ГЛАВА 4. ОСНОВЫ JAVASCRIPT

§4.1. Первые шаги

Что такое JavaScript?

Язык программирования JavaScript является сценарным языком программирования, который применяется для придания интерактивности web-страницам. Сценарий - это программа, которая вызывается из HTML-документа или непосредственно в него вложена и исполняется на компьютере-клиенте[10]. Сценарии позволяют авторам дополнить их HTML-документы динамически изменяемыми свойствами и интерактивными возможностями, а именно:

- могут исполняться при загрузке документа и динамически изменять его содержимое;
- могут вызываться для реакции на такие события, как загрузка или выгрузка документа, движение мыши, нажатие клавиши и т.п.;
- могут использоваться для обработки данных, вводимых пользователем в элементы формы. В частности, они позволяют заполнять поля формы на основе содержимого других полей и/или контролировать правильность введенных данных;
- могут присоединяться к элементам форм (например, кнопкам) для создания графического пользовательского интерфейса.

Существуют два типа сценариев, которые могут быть присоединены к HTML-документу:

- сценарии, которые исполняются в процессе загрузки документа обозревателем. Эти сценарии помещаются в элемент SCRIPT. Для обозревателей, не поддерживающих сценарии, может быть задано альтернативное содержание элементом NOSCRIPT;
- сценарии, которые вызываются каждый раз, когда происходит определенное событие. Эти сценарии

связываются с соответствующими элементами через атрибуты обработки событий.

JavaScript - это не Java!

Поддержка сценариев для HTML-страниц появилась во второй версии браузера Netscape Navigator. Новый язык фирмы Netscape Communication главным образом использовался для контроля заполняемых пользователем форм не на стороне сервера (как это делалось ранее), а на стороне клиента. Так появился LiveScript. Его создателем считают Брендана Эйка (Brendan Eich). По времени появление LiveScript совпало с представлением компанией Sun своего языка Java, вокруг которого наблюдался ажиотаж. Netscape, решила использовать эту шумиху в своих целях, для популяризации нового языка; получила лицензию от Sun и переименовала LiveScript в JavaScript. Несмотря на название, это фактически был тот же LiveScript. Некоторые утверждают, что в язык были добавлены возможности Java, что позволило улучшить его. Хотя это не так. Те возможности, которые появились в JavaScript версий 1.1, 1.2 и т.д. появились бы и без влияния Java[1,7,10].

Так что JavaScript, это не диалект и не упрощенная версия языка Java, как думают некоторые, а самостоятельный язык сценариев.

Время шло, язык становился все более популярным среди разработчиков Web-страниц. К его дальнейшему развитию подключилась корпорация Microsoft, чьи обозреватели Internet Explorer поддерживают JavaScript, начиная с версии 3.0. Версия Microsoft получила название JScript, поскольку JavaScript является зарегистрированной маркой фирмы Netscape. И в его основе лежала не Java, а тот же самый LiveScript.

С этого момента развитие JavaScript расходится в разных направлениях: JavaScript от Netscape (версии 1.0, 1.1… 1.4) и MS JScript (версии 1.0…5.0)[10].

Если рассмотреть сейчас их последние версии, то можно убедиться, что это уже разные языки написания сценариев, объединенные одним синтаксисом, напоминающим язык Си. Так что

утверждение, что JavaScript поддерживается всеми браузерами, уже не выдерживает критики. Единственное, что их объединяет – это их основа LiveScript, та часть JavaScript, что является мостом между Netscape JavaScript и Microsoft JScript.

В 1996 г. ECMA (European Computer Manufacturers Association - ассоциация европейских производителей компьютеров, созданная в 1961 г. и занимающаяся разработкой стандартов для информационных и коммуникационных систем) приняла решение о стандартизации этого языка, и в июне 1997 г. была принята первая версия стандарта под названием ECMAScript (ECMA-262). В апреле 1998 г. этот стандарт был принят ISO (International Organization for Standardization - Международная федерация национальных стандартизующих организаций) в качестве международного под номером ISO/IEC 16262[8,10,16]. В последующем изложении будем основываться на третьей версии стандарта ECMA (декабрь 1999 г.), но использовать название JavaScript, поскольку после того, как Netscape перестал быть коммерческим проектом и работы над JavaScript сместились в область программного обеспечения с открытым исходным кодом, эта технология была практически полностью приведена к стандарту ECMAScript. В тоже время Microsoft сориентировала Jscript на несколько иную область применения в .NET Framework.

Запуск JavaScript

Что необходимо сделать, чтобы запускать скрипты, написанные на языке JavaScript? Для этого понадобится браузер, способный работать с JavaScript. На сегодняшний день поддержку JavaScript обеспечивают современные версии всех наиболее часто используемых браузеров (Internet Explorer, Mozilla Firefox, Safari, Google Chrome, Opera). С тех пор, как эти браузера стали широко распространенными, множество людей получили возможность работать со скриптами, написанными на языке JavaScript. Несомненно, это важный аргумент в пользу выбора языка JavaScript, как средства улучшения Web-страниц.

Размещение JavaScript на HTML-странице

Код скрипта JavaScript размещается непосредственно на HTML-странице. Чтобы увидеть, как делается, давайте рассмотрим следующий простой пример:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
  <script language="JavaScript">
    document.write("А это JavaScript!")
  </script>
<br>
Вновь документ HTML.
</body>
</html>
```

С первого взгляда пример напоминает обычный файл HTML. Единственное новшество здесь - конструкция:

```
  <script language="JavaScript">
    document.write("А это JavaScript!")
  </script>
```

Это действительно код JavaScript. Чтобы видеть, как этот скрипт работает, запишите данный пример как обычный файл HTML и загрузите его в браузер, имеющий поддержку языка JavaScript. В результате Вы получите 3 строки текста:

Это обычный HTML документ.
А это JavaScript!
Вновь документ HTML.

Данный скрипт не столь полезен - то же самое и более просто можно было бы написать на «чистом» языке HTML, это всего лишь демонстрация тэга <script>. Все, что стоит между тэгами <script> и

</script>, интерпретируется как код на языке JavaScript. Здесь демонстрируется использования инструкции document.write() - одной из наиболее важных команд, используемых при программировании на языке JavaScript. Команда document.write() используется, когда необходимо что-либо написать в текущем документе (в данном случае, таким является HTML-документ). Так эта небольшая программа на JavaScript в HTML-документе пишет фразу "А это JavaScript!".

Браузеры без поддержки JavaScript

А как будет выглядеть страница, если браузер не воспринимает JavaScript? Браузеры, не имеющие поддержки JavaScript, не воспринимают и тэг <script>. Они игнорируют его и печатают все стоящие вслед за ним коды как обычный текст. Иными словами, читатель увидит, как JavaScript код, приведенный в программе, окажется вписан открытым текстом прямо посреди HTML-документа. Разумеется, это «портит» внешний вид HTML документа. На этот случай имеется специальный способ скрыть исходный код скрипта от старых версий браузеров - использование тэга комментария из HTML - <! - - >. В результате новый вариант нашего исходного кода будет выглядеть как:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
  <script language="JavaScript">
  <!-- скрывает код от старых браузеров
    document.write("А это JavaScript!")  // -->
  </script>
<br>
Вновь документ HTML.
</body>
</html>
```

В этом случае браузер без поддержки JavaScript будет печатать:

Это обычный HTML документ.
Вновь документ HTML.

А без HTML-тэга комментария браузер, не поддерживающий JavaScript, напечатал бы:

Это обычный HTML документ.
document.write("А это JavaScript!")
Вновь документ HTML.

Пожалуйста, обратите внимание, что нельзя полностью скрыть исходный код JavaScript. То, что здесь происходит, имеет цель предотвратить распечатку кода скрипта на старых браузерах - однако, тем не менее, читатель сможет увидеть этот код посредством пункта меню 'View document source'. Не существует также и способа скрыть что-либо от просмотра в исходном коде (или увидеть, как выполнен тот или иной трюк)[16].

События

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя. Если он щелкает по некоторой кнопке, происходит событие "Click". Если указатель мыши пересекает какую-либо ссылку гипертекста - происходит событие MouseOver.

Существует несколько различных типов событий[3,4,9,10,16]. Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. Это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелчка - Click. Программа - обработчик событий, которую используется в данном случае, называется onClick. И она сообщает компьютеру, что нужно делать, если произойдет данное

событие. Приведенный ниже код представляет простой пример программы обработки события onClick:

```
<form>
<input type="button" value="Click me"
onClick="alert('bonjour!')">
</form>
```

Данный пример имеет несколько новых особенностей - рассмотрим их по порядку. Из примера видно, что здесь создается некая форма с кнопкой (как создаются формы, мы с вами рассмотрели в §2.9). Первая новая особенность - onClick="alert('bonjour!')" в тэге <input>. Как уже говорилось, этот атрибут определяет, что происходит, когда нажимают на кнопку. Таким образом, если имеет место событие Click, компьютер должен выполнить вызов alert('bonjour!'). Это и есть пример кода на языке JavaScript (Обратите внимание, что в этом случае даже не используется тэг <script>). Функция alert() позволяет создавать выпадающие окна. При ее вызове необходимо в скобках задать некую строку. В данном случае это 'bonjour!'. И это как раз будет тот текст, что появится в выпадающем окне. Таким образом, когда читатель когда щелкает на кнопке, данный скрипт создает окно, содержащее текст 'bonjour!'.

Некоторое замешательство может вызвать еще одна особенность данного примера: в команде document.write() использовались двойные кавычки ("), а в конструкции alert() - только одинарные. Почему? В большинстве случаев можно использовать оба типа кавычек[1,3,7]. Однако в последнем примере мы написали onClick="alert('bonjour!')" - то есть использовались и двойные, и одинарные кавычки. Если бы написано onClick="alert("bonjour!")", то компьютер не смог бы разобраться в таком скрипте, поскольку становится неясно, к которой из частей конструкции имеет отношение функция обработки событий onClick, а к которой - нет. Поэтому в данном случае необходимо использовать оба типа кавычек. Не имеет значения, в каком порядке используются кавычки - сначала двойные, а затем одинарные или наоборот. То есть можно точно так же написать и onClick='alert("bonjour!")'.

В скрипте возможно использовать множество различных типов функций обработки событий[3,8].

Функции

В большинстве программ на языке JavaScript будут использоваться функции. Многие из них представляют собой лишь способ связывания вместе несколько команд. В качестве примера будет рассмотрен скрипт, печатающий некий текст три раза подряд. Для начала рассмотрим простой подход:

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

document.write("Добро пожаловать на мою
страницу!<br>");
document.write(Это JavaScript!<br>");
document.write("Добро пожаловать на мою
страницу!<br>");
document.write("Это JavaScript!<br>");
// -->
</script>
</html>
```

И такой скрипт напишет следующий текст «*Добро пожаловать на мою страницу!Это JavaScript!»* три раза. Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Но это неэффективно, можно решить эту же задачу более удобным способом:

```
<html>
<script language="JavaScript">
<!-- hide

function myFunction() {
```

```
  document.write("Добро пожаловать на мою
страницу!<br>");
  document.write("Это JavaScript!<br>");
}

myFunction();
myFunction();
myFunction();

// -->
</script>
</html>
```

В этом скрипте определена некая функция, состоящая из следующих строк:

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

Все команды скрипта, что находятся внутри фигурных скобок - {} - принадлежат функции *myFunction()*. Это означает, что обе команды *document.write()* теперь связаны воедино и могут быть выполнены при вызове указанной функции. И действительно, в этом примере есть три вызова функции *myFunction()* сразу после того, как дали определение самой функции. В свою очередь, это означает, что содержимое этой функции (команды, указанные в фигурных скобках) было выполнено трижды. Особую важность использования функций в JavaScript предает возможность передачи переменных при вызове функции, что обеспечивает скриптам подлинную гибкость[8,16].

Функции могут также использоваться совместно с процедурами обработки событий. Рассмотрим следующий 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="Calculate"
onClick="calculation()">
</form>
</body>
</html>
```

Здесь при нажатии на кнопку осуществляется вызов функции calculation(). Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными x, y и result. Переменную можно определить с помощью ключевого слова var. Переменные могут использоваться для хранения различных величин - чисел, строк текста и т.д. Так строка скрипта var result= x + y; сообщает браузеру о том, что необходимо создать переменную result и поместить туда результат выполнения арифметической операции x + y (т.е. 5 + 12). После этого в переменную result будет размещено число 17. В данном случае команда alert(result) выполняет то же самое, что и alert(17). Иными словами, мы получаем выпадающее окно, в котором написано число 17.

☑ Вопросы для самоконтроля

1. Что такое сценарии? Какое значение они имеют в HTML документах?
2. Существует ли различие между JavaScript и Java?
3. Как размещается JavaScript код на HTML странице?
4. Что такое функции? Как используются функции в JavaScript?

5. Что такое события и обработчики событий в JavaScript?

§4.2. Документ HTML

Иерархия объектов в JavaScript

В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент предстает в виде объекта. И каждый такой объект может иметь определенные свойства и методы[4]. В свою очередь, язык JavaScript позволяет легко управлять объектами web-страницы, хотя для этого очень важно понимать иерархию объектов, на которые опирается разметка HTML. Как это все действует, рассмотрим на следующем примере. Рассмотрим простую HTML-страницу:

```
<html>
<head>
</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=" maths.pomorsu.ru">My homepage</a>
</center>
</body>
</html>
```

Как будет выглядеть страница на экране, представлено на рисунке 40.

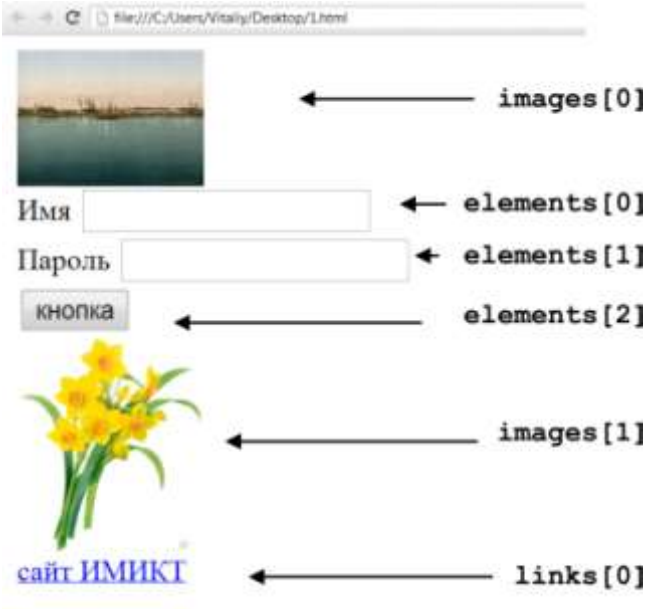

Рисунок 1. Отображение страницы на экране

Итак, есть два рисунка, одна ссылка и некая форма с двумя полями для ввода текста и одной кнопкой. С точки зрения языка JavaScript окно браузера - это некий объект window. Этот объект также содержит в свою очередь некоторые элементы оформления, такие как строка состояния. Внутрь окна можно поместить документ HTML (или файл какого-либо другого типа - однако пока мы все же ограничимся файлами HTML). Такая страница является ни чем иным, как объектом document. Это означает, что объект document представляет в языке JavaScript загруженный на настоящий момент документ HTML. К свойствам объекта document относятся, например, цвет фона для web-страницы. Однако для нас гораздо важнее то, что все без исключения объекты HTML являются свойствами объекта document. Примерами объекта HTML являются, к примеру, ссылка или заполняемая форма. На следующем рисунке 41 иллюстрируется иерархия объектов, созданная HTML-страницей из примера:

Рисунок 2. Иерархия объектов, созданная HTML-страницей

Разумеется, должна быть возможность получения информации о различных объектах в этой иерархии и управлению ею. Для этого необходимо понимать, как в языке JavaScript организован доступ к различным объектам. Как видно из изображения, каждый объект иерархической структуры имеет свое имя. Следовательно, если требуется узнать, как можно обратиться к первому рисунку на HTML-странице, то необходимо сориентироваться в иерархии объектов. И начать нужно с самой вершины. Первый объект такой структуры называется document. Первый рисунок на странице представлен как объект images[0]. Это означает, что отныне получать доступ к этому объекту можно, записав в JavaScript document.images[0]. Если же, например, необходимо узнать, какой текст ввел читатель в первый элемент формы, то сперва следует выяснить, как получить доступ к этому объекту. И снова это начинается с вершины иерархии объектов. Затем прослеживается путь к объекту с именем elements[0] и последовательно записываются названия всех объектов, через которые он проходит. В итоге выясняется, что доступ к первому полю для ввода текста можно получить, записав:

```
document.forms[0].elements[0]
```

А теперь как узнать текст, введенный читателем? Чтобы выяснять, которое из свойств и методов объекта позволят получить доступ к этой информации, необходимо обратиться к какому-либо справочнику по JavaScript. Там найдется, что элемент, соответствующий полю для ввода текста, имеет свойство value, которое как раз и соответствует

введенному тексту. Итак, теперь известно все необходимое, чтобы прочитать искомое значение. Для этого нужно написать на языке JavaScript строку:

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

Полученная строка заносится в переменную name. Следовательно, теперь можно работать с этой переменной, как необходимо. Например, можно создать выпадающее окно, воспользовавшись командой alert("Привет " + name). В результате, если читатель введет в это поле слово 'Андрей', то по команде alert("Привет " + name) будет открыто выпадающее окно с приветствием 'Привет Андрей'.

При работе с большими страницами процедура адресации к различным объектам по номеру может стать весьма запутанной. Например, придется решать, как следует обратиться к объекту document.forms[3].elements[17] или document.forms[2].elements[18]? Во избежание подобной проблемы, можно присваивать различным объектам уникальные имена. Как это делается, продемонстрировано в примере ниже:

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

Эта запись означает, что объект forms[0] получает теперь еще и второе имя - myForm. Точно так же вместо elements[0] можно написать name (последнее было указано в атрибуте name тэга <input>). Таким образом, вместо

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

Можно просто записать

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

Это значительно упрощает программирование на JavaScript, особенно в случае с большими web-страницами, содержащими множество объектов. (Обратите внимание, что при написании имен необходимо следить и за положением регистра - то есть нельзя написать *myform* вместо *myForm*).

В JavaScript многие свойства объектов доступны не только для чтения, также существует возможность записывать в них новые значения. Например, посредством JavaScript можно записать в уже упоминавшееся поле новую строку. Пример кода на JavaScript, иллюстрирующего такую возможность, представлен ниже. Здесь выполняющий этот фрагмент записан как свойство onClick второго тэга <*input*>:

```
<form name="myForm">
<input type="text" name="input" value="текст">
<input type="button" value="Write"
  onClick="document.myForm.input.value= 'Yo!'; ">
```

Еще один пример исходного кода скрипта:

```
<html>
<head>
<title>Objects</title>

<script language="JavaScript">
<!-- hide

function first() {

 // создает выпадающее окно, где размещается
// текст, введенный в поле формы

  alert("The value of the textelement is: " +
    document.myForm.myText.value);
}

function second() {
```

```
  // данная функция проверяет состояние
переключателей
  var myString= "The checkbox is ";

  // переключатель включен, или нет?
  if (document.myForm.myCheckbox.checked)
myString+= "checked"
    else myString+= "not checked";

  // вывод строки на экран
  alert(myString);
}
// -->
</script>
</head>
<body bgcolor=lightblue>
<form name="myForm">
<input type="text" name="myText" value="bla bla
bla">
<input type="button" name="button1" value="Button
1"
  onClick="first()">
<br>
<input type="checkbox" name="myCheckbox" CHECKED>
<input type="button" name="button2" value="Button
2"
  onClick="second()">
</form>
<p><br><br>
<script language="JavaScript">
<!-- hide
document.write("The background color is: ");
document.write(document.bgColor + "<br>");
document.write("The text on the second button is:
");
document.write(document.myForm.button2.value);
// -->
</script>
</body>
</html>
```

Объект location

Кроме объектов window и document в JavaScript имеется еще один важный объект - location. В этом объекте представлен адрес загруженного HTML-документа. Например, если загрузить страницу http://www.narfu.ru/page.html, то значение location.href как раз и будет соответствовать этому адресу. Впрочем, для гораздо важнее существование возможности записывать в location.href новые значения. Например, в данном примере кнопка загружает в текущее окно новую страницу:

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

Страница, управляемая при помощи мыши

Реакция на наведение

Доступ к свойствам текущего элемента осуществляется с помощью ключевого слова this[3,9]:

```
<DIV onMouseOver="this.style.color=green">Этот
текст изменит свой цвет, если навести на него
мышь!</DIV>
```

Если теперь открыть эту страницу в браузере, то при наведении указателя мыши на вторую строку текста, цвет строки действительно изменится на зеленый. Однако, один раз изменившись, он так и останется зеленым. Чтобы при убирании указателя мыши со строки цвет изменился обратно на черный, необходимо добавить обработчик событий, реагирующий на увод указателя. Он называется onMouseOut:

```
<DIV onMouseOver="this.style.color=green”
onMouseOut="this.style.color=black">Этот текст
изменит свой цвет, если навести на него мышь!</DIV>
```

Теперь при наведении указателя мыши на эту строку, ее цвет изменится на зеленый, а при уводе указателя — обратно на черный. Можно также использовать и доступ по названию элемента. Например, если установить в этом блоке атрибут ID="text1", то можно будет написать так:

```
<DIV ID="textl"
onMouseOver="textl.style.color='green'"
onMouseOut="textl.style.color='black'"> Этот текст
изменит цвет, если навести на него мышь!</DIV>
<DIV ID="textl"
onMouseOver="document.all.textl.style.color=
'green'"
onMouseOut="document.all.textl.style.color=
”'black'">Этот изменит свой цвет, если навести на
него мышь!</DIV>
```

Здесь document.all - обращение ко всем элементам, расположенным на странице.

Обратите внимание на то, что внутри кавычек расположен текст, написанный на языке JavaScript. Чтобы не загромождать текст HTML-документа, можно заранее определить соответствующие функции в разделе <HEAD>:

```
<HEAD>
<TITLE>Обработка событий мыши</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function change() { document. all. textl. style.
color="green";}
function change2() {
document.all.textl.style.color="black";}
 //--> </SCRIPT> </HEAD>
```

А при определении обработчиков событий писать только имена функций:

```
<DIV ID="textl" onMouseOver="change();"
onMouseOut="change2();">Этот текст изменит свой
цвет, если навести на него мышь!</DIV>
```

Таким же образом можно изменить не только тот блок текста, на который мы навели указатель мыши, но и любые другие элементы, если только присвоить им имя. Например, если мы хотим одновременно с изменением цвета второй строки на зеленый, изменять цвет третьей строки, скажем, на оранжевый, достаточно будет сначала присвоить третьей строке имя:

```
<DIV ID="text2">Этот текст изменит свой цвет, если
мышь навести на вторую строку!</DIV>
```

а потом соответствующим образом изменить функции:

```
function change() {
document.all.textl.style.color="green";
document.all.text2.style.color="orange";}
function change2()
{document.all.textl.style.coior="black";
document.all.text2.style.color="black"; }
```

Кнопки, влияющие на вид страницы

Теперь рассмотрим такой пример. Предположим, на web-странице с градиентным фоном размешена сказка. Однако, возможно, кому-то этот фон может помешать читать текст, и, заботясь о пользователе, хочется предусмотреть кнопку, при нажатии на которую фон становился бы равномерно зеленоватым. Для пользователей, пред почитающих читать черные буквы на белом фоне, можно предусмотреть возможность смены цвета фона на белый.

Для начала в соответствии с требованиями HTML 4.0 заменим атрибуты BGCOLOR= и BACKGROUND= тега <BODY> на соответствующие стилевые свойства:

```
<STYLE> BODY { background-color: #BFFFBF;
background-image: url("Images/gradi.jpg"); }
</STYLE>
```

Что же касается тега <BODY> , то ему желательно присвоить имя для облегчения доступа к его свойствам:

```
<BODY ID="doc">
```

Теперь добавим сверху две кнопки: одну для выключения фонового рисунка, а другую — для смены цвета фона на белый:

```
<DIV ALIGN="center"> <INPUT TYPE="button"
VALUE="Убрать фоновый рисунок">
<INPUT TYPE="button" VALUE="Сделать фон белым">
</DIV>
```

Кнопки созданы, но пока при их нажатии ничего не происходит. Нам надо написать функцию, убирающую фоновый рисунок. Для этого нужно всего лишь стилевому свойству background-image присвоить значение none:

```
function
noBg(){document.all.doc.style.backgroundImage='none
';}
```

Обратите внимание на то, что в тексте на языке JavaScript (а не на языке CSS) нужно обязательно преобразовать название стилевого свойства с дефисом, как объяснялось выше[3,4].

Теперь нужно сделать так, чтобы функция noBg() выполнялась при щелчке мыши на первой из кнопок. Для этого надо в соответствующий тег кнопки добавить обработчик событий, реагирующий на щелчок мыши. Он называется onClick:

```
<INPUT TYPE="button" VALUE="убрать фоновый рисунок"
onClick="noBg()">
```

Аналогично создается функция для смены цвета фона на белый:

```
function colChange() (
document.all.doc.style.backgroundColor='white'; }
```

и добавим ко второй кнопке обработчик события onClick:

```
<INPUT TYPE="button" VALUE="Сделать фон белым"
onClick= "colChange()">
```

Если теперь открыть эту страницу в браузере, то при нажатии на кнопку «Убрать фоновый рисунок» градиентный фоновый перелив исчезнет, уступив место зеленоватому цвету, а при нажатии на кнопку «Сделать фон белым» фон страницы действительно станет белым.

☑ Вопросы для самоконтроля

1. Для чего нужна иерархия объектов в JavaScript?
2. Для чего нужны элементы document, window, locatio?
3. Как размещается JavaScript код на HTML странице?
4. Какие обработчики событий в JavaScript вы знаете?

§4.3. Фреймы и JavaScript

Создание фреймов

Напомним, что фрейм определяется как некое выделенное в окне браузера поле в форме прямоугольника. Каждый из фреймов выдает на экран содержимое собственного документа (в большинстве случаев это документы HTML). Для создания фреймов необходимо два тэга: <frameset> и <frame>. HTML-страница, создающая два фрейма, в общем случае может выглядеть следующим образом:

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

```
</frameset>
</html>
```

В результате будут созданы два фрейма. Во фрейме <frameset> используется свойство rows. Это означает, два фрейма будут расположены друг над другом. В верхний фрейм будет загружена HTML-страница page1.htm, а в нижнем фрейме разместится документ page2.htm. Фрагмент "50%,50%" сообщает, насколько велики должны быть оба получившихся окна.

Любому фрейму можно присвоить уникальное имя, воспользовавшись в тэге <frame> атрибутом name. Такая возможность пригодится в языке JavaScript для доступа к фреймам. При создании web-страниц можно использовать несколько вложенных тэгов <frameset>.

```
<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>
```

Толщину границы между фреймами можно задать воспользовавшись в тэге <frameset> параметром border. Запись border=0 означает, что между тэгами нет какой-либо границы.

Фреймы и JavaScript

Рассмотрим, как JavaScript «видит» фреймы, присутствующие в окне браузера. Для этой цели будут созданы два фрейма, как было описано в первом примере. JavaScript организует все элементы, представленные на web-странице, в виде некой иерархической

структуры. То же самое относится и к фреймам. На рисунке 42 показана иерархия объектов, представленных в первом примере:

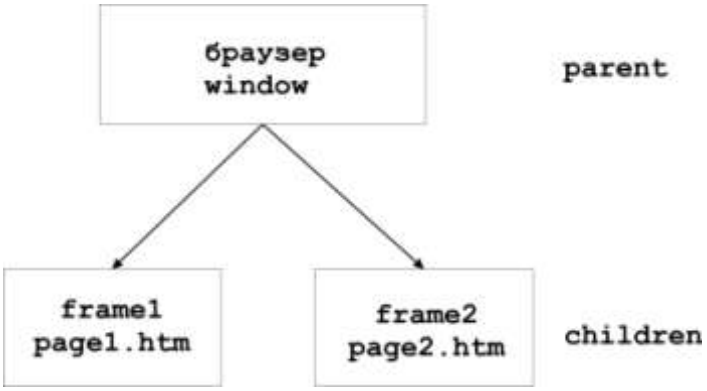

Рисунок 3. Иерархия объектов

В вершине иерархии находится окно браузера (window), в данном случае разбитое на два фрейма. Таким образом, окно, как объект, является родоначальником, родителем данной иерархии (parent), а два фрейма - соответственно, его потомки (children). Этим двум фреймам присвоены уникальные имена - *frame1* и *frame2*. И с помощью этих имен можно обмениваться информацией с двумя указанными фреймами.

С помощью скрипта можно решить следующую задачу: допустим, посетитель активирует некую ссылку в первом фрейме, однако соответствующая страница должна загружаться не в этот же фрейм, а в другой. Примером такой задачи может служить составление меню (или навигационных панелей), где один фрейм всегда остается неизменным, но предлагает посетителю несколько различных ссылок для дальнейшего изучения данного сайта[16].

Для решения этой задачи будут рассмотрены на три случая:

- главное окно/фрейм получает доступ к фрейму-потомку;
- фрейм-потомок получает доступ к родительскому окну/фрейму;
- фрейм-потомок получает доступ к другому фрейму-потомку.

С точки зрения объекта «окно» (window) два указанных фрейма называются frame1 и frame2. Как можно видеть на рисунке 43, существует прямая взаимосвязь между родительским окном и каждым фреймом. Так образом, если скрипт создается для родительского окна -

то есть для страницы, создающей эти фреймы - то обратиться к этим фреймам можно, просто называя их по имени. Например, можно написать:

```
frame2.document.write("Это сообщение передано от
родительского окна.");
```

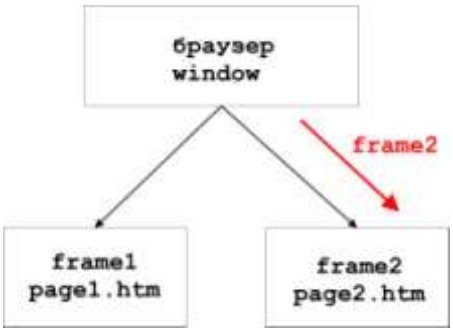

Рисунок 4. Взаимосвязь между родительским окном и фреймом

В некоторых случаях может понадобиться, находясь во фрейме, получить доступ к родительскому окну. Например, это бывает полезным, если при следующем переходе необходимо избавиться от фреймов. В таком случае удаление фреймов означает лишь загрузку новой страницы вместо содержавшей фреймы. В данном случае это загрузка страницы в родительское окно. Сделать это поможет доступ к родительскому- parent - окну (или родительскому фрейму) из фреймов, являющихся его потомками, как показано на рисунке 44. Чтобы загрузить новый документ, необходимо внести в location.href новый адрес URL. Поскольку мы хотим избавиться от фреймов, следует использовать объект location из родительского окна. (Напомним, что в каждый фрейм можно загрузить собственную страницу, и для каждого фрейма существует собственный объект location). Итак, можно загрузить новую страницу в родительское окно с помощью команды:

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

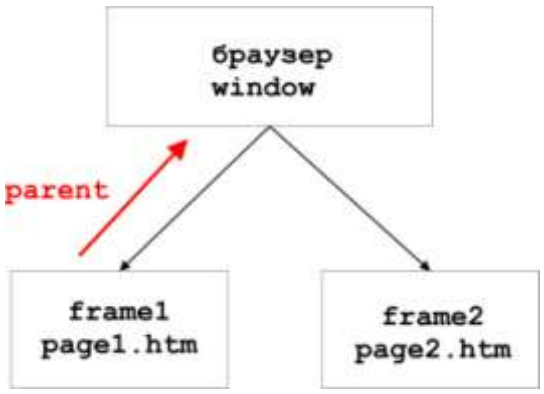

Рисунок 5. Доступ к родительскому окну через фремы

И, наконец, очень часто приходится решать задачу обеспечения доступа с одного фрейма-потомка к другому такому же фрейму-потомку. И так, как можно, находясь в первом фрейме, записать что-либо во второй - то есть, какой командой следует воспользоваться на HTML-странице page1.htm? Как видно из рисунка, между двумя этими фреймами нет никакой прямой связи. И потому нельзя просто так вызвать frame2, находясь во фрейме frame1, который просто ничего не знает о существовании второго фрейма. С точки же зрения родительского окна второй фрейм действительно существует и называется frame2, а к самому родительскому окну можно обратиться из первого фрейма по имени parent, как показано на рисунке 43. Таким образом, чтобы получить доступ к объекту document, разместившемуся во втором фрейме, необходимо написать следующее:

```
parent.frame2.document.write("Привет, это вызов из
первого фрейма.");
```

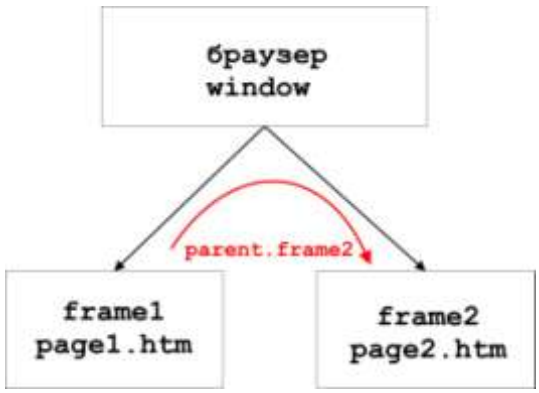

Рисунок 6. Обращение к фрейму через фрейм

Навигационные панели

Далее будет рассмотрено, как создаются навигационные панели. В одном фрейме создается несколько ссылок. Однако, если посетитель активирует какую-либо из них, соответствующая страница будет помещена не в тот же самый фрейм, а в соседний. В первую очередь необходимо написать скрипт, создающий указанные фреймы. Такой документ frames3.htm выглядит точно так же, как и тот, что рассматривался ранее:

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

Здесь start.htm - это та страница, которая первоначально будет показана в главном фрейме (main). Никаких специальных требований к содержимому этой страницы нет. Следующая web-страница будет загружена во фрейм "menu" menu.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 будет рассмотрено несколько способов загрузки новой страницы во фрейм main. В первой ссылке для этой цели используется функция load(). Следующий пример демонстрирует, как это делается:

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

Как видно из этого примера, вместо явной загрузки новой страницы браузеру предлагается выполнить некую команду на языка JavaScript - для этого необходимо всего лишь воспользоваться параметром JavaScript, вместо обычного href. Далее, внутри скобок можно увидеть 'first.htm'. Эту строка передается в качестве аргумента функции load().

Сама же функция load() определяется следующим образом:

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

Здесь показано, что внутри скобок написано url. Это означает, что в этом примере строка 'first1.htm' при вызове функции заносится в переменную url. И эту новую переменную теперь можно использовать при работе внутри функций load(). Позже будут продемонстрированы другие примеры использования важной концепции переменных.

Во второй ссылке присутствует параметр target. На самом деле это уже не относится к JavaScript. Это одна из конструкций языка HTML. Как видно из примера, там всего лишь указывается имя необходимого фрейма. Заметим, что в этом случае не обязательно ставить перед именем указанного фрейма слово parent, что несколько смущает. Причина такого отступления от правил кроется в том, что параметр target - это функция языка HTML, а не JavaScript.

И на примере третьей ссылки можно увидеть, как с помощью target можно избавиться от фреймов. И если следует избавиться от

фреймов с помощью функции load(), то необходимо написать в ней лишь parent.location.href= url.

Итак, какой способ следует выбрать? Это зависит от скрипта и от того, что собственно необходимо сделать. Параметр target использовать очень просто. Им можно воспользоваться, если необходимо всего лишь загрузить новую страницу в другой фрейм.

Решение на основе языка JavaScript (примером этого служит первая ссылка) обычно используется, если необходимо, чтобы при активизации ссылки произошло несколько вещей. Одна из наиболее часто возникающих проблем такого рода состоит в том, чтобы разом загрузить две страницы в два различных фрейма. И хотя эту задачу можно решить с помощью параметра target, использование функции JavaScript в этом случае более предпочтительно. Предположим, существует три фрейма с именами frame1, frame2 и frame3. Допустим, посетитель активирует ссылку в frame1. И требуется, чтобы при этом в два других фрейма загружались две различные web-страницы. В качестве решения этой задачи можно, например, воспользоваться функцией:

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

Если же необходимо сделать функцию более гибкой, то можно воспользоваться возможностью передачи переменной в качестве аргумента. Результат будет выглядеть как:

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

После этого можно организовать вызов функции: loadtwo("first.htm", "second.htm") или loadtwo("third.htm", "forth.htm"). Очевидно, передача аргументов делает функцию более гибкой. В

результате ее можно использовать многократно и в различных контекстах.

☑ Вопросы для самоконтроля

1. Что такое фреймы? В чем особенность работы с фреймами в JavaScript?
2. Какие виды обращений между родителями и потомками существуют?
3. Как обратиться из родительского окна к фрейму?
4. Как обратиться из фрейма к родительскому окну?
5. Как обратиться из фрейма к фрейму?
6. В чем особенность создания навигационной панели?

§4.4. Окна и динамически создаваемые документы

Создание окон

Открытие новых окон в браузере - грандиозная возможность языка JavaScript. Можно либо загружать в новое окно новые документы (например, те же документы HTML), либо (динамически) создавать новые материалы. Посмотрим сначала, как можно открыть новое окно, потом как загрузить в это окно HTML-страницу и, наконец, как его закрыть.

Приводимый далее скрипт открывает новое окно браузера и загружает в него некую web-страничку:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin() {
  myWin= open("1.htm");
}
// -->
</script>
</head>
```

```
<body>
<form>
<input type="button" value="Открыть новое окно"
onClick="openWin()">
</form>
</body>
</html>
```

В представленном примере в новое окно с помощью метода open() записывается страница 1.htm.

Заметим, что имеется возможность управлять самим процессом создания окна. Например, можно указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. Кроме того, можно задать размер окна. Например, в следующем скрипте открывается новое окно размером 300x300 пикселей. Оно не имеет ни строки статуса, ни панели инструментов, ни меню.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin2() {
  myWin= open("1.htm", "displayWindow", "width=300,
height=300, status=no, toolbar=no, menubar=no");
}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Открыть новое окно"
onClick="openWin2()">
</form>
</body>
</html>
```

Как видно из примера, свойства окна формулируются в строке "width=300, height=300, status=no, toolbar=no, menubar=no".

Метод open создает новое окно обозревателя и возвращает ссылку на него[3,10].

```
window.open(uri, имя [, свойства?][, флаг?])
```

Аргументы: uri, имя, свойства — строковые значения , флаг — логическое значение:

- аргумент uri задает URI открываемого документа,
- аргумент имя — имя фрейма для нового окна, которое может использоваться в атрибуте target элементов <A> и <FORM>. Так, если известно имя окна, то можно загрузить туда новую страницу с помощью записи

```
<a href="1.html" target="displayWindow">
```

При этом необходимо указать имя соответствующего окна (если же такого окна не существует, то с этим именем будет создано новое).

- Необязательный аргумент флаг указывает, нужно ли создавать для новой web-страницы отдельную запись в истории просмотра. Если он равен true, то новая запись не создается; вместо этого новая страница замещает в истории текущую.
- Необязательный аргумент свойства представляет собой список свойств нового окна, разделенных запятыми. В нем мы можем задать следующие свойства, представленные в таблице 42:

Таблица 1 - Свойства окна

Свойство	**Описание**	**Допустимые значения**
channelmode	Показывать панель каналов.	yes|no
directories	Показывать панель ссылок обозревателя.	yes|no
fullscreen	Открыть окно в полноэкранном режиме.	yes|no

Свойство	Описание	Допустимые значения
height	Высота окна в пикселях.	количество пикселей
left	Расстояние в пикселях от левого края экрана по горизонтали.	количество пикселей
location	Показывать адресную строку обозревателя.	yes|no
menubar	Показывать меню обозревателя.	yes|no
resizable	Пользователь может изменять размеры окна.	yes|no
scrollbars	Показывать полосы прокрутки окна.	yes|no
status	Показывать строку состояния обозревателя.	yes|no
titlebar	Показывать заголовок окна (только из HTA).	yes|no
toolbar	Показывать панель кнопок обозревателя.	yes|no
top	Расстояние в пикселях от верхнего края экрана по вертикали.	количество пикселей
width	Ширина окна в пикселях.	количество пикселей

Свойствам width, height, left и top должны быть присвоены числовые значения. Остальные свойства являются логическими; им можно присваивать значения yes или no (или, что то же самое, 1 или 0). По умолчанию все свойства имеют значение yes.

```
myWin= open("1.htm", "displayWindow", "width=300,
height=300, status=no, toolbar=no, menubar=no");
```

Обратите внимание, что myWin - это вовсе не имя окна. Но только с помощью этой переменной можно получить доступ к окну. И, поскольку это обычная переменная, то область ее действия - лишь тот скрипт, в котором она определена. А между тем, имя окна (в данном

случае это displayWindow) - уникальный идентификатор, которым можно пользоваться с любого из окон браузера.

Закрытие окон

Окна можно также закрывать с помощью языка JavaScript. Чтобы сделать это, используется метод close(). Рассмотрим на примере, откроем новое окно и загрузим туда очередную страницу:

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

Если теперь в новом окне нажать кнопку, то оно будет закрыто. open() и close() - это методы объекта window. Следует помнить, что необходимо писать не просто open() и close(), а window.open() и window.close(). Однако в данном случае объект window можно опустить - нет необходимости писать префикс window, если следует вызвать лишь один из методов этого объекта (и такое возможно только для этого объекта).

Динамическое создание документов

В этом параграфе будет рассмотрена замечательная возможность JavaScript — динамическое создание документов. То есть можно разрешить скрипту на языке JavaScript самому создавать новые HTML-страницы. Более того, можно таким же образом создавать и другие документы web, такие как VRML-сцены и т.д. Для удобства можно размещать эти документы в отдельном окне или фрейме.

Для начала создадим простой HTML-документ, который покажем в новом окне и рассмотрим следующий скрипт.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin3() {
  myWin= open("", "displayWindow",
"width=500,height=400,status=yes,toolbar=yes,menuba
r=yes");
  // открыть объект document для последующей печати
  myWin.document.open();
  // генерировать новый 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("This HTML-document has been
created ");
  myWin.document.write("with the help of
JavaScript!");
  myWin.document.write("</font></center>");
  myWin.document.write("</body></html>");
  // закрыть документ - (но не окно!)
  myWin.document.close();
}
// -->
</script>
</head>
<body>
<form>
```

```
<input type=button value="On-the-fly"
onClick="openWin3()">
</form>
</body>
</html>
```

Рассмотрим функцию winOpen3(). Очевидно, сначала открывается новое окно браузера. Поскольку первый аргумент функции open() - пустая строка (""), то это значит, что в данном случае нет необходимости указывать конкретный адрес URL. Браузер должен не только обработать имеющийся документ - JavaScript обязан создать дополнительно новый документ.

В скрипте определяем переменную myWin. И с ее помощью получаем доступ к новому окну. Обратите, пожалуйста, внимание, что в данном случае нет возможности воспользоваться для этой цели именем окна (displayWindow).

После того, как открыли окно, наступает очередь открыть для записи объект document. Делается это с помощью команды:

```
// открыть объект document для последующей печати
myWin.document.open();
```

Здесь обращение идет к open() - методу объекта document. Однако это совсем не то же самое, что метод open() объекта window! Эта команда не открывает нового окна - она лишь готовит document к предстоящей печати. Кроме того, перед document.open() необходимо поставить приставку myWin, чтобы получить возможность писать в новом окне.

В последующих строках скрипта с помощью вызова document.write() формируется текст нового документа:

```
// генерировать новый 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("This HTML-document has been
created ");
  myWin.document.write("with the help of
JavaScript!");
  myWin.document.write("</font></center>");
  myWin.document.write("</body></html>");
```

Как видно, здесь в документ записываются обычные тэги языка HTML. То есть фактически генерируется разметка HTML! При этом можно использовать абсолютно любые тэги HTML.

По завершении этого мы обязаны вновь закрыть документ. Это делается следующей командой:

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

Можно не только динамически создавать документы, но и по своему выбору размещать их в том или ином фрейме. Например, если есть два фрейма с именами frame1 и frame2, а теперь во frame2 необходимо сгенерировать новый документ, то для этого в frame1 достаточно будет написать следующее:

```
parent.frame2.document.open();
parent.frame2.document.write("Here goes your HTML-
code");
parent.frame2.document.close();
```

☑ Вопросы для самоконтроля

1. Каким образом открываются новые окна в JavaScript?
2. Каким образом закрываются окна в JavaScript?
3. Особенности работы с динамическими окнами в JavaScript?
4. В каких случаях объект window можно опустить?
5. В чем состоит различие между методами open() и close() объектов document и window?

§4.5. Строка состояния, таймеры и время

Строка состояния

Составленные программы на JavaScript могут выполнять запись в строку состояния - прямоугольник в нижней части окна браузера. Все, что необходимо для этого сделать - всего лишь записать нужную строку в *window.status*. В следующем примере создаются две кнопки, которые можно использовать, чтобы записывать некий текст в строку состояния и, соответственно, затем его стирать.

Данный скрипт 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="Писать!"
    onClick="statbar('Привет! Это окно
состо\яни\я!');">
  <input type="button" name="erase"
value="Стереть!"
    onClick="statbar('');">
</form>
</body>
</html>
```

Итак, создается форма с двумя кнопками. Обе эти кнопки вызывают функцию statbar(). Вызов от клавиши «Писать!» выглядит следующим образом:

```
statbar('Привет! Это окно состо\яни\я!');
```

В скобках написана строка: 'Привет! Это окно состо\яни\я!'. Это как раз и будет текст, передаваемый функции statbar(). В свою очередь, можно видеть, что функция statbar() определена следующим образом:

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

В заголовке функции в скобках поместили слово txt. Это означает, что строка, которую передали этой функции, помещается в переменную txt.

Передача функциям переменных - прием, часто применяемый для придания функциям большей гибкости. Можно передать функции несколько таких аргументов - необходимо лишь отделить их друг от друга запятыми[16].

Строка txt заносится в строку состояния посредством команды window.status = txt.

Соответственно, удаление текста из строки состояния выполняется как запись в window.status пустой строки.

Механизм вывода текста в строку состояния удобно использовать при работе со ссылками. Вместо того, чтобы выводить на экран URL данной ссылки, можно просто на словах объяснять, о чем будет говориться на следующей странице. Так link демонстрирует это - достаточно лишь поместить указатель вашей мыши над этой ссылкой.

Исходный код этого примера выглядит следующим образом:

```
<a href="dontclck.htm"
onMouseOver="window.status='Don\'t click me!';
return true;"
 onMouseOut="window.status='';">link</a>
```

Здесь используется процедурами *onMouseOver* и *onMouseOut*, чтобы отслеживать моменты, когда указатель мыши проходит над данной ссылкой.

Может возникнуть вопрос, а почему в onMouseOver необходимо возвращать результат true. На самом деле это означает, что браузер не

должен вслед за этим выполнять свой собственный код обработки события MouseOver. Как правило, в строке состояния браузер показывает URL соответствующей ссылки. Если же не возвратить значение true, то сразу же после того, как код будет выполнен, браузер перепишет строку состояния на свой лад - то есть текст будет тут же затерт и читатель не сможет его увидеть. В общем случае, всегда можно отменить дальнейшую обработку события браузером, возвращая true в своей собственной процедуре обработки события. В этом скрипте можно увидеть еще одну вещь - в некоторых случаях необходимо печатать символы кавычек. Например, следует напечатать текст Don't click me - однако поскольку эта строка передается в процедуру обработки события onMouseOver, то используем для этого одинарные кавычки. Между тем, как слово Don't тоже содержит символ одинарной кавычки! И в результате, если просто можно написать 'Don't ...', браузер запутается в этих символах апострофов. Чтобы разрешить эту проблему, достаточно лишь поставить обратный слэш «\» перед символом кавычки - это означает, что данный символ предназначен именно для печати (Тоже самое Вы можете делать и с двойными кавычками).

Таймеры

С помощью функции Timeout (или таймера) можно запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени. В следующем скрипте демонстрируется кнопка, которая открывает выпадающее окно не сразу, а по истечении 3 секунд.

Скрипт выглядит следующим образом:

```
<script language="JavaScript">
<!-- hide
function timer() {
  setTimeout("alert('Врем\я истекло!')", 3000);}
// -->
</script>
...
```

```
<form>
  <input type="button" value="Timer"
onClick="timer()"> </form>
```

Здесь setTimeout() - это метод объекта window. Он устанавливает интервал времени — несложно догадаться, как это происходит. Первый аргумент при вызове - это код JavaScript, который следует выполнить по истечении указанного времени. В нашем случае это вызов - "alert('Врем\я истекло!')". Обратите, пожалуйста, внимание, что код на JavaScript должен быть заключен в кавычки. Во втором аргументе компьютеру сообщается, когда этот код следует выполнять. При этом время надо указывать в миллисекундах (3000 миллисекунд = 3 секунды).

Прокрутка

Теперь, когда известно, как делать записи в строке состояния и как работать с таймерами, можно перейти к управлению прокруткой. Уже было продемонстрировано, как текст перемещается строке состояния. В Интернет этим приемом пользуются повсеместно. Теперь же рассмотрим, как можно запрограммировать прокрутку в основной линейке. Рассмотрим также и всевозможные усовершенствования этой линейки.

Создать бегущую строку довольно просто. Для начала изучим, как вообще можно создать в строке состояния перемещающийся текст - бегущую строку. Очевидно, в первую очередь необходимо записать в строку состояния некий текст. Затем по истечении короткого интервала времени следует записать туда тот же самый текст, но при этом немного переместив его влево. Если сделать это несколько раз, то у пользователя создастся впечатление, что он имеет дело с бегущей строкой.

Однако при этом следует помнить еще и о том, что необходимо каждый раз вычислять, какую часть текста следует показывать в строке состояния (как правило, объем текстового материала превышает размер строки состояния).

Эта кнопка откроет окно и покажет образец прокрутки (в исходный код скрипта добавлены еще некоторые комментарии):

```
<html>
<head>
<script language="JavaScript">
<!-- hide
// инициализация текста прокрутки
// объявление переменных и задание им значений
var scrtxt = "Это JavaScript! " +"Это JavaScript! "
+"Это JavaScript!";
var len = scrtxt.length;\\вычисляем длину строки
var width = 100;
var pos = -(width + 2);
function scroll() {
 // напечатать заданный текст справа и установить
таймер
// перейти на исходную позицию для следующего шага
  pos++;
// вычленить видимую часть текста
  var scroller = "";
  if (pos == len) {\\\ конструкция условного
оператора
    pos = -(width + 2);
  }
  // если текст еще не дошел до левой границы, то
мы должны
  // добавить перед ним несколько пробелов. В
противном случае мы должны
  // вырезать начало текста (ту часть, что уже ушла
за левую границу
  if (pos < 0) {
    for (var i = 1; i <= Math.abs(pos); i++)
{\\\конструкция цикла со счетчиком
      scroller = scroller + " ";}
    scroller = scroller + scrtxt.substring(0, width
- i + 1);
  }
  else {
    scroller = scroller + scrtxt.substring(pos,
width + pos);
  }
```

```
  // разместить текст в строке состо\яни\я
  window.status = scroller;
  // вызвать эту функцию вновь через 100
миллисекунд
  setTimeout("scroll()", 100);
}
// -->
</script>
</head>
<body onLoad="scroll()">
Это пример прокрутки в строке состояния средствами
JavaScript.
</body>
</html>
```

Большая часть функции scroll() нужна для «вычленения» той части текста, которая будет показана пользователю.

Рассмотрим использованные функции: scrtxt.substring(m, n) – копирует из строки scrtxt, начиная с символа m, ровно n символов и возвращает в качестве результата.

Для соединения строк используется символ «+».

Функция Math.abs(pos)- используется для вычисления модуля числа pos.

Функция scrtxt.length – определяет длину строки и возвращает ее в качестве результата.

Чтобы запустить этот процесс, используется процедура обработки события onLoad, описанная в тэге <body>. То есть функция scroll() будет вызвана сразу же после загрузки HTML-страницы. Через посредство процедуры onLoad вызывается функция scroll(). Первым делом в функции scroll() мы устанавливается таймер. Этим гарантируется, что функция scroll() будет повторно вызвана через 100 миллисекунд. При этом текст будет перемещен еще на один шаг и запущен другой таймер. Так будет продолжаться без конца.

Данный вызов функции не является рекурсивным! Рекурсия получается, если вызывать функцию scroll() непосредственно внутри самой же функции scroll(). А этого здесь как раз и не делается. Прежняя функция, установившая таймер, заканчивается еще до того, как

начинается выполнение новой функции.

Скроллинг используется в Интернет довольно широко. И есть риск, что быстро он станет непопулярным. В большинстве страниц, где он применяется, особенно раздражает то, что из-за непрерывного скроллинга становится невозможным прочесть в строке состояния адрес URL. Эту проблему можно было бы решить, позаботившись о приостановке скроллинга, если происходит событие MouseOver - и, соответственно, продолжении, когда фиксируется onMouseOut.

Функции даты/времени

Работа с датой и временем, установленными на данном компьютере, в JavaScript осуществляется с помощью объекта Date. Перед его использованием его необходимо создать:

```
DateObject = new Date()
```

Данной строкой создается объект Date, содержащий текущее системное время компьютера, на котором он создается.

При его создании в него можно поместить не системное время, а произвольное. Для этого применяется конструкция:

```
var DateObject = new Date([значение]),
```

где параметр «значение» может принимать следующие значения:

- -миллисекунды - количество миллисекунд прошедшее от 01/01/70 00:00:00;
- год, месяц, день;
- год, месяц, день, часы, минуты, секунды;
- год, месяц, день, часы : минуты : секунды.

Методы объекта Date, для работы с датой и временем представлены в таблице 43[10]:

Таблица 2. Методы объекта Date

Метод	Описание
getDate()	число месяца
getDay()	номер дня недели
getHours()	Час
getMinutes()	Минута
getMonth()	номер месяца
getSeconds()	Секунда
getTime()	количество миллисекунд между 1 января 1970 года и текущим временем
getTimezoneOffset()	разница в минутах между местным и гринвичским временем
getYear()	год
setDate()	устанавливает день месяца
setHours()	устанавливает час
setMinutes()	устанавливает минуту
setMonth()	устанавливает месяц
setSeconds()	устанавливает секунду
setYear()	устанавливает год
toGMTString()	преобразует местное время во время по Гринвичу
toLocaleString()	преобразует время по Гринвичу в местное время
UTC()	возвращает количество миллисекунд между 01/01/70 00:00:00 и заданным временем в формате объекта Date

getDate() возвращает текущий день месяца:

```
var D, day;
D = new Date();
day= D.getDate();
alert("Сегодня " + day + " число");
```

getDay() возвращает день недели. Возвращаемая величина представляет собой целое число от 0 до 6. Дни недели нумеруются, начиная с нуля, причем первым днем в соответствии с американской системой начинать неделю с выходного идет воскресенье.

```
function Day_of_week()
{
 var day, DateObj;
 //помещаем в массив названия дней недели:
 var Days = new Array("Воскресенье", "Понедельник",
"Вторник", "Среда", "Четверг", "Пятница",
"Суббота");
 //создаем объект Date
 DateObj = new Date();
 //получаем номер дня
 day = DateObj.getDay();
 //вызываем из массива название дня по его номеру и
выводим результат:
 alert("Сегодня у нас " + Days[day]);
}
```

getMonth() показывает номер текущего месяца как целое число от 0 до 11. Как и в днях недели, нумерация месяцев начинается с нуля (Январь = 0; Февраль = 1; … Декабрь = 11).

Для демонстрации метода getMonth() составим сценарий, выводящий название месяца:

```
function Month_of_Year()
{
 var month, DateObj;
 //помещаем в массив названия месяцев:
 var Months = new Array("Январь", "Февраль",
"Март", "Апрель", "Май", "Июнь", "Июль","Август",
"Сентябрь", "Октябрь", "Ноябрь", "Декабрь");
 //создаем объект Date
 DateObj = new Date();
 //получаем номер месяца:
 month = DateObj.getMonth();
 //вызываем из массива название дня по его номеру и
выводим результат:
 alert("Сейчас идет " + Months[month] + " месяц");
 }
```

getYear() возвращает количество лет прошедших от 1900–го года, в виде целого двухразрядного числа от 0 до 99. До 2000-го года, чтобы

вывести полную дату, нужно было к результату, возвращаемому методом getYear() прибавлять 1900:

```
...
var Year = 1900 + DataObj.getYear();
...
```

Особенность использования метода getYear() заключается в том, что современные версии браузеров, при использовании метода в 2000-х годах выведут текущий год в четырехразрядном числе: 2000, 2001 и т.д. И сценарии, которые годились до наступления 2000 года, будут работать неверно, выдавая результат на 1900 лет больше.

Следует также сказать, что некоторые браузеры по-прежнему при использовании метода getYear() считают время от 1900 года.

Для избегания подобных недоразумений можно составить подобный сценарий, контролирующий данный казус:

```
<script>
//создаем объект Date:
DateObj = new Date();
//получаем год
y = DateObj.getYear();
//если год двухразрядный:
 if (y < 2000)
{
 year = 1900 + y;
 }
 else
 {
 //оставляем год без изменений:
 year = y;
 }
alert("Сейчас идет год " + year);
</script>
```

getHours() указывает текущий час и возвращает целое число от 0 до 23,

getMinutes() выводит значение минут числом от 0 до 59, а getSeconds() отвечает за вывод текущей секунды, числом от 0 до 59.

Если возвращаемые минуты и секунды находятся в промежутке от 0 до 10, то JavaScript возвращает их без нулей: 2, 3 и т.д.

Для удобства пользователей, желательно добавить к ним нуль, например, таким образом:

```
 …
 tSec = DateObj.getSeconds();
 if (tSec < 10)
 {
 tSec = "0" + tSec;
}
 ...
```

Для вывода текущего времени составим следующий сценарий:

```
 function T()
 {
 //создаем объект Date:
 DateObj = new Date();
 //получаем текущий час:
 tHour = DateObj.getHours();
 //получаем текущую минуту
 tMin = DateObj.getMinutes();
 if (tMin < 10)
 {
 tMin = "0" + tMin;
 }
 //секунды:
 tSec = DateObj.getSeconds();
 if (tSec < 10)
 {
 tSec = "0" + tSec;
 }
 alert("Текущее время: " + tHour + ":" + tMin + ":"
+ tSec);
 }
```

Если требуется не только выводить текущее время по требованию, а поместить на экран настоящие «тикающие» часы, нужно добавить в функцию метод setTimeout().

Создадим на странице поле ввода, в которые поместим наши часы:

```
<center>
<b>Текущее время:</b>
<form>
<input type="text" size=8>
</form>
</center>
```

Чтобы часы отображались в поле ввода, заменим строку

```
alert("Текущее время: " + tHour + ":" + tMin + ":"
+ tSec);
```

на

```
//присваиваем переменной clock текущее время:
clock = tHour + ":" + tMin + ":" + tSec;
//выводим время в текстовое поле
document.forms[0].elements[0].value = clock;
//Добавляем таймер
setTimeout("T()", 1000);
```

Теперь в тег <BODY> добавим обработчик onLoad, вызывающий нашу функцию:

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

Если мы захотим, то можем поместить часы в строку состояния браузера. Для этого заменим строку

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

на

```
window.status = "Текущее время: " + clock;
```

Теперь у нас в строке состояния будут системные часы.

Метод getTime(), показывает, сколько миллисекунд прошло от 1 января 1970 года, а getTimezoneOffset() - разницу между местным и Гринвичским временем в минутах.

```
 var DateObj = new Date();
 var x = DateObj.getTimezoneOffset()
 alert("Местное время отличается от Гринвичского на
"+ x + " минут");
```

Метод toLocaleString() переводит время по Гринвичу в местное время и дату. Создавать часы на странице данным методом удобнее, чем, используя методы getHours(), getMinutes(), getSeconds(), но только метод toLocaleString() выводит еще и текущую дату[10].

Составим сценарий, выводящий дату и время в строку состояния:

```
<SCRIPT>
function T()
{
var DateObj = new Date();
var Now_Time = DateObj.toLocaleString();
window.status = "Сейчас " + Now_Time;
setTimeout("T()", 1000);
}
</SCRIPT>
```

После этого добавим в тег <BODY> вызов функции: <BODY onLoad="T()">

Аналогичные методы set позволяют задать ту или иную характеристику времени. Общий синтаксис использования данных методов эквивалентен методам get. Единственное отличие – присутствие числового значения аргумента функции.

Например,

```
DateObject.setDate(новое значение)
```

Проиллюстрируем использование методов сценарием, выводящим название предыдущего месяца. Составим этот сценарий только для примера, поскольку получить номер предыдущего месяца можно, отняв от значения возвращенного методом getMonth единицу.

```
<script>
```

```
 function PreviousMonth()
 {
 var month, DateObj;
 var Months = new Array("Январь", "Февраль",
"Март", "Апрель", "Май", "Июнь", "Июль","Август",
"Сентябрь", "Октябрь", "Ноябрь", "Декабрь");
 //создаем объект Date, содержащий текущее
системное время:
 DateObj = new Date();
 //получаем текущий месяц
 month = DateObj.getMonth();
 //если месяц не январь:
 if (month >0)
 {
 //устанавливаем предыдущий месяц:
 DateObj.setMonth(month-1);
 //получаем его:
 month = DateObj.getMonth();
 alert("Прошлый месяц был " + Months[month]);
 }
 else //если текущий месяц январь
 if (month == 0)
 {
 //устанавливаем месяц Декабрь:
 DateObj.setMonth(11);
 //получаем его:
 month = DateObj.getMonth();
 alert("Прошлый месяц был " + Months[month]);
 }
 }
 </script>
```

☑ Вопросы для самоконтроля

1. Что нам позволяет выполнять команда windows.status?
2. В каких случаях используется строка состояния?
3. Для чего при печати используется символ обратный слеш «\»?
4. В чем особенности работы с полосой прокрутки в JavaScript?
5. В чем особенности работы с таймером в JavaScript?

6. В чем особенности работы с функциями даты и времени в JavaScript?

§4.6. Диалоговые окна, основные конструкции и строки

Диалоговые окна JavaScript

JavaScript может выдавать сообщения и взаимодействовать с пользователями с помощью трех различных диалоговых окон, вызываемых с помощью методов alert, confirm и prompt.

```
alert (окно предупреждения)
```

С помощью уже знакомого нам окна предупреждения, на экран выводится информация для пользователя. Например,

```
alert(“Добрый день”);
```

Пользователю остается только прочитать сообщение и нажать кнопку OK, чтобы закрыть это окно.

```
confirm (окно подтверждения)
```

В отличие от предыдущего окна, где пользователь не может ничего сделать, кроме как нажать кнопку OK, для закрытия окна, в диалоговом окне confirm пользователь может выбрать один из двух вариантов: OK, Отмена.

При закрытии данного окна оно возвращает в сценарий значение true или false. Если выбрана кнопка OK, то возвращается true, если Отмена – false.

Например:

```
<SCRIPT>
 if (confirm("Выберите кнопку")) //Если возвращено
true
```

```
 //то выполняется данный оператор:
 alert ("Вы выбрали OK");
 //иначе, если возвращено false, т.е. нажата кнопка
[Отмена]:
 else
 {
 // выполняется данный оператор:
 alert("Вы выбрали Отмена");
 }
 </SCRIPT>
```

В зависимости от выбора пользователя, сценарий возвратит, какую кнопку он нажал.

Как уже было изучено ранее, значения true и false возвращают числовые значения 1 и 0 соответственно. Можно использовать это свойство для наглядности примера. Для этого введем в сценарий переменную x:

```
{
 //присваиваем x возвращаемое значение:
 var x = (confirm("Выберите кнопку"));
 if (x == 1)
 alert ("Вы нажали OK");
 else //если возвращаемое значение не 1 а
соответственно 0
 {
 alert("Вы нажали Отмена");
 }
```

В вышеприведенном примере можно 1 заменить true.

```
prompt (окно ввода)
```

С помощью окна ввода, вызываемого методом prompt, пользователь может ввести данные, которые будут использоваться далее в сценарии.

```
prompt (“Введите ваше имя”, “Вводите сюда”);
```

Как видно из примера первая фраза “Введите ваше имя” – предлагает пользователю ввести информацию, а вторая отображается в строке ввода. Если вторую строку опустить, то в строке ввода появится сообщение браузера по умолчанию.

Если требуется, чтобы в строке состояния было пусто, то вставьте пустые кавычки.

```
prompt (“Введите ваше имя”, “”);
```

При закрытии окно prompt возвращает значение, введенное в строке ввода.

Приведем пример использования окна prompt. Создадим документ с кнопкой и поместим в обработчик щелчков кнопки onClick="Name1()". Добавим в него следующие строки:

```
function Name1()
 {
 /*присваиваем переменной name значение
 введенное в строке ввода*/
 var name = prompt("Введите ваше имя:", "");
 alert ("Привет " + name + " !");//Выводим
результат
 }
```

С помощью условного оператора можно добавить в сценарий контроль ввода имени пользователем:

```
function Name1()
 {
 var name = prompt("Введите ваше имя:", "");
 /*присваиваем переменной
 name значение введенное в строке ввода*/
 if (name == '')
 //если ничего не введено
 {
 alert("Введите пожалуйста свое имя")
 //Снова просим ввести имя
 Name1(); //и вызываем сценарий снова
 }
 else if (name == null)
 //если пользователь нажал кнопку - Отмена
 {
 alert("Введите пожалуйста свое имя")
 //Снова просим ввести имя
 Name1(); //вызываем сценарий
 }
 else
 //если все верно
 {
 alert("Привет "+name+" !");
 }
 }
```

Как видно, при обработке ошибок ввода пользователя дважды вызываются одни и те же операторы:

```
alert("Введите пожалуйста свое имя")
//Снова просим ввести имя
Name1(); //вызываем сценарий
```

Вместо этого можно создать отдельную функцию обработки ошибок ввода:

```
function Name_error()
{
alert("Введите пожалуйста свое имя")
//Снова просим ввести имя
Name1();
}
```

После этого данные операторы в тексте сценария заменяются на вызов этой функции:

```
<SCRIPT>
function Name1()
{
var name = prompt("Введите ваше имя:", "");
/*присваиваем переменной name значение введенное в
строке ввода*/
if (name == '')
//если ничего не введено
{
Name_error();
//вызываем функцию обработки ошибок
}
else if (name == null)
//если пользователь нажал кнопку - Отмена
{
Name_error();
//вызываем функцию обработки ошибок
}
else //если все верно
{
alert("Привет "+name+" !");
}
}
```

```
 //располагаем функцию далее в этом же теге
<script>
 function Name_error()
 {
 alert("Введите пожалуйста свое имя")//Снова просим
ввести имя
 Name1(); //вызываем сценарий
 }
 </SCRIPT>
```

В завершение следует отметить, что кнопки OK и Отмена работают также как и у окна confirm.

Операции

Для работы с различными данными, в первую очередь с числовыми, в JavaScript используются знаки различных операций.

Математические операции. Математические операции используются для действий над числами и предоставляют возможность производить над ними простейшие арифметические действия (таблица 44):

Таблица 3. Математические операции

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление

%	Остаток от деления, деление по модулю
	Например:
	`var x = 10 % 3;`
	`//получим 1 т.е. x = 10 - (10/3)`

++	Увеличение на 1
--	Уменьшение на 1

Операторы присваивания. Для присвоения переменным данных используется оператор присваивания (=).

```
var result = Summa;
```

При присваивании значений можно комбинировать оператор присваивания и математические операторы, так как это показано в таблице 45:

Вместо

```
x = x + result,
```

можно использовать

```
x+=result
```

Таблица 4. Операторы присваивания

Оператор	Пример	Объяснение
=	X = X	X равен Y
+=	X+=Y	X = X + Y
-=	X-=Y	X = X – Y
*=	X*=Y	X = X * Y
/=	X/=Y	X = X / Y

Операторы. Управление последовательностью действий в ходе выполнения сценария осуществляется с помощью операторов. JavaScript содержит стандартный набор операторов, унаследованный от языков C++ и Java, а именно:

- условный оператор if…else;
- оператор выбора switch;
- операторы цикла for, while и др.

Условный оператор. Условный оператор if…else позволяет проверить определенное условие и, в зависимости от его истинности, выполнить ту или иную последовательность операторов. Он имеет две формы:

```
if (условие) оператор1
if (условие) оператор1 else оператор2
```

Здесь условие — это любое выражение, значение которого может быть преобразовано к логическому типу, оператор1 и оператор2 — любые группы операторов JavaScript; если эти группы содержат более одного оператора, то они должны быть заключены в фигурные скобки {}[10].

Первая форма оператора означает, что если значение условия истинно, то выполняются операторы1; если оно ложно, то управление передается оператору, следующему за if.

Вторая форма оператора означает, что если значение условия истинно, то выполняются операторы1; если оно ложно, то выполняются операторы2.

Пример использования условного оператора в теле функции, возвращающей наибольшее из трех чисел:

```
function maxValue(x, y, z) {
  if (x >= y) {
    if (x >= z)
      return x;
    else
      return z;
  }
  else {
    if (y >= z)
      return y;
    else
      return z;
  }
}
```

Оператор выбора. Оператор выбора switch выполняет ту или иную последовательность операторов в зависимости от значения определенного выражения. Он имеет вид:

```
switch (выражение) {
```

```
    case значение:
      операторы
      break;
    case значение:
      операторы
      break;
    ...
    default:
      операторы
}
```

Здесь выражение — это любое выражение, значение — это возможное значение выражения, а операторы — любые группы операторов JavaScript.

Оператор выбора сначала вычисляет значение выражения, а затем проверяет, нет ли этого значения в одной из меток case значение. Если такая метка есть, то выполняются операторы, следующие за ней; если нет, то выполняются операторы, следующие за меткой default (если она отсутствует, то управление передается оператору, следующему за switch).

Необязательный оператор break указывает, что после выполнения операторов управление передается оператору, следующему за switch. Если break отсутствует, то после выполнения операторов начинают выполняться операторы, стоящие после следующей метки case (управление как бы "проваливается" в следующую метку).

В следующем примере значение переменной `length` преобразуется в метры в зависимости от начальной единицы измерений, заданной в переменной `str`. Обратите внимание, что после `case "м"` нет оператора break; в данном случае это означает, что эта метка и метка default обрабатываются одинаково, а именно значение переменной `length` не изменяется.

```
var str = "см";
var length = 25;
switch (str) {
  case "км":
    length *= 1000;
    break;
  case "см":
```

```
    length /= 100;
    break;
  case "м":
  default:
    break;
}
```

Операторы цикла. Цикл — это последовательность операторов, выполнение которой повторяется до тех пор, пока определенное условие не станет ложным.

Оператор цикла for имеет вид:

```
for (инициализация; условие; изменение) оператор
```

Здесь инициализация и изменение — это любое выражения, условие — любое выражение, значение которого может быть преобразовано к логическому типу, оператор — любая группа операторов JavaScript; если эта группа содержит более одного оператора, то она должны быть заключена в фигурные скобки `{}`. Инициализация может содержать декларацию переменной.

Оператор for выполняется следующим образом:

- Выполняется выражение инициализация (обычно это выражение инициализирует счетчик или счетчики цикла).
- Вычисляется значение выражения условие. Если оно ложно, то управление передается оператору, следующему за данным оператором.
- Выполняется оператор.
- Выполняется выражение изменения (обычно это выражение увеличивает или уменьшает счетчик или счетчики цикла) и управление передается этапу 2.

Данный оператор обычно используется в тех случаях, когда количество повторений цикла известно заранее. Например, следующая функция обнуляет все элементы массива, переданного ей в качестве аргумента:

```
function initArray(a) {
```

```
  for (var i = 0; i < a.length; i++)
    a[i] = 0;
}
```

Оператор цикла while имеет вид:

```
while (условие) оператор
```

Здесь условие— любое выражение, значение которого может быть преобразовано к логическому типу, оператор - любая группа операторов JavaScript; если эти группа содержит более одного оператора, то она должны быть заключена в фигурные скобки `{}`.

Оператор while выполняется следующим образом:

- Вычисляется значение выражения условие. Если оно ложно, то управление передается оператору, следующему за данным оператором.
- Выполняется оператор и управление передается этапу 1.

При использовании данного оператора необходимо убедиться, что рано или поздно условие станет ложным, т.к. иначе сценарий войдет в бесконечный цикл, например:

```
while (true)
 document.write("Привет всем!");
```

☑ Вопросы для самоконтроля

1. Для чего используются диалоговые окна?
2. В чем заключается разлиие между диалоговыми окнами alert, confirm и promt?
3. Какие операторы присваивания существуют в JavaScript и в чем их различия?
4. Какие математические операции в JavaScript вы знаете?
5. Какие условные операторы в JavaScript вы знаете?
6. В чем особенности работы с функциями даты и времени в JavaScript?

§4.7. Работа с графическими объектами в JavaScript

В первой главе уже были рассмотрены возможности работы с изображениями на Web-страницах и, рассматривая новые элементы HTML5, упоминался элемент <canvas>. Пришло время подробнее разобраться в особенностях работы с этим элементом.

С английского canvas переводится как холст, по сути, на web-страницах этот элемент и является холстом, предназначенным для создания растровых двухмерных изображений с использованием языка JavaScript. Элемент canvas используется, как правило, для отрисовки графиков для статей и игровых полей в браузерных играх, а также для встраивания видео на web-страницы и создание полноценного плеера.

Элемент canvas был разработан компанией Apple на системе WebKit для Mac OS X с целью последующего его использования в приложениях Dashboard и Safari. Ситуацию с отсутствием canvas в IE исправила компания Google, выпустившая собственное расширение, написанное на JavaScript, под названием ExplorerCanvas. В настоящее время большинство современных браузеров предоставляют возможности 2D-контекста (2D Canvas) — Opera, Firefox, Konqueror и Safari. Кроме того, существуют экспериментальные сборки браузера Opera, которые включают поддержку 3D-контекста (3D Canvas), а также дополнение к Firefox, которое реализует поддержку 3D Canvas[14].

Чтобы создать Canvas-контекст, достаточно просто добавить элемент <canvas> в HTML-документ:

```
<canvas id="myCanvas" width="200" height="100">
Альтернативное содержимое, которое будет показано,
если браузер не поддерживает Canvas.
</canvas>
```

Нужно добавить идентификатор id к элементу canvas, чтобы потом обратиться к нему в JavaScript, также необходимо задать атрибуты width и height для определения ширины и высоты элемента canvas.

JavaScript используется для рисования внутри элемента canvas. Но сначала с помощью функции getElementById необходимо найти созданный тег canvas, а затем инициализировать нужный контекст.

После этого можно начинать рисование на канве с использованием доступных API-команд выбранного контекста. Следующий скрипт рисует простой прямоугольник:

```
<script>
var c=document.getElementById("myCanvas");
// получение ссылки на на элемент canvas по
идентификатору
var ctx=c.getContext("2d"); // инициализация
контекста
// рисуем прямоугольник, задав координаты (x,y), а
также его ширину
ctx.fillRect(0,0,150,75);
</script>
```

Заливки и границы фигур

Настройка цветов для заливки и линий объектов осуществляется с помощью свойств fillStyle и strokeStyle. Значения цветов, используемые в этих методах, такие же, как и в CSS: шестнадцатеричные коды (#F5E6AB), rgb(), rgba. В следующем примере в предыдущий код добавлена строчка, с помощью которой прямоугольник заполняется красным цветом:

```
<script>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

Использование метода fillRect позволяет нарисовать прямоугольник с заливкой, а метода strokeRect - нарисовать прямоугольник только с границами, без заливки. Для очистки некоторой части канвы используется метод clearRect. Три этих метода используют одинаковый набор аргументов: x, y, width, height. Как и в предыдущем примере, первые два аргумента задают координаты (x,y), а следующие два — ширину и высоту прямоугольника. Изменение толщины линий

происходит с помощью свойства lineWidth. Следующий пример демонстрирует использование функций fillRect, strokeRect, clearRect:

```
<html>
<body>
<canvas id="myCanvas" width="200" height="150"
style="border:1px solid #c3c3c3;">
Альтернативное содержимое, которое будет показано,
если браузер не поддерживает Canvas.
</canvas>
<script>
var c=document.getElementById("myCanvas");
// получение ссылки на на элемент canvas по
идентификатору
var ctx=c.getContext("2d"); // инициализация
контекста
ctx.fillStyle = '#00f'; //задаем цвет заливки
ctx.strokeStyle = '#f00'; // задаем цвет линии
ctx.lineWidth = 6; // задаем толщину линии
// Рисуем прямоугольники
ctx.fillRect (0, 0, 150, 50);
ctx.strokeRect(0, 60, 150, 50);
ctx.clearRect (30, 25, 90, 60);
ctx.strokeRect(30, 25, 90, 60);
</script>
</body>
</html>
```

Этот пример приведет к следующему результату представленному на рисунке 46:

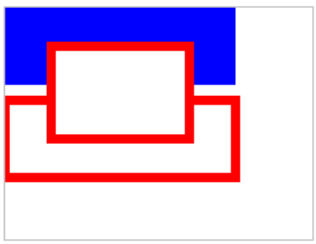

Рисунок 7 - Результат использования функций fillRect, strokeRect, clearRect

Окружность и круг

Для рисования окружности используется метод arc(x,y,r,start,stop). Для рисования круга используются методы stroke() или fill(), которые позволяют залить окружность. Следующий скрипт демонстрирует, как нарисовать круг:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

Контуры

Используя контуры Canvas можно нарисовать фигуры любой формы. Сначала нужно нарисовать «каркас», а потом можно использовать стили линий или заливки, если это необходимо. Чтобы начать рисование контура, используется метод beginPath(), потом рисуется контур, который можно составить из линий, кривых и других примитивов. Как только рисование фигуры окончено, можно вызвать методы назначения стиля линий и заливки, и только потом вызвать функцию closePath() для завершения рисования фигуры. Следующий скрипт демонстрирует рисование треугольника:

```
<script>
var c=document.getElementById("myCanvas");
// получение ссылки на на элемент canvas по
идентификатору
var ctx=c.getContext("2d"); // инициализация
контекста
ctx.fillStyle = '#00f';
ctx.strokeStyle = '#f00';
ctx.lineWidth = 4;

ctx.beginPath();
// Начинаем рисовать треугольник с верхней левой
точки.
ctx.moveTo(10, 10); // перемещаемся к координатам
(x,y)
ctx.lineTo(100, 10);
```

```
ctx.lineTo(10, 100);
ctx.lineTo(10, 10);
// Заполняем фигуру заливкой и применяем линии
// Фигура не будет отображена, пока не будет вызван
хотя бы один из этих методов.
ctx.fill();
ctx.stroke();
ctx.closePath();
</script>
```

Этот пример будет отображен в браузере следующим образом, как показано на рисунке 47:

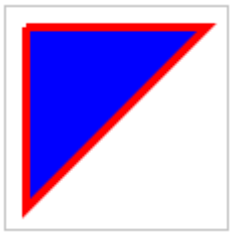

Рисунок 8. Пример рисования треугольника

Вставка изображений в Canvas

Метод drawImage(image, x, y) позволяет вставлять другие изображения на канву. Метод drawImage включает один аргумент для указания изображения, которое необходимо вывести на канве, и два аргумента для задания координат[14].

Следующий скрипт демонстрирует использование этого метода:

```
<script>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var img=document.getElementById("scream");
// элемент img, координаты для вывода на канву
(x,y).
ctx.drawImage(img, 10, 10);
</script>
```

Добавление текста на холст

Следующие свойства текста доступны для объекта контекста:

- font: используется для определения шрифта текста
- textAlign: используется для горизонтального выравнивания текста. Допустимые значения: start, end, left, right, center. Значение по умолчанию: start.
- textBaseline: используется для вертикального выравнивания текста. Допустимые значения: top, hanging, middle, alphabetic, ideographic, bottom. Значение по умолчанию: alphabetic.

Существуют два метода для вывода текста: fillText и strokeText. Первый метод используется для отрисовки текста, заполняя его заливкой стиля fillStyle, другой для рисования обводки текста, используя стиль strokeStyle. Оба метода принимают три аргумента: текст и координаты (x,y), в которых будет выводиться текст. Также существует четвертый необязательный аргумент — максимальная ширина, расположения текста в заданной ширине.

Следующий скрипт демонстрирует вывод на Canvas слов "hello world":

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

☑ Вопросы для самоконтроля

1. Что позволяет делать элемент canvas?
2. С помощью каких методов осуществляется отрисовка различных фигур, их заливка?
3. Возможно ли загрузить в canvas обычное изображение?
4. Как вывести текст в элемент canvas?

☑ Лабораторная работа №3

☑ *Задание 1.*

Используя полученные знания, создайте страницу, содержащую:

1. кнопку, при нажатии на которую выполняется скрипт, записывающий на ней следующие 3 строчки: первая будет полужирным шрифтом, вторая курсивом, третья – размещена по центру.
2. картинку, которая изменяется при наведении на нее, и увеличивает свой размер при нажатии на нее,
3. идущие часы в правом верхнем правом углу, а в нижнем правом углу текущая дата в формате «1 января 2015 года, понедельник».
4. кнопку, при нажатии на которую выполняется скрипт, заменяющий во введенном в текстовое поле тексте все символы % на символ @
5. Добавьте холст и нарисуйте на нем:

- Нарисуйте два прямоугольника, один прямоугольник заполните зеленым цветом, а другой желтым.
- Нарисуйте ромб с красным цветом линии и желтым заполнением.
- Выведите приветственный текст желтым цветом и фиолетовой обводкой.

6. Добавьте мини-калькулятор с 4 основными арифметическими операциями и 3 математическими (на Ваш выбор)