О.Н. Рева

просто о сложном

# JavaScript

в кармане

УДК 004.43
ББК 32.973.26-018.1
Р 32

**Рева О. Н.**

Р 32 JavaScript в кармане / О. Н. Рева. — М. : Эксмо, 2008. — 256 с.: ил. — (Компьютер в кармане).

Если вы почувствовали, что вашим Web-страницам недостает динамичности, гибкости и собственного характера, обратитесь к сценариям. С помощью сценариев вы сможете сделать документ таким же умным, как вы сами, вдохнуть в него жизнь и собственный характер. Сценарии — это небольшие программы, вписанные в HTML-код Web-страницы, а JavaScript, пожалуй, наиболее популярный язык написания сценариев для Web-страниц. Этот язык создавался для широкого круга разработчиков Web-страниц, не являющихся профессиональными программистами. Желательно, чтобы читатель имел представление о коде HTML Web-страниц. Впрочем, сведений о HTML, представленных в этой книге, будет достаточно для создания новичками небольших персональных Web-страниц.

**УДК 004.43**
**ББК 32.973.26-018.1**



**Рева Олег Николаевич**

**JAVASCRIPT В КАРМАНЕ**

Директор редакции *И. Е. Федосова*
Выпускающий редактор *В. А. Обручев*
Художественный редактор *П. Е. Петров*

ООО «Издательство «Эксмо»
127299, Москва, ул. Клары Цеткин, д. 18/5. Тел. 411-68-86, 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Подписано в печать 10.01.2008.
Формат 70×100 $^{1}/_{32}$. Печать офсетная. Бумага тип. Усл. печ. л. 10,4.
Тираж 3000 экз. Заказ № 7260.

Отпечатано с предоставленных диапозитивов
в ОАО «Тульская типография». 300600, г. Тула, пр. Ленина, 109.

**ISBN 978-5-699-26260-1**

# Содержание

*Эта книга посвящается моему отцу.*
*Надеюсь, она его порадует, несмотря на то,*
*что он совершенно равнодушен*
*к программированию на JavaScript*

## Благодарности

Я хочу выразить свою искреннюю благодарность всем, кто внес свою лепту в создание этой книги, — редакторам, корректорам, художникам, верстальщикам. Выполняя свою работу, они прокладывают надежный мост между автором и читателями.

# Введение

Человек, увлеченный работой на компьютере, рано или поздно придет к выводу, что все коммерческие приложения, которыми он пользуется, недостаточно хороши для него. Есть множество средств создания персональных Web-страниц — от простейших редакторов, бесплатно предоставляемых администраторами Web-серверов, до таких высокопрофессиональных, как Microsoft FrontPage и Macromedia Dreamweaver. Но созданным с их помощью документам HTML недостает интерактивности — способности изменяться в ответ на запросы пользователя и быть посредником в диалоге между вами и посетителями Web-страницы. С помощью сценариев, встроенных в HTML-код Web-страницы, вы сможете сделать ваш документ таким же «умным», как вы сами, вдохнуть в него жизнь и собственный характер.

Сценариями называются программы, написанные на языке, понятном обозревателю. В этой книге мы изучим JavaScript — пожалуй, наиболее популярный язык написания сценариев для Web-страниц. JavaScript нельзя отнести к «серьезным» языкам программирования. Разработчики стандартов этого языка из компании Netscape постарались максимально упростить его, потому что, во-первых, многие разработчики Web-страниц не являются профессиональными программистами, а во-вторых — использование сценариев не предполагает решения грандиозных проблем. Сценарии — это специи, при разумном и умеренном использовании которых повар превращает обычную похлебку в кулинарный шедевр. Кроме того, если до сих пор вам не приходилось заниматься программированием, написание сценариев станет для вас первым шагом на этом увлекательном пути.

## Структура книги

Данная книга состоит из семи глав. В главе 1, «Знакомство с JavaScript», вы получите общее представление о принципах написания сценариев и круге проблем, которые можно решить с помощью этих средств.

Сценарии не существуют сами по себе. Они всегда являются частью Web-страницы и должны быть органически вписаны в HTML-код документа. В главе 2, «JavaScript и HTML», вам будет представлен об-

зор дескрипторов HTML, а также описаны способы добавления сценариев и активизации их в коде Web-страницы.

Более подробно о том, как заставить сценарий выполняться тогда, когда в этом возникнет необходимость, рассказывается в главе 3, «Использование сценариев в динамических Web-страницах».

Что отличает Web-страницы от текстовых иллюстрированных документов для печати? Любая программа — это работа с данными, т.е. с информацией, которая поступает от пользователя, компьютера или из базы данных. Глава 4, «Работа с данными в сценариях JavaScript», посвящена типам данных, а также особенностям хранения и обработки данных разных типов в сценариях JavaScript. Полезным будет знакомство с коллекциями методов выполнения сложных математических вычислений и обработки текста, собранных в объектах `Math` и `String`.

«Умная» программа отличается тем, что ее выполнение зависит от условий, заданных пользователем. В главе 5, «Управление ходом выполнения сценария», речь пойдет о том, как разветвлять код сценария, организовывать циклы и устанавливать таймер вызова функций.

Из главы 6, «Элементы мультимедиа в динамических Web-страницах», вы узнаете об управлении графикой, звуковыми и видеоклипами с помощью сценариев.

В главе 7, «Обмен данными и сохранение информации на диске», рассматриваются средства, предоставляемые языком JavaScript для организации диалога между разработчиком и пользователем Web-страницы, между окнами обозревателя и между документом HTML и простейшими внешними базами данных. В этой главе также описан простейший редактор базы данных, удобный для использования на собственном компьютере или в сети intranet.

## Соглашения, принятые в книге

Прежде чем переворачивать страницу и переходить к чтению глав этой книги, обратите внимание на следующие моменты. Названия элементов пользовательского интерфейса программы (кнопок, команд, меню, диалоговых окон и т.п.) в тексте книги выделены **специальным шрифтом**. Для обозначения команд, расположенных в меню программы, будет использоваться стрелка (⇨). Например, запись «выберите команду **Вставка⇨Рисунок⇨Из файла**» означает, что вам необходимо

открыть меню **Вставка** (щелкнув на его названии в строке меню), выбрать в нем подменю **Рисунок** и затем в этом подменю щелкнуть на команде **Из файла**. Кроме того, вам будет предлагаться ввести с клавиатуры HTML-код страницы, код сценария или URL-адрес. Слова, которые нужно набрать, также выделены `соответствующим образом`. В некоторых случаях в листингах и в примерах программного кода показан *`временный текст`*, который нужно заменить в конечном коде. Новые термины, при первом упоминании или определении их в книге, выделяются *курсивом*.

В данной книге используются пиктограммы, акцентирующие ваше внимание на отдельных абзацах.


Такой пиктограммой отмечены полезные сведения, которые помогут вам более эффективно использовать инструменты и возможности Web-страниц.


Очень внимательно отнеситесь к сведениям, помеченным такой пиктограммой. Они помогут вам избежать ошибок, которые могут стать причиной как мелких, так и более серьезных проблем.


Разумеется, в данной книге описывается довольно большое количество примеров выполнения практических задач. Для удобства абзацы, в которых демонстрируются примеры, применимые на практике, будут помечены этой пиктограммой.

Все примеры в этой книге были опробованы с помощью обозревателя Internet Explorer 4.0 на компьютере PC. Некоторые из примеров, представленных в листингах этой книги, не будут выполняться при использовании более ранних версий Internet Explorer, в других обозревателях и на компьютерах с иной платформой.

# Глава 1

# Знакомство с JavaScript

Эта книга предназначена для начинающих разработчиков Web-страниц, имеющих наиболее общие представления о технологиях Web и HTML-коде. Язык JavaScript используется для написания сценариев, с помощью которых статические Web-страницы превращаются в интерактивные средства общения разработчика с посетителями Web-страниц. Эффективность документов HTML существенно возрастет, если пользователи смогут динамически изменять вид и содержимое Web-страниц в соответствии с собственными предпочтениями и запросами, вводить пользовательскую информацию в поля формы, направлять сообщения и комментарии разработчику Web-страницы. Если вам никогда не приходилось заниматься программированием, пусть вас это не пугает. JavaScript — один из наиболее простых языков программирования, не требующий глубоких познаний в области компьютерных технологий. Многие сценарии на JavaScript состоят всего из одной или нескольких команд, которые изменяют свойства элементов документа HTML. Вам не нужно знать все тонкости взаимодействия сценария JavaScript с HTML-кодом и приложением обозревателя, за исключением общих принципов, изложенных в этой книге.

## Что такое JavaScript

Язык JavaScript для написания сценариев Web-страниц был разработан компанией Netscape Corporation. Сценарии JavaScript не могут существовать как самостоятельные приложения. Они привязаны к HTML-коду Web-страницы и могут выполняться только при участии приложения обозревателя.

По синтаксису JavaScript немного напоминает язык Java, но это только внешнее сходство.

С помощью сценариев JavaScript можно выполнять следующие задачи:

- создавать анимационные эффекты с текстом и графикой;
- воспроизводить звуковое сопровождение страницы в соответствии с контекстом;
- динамически изменять вид и содержимое страницы;
- проверять и передавать данные, введенные пользователем в поля формы.


Несмотря на то что язык JavaScript был разработан в компании Netscape, сценарии JavaScript лучше работают в Microsoft Internet Explorer, чем в Netscape Navigator. Другой язык написания сценариев — VBScript — поддерживается исключительно обозревателем Internet Explorer, что делает его менее популярным.


Сценарии JavaScript выполняются *на стороне клиента*, т.е. код сценария выгружается по сети на компьютер клиента и выполняется обозревателем. Альтернативным решением может быть использование программ CGI (Common Gateway Interface — интерфейс общего доступа), которые выполняются на Web-сервере. Сценарии JavaScript работают быстрее, но недостаток этого подхода состоит в том, что выполнение сценариев может быть отключено в параметрах обозревателя.

## JavaScript и другие языки программирования

Чтобы лучше оценить достоинства и недостатки языка JavaScript, сравним его с другими популярными языками программирования.

### Программные коды

Компьютеры способны понимать только язык, в основе которого лежит бинарный машинный код. Этот код напоминает азбуку Морзе, состоящую из точек и тире, за тем исключением, что все «слова» машинного кода имеют одинаковую длину, соответствующую разрядности машины.

Машинный код, понятный компьютеру, совершенно непостижим для обычного человека. Трудно представить, каким авторитетом пользовались в 60-х годах прошлого столетия компьютерные гуру, вырезавшие код программы в виде дырочек на перфокартах и перфолентах.

Сейчас в программировании на машинном коде нет необходимости. Люди давно изобрели много разных языков программирования, которые выступают в роли посредника между человеком и машиной. *Язык программирования* — это программа, которая «читает» текст *исходного кода*, содержащего более привычные команды, такие как `print` (печать) или `copy` (копировать), и автоматически преобразует их в двоичные машинные команды. По способу преобразования исходного кода в машинный код языки программирования подразделяются на *компилируемые* и *интерпретируемые*.

## Компилируемые языки программирования

Код, написанный на компилируемом языке, передается в программу-компилятор, которая прежде всего проверяет код на наличие ошибок, а затем переводит текст программы в бинарный машинный код и сохраняет результат в исполняемом файле (с расширением `.exe`) или в библиотеке программных модулей (с расширением `.dll`). Если компилятор обнаруживает в исходном коде ошибки, процесс компиляции останавливается и разработчику отправляется список ошибок. После исправления ошибок компиляция начинается сначала. Таким образом, невозможно скомпилировать программу, в коде которой есть синтаксические ошибки. К сожалению, компилятор не защищает разработчика и будущих пользователей от логических ошибок в коде программы, которые уже невозможно исправить в конечном файле. Другая проблема состоит в несовместимости программ, созданных для разных операционных систем и компьютерных платформ. Так, для того чтобы программа, написанная на C++, работала на компьютерах с операционными системами Windows, OS/2 и Macintosh, необходимо использовать соответствующие компиляторы. Кроме этого, часто приходится вносить изменения в исходный код программы, поскольку компиляторы разных систем стандартизированы не в полной мере.

К компилируемым языкам программирования относятся языки семейства C (C, C++ и C#), Java и многие другие.

## Интерпретируемые языки программирования

Программы-интерпретаторы, в отличие от компиляторов, считывают исходный код построчно и переводят в машинный код каждую строку отдельно. Программы на интерпретируемых языках могут выполняться только в том случае, если на компьютере установлен соответствующий интерпретатор. Из-за того, что каждый раз при запуске программы интерпретатору приходится вновь и вновь переводить исходный текстовый код в машинный, интерпретируемые программы выполняются почти в 1 000 раз медленнее компилированных программ. С другой стороны, поскольку исходный код всегда находится под рукой, легко изменять его и исправлять обнаруженные логические ошибки.

JavaScript относится к интерпретируемым языкам программирования. В качестве интерпретатора выступает обозреватель Internet Explorer или Netscape Navigator. Поэтому сценарии на JavaScript не могут выполняться вне обозревателя. Вместе с тем привязка сценариев JavaScript к рабочей среде обозревателя делает их менее чувствительными к платформе и операционной системе пользователя. Базовые команды JavaScript должны выполняться одинаково как на компьютерах PC, так и Macintosh в соответствующих обозревателях.


Выполнение сценариев JavaScript стандартизировано еще далеко не в полной мере.

## Добро пожаловать в мир JavaScript

Чтобы лучше познакомиться с JavaScript, просто приступайте к написанию сценариев в Web-страницах. Сценарий в действии можно посмотреть, открыв Web-страницу в окне обозревателя. На рис. 1.1 сценарий на JavaScript приветствует нас в окне сообщения, которое открывается автоматически во время загрузки Web-страницы, код которой показан в листинге 1.1.

**Листинг 1.1. Добро пожаловать в мир JavaScript**

```
<HTML>
<HEAD>
<TITLE>Добро пожаловать в мир JavaScript</TITLE>
</HEAD>
```

```
<BODY ONLOAD="alert('Добро пожаловать в мир JavaScript!')">
<P>Окно с приветствием показывает функция <B>alert</B>,
присвоенная событию <B>ONLOAD</B></P>

</BODY>
</HTML>
```

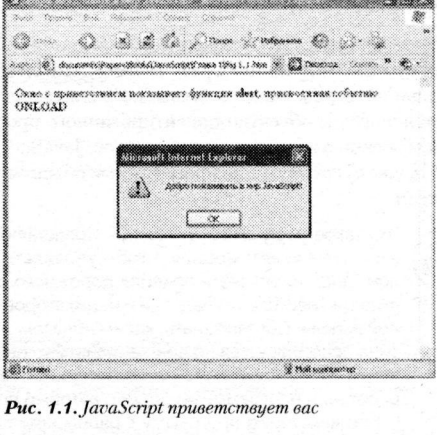
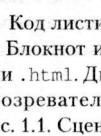



***Рис. 1.1.*** *JavaScript приветствует вас*




С дескрипторами языка HTML, функциями и назначением функций событиям вы познакомитесь более подробно в главе 2.

Код листинга 1.1 можно ввести в простейшем текстовом редакторе Блокнот или WordPad и сохранить в файле с расширением `.htm` или `.html`. Двойной щелчок на имени файла откроет документ в окне обозревателя, установленного на компьютере, как показано на рис. 1.1. Сценарий JavaScript на этой странице сводится к единственной строке:

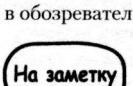
```
ONLOAD="alert('Добро пожаловать в мир JavaScript!')"
```

В этой строке стандартная функция языка JavaScript — `alert()` — назначается событию `ONLOAD`, которое активизируется каждый раз при загрузке Web-страницы. В результате выполняется функция

alert, которая отображает в окне сообщения текст, добавленный между скобками функции. Любые изменения функции вступят в силу сразу же после повторного открытия Web-страницы.

## Основы объектно-ориентированного программирования

Вам, конечно, приходилось слышать об объектно-ориентированных языках программирования. Сразу скажем, что JavaScript не относится в полной мере к таким языкам, но в синтаксисе JavaScript используются подходы объектно-ориентированного программирования. И, что особенно важно для разработчиков, JavaScript позволяет использовать уже существующие программные объекты, встроенные в обозреватели.




Что такое программный *объект*? Представьте себе, что вы водитель автомобиля. Чтобы управлять автомобилем, достаточно знать правила дорожного движения и функции нескольких педалей и индикаторов на приборной панели. Полезно знать, каким образом энергия бензина преобразуется во вращение колес, но для управления автомобилем это не обязательно. Точно так же работает и программный объект, который представляет собой некоторую программу с рядом ключевых слов, с помощью которых можно управлять объектом, или использовать его для выполнения определенных задач.

Чтобы воспользоваться объектом на практике, нам необходимы сведения о свойствах, методах и событиях данного объекта, а также ключевое слово, с помощью которого можно создать *экземпляр* объекта. Посмотрите, например, как легко на Web-странице можно создать командную кнопку:

```
<INPUT TYPE='button' VALUE='Щелкни здесь' ONCLICK="self.
status='Выполнен щелчок на кнопке'">
```

Нам не нужно писать программу, которая управляла бы самой кнопкой в окне обозревателя в ответ на наведение указателя мыши и при выполнении щелчка. Эта программа уже встроена в обозреватель и вызывается дескриптором <INPUT TYPE='button'>, или просто <BUTTON>. Кроме того, в данной программе описано свойство VALUE,

которому присваивается надпись на кнопке, и событие ONCLICK, которому назначается функция обработки события щелчка на кнопке. В нашем примере щелчок на кнопке **Щелкни здесь** выводит в строке состояния текст «Выполнен щелчок на кнопке», как показано на рис. 1.2.

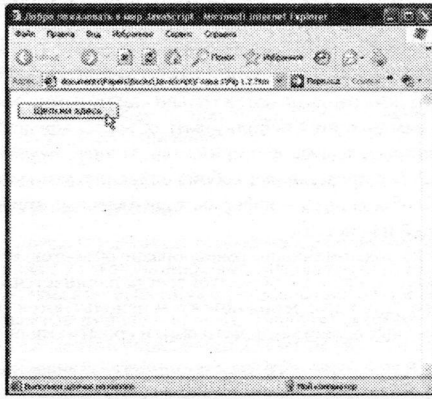



***Рис. 1.2.*** *Использование объекта командной кнопки*


Выше говорилось, что в программах используются объекты. При этом создаются их экземпляры. Следует правильно понимать взаимосвязь между *объектом* и *экземпляром*. Объект — это то же самое, что рецепт яичницы в кулинарной книге, по которому можно приготовить настоящую яичницу, — экземпляр объекта. Такое же отличие между описанием кнопки в программном коде и кнопкой в окне обозревателя, показанной на рис. 1.2. Экземплярам объектов в коде программы присваиваются уникальные *имена*.

## Объекты и их атрибуты

Наиболее часто в сценариях JavaScript нам придется обращаться к объектам, представляющим собой элементы Web-страницы. С одним из таких элементов — командной кнопкой — вы уже познакомились в предыдущем разделе. Кнопка относится к элементам управления

формы, среди которых еще есть текстовые поля, флажки, переключатели, списки и пр. Базовые объекты часто объединяются в объектах-контейнерах. Например, таким объектом является форма, содержащая коллекцию элементов управления. Форма, в свою очередь, может быть элементом объекта рамки, а рамка — элементом объекта `document`, содержащего все элементы Web-страницы, отображенные в объекте еще более высокого уровня, — `window`. Даже абзац или выделенный фрагмент текста также являются объектами. Таким образом, все, что мы видим на Web-странице, — это объекты разных типов.

Но объекты не ограничиваются только «видимыми» элементами. В сценариях нам придется использовать объекты, представляющие собой коллекции полезных функций, или, точнее, *методов*. Например, объект `String` представляет собой коллекцию методов для работы с текстом, а объект `Math` — набор математических и тригонометрических функций и констант.

Несмотря на ошеломляющее разнообразие объектов, между ними есть много общего. Работа с объектом всегда начинается с создания его экземпляра. Для управления объектом используются его атрибуты, представленные свойствами, методами и событиями объекта.


В этой главе мы рассмотрим многие стандартные объекты, используемые в сценариях JavaScript. Список и описание дополнительных объектов вы можете найти в документации обозревателей, в том числе и в Интернет.

- ***Свойства*** определяют вид и особенности (*поведение*) объекта. К свойствам относятся такие атрибуты, как ширина и высота рамки элемента в окне обозревателя, цвет, текст и пр. Все свойства могут устанавливаться и изменяться динамически во время просмотра Web-страницы с помощью сценариев JavaScript.
- ***Методы*** представляют собой встроенные функции, предназначенные для выполнения объектом определенных задач. Например, методы объекта `Math` используются для выполнения математических вычислений, а метод `focus` элементов Web-страницы выделяет соответствующий экземпляр объекта цветом, рамкой и переносом курсора.
- ***События*** устанавливают взаимосвязь между действием пользователя над объектом и внешней функцией обработки

события. Например, щелчок мышью на кнопке вызывает событие ONCLICK и, соответственно, функцию, назначенную этому событию.

Синтаксис обращения ко всем атрибутам объектов одинаков: *имя_экземпляра.атрибут*. Более подробно об именовании и иерархии объектов будет рассказано в следующем разделе.


Можно использовать не только готовые встроенные объекты, но и создавать свои пользовательские объекты, например для временного сохранения промежуточных данных. Создание объектов новых типов выходит за рамки этой книги.

## Объекты и их имена

Web-страница представляет собой набор объектов. Одни объекты содержат в себе полезную информацию, например: текст или рисунок. Другие объекты выполняют роль контейнеров и содержат коллекции объектов нижнего уровня. Так, форма представляет собой коллекцию элементов управления. Объекты образуют многоступенчатую иерархию, как показано на рис. 1.3.

Объект высшего уровня представлен окном обозревателя. Окно содержит либо отдельный документ, либо структуру рамок, в каждой из которых содержится свой документ. Документы могут содержать абзацы текста, таблицы, рисунки, формы, причем эти элементы могут быть вложенными друг в друга (см. рис. 1.3)

Чтобы управлять объектами Web-страницы с помощью сценариев, необходимо иметь возможность обращаться к каждому объекту по имени.

- Объектам верхнего уровня автоматически присваиваются стандартные имена:
  - `window` — окно;
  - `document` — документ в окне или рамке;
  - `form` — форма.


Стандартные имена можно использовать только в том случае, если на данном уровне в иерархии больше нет объектов такого типа.

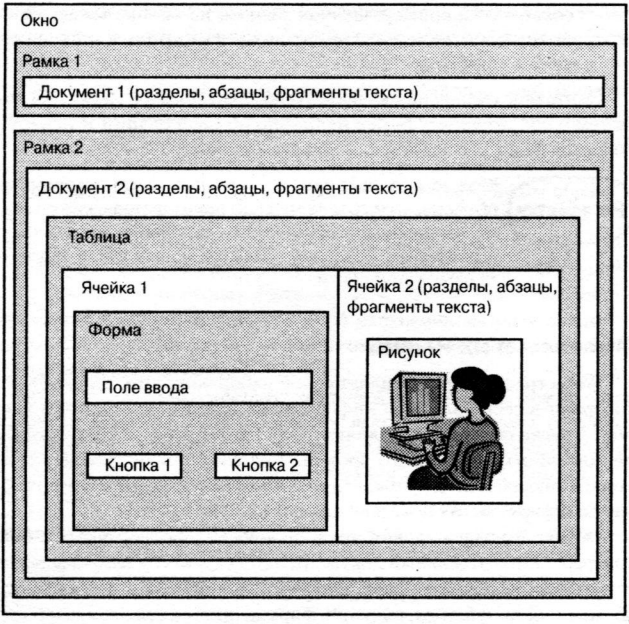



***Рис. 1.3.*** *Иерархия объектов окна обозревателя*

- Объекты автоматически добавляются в соответствующие коллекции под пользовательскими именами и порядковыми номерами в той последовательности, в какой они создавались в коде Web-страницы (подробнее коллекции объектов описаны в главе 6). Вот некоторые стандартные коллекции:
  - `all[]` — все элементы страницы;
  - `frames[]` — коллекция рамок;
  - `forms[]` — коллекция форм;
  - `elements[]` — коллекция элементов управления формы;
  - `images[]` — коллекция рисунков.
- Объектам можно присваивать пользовательские имена с помощью атрибутов `NAME` и `ID` (подробнее об этом см. в главе 2).

Пользовательское именование объектов не препятствует обращению к объекту по стандартному имени (если объект уникален) или как к элементу коллекции.

Обращаясь к объекту, указывайте путь в иерархии. При этом используется синтаксис, показанный на рис. 1.4.

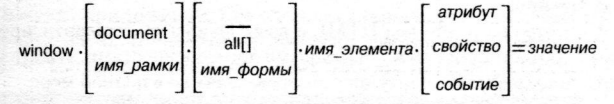



***Рис. 1.4.*** *Синтаксис обращения к элементам Web-страницы*

Имена объектов в иерархии разделяются точками. После имени объекта указывается имя атрибута (свойства или события), значение которого нужно изменить. Например, в следующем примере устанавливается цвет фона документа в рамке с именем `main`:

```
window.main.document.bgcolor = 'red'
```

(Подробнее об установке цвета фона документа см. в главе 2.)

Если вы не в полной мере уяснили принципы взаимодействия объектов Web-страницы, не отчаивайтесь. Многое станет ясно в последующих главах, когда мы перейдем к рассмотрению практических примеров.

## Инструменты для создания и тестирования сценариев

Для работы над сценариями вам не потребуется устанавливать на компьютер никаких специальных программ. Все необходимое наверняка у вас уже есть. Для написания сценария, как и для редактирования HTML-кода Web-страницы, можно использовать простейшие текстовые редакторы — Блокнот и WordPad. Проверить работоспособность сценария сразу же после написания можно с помощью приложения обозревателя, установленного на вашем компьютере. Для создания эффектов анимации вам еще могут потребоваться графический редактор или другие специальные редакторы файлов мультимедиа.

## Обозреватели Интернет

Обозреватели служат для просмотра Web-страниц на серверах, подключенных к сети Интернет. Обозреватель преобразует HTML-код в форматированный текст с рисунками и внедренными объектами мультимедиа, а также выполняет сценарии, находящиеся в коде Web-страницы.

Несмотря на то что код HTML и язык JavaScript стандартизированы в соответствии с международными договоренностями, эти стандарты поддерживаются в разных обозревателях не в полной мере.

### Netsacape и Mozilla

В начале и середине 90-х годов прошлого столетия приложение Netscape Navigator было безусловным лидером среди обозревателей. Именно в компании Netscape разработан язык JavaScript для написания сценариев. Более того, начиная с версии Netscape Navigator 2.0 в программу добавлена утилита для написания и редактирования сценариев — Composer. Для открытия окна редактора в приложении Netscape Navigator нужно выбрать команду **Communicator⇨Composer**. Впрочем, Composer представляет собой довольно примитивный текстовый редактор, сопоставимый с приложением WordPad.

Во второй половине 90-х Navigator был основательно вытеснен приложением Internet Explorer, которое распространялось бесплатно вместе с операционной системой Windows. После неудачной попытки обвинить компанию Microsoft в нерыночных методах конкуренции компания Netscape решила пойти дальше и выступила с инициативой открытия программных кодов приложений-обозревателей, чтобы мировое сообщество пользователей Интернет получило возможность конструировать и настраивать обозреватели в точном соответствии со своими требованиями. Результатом реализации этой инициативы стал обозреватель Mozilla. Доступ к этому приложению и к его программному коду открыт на сервере `www.mozilla.org`.

### Internet Explorer

Сценарии на JavaScript поддерживались исключительно обозревателем Netscape Navigator до тех пор, пока не появился Microsoft Internet Explorer 3.0. Netscape не делилась с Microsoft технологией JavaScript. Разработчики Microsoft создали собственную версию языка, называемого JScript; в его основу были положены стандарты

JavaScript. Таким образом, JavaScript и JScript — это два языка с разными интерпретаторами, но почти идентичным синтаксисом. Для нас данные нюансы не столь важны, поэтому далее мы будем говорить о языке JavaScript независимо от того, для какого обозревателя создаются сценарии. Поскольку приложение Internet Explorer более популярно в странах СНГ, далее в примерах мы будем использовать именно это приложение. Его окно показано на рис. 1.5. Обратите внимание на то, что некоторые сценарии, приведенные в листингах, могут не выполняться в приложениях Netscape Navigator и Mozilla.

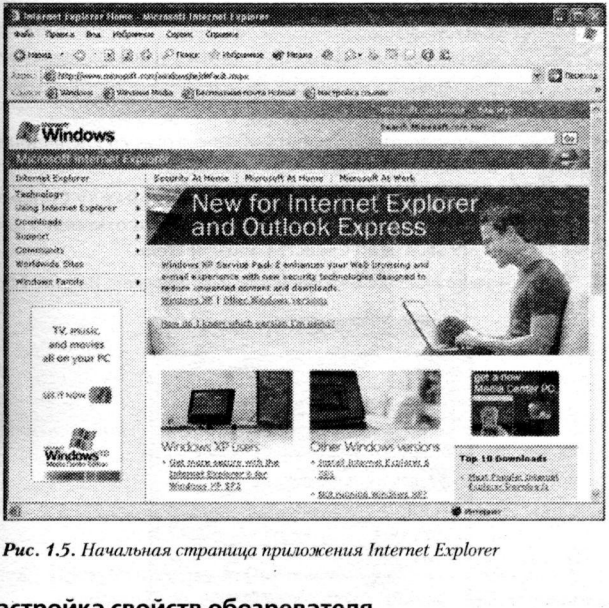



***Рис. 1.5.*** *Начальная страница приложения Internet Explorer*

## Настройка свойств обозревателя

Способность обозревателей воспроизводить элементы Web-страницы зависит от пользовательских настроек свойств обозревателей. Интернет является небезопасной средой работы, через которую распространяются компьютерные вирусы, черви и другие вредоносные программы, в том числе реализованные в виде сценариев

Web-страниц. В обозревателях предусмотрены средства безопасности, ограждающие ваш компьютер от вирусных атак. Но у этой медали есть обратная сторона — чем надежнее системы безопасности, тем хуже обозреватель отображает содержимое Web-страниц, поскольку некоторые элементы попадают в разряд запрещенных. Прежде всего это относится к сценариям на JavaScript и элементам управления ActiveX. Необходимо найти золотую середину, при которой безопасность компьютера будет сочетаться с удобством работы.


Средства безопасности обозревателей не смогут полностью защитить ваш компьютер от вредоносных программ, распространяемых по Интернет. Установите на компьютере лицензионное антивирусное приложение и регулярно обновляйте базу данных вирусов в соответствии с рекомендациями изготовителя программы.


Чтобы настроить работу обозревателя со сценариями, выполните следующее.

1. В окне обозревателя выберите команду **Сервис⇨ Свойства обозревателя**. После этого откроется диалоговое окно **Свойства обозревателя**.
2. Перейдите ко вкладке **Безопасность**, как показано на рис. 1.6.

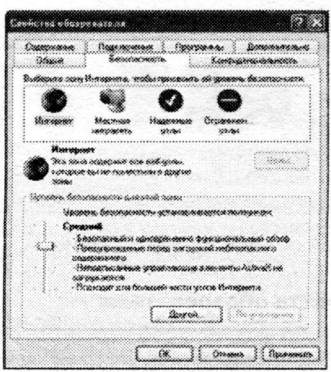



***Рис. 1.6.*** *Вкладка* ***Безопасность*** *диалогового окна* ***Свойства обозревателя***

3. Для настройки работы обозревателя в сети Интернет выберите соответствующий значок в группе **Выберите зону Интернета**, как показано на рис. 1.6.
4. С помощью бегунка в разделе **Уровень безопасности для этой зоны** выберите **Высокий**, **Средний**, **Ниже среднего** или **Низкий** уровень безопасности. По умолчанию установлен уровень **Средний**, что обычно соответствует искомой золотой середине. Если вы подключаетесь к Интернет через локальную сеть, защищенную надежной системой безопасности (технология *брандмауэра* или *огненной стены*), то уровень безопасности обозревателя можно снизить до значений **Ниже среднего** или **Низкий**.
5. Щелкните на кнопке **Другой**. Откроется диалоговое окно **Параметры безопасности**, показанное на рис. 1.7.

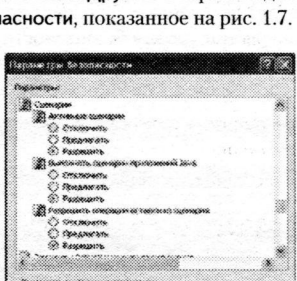


***Рис. 1.7.*** *Настройка выполнения сценариев обозревателем Internet Explorer*

6. Прокрутите список **Параметры** до раздела **Сценарии**. Можно установить для обозревателя выполнение определенного действия в случае обнаружения активного сценария или Java-аплетов на Web-странице либо в том случае, если сценарий пытается записать данные на диск или изменить содержимое окна. Предлагаются следующие варианты:
   - **Отключить** – сценарий не выполняется;

- **Предлагать** — обозреватель показывает диалоговое окно с предложением подтвердить выполнение сценария;
- **Разрешить** — обозреватель выполняет сценарий без предупреждений или каких-либо сообщений.

7. Установите параметры так, как считаете нужным. Если затем вы решите восстановить исходные установки, воспользуйтесь кнопкой **Восстановить** (см. рис. 1.7). Щелкните на кнопке **OK**.
8. Вы вернетесь к диалоговому окну **Свойства обозревателя** (см. рис. 1.7), но в нем исчезнет бегунок выбора уровня безопасности. Чтобы восстановить бегунок, щелкните на кнопке **По умолчанию**.
9. Завершив установку параметров, щелкните на кнопке **OK**.

Обозреватель будет реагировать на сценарии в Web-страницах в соответствии с установками уровня безопасности и параметров. При открытии Web-страницы с активным сценарием может появиться предупреждающее сообщение вроде того, что показано на рис. 1.8.

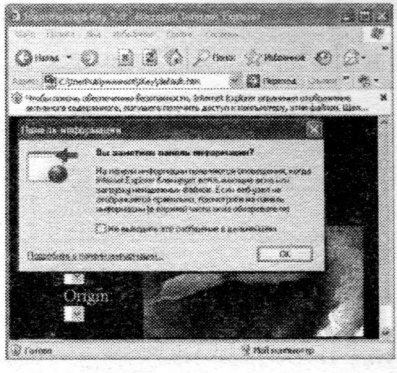



***Рис. 1.8.*** *Предупреждение об обнаружении на странице активного сценария*

Внешний вид окна предупреждения зависит от типа и версии обозревателя, а также от операционной системы, установленной на вашем компьютере. Если вы уверены в надежности и безопасности просматриваемого документа, разрешите выполнение сценариев.

## Текстовые редакторы

Для написания сценариев подойдет любой текстовый редактор. Впрочем, такие сложные редакторы как Microsoft Word лучше не использовать, поскольку эта программа будет пытаться автоматически отформатировать текст документа, что может привести к ошибке. Идеально подходят простейшие текстовые редакторы, такие как Блокнот или WordPad.

Приложение Internet Explorer позволяет извлекать HTML-код Web-страницы и переносить его в текстовый редактор для изменения и сохранения.


Выполните следующие действия.

**1.** Откройте в окне обозревателя интересующую вас Web-страницу. Для этого введите URL-адрес ресурса Интернет, как было показано на рис. 1.5.

**2.** Выберите команду меню **Вид⇨Просмотр HTML-кода**. Обозреватель автоматически запустит приложение Блокнот и отобразит в нем HTML-код Web-страницы, как показано на рис. 1.9.

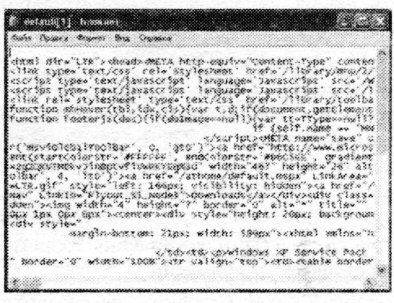


***Рис. 1.9.** HTML-код Web-страницы в окне приложения Блокнот*

**3.** Измените код страницы. Например, добавьте сценарий (вы научитесь это делать в главе 2), выберите команду **Файл⇨Сохранить как** и сохраните документ в файле с расширением `.htm` или `.html`.

## Графические редакторы

На Web-страницах помимо текста часто используются звуковые и видеоклипы, элементы управления ActiveX, рисунки и графические элементы оформления. Для создания объектов разных типов применяются определенные редакторы. У нас нет возможности рассматривать средства редактирования всех элементов Web-страниц. Остановимся вкратце только на графических редакторах, поскольку рисунки и другие графические элементы встречаются на Web-страницах так же часто, как и текст.

На Web-страницах используются графические элементы, сохраненные в файлах следующих форматов.

- **JPEG** (Joint Photography Experts Group — Объединенная группа экспертов в области фотографии). Файлы этого формата сохраняются с расширением `.jpg`. Применяется для сохранения полноцветных изображений с миллионами цветовых оттенков. При этом файлы JPEG характеризуются относительно небольшими размерами за счет эффективного сжатия, хотя сжатие файлов происходит с потерей качества изображения. Идеальный формат для цифровых фотографий и сканированных изображений.
- **GIF** (Graphics Interchange Format — формат графического обмена). Файлы этого формата сохраняются с расширением `.gif`. Применяется для сохранения рисунков с ограниченным набором цветов — не более 256. Поддерживает эффект прозрачности областей изображения. Эффективно сжимает файлы без потери качества. В одном файле можно сохранять серии изображений. Идеально подходит для сохранения рисунков и элементов графического оформления Web-страниц, а также для создания эффектов анимации.
- **PNG** (Portable Network Graphics — переносимая сетевая графика). Файлы данного формата сохраняются с расширением `.png`. Сочетает в себе все преимущества форматов JPEG и GIF, включая полноцветность, эффективное сжатие без потери качества, поддержку эффекта прозрачности. Недостаток состоит в том, что не все обозреватели способны отображать изображения в этом формате.

Есть много графических редакторов, которые можно использовать для создания, изменения и сохранения графических изображений для использования на Web-страницах. Наибольший выбор средств и инструментов предоставляет профессиональное приложение Adobe Photoshop. Замечательное свойство этого приложения состоит в том, что оно предоставляет алгоритм оптимизации графики для использования в Интернет. Этот алгоритм запускается командой меню **Файл⇨Сохранить для Web**. В открывшемся диалоговом окне **Save For Web** (Сохранить для Web) можно установить различные параметры оптимизации графики: выбрать формат, выполнить индексирование цветов рисунка, установить прозрачность, применить режим чересстрочного вывода изображения на экран (рис. 1.10).

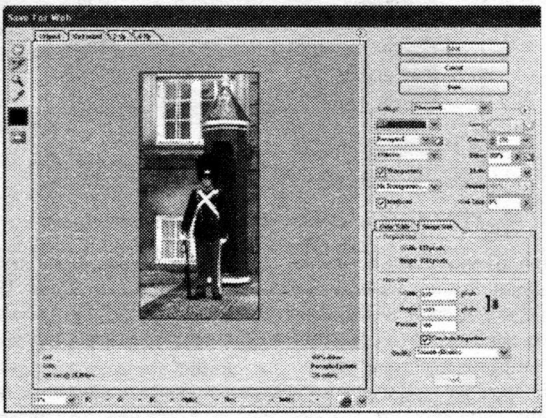



***Рис. 1.10.*** *Сохранение изображения для Web в приложении Adobe Photoshop*

Недостаток приложения Adobe Photoshop заключается в его сложности. Начинающему пользователю не просто научиться применять средства и инструменты этого приложения, а также разобраться в смысле параметров, которые необходимо установить в диалоговых окнах.

К счастью, рисунки для Web-страниц можно создавать с помощью такого простого графического редактора, как Paint, который входит

в набор стандартных приложений Windows. Для использования этого редактора, окно которого показано на рис. 1.11, не нужно профессиональных знаний и навыков.

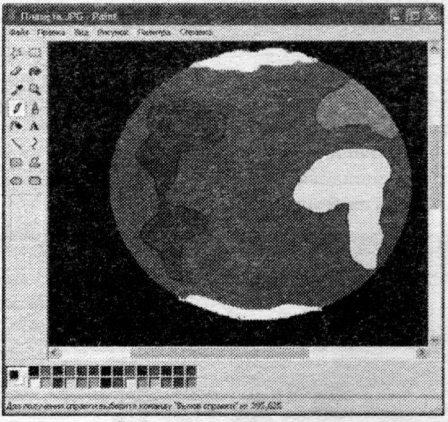


***Рис. 1.11.*** *Окно графического редактора Paint*

Приложение Paint позволяет сохранять изображения в любом из трех форматов, поддерживаемых обозревателями Интернет. Правда, данное приложение не позволяет создавать прозрачные области, изменять степень сжатия файлов или устанавливать чересстрочный режим выгрузки изображений.

При выборе графического редактора обратите внимание на приложение Paint Shop Pro, которое сочетает в себе простоту интерфейса и достаточно профессиональный набор инструментов редактирования. Более подробно об этом приложении можно узнать на Web-узле `www.jasc.com`.


Следует помнить о том, что потеря качества изображения JPEG происходит при каждом сохранении. Поэтому все редактирование желательно выполнять с изображением в ином формате, например BMP или TIF, а в формате JPEG сохранить только окончательный вариант.

# Глава 2

# JavaScript и HTML

Коды сценариев JavaScript неразрывно связаны с HTML-кодом Web-страницы и предназначены для динамического взаимодействия с элементами Web-страницы, созданными с помощью кода HTML. Поэтому невозможно научиться программированию на JavaScript без элементарных знаний о принципах кодирования содержимого Web-страниц. Но код HTML чрезвычайно прост для понимания и легок в использовании. Прочитав эту главу, вы почувствуете себя достаточно уверенно в этом вопросе, чтобы сделать следующий шаг и приступить к программированию сценариев.

## Общие сведения о коде HTML

HTML — HyperText Markup Language (язык гипертекстовой разметки) — это средство создания Web-страниц. Web-страница представляет собой текстовый документ, содержащий обычный текст и *дескрипторы* HTML — специальные текстовые командные символы, с помощью которых:

- текст формируется в абзацы и задаются отступы текста;
- выполняется форматирование текста;
- прочерчиваются линии;
- добавляются рисунки и другие объекты мультимедиа;
- создаются гиперссылки на другие Web-страницы;
- добавляются коды сценариев.

### Типы дескрипторов и принципы их использования

В документе HTML дескрипторы выделяются с помощью угловых скобок: `<дескриптор>`. Различают *парные* и *непарные* дескрипторы. Пара состоит из открывающего и закрывающего дескриптора: `<дескриптор>...</дескриптор>`. Эффект парных дескрипторов

распространяется на текст или объект, заключенный между ними. Отсутствие закрывающего дескриптора приводит к ошибке, за исключением случаев использования *непарных* дескрипторов, которые не требуют закрытия.

Различают *блочные* дескрипторы и дескрипторы *форматирования*. Первые отличаются тем, что создают на Web-странице объекты, которым можно присвоить имя и использовать в коде сценария. Дескрипторы форматирования просто изменяют вид фрагмента текста или объекта на экране, но не создают никаких новых объектов.

Различают также *обязательные* и *необязательные* дескрипторы. Большинство дескрипторов используются по необходимости, но без дескрипторов, показанных в листинге 2.1, невозможно создать Web-страницу.

**Листинг 2.1. Обязательные дескрипторы создают заголовок и основной раздел Web-страницы**

```
<HTML>
<HEAD>
     <TITLE>Моя страница</TITLE>
</HEAD>
<BODY>
     <P>Текст страницы</P>
</BODY>
</HTML>
```

Пара дескрипторов `<HTML>...</HTML>` создает объект документа Web-страницы, к которому в сценарии можно обратиться по ключевому слову `document`. Документ HTML состоит из двух разделов — заголовка и основного раздела. Содержимое Web-страницы, отображаемое в окне обозревателя, находится в основном разделе, тогда как заголовок содержит вспомогательную информацию о странице, как показано на рис. 2.1.

## Ввод кода Web-страницы

Для ввода и изменения кода Web-страницы лучше всего использовать простейший текстовый редактор Блокнот или WordPad. Обозреватель не различает регистр букв и игнорирует все дополнительные пробелы и разрывы строк. Тем не менее для упрощения работы с кодом Web-страницы желательно придерживаться некоторых общих стилистических правил. Например: вводите дескрипторы прописны-

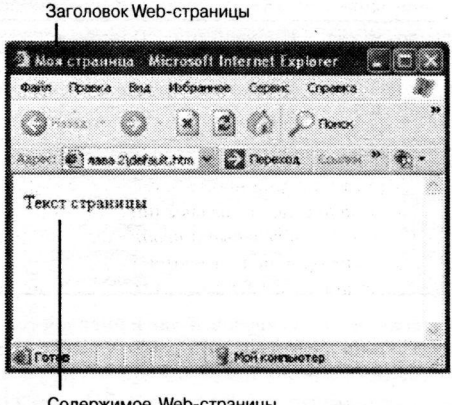



***Рис. 2.1.*** *Отображение в окне обозревателя данных из основного раздела документа HTML и раздела заголовка*

ми буквами, чтобы выделить их в тексте документа; используйте отступы и разрывы строк для выделения логических блоков программного кода.

## Ввод символов

В тексте документа HTML можно использовать все текстовые символы, вводимые с клавиатуры, а также специальные символы. Для ввода специальных символов используется код, представленный в табл. 2.1.

**Таблица 2.1. Коды специальных символов**

| Символ | Числовой код | Именной код | Описание |
|---|---|---|---|
| " | &#34 | &quot | прямые двойные кавычки |
| & | &#38 | &amp | амперсанд |
| < | &#60 | &lt | меньше |
| > | &#62 | &gt | больше |
| € | &#162 | &cent | евро |
| £ | &#163 | &pound | фунт |
| ¥ | &#165 | &yen | иена |

*Окончание табл. 2.1*

| Символ | Числовой код | Именной код | Описание |
|---|---|---|---|
| § | &#167 | &sect | параграф |
| © | &#169 | &copy | авторское право |
| ® | &#174 | &reg | торговый знак |
| ± | &#177 | &plusmn | плюс-минус |
| ´ | &#180 | &acute | ударение |
| µ | &#181 | &micro | микрон |
| ¶ | &#182 | &para | абзац |
| × | &#215 | &times | умножение |
| ÷ | &#247 | &divide | деление |

Можно использовать как числовой, так и именной код. Оба варианта являются стандартными для всех обозревателей. Использовать код вместо символов следует в том случае, когда в тексте необходимо ввести символ, зарезервированный в языке HTML в качестве командного. Так, символы угловых скобок < и > выделяют в тексте Web-страницы дескрипторы HTML. Что же делать, если эти символы нужно ввести как обычный текст. Например, набор символов `<P>` автоматически создаст новый абзац в тексте. Чтобы ввести текст «**<P>**», воспользуемся кодом из табл. 2.1: `&#60P&#62`.

## Комментарии в коде HTML

Во время работы над HTML-кодом Web-страниц рекомендуется добавлять комментарии, напоминающие разработчику, какой элемент страницы определяется в данной части кода. Комментариями называются строки текста в программном коде, которые не отображаются в окне обозревателя и не влияют на внешний вид и работу других элементов Web-страницы. В код HTML комментарии добавляются с помощью управляющих символов `<!-...-->`, между которыми заключается текст комментария.

## Создание новой Web-страницы


Создание Web-страницы можно очень удобно совместить с проверкой ее работоспособности. Давайте создадим простейшую Web-страницу.

1. В приложении Проводник перейдите к папке, или создайте новую папку, в которую вы хотите поместить файл Web-страницы.
2. На правой панели щелкните правой кнопкой мыши и выберите в контекстном меню команду **Создать⇨Текстовый документ**, как показано на рис. 2.2.

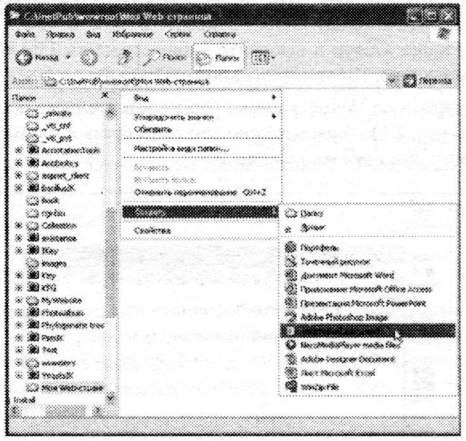


***Рис. 2.2.*** *Создание нового текстового документа*

3. В папке появится значок нового текстового документа, имя которого, `Текстовый документ.txt`, будет выделено для редактирования. Введите новое имя, например `default.htm` или `index.html`. Нажмите <Enter>. Появится предупреждающее сообщение о том, что было изменено расширение файла. Щелкните на кнопке **Да**.

**На заметку**

Файлам начальных страниц обычно присваивается имя `default.htm` или `index.html`. Обозреватель автоматически открывает эти файлы, если в URL-адресе имя файла не было указано.

4. Щелкните на значке файла правой кнопкой мыши и выберите в контекстном меню команду **Открыть с помощью⇨Блокнот**.

Документ, пока что пустой, будет открыт в окне приложения Блокнот.

5. Введите HTML-код Web-страницы, например, показанный в листинге 1.1. Выберите команду **Файл⇨Сохранить**, чтобы сохранить документ.
6. Запустите обозреватель, установленный на вашем компьютере. Выберите команду **Файл⇨Открыть**. В окне **Открыть** щелкните на кнопке **Обзор** и выберите вновь созданный файл на диске компьютера. Щелкните на кнопке **Открыть**.
7. Содержимое Web-страницы отобразится в окне обозревателя (см. рис. 2.1). Теперь можно работать сразу с двумя приложениями, как показано на рис. 2.3.

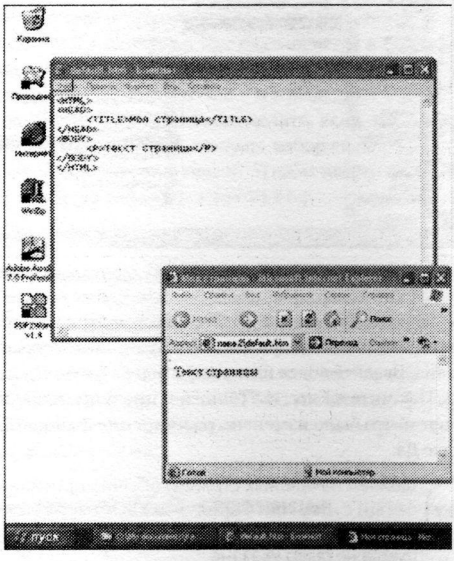



***Рис. 2.3.*** *Файл Web-страницы можно одновременно открыть в окне текстового редактора и окне обозревателя*

8. Внесите изменения в HTML-код или код сценария и сохраните документ, после чего в окне обозревателя щелкните на кнопке **Обновить**, чтобы посмотреть, какие изменения произошли во внешнем виде и функционировании Web-страницы.

# Настройка свойств элементов Web-страниц

Элементы Web-страниц создаются с помощью дескрипторов HTML. Для создания элементов разных типов используются свои специфические дескрипторы, о чем подробнее речь пойдет в следующем разделе этой главы. Но настройка свойств объектов выполняется по общему плану:

```
<ДЕСКРИПТОР АТРИБУТ_1='значение' АТРИБУТ_2='значение'
```

## Атрибуты дескрипторов HTML

Дескрипторы разных типов имеют неодинаковые наборы атрибутов. Дескрипторы форматирования, как правило, не имеют никаких настраиваемых свойств. В табл. 2.2 перечислены некоторые атрибуты, которые используются наиболее часто.

**Таблица 2.2. Атрибуты дескрипторов HTML**

| Атрибут | Описание | Дескрипторы, в которых применяется атрибут |
|---|---|---|
| ID и NAME | Уникальное имя объекта, которое можно использовать в сценариях. С назначением этих атрибутов возникла некоторая путаница. Вы поступите благоразумно, если присвоите одинаковое имя обоим атрибутам объекта, к которому хотите получить доступ в сценариях | Все блочные дескрипторы |
| ALIGN | Выравнивание объекта в окне обозревателя или в рамке | Рисунок, таблица, абзац |
| BACKGROUND | Фоновый рисунок | Документ, таблица, ячейка таблицы |

*Окончание табл. 2.2*

| Атрибут | Описание | Дескрипторы, в которых применяется атрибут |
|---|---|---|
| BORDER | Прорисовка границ объекта | Рамка, рисунок, таблица, |
| COLOR, BGCOLOR и BORDERCOLOR | Цвет объекта, цвет фона объекта и цвет границы объекта | Документ, таблица, ячейка таблицы, текст |
| HEIGHT и WIDTH | Высота и ширина рамки объекта | Бегущая строка, плавающая рамка, рисунок, проигрыватель файлов мультимедиа, таблица, ячейка таблицы |
| INNERHTML и INNERTEXT | Содержимое блочных текстовых объектов | Абзац, раздел текста, фрагмент текста, таблица, ячейка таблицы |
| SRC | Файл источника данных | Рамка, рисунок, сценарий |
| STYLE | Стиль объекта | Все блочные объекты |

## Дескрипторы форматирования текста и фона Web-страницы

Обозреватель игнорирует все пробелы в тексте, в том числе и символы конца абзаца, вводимые нажатием клавиши <Enter>. Чтобы добавить новый абзац, нужно расположить текст абзаца между дескрипторами <P>...</P>. Обозреватель автоматически добавит пустую строку между блоками текста, относящимися к разным абзацам. По умолчанию обозреватель выравнивает текст абзаца влево и отображает текст шрифтом черного цвета. Тип и размер шрифта выбираются в соответствии с текущими установками в параметрах обозревателя.

Разработчики Web-страниц могут отформатировать текст абзаца, используя для этого дескрипторы форматирования и позиционирования, представленные ниже.

### Выравнивание абзацев текста

Выравнивание текста в абзаце можно изменить, установив соответствующий параметр атрибута ALIGN в дескрипторе <P>, как показано в листинге 2.1.

**Листинг 2.1. Выравнивание абзацев**

```
<BODY>
  <P ALIGN = "left" >Абзац выровнен влево</P>
  <P ALIGN = "center" >Абзац выровнен по центру</P>
  <P ALIGN = "right" >Абзац выровнен вправо</P>
  <P ALIGN = "justify" >Абзац выровнен по ширине</P>
</BODY>
```

Результаты форматирования текста показаны на рис. 2.4.

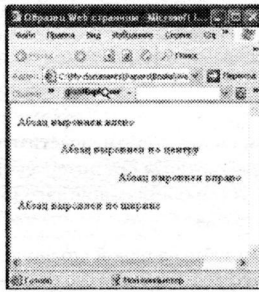
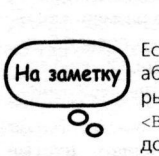


***Рис. 2.4.*** *Абзацы Web-страницы с установкой различных параметров выравнивания текста*



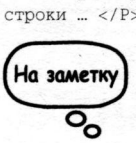

Если текст нужно начать с новой строки, но не с нового абзаца, воспользуйтесь непарным дескриптором разрыва строки <BR>. Текст, следующий за дескриптором <BR>, обозреватель покажет с новой строки, но не будет добавлять пустую строку, как в случае с дескрипторами <P>...</P>.

## Пробелы и отступы

Как уже говорилось выше, обозреватель игнорирует все дополнительные символы пробелов в тексте, включая символы табуляции. Поэтому средства текстовых редакторов, применяемые для установки отступов и пробелов в тексте, обычно не подходят для форматирования текста документа HTML. Чтобы ввести дополнительный пробел, воспользуйтесь комбинацией управляющих символов  . Повторите эту комбинацию символов несколько раз, чтобы ввести

подряд несколько пробелов. Так, в следующем примере создается абзац, начинающийся с красной строки в 5 символов:

```
<P>    &nbspЭтот абаза начинается с красной
строки … </P>
```




Комбинация символов   устанавливает неразрывный пробел. Слова, разделенные неразрывным пробелом, не будут отделяться при формировании строк в окне обозревателя. Используйте неразрывный пробел вместо обычного, если необходимо гарантировать размещение нескольких слов или символов, разделенных пробелами, в одной строке текста.

## Заголовки

Пробежав взглядом по заголовкам, посетитель страницы быстро отыщет ту информацию, которая его интересует. В HTML используются дескрипторы, устанавливающие заголовки шести уровней от <H1>…</H1> до <H6>…</H6>.


Не используйте на одной странице больше трех уровней заголовков. Слишком сложная иерархия заголовков затруднит восприятие информации документа. Кроме того, примите к сведению, что шрифты только первых трех уровней больше размера основного текста страницы. Шрифт 4-го уровня имеет такой же размер, как и основной шрифт, а размеры шрифтов заголовков 5-го и 6-го уровней меньше основного шрифта. Обычно заголовки 5-го и 6-го уровней используют для выделения примечаний или сносок.


Обозреватели автоматически добавляют пустые строки перед текстом заголовка и после него. Введем код, приведенный в листинге 2.2.

**Листинг 2.2. Заголовки разных уровней**

```
<H1>Так выглядит заголовок 1-го уровня</H1>
<H2>Так выглядит заголовок 2-го уровня</H2>
<H3>Так выглядит заголовок 3-го уровня</H3>
<H4>Так выглядит заголовок 4-го уровня</H4>
<H5>Так выглядит заголовок 5-го уровня</H5>
<H6>Так выглядит заголовок 6-го уровня</H6>
```

Введенные нами заголовки отобразятся в окне обозревателя, как показано на рис. 2.5.

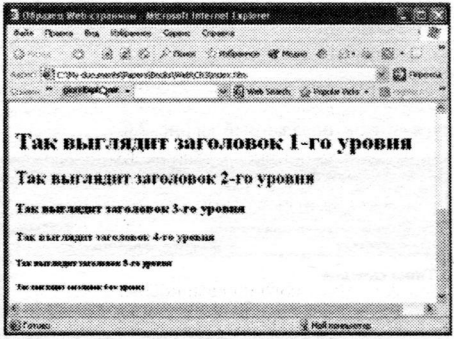



***Рис. 2.5.*** *Заголовки текста разных уровней*

## Списки

Список выглядит более наглядно и легче воспринимается с экрана компьютера, чем обычное перечисление пунктов в тексте. На Web-страницах можно использовать нумерованные, маркированные или смешанные списки. Выбор типа списка зависит от того, насколько важна очередность пунктов в списке. Например, рецепт приготовления какого-либо блюда лучше представить с помощью нумерованного списка, тогда как для перечисления блюд в меню лучше использовать маркированный список.

### *Создание нумерованного списка*

Для создания нумерованного списка используется комбинация двух пар дескрипторов:

- `<OL>...</OL>` — устанавливают начало и конец нумерованного списка;
- `<LI>...</LI>` — отмечают отдельные пункты списка.

Пример

Введем код листинга 2.3.

**Листинг 2.3. Нумерованный список**

```
<H2>Типы списков</H2>
<OL>
  <LI>Нумерованный</LI>
  <LI>Маркированный</LI>
  <LI>Многоуровневый</LI>
</OL>
```

Получится список, показанный на рис. 2.6.

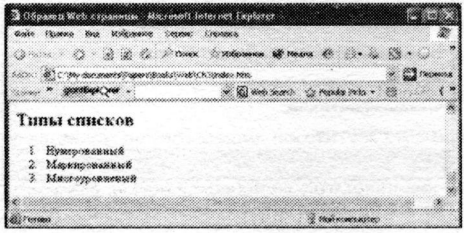



***Рис. 2.6.*** *Нумерованный список*

### *Маркированные и многоуровневые списки*

Маркированный список следует применять для перечисления неупорядоченных равнозначных объектов, например блюд в меню ресторана. В том случае если существует иерархия объектов, удобно использовать многоуровневые списки. Пункты списка всегда задаются дескрипторами <LI>...</LI>, но чтобы указать обозревателю, что данный список является маркированным, используются дескрипторы <UL>...</UL>.


Многоуровневый список создать очень легко. Просто добавьте новый список после пункта, требующего детализации, внутри другого списка, как в листинге 2.4.

**Листинг 2.4. Маркированный и многоуровневый списки**

```
<H2>Типы списков</H2>
<OL>
  <LI>Нумерованный</LI>
  <OL>
    <LI>Пункт 1</LI>
    <LI>Пункт 2</LI>
```

```
    <LI>Пункт 3</LI>
  </OL>
  <LI>Маркированный</LI>
  <UL>
    <LI>Первый вариант</LI>
    <LI>Второй вариант</LI>
    <LI>Третий вариант</LI>
  </UL>
  <LI>Многоуровневый</LI>
  <OL TYPE="a">
    <LI>Пункт a</LI>
    <LI>Пункт b</LI>
    <LI>Пункт c</LI>
  </OL>
</OL>
```

Созданный нами многоуровневый список показан на рис. 2.7.

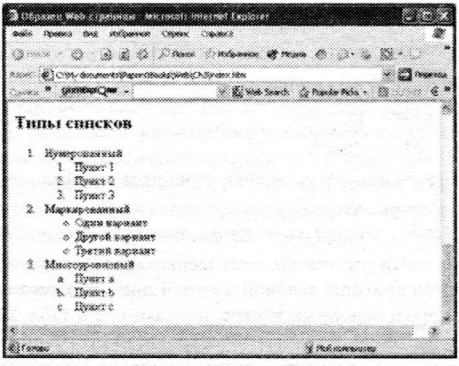
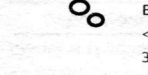



***Рис. 2.7.*** *Многоуровневый список*




Обратите внимание на то, что в последнем вложенном списке пункты отмечаются не цифрами, а латинскими буквами в алфавитном порядке. Это обычный нумерованный список, но в открывающем дескрипторе списка `<OL>` мы установили атрибут `TYPE="a"`. Эта установка заменила цифровую нумерацию, заданную по умолчанию, алфавитной нумерацией.

## Форматирование элементов текста

Дескрипторы форматирования текста, с которыми мы познакомимся в этом подразделе, позволяют изменять формат отдельных слов или даже отдельных текстовых символов. Форматирование используется для того, чтобы акцентировать внимание посетителя на наиболее важных моментах текста. Можно изменять цвет, начертание и размер шрифта, добавлять подчеркивание или перечеркивание, вводить надстрочные и подстрочные символы, применять логическое выделение. Для этого используются следующие дескрипторы:

- `<B>…</B>` — полужирный шрифт;
- `<I>…</I>` — курсив;
- `<U>…</U>` — подчеркивание;
- `<STRIKE>…</STRIKE>` — перечеркивание;
- `<SUP>…</SUP>` — надстрочный шрифт;
- `<SUB>…</SUB>` — подстрочный шрифт;
- `<FONT атрибуты>…</FONT>` — установка атрибутов шрифта:
  - `face` — имя шрифта;
  - `size` — размер;
  - `color` — цвет.

Ниже показан пример установки атрибутов шрифта.

```
<FONT FACE='Times New Roman' SIZE='14' COLOR='#FF0000'>Шрифт
Times New Roman, размер 14 кеглей, красного цвета</FONT>
```

Значение цвета устанавливается шестнадцатеричными числовыми значениями красной, зеленой и синей цветовых составляющих. Для шестнадцати основных цветов, показанных в табл. 2.3, заданы именные коды, которые можно использовать вместо числовых значений. Например, в следующих двух дескрипторах устанавливается красный цвет шрифта:

```
<FONT color='#FF0000'>
<FONT color='red'>
```

**Таблица 2.3. Шестнадцатеричный и именной цветовые коды**

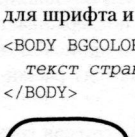
| Цвет | Шестнадцатеричный код | Именной код |
|---|---|---|
| Белый | `#FFFFFF` | `white` |
| Бирюзовый | `#008080` | `teal` |

*Окончание табл. 2.3*

| Цвет | Шестнадцатеричный код | Именной код |
|---|---|---|
| Желтый | #FFFF00 | yellow |
| Зеленый | #008000 | green |
| Золотой | #FFD700 | gold |
| Красно-коричневый | #800000 | maroon |
| Красный | #FF0000 | red |
| Лимонный | #00FF00 | lime |
| Морской волны | #00FFFF | aqua |
| Оливковый | #808000 | olive |
| Пурпурный | #800080 | purple |
| Серебряный | #C0C0C0 | silver |
| Серый | #808080 | gray |
| Синий | #0000FF | blue |
| Слоновой кости | #FFFFF0 | ivory |
| Темно-голубой | #000080 | navy |
| Фуксиновый | #FF00FF | fuchsia |
| Черный | #000000 | black |

Чтобы изменить цвет шрифта для всей страницы — вместо черного, который задан по умолчанию, установить какой-нибудь другой, — воспользуйтесь атрибутом TEXT в дескрипторе <BODY>. Еще один атрибут этого дескриптора — BGCOLOR — задает цвет фона страницы. Например, в следующем примере устанавливается синий цвет для шрифта и желтый цвет для фона страницы:

```
<BODY BGCOLOR='yellow' TEXT='blue'>
  текст страницы...
</BODY>
```




Установки параметров шрифта с помощью дескриптора <FONT> имеют более высокий приоритет и отменяют установки, заданные в дескрипторе <BODY>.

## Графика на Web-страницах

Web-страница без рисунков выглядит скучно и уныло. Но за рисунки и графические элементы оформления Web-страницы приходится платить свою цену. Графика часто значительно увеличивает размер

файла страницы, что замедляет время выгрузки документа по сети Интернет. Так, файл с довольно длинным текстом занимает, как правило, 500–1000 байт и выгружается по сети практически моментально. Наличие небольшого рисунка на странице, размером в 50 Кбайт, приведет к тому, что при подключении к Интернет по телефонному модему Web-страница будет выгружаться на компьютер пользователя примерно 10 с. Пользователи Интернет — нетерпеливые люди. Один щелчок на гиперссылке, и пользователь покинет вашу страницу еще до того, как она выгрузится в окно обозревателя. Поэтому используйте графику на Web-страницах в разумных пределах.

## Добавление рисунков

В Интернет используются графические изображения только трех следующих форматов.

- **GIF** (Graphics Interchange Format — Формат графического обмена). Пожалуй, является основным форматом графики в Интернет, поскольку он поддерживает сжатие без потери данных, передает прозрачность и позволяет сохранять в одном файле серии рисунков для воспроизведения эффектов анимации. Но цветовая палитра не может содержать более 256 цветов. Файлы этого формата сохраняются с расширением `.gif`.
- **JPEG** (Joint Photography Experts Group — Объединенная группа экспертов в области фотографии). Его название является аббревиатурой названия разработавшей его компании. Отличается максимальной скоростью передачи, поддерживает передачу цветовых полутонов, но сжатие файла происходит с потерей данных. Файлы этого формата сохраняются с расширением `.jpg`.
- **PNG** (Portable Network Graphics — Переносимая сетевая графика) является альтернативой форматам GIF и JPEG. В формате PNG реализован метод эффективного сжатия изображений без потери данных, поддержка прозрачности и полноцветности изображений. Предполагалось, что данный формат станет универсальным для Интернет и заменит все остальные форматы, но пока многие обозреватели все еще не поддерживают показ изображений в этом формате. Файлы данного формата сохраняются с расширением `.png`.

Чтобы добавить рисунок на Web-страницу, используется дескриптор <IMG>. Ниже показан пример добавления рисунка в коде HTML:

```
<IMG SRC='images\MyPicture.jpg' HEIGHT=50 WIDTH=100
ALIGN=left ALT='Фотография'>
```

Пример

Давайте добавим рисунок в документ HTML.

1. Прежде всего нужно подготовить рисунок к использованию в Интернет. Откройте файл рисунка в графическом редакторе и выполните следующие действия.

- Проверьте разрешение изображения, определите его размер. В большинстве случаев для демонстрации рисунков на Web-страницах достаточно иметь разрешение в 72 пикселя/дюйм. (В приложении для этого используется диалоговое окно **Размер рисунка**, показанное на рис. 2.8, которое можно открыть командой меню **Изображение⇨Размер изображения**.)
- Уменьшите размер рисунка до требуемого. Размер изображения можно изменять непосредственно в документе HTML, но это не приведет к уменьшению файла рисунка. (Используйте для этого диалоговое окно **Размер изображения**, как показано на рис. 2.8.)

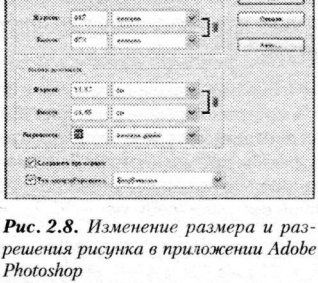


***Рис. 2.8.*** *Изменение размера и разрешения рисунка в приложении Adobe Photoshop*

- Если рисунок содержит ограниченное число цветов, выполните индексирование, чтобы отбросить лишнюю информацию и уменьшить размер файла. В Adobe Photoshop

для этого используется команда меню **Изображение⇨Режим⇨Индексированный**. Установите количество цветов в открывшемся диалоговом окне **Цвет с индексом** (рис. 2.9).

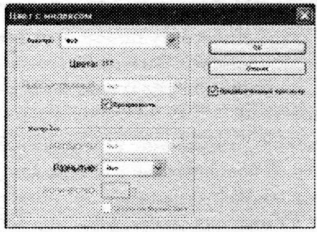


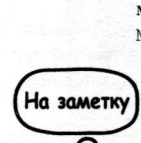
***Рис. 2.9.*** *Индексирование цвета в приложении Adobe Photoshop*

- Сохраните рисунок в одном из форматов, поддерживаемых обозревателями Интернет. Многие современные графические редакторы содержат в своем меню команду **Файл⇨Сохранить для Веб**. Эта команда открывает специальное диалоговое окно, которое позволяет выбрать правильный формат, установить степень сжатия и поддержку прозрачных областей, оптимизировать рисунок для просмотра в обозревателе Интернет, применить к рисунку *чересстрочный* или *прогрессивный* режим выгрузки.


Чересстрочный режим применим к файлам в формате GIF, а прогрессивный – к файлам JPEG. В обоих случаях суть сводится к тому, что изображение выгружается с эффектом постепенного улучшения качества на Web-странице, что позволяет читать документ, не дожидаясь полной выгрузки графики.


Сохранять рисунок в формате JPEG следует только один раз, когда вы полностью завершили редактирование изображения в другом формате. При каждом последующем сохранении качество рисунка в формате JPEG будет ухудшаться.

2. Откройте файл Web-страницы в текстовом редакторе. Выберите в документе место, где должен появиться рисунок (более подробно о позиционировании элементов Web-страницы рассказано далее, в разделе «Разметка Web-страницы»).
3. Введите с клавиатуры дескриптор `<IMG>` и установите в нем следующие атрибуты.
   - `SRC` — путь к графическому файлу. Обычно указывается относительный путь от папки, в которой находится файл Web-страницы, например: `image/MyPicture.jpg`. Можно также указать путь к графическому файлу, размещенному по URL-адресу в Интернет: `http://www.MyPage.org/MyPicture.jpg`.




Если на нескольких Web-страницах имеются одинаковые рисунки или графические элементы, всегда используйте для всех страниц один и тот же графический файл, указав путь к нему в атрибутах `SRC` в разных дескрипторах `<IMG>`. Тем самым вы упростите редактирование Web-страниц в дальнейшем, сохраните место на диске Web-сервера и ускорите время выгрузки Web-страниц по Интернет.

   - `HEIGHT` и `WIDTH` — устанавливают, соответственно, высоту и ширину рисунка в пикселях или в процентах от размера окна обозревателя. Размер файла изображения при этом не изменяется.
   - `ALIGN` — значения `left` и `right` устанавливают выравнивание рисунка на странице, соответственно, влево и вправо с обтеканием текста с противоположной стороны, а значения `top`, `middle` и `bottom` выравнивают текст абзаца, в который добавлен рисунок, по верхнему краю, центру или нижнему краю рисунка.
   - `ALT` — текст для обозревателей, в которых отображение изображений отключено пользователем.
   - `LOWSRC` — путь к рисунку, который будет отображаться до того момента, пока не будет выгружено основное изображение, указанное в атрибуте `SRC`. Обычно в атрибуте `LOWSRC` устанавливается вспомогательное изображение маленького размера с низким разрешением.

## Использование рисунков в виде фона Web-страницы

Любой рисунок в формате JPEG, GIF или PNG может быть использован как в тексте документа HTML, так и в качестве фона Web-страницы. Для этого путь к файлу рисунка присваивается атрибуту BACKGROUND в дескрипторе <BODY>, как в следующем примере:

```
<BODY BACKGROUND='images/MyPicture.jpg'>
```


Для установки цвета фона страницы используется атрибут BGCOLOR. Этому атрибуту присваивается значение цвета (см. табл. 2.3).

Следует помнить, что графика в тексте и в фоне документа применяется по-разному. Обозреватель многократно копирует рисунок в фоне страницы, заполняя им все окно обозревателя. Не используйте слишком сложные рисунки в качестве фона, так как многократное повторение одного и того же изображения затрудняет чтение текста (рис. 2.10).

***Рис. 2.10.*** *Использование рисунков в виде фона может испортить Web-страницу*

В то же время благодаря многократному повторению фон страницы может быть создан с помощью небольшого исходного изображе-

ния, как показано на рис. 2.11. В результате сложный узор в фоне не повлияет на время выгрузки Web-страницы.

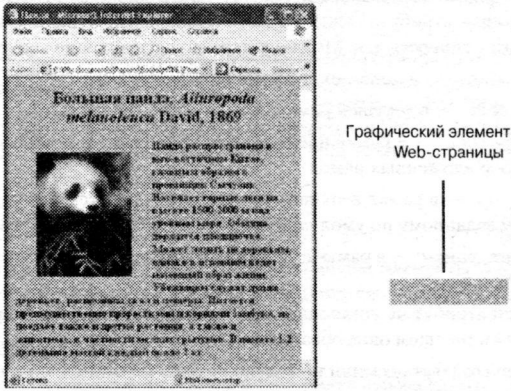



***Рис. 2.11.*** *Фон Web-страницы построен из повторяющихся элементарных рисунков*

## Гиперссылки

Отличительной особенностью Web-страниц в сети Интернет является их взаимосвязанность. Если в Интернет уже существует Web-страница с полезной для вас информацией, нет необходимости копировать ее содержимое в новый готовящийся документ. Просто добавьте в текст своей страницы *гиперссылку* на ресурс, находящийся где-либо в Интернет, адрес которого вам известен.

Ниже показано, как гиперссылка выглядит в коде HTML.

```
<A HREF='http://www.MySite.com/MyPage.htm' TARGET='_blank'>Моя страница</A>
```

Гиперссылка задается парным дескриптором <A>…</A>, а адрес страницы, на которую будет осуществляться переход по щелчку на гиперссылке, устанавливается как значение атрибута HREF в дескрипторе <A>. Чтобы создать гиперссылку на адрес электронной почты, атрибуту HREF нужно присвоить значение следующего типа:

```
href='mailto:MyAddress@MailServer'
```

Команда `mailto` запускает на компьютере пользователя приложение клиента электронной почты, открывает окно нового сообщения и вставляет в поле **Кому** заданный адрес *`MyAddress@MailServer`*.

С помощью атрибута `TARGET` задается способ открытия документа, указанного в гиперссылке. Можно установить следующие значения:

- `'_blank'` — в новом окне обозревателя;
- `'_self'` — в текущей рамке;
- `'_parent'` — в рамке предыдущего уровня, если используется набор вложенных рамок;
- `'_top'` — в рамке верхнего уровня, что соответствует параметру, заданному по умолчанию;
- *`'имя_рамки'`* — в рамке с указанным именем, заданным в атрибуте `name`.
- если атрибут не установлен (по умолчанию), ссылка открывается в текущем окне обозревателя.

Можно создавать ссылки не только на Web-страницы, но и на некоторые документы других форматов. Например, можно создать гиперссылку на звуковой файл, содержащий музыкальное произведение, звуковой эффект или сообщение, записанное в файл с помощью средств цифровой звукозаписи. Обозреватели поддерживают воспроизведение звуковых файлов следующих форматов:

- **MIDI** — музыкальные произведения в файлах с расширением `.mid`;
- **MP3** — музыка, песни и звуковые сообщения в файлах с расширением `.mp3`;
- **RealAudio** — интерактивное радио и звукозапись в файлах с расширением `.ra`;
- **WAV** — звуковые эффекты в файлах с расширением `.wav`.

Обозреватели поддерживают просмотр видеоклипов таких форматов:

- **AVI** — файлы с расширением `.avi`;
- **MPEG** — файлы с расширением `.mpg` или `.mpeg`;
- **QuickTime** — небольшие клипы в файлах с расширением `.mov` или `.qt`.

В атрибуте `HREF` можно также указать путь к графическому файлу в формате JPEG, GIF или PNG, или к документу в формате PDF.

Во всех перечисленных выше примерах при обращении пользователя к гиперссылке обозреватель запустит на компьютере пользователя специальное приложение для воспроизведения или отображения документа в окне обозревателя.


Специальные приложения для работы с документами мультимедиа являются надстройками обозревателя, т.е. они необязательны и могут отсутствовать. В результате мультимедийная гиперссылка может не работать на компьютерах некоторых пользователей.

# Разметка Web-страниц

Чтобы разместить текст, рисунки и другие элементы Web-страницы соответствующим образом, используются таблицы, рамки и формы.

## Таблицы

Для создания таблиц применяются пять пар дескрипторов:

- <TABLE>...</TABLE> — устанавливают начало и конец таблицы;
- <CAPTION>...</CAPTION> — устанавливают заголовок таблицы;
- <TR>...</TR> — добавляют новую строку в таблицу;
- <TH>...</TH> — добавляют в строку ячейку заголовка;
- <TD>...</TD> — добавляют в строку обычную ячейку.

Обозреватель по умолчанию выравнивает заголовок по ширине таблицы и отображает текст в ячейках заголовка полужирным шрифтом с выравниванием по центру ячейки. Для обычных ячеек по умолчанию задается выравнивание по левому краю.


Пример создания таблицы показан в листинге 2.5.

**Листинг 2.5. Создание таблицы**

```
<TABLE>
<CAPTION>Эта таблица создана с использованием<BR>дескрипторов
HTML</CAPTION>
<TR><TH>Фамилия</TH><TH>Имя</TH><TH>Отчество</TH><TH>Год
рождения</TH></TR>
<TR><TD>Скворцова</TD><TD>Валентина</TD><TD>Васильевна</
TD><TD>1973</TD></TR>
```

```
<TR><TD>Баранов</TD><TD>Сергей</TD><TD>Анатольевич</
TD><TD>1969</TD></TR>
<TR><TD>Миронов</TD><TD>Тимофей</TD><TD>Максимович</
TD><TD>1980</TD></TR>
</TABLE>
```

Внешний вид таблицы в окне обозревателя показан на рис. 2.12.

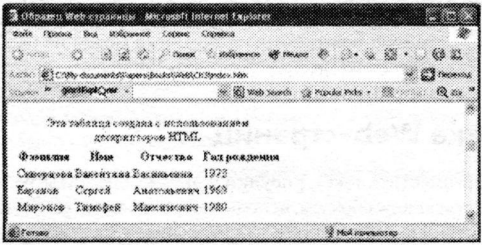



***Рис. 2.12.*** *Таблица, созданная с помощью дескрипторов HTML*

## Установка ширины и высоты таблицы

Таблицы удобно использовать в тех случаях, когда нужно разместить элементы Web-страницы в определенных областях окна обозревателя. Например, чтобы создать колонку текста заданной ширины, достаточно добавить таблицу с единственной ячейкой и установить ширину ячейки в пикселях с помощью атрибута `WIDTH`, как в следующем примере:

```
<TABLE WIDTH='200'><TR><TH>текст</TH></TR></TABLE>
```

В данном примере ширина таблицы установлена в 200 пикселей. Но атрибут `WIDTH` позволяет устанавливать ширину таблицы также в процентах от ширины окна обозревателя:

```
<TABLE width='20%'><TR><TH>текст</TH></TR></TABLE>
```

Атрибут `WIDTH` можно устанавливать не только для всей таблицы, но и для отдельных ячеек. В следующем примере мы создаем две колонки текста шириной в 100 и 200 пикселей:

```
<TABLE><TR><TH width='100'>текст</TH><TH WIDTH='200'>текст</
TH></TR></TABLE>
```

Аналогичным образом высота таблицы и отдельной ячейки устанавливается с помощью атрибута `HEIGHT`.


Установка атрибута WIDTH в одной ячейке таблицы изменит ширину всех остальных ячеек столбца, а установка атрибута HEIGHT соответственно изменит высоту всех ячеек строки таблицы.

## Выравнивание текста в таблице

Выравнивание текста по высоте ячеек можно установить с помощью атрибута VALIGN. По умолчанию текст в ячейке таблицы выравнивается по центру. Атрибуту можно присваивать следующие значения:

- **top** — вверх;
- **middle** — по центру;
- **bottom** — вниз.

Для выравнивания текста по горизонтали используется атрибут ALIGN. Этому атрибуту присваиваются значения left, center и right точно так же, как при выравнивании текста абзаца (см. листинг 2.1 и рис. 2.4).


Атрибуты выравнивания текста можно устанавливать для строк и для отдельных ячеек, причем установки для ячеек таблицы отменяют установки для строк.

Атрибут ALIGN в дескрипторе <TABLE> используется для управления обтеканием текста страницы вокруг таблицы. Этот прием удобно применять для создания с помощью таблиц панелей элементов управления и текстовых выносок. Атрибуту ALIGN присваивают следующие значения:

- **left** — таблица выравнивается по левому краю окна с обтеканием текста справа от таблицы;
- **right** — таблица выравнивается по правому краю окна с обтеканием текста слева от таблицы;
- **all** — таблица выравнивается по центру окна с обтеканием текста справа и слева от таблицы (работает не во всех обозревателях);
- **center** — таблица выравнивается по центру окна, текст размещается над и под таблицей.

На рис. 2.13 показан пример создания панели ссылок и текстовой выноски в документе HTML. Код страницы показан в листинге 2.6.

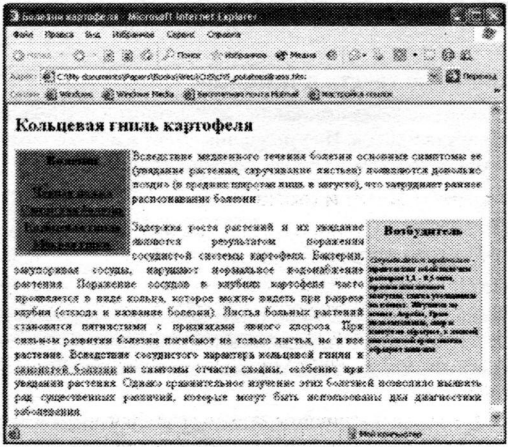



***Рис. 2.13.*** *Установка параметров обтекания текста вокруг таблицы для создания панели ссылок и текстовой выноски*

**Листинг 2.6. Создание панели ссылок с помощью таблицы**

```
<HTML>
<HEAD>
  <TITLE>Болезни картофеля</TITLE>
</HEAD>
<BODY>
<H2>Кольцевая гниль картофеля</H2>
<!- создаем таблицу для панели ссылок -->
<TABLE WIDTH='150' BORDER='2' FRAME='border' RULES='none'
BGCOLOR='gray' ALIGN='left'>
<TR><TH><H3>Болезни</H3></TH></TR>
<TR><TH><A HREF='?'>Черная ножка</A></TH></TR>
<TR><TH><A HREF='?'>Слизистая болезнь</A></TH></TR>
<TR><TH><A HREF='?'>Кольцевая гниль</A></TH></TR>
<TR><TH><A HREF='?'>Мокрая гниль</A></TH></TR>
</TABLE>
<P ALIGN='justify'>Вследствие медленного течения болезни …</P>
```

```
<!- создаем таблицу для выноски -->
<TABLE WIDTH='150' BORDER='1' FRAME='border' RULES='none'
BGCOLOR='yellow' ALIGN='right'>
<TR><TH><H3>Возбудитель</H3></TH></TR>
<TR><TD><H6><I>Corynebacterium sepedonicum</I> - представляет
собой палочки …</H6></TD></TR>
</TABLE>
<P ALIGN='justify'>Задержка роста растений и их увядание …</P>
</BODY>
</HTML>
```

## Добавление границ и фона

Толщина границ таблицы и ячеек устанавливается в пикселях с помощью атрибута BORDER в дескрипторе <TABLE>. Чтобы скрыть границы, атрибуту BORDER присваивается значение 0. Для выборочной прорисовки границ таблицы используются атрибуты FRAME и RULES. Выбор границ для показа производится путем присвоения этим атрибутам определенных стандартных значений, перечисленных в табл. 2.4.

**Таблица 2.4. Значения атрибутов FRAME и RULES для прорисовки внешних и внутренних границ таблицы**

| Значение | Добавляемые границы |
|---|---|
| **Внешние границы, устанавливаемые атрибутом FRAME** | |
| void | Нет внешних границ |
| above | Граница по верхнему краю таблицы |
| below | Граница по нижнему краю таблицы |
| rhs | Граница по правому краю таблицы |
| lhs | Граница по левому краю таблицы |
| hsides | Границы по верхнему и нижнему краям таблицы |
| vsides | Границы по левому и правому краям таблицы |
| border | Все границы (задано по умолчанию) |
| **Внутренние границы, устанавливаемые атрибутом RULES** | |
| none | Нет внутренних границ |
| cols | Границы между столбцами |
| rows | Границы между строками |
| all | Все внутренние границы (задано по умолчанию) |

Атрибут BGCOLOR в дескрипторе <TABLE> устанавливает цвет фона таблицы (см. табл. 2.3), а в атрибуте BACKGROUND можно указать путь

к фоновому рисунку. Оба атрибута можно установить для отдельных строк в дескрипторе <TR> или для отдельных ячеек в дескрипторах <TD> и <TH>.


Результат установки фонового рисунка для всей таблицы не вполне предсказуем. В одних обозревателях фоновый рисунок отобразится без учета границ между ячейками, тогда как в других обозревателях рисунок повторится в каждой ячейке.

В листинге 2.7 показан пример форматирования таблицы.

**Листинг 2.7. Форматирование таблицы**

```
<TABLE FRAME='hsides' RULES='rows' BORDER=1>
<CAPTION>Пример форматирования таблицы</CAPTION>
<TR BGCOLOR='yellow'><TH>Фамилия</TH><TH>Имя</
TH><TH>Отчество</TH><TH>Год рождения</TH></TR>
<TR><TD>Скворцова</TD><TD>Валентина</TD><TD>Васильевна</
TD><TD>1973</TD></TR>
<TR><TD>Баранов</TD><TD>Сергей</TD><TD>Анатольевич</
TD><TD>1969</TD></TR>
<TR><TD>Миронов</TD><TD>Тимофей</TD><TD>Максимович</
TD><TD>1980</TD></TR>
</TABLE>
```

Результат форматирования показан на рис. 2.14.

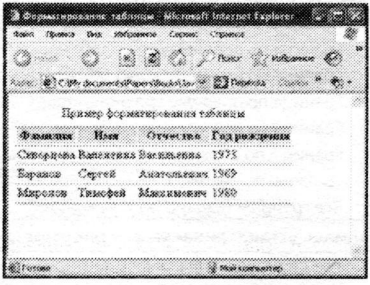



***Рис. 2.14.*** *Выборочная прорисовка границ и установка фона для строки заголовков таблицы*

## Объединение ячеек

Иногда бывает необходимо создать в таблице ячейку такого размера, что она будет занимать несколько столбцов или строк. Это особенно важно при выполнении разметки макета страницы с помощью таблицы. Для объединения ячеек в дескрипторах <TD> и <TH> устанавливаются следующие параметры:

- **COLSPAN** — число столбцов, занятых ячейкой;
- **ROWSPAN** — число строк, занятых ячейкой.


Часто при объединении ячеек разработчики допускают ошибку, в результате которой одна из строк таблицы оказывается длиннее остальных. Следует помнить, что при объединении *n* строк с помощью атрибута COLSPAN в текущей строке остается на *n*-1 ячеек меньше, а при использовании атрибута ROWSPAN число ячеек сокращается на 1 в *n*-1 строках, расположенных ниже текущей.

Пример использования объединения ячеек показан в листинге 2.8, а внешний вид полученной таблицы — на рис. 2.15.

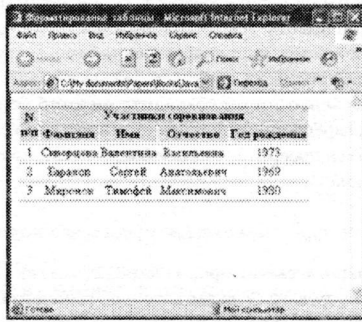



***Рис. 2.15.*** *Объединение ячеек для создания подзаголовков внутри таблицы*

**Листинг 2.8. Объединение ячеек**

```
<TABLE FRAME=hsides RULES=rows BORDER=1>
<TR BGCOLOR='yellow'><TH ROWSPAN=2>N<BR>п&#47п</TH><TH
COLSPAN=4>Участники соревнования</TH></TR>
```

```
<TR BGCOLOR='yellow'><TH>Фамилия</TH><TH>Имя</
TH><TH>Отчество</TH><TH>Год рождения</TH></TR>
<TR ALIGN='center'><TD>1</TD><TD>Скворцова</
TD><TD>Валентина</TD><TD>Васильевна</TD><TD>1973</TD></TR>
<TR ALIGN='center'><TD>2</TD><TD>Баранов</TD><TD>Сергей</
TD><TD>Анатольевич</TD><TD>1969</TD></TR>
<TR ALIGN='center'><TD>3</TD><TD>Миронов</TD><TD>Тимофей</
TD><TD>Максимович</TD><TD>1980</TD></TR>
</TABLE>
```

## Преимущества и недостатки использования таблиц

Преимущество использования таблиц для разметки Web-страниц состоит в том, что таблица является стандартным базовым элементом документа HTML, поэтому одинаково хорошо отображается во всех обозревателях даже самых старых версий. В то же время разметка документов требует тщательного планирования размера и структуры таблицы. Последующее редактирование больших таблиц на Web-страницах может быть затруднено, поскольку элементы таблицы жестко взаимосвязаны друг с другом. Так, изменение значений атрибутов `WIDTH` и `HEIGHT` в одной ячейке влечет за собой автоматическое изменение ширины или высоты всего столбца или строки.

Еще один недостаток использования таблиц состоит в том, что невозможно точно соотнести ячейки таблицы с областями окна обозревателя. Например, если вы создадите нижний колонтитул, то в зависимости от разрешения монитора пользователя ваш колонтитул может как оказаться посередине окна обозревателя, так и скрыться за нижним краем окна.

## Рамки

С помощью пары дескрипторов `<FRAMESET>...</FRAMESET>` можно разделить окно обозревателя на две рамки по горизонтали или по вертикали, причем в каждой рамке будет отображаться своя Web-страница, независимо от содержимого другой рамки. Технологию рамок удобно использовать для разметки окна обозревателя, создания панелей управления и колонтитулов. Прокручивание содержимого или смена Web-страницы в основной рамке не повлияет на внешний вид и положение в окне обозревателя вспомогательных рамок с гиперссылками и средствами навигации по Web-узлу.

## Создание структуры рамок

Ориентация рамок, а также их размер задаются атрибутами ROWS и COLS. В одном дескрипторе <FRAMESET> может быть установлен только один из них — ROWS или COLS. В первом случае окно делится по горизонтали, а во втором — по вертикали. Значения атрибутов задают высоту верхней и нижней рамок или ширину левой и правой рамок в процентах от размера окна либо в пикселях. Обычно указывается размер только одной рамки, а вместо размера другой рамки ставится символ звездочки (*). Это означает, что размер рамки будет подобран таким образом, чтобы заполнить всю оставшуюся часть окна.


Чтобы разделить одну из рамок на две дополнительные рамки, используется структура вложенных наборов рамок, как показано в листинге 2.9.

**Листинг 2.9. Набор рамок**

```
<HTML>
<HEAD>
<TITLE>Моя страница</TITLE>
</HEAD>
<FRAMESET ROWS='110,*'>
  <FRAME NAME='колонтитул' SRC='banner.htm'>
  <FRAMESET COLS='30%,*'>
    <FRAME NAME='оглавление' SRC='contents.htm'>
    <FRAME NAME='главная' SRC='MyPage.htm'>
  </FRAMESET>
</FRAMESET>
</HTML>
```

1. Введите код в текстовом редакторе Блокнот и сохраните в файле frame.htm.
2. Создайте и сохраните в этой же папке на диске файлы banner.htm, contents.htm и MyPage.htm, добавив в них абзацы <P>Колонтитул</P>, <P>Оглавление</P> и <P>Главная</P>.
3. Щелкните дважды на файле frames.htm, чтобы открыть его в окне обозревателя, как показано на рис. 2.16.

Имена рамок и файлы источников данных задаются с помощью непарного дескриптора <FRAME> и его атрибутов NAME и SRC (см. листинг 2.9). Имя в атрибуте NAME используется для навигации по рамкам с помощью гиперссылок (см. следующий подраздел).

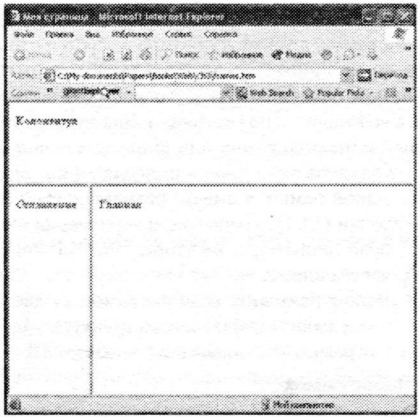


***Рис. 2.16.** Набор рамок*


В значении атрибута `SRC` можно использовать имя файла источника только в том случае, если файл документа находится в одной папке с файлом набора рамок. Если файл источника сохранен в другой папке на диске, введите абсолютный путь к файлу: `'C:\\MyWeb\Folder\MyPage.htm'`, или относительный путь от текущей папки, например: `'MyFolder/MyPage.htm'`. В атрибуте `SRC` можно также указать URL-адрес ресурса в Интернет.

## Навигация по рамкам с помощью гиперссылок

Для управления содержимым рамок используются гиперссылки. Например, щелчком на гиперссылке в одной рамке можно сменить документ в другой рамке или открыть новый документ в окне обозревателя, который заменит собой текущую структуру рамок. Навигация по рамкам осуществляется с помощью атрибута `TARGET` в дескрипторе гиперссылки, с которым вы уже познакомились ранее (см. раздел «Гиперссылки»). Например, следующая гиперссылка:

```
<A HREF='article2.htm' TARGET='main'>Статья 2</A>
```

открывает документ `article2.htm` в рамке с именем `main`. Причем сама гиперссылка может находиться в документе HTML, отображаемом в любой другой рамке окна обозревателя.

## Атрибут STYLE

Атрибут STYLE отличается от всех других атрибутов HTML. С помощью этого атрибута можно устанавливать одновременно несколько свойств объекта для его форматирования и позиционирования на странице.


Данный атрибут появился относительно недавно, поэтому не поддерживается некоторыми устаревшими обозревателями.

### Форматирование элементов страницы с помощью атрибута STYLE

С помощью атрибута STYLE можно отформатировать текст, принадлежащий блочным дескрипторам, перечисленным в табл. 2.5.

**Таблица 2.5. Блочные дескрипторы, определяющие фрагменты текста**

| Дескриптор | Описание |
|---|---|
| <BLOCKQUOTE>… </BLOCKQUOTE> | Врезка текста с увеличенными размерами полей |
| <CODE>…</CODE> | Выделяет текст, применяет шрифт Courier New |
| <DEL>…</DEL> | Выделяет текст перечеркиванием |
| <DIV>…</DIV> | Выделяет фрагмент текста, содержащий другие блочные дескрипторы |
| <EM>…</EM> | Выделяет текст курсивом |
| <H1>…</H1> – <H6>…</H6> | Заголовки текста |
| <INS>…</INS> | Выделяет текст подчеркиванием |
| <KBD>…</KBD> | Выделяет текст, применяя шрифт Courier New |
| <P>…</P> | Текст абзаца |
| <Q>…</Q> | Выделяет цитату, заключая текст в парные кавычки |
| <SAMP>…</SAMP> | Выделяет текст, применяя шрифт Courier New |
| <SPAN>…</SPAN> | Выделяет фрагмент текста внутри другого блочного дескриптора |
| <STRONG>… </STRONG> | Выделяет текст полужирным шрифтом |
| <TT>…</TT> | Выделяет текст, применяя шрифт Courier New |


Блочные дескрипторы, перечисленные в табл. 2.5, используются для создания как визуальных, так и логических фрагментов текста. Стандартное форматирование текста выполняется далеко не во всех обозревателях. Но с помощью атрибута `STYLE` можно отформатировать требуемым образом текст, заключенный между парой соответствующих дескрипторов.

Для форматирования текста атрибуту `STYLE` присваивается список параметров с определенными значениями, как в следующем примере:

```
<P STYLE='font-style: italic; background: #FFD700'>Текст
этого абзаца будет показан курсивом на золотом фоне</P>
```

Список параметров атрибута `STYLE` заключается в одинарные или двойные кавычки. Названия параметров отделяются от значений двоеточиями, а параметры в списке отделяются символами точки с запятой. Число параметров в списке не ограничено. В данном примере текст выбранного абзаца будет показан в окне обозревателя курсивом на золотом фоне (рис. 2.17).

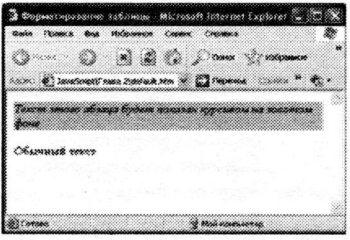



***Рис. 2.17.*** *Пример форматирования текста абзаца с помощью атрибута* `STYLE`

С атрибутом `STYLE` для форматирования текста можно использовать следующие параметры:

- **`background`** — цвет фона фрагмента текста;
- **`color`** — цвет текста;
- **`text-align`** — выравнивание текста на странице:
  - `center` — по центру;

- justify — по ширине страницы;
- left — по левому краю (по умолчанию);
- right — по правому краю;

◆ **text-decoration** — прочерчивание линий:

- none — нет линий (по умолчанию);
- underline — подчеркивание;
- line-through — перечеркивание;
- overline — линия над текстом;

◆ **font-stretch** — интервал:

- normal — обычный шрифт;
- condensed — сжатый;
- expanded — разреженный;

◆ **font-style** — стиль текста:

- normal — обычный текст;
- italic и oblique — курсив;

◆ **font-weight** — толщина букв:

- 100-900 — числовые значения толщины букв;
- normal — обычный шрифт, соответствует значению 400;
- bold — полужирный шрифт, соответствует значению 700;

◆ **line-height** — междустрочный интервал, задается числом с плавающей запятой: 1.0 — одинарный; 2.0 — двойной и т.д.

## Позиционирование элементов Web-страницы

Атрибут STYLE можно использовать для позиционирования элементов Web-страницы. Рассмотрим пример на рис. 2.18.

Web-страница содержит текст и рисунки, размещенные в определенном порядке. Некоторые рисунки перекрывают друг друга. Такую разметку страницы невозможно выполнить с использованием таблиц и рамок. Посмотрите, как эта задача выполняется с помощью параметров атрибута STYLE (листинг 2.10).

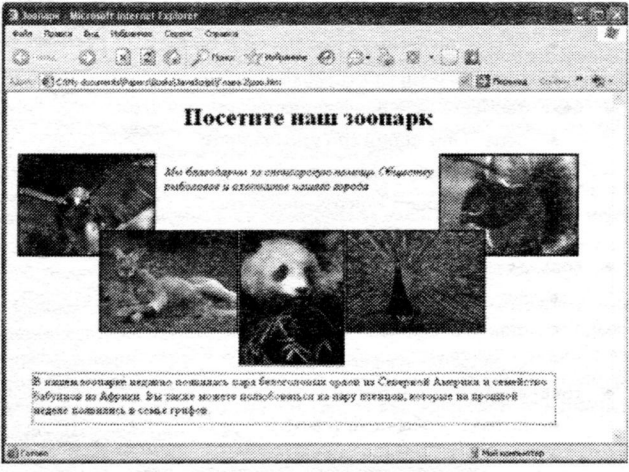



***Рис. 2.18.*** *Позиционирование элементов Web-страницы с помощью атрибута* `STYLE`

**Листинг 2.10. Позиционирование элементов страницы с помощью атрибута *STYLE***

```
<HTML>
<HEAD>
  <TITLE>Зоопарк</TITLE>
</HEAD>
<BODY>
  <H1 STYLE='text-align: center;
   font-weight: bold'>Посетите наш зоопарк</H1>
  <P STYLE='font-style: italic; position: absolute; top: 97;
   left:208; width:358; height:72'>Мы благодарны за
   спонсорскую помощь Обществу охотников и рыболовов
   нашего города</P>
  <IMG SRC='images\ANML01.jpg' STYLE='position: absolute;
   top: 83; left:15; z-index: 1; border: thin solid black'>
  <IMG SRC='images\ANML02.jpg' STYLE='position: absolute;
   top: 83; left:564; z-index: 2; border: thin solid black'>
```

```
  <IMG SRC='images\ANML03.jpg' STYLE='position: absolute;
   top: 185; left:305; z-index: 3; border: thin solid black'
   width="133" height="177"> 
  <IMG SRC='images\ANML05.jpg' STYLE='position: absolute;
   top: 185; left:443; z-index: 4; border: thin solid black'>
  <IMG SRC='images\ANML06.jpg' STYLE='position: absolute;
   top: 185; left:123; z-index: 5; border: thin solid black'>
  <P STYLE='border: thin dotted red; position: absolute;
   top: 376; left:34; width:682; height:70'>В нашем зоопарке
   недавно появилась пара белоголовых орлов из Северной
   Америки и семейство бабуинов из Африки. Вы также можете
   полюбоваться на пару птенцов, которые на прошлой неделе
   появились в семье грифов.</P>
</BODY>
</HTML>
```

В листинге 2.10 атрибут `STYLE` используется как для форматирования текста, так и для позиционирования элементов страницы. Например, заголовок «Посетите наш зоопарк» выравнивается по центру страницы и выделяется полужирным шрифтом с помощью параметров `text-align: center` и `font-weight: bold`. В следующем абзаце текст выводится курсивом с помощью параметра `font-style: italic`.

Далее мы размещаем рисунки и блоки текста на странице требуемым образом с помощью параметров позиционирования. Прежде всего задается тип позиционирования — `position: absolute`. В нашем примере установлено абсолютное позиционирование. Альтернативным вариантом является относительное позиционирование элементов Web-страницы: `position: relative`. Разница между этими типами позиционирования состоит в том, что относительный сдвиг элемента не влияет на положение других элементов Web-страницы, тогда как при абсолютном сдвиге образовавшееся пустое место заполняется текстом или другими элементами.

Сдвиг элемента в определенном направлении относительно исходной позиции устанавливается в пикселях с помощью параметров `top` (вверх), `bottom` (вниз), `right` (вправо) и `left` (влево). При этом элементы Web-страницы могут перекрывать друг друга. Чтобы указать очередность следования элементов от нижнего к верхнему, используется параметр `z-index`. Размеры элемента (рисунка или рамки текстового фрагмента) устанавливаются в пикселях с помощью параметров `width` (ширина) и `height` (высота). Чтобы лучше выделить элемент Web-страницы в окне обозревателя, прочертите рамку вокруг

него. Отображение рамки устанавливается параметром `border`, за которым в произвольном порядке следуют три значения, определяющие вид рамки:

- **толщина рамки**:
  - `thin` — тонкая;
  - `medium` — средняя;
  - `thick` — толстая;
- **стиль рамки**:
  - `solid` — сплошная;
  - `double` — двойная;
  - `dotted` — пунктирная;
  - `dashed` — штрих-пунктирная;
  - `groove` — вдавленная;
  - `inset` и `outset` — объемные;
- **цвет** устанавливается по имени или шестнадцатеричным значением (см. табл. 2.3).

Например, в листинге 2.10 вокруг рисунков прочерчены тонкие сплошные черные рамки (`border: thin solid black`), а последний абзац на странице очерчен тонкой пунктирной красной рамкой (`border: thin dotted red`).

## Добавление сценариев в код HTML

Сценариями называются небольшие программы, предназначенные для выполнения определенных операций и функций. Сценарии используются в том случае, если стандартный набор функций обозревателей оказывается недостаточным для правильного отображения Web-страницы. Например, сценарии необходимы при использовании на странице форм с элементами управления. Обозреватели отображают элементы управления на экране, но пользу они принесут только в том случае, если разработчик снабдит их соответствующими функциями.

Сценарии добавляются непосредственно в код HTML. Обозреватели способны различать программные коды сценариев и выполнять их. Стандартным языком сценариев, поддерживаемым большинством обозревателей, является JavaScript.


Часто сценарии требуются для выполнения рутинных задач, с которыми до вас сталкивались другие разработчики Web-страниц. Чтобы не изобретать велосипед снова и снова, посетите Web-узлы `http://javascript.internet.com/`, `www.javascript.com` и `www.javascripts-galore.com`.

Здесь вы найдете большую коллекцию готовых кодов сценариев на JavaScript для решения многих задач. Вам останется только скопировать код сценария и правильно вставить его в код HTML.

## Добавление сценариев JavaScript

Для добавления сценария в код HTML используется пара дескрипторов `<SCRIPT>...</SCRIPT>`. Текст, заключенный между этими дескрипторами, рассматривается обозревателем как код программы и не отображается на экране.

Сценарии можно добавлять как в основной раздел Web-страницы (`<BODY>...</BODY>`), так и в раздел заголовка (`<HEAD>...</HEAD>`). Важно правильно выбрать место сценария в коде HTML, поскольку строки кода выполняются обозревателем по мере загрузки Web-страницы. В коде сценария не должно быть обращений к элементам Web-страницы, которые загружаются позже (т.е. расположены в коде HTML после кода сценария). Сценарии в разделе заголовка обычно используются для определения основных функций, констант и переменных, которые затем могут быть использованы в сценариях основного раздела.

Код сценария может храниться в отдельном текстовом файле. (Обычно такие файлы сохраняют с расширением `.js`.) Чтобы воспользоваться этим кодом, путь к файлу указывают в атрибуте `SRC` дескриптора `<SCRIPT>`:

```
<SCRIPT SRC='scripts/MyScript.js' LANGUAGE='JavaScript'></
SCRIPT>
```

В примере установлен еще один атрибут — `LANGUAGE`. Для написания сценариев могут использоваться некоторые другие языки программирования, например VBScript. Обозреватель может самостоятельно определить язык сценария, но это увеличит время выгрузки страницы. Поэтому язык сценария лучше явно указать в атрибуте `LANGUAGE`.

## Сокрытие сценариев

Устаревшие обозреватели, не поддерживающие дескриптор <SCRIPT>, рассматривают код сценария как обычный текст и отображают его на экране, что портит Web-страницу, и без того утратившую часть своей функциональности. Есть очень простой способ предупредить показ кода сценария в окнах всех обозревателей — поместите код сценария между командными символами комментариев, как в следующем примере:

```
<SCRIPT>
<!--
    код сценария
//-->
</SCRIPT>
```


Перед закрывающим символом комментария «-->» находятся два символа обратного слеша «//», которые необходимы для того, чтобы обозреватель не воспринял командные символы как часть кода сценария.


Если код сценария сохранен в отдельном файле и добавлен с помощью атрибута SRC, то в этом случае код никогда не отобразится в окне обозревателя.

## Добавление альтернативного текста

Еще одна проблема состоит в том, что страницы со сценариями утрачивают в устаревших обозревателях часть своей функциональности и могут работать неправильно. Следует предупредить пользователей о возможных неполадках в работе Web-страницы. Для этого используется альтернативный текст, добавляемый на страницу с помощью дескрипторов <NOSCRIPT>...</NOSCRIPT>. На странице может быть несколько сценариев, но альтернативный текст достаточно добавить один раз в любом месте основного раздела страницы.


Альтернативный текст сценариев просматривается и индексируется поисковыми роботами. Это дает возможность использовать его для описания сценариев, что позволит успешно вести поиск сценариев в Интернет с помощью поисковых роботов.


Следует помнить, что альтернативный текст будет показан только в том случае, если обозреватель не поддерживает выполнение сценариев. Когда ошибка возникает в ходе выполнения сценария, альтернативный текст не отображается.

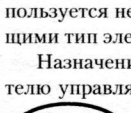
## Создание форм

На Web-страницы можно добавлять элементы управления, которые обычно используются в диалоговых окнах: командные кнопки, флажки, переключатели, текстовые поля, списки и пр. Для добавления этих элементов управления сначала нужно создать форму, которой они будут принадлежать.


На Web-странице может использоваться несколько форм с различными наборами элементов управления. Может быть создана пустая форма, элементы управления в которую будут добавляться динамически во время использования Web-страницы. Но на странице нельзя применять элементы управления без формы.

Форма создается с помощью пары дескрипторов `<FORM>...</FORM>`. Все элементы управления формы находятся в коде HTML между этими дескрипторами. Для создания элемента управления чаще всего используется непарный дескриптор `<INPUT>` с атрибутами, определяющими тип элемента управления.

Назначение форм состоит в том, чтобы дать возможность посетителю управлять содержимым Web-страницы, вызывать на выполнение *сценарии*, а также вводить собственные данные и отправлять их на Web-сервер или по адресу электронной почты.




Создадим пример формы, введя код листинга 2.11.

**Листинг 2.11. Форма**

```
<FORM>
  <!- Поле ввода -->
  <BR>Фамилия:
  <BR><INPUT TYPE='text' VALUE='Иванов'><BR>

  <!- Группа переключателей -->
  <BR><INPUT TYPE='radio' NAME='color'>Белый
  <BR><INPUT TYPE='radio' NAME='color'>Красный
```

```
<BR><INPUT TYPE='radio' NAME='color'>Желтый
<BR><INPUT TYPE='radio' NAME='color'>Зеленый<BR>

<!- Группа флажков -->
<BR><INPUT TYPE='checkbox' NAME='fruit'>Груша
<BR><INPUT TYPE='checkbox' NAME='fruit'>Персик
<BR><INPUT TYPE='checkbox' NAME='fruit'>Яблоко<BR>

<!- Кнопки -->
<BR><INPUT TYPE='button' VALUE='Пишите мне'
 ONCLICK=window.open('mailto:MyAddress@mail.ru');>
<BR><INPUT TYPE='button' VALUE='Посмотрите мои фотографии'
 ONCLICK=window.open('http:\\\\www.MySite.org\\MyFoto.htm');>
</FORM>
```

На рис. 2.19 показано, как созданные элементы управления будут выглядеть в окне обозревателя.

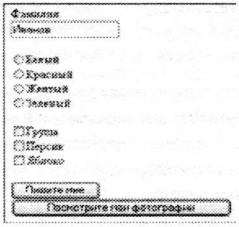



***Рис. 2.19.*** *Элементы управления формы*

Тип элемента управления устанавливается с помощью атрибута `TYPE`. Давайте рассмотрим по порядку свойства и использование элементов управления формы, показанных на рис. 2.19.

## Поле ввода

Поле ввода `<INPUT TYPE='text' VALUE='Иванов'>` позволяет пользователю ввести с клавиатуры одну строку текста. Строка, введенная в поле, автоматически присваивается атрибуту `VALUE` этого элемента. И наоборот, строка текста, назначенная атрибуту `VALUE`

в коде HTML, отобразится в этом поле в окне обозревателя по умолчанию (см. рис. 2.19).

## Переключатели и флажки

Переключатели и флажки используются для того, чтобы предоставить посетителям Web-страницы возможность сделать свой выбор, например: выбрать цвет фона страницы или выбрать для просмотра одну из нескольких статей или фотографий. Разница между флажками и переключателями состоит в том, что в группе переключателей можно выбрать только один из них. Флажки, принадлежащие одной группе, устанавливаются независимо друг от друга.

Чтобы объединить переключатели или флажки в группу, назначьте атрибутам `NAME` этих элементов одно и то же значение. Например, в листинге 2.11 всем переключателям присвоено имя `color`, а всем флажкам — имя `fruit`.

Подпись переключателя или флажка добавляется сразу после закрывающей скобки дескриптора `<INPUT>`:

```
<INPUT type='checkbox' name='name'>Подпись флажка
```

## Кнопка

Кнопки `<INPUT TYPE='button'>` используются для вызова сценариев, присвоенных атрибуту кнопки `ONCLICK`. На кнопке в окне обозревателя отображается надпись, присвоенная атрибуту `VALUE`. Вкратце рассмотрим пример сценариев в листинге 2.11, где создаются две кнопки. В обоих случаях атрибуту `ONCLICK` назначается функция `window.open()`, которая используется для открытия нового окна обозревателя. В первой кнопке в качестве аргумента функции вводится строка `'mailto:MyAddress@mail.ru'`:

```
<INPUT TYPE='button' VALUE='Пишите мне' ONCLICK=window.open
('mailto:MyAddress@mail.ru');>
```

Как и в адресе гиперссылки, команда `mailto` запускает приложение клиента электронной почты, открывает окно нового сообщения и вводит адрес абонента, указанный после двоеточия.

Во второй кнопке в аргументе функции `open.window` указывается URL-адрес Web-страницы, которая отобразится в новом окне обозревателя после щелчка на кнопке **Посмотрите мои фотографии**:

```
<INPUT type='button' value='Посмотрите мои фотографии'
onClick=window.open('http:\\\\www.MySite.org\\MyFoto.htm');>
```


Обратите внимание на то, что URL-адрес Web-страницы в аргументе команды `open.window` выглядит не совсем обычно — в нем вдвое больше слешей. Дело в том, что в языке JavaScript слеш (`\`) является управляющим символом, который указывает, что следующий за ним символ является текстом, а не командой — дополнительные слеши как раз и используются для отмены управляющих символов. Поэтому в текстовом аргументе функции перед каждым слешем, необходимым в адресе URL, устанавливается дополнительный слеш, указывающий, что следующий за ним символ является текстом.

# Глава 3

# Использование сценариев в динамических Web-страницах

Идея создания мировой сети состояла в том, чтобы предоставить возможность свободного обмена информацией между пользователями во всем мире. Достаточно подготовить документ в формате HTML и опубликовать его на Web-сервере, чтобы любой человек в мире смог открыть и прочесть ваш файл. Быть автором публикаций в Интернет теперь так же просто, как и быть пользователем Интернет.

Но статическое представление данных в Интернет быстро исчерпало себя. Живой диалог с посетителями Web-страниц открывает значительно больше возможностей перед автором для раскрытия темы, рекламы товаров и предоставления именно тех услуг, которые ждет пользователь. Идея динамически изменяемых Web-страниц, внешний вид и содержимое которых может настраиваться самим посетителем, была реализована за счет использования сценариев — программ, загружаемых вместе с Web-страницей и выполняемых при определенных условиях. В этой главе вы познакомитесь с основными концепциями создания сценариев и назначения их определенным событиям.

## Общие сведения о событиях и функциях обработки событий

Интерактивность Web-страниц состоит в том, что в ответ на какое-либо действие пользователя происходит изменение внешнего вида страницы, отображается дополнительная информация или выполняются другие процессы, облегчающие пользователю работу с документом. Действия пользователей, в ответ на которые происходит

изменение страницы, называются *событиями*. Чтобы назначить событию вызов сценария на выполнение, используются атрибуты событий элементов Web-страницы. В главе 2, в листинге 2.11, мы уже назначали сценарии событиям щелчка на кнопке формы. Для установки такого события используется атрибут ONCLICK, как в следующем примере:

```
<INPUT TYPE='button' VALUE='Посмотрите мои фотографии'
ONCLICK=window.open('http:\\\\www.MySite.org\\MyFoto.htm');>
```

В данном случае в форме создается кнопка **Посмотрите мои фотографии**, после щелчка на которой выполняется стандартная функция JavaScript `window.open()`. В аргументе этой функции указан URL-адрес Web-страницы, которая будет открыта в новом окне обозревателя после того, как пользователь щелкнет на кнопке **Посмотрите мои фотографии**.

В табл. 3.1 представлены некоторые атрибуты событий, которым можно назначать выполнение сценариев.

**Таблица 3.1. События, поддерживаемые элементами Web-страницы**

| Событие | Описание | Дескрипторы, поддерживающие событие |
|---|---|---|
| ONCLICK | Щелчок мышью на элементе | Большинство дескрипторов |
| ONDBCLICK | Двойной щелчок мышью на элементе | Большинство дескрипторов |
| ONMOUSEOVER | Пользователь наводит указатель мыши на элемент | Большинство дескрипторов |
| ONMOUSEOUT | Пользователь убирает указатель мыши с элемента | Большинство дескрипторов |
| ONFOCUS | Пользователь выбирает элемент с помощью клавиши <Tab> | <A>, <AREA>, <LABLE>, <INPUT>, <SELECT>, <BUTTON>, <TEXTAREA> |
| ONBLUR | Элемент теряет выделение при следующем нажатии клавиши <Tab> | <A>, <AREA>, <LABLE>, <INPUT>, <SELECT>, <BUTTON>, <TEXTAREA> |
| ONSELECT | Пользователь выделяет текст элемента | <INPUT>, <TEXTAREA> |
| ONCHANGE | Пользователь изменяет текст элемента | <INPUT>, <SELECT>, <TEXTAREA> |
| ONSUBMIT | Пользователь щелкнул в форме на кнопке **Подача запроса** | <FORM> |

*Окончание табл. 3.1*

| Событие | Описание | Дескрипторы, поддерживающие событие |
|---|---|---|
| ONRESET | Пользователь щелкнул в форме на кнопке Сбросить | <FORM> |
| ONLOAD | Web-страница загружается в окно обозревателя или в рамку | <BODY>, <FRAMESET>, <FRAME> |
| ONUNLOAD | Web-страница замещается другой страницей в окне обозревателя или в рамке | <BODY>, <FRAMESET>, <FRAME> |

## Именование элементов Web-страницы

Для любого элемента Web-страницы можно задать уникальное имя, присвоив его атрибуту ID и/или NAME.


Если вы предполагаете обращаться к элементу Web-страницы в сценариях, установите одно и то же имя для обоих атрибутов — ID и NAME. Эти атрибуты частично дублируют друг друга, а их использование в разных обозревателях не в полной мере стандартизировано, поэтому лучше подстраховаться.

Например, в следующем примере мы присваиваем элементу основного раздела страницы имя body:

```
<BODY ID='body' NAME='body'>
```

Смысл именования элементов состоит в том, что разработчик получает возможность в сценариях обращаться к элементам Web-страницы по имени и динамически изменять их атрибуты. Например, после того как мы присвоили разделу <BODY> имя, в коде сценария можно установить или изменить цвет фона страницы, используя следующую команду:

```
window.body.bgColor='red'
```

Эту команду можно присвоить атрибуту события, например событию ONCLICK кнопки формы.


Web-страницу можно представить как контейнер, содержащий элементы страницы, которые, в свою очередь, являются контейнерами атрибутов. Иерархия контейнеров записывается слева направо, и имена контейнеров

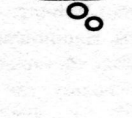
разделяются точками. Для обращения к объекту Web-страницы в сценарии используется ключевое слово `window`. Далее следуют имя элемента и имя атрибута. (См. также основы объектно-ориентированного программирования в главе 1.)

## Создание пользовательских функций

В сценариях часто бывает необходимо выполнить не одну, а много команд. В таком случае удобно создавать и применять пользовательские функции. Пользовательские функции создаются с помощью ключевого слова `function` следующим образом:

```
<SCRIPT LANGUAGE='JavaScript'>
function ИмяФункции(список аргументов) {
  код функции
  };
др. функции …
</SCRIPT>
```

Определения функций всегда отделяются от остального кода Web-страницы с помощью пары дескрипторов сценария <SCRIPT>…</SCRIPT>. Сценарий может содержать определения многих функций, а также выполняемые команды и стандартные функции языка JavaScript. В свою очередь Web-страница может содержать несколько сценариев в основном разделе и в разделе заголовка (см. главу 2).




Сценарии выполняются во время загрузки Web-страницы в той очередности, в которой они находятся в коде HTML. Но если команды и стандартные функции JavaScript в сценариях выполняются автоматически, то последовательности команд функций сохраняются в оперативной памяти компьютера под именем функции и выполняются только в том случае, если функция была вызвана на выполнение.

За ключевым словом `function` следует имя функции. Имя функции должно быть уникальным для данной Web-страницы. Если в текущем или следующем сценарии на Web-странице будет определена другая функция с таким же именем, то прежняя функция будет замещена новой.

Имя функции завершается парой скобок, между которыми находится список аргументов функции. Список аргументов может быть пустым «()», или содержать имена аргументов, используемых в дан-

ной функции: *ИмяФункции*(a, b, c, …). В соответствии с синтаксисом JavaScript код функции заключается в фигурные скобки {…}, а каждая строка кода должна завершаться символом точки с запятой «;».

После того как функция была определена, ее можно вызвать на выполнение в коде сценария следующим образом:

```
ИмяФункции(список аргументов);
```


Число аргументов в вызове функции должно соответствовать числу аргументов в определении функции.

Чаще всего функции используют для обработки событий, выполняемых над документом HTML или его элементами. Функция, назначенная атрибуту события, называется *функцией обработки события*. При использовании гиперссылок функции могут назначаться как атрибутам событий, так и непосредственно HREF. Но при этом используется особый синтаксис назначения функции с помощью ключевого слова javascript:

```
<A href='javascript:def_show()'>Текст гиперссылки</A>
```

В этом случае текст будет выглядеть как обычная гиперссылка, но функционировать как кнопка формы.


Если вы хотите, чтобы во время щелчка на гиперссылке одновременно с открытием новой страницы выполнялась какая-нибудь дополнительная функция, назначьте эту функцию событию ONCLICK данной гиперссылки.

Пользовательские функции можно вызывать непосредственно из кода сценария, как и стандартные функции JavaScript.


Вызов в коде сценария функции, определение которой находится ниже в коде HTML, приведет к ошибке.

Функции в сценариях используют для достижения следующих двух наиболее общих целей:

- для динамического добавления или удаления элементов Web-страницы, либо изменения их свойств;

- для выполнения вычислений и возвращения результата по месту вызова функции.

Для возвращения полученного значения в функции используется команда `return`. Например, ниже показана функция для вычисления суммы двух чисел:

```
function sum(a, b) {
  return (a + b);
}
```

В этой главе мы рассмотрим применение функций для динамического редактирования Web-страниц, а с вычислениями в сценариях ознакомимся в главе 4.

# Добавление и изменение текста Web-страницы с помощью сценариев

Поскольку основным содержимым Web-страниц обычно является текст, прежде всего рассмотрим примеры динамического ввода и редактирования текста Web-страниц с помощью сценариев JavaScript. Сначала познакомимся с общими принципами динамического редактирования текста, а затем рассмотрим несколько примеров, которые могут пригодиться на практике.

## Динамическое редактирование блочных элементов Web-страницы

С помощью функций можно также изменять или возвращать содержимое блочных элементов Web-страницы. Для этого используются атрибуты `innerText` и `innerHTML`. Чтобы понять смысл этих атрибутов, рассмотрим два следующих примера абзацев текста, которые на Web-странице будут выглядеть совершенно одинаково:

```
<P>Текст абзаца</P>
<P innerText='Текст абзаца'></P>
```

Таким образом, атрибут содержит весь текст блочного элемента, в данном случае абзаца. Атрибут `innerHTML` отличается тем, что содержит не только текст, но и другие дескрипторы HTML, такие, как дескрипторы форматирования и вложенные блочные элементы. Рассмотрим примеры динамического изменения текста Web-страницы.

## Ввод текста непосредственно на Web-странице

Обозреватели предназначены для просмотра, но не для редактирования Web-страниц. Как правило, посетитель Web-страницы может только читать текст, но не изменять его, за исключением текстовых полей форм, о которых мы поговорим подробнее в главе 7. Тем не менее Web-страницу можно снабдить сценариями, которые дадут возможность пользователю изменять текст документа. Эта возможность может пригодиться, например, при создании развлекательных игровых Web-страниц. Правда, и в этом случае не удастся обойтись без элементов управления форм.


Введите в текстовом редакторе код Web-страницы, показанный в листинге 3.1.

**Листинг 3.1. Ввод текста на Web-страницу**

```
<BODY>
<FORM>
<INPUT TYPE=text ID=txt_field NAME=txt_field SIZE=50><BR>
<INPUT TYPE=button VALUE='Ввести текст'
 ONCLICK="insert_text(txt_field.value)">
</FORM>
<P ID=par NAME=par></P>
<SCRIPT>
function insert_text(text) {
  par.innerText=text;
  }
</SCRIPT>
</BODY>
```

Web-страница содержит форму с текстовым полем `txt_field` и кнопкой **Ввести текст**. Событию `ONCLICK` этой кнопки назначено выполнение пользовательской функции `insert_text`, определенной ниже в разделе сценария. В функцию обработки события передается один аргумент — текущее значение текстового поля `txt_field.value`. Web-страница также содержит пустой именованный абзац: `<P ID=par NAME=par></P>`. Функция `insert_text` вставляет переданный ей текст в абзац: `par.innerText=text;`. Результат показан на рис. 3.1.

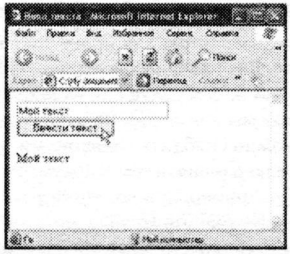
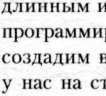



***Рис. 3.1.*** *Текст из поля был вставлен на Web-страницу щелчком на кнопке* ***Ввести текст***

## Всплывающий текст

Освоив принципы динамического ввода и редактирования текста Web-страницы, приступим к созданию всплывающих подсказок и определений. В тексте страницы могут встречаться термины или упоминаться события, требующие кратких пояснений. Многочисленные пояснения и определения сделают текст Web-страницы слишком длинным и отвлекут от основной идеи. Воспользуйтесь навыками программирования на JavaScript, чтобы решить эту проблему. Давайте создадим в тексте всплывающие определения. Предположим, что у нас на странице есть текст: «Чтобы добавить цифровую подпись к документам, передаваемым с вашего компьютера, щелкните *здесь* для загрузки и установки цифрового сертификата». Теперь добавим к этому тексту определение термина *цифровая подпись*.




Введите в текстовом редакторе код листинга 3.2.

**Листинг 3.2. Добавление всплывающего текста**

```
<BODY>
<P> Чтобы добавить <A HREF='javascript:def_show()'>цифровую
подпись</A> <SPAN ID='def_text' NAME='def_text'
STYLE='color: red' ONCLICK=def_move();></SPAN>  к
документам, передаваемым с вашего компьютера, щелкните здесь
для загрузки и установки цифрового сертификата.</P>

<SCRIPT>
function def_show() {
```

```
  def_text.innerText="(Цифровая подпись: шифрованная
электронная подпись, подтверждающая подлинность документа.
Наличие цифровой подписи подтверждает, что документ был
получен от владельца подписи и не был изменен.)";
};

function def_move(){
  def_text.innerText="";
};
</SCRIPT>
</BODY>
```

Сохраните документ в формате HTML и дважды щелкните на имени файла в окне **Мой компьютер**, чтобы открыть файл с помощью обозревателя. Текст отобразится в окне обозревателя, как показано на рис. 3.2. Щелкните на словах «*цифровую подпись*», отформатированных как гиперссылка. В текст абзаца будет добавлено определение термина «*цифровая подпись*», как показано на рис. 3.3. Текст определения взят в скобки и выделен красным цветом (цвет текста задан в дескрипторе `<SPAN>` с помощью атрибута `STYLE`). Щелкните на тексте определения, чтобы убрать его с экрана и восстановить текст в том виде, в каком он был показан на рис. 3.2.

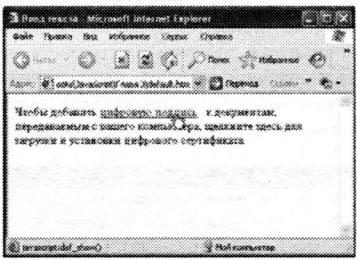



***Рис. 3.2.** Щелкните на выделенном термине, чтобы добавить его определение*

В коде листинга 3.2, в отличие от листинга 3.1, вместо пустого абзаца используется пустой фрагмент текста, заданный дескрипторами `<SPAN>...</SPAN>`. Эти дескрипторы используются для того, чтобы добавляемый текст появился внутри текущего абзаца.

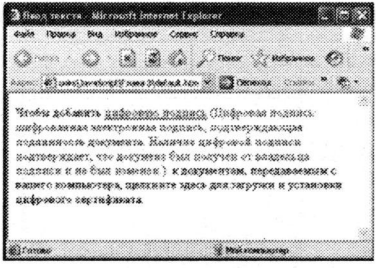


***Рис. 3.3.** После того как пользователь прочитает определение термина, он сможет удалить определение, щелкнув на его тексте*

Обратите внимание: такой же подход может быть применен как для добавления текста на Web-страницу, так и для удаления текста из нее. Щелчок на тексте определения вызывает функцию `def_move`, которая присваивает динамически изменяемому фрагменту текста пустую строку: `def_text.innerText=""`.

## Меню гиперссылок

Если у вас на странице много гиперссылок, то для экономии места на странице и облегчения поиска нужной ссылки можно сгруппировать их по категориям и добавить на страницу список категорий. Когда пользователь подводит указатель мыши к заголовку категории, открывается вложенный список гиперссылок, относящихся к этой категории, как показано на рис. 3.4. Если пользователь перемещает

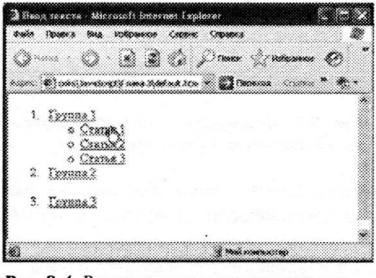


***Рис. 3.4.** Всплывающее меню гиперссылок*


указатель на другой заголовок категории, список ссылок предыдущей категории сворачивается, а под заголовком категории возникает новый список.

Введите в основной раздел страницы код листинга 3.3.

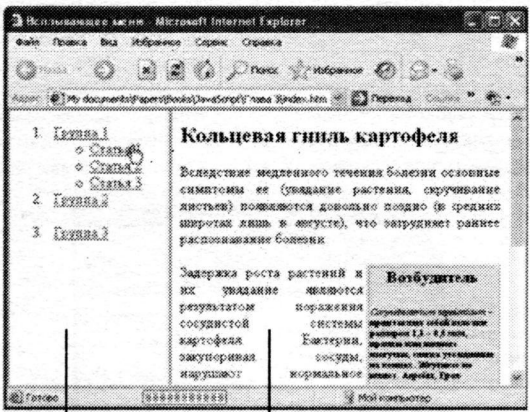
**Листинг 3.3. Всплывающие списки гиперссылок**

```
<FORM><DIV STYLE='color: blue; text-decoration: underline'>
  <P ONMOUSEOVER=f1()>Группа 1</P>
  <UL ID='group_1' NAME='group_1'></UL>
  <P ONMOUSEOVER=f2()>Группа 2</P>
  <UL ID='group_2' NAME='group_2'></UL>
  <P ONMOUSEOVER=f3()>Группа 3</P>
  <UL ID='group_3' NAME='group_3'></UL>
</DIV></FORM>

<SCRIPT>
function f1() {
  window.group_2.innerHTML=""
  window.group_3.innerHTML=""
  window.group_1.innerHTML="<LI ONMOUSEOVER=on_select
    ('art1.htm')>Статья 1</LI><LI ONMOUSEOVER=on_select
    ('art2.htm')>Статья 2</LI><LI ONMOUSEOVER=on_select
    ('art3.htm')>Статья 3</LI>"
};

function f2(){
  window.group_1.innerHTML=""
  window.group_3.innerHTML=""
  window.group_2.innerHTML="<LI ONMOUSEOVER=on_select
    ('art4.htm')>Статья 4</LI><LI onMouseOver=on_select
    ('art5.htm')>Статья 5</LI>"
};

function f3(){
  window.group_1.innerHTML=""
  window.group_2.innerHTML=""
  window.group_3.innerHTML="<LI ONMOUSEOVER=on_select
    ('art6.htm')>Статья 6</LI>"
};

function on_select(name){
```

```
  top.main.location.replace(name);
};
</SCRIPT>
```

Давайте шаг за шагом разберем предложенный код HTML. Сначала создаются заголовки меню верхнего уровня **Группа 1**—**Группа 3**, каждый в своем абзаце с установленными функциями обработки событий `ONMOUSEOVER`. За заголовками следуют объекты пустого маркированного списка с уникальным именем: `<UL ID='group_#' NAME=='group_1'></UL>`. Программа работает следующим образом.

1. Пользователь наводит указатель мыши на заголовок группы. При этом происходит событие `ONMOUSEOVER`, вызывающее на выполнение соответствующую функцию.
2. В первых строках функция обработки события обнуляет списки ссылок других категорий, присваивая пустые строки атрибутам `innerHTML` соответствующих маркированных списков.
3. В третьей строке функция создает список ссылок под выбранной категорией, присвоив HTML-код списка атрибуту `innerHTML` соответствующего маркированного списка.




Панель навигации Основная рамка *main*

***Рис. 3.5.*** *Использование всплывающего меню в панели навигации*

4. При наведении указателя мыши на пункт всплывающего меню в основной рамке окна обозревателя отображается текст новой Web-страницы.

Такое раскрывающееся меню удобно разместить в панели навигации, как показано на рис. 3.5.

В окне обозревателя пункты меню выглядят как гиперссылки, но в действительности они таковыми не являются. Цвет и подчеркивание заданы параметрами атрибута `STYLE` в блочном дескрипторе `<DIV>`. Активность пунктов меню обеспечивается функцией `on_select`, которая назначена соответствующим событиям `ONMOUSEOVER`. Эта функция открывает новую Web-страницу в рамке `main`. (Более подробно об открытии Web-страниц в окне обозревателя с помощью сценариев см. в главе 6.)

## Ввод и редактирование кода HTML с помощью сценариев

В главе 2 вы ознакомились с кодом HTML и узнали, как создавать Web-страницы в текстовом редакторе с чистого листа. В предыдущих разделах этой главы указывалось, что элементы Web-страницы, созданные с использованием дескрипторов HTML, могут быть изменены динамически уже после открытия Web-страницы с помощью сценариев. В этом разделе речь пойдет о динамическом создании кода HTML посредством сценариев, т.е. о создании новых Web-страниц по требованию пользователей.

### Функция write

Функция `write` языка JavaScript используется для динамического ввода текста и дескрипторов HTML в окно обозревателя. Синтаксис использования этой функции следующий:

```
document.write("текст")
```

Прежде всего нужно указать документ, в котором следует ввести новый текст. Ключевое слово `document` соответствует текущему окну обозревателя. Если текст вводится в определенную рамку или в другое окно обозревателя, то перед словом `document` необходимо указать имя рамки или окна:

```
main.document.write("текст")
NewWin.document.write("текст")
```

Аргументом функции может быть строка текста, число или вызов функции JavaScript (стандартной или пользовательской), возвращающей текстовое или числовое значение. Каждая последующая функция `write` вводит текст, вслед за текстом, добавленным предыдущей функцией. Чтобы создать новый абзац, используйте в строке текста дескрипторы HTML:

```
document.write("<P>Первый абзац</P>")
document.write("<P>Второй абзац</P>")
```

или одну команду:

```
document.write("<P>Первый абзац</P><P>Второй абзац</P>")
```

В строке функции `write` можно использовать любые дескрипторы как для форматирования текста, так и для создания таблиц, форм, списков и других элементов Web-страницы.

Функции `write` можно передать несколько аргументов, разделив их запятыми. Функция автоматически объединит строки текста, переданные в аргументах, и выведет суммарную строку в окне обозревателя. Это свойство удобно использовать для включения в текст результатов выполнения пользовательских функций, как в следующем примере:

```
WinCulc.document.write("<P>Сумма налога за автомобиль составит",
culc_tax(car_type.value,year.value)," рублей</P>")
```

Данная функция выводит строку с указанием суммы налога за автомобиль. Предположим, что на Web-странице есть форма с двумя полями **Тип автомобиля** и **Год выпуска**, которым присвоены имена `car_type` и `year` соответственно. Вам не составит труда создать такую форму самостоятельно. В Web-странице также есть сценарий с определением функции `culc_tax`, которая возвращает значение, рассчитанное по данным из полей формы. (Использование сценариев для вычислений рассматривается в главе 4.) Функция `write` выводит результат вычислений с сопровождающим текстом одним абзацем в новое окно обозревателя с именем `WinCulc`. (Подробно об открытии новых окон обозревателя из кода сценария рассказывается в главе 6.)

Очень важно правильно выбрать место функции `write` в сценарии на Web-странице, чтобы получить требуемый результат. Если функция `document.write` используется в сценарии, то строки текста будут добавляться за элементами Web-страницы, созданными ранее в коде HTML. Функцию `write` также можно применять внутри пользова-

тельских функций обработки событий. Обычно в этом случае функция `write` применяется для заполнения текстом определенных рамок или окон обозревателя.


При использовании `write` в функции обработки события следует учесть одну важную особенность обозревателя Netscape Navigator. Текст, добавленный в уже открытое окно обозревателя или рамку, появится только после перезагрузки содержимого Web-страницы. В коде JavaScript для перезагрузки содержимого окна или рамки используется команда *имя_окна/рамки*`.location.reload()`. Эта команда не помешает выполнению сценария в Internet Explorer.


Давайте рассмотрим использование функции `write` на примере листинга 3.4.

**Листинг 3.4. Ввод текста Web-страницы с помощью функции `write`**

```
<HTML>
<HEAD>
<TITLE>Функция write</TITLE>
<SCRIPT>
function input1() {
  document.write("<P>Ввод текста с помощью
   пользовательской функции.</P>");
}
function input2() {
  win2 = window.open("","output","width=400,height=200");
  win2.document.write("<P>Ввод текста в новое окно</P>");
  win2.location.reload();
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write("<P>Ввод текста из кода сценария.</P>")
input1()
document.write("<FORM><INPUT TYPE=button VALUE='Вывести текст
в новое окно' ONCLICK=input2()></FORM>")
</SCRIPT>
</BODY>
</HTML>
```

Заметьте, что раздел <BODY> содержит только код сценария, который создает все элементы Web-страницы. Первая строка сценария добавляет текст «Ввод текста из кода сценария». Затем следует вызов функции input1, определенной в коде сценария в разделе заголовка. Несмотря на то что команда write в этой функции записана в коде Web-страницы еще в разделе заголовка, на выполнение эта функция вызывается только сейчас. Поэтому текст «Ввод текста с помощью пользовательской функции» появится во втором абзаце в окне обозревателя. Затем, опять таки с помощью функции write, создается форма и командная кнопка **Вывести текст в новое окно**. Щелчок на этой кнопке запускает на выполнение функцию input2, которая открывает новое окно обозревателя и вводит в него строку текста. (Более подробно об управлении окнами обозревателя рассказывается в главе 6.)

Результат выполнения листинга 3.4 показан на рис. 3.6.

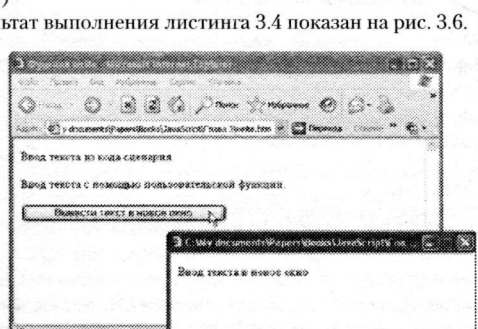



***Рис. 3.6.** Использование функции write для ввода данных в текущий документ и в новое окно обозревателя*

## Динамическое редактирование Web-страницы

Поскольку функцию write удобно использовать для последовательного ввода текста и других элементов на Web-страницу, этот прием чаще используется для динамического создания новых документов в окнах и рамках. Для произвольного доступа к элементам теку-

щей Web-страницы лучше использовать другие подходы. Так, язык JavaScript позволяет создавать объекты любых элементов Web-страницы, вставлять их в определенной позиции в текущем документе и динамически изменять в ходе работы с Web-страницей.

### *Создание объектов элементов Web-страницы*

Для создания объектов Web-страницы используется функция `createElement`. Синтаксис этой функции очень простой: в аргументе указывается дескриптор HTML соответствующего объекта в кавычках, но без угловых скобок <...>. Например:

- `document.createElement('P')` — создает новый абзац, аналогичен `<P></P>`;
- `document.createElement('TABLE')` — создает новую таблицу, аналогичен `<TABLE></TABLE>`;
- `document.createElement('TD')` — создает новую ячейку таблицы, аналогичен `<TD></TD>`, и т.д.


Команда `createElement` создает новый объект в оперативной памяти компьютера. Теперь этот объект можно добавить в определенное место на Web-странице. Для примера в листинге 3.5 создается новый абзац текста.

**Листинг 3.5. Динамическое создание и добавление объектов Web-страницы**

```
<BODY>
<DIV ID='tag' NAME='tag'></DIV>
<SCRIPT>
par = document.createElement("P");
par.innerText = "Текст абзаца";
tag.appendChild(par);
</SCRIPT>
</BODY>
```

При загрузке Web-страницы с помощью дескриптора `<DIV ID= ='tag' NAME='tag'>` создается пустой именованный раздел, после чего выполняется код сценария. В сценарии создается новый объект абзаца под именем `par`, которому присваивается текст: `par.innerText = = "Текст абзаца";`. Итак, абзац уже содержит текст, но его еще нет на странице. В следующей строке программы абзац добавляется в раздел `tag` с помощью команды `tag.appendChild(par);`. В данном случае

команда `appendChild` добавляет *дочерний* объект в *родительский* блочный объект. Эта команда применима ко всем объектам, однако необходимо учитывать логику отношений дочерних и родительских объектов. Объект абзаца не может быть дочерним по отношению к объекту другого абзаца, а объект ячейки таблицы может быть дочерним по отношению к объекту строки таблицы, но не самой таблицы. При работе с таблицами, списками и некоторыми другими объектами помимо общей команды `appendChild` можно использовать специальные команды, которые мы рассмотрим в следующих подразделах.

### Динамическое редактирование таблиц

Для добавления строк и ячеек к таблице в JavaScript применяются специальные функции: `insertRow(`*номер_строки*`)` и `insertCell(`*номер_ячейки*`)`. Удобство использования этих функций состоит в том, что они позволяют добавлять строки и ячейки в определенном месте таблицы.


Предположим, нам необходимо иметь возможность добавлять в таблицу на Web-странице новые записи с помощью полей формы, как показано на рис. 3.7.

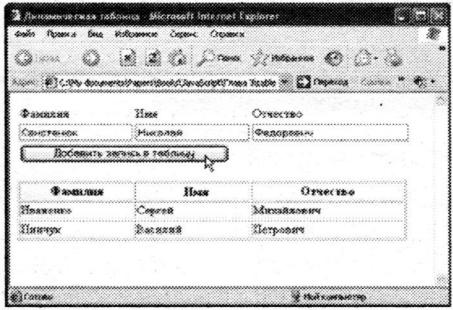



***Рис. 3.7.*** *Добавление записей в таблицу с помощью текстовых полей и кнопки*

Вам не составит труда создать подобную форму самостоятельно. За формой в коде HTML была создана таблица под именем `table` со строкой заголовков, а кнопке **Добавить запись в таблицу** присвоена функция обработки события `add_entry`. Код функции показан в листинге 3.6.

**Листинг 3.6. Динамическое добавление строк в таблицу**

```
function add_entry(surname,name,middle) {
  row = table.insertRow(1);
  cell3 = row.insertCell(0);
  cell3.innerText = middle;
  cell2 = row.insertCell(0);
  cell2.innerText = name;
  cell1 = row.insertCell(0);
  cell1.innerText = surname;
  }
```

В функцию передаются значения полей **Имя**, **Фамилия** и **Отчество**. В первой строке функции `add_entry` в таблицу добавляется новая строка. Указан индекс ввода — `1`. Таким образом, каждый щелчок на кнопке будет вставлять новую строку под строкой заголовка, смещая все остальные строки вниз.


Обратите внимание: нумерация строк, как и ячеек, начинается с числа `0`, т.е. первая строка в таблице является нулевой. Номер последней строки равняется числу строк таблицы минус единица. Если индекс окажется больше этого числа, строка не будет добавлена в таблицу.

Затем происходит добавление ячеек в новую строку и ввод соответствующего текста. Все ячейки вводятся под индексом `0` и каждая новая ячейка смещает предыдущие вправо по строке. Поэтому первой добавленной ячейке мы присваиваем текст последней ячейки в строке: Попробуйте самостоятельно изменить код таким образом, чтобы ячейки вводились в строку последовательно, с первой по последнюю.

Данную программу можно усовершенствовать — таблицу со строкой заголовков не вводить жестко в код HTML, а создать динамически при первом щелчке на кнопке. Можно добавить программу, которая проверяла бы, есть ли данные в полях и не совпадают ли они с уже существующими записями, чтобы избежать ввода повторяющихся и пустых строк. Можно добавить средства удаления записей из таблицы. Но все эти инструменты требуют более сложного программирования с проверкой условий, выполнением циклов и вычислениями положения ячеек в таблице. Об управлении ходом выполнения программы речь пойдет в главе 5.

### Динамическое изменение списков

Заменим на Web-странице таблицу объектом раскрывающегося списка `<SELECT ID='selector' NAME='selector'></SELECT>`. Заменим также код функции `add_entry`, как показано в листинге 3.7.

**Листинг 3.7. Динамическое добавление пунктов в список**

```
function add_entry(surname,name,middle) {
  entry = surname + " " + name + " " + middle;
  opt = document.createElement('OPTION');
  opt.value = entry;
  opt.text = entry;
  selector.options.add(opt,0);
  }
```

Теперь записи из текстовых полей формы заносятся в раскрывающийся список (рис. 3.8).

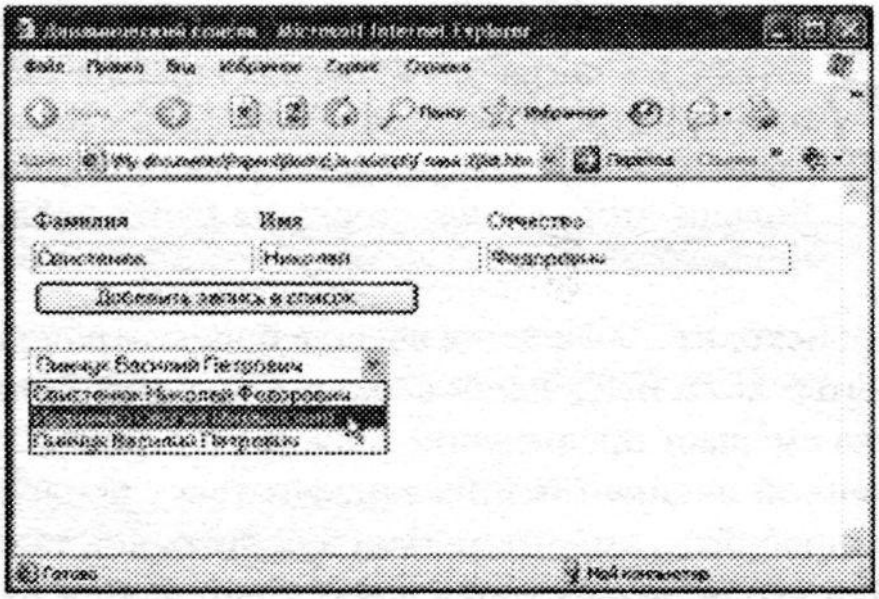



***Рис. 3.8.*** *Динамическое добавление пунктов в список*

В первой строке функции фамилия, имя и отчество, взятые из полей формы, объединяются в одну строку `entry`. Затем создается объект пункта списка `opt`, параметрам которого `value` и `text` присваивается полученная строка текста. Пункт списка добавляется в начало списка командой *имя_списка*`.options.add(`*объект_пункта, индекс*`)`.

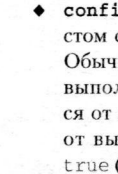
# Сообщения для посетителей Web-страниц

JavaScript предоставляет много средств для организации диалога между создателем и посетителями Web-страницы. Наиболее эффективны в этом плане диалоговые окна, с помощью которых можно как сообщить информацию, так и попросить подтверждения действия или задать вопрос посетителю. Кроме того, сообщения для пользователей можно показывать в строке состояния и строке заголовка окна обозревателя.

## Использование диалоговых окон

В JavaScript используются три встроенных диалоговых окна. Заголовки и набор кнопок в этих окнах постоянны. Текст сообщения задается аргументом функции открытия диалогового окна.

- **`alert('сообщение')`** — открывает диалоговое окно с текстом сообщения и единственной кнопкой OK (рис. 3.9). В качестве сообщения можно использовать обычные строки текста, заключенные в кавычки, цифры, переменные и вызовы функций, возвращающие текстовые или цифровые значения. Это диалоговое окно применяется для показа предупреждений или информационных сообщений, не требующих от пользователя принятия каких-либо решений.




Функцию `alert` удобно использовать для отладки сценариев. Поместите функцию `alert` в любой строке кода сценария, чтобы определить сбойную строку или отобразить текущие значения переменных.

- **`confirm('сообщение')`** — открывает диалоговое окно с текстом сообщения и двумя кнопками — OK и Cancel (рис. 3.10). Обычно в сообщении пользователю предлагается подтвердить выполнение операции, щелкнув на кнопке OK, или отказаться от выполнения щелчком на кнопке Cancel. В зависимости от выбора кнопки функция `confirm` возвращает значение `true` (OK) или `false` (Cancel). Возвращенное значение затем анализируется в конструкции с оператором `if`. (Подробнее переменные, типы данных и операторы рассматриваются в главе 4.)

- **prompt('сообщение','текст по умолчанию')** — открывает окно для ввода данных пользователем. Кроме кнопок **OK** и **Cancel** данное окно содержит текстовое поле ввода (рис. 3.11). В текстовом поле по умолчанию отображается текст, заданный во втором аргументе функции `prompt`. Чтобы оставить текстовое поле пустым, введите во втором аргументе пустую строку — `""`. Если второй аргумент пропустить, то в текстовом поле отобразится текст ***undefined***, как показано на рис. 3.11. После щелчка на кнопке **OK** функция `prompt` возвращает значение текстового поля, а после щелчка на кнопке **Cancel** — значение `false`.


Пример использования диалоговых окон показан в листинге 3.8.

**Листинг 3.8. Применение стандартных диалоговых окон в коде сценария**

```
<HTML>
<HEAD>
<TITLE>Диалоговые окна</TITLE>
<SCRIPT>
function do_payment(cart) {
  alert(cart);
}

function finish() {
  alert("До свидания,вы покидаете систему
        электронных платежей!");
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
alert("Вы вошли в систему электронных платежей!");
if(confirm("Хотите ли вы выполнить платеж?")) {
  cart = prompt("Введите номер кредитной карточки.");
  if(confirm("Вы ввели номер карточки " + cart + ",
     пожалуйста подтвердите."))
    do_payment(cart);
  else
    finish();
  }
```

```
else
  finish();
</SCRIPT>
</BODY>
</HTML>
```

Основной раздел содержит сценарий, который выполняется автоматически во время загрузки Web-страницы. В первой строке сценария функция `alert` открывает окно сообщения, показанное на рис. 3.9.

Далее выполняется конструкция `if(confirm("Хотите ли вы выполнить платеж?"))`. Функция `confirm` открывает диалоговое окно, показанное на рис. 3.10.

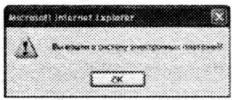



***Рис. 3.9.*** *Окно сообщения*

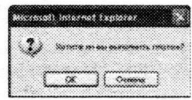



***Рис. 3.10.*** *Окно подтверждения*

После щелчка на кнопке **OK** выполняются команды, следующие за конструкцией с оператором `if`, а если щелкнуть на кнопке **Cancel** — выполняются команды, следующие за оператором `else`. Таким образом, если вы подтвердите желание выполнить платеж, следующая команда `prompt("Введите номер кредитной карточки.")` откроет диалоговое окно, показанное на рис. 3.11.

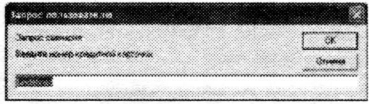



***Рис. 3.11.*** *Окно запроса сценария*

Введите номер карточки и щелкните на кнопке **OK**. Введенное число будет сохранено в переменной `cart`.

Далее в сценарии следует еще одна конструкция с оператором `if` и функцией `confirm`. Попробуйте самостоятельно на практике разобраться в работе вложенных операторов `if`. Если не получится, не огорчайтесь — мы вернемся к этой теме в главе 4. Обратите также внимание на то, что в функции `confirm` текст сообщения формируется из строк и значения переменной `cart`:

```
confirm("Вы ввели номер карточки " + cart + ",
        пожалуйста, подтвердите.")
```

Результатом выполнения сценария будет либо вызов функции do_payment, в которую передается номер карточки (в нашем примере эта функция просто покажет номер в окне сообщения), либо вызов функции finish, которая сообщает о завершении выполнения программы.

## Сообщения в строке состояния

Для управления текстом строки состояния используются два метода.

- **defaultStatus = text** — устанавливает текст строки состояния по умолчанию. Обычно эта установка производится во время загрузки Web-страницы в сценарии основного раздела.
- **status = text** — используется в функциях обработки событий для установки контекстно-зависимых сообщений.

Строка состояния принадлежит объекту window, поэтому данные методы вызываются либо для объекта окна обозревателя по его имени, либо для текущего окна с ключевым словом self или window. Например, ниже показан синтаксис обработки события ONFOCUS:

```
ONFOCUS = "self.status = 'Введите свой
          адрес электронной почты'"
ONFOCUS = "window.status = 'Введите свой
          адрес электронной почты'"
ONFOCUS = "MyWindow.status = 'Введите свой адрес
          электронной почты'"
```

В первых двух случаях устанавливается текст строки состояния текущего окна, а в третьем случае — строка состояния окна с именем MyWindow.


В случае использования метода status для обработки событий ONMOUSEOVER и ONMOUSEOUT функция должна завершаться командой return true:

```
ONMOUSEOVER = "self.status = 'Моя электронная
почта'; return true;"
```


Познакомимся на примере листинга 3.9 с использованием сообщений в строке состояния.

**Листинг 3.9. Установка сообщений в строке состояния**

```
<BODY>
<SCRIPT>
self.defaultStatus = "Глава: Детские годы"
</SCRIPT>
<H1 ALIGN="center">Детские годы</H1>
<TABLE ALIGN='left' WIDTH='200'><TR><TH>
<A HREF='house.jpg'><IMG SRC='house_small.jpg'
ONMOUSEOVER="self.status='Щелкните, чтобы посмотреть большую
фотографию'; return true;" TITLE="Щелкните, чтобы открыть"></
A><BR>Дом, где прошло детство художника
</TH></TR></TABLE>
<P>Внешне история карьеры … </P>
</BODY>
```

Раздел <BODY> начинается со сценария, в котором устанавливается текст в строке состояния по умолчанию. Далее следует разметка Web-страницы с помощью таблицы, в ячейках которой находятся рисунок и подпись к рисунку, причем рисунок является графической гиперссылкой, открывающей копию рисунка большего формата в новом окне обозревателя. Чтобы сообщить пользователю об этом, событию

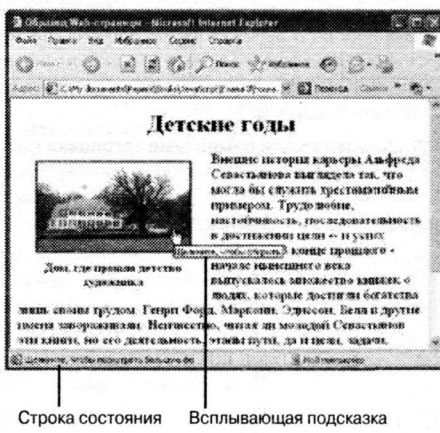



***Рис. 3.12.*** *Показ сообщения для пользователя в строке состояния и с помощью всплывающей подсказки*

ONMOUSEOVER в дескрипторе рисунка <IMG> задан метод вывода соответствующего текста в строке состояния. Как только указатель мыши выйдет за рамку рисунка, в строке состояния восстановится текст, заданный по умолчанию. Результат выполнения программы показан на рис. 3.12.


Показ сообщений в строке состояния — не самый эффективный метод информирования посетителей Web-страницы. Часто это сообщение просто остается незамеченным. Кроме того, размеры строки состояния недостаточны, чтобы вместить подробное объяснение. Использование всплывающей подсказки, устанавливаемой с помощью атрибута TITLE, будет гораздо эффективнее (см. рис. 3.12).

## Динамическое изменение заголовков окна обозревателя

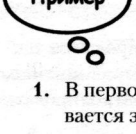
Динамическое изменение заголовков полезно при работе с наборами рамок. Содержимое рамок постоянно меняется, но в окне обозревателя отображается заголовок, заданный в файле набора рамок, этот заголовок может не соответствовать основному содержимому окна. (Более подробно о наборах рамок рассказывалось в главе 2.) В листинге 3.10 показан код страницы с набором рамок, в котором текст заголовка изменяется динамически каждый раз при смене содержимого основной рамки main.

**Листинг 3.10. Динамическое изменение заголовка набора рамок**

```
<HTML>
<HEAD>
<TITLE>Заголовок по умолчанию</TITLE>
<SCRIPT>
function update_title() {
  var i = 1;
  while(i < 100) {
  if(main.document.title) {
    document.title = main.document.title;
    break;
  }
  i = i + 1
}}
```

```
</SCRIPT>
</HEAD>
<FRAMESET ROWS='10%,*'>
  <FRAME NAME='top' SRC='top.htm'>
  <FRAME NAME='main' SRC='main.htm' ONLOAD='update_title();'>
</FRAMESET>
</HTML>
```

В коде листинга 3.10 создается набор, состоящий из двух горизонтальных рамок — `top` (для панели навигации) и `main` (для отображения основного текста). Содержимое рамки `main` будет постоянно меняться в результате щелчков на гиперссылках в панели навигации. Чтобы решить проблему, связанную с отображением в окне обозревателя заголовка текущего документа, в дескрипторе `<FRAME>`, который определяет основную рамку, для события `ONLOAD` устанавливается функция `update_title`. Код функции определен в сценарии в разделе заголовка. Идея очень проста: заголовку окна `document.title` присваивается заголовок документа, находящегося в основной рамке `main.document.title`. Но эту идею не так-то просто реализовать. Проблема в том, что событие `ONLOAD` активизируется с началом загрузки документа в рамку, т.е. функция `update_title` вызывается в тот момент, когда еще ни документа, ни его заголовка в рамке нет.




Эту проблему можно решить с помощью конструкции оператора цикла `while` и условного оператора `if` (более подробно о циклах и условных операторах рассказывается в главе 5).

1. В первой строке определяется переменная `i`, которой присваивается значение `1`.
2. Затем задается цикл `while(`*условие*`) {`*тело цикла*`;}`. Команды *тела цикла* будут выполняться до тех пор, пока выполняется заданное *условие*. В данном случае условие состоит в том, что переменная `i` меньше `100`: `i<100`. Поскольку с каждым циклом `i` увеличивается на `1`, цикл повторится как минимум 99 раз — этого достаточно для того, чтобы дождаться загрузки заголовка страницы в рамке `main`.
3. Помимо приращения переменной `i` в теле оператора выполняется проверка еще одного условия с помощью оператора

`if(`*условие*`) {`*операторы*`;}`. Операторы конструкции `if` выполняются только в случае выполнения заданного *условия*.

4. В качестве условия оператора `if` проверяется наличие объекта заголовка в документе рамки `main`: `main.document.title`. Как только заголовок будет обнаружен, первый оператор конструкции `if` присвоит заголовок документу с набором рамок, а следующий за ним оператор `break` прервет цикл.


Объект заголовка будет истинным, даже если он окажется пустым. Попробуйте самостоятельно добавить в код функции `update_title` еще одну конструкцию с оператором `if`, который будет проверять, не является ли заголовок открываемого документа пустым: `main.document.title != ''`. Только если это условие выполняется, странице присваивается заголовок загружаемого документа, в противном случае восстанавливается заголовок по умолчанию: `else {document.title = '`*Заголовок по умолчанию*`';}`. Чрезвычайно важно проследить, чтобы в коде функции число открывающих фигурных скобок ({) соответствовало числу закрывающих скобок (}).

## Динамическое форматирование элементов Web-страницы

В главе 2 вы познакомились с принципами форматирования текста и прочих элементов Web-страницы. Цвет, стиль, выравнивание, наличие рамки и некоторые другие свойства можно контролировать следующими способами:

- с помощью специальных дескрипторов форматирования;
- с помощью атрибутов в дескрипторах элементов Web-страницы;
- с помощью параметров атрибута `STYLE`.

Дескрипторы форматирования можно добавить или удалить, используя атрибут `innerHTML`, но это самый длинный и неэффективный путь. Для динамического форматирования элементов страницы лучше применять атрибуты дескрипторов HTML.

## Установка атрибутов в коде HTML

Чтобы получить доступ к атрибутам элемента Web-страницы из кода сценария, необходимо присвоить этому элементу уникальное имя, установив его в дескрипторе элемента для атрибутов ID и NAME. Если уникальное имя задано, то значение атрибута можно изменить динамически в коде сценария следующим образом:

```
имя_элемента.атрибут = значение
```

Например, скроем границы таблицы с именем tb:

```
tb.border = 0
```

Гораздо больше возможностей для форматирования элементов Web-страницы предоставляет атрибут STYLE. В главе 2 вы узнали, что значением атрибута STYLE является текстовая строка со списком параметров форматирования и их значений, например:

```
<P ID=par1 NAME=par1 STYLE='color: red; font-style: italic'>
```

В коде сценария JavaScript стиль объекта можно изменить двумя способами: присвоить атрибуту STYLE строку текста с параметрами так же, как в коде HTML, или обратиться непосредственно к параметру, как в следующем примере:

```
par1.style.color = "FFFF00";
```

## Изменение цвета фона и гиперссылок Web-страницы

В коде HTML можно именовать все объекты Web-страницы, создаваемые с помощью соответствующих дескрипторов. Но именование никогда не применяется к обязательным дескрипторам:

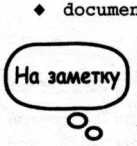
- <HTML> — соответствует объекту window;
- дескрипторы заголовка Web-страницы и <BODY> — соответствуют объекту document.

Например, для доступа к тексту заголовка использовался следующий код (см. раздел «Динамическое изменение заголовков окна обозревателя» этой главы):

```
document.title = текст;
```

Доступ к атрибутам, устанавливаемым в дескрипторе <BODY>, также можно получить посредством объекта document:

- **document.bgColor** — цвет фона;
- **document.fgColor** — цвет текста страницы по умолчанию;
- **document.linkColor** — цвет неиспользованной гиперссылки;
- **document.alinkColor** — цвет активной гиперссылки в тот момент, когда на нее наведен указатель мыши;
- **document.vlinkColor** — цвет использованной гиперссылки.




В случае изменения настроек цвета в одной из рамок или в другом окне обозревателя перед словом document нужно указать имя рамки или окна:

```
win2.document.bgColor = blue;
main.document.linkColor = white;
```


Пример динамического изменения фона Web-страницы показан в листинге 3.11.

**Листинг 3.11. Динамическое изменение фона Web-страницы**

```
<HTML>
<HEAD>
<TITLE>Выбор фона</TITLE>
<SCRIPT>
function change_color(bg,fg) {
  document.bgColor = bg;
  document.fgColor = fg;
}
</SCRIPT>
<BODY BGCOLOR="ivory" ONFOCUS="document.bgColor='ivory'"
ONBLUR="document.bgColor='gray'">
<H2>Выберите цвет фона</H2>
<FORM>
Щелкните на кнопке:
<BR><INPUT TYPE=button VALUE="Белый" ONCLICK="change_color
('white','black');">
<INPUT TYPE=button VALUE="Желтый" ONCLICK="change_color
('yellow','blue');">
<INPUT TYPE=button VALUE="Синий" ONCLICK="change_color
('blue','yellow');">
```

```
<INPUT TYPE=button VALUE="Черный" ONCLICK="change_color
('black','white');">
</BODY>
</HTML>
```

Страница содержит набор кнопок, изменяющих цвет фона и текста по умолчанию (рис. 3.13). Событиям ONCLICK кнопок присвоено выполнение функции change_color, в аргументах которой передаются соответствующие значения цветов. Определение функции находится в сценарии в разделе заголовка.

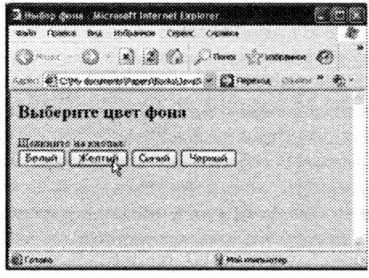



***Рис. 3.13.*** *Щелчок на кнопке изменяет цвет фона и текста Web-страницы*

Обратите внимание на использование событий ONFOCUS и ONBLUR в дескрипторе <BODY>. Изначально фону Web-страницы задан цвет слоновой кости — ivory. Если вы щелкнете на рабочем столе Windows или перейдете к окну другого приложения, в окне обозревателя произойдет событие потери фокуса — ONBLUR. Функция обработки этого события изменяет цвет фона страницы, он становится серым. Но как только вы вновь выберете окно обозревателя, событие ONFOCUS восстановит цвет фона, заданный по умолчанию.

## Документирование кода сценариев

Код сценариев должен не только правильно работать, но и быть понятным настолько, чтобы другой разработчик или вы через определенный период времени смогли легко разобраться в программе, когда придет время обновить Web-страницу.

Чтобы облегчить работу с кодом сценария, всегда подбирайте для функций и переменных такие имена, которые объясняли бы их назначение. Но использование слишком длинных имен затрудняет работу и усложняет код. Добавить более длинное описание назначения функции, переменной или логического блока внутри кода сценария можно с помощью *комментариев*.

Комментарии вводятся в код сценария двумя способами. Короткий однострочный комментарий вводят после символов //. Такой комментарий может занимать целую строку или находится в конце строки программного кода:

```
// Выполняем цикл по элементам массива
var i = 0;  // устанавливаем начальное
            // значение счетчика циклов
```

Длинные многострочные комментарии начинаются символами /* и заканчиваются символами */, например:

```
/*************************************************
* Этот сценарий предназначен для вычисления     *
* налога за автомобиль. Данные о типе автомобиля*
* годе выпуска берутся из соответствующих полей *
* формы.
*************************************************/
```

Обозреватель сумеет распознать символы начала и окончания комментария в примере, показанном выше, даже если вы окажетесь в затруднении.

# Глава 4

# Работа с данными в сценариях JavaScript

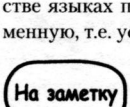
На вашем компьютере установлено множество программ для работы с текстом и электронными таблицами, рисунками и фотографиями, звукозаписями и видеоклипами, а также множество других приложений, необходимых для повседневной работы и отдыха. Что общего между всеми этими программами? Они, как изначально и сам компьютер, предназначены для хранения и обработки данных. Все данные, будь то текст, рисунок или звук, хранятся в памяти компьютера в виде универсального двоичного кода. В то же время для работы с данными разных типов требуются свои специфические методы и подходы. Программист должен иметь возможность выбрать из множества данных на компьютере требуемое значение, использовать его и сохранить результат таким образом, чтобы его вновь можно было использовать в дальнейшем. Все эти проблемы решаются с помощью *переменных* разных *типов*, речь о которых пойдет в этой главе.

## Создание переменных

Первое, что вам необходимо уяснить, — это смысл использования переменных. Итак, что же такое *переменная*? Все данные на компьютере хранятся в ячейках памяти на встроенных или съемных носителях, таких, как жесткий диск, дискета, компакт-диск или устройства флэш-памяти, либо в оперативной памяти компьютера. Все ячейки памяти имеют свои уникальные двоичные адреса. Кроме того, для записи разных данных требуется разное количество элементарных ячеек памяти. Чтобы упростить работу, связанную с выделением памяти и записью данных в ячейки, для языков программирования была разработана концепция переменных. Переменную можно представить как именованный резервуар, в котором хранятся определенные данные, называемые *значениями* переменных. На компьютере с *именем* переменной ассоциируется адрес массива ячеек памяти, в которых

хранится значение. Но как компьютер узнает, сколько ячеек нужно выделить для сохранения данных? Об этом компьютеру сообщает *тип* переменной.

Таким образом, прежде чем использовать переменную, в большинстве языков программирования необходимо сначала *объявить* переменную, т.е. установить ее имя и тип.




JavaScript представляет собой упрощенную версию языка программирования. В JavaScript объявление переменных происходит автоматически при присвоении им первого значения, что не допускается в других, профессиональных, языках. Более того, JavaScript позволяет динамически изменять тип переменной, просто присвоив ей значение иного типа. Хотя это возможно, такое поведение считается дурным тоном программирования, как и отсутствие объявлений переменных.

Для объявления переменных в JavaScript используется ключевое слово `var`. Хотя объявления можно пропустить, код сценария станет понятнее, если в самом начале будет дан список используемых переменных:

```
var name = prompt("Введите ваше имя.");  // имя пользователя
var amount = currency.value;             // сумма денег, взятая
                                         // из поля "Сумма"
var date = new Date();                   // объект текущей даты
var rate = 5.05;                         // курс
```

За ключевым словом `var` следует имя переменной. Тип переменной устанавливается автоматически в ходе присвоения переменной первого значения. Оператором присваивания служит знак равенства (=). Значение, находящееся справа от знака равенства, присваивается переменной слева. Справа от знака равенства может находиться:

- **константное значение** — число или текст (текстовое значение должно быть заключено в кавычки;
- **вызов функции** — в примере выше использовались функция `prompt`, открывающее диалоговое окно и окно, возвращающее текст, введенный пользователем (см. главу 3), а также функция `Date`, возвращающая объект текущей даты (подробнее об объекте даты рассказывается в главе 6);

- **другая переменная** — имя переменной, которая была объявлена ранее, в том числе атрибуты элементов Web-страницы, например, значение текстового поля, как в примере выше.

## Имя переменной

В JavaScript следует придерживаться определенных правил при именовании переменных:

- в имени можно использовать латинские буквы, цифры и подстрочный символ «_»;
- имя переменной должно начинаться с буквы или с подстрочного символа, но не с цифры;
- в имени переменной нельзя использовать знаки пунктуации, специальные символы, а также пробелы;
- имя переменной чувствительно к регистру букв (`sum` и `Sum` будут рассматриваться как две разные переменные);
- длина имени переменной не ограничена.


В сценарии у всех переменных с перекрывающимися областями видимости должны быть уникальные имена. (Об областях видимости переменных пойдет речь далее в этой главе.) Если в сценарии будет объявлена переменная с таким же именем, как у ранее созданной переменной, новая переменная заменит собой предыдущую.


Желательно использовать такие имена переменных, которые объясняли бы тип и назначение хранимых в них данных. Но при этом следует быть максимально лаконичным и использовать понятные аббревиатуры слов и понятий.

## Типы данных

Хотя JavaScript относится довольно демократично к типу данных переменной и позволяет изменять его в ходе выполнения программы, знание типов данных вам необходимо, поскольку с каждым из них связаны определенные ограничения в использовании переменной. В JavaScript различают следующие типы данных:

- целое число;
- число с плавающей запятой;

- строка текста;
- логическая переменная;
- объект.


Как уже отмечалось, смена типа переменной в ходе выполнения программы — это дурной тон программирования. Напротив, хорошим тоном будет маркирование типа переменной в ее имени. Например, имена переменных могут выглядеть следующим образом: `iVal`, `fVal`, `sVal`, `bVal` и `oVal` (в соответствии с типами данных, представленными в списке выше). Первая строчная буква указывает тип переменной, а за ней начинается собственно имя переменной с прописной буквы — этим подчеркивается, что первая буква была служебной.


Выделять объекты в особый тип данных не совсем верно. В действительности переменные других типов также являются объектами. Но далее в этой книге под объектами мы будем понимать составные переменные, содержащие несколько значений одного типа или разных типов. К этой же группе можно отнести переменную `null`, которая не содержит никаких значений. Подробнее об объектах в качестве переменных вы узнаете далее в этой главе, в разделе «Объекты и массивы данных».

## Целые числа

Переменные этого типа могут содержать целочисленные значения, как положительные, так и отрицательные. Отрицательные числа начинаются со знака «-». В сценариях можно использовать целые числа разных нумерических систем: привычной нам десятеричной, восьмеричной и шестнадцатеричной. На то, что данное число записано в восьмеричной системе, указывает лидирующий `0`, а признаком шестнадцатеричной системы являются лидирующие `0x`. Например, запишем число 14 в разных нумерических системах.

- Десятеричная: `14`.
- Восьмеричная: `016`.
- Шестнадцатеричная: `0xE`.

Целочисленные переменные можно использовать в математических операциях, с которыми мы познакомимся ниже в этой главе.

В листинге 4.1 показан пример объявления и использования целочисленных переменных.

**Листинг 4.1. Использование целых чисел**

```
<HTML>
<HEAD>
<TITLE>Целые числа</TITLE>
</HEAD>
<BODY>
<SCRIPT>
// Объявления переменных
var iVal = 14;    // десятеричное число
var hVal = 0xE;   // шестнадцатеричное число
var oVal = 016;   // восьмеричное число
// Вывод переменных в окне обозревателя
document.write("Это десятеричное число: " + iVal + "<BR>");
document.write("Это шестнадцатеричное число: "
               + hVal + "<BR>");
document.write("Это восьмеричное число: " + oVal + "<BR>");
document.write("Это сумма чисел: " + (oVal + hVal) + "<BR>");
</SCRIPT>
<BODY>
</HTML>
```

Результат выполнения сценария показан на рис. 4.1. Обратите внимание: функция `write` выводит все значения как десятеричные числа. Целые числа можно суммировать и использовать в других вычислениях независимо от того, в какой нумерической системе они были введены.

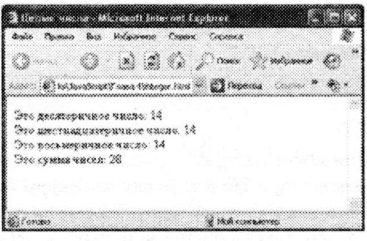



***Рис. 4.1.*** *Результат выполнения сценария, код которого приведен в листинге 4.1*

## Числа с плавающей запятой

Числовые значения этого типа могут представлять целую и дробную часть числа, отделенные точкой. На уроках математики в школе вы ставили десятичную запятую, но в англоязычных странах принято использовать десятичную точку, поэтому в JavaScript число 3,2 следует вводить так: `3.2`.

Зачем вообще потребовалось вводить два типа числовых значений? Дело в том, что для значений с плавающей запятой компьютер резервирует значительно больше ячеек памяти, чем для целых чисел, даже если дробная часть отсутствует. Например, число 2.0 займет больше памяти, чем целое число 2. Результаты математических операций с целыми числами будут автоматически округляться до целочисленного значения, что может привести к ошибке. И наоборот, в том случае, когда дробное значение не имеет смысла, например при вычислении необходимого числа упаковок товара, к ошибке может привести использование чисел с плавающей запятой.

Числа с плавающей запятой в JavaScript могут быть представлены в двух форматах: обычном и логарифмическом. Логарифмический формат имеет следующий синтаксис:

```
#.##E??
```

Это соответствует следующей математической операции:

$$\#.\#\# \times 10^{??}, \text{ где } 0.0 \leq \#.\# \leq 1.0.$$

Например, число 5 140 000.0 можно представить как 5.14E6, а число 0.0006023 — как 6.023E-4.


JavaScript автоматически выводит в окне обозревателя числа с плавающей запятой в логарифмическом формате, если число превышает значение 1.0E20 или меньше числа 1.0E-7.

## Строки текста

Строки текста могут содержать последовательности любых символов, в том числе и цифр. Но в этом случае цифры будут рассматриваться программой как текст, а не как числовые значения.

Чтобы присвоить текст переменной, строку следует заключить в кавычки. Можно использовать как одинарные (`'`), так и двойные (`"`) кавычки, но они должны быть одинаковыми в начале и в конце

строки. Если строка заключена в двойные кавычки, то в тексте можно использовать одинарные. Чтобы ввести в текст двойные кавычки, используйте символ обратного слеша: `\"`. Этот символ указывает программе, что за ним идет обычный текст, а не управляющий символ. Тогда добавление в текст одного символа обратного слеша будет выглядеть так: `\\`.

Если текст выводится в окно обозревателя с помощью команды `document.write`, то в нем для форматирования можно использовать дескрипторы HTML. Ниже приведен фрагмент программного кода присвоения текста переменной и вывода значения на экран. Результат выполнения программы показан на рис. 4.2.

```
sVal = "В тексте мы можем использовать<I>\"кавычки\"</I>,
заключать слова в <B>\<угловые\></B> или \\косые\\ скобки.";
document.write(sVal);
```

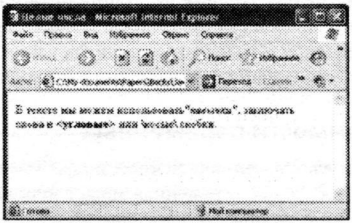


***Рис. 4.2.*** *Вывод текстовой переменной в окно обозревателя*

## Логические переменные

Логические переменные могут принимать только значения `true` (истинно) и `false` (ложно). Впрочем, этим переменным также можно присваивать целочисленные значения, при этом `0` будет соответствовать `false`, а любое другое число — `true`. Обычно логические переменные используются в логических конструкциях управления выполнением программы с использованием операторов `if` и `else`. Более подробно о логических операторах рассказывается в следующей главе.

# Управление переменными

Переменные в сценариях создаются для последующего использования в вычислениях и вывода результатов в окне обозревателя, диалоговом окне или для подачи запроса формы. Начинающие программисты часто сталкиваются с тем, что переменная, созданная ранее, оказывается недоступной для использования там, где это значение необходимо. При работе с переменными следует учитывать такое понятие, как *область видимости* переменной. Другая довольно распространенная проблема связана с несоответствием типа переменной требованиям операции, в которой эта переменная используется. Как вы узнаете ниже, в разделе «Выражения и операции», использование одного и того же оператора приведет к совершенно разным результатам или даже вызовет ошибку, в зависимости от типа переменных, к которым он применен. Данная проблема становится особенно острой в JavaScript, поскольку этот язык позволяет произвольно изменять тип переменной в ходе выполнения сценария. Далее мы рассмотрим способы управления переменными и их проверки.

## Область видимости переменных

Под *областью видимости* переменных подразумевают часть программного кода, в которой переменная может использоваться. При создании переменной компьютер временно выделяет ячейки памяти для хранения ее данных. Как только необходимость в переменной исчезает, память освобождается и становится доступной для записи других переменных. В JavaScript нет средств, подобных тем, что имеются в профессиональных языках программирования, которые позволяли бы управлять памятью компьютера, поэтому выделение ячеек памяти и их очистка происходят автоматически. Разработчик сценариев должен точно знать, в какой части программного кода переменная доступна, а в какой — нет.

В зависимости от области видимости различают два вида переменных.

- **Глобальные переменные** — это переменные, созданные непосредственно в коде сценария в блоке `<SCRIPT>...</SCRIPT>` где-либо в коде Web-страницы. Во время загрузки Web-страницы эти переменные доступны для использования в функциях и сценариях, код которых находится ниже в коде HTML

Web-страницы. После окончания загрузки Web-страницы эти переменные доступны для всех функций текущего документа, а также для сценариев, запускаемых в других окнах обозревателя, если есть возможность обратиться к текущему окну по имени: *ИмяОкна.имя_переменной*. (Об управлении окнами обозревателя речь пойдет в главе 6.)


Поскольку сценарии выполняются по мере загрузки Web-страницы, глобальные переменные следует объявлять раньше, чем они будут использованы в данном сценарии, другом сценарии или в функции. Например, предположим, что есть функция `funk1`, в которой используется глобальная переменная `gVal`. В табл. 4.1 показано правильное и ошибочное использование переменной.

**Таблица 4.1. Примеры использования глобальной переменной**

| Правильное использование | Ошибочное использование |
|---|---|
| `var gVal = 10;` | `funk();` |
| *… код сценария* | *… код сценария* |
| `funk();` | `var gVal = 10;` |

Подобные ошибки случаются довольно часто. Вызов функции, использующей глобальную переменную, может произойти опосредованно в результате выполнения определенной серии событий, вызвавших соответствующие функции обработки, что не всегда очевидно во время разработки сценария. Будьте внимательны.

- **Локальные переменные** — это переменные, которые объявлены в коде функции. Использование таких переменных ограничено кодом функции ниже того места, где они были объявлены. После завершения функции все ее локальные переменные автоматически удаляются из памяти компьютера, поэтому не могут использоваться другими функциями или командами сценария, даже в том же сценарии, где функция была определена или вызвана на выполнение.


Всегда объявляйте локальные переменные в функциях с помощью ключевого слова `var`, чтобы избежать *конфликта имен* с одноименными глобальными переменными. Если переменная в функции создается путем простого присвоения значения и в данной странице

есть глобальная переменная с таким же именем, то в действительности будет происходить не создание новой переменной, а присвоение глобальной переменной нового значения. Позже это может привести к ошибке в других функциях, в которых используется глобальная переменная. Напротив, ключевое слово `var` указывает, что код функции нужно исключить из области видимости одноименной глобальной переменной и вместо нее в данной функции использовать локальную переменную. Любые изменения локальной переменной в функции никак не повлияют на значение одноименной глобальной переменной.

В листинге 4.2 показан пример правильного и ошибочного использования локальных переменных.

**Листинг 4.2. Использование локальных и глобальных переменных**

```
<HTML>
<HEAD>
<TITLE>Использование переменных</TITLE>
<SCRIPT>
// В функциях используются глобальная и локальная переменные
function right() {
  var inText
  inText = prompt("Введите ваш текст:");
  if (inText)
    alert(gText + inText);
}

function wrong() {
  alert(gText + inText);
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
// Глобальная переменная
var gText = "Был введен следующий текст: ";
</SCRIPT>
<FORM>
<INPUT TYPE='button' VALUE="Правильное использование
переменной" ONCLICK='right()'><BR>
```

```
<INPUT TYPE='button' VALUE="Ошибочное использование
переменной" ONCLICK='wrong()'>
</FORM>
<BODY>
</HTML>
```

В сценарии основного раздела Web-страницы определена глобальная переменная `gVal`, которой присвоена строка `"Был введен следующий текст: "`. Затем создаются две кнопки **Правильное использование переменной** и **Ошибочное использование переменной**, вызывающие на выполнение функции `right` и `wrong` соответственно. Первая функция сначала открывает диалоговое окно с предложением ввести текст, сохраняет этот текст в локальной переменной `inText`, а затем показывает в окне сообщения строку, состоящую из объединенных значений глобальной и локальной переменных (рис. 4.3). Обратите внимание: функция работает, несмотря на то, что в коде HTML объявление глобальной переменной находится ниже определения функции, поскольку вызов функции происходит уже после того, как Web-страница будет загружена в окно обозревателя.

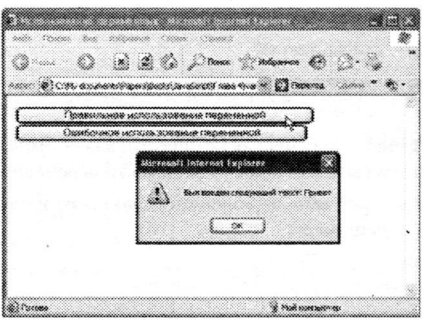



***Рис. 4.3.** Правильное и ошибочное использование локальных переменных*

Если щелкнуть на другой кнопке, то в строке состояния окна обозревателя появится сообщение: **Ошибка на странице**. Ошибка возникла вследствие того, что функция обращается к переменной `inText`, которая определена в другой функции и не существует в данной.


В JavaScript области видимости переменных функций разделены не в полной мере. В серии последовательных вызовов в теле одной функции можно обратиться к переменной, которая была создана в теле ранее вызванной функции. Чтобы избежать конфликта одноименных переменных, всегда в функциях объявляйте локальные переменные с помощью оператора `var`.

## Динамическое определение типа переменной

Возможность смены типа переменной в ходе выполнения сценария JavaScript создает ситуацию, которая невозможна во многих других языках программирования. Иногда нельзя предугадать, каким будет тип переменной перед ее использованием. Например, можно предложить пользователю ввести в поле текст или число, после чего выполнить ветвление программы в зависимости от типа введенных данных. Для определения типа переменной в коде сценария используется команда `typeof` *имя_переменной*. Команда возвращает текстовое значение, соответствующее типу переменной:

- **`number`** — целое число или число с плавающей запятой;
- **`string`** — строка текста;
- **`boolen`** — логическое значение;
- **`object`** — объект;
- **`underfined`** — переменная не определена (не была создана или программа вышла за область видимости переменной).

Обычно проверку типа переменной выполняют в конструкции с оператором `if`, например:

```
if (typeof val == 'string')
  {команды обработки строкового значения}
else
  {команды обработки значения другого типа}
```


Если вы забыли ключевое слово, возвращаемое командой `typeof` для данных определенного типа, проверку можно организовать следующим образом:

- `if (typeof val == typeof 'a')` — возвращает `true`, если `val` — строка текста;

- `if (typeof val != typeof '1')` — возвращает `true`, если `val` не является числом.

(Операторы сравнения описаны ниже в этой главе, в разделе «Выражения и операции».)

## Преобразования переменных

Чаще всего конфликты типов данных возникают при использовании строк вместо чисел и наоборот. Иногда бывает необходимо число представить как строку или, скажем, вырезать из строки часть информации и преобразовать ее в число. Для выполнения этих операций в JavaScript предусмотрен ряд функций.

Чтобы преобразовать числа в строки текста, используется функция `toString()`. Вот пример использования этой функции:

```
var iVal = 5;
var sVal = iVal.toString();
```

В результате получим переменную `sVal`, значением которой будет строка `'5'`.


Строковую переменную из числа можно получить, прибавив число к пустой строке:

```
var sVal = '' + iVal;
```

Результат будет таким же, как и в предыдущем примере.

Для выполнения обратного действия используются функции `parseInt()` и `parseFloat()`, например:

```
var sVal = '5';
var iVal = parseInt(sVal);
var fVal = parseFloat(sVal);
```

Первая функция возвращает целое число, а вторая — число с плавающей запятой.


Использование `toString()` совершенно безопасно. Любое число можно преобразовать в строку, а если переменная окажется строкой, применение функции `toString()` никак ее не изменит. Напротив, использование функций `parseInt()` и `parseFloat()` может вызвать ошибку или привести к потере данных. Если в начале строки содержится какой-либо символ, кроме цифр, функции возвратят значение `NaN`, что значит *ничего*. Допускается только наличие пробелов в начале или

в конце строки, но пробел или другой символ внутри числового значения разорвет число. Так, `parseInt(1 983)` возвратит `1`. Аналогично, отсечение целочисленной части произойдет в случае применения функции `parseInt` к числу с плавающей запятой или к соответствующей строке. В то же время эти функции можно использовать для преобразования в числа строк, представляющих собой значения в восьмеричном или шестнадцатеричном формате, а также числа с плавающей запятой в логарифмическом формате.

## Объекты и массивы данных

*Массивы* данных используются во многих языках программирования. Идея состоит в том, чтобы собрать несколько однотипных или взаимосвязанных значений под одним именем, вместо создания множества переменных. *Объекты* представляют собой более сложно организованные массивы данных, поскольку они кроме значений разных типов содержат также *методы* обработки этих значений.


Методами называются функции, являющиеся частью объекта. Вызов метода происходит следующим образом: *`имя_объекта.метод()`*. Как видите, с методами вам уже приходилось иметь дело, например с методом `write`, который принадлежит объекту `document`.

Но в JavaScript все переменные, даже самые простые, являются объектами со своими методами (в частности метод `toString`, с которым вы познакомились выше). Массивы данных JavaScript также являются одним из типов встроенных объектов. JavaScript содержит обширную коллекцию встроенных объектов; для их описания и описания всех их методов потребуется отдельная книга. В этой книге вы познакомитесь с общими принципами использования некоторых встроенных объектов: `Math` и `String` (далее в этой главе, в разделе «Выражения и операции»), а также с объектом `Date` (в главе 6). Перечисленные объекты содержат коллекции полезных методов, которые можно использовать для вычислений и обработки данных.

Объекты массивов и словарей, которые мы рассмотрим в этом разделе, кроме предоставления полезных методов, позволяют создавать коллекции данных.

Общим для объектов JavaScript является использование ключевого слова new при создании нового *экземпляра* объекта. Команда typeof, вызванная для экземпляра объекта, возвращает значение object.

## Массивы

Массивы содержат коллекции взаимосвязанных данных. В JavaScript, в отличие от многих других языков программирования, один массив может содержать данные разных типов. Можно создать массив значений, а также массивы объектов других типов (в том числе и массивов — *многомерные массивы*), массивы методов, гиперссылок, файлов мультимедиа и т.д. В этом разделе мы рассмотрим общие принципы создания и использования массивов. Пример массива звуковых файлов будет рассмотрен в главе 6.

### Создание и использование массивов

Массив представляет собой особый тип переменной, содержащей не одно, а много значений, называемых *элементами* массива. Элементы массива хранятся в той последовательности, в которой они добавлялись в массив. К любому элементу можно обратиться по имени массива, указав порядковый номер элемента, называемый *индексом*. Синтаксис обращения следующий:

```
элемент = имя_массива[индекс];
```

Нумерация элементов массива начинается с нуля. Таким образом:

```
первый_элемент = имя_массива[0];
последний_элемент = имя_массива[размер_массива - 1];
```

Индекс может быть задан числом, целочисленной переменной или функцией, возвращающей целочисленное значение.


Установка в индексе числа, превышающего размер массива, приведет к ошибке выполнения сценария. Чаще всего ошибка происходит при обращении к последнему элементу массива, когда в качестве индекса используют значение размера массива, в то время как индекс последнего элемента на единицу меньше.

Новый массив создается с помощью ключевого слова new:

array1 = new Array(#), где # — размер массива.


Размер массива указывает число элементов массива. Так, в строке `array1 = new Array(10)` создается массив из десяти элементов с `array1[0]` по `array1[9]`.


При создании объектов в JavaScript используется следующий синтаксис: `new` *`класс_объекта(аргументы)`*. Несколько напоминает вызов функции, не так ли? Действительно, эту функцию называют *конструктором класса*. Обратите внимание на то, что имя конструктора класса чувствительно к регистру букв.

При создании массива можно задать его размер. Впрочем, размер массива очень просто изменить в ходе выполнения сценария. Если размер не указан, создается пустой массив.

Следующий этап состоит в заполнении элементов массива данными, как в следующем примере:

```
months = new Array(12);
months[0] = "январь";
months[1] = "февраль";
```

Мы заполнили два первых элемента массива названиями месяцев года. Остальные 10 элементов остались пустыми. Если сейчас с помощью функции `write` или `alert` мы выведем содержимое третьего элемента, `months[2]`, то отобразится значение `undefined`.

Существует другой способ создания массива одновременно с присвоением значений всем его элементам:

```
months = new Array("январь", "февраль", "март", "апрель",
"май", "июнь", "июль", "август", "сентябрь", "октябрь",
"ноябрь");
```

Одной строкой мы создали массив месяцев года из 11 элементов, по оплошности пропустив `"декабрь"`. Позже мы исправим эту ошибку.

## Методы массивов

Массив, как любой другой стандартный объект JavaScript, содержит подборку полезных методов. Методы массивов предназначены для преобразования массива в строку и для изменения порядка следования элементов.

### *Изменение размера массива и порядка следования элементов*

Увеличить размер массива очень просто. Присвойте значение элементу с индексом, превышающим текущий размер массива. Массив автоматически будет увеличен до элемента с максимальным индексом. Например, в следующем примере создается массив из 10 элементов, причем значение присвоено только последнему элементу.

```
array1 = new Array();
array1[9] = "значение последнего элемента";
```

Все остальные элементы массива остались неопределенными.

Значение размера массива хранится в переменной `length`, встроенной в объект массива. В любой момент мы можем определить текущий размер массива и использовать это значение, например вывести в окне сообщения:

```
alert(month.length);
```


Обратите внимание на то, что `length` — это не метод, а имя переменной. Переменные, встроенные в объект, называются *переменными-членами* или *свойствами* объекта. В переменной `length` хранится информация о числе элементов массива. Максимальный индекс массива можно вычислить так:

```
max_index = имя_массива.length - 1;
```

Объект `Array` содержит ряд методов, которые позволяют изменять структуру массивов:

- **`concat(массив2)`** — объединяет два массива и возвращает новый массив, в котором элементы массива 2 следуют за элементами исходного;
- **`slice(индекс1,индекс2)`** — возвращает часть массива от элемента с индексом 1 до элемента с индексом 2, не включая его (чтобы скопировать массив до конца, просто пропустите второй аргумент);
- **`splice(индекс1,индекс2)`** — работает так же, как и предыдущий метод, но извлекаемый подмассив удаляется из исходного массива;
- **`push(значение)`** — добавляет новый элемент в конец массива;
- **`pop()`** — возвращает последний элемент и удаляет его из массива;

- **shift()** — возвращает первый элемент и удаляет его из массива;
- **unshift(значение)** — добавляет новый элемент в начало массива;
- **reverse()** — инвертирует порядок следования элементов в массиве;
- **sort()** — сортирует элементы массива в алфавитном порядке.


При сортировке текстовых элементов массива следует учесть, что на нее повлияет регистр букв.

### *Преобразование массива в строку и поиск элементов*

Для преобразования массива в строку можно использовать два метода: `toString()` и `join()`. Оба метода по умолчанию возвращают список элементов, разделенных запятыми (рис. 4.4). Метод `join()` позволяет установить другой разделитель, указав его в аргументе функции, например `months.join('; ')`.

Методы преобразования массива в строку элементов можно использовать для быстрого поиска элемента в массиве. Предположим, у вас есть огромный массив элементов, которые добавлялись в массив в неизвестной для вас последовательности. Как определить индекс искомого элемента? Один подход состоит в использовании цикла `for` (см. главу 5), в котором последовательно каждый элемент массива будет сравниваться с искомым. Этот поиск займет много времени, особенно если вам нужно определить индексы многих элементов. Быстрее будет работать программа поиска элементов в строке текста. Пример такой программы реализован в листинге 4.3 в функции `find_month`. В этой функции используются пока еще не известные вам методы строковых объектов, с которыми вы познакомитесь далее в этой главе, в разделе «Строковые операции».


Использование методов объекта `Array` показано в листинге 4.3.

**Листинг 4.3. Создание массива данных и управление им**

```
<HTML>
<HEAD>
<TITLE>Массив</TITLE>
```

```
<SCRIPT>
// Поиск элемента в массиве
function find_month() {
  month = prompt("Введите месяц: ");
  if (month) {
    var i = NaN;
    list = months.join("$");
    position = list.indexOf(month);
    if (position > -1) {
      var index = 0;
      if (position) {
        sub_list = list.substring(0,position);
        sub_array = sub_list.split("$");
        index = sub_array.length - 1;
      }
      alert("Месяц " + months[index] + " в массиве
            находится под индексом " + index);
    }
    else
      alert("Месяц " + month + " в массиве
            отсутствует")
  }
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
// Создаем массив
months = new Array("январь", "февраль", "март", "апрель",
"май", "июнь", "июль", "август", "сентябрь", "октябрь",
"ноябрь");
document.write("Исходный массив: ", months.toString(),
"<BR><BR>");
// Добавляем в массив новый элемент
months[11] = "декабрь";
document.write("Новый массив: ", months.join("; "),
"<BR><BR>");
// Извлекаем два новых массива и объединяем их
spring_months = months.slice(2,5);
autumn_months = months.slice(8,11);
offseason = spring_months.concat(autumn_months);
```

```
document.write("Весенние месяцы: ", spring_months.join("; "),
"<BR><BR>");
document.write("Осенние месяцы: ", autumn_months.join("; "),
"<BR><BR>");
document.write("Весенние и осенние месяцы: ", offseason.
join("; "), "<BR><BR>");
// Изменение порядка следования элементов
months.reverse();
document.write("Инвертированный список: ", months.join("; "),
"<BR><BR>");
months.sort();
document.write("месяцы в алфавитном порядке: ", months.
join("; "), "<BR><BR>");
</SCRIPT>
<FORM>
<INPUT TYPE='button' VALUE='Найди месяц' ONCLICK='find_
month()'>
</FORM>
<BODY>
</HTML>
```

В сценарии основного раздела Web-страницы создается массив `month` с названиями месяцев. Список месяцев выводится на страницу в виде строки текста с помощью методов `document.write()` и `month.toString()`. Затем мы добавляем в массив еще один месяц и выводим на экран новый список с помощью метода `join`. Далее демонстрируется использование методов `slice` и `concat` для разделения и объединения массивов, и методов `reverse` и `sort` — для изменения порядка элементов в массиве. Результат выполнения сценария показан на рис. 4.4.

Сортировка массива изменила начальный порядок следования месяцев (см. рис. 4.4). Чтобы определить новый индекс месяца, на Web-страницу добавлена кнопка **Найди месяц**. Эта кнопка запускает на выполнение функцию `find_month`, определенную в сценарии в разделе заголовка. Функция открывает диалоговое окно для ввода названия месяца, после чего определяет индекс месяца в массиве и возвращает это значение в окне сообщения (см. рис. 4.4). В коде функции использовались методы строчных объектов, с которыми мы познакомимся ниже в этой главе.

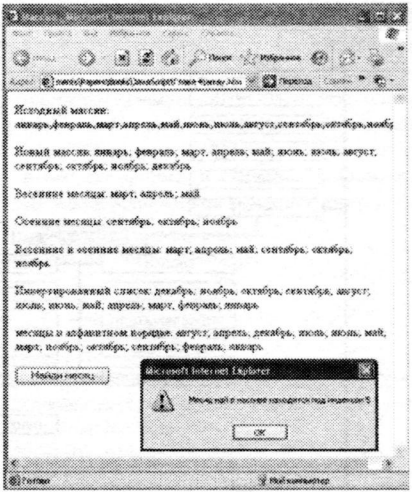



***Рис. 4.4.*** *Использование массивов в сценарии*

## Стандартные массивы Web-страницы

Вам будет полезно узнать, что любая Web-страница уже содержит ряд стандартных встроенных массивов, в которых представлены коллекции ее элементов. Имена массивов и их иерархия показаны на рис. 4.5.

Элементы Web-страницы автоматически добавляются в соответствующие массивы в той последовательности, в которой они создаются в коде HTML. Все массивы элементов принадлежат объекту `document` (соответственно, к ним можно обратиться: `document.`*`имя_массива`*), но к массиву рамок `frames` можно обратиться непосредственно из документа window: `имя_окна.frames[0]`.

Удобство использования массивов элементов состоит в том, что с их помощью можно обратиться к любому элементу Web-страницы, даже если для него не было задано уникальное имя.

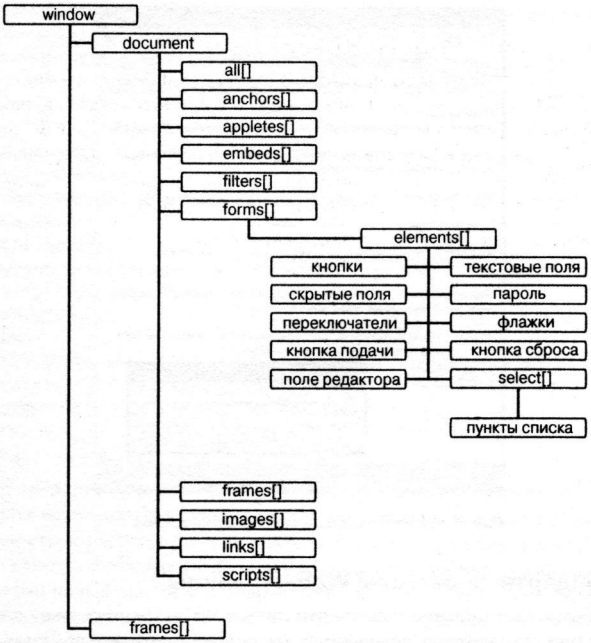



***Рис. 4.5.** Стандартные массивы элементов Web-страницы*


Обращение к элементам Web-страницы по именам все же предпочтительнее, так как число и порядок элементов в массиве могут измениться в результате динамического изменения самой Web-страницы.

Наиболее часто используются следующие массивы элементов:

- **`all`** — все элементы Web-страницы;
- **`frames`** — рамки;
- **`images`** — рисунки;
- **`links`** — гиперссылки;

- **forms** — формы;
- **elements** — элементы формы.


Чтобы на практике показать примеры обращения к массивам элементов, давайте воспользуемся ранее созданной нами Web-страницей зоопарка (см. листинг 2.10 и рис. 2.18). Усложним страницу, добавив в нее две формы и сценарий функции для кнопки. Дополнительный код показан в листинге 4.4.

**Листинг 4.4. Пример обращения к массивам элементов Web-страницы**

```
<HTML>
<HEAD>
<SCRIPT>
function analyse() {
  WinReport = window.open("","","width=250,height=100");
  WinReport.document.write("Всего элементов на странице: ",
   document.all.length, "<BR>");
  WinReport.document.write("рисунков: ", document.images.
   length, "<BR>");
  WinReport.document.write("форм: ", document.forms.
   length, "<BR>");
  WinReport.document.write("элементов в первой форме: ",
   document.forms[0].elements.length, "<BR>");
}
</SCRIPT>
</HEAD>
<BODY>
…
  код из листинга 2.10
…
<FORM>
<H2>Зарегистрируйтесь на нашем сервере</H2>
Фамилия<INPUT TYPE='text'> Имя<INPUT TYPE='text' ><BR>
Отчество<INPUT TYPE='text'><BR>
<INPUT TYPE='submit'>
</FORM>
<FORM>
<INPUT TYPE="button" VALUE="Сколько на странице элементов"
ONCLICK="analyse()"></p>
</FORM>
```

```
</BODY>
</HTML>
```

На страницу добавлена кнопка **Сколько на странице элементов**, запускающая на выполнение функцию `analyse`, определение которой находится в заголовке Web-страницы. С помощью функции мы определяем общее число элементов на странице, число рисунков, форм и элементов управления в первой форме. Результаты выводятся в новое окно обозревателя, как показано на рис. 4.6.

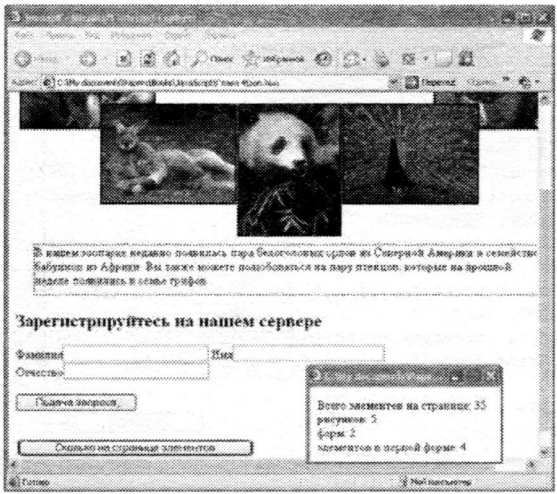



***Рис. 4.6.** Определение числа элементов на странице*

## Словари

Словари, как и массивы, используются для хранения коллекций данных. Но в отличие от массивов значения в словарях сохраняются не под номерами, а под уникальными именами, называемыми *ключами*. Ниже для сравнения показаны обращения к элементу массива и элементу словаря:

```
значение = имя_массива[индекс];
значение = имя_словаря('ключ');
```

Словарь является встроенным элементом ActiveX, доступным для использования в сценариях JavaScript. Синтаксис создания нового экземпляра ActiveX несколько отличается от знакомого вам синтаксиса создания объектов JavaScript:

```
NewDictionary = new ActiveXObject("Scripting.Dictionary");
```

Новый словарь создается пустым. Для заполнения его данными используется метод Add. Ниже показан пример создания словаря записной книжки с днями рождений:

```
dicBirthdays = new ActiveXObject("Scripting.Dictionary");
  dicBirthdays.Add("Света", "12.05.1982");
  dicBirthdays.Add("Лена", "07.11.1985");
  dicBirthdays.Add("Юля", "26.02.1983");
```

В метод Add передается пара аргументов — ключ и значение. Совсем не обязательно, чтобы это были строки текста. В качестве ключа и значения можно использовать данные любого типа, в том числе массивы, словари и другие объекты, хотя для ключей больше подходят строки текста. Вместо значений в качестве аргументов методу Add можно передавать переменные и вызовы функций, возвращающих ненулевые значения.

Оба элемента, ключ и значение, могут быть изменены в ходе выполнения сценария следующим образом:

```
dicBirthdays.Item("Лена") = "08.11.1985"; // изменяем значение
dicBirthdays.Key("Лена") = "Лена П."; // изменяем ключ
```

Так же, как и в случае использования массивов, мы можем определить размер словаря — количество пар ключ–значение в словаре. Но вместо length в словарях используется свойство Count:

```
size = dicBirthdays.Count;
```

Рассмотрим некоторые полезные методы словарей:

- **Exists("ключ")** — проверяет наличие ключа в словаре;
- **Items()** — возвращает массив значений словаря;
- **Keys()** — возвращает массив ключей словаря;
- **Remove("ключ")** — удаляет из словаря пару ключ–значение;
- **RemoveAll()** — удаляет из словаря все элементы.


В действительности извлечение из словарей массивов значений и ключей происходит не так просто. Это связано с тем, что объекты ActiveX не являются стандартными элементами языка JavaScript. Чтобы адаптировать массивы элементов к стандартам JavaScript используется следующий синтаксис:

```
var KeyArray = (new VBArray(имя_словаря.Keys()).
   toArray();
var ItemArray = (new VBArray(имя_словаря.Items()).
   toArray();
```

## Выражения и операции

Сценарии в Web-страницах часто используют для того, чтобы произвести анализ данных, введенных пользователем, и возвратить результат. Например, можно рассчитать размер текущего платежа по кредиту или проверить правильность данных в полях формы, применяя для этого методы текстового анализа. Для выполнения операций с переменными используются операторы языка JavaScript.

Выделяют следующие группы операторов:

- арифметические;
- присваивания;
- сравнения;
- логические;
- строковые.


Вместо того чтобы вновь и вновь создавать функции для выполнения рутинных операций, воспользуйтесь коллекциями методов в стандартных объектах, доступных для использования в JavaScript. Ниже мы рассмотрим применение объекта `Math` для выполнения математических вычислений и объекта `String` — для работы со строками текста.

### Арифметические операции

В сценариях JavaScript можно выполнять простые и довольно сложные вычисления с помощью стандартных операторов JavaScript, а также с помощью методов объекта `Math`. Арифметические операторы

применяются для создания выражений, в которых помимо операторов используются переменные (целочисленные и с плавающей запятой) и собственно числовые значения, называемые *литералами*. Обычно выражение также содержит оператор присваивания, который присваивает результат вычисления, находящийся справа от него, переменной, находящейся слева. Пример выражения показан на рис. 4.7.


***Рис. 4.7.*** *Пример арифметического выражения в сценарии JavaScript*

## Выполнение базовых математических операций

В JavaScript используются 5 арифметических операторов:

- `+` — суммирования;
- `-` — вычетания;
- `*` — умножения;
- `/` — деления;
- `%` — вычисление модуля.

Первые четыре оператора не требуют объяснений. Оператор вычисления модуля используется для извлечения остатка от деления. Например, в результате выполнения следующего сценария:

```
mod = 14 % 2;
```

в переменной `mod` будет сохранено значение `2`. Действительно, при делении 14 на 2 получаем число 3 и 2 в остатке. Именно остаток и возвращает оператор `%`.

## Операторы инкремента

В сценариях JavaScript часто используются счетчики — переменные, значение которых изменяется на единицу каждый раз при выполнении определенной операции. Например, можно создать на Web-странице счетчик щелчков на кнопке или на гиперссылке. Часто

счетчики применяются в циклах (см. главу 5). Создать счетчик очень просто:

```
// Начальное значение счетчика
var count = 0;
… конструкция с оператором цикла или код функции обработки
события
  // инкрементация счетчика
  count = count + 1;
```

Чтобы упростить запись, в JavaScript используются операторы инкремента:

- **count++;** — увеличивает значение переменной count на единицу;
- **count--;** — уменьшает значение переменной count на единицу.

В данном случае не важно, находится оператор инкремента до или после переменной. Конструкции ++count и count++ дадут одинаковый результат. Но положение оператора инкремента становится важным, если он используется в сочетании с оператором присваивания. Рассмотрим следующие примеры:

```
num1 = ++count; // значение count увеличивается на 1 и
                // присваивается переменной num1
num2 = count++; // значение count присваивается переменной
                // num2, после чего увеличивается на 1
```

## Приоритеты выполнения операторов

Выше мы рассмотрели примеры простейших вычислений, в которых использовался только один оператор. Но в JavaScript можно создавать сложные математические выражения, содержащие много разных операторов. В этом случае возникает вопрос, в какой последовательности будут выполняться операции. Приоритеты выполнения операторов были распределены в соответствии с общими правилами математики, известными вам еще со школы. Например, вполне очевидно, что в выражении `4 + 8/2` сначала следует разделить 8 на 2, а затем прибавить результат деления к 4. Если мы прибавим 4 к 8, а затем сумму разделим на 2, результат будет совершенно иным. Но если именно это нам и нужно, то выражение следует записать иначе: `(4 + 8)/2`.

Именно в такой последовательности будут выполняться операции в JavaScript, поскольку операторы умножения и деления имеют более

высокий приоритет, чем операторы суммирования и вычитания. Но символы скобок в коде изменяют распределение приоритетов, заставляя программу выполнять в первую очередь действия в скобках.

## Объект Math

Объект `Math` поддерживается всеми обозревателями, поэтому можно смело использовать коллекцию методов и констант этого объекта для выполнения вычислений. При этом не нужно создавать объект `Math` или добавлять его в сценарии, он всегда доступен по умолчанию. Например, чтобы найти квадратный корень от числа в переменной `fVal`, выполните следующее:

```
var result = Math.sqrt(fVal);
```

Польза от применения объекта `Math` очевидна уже по этому примеру, так как JavaScript не содержит операторов для извлечения квадратного корня, а написать самостоятельно такую программу будет весьма затруднительно. Давайте рассмотрим ресурсы объекта `Math`.

### *Константы*

В вычислениях нам часто приходится использовать константы, например число $\pi$ (пи) или число Авогадро (количество молекул в одном моле вещества). Первую константу вы найдете в объекте `Math`, вторую, к сожалению, нет. Список констант небольшой, но полезный (табл. 4.2).

**Таблица 4.2. Математические константы объекта `Math`**

| Константа | Описание |
|---|---|
| `E` | Число *e* – основание натурального логарифма (≈ 2,72) |
| `LN10` | Число ln(10) ≈ 2,3 |
| `LN2` | Число ln(2) ≈ 0,69 |
| `LOG10E` | Число lg(e) ≈ 0,43 |
| `LOG2E` | Число $\log_2(e) \approx 1{,}44$ |
| `PI` | Число $\pi \approx 3{,}14$ |
| `SQRT1_2` | Квадратный корень от ½ ≈ 0,71 |
| `SQRT2` | Квадратный корень от 2 ≈ 1,41 |

### *Методы*

Коллекция методов, предназначенных для выполнения математических операций, представлена в табл. 4.3.

**Таблица 4.3. Методы объекта `Math`**

| Метод | Описание |
|---|---|
| Тригонометрические методы | |
| `abs(x)` | Возвращает абсолютное (положительное) значение от числа $x$ |
| `acos(x)` | Возвращает арккосинус числа $x$ ($-1 \le x \le 1$) |
| `asin(x)` | Возвращает арксинус числа $x$ ($-1 \le x \le 1$) |
| `atan(x)` | Возвращает арктангенс числа $x$ |
| `cos(x)` | Возвращает косинус числа $x$ |
| `sin(x)` | Возвращает синус числа $x$ |
| `tan(x)` | Возвращает тангенс числа $x$ |
| Методы округления | |
| `ceil(x)` | Возвращает ближайшее целое число, которое меньше или равно $x$ |
| `floor(x)` | Возвращает ближайшее целое число, которое больше или равно $x$ |
| `round(x)` | Возвращает целое число, ближайшее к $x$ (`Math.round(1.5) = 2`) |
| Методы сравнения | |
| `max(x,y)` | Возвращает большее из двух чисел |
| `mix(x,y)` | Возвращает меньшее из двух чисел |
| Методы вычислений | |
| `exp(x)` | Возвращает $e^x$ |
| `log(x)` | Возвращает натуральный логарифм $ln(x)$ |
| `pow(x,y)` | Возвращает $x^y$ |
| `sqrt(x)` | Возвращает квадратный корень от $x$ |

### *Генератор случайных чисел*

Объект `Math` содержит *рандомайзер* — функцию, возвращающую случайное число. Вызов генератора случайных чисел выполняется следующим образом:

```
var rand = Math.random();
```

Переменной `rand` будет присвоено случайное число с плавающей запятой в диапазоне от 0 до 1. Используя различные ухищрения, можно получить числа в любом диапазоне. Например, ниже показана программная модель игральной кости, возвращающей целое число от 1 до 6:

```
var rand = Math.floor(5.0*Math.random() + 0.5);
```

## Операции присваивания

Оператор присваивания = уже использовался нами ранее во многих листингах. Но в JavaScript есть ряд дополнительных операторов присваивания, полезных в тех случаях, когда переменной присваивается новое значение исходя из текущего, т.е. одна и та же переменная находится слева и справа от оператора присваивания, как в следующем примере:

```
num = num + 5;
```

Это выражение можно упростить:

```
num += 5;
```

Полный список операторов присваивания показан в табл. 4.4.

**Таблица 4.4. Операторы присваивания**

| Оператор | Описание |
|---|---|
| `x = y` | Присваивает переменной x значение переменной y |
| `x += y` | Увеличивает значение переменной x на значение переменной y |
| `x -= y` | Уменьшает значение переменной x на значение переменной y |
| `x *= y` | Увеличивает значение переменной x в y раз |
| `x /= y` | Уменьшает значение переменной x в y раз |
| `x %= y` | Присваивает переменной x остаток от деления значения переменной x на значение переменной y |

## Операции сравнения

В том случае, если нужно выяснить отношения равенства или отличий значений в двух переменных, применяйте операторы сравнения:

- **==** — равенство;
- **!=** — неравенство;
- **>** — больше;
- **>=** — больше или равно;
- **<** — меньше;
- **<=** — меньше или равно.

Все операторы сравнения возвращают логические значения `true` (истинно) и `false` (ложно). Например, в выражении

```
var bVal = 5 == 3;
```

переменной bVal будет присвоено значение false. Чаще всего операторы сравнения используются для проверки условия в конструкции с оператором if (подробнее об управлении выполнением сценария в зависимости от выполнения условия рассказывается в главе 5).


В синтаксисе операторов сравнения часто возникают ошибки. Обратите внимание на то, что оператор равенства содержит два символа равенства (==), тогда как один символ равенства имеет в JavaScript совершенно иной смысл (см. выше раздел «Операции присваивания»). Следует также помнить, что в операторах неравенства, больше или равно и меньше или равно символ равенства находится справа. Комбинации символов =!, => и =< вызовут ошибку.

## Строковые операции

Со строками текста на Web-страницах приходится работать даже чаще, чем с числовыми значениями. В связи с этим в JavaScript предусмотрен обширный набор средств для анализа текста и манипуляций со строками. Как вы увидите, к строковым переменным можно применять даже некоторые математические операторы, с которыми вы познакомились выше. Набор удобных методов для работы с текстом предоставляет объект String.

## Конкатенация строк и полиморфизм операторов

Чаще всего в сценариях приходится объединять фрагменты текста, хранящиеся в разных переменных, в одну фразу для вывода ее в окне обозревателя. Процесс объединения строк называется *конкатенацией*. Проще всего это сделать с помощью оператора суммирования «+», с которым вы познакомились выше, в разделе «Выполнение базовых математических операций». Рассмотрим пример в листинге 4.5.

**Листинг 4.5. Конкатенация строк с помощью оператора суммирования**

```
<SCRIPT>
  str1 = "Глубокоуважаемый ";
  str2 = prompt("Введите имя и фамилию");
  if (str2)
    document.write(str1 + str2 + "!");
</SCRIPT>
```

Сценарий выводит приветственную строку, которую можно использовать в начале делового письма. Сначала создается строковая переменная `str1` с текстом `"Глубокоуважаемый "`. Затем нужно ввести имя и фамилию человека, к которому мы обращаемся. Программа открывает диалоговое окно с просьбой к пользователю ввести требуемую информацию. Введенная строка сохраняется в переменной `str2`. Далее программа выводит текст на экран с помощью метода `write`, в аргументе которого происходит суммирование строк. Результат показан на рис. 4.8.

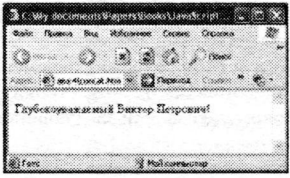



***Рис. 4.8.*** *Результат конкатенации строк*

Для конкатенации строк мы использовали оператор суммирования, который ранее применяли для выполнения сложения двух числовых переменных. Такое свойство операторов, при котором результат их выполнения зависит от типа переменных, к которым они применяются, называется *полиморфизмом* операторов. К полиморфизму также можно отнести автоматическое изменение оператором типа одной из переменных, чтобы избежать возникновения конфликта. Рассмотрим следующее выражение:

```
val = 3 + "2";
```

Как вы думаете, какой результат будет занесен в переменную `val`? Возможно, вас удивит, когда функция `alert(val);` выведет в окне сообщения значение `32`. Все дело в том, что если одна из переменных оператора суммирования окажется строкой текста, даже при условии, что она содержит цифру, вторая переменная автоматически преобразуется в строку и выполняется конкатенация строк. Такой же результат получится в выражении:

```
val = 3;
val += "2";
```

Все остальные математические операторы, -, *, /, %, действуют в противоположном направлении: пытаются преобразовать строку в число. Если это не удается, выражение возвращает значение NaN. Рассмотрим следующие примеры:

```
val = 3 - "2"; // в результате получаем 1
val = "s" * 3; // в результате получаем NaN
```

## Объект String

Все элементы текста Web-страницы и текстовые переменные, включая содержимое текстовых полей, являются экземплярами (объектами) класса String. В связи с этим к методам класса String можно обращаться напрямую: *строковая_переменная.метод(аргументы)*.

Все объекты класса String содержат переменную-член length, в которой хранится информация о длине строки — числе символов, включая пробелы. Попробуйте ввести следующий код:

```
sVal = "Привет!";
alert(sVal.length);
```

В данном случае сценарий покажет в окне сообщения число 7.

Объекты String содержат много методов, полезных для работы со строками текста, часть которых представлена в табл. 4.5.

**Таблица 4.5. Методы объекта String**

| Метод | Описание |
|---|---|
| Методы манипуляций со строками | |
| concat(str2, str3, …) | Последовательно конкатенирует строки в списке аргументов со строкой, к которой был вызван метод, например:<br>`str1.concat(str2, str3, str4);` |
| split("*разделитель*") | Разбивает строку на массив подстрок по позициям заданного символа разделителя. Например:<br>`sVal = "А,Б,В,Г";`<br>`sArray = sVal.split(",");`<br>`document.write(sArray[0]); // Выведет букву "А"`<br>`document.write(sArray[1]); // Выведет букву "Б"`<br>`document.write(sArray.length); // Выведет 4` |

*Окончание табл. 4.5*

| Метод | Описание |
|---|---|
| `substr(позиция, длина)` | Возвращает подстроку, заданную позицией и длиной. Отсчет позиций начинается с `0` |
| `substring(позиция, позиция)` | Возвращает подстроку, заданную позициями начала и конца подстроки в строке. Например:<br>`sVal = "абвгде";`<br>`alert(sVal.substring(1,3)); // возвратит "бв"` |
| Методы поиска | |
| `charAt(позиция)` | Возвращает символ по указанной позиции в строке |
| `indexOf("подстрока")` | Возвращает позицию первого вхождения подстроки в строку |
| `lastIndexOf("подстрока")` | Возвращает позицию последнего вхождения подстроки в строку |
| Методы форматирования | |
| `big()` | Увеличивает размер шрифта на единицу (см. ниже метод `fontsize`) |
| `bold()` | Устанавливает полужирный шрифт |
| `fontcolor('цвет')` | Устанавливает цвет строки |
| `fontsize('n')` | Устанавливает номер размера строки от `1` до `7` |
| `italics()` | Задает курсив |
| `link('URL')` | Создает из строки гиперссылку на заданный URL-адрес или команды `'mailto:'` и `'javascript:'` |
| `small()` | Уменьшает размер шрифта на единицу |
| `strike()` | Перечеркивает текст |


Все перечисленные выше методы форматирования текста выполняются путем заключения строки в соответствующие дескрипторы HTML (см. главу 2).


Ранее в листинге 4.3 мы создали функцию `find_month`, в которой использовались методы объекта `String`. Давайте подробно рассмотрим работу этой функции в листинге 4.6.

**Листинг 4.6. Функция `find_month`**

```
function find_month() {
  month = prompt("Введите месяц: ");
  if (month) {
    var i = NaN;
    list = months.join("$");
    position = list.indexOf(month);
    if (position > -1) {
      var index = 0;
      if (position) {
        sub_list = list.substring(0,position);
        sub_array = sub_list.split("$");
        index = sub_array.length - 1;
      }
      alert("Месяц " + months[index] + " в массиве
            находится под индексом " + index);
    }
    else
      alert("Месяц " + month + " в массиве отсутствует")
  }
}
```

Идея состояла в том, чтобы с помощью функции `find_month` определить индекс текстового элемента в массиве. Для этого мы преобразовали массив в строку с помощью метода `join`, установив в качестве разделителя символ доллара (`$`). (Мы специально выбрали символ, который не ожидали увидеть в строках нашего текста.) Затем с помощью метода `indexOf` определили позицию вхождения искомого слова в строке текста. Если искомое слово не обнаружено, метод `indexOf` возвращает значение `-1`. Убедившись с помощью оператора `if`, что поиск завершился успехом, мы вырезаем часть строки от начала до позиции искомого слова, воспользовавшись для этого методом `substring`. Полученную строку вновь преобразуем в массив с помощью метода `split`, установив в качестве разделителя все тот же символ доллара. Нетрудно понять, что индекс искомого элемента в исходном массиве будет на единицу меньше размера нового урезанного массива. Результат выполнения функции был показан на рис. 4.4.


Рассмотрим теперь пример форматирования текста с помощью методов объекта `String` (листинг 4.7).

**Листинг 4.7. Изменение размера текста**

```
<SCRIPT>
  var sVal = "Установка размера шрифта";
  for (i=1; i<=7; i++)
    document.write(sVal.fontsize(i) + "<BR>");
</SCRIPT>
```

В листинге 4.7 мы используем метод `fontsize` для установки размера шрифта. Размер шрифта задается номером от 1 до 7. Как будет выглядеть текст на экране, зависит не только от номера шрифта, но и от установок параметров обозревателя. В листинге 4.7 мы выводим один и тот же текст с размером шрифта от 1 до 7, воспользовавшись оператором цикла `for`. (Подробнее о циклах рассказывается в главе 5.) Результат показан на рис. 4.9.

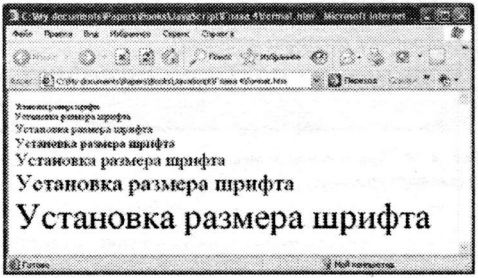



***Рис. 4.9.*** *Изменение размера шрифта с помощью метода* `fontsize`


Методы форматирования объекта `String` добавляют дескрипторы в строку текста только в том случае, если текст выводится на экран с помощью метода `write`. Если просто применить метод форматирования к строковой переменной, результат не сохранится и никак не повлияет на формат текста при последующем выводе значения переменной на экран.

Вышеуказанная проблема затрудняет установку для строки сразу нескольких параметров форматирования, например размера шрифта и курсива.


Эту проблему можно решить путем добавления необходимых дескрипторов непосредственно в строку текста. Например, добавим в листинг 4.7 следующую строку кода где-нибудь после создания переменной `sVal`, но до цикла `for`:

```
sVal = "<I>" + sVal + "</I>";
```

Результат показан на рис. 4.10.

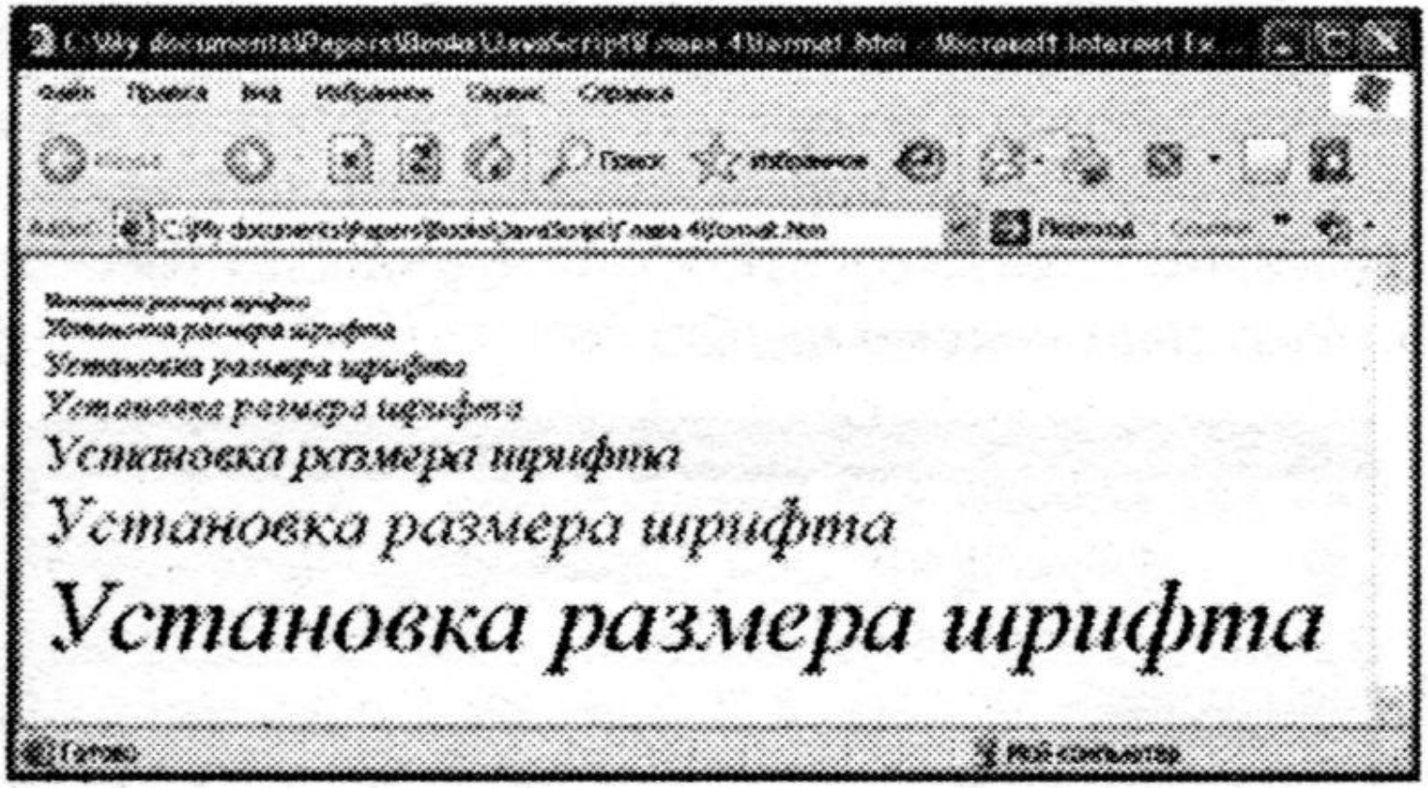



***Рис. 4.10.*** *Применение к тексту нескольких параметров форматирования*

# Глава 5

# Управление ходом выполнения сценария

В простейшем случае код сценария выполняется линейно от первой до последней строки. Но даже в предыдущих главах книги, когда мы изучали базовые основы программирования на JavaScript, нам не всегда удавалось удержаться в пределах простой линейной программы. Часто бывает необходимо разветвить программу на два альтернативных кода (подпрограммы), каждый из которых выполняется только при определенном условии. Например, в случае показа диалоговых окон нужно выполнить анализ данных, введенных пользователем, и в зависимости от того, на какой кнопке диалогового окна был сделан щелчок, выбрать для исполнения требуемый код подпрограммы.

Таким образом, нам нужно научиться решать с помощью сценариев следующие задачи:

- определять выполнение условия, критического для последующего хода программы;
- разветвлять программу на подпрограммы по факту выполнения условия;
- устанавливать временной промежуток выполнения программы;
- организовывать повторное выполнение кода программы по циклу.

В этой главе вы познакомитесь с операторами, применяемыми в JavaScript для управления ходом выполнения программы.

## Сравнение значений

Чтобы установить момент выполнения условия, нужно сравнить текущее значение переменной со значением, характеризующим данное условие. Выбор переменной и стандартного значения зависит исключительно от логики процесса, который вы хотите смоделировать

с помощью сценария. Например, таким условием может быть значение переменной-счетчика. Если доступ к странице заблокирован с помощью пароля, можно установить счетчик попыток пользователя ввести пароль. Контрольное значение счетчика может равняться трем, пяти, во всяком случае меньше десяти, после чего программа блокирует доступ к серверу для этого пользователя.

Условием выполнения программы может быть значение, возвращенное диалоговым окном. Если пользователь щелкнет на кнопке **Отменить**, функция диалогового окна возвратит значение `false`. Щелчок на кнопке **OK** возвратит значение `true` или текст, введенный пользователем в диалоговое окно. Условием может быть соответствие определенному формату значения, введенного в текстовое поле диалогового окна и формы. (Подробно о формах и диалоговых окнах рассказывалось в главе 3.)

Другим часто используемым условием может быть наличие или отсутствие на Web-странице определенного элемента. Отсутствию элемента Web-страницы соответствует значение `NaN`. Это же значение возвращается некоторыми функциями в случае возникновения в них ошибки.

В этом разделе мы рассмотрим способы определения соответствия значения переменной определенному критерию.

## Что такое «истинно» и «ложно»

С операторами сравнения вы познакомились в главе 4. В JavaScript используются шесть операторов, `==`, `!=`, `<`, `<=`, `>` и `>=`, с помощью которых можно установить равенство, неравенство или превышение значения одной переменной относительно другой. Общим для всех этих операторов является то, что все они возвращают значения логического типа `true` (истинно) и `false` (ложно). Смысл этих значений очевиден: выражение `5 == 5` возвратит `true`, тогда как выражение `5 == 2` возвратит `false`.

Переменные логического типа тесно связаны с переменными других типов и могут быть заменены ими в логических выражениях. Так, числовое значение `0` соответствует логическому `false`, в то время как все другие числа соответствуют значению `true`. Если переменная содержит в себе объект (элемент Web-страницы или текст), то в логическом выражении эта переменная будет соответствовать значению `true`, тогда как значение `NaN` и пустая строка текста `""` будут рассматриваться как `false`.

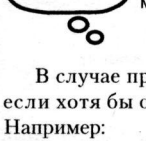
С логическими переменными работают условный оператор `if` и операторы циклов. Если условная переменная принимает значение `true`, то оператор `if` запускает на выполнение код одной подпрограммы, а если она принимает значение `false` — другой. Циклы выполняются до тех пор, пока условная переменная имеет значение `true`. Более подробно о циклах и конструкциях `if-else` вы узнаете далее в этой главе.

## Конструирование логических выражений

С помощью оператора сравнения можно проверить значение одной переменной. Но иногда бывает необходимо в качестве условия установить попадание значения переменной в определенный диапазон или связать выполнение подпрограммы с несколькими условиями. В подобных случаях нужно применять логические выражения. Для создания таких выражений используются специальные логические операторы `&&` (логическое *и*) и `||` (логическое *или*).

С помощью оператора `&&` можно определить одновременное выполнение двух условий, как в следующих примерах:

```
bVal = (x >= 20) && (x <= 30) // возвращает true, если x
                              // находится в диапазоне [20:30]
bVal = (x > 20) && y // возвращает true, если x больше 20
                     // и значение переменной y соответствует true
```




Обратите внимание на то, что вместо выражения сравнения `y == true` можно использовать только имя самой переменной `y`.

В случае применения оператора `||` выражение возвратит `true`, если хотя бы одно из двух условий выражения будет истинным. Например:

```
bVal = (txtField.value != "") || (MyArray.length > 0)
// возвращает true, если текстовое поле txtField содержит
// непустую строку текста или массив MyArray содержит
// хотя бы один элемент
```

Логические выражения могут быть еще более сложными и содержать несколько логических операторов, например, как следующее:

```
bVal = ((x >= 20) && (x <= 30)) || (y == x)
// возвращает true, если x находится в диапазоне [20:30],
// или значение y равно значению x
```


В наших примерах мы присваивали результат выражения переменной `bVal`. Эту переменную затем мы можем использовать в логической конструкции с оператором `if` или для проверки условия цикла. Но часто программисты обходятся без посредника и просто устанавливают все логическое выражение в месте кода программы, где проверяется выполнение условия.

## Выполнение задач по циклу

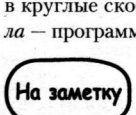
Если нам необходимо выполнить одну и ту же операцию с разными объектами или с разными значениями переменной, проще всего это сделать, организовав выполнение части кода программы по циклу. Для организации циклов в JavaScript используются два оператора: `for` и `while`. Помимо синтаксиса, разница между этими операторами состоит в логике организации цикла. Оператор `for` лучше использовать в тех случаях, когда цикл нужно выполнить определенное число раз, а оператор `while` лучше подходит для циклов, повторяющихся до достижения определенного условия или для создания бесконечных циклов.

### Цикл for

Для организации цикла с помощью оператора `for` используется следующий синтаксис:

```
for (счетчик = начальное_значение; условие_завершения;
приращение_счетчика)
  { тело цикла }
```

За оператором `for` следует код определения цикла, заключенный в круглые скобки, и заключенный в фигурные скобки код *тела цикла* — программа, повторяемая по циклу.

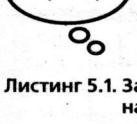



Если тело цикла `for` состоит всего из одной строки кода, то фигурные скобки можно не устанавливать. Это справедливо также для исполняемого кода операторов `while`, `if` и `else`.

Определение цикла состоит из трех выражений, разделенных символами точки с запятой.

- Первое выражение создает целочисленную переменную-счетчик и присваивает ей начальное значение.
- Второе выражение устанавливает условие, при невыполнении которого цикл прерывается.
- Третье выражение устанавливает способ приращения счетчика. Чаще всего это *счетчик*++, что означает приращение счетчика на единицу после каждого цикла. Но можно использовать и более сложные выражения, например: `счетчик = счетчик - 5;`

Можно показать множество примеров применения цикла `for` на практике. Этот оператор весьма полезен и чрезвычайно популярен у программистов. Одна из часто встречаемых задач, где используется цикл `for`, — это заполнение данными массивов и словарей или же, наоборот, вывод данных из массивов и словарей на экран.




Давайте рассмотрим пример выполнения обеих этих задач в листинге 5.1.

**Листинг 5.1. Заполнение массива и вывод элементов массива на экран с помощью цикла `for`**

```
<SCRIPT>
var size = parseInt(prompt("Введите размер массива","5"));
var ar = new Array(size);
for (n = 0; n < ar.length; n++) {
  sVal = prompt("Введите текст элемента массива","");
  ar[n] = sVal;
  document.write(ar[n] + "<BR>");
}
</SCRIPT>
```

Сначала программа показывает диалоговое окно с предложением ввести размер массива (по умолчанию предлагается значение `5`). Значение размера преобразуется в целое число с помощью стандартной функции `parseInt` и присваивается переменной `size`. В следующей строке программа создает новый массив `ar` указанного размера. Затем следует предмет нашего изучения — цикл `for`. В определении цикла создается переменная-счетчик `n`, которой присваивается начальное значение `0`. Цикл будет выполняться до тех пор, пока значение `n`

меньше размера созданного массива ar, и n увеличивается на единицу после выполнения каждого цикла.

Тело цикла содержит три строки (поэтому заключено в фигурные скобки). Сначала программа предлагает пользователю ввести текстовое значение для текущего элемента массива. В следующей строке мы воспользовались счетчиком n как индексом массива для последовательного доступа ко всем его элементам. Текущему элементу присваивается значение, введенное пользователем. Последняя строка тела цикла выводит значение текущего элемента массива на экран с помощью метода document.write.

Промежуточный этап заполнения массива показан на рис. 5.1.

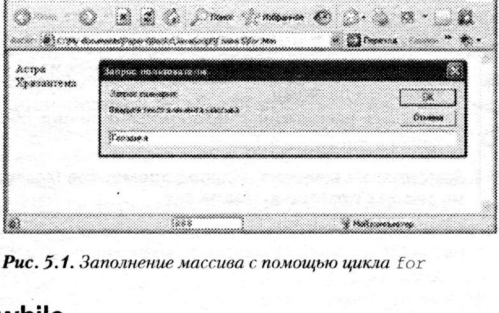



***Рис. 5.1.*** *Заполнение массива с помощью цикла* `for`

## Цикл while

Цикл while, в отличие от цикла for, не содержит счетчика. Цикл while длится до тех пор, пока выполняется заданное логическое условие. Как только логическое выражение, установленное в определении цикла, возвратит false, цикл прекратится. Это условие может никогда не наступить, тогда цикл продолжается бесконечно. Возникновение бесконечного цикла обычно вызывает ошибку в выполнении программы и зависание приложения. Но иногда бесконечные циклы создаются целенаправленно. Ниже вы узнаете об использовании оператора break для прерывания цикла, что позволяет сделать цикл зависимым от нескольких условий.

Оператор while дает возможность создавать более гибкие циклы по сравнению с жестко детерминированными циклами оператора for. Приведем некоторые полезные свойства цикла while:

- выполнение цикла можно связать с любым условием, например значением системного времени или цветом фона Web-страницы и пр., тогда как в цикле `for` контролируется только целочисленное значение счетчика;
- проверка условия происходит перед входом в цикл и после выполнения последней строки в теле цикла. Таким образом, строки тела цикла могут вообще не выполняться, если заданное условие изначально было `false`.

Синтаксис цикла `while` проще, чем у цикла `for`:

```
while (логическое_выражение)
  { тело цикла }
```


В листинге 5.2 показан простой пример, в котором раскрывается одна интересная особенность цикла `while`.

**Листинг 5.2. Особенности проверки условия в цикле `while`**

```
<SCRIPT>
count = 1;
while (count< 5) {
  document.write("Счетчик " + count + "<BR>");
  count++;
}
document.write("<BR><BR>");
count = 0;
while (count< 5) {
  count++;
  document.write("Счетчик " + count + "<BR>");
}
</SCRIPT>
```

В сценарии используются два на первый взгляд одинаковых цикла `while`. В условии проверяется значение счетчика, который инкрементируется при каждом выполнении тела цикла. Читатель может предположить, что для цикла со счетчиком лучше подойдет оператор `for`, и будет совершенно прав. Но этот пример был создан, чтобы обратить ваше внимание на особенности проверки условия в цикле `while`. Повлияет ли на работу цикла положение в теле цикла строки с приращением счетчика? Результат выполнения сценария показан на рис. 5.2.

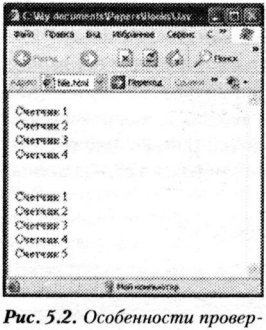



***Рис. 5.2.** Особенности проверки условия в цикле `while`*

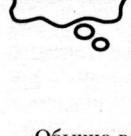
Метод `write` в теле обоих циклов выводит на страницу текущее значение счетчика. Разница только в том, что в первом цикле приращение счетчика происходит в последней строке тела цикла, а во втором цикле это первая строка. Начальные значения счетчика были подобраны таким образом, чтобы первым значением, выводимым на экран, была единица. Несмотря на то что условия циклов были одинаковыми, второй цикл вывел значения от 1 до 5, а первый — от 1 до 4. Все дело в том, что во втором цикле приращение счетчика происходило сразу после проверки условия, тогда как в первом — непосредственно перед проверкой. В случае создания аналогичного цикла `for`:

```
for (count = 1; count < 5; count++)
  document.write("Счетчик " + count + "<BR>");
```

результат будет однозначным (`1, 2, 3, 4`). Именно поэтому циклы со счетчиками лучше организовывать с помощью оператора `for`.

## Вложенные циклы

В сценариях иногда используются конструкции, когда один цикл создается в теле другого цикла. Такие циклы называются *вложенными*. Нет никаких ограничений на использование вложенных циклов и число вложений, кроме логики и здравого смысла. Это означает, что цикл `for` может быть вложен в тело другого цикла `for` или цикла `while` и иметь в своем теле любой другой цикл.




Выполнение циклов в сценарии может занять много времени и замедлить выгрузку Web-страницы. Следует помнить, что в случае использования вложенных циклов время выполнения сценария возрастет геометрически, т.е. время выполнения двух вложенных циклов будет равняться произведению времени выполнения одного цикла на время выполнения другого цикла.

Обычно вложенные циклы используют для математической обработки многомерных массивов данных или для вывода данных на

экран в табличной форме. Причем, манипулируя условиями циклов, можно создать довольно сложные многомерные композиции, как в примере на рис. 5.3.

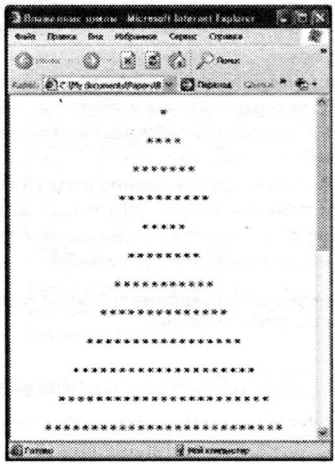


***Рис. 5.3.*** *Елочка создана с помощью трех вложенных циклов* `for`

Код сценария, с помощью которого была создана композиция на рис. 5.3, показан в листинге 5.3.

**Листинг 5.3. Создание двухмерной композиции с помощью вложенных циклов**

```
<SCRIPT>
for (i = 1;i < 10;i += 4) {
  for (j = 0; j < i + 3; j++) {
    document.write("<CENTER>");
    for (n = 0; n < i + j*3; n++)
      document.write("* ");
    document.write("</CENTER><BR>");
  }
}
</SCRIPT>
```

Первый цикл for со счетчиком i устанавливает число уровней елочки и число символов звездочки в вершине каждого уровня. Следующий цикл со счетчиком j определяет число строк в каждом уровне, а следующий вложенный цикл со счетчиком n устанавливает количество звездочек в каждой строке в зависимости от уровня. Взаимодействие между вложенными циклами достигается за счет того, что в условии каждого вложенного цикла используются значения счетчиков цикла верхнего уровня. Обратите также внимание на то, как нам удалось выровнять звездочки в окне обозревателя с помощью дескриптора <CENTER>.


Приведем другой, более практичный, пример использования вложенного цикла для динамического вывода данных многомерного массива в таблице. Код сценария показан в листинге 5.4.

**Листинг 5.4. Динамическое создание таблицы по данным многомерного массива**

```
<BODY>
<TABLE NAME='tb' ID='tb' BORDER=1></TABLE>
<SCRIPT>
arData = new Array(3);
arData[0] = new Array("1","2","3","4","5");
arData[1] = new Array("а","б","в","г","д");
arData[2] = new Array("@","#","$","%","&");
for (i = 0;i < arData.length;i++) {
  tb.insertRow(i);
  for (j = 0; j < arData[i].length; j++) {
    tb.rows[i].insertCell(j);
    tb.rows[i].cells[j].width = 50;
    tb.rows[i].cells[j].align = 'center';
    tb.rows[i].cells[j].innerText = arData[i][j];
  }
}
</SCRIPT>
</BODY>
```

В основной раздел Web-страницы добавляется пустая таблица под именем tb. Затем выполняется код сценария, в котором прежде всего создается и заполняется данными многомерный массив arData. В следующих строках сценария выполняется конструкция из двух вложенных циклов for. Первый цикл добавляет в таблицу новую

строку, тогда как следующий цикл заполняет строку таблицы ячейками и присваивает им значения из массива. (О динамическом создании таблицы речь шла в главе 3.) Обратите внимание на то, как счетчики циклов используются для адресации ячеек таблицы и элементов многомерного массива. Полученная таблица показана на рис. 5.4.

***Рис. 5.4.** Вывод данных многомерного массива с помощью конструкции вложенных циклов*

## Прерывание и продолжение цикла

В JavaScript есть специальные команды, с помощью которых можно задействовать дополнительные уровни контроля за выполнением циклов:

- **`break`** — прерывает цикл в любой точке тела цикла;
- **`continue`** — заставляет программу пропустить все следующие строки тела цикла и начать новый цикл.

Эти команды используются внутри циклов в конструкции с операторами `if-else`, чтобы проверить дополнительные условия выполнения цикла.

### Команда break

Команда `break` является необходимым элементом бесконечных циклов.


Рассмотрим следующий пример. Предположим, вы хотите защитить свою страницу с помощью своеобразного пароля и сделать ее доступной только для тех, кто умеет умножать числа. Введите код листинга 5.5.

**Листинг 5.5. Прерывание цикла с помощью команды `break`**

```
<SCRIPT>
var c = 1;
while(c) {
  num1 = Math.round(Math.random()*19 + 1);
  num2 = Math.round(Math.random()*19 + 1);
  result = prompt(("Введите произведение " + num1 + " x " +
           num2),"");
  if (result && parseInt(result)) {
    if (result == num1*num2)
      break
  }
}
</SCRIPT>
```

В начале сценария запускается бесконечный цикл `while`. Для этого переменной `c` присваивается значение `1`, а в условии оператора проверяется, является ли значение `c` истинным. Поскольку все ненулевые значения истинны (см. главу 4), а переменная `c` нигде не изменяется, цикл `while` будет продолжаться вечно, если его не остановить с помощью команды `break`. Далее программа генерирует два случайных целых числа в диапазоне [1:20] и предлагает пользователю ввести результат произведения этих чисел. В следующих строках с помощью двух операторов `if` программа проверяет, является ли введенное значение числом и соответствует ли оно произведению. Если да, цикл прерывается и продолжается загрузка Web-страницы. Если результат неправильный или пользователь щелкнул на кнопке **Отмена**, цикл будет повторяться вновь и вновь. Промежуточный этап выполнения программы показан на рис. 5.5.

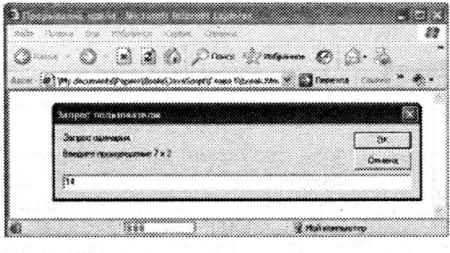



***Рис. 5.5.*** *Надеюсь, на этот раз я ввел правильное число*

## Команда continue

Команду `continue` часто применяют в случаях, когда необходимо отследить недопустимое значение счетчика или другой изменяемой переменной цикла, пропустить выполнение цикла для этой переменной, но продолжить цикл для последующих значений. Например, предположим, что мы хотим вывести обратные значения для ряда целых чисел. Для любого числа *x* можно получить значение *1/x* за единственным исключением — число нуль. Деление на нуль вызовет ошибку. Поэтому в программе следует установить проверку равенства переменной нулю, прежде чем выполнять деление.


Рассмотрим пример в листинге 5.6.

**Листинг 5.6. Прокручивание цикла вхолостую с помощью команды `continue`**

```
<HTML>
<HEAD>
<SCRIPT>
function culc() {
  var start = Math.round(document.forms[0].elements[0].
              value);
  var end = Math.round(document.forms[0].elements[1].
            value);
  if (start && end && (start < end)) {
    output.innerHTML = "";
    for(n = start;n <= end;n++) {
      if (n == 0)
        continue
      sVal = 1/n + ";<BR>";
      output.innerHTML += sVal;
    }
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Первый член ряда: <INPUT TYPE='text' VALUE='-5'><BR>
Последний член ряда: <INPUT TYPE='text' VALUE='5'><BR><INPUT
TYPE='button' VALUE='Вывести ряд обратных чисел'
ONCLICK='culc()'><BR>
```

```
</FORM>
<P NAME='output' ID='output'></P>
</BODY>
</HTML>
```

Web-страница предлагает ввести в поля формы значения начала и окончания ряда целых чисел, для которого программа рассчитает и выведет обратные значения *1/x*, как показано на рис. 5.6.

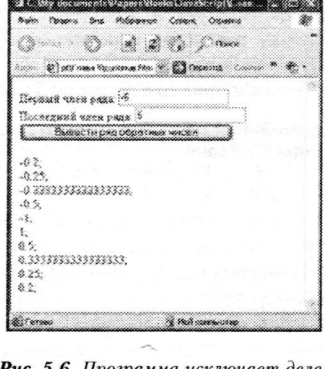



***Рис. 5.6.*** *Программа исключает деление на нуль*

Выполнение сценария запускается щелчком на кнопке **Вывести ряд обратных чисел**. Обратите внимание на то, как в сценарии мы можем обратиться к значениям полей, даже если им не были присвоены уникальные имена. Для этого воспользуемся массивами элементов, о которых вы узнали в главе 4.

Далее программа проверяет, были ли введены какие-либо данные в поля, а также следит за тем, чтобы первый член числового ряда был меньше последнего члена. Затем очищается текст именованного абзаца и запускается цикл `for`, в котором вычисляются значения *1/x*. Если в числовом ряду оказывается нуль, то выполняется команда `continue`, которая запускает следующий шаг цикла со значением счетчика, увеличенным на единицу.

# Ветвление программного кода

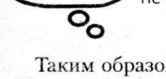
Циклы делают программный код мощнее, позволяя многократно использовать один и тот же фрагмент кода для последовательной обработки множества значений в массиве данных. Благодаря ветвлению программного кода сценарий становится более гибким, чувствительным к контексту выполнения программы и к требованиям пользователя. Выбор той или иной подпрограммы зависит от выполнения или невыполнения определенного условия. Для ветвления программного кода JavaScript используют конструкции операторов `if-else`, `switch-case` и условный оператор `?`.

## Оператор if

Оператор `if` наиболее часто используется для ветвления программного кода или выполнения части кода по условию. В простейшем виде синтаксис оператора `if` выглядит так:

```
if (логическое выражение)
  { код, выполняемый в случае возвращения
  логическим выражением значения true }
```

Если логическое выражение в операторе `if` возвратит `false`, весь блок кода, заключенный в фигурные скобки, будет пропущен, и управление перейдет к строке сценария, следующей за блоком оператора `if`.




Если код, находящийся под управлением оператора `if`, состоит всего из одной строки, фигурные скобки можно не вводить.

Таким образом, в первом варианте с помощью оператора `if` выделяется программный блок, выполнение которого необязательно и зависит от условия. Однако зачастую необходимо создать два альтернативных программных блока, один из которых обязательно выполняется в зависимости от выполнения или невыполнения условия. Для моделирования таких ситуаций используется конструкция операторов `if-else`:

```
if (логическое выражение)
  { код, выполняемый в случае возвращения
  логическим выражением значения true }
```

```
else
  { код, выполняемый в случае возвращения
  логическим выражением значения false }
```

Конструкции `if-else` мы довольно часто использовали в предыдущих листингах (например, листинг 4.3 в предыдущей главе).

## Множественное ветвление программного кода

Мы рассмотрели простейший пример ветвления программного кода на две подпрограммы по условию, возвращающему одно из двух значений: `true` или `false`. Но в реальных ситуациях, которые мы пытаемся смоделировать с помощью сценариев, не все так просто. Если нужно сделать выбор между тремя или большим числом подпрограмм, одной конструкции `if-else` будет недостаточно. В зависимости от логики выполнения программы, проблему можно будет решить за счет вложенных конструкций `if-else` или с помощью операторов `switch-case`. В следующих подразделах вы узнаете, в каком случае нужно использовать тот или иной подход.

### Вложенные операторы if

Часто ответ на один вопрос вызывает дополнительные вопросы. Логика таких сценариев следующая: если условие «А» выполняется, то нужно выполнить подпрограмму 1, если не выполняется, то нужно проверить условие «В» и по результатам проверки выполнить подпрограмму 2 или 3. Для решения этой проблемы можно использовать следующую комбинацию операторов `if` и `else`:

```
var A = логическое выражение;
var B = логическое выражение;
if (A)
  { код 1-й подпрограммы }
else {
  if (B)
    { код 2-й подпрограммы }
  else
    { код 3-й подпрограммы }
}
```

Выбор из трех подпрограмм — это еще довольно простая задача. Часто в сценариях приходится проверять много разных условий, и бесконечные чередования `if` и `else` затрудняют чтение и понимание

кода сценария. Чтобы упростить код, в JavaScript используется модифицированный вариант конструкции `if-else` для проверки ряда условий: `if-else-if`. Предыдущая схема программного кода изменится следующим образом:

```
if (A)
  { код 1-й подпрограммы }
else if (B){
  { код 2-й подпрограммы }
…
else
  { код 3-й подпрограммы }
}
```

Сначала проверяется условие «A». Если «A» — `false`, проверяется условие «B». Если «B» окажется `false`, можно таким же образом добавить еще проверку условия «C» и т.д. Конструкцию завершает одиночный оператор `else`, блок которого выполняется только в том случае, если все условия оказались `false`.


Рассмотрим применение этого подхода на примере. В листинге 5.7 представлен код сценария, который анализирует текстовый символ, введенный пользователем, и сообщает, является этот символ цифрой, буквой или знаком.

**Листинг 5.7. Проверка типа символа с помощью вложенных операторов `if`**

```
<SCRIPT>
var symbol = "";
while (symbol != null) {
  symbol = prompt("Введите любой одиночный символ",symbol);
  if (symbol == null)
    break
  else if ((symbol >= "a" && symbol <= "z") || (symbol >= "A"
           && symbol <= "Z"))
    alert(symbol + " - буква латинского алфавита");
  else if (symbol >= "А" && symbol <= "я")
    alert(symbol + " - буква русского алфавита");
  else if (symbol >= "0" && symbol <= "9")
    alert(symbol + " - цифра");
  else if (symbol == "")
    alert("\"\" - пустая строка");
  else
```

```
    alert(symbol + " - знак препинания или
          специальный символ")
}
</SCRIPT>
```

В сценарии запускается цикл `while`, в первой строке которого открывается диалоговое окно с просьбой ввести символ. Далее следует программа анализа введенного символа. Первый оператор `if` проверяет, не была ли нажата клавиша **Отмена**. (В этом случае функция диалогового окна возвратит объект `null`, и оператор `break` прервет выполнение цикла.) Далее строки `else if` последовательно проверяют принадлежность символа к остальным группам: буквам латинского алфавита, русского алфавита, цифрам или пустой строке. Если символ не принадлежит ни к одной из этих групп, последний оператор `else` выводит сообщение, что был введен знак препинания или специальный символ. Пример выполнения сценария показан на рис. 5.7.

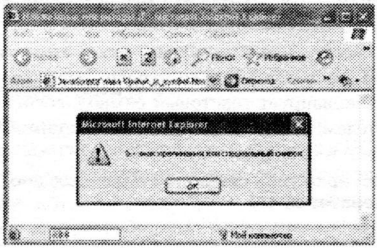



***Рис. 5.7.*** *Программа анализа введенного символа*


Обратите внимание: последовательность проверки условий в листинге 5.7 не произвольна, а подчинена определенной логике. Прежде всего проверяется условие прерывания цикла. Принадлежность символа к знакам препинания и специальным символам определяется в конце программы по той причине, что эти символы рассредоточены по кодировочной таблице. Поэтому сначала мы проверяем принадлежность символа к группам, образующим в кодировочной таблице сплошные массивы. Чтобы посмотреть кодировочную таблицу, выберите в Word команду меню **Вставка⇨Символ**.

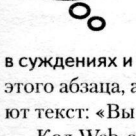
## Оператор switch

Оператор `switch` удобно применять для анализа переменных, которые могут принимать более двух значений. Типичный пример — это обработка значений раскрывающегося списка с несколькими пунктами. Синтаксис оператора `switch` следующий:

```
switch (выражение или переменная) {
  case значение:
    строки кода
    break;
  case значение:
    строки кода
    break;
  …
  default:
    строки кода
    break;
}
```

В условии оператора `switch` может быть имя переменной или выражение, возвращающее ряд значений. Тело оператора `switch` заключается в фигурные скобки и состоит из повторяющихся операторов `case`, проверяющих совпадение значения оператора `switch` с константным значением, установленным для каждого оператора `case`. Если совпадения не обнаружены, управление передается программному коду оператора `default` в конце блока оператора `switch`.


Обратите внимание на то, что блоки операторов `case` и `default` не выделяются фигурными скобками. Используется другой синтаксис: блок начинается с символа двоеточия и завершается оператором `break`.




Рассмотрим пример использования на практике оператора `switch`. На рис. 5.8 показан небольшой шуточный психологический тест.

Щелчок на картинке отображает текст в нижней части окна под заголовком **Насколько вы независимы в суждениях и поступках**. Щелчок на кнопке **Очистить** удаляет текст из этого абзаца, а щелчки на других элементах Web-страницы показывают текст: «Вы промахнулись, щелкните еще!».

Код Web-страницы показан в листинге 5.8.

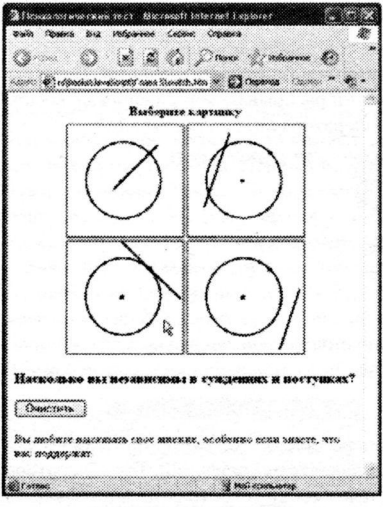



***Рис. 5.8.*** *Выберите картинку, чтобы больше узнать о своем характере*

### Листинг 5.8. Использование оператора `switch` для оценки выбора пользователя

```
<HTML>
<HEAD>
  <TITLE>Психологический тест</TITLE>
<SCRIPT>
function test(n) {
  switch (n) {
    case 0:
      output.innerText = "Вы промахнулись, щелкните еще!"
      break;
    case 1:
      output.innerText = "Вы законопослушны.
          Законопослушные люди - основа нации и государства."
      break;
    case 2:
      output.innerText = "Чтобы заставить вас что-то
        сделать, достаточно сказать, чтобы вы этого не делали."
```

```
      break;
    case 3:
      output.innerText = "Вы любите высказать свое мнение,
        особенно если знаете, что вас поддержат."
      break;
    case 4:
      output.innerText = "Вы - кот, который
        гуляет сам по себе."
      break;
    default:
      output.innerText = ""
      break;
  }
}
</SCRIPT>
</HEAD>
<BODY>
<TABLE ALIGN='center' BORDER=0>
<CAPTION ONCLICK='test(0);'><B>Выберите картинку</B></
CAPTION>
<TR><TH ALIGN='center'><IMG SRC='images\cartoon1.jpg'
BORDER=1 WIDTH='150' HEIGHT='150' ONCLICK='test(1);'></TH>
<TH ALIGN='center'><IMG SRC='images\cartoon2.jpg' BORDER=1
WIDTH='150' HEIGHT='150' ONCLICK='test(2);'></TH></TR>
<TR><TH ALIGN='center'><IMG SRC='images\cartoon3.jpg'
BORDER=1 WIDTH='150' HEIGHT='150' ONCLICK='test(3);'></TH>
<TH ALIGN='center'><IMG SRC='images\cartoon4.jpg' BORDER=1
WIDTH='150' HEIGHT='150' ONCLICK='test(4);'></TH></TR>
</TABLE>
<H3 ONCLICK='test(0);'>Насколько вы независимы в суждениях и
поступках?</H3>
<FORM><INPUT TYPE='button' VALUE='Очистить'
ONCLICK="test('clear')"> <P NAME='output' ID='output'
ONCLICK='test(0);'></P>
</BODY>
</HTML>
```

Вся интерактивность Web-страницы реализована с помощью единственной функции обработки событий, `test`, определенной в разделе заголовка кода HTML. Функция принимает один аргумент, значение которого меняется в зависимости от того, на каком элементе Web-страницы был сделан щелчок. Так, щелчки на рисунках

передают в функции значения от 1 до 4. Щелчок на кнопке Очистить передает в функцию текстовое значение, а щелчки на других элементах Web-страницы передают значение 0. В функции test значение аргумента анализируется с помощью конструкции операторов switch-case-default, в результате чего каждое значение аргумента от 0 до 4 и "clear" соответствует выводу определенного текста или очистке поля. В результате с помощью одного оператора switch программный код разветвляется сразу на шесть подпрограмм.

## Условный оператор

В том случае если программные блоки операторов if и else состоят из одиночных строк, программный код можно сделать более компактным, воспользовавшись вместо конструкции if-else условным оператором ?...:... . Приведем простейший пример использования данного оператора, где в одной строке кода мы определяем цену билета в зависимости от возраста покупателя:

```
price = (age < 18 ? "5 p." : "10 p.");
```

Если заменить эту строку на обычную конструкцию if-else, получится более длинный код:

```
if (age < 18)
  price = "5 p.";
else
  price = "10 p.";
```

Конструкции с условным оператором можно использовать внутри выражений. Предположим, что в вашей компании оптовым покупателям, закупившим товар более чем на 10 000 р., предоставлена скидка в виде бесплатной доставки товара, тогда как покупатели, которые приобрели товар в розницу, платят за доставку еще 300 р. Создадим соответствующий код программы с использованием условного оператора:

```
total_price = price + (price < 10000 ? 300 : 0);
```

Теперь эта же программа с использованием конструкции if-else:

```
if (price < 10000)
  total_price = price + 300;
else
  total_price = price
```

Экономия места очевидна.


Посмотрите еще один пример использования условного оператора для выравнивания вывода числовых значений на экран без применения таблицы. Допустим, нам нужно вывести массив данных со значениями в диапазоне от единиц до сотен. Мы хотим разместить значения в строках и колонках так, чтобы цифры в разных строках располагались строго по линии друг над другом. Это не так просто сделать, учитывая, что однозначные, двузначные и трехзначные числа имеют разную длину. Введите код программы из листинга 5.9.

**Листинг 5.9. Выравнивание значений в колонках**

```
<SCRIPT>
for (i=1;i<56;i++) {
  num = Math.round(Math.random()*998 + 1);
  output = (num < 10 ? "....." : "...");
  output += (num < 100 ? ".."+num : num);
  document.write(output);
  if (i % 10 == 0)
    document.write("<BR>");

}
</SCRIPT>
```

В сценарии запускается цикл `for`. Число циклов в данном случае не столь важно, лишь бы их было достаточно для создания наглядного массива данных. Массив целых чисел в диапазоне от 1 до 999 генерирует строка кода `Math.round(Math.random()*998 + 1)`. Затем мы формируем строку вывода `output`, прибавляя к числовому значению точки таким образом, чтобы строки с однозначными, двузначными и трехзначными цифрами были одинаковой длины. Обратите внимание на использование условного оператора для заполнения строк вывода. После каждого десятого значения выводится новая строка. Для этого проверяем, делится ли счетчик цикла `for` на 10 без остатка, используя оператор деления по модулю. (Подробно об операторе деления по модулю рассказывалось в главе 4.) Результат выполнения программы показан на рис. 5.9.

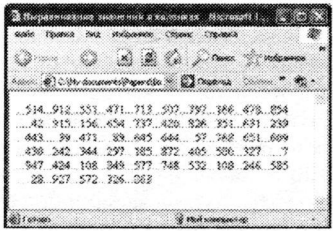


***Рис. 5.9.*** *Выравнивание чисел в колонках путем заполнения строк вывода до фиксированной длины*


Вы можете видоизменить сценарий в листинге 5.9 таким образом, чтобы выравнивались не только числовые значения, но и небольшие слова. Попытайтесь сделать это самостоятельно. Вспомните, что длину слова в сценарии JavaScript можно определить следующим образом: `слово.length`.

## Установка таймера выполнения функций

JavaScript позволяет устанавливать промежуток времени между вызовом функции и ее исполнением. Для этого используется метод `setTimeout` объекта `window`. Установка параметров метода производится следующим образом:

```
window.setTimeout("вызов_функции", время_задержки);
```

Можно указать не одну функцию, а строку из нескольких выражений и вызовов функций, отделяя каждый элемент символами точки с запятой. Вызов функции следует взять в кавычки.


Время задержки исполнения функции устанавливается в миллисекундах, т.е. значение 5000 будет соответствовать 5 с, а 10 000 — 10 с.

Пример использования таймера показан в листинге 5.10.

**Листинг 5.10. Задержка исполнения функции**

```
<HTML>
<HEAD>
```

```
<SCRIPT>
function rabbit() {
  document.write("<H1>О господи, что это?</H1><BR/><IMG
                SRC='images/goose.gif'/>");
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Сейчас я достану кролика из своей шляпы.</H1>
<FORM>
<INPUT TYPE='button' VALUE='Щелкни здесь' ONCLICK="window.
setTimeout('rabbit()', 5000)">
</FORM>
</BODY>
</HTML>
```

После загрузки Web-страницы отобразится картинка, показанная на рис. 5.10.

Текст заголовка обещает, что сейчас появится кролик из шляпы, щелкните только на кнопке. После щелчка на кнопке не происходит ничего, видимо что-то не получилось?! Но прошло 5 с, и на экране появился совсем не тот, кого мы ждали (рис. 5.11).

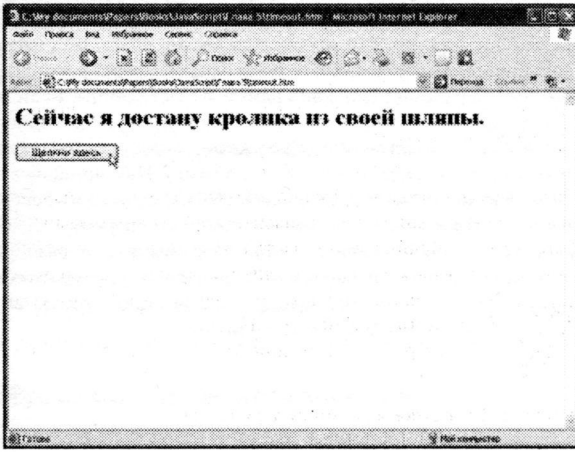



***Рис. 5.10.*** *Щелкните на кнопке, чтобы достать кролика из шляпы*

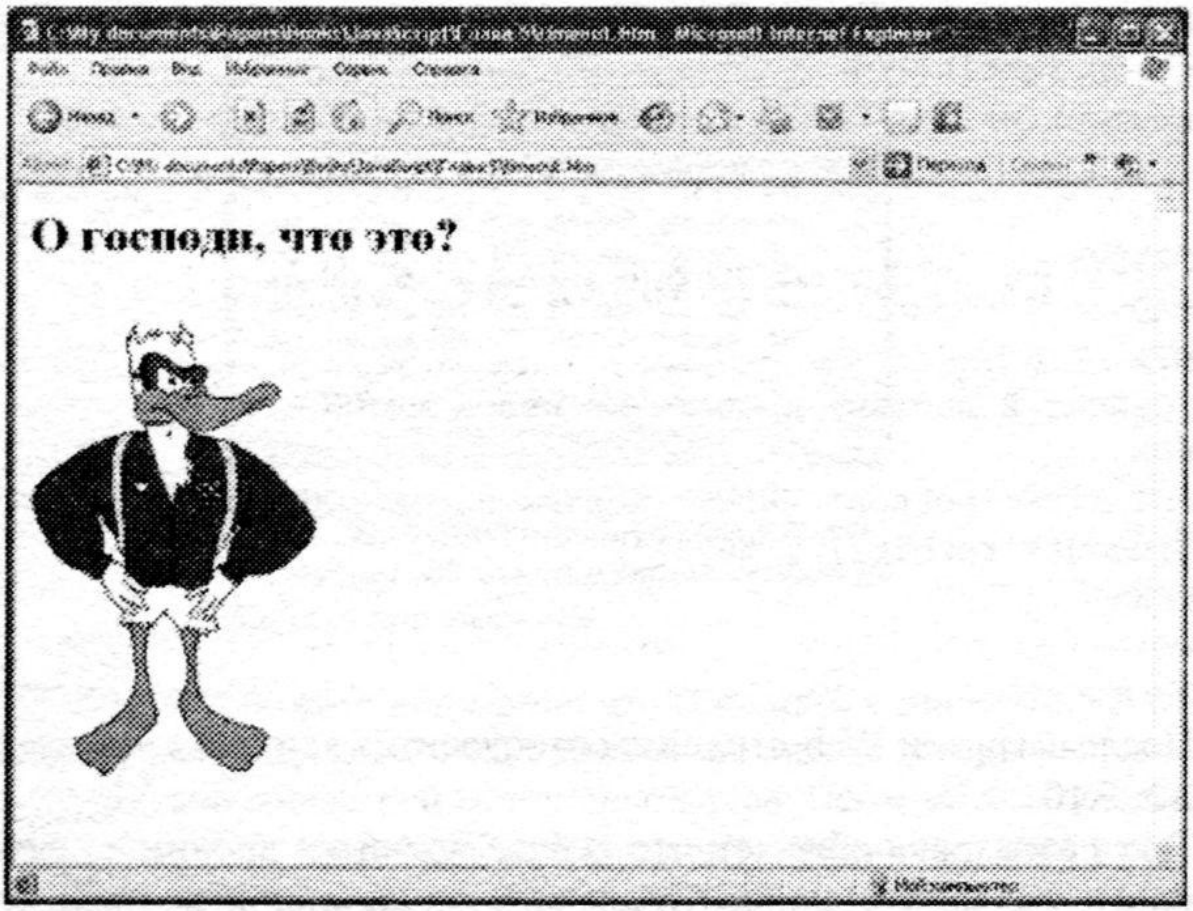



***Рис. 5.11.*** *На экране появился разъяренный гусь*

Простая шутка, но нам в этом примере следует обратить внимание на то, что выполнение функции `rabbit` было отложено на 5 с.

В главе 3 в листинге 3.10 мы создали весьма полезный сценарий для динамического изменения заголовка окна обозревателя в ответ на изменение содержимого основной рамки Web-страницы. Если помните, тогда мы столкнулись с тем, что выполнение функции обновления заголовка необходимо было задержать до того момента, пока не завершится загрузка нового документа в рамку. Нам пришлось создать довольно хитроумную комбинацию с циклом `while` и проверкой наличия заголовка нового документа с помощью оператора `if`. Суть этой конструкции не разъяснялась, но теперь вы можете разобраться в ней самостоятельно и, более того, значительно упростить ее, отложив выполнение функции `update_title` на 5 с, используя таймер `window.setTimeout`. Попробуйте сделать это.

# Глава 6

# Элементы мультимедиа в динамических Web-страницах

Web-страница, содержащая только текст, выглядит скучно и неинтересно. В этой главе вы узнаете об объектах мультимедиа и возможностях управления ими с помощью сценариев. Разумное использование мультимедийных объектов, к которым относятся рисунки, графическая анимация, видео, звуки, аплеты и элементы управления ActiveX, значительно повысит привлекательность Web-страницы. Недостаток применения спецэффектов состоит в том, что они отвлекают внимание посетителей и затрудняют чтение текста. Поэтому чрезвычайно важно, чтобы мультимедийный объект или эффект анимации появился тогда, когда это необходимо автору.

Помимо объектов мультимедиа, в этой главе мы рассмотрим средства создания эффектов анимации и анализа системной информации. Начнем главу со знакомства с объектом `Date`, возвращающим системное время компьютера.

## Возвращение даты и времени

Если в работе вы хотите использовать текущую дату и время не только для показа на Web-странице, но и для расчетов в сценариях, познакомьтесь с объектом `Date`. Этот объект позволяет:

- возвращать системную дату и время;
- отображать дату и время в разных форматах;
- производить вычисления с датой и временем.

### Отображение системного времени

Объект `Date` создается следующей строкой кода:

```
var today = new Date();
```

В переменную today заносится дата и время, извлеченные из системного таймера компьютера в момент выполнения строки кода сценария. Но в сценарии можно создать объект Date с информацией о любой произвольной дате, как в следующих примерах:

```
var today = new Date("November 5th, 2005 14:30:00")
var today = new Date(2005, 10, 5, 14, 30, 0)
```


Обратите внимание на то, что дата может вводиться как строка текста и как набор числовых значений. Данные времени (часы, минуты и секунды) в обоих случаях можно не указывать.

Объект Date по умолчанию возвращает строку текста. Чтобы убедиться в этом, введите в код сценария такую строку:

```
alert(new Date());
```

Текущее системное время отобразится в окне сообщения (рис. 6.1).

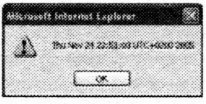



***Рис. 6.1.*** *Текущее время в окне сообщения*

## Формат данных о текущей дате и времени

От обычной строки текста объект Date отличается тем, что у него есть набор методов, которые позволяют извлекать составляющие элементы текущей даты:

- **getYear()** — год;
- **getMonth()** — месяц;
- **getDay()** — день недели;
- **getDate()** — число месяца;
- **getHours()** — часы;
- **getMinutes()** — минуты;
- **getSeconds()** — секунды.

Выполнение всех этих методов в той последовательности, как они представлены в списке, показано на рис. 6.2. (На экран текст

выводился с помощью метода `write`, например: `document.write("Год: "today.getYear(),"<BR>");`.)

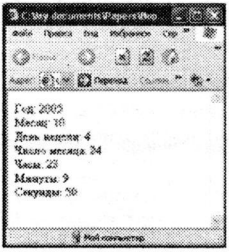



***Рис. 6.2.** Составляющие элементы даты*


Данные, показанные на рис. 6.2, были получены 24 ноября 2005 года (четверг) в 23:09:50. Вас не удивило несоответствие с некоторыми датами на рис. 6.2? На экран был выведен 10-й месяц. Это не ошибка компьютера. Просто компьютер считает месяцы с нуля, т.е. для января номер месяца будет `0`. Точно так же с нулевой отметки ведется отсчет дней недели. Четверг был выведен под номером `4` по той причине, что первым днем недели у англичан считается воскресенье.

Если вы хотите отобразить на странице текущий день недели или месяц словами, вам придется немножко потрудиться и создать два массива, как в листинге 6.1.

**Листинг 6.1. Преобразование даты в слова**

```
<BODY>
<P NAME='clock' ID='clock' ALIGN='center'></P>
<SCRIPT>
days = new Array("воскресенье", "понедельник", "вторник",
"среда", "четверг", "пятница", "суббота");
months = new Array("января", "февраля", "марта", "апреля",
"мая", "июня", "июля", "августа", "сентября", "октября",
"ноября", "декабря");
today = new Date();
```

```
clock.innerHTML = today.getHours() + ((today.getHours()<10) ?
"0:" : ":") + ((today.getMinutes()<10) ? "0" : "") + today.
getMinutes() + "<BR>";
clock.innerHTML += (days[today.getDay()] + "<BR>");
clock.innerHTML += (today.getDate() + " " + months[today.
getMonth()] + " " + today.getYear());
</SCRIPT>
</BODY>
```

Результат выполнения сценария показан на рис. 6.3.

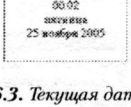
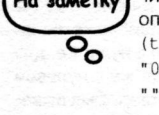



***Рис. 6.3.** Текущая дата выведена словами и цифрами*

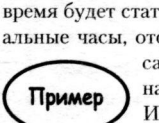



Чтобы часы и минуты всегда выводились двухзначным числом, мы воспользовались знакомым вам условным оператором (см. главу 5):

```
(today.getHours() + ((today.getHours()<10) ?
"0:" : ":") + ((today.getMinutes()<10) ? "0" :
"") + today.getMinutes() + "<BR>")
```

## Часы на Web-странице

Вы научились извлекать системное время компьютера и показывать его на экране. Этим можно воспользоваться для отображения на Web-странице даты и времени ее открытия, если такая информация важна для понимания содержимого Web-страницы. Но показанное время будет статичным. Если вы хотите показать на странице виртуальные часы, отображающие реальное время, то необходимо написать сценарий для регулярного обновления строки на экране.




Имея понятие о функциях, бесконечных циклах и таймере задержки выполнения функции, вы без труда выполните эту задачу.

1. В коде HTML Web-страницы создайте пустой именованный абзац, например: `<P NAME='clock' ID='clock'><P>`. Поместите абзац в той части страницы, где должны отображаться часы.

2. Преобразуйте код листинга 6.1 в пользовательскую функцию с именем, например: `show_clock()`. (Если вы забыли, как создавать пользовательские функции, вернитесь к главе 3.) Добавьте в код функции еще одну строку:

```
setTimeout("show_clock()",60000);
```

3. В конце основного раздела Web-страницы (где-нибудь между дескрипторами `<BODY>...</BODY>`, но после дескрипторов абзаца, в котором будут выводиться часы) добавьте вызов функции `show_clock()`.

4. После загрузки Web-страницы на ней отобразятся часы, такие же, как на рис. 6.3, но теперь время будет обновляться каждую минуту.


Обратите внимание на то, что в код функции `show_clock` мы добавили вызов этой же функции с задержкой в 60 с. В результате функция `show_clock` будет выполняться каждые 60 с, вызывая саму себя. Вам может показаться, что эту задачу можно было бы решить с помощью бесконечного цикла, например так:

```
<SCRIPT>
var i = 1;
while (i)
  setTimeout("show_clock()",60000);
</SCRIPT>
```

Даже не пытайтесь это сделать, поскольку данный бесконечный цикл приведет к зависанию приложения обозревателя.

## Вычисление даты и времени

Несмотря на то что объект `Date` по умолчанию возвращает строку текста (см. рис. 6.1), данные в нем хранятся в другом формате — целое число, представляющее количество миллисекунд, прошедших от *базовой даты*, — полуночи 1 января 1970 года. Это число называется примитивным значением объекта `Date`. Для его возвращения используется метод `valueOf()`.

Возможность получить текущую дату одним числом позволяет нам производить вычисления времени между событиями.


Для примера давайте слегка преобразуем код функции show_clock, с которой мы работали в предыдущем разделе (листинг 6.2).

**Листинг 6.2. Определение времени между двумя вызовами функции**

```
<HTML>
<HEAD>
<SCRIPT>
function show_clock(last_time) {
  // код листинга 6.1.
  alert(today.valueOf() - last_time.valueOf());
  setTimeout("show_clock(today)",60000);}
</SCRIPT>
</HEAD>
<BODY>
<P NAME='clock' ID='clock' ALIGN='center'></P>
<SCRIPT>
show_clock(new Date());
</SCRIPT>
</BODY></HTML>
```

Были внесены следующие изменения. В определение функции show_clock добавлен аргумент last_time, принимающий объект Date. Код функции остался таким же, как в листинге 6.1, за исключением двух последних строк. В предпоследней строке сравниваются примитивные значения объекта Date, переданного с аргументом, и переменной today, созданной в теле функции. Результат выводится в стандартном окне сообщения. Затем выполняется вызов этой же функции show_clock с задержкой в 60 с и с передачей переменной today в качестве аргумента. Мы можем ожидать, что окно сообщения покажет нам интервал между вызовами функции в 60 000 миллисекунд. В действительности интервал будет больше (рис. 6.4), поскольку какое-то время будет затрачено на выполнение самой функции.

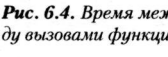



***Рис. 6.4.** Время между вызовами функции*

Мы можем воспользоваться этим подходом, чтобы разместить на персональной Web-странице напоминание о том, сколько дней осталось до какого-нибудь знаменательного события: Нового года, дня рождения, очередного чемпионата мира по футболу и т.п. Но чтобы текст напоминания был понятен, значение в миллисекундах нужно преобразовать в дни, часы или минуты. Для этого полученное значение нужно разделить на число миллисекунд в желаемой единице времени:


- сутки — 86 400 000 мс;
- минута — 60 000 мс;
- час — 360 000 мс;
- секунда — 1000 мс.

Рассмотрим пример в листинге 6.3.

**Листинг 6.3. Определение числа дней до знаменательной даты**

```
<HTML>
<HEAD>
<TITLE>Скоро Новый год!</TITLE>
</HEAD>
<BODY>
<SCRIPT>
var NewYear = new Date("January 1, 2006");
var today = new Date();
var day_left = Math.ceil((NewYear.valueOf()-today.valueOf())/
                         86400000)
if (day_left == 1)
  document.write("<H1>До Нового года остался 1 день!!!</H1>");
else
  document.write("<H1>До Нового года осталось ",day_
      left,(day_left<5 ? " дня":" дней"),"!!!</H1>")
</SCRIPT>
</BODY></HTML>
```

Сценарий Web-страницы очень простой. Создается объект `Date` с датой Нового года. Затем возвращается текущая дата. Находим разницу между примитивными значениями этих переменных, делим полученное значение на число миллисекунд в сутках и округляем до ближайшего большего целого числа методом `Math.ceil`. Текст заголовка выбирается с помощью конструкции операторов `if-else` и с помощью условного оператора. Результат показан на рис. 6.5.

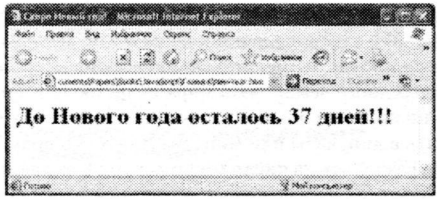



***Рис. 6.5.*** *Сколько дней осталось до Нового года?*

# Графика и эффекты анимации

Занимаясь разработкой Web-страниц, вы можете проявить себя не только как писатель, но и как режиссер. Мир Интернет динамичен и полон движения. Эффекты анимации позволят привлечь внимание посетителей Web-страницы и выделить наиболее важные информационные блоки. Но будьте сдержанны, применяя спецэффекты. Анимация должна привлекать, но не отвлекать внимание посетителей.

Обычно для анимации используют графические объекты — изображения в формате GIF, которые сменяют друг друга с определенной частотой, часто по циклу. В результате получается эффект движения или изменения графического изображения. К анимации также относится динамическое изменение или перемещение текста — эффект бегущей строки. На Web-страницах можно показывать целые фильмы, но эта возможность уже относится не к анимации, а к созданию встроенных объектов мультимедиа, о чем речь пойдет в последующих разделах этой главы.

## Бегущая строка

Интересный способ привлечения внимания к короткому, но емкому сообщению состоит в использовании бегущей строки. Бегущая строка — это анимационный эффект, при котором строка текста перемещается по странице один раз, несколько раз, или постоянно.

### Объект MARQUEE

Для создания бегущей строки используется специальный дескриптор `<MARQUEE>`. Атрибуты этого дескриптора контролируют параметры перемещения текста.

- **BEHAVIOR** — способ выполнения эффекта бегущей строки:
  - `scroll` — текст исчезает за краем Web-страницы;
  - `slide` — после выполнения заданного числа циклов текст остается у левого или правого поля Web-страницы;
  - `alternate` — направление перемещения текста в строке меняется на противоположное после выполнения каждого цикла, по завершении последнего цикла текст остается у края страницы.
- **BGCOLOR** — цвет фона бегущей строки (см. табл. 2.3).
- **DIRECTION** — направление перемещения текста:
  - `left` — влево;
  - `right` — вправо.
- **HEIGHT** — высота бегущей строки.
- **HSPACE** — отступ в пикселях текста бегущей строки от левого и правого полей Web-страницы.
- **LOOP** — число показов текста в строке:
  - `?` — целочисленное значение, указывающее число повторов;
  - `infinite` — бесконечное повторение эффекта по циклу.
- **SCROLLAMOUNT** — смещение текста в пикселях за один шаг.
- **SCROLLDELAY** — временной промежуток между смещениями в миллисекундах, по умолчанию 60 мс. Используйте этот атрибут, чтобы замедлить перемещение текста в строке, так как значения менее 60 мс будут игнорироваться обозревателем. Чтобы ускорить перемещение текста, переустановите минимальное значение смещения текста в атрибуте `TRUESPEED`.
- **VSPACE** — отступ в пикселях по вертикали от текста до рамки бегущей строки.
- **TRUESPEED** — минимальное значение смещения текста, по умолчанию 60 мс.
- **WIDTH** — ширина бегущей строки в пикселях.


Текст, заключенный между дескрипторами `<MARQUEE>...</MARQUEE>`, можно форматировать так же, как обычный текст Web-страницы.


Объект бегущей строки, созданный с помощью дескрипторов <MARQUEE>...</MARQUEE>, доступен для динамического редактирования с помощью сценариев. Посмотрите код листинга 6.4.

**Листинг 6.4. Управление свойствами бегущей строки с помощью кнопок формы**

```
<HTML>
<HEAD>
<TITLE>Бегущая строка</TITLE>
<SCRIPT>
function set_marquee(text,color) {
  marquee.innerText = text;
  marquee.bgColor = color;
}
</SCRIPT>
</HEAD>
<BODY>
<MARQUEE NAME='marquee' ID='marquee'></MARQUEE>
<FORM>
<P>
<INPUT TYPE='button' VALUE="Красная кнопка" ONCLICK="set_
marquee('Текст красной кнопки','red');">
<INPUT TYPE='button' VALUE="Желтая кнопка" ONCLICK="set_
marquee('Текст желтой кнопки','yellow');">
<INPUT TYPE='button' VALUE="Зеленая кнопка" ONCLICK="set_
marquee('Текст зеленой кнопки','green');">
</P>
</FORM>
</BODY></HTML>
```

Web-страница содержит именованный объект бегущей строки и форму с тремя кнопками. Щелчок на кнопке выводит текст в бегущей строке и изменяет цвет фона. Все кнопки вызывают на выполнение функцию set_marquee, определенную в сценарии в заголовке Web-страницы. Эта функция принимает в аргументах текст и значение цвета. В строках функции происходит присвоение полученных значений соответствующим атрибутам объекта бегущей строки. Результат выполнения сценария показан на рис. 6.6.

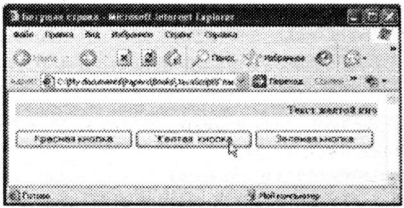



***Рис. 6.6.*** *Вывод текста в бегущей строке с помощью кнопок*


Эффект бегущей строки поддерживается только в обозревателе Internet Explorer. В других обозревателях текст будет отображаться как обычная строка.

## Бегущий текст в строке состояния

Показ сообщений в строке состояния — это не самый лучший способ информирования посетителей Web-страницы, поскольку часто пользователи Интернет не обращают ни малейшего внимания на сообщения в этой части окна обозревателя. Но движущийся текст безусловно привлечет внимание.


Чтобы заставить текст двигаться, введите код Web-страницы, показанный в листинге 6.5.

**Листинг 6.5. Бегущий текст в строке состояния**

```
<HTML>
<HEAD>
<TITLE>Бегущая строка</TITLE>
<SCRIPT>
function set_phrase(tVal) {
  if (tVal) {
    phrase = "..." + tVal + "...";
    set_status();
  }
  else
    stopscroll();
}

function set_status() {
  len = phrase.length;
  first = phrase.substring(0,1);
```

```
  rest = phrase.substring(1,len);
  phrase = rest + first;
  self.status = phrase;
  timerID = setTimeout("set_status()",100)
}

function stopscroll() {
  clearTimeout(timerID);
  self.status = "Бегущий текст остановлен";
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE='text' NAME='input' ID='input'
STYLE="width:'500'"><BR>
<INPUT TYPE='button' VALUE='Запустить бегущую строку '
ONCLICK="set_phrase(input.value)"><BR>
<INPUT TYPE='button' VALUE='Остановить бегущую строку'
ONCLICK="stopscroll()">
</FORM>
</BODY></HTML>
```

Полученная Web-страница (рис. 6.7) содержит текстовое поле и две кнопки.

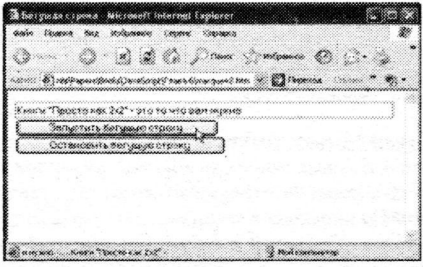


***Рис. 6.7.*** *Бегущий текст в строке состояния*

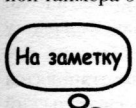
Первая кнопка, **Запустить бегущую строку**, извлекает строку из текстового поля и передает ее функции `set_phrase` (код функций находится в сценарии в области заголовка Web-страницы). Данная

функция добавляет к строке начинающие и замыкающие символы точек, чтобы можно было отделить окончание одной строки от начала следующей в строке состояния. Далее следует вызов функции `set_status`. Функция `set_status` работает так.

1. С помощью метода `substring` первый символ извлекается из строки и помещается в ее конец.
2. Полученная строка передается в строку состояния командой `self.status = phrase;`.
3. Вновь происходит вызов функции `set_status` с установкой таймера на 100 мс. Таймер присваивается переменной `timerID`.

Таким образом, эффект бегущего текста достигается за счет того, что функция `set_status` вызывается рекурсивно по бесконечному циклу и на каждом этапе переносит первый символ в конец строки. С помощью таймера устанавливается скорость перемещения строки по полю.

Вторая кнопка, **Остановить бегущую строку**, прекращает рекурсию функции `set_status` путем очистки таймера. Для этого для переменной таймера `timerID` вызывается метод `clearTimeout`.




На примере листинга 6.5 вы познакомились с новыми методами и свойствами таймера: таймер можно присвоить переменной, можно прервать выполнение функции по таймеру с помощью метода `clearTimeout`. На Web-странице можно установить несколько таймеров с разными именами, контролирующих различные процессы.

## Управление графическими объектами

Рисунки на Web-страницы можно добавлять несколькими способами:

- с помощью дескриптора `<IMG>` (см. главу 2);
- с помощью дескриптора `<OBJECT>` (см. раздел «Создание объектов мультимедиа с помощью дескриптора <OBJECT>», далее в этой главе);
- с помощью конструктора графического объекта `Image()`.

## Конструктор графических объектов

Объект графического изображения, так же, как другие объекты, создается с помощью ключевого слова `new` и конструктора графических объектов `Image()`:

```
MyImage = new Image(ширина,высота);
```

Ширина и высота изображения в пикселях задаются целочисленными значениями. Но эти аргументы необязательны и могут быть пропущены. В этом случае создается пустой объект графического изображения. Далее следует связать с этим объектом выбранный графический файл:

```
MyImage.src = "myimage.jpg"
```

Как и в случае с дескриптором `<IMG>`, графический файл рисунка устанавливается в свойстве `src`. Объект изображения имеет еще одно весьма полезное свойство — `complete`. Это свойство принимает значение `true` после завершения выгрузки изображения на Web-страницу.

Есть еще одно отличие графического объекта, созданного с помощью конструктора, от объекта, добавленного в коде HTML. Каждый рисунок, добавленный с помощью дескриптора `<IMG>`, автоматически становится элементом стандартного массива `images`. Так, к первому рисунку на странице можно обратиться следующим образом: `document.images[0]`. Объекты, созданные в сценариях, не становятся элементами массива `images`. Но мы можем создать пользовательский массив графических объектов:

```
pic0 = new Image();
pic0.src = 'reddart.gif';
pic1 = new Image();
pic1.src = 'bluedart.gif';
…
// Создаем массив рисунков
picA = new Array(pic0,pic1,pic2,pic3,…);
```


Воспользуемся массивом графических изображений для создания простейшего эффекта анимации. Поместим на Web-страницу ряд цветных стрелок, которые будут двигаться в направлении справа налево. Для этого введем код листинга 6.6.

**Листинг 6.6. Бегущие огни**

```
<HTML>
<HEAD>
<TITLE>Гирлянда</TITLE>
<SCRIPT>
// Загрузка рисунками массива picA
function loadimages() {
  pic0 = new Image();
  pic0.src = 'images\\reddart.gif';
  pic1 = new Image();
  pic1.src = 'images\\bluedart.gif';
  pic2 = new Image();
  pic2.src = 'images\\grndart.gif';
  pic3 = new Image();
  pic3.src = 'images\\yelldart.gif';
  picA = new Array(pic0,pic1,pic2,pic3);
}
// Программа анимации
function animate() {
  // сброс текущего таймера
  if (going)
    clearTimeout(timer);
  for (loop=0;loop<4;loop++) {
    index = (index<3) ? ++index : 0;
    document.images[index].src = picA[loop].src;
  }
  index--;
  timer = setTimeout("animate()",200);
}
// Все функции приостанавливаются, пока не будет
// загружен на Web-страницу последний рисунок
function checkload() {
  if (pic3.complete == false)
    timer = setTimeout("checkload()",100);
}

</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
loadimages();
index = 0;
```

```
going = false;
timer = null;
checkload();
</SCRIPT>
<P><IMG SRC='images\reddart.gif'>
<IMG SRC='images\yelldart.gif'>
<IMG SRC='images\grndart.gif'>
<IMG SRC='images\bluedart.gif'>
</P>
<FORM>
<INPUT TYPE='button' VALUE="Включить" ONCLICK="going = true;
animate();">
<INPUT TYPE='button' VALUE="Выключить" ONCLICK="clearTimeout
(timer);">
</FORM>
</BODY></HTML>
```

Сценарий может показаться вам сложным, поэтому давайте рассмотрим его подробно.

Результат выполнения сценария показан на рис. 6.8.

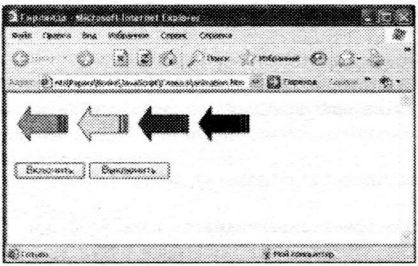


***Рис. 6.8.*** *Анимация «Бегущие огни»*

Четыре разноцветные стрелки в окне обозревателя постоянно сменяют друг друга в направлении справа налево. Бесконечное чередование стрелок обеспечивается следующим циклом в теле функции `animate`:

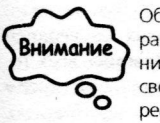
```
for (loop=0;loop<4;loop++) {
  index = (index<3) ? ++index : 0;
  document.images[index].src = picA[loop].src;
}
index--;
```

Переменная index определяет элементы стандартного массива images, тогда как переменная loop используется в качестве индекса массива графических объектов picA, созданного ранее в функции loadimages. В строке document.images[index].src = picA[loop].src; происходит копирование графического объекта из массива picA в элемент массива images, представленного соответствующим дескриптором <IMG> в коде HTML. В результате графический объект становится видимым в окне обозревателя в том месте, где в коде HTML находится соответствующий дескриптор <IMG>. Благодаря строке кода index = (index<3) ? ++index : 0; графические объекты выносятся на экран последовательно. В самом деле, значение переменной index увеличивается на единицу при каждом выполнении цикла, пока не достигнет значения 3, после чего index принимает значение 0. Таким образом, index может быть равным 0, 1, 2, 3. Благодаря оператору index-- за телом цикла при каждом новом вызове функции animate начальное значение переменной index смещается с шагом в единицу.


Вам может быть не понятно, зачем понадобился сброс таймера:

```
if (going)
  clearTimeout(timer);
```

в начале функции animate. Если вы удалите эти строки, то увидите странный эффект: каждый раз после щелчка на кнопке **Включить** скорость мигания цветных стрелок будет возрастать вдвое. Дело в том, что если не сбросить текущий таймер, то вызов функции animate щелчком на кнопке будет запускать параллельный процесс, в результате чего скорость анимации удвоится. Чтобы остановить анимацию, вам придется затем столько же раз щелкнуть на кнопке **Выключить**. Впрочем, получится довольно интересный эффект, который может пригодиться вам на практике.




Обратите также внимание на функцию checkload, которая проверяет, был ли загружен на Web-страницу последний графический объект. При испытании сценария на своем компьютере эта функция не имеет значения. Но на реальной Web-странице отсутствие проверки может привести к ошибке из-за того, что сценарий обратится к графическому объекту, которого на странице пока еще нет, поскольку он не успел выгрузиться по медленной сети.

## Назначение гиперссылок областям изображения

В главе 2 вы узнали, как с помощью дескриптора <A> преобразовать фрагмент текста или рисунок в гиперссылку. Но этим ваши возможности не исчерпываются. Интересный прием состоит в разделении изображения на области, каждой из которых можно назначить свою гиперссылку. Такой прием удобно использовать, например, при создании карты Web-узла для навигации по его страницам. Предположим, что вы создаете Web-узел с описанием технических характеристик автомобиля. Можно спланировать узел следующим образом. На начальной странице поместите изображение автомобиля и выполните разметку изображения таким образом, чтобы щелчок мыши на узле или агрегате открывал Web-страницу с соответствующим описанием.


Выполните следующие действия.

1. Подберите или создайте необходимый рисунок.
2. Добавьте рисунок на Web-страницу с помощью дескриптора <IMG>.
3. В любом месте HTML-кода страницы в разделе <BODY> создайте элемент разметки страницы с уникальным именем: <MAP NAME='*имя_разметчика*'>...</MAP>.
4. В дескрипторе <IMG> рисунка присвойте атрибуту USEMAP имя элемента разметки. (Имя элемента разметки следует после символа #.)
5. Между открывающим и закрывающим дескрипторами элемента разметки рисунка добавьте области с помощью дескрипторов <AREA> и установите следующие атрибуты областей:
   - SHAPE — присвойте стандартные значения формы области rect (прямоугольник), circle (окружность), polygon (многоугольник) и default (незанятая часть рисунка);
   - COORDS — пары координат $x$ и $y$ в пикселях для верхней левой и нижней правой точек прямоугольника, всех точек многоугольника и центральной точки с радиусом окружности;
   - HREF — URL-адрес гиперссылки;
   - TITLE — текст экранной подсказки.


Чтобы определить координаты области, откройте изображение в графическом редакторе. Большинство приложений указывают в окне **Info** (Информация) или в строке состояния текущее положение курсора – расстояние в пикселях от верхнего левого угла изображения. Именно эти координаты необходимо присвоить атрибутам `COORDS` дескрипторов `<AREA>`. Если на странице размеры рисунка изменены с помощью атрибутов `WIDTH` и `HEIGHT`, вам придется пропорционально пересчитать координаты областей.

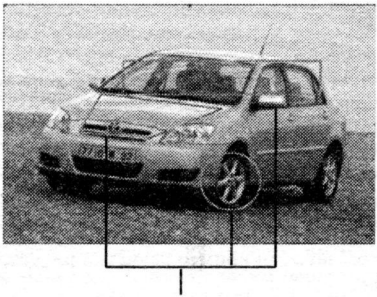
В листинге 6.7 показан пример разметки рисунка для создания карты Web-узла.

**Листинг 6.7. Назначение гиперссылок областям рисунка**

```
<HTML>
<HEAD>
<TITLE>Hyundai</TITLE>
</HEAD>
<BODY>
<H2>Модель 7710W92</H2>
<IMG SRC='images/7710W92.jpg' BORDER='0' WIDTH='640'
HEIGHT='425' USEMAP='#imagemap'>
<MAP NAME='imagemap'>
     <AREA SHAPE='circle' COORDS='389, 312, 52'
HREF='chassis.htm' TITLE='Ходовая'>
     <AREA SHAPE='polygon' COORDS='374, 184, 168, 158, 130,
          225, 248, 237' HREF='engine.htm' TITLE='Двигатель'>
     <AREA SHAPE='rect' COORDS='205, 99, 544, 160'
HREF='salon.htm' TITLE='Салон'>
</MAP>
</BODY></HTML>
```

Расположение областей на рисунке можно посмотреть в профессиональном редакторе Web-страниц Microsoft FrontPage, как показано на рис. 6.9. А внешний вид страницы в окне обозревателя приведен на рис. 6.10.

В листинге 6.7 вы познакомились с постоянными областями рисунка. Ниже в разделе «Создание объектов мультимедиа с помощью дескриптора <OBJECT>» вы узнаете о возможности динамического создания и изменения областей с помощью сценариев.



Прямоугольная, полигональная и округлая области рисунка, созданные с помощью дескрипторов <AREA>

***Рис. 6.9.*** *Области разметки рисунка*

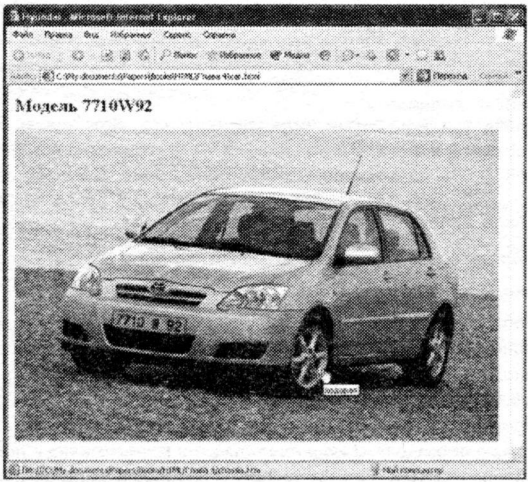
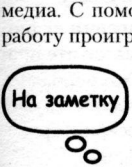



***Рис. 6.10.*** *Внешний вид рисунка с областями в окне обозревателя Internet Explorer*

# Воспроизведение звуковых и видеоклипов на Web-странице

В языке JavaScript нет никаких функций для непосредственного контроля за воспроизведением аудио- или видеофайлов, но предусмотрены функции управления стандартными проигрывателями мультимедиа. С помощью сценариев можно настроить требуемым образом работу проигрывателей, встроенных в приложение обозревателя.




Возможности обозревателя можно расширить за счет установки на компьютере дополнительной программы (*плагина*). Например, для просмотра файлов в формате PDF используется Acrobat Reader, а для воспроизведения звуковых файлов – LiveAudio. Указанные программы настолько широко распространены, что многие считают их обязательными составными частями обозревателей. На самом деле это не так, в результате чего объекты мультимедиа, добавленные вами на Web-страницу, могут оказаться недоступными для некоторых пользователей.

## Использование внедренных проигрывателей

Для внедрения проигрывателей и файлов мультимедиа в документ HTML применяется дескриптор `<EMBED>`. Обозреватель различает формат внедренного файла и добавляет на страницу элементы управления, необходимые для контроля за воспроизведением файла данного формата. В дескрипторе `<EMBED>` устанавливаются следующие атрибуты:

- **SRC** — имя файла и путь к файлу мультимедиа, или URL-адрес;
- **HIDDEN** — если этот атрибут установлен, элементы управления воспроизведением файла скрыты;
- **WIDTH** — ширина рамки для элементов управления воспроизведением файла;
- **HEIGHT** — высота рамки для элементов управления воспроизведением файла;
- **ALIGN** — параметры выравнивания рамки с элементами управления в окне обозревателя:

- left — влево;
- right — вправо;
- center — по центру;

- **AUTOSTART** — устанавливает автоматическое воспроизведение файла при открытии Web-страницы;
- **LOOP** — устанавливает режим воспроизведения по циклу.

Атрибут HIDDEN устанавливается обычно при внедрении аудиофайла для фонового сопровождения. Например, если вы хотите просто добавить звуковое сопровождение из файла background.mid, которое будет воспроизводиться автоматически сразу после загрузки Web-страницы, введите в разделе <BODY> следующий дескриптор:


```
<EMBED SRC='background.mid' HIDDEN=TRUE>
```

Ниже показаны примеры внедрения проигрывателей аудио- и видеофайлов. Введите в раздел <BODY> документа HTML код листинга 6.8.

**Листинг 6.8. Внедрение проигрывателей аудио- и видеофайлов**

```
<!- Добавляем фоновый звук -->
<EMBED SRC='bgsound.mid' HIDDEN AUTOSTART='true' LOOP='true'>

<!- Добавляем звуковой файл -->
<P>Прослушайте запись моего концерта</P>
<EMBED SRC='MyMusic.mid' WIDTH='200' HEIGHT='25'
AUTOSTART='false'>
<P> </P>
<P> </P>
<!- Добавляем видео -->
<P>Время</P>
<EMBED SRC='clock.avi' WIDTH='200' HEIGHT='200'
ALIGN='center' AUTOSTART='true' LOOP='true'>
```

Прежде чем открывать полученный файл в обозревателе, выполните следующие действия.

1. Скопируйте в папку документа необходимые аудио- и видеофайлы. (Образцы файлов вы можете найти в папке Windows на вашем компьютере или в Интернет.)
2. Измените значения атрибутов <EMBED SRC='?'> в соответствии с именами ваших файлов или переименуйте файлы.

3. Откройте Web-страницу в обозревателе. Она будет выглядеть примерно так, как показано на рис. 6.11.

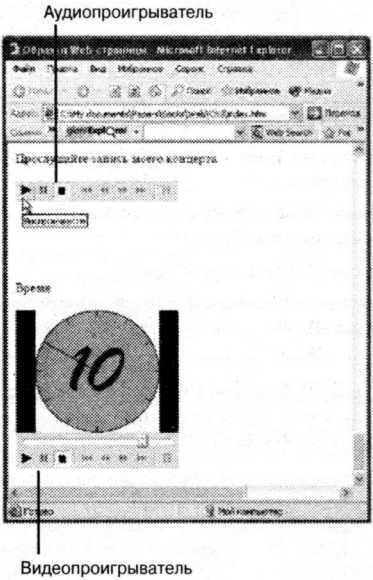



***Рис. 6.11.*** *Внедренные проигрыватели аудио- и видеофайлов*

Файл `bgsound.mid` автоматически начнет воспроизводиться по циклу в фоновом режиме. Автоматически по циклу также будет воспроизводиться видеоклип. Чтобы прослушать файл `MyMusic.mid`, щелкните на кнопке **Воспроизводить** в соответствующей группе элементов управления, как показано на рис. 6.11.


Файлы мультимедиа не будут воспроизводиться в некоторых устаревших обозревателях или в тех приложениях, в которых данные параметры были отключены пользователями. Чтобы сообщить пользователю о том,

что страница содержит недоступный для него объект мультимедиа, введите альтернативный текст вроде следующего:

```
<NOEMBED>Ваш обозреватель не поддерживает воспроизведение внедренных файлов мультимедиа</NOEMBED>
```

## Управление звуковыми файлами

Проигрыватель LiveAudio поддерживает воспроизведение файлов следующих форматов:

- **AU** — формат, разработанный компанией Sun для использования в программах на языке Java;
- **AIFF** — формат компьютеров Apple;
- **WAV** — широко распространенный формат Windows для записи звуков и речи;
- **MIDI** — стандартный формат для сохранения цифровой музыки.

Если на вашем компьютере PC установлен микрофон и звукозаписывающее программное обеспечение, вы можете записать свои комментарии в файлах формата WAV и добавить их на Web-страницу.


Звуковые файлы обычно занимают много места, что существенно замедлит выгрузку Web-страницы по Интернет. В среднем 1 с звука в формате WAV занимает 22 Кбайт. Файл такого же размера в формате MIDI будет соответствовать нескольким минутам звукозаписи.

## Методы управления воспроизведением звука

Для взаимодействия с внедренным на Web-странице проигрывателем LiveAudio в JavaScript используются следующие методы:

- **play()** — запускает воспроизведение звукового файла;
- **pause()** — приостанавливает выполнение звукового файла, следующий вызов функции `play` продолжает воспроизведение с того места, где запись была остановлена;
- **stop()** — останавливает выполнение звукового файла, следующий вызов функции `play` начинает воспроизведение сначала.

Выполнение этих методов можно назначить любым событиям элементов Web-страницы. Так, в следующем примере мы создаем кнопки для управления скрытым проигрывателем LiveAudio:

```
<EMBED ID='sound' NAME='sound' SRC='bgsound.wav' HIDDEN
AUTOSTART='false'>
<FORM><P>
<INPUT TYPE='button' VALUE='Воспроизвести' ONCLICK='sound.
play();'>
<INPUT TYPE='button' VALUE='Пауза' ONCLICK='sound.pause();'>
<INPUT TYPE='button' VALUE='Остановить' ONCLICK='sound.
stop();'>
</P></FORM>
```

Данные методы могут быть присвоены любым другим событиям, например щелчку на гиперссылке, открытию или закрытию документа и пр., что открывает перед нами неограниченные возможности по управлению звуковым сопровождением в ходе работы пользователя с Web-страницей.

## Массив внедренных объектов мультимедиа

На Web-странице можно использовать много разных звуков и видеороликов, добавленных с помощью дескриптора <EMBED>. Все объекты мультимедиа, созданные таким способом, автоматически добавляются в стандартный массив embeds в том порядке, в каком дескрипторы <EMBED> следуют в коде HTML (подробнее о стандартных массивах обозревателя рассказывалось в главе 4). К каждому из объектов мы можем обратиться по индексу в массиве embeds и вызвать для него метод воспроизведения, паузы или остановки.


Предположим, что на Web-странице используется семь звуковых файлов. Давайте создадим кнопки для выборочного воспроизведения файлов, воспроизведения всех мелодий по циклу и кнопку случайного выбора. Рассмотрим листинг 6.9.

**Листинг 6.9. Использование массива внедренных звуков**

```
<HTML>
<HEAD>
<TITLE>Гамма</TITLE>
<SCRIPT>
note = 0;
function play_scale() {
```

```
  document.embeds[note].play();
  note++;
  if (note < document.embeds.length)
    setTimeout("play_scale()",2000);
  else
    note = 0;
}
function random_play() {
  i = Math.floor(Math.random()*document.embeds.length);
  document.embeds[i].play();
}
</SCRIPT>
</HEAD>
<BODY>
<!- Добавляем фоновый звук -->
<EMBED SRC='sounds\do.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\re.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\mi.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\fa.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\sol.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\la.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\si.wav' HIDDEN AUTOSTART='false'>
<FORM>
<P>
<INPUT TYPE='button' VALUE=' До  ' ONCLICK="document.
embeds[0].play();">
<INPUT TYPE='button' VALUE=' Ре  ' ONCLICK="document.
embeds[1].play();">
<INPUT TYPE='button' VALUE=' Ми  ' ONCLICK="document.
embeds[2].play();">
<INPUT TYPE='button' VALUE=' Фа  ' ONCLICK="document.
embeds[3].play();">
<INPUT TYPE='button' VALUE='Соль' ONCLICK="document.
embeds[4].play();">
<INPUT TYPE='button' VALUE=' Ля  ' ONCLICK="document.
embeds[5].play();">
<INPUT TYPE='button' VALUE=' Си  ' ONCLICK="document.
embeds[6].play();"><BR>
<INPUT TYPE='button' VALUE='Играть гамму' ONCLICK='play_
scale();'>
<INPUT TYPE='button' VALUE='Случайный выбор' ONCLICK='random_
play();'>
```

```
</P>
</FORM>
</BODY></HTML>
```

Наша Web-страница содержит набор командных кнопок, как показано на рис. 6.12.

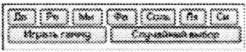
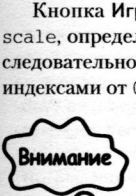


***Рис. 6.12.** Кнопки для воспроизведения звуковых файлов из массива* `embeds`

Кнопки верхнего ряда от **До** до **Си** воспроизводят отдельные звуковые файлы, хранящиеся в массиве `document.embeds`. Все очень просто: каждая кнопка обращается к своему элементу массива и вызывает для него метод `play`.

Кнопка **Играть гамму** вызывает на выполнение функцию `play_scale`, определенную в сценарии в разделе заголовка. Функция последовательно вызывает на выполнение элементы массива `embeds` с индексами от `0` до `document.embeds.length - 1`.




Казалось бы логичным для обращения к элементам массива воспользоваться циклом `for`:

```
for (note=0;note<document.embeds.length;note++)
  document.embeds[note].play();
```

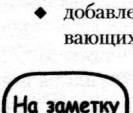
Но вместо последовательного выполнения звуковых файлов мы получим какофонию звуков. Вызов следующего файла на воспроизведение будет происходить в цикле значительно быстрее, чем первый файл успеет завершиться. Чтобы выделить достаточно времени на исполнение каждого звукового клипа, воспользуйтесь методом `setTimeout`.

Следующая кнопка, **Случайный выбор**, воспроизводит один из звуковых файлов. Для случайного выбора элементов массива используется рандомайзер `Math.floor(Math.random()*document.embeds.length)`, который возвращает целые числа в диапазоне от `0` до `document.embeds.length - 1`.

# Создание объектов мультимедиа с помощью дескриптора <OBJECT>

Для работы с объектами в код HTML был добавлен новый парный дескриптор <OBJECT></OBJECT>, который, благодаря своей универсальности, не только стал стандартом для всех обозревателей, но и вытеснил из языка HTML некоторые более ранние дескрипторы. Дескриптор <OBJECT> может применяться в следующих целях:

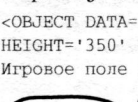
- добавление аплетов (небольшие программы, которые автоматически выгружаются по сети вместе с Web-страницей и выполняются на машине пользователя);
- добавление изображений в любом формате, поддерживаемом обозревателем;
- добавление некоторых элементов Web-страницы, например плавающих рамок.




Ранее для добавления аплетов использовался дескриптор <APPLET>. Недостатком данного дескриптора было то, что он поддерживал только аплеты, написанные на Java. Дескриптор <OBJECT> позволяет внедрять на Web-страницу программы разных типов, включая элементы управления ActiveX, аплеты на языке Python, исполняемые файлы и пр. Дескриптор <APPLET> рассматривается как устаревший, поэтому удален из стандартов языка HTML.

Ниже показан пример добавления на Web-страницу изображения из файла JPG с помощью атрибута <OBJECT>:

```
<OBJECT DATA='images/sea.jpg' TYPE="image/jpeg" WIDTH='650'
HEIGHT='350' USEMAP='#imagemap'>
Игровое поле для игры в морской бой</OBJECT>
```




Следует учесть, что рисунки, добавленные с помощью атрибутов <OBJECT></OBJECT>, не становятся членами стандартного массива images.

Текст, заключенный между парой дескрипторов <OBJECT></OBJECT>, является альтернативным текстом и отображается на экране только в том случае, если сам объект не поддерживается обозревателем. Как

вы видите в примере выше, набор атрибутов графического объекта, добавленного с помощью дескриптора <OBJECT>, соответствует атрибутам дескриптора <IMG>. В действительности набор атрибутов <OBJECT> значительно шире, чем у какого-либо другого дескриптора HTML. Универсальность имеет свою цену: при установке атрибутов <OBJECT> следует учитывать тип добавляемого объекта. Некоторые атрибуты становятся бессмысленными, а смысл других радикально меняется в зависимости от типа объекта. Некоторые наиболее используемые атрибуты дескриптора <OBJECT> перечислены в табл. 6.1.

**Таблица 6.1. Атрибуты дескриптора <OBJECT>**

| Атрибут | Описание |
|---|---|
| **Атрибуты объявления объекта** | |
| DATA | Путь к файлу-первоисточнику. В качестве пути можно указать URL-адрес |
| TYPE | Тип объекта в соответствии с классификацией Internet Media Types (типы данных Интернет). Вот некоторые примеры типов объектов:<br>"image/jpg" — изображение в формате JPG;<br>"image/gif" — изображение в формате GIF;<br>"application/x-webfont" — шрифт Web-страницы |
| CODETYPE | Язык программирования |
| DECLARE | Индикатор виртуального объекта, который используется в качестве прототипа для создания экземпляров других объектов |
| ID и NAME | Уникальные дескрипторы объекта в коде HTML, позволяющие обращаться к объекту из сценариев JavaScript |
| **Атрибуты форматирования объекта** | |
| ALIGN | Выравнивает объект в тексте Web-страницы |
| WIDTH и HEIGHT | Ширина и высота рамки объекта |
| BORDER | Прорисовка границ объекта |
| HSPACE | Размер поля слева и справа от объекта |
| VSPACE | Размер поля над и под объектом |
| STANDBY | Строка текста, отображаемая во время выполнения программы объекта |
| **Другие атрибуты** | |
| USEMAP | Имя схемы областей рисунка, созданной с помощью дескриптора <MAP> |

*Окончание табл. 6.1*

| Атрибут | Описание |
|---|---|
| SHAPES | Наличие в рисунке областей с установленными гиперссылками |

Все атрибуты объекта могут быть установлены или изменены динамически с помощью сценариев. Таким образом, даже тип объекта, созданного с помощью дескриптора <OBJECT> может быть изменен в ходе работы с Web-страницей. Например, рисунок можно заменить на выполняемый аплет или объект другого типа.


Чтобы закрепить материалы, изученные нами в этой главе, давайте создадим Web-страницу для игры в морской бой. Играть будим следующим образом.

1. На странице покажем игровое поле — фотография морской глади с линией горизонта вдалеке. Фотография сохранена в файле JPG и будет добавлена с помощью дескриптора <OBJECT>.
2. Игровое поле разбито на 45 клеток. Десять случайно выбранных клеток содержат корабли неприятеля. Границы между клетками не видны, но при наведении указателя мыши появляется всплывающая подсказка с номером текущей клетки.
3. Огонь по кораблям противника ведем с помощью мыши. Если щелчок выполнен на пустой клетке — слышен звук улетающего снаряда. Если клетка содержала корабль — слышен взрыв, число кораблей уменьшается на единицу и клетка становится пустой.
4. Программа показывает число кораблей противника, которые все еще плавают по морю, и подсчитывает число выстрелов. (Чтобы облегчить отладку программы, выведем на экран также информацию о том, в каких клетках находятся корабли. В конечной версии программы эту часть кода следует удалить.) Нужно потопить все корабли противника, выполнив минимальное число выстрелов.
5. Web-страница содержит кнопку **Начать новую игру**.

Окно программы «Морской бой» показано на рис. 6.13.

Давайте рассмотрим код HTML этой Web-страницы, приведенный в листинге 6.10.

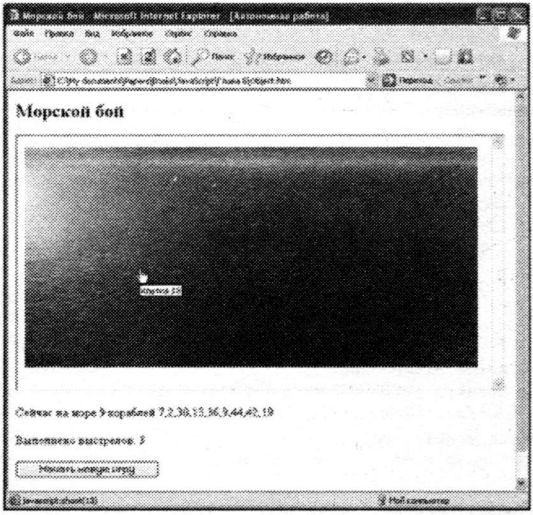



***Рис. 6.13.*** *Морской бой*

**Листинг 6.10. Код программы «Морской бой»**

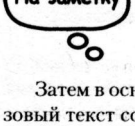
```
<HTML>
<HEAD>
<TITLE>Морской бой</TITLE>
<SCRIPT>
// Проверяем попадание в корабль и удаляем корабль из массива
function shoot(n) {
  // Функция завершается, если кораблей больше нет
  if (ships.length == 0)
    return;
  // Счетчик выстрелов
  counter++;
  count.innerText = counter;
  strShips = ships.join("$");
  if (strShips.indexOf(n) >= 0) {
    document.embeds[1].play();
    index = find_index(strShips,n);
```

```
    switch (index) {
      case 0:
        ships.shift();
        break;
      case ships.length-1:
        ships.pop();
        break;
      default:
        tmpShips = new Array();
        for (i=0;i<ships.length;i++) {
          if (i==index)
            continue
          else
            tmpShips.push(ships[i]);
        }
        ships = tmpShips;
        break;
    }
    update_statistics()
  }
  else
    document.embeds[0].play();
}
// Определяем индекс пораженного корабля
function find_index(strShips,n) {
  position = strShips.indexOf(n);
  var index = 0;
  if (position) {
    sub_list = strShips.substring(0,position);
    sub_array = sub_list.split("$");
    index = sub_array.length - 1;
  }
  return index;
}
// Обновляем текст Web-страницы
function update_statistics() {
  var num = ships.length;
  if (num == 1)
    statistics.innerText = num + " корабль";
  else if (num && num<5)
    statistics.innerText = num + " корабля";
  else if (num>4)
```

```
    statistics.innerText = num + " кораблей";
  else {
    statistics.innerText = num + " кораблей";
    alert("Вы победили!");
  }
  // Эту строку нужно удалить в окончательном
  // варианте Web-страницы
  statistics.innerText += " " + ships.toString();
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Морской бой</H2>
<OBJECT ID='imgObj' NAME='imgObj' DATA='images/sea.
jpg' TYPE="image/jpeg" WIDTH='650' HEIGHT='350'
USEMAP='#imagemap'>
Игровое поле для игры в морской бой</OBJECT>
<P>Сейчас на море <SPAN ID='statistics' NAME='statistics'></
SPAN></P>
<P>Выполнено выстрелов: <SPAN ID='count' NAME='count'>0</
SPAN></P>
<FORM><INPUT TYPE='button' VALUE='Начать новую игру'
ONCLICK='document.location.reload()'></FORM>
<MAP ID='imagemap' NAME='imagemap'></MAP>
<EMBED SRC='sounds\miss.wav' HIDDEN AUTOSTART='false'>
<EMBED SRC='sounds\hit.wav' HIDDEN AUTOSTART='false'>
<SCRIPT>
// Создаем массив кораблей
var ships = new Array(10);
var ship = 0;
while (ship < ships.length) {
  n = Math.round(Math.random()*44);
  strShips = ships.toString();
  // в массив добавляются только неповторяющиеся значения
  if (strShips.indexOf(n) >= 0)
    continue
  ships[ship] = n;
  ship++;
}
var counter = 0;
// Создаем области рисунка
var mapObj = document.getElementById("imagemap");
```

```
var n = 0;
for (x=0;x<600;x += 120) {
  for (y=30;y<300;y += 30) {
    area = document.createElement("AREA");
    area.shape = 'rect';
    area.coords = x + "," + y +"," + (x+120) + "," + (y+30);
    area.href = "javascript:shoot(" + n + ")";
    area.title = "клетка " + n;
    mapObj.appendChild(area);
    n++;
  }
}
update_statistics()
</SCRIPT>
</BODY></HTML>
```

Рисунок на Web-страницу добавляется с помощью дескриптора <OBJECT> с установкой линейных размеров рамки объекта и имени схемы областей рисунка.




Размеры рамки объекта должны быть примерно на 50 пикселей больше размеров самого рисунка. Если рамка объекта будет меньше изображения, появятся полосы прокрутки.

Затем в основном разделе Web-страницы вводятся заголовки и базовый текст сообщений о числе кораблей и выстрелов. В тексте с помощью дескрипторов <SPAN>...</SPAN> выделены именованные фрагменты, которые будут редактироваться динамически.

Кнопка **Начать новую игру** просто выполняет перезагрузку Web-страницы с помощью команды `document.location.reload()`.

Затем следует код сценария. Сначала мы создаем массив `ships` размером в 10 элементов. Массив заполняется целыми числами, случайно сгенерированными в диапазоне от 0 до 44. Обратите внимание, как с помощью оператора `if` мы проверяем, чтобы в массиве не было повторяющихся значений.

В следующей части кода сценария мы создаем области рисунка (клетки игрового поля):

```
// возвращаем объект <MAP>
var mapObj = document.getElementById("imagemap");
var n = 0;
```

```
// создаем смежные области, заполняющие все
// пространство рисунка до линии горизонта
for (x=0;x<600;x += 120) {
  for (y=30;y<300;y += 30) {
    // создаем новый элемент <AREA>
    area = document.createElement("AREA");
    // определяем атрибуты области
    area.shape = 'rect';
    area.coords = x + "," + y +"," + (x+120) + "," + (y+30);
    area.href = "javascript:shoot(" + n + ")";
    area.title = "клетка " + n;
    // добавляем область в объект <MAP>
    mapObj.appendChild(area);
    n++;
  }
}
```

Некоторые методы в данном фрагменте кода вам не знакомы. Прежде всего мы возвращаем переменную объекта схемы областей, созданного в коде HTML с помощью дескриптора `<MAP>` по значению атрибута `ID`: `document.getElementById("imagemap")`.


Метод `getElementById(ID)` можно использовать для возвращения в код сценария объекта любого элемента Web-страницы, в дескрипторе которого был установлен уникальный атрибут `ID`. Заметьте, что в данном случае не создается новый объект, а просто открывается доступ к уже существующему объекту для его динамического изменения.

Затем известным вам методом `crearteElement` создается новый объект области рисунка и устанавливаются его атрибуты (подробно создание областей рисунка мы рассматривали выше в этой главе, в разделе «Назначение гиперссылок областям изображения»). Обратите внимание на то, как мы формируем значения атрибутов из текстовых констант и переменных сценария. После завершения установки атрибутов область добавляется в родительский объект `<MAP>` с помощью метода `appendChild`.

Атрибутам `HREF` во всех областях рисунка присваивается выполнение функции `shoot`, определенной в сценарии в разделе заголовка Web-страницы. В аргументе функции передается номер текущей ячейки. Функция проверяет, содержится ли данный номер в массиве ко-

раблей ships. Если номер присутствует в массиве, то данный элемент удаляется и в тексте сообщения на Web-странице выводится новый размер массива. При этом звучит аудиофайл hit.wav, добавленный с помощью дескриптора <EMBED>. Если номер отсутствует, звучит файл miss.wav. Но в любом случае выполняется приращение глобальной переменной counter, с помощью которой мы подсчитываем выстрелы.

Как видите, для создания компьютерной игры нам потребовались всего несколько небольших сценариев. Попробуйте создать собственную игру или усовершенствовать данную. Например, не трудно сделать так, чтобы после каждого выстрела корабли перемещались по полю.

# Глава 7

# Обмен данными и сохранение информации на диске

В главе 2 мы познакомились с дескрипторами для создания форм и связанных с ними элементов управления. Чаще всего в предыдущих листингах мы использовали кнопки формы как средство вызова пользовательских функций, созданных в сценариях Web-страницы, и управления ими. Но основное назначение форм состоит в поддержании диалога между разработчиком и посетителями Web-страницы. Можно наладить непосредственный контакт с пользователями, при котором информация, введенная в поля формы, передается разработчику по электронной почте. Чаще под диалогом подразумевается взаимодействие посетителя с Web-сервером, когда записи в полях формы сохраняются в базе данных или обрабатываются специальными программами для формирования запроса к базе данных и возвращения пользователю запрошенной информации.

В любом случае диалог предполагает обмен данными. В этой главе мы рассмотрим принципы организации обмена данными с помощью сценариев JavaScript и элементов управления форм. Будут рассмотрены примеры обмена данными как между удаленными компьютерами, так и на уровне компьютера клиента между окнами обозревателя и файловой системой машины.

## Выбор и передача данных с помощью элементов управления формы

Элементы управления, с помощью которых можно вводить данные или выбирать предустановленные значения, называются *полями* формы. К полям формы относятся элементы управления следующих типов:

- текстовое поле ввода;
- поле редактора;
- флажок;
- переключатель;
- список;
- раскрывающийся список.

Общим для всех полей является то, что они могут хранить данные, вводимые или изменяемые пользователями. С помощью методов формы, о чем вы узнаете далее в этой главе, можно автоматически извлечь данные из всех полей и передать их на сервер для программной обработки или направить в виде текста по адресу электронной почты.

## Поле ввода и поле редактора

Поля ввода мы уже довольно часто использовали в листингах предыдущих глав. Такое поле создается с помощью дескриптора `<INPUT TYPE="text" VALUE="`*строка текста*`">` и может содержать строку неформатированного текста, которая либо задается по умолчанию с помощью атрибута `VALUE`, либо автоматически присваивается этому атрибуту при вводе текста пользователем.

Если необходимо предоставить пользователю возможность вводить многострочный текст, воспользуйтесь таким элементом управления, как поле редактора. Для создания этого элемента формы применяется специальный парный дескриптор `<TEXTAREA>...</TEXTAREA>`. Для поля редактора устанавливаются следующие атрибуты:

- **`NAME`** и **`ID`** — идентификация элемента на Web-странице;
- **`VALUE`** — содержимое поля, представленное текстом, заключенным между дескрипторами `<TEXTAREA>...</TEXTAREA>`;
- **`COLS`** — ширина поля в окне обозревателя в текстовых символах;
- **`ROWS`** — высота поля в строках.


Линейные размеры поля редактора в окне обозревателя, задаваемые с помощью атрибутов `ROWS` и `COLS`, не ограничивают число символов и строк, вводимых пользователем с клавиатуры. Если строка содержит больше символов, чем указано в атрибуте `COLS`, то текст авто-

матически будет перенесен на следующую строку. Если число строк превысит значение ROWS, появится полоса прокрутки. К тексту поля редактора можно обратиться как с помощью атрибута VALUE, так и с помощью свойства innerText.


Пример использования поля редактора показан в листинге 7.1.

**Листинг 7.1. Поле редактора**

```
<FORM NAME="form1" ID="form1">
<TEXTAREA NAME="ta1" ID="ta1" COLS=50 ROWS=5>
Варкалось. Хливкие шорьки
Пырялись по наве.
И хрюкотали зелюки
Как мюмзики в мове.
  (Льюис Кэрролл)</TEXTAREA>
</FORM>
<SCRIPT>
  document.write("Атрибуты текстового поля:<BR>");
  document.write("Тип элемента формы: " + document.form1.ta1.
                 type + "<BR>");
  document.write("The value of this field is: " + document.
                 form1.ta1.cols + "<BR>");
  document.write("The value of this field is: " + document.
                 form1.ta1.rows + "<BR><BR>");
  document.write("Содержимое текстового поля<BR>");
  document.write("The value of this field is: " +
          document.form1.ta1.innerText + "<BR><BR>");
  document.write("Вывод текста построчно:<BR>");
  text_strings = document.form1.ta1.value.split("\n");
  for (i=0;i<text_strings.length;i++)
    document.write(text_strings[i]+"<BR>");
</SCRIPT>
```

В этом листинге мы создаем форму с полем редактора, а затем анализируем атрибуты поля и выводим их значения на экран с помощью метода document.write. Полученное поле редактора и результат выполнения сценария показаны на рис. 7.1.

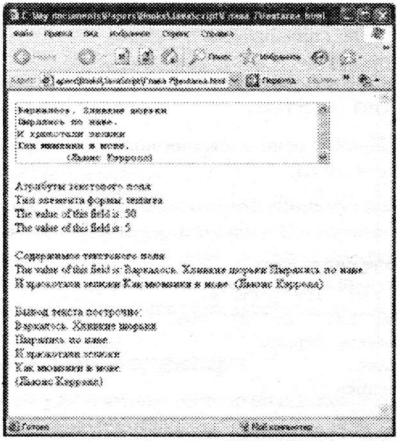



***Рис. 7.1.*** *Поле редактора и значения его атрибутов*

Вы помните, что обозреватели игнорируют все символы абзацев и табуляции в тексте Web-страницы, если они не заданы соответствующими дескрипторами. Но поле редактора чувствительно к этим служебным символам и воспроизводит их так, как они были введены в исходном тексте между дескрипторами `<TEXTAREA>...</TEXTAREA>` (см. рис. 7.1). Далее программа сценария выводит на экран тип данного элемента управления (свойство `type`) и значения атрибутов `COLS` и `ROWS`. Этот же синтаксис обращения к атрибутам поля редактора можно использовать в коде сценариев для динамического изменения внешнего вида поля на Web-странице.

Затем с помощью метода `document.write` мы выводим на экран содержимое поля. Обратите внимание, что теперь обозреватель игнорирует служебные символы и выводит текст одной строкой без абзацев и табуляции (см. рис. 7.1). Чтобы вывести текст построчно, вам следует узнать о кодировании служебных непечатных символов в тексте:

- `"\n"` — символ абзаца;
- `"\t"` — символ табуляции.

Теперь, воспользовавшись методом `split` (см. главу 4), мы можем преобразовать текст поля редактора в массив, значениями которого будут отдельные строки текста. Затем выводим элементы массива на экран с помощью цикла `for`:

```
text_strings = document.form1.ta1.value.split("\n");
for (i=0;i<text_strings.length;i++)
  document.write(text_strings[i]+"<BR>");
```


Служебные символы можно использовать при вводе текста в поле редактора с помощью сценария. Например, текст, приведенный на рис. 7.1, можно ввести следующим образом:

```
document.form1.ta1.value = "Варкалось. Хливкие
шорьки\nПырялись по наве.\nИ хрюкотали зелюки\
nКак мюмзики в мове.\n\t(Льюис Кэрролл)"
```


Поле редактора не поддерживает использование дескрипторов для форматирования текста или создания в поле каких-либо элементов Web-страницы. Все введенные дескрипторы будут показаны как обычный текст.

## Выбор параметров с помощью переключателей и списков

В главе 2 вы узнали, как создавать в форме флажки и переключатели с помощью дескрипторов `<INPUT TYPE='checkbox' NAME='`*имя*`'>` и `<INPUT TYPE='radio' NAME='`*имя*`'>` соответственно. Как вы помните, у каждого флажка может быть уникальное имя, но можно создать и группу флажков под одним именем. Переключатели всегда образуют группу, в которой выбранным может быть только один переключатель. Выбор другого переключателя в группе автоматически сбрасывает предыдущий выбор. В группе же флажков возможна любая комбинация выбранных и сброшенных элементов группы.

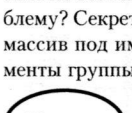
Данные элементы управления формы дают возможность пользователям сделать свой выбор между двумя (индивидуальный флажок) или несколькими независимыми (группа флажков) или альтернативными (группа переключателей) значениями. Теперь нам нужно научиться анализировать значения флажков и переключателей с помощью сценариев.

## Обращение к флажкам и переключателям

С индивидуальными флажками все очень просто. К такому флажку можно обратиться по имени следующим образом:

```
document.имя_формы.имя_флажка.атрибут
```

Но элементы группы не имеют индивидуальных имен, а группа сама по себе не содержит никаких атрибутов. Как же решить эту проблему? Секрет состоит в том, что обозреватель автоматически создает массив под именем группы переключателей или флажков. Все элементы группы добавляются в этот массив в порядке их определения в коде HTML.




Так, в листинге 7.2 в форме `form1` создаются два массива — `color`, содержащий в себе четыре объекта переключателей, и `fruit` с тремя объектами флажков.

**Листинг 7.2. Создание в форме группы переключателей и группы флажков**

```
<FORM NAME='form1' ID='form1'>
  <!- Группа переключателей -->
  <BR><INPUT TYPE='radio' NAME='color' CHECKED>Белый
  <BR><INPUT TYPE='radio' NAME='color'>Красный
  <BR><INPUT TYPE='radio' NAME='color'>Желтый
  <BR><INPUT TYPE='radio' NAME='color'>Зеленый<BR>

  <!- Группа флажков -->
  <BR><INPUT TYPE='checkbox' NAME='fruit'>Груша
  <BR><INPUT TYPE='checkbox' NAME='fruit'>Персик
  <BR><INPUT TYPE='checkbox' NAME='fruit'>Яблоко<BR>
</FORM>
```

Теперь мы можем обратиться к любому флажку или переключателю следующим образом:

```
document.form1.color[1]  // соответствует переключателю
                         // "Красный"
document.form1.fruit[0]  // соответствует флажку "Груша"
```

## Атрибут CHECKED

Флажки и переключатели имеют атрибут `CHECKED`, который может принимать значение `true` или `false` и указывает, является данный элемент выбранным или сброшенным. По умолчанию этот атрибут принимает значение `false`, т.е. все флажки и переключатели

по умолчанию сброшены. Чтобы сделать какой-либо из этих элементов установленным сразу при открытии окна, укажите атрибут CHECKED в списке атрибутов дескриптора <INPUT>. Например, в листинге 7.2 по умолчанию мы устанавливаем переключатель **Белый**, как показано на рис. 7.2.

Анализируя значения атрибута CHECKED с помощью функции сценария, мы можем теперь определить, какие флажки и переключатели были выбраны пользователем. Давайте изменим код листинга 7.2 и добавим на Web-страницу кнопку **Показать выбранные элементы** и функцию show_selection для обработки события ONCLICK. Код сценария показан в листинге 7.3.

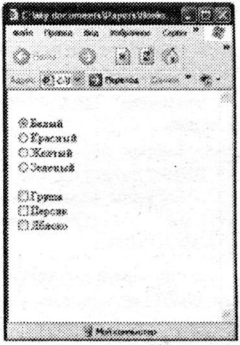



***Рис. 7.2.*** *Группы переключателей и флажков в окне обозревателя*

**Листинг 7.3. Определение флажков и переключателей, выбранных пользователем**

```
<HTML>
<HEAD>
<SCRIPT>
var colors = new Array("Белый","Красный","Желтый","Зеленый");
var fruits = new Array("Груша","Персик","Яблоко");
function show_selection() {
  for (i=0;i<document.form1.color.length;i++) {
    if (document.form1.color[i].checked)
      selected_radio.innerText = colors[i]+";";
  }
  selected_flags.innerText = "";
  for (j=0;j<document.form1.fruit.length;j++) {
    if (document.form1.fruit[j].checked)
      selected_flags.innerText += (fruits[j]+"; ");
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME='form1' ID='form1'>
```

*... код сценария листинга 7.2*

```
  <!- Кнопка -->
  <BR><INPUT TYPE='button' VALUE="Показать выбранные
   элементы" ONCLICK="show_selection()">
</FORM>
<H3>"Выбран переключатель:"</H3>
<P NAME="selected_radio" ID="selected_radio"></P>
<H3>"Выбраны флажки:"</H3>
<P NAME="selected_flags" ID="selected_flags"></P>
</BODY></HTML>
```

В сценарии создаются два массива с литеральными константами, соответствующими подписям переключателей и флажков. Затем следует код функции `show_selection`. В функции выполняются два цикла по массиву переключателей и массиву флажков. С помощью оператора `if` проверяются значения атрибутов `CHECKED` у каждого элемента массива, и если значение `true`, на страницу выводится соответствующий элемент массива `colors` или `fruits`. Результат выполнения программы показан на рис. 7.3.

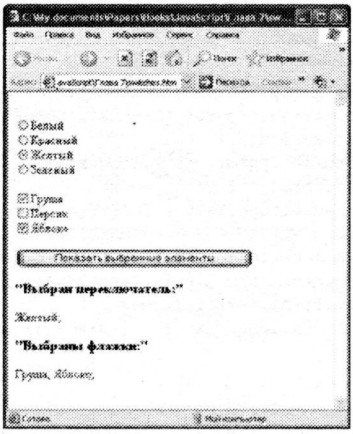



***Рис. 7.3.*** *Программа определяет переключатели и флажки, выбранные пользователями*

## Работа со списками формы

Списки в форме создаются с помощью парных дескрипторов <SELECT>...</SELECT> и вложенных дескрипторов <OPTION>...</OPTION> (см. также главу 3). Например:

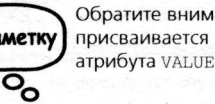
```
<SELECT NAME="form1">
  <OPTION VALUE=0>Газета.Ru</OPTION>
  <OPTION VALUE=1>Утро.Ru</OPTION>
  <OPTION VALUE=2>Комсомольская правда</OPTION>
  <OPTION VALUE=3>Экспресс газета</OPTION>
  <OPTION VALUE=4>Наша газета</OPTION>
  <OPTION VALUE=5>Футбол Хоккей</OPTION>
</SELECT>
```

Пункты списка автоматически добавляются в массив с именем списка. В списке отображается текст, заключенный между дескрипторами <OPTION>...</OPTION>. Этот же текст автоматически присваивается атрибуту списка TEXT.

В формах используются списки двух типов: с исключительным и множественным выделением элементов списка. В первом случае в списке можно выбрать только один пункт. Выбор другого пункта отменяет выбор предыдущего. В списках с множественным выделением можно выбрать произвольный набор списков. Тип списка задается атрибутом MULTIPLE в теле дескриптора <SELECT>. Анализ и обработка данных списков производится по-разному, в зависимости от типа списка.

### Списки с исключительным выбором пунктов

Если в списке разрешен выбор только одного пункта, индекс выбранного элемента заносится в свойство selectedIndex.




Обратите внимание на то, что свойству selectedIndex присваивается индекс пункта в массиве, а не значение атрибута VALUE и не текст пункта, показанный в списке.

Таким образом, в сценарии можно проверить, какой пункт списка выбран в данный момент, и выполнить действие, соответствующие выбранному пункту, используя для этого конструкцию с оператором switch (см. главу 5). Другое решение состоит в выборе параметра или функции из массива, содержащего столько же элементов, сколько

пунктов в списке. Давайте создадим список гиперссылок. Идея состоит в том, чтобы дать пользователям возможность выбрать из списка ресурс Интернет и перейти к нему щелчком на кнопке, как показано на рис. 7.4.

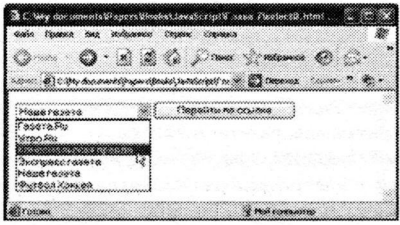


***Рис. 7.4.** Переход на Web-страницу, выбранную из списка*

HTML-код Web-страницы приведен в листинге 7.4.

**Листинг 7.4. Выбор гиперссылки из списка**

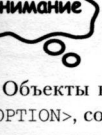
```
<HTML>
<HEAD>
<SCRIPT>
links = new Array();
links[0] = "http://www.gazeta.ru/";
links[1] = "http://www.utro.ru/";
links[2] = "http://www.kp.ru/";
links[3] = "http://www.eg.ru/";
links[4] = "http://ng.apress.ru/";
links[5] = "http://www.hockeynews-online.com/";

function relocate() {
  choice = document.form1.linklist.selectedIndex;
  window.location = links[choice];
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1">
<SELECT NAME="linklist">
  <OPTION VALUE=0>Газета.Ru</OPTION>
  <OPTION VALUE=1>Утро.Ru</OPTION>
```

```
  <OPTION VALUE=2>Комсомольская правда</OPTION>
  <OPTION VALUE=3>Экспресс газета</OPTION>
  <OPTION VALUE=4>Наша газета</OPTION>
  <OPTION VALUE=5>Футбол Хоккей</OPTION>
</SELECT>
<INPUT TYPE="button" VALUE="Перейти по ссылке"
ONCLICK="relocate()">
</FORM>
</BODY>
</HTML>
```

Чтобы выполнить переход, URL-адрес просто присваивается свойству `location` объекта `window`. В окне обозревателя автоматически отобразится Web-страница по заданному адресу. Список URL-адресов хранится в массиве `links`, причем элементы этого массива соответствуют названиям Web-узлов в списке. Поэтому индекс выбранного пункта списка используется в качестве индекса массива URL-адресов.

## Списки со множественным выбором пунктов

В списке с установленным атрибутом `MULTIPLE` можно выбрать несколько произвольных пунктов. Для этого нужно щелкнуть по порядку на всех пунктах, которые вас интересуют, удерживая нажатой клавишу <Ctrl>. Если отпустить клавишу, то список будет работать в обычном режиме с исключительным выбором пунктов, т.е. щелчок на пункте списка без нажатия клавиши <Ctrl> сбросит все ранее выбранные пункты списка.




Для анализа списков со множественным выделением нельзя использовать свойство `selectedIndex`, так как в эту переменную заносится индекс только первого пункта, выбранного в списке.

Объекты пунктов списка, заданные дескрипторами `<OPTION>...</OPTION>`, содержат атрибут `SELECTED`, который работает точно так же, как и уже знакомый вам атрибут `CHECKED` в объектах флажков и переключателей (см. предыдущий раздел), т.е. данный атрибут принимает значения `true` и `false` в зависимости от того, выбран этот пункт списка или нет. Атрибут `SELECTED` может быть установлен в HTML-коде списка. В этом случае данный пункт будет выбран на Web-странице по умолчанию.


Если в коде списка с исключительным выбором пунктов вы установите атрибуты SELECTED в нескольких дескрипторах <OPTION>, то выбранным по умолчанию будет только последний установленный пункт.

Теперь все ясно. Если в объектах пунктов списка есть атрибут SELECTED, а сами объекты являются членами массива с именем списка, то в сценарии мы можем определить выбранные массивы с помощью цикла for, так же, как мы делали это для анализа групп флажков.


Пример использования списка со множественным выделением показан в листинге 7.5 и на рис. 7.5.

**Листинг 7.5. Использование списка со множественным выделением**

```
<HTML>
<HEAD>
<SCRIPT>
function show_list() {
  document.form1.selection.value = "";
  for (i=0;i<document.form1.flights.length;i++) {
    if (document.form1.flights[i].selected)
      document.form1.selection.value +=
       (document.form1.flights[i].text + ". " +
       document.form1.flights[i].value + "\n");
  }
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Авиарейсы</H2>
<FORM NAME="form1">
<P><SELECT NAME="flights" MULTIPLE>
  <OPTION VALUE="Рейс SA0316, 8:05">Москва -> Киев
              </OPTION>
  <OPTION VALUE="Рейс SA2505, 8:37">Москва -> Ереван
              </OPTION>
  <OPTION VALUE="Рейс SA0180, 8:55">Москва ->
              Санкт-Петербург</OPTION>
  <OPTION VALUE="Рейс SA5513, 9:12">Москва -> Варшава
              </OPTION>
```

```
  <OPTION VALUE="Рейс SA3346, 9:33">Москва ->
              Санкт-Петербург</OPTION>
  <OPTION VALUE="Рейс SA5513, 10:00">Москва -> Киев
              </OPTION>
</SELECT>
<INPUT TYPE="button" VALUE="Показать выбранные"
ONCLICK="show_list()"></P><P>
<TEXTAREA NAME='selection' ID='selection' ROWS=5 COLS=50>
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

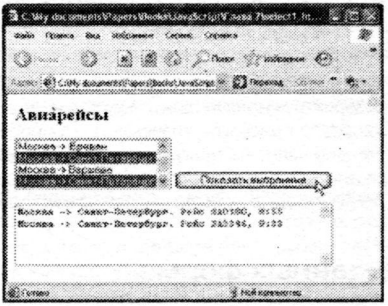
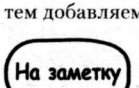



***Рис. 7.5.** Выберите рейсы из списка и посмотрите детальную информацию в поле редактора*


Обратите внимание: добавление атрибута `MULTIPLE` не только позволило выбирать несколько пунктов списка одновременно, но и изменило внешний вид списка. Список расширился для отображения четырех пунктов, тогда как список с исключительным выбором пунктов по умолчанию был раскрывающимся с полем в одну строку (см. рис. 7.4). Вы можете изменить размер окна списка с помощью атрибута `SIZE`, в котором нужно указать число строк, отображаемых в списке. Например, `SIZE=1` задаст список размером в одну строку.

Для того чтобы запустить функцию анализа списка, щелкните на кнопке **Показать выбранные**. Функция `show_list` сначала очищает

поле редактора, а затем выполняет цикл `for`, в котором проверяются атрибуты `SELECTED` пунктов списка. Если пункт выбран, сопутствующая информация выводится отдельной строкой в поле редактора. Обратите внимание на то, как мы комбинируем в одной строке данные из атрибутов `TEXT` и `VALUE` выбранного пункта списка, а затем добавляем символ абзаца `"\n"`.




Иногда бывает необходимо дать возможность пользователю сбросить все пункты списка. Оперируя только списком, сделать это невозможно, поскольку для того чтобы сбросить текущий пункт, надо выбрать другой. Вам нужно будет добавить в форму дополнительную кнопку **Очистить**, которая запустит следующий цикл:

```
for(i=0;i<размер_списка;i++)
  document.имя_формы.имя_списка[i].selected = 0;
```

Можно также добавить кнопку **Выделить все**, в функции которой нужно выполнить такой же цикл, но атрибуту `SELECTED` присвоить значение `1`. Попробуйте самостоятельно написать программу для кнопки **Инвертировать список**, которая сбросит все выделенные пункты и выберет те, которые не были выделены в исходном списке.

## Передача данных формы

Основное назначение форм состоит в сборе информации от пользователей и передачи этих данных на сервер разработчика Web-страницы. Типичный пример использования форм — регистрация пользователей Web-узла. Посетителю при первом обращении к Web-странице предлагается ввести в поля формы информацию о себе: фамилию, имя и отчество, год рождения, пол, род занятий. В форме могут быть представлены списки телеконференций, групп по интересам и прочих неформальных объединений, в которых посетитель может принять участие (или подписаться на них), выбрав соответствующий пункт списка, переключатель или флажок. Регистрация посетителей полезна как самим пользователям, поскольку позволяет им выбрать наиболее приемлемую конфигурацию предоставления услуг, так и разработчикам, так как дает возможность оценить качественный и количественный состав потенциальных клиентов. Такая оценка поможет спланировать дальнейшее развитие и продвижение Web-узла

для достижения максимальной эффективности и популярности среди пользователей.

После того как форма будет заполнена, информацию нужно передать на сервер для обработки, регистрации и сохранения в базе данных. Обычно на сервере для этого используются специальные программы, написание которых выходит далеко за рамки этой книги. Кроме того, программирование обработки данных форм на сервере требует знания технических характеристик и программного обеспечения самого сервера. Обратитесь за помощью к своему системному администратору.

В этом разделе мы рассмотрим лишь наиболее общие принципы управления передачей данных с формы на сервер. Но прежде чем передать данные, нужно проверить содержимое полей формы. Пользователь мог забыть заполнить какое-нибудь важное поле, либо ввести данные не в том формате. В следующем подразделе мы познакомимся с методами проверки значений в полях формы.

## Проверка полей формы

С помощью сценариев можно выполнять проверку правильности заполнения полей формы, по мере того как пользователь вводит текст с клавиатуры. Для этого используется событие ONCHANGE, которое активизируется каждый раз, когда пользователь покидает элемент управления, в поле которого были внесены изменения. Для проверки содержимого поля используется встроенная функция test() в сочетании с переменной особого типа, содержащей список символов, применение которых недопустимо. Синтаксис проверки содержимого поля показан в следующем примере:

```
<INPUT TYPE='text' NAME='first_name' SIZE='30' ONCHANGE="var
pattern=/['0-9',',',':']/; if(pattern.test(this.value))
alert('Проверьте данные в поле Имя');"> Имя
```




Обратите внимание на то, что событию ONCHANGE назначен сценарий, состоящий из нескольких команд, разделенных символами точки с запятой. Ранее в подобных случаях вы определяли отдельную функцию и присваивали ее событию. Эти способы взаимозаменяемы.

В сценарии определяется переменная pattern особого типа. Имя переменной может быть другим, но важно соблюсти правильный синтаксис присвоения значения этой переменной:

```
var имя_переменной = /[список запрещенных символов]/
```

В списке можно указывать диапазон символов:

- `'0-9'` или `'\d'` — все цифры;
- `'а-я'` — все строчные буквы кириллицы;
- `'А-Я'` — все прописные буквы кириллицы;
- `'А-я'` — все буквы кириллицы;
- `'a-z'` — все строчные латинские буквы (первая буква — латинская «a»);
- `'A-Z'` — все прописные латинские буквы (первая буква — латинская «A»);
- `'A-z'` — все буквы латинского алфавита (первая буква — латинская «A»);
- `'A-я'` — все буквы (первая буква — латинская «A»);
- `'\w'` — все буквы, цифры и символ подчеркивания «_»;
- `'\s'` — символы пробела, табуляции и абзаца.

Далее в сценарии используется конструкция с оператором `if`. Если условие оператора `if` выполняется (в данном случае функция `test` обнаруживает в значении текущего элемента управления `this.value` символы, представленные в переменной `pattern`), выполняется команда, следующая за оператором `if(условие)`. В данном примере используется функция `alert`, которая покажет на экране окно сообщения с предупреждением о том, что в поле введено неверное значение (рис. 7.6).

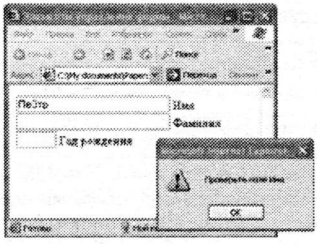



***Рис. 7.6.** В текстовое поле **Имя** были введены недопустимые символы*


В примере выше в переменной pattern мы указали диапазоны символов, которые не должны использоваться в поле. Можно поступить наоборот — указать диапазон допустимых символов. Например, в следующем примере указывается, что в поле допустимо использовать только прописные и строчные буквы кириллического алфавита:

```
ONCHANGE="var pattern=/['А-я']/; if(!pattern.test
(this.value)) alert('Проверьте данные в поле');"
```

Обратите внимание на восклицательный знак перед функцией pattern.test. Это логический оператор отрицания, который изменяет значение, возвращенное функцией test, на противоположное, т.е. окно с предупреждением будет показано в том случае, если введенные символы не будут принадлежать диапазону, заданному в переменной pattern.

## Передача данных на сервер

Для передачи данных на сервер используется метод формы submit(). Метод можно вызвать непосредственно в коде сценария: *имя_формы*.submit(), но чаще данные передаются с помощью специальной кнопки **Подача запроса**. Эта кнопка создается следующим образом:

```
<INPUT TYPE='submit'>
```

Щелчок на кнопке **Подача запроса** автоматически вызывает для формы метод submit(). Адрес, по которому передаются данные, и способ передачи устанавливаются в атрибутах дескриптора <FORM>.

- **ACTION** — строка адреса передачи данных:
  - URL-адрес Web-сервера и имя файла программы обработки данных;
  - адрес электронной почты в формате "mailto:*адрес_электронной_почты*".
- **METHOD** — метод передачи get или post; отличаются форматом передачи данных и именем стандартной переменной среды Web-сервера, которой будет присвоена строка "*имя_поля=значение*,…".

В формах используется еще одна полезная специальная кнопка — **Сброс**. Кнопка создается следующим образом:

```
<INPUT TYPE='reset'>
```

Кнопка **Сброс** вызывает для формы метод `reset()`, который восстанавливает поля формы в том виде, в каком они были установлены по умолчанию.


Кнопки типов `submit` и `reset` имеют такой же набор атрибутов и событий, как и обычные кнопки. Можно заменить стандартные подписи кнопок **Подача запроса** и **Сброс** на другие, присвоив текст подписи атрибутам `VALUE`. Примите к сведению, что стандартные подписи кнопок выводятся операционной системой пользователя, т.е. на компьютере с англоязычным интерфейсом Windows будут выведены надписи **Submit** и **Reset**, а на Windows с китайским языком эти кнопки будут подписаны по-китайски. В результате назначение этих важных кнопок будет понятным любому пользователю. Если же вы зададите альтернативные подписи, они останутся такими на компьютерах всех пользователей.


Иногда бывает необходимо выполнить сценарий во время передачи данных. Если вы присвоите выполнение сценария событию `ONCLICK` кнопки `submit` или `reset`, они перестанут выполнять свои непосредственные функции. Воспользуйтесь для решения этой проблемы событиями формы `ONSUBMIT` и `ONRESET`. Впрочем, передачу данных формы можно организовать из любого сценария, добавив в конце функции обработки события строку

`document.`*`имя_формы`*`.submit()`.


Пример формы с кнопками **Подача запроса** и **Сброс** приведен ниже.

```
<FORM NAME="form1" ACTION="mailto:mymail@mail.
ru" METHOD="post">
  <INPUT TYPE="text" NAME="text1">
  <INPUT TYPE='submit'><INPUT TYPE='reset'>
</FORM>
```

В данной форме установлена передача текста поля `text1` по адресу `mymail@mail.ru` методом `post`. После щелчка на кнопке **Подача запроса** обозреватель покажет окно с предупреждением о небезопасной передаче данных (рис. 7.7).

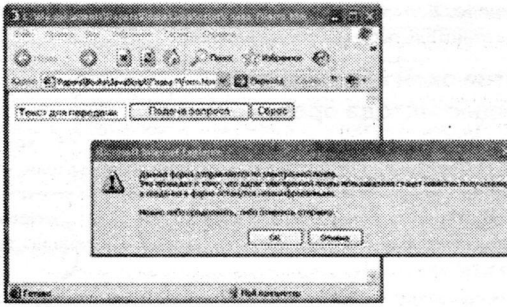



***Рис. 7.7.*** *Подача запроса по адресу электронной почты*

Если щелкнуть на кнопке OK, имя текстового поля с его содержимым будет передано по адресу электронной почты, заданному в атрибуте `ACTION` формы. Впрочем, дойдет ли сообщение по указанному адресу, следует выяснить у администратора сервера электронной почты, где зарегистрирован ваш почтовый ящик.

## Создание пользовательских диалоговых окон

Ранее в главе 3 вы познакомились со стандартными диалоговыми окнами, открываемыми в сценариях JavaScript с помощью функций:

- **`alert`** — окно сообщения с единственной кнопкой OK;
- **`confirm`** — окно с предложением подтвердить действие пользователя с кнопками OK и Отмена;
- **`prompt`** — окно ввода с текстовым полем и двумя кнопками, OK и Отмена.

Стандартные диалоговые окна предоставляют разработчикам довольно ограниченный выбор средств ведения диалога с посетителями Web-страниц. Но мы можем не ограничиваться стандартными диалогами и разработать собственные. Пользовательские диалоговые окна представляют собой отдельные Web-страницы, обычно содержащие формы, которые открываются в особым образом сконфигурированных окнах обозревателя, сохраняя при этом связь с окном основной

Web-страницы. В этом разделе вы научитесь открывать и контролировать дополнительные окна обозревателей.

## Открытие окон обозревателя с помощью метода open

В главе 2 при изучении гиперссылок вы узнали, что документ HTML по ссылке можно открыть в новом окне обозревателя, установив в дескрипторе <A> атрибут TARGET=_blank. Но это не единственный способ открытия нового окна обозревателя. В JavaScript в этих целях используется метод open объекта window. Синтаксис данного метода следующий:

```
var имя_переменной = window.open(URL, имя_окна, параметры);
```

Метод возвращает объект нового окна, который можно присвоить переменной для управления окном в коде сценария. Все аргументы, передаваемые методу open, необязательны. Команда window.open() откроет пустое окно обозревателя. В первом аргументе можно указать URL-адрес существующей Web-страницы, которая отобразится в новом окне. Имя окна, заданное во втором аргументе, можно будет использовать в текущей Web-странице как значение атрибута TARGET в гиперссылках. В результате щелчок на гиперссылке откроет документ в этом окне. Третий аргумент содержит в себе список параметров конфигурации окна обозревателя. Используется следующий формат списка: "*параметр1=значение,параметр2=значение,…*".


Обратите внимание: знак равенства между именем параметра и значением не должен отделяться пробелами, иначе это вызовет ошибку.

Для настройки конфигурации окна обозревателя используются параметры, перечисленные в табл. 7.1.

**Таблица 7.1. Параметры конфигурации нового окна обозревателя**

| Параметр | Описание |
|---|---|
| width | Ширина окна в пикселях |
| height | Высота окна в пикселях |
| Остальные параметры принимают логические значения в формате yes-no, или 1-0. | |
| resizable | Если no или 0, то пользователь не может изменять размер открывшегося окна |

*Окончание табл. 7.1*

| Параметр | Описание |
|---|---|
| location | Если no или 0, то в окне не отображается поле Адрес |
| menubar | Если no или 0, то в окне не отображается строка меню |
| scrollbars | Если no или 0, то в окне не отображаются полосы прокрутки |
| status | Если no или 0, то в окне не отображается строка состояния |
| toolbar | Если no или 0, то в окне не отображается панель инструментов |

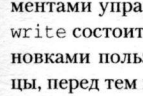

Если третий аргумент метода open пропущен, то новое окно обозревателя выглядит так же, как текущее окно. Но если установлен хотя бы один параметр конфигурации окна, всем остальным параметрам автоматически присваивается значение no, т.е. в окне не отображаются никакие строки и панели пользовательского интерфейса, и размеры окна не могут быть изменены.

## Ввод элементов Web-страницы в новое окно

Открыв пустое новое окно обозревателя, мы можем заполнить его содержимым с помощью метода write. Например, так вводится текст в новое окно:

```
имя_окна.write("<H2>Заголовок нового окна</H2>");
```

Таким способом можно создать диалоговое окно с сообщением, сформированным с помощью сценария в исходной Web-странице. Метод write позволяет создавать с чистого листа довольно сложные документы HTML, содержащие не только текст, но и формы с элементами управления. Удобство заполнения окна с помощью метода write состоит в том, что содержимое окна может определяться установками пользователя, сделанными в форме исходной Web-страницы, перед тем как было открыто новое окно.




Например, давайте создадим диалоговое окно выбора файлов для выгрузки на удаленный сервер. Вы не знаете, сколько файлов собирается выгрузить пользователь за один сеанс, поэтому хотите дать возможность ему установить число полей в форме перед ее открытием. Для этого пользователь указывает число файлов в поле исходной Web-страницы. Предположим, что за один сеанс можно выгружать от одного до девяти файлов, т.е. в поле нельзя вводить 0,

число больше 9, буквы или оставлять поле незаполненным. В листинге 7.6 вам предлагается решение этой задачи.

**Листинг 7.6. Открытие диалогового окна, заполняемого из исходной Web-страницы**

```
<HTML>
<HEAD>
<SCRIPT>
function show_dialog() {
  upload_dialog = window.open("","","height=300,width=400,
                              scrollbars=yes");
  upload_dialog.document.write("<H3>Выберите файлы на диске</
                               H3>");
  upload_dialog.document.write("<FORM ONSUBMIT='window.
   close()' METHOD='get' ACTION='http://www.myserver.com/
   cgi-bin/getfiles.py'>");
  for(i=0;i<document.forms[0].file_num.value;i++)
    upload_dialog.document.write("<INPUT TYPE='file'><BR>");
    upload_dialog.document.write("<INPUT TYPE='submit'>");
    upload_dialog.document.write("<INPUT TYPE='button'
     VALUE='Отмена' ONCLICK='window.close()'></FORM>");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<P> Открыть диалоговое окно для загрузки
<INPUT TYPE='text' NAME='file_num' SIZE='1' VALUE=5
ONCHANGE="var pattern=/['1-9']/; if(!pattern.test(this.value)
|| !this.value || this.value.length>1) alert('Введите в поле
число от 1 до 9');">
файлов</P>
<INPUT TYPE='button' VALUE="Открыть диалог" ONCLICK="show_
dialog()">
</FORM>
</BODY>
</HTML>
```

Внешний вид исходной Web-страницы и пользовательского диалогового окна, создаваемого с помощью функции show_dialog, показаны на рис. 7.8.

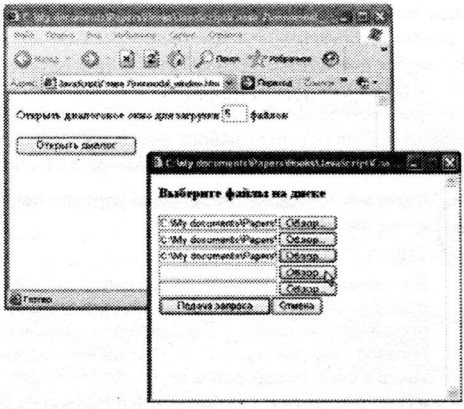



***Рис. 7.8.** Открытие диалогового окна выбора файлов для выгрузки на сервер*

Исходная Web-страница очень проста. Она содержит текст предложения выгрузить файлы на сервер. Прямо в текст встроено небольшое поле, в котором указывается число выгружаемых файлов. Обратите внимание на строку проверки содержимого поля, присвоенную событию ONCHANGE:

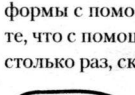
```
"var pattern=/['1-9']/; if(!pattern.test(this.value) ||
!this.value || this.value.length>1) alert('Введите в поле
число от 1 до 9');"
```

Диапазон допустимых значений от 1 до 9 задается в переменной `pattern`, как мы делали это ранее в разделе «Проверка полей формы». Но в методе `test` мы выполняем более сложную проверку условий неверности значения поля.

1. Значение не принадлежит диапазону переменной `pattern`.
2. Значение неопределенно (т.е. поле осталось незаполненным).
3. Введено многозначное число.

При выполнении любого из этих условий (поскольку они объединены логическим оператором ИЛИ — `||`), будет выведено на экран окно предупреждения.

Щелчок на кнопке **Открыть диалог** запускает на выполнение функцию `show_dialog`, определенную в сценарии в разделе заголовка. Эта функция открывает новое окно обозревателя и присваивает его переменной `upload_dialog`, после чего вводит в новое окно код HTML формы с помощью метода `upload_dialog.document.write()`. Учтите, что с помощью цикла `for` поле выбора файла добавляется в форму столько раз, сколько было определено пользователем в поле `file_num`.




В форме используется элемент управления, который ранее мы не использовали:

```
<INPUT TYPE='file'>
```

Этот элемент управления представляет собой текстовое поле с кнопкой **Обзор** (см. рис. 7.8). Щелчок на кнопке открывает стандартное для операционной системы диалоговое окно **Выбор файла**. При щелчке на кнопке **Открыть** в окне **Выбор файла** путь к файлу будет помещен в поле элемента управления, как показано на рис. 7.8.

За полями выбора файлов следуют две кнопки: **Подача запроса** и **Отмена**. Кнопка **Отмена** просто закрывает окно обозревателя, вызвав для этого метод `window.close()`. Кнопка **Подача запроса** выполняет передачу выбранных файлов на сервер по адресу, указанному в атрибуте `ACTION` дескриптора `<FORM>`. При этом окно обозревателя тоже закрывается. Мы добились этого, назначив метод `window.close()` событию `ONSUBMIT` в дескрипторе `<FORM>`.

## Модальные диалоговые окна

Пользовательское диалоговое окно, созданное нами в листинге 7.6, имеет ряд недостатков.

- С помощью метода `write` невозможно ввести сценарий, исполняемый в новом окне.
- Диалоговое окно существует независимо от окна исходного окна Web-страницы. Если щелкнуть на исходном окне, оно скроет от пользователя диалоговое окно, что может привести к ошибке.
- Невозможно организовать обмен данными между новым окном и исходным окном.

Эти проблемы можно решить в JavaScript с помощью функции создания *модальных* диалоговых окон.


Модальными называются такие окна, которые открываются поверх исходного окна и не дают пользователю вернуться к исходному окну до тех пор, пока не будет закрыто модальное окно. Модальными были стандартные диалоговые окна JavaScript, открываемые с помощью функций `alert`, `confirm` и `prompt`. Окно, открываемое с помощью метода `window.open`, является *немодальным*.

## Создание пользовательского модального окна

Для создания пользовательского модального окна в JavaScript используется метод `showModalDialog(`*`URL, значение, параметры`*`)` объекта `window`.

- *`URL`* — адрес URL или путь на диске к файлу Web-страницы, в котором записан HTML-код диалогового окна. Это обязательный аргумент. В модальном диалоговом окне может отображаться только документ HTML, сохраненный в отдельном файле.
- *`значение`* — любая переменная, передаваемая в диалоговое окно из основного документа.
- *`параметры`* — список параметров диалогового окна. Для настройки конфигурации модального диалогового окна используются почти те же параметры, перечисленные в табл. 7.1, что и для немодального окна, за исключением параметров размера окна. В данном случае параметры размеров окна выглядят так:
  - `dialogWidth` — ширина диалогового окна в колонках, например: `"dialogWidth=25em"`;
  - `dialogHeight` — высота диалогового окна в рядах, например: `"dialogHeight=25em"`.


`em` — это особая полиграфическая единица измерения. Она соответствует площади, занимаемой буквой `m`.

Ниже показан пример открытия модального диалогового окна:

```
Dialog = window.showModalDialog("PersonalInf.html", "",
dialogWidth:25em; dialogHeight:28em;");
```

В этом примере открывается модальное диалоговое окно, код которого хранится в файле `PersonalInf.html`, находящемся на компьютере в одной папке с файлом основной Web-страницы. Пример модального диалогового окна показан на рис. 7.9, а его код — в листинге 7.7.

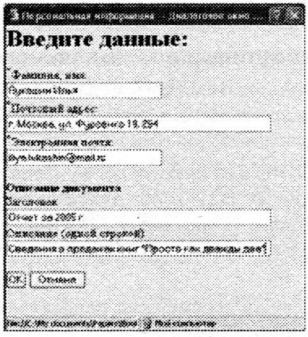



***Рис. 7.9.** Модальное диалоговое окно Персональная информация*

**Листинг 7.7. HTML-код диалогового окна `PersonalInf.html`**

```
<HTML>
<HEAD>
<TITLE>Персональная информация</TITLE>
<SCRIPT>
function setValues(){
  var arReturnValues = new Array();
  arReturnValues[0] = document.forms[0].elements[0].value;
  arReturnValues[1] = document.forms[0].elements[1].value;
  arReturnValues[2] = document.forms[0].elements[2].value;
  arReturnValues[3] = document.forms[0].elements[3].value;
  arReturnValues[4] = document.forms[0].elements[4].value;
  window.returnValue = arReturnValues;
  window.close();
}
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFF66">
```

```
<H1>Введите данные:</H1>
<FORM><P><FONT COLOR='red'><SUP>*</SUP></FONT>Фамилия,
имя:<BR>
    <INPUT NAME="fieldAuthor" ID="fieldAuthor" SIZE="30"
    ONCHANGE="var pattern=/['A-я']/; if(!pattern.test(this.
    value) || !this.value) alert('Неверный формат в поле
    \'Фамилия, имя\'');"><BR>
    <FONT COLOR='red'><SUP>*</SUP></FONT>Почтовый адрес:<BR>
    <INPUT NAME="fieldAddress" SIZE="54" ONCHANGE="var
    pattern=/['A-я','0-9',',-.',';',':']/; if(!pattern.
    test(this.value) || !this.value) alert('Неверный формат
    в поле \'Почтовый адрес\'');"><BR>
    <FONT COLOR='red'><SUP>*</SUP></FONT>Электронная
    почта:<BR>
    <INPUT NAME="fieldEMail" SIZE="30" ONCHANGE="var
    pattern=/['A-я',' ']/; if(pattern.test(this.value)
    || !this.value || this.value.indexOf('@')==-1)
    alert('Неверный формат в поле \'Электронная почта\
    '');"></P>
  <P><B>Описание документа</B><BR>
  Заголовок<BR>
  <INPUT NAME"fieldTitle" SIZE="54"><BR>
  Описание (одной строкой)<BR>
  <INPUT NAME="fieldDescription" SIZE="54"></P>
  </P><INPUT TYPE="button" VALUE="OK"
  ONCLICK="setValues()">
  <INPUT TYPE="button" VALUE="Отмена" ONCLICK="window.
  close()"></P>
</FORM>
</BODY>
</HTML>
```

Диалоговое окно содержит пять текстовых полей. Первые три поля — **Фамилия, имя**; **Почтовый адрес**; **Электронная почта** — помечены красными звездочками. Это означает, что данные поля обязательно должны быть заполнены пользователем. Обратите внимание на то, как настроена проверка полей формы:

- **Фамилия, имя** — только буквы русского алфавита, пустая строка не допускается;
- **Почтовый адрес** — только буквы русского алфавита, цифры и знаки препинания, пустая строка не допускается;

- **Электронная почта** — нельзя использовать буквы русского алфавита и пробелы, обязательно должен присутствовать символ `@`, пустая строка не допускается.

Два оставшихся поля диалогового окна, **Заголовок** и **Описание**, необязательны и могут содержать любые комбинации символов.

Функционирование кнопок диалогового окна мы рассмотрим в следующем подразделе.

## Обмен данными с диалоговым окном

Модальное диалоговое окно открывается следующей строкой кода JavaScript:

```
Dialog = window.showModalDialog("PersonalInf.html", "",
dialogWidth:25em; dialogHeight:28em;");
```

Это напоминает открытие немодального диалогового окна с помощью метода `window.open`. В обоих случаях создается переменная, которой присваивается результат выполнения метода `showModalDialog` или `open`. Но результаты этих методов совершенно разные. Если метод `open` возвращает объект окна обозревателя для манипулирования им в следующих строках кода сценария, то метод `showModalDialog` работает иначе. Во время отображения на экране модального диалогового окна выполнение сценария прерывается. После закрытия окна метод `showModalDialog` возвращает в переменную результат. Какой же результат будет возвращен в переменную `Dialog`, если в нашем окне, показанном на рис. 7.9, содержится целых пять текстовых полей?

Модальное диалоговое окно возвращает значение, содержащееся в стандартной переменной `window.returnValue`. Этой переменной можно присвоить любое значение — логическое `true` или `false`, число, текст, массив или объект. В нашем примере в листинге 7.7 щелчок на кнопке **OK** приводит к вызову функции `setValues`. Эта функция возвращает значения всех полей формы и заносит их в массив `arReturnValues`, который затем присваивается переменной `window.returnValue`. Именно этот массив и будет возвращен переменной `Dialog` сразу после выполнения метода `window.close()`, который закрывает диалоговое окно.

Кнопка **Отмена** просто выполняет метод `window.close()`. В этом случае в переменную `Dialog` будет возвращено значение `NaN`.


Чтобы убедиться, что все происходит именно так, давайте создадим Web-страницу, использующую модальное диалоговое окно. HTML-код этой страницы показан в листинге 7.8.

**Листинг 7.8. Код основной Web-страницы, использующей модальное диалоговое окно**

```
<HTML>
<HEAD>
<SCRIPT>
var Dialog = new Array();
var Fields = new Array("Фамилия, имя","Почтовый
адрес","Электронная почта","Заголовок","Описание")
function show_dialog() {
  output.innerHTML = "";
  Dialog = window.showModalDialog("PersonalInf.html", "",
           "dialogWidth:25em; dialogHeight:28em;");
  for (i=0;i<Dialog.length;i++)
    output.innerHTML += ("<B>" + Fields[i] + "</B>: " +
                         Dialog[i] + "<BR>")
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE='button' VALUE="Открыть диалог"
ONCLICK="show_dialog()">
</FORM>
<P NAME='output' ID='output'></P>
</BODY></HTML>
```

Эта Web-страница содержит единственную кнопку — **Открыть диалог**. Щелчок на кнопке вызывает на выполнение функцию `show_dialog`. Функция содержит уже известную вам строку открытия модального диалогового окна. После закрытия диалогового окна выполняется цикл `for`, который выводит на Web-страницу значения массива, возвращенного модальным окном (рис. 7.10).

Обмен данными между Web-страницей и диалоговым окном двусторонний. Мы можем не только получать результат работы пользователя с диалоговым окном, но и передавать исходные данные во втором аргументе метода `showModalDialog`.

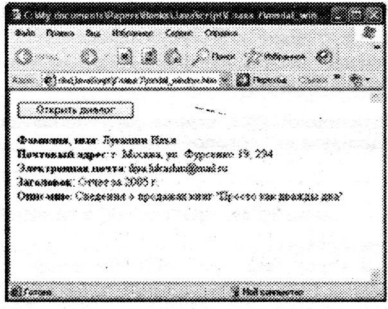



***Рис. 7.10.*** *Текст, введенный в поля формы (см. рис. 7.9), отображен в окне основной Web-страницы*

**Пример**

Например, давайте сделаем так, чтобы при повторном открытии диалогового окна в нем отобразилась ранее введенная информация.

1. Передадим в диалоговое окно исходную переменную `Dialog`, добавив ее во второй аргумент метода `showModalDialog`:

```
Dialog = window.showModalDialog("PersonalInf.html",
Dialog, "dialogWidth:25em; dialogHeight:28em;");
```

**Внимание**

Обратите внимание на то, что в листинге 7.8 пустой массив `Dialog` создается в сценарии вне кода функции `show_dialog`, а в функции перед переменной `Dialog` не установлен оператор `var`, что указывает на использование глобальной переменной (о глобальных и локальных переменных речь шла в главе 4). Если не соблюсти эти условия, при открытии диалогового окна произойдет ошибка.

2. В модальном окне значение второго аргумента автоматически присваивается стандартной переменной `window.dialogArguments`. Добавим в основной раздел HTML-кода Web-страницы `PersonalInf.html` сценарий обработки значения переменной `window.dialogArguments`:

```
<SCRIPT>
for (i=0;i<window.dialogArguments.length;i++)
  document.forms[0].elements[i].value = window.
dialogArguments[i];
</SCRIPT>
```

3. Этот сценарий заполняет поля формы данными из переданного массива. Чтобы программа работала без ошибок, нам нужно еще изменить код кнопки **Отмена**. Действительно, сейчас эта кнопка просто закрывает диалоговое окно, возвращая значение `NaN`, которое присваивается переменной `Dialog`. При следующем открытии диалога это же значение `NaN` будет передано переменной `window.dialogArguments`, что приведет к ошибке при выполнении сценария. Нужно, чтобы кнопка **Отмена** возвратила исходный массив данных. Проблема решается очень просто. Изменим код события `ONCLICK` кнопки **Отмена** следующим образом:

```
ONCLICK="window.returnValue = window.dialogArguments;
window.close()"
```

Прежде чем закрыть окно, массив из переменной `window.dialogArguments` присваивается переменной `window.returnValue` и возвращается обратно в переменную `Dialog`.

Теперь, щелкнув на кнопке `Открыть диалог`, мы получим диалоговое окно, в котором все поля будут заполнены исходными данными (рис. 7.9). Окончательный вариант основного раздела HTML-кода диалогового окна показан в листинге 7.9.

**Листинг 7.9. Окончательный вариант HTML-кода файла `PersonalInf.html`**

```
<BODY BGCOLOR="#FFFF66">
<H1>Введите данные:</H1>
<FORM><P><FONT COLOR='red'><SUP>*</SUP></FONT>Фамилия,
имя:<BR>
    <INPUT NAME="fieldAuthor" ID="fieldAuthor" SIZE="30"
    ONCHANGE="var pattern=/['А-я']/; if(!pattern.test(this.
    value) || !this.value) alert('Неверный формат в поле
    \'Фамилия, имя\'');"><BR>
  <FONT COLOR='red'><SUP>*</SUP></FONT>Почтовый адрес:<BR>
    <INPUT NAME="fieldAddress" SIZE="54" ONCHANGE="var
    pattern=/['А-я','0-9',',-.',';',':']/; if(!pattern.
```

```
    test(this.value) || !this.value) alert('Неверный формат
    в поле \'Почтовый адрес\'');"><BR>
  <FONT COLOR='red'><SUP>*</SUP></FONT>Электронная почта:<BR>
    <INPUT NAME="fieldEMail" SIZE="30" ONCHANGE="var
    pattern=/['A-я',' ']/; if(pattern.test(this.value) ||
    !this.value || this.value.indexOf('@')==-1) alert
    ('Неверный формат в поле \'Электронная почта\'');"></P>
  <P><B>Описание документа</B><BR>
  Заголовок<BR>
  <INPUT NAME"fieldTitle" SIZE="54"><BR>
  Описание (одной строкой)<BR>
  <INPUT NAME="fieldDescription" SIZE="54"></P>
  </P><INPUT TYPE="button" VALUE="OK"
  ONCLICK="setValues()">
  <INPUT TYPE="button" VALUE="Отмена" ONCLICK="window.
   returnValue = window.dialogArguments; window.close()"></P>
</FORM>
<SCRIPT>
for (i=0;i<window.dialogArguments.length;i++)
  document.forms[0].elements[i].value = window.
dialogArguments[i];
</SCRIPT>
</BODY>
```

## Работа с внешними файлами в JavaScript

На Web-страницах можно использовать и показывать данные, хранящиеся в других файлах. Более того, сценарии документов HTML применяют для сохранения данных на диске компьютера. Использование данных из внешних файлов позволяет облегчить и сделать более понятным HTML-код Web-страницы. Запись на диск может осуществляться с применением нескольких разных технологий.

- Технология *cookies* позволяет записывать на диск в особую папку временных Интернет-файлов информацию о текущей настройке пользователем конфигурации Web-страницы. В результате при следующем открытии Web-страница появится в том виде, в каком пользователь завершил с ней работу во время предыдущего сеанса. Очень полезная технология, но ее изучение выходит за рамки этой книги.

- Другой подход состоит в записи информации в любом месте на диске пользователя с помощью методов непосредственного обращения к файловой системе компьютера. В отличие от обмена данными с помощью форм запись будет производиться не на сервер, а на компьютер пользователя. Это имеет смысл только при работе в intranet или при использовании Web-приложений на своем компьютере для редактирования собственных баз данных.

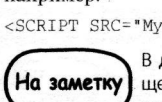
## Извлечение данных из внешних файлов

Web-страницы часто используют для представления информации, хранящейся в базе данных. Доступ к профессиональным базам данных осуществляется с помощью программ, запускаемых на Web-сервере. Но чтобы получить доступ к простой базе данных, записанной в текстовом файле на вашем компьютере, можно обойтись средствами JavaScript.

### Ввод данных из внешних файлов сценариев

При изучении основ языка HTML в главе 2 вы узнали о том, что на Web-страницах можно использовать сценарии, сохраненные во внешних файлах, с помощью атрибутов `SRC` дескриптора `<SCRIPT>`, например:

```
<SCRIPT SRC="MyScript.js"></SCRIPT>
```




В данном примере добавляется код сценария, хранящегося в файле `MyScript.js`. Файл может иметь любое расширение. Важно только, чтобы он содержал обычный текст без форматирования, а код, хранящийся в файле, соответствовал синтаксису и структуре языка JavaScript. Сценарий, добавленный из файла, сразу же исполняется обозревателем.

В отдельном файле можно сохранять любые сценарии, включая исполняемые команды, определения пользовательских функций и переменных. Сценарии будут исполняться точно так же, как если бы код сценария был вписан непосредственно на Web-страницу.


Рассмотрим пример базы данных, сохраненной в виде внешнего сценария в файле `db.js`, как показано в листинге 7.10.

**Листинг 7.10. База данных в файле `db.js`**

```
var db = new Array();
db[0] = new Array(1,1,0.25);
db[1] = new Array(1,2,0.75);
db[2] = new Array(1,3,1.00);
...
db[1037] = new Array(215,74,0.11);
db[1038] = new Array(215,75,0.00);
```

Это матрица из множества столбцов и строк, сохраненная в виде двумерного массива db. Каждый элемент массива db представляет собой вложенный массив из трех элементов — номера столбца матрицы, номера строки и значения в формате числа с плавающей запятой. Если в код Web-страницы добавить строку `<SCRIPT SRC="db.js"> </SCRIPT>`, то во время загрузки документа автоматически будет создан глобальный массив db, который можно будет использовать для демонстрации данных на Web-странице. Вы можете себе представить, насколько тяжело было бы работать с кодом Web-страницы в случае добавления этого огромного массива данных непосредственно в код HTML.

Впрочем, редактировать такую базу данных вручную с помощью текстового редактора тоже тяжело. Ниже в разделе «Редактор базы данных» вы найдете пример Web-страницы, которую можно использовать на своем компьютере в качестве редактора для изменения и сохранения небольших баз данных.

## Открытие текстовых документов

Если вы захотите с помощью JavaScript создать небольшое Web-приложение для работы на собственном компьютере, то сможете использовать текстовые файлы как источник данных. JavaScript содержит ряд встроенных функций для обращения к файловой системе компьютера. Понятно, что эта технология неприменима к Web-страницам, публикуемым в Интернет.

Для открытия текстового файла используется следующий синтаксис:

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var file = fso.OpenTextFile("путь",параметр);
```


Текстовый файл открывается в два этапа. Сначала создается переменная `fso` с объектом файловой системы компьютера, а затем с помощью метода `OpenTextFile` этого объекта открывается файл на диске, который присваивается новой переменной `file`.

Методу `OpenTextFile` передаются следующие аргументы:

- *путь* — путь к файлу на диске;
- *параметр* — числовое значение, указывающее режим открытия файла:
  - `0` — только для чтения (по умолчанию);
  - `2` — для записи с затиранием имеющейся информации;
  - `8` — для добавления новых данных в конец файла.

Объект файла содержит методы доступа к данным, перечисленные в табл. 7.2.

**Таблица 7.2. Методы чтения данных из файла**

| Метод | Описание |
|---|---|
| `Read(n)` | Считывает следующие `n` символов |
| `ReadAll()` | Считывает весь текст до конца файла |
| `ReadLine()` | Считывает текущую строку |
| `Skip(n)` | Пропускает следующие `n` символов |
| `SkipLine()` | Пропускает текущую строку |

Считывание данных из файла происходит последовательно только в одном направлении от начала до конца, т.е. несколько вызовов `file.ReadLine()` один за другим будут считывать последовательно строку за строкой. Методы `Skip` и `SkipLine` сдвигают *точку считывания*, не возвращая данных из файла.


Давайте в текстовом редакторе Блокнот введем текст, показанный на рис. 7.11, и сохраним его в файле `test.txt` в корневой папке диска `C`. Теперь создадим Web-страницу со сценарием из листинга 7.11.

**Листинг 7.11. Считывание данных из текстового файла**

```
<SCRIPT>
var fso = new ActiveXObject("Scripting.FileSystemObject");
var f = fso.OpenTextFile("c:\\test.txt");
while (!f.AtEndOfStream)
  document.write(f.ReadLine()+"<BR>");
f.Close();
</SCRIPT>
```

Результат выполнения сценария показан на рис. 7.12.

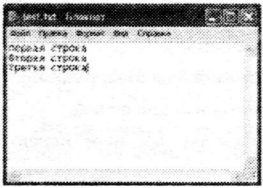



***Рис. 7.11.*** *Содержимое файла* `test.txt` *в приложении Блокнот*

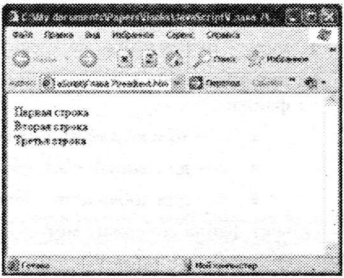



***Рис. 7.12.*** *Результат выполнения сценария листинга 7.11*


Обратите внимание на то, как в листинге 7.11 задается цикл `while` до конца файла. Признаком окончания файла служит свойство `AtEndOfStream`, которое принимает значение `true`, как только точка считывания дойдет до конца файла. Работа с файлом обязательно должна завершаться закрытием объекта с помощью метода `close()`.


Обозреватель обнаруживает на Web-странице потенциально опасный код с функциями обращения к файловой системе компьютера и выводит окно предупреждения (рис. 7.13). Предупреждение не напрасно, так как некоторые вирусы и другие вредоносные программы используют данные функции для копирования себя на диск. Если вы уверены в программе, просто щелкните на кнопке **Да**.

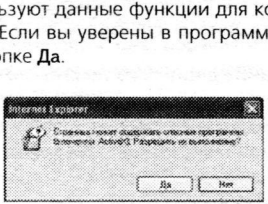



***Рис. 7.13.*** *Окно предупреждения о наличии на Web-странице потенциально опасных элементов*

## Сохранение данных в файле

Для того чтобы сохранить данные в текстовом файле, прежде всего необходимо открыть файл так, как мы делали это в предыдущем примере. Только теперь нужно установить такие параметры режима открытия, которые позволяли бы вести запись в файл (2 или 8, см. выше). Для записи используются методы файлового объекта, представленные в табл. 7.3.

**Таблица 7.3. Методы записи данных в файл**

| Метод | Описание |
|---|---|
| `Write("текст")` | Вводит текст в текущую позицию |
| `WriteBlankLines(n)` | Вводит n пустых строк |
| `WriteLine("текст")` | Вводит текст и добавляет символ конца строки |

Запись данных, так же, как и считывание, происходит в одном направлении с начала до конца файла. Об использовании методов записи речь пойдет в следующем подразделе главы.

## Редактор базы данных

Здесь мы рассмотрим пример редактора базы данных, основанного на Web-странице. Данный пример поможет закрепить знания, полученные как в этой, так и в предыдущих главах книги.

Безусловно, редактор базы данных на основе сценария JavaScript можно использовать только на персональном компьютере или в intranet. По сети Интернет доступ к базе данных таким способом получить невозможно. Хотя Web-интерфейсы доступа и редактирования баз данных чрезвычайно широко распространены, в их основе лежат принципиально иные технологии, рассмотрение которых выходит за рамки этой книги.

Тем не менее Web-страница, код которой приведен в листинге 7.12, может оказаться полезной для создания и редактирования небольшой собственной базы данных, сведения из которой можно представлять в Интернет с помощью других Web-страниц узла.

**Листинг 7.12. Редактор базы данных**

```
<HTML>
<HEAD>
<TITLE>Редактор базы данных</TITLE>
<SCRIPT>
```

```
// пустая база данных
var db = FALSE;
// функция создания HTML-кода раскрывающегося списка
function get_selector(cell_id) {
  var selector = "<SELECT ONBLUR=fillCell('" + cell_id +
   "',this.value);><OPTION VALUE='нд'>нд</OPTION>";
  for (n=0;n<=10;n+=1)
    selector += "<OPTION VALUE=" + n/10 + ">" + n/10 +
     "</OPTION>"
  selector += "</SELECT>"
  return selector;
}
// функция добавления раскрывающегося списка в ячейку
function createSelector(cell_id) {
  // определяем координаты ячейки
  var cell_coord = cell_id.split(":");
  var row_num = parseInt(cell_coord[0]);
  var col_num = parseInt(cell_coord[1]);
  // вставляем в ячейку раскрывающийся список
  matrix.rows[row_num].cells[col_num].innerHTML = get_
   selector(cell_id);
}
// функция заполнения ячейки выбранным значением
function fillCell(cell_id,value) {
  // определяем координаты ячейки
  var cell_coord = cell_id.split(":");
  var row_num = parseInt(cell_coord[0]);
  var col_num = parseInt(cell_coord[1]);
  // заполняем ячейку значением
    matrix.rows[row_num].cells[col_num].innerHTML = "<SPAN
    ONCLICK=createSelector('" + cell_id + "')>" + value +
    "</SPAN>";
}

// функция добавляет в таблицу пустую строку
function add_row(index) {
  var row = matrix.insertRow(index);
  var cell = row.insertCell(0);
  cell.innerText = index;
  for (i=1;i<=6;i++) {
    cell = row.insertCell(i);
    fillCell(index + ":" + i,"нд");
```

```
  }
}

// функция сохранения базы данных
function save_db() {
  // определяем путь к текущей папке
  var currPath = document.location.pathname
  pos = currPath.lastIndexOf("\\") + 1;
  path = currPath.substring(1,pos) + "db.js";
  // открываем файл db.js для записи с замещением
  var fso = new ActiveXObject("Scripting.FileSystemObject");
  var f = fso.CreateTextFile(path);
  // записываем значения таблицы в файл
  index = 0;
  f.WriteLine("var db = new Array()");
  for (i=1;i<matrix.rows.length;i++) {
    for (j=1;j<matrix.rows[i].cells.length;j++) {
      f.WriteLine("db[" + index + "] = new Array('" +
      i + "','" + j + "','" + matrix.rows[i].
      cells[j].innerText + "');");
      index++;
      }
  }
  f.close();
}
</SCRIPT>
<SCRIPT SRC='db.js'></SCRIPT>
</HEAD>
<BODY>
<H1>Электронная таблица</H1>
<TABLE NAME='matrix' ID='matrix' BORDER=1>
<TR><TH WIDTH=20> </TH>
<TH WIDTH=100>A</TH><TH WIDTH=100>B</TH><TH WIDTH=100>C</
TH><TH WIDTH=100>D</TH><TH WIDTH=100>E</TH><TH WIDTH=100>F</
TH></TR>
</TABLE>
<FORM>
<INPUT TYPE='button' VALUE='Добавить строку' ONCLICK='add_
row(matrix.rows.length)'>
<INPUT TYPE='button' VALUE='Сохранить' ONCLICK='save_db()'>
</FORM>
<SCRIPT>
```

```
if (!db)
  add_row(1);
else {
  for (s=0;s<db.length;s++) {
    row_num = db[s][0];
    col_num = db[s][1];
    value = db[s][2];
    if (row_num > matrix.rows.length-1) {
      add_row(matrix.rows.length);
    }
    fillCell(row_num + ":" + col_num,value);
  }
}
</SCRIPT>
</BODY>
</HTML>
```

Работу мы начнем с чистого листа — с пустой базы данных, которую в дальнейшем мы собираемся хранить в файле `db.js`. Этот файл добавляется в виде внешнего сценария в строке:

```
<SCRIPT SRC='db.js'></SCRIPT>
```

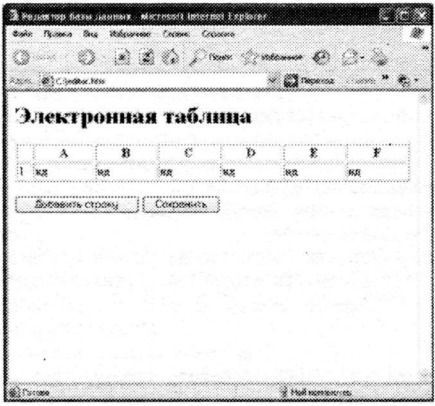



***Рис. 7.14.*** *Пустая база данных*

Если файл db.js отсутствует, эта строка кода будет пропущена без возникновения ошибки и управление программой перейдет к выполнению кода основного раздела Web-страницы. На странице создается таблица из одной строки с заголовками столбцов и форма с двумя кнопками — **Добавить строку** и **Сохранить**. Код сценария проверяет наличие массива базы данных db, и если db == NaN, с помощью функции add_row добавляется новая строка таблицы, как показано на рис. 7.14.

Ячейки добавленной строки заполнены текстом нд, что означает «нет данных». Кнопка Добавить строку вновь вызывает функцию add_row, добавляя новую строку. Текст в ячейках таблицы ведет себя очень интересно — щелчок на ячейке преобразует текст в раскрывающийся список со значениями от 0 до 1 с шагом в 0,1, плюс значение нд. Как это происходит?

1. В коде функцией add_row для заполнения каждой ячейки таблицы вызывается функция fillCell, которой передаются адрес ячейки (номер строки и столбца) и текущее значение.
2. Функция fillCell вставляет значение в ячейку как фрагмент текста в окружении дескрипторов <SPAN>...</SPAN>. Событию ONCLICK дескриптора назначается вызов функции createSelector, которой передается адрес текущей ячейки.
3. Функция createSelector вставляет в ячейку по принятому адресу HTML-код раскрывающегося списка, возвращаемого функцией get_selector. Функция get_selector создает код списка с учетом адреса ячейки, в которую список добавляется. Адрес вновь используется в качестве аргумента знакомой вам функции fillCell — в этот раз она назначена событию списка ONBLUR.
4. Событие ONBLUR наступает при переходе от текущего элемента списка к любому другому элементу Web-страницы в результате щелчка мыши или нажатия клавиши <Tab>. При этом в функцию fillCell передаются адрес текущей ячейки и значение, выбранное в списке.
5. Функция fillCell замещает раскрывающийся список текстом выбранного значения, вставленного между дескрипторами <SPAN>...</SPAN> с установленным событием ONCLICK. Цикл замкнулся.

Пример

Давайте введем пять новых строк и заполним их так, как показано на рис. 7.15.

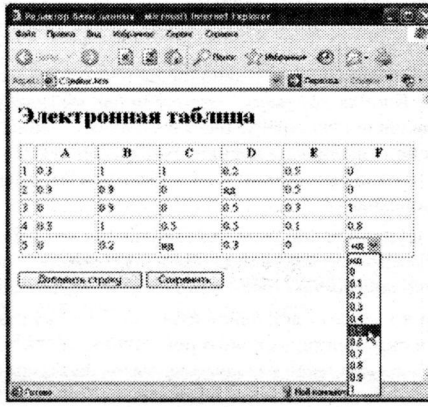



***Рис. 7.15.*** *Заполненная база данных*

Осталось сохранить базу данных щелчком на кнопке **Сохранить**. Эта кнопка вызывает на выполнение функцию `save_db`.

1. В первых строках кода функции происходит обращение к файлу Web-страницы, который хранится в свойстве документа `document.location.pathname`. Это значение принимается за основу для формирования пути к файлу `db.js`, который должен находиться в одной папке с файлом Web-страницы.
2. Далее в функции создается объект файловой системы компьютера

   ```
   var fso = new ActiveXObject("Scripting.FileSystem Object");.
   ```

   При первом обращении к этой функции обозреватель покажет окно предупреждения (см. рис. 7.13). Щелкните на кнопке **OK**, чтобы продолжить работу.

3. В следующей строке открывается текстовый файл командой

   ```
   var f = fso.CreateTextFile(path);.
   ```

   Метод `CreateTextFile` отличается от известного вам метода `OpenTextFile` тем, что он создает новый текстовый файл и сразу открывает его в режиме записи. Если такой файл уже существует, метод замещает его (изначально файл `db.js` в нашем примере отсутствует). Это вызвало бы ошибку в случае применения метода `OpenTextFile`.

4. С помощью метода `WriteLine` в файл записываются данные таблицы, после чего файл закрывается с использованием метода `close()`.

Зайдите в программе Проводник в папку с Web-страницей. Теперь там должен появиться новый файл `db.js`. Откроем его с помощью приложения Блокнот, как показано на рис. 7.16.

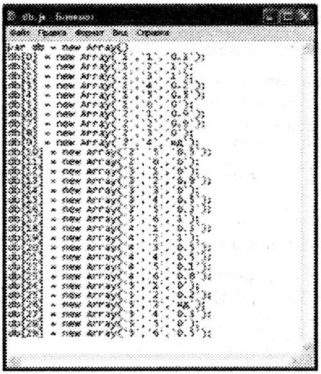



***Рис. 7.16.*** *Файл базы данных в окне приложения Блокнот*


В именах папок не должно быть пробелов, иначе методы `CreateTextFile` и `OpenTextFile` вызовут ошибку. Обязательно нужно закрыть объект файла методом `close()`, иначе он останется недоступным при попытке открыть его в следующий раз.

Обратите внимание, что данные записаны именно в том формате, какой мы использовали ранее в листинге 7.10. Закройте окно обозревателя с Web-страницей и откройте ее вновь. Теперь содержимое файла `db.js` будет добавлено как код внешнего сценария, и вы увидите все ваши данные в сохранности в таком виде, как на рис. 7.15. Можно снова отредактировать значения таблицы и сохранить изменения в файле базы данных.

На этом мы заканчиваем изучение основ программирования сценариев на JavaScript. Много полезных функций, методов, свойств и объектов остались за рамками этой книги. Если вас увлек JavaScript, имеет смысл не останавливаться на достигнутом и продолжить изучение этого простого, но мощного языка программирования. Воспользуйтесь справочниками и технической литературой, доступной в Интернет:

- `http://digkiller.narod.ru/man1/menu.htm`;
- `http://web.compdoc.ru/javascript.php`;
- `http://www.js-examples.com/javascript/core_js15/`;
- `http://www.dannyg.com/ref/jsquickref.html`.

# Предметный указатель

## Г



## Д



## З



## К



## Л



## М



## Н



## О

## П



## Р



## С



## Т

## Ф



## Ц



## Ч



## Э



## Я

# JavaScript

Если вы почувствовали, что вашим Web-страницам недостает динамичности, гибкости и собственного характера, обратитесь к сценариям. С помощью сценариев вы сможете сделать документ таким же «умным», как вы сами, вдохнуть в него жизнь и собственный характер. Сценарии – это небольшие программы, вписанные в HTML-код Web-страницы, а JavaScript – пожалуй, наиболее популярный язык написания сценариев для Web-страниц. Этот язык создавался для широкого круга разработчиков Web-страниц, не являющихся профессиональными программистами. Желательно, чтобы читатель имел представление о коде HTML Web-страниц. Впрочем, сведений об HTML, представленных в этой книге, будет достаточно для создания новичками небольших персональных Web-страниц.

ISBN 978-5-699-26260-1