конспект программиста

Д.В. Николенко

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ ПО JavaScript

Дискета содержит примеры всех программ

Наука и Техника
ИЗДАТЕЛЬСТВО

Идеальная книга по возможностям и принципам организации языка JavaScript, обучающая на примерах созданию интерактивных Web-страниц. Освоить уроки поможет прилагаемая дискета.

Николенко Д. В.

Практические занятия по JavaScript

для начинающих

Наука и техника
2000

Издательство «НАУКА И ТЕХНИКА»

Николенко Д. В.

Практические занятия по JavaScript

Под редакцией С.Л. Корякина-Черняка, члена Международной академии информационных процессов и технологий

Серия «Конспект программиста»

Эта книга уникальна. Она написана специально для Вас, если Вы чувствуете необходимость реализовать на создаваемых Вами Web-страницах собственные сценарии. Пусть у Вас мало свободного времени, а детальное изучение используемых для этого языков не представляется Вам лучшим способом включиться в работу. Хорошо, если Вы уже имеете некоторый опыт создания Web-страниц, знакомы с HTML и можете разобраться в HTML-кодах. В этом случае книга, которую Вы держите в руках — это именно то, что Вам необходимо.

Издание состоит из 12 глав. В каждой главе раскрываются наиболее важные особенности написания кодов и возможности обогащения HTML с использованием JavaScript. В тексте приводятся листинги программ. В целях приобретения опыта написания программ мы рекомендуем вводить примеры с клавиатуры, запоминать их в отдельном файле, который затем может быть загружен в броузер и проверен на работоспособность.

Книга является практическим руководством, рассчитанным на начинающих пользоваться возможностями JavaScript для обогащения своих страниц. Прочитав и изучив пособие, Вы будете иметь достаточное представление о том, как создаются скрипты на JavaScript. Для дальнейшей работы Вам будет необходимо использовать лишь справочные материалы по JavaScript.

Приложение этой книги представляет собой небольшой лаконичный справочник по объектам, свойствам и методам JavaScript. Он ни в коей степени не создавался полным или исчерпывающим, давая право на существование объемным томам по этому языку для опытных пользователей. В справочном приложении приведены сведения о наиболее важных объектах JavaScript с указанием их основных свойств и методов.

К книге прилагается дискета, содержащая файлы со скриптами примеров, благодаря которым Вам будет гораздо легче изучить язык JavaScript и работать.

(812) 567-70-25, (044) 559-27-40 **WWW.NIT.ALFACOM.NET**

9795793 101263

Иллюстрации: Д.В. Николенко
Компьютерная верстка: К.В. Болдырев
Компьютерный дизайн обложки: К.В. Болдырев

ISBN 5-7931-0126-8

ООО «КОРОНА принт», лицензия № 065007 от 18 февраля 1997 г.
198005, Санкт-Петербург, Измайловский пр., 29
Подписано в печать 11.08.2000. Формат 70x100
Бумага газетная. Печать офсетная. Объем 8 печ. л.
Тираж 5 000 экз. Заказ № 1046.

Отпечатано с готовых диапозитивов в ордена Трудового Красного Знамени ГП -Техническая книга» Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
198005, Санкт-Петербург, Измайловский пр., 29.

Содержание

1 ПЕРВЫЕ ШАГИ

- *Что такое JavaScript?*
- *JavaScript — это не то же самое, что Java!*
- *Как запустить скрипт, написанный на JavaScript?*
- *Включение скриптов JavaScript в HTML-страницу*
- *Броузеры, не поддерживающие JavaScript*
- *События*
- *функции*

Что такое JavaScript?

JavaScript — это сравнительно новый язык для написания сценариев, разработанный компанией Netscape. При помощи языка JavaScript мы можем создавать интерактивные web-страницы наиболее удобным и эффективным способом. В настоящем практическом пособии приведены примеры, которые, по мнению автора, являются наиболее важными. Они смогут продемонстрировать возможности JavaSript и принципы организации языка. В этом пособии приводятся примеры того, что можно сделать, используя JavaScript, а также, что не менее важно, рассказывается, как это сделать.

JavaScript — это не то же самое, что Java!

Многие люди считают, что язык JavaScript — это то же самое, что язык Java, недаром они носят одинаковые имена. Однако это неверно. Не будем разбираться в существующих различиях, важно лишь помнить, что JavaScript и Java — это разные языки, хотя в них имеется много общего.

Как запустить скрипт, написанный на JavaScript?

Что требуется для того, чтобы запустить программу-скрипт, написанный на JavaScript? Все что для этого требуется — это обычный интернет-броузер, с активированной способностью распознавать JavaScript. Таким броузером, например, может быть Netscape Navigator версии 2.0 и выше или Microsoft Internet Explorer версии 3.0 и выше. Поскольку большинство пользователей сети Интернет используют именно эти два броузера, то становится понятным, почему именно язык JavaScript является наиболее предпочтительным языком при решении задач улучшения качества web-страниц. Заметим также, что для понимания изложенного здесь материала требуется знание основ HTML — языка разметки, при помощи которого создаются web-страницы.

Включение скриптов JavaScript в HTML-страницу

Текст программы на языке JavaScript вставляется непосредственно в файл HTML-страницы. Чтобы посмотреть, как это выглядит, приведем пример простейшей HTML-страницы.

```
<html>
 <body>
  <br>
  Это обычный HTML документ
  <br>
   <script language="JavaScript">
   document.write("Это и есть JavaScript!")
   </script>
  <br>
  Выходим обратно в HTML
 </body>
</html>
```

На первый взгляд все выглядит как обычный HTML-файл. Единственным отличием является то, что в нем содержится следующий фрагмент:

```
<script language="JavaScript">
document.write("Это и есть JavaScript!")
</script>
```

Этот фрагмент представляет собой скрипт, написанный на языке JavaScript. Чтобы посмотреть, как такой скрипт будет работать, наберите текст HTML-страницы, приведенный выше. Затем сохраните его, как обычную HTML-страницу и загрузите в броузер с активированным JavaScript. На экране своего монитора в окне броузера Вы увидите три строчки, которые будут выглядеть примерно так:

```
Это обычный HTML-документ.
Это и есть JavaScript!
Выходим обратно в HTML.
```

Конечно, этот скрипт вряд ли может оказаться сколько-нибудь полезным, ведь подобную информацию можно записать при помощи обычного HTML гораздо более простым способом. Целью этого примера было показать, как вставляется JavaScript в файл HTML-документа и продемонстрировать назначение новых ярлыков <script> и </script>.

Вся информация, которая располагается между этими ярлыками, воспринимается как программа на языке JavaScript. В тексте программы встречается строчка document.write() — одна из наиболее важных команд в программировании на JavaScript. Строчка document.write() используется тогда, когда нужно записать что-либо в документ, в данном случае, в HTML-документ. Так, наша маленькая программа на JavaScript записывает текст

Это и есть JavaScript!

в наш HTML-документ.

Броузеры, не поддерживающие JavaScript

Как будет выглядеть наша страничка, если просматривающим ее броузер не поддерживает JavaScript? Броузер, который не понимает JavaScript, также не поймет и ярлык <script>. Такой броузер проигнорирует этот ярлык и воспримет все, что за ним следует, как обычный текст. А это значит, что пользователь увидит на своем экране весть текст программы на JavaScript, вставленный в HTML-страничку.

Конечно, нам бы этого не хотелось. Существует возможность скрыть текст JavaScript от пользователя, который просматривает страничку с помощью старых версий броузера. Для этого используется ярлык комментария <!-- -->. С использованием обозначения комментария наша страничка примет такой вид:

```
<html>
<body>
<br>
Это обычный HTML-документ.
<br>
  <script language="JavaScript">
   <!— спрячемся от старых версий броузеров
   document.write("Это и есть JavaScript")
   // ->
  </script>
<br>
Возвращаемся назад в HTML
</body>
</html>
```

Если броузер не понимает JavaScript, то на экране мы увидим:

```
Это обычный HTML-документ.
Возвращаемся назад в HTML.
```

Если же мы не обнесем текст программы JavaScript скобками ярлыка комментария, то в окне броузера, не понимающего JavaScript, будет следующий текст:

```
Это обычный HTML-документ.
document.write("Это и есть JavaScript!")
Возвращаемся обратно в HTML.
```

Конечно, не существует способа скрыть от пользователя текст JavaScript программы полностью, раз и навсегда. Здесь мы лишь можем скрыть текст программы-скрипта от возможности его просмотра в окне старых версий броузеров в тексте основного HTML-документа. Пользователь все же имеет возможность просмотреть текст программы-срипта, если воспользуется опцией просмотра источника HTML-документа, имеющейся в меню "вид". Не существует способа скрыть от глаз пользователя внутреннее устройство того или иного эффекта, созданного при помощи JavaScript.

События

Событие — это очень важное в программировании на JavaScript понятие. События главным образом порождаются пользователем, являются следствиями его действий. Если пользователь нажимает кнопку мыши, то происходит событие, которое называется *Click*. Если экранный указатель мыши движется по ссылке HTML-документа, происходит событие *MouseOver*.

Существует несколько различных событий. Пусть мы хотим, чтобы наша JavaScript программа реагировала на несколько разных событий. Это можно сделать при помощи event-handlers. Например, можно заставить появиться новое всплывающее окно, которое появляется при нажатии кнопки. Появление нового окна будет следствием наступления события *Click*.

Event-handler, иначе говоря обработчик событий или средство управления событиями, который требуется для организации такого действия, называется *onClick*. Это средство управления событиями сообщает компьютеру, какие действия необходимо совершить при наступлении данного события. Ниже приведена простая программа, иллюстрирующая то, как может быть использовано средство *onClick*.

```
<form>
 <input type="button" value="щелкни меня" onClick="alert('Ой-ой')">
</form>
```

В приведенном выше примере есть несколько новых моментов. Рассмотрим все подробно, шаг за шагом. В этом примере мы описали *форму* или *бланк,* в котором имеется кнопка. Это осуществляется при помощи инструкций HTML, и поэтому мы не будем на этом останавливаться. Для нас важной частью является инструкция onClick="alert('Ой-ой')" , которая размещена внутри ярлыка <input>.

Как уже было замечено ранее, она определяет то действие, которое будет выполнено при нажатии кнопки. Если происходит событие *Click,* то компьютер выполнит инструкцию alert('Ой-ой'). Эта инструкция — фрагмент . JavaScript. Заметьте, что в этом случае мы не использовали ярлык <script>.

Инструкция *alert()* позволяет создавать всплывающие окна. В скобках следует записать строку, в нашем случае это 'Ой-ой'. Текст строки будет виден во всплывшем окне. Наш скрипт заставляет появиться на экране новое окно с текстом 'Ой-ой', когда пользователь нажимает на кнопку.

Отметим маленькую деталь, которая, однако, может создать некоторые недоразумения. В команде document.write() мы писали двойные кавычки ", а в комбинации инструкций, содержащей alert(), мы используем одиночные кавычки ' — почему? Вообще говоря, можно пользоваться как теми, так и другими кавычками.

В нашем последнем примере мы используем две пары кавычек и пишем onClick="alert('Ой-ой')". Но если мы напишем onClick="alert('Ой-ой')", то компьютер не поймет, какая часть относится к onClick, а какая — нет.

Для того, чтобы не возникали недоразумения такого рода, необходимо использовать разные типы кавычек, при этом нет никакой разницы, в каком порядке они будут применяться. Ту же формулу можно переписать в другом виде, поменяв кавычки местами: onClick='alert("Ой-ой")'.

Существует множество средств управления событиями. С некоторыми из них мы познакомимся далее, однако не со всеми. Если у Вас появляется необходимость познакомиться с другими средствами управления событиями, обратитесь к справочным пособиям.

Если Вы используете Netscape Navigator, то во всплывающем окне будет содержаться надпись *JavaScript alert,* что означает *предупреждение JavaScript.* Смысл этой надписи — соображения безопасности. Можно создать подобное всплывающее окно с помощью метода prompt(). В окне, созданном при помощи этого метода, предусмотрена возможность осуществления ввода информации.

Можно создать такой скрипт хулиган, который будет имитировать системное сообщение, запрашивающее пароль. Вспомогательный текст во всплывающем окне сообщает, что данное окно создано Вашим броузером, а не операционной системой. Эту надпись на всплывающем окне никак нельзя уничтожить.

Функции

В большинстве программ на JavaScript мы будем пользоваться функциями . Рассмотрим пример и напишем скрипт, который будет выводить текст, причем один и тот же текст будет выводиться три раза. Вот возможный вариант программы:

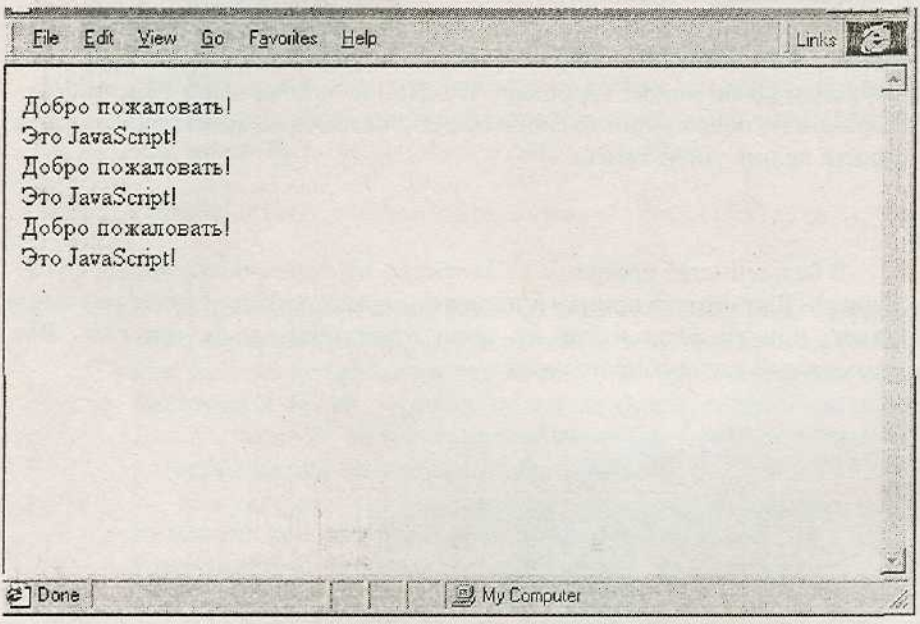
```
<html>
 <script language="JavaScript">
 <!- hide
   document.write("Добро   пожаловать!<br>");
   document.write("Это   JavaScript!<br>");
   document.write("Добро   пожаловать!<br>");
   document.write("Это   JavaScript!<br>");
   document.write("Добро   пожаловать!<br>");
   document.write("Это   JavaScript!<br>");
 // ->
 </script>
</html>
```

Этот скрипт выведет такой текст:

Добро пожаловать!
Это JavaScript!

Этот текст будет выведен три раза. Насколько это эффективно? Можно решить эту задачу более удобным способом. Ниже приводится другой вариант решения задачи.

```
<html>
<script language="JavaScript">
<!- hide
   function myFunction() {
    document.write("Добро    пожаловать!<br>");
    document.write("Это JavaScript!<br>") ;
   }
 myFunction();
 myFunction();
 myFunction();
// ->
</script>
</html>
```



Так выглядит окно броузера после загрузки скрипта, содержащего описанную выше функцию

В программе этого скрипта мы использовали функцию. Для определения функции нам понадобилось три строчки:

```
function myFunction() {
 document.write("Добро пожаловать!<br>") ;
 document.write("Это JavaScript!<br>");
}
```

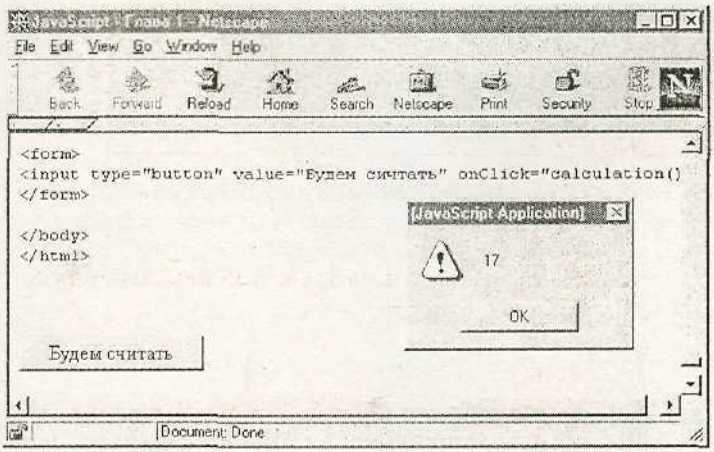
Слова, помещенные между скобок {}, составляют команды, образующие функцию *myFunction()*. Это значит, что две используемых нами функции document.write() объединяются в одно целое, и они могут быть

выполнены совместно при помощи обращения к функции. В нашей программе мы обращаемся к функции три раза, мы три раза написали *myFunction()* после того, как определили эту функцию. Мы три раза вызвали функцию, т.е. содержание функции будет исполнено три раза.

Это был пример очень простой функции. По мере чтения настоящего пособия Вы постепенно будете убеждаться в том, что функции действительно оказываются очень полезны.

Функции также можно использовать в сочетании со средствами управления событиями. Рассмотрим пример:

```
<html>
<head>
 <script language="JavaScript">
 <!- hide
  function calculation() {
    var x= 12;
    var y= 5;
    var result- x + y;
    alert (result) ;
  }
// ->
</script>
</head>
<body>
 <form>
 <input type="button" value-"Будем считать"
 onClick="calculation()">
 </form>
</body>
</html>
```



Окно броузера и всплывающее окно, содержащее результат функции, описанной в вышеприведенном скрипте, после нажатия кнопки "Будем считать"

Нажатая кнопка заставляет компьютер выполнить функцию *calculation()*. Эта функция производит определенные вычисления с переменными *x, y и result*. Мы можем определить переменные с помощью специального слова *var*. Переменные могут использоваться для хранения различных величин, например для хранения чисел, текстовых строк и т.д.

Строка программы var result= x + y; сообщает броузеру, что необходимо создать переменную *result* и в этой переменной сохранять значение, равное сумме значений переменных *x + y* (т.е. 5 + 12). После выполнения этой функции переменная *result* будет содержать число 17. Команда alert(result) в этом случае эквивалентна команде alert{17). Мы видим всплывающее окно, в котором записано число 17.

2

HTML ДОКУМЕНТЫ

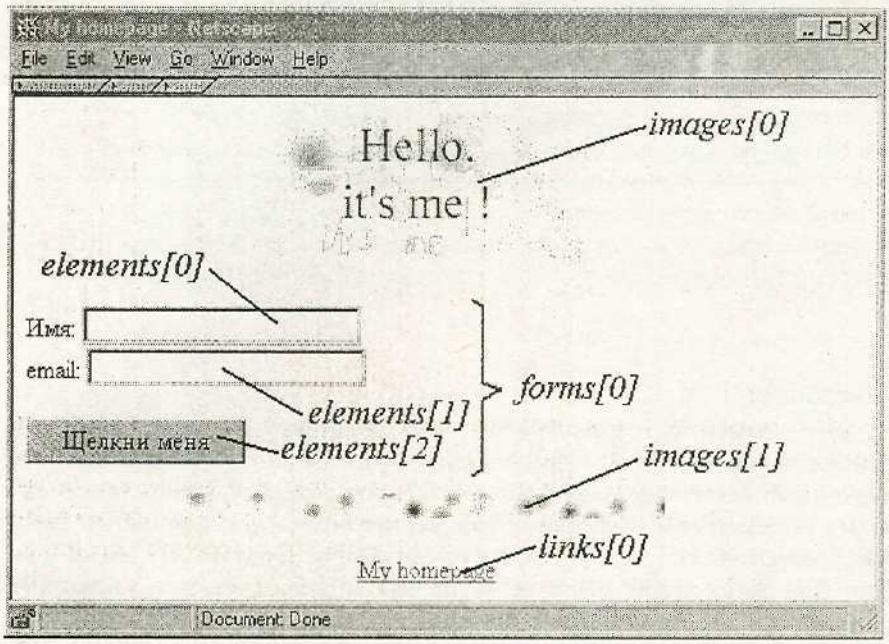
- *Иерархии JavaScript*
- *Объект местоположения — location-object*

Иерархии JavaScript

Все элементы, составляющие Web-страницу, рассматриваются в JavaScript как элементы, иерархически организованные. Каждый элемент страницы представляет собой *объект*. Каждый объект обладает определенными *свойствами* и располагает своими *методами*. Для понимания того, как устроен объект, важно понимать устройство иерархии HTML-объектов. Для того, чтобы было легче понять, о чем идет речь, приведем пример. Вот простая HTML-страница:

```
<html>
<head>
<title>My homepage</title>
</head>
<body bgcolor=#ffffff>
 <center>
  <img src="home.gif" name="pic1" width=200 height=100>
 </center>
 <p>
 <form name="myForm">
  Name:
  <input type="text" name="name" value=""><br>
  e-Mail:
  <input type="text" name="email" value=""><br><br>
  <input type="button" value="Push me" name="myButton"
  onClick="alert('Yo')">
 </form>
 <p>
 <center>
 <img src="ruler.gif" name="pic4" width=300 height=15>
 <P>
 <a href="http://www.chat.ru/~oollyymmpp">Моя страничка</a>
 </center>
</body>
</html>
```

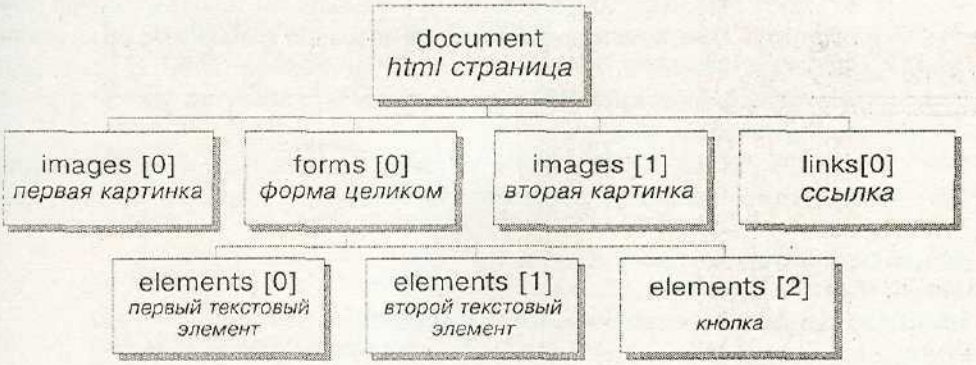
Эта страница на экране выглядит следующим образом:



Здесь мы имеем две картинки, одну ссылку и одну форму, содержащую Два поля для ввода текстовой информации. С точки зрения языка JavaScript окно броузера представляет собой объект, называемый *window.* Этот объект содержит в себе несколько элементов, например, панель состояния. В объект — окно *window* может быть загружен HTML-документ (или файл другого типа, здесь же мы ограничимся HTML-документами).

HTML-страница представляет собой новый объект — *document.* Объект *document* представляет собой HTML-документ, который загружен в броузер в данное время. Этот объект очень важен в языке JavaScript, и мы будем пользоваться им очень часто. Этот объект имеет свои свойства, например, одним из свойств объекта *document* является цвет фона документа.

Существенным моментом является тот факт, что все объекты, описанные в HTML, являются свойствами объекта *document.* Например, HTML-объект, представляющий собой ссылку, содержащуюся в документе, или объект, который представляет собой форму — все это свойства объекта *document.* Следующий рисунок иллюстрирует структуру иерархии, образующейся в нашей HTML-странице.



Мы хотим научиться получать информацию о разных объектах и управлять объектами. Для этого мы должны знать, как можно управлять объектами. Каждый объект в иерархии имеет свое имя в виде своеобразного адреса. Чтобы назвать, например, первую картинку на HTML-странице, нужно посмотреть, где в иерархии находится эта страница. Начинать нужно с самого начала, с основания.

Первый, и основной, здесь объект, — это документ, его имя *document.* Первая картинка на странице имеет имя *images[0].* Все это значит то, что мы можем обратиться к этой картинке посредством языка JavaScript, назвав ее *document.images[0].* Если же, например, у нас появиться необходимость посмотреть, какую информацию пользователь ввел в элемент первой формы, то вначале нам необходимо позаботиться о том, как мы сможем обратиться к самому объекту, представленному данной формой.

И опять начинаем с самого основания иерархии, с его главного объекта. Далее обращаемся к формам и затем переходим к элементам форм, таким образом, первый элемент нашей формы будет иметь имя *document.forms[0].element[0].* И все же, как мы сможем узнать, какой в этом элементе введен текст? Для того, чтобы узнать, какими свойствами и методами может обладать тот или иной объект, удобнее всего обратиться к справочному пособию по JavaScript.

Из справочника легко можно узнать, что текстовый элемент имеет свойство, носящее имя *value,* т.е. значение. То, чему равняется *value,* и представляет собой текст, введенный пользователем. Вот как мы можем прочитать то, что ввел пользователь:

```
name= document.forms[0].elements[0].value;
```

Текстовая строка, введенная пользователем, сейчас сохранена в переменной *name.* Далее мы можем пользоваться этой переменной. Например, мы можем создать всплывающее окно, используя эту переменную: alert("Привет, " + name). Если в данный элемент формы введена строка 'Яков', то в открывшемся окне будет написано ' *Привет, Яков'.*

Но если наша страничка велика, и в ней много различных объектов, то может оказаться затруднительным указывать имена объектов при помощи чисел. При этом могут появляться довольно громоздкие конструкции, например, *document.forms[3].elements[17]* и *document.forms[2].elements[18].* Эту трудность можно обойти, если присвоить объектам уникальные, неповторяющиеся имена.

Два разных объекта не могут иметь одинаковые имена. Посмотрим, как может выглядеть программа на JavaScript, в которой используются такие имена.

```
<form name="myForm">
Name:
<input type="text" name="name" value=""><br>
...
```

В этом примере первая форма *forms[0]* имеет собственное уникальное имя *my Form*. Вместо *elements[0]* мы можем использовать you имя, которое задано этому элементу при помощи свойства имени *name,* которое может быть указано для задания собственного уникального имени в ярлыке <input>, при помощи которого описывается данный элемент. Тогда вместо строчки

```
name=  document.forms[0].elements[0].value;
```

мы можем написать:

```
name=  document.myForm.name.value;
```

Используя такие имена мы облегчаем себе процесс написания программы, особенно в том случае, если создаваемая страница довольно велика. Здесь немаловажно помнить, что имена чувствительны к регистру, и имя *myForm* — это вовсе не то же самое, что *myform*. Свойства объектов JavaScript не ограничиваются операциями прочтения значений полей, заполненных информацией, которая вводится пользователем.

Свойствам объектов можно задать и иные значения. Например, можно организовать ввод строки текста в текстовый элемент формы. Посмотрите, как это делает скрипт в следующем примере:

```
<form  name="myForm">
<input  type="text"  name="input"  value="**  **  **">
<input  type="button"  value="Write"
 onClick="docurnent .myForm. input.value=  'Ой' ;  ">
```

Интересующая нас часть находится справа от onClick во втором ярлыке <input>.

Объект местоположения — location-object

Помимо объектов-документов, в JavaScript существуют объекты-окна. Используя эти объекты, можно, например, создавать новые всплывающие окна. Пример такой программы приведен на рисунке. Введите ее и сохраните в виде HTML-файла, а затем загрузите в броузер и поэкспериментируйте. Существует также объект местоположения. Этот объект представляет собой адрес загруженного HTML-документа.

Например, если Вы загрузите страницу *http://www.somehost.ru/page.html,* то *location.href* будет иметь значение, равное именно указанной строчке адреса страницы. Интересно то, что мы можем присвоить *location.href* новые значения. Например, нажимая кнопку, описанную в следующем примере, мы загружаем новую страницу в существующее окно броузера.

```
<form>
<input type=button value="Yahoo"
 onClick="location.href='http://www.yahoo.com'; ">
</form>
```

3

ФРЕЙМЫ

- *Понятие фреймов*
- *Создание и использование фреймов*
- *Фреймы и JavaScript*
- *Меню навигации*

Понятие фреймов

Фреймы — это английское слово, в переводе означающее *рамки*. HTML-страница, в которой используются фреймы, выглядит разделенной на несколько частей, которые окаймлены границами, как рамками. Мы не будем переводить слово фрейм, более того, даже будем использовать его русифицированную форму множественного числа.

Все это оправдывается тем фактом, что все-таки фреймы — это не совсем то, что обозначает русское слово *рамки*. К тому же в программировании фреймов мы будем вынуждены пользоваться английскими словами, а в "родном" написании слово фреймы выглядит как frames.

Создание и использование фреймов

Часто возникает вопрос о том, как могут быть связаны друг с другом, казалось бы, совершенно разные вещи, такие как фреймы и язык JavaScript. Не будем спешить, и остановимся вначале на рассказе о том, что представляют собой фреймы и для чего они используются. После этого рассмотрим вопрос о том, как могут быть взаимосвязаны фреймы и язык JavaScript.

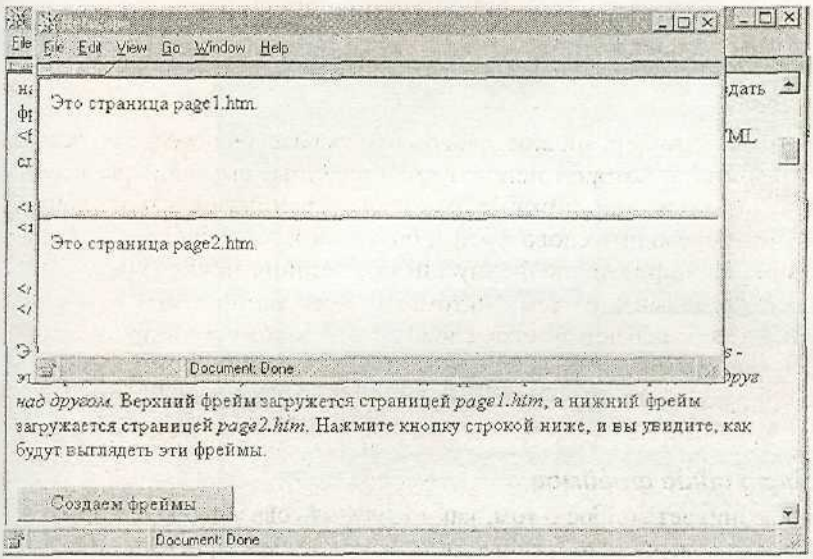
Окно броузера может быть разделено на несколько фреймов. Фрейм представляет собой прямоугольную область экрана, расположенную в окне броузера. Каждый фрейм показывает свой документ, как правило, это обычный HTML-документ. Например, можно создать два окна и в одно из них загрузить домашнюю страничку компании Netscape, а в другое — домашнюю страничку компании Microsoft.

Несмотря на то, что создание фреймов осуществляется в рамках языка HTML, напомним основные принципы и способы создания фреймов. Для того, чтобы создать фреймы, мы пользуемся двумя ярлыками, а именно ярлыком <frameset> и ярлыком <frame>. Страница, разбитая на

два фрейма, может быть описана при помощи HTML следующим образом:

```
<html>
<frameset    rows="50%,50%">
  <frame   src="page1.htm"   name="frame1">
  <frame   src="page2.htm"   name="frame2">
</frameset>
</html>
```

Этот код порождает два фрейма. В ярлыке <frameset> указано свойство *rows,* а *rows* — это в ряды. Это свойство указывает на то, что два фрейма расположены рядами, *друг над другом.* Верхний фрейм загружается страницей *page1.htm,* а нижний фрейм загружается страницей *page2.htm.*



Такие окна фреймов образуются в результате исполнения приведенного выше HTML-кода

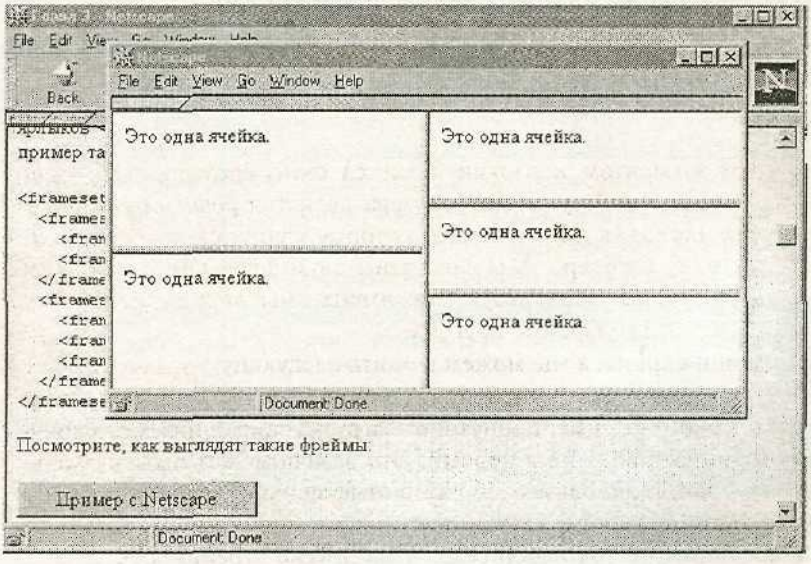
Если Вы захотите расположить фреймы столбцами, то вместо свойства *rows* в ярлыке <frameset> укажите *cols.* Процентные величины *"50%,50%"* указывают относительные размеры окон фреймов. Если Вы не хотите утруждать себя вычислениями того, каким должен быть размер второго окна, чтобы вцелом получилось 100%, то можно указать размер окна в пикселях, при этом знак % опускается.

Каждый фрейм имеет свое уникальное имя, которое задается при помощи свойства *name* в ярлыке <frame>. При помощи этого имени фрейм легко доступен при. описаниях на JavaScript.

Можно использовать несколько последовательных описаний фреймов при помощи ярлыков <frameset>, при этом фреймы могут быть расположены друг в друге. Вот пример такого расположения фреймов:

```
<frameset  cols="50%,50%">
  <frameset  rows="50%,50%">
    <frame  src="cell.htm">
    <frame  src="cell.htm">
  </frameset>
  <frameset  rows="33%,33%,33%">
    <frame  src="cell.htm">
    <frame  src="cell.htm">
    <frame  src="cell.htm">
  </frameset>
</frameset>
```

Посмотрите, как выглядят такие фреймы:



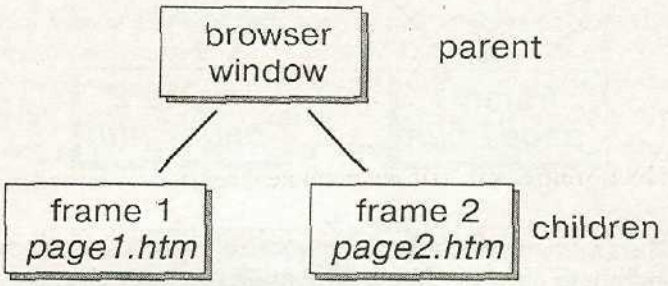
Сложная структура фреймов

Существует возможность установить толщину рамки фрейма при помощи свойств *border* в ярлыке .<frameset>. Если border = 0, то это обозначает, что Вы не хотите, чтобы фреймы имели видимых рамок.

Фреймы и JavaScript

Сейчас мы переходим к обсуждению того, как с точки зрения JavaScript выглядят фреймы, расположенные в окне броузера. Для этого создадим два фрейма, точно так, как это было сделано в первом примере.

Ранее мы видели, что все элементы web-страницы иерархически организованы. Фреймы — не исключение. Иерархия страницы первого примера выглядит таким образом:



Основным элементом иерархии является окно броузера. Это окно разбивается на два фрейма. Окно броузера является *родительским* элементом по отношению к двум фреймам, которые являются дочерними по отношению к окну броузера. Мы дали этим двум фреймам уникальные имена *frame1* и *frame2*. Используя эти имена, мы можем передавать информацию из одного фрейма другому.

При помощи скриптов мы можем решить следующую задачу. Нужно создать такую страницу, в которой при нажатии ссылки, расположенной в документе первого фрейма, происходит загрузка запрашиваемого документа во второй фрейм, а не в первый. Это задача может быть актуальной в случаях, когда используются различные меню. Фрейм, в котором содержится меню или содержание, остается неподвижным и неизменным, и в котором имеется набор ссылок на другие страницы.

Рассмотрим три следующих варианта:

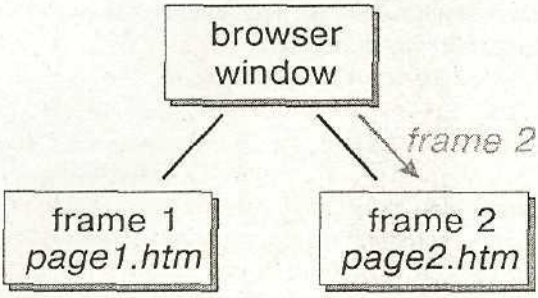
родительское окно или фрейм обращаются к дочернему фрейму;
дочерний фрейм обращается к родительскому окну или фрейму;
дочерний фрейм обращается к другому дочернему фрейму.

С точки зрения окна броузера (простите за жаргон) дочерние фреймы имеют имена *frame1* and *frame2*. Выше на рисунке показано, что окно броузера имеет прямые (родительские) связи с этими фреймами.

Поэтому, если Вы используете скрипт, расположенный в родительском фрейме (в окне броузера), т.е. во фрейме, который содержит два дочерних фрейма, и хотите обратиться из родительского фрейма к дочер-

ним, то у вас есть простой метод — укажите имя дочернего фрейма, к которому необходимо обратиться. К примеру, можно написать так:

```
frame2.document.write ("Обращение из родительского окна.");
```

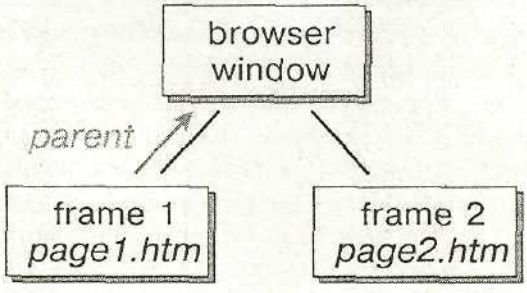


Иногда появляется необходимость обратиться к родительскому окну из дочернего фрейма. Эта необходимость может возникнуть, например, тогда, когда нужно избавиться от фреймов. Освобождение от фреймов означает, что в окно броузера загружается другая страница, которая отличается от той, что породила фреймы. В нашем примере — это страница, которая содержится в дочернем фрейме.

Мы можем обратиться к родительскому окну или родительскому фрейму при помощи *parent* (родитель) . Для того, чтобы загрузить в окно броузера новый документ, необходимо задать новый адрес URL, присвоив его объекту местоположения *location.href.* Поскольку мы хотим удалить фреймы, то нам необходимо обратиться к объекту местоположения, который расположен в родительском окне.

Поскольку каждый фрейм может быть загружен своим собственным документом, то у каждого фрейма есть свой собственный объект местоположения. Мы можем загрузить новую страницу в родительское окно при помощи следующей команды, расположенной в документе дочернего фрейма:

```
parent.location.href= "http://...";
```

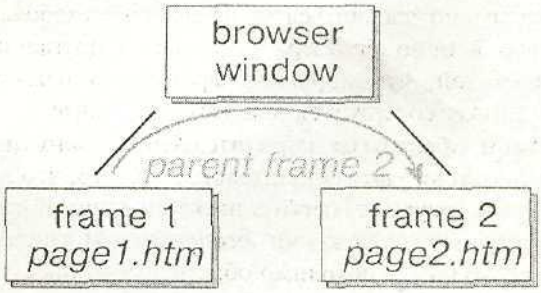


Довольно часто приходится сталкиваться с задачей, когда из одного дочернего документа необходимо обратиться к другому дочернему документу. Как записать чего-либо во второй фрейм, находясь в первом фрейме? Иными словами, такая команда должна быть записана в документе *page1.htm,* загруженном в первый фрейм, которая приводила бы появлению во втором фрейме необходимых записей.

На нашей картинке нет прямых связей между первым и вторым дочерними фреймами. Это означает, что мы не имеем возможности обратиться из первого фрейма *frame1* ко второму, указав имя второго фрейма *frame2* в документе *page1.htm,* поскольку первый фрейм ничего "не знает" о существовании второго фрейма.

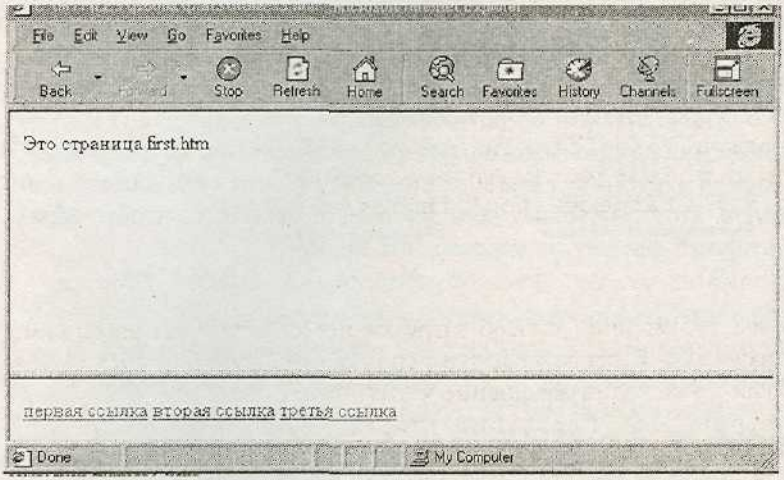
С точки зрения родительского окна второй фрейм носит *имя frame2,* а с точки зрения первого фрейма к родительскому окну обращаются при помощи *parent.* Поэтому, объединяя все имеете, для того, чтобы обратиться ко второму фрейму из первого, мы пишем:

```
parent.frame2.document.write("Это обращение из frame1.");
```



Меню навигации

Рассмотрим пример того, как можно создавать меню навигации. Меню навигации — это набор нескольких ссылок, расположенных в одном фрейме. Если пользователь выбирает какую-либо ссылку из этого фрейма, то выбранная страница загружается не в этот фрейм, а во второй. Вот пример:



Окно броузера после нажатия кнопки Navigationbar — Меню навигации

В начале мы пишем документ, который создает фреймы. Этот документ выглядит подобно тому, что был приведен в первом примере.

Документframes3.htm:

```
<html>
<frameset rows="80%,20%">
 <frame src="start.htm" name="main">
 <frame src="menu.htm" name="menu">
</frameset>
</html>
```

Документ *start.htm* — это стартовая страница, которая будет вначале загружена во фрейм, носящий имя *MAIN.* К этой странице нет никаких особых требований.

Во фрейм, который назван *MENU,* будет загружена страница *menu.htm.*

Документmenu.htm:

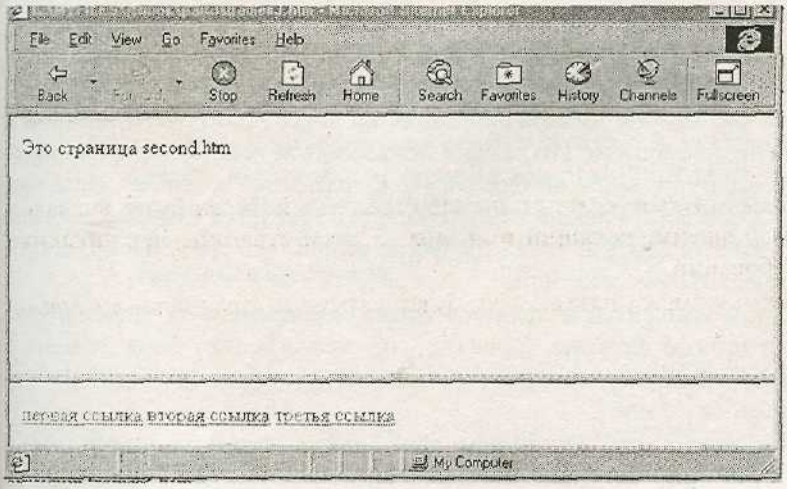
```
<html>
<head>
<script language="JavaScript">
 <!- hide
  function load(url) {
   parent.main.location.href= url;
  }
 // ->
</script>
</head>
<body>
 <a href="javascript:load('first.htm')">first</a>
 <a href="second.htm" target="main">second</a>
 <a href="third.htm" target="_top">third</a>
</body>
</html>
```

Здесь мы видим иной способ загрузки новой страницы во фрейм, носящий имя *MAIN.* В первой ссылке использована функция *load()*. Посмотрим, как происходит обращение к этой функции?

```
<a href="javascript:load('first.htm')">first</a>
```

Эта строчка заставляет броузер выполнять программу-скрипт, написанную на JavaScript, вместо того, чтобы обратиться к обычной загрузке страницы. Имя документа *'first.htm'* записано в скобках. Это имя передается в виде аргумента функции *load()*. Функция *load()* определена следующим образом:

```
function load(url) {
 parent.main.location.href= url;
}
```

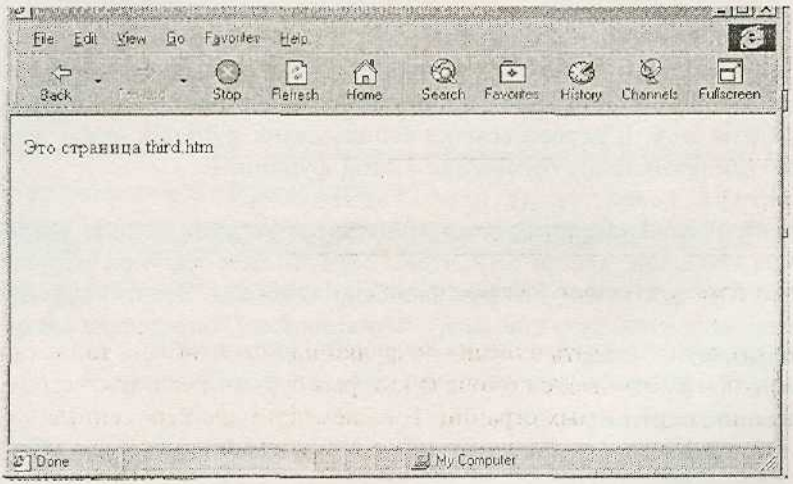


Окно броузера после перехода по ссылке "вторая ссылка"

Здесь в скобках используется url, Подразумевается, что строка "first1.htm" хранится в переменной url. В функции load() мы можем использовать эту переменную.

Во второй ссылке использовано свойство *target.* Вообще говоря, оно не является специфическим для JavaScript, оно скорее принадлежит языку HTML. Здесь мы лишь указали имя фрейма. Заметьте, что нам не понадобилось писать слово *parent* перед именем фрейма. Быть может это и выглядит немного странным, но этому есть простое объяснение — это свойство не является свойством JavaScript, оно является свойством HTML.

Третья ссылка показывает, как можно избавиться от фреймов, пользуясь свойством *target.* Если возникает необходимость избавиться от фреймов и при этом использовать функцию *load(),* то все что нужно сделать — это написать строчку *parent.location.href= url* в скобках у этой функции.



Окно броузера после перехода по ссылке "третья ссылка"

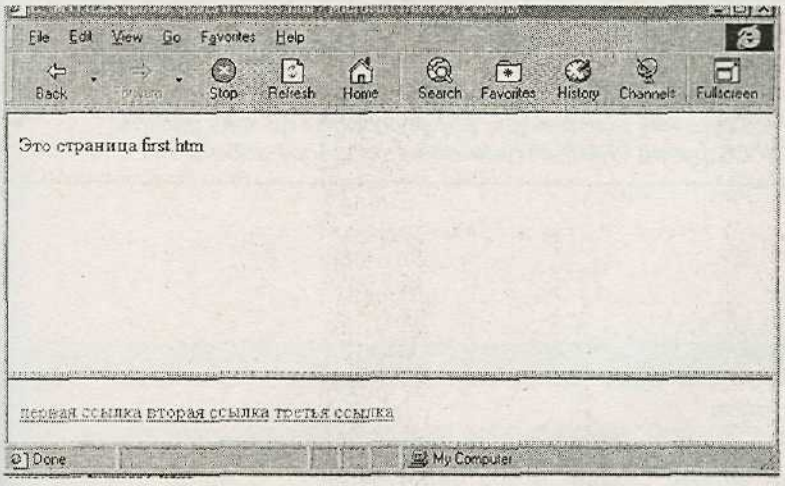
Как поступить — выбор остается за Вами. Это зависит от того, что Вы хотите сделать. Использование свойства target — это очень простой и эффективный механизм. Им можно пользоваться всегда, если требуется загрузить страницу в другой фрейм. Использование JavaScript здесь может быть оправдано только в том случае, если требуется сложная реакция на факт нажатия ссылки, когда требуется выполнить сразу набор, состоящий из нескольких действий.

Обычной задачей здесь является задача одновременной загрузки двух страниц в разные фреймы. Конечно, эта задача может быть решена с использованием свойства *target,* однако использование JavaScript позволяет решить ее "в лоб".

Предположим, что мы имеем три фрейма, которые носят имена *frame1, frame2 и frame3.* Пользователь щелкает ссылку, расположенную во фрейме *frame1.* При этом необходимо осуществить загрузку двух разных

страниц во фреймы *frame2* и *frame3*. Вот как эта задача может быть решена средствами JavaScript:

```
function loadtwo () {
 parent.frame1.location.href=  "first.htm";
 parent.frame2.location.href=  "second.htm";
}
```



Окно броузера после перехода по ссылке "первая ссылка".

Если Вы хотите сделать описанную функцию более гибкой, то можно использовать переменные, с помощью которых будут передаваться строчные значения загружаемых страниц. В нашем примере переменные названы url1 и url2.

```
function loadtwo(url1, url2) {
 parent.frame1.location.href=  url1;
 parent.frame2.location.href=  url2;
}}
```

Далее к этой функции следует обратиться, указав конкретные имена — адреса загружаемых страниц: loadtwo("first.htm", "second.htm") или Ioadtwo{"third.htm", "forth.htm"). Использование переменных делает функцию более гибкой, ее можно использовать снова и снова, меняя лишь значения переменных.

4

ОКНА И ДОКУМЕНТЫ, СОЗДАВАЕМЫЕ В ПРОЦЕССЕ РАБОТЫ JavaScript

- *Создание новых окон*
- *Имя окна*
- *Закрытие окна*
- *Создание документов, образующихся по ходу работы*
- *Создание VRML-документов в процессе работы*

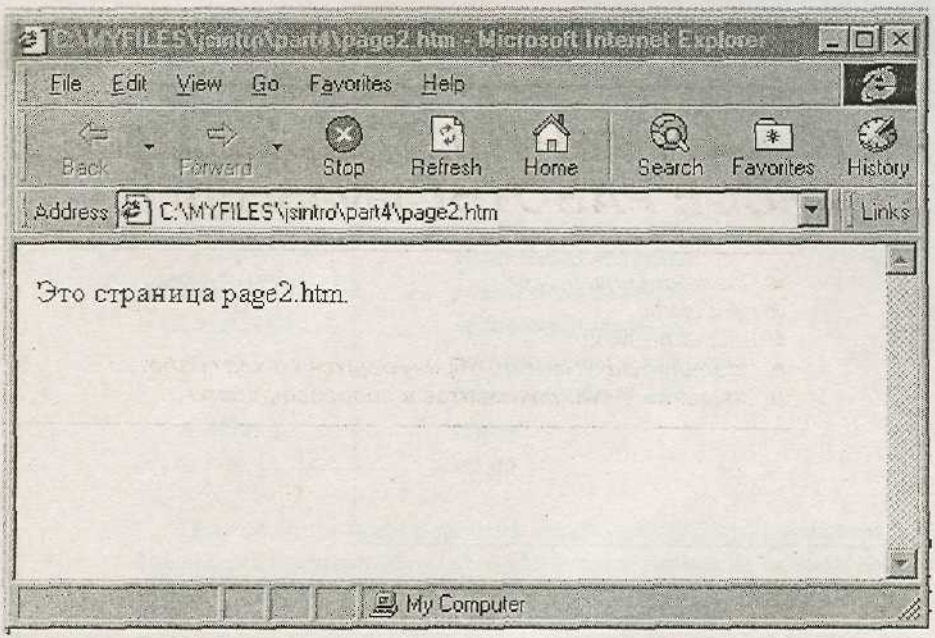
Создание новых окон

JavaScript обладает одним замечательным свойством. С помощью JavaScript можно создать новое окно броузера. В новое открывшееся окно можно загрузить уже существующий HTML-документ, но в нем также можно разместить абсолютно новый, создаваемый по ходу работы, документ. Сначала мы рассмотрим пример того, как можно создать новое окно броузера и загрузить в него существующий документ.

Следующий скрипт открывает новое окно броузера и загружает в него HTML-страничку.

```
<html>
<head>
 <script language="JavaScript">
  <!- hide
  function openWin() {
   myWin= open("page.htm");
  }
  // ->
 </script>
</head>
<body>
 <form>
  <input type="button" value="Открываем новое окно"
   onClick="openWin()">
  </form>
</body>
</html>
```

Страничка page.htm загружается в созданное окно при помощи метода open().

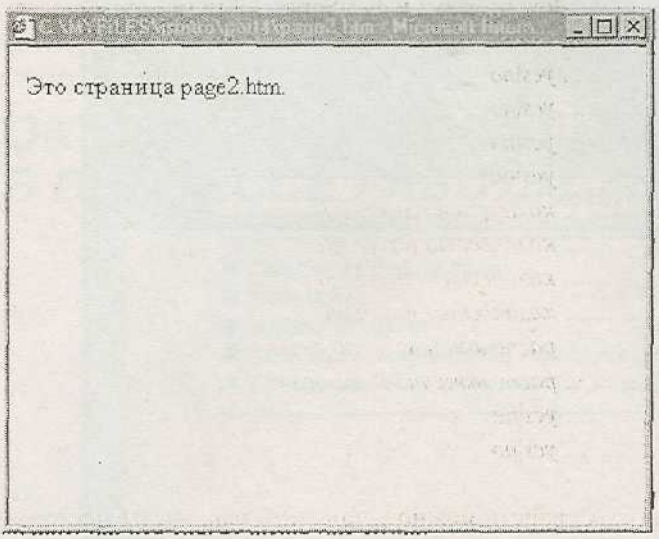


Новое окно броузера, открывшееся после нажатия кнопки "Открываем новое окно"

При открытии нового окна существует возможность выбора параметров внешнего вида создаваемого окна. Например, можно выбрать, будет или нет новое окно иметь панель инструментов, панель меню или панель строки состояния.

Также возможно задание размера открываемого окна. Следующая программа открывает окно, размер которого устанавливается 400x300. Окно не будет иметь строки состояния, меню и панели инструментов.

```
<html>
<head>
 <script language="JavaScript">
 <!— hide
  function openWin2() {
   myWin= open("page.htm", "newWindow",
   "width=400,height=300,status=no,toolbar=no,menubar=no");
  }
 // ->
 </script>
</head>
<body>
 <form>
  <input type="button" value="Открываем новое окно 400x300"
  onClick="openWin2() ">
 </form>
</body>
</html>
```



Окно, открывающееся при нажатии кнопки "Открываем новое окно 400x300"

В приведенной выше программе мы использовали следующие свойства объекта window, записанные в строке

"width=400,height=300,status=no,toolbar=no,menubar=no".

Заметьте, что внутри строки не следует использовать пробелы.

Данная таблица содержит свойства объекта window (левый столбец) и значения, которые могут принимать эти свойства (правый столбец).

directories.....*yes\no*
height.....*количество пикселей*
location.....*yes\no*
menubar.....*yes\no*
resizable.....*yes\no*
scrollbars.....*yes\no*
status.....*yes\no*
toolbar.....*yes\no*
width ..... *количество пикселей*

Список свойств в вышеприведенной таблице не является исчерпывающим. Так новая версия языка JavaScript 1.2 предусматривает возможность задания других свойств. Этот вариант JavasScript понятен для броузеров Netscape Navigator версии 4.0 и выше.

Более ранние броузеры не поймут такие инструкции, и если страничка предполагает использование броузеров Netscape Navigator 2.x, Netscape Navigator 3.x или Microsoft Internet Explorer 3.x, то в ней следует избегать инструкций, содержащихся исключительно в языке JavaScript 1.2.

Свойства объекта window по версии JavaScript 1.2 приведены в следующей таблице.

alwaysLowered.....*yes\no*
alwaysRaised.....*yes\no*
dependent.....*yes\no*
hotkeys.....*yes\no*
innerWidth.....*количество пикселей*
innerHeight.....*количество пикселей*
outer Width.....*количество пикселей*
outerHeight.....*количество пикселей*
screenX.....*расположение в пикселях*
screenY.....*расположение в пикселях*
titlebar.....*yes\no*
z-lock.....*yes\no*

При помощи новых свойств можно установить параметры месторасположения открываемого окна. Подробное описание новых свойств можно найти в справочнике по JavaScript. Примеры будут рассмотрены нами ниже.

Имя окна

При открытии нового окна функция open содержала три аргумента:

```
myWin= open("page.htm", "newWindow",
"width400,height=300,status=no,toolbar=no,menubar=no");
```

Ответьте на вопрос: "Что означает второй аргумент функции и зачем он используется?" Второй аргумент функции open — это имя окна. Ранее мы уже видели, как используется свойство *target*. Если мы знаем имя существующего окна, то загрузить в это окно новый документ можно с использованием свойства *target*. Это можно осуществить следующим образом.

```
<a href="page.html" target="newWindow">
```

Здесь мы указали имя окна. Если окна с таким именем не существует, то указанная выше ссылка приведет к созданию нового окна.. Также отметим, что *myWin* не является именем этого окна. При помощи переменной *myWin* мы можем лишь обратиться к этому окну.

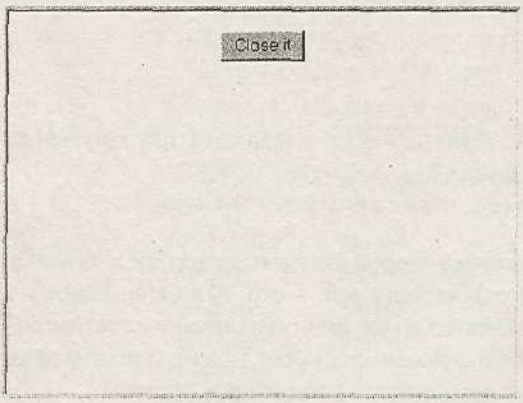
Эта переменная может использоваться в пределах данного скрипта JavaScript, т.е. в пределах того модуля JavaScript, в пределах которого она была описана. Имя же окна (в нашем случае это *newWindow)* является уникальным именем и может быть использовано во всех существующих окнах броузера.

Закрытие окна

JavaScript также позволяет закрывать окна. Для закрытия окон используется метод *close()*. Откроем окно, как это было описано выше, и загрузим в него следующую страницу:

```
<html>
<script language="JavaScript">
<!- hide
  function closeIt() {
   close ();
  }
//  ->
</script>
<center>
<form>
 <input type=button value="Закроем окно" onClick="closeIt()">
</form>
</center>
</html>
```

Если нажать кнопку в этом окне, то окно закроется.



Окно, появляющееся принажатиикнопки "откроем и закроем окно"

Метод open() и метод close() — это методы, свойственные объекту window. Если быть последовательным, то необходимо помнить, что полный вид записей инструкций открытия и закрытия окон должен быть *window.open() и window.close(), вместо open()* и *close()*.

Объект window представляет собой исключение из общего правила, и для вызова методов объекта window не нужно указывать объект (т.е. писать слово window) при обращении к тому или иному методу объекта window. Это справедливо лишь для данного конкретного объекта (объекта window).

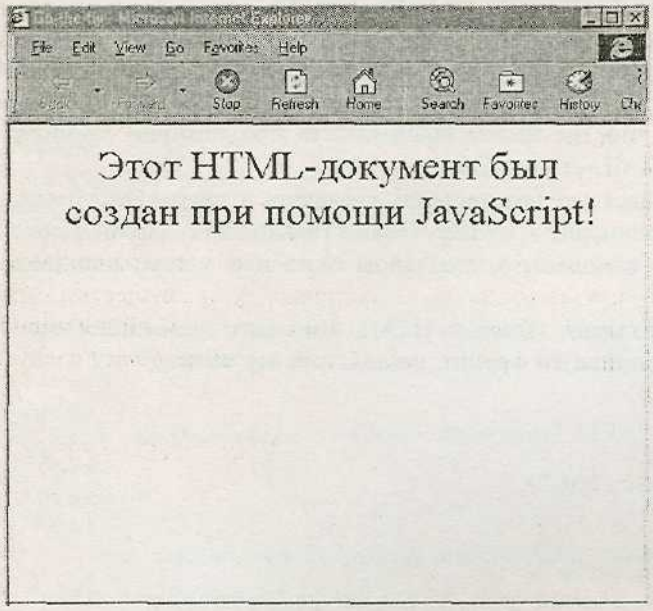
Создание документов, образующихся по ходу работы

Сейчас мы подошли к одному из замечательных свойств JavaScript — созданию документов, формируемых в процессе работы. В английском языке для таких документов используется термин on-the-fly документы. Это означает, что что мы имеем возможность при помощи JavaScript создавать новые HTML-страницы.

Более того, существует возможность создавать и другие документы, например, VRML-сценарии и прочее. Новые только что созданные документы могут быть выведены в отдельном окне или в том, или ином заданном фрейме.

Сначала мы создадим простой HTML-документ и покажем его в новом окне. Ниже приведен скрипт, решающий эту задачу.

```
<html>
<head>
<script language="JavaScript">
<!- hide
 function openWin3() {
  myWin= open("", "newWindow",
  "width=500,height=400,status=yes,toolbar=yes,menubar=yes");
  // открываем документ
  myWin.document.open();
  // создаем документ
  myWin.document.write("<html><head><title>On-the-fly");
  myWin.document.write("</title></headxbody>");
  myWin.document.write("<center><font  size=+3>");
  myWin.document.write("Этот HTML-документ создан ");
  myWin.document.write("при помощи JavaScript!");
  myWin.document.write("</font></center>");
  myWin.document.write("</body></html>");
  // закрываем документ (но не окно!)
  myWin.document.close();
 }
// ->
</script>
</head>
<body>
<form>
 <input type=button value="On-the-fly" onClick="openWin3()">
</form>
</body>
</html>
```



Этоокнопоявляетсяпринажатиикнопки"On-the-fly"

Рассмотрим функцию winOpen3(). Сначала мы открываем новое окно броузера. Первый аргументфункции— пустая строка "" . Это означает, что мы не указываем никакой URL. Броузер не долен загружать существующий документ, JavaScript создаст новый документ.

Мы определяем переменную *myWin.* При помощи этой переменной мы будем обращаться к новому окну. Здесь мы не можем использовать имя нового окна *{newWindow).* После того, как окно открыто, мы открываем документ при помощи

```
myWin.document.open ();
```

Мы вызываем метод open(), относящийся к объекту document. Это иной метод, отличающийся от метода open() для объекта window! Эта команда не открывает нового окна, она подготавливает документ для его последующего вывода. Перед *document.open()* следует указать *myWin* для того, чтобы обратиться к новому окну.

Далее следующие команды создают содержание нового документа.

```
// создаем документ
myWin . document.write ("<html><head><title>On-the-fly") ;
myWin.document.write ("</title></head><body>" ) ;
myWin.document.write ("<center><font size=+3>") ;
myWin.document.write("Этот HTML-документ создан ") ;
myWin.document.write("при помощи JavaScript!");
myWin.document. write ("</font></center>") ;
myWin.document.write("</body></html>");
```

Мы видим, что здесь записаны обычные флажки, используемые в HTML-документах. Таким образом мы создаем HTML-страничку. Здесь можно использовать любые флажки, допустимые в HTML. После окончания вывода необходимо закрыть документ. Это делается при помощи следующей инструкции.

```
// закрываем документ (но не окно!)
myWin.document.close() ;
```

Раньше мы упоминали, что создаваемые в процессе работы документы можно выводить и во фреймы. Если, например, у нас существует два фрейма с именами *frame1* и *frame2,* и мы хотим создать новый документ, который будет размещен во фрейме *frame2,* то во фрейме *frame1* следует написать следующее.

```
parent.frame2.document.open() ;
parent.frame2.document.write("Здесь мы пишем наш HTML код") ;
parent.frame2.document.close() ;
```

Создание VRML-документов в процессе работы

Для демонстрации удивительной гибкости языка JavaScript приведем пример создания документа VRML, формируемого в процессе работы. Аббревиатура VRML обозначает язык моделирования виртуальной реальности (Virtual Reality Modelling Language). При помощи этого языка создаются трехмерные образы. Для проверки работы данного примера необходимо иметь установленный VRML plug-in. Сам скрипт не осуществляет проверку того, установлены или нет необходимые плаг-ины.

Вот текст программы.

```
<html>
<head>
<script language="JavaScript">
<!- hide
 function vrmlScene() {
  vrml= open("", "newWindow",
  "width=500,height=400,status=yes,toolbar=yes,menubar=yes");
  // открываем документ
  vrml.document.open{"x-world/x-vrml");
  vr= vrml.document;
  // создаем VRML-сцену
  vr.writeln("#VRML V1.0 ascii");
  // свет
  vr.write("Separator { DirectionalLight { ");
  vr.write("direction 3 -1 -2.5 } ");
  // камера
  vr.write("PerspectiveCamera { position -8.6 2.1 5.6 ");
  vr.write("orientation -0.1352 -0.9831 -0.1233 1.1417 ");
  vr.write("focalDistance 10.84 } ");
 // куб
   vr.write("Separator { Material { diffuseColor 0 0 1 } ");
   vr.write("Transform { translation -2.4 .2 1 rotation 0 0.5 1 .9 } ");
  vr.write("Cube { } } }");
  // закрываем документ (но не окно!)
  vrml.document.close();
  }
 // ->
</script>
</head>
<body>
 <form>
 <input type=button value="VRML on-the-fly"
 onClick="vrmlScene()">
 </form>
</body>
</html>
```

Эта программа во многом подобна предыдущему примеру. Сначала мы открываем окно. Затем мы открываем документ, подготавливая его вывод. Взгляните на пример:

```
// открываем документ
vrml.document.open("x-world/x-vrml");
```

В предыдущем примере мы ничего не писали в скобках. Что значат эти *"x-world/x-vrml"* ? Это MIME-тип создаваемого файла. При помощи этой записи мы сообщаем броузеру о том, какой тип данных будет ему предложен далее. Если мы не укажем в скобках ничего, то по умолчанию предполагается тип "text/html", этот MIME-тип соответствует обычным HTML-файлам.

Для создания 3D сцены мы пишем *vrml.document.write()*.Это сравнительно длинная запись, поэтому удобно определить *vr= vrml.document*. Сейчас можно записать *vr.write()* вместо *vrml.document.write()*. Далее мы вводим VRML-код:

```
#VRML V1.0 ascii
Separator {
 DirectionalLight { direction 3 -1 -2.5 }
 PerspectiveCamera {
 position -8.6 2.1 5.6
 orientation -0.1352 -0.9831 -0.1233 1.1417
 focalDistance 10.84
 }
 Separator {
 Material {
  diffuseColor 0 0 1
 }
 Transform {
  translation -2.4.2 1
  rotation 0 0.5 1 .9
 }
 Cube {}
 }
}
```

Мы вводим этот код при помощи команд *document.write()*.

Конечно, вводить VRML-код из существующего ранее созданного файла в создаваемый в процессе работы документ довольно бессмысленное занятие. Гораздо более интересно, если на странице используются те или иные формы, когда пользователь может сам манипулировать объектами и изменять их.

5

СТРОКА СОСТОЯНИЯ И УПРАВЛЕНИЕ ВРЕМЕНЕМ

- *Строка СОСТОЯНИЙ*
- *Управление временем. Создание временных задержек*
- *Движущийся текст*

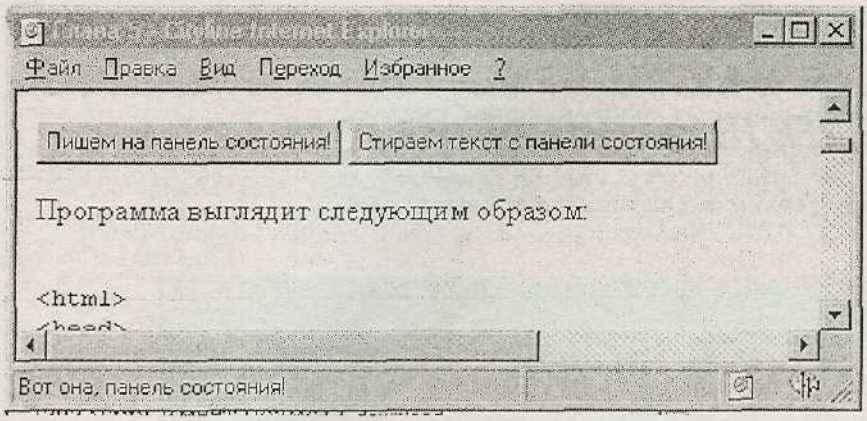
Строкасостояния

JavaScript имеет возможности, которые позволяют осуществлять вывод информации на панель состояния (мы будем пользоваться терминами "строка состояния" и "панель состояния" как синонимами, не делая между ними различия).

Панель состояния находится на нижней части рамки окна броузера. Для того, чтобы вывести текстовую строку на панель состояния необходимо присвоить *window.status* значение, равное этой строке. Нижеследующий пример осуществляет запись и уничтожение текста на панели состояния.

Программа выглядит следующим образом:

```
<html>
<head>
<script  language="JavaScript">
<!- hide
function statbar(txt)  {
 window.status = txt;
}
// ->
</script>
</head>
<body>
<form>
 <input type="button" name="look" value=Пишем на панель
состояния!"
 onClick="statbar(' Вот она, панель состояния!');">
 <input type="button" name="erase" value=Стираем текст с
панели состояния!"
 onCiick="statbar(' ') ; ">
</form>
</body>
</html>
```



При нажата кнопки "Пишем на панель состояния" в панели состояния, расположенной в нижней части окна, появляется текст: "Вот она, панель состояния!"

Здесь мы создаем форму с двумя кнопками. Обе кнопки приводят к обращению к функции statbar(). Кнопка "Пишем на панель состояния!" приводит к исполнению следующей функции:

```
statbar('Вот она, панель состояния!');
```

В скобках мы записали выражение 'Это и есть панель состояния'. Эта строка передается для обработки функции в виде параметра. Сама функция statbar() была определена нами следующим образом:

```
function statbar(txt) {
 window.status = txt;
}
```

Здесь мы поместили в скобки выражение *txt,* параметр функции. Это значит, что строчное выражение, которое будет передано для обработки функции, будет храниться в переменной с именем *txt.* Передача значений функции с помощью переменных позволяет использовать функции наиболее гибким способом.

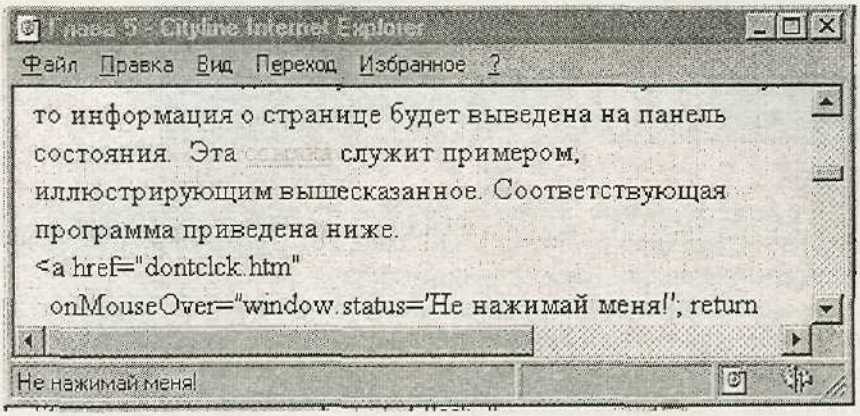
Функции можно передавать несколько переменных, для этого в списке переменные должны быть отделены друг от друга при помощи запятой. Значение переменной *txt,* т.е. строчное выражение, присвоенное этой переменной, можно вывести на панель состояния при помощи команды window.status = txt. Стирание текста с панели состояния осуществляется путем присвоения window.status пустой строки.

Использование возможности вывода текстовой информации на панель состояния полезно использовать в сочетании с гипертекстовыми ссылками. Вместо URL-адреса на панель состояния можно вывести краткую информацию о содержании страницы, на которую имеется ссылка.

Если подвести указатель мышки на такую ссылку, то информация о странице будет выведена на панель состояния.

Эта ссылка служит примером, иллюстрирующим вышесказанное. Соответствующая программа приведена ниже.

```
<a href="dontclck.htm"
 onMouseOver="window.status='Не нажимай меня!'; return true;"
 onMouseOut="window.status='';">ссылка</a>
```



На панель состояния выведен текст "Не нажимай меня!"

Для того, чтобы определить, где находится указатель мыши, мы использовали *onMouseOver* и *onMouseOut*. Сообщение "Не нажимай меня!" появляется, тогда, когда указатель мыши находится на ссылке. Возможно Вас удивило, что внутри скобок в выражении, присваиваемом свойству onMouse, указано *return true*. Эта инструкция заставляет броузер отказаться от выполнения собственных, свойственных броузеру действий в момент наступления данного события.

Обычно, когда указатель мыши находится на той или иной ссылке, броузер выводит на панель состояния адрес ссылки, ее URL. Если мы не напишем *return true,* то немедленно после того, как будет выполнена наша команда, т.е. на панель состояния будет выведен заданный нами текст "Не нажимай меня!", броузер вернется в свое стандартное состояние и запишет на панель состояния URL данной ссылки. Наша строка будет "затерта", и мы не сможем ее прочитать. Во избежание этого в средстве обработки событий мы пишем инструкцию *return true*.

Событие *onMouseOut* не существует в языке JavaScript 1.0. При использовании броузера Netscape Navigator 2.x результат действия, связанного с перемещением указателя мыши и его удалением со ссылки, будет зависеть от типа компьютерной платформы. Так, если Вы пользуетесь операционной системой Unix, выведенный на панель состояния, текст

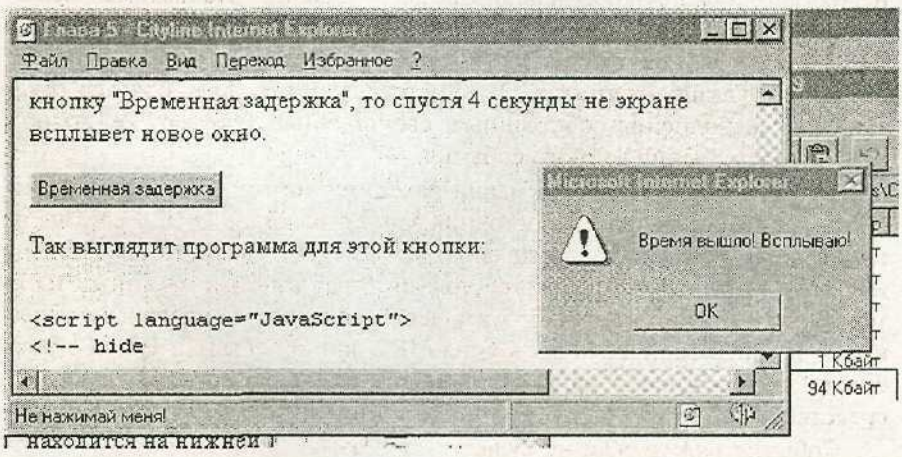
исчезнет даже в том случае, если броузер не понимает, как обрабатывается событие onMouseOut.

В системе Windows текст не исчезнет после перемещения указателя со ссылки. Однако существует возможность создать приемлемый скрипт, который будет понятен для Netscape Navigator 2.x в среде Windows. Для того можно, например, описать функцию, которая будет записывать текст на панель состояния и стирать его по прошествии некоторого времени.

Для создания такой функции потребуется использование средств управления временем. В следующей части мы познакомимся с возможностями управлять временем более подробно.

Управление временем. Создание временных задержек

Создав временную задержку, мы можем заставить компьютер выполнить те или иные действия лишь по истечении заданного промежутка времени. Ниже приведен пример. Если нажать на кнопку "Временная задержка", то спустя 4 секунды на экране всплывет новое окно.



Спустя четыре секунды после нажатия кнопки "Временная задержка" на экране появляется окно с текстом предупреждения : "Время вышло! Всплываю!"

Так выглядит программа для этой кнопки:

```
<script language="JavaScript">
<!- hide
 function timer() {
  setTimeout("alert('Время вышло! Всплываю!')", 4000);
 }
// ->
</script>
...
<form>
 <input type="button" value="Временная задержка"
onClick="timer ()">
</form>
```

Команда *setTimeout()* — это метод, относящийся к объекту window. При помощи данного метода происходит установка величины задержки, о чем нетрудно догадаться. Первым аргументом в выражении, присваеваемом данному методу, является код на языке JavaScript, который будет выполнен по истечении определенного промежутка времени.

В данном случае будет исполнена команда *"alert('Время вышло! Всплываю!')*. Заметьте, что команды JavaScript должны быть записаны в кавычках. Второй аргумент сообщает компьютеру продолжительность задержки — время, через которое необходимо выполнить указанные команды. Время указывается в миллисекундах (4000 миллисекунд = 4 секунды).

Движущийсятекст

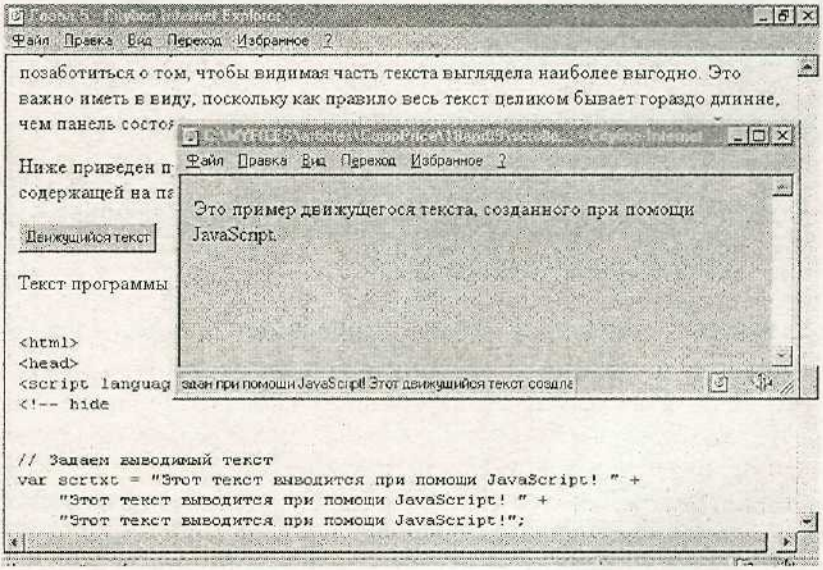
После того как мы научились записывать текст на панель состояния и создавать временные задержки, перейдем к рассмотрению движущихся текстов. Вы уже вероятно встречались с движущимся текстом на панели состояния. Его можно встретить на многих сайтах в Интернете. Мы познакомимся с тем, как можно запрограммировать элементарный вариант движущегося текста на панели состояния.

Кроме этого мы подумаем, как можно улучшить качество движущегося текста. Движущийся текст можно создать довольно просто . Представим себе, что нужно для создания движущегося текста на панели состояния. Сначала необходимо просто записать текст на панель состояния.

Затем, по истечении небольшого промежутка времени, нам необходимо записать этот текст, но слегка смещенный влево. Если мы будем повторять эту последовательность снова и снова, то создастся впечатление, что перед нами движущийся справа налево текст. Кроме того, нам необходимо позаботиться о том, чтобы видимая часть текста выглядела наиболее выгодно.

Это важно иметь ввиду, поскольку, как правило, весь текст целиком бывает гораздо длиннее, чем панель состояния, поэтому весь текст целиком не может поместиться на ней.

Ниже приведен пример. Нажатие кнопки приведет к загрузке новой страницы, содержащей на панели состояния движущийся текст.



После нажатия кнопки "Движущийся текст" в новом окне в панели состояния появляется движущаяся строка "Этот движущийся текст создан при помощи JavaScript!"

Текст программы выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!- hide
// Задаем выводимый текст
var scrtxt = "Этот текст выводится при помощи JavaScript! "
+
 "Этот текст выводится при помощи JavaScript! " +
 "Этот текст выводится при помощи JavaScript!";
var length = scrtxt.length;
var width = 100;
var pos = -(width + 2);
function scroll() {
 // выводим текст с правой позиции и устанавливаем временную
задержку
 // сдвигаем позицию на один шаг
 pos++;
 // вычисляем, какая часть текста будет видима
 var scroller = "";
 if (pos == length) {
 pos = -(width + 2);
 }
```

```
//  если текст не достиг пока левого края,
 // то необходимо добавить несколько пробелов — иначе
придется
 //отсекать переднюю часть текста
 if (pos < 0) {
 for (var i = 1; i <= Math.abs(pos); i++) {
  scroller = scroller + " ";}
 scroller = scroller + scrtxt.substring(0, width - i + 1);
 }
 else {
 scroller = scroller + scrtxt.substring(pos, width + pos);
 }
 // выводим текст на панель состояния
 window.status = scroller;
 // вновь обращаемся к функции спустя 100 миллисекунд
 setTimeout ("scroll()", 100);
}
//  ->
</script>
</head>
<body onLoad="scroll()">
Здесь располагается текст HTML-странички.
</body>
</html>
```

Основная часть функции scroll() осуществляет вычисления того, какая часть текста будет видима на панели состояния. Мы не будем детально задерживаться на объяснении того, как устроена данная часть программы. Для того, чтобы запустить движущийся текст, мы использовали средство управления событиями onLoad, указав его в ярлыке <body>. Следовательно, функция scroll() будет выполнена сразу же после загрузки страницы.

Мы обращаемся к функции scroll() посредством свойства onLoad. Производится первая часть расчетов строки текста и ее вывод на панель состояния. В конце описания функции scroll() мы определяем временную задержку. В результате задержки повторное обращение к функции scroll() произойдет лишь после указанного промежутка времени (100 миллисекунд). На очередном шаге текст смещается на один шаг влево, затем снова временная задержка. Это повторяется неограниченно долго.

Постоянно движущийся текст на панели состояния — это не лучшее решение. Такой текст скоро становится надоедливым. К тому же он мешает читать информацию о ссылках. Быть может лучшим было бы решение, когда одна часть текста движется справа налево, а другая часть — слева направо.

Затем в центре, натолкнувшись друг на друга, текст останавливается, некоторое время остается неподвижным, затем исчезает. При наличии фантазии движущийся текст может оказаться неплохим инструментом в арсенале создателя HTML-страничек.

6

СТАНДАРТНЫЕ ОБЪЕКТЫ JavaScript

- *Объект Date*
- *Объект массива Array*
- *Массивы и JavaScript 1.0*
- *Математический объект Math*

Объект Date

В языке существуют такие объекты, тип которых предусмотрен стандартом языка JavaScript. Под термином *стандартные объекты* мы понимаем такие объекты, которые в английском языке (языке оригинала, описывающего JavaScript) определяются словом *predefined,* т.е. объекты, которые были определены заранее.

Это специального типа объекты, существование которых обогащает язык JavaScript, придает ему новые свойства. Примером стандартных объектов может служить объект Data или Data-odject. Существуют и другие стандартные объекты, например, Array-object, Math-object.

Этим не исчерпывается весь список стандартных объектов. Обо всех типах стандартных объектов можно получить информацию, обратившись к справочнику по JavaScript.

Рассмотрим подробнее объект Date. Как подсказывает само название объекта (в переводе слово Data обозначает *дата* или *число месяца),* этот объект позволяет работать со временем и числами месяца. Можно, например, подсчитать, сколько дней осталось до дня рождения Вашего друга. Можно вставить в Ваш HTML-документ информацию о текущем времени.

Перейдем к примеру, в котором рассмотрим, как можно вывести на страницу реальное время. Первым делом нам необходимо создать новый объект Date. Для этой цели служит оператор *new.* Посмотрите на следующую строчку:

```
today=  new  Date()
```

При помощи этой команды мы создаем новый объект Date, который называется *today*. Если мы не укажем определенных значений времени и числа, то по умолчанию будет использовано текущее время. Т.е. во время выполнения команды today= new Date() новый объект Date с именем *today* будет иметь значение, равное текущему времени, соответствующему текущему времени на компьютере.

Объект Date может быть использован в сочетании с некоторыми методами. Нижеперечисленные методы (это не полный список методов, с которыми может работать объект Date) позволят использовать наш объект *today* более гибко: getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth(). При помощи этих методов можно получать информацию о текущем значении часов, минут, месяцев, а также устанавливать новые значения этих параметров.

Полный перечень всех методов, присущих объекту Date, можно найти в справочниках. При использовании объекта Date помните, что это все же не часы, которые меняют свои показания каждую секунду и миллисекунду, и эти изменения происходят автоматически. Объект Date позволяет получать значения времени и устанавливать новые значения времени лишь в момент выполнения команды.

Чтобы задать значения времени, мы можем использовать следующую команду (здесь метод Date() вызывается посредством оператора new):

```
today= new Date(2000, 3, 17, 15, 35, 23)
```

С помощью такой строчки мы создаем объект Date, который соответствует 15 часам 35 минутам 23 секундам 17 апреля 2000 года. Время в скобках записывается в следующей последовательности:

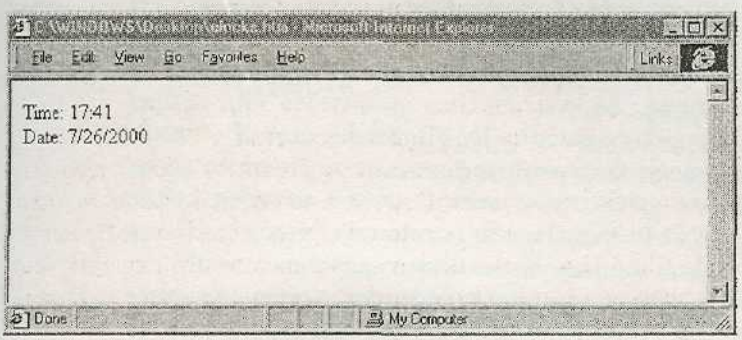
Date(год, месяц, день, часы, минуты, секунды).

Заметьте, что для обозначений месяцев используются числа, на единицу меньшие привычного номера месяца. Так для обозначения января используется 0, февралю соответствует 1 и т.д.

Сейчас мы напишем программу, которая будет выводить реальное текущее время. Вот она:

```
<script language="JavaScript">
<!- hide
now= new Date();
document.write("Time: " + now.getHours() + ":" +
now.getMinutes() + "<br>");
document.write("Date: " + (now.getMonth() + 1) + "/" +
now.getDate() + "/" +
     (1900 + now.getYear()));
// ->
</script>
```

Однако, такая программа не всегда работает корректно. Следует дополнительно проверить, в каком формате будет выведен текущий год.



Скрипт выводит на экран текущее время (но не идущие часы)

Здесь мы использовали getHours(), с помощью getHours() мы получили значение часов нашего нового объекта Date с именем new. Вы наверное заметили, что мы прибавляем 1900 к значению года, получаемого при помощи getYear(). Метод getYear(), возвращает значения года, начиная с 1900. Если это 1997 год, то getYear() возвратит величину 97, для 2005 года getYear() возвратит число 105.

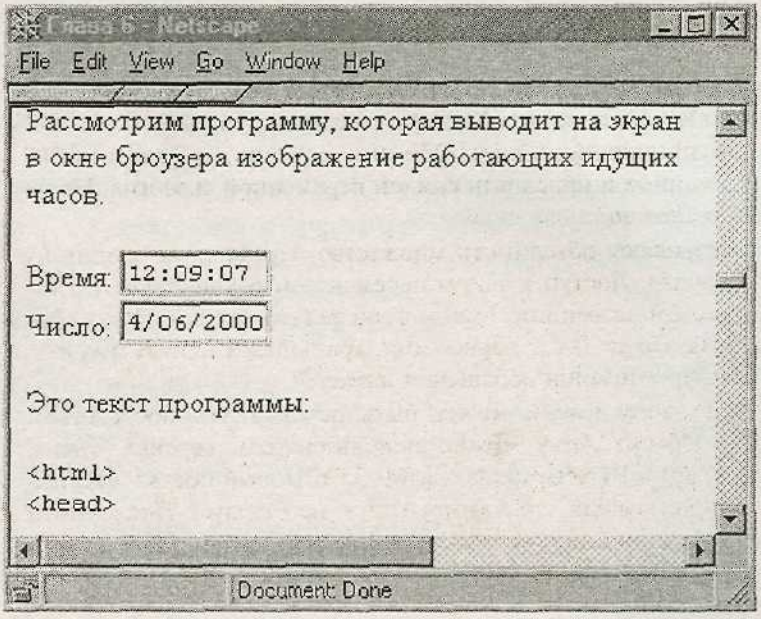
Полезно иметь ввиду, что при написании программ, работающих со временем, легко получить результат, когда минуты и секунды, представленные в строчной записи, не будут содержать незначимый нуль впереди значащей цифры, если значение окажется меньше десяти, т.е. время будет выглядеть примерно так: *14:3,* — тогда как следовало бы вывести *14:03.* В приведенной ниже программе этот момент учтен, и внесены необходимые исправления.

Рассмотрим программу, которая выводит на экран в окне броузера изображение работающих идущих часов.

Это текст программы:

```
<html>
<head>
<script Language="JavaScript">
<!- hide
var timeStr, dateStr;
function clock() {
  now= new Date() ;
  // время
  hours= now.getHours() ;
  minutes= now.getMinutes() ;
  seconds= now.getSeconds() ;
  timeStr= "" + hours;
  timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
  timeStr+= ( (seconds < 10) ? ":0" : ":") + seconds;
  document.clock.time.value = timeStr;
```

```
   // дата
   date= now.getDate() ;
   month= now. getMonth 0+1 ;
   year= 1900 + now.getYear();
   dateStr= "" + month;
   dateStr+= ((date < 10) ? "/0" : "/") + date;
   dateStr+= "/" + year;
   document.clock.date.value = dateStr;
   Timer= setTimeout ("clock () ", 1000);
}
// ->
</script>
</head>
<body onLoad="clock()">
<form name="clock">
 Время:
 <input type="text" name="time" size="8" value=""><br>
 Дата:
 <input type="text" name="date" size="8" value="">
</form>
</body>
</html>
```



В окне броузера видны идущие часы

Для того, чтобы получать значение текущего времени и даты каждую секунду, мы используем метод setTimeout(). Таким образом, каждую секунду мы создаем новый объект Data, в котором установлено текущее время.

Вы видите, что функция clock() вызывается при помощи средства управления событиями onLoad, которое использовании внутри ярлыка <body>. В теле нашей HTML-странички содержится два элемента для ввода текста. При помощи функции clock() в эти элементы производится запись времени и даты, причем учитывается особенность представления времени, когда указывается незначащий нуль на старшей позиции.

Мы использовали строчные переменные *timeStr* и *dateStr,* при помощи этих переменных мы сформатировали вывод времени в соответствии с требованиями. Для этого мы использовали следующую команду:

```
timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
```

При помощи этой команды мы дописываем число минут в строку *timeStr.* Если количество минут меньше 10, то мы добавляем 0. Эту строчку можно записать и в более удобном (привычном) виде:

```
if (minutes < 10) timeStr+= ":0" + minutes
  else timeStr+= ":" + minutes;
```

Объект массива Array

Массивы всегда бывают полезны. Представим пример, когда Вам нужно хранить список, состоящий из 300 различных имен. Как осуществить это при помощи JavaScript? Можно, конечно, определить 100 различных переменных и присвоить каждой переменной свое имя. Но не слишком ли это сложно?..

Массивы позволяют объединить множество переменных, связав их вместе друг с другом. Доступ к таким переменным осуществляется посредством одного единственного имени и числа (или чисел). Пусть наш массив носит имя *names.* Тогда первое имя, хранящееся в этом массиве, можно получить при помощи обращения *names[0].*

Доступ ко второму имени может быть осуществлен посредством *names[1]* и т.д. Объект Array можно использовать в версиях языка, начиная с JavaScript 1.1 (Netscape Navigator 3.0). Новый объект массива можно создать при помощи строчки myArray= new Array(). После этого массиву можно присваивать конкретные величины:

```
myArray[0]= 17;
myArray[1]= "Dima";
myArray[2]= "Petya";
```

Массивы в языке JavaScript являются весьма гибкими. Вам даже не придется заботиться о размере массива, так как размер массива устанавливается динамически. Так если мы пишем myArray[99]= "xyz", то величина массива автоматически устанавливается в размере 100 элементов. Массивы в JavaScript могут лишь увеличиваться в размерах, но не могут уменьшаться.

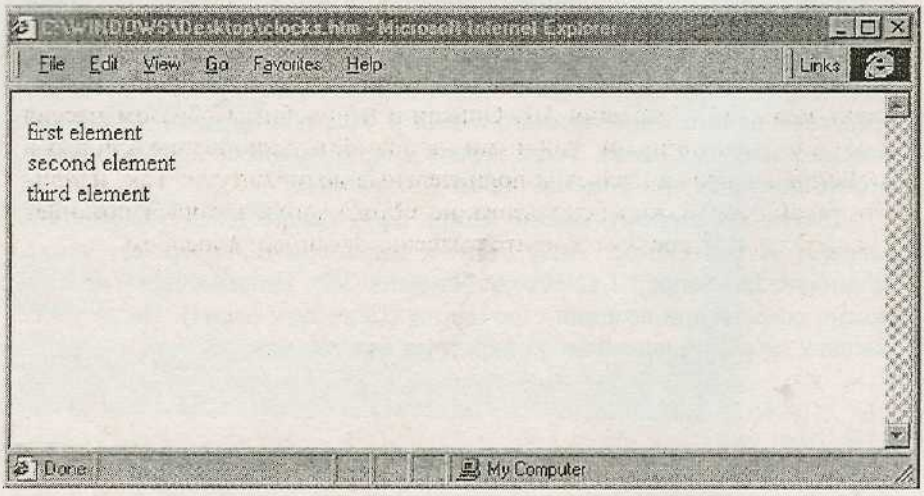
Поэтому необходимо следить за тем, что размерь: массивов были по возможности минимальными. В массивах можно хранить данные различных типов, например, числа, строки или другие объекты. Многое гораздо нагляднее видно на примерах.

Вот информация, которая выводится как результат работы программы, приведенной ниже:

```
first element
second element
third element
```

Здесь сама программа:

```
<script language="JavaScript">
<!- hide
myArray= new Array() ;
myArray[0]= "first element";
myArray[1]= "second element";
myArray[2]= "third element";
for (var i= 0; i< 3; i+.+) {
 document.write(myArray[i] + "<br>");
}
// ->
</script>
```



Приведенный выше скрипт выводит в окне броузера три строки

Сначала мы создаем новый массив с именем *myArray*. Затем мы присваиваем элементам массива различные значения. После присваивания мы организуем цикл. В этом цикле происходит исполнение команды: document.write(myArray[i] + "
");

Это команда исполняется три раза. Переменная i пробегает значения от 0 до 2. Внутри цикла мы обращаемся к элементам массива при помощи *myArray[i]*. Поскольку значение изменяется в пределах от 0 до 2, то мы три раза обращаемся к document.write():

```
document.write(myArray[0] + "<br>");
document.write{myArray[1] + "<br>");
document.write(myArray[2] + "<br>");
```

Массивы и JavaScript 1.0

Массивы не существуют в языке JavaScript 1.0 (Netscape Navigator 2.x и Microsoft Internet Explorer 3.x), поэтому необходимо позаботиться об альтернативных решениях для устаревших броузеров. В документации компании Netscape имеется следующий фрагмент программы, позволяющей создавать массивы:

```
function initArray() {
 this.length = initArray.arguments.length
 for (var i = 0; i < this.length; i++)
 this[i+1] = initArray.arguments[i]
}
```

Далее можно создавать массивы при помощи таких инструкций:

```
myArray= new initArray(17, 3, 5);
```

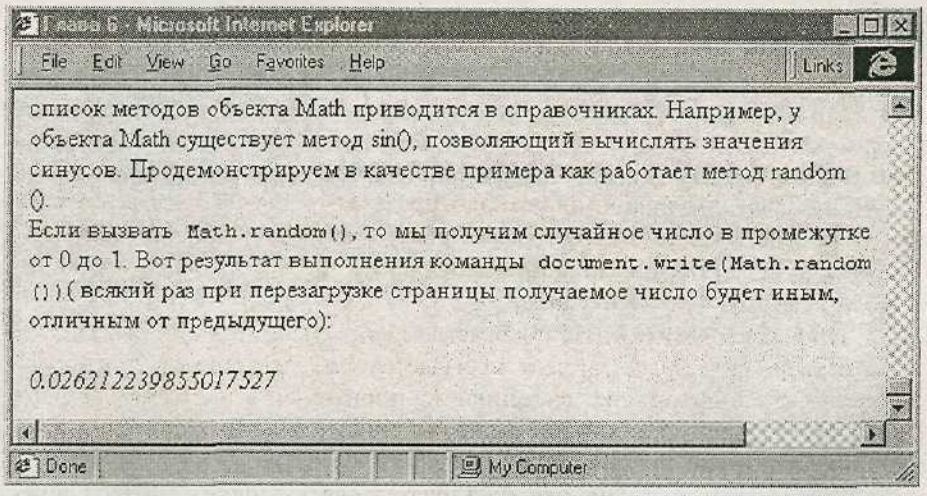
Числа в скобках — начальные параметры массива. Их также можно использовать и в JavaScript 1.1. Описанный подобным образом массив JavaScript 1.0 некоторыми удобными методами, использование которых в JavaScript 1.1 предоставляет дополнительные возможности. Так, например, здесь невозможно использование метода sort(), который помогает осуществить сортировку элементов массива заданным способом.

Математический объект Math

Если появляется необходимость производить математические вычисления, то математический объект Math предоставляет целый набор методов, позволяющих осуществить множество математических операций. Полный список методов объекта Math приводится в справочниках.

Например, у объекта Math существует метод sin(), позволяющий вычислять значения синусов. Продемонстрируем на примере, как работает метод random().

Если вызвать Math.random(), то мы получим случайное число в промежутке от 0 до 1. Вот результат выполнения команды document.write(Math.random()). Всякий раз при перезагрузке страницы получаемое число будет иным, отличным от предыдущего.



В конце HTML-документа помещена команда вывода случайного числа, которое записывается в окне броузера

7 ФОРМЫ

- *Проверка информации, вводимой посредством форм*
- *Проверка наличия в вводимой строке определенных символов*
- *Доставка информации, вводимой посредством форм*
- *Как обратить внимание пользователя на тот или иной элемент формы*

Проверка информации, вводимой посредством форм

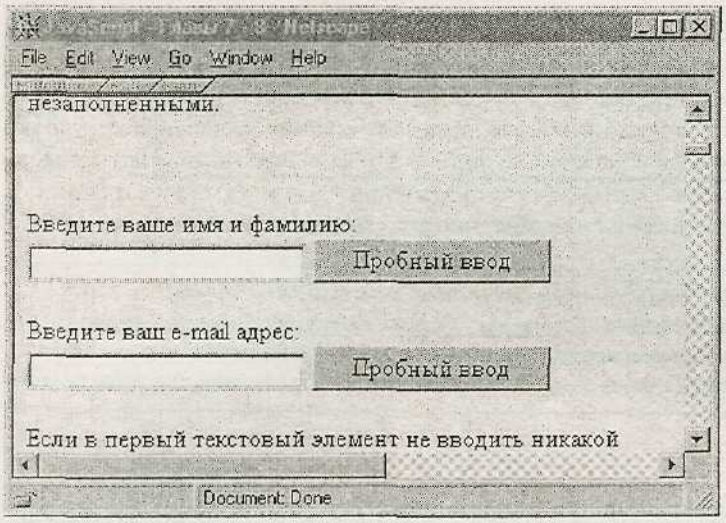
Формы — это бланки, в которых предусмотрены свободные места, в которые пользователь вводит необходимую информацию, это документы-бланки, предназначенные для заполнения. Формы в большом количестве используются в Интернете.

Информация, вводимая посредством форм, как правило, отсылается на сервер для обработки или посылается по электронной почте. Но прежде чем информация, введенная пользователем в пустые поля формы, будет передана по назначению, возникает задача о проверке правильности вводимой в форму информации.

Как убедиться в том, что пользователь ввел в форму подходящую информацию? При помощи языка JavaScript мы можем проверить правильность вводимой информации до того, как она будет передана по сети Интернет.

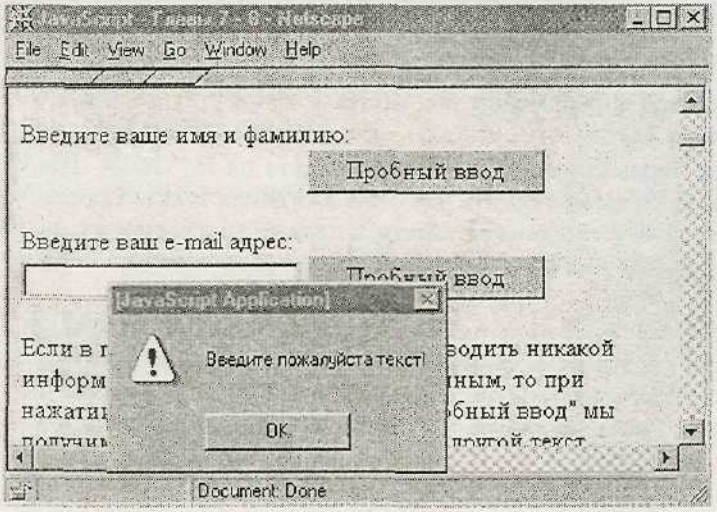
Рассмотрим первый пример простого скрипта на JavaScript. Наша HTML-страничка будет содержать два окна для ввода текста. Предполагается, что пользователь должен ввести в первый текстовый элемент информацию о своем имени и фамилии, а во второй элемент — свой e-mail адрес.

В действительности же пользователь имеет возможность ввести любую информацию в эти окна. Попробуем испытать нашу программу в работе. Поэкспериментируем с различными вариантами вводимой информации, в том числе попробуем оставить текстовые элементы незаполненными.

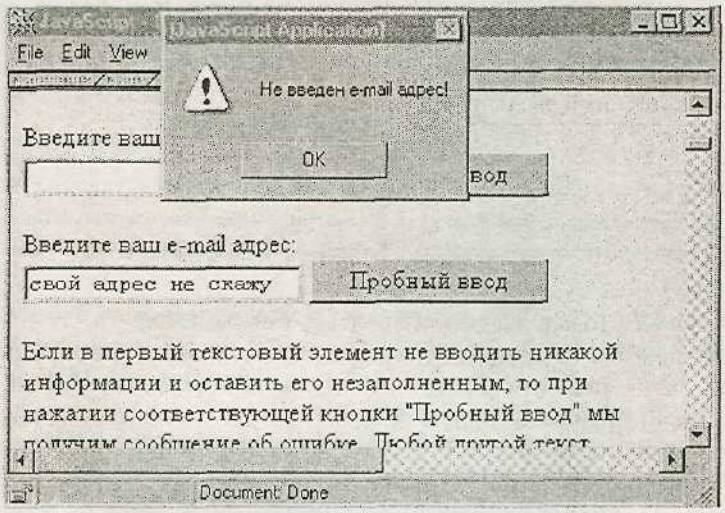


Если в первый текстовый элемент не вводить никакой информации и оставить его незаполненным, то при нажатии соответствующей кнопки "Пробный ввод" мы получим сообщение об ошибке. Любой другой текст, размещенный в первом текстовом элементе, будет воспринят как допустимый, сообщение об ошибке не появится. При этом, конечно, пользователь может ввести всякую постороннюю информацию, это не обязательно будет его именем.

Помимо буквенной информации можно вводить и цифры, и значки. Если пользователь введет число, скажем, "777", то в ответ после нажатия кнопки в данном примере он получит ответ "Привет, 777!". Поэтому предложенная в этой программе проверка может оказаться не слишком эффективной.



Пример со второй текстовой формой несколько более сложен и интересен. Попробуем ввести какую-либо простую строчку, например, имя, в котором не содержится значок @. Такой ввод будет воспринят как ошибочный. Критерием отбора является наличие в вводимой строчке значка @. Поскольку каждый интернет-адрес электронной почты содержит в себе значок @, то проверка наличия в данной последовательности символов представляется оправданной.



Программа данных текстовых элементов, в которой имеются некоторые элементы проверки, приведена ниже:

```
<html>
<head>
<script language="JavaScript">
<!- Hide
function test1 (form) {
 if (form.text1.value == "")
 alert("Введите пожалуйста текстовую информацию!")
 else {
 alert("Привет, "+form.text1.value+"! Form input ok!");
 }
}
function test2(form) {
 if (form.text2.value == "" ||
  form.text2.value.index0f('@', 0) == -1)
  alert("Вы не ввели верного e-mail адреса!");
 else alert("OK!");
}
// ->
</script>
</head>
```

```
<body>
<form name="first">
Введите ваше имя и фамилию:<br>
<input type="text" name="textl">
<input type="button" name="button1" value="Пробный ввод"
onClick="testl(this.form)">
<P>
Введите Ваш e-mail адрес:<br>
<input type="text" name="text2">
<input type="button" name="button2" value="Пробный ввод"
onClick="test2(this.form)">
</body>
</html>
```

Рассмотрим сначала ту часть программы, которая представлена в виде HTML-кода и размешена в теле HTML-страницы. Мы здесь создали два текстовых элемента и две кнопки. Кнопки обращаются к функциям testl(...) и test2(...).

Функция test1(form) осуществляет проверку того, является ли вводимая строчка пустой. Проверка осуществляется при помощи оператора *if (form.text1value == "")...* Переменной 'form' (в функции testl(form)) передается значение *this.form*. Значение (строка) текстового ввода может быть получено при использовании 'value' в сочетании с *form.text1*. Чтобы проверить, является ли ввод пустым, мы сравниваем это значение с пустой строчкой "". Если вводимая строка равна "", то ввод не был осуществлен. Программа генерирует сообщение об ошибке. Если какая-либо информация введена (вводимая строчка не пуста), то программа воспринимает ее как допустимую для ввода.

Единственной проблемой здесь является возможность ввода пробелов. Если пользователь введет пробел — то такой ввод будет воспринят программой как допустимый. Конечно, этот недостаток, при необходимости, можно исправить. Это легко можно сделать, если Вы познакомитесь со следующим примером, описывающим второй текстовый ввод в нашем примере.

Рассмотрим функцию *test2(form)*. Эта функция, как и в предыдущем случае, осуществляет проверку того, осуществлен ли ввод какой-либо информации в текстовый элемент, сравнивая его с пустой строкой "". Но далее мы добавили новые инструкции в команду if. Двойная вертикальная черта || — это оператор OR. Мы его уже использовали ранее (в предыдущей главе).

Оператор if проверяет, является ли хотя бы одна из двух частей (правая или левая части, объединенные оператором |) истинными. Если хотя бы одно из этих двух выражений истинно, то оператор if принимает истинное значение (true), и программа переходит к выполнению следующей команды. В данном случае это означает, что мы получим сообщение об ошибке в том случае, если либо вводимая строчка оказывается пустой, либо если в водимой строчке не содержится символа @. Вторая часть оператора if осуществляет проверку наличия символа @ в водимой строчке.

Проверка наличия в вводимой строке определенных символов

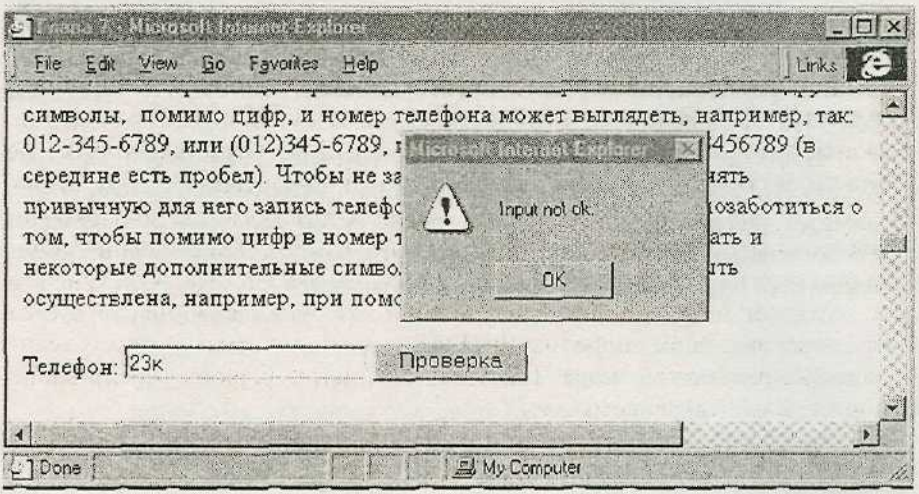
Может возникнуть необходимость проверки того, содержатся ли в вводимой строке те или иные символы. Представим, например, что предполагается, что вводимая информация представляет собой номер телефона. Это значит, что в строке должны содержаться только цифры (мы предполагаем, что номера телефонов содержат в себе лишь цифры).

Иногда же необходимо, чтобы вводимая информация содержала лишь ограниченный определенный набор символов. В примере с номером телефона мы будем проверять, являются ли вводимые символы цифрами. Однако некоторые люди при вводе номеров телефонов используют и другие символы, помимо цифр, и номер телефона может выглядеть, например, так: 012-345-6789, или (012)345-6789, или 012/345-67-89, или 012 3456789 (в середине есть пробел).

Чтобы не заставлять пользователя изменять привычную для него запись телефонного номера, мы должны позаботиться о том, чтобы помимо цифр в номер телефона можно было включать и некоторые дополнительные символы. Такая проверка может быть осуществлена, например, при помощи следующей программы:

```
<html>
<head>
<script language="JavaScript">
<!- hide
function check(input) {
 var ok = true;
 for (var i = 0; i < input.length; i++) {
 var chr = input.charAt(i);
 var found = false;
 for (var j = 1; j < check.length; j++) {
  if (chr == check[j]) found = true;
 }
 if (!found) ok = false;
 }
 return ok;
}
function test(input) {
 if (!check(input, "1", "2", "3", "4",
  "5", "6", "7", "8", "9", "0", "/", "-", " ")) {
 alert("В вводимой информации содержатся ошибки");
 }
 else {
 alert("Ввод принят.");
 }
}
// ->
</script>
</head>
```

```
<body>
<form>
Телефон:
<input type="text" name="телефон" value="">
<input type="button" value="Check"
 onClick="test{this.form.telephone.value)">
</form>
</body>
</html>
```



При помощи функции test() мы проверяем, являются ли вводимые символы допустимыми (т.е. содержатся ли они в списке допустимых символов).

Доставка информации, вводимой посредством форм

Какие существуют методы доставки вводимой пользователем посредством форма информации? Самый простой способ доставки информации из форм — электронная почта (e-mail). Этот метод далее мы рассмотрим немного более подробно.

Если же Вы захотите, чтобы введенная пользователем информация была обработана на сервере, то Вам придется использовать возможности CGI (common Gateway Interface). Пользуясь возможностями серверной обработки форм, мы имеем возможность обрабатывать поступающую информацию автоматически.

Эта информация может быть, например, занесена в базу данных, для этого не понадобится вмешательства человека, сервер может обновить

базу данных автоматически. Еще один пример того, как может обрабатываться информация из заполненных пользователем форм на сервере — это поисковая система{ например, Yahoo).

Пользователь вводит в форму искомый текст, нажимает кнопку и через несколько мгновений получает ответ сервера. Пользователю не приходится ждать пока человек, который отвечает за работу сервера рассмотрит его запрос и произведет поиск в базе данных. Это автоматически делает сервер. Однако отметим, что такие задачи не по силам JavaScript.

Мы также не сможет записать отзывы визитеров в гостевую книгу, расположенную на сервере при помощи JavaScript, поскольку JavaScript не способен осуществлять записи в файлы, расположенные на сервере. Это возможно осуществить лишь средствами CGI.

Но, конечно, возможность создания гостевой книги, тем не менее, существует, ее можно создать посредством отправки e-mail сообщений. Однако это возможно только тогда, когда предполагается, что Вы не будете получать более сотни-двух сообщений ежедневно.

Посылка формы посредством e-mail осуществляется при помощи программы-скрипта, написанного в рамках языка HTML. Нам не понадобится даже JavaScript. Конечно, JavaScript может оказать помощь в случае, если перед тем как будет послана заполненная форма, требуется проверка введенной информации. Еще необходимо отметить, что команда mailto работает не везде. Так Microsoft Internet Explorer версии 3.0 не обрабатывает такую команду.

```
<form method=post action="mailto:your.address@runs.here"
enctype="text/plain">
Вам понравилась эта страничка?
 <input name="choice" type="radio" value="1">Ни капельки.<br>
 <input name="choice" type="radio" value="2" CHECKED>Зря
потратил время.<br>
 <input name="choice" type="radio" value="3">Самая плохая
страничка в сети.<br>
 <input name="submit" type="submit" value="Send">
</form>
```

Свойство *enctype— "text/plain "* использовано для того, чтобы вводимый текст посылался в "чистом" виде, т.е. как простой текст, без вспомогательной информации (программы, описывающей саму страничку). Тогда ответы становится читать намного удобнее.

Если до того, как форма будет послана по сети, требуется проверка того, правильно ли осуществлен ввод информации в форму, то необходимо использовать средство обработки событий onSubmit. В теле формы (после ярлыка <form> необходимо обратиться к функции, осуществляющей про-

верку вводимой информации, эта функции должна выполняться при нажатии кнопки Submit). Программа может выглядеть следующим образом:

```
function validate() {
 // check if input ok
 // ...

 if (inputOK) return true
 else return false;
}

...

<form ... onSubmit="return validate()">

...
```

Если использовать эту программу, то форма не будет послана через Интернет, если вводимая информация не соответствует установленным требованиям.

Какобратитьвниманиепользователянатотилиинойэлемент формы

При помощи метода focus() мы имеем возможность сделать нашу форму более удобной для пользователя. Мы сможем подсказать пользователю, на какую форму следует обратить внимание первым делом. Или же мы сможем подсказать пользователю и обратить его внимание на тот элемент, в котором был введен неправильный текст или допущена ошибка.

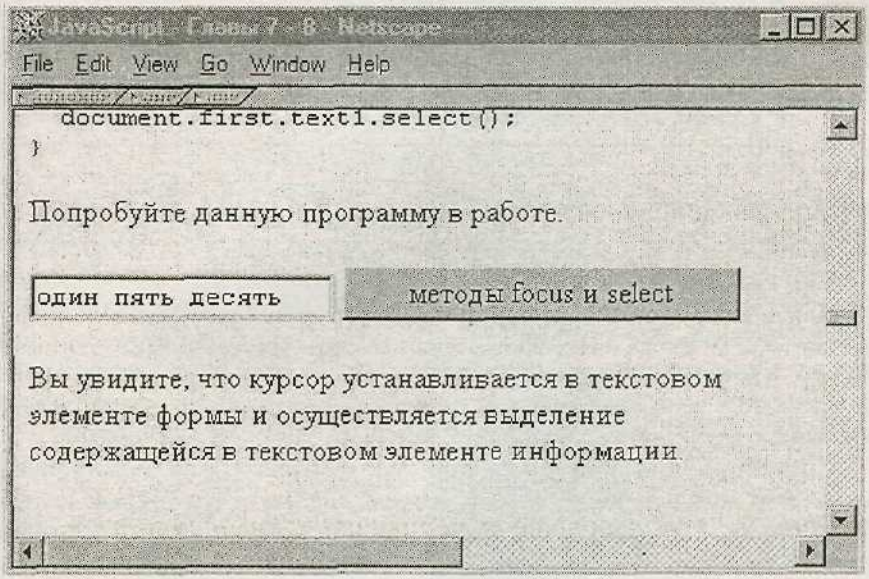
Это осуществляется при помощи того, что броузер устанавливает курсор в тот элемент, на который мы указываем посредством программы, использующей метод focus(). Метод focus() как бы фокусирует внимание пользователя на определенном элементе формы, на котором устанавливается курсор. Это осуществляется при помощи следующего фрагмента программы на JavaScript:

```
function setfocus() {
 document.first.textl.focus() ;
}
```

При помощи следующего фрагмента программы мы устанавливаем курсор в первом элементе выше написанного скрипта на JavaScript. Нам необходимо указать имя всей формы, которое в нашем случае представляет собой first, а затем указать имя текстового элемента данной формы — textl.

Для того, чтобы курсор устанавливался в текстовом элементе этой формы сразу после загрузки страницы, мы используем свойство onLoad внутри действия ярлыка <body>. Программа выглядит так:

```
<body  onLoad="setfocus()">
```



Мы можем слегка усложнить ее, определив функцию setfocus() следующим образом:

```
function  setfocus()  {
 document.first.text1.focus();
 document.first.text1.select();
}
```

Попробуйте данную программу в работе.

Вы увидите, что курсор устанавливается в текстовом элементе формы и осуществляется выделение содержащейся в текстовом элементе информации.

8

РИСУНКИ НА ВЕБ-САЙТЕ

- *Первые навыки создания рисунков*
- *Загрузка изображений на страничку*
- *Предварительная загрузка картинок, используемых на страничке*
- *Изменение картинок в ответ на действия пользователя*
- *Скрипт без недостатков*

Первыенавыкисозданиярисунков

В этой главе мы рассмотрим объект Image, рисунок. Этот объект понятен для броузеров, начиная с версий Netscape Navigator 3.0 и Microsoft Internet Explorer 3.0. Если Вы пользуетесь броузером более ранней версии, то приведенные в настоящей главе скрипты не будут работать в том объеме, в котором они изначально задуманы. Объект Image, появившийся в версии языка JavaScript 1.1, позволяет осуществлять эффективную обработку изображений, например, создавать эффекты анимации.

Сначала рассмотрим пример того, как к объекту Image можно обратиться при помощи JavaScript. Как мы уже видели в первой главе настоящего пособия, подобно многим другим объектам JavaScript, все дочерние изображения в пределах родительского объекта организуются в массив.

Массив изображений носит имя *images.* Изображения images — это свойства объекта document. Каждый документ на веб-странице имеет свой номер. Первое изображение имеет номер 0, второе изображение — номер один, и так далее. Таким образом, мы можем обратиться к рисунку посредством document.images[0].

Каждое изображение (картинка), содержащееся в HTML-документе, преставляет собой самостоятельный объект Image. Объект Image обладает набором своих свойств, к которым можно обратиться при помощи JavaScript. Например, можно узнать размер изображения, используя свойства *width* и *height.*

Так document.images[0].width содержит информацию о ширине первой картинки на страничке в пикселях, соответственно documet.images[0].height содержит высоту картинки. Иногда использование порядковых номеров картинок на странице оказывается неприемлемым. Это особенно сильно ощущается, если страница содержит большое количество изображений. Обратиться к изображению в таком случае помогают имена, данные конкретным изображениям. Имена изображений можно записать, например, используя следующий ярлык:

```
<img  src="img.gif"  name="ImageName"width=100  height=100>
```

Затем к этому изображению можно обращаться при помощи document.myName или document.images["ImageName"]. Здесь ImageName — это собственное имя данного рисунка.

Загрузка изображений на страничку

Мы познакомились с тем, как можно узнать размер изображения, используя свойства объекта Image. Однако это не есть самое интересное из того, чем располагает объект Image. Познакомимся с тем, как можно изменить уже загруженное изображение на новое при помощи свойства *src*.

В ярлыке <img> свойство src представляет адрес (URL) изображения, размещенного на страничке. При помощи JavaScript 1.1 мы имеем возможность присвоить свойству src новое значение адреса для уже существующего развернутого на странице изображения.

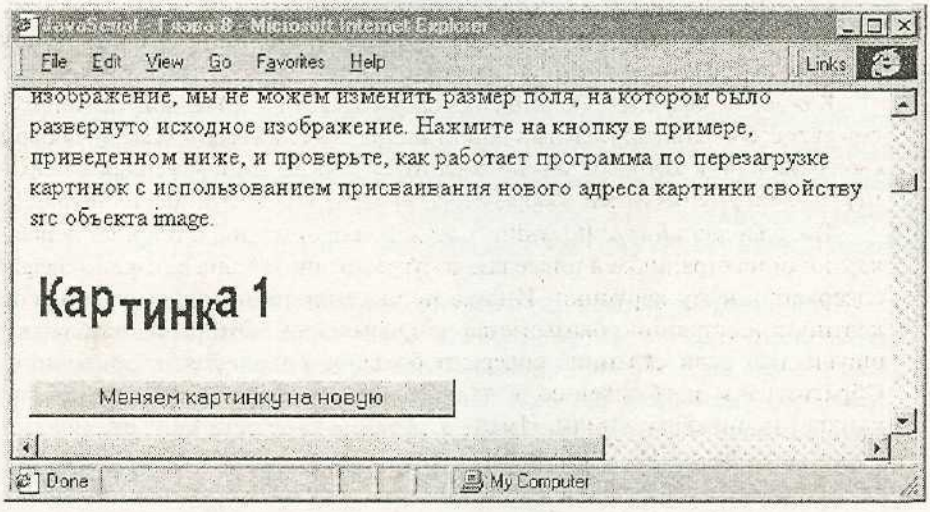
В результате данного присваивания на страницу будет загружено новое изображение, и старое изображение будет заменено новым. Посмотрите на этот пример:

<img src="img1.gif" name="ImageName" width=100 height=100>

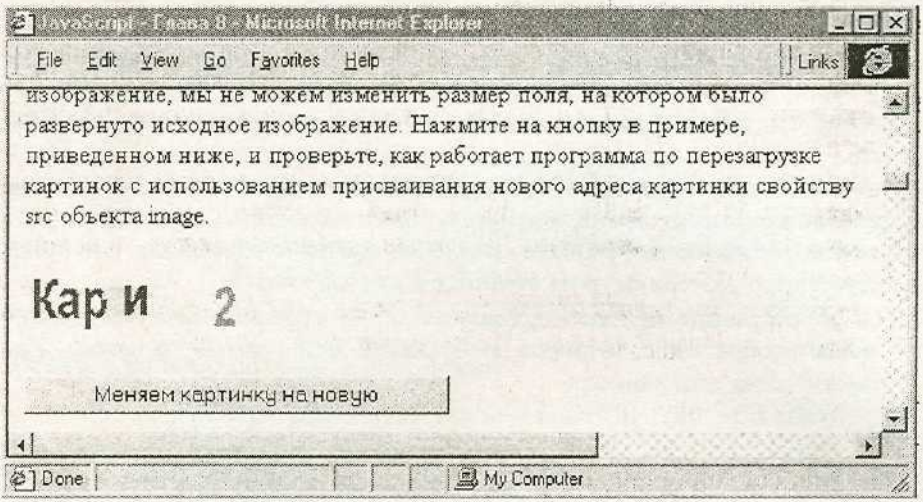
Здесь картинка img1.gif загружается на страничку, и имеет имя ImageName. Следующая строчка программы заменяет картинку img1.gif на новую картинку image2.gif.

document.ImageName.src= "img2.src";

Новая картинка будет иметь тот же размер, что и первоначально загруженное изображение, мы не можем изменить размер поля, на котором было развернуто исходное изображение. Нажмите на кнопку в примере, приведенном ниже, и проверьте, как работает программа по перезагрузке картинок с использованием присваивания нового адреса картинки свойству src объекта image.



Так ВЫГЛЯДИТ окно броузера до нажатия кнопки "Меняем картинку на новую"



После того, как кнопка нажата, картинка меняется на новую

Предварительная загрузка картинок, используемых на страничке

В приведенном выше примере есть одна тонкость. Дело в том, что новая картинка будет загружаться на компьютер лишь после того, как будет присвоен новый адрес картинки свойству src. Загрузка картинки из сети Интернет может занять некоторое время.

В некоторых ситуациях такая временная задержка может быть и приемлемой, однако не всегда допустимо ждать долгое время до окончания загрузки страницы. Как можно решить такую проблему? Решение существует!

Можно заранее загрузить рисунок, который потребуется при дальнейшей работе со страницей. Для этой цели мы создадим новый объект Image. Посмотрите на фрагмент программы, приведенный ниже:

```
ImgNew= new Image() ;
ImgNew.src= "img3.gif" ;
```

При помощи первой строчки мы создали новый объект Image и назвали его ImgNew. Вторая строчка задает адрес этого нового объекта. Как мы уже видели ранее, присваивание адреса свойству src приводит к тому, что указанный рисунок загружается из сети. При выполнении второй строчки броузер приступит к загрузке картинки img2.gif.

Поскольку картинка с именем ImgNew нигде на HTML-страничке не используется, то картинка ImgNew останется скрытой от пользователя, она будет загружена из сети и останется где-то в памяти компьютера, откуда она быстро может быть извлечена и помещена на страничку. Для

того, чтобы показать этот рисунок на страничке, мы можем использовать такую строчку:

```
document.myImage.src= hiddenImg.src;
```

После исполнения этой команды изображение будет извлечено из памяти и тотчас показано на страничке. Таким образом мы можем использовать картинки, которые были заранее загружены на компьютер.

Конечно, для того, чтобы рисунок, который должен быть заранее загружен на компьютер, смог появиться в окне броузера без задержки, этот рисунок должен быть полностью загружен на компьютер. Лишь тогда не возникнет никакой задержки при его раскрытии на экране.

Поэтому, если картинка велика по размеру или на странице содержится большое число заранее загружаемых картинок, может все же возникнуть нежелательная задержка ввиду того, что броузер будет занят работой по загрузке изображений из сети. Поэтому при работе с изображениями всегда следует иметь ввиду ограничения скорости загрузки информации из сети Интернет.

Используемый метод заблаговременной загрузки изображений ускоряет процесс развертывания изображения на экране компьютера, но он ни в коей мере не ускоряет процесс загрузки изображения из сети. При заблаговременной загрузке мы лишь ускоряем начало загрузки изображений, мы заставляем броузер приступить к загрузке нужных в дальнейшем изображений как можно скорее, не дожидаясь непосредственного обращения к этим изображениям.

Чем скорее мы начнем загрузку изображений, тем скорее мы будем иметь на своем компьютере готовые для развертывания в окне броузера изображения. При этом весь процесс работы происходит более равномерно и более удобно для пользователя.

Конечно, если Вы пользуетесь быстрым соединением с сетью Интернет, то все эти рассуждения могут показаться неубедительными. "И о каких таких задержках Вы тут рассуждаете?"- воскликнут они. Но если учесть качество многих российских телефонных станций и линий связи, то даже при наличии хороших модемов со скоростью 56 килобит, проблема оптимизации загрузки изображений может оказаться актуальной.

Изменение картинок в ответ на действия пользователя

Мы имеем возможность создавать интересные эффекты перемены изображений в ответ на те или иные действия пользователя. Например, мы можем изменить изображение на новое, если указатель мыши будет находиться над той или иной областью изображения. В следующем примере изменение изображения происходит тогда, когда указатель мыши находится над изображением, т.е. тогда, когда пользователь перемещает курсор по изображению.

Если же Вы пользуетесь старым броузером (JavaScript 1.0), то на экран будет выдано сообщение об ошибке. Этого, однако, можно избежать, далее мы на этом остановимся немного подробнее.

Так выглядит картинка, если указатель мыши находится за пределами изображений

Так выглядит картинка, когда указатель мыши расположен непосредственно на изображении

Вот соответствующая программа:

```
<a href="#"
  onMouseOver="document.myImage2.src='img2.gif'"
  onMouseOut="document.myImage2.src='img1.gif'">
<img src="img1.gif" name="myImage2" width=160 height=50
border=0></a>
```

Эта программа не лишена некоторых недостатков:

- пользователь может пользоваться старой версией броузера, программа в этом случае будет работать неверно;
- вторая картинка не будет заранее загружена на компьютер;
- для каждой картинки на страничке придется писать весь этот фрагмент программы заново;
- хотелось бы иметь такую программу, которую можно использовать на различных веб-страничках по много раз, причем сама программа при этом не претерпевала бы значительных изменений.

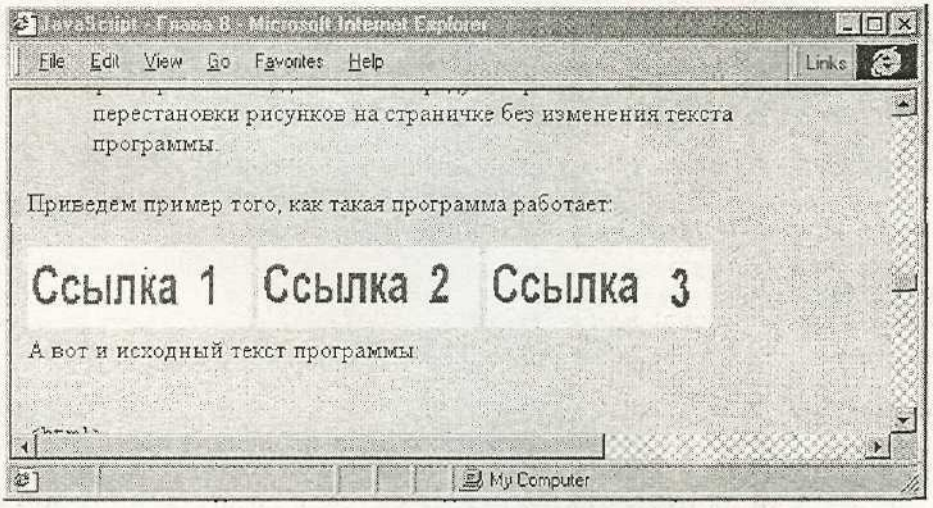
Скрипт без недостатков

Сейчас мы перейдем к рассмотрению известного скрипта, в котором отсутствуют все перечисленные выше недостатки. Эта программа оказывается несколько больше предыдущей и требует большее количество времени для ее написания. Но временные затраты и увеличение размера программы оправдываются сполна, поскольку после того, как эта программа написана, нам не придется больше возвращаться к старым проблемам снова и снова.

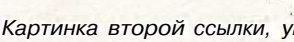
При создании этой программы учитывались следующие требования, без выполнения которых программа не обладала бы достаточной гибкостью.

- Количество используемых на странице рисунков не должно быть раз и навсегда фиксированным, безразлично, будет ли на страничке всего 20 рисунков, или же на страничку вмещается сотня изображений.
- Порядок следования рисунков на страничке также не должен быть фиксированным, должна быть предусмотрена возможность перестановки рисунков на страничке без изменения текста программы.

Приведем пример того, как исправленная программа работает:



При помещении указателя мыши на любую ссылку (представленную картинкой), соответствующая ссылке картинка меняется



Картинка второй ссылки, указатель мыши отсутствует

Так выглядит картинка второй ссылки, когда указатель мыши расположен на ней (фактически это другая картинка)

А вот и исходный текст программы:

```
<html>
<head>
<script language="JavaScript">
<!- hide
 // переменная для проверки номера версии броузера/языка
JavaScript
 var browserOK" = false;
 var pics;
// ->
</script>
<script language="JavaScript1.1">
<!- hide
 // переменная проверки версии принимает истинное значение,
 // если броузер поддерживает JavaScript 1.1
 browserOK = true;
 pics = new Array() ;
// ->
</script>
<script language="JavaScript">
<!- hide
var objCount = 0; // количество изображений на странице
function preload(name, first, second) {
 // предварительно загружаем все рисунки, используемые на
страничке, сохраняя их в виде массива
 if (browserOK) {
 pics[objCount] = new Array(3);
 pics[objCount][0]= new Image ();
 pics[objCount][0].src = first;
 pics[objCount][1]= new Image ();
 pics[objCount][1].src = second;
 pics[objCount][2] = name;
 objCount++;
 }
}
function on(name){
 if (browserOK) {
  for (i = 0 ; i < objCount; i++) {
  if (document.images[pics[i] [2] ] != null)
  if (name != pics[i][2]) {
   // все прочие картинки не показываем
   document.images[pics[i][2]].src = pics[i][0].src;
  } else {
   // вторую картинку показываем, когда по ней передвигается
курсор
   document.images [pics[i][2]].src = pics[i][1].src;
  }
 }
 }
```

```
}

function off () {
 if (browserOK) {
  for (i = 0; i < objCount; i++) {
  // все прочие картинки скрываем
  if (document.images[pics[i][2]] != null)
  document.images[pics[i] [2]].src = pics[i][0].src;
 }
 }
}

// предварительная загрузка картинок, необходимо указать,
какие картинки должны быть загружены,
// а также также какому объекту Image на страничке они
соответствуют
// (первый аргумент функции).
preload("link1", "img1f.gif", "img1t.gif");
preload("link2", "img2f.gif", "img2t.gif") ;
preload("link3", "img3f.gif", "img3t.gif");
// ->
</script>
<head>
<body>
<a href="link1.htm" onMouseOver="on ( 'link1') "
 onMouseOut="off ()">
<img name="link1" src="link1f.gif"
 width="140" height="50" border="0"></a>
<a href="link2.htm" onMouseOver "on('link2' ) "
 onMouseOut="off() ">
<img name="link2" src="link2f.gif"
 width="140" height="50" border="0"></a>
<a href="link3.htm" onMouseOver="on('link3' ) "
 onMouseOut="off() ">
<img name="link3" src="link3f.gif"
 width=" 140" height="50" border="0"></a>
</body>
</html>
```

Этот скрипт располагает все картинки в массиве *pics*. Этот массив создается при помощи обращений к функции preload() (см. начало программы). Мы видели, что обращение к функции preload() можно осуществить таким образом:

```
preload("link1", "img1f.gif", "img1t.gif");
```

Смысл данной строчки состоит в том, что происходит загрузка двух изображений *img1f.gif* и *img1t.gif*. Первая картинка будет демонстрироваться на страничке тогда, когда указатель мыши не будет располагаться

над полем картинки. Если же курсор находится над полем картинки, на страничке будет представлено второе изображение.

Первый аргумент функции *"img1"* при обращении к функции preload() — это имя объекта Image на данной страничке, к которому относятся эти два загружаемые изображения. В теле странички, за ярлыком <body> мы встретим рисунок с именем img1. Мы используем имя изображения, а не его порядковый номер. Это делается для того, чтобы оставалась возможность изменить порядок следования картинок на страничке и при этом не менять текст программы.

Функции on() и off() вызываются путем обращения к ним внутри средств обработки событий onMouseOver и onMouseOut. Поскольку изображения сами по себе не могут реагировать на события MouseOver и MouseOut, то мы использовали ссылки, поместив изображения внутри ссылок.

Из приведенной программы мы видим, что функция on() "убирает" все картинки, за исключением одной. Это делать необходимо, потому что может создаться такая ситуация, когда выделенными (как если бы на них находился указатель мыши) окажутся несколько картинок, так как требуемое для снятия выделения событие MouseOut может не наступить. Например, такая ситуация возможна, если курсор был передвинут с изображения сразу за пределы окна броузера.

Использование картинок — это мощный механизм обогащения веб-страниц. Объекты Image позволяют создавать изысканные эффекты. Но не надо забывать, что не каждая картинка и не любая программа на JavaScript подойдет именно для Вашей странички. В Интернете можно встретить множество примеров того, как изображения используется на страничках не самым лучшим, а порой и просто ужасным способом.

Новое качество страничке придает не количество изображений, а их качество. А навязчивое использование больших картинок, требующих временных затрат для их загрузки при сомнительном качестве изображений, делает просмотр некоторых сайтов в сети просто раздражающим занятием. Если это иметь в виду при создании веб-страниц, то качество их несомненно будет выше, а посетители Вашего сайта будут возвращаться к Вашим страничкам чаще.

9

СЛОИ

- *Что такое слои*
- *Как создать слои*
- *Слои и JavaScript*
- *Перемещение слоев в окне*

Чтотакоеслои

Слои обозначаются английским словом *layers*. Слои появились с возникновением версии 4.0 броузера Netscape Navigator. При помощи слоев стало возможным позиционирование элементов веб-страницы при помощи задания абсолютных координат слоя или координат относительных, задающих положение слоев друг относительно друга.

Использование координат слоев предоставляет возможность перемещения объектов, расположенных на данном слое, по HTML-странице. Возможно установить частичную видимость тех или иных объектов на слое, использовать возможности языка JavaScript для манипулирования объектами слоев.

Чтобы понять, что представляют собой слои, приведем пример. Возьмем несколько листов чистой бумаги. На одном листе мы напишем текст, на другом мы нарисуем картинку, на третьем мы снова напишем какой-нибудь текст, и так далее.

Расположим наши листы бумаги на столе. Каждый лист бумаги — это слой. Это своеобразный носитель той информации, которую мы в этот слой вложили — на листах мы написали текст, нарисовали рисунок, эта информация содержится на наших листах, на слоях. Слои содержат объекты.

Возьмем лист с рисунком и будем передвигать его по столу. Рисунок передвигается вместе с листом, повторяя все его движения. Точно так же происходит и со слоем на веб-страничке. Слой может содержать несколько объектов, например, рисунков, форм, текстов и так далее, все эти объекты могут быть расположены на HTML-страничке и могут передвигаться по ней вместе со слоем.

Слой — это носитель этих объектов. Если слой передвинуть, то все объекты данного слоя переместятся вместе со слоем. Слои могут накладываться друг на друга точно так же, как можно наложить друг на друга листы бумаги на столе. Каждый слой может иметь прозрачные части.

Если вырезать на листе бумаги круг, то через получившееся отверстие можно видеть часть другого листа, расположенного под тем, на котором вырезан круг. Круглая дырка — это прозрачная часть нашего слоя. Через прозрачные части слоя видно содержание того слоя, который расположен непосредственно за данным слоем.

Как создать слои

Для того, чтобы создать слой, используется ярлык <layer> или ярлык <ilayer>. Слои могут обладать следующими свойствами:

name="*layerName*" ..... Название слоя

left=*xPosition* ..... Горизонтальная (x) координата верхнего левого угла слоя

top=*yPosition* ..... Вертикальная (y) координата верхнего левого угла слоя

z-index=*layerIndex* ..... Номер слоя, его "индекс глубины"

width=*layerWidth* ..... Ширина слоя в пикселях

clip="*x1, y1, x2, y2*" ..... Данные координаты определяют расположение видимой области слоя

above="*layerName*" ..... Это свойство определяет имя слоя, над которым будет расположен данный слой

below="*layerName*" ..... Это свойство определяет имя слоя, под которым будет располагаться данный слой

Visibility=show|hide|inherit ..... Свойство задает параметр видимости слоя

bgcolor="*rgbColor*" ..... Свойство задает цвет фона слоя, это либо название стандартного цвета, либо значение rgb цвета

background="*imageURL*" ..... Свойство задает рисунок фона слоя

Ярлык <layer> используется для создания слоев, положение которых задается в явном виде посредством свойств *left* и *top*. Если положение слоя на страничке не указано, то слой будет расположен в левом верхнем углу окна странички.

Ярлык <ilayer> используется при создании слоя, который располагается в соответствии с условиями формирования HTML-документа по ходу его создания.

Начнем наше знакомство со слоями с простого примера. Создадим два слоя, на первом слое поместим рисунок, а на втором слое — текст. Мы хотим показать текст поверх рисунка.

На этом рисунке мы видим, что текст располагается поверх рисунка

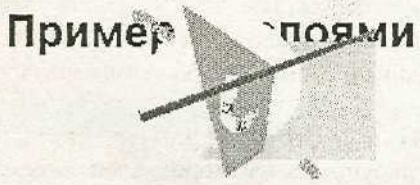
Исходный текст программы таков:

```
<html>
 <layer name=pic z-index=0 left=200 top=100>
  <img src="img.gif" width=160 height=120>
 </layer>
 <layer name=txt z-index=1 left=200 top=100>
  <font size=+4> <i> Пример со слоями </i> </font>
 </layer>
</html>
```

С помощью ярлыков <layer> мы определили два слоя. Оба эти слоя расположены в точке 200/100 (координаты в пикселях соответствуют положению верхнего правого угла слоя). Координаты слоя мы задали с помощью свойств left и top. Все, что располагается между ярлыками <layer> и </layer> (или между ярлыками <ilayer> и </ilayer>), принадлежит этому слою.

Мы также использовали свойство z-index. При помощи этого свойства мы установили порядок следования слоев или порядок их наложения друг на друга. Слой с самым большим значением z-index'a будет находиться на самом верху. Для значений z-index'a не обязательно использовать последовательные числа.

Достаточно того, чтобы эти числа были целыми, порядок наложения слоев друг на друга определяется этими числами, чем больше значение z-index, тем "выше" расположен слой. Если, например, мы написали бы в ярлыке <layer> в первом слое выражение z-index=30, то этот слой был бы показан поверх другого, т.е. текст был бы расположен за рисунком, поскольку слой, содержащий текст, имел бы тогда меньшее значение z-index'a.



На этом рисунке текст расположен позади картинки.

Слои и JavaScript

Сейчас мы узнаем, как можно управлять слоями при помощи JavaScript. И опять мы начнем с рассмотрения примера, в котором пользователь имеет возможность скрыть или показать слой путем нажатия кнопки. Вспомним, что объекты в JavaScript могут быть представлены несколькими способами, и слои здесь не являются исключением.

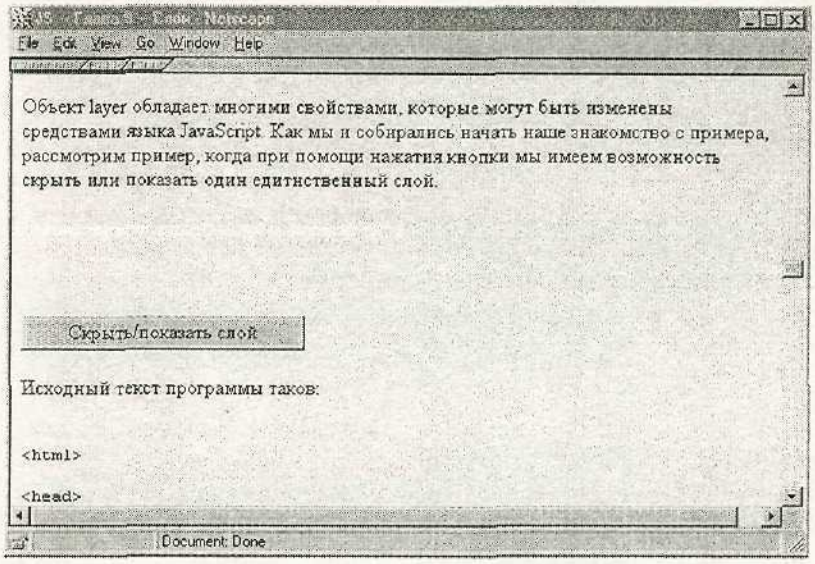
Возможно наилучшим способом обращения к слоям является присвоение каждому слою его имени и обращение к слою по его имени. Если мы зададим имя слоя при помощи свойства name, указав его в ярлыке <layer> следующим образом:

```
<layer ... name=myLayer>
...
</layer>,
```

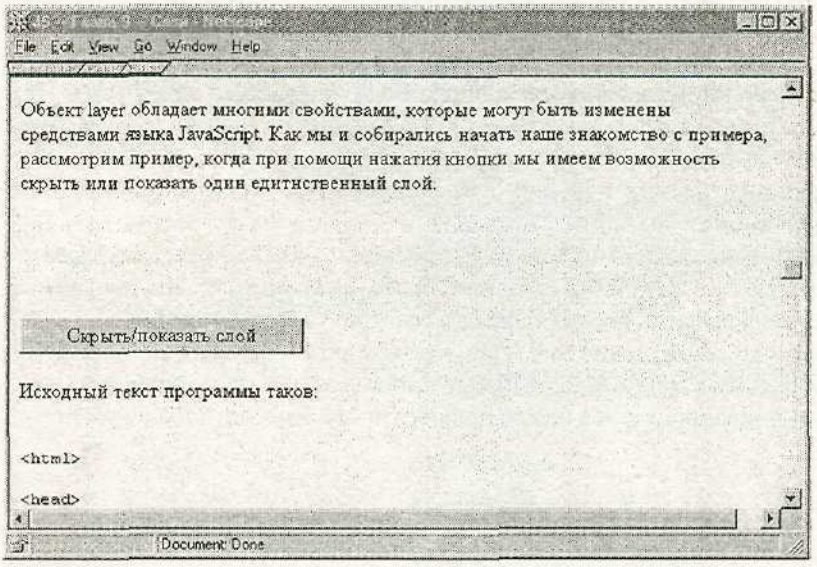
то мы сможем в дальнейшем обратиться к слою посредством выражения *document.layers["myLayer"]* или выражения *document.myLayer.* Конечно остается возможность обращения к слоям посредством числовых индексов, самый нижний слой при этом может быть описан при помощи выражения *document.layers[0].* Не следует путать числовой индекс слоя в выражениях только что приведенного выше типа с z-index'ом слоя. Это две разные, независимые друг от друга вещи.

Так, например, у нас есть два слоя, названные, скажем, layer1 и layer2, причем layer1 имеет z-index, равный 11, а layer2 имеет z-index, равный 20. Обращаться к этим слоям надо при помощи выражений *document.layers[0]* и *document.layers[1],* но никак не при помощи *document.layers[11]*и*document.layers[20].*

Объект layer обладает многими свойствами, которые могут быть изменены средствами языка JavaScript. Как мы и собирались начать наше знакомство с примера, рассмотрим вариант, когда при помощи нажатия кнопки мы имеем возможность скрыть или показать один единственный слой.



При загрузке HTML-документа строка "Этот текст расположен внутри слоя" видна



После нажатия кнопок "Скрыть/показать слой" строка исчезает. Повторное нажатие кнопки заставляет слой вновь стать видимым

Исходный текст программы таков:

```
<html>
<head>
<script language="JavaScript">
<!— hide
function showHide() {
 if (document.layers["myLayer"].visibility == "show")
 document.layers["myLayer"].visibility="hide"
 else document.layers["myLayer"].visibility= "show";
}
// ->
</script>
</head>
<body>
<ilayer name=myLayer visibility=show>
<font size=+l color="#0000ff"><i> Этот текст расположен
внутри слоя</i></font>
</ilayer>
<form>
<input type="button" value="Скрыть/показать слой"
onClick="showHide()">
</form>
</body>
</html>
```

Нажатие кнопки приводит к обращению к функции *showHide()*. При помощи этой функции мы обращаемся к свойству *visibility* объекта layer с именем *myLayer*. Путем присваивания значений *"show"or "hide"*выражению *document.layers["myLayer"].visibility* мы имеем возможность прятать или показывать этот слой.

Выражения *"show"* и *"hide"*— это строки, они не являются зарезервированными в JavaScript словами, а это значит, что мы не можем написать выражение *document.layers["myLayer"].visibility= show*.

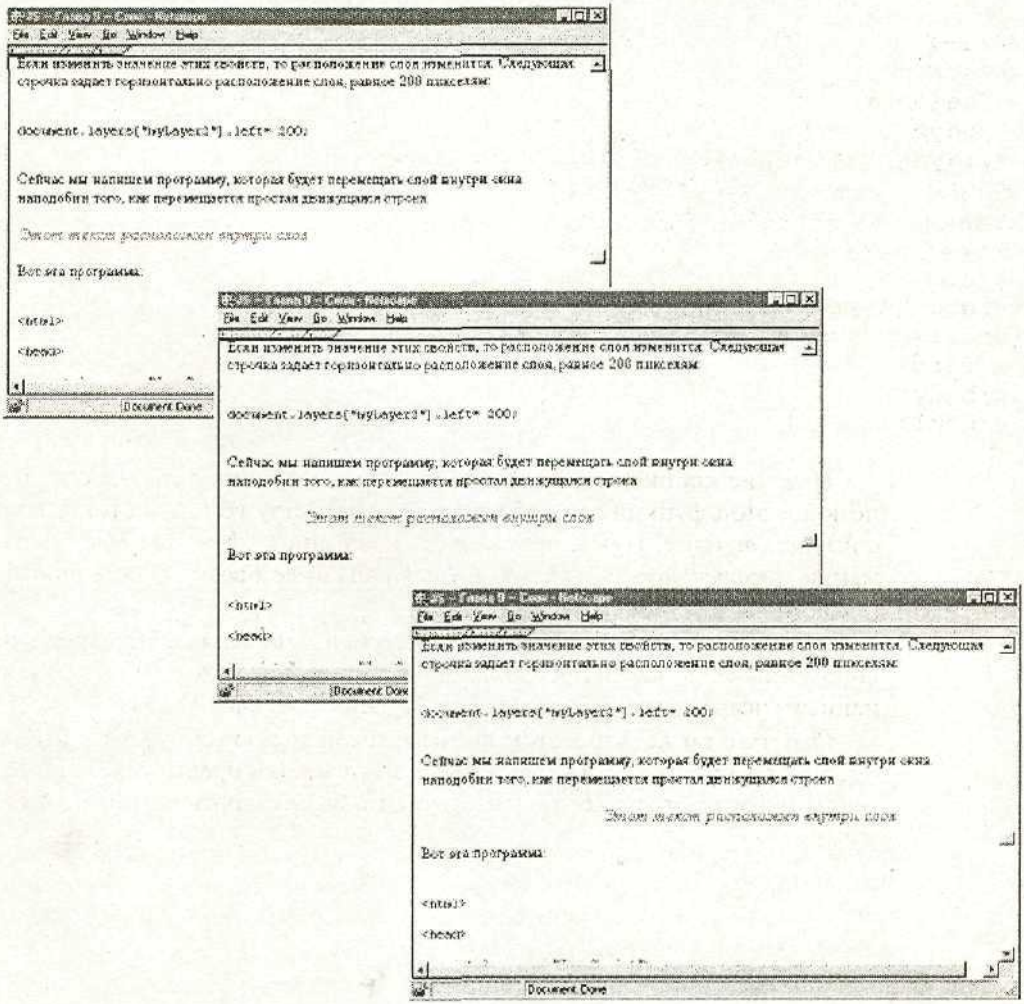
Отметьте также, что в этом примере мы использовали ярлык <ilayer> вместо ярлыка <layer>, который был использован в предыдущем примере, слой появляется в контексте документа без абсолютного позиционирования.

Перемещение слоев в окне

Как мы уже знаем, при помощи свойств *left* и *top* можно задать положение слоев. Если изменить значение этих свойств, то расположение слоя изменится. Следующая строчка задает горизонтально расположение слоя, равное 200 пикселям:

```
document.layers ["myLayer2"].left= 200;
```

Сейчас мы напишем программу, которая будет перемещать слой внутри окна наподобие того, как перемещается простая движущаяся строка.



Эти три картинки показывают последовательные положения текста "Этот текст расположен внутри слоя" в окне броузера. Текст движется вместе со слоем, относительное положение которого в окне броузера меняется со временем так, как это описано в нашем скрипте

Вот эта программа:

```
<html>
<head>
<script language="JavaScript">
<!- hide
function move () {
 if (pos < 0) direction=true;
 if (pos > 200) direction= false;
 if (direction) pos++
 else pos-;
 document.layers["myLayer2"].left= pos;
}
// ->
</script>
</head>
<body onLoad="setInterval('move() ', 20)">
<ilayer name=myLayer2 left=0>
<font size=+1 color="#0000ff"><i>Этот текст расположен внутри слоя</i></font>
</ilayer>
</body>
</html>
```

Здесь мы создали слой с именем *myLayer2.* Внутри ярлыка <body> мы описали свойство *onLoad,* потому что мы хотим, чтобы движение этого слоя началось сразу же после загрузки странички. При описании средства обработки событий onLoad мы использовали функцию *setInterval().* Этот метод позволяет обращаться к функции множество раз через заданный промежуток времени. Этот метод появился в JavaScript версии 1.2.

В предыдущей главе мы пользовались методом *setTimeout().* Метод *setInterval()* работает аналогичным образом, требуется лишь единственное обращение к этому методу. С помощью метода *setInterval()* происходит обращение к функции *move()* через каждый 20 миллисекунд. Каждый раз функция *move()* устанавливает новое положение слоя.

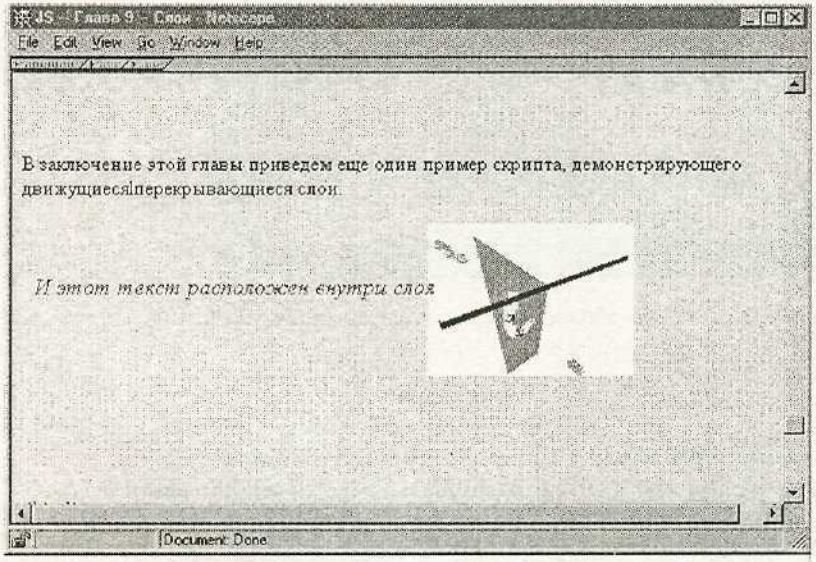
Регулярное обращение к этой функции приводит к появлению эффекта равномерного передвижения слоя по экрану внутри окна броузера. Все что требуется от функции *move()* — это сосчитать очередное значение координаты положение слоя и присвоить его посредством *document.layers["myLayer2"].left=pos.*

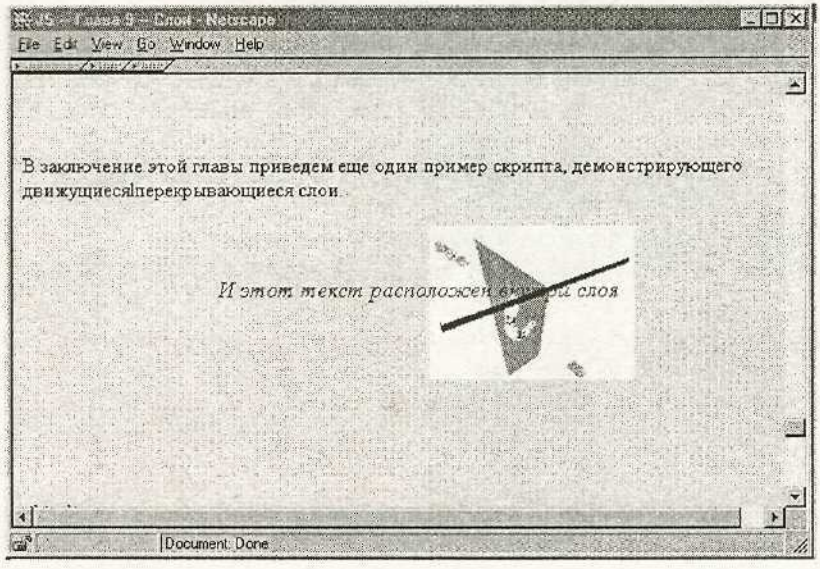
Если пользователь работает со старой версией броузера, "незнакомой" с языком JavaScript версии 1.2, то при выполнении скриптов, приведенных в настоящей главе, возможно появление сообщений об

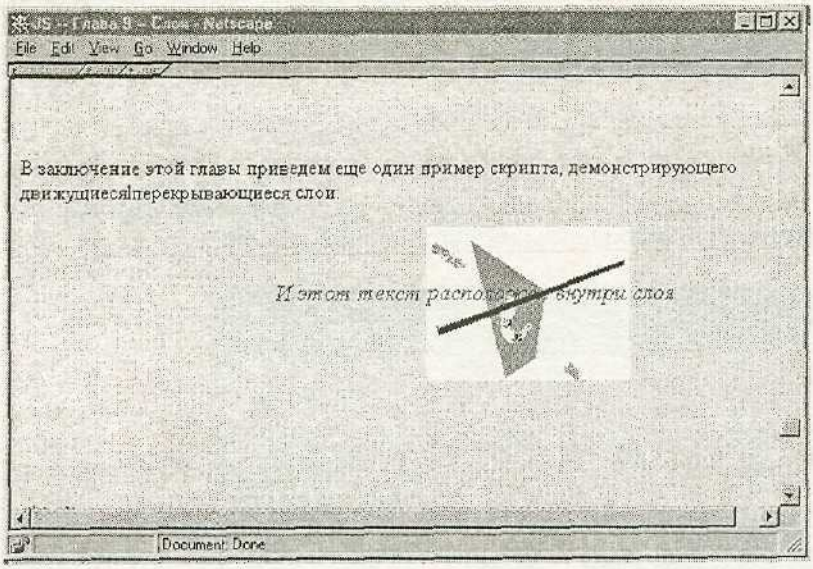
ошибке. Этого можно избежать. Для этого в ярлыке <script> необходимо указать версию языка, т.е. попросту написать так:

```
<script language="JavaScript1.2">
<!- hide
... (здесь расположен текст программы на JavaScript 1.2)
// ->
</script>
```

В заключение этой главы приведем еще один пример скрипта, демонстрирующего движущиеся перекрывающиеся слои.







Слой, содержащий текст, является прозрачным и передвигается поверх неподвижного слоя с картинкой

```
<html>
<head>
<script language="JavaScript">
<!- hide
function move2() {
 if (pos2 < 0) direction= true;
 if (pos2 > 200) direction= false;
 if (direction) pos2++
 else pos2-;
 document.layers["myLayer3"].left= pos2;
}
// ->
</script>
</head>
<body onLoad="setInterval('move2() ', 20)">
<ilayer name="myLayer3" left=0 z-index=21>
<font size=+lcolor="#0000ff"><1>ЭТОТтекст расположен внутри
слоя</i></font>
</ilayer>
<ilayer name="myLayer4" z-index="19">
<img src="img.gif" width="160" height="120">
</ilayer>
</body>
</html>
```

10 И ВНОВЬ О СЛОЯХ

- *Выделение видимых участков*
- *Вложенные слои*
- *Прозрачные слои и эффекты с ними*

В предыдущей главе мы рассмотрели основные свойства слоев и методы обращения с ними. Б этой главе мы рассмотрим следующие вопросы:

- выделение видимых участков;
- вложенные слои;
- прозрачные слои и эффекты с ними.

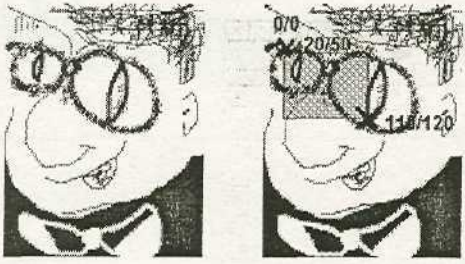
Выделениевидимыхучастков

При работе со слоями мы имеем возможность задать прямоугольную область внутри слоя, которая будет видима. Вся остальная часть слоя будет невидима. Для этого можно воспользоваться свойством, используемом в HTML, свойством clip. Пример задания видимой области приведен ниже:

```
<ilayer left=0 top=0 clip="20,50,110,120">
<img src="davinci.jpg" width=209 height=264>
</ilayer>
```

Здесь мы установили нулевые значения положения слоя (они также устанавливаются по умолчанию) и задали координаты углов видимой части слоя.

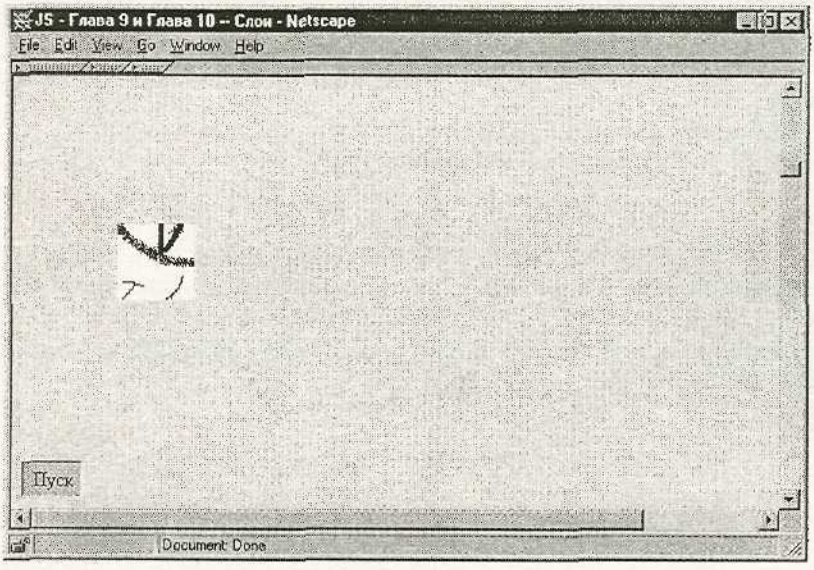
Несмотря на то, что полный размер изображения составляет 209x264 пикселей, мы будем видеть лишь малую часть от всего изображения.

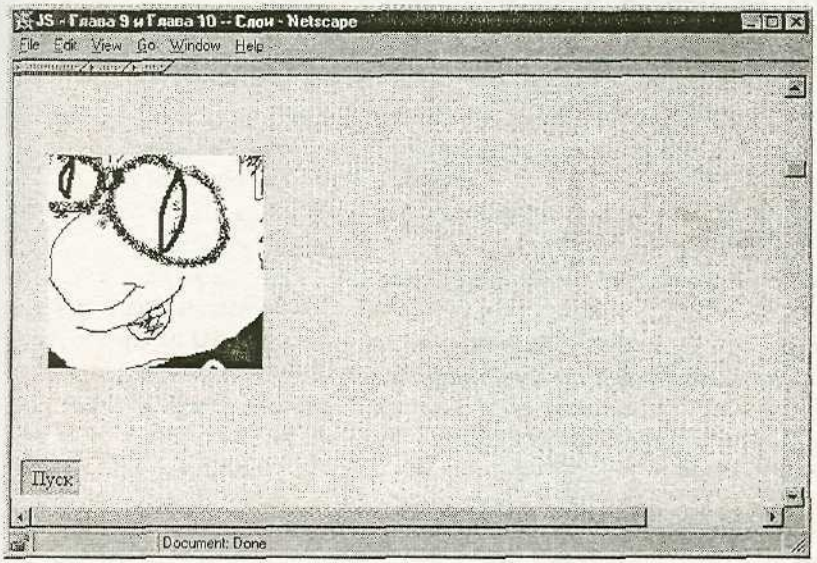


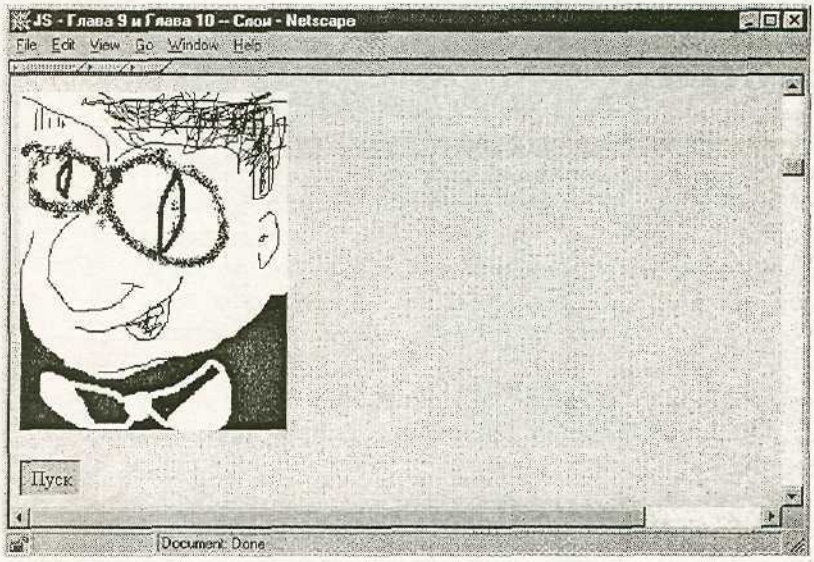
Размер видимой части — 90x70 пикселей. Первые два числа в значении clip в ярлыке <ilayer> указывают координаты левого верхнего угла видимой области, следующие два значения соответствуют координатам нижнего правого угла этой области. На следующем рисунке видимая область выделена четырехугольником и заштрихована.

Наиболее интересным представляется использование возможностей задания видимых областей изображений при помощи JavaScript. При помощи JavaScript мы можем изменить значения таких свойств объекта layer, как *clip.left, clip.top, clip.right* и *clip.bottom*.

Стоит лишь присвоить новые значения этим свойствам, как видимая часть слоя изменится. В приведенном ниже примере видимая область динамически меняется, создавая впечатление, будто бы постепенно изображение медленно вырастает из ничего.




JS - Глава 9 и Глава 10 -- Слои - Netscape
File Edit View Go Window Help
Пуск
Document: Done


JS - Глава 9 и Глава 10 -- Слои - Netscape
File Edit View Go Window Help
Пуск
Document: Done

Текст программы выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!- hide
var middleX, middleY, pos;
function start () {
 // получаем размер рисунка
 var width=
document.layers["imgLayer"].document.davinci.width;
 var height=
document.layers["imgLayer"].document.davinci.height;
 // вычисляем координаты центра рисунка
 middleX= Math.round(width/2);
 middleY= Math.round(height/2) ;
 // задаем начальное положение
 pos= 0;
 // запускаем картинку
 show();
}
function show() {
 // увеличиваем размер видимой области
 pos+= 2; // величина шага
 document.layers["imgLayer"].clip.left= middleX- pos;
 document.layers["imgLayer"].clip.top= middleY- pos;
 document.layers["imgLayer"].clip.right= middleX+ pos;
 document.layers["imgLayer"].clip.bottom= middleY+ pos;
 // проверяем, вся ли картинка видима
 if ( ! ( (pos > middleX) && (pos > middleY)))
 setTimeout("show()", 20);
}
// ->
</script>
</head>
<body>
<ilayer name="imgLayer" clip="0,0,0,0">
<img name=davinci src="davinci.jpg" width=209 height=264>
</ilayer>
<form>
<input type=button value="Пуск" onClick="start () ">
</form>
</body>
</html>
```

Кнопка, расположенная в теле странички за ярлыком <body>, приводит к обращению к функции start(). Первое, что необходимо вычислить — это координаты точки, расположенной в самом центре картинки. Эти координаты сохраняются в переменных *middleX* и *middleY*. Затем происходит обращение к функции *show()*. Эта функция устанавливает размер видимой области рисунка, который определяется значениями переменных *middleX, middleY* и *pos*.

Переменная *pos* увеличивается всякий раз, когда происходит обращение к функции *show()*. С каждым новым обращением к этой функции видимая часть рисунка становится все больше и больше. В конце тела функции *show()* мы используем *setTimeout()*.

Процесс увеличения видимой области, т.е. обращения к функции show(), повторяется снова и снова до тех пор, пока вся картинка не станет видимой. После этого работа программы заканчивается. Обратите внимание на то, каким образом мы получили информацию о размере картинки:

```
 var width=
document.layers["imgLayer"].document.davinci.width;
 var height=
document.layers["imgLayer"].document.davinci.height;
```

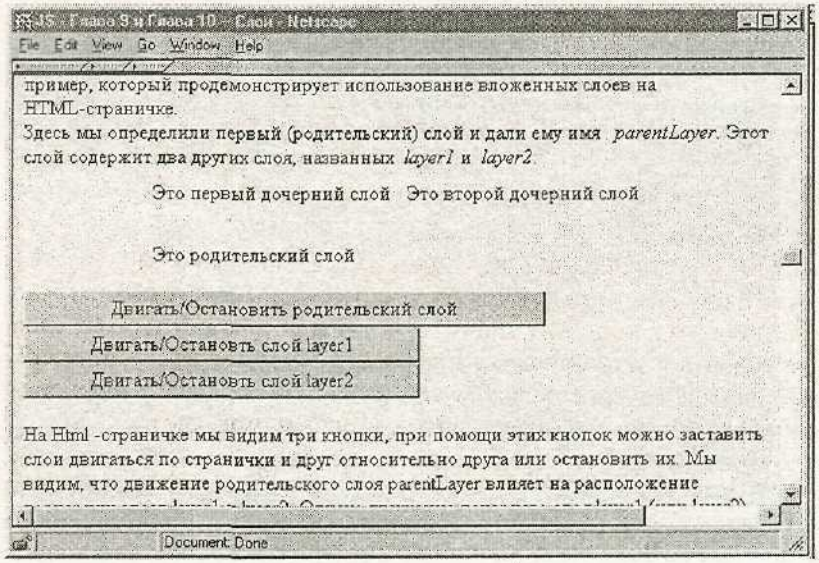
При помощи выражения *document.layers["imgLayer"]* мы можем обратиться непосредственно к слою с именем *imgLayer*. Зачем мы используем дополнительно слово *document* и располагаем его после *document.layers["imgLayer"]*?

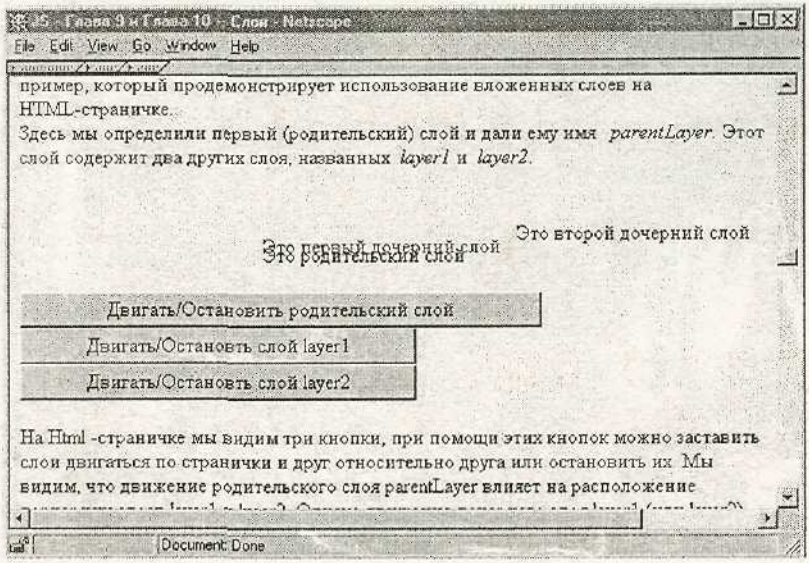
Дело в том, что каждый слой содержит в себе свою собственную HTML-страницу, иными словами, каждый слой имеет свой собственный объект document. Для того, чтобы обратиться конкретно к рисунку, расположенному в слое, а значит, на страничке этого слоя, нам необходимо вначале обратиться к объекту document этого слоя. Картинка в этом документе носит имя *davinci,* поэтому за словом document мы пишем davinci — имя картинки, к которой обращаемся.

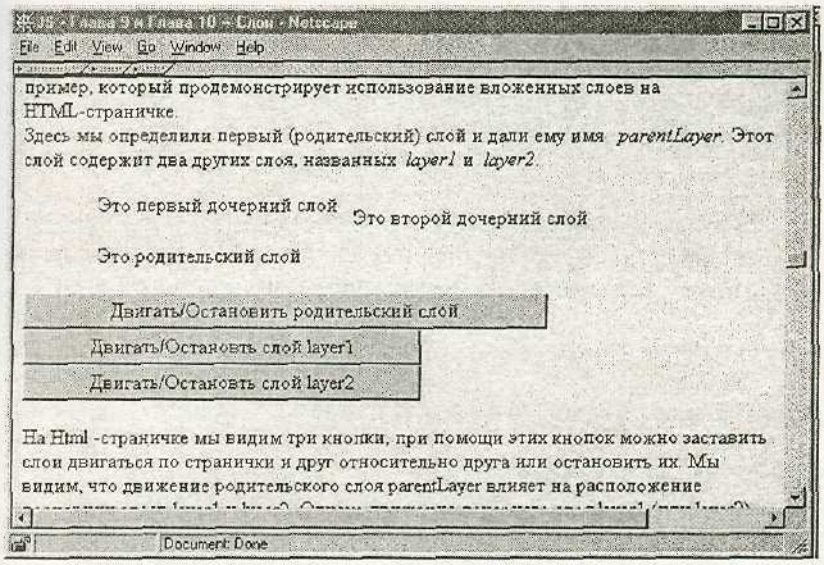
Вложенные слои

Ранее мы уже видели, что слои могут содержать в себе несколько различных объектов. Слои также могут содержать в себе другие слои. Можно задаться вопросом, когда такие вложенные слои бывают полезны? Можно назвать несколько полезных моментов, связанных с использованием вложенных слоев. И опять мы рассмотрим пример, который продемонстрирует использование вложенных слоев на HTML-страничке.

Здесь мы определили первый (родительский) слой и дали ему имя *parentLayer*. Этот слой содержит два других слоя, названных *layer1* и *layer2*.







На HTML-страничке мы видим три кнопки, при помощи этих кнопок можно заставить слои двигаться по страничке и друг относительно друга или остановить их. Мы видим, что движение родительского слоя parentLayer влияет на расположение дочерних слоев layer1 и layer2. Однако движение дочернего слоя layer1 (или layer2) приводит к изменению расположения текста, содержащегося лишь в этом слое.

Рассмотрим программу, создающую эти слои:

```
<html>
<head>
<script language="JavaScript">
<!- hide
// начальное положение
var pos0= 0;
var pos1= -10;
var pos2= -10;
// параметры| задающие состояние движения или покоя слоев
var move0= true;
var move1= false;
var move2= false;
// проверка направления движения
var dir0= false;
var dir1= false;
var dir2= true;
function startStop(which) {
 if (which == 0) move0= !move0;
 if (which == 1) move1= !move1;
 if (which == 2) move2=!move2;
}
function move() {
 if (moved) {
```

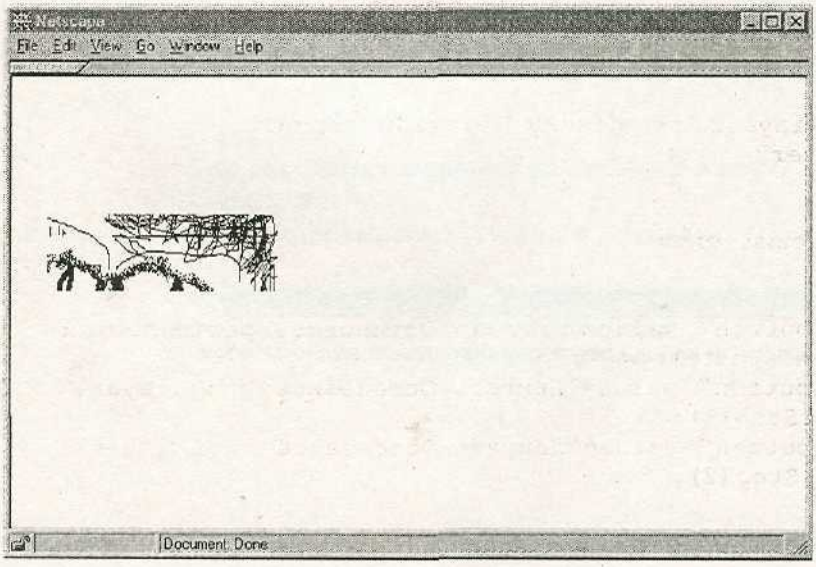
```
 // движение слоя parentLayer
 if (dir0) pos0-
  else pos0++;
 if (pos0 < -100) dir0= false;
 if (pos0 > 100) dir0= true;
 document.layers["parentLayer"].left= 100 + pos0;
 }
 if (move1) {
 // движение слоя layer1
 if (dir1) pos1—
  else pos1++;
 if (pos1 < -20) dir1= false;
 if (pos1 > 20) dir1= true;
 document.layers["parentLayer"].layers["layer1"].top= 10 +
pos1;
 }
 if (move2) {
 // движение слоя 1ayer2
 if (dir2) pos2-
  else pos2++;
 if (pos2 < -20) dir2= false;
 if {pos2 > 20) dir2= true;
 document.layers["parentLayer"].layers["layer2"].top= 10 +
pos2;
 }
}
// ->
</script>
</head>
<body onLoad="setInterval('move() ', 20)">
<ilayer name=parentLayer left=100 top=0>
 <layer name=layer1 z-index=l0 left=0 top=-10>
 Это слой layer1
 </layer>
 <layer name=layer2 z-index=20 left=200 top=-10>
 Это слой layer2
 </layer>
 <br><br>
 Это родительский слой
</ilayer>
<form>
<input type="button" value="Двигать/Остановить родительский
слой" onClick="startStop(0);">
<input type="button" value="Двигать/Остановить слой layer1"
onClick="startStop (1) ;">
<input type="button" value="Двигать/Остановить слой layer2"
onClick="startStop (2);">
</form>
</body>
</html>
```

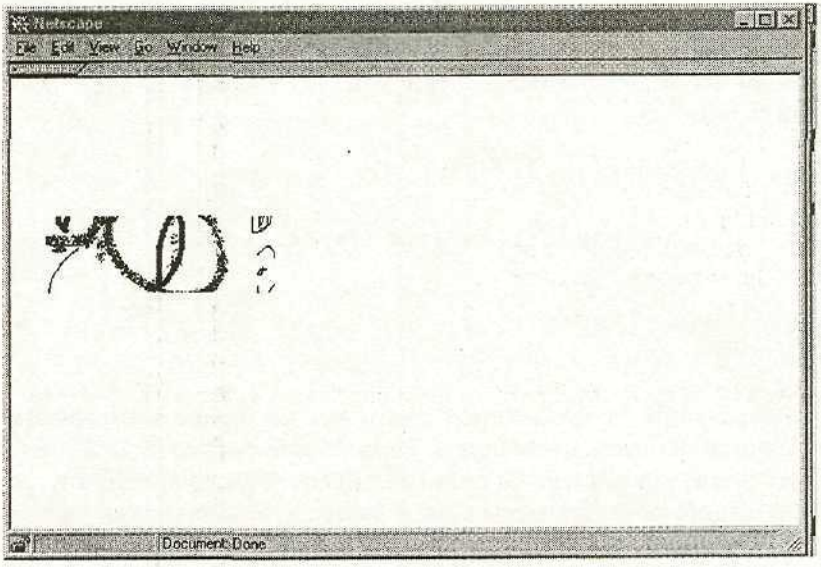
Из примера мы видим, что внутри родительского слоя мы задали два самостоятельных слоя. Эти слои являются вложенными слоями по отношению к родительскому слою. Каким способом мы можем обратиться к этим слоям на языке JavaScript? Вот как это сделано в примере в функции move():

```
document.layers["parentLayer"].left= 100 + pos0;
  ...
document.layers["parentLayer"] .layers["layer1"] .top= 10 +
pos1 ;
  ...
document.layers["parentLayer"].layers["layer2"].top= 10 +
pos2;
```

При обращении ко вложенным слоям мы не имеем возможности писать коротко *document.layers["layer1"]* или *document.layers["layer2"].* Это объясняется тем, что вложенные слои layer1 и layer2 расположены внутри самостоятельного объекта parentLayer, и доступ к ним возможен только через объект parentLayer.

Мы научились задавать параметры видимой области рисунка. На следующем примере мы посмотрим, как можно совместить возможность задания видимой области слоя и движение рисунка по HTML-странице. Здесь мы ставим перед собой задачу создания неподвижной видимой области на экране, и написание программы, движущейся по экрану картинки. Видимая область не будет передвигаться по экрану вместе с картинкой, она будет оставаться неподвижной.





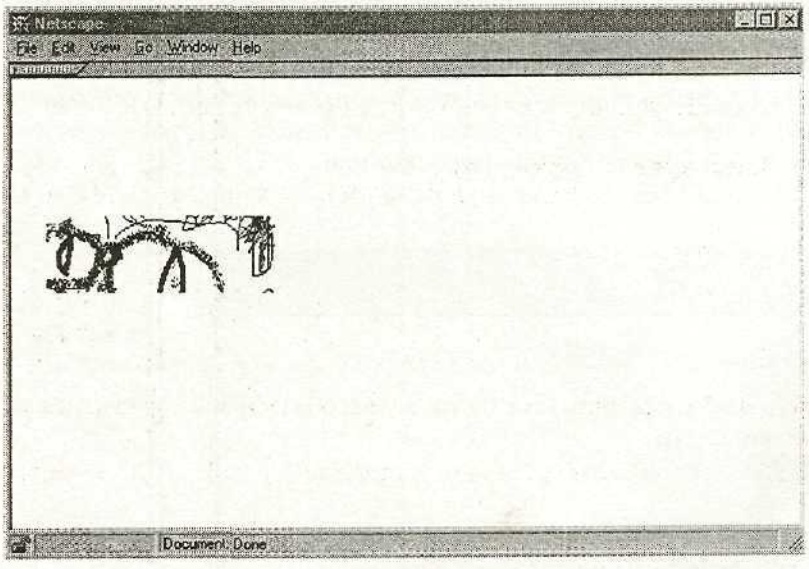


Рисунок "прокручивается" в неподвижной фиксированной видимой области экрана броузера

Программа выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!- hide
var pos= 0; // начальное положение
var direction= false;
function moveNclip() {
 if (pos<-180) direction= true;
 if (pos>40) direction= false;
 if (direction) pos+= 2
 else pos-= 2;

 document.layers["clippingLayer"].layers["imgLayer"].top= 100
+ pos;

}

// ->
</script>

</head>
<body onLoad="setInterval('moveNclip() ', 20);">

<ilayer name="clippingLayer" z-index=0 clip="20,100,200,160"
top=0 left=0>
 <ilayer name="imgLayer" top=0 left=0>
 <img name=davinci src="davinci.jpg" width=209 height=2 64>
 </ilayer>
</ilayer>

</body>
</html>
```

Еще раз обратим внимание на то, каким образом мы обращаемся ко вложенным слоям:

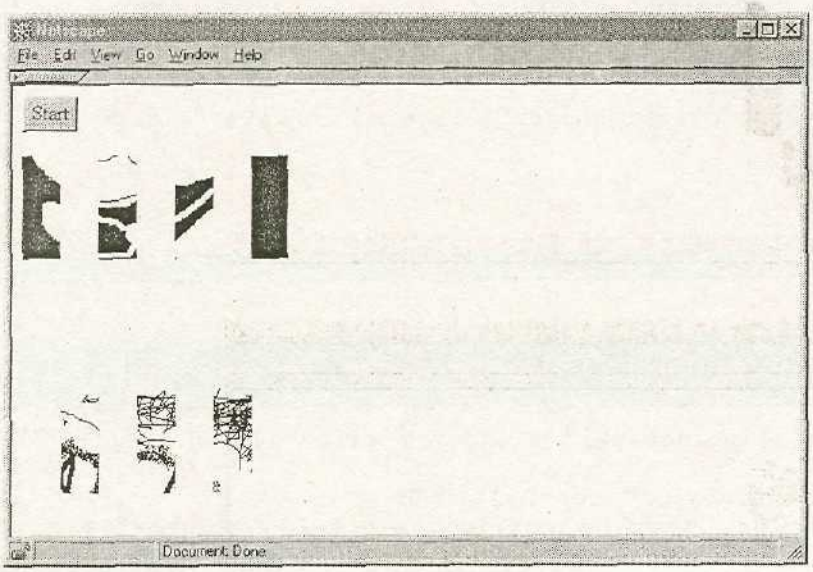
```
document.layers["clippingLayer"].layers["imgLayer"].top= 100 + pos;
```

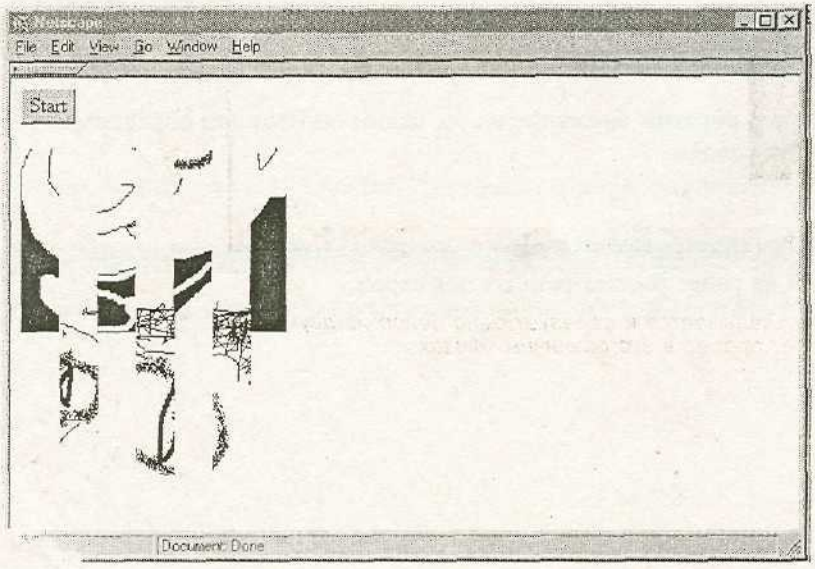
Все прочие элементы, содержащиеся в этой программе, нам уже знакомы из ранее рассмотренных примеров.

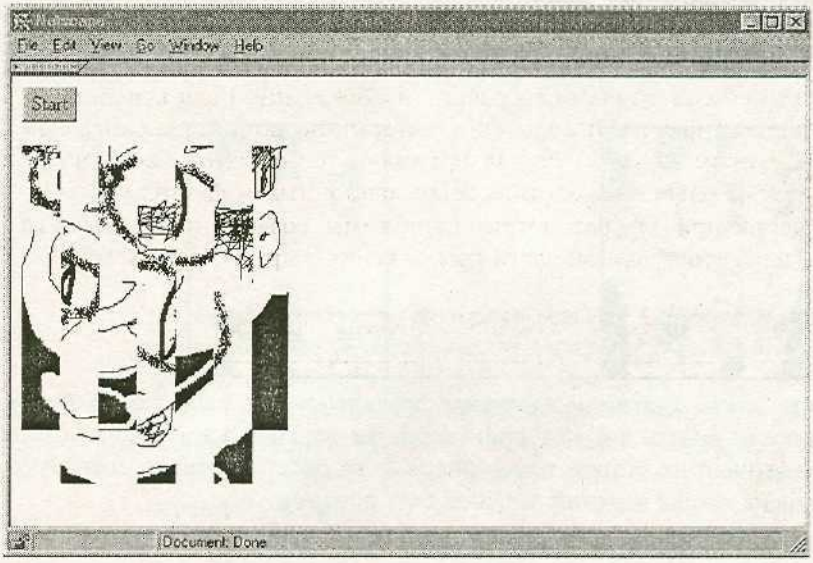
Прозрачные слои и эффекты с ними

При помощи прозрачных слоев (или частично прозрачных слоев) можно создавать довольно интересные эффекты. Особенно интересны эффекты с использованием прозрачных изображений. Если используются графические файлы, изображение которых не затеняет расположенную слоем ниже графическую информацию, то мы имеем счастливую возможность создания качественных анимационных эффектов.

В качестве примера рассмотрим программы, создающий) несложный эффект "вползания" одной части изображения в другую его часть.







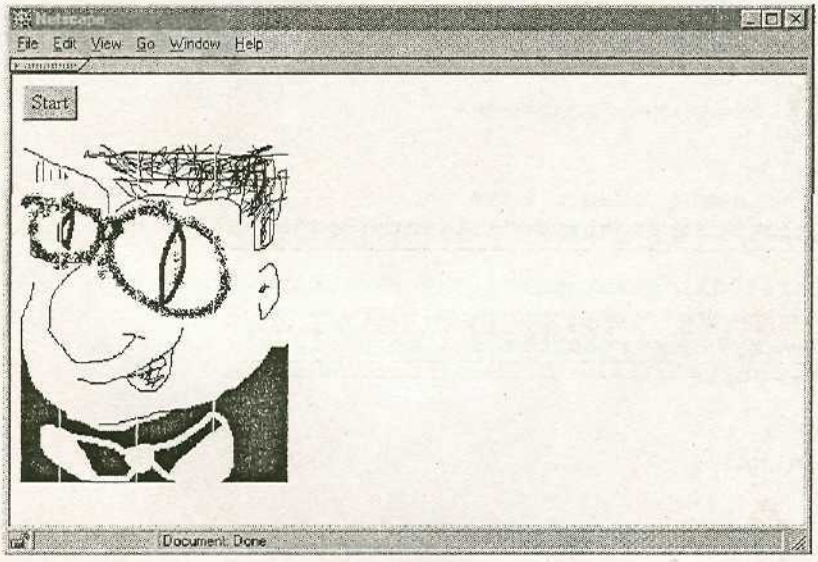
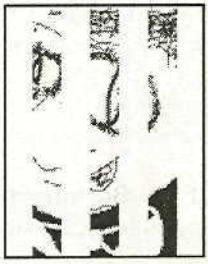


Рисунок постепенно складывается и образует одно целое из двух частей. На картинках показан этот процесс в его различных фазах

Для создания этого эффекта использованы две картинки. Эти картинки движутся друг другу навстречу, одна — снизу вверх, другая — сверху вниз. Вот эти картинки:



Тогда, когда картинки начинают перекрываться, необходимо иметь возможность видеть то, что происходит за верхней картинкой, иначе расположенный на втором плане рисунок не будет видимым. Для этого необходимо, чтобы верхний рисунок был прозрачным.

Вот программа этого примера:

```
<html>
<head>
<script language="JavaScript">
<!- hide
var pos= 0; // начальное положение
var myTimer= null;
function move() {
 pos+= 3; // величина одного шага
 document.layers["clippingLayer"].layers["imgLayer1"].top= -
264 + pos;
 document.layers["clippingLayer"].layers["imgLayer2"].top= 264
- pos;
 if {pos< 264) myTimer= setTimeout('move()', 10)
 else myTimer= null;
}
function start() {
 if (myTimer== null) {
 pos= 0;
 move ();
  }
}
// ->
</script>
</head>
<body onLoad="move() ">
<form>
<input type=button value="Start" onClick="start();">
</form>
```

```
<ilayer name="clippingLayer" z-index=0 clip="209,264" top=0 left=0>
 <layer name="imgLayer1" top=-264 left=0>
 <img name=davinci1 src="davinci1.gif" width=209 height=264>
 </layer>
 <layer name="imgLayer2" top=264 left=1>
 <img name=davinci2 src="davinci2.gif" width=209 height=264>
 </layer>
</ilayer>
</body>
</html>
```

Если Вы читаете Интернет-версию этой книги, то для запуска примеров Вам достаточно нажимать кнопки. Если Вы читаете текстовую версию книги на экране своего компьютера, то для демонстрации примеров Вам достаточно скопировать тексты программ через буфер обмена в стандартный HTML-файл, обзавестись необходимыми для правильной работы странички графическими файлами и загрузить вновь созданную страничку в броузер.

В качестве графических файлов можно использовать любые картинки, важно понять, как работают скрипты на JavaScript. Если Вы пользуетесь бумажной версией, то примеры скриптов можно проверить, введя тексты программ вручную и запомнив их в виде HTML-файлов.

11

СОБЫТИЯ В JavaScript 1.2

- *Новые события*
- *Объект Event (событие)*
- *Перехват событий*

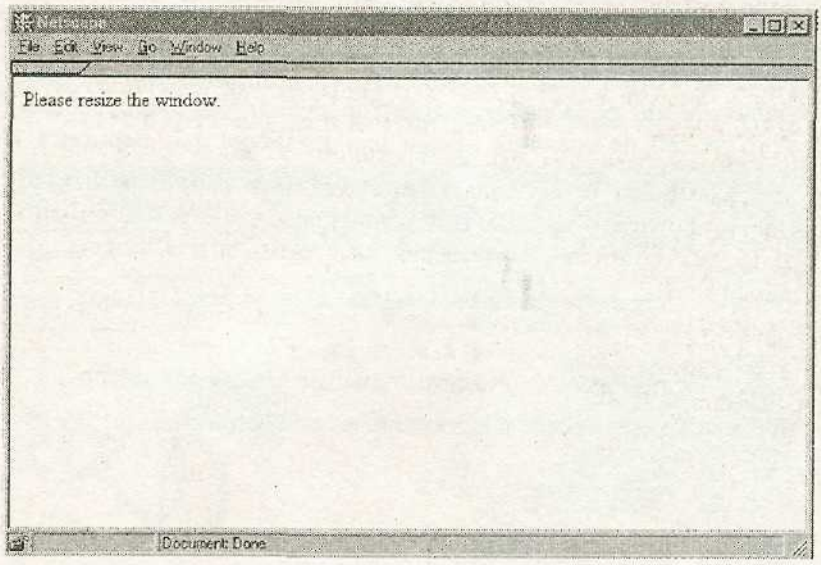
Новые события

Ранее мы уже упоминали о том, что JavaScript 1.2 имеет возможность обрабатывает новые события, которые не описаны в более ранних версиях языка. В таблице, расположенной ниже, перечислены события JavaScript 1.2.

Abort	*Focus*	*MouseOut*	*Submit*
Blur	*KeyDown*	*MouseOver*	*Unload*
Click	*KeyPress*	*Mouse Up*	
Change	*KeyUp*	*Move*	
DblClick	*Load*	*Reset*	
DragDrop	*Mouse Down*	*Resize*	
Error	*Mouse Move*	*Select*	

В настоящей главе мы рассмотрим некоторые новые события более подробно. Сначала обратимся к событию Resize. При помощи этого события мы имеем возможность узнать о том, был ли изменен пользователем размер окна броузера. Как работает это событие, показывает следующий пример:

```
<html>
<head>
<script language="JavaScript">
window.onresize= message;
function message() {
 alert("Размер окна был изменен!");
}
</script>
</head>
<body>
Измените размер окна.
</body>
</html>
```



Первоначально окно выглядело так

Размер окна был изменен и JavaScript позволил отследить факт изменения размера окна

При помощи строчки

```
window.onresize= message;
```

мы задаем действие, которое будет осуществлено при наступлении события Resize. Мы определили средство управления событием onResize. Функция *message()* будет выполнена при изменении размеров окна. Здесь мы определили средство обработки события новым для нас способом, однако этот способ не является новым для JavaScript 1.2. Вернемся к ранее рассмотренным примерам. Если мы хотим определить средство обработки события, связанного с нажатием кнопки, то мы можем написать программу привычным способом, как мы это делали ранее:

```
<form name="myForm">
<input type="button" name="myButton" onClick="alert('Click
event occurred!')">
</form>
```

Тот же смысл несет в себе следующий текст программы:

```
<form name="myForm">
<input type="button" name="myButton">
</form>
...
<script language="JavaScript>
document.myForm.myButton.onclick= message;
function messaged {
 alert('Click event occurred!');
}
</script>
```

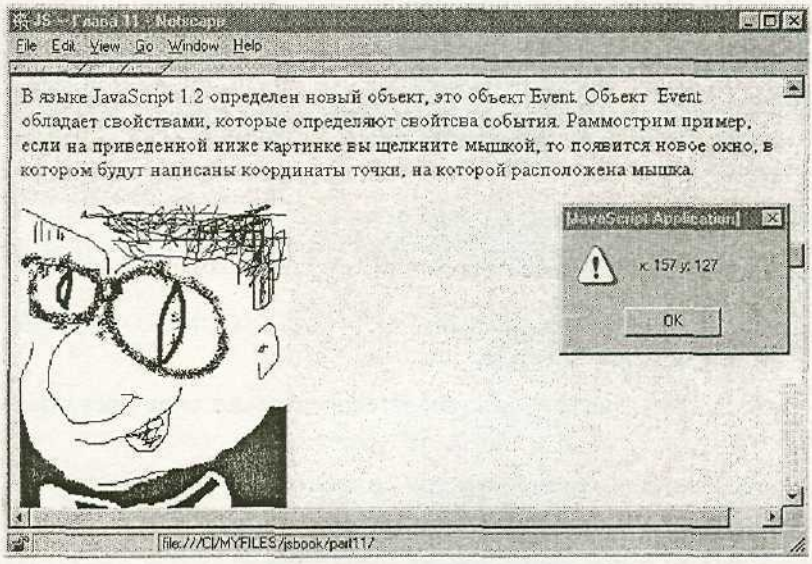
Может показаться, что второй вариант является более сложным, занимает больше места и неудобен при составлении программ. В примерах, которые мы рассматривали в предыдущих главах, это действительно так, поэтому мы и пользовались первым способом.

Однако существуют ситуации, когда наиболее предпочтительным оказывается второй способ. Более того, такой объект, как объект window, нельзя описать при помощи обычных ярлыков, подобно тому, как описываются при помощи ярлыков формы и кнопки. И здесь мы вынуждены пользоваться вторым способом задания средств обработки событий.

Здесь необходимо сделать два замечания. Первое, не следует писать window.onResize, можно пользоваться только прописными буквами, т.е. писать window.onresize. Второе, после слова message мы не должны писать скобки, т.е. мы не можем написать window.onresize=message(). Если мы напишем такую строчку (что будет ошибкой в нашем случае), то броузер воспримет messgae() как обращение к функции, чего не должно быть в нашем случае, так мы не требуем обращения к функции, но лишь хотим определить средство обработки события.

Объект Event (событие)

В языке JavaScript 3.2 определен новый объект, это объект Event. Объект Event обладает свойствами, которые определяют свойства события. Рассмотрим пример, если на приведенной ниже картинке Вы щелкните мышкой, то появится новое окно, в котором будут написаны координаты точки, на которой расположена мышка.



Скрипт позволяет узнать координаты курсора (указателя мыши)

Текст программы выглядит следующим образом:

```
<layer>
<a href="#" onClick="alert('x:' + event.x + ' y: ' +
event.y) ; return false;">
<img src="davinci.jpg" width=209 height=264 border=0></a>
</layer>
```

Здесь мы используем средство обработки событий onClick, расположенное внутри ярлыка <a>, что мы бы могли писать и в старых версиях JavaScript.

Новым здесь является то, что далее мы используем событие event и посредством event.x и event.y обращаемся к его свойствам (в данном случае это координаты точки щелчка) для того, чтобы вывести координаты положения точки щелчка во всплывающем окне. Здесь мы использовали объект Event для того, чтобы определить координаты события.

Мы поместили весь пример внутри слоя (текст программы размещен после ярлыка <layer>), поэтому мы получим координаты относительно этого слоя, в нашем случае весь слой состоит из рисунка, и координаты будут выдаваться относительно этого рисунка.

Если бы мы не размещали наш пример в слое, то координаты были бы определены относительно окна броузера. Инструкция return false здесь используется для того, чтобы броузер не переходил по указанной ссылке, а игнорировал ее.

Объект Event обладает следующими свойствами (некоторые из них мы рассмотрим далее в примерах):

data.....Массив адресов URL тех объектов, которые были перетащены, событие *DragDrop* (перетаскивание)

layerX.....Горизонтальное положение курсора относительно слоя в пикселях. В сочетании с событием Resize это свойство определяет ширину окна броузера

layerY.....Вертикальное положение курсора относительно слоя в пикселях. В сочетании с событием Resize это свойство определяет высоту окна броузера

modifiers.....Строчка, определяющая признаки: ALT_MASK, CONTROL_MASK, META_MASK, SHIFT_MASK

pageX.....Горизонтальное положение курсора относительно окна броузера в пикселях

pageY.....Вертикальное положение курсора относительно окна броузера в пикселях

screenX.....Горизонтальное положение курсора относительно экрана в пикселях

screenY.....Вертикальное положение курсора относительно экрана в пикселях

target.....Строка, указывающая объект, которому направляется данное событие

type.....Строка, задающая тип события

which.....ASCII значение нажатой клавиши или номер кнопки мыши

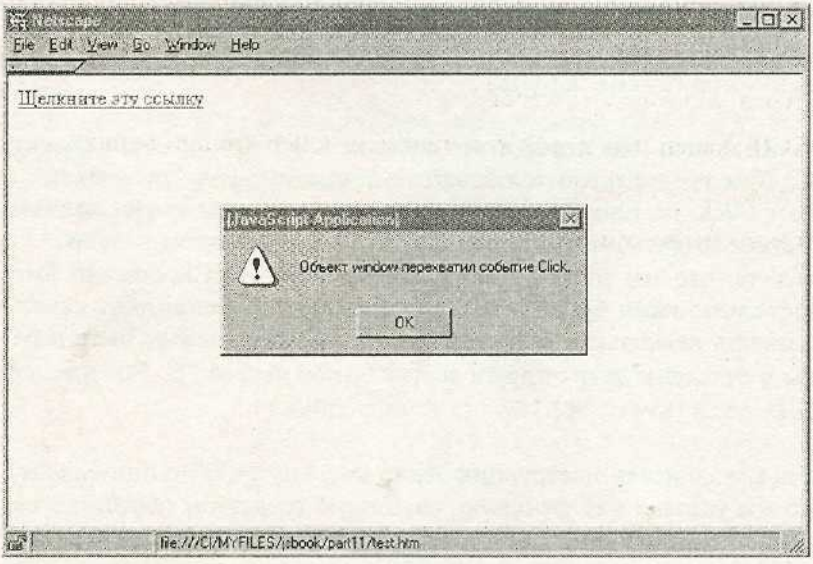
x.....Эквивалентно layerX

y.....Эквивалентно layerY

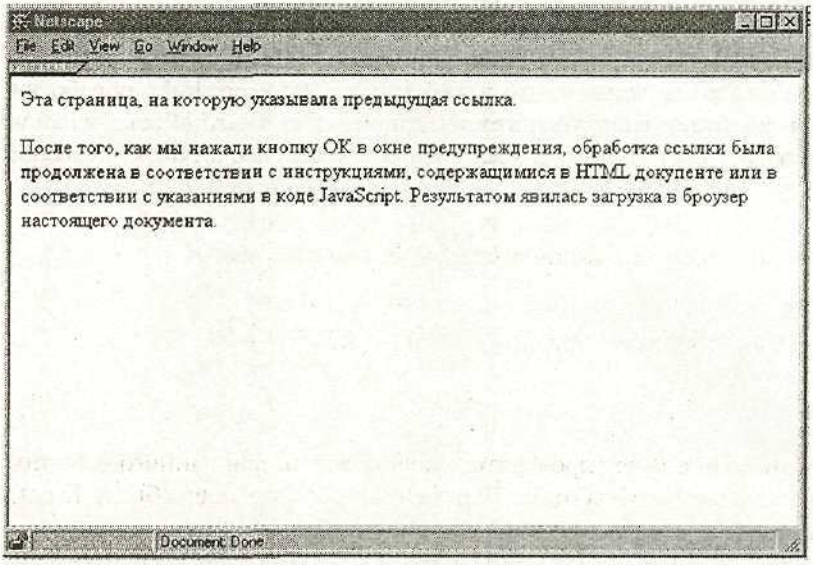
Перехват событий

Еще одно важное свойство — это возможность перехвата событий. Если Вы, например, щелкаете кнопку мыши, то происходит событие Click, и выполняется инструкция, указанная в средстве обработки события onClick. При помощи перехвата событий можно сделать так, что окно, документ или слой "перехватят" данное событие до того, как оно будет обработано. Рассмотрим пример, чтобы понять, что в этом свойстве есть полезного.

```
<html>
<head>
<script language="JavaScript">
window.captureEvents(Event.CLICK) ; \
window.onclick= handle;
function handle(e) {
 alert("Объект window перехватил событие Click!");
 return true; // выполняем инструкции далее, т.е. переходим
по ссылке
}
</script>
</head>
<body>
<a href="test.htm">Нажмите эту ссылку</a>
</body>
</html>
```



После нажатия на ссылку происходит перехват события Click



Как только перехват был обработан произошел переход по указанной ссылке

В этом примере (пример загружается в отдельное окно) после того, как мы нажмем указанную ссылку, появляется новое окно с предупреждением "Объект window перехватил событие Click. В окне "Предупреждение" содержится кнопка "OK". После того, как мы щелкаем эту кнопку, происходит переход по указанной ссылке. Здесь мы не описывали средство обработки события внутри ярлыка <a>, вместо этого мы написали строчку:

```
window.captureEvents(Event.CLICK);
```

Это позволило нам перехватить событие Click при помощи объекта window. При обычных обстоятельствах объект window "не знаком" с событием Click, но при помощи свойства перехвата мы имеем возможность перенаправить это событие и связать его с объектом window.

Заметьте, что мы пишем Event.CLICK, здесь CLICK должно быть написано заглавными буквами. Если существует необходимость осуществить захват нескольких событий, то эти события должны быть перечислены и отделены друг от друга вертикальной чертой "|". Вот пример:

```
window.captureEvents(Event.CLICK | Event.MOVE);
```

Здесь мы записали инструкцию *return true;* внутри функции *handle(),* которую мы указали как функцию, служащую средством обработки события onClick. Это означает, что броузер перейдет по указанной ссылке сразу после выполнения функции handle(). Если же мы напишем вместо

return true строчку return false, то все действия, которые должны были произойти при наступлении события Click, не будут выполнены.

Если мы определим средство обработки событий onClick внутри ярлыка <a>, то мы увидим, что в этом случае наше средство обработки события не будет работать. Это очевидно, поскольку объект window перехватывает событие перед тем, как он обращается к объекту ссылки (link)

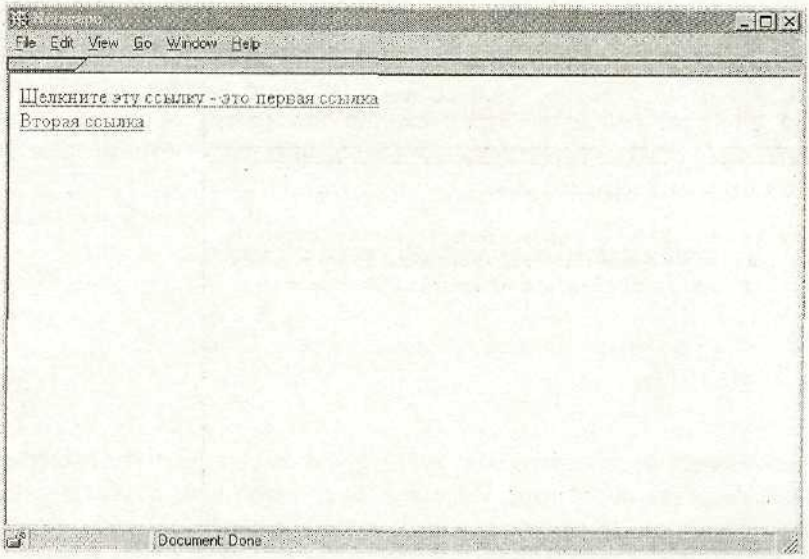
Если мы опишем функцию *handle()* таким образом:

```
function handle(e) {
 alert("The window object captured this event!");
 window.routeEvent(e);
 return true;
}
```

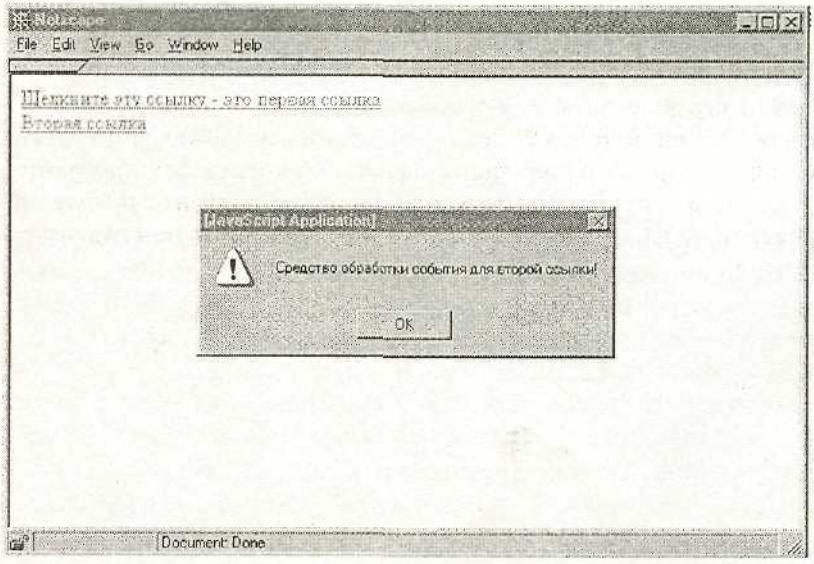
то компьютер будет проверять, существуют ли для данного события еще иные средства обработки. Переменная e — это наш объект Event, который передается посредством нашей функции handle(e).

Мы также имеем возможность осуществлять пересылку событий прямо какому-либо конкретному объекту. Для этого мы можем использовать метод *handleEvent()*. Пример выглядит так:

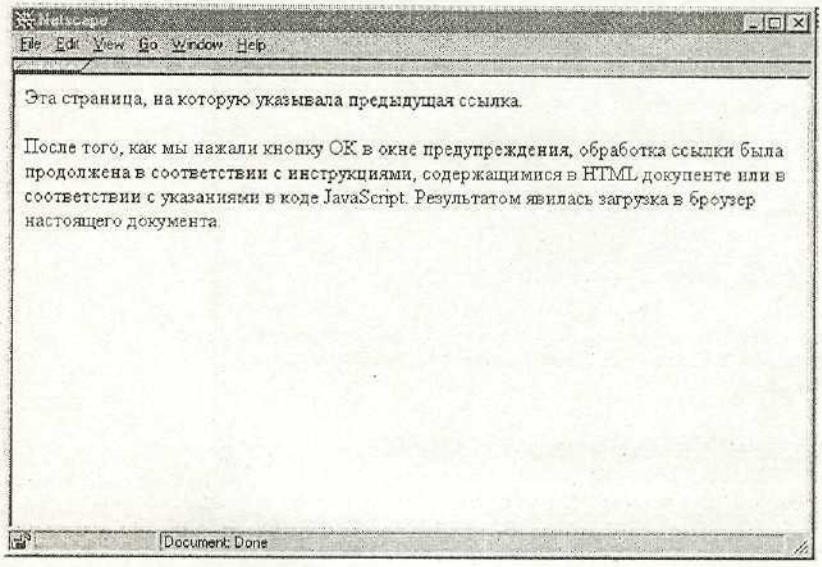
```
<html>
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
 document.links[1].handleEvent(e);
}
</script>
<a href="test.htm">Щелкните эту ссылку</a><br>
<a href="test.htm"
 onClick="alert('Средство обработки второй ссылки!');">Вторая
ссылка</a>
</html>
```



На странице предложены две ссылки. Однако после того, как будет перехвачено событие Click (нажать мышку можно в любой части окна броузера), переход по ссылке будет обработан



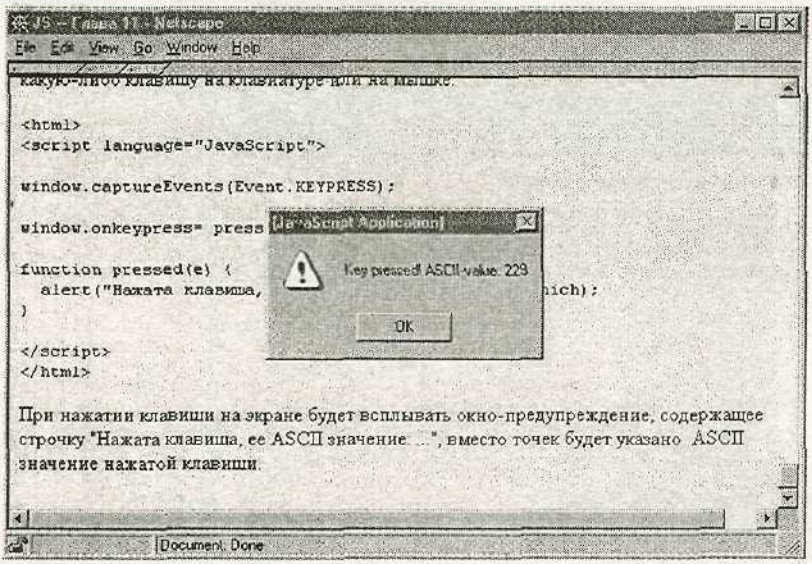
Осуществлен перехват события Click



Произведен переход в соответствии с инструкциями JavaScript

Все события Click будут направлены на вторую ссылку, даже в том случае, если Вы не нажимаете эту ссылку непосредственно!

Следующий пример показывает, что программа может реагировать на события, связанные с нажатиями клавиш. Запишите эту программу (или нажмите кнопку для загрузки примера) и загрузите в броузер. Для проверки ее работы нажмите какую-либо клавишу на клавиатуре или на мышке.



При нажатии клавиши на экран выводится окно, содержащее значение этой клавиши.

```
<html>
<script language="JavaScript">
window.captureEvents(Event.KEYPRESS);
window.onkeypress= pressed;
function pressed(e) {
 alert("Нажата клавиша, ее значение ASCII: " + e.which);
}
</script>
</html>
```

При нажатии клавиши на экране будет всплывать окно-предупреждение, содержащее строчку "Нажата клавиша, ее ASCII значение: ...", вместо точек будет указано ASCII значение нажатой клавиши.

12

ПЕРЕТАСКИВАНИЕОБЪЕКТОВ

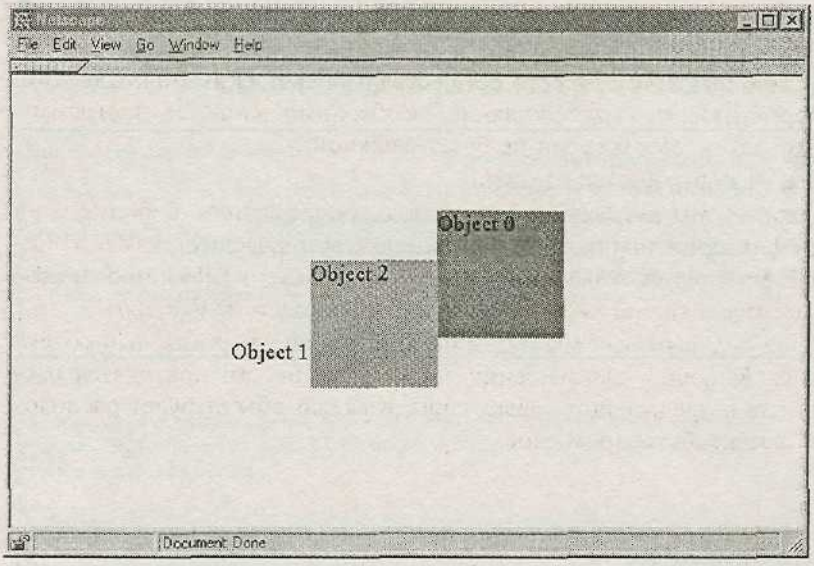
- *Что такое перетаскивание?*
- *События Mouse и JavaScript 1.2*
- *События MouseDown, MouseMove и MouseUp*
- *Изображение перетаскиваемых объектов на экране*
- *Кидание объектов*

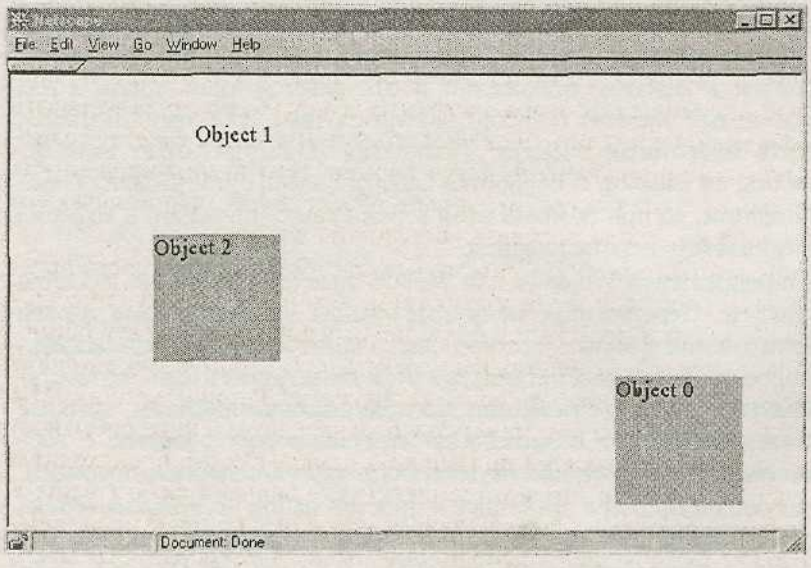
Чтотакоеперетаскивание?

Перетаскивание — перевод английского термина drag & drop. Перетаскивание — это выбор элемента при помощи нажатия кнопки маши и перемещение этого элемента в новое место, при этом кнопка мыши удерживается в нажатом положении и отпускается лишь тогда, когда перетаскиваемый элемент (курсор) находится там, куда предполагается перетащить выбранный элемент. Поскольку JavaScript имеет возможность работы со слоями, и обработка событий здесь организована более гибким образом, то при помощи этой версии языка появляется возможность организовать перетаскивания.

Для пользователей Windows или MacOS перетаскивание не является чем-то новым. Перетаскивая ненужные файлы в корзину, мы можем удалить эти лишние файлы с жесткого диска компьютера. То перетаскивание, которое мы будем рассматривать в этой главе, может быть осуществлено только в пределах веб-страниц, т.е. при помощи программ, которые мы напишем далее, нет возможности перетаскивания объектов с веб-страницы на жесткий диск или осуществлять другие подобные операции.

В следующем примере представлен простой набор нескольких объектов. Нажмите на кнопку, после этого произойдет загрузка программы в ваш броузер, и попробуйте поперетаскивать по окну эти объекты. Если Вы читаете текстовую версию, создайте новый HTML файл с текстом HTML-странички, который приведен ниже.





Установив указатель мыши на любой из объектов и удерживая кнопку мыши в нажатом положении, мы можем переместить этот объект в любое место в окне броузера

JavaScript не поддерживает перетаскивания в явном виде, т.е. в языке нет такого свойства, котором бы описывалось перетаскивание. У объекта Image нет свойства, назовем его например, dragging, задавая которое можно было бы легко и просто осуществлять перетаскивания. Для того, чтобы организовать перетаскивание, необходимо написать программу. Радует то, что программа эта не будет сложной.

Итак, что потребуется сделать?

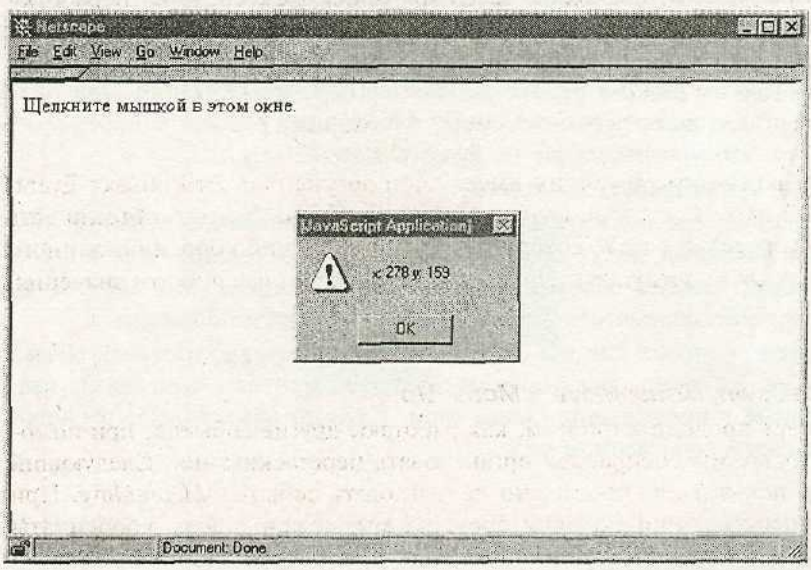
Первое — мы должны зафиксировать определенное событие, при помощи фиксации такого события мы сможем определить, какой конкретно объект следует перемещать и в какое положение. Мы произведем выбор перетаскиваемого и объекта и его новое местоположение.

Второе — это то. как мы будем отображать перетаскиваемый объект на экране. Конечно, для решения данной задачи нам пригодятся рассмотренные в предыдущих главах слои. Каждый объект будет расположен на своем собственном слое.

События Mouse и JavaScript 1.2

Итак, какие события мы будем использовать? В нашем распоряжении нет события, которое в явном виде задавало бы перетаскивание, однако мы можем достичь своей цели, используя события *MouseDown, MouseMove* и *MouseUp.* Если пользователь нажмет где-нибудь в окне броузера кнопку мыши, то наша программа должна будет прореагировать на это событие и определить, на каком объекте (на каком слое) находится указатель мыши в момент нажатия кнопки. Нам нужны координаты данного события. Для этого мы имеем объект Event, с помощью его свойств мы легко можем получить координаты события.

Второй важный момент — это то, что мы имеем возможность организовать захват события. Если мы, скажем, нажимаем кнопку, то событие нажатия кнопки посылается сразу непосредственно объекту button. Но в нашем случае нам необходимо сделать так, чтобы объект window смог обработать наступившее событие {нажатие кнопки мыши). Мы создаем ситуацию, когда объект window получает захваченное событие и реагирует на него. Следующий пример показывает, как это происходит в случае с событием Click. Вы можете щелкнуть мышью в любом месте в пределах окна броузера, в ответ появится новое окно с предупреждением, в котором будут указаны координаты мыши, т.е. координаты места щелчка мышью,



Щелкнув мышью в окне броузера, мы получим координаты положения указателя мыши во время щелчка

Так выглядит текст программы:

```
<html>
<script language="JavaScript">
<!-
   window.captureEvents(Event.CLICK);
   window.onclick= displayCoords;
   function dispiayCoords(e) {
      alert("x: " + e.pageX + " y: " + e.pageY);
   }
// ->
</script>
Щелкните где-нибудь в окне броузера.
</html>
```

Первое, что мы потребовали от объекта window, — это осуществить перехват события Click. Для этого мы использовали метод *captureEvent()*.

Строчка

```
window.onclick=  displayCoords;
```

задает то, что будет происходить в случае наступления события *Click*. Эта инструкция сообщает броузеру, что в случае наступления события Click необходимо вызвать функцию *displayCoords()*. Здесь при описании средства обработки события не следует использовать скобки после выраже-

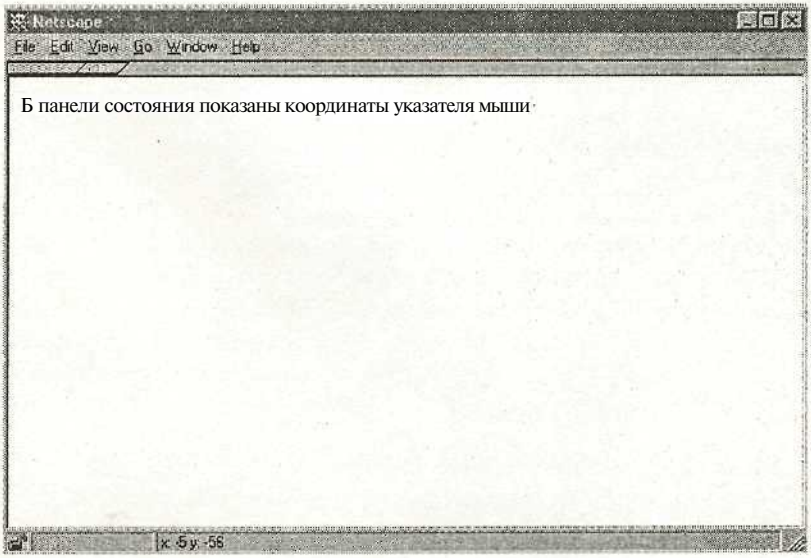
ния displayCoords, поскольку здесь мы описываем средство обработки события, передавая ему лишь имя функции, но не саму функцию. Функция *display Coords()* выглядит следующим образом:

```
function displayCoords(e) {
    alert("x: " + e.pageX + " y: " + e.pageY);
}
```

Мы видим, что функция имеет один аргумент e. Это объект Event, который передается этой функции для обработки. У этого объекта есть свойства pageX и pageY, которые содержат значения координат данного события. Окно предупреждения выводит на экран именно эти значения.

События MouseDown, MouseMove и MouseUp

Теперь продемонстрируем, как работают другие события, при помощи которых мы собираемся организовать перетаскивания. Следующий пример показывает, как можно использовать событие *MouseMove*. При перемещении мыши по окну броузера мы можем видеть координаты курсора в строке состояния.



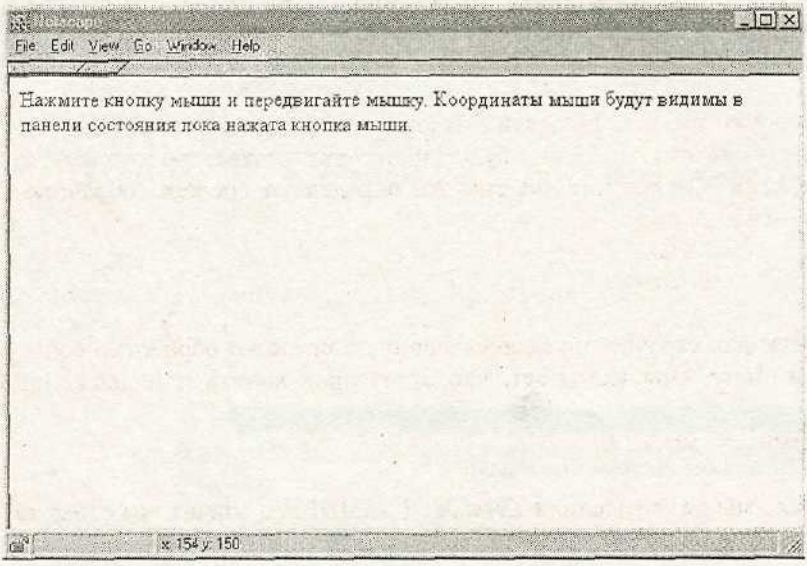
В панели состояния показаны координаты мыши

Текст программы очень сильно похож на текст предыдущего примера:

```
<html>
<script language="JavaScript">
<!-
   window.captureEvents(Event.MOUSEMOVE);
   window.onmousemove= displayCoords;
   function displayCoords(e) {
      status= "x: " + e.pageX + " y: " + e.pageY;
   }
// ->
</script>
Координаты мыши показаны в строке состояния.
</html>
```

Здесь необходимо писать выражение *Event MOUSEMOVE,* причем *MOUSEMOVE* должно быть написано заглавными буквами. Однако при определении средства обработки события мы должны использовать строчные буквы: *window.onmousemove= displayCoords.*

Сейчас мы уже можем объединить эти два примера вместе. Координаты указателя мыши будут показаны в строке состояния до тех пор, пока мышь будет передвигаться с нажатой кнопкой. Пример выглядит так:



Координаты мыши в этом случае видны только тогда, когда нажата кнопка мыши

Текст программы:

```
<html>
<script language="JavaScript">
<!-
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
function startDrag(e) {
   window.captureEvents(Event.MOUSEMOVE);
}
function moveIt(e) {
   // показываем координаты
   status= "x: " + e.pageX + " y: " + e.pageY;
}
function endDrag(e) (
   window.releaseEvents(Event.MOUSEMOVE);
}
// ->
</script>
Нажмите кнопку и перемещайте мышь по экрану.
Координаты мыши показаны в строке состояния.
</html>
```

Здесь мы, как это было и в предыдущих примерах, первым делом заставляем объект window перехватить события. Сейчас перехватываемыми событиями являются события *MouseDown* и *MouseUp:*

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

В скобках мы использовали значок | для разделения событий, его значение — логическое "или", будет происходить захват любого из этих двух событий. Следующие две строчки определяют средства обработки этих событий:

```
window.onmousedown= startDrag;
window.onmouseup= endDrag;
```

Следующая строчка определяет еще одно средство обработки события *MouseMove.* Она указывает, что будет происходить при движении мыши:

```
window.onmousemove= moveIt;
```

Однако мы не определили *Event.MOUSEMOVE,* а значит это событие не будет захвачено объектом window. И кажется, что мы зря обращаемся к функции moveIt при помощи той строчки, что написана выше, поскольку событие MouseMove не достигнет объекта window. Не спешите,

посмотрите на функцию *startDrag(),* обращение к ней происходит сразу же после того, как наступит событие *MouseDown:*

```
function startDrag(e) {
  window.captureEvents(Event.MOUSEMOVE) ;
}
```

Сейчас мы видим, что объект window наделяется возможностью осуществлять перехват событий MouseMove сразу после того, как нажимается кнопка мыши. Мы должны позаботиться и о том, чтобы объект window не смог захватывать событие MouseMove после того, как кнопка мыши была отпущена. Это делает функция *endDrag(),* в которой использован метод *releaseEvents():*

```
function endDrag(e) {
  window.releaseEvents(Event.MOUSEMOVE);
}
```

Функция *moveIt()* также показывает координаты мыши в строке состояния.

Теперь мы имеем все необходимое для фиксации перетаскиваемого объекта и определения его нового местоположения. Вернемся назад ко второй части задачи и подумаем о том, как будет изображаться перетаскиваемый объект.

Изображение перетаскиваемых объектов на экране

В предыдущих частях мы уже познакомились с тем, как можно создавать движущиеся объекты при помощи слоев. Сейчас нам необходимо лишь написать программу, которая будет определять, над каким слоем произошло нажатие кнопки мыши, затем нужно лишь заставить этот слой двигаться вместе с мышью. Это можно сделать следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!-
var dragObj = new Array();
var dx, dy;
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
function startDrag(e) {
  currentObj= whichObj (e);
  window.captureEvents(Event.MOUSEMOVE) ;
}
function moveIt(e) {
  if (currentObj != null) {
    dragObj[currentObj].left= e.pageX - dx;
    dragObj[currentObj].top= e.pageY - dy;
  }
```

```
}
function endDrag(e) {
   currentObj= null;
   window.releaseEvents(Event.MOUSEMOVE);
}
function init() {
   dragObj[0]= document.layers["layer0"];
   dragObj[1]= document.layers["layer1"];
   dragObj[2]= document.layers["layer2"];
}
function whichObj (e) {
   // определяем, какой объект "щелкнут" мышкой
   var hit= null;
   for (var i= 0; i < dragObj.length; i++) {
      if ((dragObj[i].left < e.pageX) &&
          (dragObj[i].left + dragObj[i].clip.width > e.pageX) &&
          (dragObj[i] .top < e.pageY) &&
          (dragObj [i] . top + dragObj [i] .clip.height >
e.pageY)) {
             hit= i;
             dx= e.pageX- dragObj[i].left;
             dy= e.pageY- dragObj[i].top;
             break;
      }
   }
   return hit;
}
//->
</script>
</head>
<body onLoad="init() ">
<layer name="layer0" left=100 top=200 clip="100,100"
bgcolor="#0000ff">
<font size=+1>Object 0</font>
</layer>
<layer name="layer1" left=300 top=200 clip="100,100"
bgcolor="#00ff00">
<font size=+1>Object 1</font>
</layer>
<layer name="layer2" left=500 top=200 clip="100,100"
bgcolor="#ff0000">
<font size=+1>Object 2</font>
</layer>
</body>
</html>
```

Мы задали три слоя, которые описаны в теле страницы после ярлыка <body>. После того, как вся страница будет загружена, происходит обращение к функции init(), которая вызывается средством управления событием onLoad, которое мы расположили в ярлыке <body>:

```
function init() {
  dragObj[0]= document.layers["layer0"] ;
  dragObj[1]= document.layers["layer1"] ;
  dragObj[2]= document.layers["layer2"] ;
}
```

Массив *dragObj* содержит три слоя, которые могут быть перемещены по желанию пользователя. Каждый слой имеет свой номер, соответствующий номеру элемента массива *dragObj*. В дальнейшем нам эти номера пригодятся. Для захвата событий мы использовали в точности такой же фрагмент программы, что был приведен в только что рассмотренном нами примере:

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
```

В функции *startDrag()* добавлены строчка:

```
currentObj= whichObj(e);
```

Функция *whichObj()* определяет тот объект, который был выбран пользователем, т.е. тот объект, на котором щелкнула мышка. Функция возвращает число — номер слоя. Если не выбран ни один слой, то функция возвращает значение null. Возвращаемое значение хранится в переменной *currentObj*.

В функции *whichObj()* осуществляется проверка значений свойств *left, top, width* и *height* каждого слоя. С помощью такой проверки происходит определение того, какой слой был выбран щелчком мыши.

Киданиеобъектов

Перетаскивание объектов состоит из двух частей Первая часть — это собственно перемещение объекта. Вторая часть — это бросание объекта, кидание его в новом месте. Для того чтобы избавиться от ненужного фала, мы перетаскиваем его к корзине и затем кидаем его в корзину (отпускаем клавишу мыши). Пока мы не рассмотрели вопрос о том, как правильно кинуть предмет. Правильно кинуть — это тоже важно. Предположим, что мы хотим создать виртуальный магазин. Пользователю необходимо разместить несколько приобретаемых предметов в своей

корзине. Мы должны знать, что пользователь кинул эти предметы в свою корзину, т.е. имеет намерение оплатить их.

Как определить, кинул покупатель предметы в корзину или нет? Для этого нам нужно знать координаты объекта непосредственно после события MouseUp. Для этого нам необходимо добавить несколько инструкций в функцию endDrag(). Мы, например, можем установить проверку того, попадают ли координаты мыши в пределы установленной прямоугольной области (соответствующей корзине), если координаты попадают в эту область, то можно, например, осуществить обращение к функции, которая регистрирует факт покупки путем, скажем, размещения покупаемых объектов в массиве. Демонстрируя элементы этого массива, мы показываем предметы, расположенные в корзине.

Конечно, наша программа не лишена недостатков. Может показаться странным, что перетаскиваемый объект вдруг оказывается позади покоящегося объекта. Эту задачу можно решить, если изменить порядок следования объектов, задав новый порядок внутри функции startDrop(). Конечно, объекты в виде красной, синей и зеленой фигур — это не лучший пример. Вы можете использовать более изящные картинки, описав их при помощи ярлыка <img>. Более того, в пределах каждого слоя можно использовать и любые другие объекты.

НЕКОТОРЫЕ ОБЪЕКТЫ, МЕТОДЫ, СВОЙСТВА В JAVASCRIPT

ОбъектСсылка

Способ создания

Может быть создана с использованием ярлыка <a> или путем вызова метода String.anchor. При создании ссылки по второму методу используется строка theString.anchor(nameAttribute). Здесь:

theString.....*строковый объект*

nameAttribute.....*строка*

При определении объекта ссылки с помощью ярлыка <a> используется стандартный синтаксис HTML. Если будет задано имя Ссылки, то обращение к ней может быть осуществлено с помощью индексированного массива с указанием этого имени.

Описание

Объект Ссылка Anchor является также объектом Link. Он учитывается в массиве объектов Anchor и в массиве объектов Link.

Свойства

name.....Строка, представляющая собой имя Ссылки

text.....Текст, по которому предоставляется возможность Ссылки

x.....Горизонтальное положение Ссылки в пикселях по отношению к левому краю документа

y.....Вертикальное положение Ссылки в пикселях по отношению к верхнему краю документа

Методы

С этим объектом могут использоваться методы watch и unwatch.

Объект Applet

Используется для включения Java-апплета в веб-страницу.

Создается при помощи ярлыка <APPLET>. Для каждого включенного в веб-страницу апплета создается свой собственный объект, объекты образуют массив document.applets. Доступ к апплету может быть осуществлен с помощью индексации этого массива.

Объект Array— Массив

Объект создается при помощи таких строчек:

new *Array(arrayLength)*

new *Array(eletnent0, element1, ..., elementN)*

Параметры

arrayLength.. Начальная длина массива. Эта величина может быть вызвана путем обращения к свойству length. Если в указанном свойстве будет содержаться величина, не являющаяся числом, то будет создан массив, длина которого равна единице.

element*N*..... Список значений элементов массива. Когда используется такой список, то создается массив, элементы которого инициализированы указанными в списке значениями, а длина массива равна количеству указанных в списке значений.

Свойства

constructor ...Описывает функцию, которая создает прототип объекта

length.....Соответствует количеству элементов массива

prototype..... Позволяет вводить дополнительные свойства объекта

Методы

concat.....Объединяет два массива и возвращает новый массив

join.....Объединяет все элементы массива в строку

pop.....Удаляет последний элемент из массива и возвращает этот элемент в качестве значения

push.....Добавляет один или несколько элементов к концу массива и возвращает длину нового массива

reverse.....Переставляет элементы массива, меняя направление, первый элемент массива становится последним, а последний — первым

shift.....Удаляет первый элемент массива и возвращает этот элемент в качестве значения

slice.....Выделяет фрагмент массива и возвращает новый массив

splice.....Добавляет и/или удаляет элементы массива

sort.....Осуществляет сортировку элементов массива

toSource.....Возвращает литерал того или иного массива, может быть использован для создания нового массива

toString.....Возвращает строку, соответствующую массиву и его элементам
unshift.....Добавляет один или несколько элементов, присоединяя их к началу массива, и возвращает длину нового массива
valueOf.....Возвращает значение массива

Объект document

В этом объекте содержится информация о текущем документе и используются методы, позволяющие установить вид документа на основе кода HTML. Объект может быть создан при помощи ярлыка <body>. Для каждой HTML-страницы создается свой объект document. Каждый объект window обладает свойством object, значением которого является загруженный в окно объект document. При создании объекта document в ярлыке <body> могут быть заданы те или иные средства обработки событий.

Средства обработки событий

- onClick
- onDblClick
- onKeyDown
- onKeyPress
- onKeyUp
- onMouseDown
- onMouseUp

Эти средства обработки событий указываются в ярлыке <body>, однако они относятся к объекту window. Новый документ может быть загружен путем обращения к свойству window.location. Закрыть панель с документом и удалить весь текст, формы и т.д. можно при помощи следующих инструкций:

```
document.close();
document.open();
document.write();
```

Свойства

alinkColor.....Строка, задающая параметр для ALINK
anchors.....Массив, содержащий все Ссылки документа
applets.....Массив, содержащий все апплеты документа
bgColor.....Строка, задающая параметр BGCOLOR
classes.....Создает объект стилей Style, который может задавать стили для HTML
cookie.....Задает cookie
domain.....Задает имя сервера, соответствующего документу

embeds.....Массив всех плагинов в документе
fgColor.....Строка, задающая параметр fgColor для текста
formName.....Свойство для каждой именованной формы в документе
forms.....Массив всех форм, содержащихся в документе
height.....Высота документа в пикселях
ids.....Создает объект Style
images.....Массив всех изображений, содержащихся в документе
lastModified... Строка, указывающая дату последнего изменения документа
layers.....Массив всех слоев, содержащихся в документе
linkColor.....Строка, задающая цвет текста Ссылки
links.....Массив всех ссылок, содержащихся в документе
plugins.....Массив всех плагинов, содержащихся в документе
referrer.....Строка, содержащая URL вызывающего документа
tags.....Создает объект Style
title.....Строка, которая задает заголовок в ярлыке **TITLE**
URL.....Строка, которая содержит полный URL документа
vlinkColor.....Строка, которая задает параметр VLINK
width.....Ширина документа в пикселях

Методы

captureEvents Устанавливает возможность захвата всех событий указанного типа
close.....Закрывает поток вывода и показывает данные
contextual.....Использует возможность контекстного отбора
getSelection ... Возвращает строку, содержащую текст текущего выбора
handleEvent ...Активизирует средство обработки того или иного события
open.....Открывает поток вывода для методов **write** и **writeln**
releaseEvents. Устанавливает состояние документа или окна осуществлять захваченные события заданного вида
routeEvent.....Передает захваченные события в соответствии с обычной иерархией
write.....Записывает одно или несколько HTML выражений в документ, расположенный в заданном окне
writeln.....Записывает одно или несколько **HTML** выражений в документ, расположенный в заданном окне и добавляет символ перевода строки

Объект Form

Позволяет использовать формы для ввода текста и осуществления выбора с использованием таких элементов, как поля для галочек, радио кнопки, списки для выбора. Формы могут также использоваться для передачи данных серверу.

Создается при помощи ярлыка <FORM>. Для каждой формы в документе создается свой собственный объект FORM.

Средства обработки событий

- onReset
- onSubmit

Описание

Каждая форма в документе — это отдельный объект. К этому объекту можно обратиться либо при помощи имени, задав его с использованием свойства NAME, или с использованием массива объектов форм Form.elements. Массив элементов включает в себя все элементы, такие как checkbox, radio, text.

Свойства

action.....Задает свойство ACTION
elements.....Массив всех элементов формы
encoding.....Задает свойство ENCTYPE
length.....Задает количество элементов в форме
method.....Задает свойство METHOD
name.....Задает свойство NAME
target.....Задает свойство TARGET

Методы

handleEvent ...Активизирует средство обработки события
reset.....Имитирует нажатие кнопки reset в форме
submit.....Посылает форму

Объект Frame

В окне броузера может быть показано несколько независимых фреймов, в каждом из которых может быть представлен документ со своим собственным независимым URL. Фреймы создаются при помощи ярлыка <frame>, который используется внутри фрагмента, открытого с помощью другого ярлыка — ярлыка <frameset>. Набор всех фреймов образует законченную страничку. Каждый фрейм может быть направлен для загрузки своего собственного URL, а также в него можно загрузить URL по указанию из другого фрейма, расположенного на той же страничке (в том же основном окне броузера).

В JavaScript к фреймам происходит обращение, как к объектам window. Каждый фрейм — это объект window.

Объект window

Объект window описывает окно броузера или фрейм. Этот объект является объектом top-level для всех объектов document.location и history.

Если появляется ярлык <body> или <frameset>, то для каждого такого ярлыка создает свой собственный (или несколько) объект window. Для каждого ярлыка <frame> создается объект window. Объект window также можно создать, вызвав метод window.open.

Средства обработки событий

- onBlur
- onDragDrop
- onError
- on Focus
- onLoad
- onMove
- onResize
- onUnload

В иерархии объектов JavaScript объект window является самым старшим объектом.

- К объекту окна свойства parent и top могут быть применены, однако они указывают на само это окно. Для фреймов свойство top указывает на самое главное (основное) окно броузера, а parent указывает на родительское окно внутри главного окна броузера.
- Для окна верхнего уровня свойства defaultStatus и status задают текст, который появится в строке состояния. Для фреймов эти свойства задают статус фрейма, который будет показан только в том случае, если указатель мыши находится над данным окном фрейма.
- Метод close не используется для окон фреймов.
- Для того, чтобы использовать средства обработки событий onBlur и onFocus, следует задать их (строчные буквы), например, в коде html.

Для всех окно свойства self и window являются синонимами. Например, для закрытия окна можно использовать метод close по отношению либо к свойству window, либо к свойству self.

Свойства

closed.....Определяет, было ли окно закрыто
crypto.....Объект, который позволяет осуществлять доступ к закодированным данным
defaultStatus.. Содержит статус, присваиваемый по умолчанию, который будет показан в панели состояния
document.....Содержит информацию о текущем документе и предоставляет методы представления документа пользователю
frames.....Массив фреймов в окне
history.....Содержит информацию об адресах посещенных пользователем страниц
innerHeight....Указывает вертикальный размер в пикселях
innerWidth.....Указывает горизонтальный размер в пикселях
length.....Количество фреймов в данном окне
location.....Содержит адрес URL
locationbar.....Представляет панель местоположения для данного окна
menubar.....Представляет панель меню окна броузера
name.....Уникальное имя, присвоенное данному окну, по которому к нему можно обращаться
opener.....Содержит имя окна, из которого произошло открытие данного окна при помощи метода open
outerHeight ...Содержит вертикальный размер окна в пикселях по внешней границе окна
outerWidth.....Содержит горизонтальный размер окна в пикселях по внешней границе
pageXOffset... Содержит текущее положение по горизонтали демонстрируемой страницы
pageYOffset... Содержит текущее положение по вертикали для демонстрируемой страницы
parent.....Синоним для окна или для фрейма, в котором содержится описание <frameset> для данного фрейма
personalbar .... Содержит персональную пользовательскую панель броузера
screenX.....Содержит горизонтальную координату левого края окна
screenY.....Содержит вертикальную координату верхнего края окна
scrollbars.....Представляет панель прокрутки броузера
self.....Синоним для window в текущем окне
status.....Устанавливает приоритет сообщения в панели состояния
statusbar.....Представляет панель состояния окна
toolbar.....Представляет панели инструментов окна
top.....Синоним самого старшего окна броузера
window.....Синоним текущего окна

Методы

alert.....Выводит диалоговое окно предупреждения, содержащее текст предупреждения и кнопку OK

atob.....Производит декодирование строки данных, закодированных на основе base-64

back.....Производит возвращение на один шаг вперед в history

blur.....Удаляет фокус с указанного объекта

btoa.....Создает закодированную в base-64 строку

captureEvents Устанавливает захват всех событий определенного типа

clearInterval .. Отменяет задержку, установленную с помощью setInterval

clearTimeout.. Отменяет установку, которая была сделана с использованием метода setTimeout

close.....Закрывает определенное окно

confirm.....Выводит диалоговое окно подтверждения, содержащее некоторое сообщение, а также кнопки OK и Cancel

crypto.random.. Возвращает псевдослучайную строку символов с указанной длиной в байтах

find—Позволяет находить указанную текстовую строку в содержимом указанного окна

focus.....Устанавливает фокус на указанном объекте

forward.....Загружает документ, соответствующий последующему адресу URL в списке history

home.....Направляет броузер к тому URL, который записан в качестве домашней страницы в соответствии с выбором пользователя

moveBy.....Передвигает окно на указанную величину

moveTo.....Передвигает верхний левый угол окна к указанным координатам экрана

open.....Открывает новое окно броузера

print.....Выводит на печать содержимое окна или фрейма

prompt.....Выводит диалоговое окно, содержащее сообщение и окно для ввода текста

resizeBy.....Производит изменение размера окна путем передвижения нижнего правого угла окна на заданную величину

resizeTo.....Производит изменение размера окна в соответствии с указанными размерами внешних границ окна по высоте и ширине

scroll.....Перекручивает окно к указанным координатам

scrollBy.....Перематывает видимую область экрана на указанную величину

scrollTo.....Перекручивает видимую область окна к указанным координатам, указанная точка становится верхним правым углом видимой области

setHotKeys .... Включает или выключает специальные комбинации кнопок

setInterval.....Выполняет то или иное действие (выполняет функцию, осуществляет проверку) каждый раз по истечении определенного интервала времени в миллисекундах

setResizable.....Разрешает пользователю производить изменение размеров окна

setTimeout.....Осуществляет то или иное действие (выполняет функцию, осуществляет проверку) единожды по истечении указанного промежутка времени в миллисекундах

stop.....Останавливает процесс загрузки

Объект Layer

Соответствует слоям, расположенным в HTML-документе, позволяет осуществлять манипуляции со слоями.

Слой создается при помощи ярлыков HTML <layer> и/или <ilayer> или при использовании каскадных стилей синтаксиса. Для каждого слоя в документе создается свой собственный объект Layer. Все слои в документе образуют массив document.layers. Обращение к слою можно осуществить путем обращения к этому массиву с помощью индексов.

Средства обработки событий

- onMouseOver
- onMouseOut
- onLoad
- onFocus
- onBlur

Свойства

above.....Объект Layer, расположенный над данным объектом или объемлющее окно, если данный слой является самым высшим

background .... Картинка, используемая в качестве фона

bgColor.....Цвет фона слоя

below.....Объект Layer, расположенный ниже данного слоя, или null, если данный слой является самым нижним

clip.bottom.....Нижний край прямоугольника зоны видимости слоя

clip.height.....Верхний край прямоугольника зоны видимости слоя

clip.left.....Левый край прямоугольника зоны видимости слоя

clip.right.....Правый край прямоугольника зоны видимости слоя

clip.top.....Высота зоны видимости

clip.width.....Ширина зоны видимости

document.....Документ, связанный со слоем

left.....Горизонтальное положение левого края слоя в пикселях по отношению к родительскому слою

name.....Имя слоя, указанное с помощью свойства ID в ярлыке <layer>

pageX.....Горизонтальная координата слоя в пикселях по отношению к странице

pageY.....Вертикальная координата слоя в пикселях по отношению к странице

parentLayer ...Родительский слой, который содержит данный слой, или окно, если нет родительского слоя

siblingAbove ...Слой, расположенный выше данного слоя в пределах одного родительского слоя, или null, если данный слой является самым высоким

siblingBelow... Слой, расположенный ниже данного слоя в пределах одного родительского слоя, или null, если данный слой является самым нижним

src.....Строка, указывающая на адрес URL содержимого, расположенного в данном слое

top.....Вертикальное расположение верхнего края слоя в пикселях по отношению к родительскому слою

visibility.....Является или нет данный слой видимым

window.....Объект window или объект frame, в котором содержится данный слой, вне зависимости от того, содержатся ли в данном слое вложенные слои

x.....Синоним для layer.left

y.....Синоним для layer.top

zIndex.....Относительный индекс слоя, указывающий его положение по отношению к соседям

Методы

captureEvents Устанавливает захват всех элементов заданного типа

handleEvent ...Активизирует средство обработки события

load.....Изменяет содержимое слоя на содержимое указанного файла

moveAbove.....Фиксирует данный слой поверх указанного слоя без изменения вертикального и горизонтального положения

moveBelow.....Фиксирует данный слой снизу от указанного слоя без изменения вертикального и горизонтального положения слоя

moveBy.....Изменяет положение слоя на указанные величины смещений в пикселях

moveTo.....Перемещает верхний левый угол окна к указанным координатам на экране

moveToAbsolute Изменяет положение слоя и перемешает его к указанным координатам на странице (но не по отношению к родительскому слою)

releaseEvents . Освобождает захваченные события указанного типа и осуществляет обработку событий согласно нормальной иерархии

resizeBy.....Изменяет размер слоя в соответствии с указанными величинами, Р на которые происходит изменение высоты и ширины в пикселях

resizeTo.....Изменяет размер слоя так, чтобы его высота и ширина равнялись указанным величинам в пикселях

routeEvent.....Передает захваченные события в соответствии с нормальной иерархией

Заключение

Вы прочли пособие по языку JavaScript, познакомились с примерами, почувствовали возможности этого языка. Вы приобрели некоторый опыт создания сценариев и, наверное, ощутили потребность дальнейшего овладения возможностями этого языка. Мы рекомендуем Вам продолжать изучение языка JavaScript, углублять полученные знания. Полезными окажутся в этом случае существующие немногочисленные справочные пособия, которые были использованы при создании данной книги и которые могут пригодиться в Вашей дальнейшей работе, например, книга **Р. Дарнелла JavaScript. Справочник,** выпущенная в издательстве "Питер" в 2000 году.

Всегда самую свежую информацию и нужные сведения, полезность которых невозможно переоценить, Вы найдете в Интернете на сайте разработчика JavaScript по адресу: http://developer.netscape.com/library/documentation/.