

JavaScript

WEB-ПРОФЕССИОНАЛАМ



**Дэн Баррет, Дэн Ливингстон,
Мика Браун**

cisarte



по эту сторону - искусство

636352 *am*

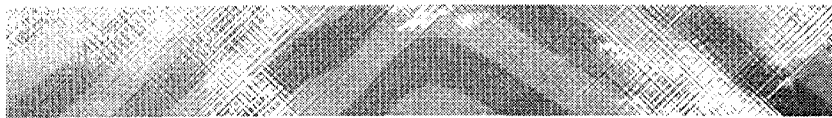


om



JavaScript

Web-профессионалам



Дэн Баррет, Дэн Ливингстон, Мика Браун

Essential JavaScript for Web Professionals

Dan Barrett

Clear Ink Corporation

Dan Livingston

Wire Man Productions

Micah Brown

Etail Enterprises

JavaScript

Web-профессионалам

Перевод с английского

Prentice Hall PTR
Upper Saddle River, NJ 07458
<http://www.phptr.com>



BHV, Киев, 2001

НБ ПНУС



636352

Д. Баррет, Д. Ливингстон, М. Браун

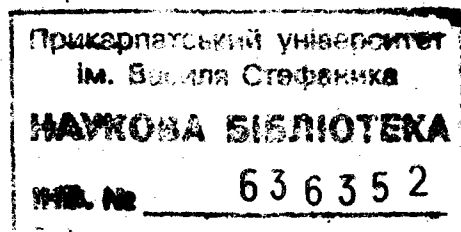
JavaScript. Web-профессионалам: Пер. с англ. — К.: Издательская группа BHV, 2001. — 240 с.

ISBN 966-552-081-4

ISBN 5-7315-0110-6

Книга рассчитана на читателей, знающих Web-технологии и язык разметки HTML, и станет хорошим практическим пособием в изучении языка JavaScript. Содержащиеся в ней примеры взяты из жизни, они иллюстрируют способы решения задач, которые очень часто возникают при создании реальных Web-узлов. Книга поможет вам получить твердые знания, на которые можно опереться в дальнейшей работе, и оценить возможности языка JavaScript.

Обложка А.А. Стеценко



ISBN 0-13-013056-7

© Prentice Hall PTR, 2000

ISBN 966-552-081-4

© Издательская группа BHV, Киев, 2001

ISBN 5-7315-0110-6

Содержание

Введение	9
Благодарности	15
Об авторах	17
Глава 1. Динамическое создание информационного наполнения	19
Проект I: Создание информационного наполнения для конкретной платформы	22
Вставка сценария в код HTML	22
Иерархия объектов JavaScript	24
Получение сведений о платформе и броузере	25
Динамическое создание кода HTML из сценария JavaScript	30
Проект II: Вывод информации об авторских правах и даты последнего изменения	33
Описание свойства lastModified	34
Динамический вывод данных, отличных от строковых	35
Глава 2. Сменяющиеся изображения	41
Проект I: Сценарий смены изображения	44
Создание объектов IMAGE	44
Функции смены изображения	50
Вставка обработчиков событий	53
Проект II: Расширение функциональных возможностей сменяющихся изображений	58
Вставка обработчиков событий	60
Создание объектов IMAGE	62

Вставка функций смены изображения	64
Некоторые особенности сменяющихся изображений	73
Глава 3. Применение JavaScript для навигации	75
Проект I: JavaScript и разворачивающиеся меню	78
Создание функции навигации	78
Вставка обработчика событий	79
Использование обработчика событий onChange для непосредственного перехода	81
Проект II: Применение нескольких разворачивающихся меню для навигации	84
Создание массивов	85
Создание функции многоступенчатого меню	90
Создание функции навигации	100
Вставка обработчиков событий	101
Проект III: Применение JavaScript на странице регистрации	108
Вставка обработчика событий	109
Глава 4. Обработка ошибок в формах	111
Проект I: Контроль пустых полей формы	113
Проверка формы на отсутствие пустых текстовых полей	117
Проверка выбора опции из разворачивающегося меню	121
Проект II: Контроль данных в полях формы	122
Проверка символов в адресах электронной почты	122
Проверка кода города в телефонном номере	123
Проверка наличия цифр в значении почтового индекса	126
Проект III: Уведомление пользователя об ошибках	132
Глава 5. Окна и кадры JavaScript	141
Общее представление об объекте WINDOW	143
Проект I: Создание, заполнение и закрытие окон	143
Создание и определение массивов	144
Создание функции	147
Вставка обработчиков событий	152
Проект II: Обмен информацией между кадрами в коде JavaScript	156
Определение иерархии кадров	157
Создание функции для проверки того, в каком разделе мы находимся	159
Вставка обработчика событий	162

Глава 6. Развитые средства JavaScript	167
Проект I: Создание подменю для вторичных страниц Stitch	169
Постановка задачи	169
Создание и заполнение массивов для хранения данных	174
Создание функций управления системой подменю	177
Вставка обработчиков событий	187
Приложение А. Обработчики событий	197
Приложение Б. Объекты JavaScript	209
Предметный указатель	235



Введение

Добро пожаловать в мир JavaScript! Мы сами хотели бы иметь перед собой такую книгу, приступая к изучению этого языка. Но в наше время в продаже имелись только два типа учебников: увесистые тома объемом более тысячи страниц, предельно сложные для восприятия, и небольшие брошюры, написанные донельзя просто и не содержавшие информации, которая могла бы пригодиться для практической работы. К сожалению, мы тогда не могли найти книги, достаточно информативные и в то же время способные научить пользоваться этой информацией при осуществлении реальных проектов.

Предлагаемый труд может стать вашим проводником в изучении JavaScript. Содержащиеся в нем примеры взяты непосредственно из ситуаций, которые очень часто возникают при построении Web-узлов. При этом от простых примеров мы с каждой главой переходим ко все более сложным.

А теперь подробнее о том, как построена эта книга, каким образом используются интерпретаторы, а также что позволяет и чего не позволяет делать язык JavaScript.

♦ Как построена эта книга

Вполне возможно, что по крайней мере некоторые из вас взялись за эту книгу после того, как руководитель, вызвав вас в свой кабинет, показал понравившийся ему Web-узел, созданный с использованием JavaScript. “Надо бы и нам сделать не хуже, — веско заметил он. — Сможете?” “Конечно, шеф”, — ответили вы. И подумали: “Надо поскорее изучить язык JavaScript!”

Чаще всего нам приходится осваивать новое именно так: мы получаем задание, и, если не знаем точно, как его выполнить, то стараемся побыстрее научиться. В данной книге описан ход реализации двух больших проектов, максимально подобных тем, что осуществляются в повседневной практике. В обоих случаях перед нами стоит задача создать или обновить Web-узел.

В первых трех главах мы будем обновлять начальную страницу быстро развивающейся биотехнологической компании Shelley Bio-technologies. При этом в каждой главе рассматривается, по меньшей мере, одна задача, выполняемая с использованием средств JavaScript, относящихся к категории простых или умеренно сложных. По завершении каждой главы вы получаете более сложное задание, которое должны выполнить самостоятельно для лучшего освоения материала. Во второй части книги описан процесс внесения существенных дополнений в Web-узел некоего журнала Stitch. Приводимые здесь примеры гораздо сложнее тех, что рассматривались в первом проекте, и позволяют продемонстрировать значительно более широкие возможности JavaScript.

Примеры во всех главах подобраны таким образом, чтобы помочь вам обрести твердые знания, на которые можно опереться в дальнейшей работе. Вы увидите, что в работе с JavaScript часто можно прийти к успеху разными путями. Об этом языке действительно можно сказать, что в нем все средства хороши.

Для тщательного изучения примеров книги вы можете перейти на Web-узел издательства, расположенный по адресу <http://www.phptr.com/essential>, и загрузить код HTML и все рисунки, необходимые для выполнения упражнений.

♦ Основные сведения о языке JavaScript

Что такое JavaScript

Для тех из вас, кто недавно занялся разработками для Web и изучает JavaScript в сочетании с HTML, может оказаться полезным следующее небольшое пояснение. JavaScript — это встроенный, мультиплатформенный, интерпретируемый язык, созданный компанией Netscape. Как и HTML, он совместим со всеми платформами.

JavaScript позволяет расширять функциональные возможности Web-страниц, встраивая прикладной код непосредственно в код HTML. С помощью этого языка можно создавать приложения с широким диапазоном: начиная от применения средств взаимодействия пользователей на Web-страницах и заканчивая обменом информацией с базами данных. Хотя JavaScript был создан компанией Netscape, он способен работать практически во всех современных браузерах, включая Internet Explorer (IE) компании Microsoft. Однако браузер IE непосредственно не поддерживает данный язык. Он имеет собственный интерпретируемый язык JScript, обладающий большинством возможностей JavaScript. Все немногочисленные случаи различного действия языков указаны в нашей книге, и мы предлагаем способы преодоления этих расхождений. Поскольку браузеры Netscape и IE получили самое широкое распространение, представляемые нами сценарии в основном предназначены для них.

Существуют два метода встраивания кода JavaScript в Web-страницы: включение операторов языка в код, выполняемый либо на клиентской, либо на серверной платформе. Оба метода содержат один и тот же базовый набор средств, определяемых ядром языка. В то же время каждый метод располагает и собственным, расширенным, набором языковых объектов и средств.

Как действует клиентский код JavaScript

Клиентские приложения JavaScript представляют собой сценарии, встроенные непосредственно в HTML-страницы и обрабатываемые браузером пользователя в процессе загрузки страницы. Не менее 90 % всех сценариев, применяемых в Web, относятся к подобной категории. Поэтому в данной книге описан именно этот метод.

Когда браузер пользователя вызывает HTML-страницу со встроенным в нее сценарием JavaScript, интерпретатор JavaScript браузера читает сценарий сверху вниз, выполняя операторы по мере их появления.

Одним из преимуществ применения клиентского сценария является то, что он разрешает распознавать события, вызванные действиями пользователя (например, ввод данных в форме или появление курсора мыши над конкретным графическим изображением). Затем, получая информацию о действиях пользователя, он может вызывать другие части сценария и выполнять иную работу, не обращая снова к Web-серверу и не требуя дополнительной информации. Поскольку сценарии выполняются в форме интерпретации браузером пользователя, следует сказать несколько слов и о том, чем отличается действие разных браузеров при обработке кода JavaScript.

Браузеры и версии

Как мы уже сказали, в браузерах IE и Navigator интерпретируемые языки реализованы немного по-разному. Жизнь программиста от этого становится сложнее, поскольку применяемые средства порой работают в разных браузерах неодинаково или вообще не действуют в одном из браузеров. Со временем ситуация усложняется еще больше, ведь с каждой новой версией браузеров Netscape и Microsoft набор их средств постоянно расширяется. А это значит, что сценарии, написанные с использованием новых средств, могут не работать в браузерах предыдущих версий. Но не стоит расстраиваться — все не так плохо, как кажется. Просто программист должен более внимательно определять набор средств и проверять свои сценарии. Существует много способов, позволяющих обеспечить применение сценариев на любых платформах. Мы их рассмотрим и объясним, когда их необходимо использовать. Но поскольку настоящая книга посвящена JavaScript, а не JScript, в ней в основном описан язык, разработанный компанией Netscape.

Что позволяет и чего не позволяет делать JavaScript

Хотя трудно даже себе представить все разнообразие приложений, которые могут быть созданы с использованием JavaScript, этот язык имеет и определенные ограничения. Например, он не разрешает получить доступ к компьютеру пользователя или контроль над ним. Из соображений защиты запись информации на ПК строго ограничена. Существует возможность записать данные только с помощью

файлов “cookie”, но даже и в этом случае приходится ограничиваться простыми текстовыми файлами. Такое ограничение позволяет защитить пользователей от сценариев, способных причинить вред их компьютерам, а также от несанкционированного доступа к конфиденциальной информации.

Существует и средство защиты, которое называют “правилом одинакового источника” (Same Origin Policy). Его суть в том, что сценарии из одного источника не могут получить доступ к определенным свойствам или файлам, принадлежащим другому источнику (находящимся по другому адресу). Например, если сценарий, расположенный по адресу `http://www.yoursite.com/test.html`, пытается получить доступ к определенным свойствам HTML-страницы, находящейся по адресу `http://www.theirsite.com/test.html`, то правило одинакового источника исключит такую возможность для данного сценария. Свойства, ограниченные этим правилом, показаны на рис. 1.1.

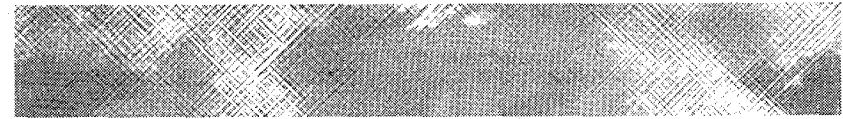
Объекты	Ограниченные свойства
images	src, lowsrc
document	anchors, applets, cookie, domain, elements, embeds, forms, fastModified, length, links, referrer, title, URL
layer	src
location	Все свойства, кроме x и y
window	find

Рис. 1.1. Ограничения, налагаемые правилом одинакового источника

Таковы основные ограничения, с которыми приходится сталкиваться, создавая приложения JavaScript. Безусловно, вы можете оказаться в аналогичной ситуации и при попытке применить объект или свойство для выполнения того, что они не позволяют выполнить. Но это уже будет скорее следствием ограничений не самого языка, а ваших знаний о нем.

Если вы в процессе работы с книгой обнаружите, что способны предложить какое-либо решение, отличающееся от нашего, проверьте и свой вариант. Возможно, он окажется абсолютно оригинальным.

А теперь приступим к изучению языка.



Благодарности

◆ Дэн Баррет

Я хочу обратиться со словами благодарности к Академии... Впрочем, извините, кажется, это не из той речи. Ну а если серьезно, я хотел бы поблагодарить свою жену Кристин за ее понимание и поддержку во время работы над данной книгой.

Спасибо моей семье — родителям Чаку и Джуди и брату Рику, которые позволяли мне не расставаться с книгами, пока я был ребенком. Я хотел бы также выразить свою признательность Дэну Ливингстону и Мике Брауну за то, что они предложили мне принять участие в этом проекте, а также Брэду Скотту, который подготовил технический обзор по материалам данного пособия.

И наконец, книга не была бы написана, если бы я не мог уединиться в этом чудесном уголке, пропитанном запахом кофе, который носит название Маунтин Дью.

◆ Дэн Ливингстон

Нам посчастливилось работать над этим проектом вместе с Марком Таубом и Карен Маклин из издательства Prentice Hall. Мы особенно оценили организаторский талант Карен, которая оставалась выдержанной и собранной на протяжении всего издательского процесса.

Я хотел бы поблагодарить свою невесту, Таню Мюллер, за неизменное терпение и помощь. Без нее я не смог бы написать эту книгу, создавая одновременно собственную компанию. Ее поддержка была и остается бесценной.

Я хочу также поблагодарить В. Брэдли Скотта из компании Clear Ink за его идею использовать в качестве проекта, на примере осуществления которого построена наша книга, интерактивный журнал мод. Кроме того, он выполнял функции технического рецензента и оказал нам очень большую помощь.

И, наконец, я хотел бы выразить признательность своему преподавателю по дизайну Бреду Эйгену из компании MadBoy Productions. Он — Наставник с большой буквы!

◆ Мика Браун

Я желал бы выразить особую признательность моей жене, Даун, которая помогала мне во всем. Она — моя любовь, моя жизнь и, что самое важное, мой лучший друг. Я посвящаю эту книгу ей и нашей дочери Эшли Нова, которая вскоре должна родиться. Мы ее очень-очень ждем!

Хочу также поблагодарить моих родителей, Уильяма и Донну, и моих крестных, Беппе и Джой, за все, что они для меня сделали за мои 29 лет. Без них я не смог бы стать тем, кем сейчас являюсь. Я буду всегда благодарен им за то, что они учили меня все эти годы и помогли мне стать личностью.

Спасибо Марку Таубу и Карен Маклин за помощь Дэну и мне в подготовке ряда книг этой серии. Вы были правы: это немного тяжелее, чем мы вначале думали! Кроме того, благодарю Карла Гормана, моего сотрудника из компании Etail Enterprises (www.etail.com), и своих друзей Келли и Карла из компании Nitrus за то, что они помогли мне пройти через все это.

И, наконец, спасибо моим соавторам Дэну Баррету и Дэну Ливингстону, а также всем тем, кто работал над книгами данной серии. Без них я не мог бы ничего этого написать.

Об авторах

◆ Дэн Баррет

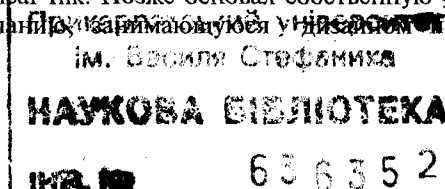
Работает программистом и дизайнером в преуспевающей компании из Сан-Франциско, среди клиентов которой — Hewlett-Packard, Novell и Pacific Bell. Постоянное времяпрепровождение — сидение перед монитором: если не колдует над Web-узлами, значит проводит исследования по компьютерной анимации.

В настоящее время Дэн Баррет живет в Калифорнии, в Силиконовой Долине, со своей женой Кристин.

◆ Дэн Ливингстон

Получил образование в области морской биологии. Web-дизайном начал заниматься с 1996 года. Создавал Web-узлы для таких крупных компаний, как Apple, Pacific Bell и Novell. Его работы завоевали многочисленные награды и были отмечены в литературе по дизайну и в передачах CNN Prime Time. Выполненный им на основе технологии DHTML узел Palette Map стал новым словом в Web-дизайне и получил международное признание, а также высокую оценку в номинации "Cool Site" ("великолепный узел") на узлах Yahoo!, Macromedia и USA Today.

Дэн Ливингстон работал Web-дизайнером и Web-программистом в корпорации Clear Ink. Позже основал собственную успешно действующую компанию, занимающуюся разработкой



пользовательского интерфейса Wire Man Productions. Продолжает выпускать книги серии Essential в издательстве Prentice Hall.

◆ Мика Браун

Отдав несколько лет печатному делу, с 1995 года начал свою карьеру в Web-индустрии в качестве программиста и дизайнера. Среди Web-узлов, которые он разработал, Dr. Laura, Pacific Bell, Amazing Discoveries и Ascend Communications.

В течение трех последних лет осуществлял также техническое рецензирование литературы, выпускаемой издательством Prentice Hall. В этом плане особенно следует отметить его работу с книгой “Perl by Example”, написанной Элли Квигли.

В настоящее время Мика Браун — совладелец консультационной Web-компании Etail Enterprises. Она расположена в южной части штата Калифорния и специализируется на оказании помощи организациям, желающим работать в новой для них области оперативной рекламы.

1 Динамическое создание информационного наполнения

В этой главе

- Проект I: Создание информационного наполнения для конкретной платформы
- Проект II: Вывод информации об авторских правах и даты последнего изменения
- Резюме
- Дополнительное задание

Итак, задание по программированию Web-узла для компании Shelley Biotechnologies выполнено. Однако динамика узла оставляет желать лучшего, и ваш новый руководитель поручил вам его усовершенствовать и сделать более конкурентоспособным. Прежде чем приступить к выполнению задания, вы должны разобраться с претензиями, которые возникают у некоторых посетителей. Выясняется, что пользователи Macintosh видят текст не так, как те, кто просматривает узел в операционной системе Windows (рис. 1.1). После внимательного исследования вы обнаруживаете, что шрифт используемого размера отображается на этих платформах по-разному. Тем не менее, хотя проблема не кажется вам столь уж значительной, ваш руководитель жаждет во всем добиться совершенства. Похоже, что возникает идеальная ситуация для применения JavaScript.

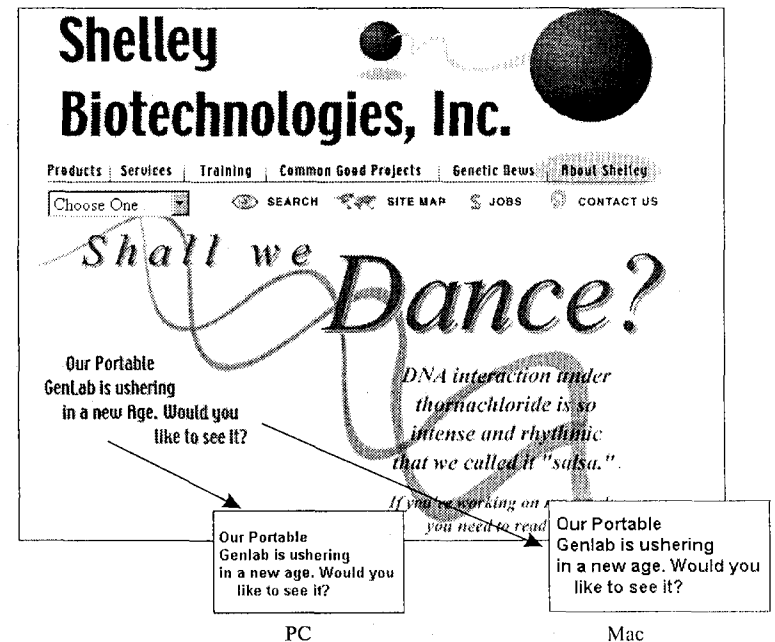


Рис. 1.1. Шрифт на разных платформах отображается по-разному

С помощью данного языка мы можем узнать, какая платформа и какой браузер применяются для просмотра узла, и направить пользователя наиболее подходящее для него информационное наполнение.

◆ Проект I: Создание информационного наполнения для конкретной платформы

Допустим, после экспериментов с имеющимся кодом HTML вы пришли к выводу, что страница будет выглядеть одинаково на обеих платформах, если применять шрифт размера 2 для Macintosh и шрифт размера 3 для системы Windows. Чтобы решить проблему, нужно записывать тег FONT в код HTML динамически, поэтому в том месте в теле HTML-страницы, где обычно находится тег FONT, должен быть помещен сценарий.

Этот первый сценарий будет состоять из двух частей: во-первых, нам надо узнать, на какой платформе работает пользователь, а во-вторых, мы должны динамически выводить в браузере разные коды для каждой из платформ.

Вставка сценария в код HTML

На первом этапе создания любого сценария JavaScript нужно сообщить браузеру, что код, размещенный на Web-странице, представляет собой не код HTML, а приложение JavaScript. Получив такое указание, браузер отправляет соответствующую информацию интерпретатору JavaScript, который выполнит задачу. В большинстве случаев мы будем осуществлять этот первый этап, заключая сценарии в тег <SCRIPT>:

```
<SCRIPT>
...
</SCRIPT>
```

Существует еще несколько способов ввода сценариев в код HTML:

- указание внешнего файла, содержащего сценарий;
- указание выражения JavaScript как значения для атрибута HTML;
- вставка сценариев в некоторые теги HTML в качестве обработчиков событий.

Способ, основанный на использовании тега <SCRIPT>, является наиболее распространенным. Мы в нашей книге иногда применяем и иные способы — при изучении работы кода JavaScript с изображениями и формами. Если браузер обнаруживает, что данный код находится за пределами тега <SCRIPT> или не применяется в одном из других упомянутых выше способов, то сценарий не станет работать, поскольку будет рассматриваться как текст, который должен быть выведен на экран.

С каждой последующей версией языка JavaScript его инструментальные средства дополняются новыми свойствами и выражениями. Если вы примените их в своих сценариях и при этом вашу страницу загрузит система с более старым браузером, вполне вероятно возникновение ошибки, которая вызовет неправильную работу сценария. Чтобы помешать этому, можно добавить атрибут к тегу <SCRIPT>, сообщая браузеру, для какой именно версии JavaScript предназначен написанный код. Когда браузер пользователя получит сценарий, он проверит, поддерживается ли им указанная версия, и если нет, то пропустит данный сценарий. С этой целью можно использовать атрибут LANGUAGE тега <SCRIPT>.

```
<SCRIPT LANGUAGE="JavaScript1.2">
...
</SCRIPT>
```

Данное дополнение к приведенному выше оператору сообщает браузеру, что он не должен обрабатывать сценарий, если в нем не предусмотрена поддержка версии 1.2 языка JavaScript. Этот способ позволяет скрыть код более поздней версии языка от браузеров, работающих с JavaScript. Возможно, однако, что некоторые пользователи имеют дело с браузерами еще более ранних версий, которые вообще не совместимы с изучаемым нами языком. Проблему можно решить, поместив весь код JavaScript между тегами комментария HTML следующим образом:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Теперь мы знаем, как ввести сценарий в код HTML. Однако прежде чем всерьез взяться за решение этой задачи, необходимо изучить иерархию объектов JavaScript.

Иерархия объектов JavaScript

По мере того, как браузер осуществляет загрузку HTML-страницы, машина обработки кода — интерпретатор JavaScript — автоматически формирует набор объектов на основе содержимого кода HTML. Они выстраиваются в виде определенной иерархии, которую затем можно использовать для вызова объектов и их свойств или ссылки на них (рис. 1.2).

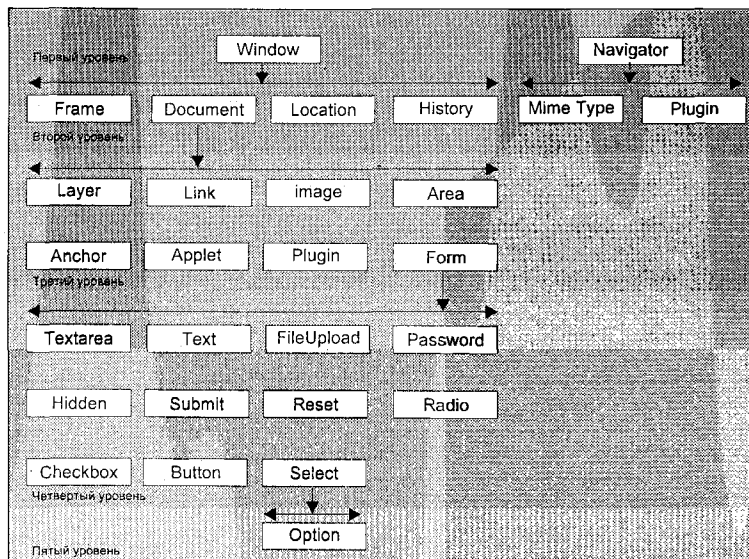


Рис. 1.2. Иерархия объектов JavaScript

Объект WINDOW находится на вершине данной иерархии. Он фактически представляет собой окно, которое показывает все информационное наполнение, отображаемое браузером. Потомки объекта WINDOW являются его свойствами и сами выступают объектами, способными иметь потомков. Например, если на странице есть изображение с именем product, то product — это объект, относящийся к типу изображения, но он также — свойство объекта DOCUMENT, который в свою очередь является свойством объекта WINDOW. Без понимания этой иерархии, ее объектов и свойств, невозможно

создавать приложения в коде JavaScript. Справочник по всем объектам языка и их свойствам вы найдете в приложении Б.

Чтобы обратиться к объекту в коде JavaScript, нужно вызвать сам этот объект и все объекты, иерархически расположенные над ним. Вот пример вызова объекта product:

```
document.product
```

Обращение к конкретному свойству любого объекта аналогично, надо лишь сделать один шаг вниз по иерархии. Например, вызов свойства с обозначением источника изображения product выглядит так:

```
document.product.src
```

Обратите внимание: объект WINDOW не был указан в предыдущих примерах как первый. Если ссылка на конкретное окно отсутствует, JavaScript автоматически предполагает, что здесь имеется в виду окно, в котором загружен документ. Точное указание требуется только в том случае, когда предпринимается попытка обратиться к объектам, расположенным в другом окне или кадре.

Итак, теперь вы имеете хотя бы основное представление об иерархии объектов JavaScript. По мере ознакомления с другими примерами, приведенными в этой книге, вы сможете существенно пополнить багаж ваших знаний.

Получение сведений о платформе и браузере

Мы можем приступить к написанию сценария. Первым делом нужно обратиться к свойству платформ объекта NAVIGATOR, и значение этого свойства сообщит нам, на какой платформе работает пользователь. Теперь оно будет применяться для определения в операторе if размера необходимого на странице шрифта.

Оператор if — условный. Он сообщает интерпретатору, что при выполнении определенного условия тот должен реализовать код, который указан за ним в фигурных скобках. В ином случае интерпретатору надо пропустить данный код и перейти к следующему оператору.

В большинстве случаев сценарии размещаются в разделе <HEAD> HTML-страницы. Однако при динамическом создании текста необходимо ввести, по меньшей мере, часть сценария в тело HTML-страницы, туда, где должен находиться данный текст. В нашем проекте мы поместим сценарий в то место документа HTML, где будет

располагаться текст, замена которого должна происходить динамически. Приводим код HTML с указанием места размещения спенария.

[illegible]

Теперь, определив место расположения сценария в коде HTML, вставим его первый фрагмент.

```
<SCRIPT LANGUAGE="JavaScript"> '  
<!-- Бrowsers ранних версий игнорируют следующий код  
// Присвоить тип платформы переменной  
var platform = navigator.platform.substr(0,3);  
...  
// Здесь сокрытие кода прекращается -->  
</SCRIPT>
```

Рассмотрим этот фрагмент строка за строкой и разберемся, что в нем происходит. В начале сценария размещен открывающий тег `<SCRIPT>` (как было описано ранее). Для данного сценария будет применяться только код, поддерживаемый всеми версиями JavaScript, поэтому атрибут `LANGUAGE` тега `<SCRIPT>` можно не использовать. За тегом `<SCRIPT>` находится строка комментария HTML, который скрывает код от браузеров, не поддерживающих JavaScript. С этих двух первых строк будут начинаться все создаваемые нами сценарии, поэтому они должны стать для вас привычными.

Третья строка представляет собой комментарий JavaScript. Как и в HTML, мы можем вносить заметки в свой код, чтобы сделать сценарии более удобными для сопровождения. Существуют два способа ввода комментария в код JavaScript. Первый (который мы здесь

и используем) заключается в размещении в начале строки кода двух прямых слэшей.

```
// Эта строка рассматривается как комментарий JavaScript
```

При таком способе любой текст, следующий за прямыми слэшами в данной отдельной строке, рассматривается как комментарий, а не интерпретируется как код, предназначенный для выполнения.

Второй способ применяется, когда нужно вставить комментарий, занимающий несколько строк. Для этого следует заключить его между двумя парами символов /* и */. Например:

```
/* Это пример комментария, который
   занимает несколько строк программы */
```

Далее в ходе разработки сценариев мы будем использовать комментарии JavaScript для обозначения главных разделов кода и указания их основного назначения.

После ввода комментария можно перейти к основной части сценария. В четвертой строке для присвоения значения свойству переменной `platform` применяется простой оператор присваивания (`=`). В этой строке кода есть несколько новых конструкций, поэтому рассмотрим ее более внимательно. Прежде всего, для установки значения правого операнда в качестве значения левого операнда применяется оператор присваивания. С обеих сторон от знака равенства происходит еще кое-что: слева от него впервые создается переменная. Переменные играют очень важную роль в разработке программ любого типа, и язык JavaScript не составляет исключения. Переменную правомерно считать контейнером с произвольным именем, который позволяет хранить информацию.

После создания переменной можно просто вызывать ее по имени для доступа к содержащейся в ней информации или переназначать ей для хранения свежую информацию. Для образования новой переменной применяется следующий синтаксис:

```
var variablename;
```

В левой части оператора создается переменная с именем platform. Теперь она готова принять значение, которое ей передается из правой части оператора. В правой части выполняются два действия. Прежде всего мы вызываем свойство platform объекта NAVIGATOR. Данное свойство хранит строковое значение, позволяющее узнать, на какой платформе работает пользователь. Если это — ПК с операционной системой Windows, то свойство navigator.platform

получит значение либо “Win32”, либо “WinNT”. Если пользователь работает на компьютере Macintosh, свойство возвратит значение “Macintosh”.

Нас в действительности не интересует, с какой версией Windows работает пользователь. Поэтому было бы просто замечательно, найди мы способ сократить значение, возвращаемое с компьютеров с Windows, до первых трех символов — “Win”. В таком случае переменной `platform` можно было бы присвоить единственное значение, не зависящее от используемой версии ОС. К счастью, такой способ есть — это второе действие, выполняемое в правой части оператора. Атрибут `substr()` возвращает указанное число символов из любой заданной строки. Поскольку известно, что первые три буквы, возвращаемые из всех версий Windows, являются одинаковыми, для нас это идеальное решение. Перед вами синтаксис метода `substr()`:

```
string.substr(start, length);
```

Здесь `start` обозначает позицию начального символа, а `length` — число символов, которое должен вернуть `substr()`. В приведенной выше строке мы сообщаем этому методу, чтобы он стартовал с первого символа и прошел до третьего. Теперь оператор присваивания поместит в переменную `platform` либо значение “Win” для компьютеров с Windows, либо значение “Mac” для компьютеров Macintosh.

Итак, у нас есть уже вся информация, необходимая для создания следующей части кода. А для проверки значения переменной `platform` будет применяться оператор `if ... else`. Рассмотрим синтаксис этого оператора.

```
if (condition1)
{
    statements1
}
else if(condition2)
{
    statements2
}
else
{
    statements3
}
```

Здесь `condition1` и `condition2` представляют собой любое выражение, возвращающее значение “истина” или “ложь”. Если условие `condition1` является истинным, то интерпретатор JavaScript выполнит код, содержащийся в блоке `statements1`. Однако, если `condition1` оказывается ложным, интерпретатор пропустит блок операторов `statements1` и перейдет к проверке условия `condition2`. Если `condition2` будет истинным, интерпретатор выполнит операторы `statements2`. Если же обнаружится, что и `condition1`, и `condition2` ложные, то выполняться станут операторы `statements3`. Оператор `if` является одной из наиболее часто применяемых конструкций языка JavaScript, и вы будете регулярно с ним встречаться на протяжении этой книги.

Перейдем к следующему разделу кода, содержащему оператор `if`.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Присвоить тип платформы переменной
var platform = navigator.platform.substr(0,3);
// С помощью оператора if осуществляется проверка того, на
// какой платформе работает пользователь
if (platform == "Win")
{
    ...
}
else if (platform == "Mac")
{
    ...
}
else
{
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В операторе `if` мы указываем интерпретатору JavaScript, чтобы он вначале проверил значение переменной `platform` и, если она равна строке “Win”, выполнил операторы, следующие за выражением `if` и заключенные в фигурные скобки. Если значение переменной `platform` не равно “Win”, интерпретатор переходит к строке `else if`, в которой проверяется равенство переменной строке Mac. Обнаружив соответствие, интерпретатор переходит к выполнению операторов, следующих за выражением `else if` и заключенных в фигурные скобки. Не установив соответствие строкам Win или Mac, оператор `if` выполнит код, находящийся за строкой `else`.

Обратите внимание: для сравнения значений при проверке условий в языке JavaScript применяется двойной знак равенства (==). Если бы мы в этом случае использовали одинарный знак равенства, интерпретатор рассматривал бы данный оператор как оператор присваивания и переустанавливал значение переменной platform.

Мы создали основную структуру сценария, определяющего, какой браузер применяется для просмотра страницы. Теперь осталось только вставить код, который браузер должен выполнять для каждой из платформ.

Динамическое создание кода HTML из сценария JavaScript

Поскольку наша цель — добиться одинакового результата при применении шрифтов разных размеров на различных платформах, необходимо записывать тег динамически с помощью сценария JavaScript. Для вывода данных в документ HTML можно использовать два оператора: document.write() и document.writeln(). Оба они выводят всю информацию, переданную им в качестве параметра, но document.writeln() вставляет символ перевода строки в конце показанного им текста. Обычно можно использовать любой из названных методов, но в нашем сценарии будет задействован document.writeln(), чтобы после вывода исходного кода HTML-страницы она в результате выполнения операторов была отформатирована для удобства чтения.

Можно применять эти команды для показа информации разных типов. Все, что заключено в кавычки внутри круглых скобок, будет выведено в виде строки, а в качестве параметра допустимо также задавать переменные или даже другие команды JavaScript, которые будут выполнены перед отправкой их результата в документ. В данном случае вам нужно только распечатать строку, содержащую тег . Вставим document.writeln() в описанные выше операторы if.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Присвоить тип платформы переменной
var platform = navigator.platform.substr(0,3);
// С помощью оператора if осуществляется проверка того,
// на какой платформе работает пользователь
if (platform == "Win")
{
    document.writeln("<FONT SIZE=\"3\">");
}
```

```
}
else if (platform == "Mac")
{
    document.writeln("<FONT SIZE=\"2\">");
}
else
{
    document.writeln("<FONT SIZE=\"3\">");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Таким образом, если пользователь работает на компьютере с системой Windows, для него будет выведен тег с атрибутом SIZE, равным 3. На компьютере Macintosh атрибут примет значение 2. Если же не используется ни одна из этих платформ (т.е. на компьютере установлена какая-то из версий Unix), выводится тег с размером шрифта 3. Обратите внимание: в операторе document.writeln() перед кавычками добавлены символы обратного слэша (или просто обратные слэши), которые окружают числовое значение атрибута FONT SIZE. Обратные слэши помещены здесь, чтобы сообщить оператору document.writeln(), что он должен фактически вывести эти кавычки, а не рассматривать их как собственные кавычки выражения, переданного в качестве параметра. Такой прием, называющийся “применением управляющего символа”, позволяет указать интерпретатору JavaScript, чтобы он воспринимал собственное значение зарезервированных символов, а не пытался их обработать.

Описание сценария

Поздравляем, вы написали свой первый сценарий. Он даст возможность доставлять разное информационное наполнение пользователям различных браузеров. Рассмотрим его в последний раз и вкратце напомним, какие действия он выполняет.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Присвоить тип платформы переменной
var platform = navigator.platform.substr(0,3);
// С помощью оператора if осуществляется проверка того,
// на какой платформе работает пользователь
if (platform == "Win")
{
    document.writeln("<FONT SIZE=\"3\">");
}
```

```

else if (platform == "Mac")
{
    document.writeln("<FONT SIZE=\"2\">");
}
else
{
    document.writeln("<FONT SIZE=\"3\">");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Ниже перечислены этапы создания сценария:

1. Установлено значение переменной для хранения определенной части значения объекта navigator.platform с использованием метода substr().
2. Значение переменной platform применяется для проверки в операторе if и определения того, какие коды должны быть в нем выполнены.
3. Для динамического вывода соответствующего тега применяется оператор document.writeln().

В данном сценарии было впервые описано несколько новых средств языка JavaScript:

- вставка сценариев в документы HTML;
- сокрытие сценариев от браузеров, не поддерживающих JavaScript;
- иерархия объектов JavaScript;
- ввод комментариев в код JavaScript;
- создание переменной и использование оператора присваивания для передачи ей значения;
- применение метода substr();
- синтаксис и использование оператора if ... else;
- операторы document.write и document.writeln;
- применение обратного слэша (\) в качестве управляющего символа для зарезервированных символов.

◆ Проект II: Вывод информации об авторских правах и даты последнего изменения

Поскольку ваш руководитель убедился, что вы можете динамически изменить информационное наполнение Web-страницы, он приготовил для вас новый проект. Теперь вам надо разместить в нижней части каждой страницы информацию об авторских правах и дату последнего изменения (рис. 1.3). Добавление даты представляется особенно полезным на Web-узле компании Shelley. В страницы узла вносят коррективы очень многие, и было бы неплохо располагать простым способом узнать, трансформировался ли файл с того времени, как вы последний раз его редактировали. Как всегда, задача, о существовании которой всего лишь двадцать минут назад никто, в том числе и ваш босс, даже не подозревал, вдруг становится самой главной, причем решить ее надо немедленно. Итак, приступим к работе.

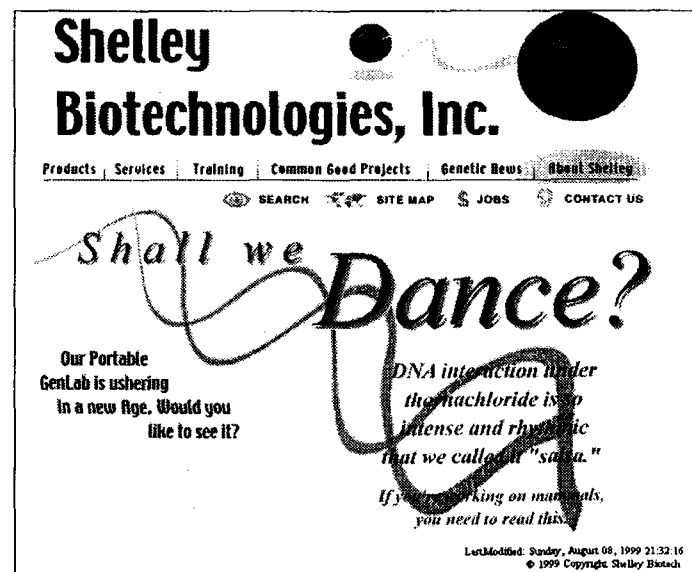


Рис. 1.3. Размещение даты последнего изменения и информации об авторских правах

Описание свойства lastModified

После краткого знакомства с различными возможностями объекта DOCUMENT вы заметите свойство, позволяющее значительно упростить написанный сценарий. Судя по всему, отнюдь не нам первым пришла в голову мысль о том, что было бы неплохо разместить на странице дату последнего изменения. Создатели языка JavaScript, будучи людьми умными и предусмотрительными, включили в объект DOCUMENT свойство lastModified. В процессе загрузки страницы и построения иерархии JavaScript интерпретатор переходит к заголовку HTTP, получает дату последней трансформации страницы и назначает ее этому свойству. Однако описанный способ имеет один недостаток: не все Web-серверы включают такую информацию в заголовок, поэтому мы должны воспользоваться оператором if, чтобы вывести ее только в том случае, если данное значение указано. Но выбирать не приходится, и к тому же подготовка еще одного оператора if позволит нам приобрести дополнительный опыт.

Если Web-сервер не предоставляет дату, то свойство lastModified будет иметь значение 0. Это значение можно использовать для проверки наличия даты последнего изменения в операторе if. Прежде всего рассмотрим тот раздел кода HTML, в котором разместится сценарий.

```
<TR>
  <TD COLSPAN="3" ALIGN="RIGHT" VALIGN="TOP">
    <FONT SIZE="-2">
      **** ЗДЕСЬ ДОЛЖЕН НАХОДИТЬСЯ СЦЕНАРИЙ ****
    </FONT>
  </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Теперь, когда мы выяснили, где должен находиться код, начнем со вставки тегов <SCRIPT> и комментариев HTML, которые скроют этот сценарий от браузеров более ранних версий.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Первый шаг в сценарии предусматривает создание переменной и присвоение ей свойства lastModified.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Присвоить дату последнего изменения переменной lastmoddate
var lastmoddate = document.lastModified;
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Поскольку у нас уже есть переменная с именем lastmoddate, хранящая значение свойства lastModified, подготовим оператор if для проверки того, действительно ли это свойство содержит дату.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Присвоить дату последнего изменения переменной lastmoddate
var lastmoddate = document.lastModified;
// Создать оператор if для проверки значения lastmoddate
if(lastmoddate == 0)
{
  ...
}
else
{
  ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Оператор if в приведенном коде очень похож на тот, что использовался в первом сценарии данной главы. Однако он немного проще, поскольку его структура состоит только из выражений if ... else. Поэтому, если значение lastmoddate равно 0, то будут выполнены операторы, которые следуют за ним. Если же эта переменная будет иметь иное значение, оператор if пропустит блок операторов, содержащийся в выражении if, и выполнит коды, которые следуют за выражением else.

Теперь, после подготовки оператора if, определим, что нам нужно вывести на страницу в качестве кода HTML.

Динамический вывод данных, отличных от строковых

В первом сценарии главы для вывода конкретной строки в качестве кода HTML применялся метод document.writeln().

Остановимся на нем более подробно, поскольку в том сценарии, над которым мы работаем сейчас, этот уже знакомый нам метод следует использовать для вывода строковых и нестроковых данных. Однако вначале мы будем применять его при выводе необходимого информационного наполнения в том случае, если в свойстве `lastModified` не обнаружена дата. При таком условии мы должны показать две строки: в первую очередь — строку с сообщением о том, что дата последнего изменения неизвестна. Затем следует вывести в документ тег перевода строки HTML и вторую строку, содержащую информацию об авторских правах.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Присвоить дату последнего изменения переменной lastmoddate
var lastmoddate = document.lastModified;
// Создать оператор if для проверки значения lastmoddate
if (lastmoddate == 0)
{
    document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
else
{
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Предусмотрев данные действия, надо вставить код, который будет выводить в документ информационное наполнение, необходимое в том случае, если в свойстве `lastModified` содержится дата. Как было сказано выше, когда речь идет о `document.writeln()`, мы вступаем на не совсем знакомую нам территорию. До сих пор этот метод применялся только для вывода строки. Теперь же он будет использоваться для печати комбинации строковых данных и значения, содержащегося в переменной `lastmoddate`. Это можно выполнить с помощью следующей строки кода:

```
document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech")
```

Для объединения разных типов данных может использоваться оператор конкатенации (+). Он принимает две или несколько строк и соединяет их в одну новую. Несмотря на то, что сама переменная `lastmoddate` не является строкой, хранящаяся в ней значение может

быть преобразовано в строку, поэтому оператор конкатенации будет работать. Таким образом, в предыдущей строке мы вначале объединяем свойство `LastModified` со значением, которое хранится в переменной `lastmoddate`, а затем объединяем эту новую строку со строкой, содержащей тег перевода строки HTML и информацию об авторских правах. Рассмотрим, как эта строка программы выглядит в остальной части сценария:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Присвоить дату последнего изменения переменной lastmoddate
var lastmoddate = document.lastModified;
// Создать оператор if для проверки значения lastmoddate
if (lastmoddate == 0)
{
    document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
else
{
    document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Описание сценария

Последняя добавленная строка представляет собой заключительный фрагмент, необходимый для завершения программы. Итак, у нас есть сценарий, который динамически выводит дату последнего изменения и информацию об авторских правах на всех страницах, где он размещен. Теперь, как и после выполнения каждого задания нашей книги, рассмотрим законченный сценарий, опишем, как мы его получили и с какими новыми свойствами при этом ознакомились.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Присвоить дату последнего изменения переменной lastmoddate
var lastmoddate = document.lastModified;
// Создать оператор if для проверки значения lastmoddate
if (lastmoddate == 0)
{
    document.writeln("Lastmodified: Unknown<BR>&copy; 1999
Copyright Shelley Biotech");
}
}
```

```

else
{
document.writeln("LastModified: " + lastmoddate +
"<BR>&copy; 1999 Copyright Shelley Biotech");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Перечислим этапы создания сценария:

1. Переменной `lastmoddate` присвоено значение свойства `document.lastModified`.
2. Разработан оператор `if` для проверки того, действительно ли Web-сервер передал дату последнего изменения интерпретатору JavaScript.
3. Выполнена вставка оператора с использованием свойства `document.writeln()`, выводящего информацию о том, что дата последнего изменения неизвестна, а также об авторских правах, если в переменной `lastmoddate` не содержится дата.
4. Подготовлен оператор, в котором используется метод `document.writeln()` для вывода сочетания строковых данных и информации о дате, содержащихся в переменной `lastmoddate` наряду с информацией об авторских правах, если указана дата последнего изменения.

А вот новые свойства, применявшиеся в ходе выполнения проекта:

- свойство `lastModified` объекта `DOCUMENT`;
- способ объединения строковых и нестроковых значений с использованием оператора конкатенации (+).

◆ Резюме

Итак, мы успешно приступили к внесению изменений в Web-узел компании Shelley Biotechnologies. Достаточно оперативно в начальную страницу были введены два новых очень полезных свойства. Дело в том, что по мере появления на рынке все более мощных браузеров, отличающихся некоторыми особенностями в обработке HTML-страниц, возможность получить от пользователя информацию и настроить страницы с учетом специфики его платформы приобретает исключительно важное значение. В этой главе мы также начали ознакомление с технологией динамического создания информационного наполнения. Именно в данной области JavaScript имеет

наибольшее преимущество перед другими языками с точки зрения существенного повышения эффективности поддержки и сопровождения Web-узла. В процессе дальнейшего изучения этой книги вам встретится много примеров реализации возможностей JavaScript в данной сфере.

◆ Дополнительное задание

Сценарии, написанные нами в главе 1, — довольно простые примеры того, чего можно достичь с помощью определения версии платформы и браузера, а также динамического создания HTML. Попробуйте применить полученные знания в других проектах, чтобы закрепить приобретенные вами навыки.

1. Воспользуйтесь средствами получения сведений о браузере для динамического создания страниц, которые бы могли настраиваться на работу с конкретными версиями браузера.
2. Напишите страницу, умеющую собирать всю информацию, содержащуюся в объекте `NAVIGATOR`, и динамически создавать таблицу, отображающую эту информацию для пользователя. (Информация, полученная с браузера пользователя и помещенная в таблицу, удобна для применения в качестве средства отладки.)
3. Если у вас есть узел, в котором на всех страницах появляется определенное информационное наполнение, примените метод `document.writeln()` для вывода этого кода, а затем сохраните сценарий во внешнем файле JavaScript. На каждой странице просто вызывайте внешний файл с использованием атрибута `SRC` тега `<SCRIPT>`. Это позволит вам обновлять только один файл для внесения изменений, относящихся ко всему узлу.

2 Сменяющиеся изображения

В этой главе

- Проект I: Сценарий смены изображения
- Проект II: Расширение функциональных возможностей сменяющихся изображений
- Некоторые особенности сменяющихся изображений
- Резюме
- Дополнительное задание

Показав свои недюжинные способности, вы вызвали цепную реакцию. Ваш руководитель жаждет столь же удачного продолжения и требует сделать узел еще более динамичным и “крутым”. Он ознакомился с узлами конкурентов и обнаружил, что на них есть сменяющиеся изображения JavaScript. Они применяются для замены одной версии изображения (ничем не выделяющейся, то есть “не-подсвеченной”) другой (оформленной иным образом, то есть “подсвеченной”). Эта замена происходит при установке курсора над изображением. Когда же пользователь убирает курсор, новая версия изображения заменяется первоначальной. Такой способ широко используется на Web-страницах, и ваш шеф решил, что узлу Shelley не хватает именно этой особенности.

Прежде всего нужно определить, для каких графических файлов следует применить технику смены изображения. Для большинства страниц лучшим местом использования сменяющихся изображений является основная навигационная панель, и узел Shelley Biotech не составляет исключения (рис. 2.1).



Рис. 2.1. Начальная страница Shelley Biotechnologies

Один из дизайнеров компании уже придумал превосходную графическую трактовку для нового эффекта (рис. 2.2), и теперь задержка только за вами.

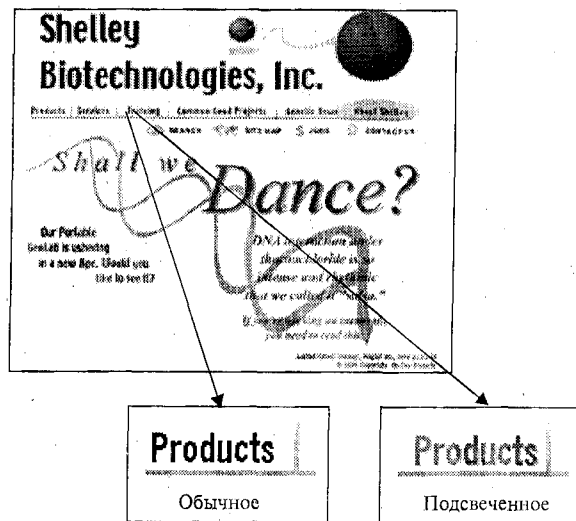


Рис. 2.2. Пример сменяющегося изображения

Создание сменяющегося изображения осуществляется в три этапа: определение объектов IMAGE; построение выполняющей функции; ввод необходимых обработчиков событий JavaScript в теги изображения и якоря.

◆ Проект I: Сценарий смены изображения

Создание объектов IMAGE

Прежде чем перейти к фактическому созданию самих объектов IMAGE, необходимо поместить тег <SCRIPT> на HTML-страницу.

В отличие от кодов, написанных в предыдущей главе, здесь нам не нужно выводить информационное наполнение в документ на странице, поэтому поместим сценарий в раздел HEAD HTML-страницы.

```
<HEAD>
<SCRIPT Language="JavaScript">
```

```
<!-- Бrowsers ранних версий игнорируют следующий код
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Теперь нам нужно скрыть объект IMAGE, необходимый для сценария смены изображения, от браузеров более ранних версий, не поддерживающих этот объект. В первой главе было показано, как использовать JavaScript для сбора информации о браузере и платформе. В данном случае можно применить вариант этого метода, в котором устанавливаются несколько операторов if для определения каждого браузера, не поддерживающего IMAGE. Но такой способ связан с большим объемом работы. К счастью, есть более простой выход. Мы можем проверить, поддерживает ли браузер пользователя сменяющиеся изображения, применив единственный оператор if, как показано в следующих строках кода:

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
// Создание объектов IMAGE
if (document.images)
{
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

В приведенном коде выражение

```
document.images
```

используется в качестве условия оператора if. Оно возвращает значение истина, если браузер поддерживает объект IMAGE, и значение ложь — в обратном случае. Вставляя код в этот оператор, мы фактически скрываем его от браузеров, которые не могут выполнить данный сценарий.

В процессе загрузки HTML-страницы и создания в браузере объектов, из которых состоит иерархия JavaScript, каждое изображение, выведенное на страницу, превращается в объект и помещается в массив с именем images. Здесь мы впервые встречаемся с массивами, поэтому рассмотрим, что это такое вообще, что собой представляет, в частности, массив images, и как мы можем им воспользоваться.

Массив можно сравнить с каталожным блоком. Предположим, что у вас есть страница с четырьмя изображениями, и, по мере того, как браузер читает файл HTML, он продвигается по ней вниз. Встретив первое изображение, он создаст для него объект IMAGE и сохраняет его в первом “ящике” каталожного блока. Дойдя до второго изображения, он опять создаст новый объект IMAGE и помещает его во второй “ящик”. Подобные действия повторяются до тех пор, пока каждое изображение на странице не получит свой собственный объект IMAGE, хранящийся в отдельном элементе массива.

Когда браузер должен сослаться на изображение, он открывает соответствующий “ящик”. Вы также можете сослаться на данный массив и хранящиеся в нем объекты. Осуществляется это одним из двух способов. Первый заключается в ссылке на то место в массиве, где находится требуемый объект IMAGE. Его недостаток связан с тем, что при добавлении на страницу еще одного изображения изменяется положение всех других следующих за ним изображений. Так что вам придется затем пройти по всему сценарию и убедиться, что вызываемое изображение не переместилось в другую позицию.

Второй способ свободен от такого недостатка. Суть его в том, что если вы в тегах задаете атрибут NAME, то обращаетесь к графическому изображению в массиве по имени, которое вы ему присвоили. При использовании данного способа вы всегда сможете сослаться на объект IMAGE одинаковым образом, даже после размещения на странице дополнительных графических изображений, и при этом сценарий не нужно будет корректировать.

Существуют шесть навигационных изображений, на которые будет воздействовать наш сценарий смены, поэтому каждому из них нужно присвоить атрибут NAME в тегах .

```
<A HREF="products/index.html"><IMG
src="images/h_products_off.gif"
WIDTH="71" HEIGHT="33" BORDER="0" NAME="Products"
ALT="Products"></A>
```

```
<A HREF="services/index.html"><IMG
src="images/h_services_off.gif"
WIDTH="67" HEIGHT="33" BORDER="0" NAME="Services"
ALT="Services"></A>
```

```
<A HREF="training/index.html"><IMG
src="images/h_training_off.gif"
WIDTH="75" HEIGHT="33" BORDER="0" NAME="Training"
ALT="Training"></A>
```

```
<A HREF="common/index.html"><IMG src="images/h_common_off.gif"
WIDTH="157" HEIGHT="33" BORDER="0" NAME="Common"
ALT="Common Good"></A>
```

```
<A HREF="genetic/index.html"> <IMG src="images/h_news_off.gif"
WIDTH="98" HEIGHT="33" BORDER="0" NAME="News"
ALT="Genetic News"></A>
```

```
<A HREF="about/index.html"> <IMG src="images/h_about_off.gif"
WIDTH="106" HEIGHT="33" BORDER="0" NAME="About"
ALT="About Shelley"></A>
```

Итак, атрибут NAME добавлен к каждому из шести тегов . Имя любого из них является первым словом категории, представленной этим графическим изображением. Обратите внимание, что начальные буквы имен прописные — язык JavaScript чувствителен к регистру. Теперь, после назначения имен, мы можем перейти к созданию некоторых новых объектов IMAGE, необходимых для сценария.

Поскольку изображения, которые должны появляться, когда пользователь проводит курсором над одним из шести пунктов меню, не размещаются явно на странице в коде HTML, мы должны создать два новых объекта IMAGE для каждого из сменяемых образов. Это повлечет за собой добавление изображений к массиву images и позволит нам обращаться к их свойствам. Начнем с построения объектов IMAGE для изображения Products.

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Создание объектов IMAGE
if (document.images)
{
ProductsOn=new Image(71, 33);
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

В приведенном коде мы инициализировали новый объект IMAGE, используя конструктор изображения со следующим синтаксисом:

```
new Image(width, height);
```

Поместив конструктор изображения с правой стороны от оператора присваивания (=) и указав с левой стороны имя, которое мы хотим дать объекту IMAGE, мы только что создали новый объект IMAGE с именем ProductsOn.

Правильный выбор имен этих новых объектов очень важен для работы сценария. Начало имени должно совпадать с именем соответствующего изображения, уже находящегося на странице. Например, в первой строке, где мы создали объект для хранения позиции варианта изображения Products, который должен появляться при прохождении курсора над пунктом меню, имя начинается с Products. А поскольку изображение показывается после того, как курсор проходит сверху, то вторая часть имени — On. Объединив эти два компонента имени, мы получаем новый объект с именем ProductsOn. Для каждого из шести навигационных изображений мы должны иметь не только объект, содержащий вариант, появляющийся при перемещении над ним курсора, но и объект для хранения обычного варианта изображения. Имя второго объекта для наших изображений продуктов также начинается со слова Products, но оканчивается суффиксом Off. Ниже показано, как вставить этот новый объект IMAGE после первого.

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Создание объектов IMAGE
if (document.images)
{
ProductsOn=new Image(71, 33);
...
ProductsOff=new Image(71, 33);
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Если мы нарушим указанное соглашение о наименовании, то функции, которые будут описаны ниже, не смогут определить, какой объект IMAGE должен применяться для замены, и возникнут ошибки. Теперь у нас есть два новых объекта IMAGE, необходимые для смены изображения Products, но на данный момент они пусты. Надо назначить им реальные изображения. Для этого можно присвоить адрес (URL) изображения свойству источника (src) объекта IMAGE:

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
if (document.images)
{
ProductsOn=new Image(71, 33);
ProductsOn.src="images/products_on.gif";
```

```
ProductsOff=new Image(71, 33);
ProductsOff.src="images/products_off.gif";
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Сделано все, что необходимо для создания объектов On и Off, применяемых с целью смены изображения Products. Но нужно построить и объекты для каждого из остальных пяти сменяющихся изображений. Осуществим это, используя такие же соглашения о наименовании, как и в первом случае.

```
<HEAD>
<SCRIPT Language="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
if (document.images)
{
ProductsOn=new Image(71, 33);
ProductsOn.src="images/products_on.gif";

ProductsOff=new Image(71, 33);
ProductsOff.src="images/products_off.gif";

ServicesOn=new Image(67, 33);
ServicesOn.src="images/services_on.gif";

ServicesOff=new Image(67, 33);
ServicesOff.src="images/services_off.gif";

TrainingOn=new Image(75, 33);
TrainingOn.src="images/training_on.gif";

TrainingOff=new Image(75, 33);
TrainingOff.src="images/training_off.gif";

CommonOn=new Image(157, 33);
CommonOn.src="images/common_on.gif";

CommonOff=new Image(157, 33);
CommonOff.src="images/common_off.gif";

GeneticOn=new Image(98, 33);
GeneticOn.src="images/genetic_on.gif";

GeneticOff=new Image(98, 33);
GeneticOff.src="images/genetic_off.gif";
```

```

AboutOn=new Image(106, 33);
AboutOn.src="images/about_on.gif";

AboutOff=new Image(106, 33);
AboutOff.src="images/about_off.gif";
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>

```

Построение объектов `IMAGE` подобным образом позволяет добиться еще одного полезного эффекта. Когда браузер создает такие объекты, он отправляет изображения в свой кэш, выполняя тем самым их предварительную загрузку в оперативную память. Это дает возможность выводить на экран новую версию изображения сразу после его смены. Важно учитывать предварительную загрузку при написании HTML-страниц: если вы не поместите в теге `` атрибуты `HEIGHT`, `WIDTH` и `ALT`, большинство браузеров не станет отображать страницу до тех пор, пока все информационное наполнение и все изображения не будут загружены в память. Вы можете столкнуться с серьезными проблемами увеличения продолжительности загрузки, если попытаетесь разместить в памяти сразу большое количество объектов `IMAGE` для сменяющихся изображений в фоновом режиме, поэтому обязательно включайте в свой код HTML все необходимые теги.

Итак, мы создали объекты `IMAGE` для всех сменяющихся изображений. Следующий этап заключается в образзовании функций, которые будут выполнять смену изображений.

Функции смены изображения

Когда пользователь проведет курсор над одним из пунктов меню, обработчики событий JavaScript вызовут функцию, которая должна фактически осуществлять смену изображений. Функции относятся к наиболее важным компонентам JavaScript. Это — набор языковых операторов, выполняющих конкретные задачи. В приводимом примере мы рассмотрим и определение, и вызов функций, но в данный момент займемся определением тех из них, что необходимы для смены изображения. Для этого надо задать четыре основных элемента: ключевое слово `function`, имя функции, набор параметров, разделенных запятыми, и операторы, которые должны быть выполнены в функции.

Для практического использования наших сменяющихся изображений требуются две функции: одна переключается на графику с

обозначением `On`, когда курсор проходит над изображением, а вторая — возвращается на графику с обозначением `Off` после того, как курсор уходит с позиции над изображением. Как правило, функции, предназначенные для всей страницы, следует определять в разделе `HEAD` кода HTML, поэтому расположим наши функции сразу после объектов `IMAGE`.

```

function on(pic)
{
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>

```

Первым этапом определения функции является вызов ключевого слова `function`. Затем следует имя функции: в данном случае назовем ее `on()`. За именем внутри пары круглых скобок нужно разместить список переменных, разделенных запятыми, — по одной для каждого параметра, который необходимо передать функции. Эти переменные позволяют сообщить в функцию значения, полученные от вызывающих ее обработчиков событий. Здесь нам достаточно передать единственный параметр. Для его хранения будем использовать переменную `pic`.

Потом следует набор команд, которые должны быть выполнены в теле функции. Эти операторы непременно заключены в фигурные скобки.

```

function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>

```

Прежде всего функция обязана выполнить оператор `if`, проверяющий наличие браузеров более ранних версий. Если браузер поддерживает объект `IMAGE`, строки, следующие за оператором `if`, будут выполнены. В данной строке мы указываем браузеру, чтобы он заменил источник изображения, над которым появился курсор, источником, представляющим собой вариант графического изображения с обозначением `On`. В строке происходит очень многое,

поэтому разобьем ее на части. Вначале рассмотрим то, что находится слева от оператора присваивания (=):

```
document.images[pic].src
```

Мы сослались на свойство источника изображения, расположенного в той позиции массива `images`, которая имеет значение переменной `pic`. Поэтому предположим, что переменная `pic` имеет значение `Products`. Мы ссылаемся на свойство источника объекта `IMAGE` с именем `Products` и присваиваем ему значение, полученное с правой стороны от оператора присваивания.

Если бы мы явно вызывали графическое изображение `product` следующим образом:

```
document.product.src
```

нам пришлось бы применять отдельные функции для каждого изображения, для которого мы хотим иметь два сменяемых варианта. Это не самый удачный способ кодирования. При разработке сценария вы всегда должны думать о том, как написать код наиболее эффективным способом. При таком использовании переменных вам удастся сэкономить много времени и сил. Однако в некоторых ситуациях вам придется вызывать конкретное изображение при каждом обращении к функции, и приведенная выше строка кода является примером того, как это можно сделать.

Теперь о расположенном с правой стороны от оператора присваивания операторе вызова метода `eval()`. Он применяется для соединения значения `pic` со строкой, позволяющей создать новое значение, присваиваемое в качестве источника изображения, над которым проходит курсор:

```
eval(pic + "On.src");
```

Если параметр `pic` имеет значение `Products`, оператор `eval()` вернет значение `ProductsOn.src`, и значение источника объекта `IMAGE` с именем `ProductsOn` будет присвоено источнику объекта `IMAGE` с именем `Products`. Когда это произойдет, изображение на странице сменится изображением с обозначением `On`, и последнее будет находиться там до тех пор, пока пользователь не уберет с него курсор.

Как только пользователь уберет курсор, нужно вызвать еще одну необходимую нам функцию. Назовем ее `off`.

```
function off(pic)
{
```

```
    if (document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Она должна быть очень похожа на функцию `on()`. Единственное различие состоит в том, что вторая половина вычисляемого справа от оператора присваивания, имеет значение `Off.src` вместо `On.src`. Поэтому при выполнении данной строки кода она назначит в качестве источника объекта `IMAGE` с именем `Products` источник объекта `IMAGE` с именем `ProductsOff`, а это равносильно возвращению изображения к обычному виду.

Теперь все функции разработаны, и подготовка сценария практически завершена. Нам осталось поместить в код HTML соответствующие обработчики событий JavaScript, чтобы написанный код мог работать.

Вставка обработчиков событий

Со временем вы узнаете, что большинство действий в коде JavaScript происходит под управлением событий. События обычно являются результатом каких-либо выполненных операций (например, перемещения курсора над ссылкой или изменения значения текстового поля). Чтобы воспользоваться этими событиями в коде JavaScript, следует определить обработчик, который станет на них реагировать. Имеется много predefined обработчиков событий, применяемых с этой целью. Для наших сменяющихся изображений мы будем использовать `onMouseOver` и `onMouseOut`. Вначале рассмотрим, как выглядит существующий код HTML без обработчиков событий.

```
<A HREF="products/index.html"><IMG
src="images/h_products_off.gif"
WIDTH="71" HEIGHT="33" BORDER="0" NAME="Products"
ALT="Products"></A>
```

Для решения тех задач, что стоят перед нами, нужно добавить обработчики событий в тег якоря. Сначала введем `onMouseOver`.

```
<A HREF="products.html" onMouseOver="on('products');
return true;">
```

При вызове обработчика событий он должен выполнить все следующие за ним операторы JavaScript. Мы заключили эти операторы в двойные кавычки и разделили их точками с запятой. Первым делом необходимо, чтобы обработчик вызвал функцию `on()`. Для этого требуется просто ввести ее имя, а после него указать в круглых скобках все разделенные запятыми значения, которые нужно передать функции:

```
on('Products');
```

Приведенный выше оператор вызывает функцию `on()` и передает ей значение `Products`. Опять-таки, мы передаем значение `Products`, поскольку это — имя изображения, на которое должна воздействовать данная функция.

Распространенным дополнением к сменяющимся изображениям является и вывод соответствующей фразы в строке состояния браузера. Для этого следует добавить к обработчику событий еще один оператор:

```
<A HREF="products.html" onMouseOver="on('products');  
window.status='Products'; return true;">
```

В строке состояния браузера можно отобразить любой указанный текст с помощью вызова свойства состояния объекта `WINDOW`, присвоив ему новое значение. Последним дополнением, которое надо внести в обработчик событий, является следующий оператор:

```
return true;
```

Он сообщает интерпретатору JavaScript о завершении всех необходимых действий в обработчике событий и о том, что теперь он может вернуться к выполнению других своих обязанностей.

Итак, первый обработчик событий добавлен. Теперь мы должны перейти к следующему — `onMouseOut`, который позаботится о выполнении всех нужных операций после того, как пользователь уберет курсор с изображения.

```
<A HREF="products.html" onMouseOver="on('products');  
window.status='Products'; return true;"  
onMouseOut="off('products'); window.status=''; return true;">
```

Для данного обработчика мы вызываем функцию `off()` и снова передаем ей имя графического изображения, на которое собираемся воздействовать. Чтобы удалить фразу, помещенную в строку состояния, мы переустанавливаем в качестве `window.status` пустое значение, а затем выходим из обработчика с помощью оператора `return`.

После введения обработчиков событий для изображения `Products` надо кое-что добавить для остальных изображений. Синтаксис остается аналогичным. Не забывайте только вместо передаваемого функциям значения указывать имя графического изображения, которое требуется сменить.

Описание сценария

Таким образом, мы располагаем всеми элементами для приведения в действие Web-узла с эффектными сменяющимися изображениями. Рассмотрим законченный сценарий и повторим то, что нам пришлось выполнить в процессе работы.

Вначале мы создали необходимые для сценария объекты `IMAGE`:

```
<SCRIPT Language="JavaScript">  
<!-- Броузеры ранних версий игнорируют следующий код  
if (document.images)  
{  
ProductsOn=new Image(71, 33);  
ProductsOn.src="images/products_on.gif";  
  
ProductsOff=new Image(71, 33);  
ProductsOff.src="images/products_off.gif";  
  
ServicesOn=new Image(67, 33);  
ServicesOn.src="images/services_on.gif";  
  
ServicesOff=new Image(67, 33);  
ServicesOff.src="images/services_off.gif";  
  
TrainingOn=new Image(75, 33);  
TrainingOn.src="images/training_on.gif";  
  
TrainingOff=new Image(75, 33);  
TrainingOff.src="images/training_off.gif";  
  
CommonOn=new Image(157, 33);  
CommonOn.src="images/common_on.gif";  
  
CommonOff=new Image(157, 33);  
CommonOff.src="images/common_off.gif";  
  
GeneticOn=new Image(98, 33);  
GeneticOn.src="images/genetic_on.gif";  
  
GeneticOff=new Image(98, 33);  
GeneticOff.src="images/genetic_off.gif";
```



```
AboutOn=new Image(106, 33);
AboutOn.src="images/about_on.gif";

AboutOff=new Image(106, 33);
AboutOff.src="images/about_off.gif";
}
```

Теперь перечислим действия, выполненные для построения объектов IMAGE.

1. Скрыт код от браузеров, не поддерживающих объекты IMAGE.
2. Добавлен атрибут NAME к шести изображениям, в которые нужно поместить сменяющиеся изображения.
3. Создан набор из двух новых объектов IMAGE для каждого из шести пунктов меню, на которые нужно воздействовать с помощью сценария смены изображения.
4. Свойству источника каждого объекта IMAGE присвоен адрес соответствующего файла изображения.

Затем были разработаны функции, выполняющие смену изображений:

```
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
}
function off(pic)
{
    if (document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Ниже перечислены этапы создания функций смены изображения.

1. Разработана функция on(), заменяющая изображение, на которое пользователь в данный момент установил курсор, вариантом, показывающим, что над ним находится курсор.

2. Разработана функция off(), устанавливающая в качестве графического то изображение, которое применяется по умолчанию, когда пользователь убирает с него курсор.
3. Внесены изменения в теги <A HREF> в существующем коде HTML:

```
<A HREF="products/index.html"
onMouseOver="on('Products'); window.status='Products';
return true;"
onMouseOut="off('Products'); window.status=' '; return
true;">
```

```
<A HREF="services/index.html"
onMouseOver="on('Services'); window.status='Services';
return true;"
onMouseOut="off('Services'); window.status=' '; return
true;">
```

```
<A HREF="training/index.html"
onMouseOver="on('Training'); window.status='Training';
return true;"
onMouseOut="off('Training'); window.status=' '; return
true;">
```

```
<A HREF="common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;"
onMouseOut="off('Common'); window.status=' '; return
true;">
```

```
<A HREF="genetic/index.html"
onMouseOver="on('Genetic'); window.status='Genetic';
return true;"
onMouseOut="off('Genetic'); window.status=' '; return
true;">
```

```
<A HREF="about/index.html"
onMouseOver="on('About'); window.status='About Us';
return true;"
onMouseOut="off('About'); window.status=' '; return
true;">
```

Теперь перечислим действия, связанные со вставкой обработчиков событий в код HTML.

1. К каждому из шести тегов изображения <A HREF> добавлены обработчики событий onMouseOver.

В обработчике событий вызывается функция `on()`, ей передается имя графического изображения, над которым появился курсор, вносятся изменения в строку состояния для отображения нового сообщения и добавляется команда `return` для выхода из обработчика.

- К каждому из шести тегов изображения `<A HREF>` добавлены обработчики событий `onMouseOut`.

В обработчике событий вызывается функция `off()`, ей передается имя графического изображения, которое должно появиться после того, как с этого изображения уйдет курсор, вносятся изменения в строку состояния для удаления из нее сообщения и добавляется команда `return` для выхода из обработчика.

В этом первом сценарии было использовано несколько новых средств JavaScript, в том числе:

- способ сокрытия кода, который позволяет определить браузеры, поддерживающие необходимые для сценария конкретные объекты;
- знакомство с массивами, в частности, с массивом `images`;
- способы создания новых объектов `IMAGE` и изменения их свойства источника;
- создание функций;
- общее описание обработчиков событий и способ применения двух из них: `onMouseOver` и `onMouseOut`;
- изменение сообщения, отображаемого в строке состояния окна браузера.

◆ Проект II: Расширение функциональных возможностей сменяющихся изображений

Вашему руководителю так понравились новые сменяющиеся изображения на начальной странице, что он захотел иметь такой эффект и на всех вторичных страницах узла Shelley (рис. 2.3). Это задание относительно несложное, поскольку для его выполнения можно в основном применить тот же сценарий, что был подготовлен для начальной страницы. Однако, чтобы сценарий для вторичных страниц работал правильно, в них все же необходимо внести некоторые изменения.

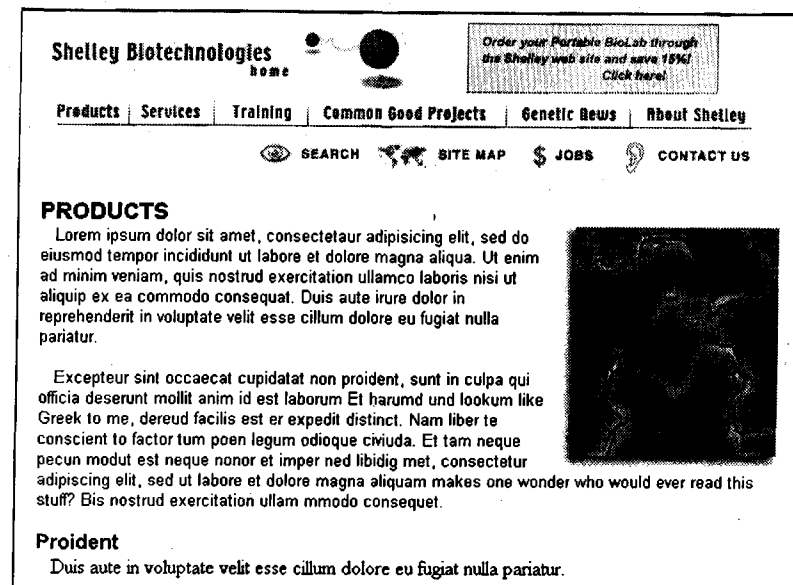


Рис. 2.3. Вторичная страница Shelley Biotech

Прежде всего нам не нужна смена изображения для раздела, в котором в настоящее время пребывает пользователь. Это значит, что в данном месте должна постоянно располагаться версия графического изображения с обозначением `On`. Таким образом, пользователь всегда сможет определить, в какой категории находится. Чтобы этого добиться, не нужно изменять сценарий. Достаточно просто изъять обработчики событий из тега `<A HREF>` и сменить изображение, вызываемое в теге ``, на вариант с обозначением `On`.

Следующее необходимое изменение немного сложнее и требует корректировки функций, написанных для начальной страницы. Прежде всего в той категории меню, где мы находимся, надо показывать версию графического изображения, над которой не проходит курсор, то есть — находящуюся в неподсвеченном состоянии, когда курсор перемещается над одним из изображений другой навигационной панели. Это позволяет исключить возможность одновременного появления двух подсвеченных изображений, способных запутать пользователя. Следует также следить за тем, чтобы после снятия курсора с изображения пункт меню для данного раздела опять получал подсветку.

Вставка обработчиков событий

Перед тем, как приступить к внесению изменений в функции, займемся корректировкой тегов `<A HREF>` и ``. Рассмотрим раздел кода HTML, содержащий навигационные изображения второй страницы Products.

```
<A HREF="index.html"><IMG src="../../images/products_off.gif"
WIDTH="71" HEIGHT="33" BORDER="0" ALT="Products"></A>
```

```
<A HREF="../../services/index.html"><IMG
src="../../images/services_off.gif"
WIDTH="67" HEIGHT="33" BORDER="0" ALT="Services"></A>
```

```
<A HREF="../../training/index.html"><IMG
src="../../images/training_off.gif"
WIDTH="75" HEIGHT="33" BORDER="0" ALT="Training"></A>
```

```
<A HREF="../../common/index.html"><IMG
src="../../images/common_off.gif"
WIDTH="157" HEIGHT="33" BORDER="0" ALT="Common Good"></A>
```

```
<A HREF="../../genetic/index.html"><IMG
src="../../images/news_off.gif"
WIDTH="98" HEIGHT="33" BORDER="0" ALT="Genetic News"></A>
```

```
<A HREF="../../about/index.html"><IMG
src="../../images/about_off.gif"
WIDTH="106" HEIGHT="33" BORDER="0" ALT="About Shelley"></A>
```

Вначале нужно сменить изображение, вызываемое в качестве графического файла Products, на вариант с обозначением On (products_on.gif):

```
<IMG src="../../images/products_on.gif"
WIDTH="71" HEIGHT="33" BORDER="0" ALT="Products">
```

Таким образом, пока пользователь пребывает в разделе Products, навигационное изображение Products остается в подсвеченном состоянии, словно над ним расположен курсор (это соответствует варианту объекта изображения с обозначением On и файлу с частью имени _on). Теперь необходимо ввести атрибуты NAME в тег `` и вставить обработчики событий в теги `<A HREF>` изображения. Когда пользователь перейдет на эту страницу, изображение Products будет всегда иметь вариант с обозначением On, поэтому нам не нужно, чтобы данная графика участвовала в смене изображений. В

связи с чем мы не должны помещать обработчики событий в тег `<A HREF>` для этой графики.

```
<A HREF="index.html">
<IMG src="../../images/products_on.gif" WIDTH="71" HEIGHT="33"
BORDER="0" NAME="Products" ALT="Products"></A>
```

```
<A HREF="../../services/index.html"
onMouseOver="on('Services'); window.status='Services'; return
true;"
onMouseOut="off('Services'); window.status=''; return true;">
<IMG src="../../images/services_off.gif" WIDTH="67" HEIGHT="33"
BORDER="0" NAME="Services" ALT="Services"></A>
```

```
<A HREF="../../training/index.html"
onMouseOver="on('Training'); window.status='Training'; return
true;"
onMouseOut="off('Training'); window.status=''; return true;">
<IMG src="../../images/training_off.gif" WIDTH="75" HEIGHT="33"
BORDER="0" NAME="Training" ALT="Training"></A>
```

```
<A HREF="../../common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;"
onMouseOut="off('Common'); window.status=''; return true;">
<IMG src="../../images/common_off.gif" WIDTH="157" HEIGHT="33"
BORDER="0" NAME="Common" ALT="Common Good"></A>
```

```
<A HREF="../../genetic/index.html"
onMouseOver="on('News'); window.status='Genetic News'; return true;"
onMouseOut="off('News'); window.status=''; return true;">
<IMG src="../../images/news_off.gif" WIDTH="98" HEIGHT="33"
BORDER="0" NAME="News" ALT="Genetic News"></A>
```

```
<A HREF="../../about/index.html"
onMouseOver="on('About'); window.status='About Us'; return
true;"
onMouseOut="off('About'); window.status=''; return true;">
<IMG src="../../images/about_off.gif" WIDTH="106" HEIGHT="33"
BORDER="0" NAME="About" ALT="About Shelley"></A>
</TD>
```

Обработчики событий наверняка покажутся вам очень знакомыми, поскольку именно они вставлялись в сценарий смены изображений на начальной странице. Поиск приемов повторного использования уже написанного кода — превосходный способ экономии времени и повышения производительности. Атрибут NAME также был

введен во все теги . Обратите внимание: хотя обработчики событий для изображения Products не вставлены, для него все равно предусмотрен атрибут NAME. Причина этого станет понятной далее. Следующий этап написания сценария предусматривает создание объектов IMAGE для всех сменяющихся изображений.

Создание объектов IMAGE

И здесь удача нам немного улыбнулась: объекты IMAGE, подготовленные для начальной страницы, при незначительных поправках станут работать вполне приемлемо на всех вторичных страницах. Единственное, что нам нужно подкорректировать, — путь, присвоенный свойству источника объекта. Поскольку вторичные страницы находятся в каталогах, расположенных одним уровнем ниже Home Page, мы должны сообщить объекту, чтобы он, прежде чем приступить к поиску каталога Images, вернулся назад на один уровень.

Сначала скопируем имеющийся объект IMAGE с начальной страницы на страницу Products, а затем внесем все необходимые изменения.

```
<SCRIPT Language="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Создание объектов IMAGE для сменяющихся изображений
if (document.images)
{
ProductsOn=new Image(71, 33);
ProductsOn.src="images/products_on.gif";

ProductsOff=new Image(71, 33);
ProductsOff.src="images/products_off.gif";

ServicesOn=new Image(67, 33);
ServicesOn.src="images/services_on.gif";

ServicesOff=new Image(67, 33);
ServicesOff.src="images/services_off.gif";

TrainingOn=new Image(75, 33);
TrainingOn.src="images/training_on.gif";

TrainingOff=new Image(75, 33);
TrainingOff.src="images/training_off.gif";

CommonOn=new Image(157, 33);
CommonOn.src="images/common_on.gif";
```

```
CommonOff=new Image(157, 33);
CommonOff.src="images/common_off.gif";

NewsOn=new Image(98, 33);
NewsOn.src="images/news_on.gif";

NewsOff=new Image(98, 33);
NewsOff.src="images/news_off.gif";

AboutOn=new Image(106, 33);
AboutOn.src="images/about_on.gif";

AboutOff=new Image(106, 33);
AboutOff.src="images/about_off.gif";
}
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
```

Теперь надо перейти ко вторичной странице и исправить путь для свойства источника объектов.

```
<SCRIPT Language="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Создание объектов IMAGE для сменяющихся изображений
if (document.images)
{
ProductsOn=new Image(71, 33);
ProductsOn.src="../images/products_on.gif";

ProductsOff=new Image(71, 33);
ProductsOff.src="../images/products_off.gif";

ServicesOn=new Image(67, 33);
ServicesOn.src="../images/services_on.gif";

ServicesOff=new Image(67, 33);
ServicesOff.src="../images/services_off.gif";

TrainingOn=new Image(75, 33);
TrainingOn.src="../images/training_on.gif";

TrainingOff=new Image(75, 33);
TrainingOff.src="../images/training_off.gif";

CommonOn=new Image(157, 33);
CommonOn.src="../images/common_on.gif";

CommonOff=new Image(157, 33);
```

```

CommonOff.src="../images/common_off.gif"

NewsOn=new Image(98, 33);
NewsOn.src="../images/news_on.gif";

NewsOff=new Image(98, 33);
NewsOff.src="../images/news_off.gif";

AboutOn=new Image(106, 33);
AboutOn.src=" ../images/about_on.gif";

AboutOff=new Image(106, 33);
AboutOff.src=" ../images/about_off.gif";
}
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>

```

Последний этап заключается в размещении функций смены изображения и внесении изменений, необходимых для того, чтобы эти функции могли работать на вторичных страницах.

Вставка функций смены изображения

Начнем просто с копирования функций, используемых на Home Page. Как уже указывалось, для применения на последующих страницах они требуют корректировки. Но поскольку основные их возможности остаются неизменными, нам не нужно приступать к разработке данных функций с нуля.

```

// Функции смены изображения
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
    }
}
function off(pic)
{
    if (document.images)
    {
        document.images[pic].src= eval(pic + "Off.src");
    }
}

```

Создав основные функции для страницы, рассмотрим проблему, которую обязательно нужно решить, чтобы заставить их работать правильно. На вторичных страницах мы изменили код HTML таким образом, что в качестве навигационного графического изображения для любого раздела, в котором находится пользователь, всегда применялась версия с частью имени On. Поэтому, когда курсор начнет перемещаться по одной из остальных категорий, будет подсвечена как графика для этой категории, так и графическое изображение для раздела, в котором в настоящее время пребывает пользователь. Чтобы не возникла путаница, необходимо сделать следующее: отключить подсветку графики для раздела, в котором в данный момент работает посетитель узла, на то время, пока курсор скользит по одной из других категорий.

С этой целью следует внести небольшие изменения и в функцию on(), и в функцию off(), а также создать новую функцию off1(), которая станет реализовывать часть необходимых нам рабочих средств. Начнем с корректировки функции on().

```

// Функции смены изображения
...
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval(pic + "On.src");
        document.images['Products'].src=eval("ProductsOff.src");
        ...
    }
}
...

```

Строка, введенная в данный сценарий, сообщает интерпретатору JavaScript, что он должен перевести в неподсвеченное состояние изображение для раздела Products после подсветки графики, на которой пользователь установил курсор. Обратите внимание: мы специально вызываем изображение Products по имени и явно ссылаемся на источник объекта, который хотим ему присвоить. Поэтому строку нужно исправить во всех разделах узла, чтобы сценарий убирал подсвечивание с соответствующего изображения.

Теперь изображение для раздела, в который мы вошли, становится неподсвеченным, как только пользователь устанавливает курсор на графику для другого раздела. Требуется способ перевести изображение для текущего раздела снова в подсвеченное состояние

в тот момент, когда курсор уходит с изображения для нерабочего раздела. Добиться этого легче всего размещением строки кода в функции `off()`. Последняя при своем вызове просто переводит графическое изображение для активного раздела в положение On (в подсвеченное состояние). К сожалению, описанный способ имеет один недостаток: если пользователь будет быстро переводить курсор с одного изображения на другое, то графика для категории, в которой пользователь находится сейчас, начнет мерцать, быстро сменяя подсвеченное состояние на неподсвеченное и наоборот. Это, конечно, не конец света, но все же недостаток довольно принципиальный и его устранение требует более сложного решения.

Нам требуется следующее. Перед тем, как переводить изображение для выбранной категории снова в подсвеченное состояние, необходимо знать не только то, убрал ли пользователь курсор с графики, но и то, убрал ли он его с одного из изображений для иных категорий. Сначала надо создать переменную, в которой мы можем хранить значение, используя его для проверки того, действительно ли курсор уведен с изображения.

```
// Функции смены изображения
var over_checker;
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval (pic + "On.src");
        document.images['Products'].src=eval ("ProductsOff.src");
        ...
    }
}
...
```

В приведенном коде мы объявляли переменную `over_checker` перед функциями смены изображения. Прежде всего эта переменная потребуется нам в функции `on()`, поскольку нужно знать, сколызит ли в настоящий момент курсор над изображением для категории. Применим переменную `over_checker` так, чтобы она принимала значение on при вызове функции `on()`.

```
// Функции смены изображения
var over_checker;
function on(pic)
{
    if (document.images)
```

```
{
    document.images[pic].src=eval (pic + "On.src");
    document.images['Products'].src=eval ("ProductsOff.src");
    over_checker='on';
}
}
...
```

Нам нужно знать не только то, что пользователь ведет курсор над изображением, но и то, что он убрал с него курсор. Для этого следует внести изменения в функцию `off()`.

```
function off(pic)
{
    if (document.images)
    {
        over_checker='off';
        document.images[pic].src=eval (pic + "Off.src");
        ...
    }
}
...
```

Так же, как и в функции `on()`, мы добавили строку, присваивающую значение переменной `over_checker`. В данном случае мы присваиваем ей значение `off`. Это позволит нам установить, что курсор убран с изображения.

Что же теперь? Мы знаем, находится ли курсор над графикой, но как воспользоваться этой информацией? Добавим третью функцию, которая станет проверять переменную `over_checker`. Если курсор не размещен над каким-либо изображением для другого раздела, она будет снова переводить в подсвеченное состояние графику для активной категории. Рассмотрим эту новую функцию:

```
function off1()
{
    if (over_checker=='off')
    {
        if (document.images)
        {
            document.images['Products'].src=
            eval ("ProductsOn.src");
        }
    }
}
```

Она выполняет несколько действий, но в ней нет каких-либо новых для нас понятий. Тем не менее, разберем ее подробнее. Для проверки того, имеет ли переменная `over_checker` значение `off`, применяется оператор `if`. При положительном ответе сценарий переходит к дальнейшим действиям и проверяет, поддерживает ли браузер пользователя сменяющиеся изображения. Если да, то сценарий переводит в подсвеченное состояние изображение для раздела, в котором находится пользователь. Как и в строке кода, добавленной к функции `on()`, мы явно вызываем изображение `Products` для его возвращения в подсвеченное состояние. Как только мы закончим подготовку кода для других вторичных категорий, нам потребуется изменить изображение, на которое воздействует данная функция.

Есть еще один последний фрагмент кода, отсутствующий в наших функциях. В настоящее время `off1()` нигде не вызывается. Разумнее всего поместить ее вызов в функцию `off()` с тем, чтобы `off1` выполнялась при отводе курсора с изображения. И если пользователь не навел курсор на другое изображение, то состояние подсветки для выбранной категории должно измениться. Однако при использовании всего лишь обычного вызова функции возникает следующая проблема: поскольку интерпретатор JavaScript обрабатывает код очень быстро, может случиться так, что пользователь не успеет перевести курсор на другое изображение достаточно быстро для того, чтобы предотвратить перевод этой функцией графического изображения для текущего раздела в подсвеченное состояние. Поэтому требуется установить задержку, так, чтобы `off1()` вызывалась после короткой паузы, разрешая пользователю постепенно перевести курсор на иное изображение.

Для осуществления этой задачи применим новый метод — `setTimeout()`. Он выполняет свои параметры `arguments` по истечении времени, указанного пользователем. Вот синтаксис данного метода:

```
setTimeout(arguments, msec)
```

Для наших целей достаточно передать методу только один параметр — вызов функции `off1()`. Чтобы предоставить пользователю время, необходимое для перевода курсора на новое изображение, нужно лишь задать короткую паузу (здесь вполне приемлемым значением является 700 миллисекунд). Рассмотрим новую строку кода после ее добавления к функции `off()`:

```
function off (pic)
{
```

```
if (document.images)
{
    over_checker='off';
    document.images[pic].src=eval(pic + "Off.src");
    setTimeout("off1()", 700);
}
}
```

Последний добавленный фрагмент кода сообщает интерпретатору JavaScript, чтобы он вызвал через 700 мс функцию `off1()`, которая затем выполнит всю остальную работу за нас. Разработка усовершенствованных функций смены изображения для раздела `Products` завершена.

Описание сценария

Хотя главная часть сценария этого проекта написана на основе функций смены изображения, созданных для начальной страницы, кое-какие изменения все же были сделаны. В том числе к функциям смены добавлены некоторые новые средства. Рассмотрим именно отличительные особенности проекта.

Прежде всего, мы вставили необходимые обработчики событий и внесли коррективы в теги ``.

```
<A HREF="index.html">
<IMG src="../../images/products_on.gif" WIDTH="71" HEIGHT="33"
BORDER="0" NAME="Products" ALT="Products"></A>

<A HREF="../../services/index.html"
onMouseOver="on('Services'); window.status='Services'; return true;"
onMouseOut="off('Services'); window.status=''; return true;">
<IMG src="../../images/services_off.gif" WIDTH="67" HEIGHT="33"
BORDER="0" NAME="Services" ALT="Services"></A>

<A HREF="../../training/index.html"
onMouseOver="on('Training'); window.status='Training'; return true;"
onMouseOut="off('Training'); window.status=''; return true;">
<IMG src="../../images/training_off.gif" WIDTH="75" HEIGHT="33"
BORDER="0" NAME="Training" ALT="Training"></A>

<A HREF="../../common/index.html"
onMouseOver="on('Common'); window.status='Common Good
Projects'; return true;"
onMouseOut="off('Common'); window.status=''; return true;">
<IMG src="../../images/common_off.gif" WIDTH="157" HEIGHT="33"
BORDER="0" NAME="Common" ALT="Common Good"></A>
```



```

<A HREF="../../../genetic/index.html"
onMouseOver="on('News'); window.status='Genetic News'; return
true;"
onMouseOut="off('News'); window.status=''; return true;">
<IMG src="../../../images/news_off.gif" WIDTH="98" HEIGHT="33"
BORDER="0" NAME="News" ALT="Genetic News"></A>

<A HREF="../../../about/index.html"
onMouseOver="on('About'); window.status='About Us'; return
true;"
onMouseOut="off('About'); window.status=''; return true;">
<IMG src="../../../images/about_off.gif" WIDTH="106" HEIGHT="33"
BORDER="0" NAME="About" ALT="About Shelley"></A>

```

Перечислим этапы разработки данной части сценария.

1. Мы ввели обработчики событий onMouseOver в теги <A HREF> всех изображений, кроме Products.
В обработчике мы вызываем функцию on(), передаем ей имя графического изображения, над которым появился курсор, изменяем строку состояния для отображения нового сообщения и добавляем команду return для выхода из обработчика.
2. Мы добавили обработчики событий onMouseOut в теги <A HREF> всех изображений, кроме Products.
В этом обработчике мы вызываем функцию off(), передаем ей имя графического изображения, с которого был убран курсор, переустанавливаем строку состояния, чтобы в ней не было никакого сообщения, и добавляем команду return для выхода из обработчика.
3. Мы вставили атрибут NAME во все шесть тегов .
Хотя обработчики событий не были помещены в тег <A HREF> изображения Products, мы должны присвоить ему атрибут NAME, поскольку придется менять его состояние, когда курсор начнет перемещаться над одним из пяти изображений для разделов.

После вставки обработчиков событий и атрибутов NAME мы создали объекты IMAGE, применяемые в сценарии смены изображения.

```

<SCRIPT Language="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Создание объектов IMAGE для сменяющихся изображений
if (document.images)
{
ProductsOn=new Image(71, 33);
ProductsOn.src="../../../images/products_on.gif";

```

```

ProductsOff=new Image(71, 33);
ProductsOff.src="../../../images/products_off.gif";

ServicesOn=new Image(67, 33);
ServicesOn.src="../../../images/services_on.gif";

ServicesOff=new Image(67, 33);
ServicesOff.src="../../../images/services_off.gif";

TrainingOn=new Image(75, 33);
TrainingOn.src="../../../images/training_on.gif";

TrainingOff=new Image(75, 33);
TrainingOff.src="../../../images/training_off.gif";

CommonOn=new Image(157, 33);
CommonOn.src="../../../images/common_on.gif";

CommonOff=new Image(157, 33);
CommonOff.src="../../../images/common_off.gif";

NewsOn=new Image(98, 33);
NewsOn.src="../../../images/news_on.gif";

NewsOff=new Image(98, 33);
NewsOff.src="../../../images/news_off.gif";

AboutOn=new Image(106, 33);
AboutOn.src="../../../images/about_on.gif";

AboutOff=new Image(106, 33);
AboutOff.src="../../../images/about_off.gif";
}

```

Теперь перечислим этапы создания объектов IMAGE.

1. Мы скопировали объекты IMAGE, созданные для смены изображений на начальной странице.
2. Мы указали в качестве свойства источника объектов правильный путь.
3. Мы включили в сценарий функции, управляющие его работой.

```

// Функции смены изображения
var over_checker;
function on(pic)
{
    if (document.images)
    {
        document.images[pic].src=eval (pic + "On.src");

```

```

document.images['Products'].src=eval("ProductsOff.src");
    over_checker='on';
}
}
function off (pic)
{
    if (document.images)
    {
        over_checker='off';
        document.images[pic].src=eval (pic + "Off.src");
        setTimeout ("off1('Products')", 700);
    }
}
function off1()
{
    if (over_checker=='off')
    {
        if (document.images)
        {
            document.images['Products'].src=
            eval("ProductsOn.src");
        }
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>

```

А это этапы создания функций.

1. Мы скопировали функции `on()` и `off()` из сценария смены изображения на начальной странице и применили их в качестве основы для создания скорректированных функций `on()` и `off()`.
2. Мы изменили функцию `on()` следующим образом:
 - создали переменную `over_checker` до начала определения функций смены изображения;
 - внутри функции `on()` присвоили значение `on` переменной `over_checker`;
 - добавили строку для перевода изображения `Products` в положение `off`.
3. Мы изменили функцию `off()` следующим образом:
 - добавили строку, которая присваивает значение `off` переменной `over_checker`;
 - ввели метод `setTimeout`, вызывающий функцию `off1()` после задержки в 700 мс.

4. Мы создали функцию `off1()`.

Она выполняет две проверки с применением операторов `if`: вначале определяет, установлено ли значение переменной `over_checker` в `off`, а затем проверяет, поддерживает ли браузер пользователя сменяющиеся изображения.

Если обе проверки возвращают значение истина, то данная функция устанавливает изображение `Products` в положение `on`.

В этом сценарии были впервые использованы такие средства JavaScript:

- метод `setTimeout`;
- принцип применения существующих сценариев для решения новых задач.

◆ Некоторые особенности сменяющихся изображений

Сменяющиеся изображения могут оказаться очень полезным и привлекательным дополнением к Web-узлу, однако у них есть некоторые особенности, которые обязательно необходимо учитывать. Начнем с того, что не все посетители узла смогут видеть ваши сменяющиеся изображения: например те, кто работает с версией браузера Netscape или Internet Explorer более ранней, чем 3.0. Хотя это соображение, безусловно, не настолько существенно, чтобы заставить вас отказаться от использования данного эффекта, вам надо иметь его в виду. Вряд ли стоит применять навигацию, доступную исключительно благодаря сменяющимся изображениям. Будет правильным, если вы станете включать, по меньшей мере, резервную текстовую навигацию для пользователей, работающих с более ранними версиями браузеров.

В обозревателе Netscape 3.0 проявляется себя еще одна особенность. Так, при использовании изображений внутри вложенных таблиц интерпретатор JavaScript создает по два входа массива для единственного изображения в том случае, если у вас есть таблица, вложенная в другую таблицу, а изображение размещено именно во вложенной таблице. Когда графика находится в таблице, вложенной в две других, то в массиве `images` создаются три входа. В результате при ссылке на изображения путем указания не имени, а индекса в массиве `images`, могут возникнуть проблемы.

И, наконец, последняя особенность, о которой всегда следует помнить, связана с использованием в сменяющихся изображениях анимационных файлов gif. Если вы будете применять одну и ту же анимацию для нескольких сменяющихся изображений и предусмотрите для них единственный объект IMAGE, то обнаружите, что анимация работает только в первой графике. Во всех последующих она будет сразу переходить к завершающему кадру анимации. Чтобы справиться с ситуацией, надо предусмотреть отдельные объекты IMAGE для каждого случая показа анимационных файлов gif в сменяющихся изображениях.

◆ Резюме

В настоящее время сменяющиеся изображения являются одной из наиболее распространенных областей применения JavaScript на Web-узлах. Они используются тем или иным образом почти на каждом узле, поэтому знать о том, как они работают и каковы их ограничения, очень важно для любого программиста. Как только вы изучите основы этих средств, попробуйте применить свои знания, проведя эксперименты с разными способами смены изображения. Вы будете приятно удивлены тем, насколько интересные эффекты сможете создать без особого труда.

◆ Дополнительное задание

1. Постройте начальную страницу, на которой сменяются несколько график при перемещении курсора над изображениями одной из главных категорий.
2. Создайте страницу, на которой при перемещении курсора над изображением категории имеющееся изображение сменяется случайно выбранным.

3 Применение JavaScript для навигации

В этой главе

- Проект I: JavaScript и разворачивающиеся меню
- Проект II: Применение нескольких разворачивающихся меню для навигации
- Проект III: Применение JavaScript на странице регистрации
- Резюме
- Дополнительное задание

Вы уже провели большую работу, используя при оформлении начальной страницы компании Shelley Biotechnologie новые функциональные средства. Теперь, когда ваш руководитель увидел, насколько впечатляющими могут быть результаты, полученные с помощью языка JavaScript, ему пришла в голову еще одна идея. Он решил, что нужно разместить на Home Page разворачивающиеся меню и предоставить пользователю возможность быстро, с помощью единственного щелчка мыши, переходить на страницы, находящиеся на узле на несколько уровней ниже. И, естественно, эта задача также должна быть решена как можно быстрее. Поэтому давайте сразу же приступим к работе.

Посоветовавшись с одним из дизайнеров, вы выбрали наиболее подходящее место для размещения нового эффекта (рис. 3.1). Вам осталось лишь позаботиться о ссылках для разворачивающегося меню. Отправив письмо по электронной почте своему руководителю, вы быстро получили необходимую информацию, и теперь все вновь зависит только от вас.



Рис. 3.1. На начальной странице размещено разворачивающееся меню

◆ Проект I: JavaScript и разворачивающиеся меню

Прежде всего необходимо вставить код HTML начальной страницы, позволяющий создать разворачивающееся меню и кнопку Go. Вот код, который мы будем использовать:

```
<SELECT NAME="PullDown">
  <OPTION VALUE=" ">Get There Quick
  <OPTION VALUE=" ">What's New
  <OPTION VALUE=" ">Feature Product
  <OPTION VALUE=" ">Press Releases
  <OPTION VALUE=" ">Company Store
</SELECT>
<INPUT TYPE="button" NAME="Go" VALUE="Go">
```

Так как мы вставляем меню в уже существующий проект, надо разместить теги `<FORM>` в начале и конце кода HTML тела страницы, а не вокруг данного элемента формы (чтобы предотвратить появление ненужных символов перевода строки или пробела). Обратите внимание и на то, что свойства `VALUE` тегов `OPTION` остались незаполненными. Мы заполним их позднее, при вставке обработчика событий, который нужен для выполнения сценария.

Создание функции навигации

Поскольку навигационное разворачивающееся меню должно быть размещено на странице, уже имеющей сменяющиеся изображения, мы можем вставить функцию навигации в существующие теги `<SCRIPT>`. Поместим ее сразу после функций смены изображения следующим образом:

```
function off(pic)
{
  if(document.images)
  {
    document.images[pic].src= eval(pic + "Off.src");
  }
}
// Функция навигации разворачивающегося меню
function PageChanger(page)
{
  ...
}
```

Сама функция довольно проста и состоит из двух частей. Заметьте, мы передаем в функцию значение, которое присваивается переменной `page`. Оно соответствует адресу страницы, выбранной пользователем из разворачивающегося меню. В первой строке данное значение станет применяться для изменения свойства `location` объекта `DOCUMENT`, в результате чего в окно браузера будет загружена новая Web-страница.

```
// Функция навигации разворачивающегося меню
function PageChanger(page)
{
  document.location=page;
  ...
}
```

Затем мы добавим строку, осуществляющую вспомогательную работу. После выполнения функции она переустанавливает разворачивающееся меню для совершения первого выбора. В данном случае это не очень важно, поскольку мы сменяем страницу. Однако если меню устанавливает в качестве нового адреса другой кадр, а не все окно, такой сервис может оказаться удобным.

```
// Функция навигации разворачивающегося меню
function PageChanger(page)
{
  document.location=page;
  document.NavForm.PullDown.options[0].selected=true;
}
```

Вставка обработчика событий

После создания функции следует вставить в код HTML обработчик событий для ее вызова. Перед этим необходимо ввести в опции разворачивающегося меню свойства `VALUE`. Как сообщалось выше, значением, передаваемым в функцию, является URL страницы, на которую направляют пользователя. Это значение будет получено из атрибута `VALUE` выбранной опции меню. Поэтому в качестве значения каждой опции мы должны указать адрес страницы, соответствующей данному выбору.

```
<SELECT NAME="PullDown">
  <OPTION VALUE=" ">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
```

```
<OPTION VALUE="store/index.html">Company Store
</SELECT>
```

Для первой опции меню, которая является простым заголовком, введенным как дополнительное пояснение, нам не нужно указывать значение, поскольку эта опция не должна куда-либо направлять пользователя. Выбрав иную желаемую опцию, пользователь должен сразу щелкнуть на кнопке Go, чтобы перейти на новую страницу. Для этого требуется вставить обработчик событий onClick в код HTML для кнопки Go.

```
<SELECT NAME="PullDown">
  <OPTION VALUE="">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
  <OPTION VALUE="store/index.html">Company Store
</SELECT>
<INPUT TYPE="button" NAME="GO" VALUE="Go"
onClick="PageChanger(document.NavForm.PullDown
.options[NavForm.PullDown.selectedIndex].value)">
```

Обработчик событий выполняет два действия: вызывает функцию PageChanger() и передает ей значение. Передаваемым является значение опции, выбранной пользователем из разворачивающегося меню. В функциях, которые мы создавали до сих пор, передаваемым параметром служила простая строка. В этом обработчике событий мы пробуем кое-что новое: ссылаемся на объект разворачивающегося меню и обращаемся к значению, выбранному в настоящий момент. Затем это значение передается в функцию и применяется для перенаправления пользователя на страницу, которую он хочет посетить.

Теперь наш сценарий работоспособен и готов к использованию. Но поскольку к странице добавлена кнопка Go, категории, находящиеся справа от разворачивающегося меню, стали занимать две строки (рис. 3.2). Однако это недопустимо, и значит нужно найти другое решение. К счастью существует такой способ приведения нашего сценария в действие, который не только устранил возникшую проблему, но и даст возможность ускорить навигацию.

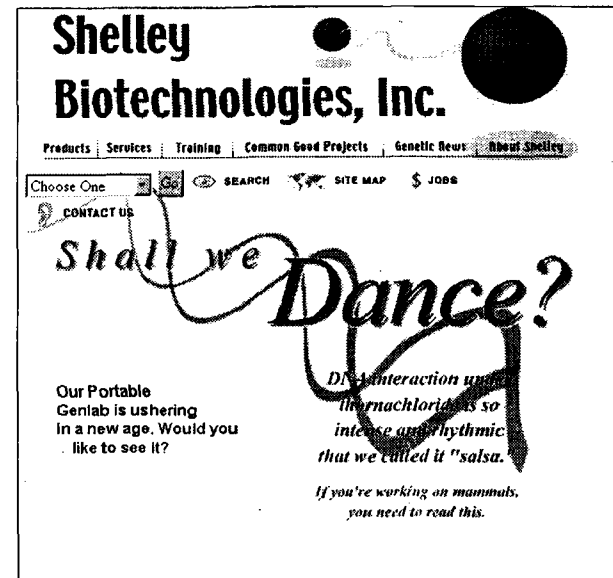


Рис. 3.2. Страница с разворачивающимся меню и кнопкой Go

Использование обработчика событий onChange для непосредственного перехода

Чтобы вернуть компоновку страницы в первоначальное состояние, надо избавиться от кнопки Go. Но в данный момент в нее включен обработчик событий, управляющий сценарием. Поэтому нам нужен обработчик, который можно будет поместить в само разворачивающееся меню, дабы он выполнял сценарий за нас. Именно таким является onChange. Он контролирует состояние содержащего его объекта и активизируется, обнаружив изменение состояния.

После вставки такого обработчика в разворачивающееся меню мы можем применить его для активизации функции смены страницы при выборе пользователем вместо опции меню, установленной по умолчанию во время загрузки страницы, другой опции. Посмотрим, как изменится код HTML.

```
<SELECT NAME="PullDown"
  onChange="PageChanger(this.options[this.selectedIndex
.value])">
```

```

<OPTION VALUE="">Get There Quick
<OPTION VALUE="whatsnew.html">What's New
<OPTION VALUE="products/featured.html">Feature Product
<OPTION VALUE="news/press.html">Press Releases
<OPTION VALUE="store/index.html">Company Store
</SELECT>

```

Обратите внимание на следующее. Во-первых, мы удалили кнопку Go, так как она больше не нужна. Во-вторых, в разворачивающемся меню был введен обработчик событий onChange. Значение, передаваемое в функцию, остается тем же самым — это значение опции, выбранное пользователем. Однако мы вызываем его иначе, чем в предыдущем варианте. Поскольку обработчик событий находится в элементе формы, из которого мы хотим получить значение, можно применить следующий способ:

```
this.options[this.selectedIndex].value
```

Данная строка кода в сценарии указывает интерпретатору JavaScript, чтобы он получил информацию из элемента формы, в котором содержится обработчик событий.

Оба варианта одинаково хороши, но лучше всегда иметь в своем распоряжении несколько способов выполнения задачи.

После вставки нового обработчика событий наш сценарий завершен и может функционировать. Теперь, с использованием onChange, страница сменится, как только будет выбран другой пункт меню, без нажатия на кнопку.

Описание сценария

Перечислим, что было сделано для написания этого сценария.

Прежде всего мы создали функцию, которая принимает значение из элемента формы и использует его для перенаправления пользователя на другую HTML-страницу. Мы также рассмотрели два разных обработчика событий, применимых для приведения сценариев в действие.

Сначала рассмотрим функцию.

```

// функция навигации разворачивающегося меню
function PageChanger(page)
{
    document.location=page;
    document.NavForm.PullDown.options[0].selected=true;
}
</SCRIPT>

```

1. Мы создали функцию PageChanger().

В ней мы первым делом установили свойство location объекта DOCUMENT в значение, содержащееся в переменной page. Это значение представляет собой URL, передаваемый в функцию из разворачивающегося меню.

Затем мы переустановили опцию, которая появляется в разворачивающемся меню, на первую опцию.

Теперь обратимся к коду HTML, необходимому для использования с обработчиком onClick.

```

<SELECT NAME="PullDown">
  <OPTION VALUE="">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
  <OPTION VALUE="store/index.html">Company Store
</SELECT>
<INPUT TYPE="button" NAME="Go" VALUE="Go"
onClick="PageChanger(document.NavForm.PullDown.
options[NavForm.PullDown.selectedIndex].value)">

```

2. Данная функция будет вызываться, когда пользователь щелкнет на кнопке Go. Мы вставили обработчик событий onClick в элемент формы button (кнопка).

В обработчике мы вызываем функцию PageChanger() и передаем ей значение, содержащееся в опции, выбранной из разворачивающегося меню.

Придя к выводу о необходимости убрать кнопку Go со страницы (по проектным соображениям), мы решили применить обработчик событий onChange.

```

<SELECT NAME="PullDown"
onChange="PageChanger(this.options[this.selectedIndex].
value)">
  <OPTION VALUE="">Get There Quick
  <OPTION VALUE="whatsnew.html">What's New
  <OPTION VALUE="products/featured.html">Feature Product
  <OPTION VALUE="news/press.html">Press Releases
  <OPTION VALUE="store/index.html">Company Store
</SELECT>

```

3. Мы вставили обработчик событий onChange в разворачивающееся меню.

В этом обработчике мы снова вызываем функцию `Page-Changer()` и передаем ей URL выбранной опции меню.

Вспомним новые понятия, с которыми мы ознакомились при выполнении данного проекта:

- доступ к свойству `location` объекта `DOCUMENT` и изменение этого свойства;
- доступ к значениям опций разворачивающегося меню и изменение выбранной опции;
- обработчики событий `onClick` и `onChange`.

По мере того, как Web-узлы начинают играть все более важную роль в успешной деятельности компаний, их информационное наполнение расширяется и углубляется. Разработка быстрых и эффективных способов навигации по этим просторам становится чрезвычайно актуальной. Применение обработчиков событий `onClick` и `onChange` вместе с разворачивающимися меню — один из удобных способов предоставления пользователям необходимой им информации.

◆ Проект II: Применение нескольких разворачивающихся меню для навигации

После успешного использования сценария навигации с разворачивающимся меню ваш руководитель наметил еще одну область Web-узла компании, которая могла бы выиграть за счет применения данного эффекта — раздел *Company Store* (магазин компании). До сих пор перемещение по различным группам товаров и их отдельным видам было сложным и не всегда очевидным для пользователя. Но компания *Shelley Biotech* готовит к выпуску новую линию своей продукции, и усовершенствованные средства навигации способны придать дополнительный блеск разделу магазина (рис. 3.3). В общем, было бы прекрасно, если бы вы смогли использовать разворачивающиеся меню и язык JavaScript для повышения эффективности работы данной области узла.

Ваш начальник предлагает разместить два разворачивающихся меню. Первое должно содержать различные группы товаров, а второе — динамически заполняться наименованиями продуктов выбранной группы. Затем пользователь должен быть направлен на страницу, посвященную товару, выбранному из второго меню.

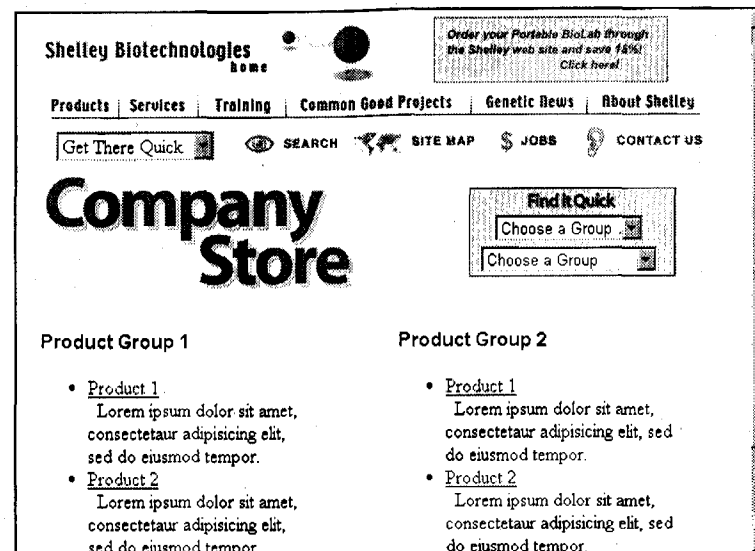


Рис. 3.3. Раздел *Company Store* Web-узла компании *Shelley*

На первый взгляд задача может показаться неосуществимой. Однако в предыдущих главах мы уже узнали многое из того, что необходимо для ее решения. Проект следует разбить на четыре раздела: создание массивов, которые будут хранить информацию, необходимую для каждой из групп товаров; построение функции, которая станет динамически заполнять второе меню; создание функции, обеспечивающей навигацию; вставка соответствующих обработчиков событий в код HTML.

Создание массивов

Поскольку к выпуску готовится очень много новых товаров, отдел маркетинга не может решить окончательно, какие группы и конкретные виды надо обязательно включить в навигацию. Ясно только, что должно быть, по крайней мере, четыре группы, в каждой из которых — содержаться не меньше пяти разновидностей товаров. С учетом данной информации можно создать систему навигационных меню с применением меток-заполнителей, что позднее будут

заменены сведениями о реальных товарах. Приступим к работе, исходя из указанных предпосылок.

В первых двух главах мы уже вкратце ознакомились с понятием массивов. Мы узнали, как обращаться к массивам, создаваемым интерпретатором JavaScript, в ходе загрузки страницы и как добавлять объекты к некоторым из них. Однако теперь нам предстоит образовать определенные массивы самим и заполнить их обновленной информацией о товарах.

Для создания массива применяется следующий синтаксис:

```
arrayObjectName = new Array([arrayLength])
```

В приведенном операторе переменная `arrayObjectName` условно обозначает имя, которое может быть присвоено массиву, а `arrayLength` — количество входов массива.

Для динамической вставки информации о товаре во второе разворачивающееся меню нужно иметь по два значения для каждого продукта. Первым является наименование товара, вторым — адрес HTML-страницы, которая ему соответствует. Для хранения этих данных создадим по два массива для каждой группы товаров: в одном из них должны храниться наименования товаров, в другом — адреса страниц. Поскольку мы не знаем, как будут называться группы, станем использовать для их обозначения переменные `group1`, `group2`, `group3` и `group4`. Как и в первом проекте, добавим этот новый код в существующий тег `<SCRIPT>` после кода, который уже применяется на странице.

```
// Массивы для товаров
// Определить массивы
group1_names = new Array(5);
group1_locations = new Array(5);

group2_names = new Array(5);
group2_locations = new Array(5);

group3_names = new Array(5);
group3_locations = new Array(5);

group4_names = new Array(5);
group4_locations = new Array(5);
...
```

Рассмотрим код, определяющий первый набор массивов:

```
group1_names = new Array(5);
group1_locations = new Array(5);
```

Первая строка описывает новый массив `group1_names` и указывает его длину, равную 5, что соответствует предполагаемому количеству наименований товаров каждой группы. Данный массив будет хранить имена продуктов группы `group1`. Вторая строка определяет массив `group1_locations`, длина которого также равна 5. Этот массив станет хранить адреса страниц для товаров группы `group1`. Аналогичным образом создадим массивы для трех других групп.

После определения массивов нужно заполнить их информацией о товарах. Сначала вкратце рассмотрим синтаксис добавления входа в массив:

```
arrayName[arrayPosition] = "arrayValue"
```

Здесь `arrayName` — имя массива, в который нужно добавить вход, `arrayPosition` — позиция в массиве, где будет храниться новое значение, а `arrayValue` — фактическое значение, которое должно храниться в массиве.

Заполним массивы группы `group1` информацией о ее товарах:

```
// Заполнить массивы группы Group1
group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';

group1_names[1] = 'Group1_product two';
group1_locations[1] = 'group1/prod_2.html';

group1_names[2] = 'Group1_product three';
group1_locations[2] = 'group1/prod_3.html';

group1_names[3] = 'Group1_product four';
group1_locations[3] = 'group1/prod_4.html';

group1_names[4] = 'Group1_product five';
group1_locations[4] = 'group1/prod_5.html';

var group1_length = group1_names.length;
```

В приведенном коде одинаковым позициям двух разных массивов были присвоены значения наименования и адреса страницы для одного и того же товара. Иными словами, выполнив первые две строки:

```
group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';
```

мы присвоили наименование товара номер один первой позиции массива `group1_names` и значение адреса страницы для этого товара, `group1/prod_1.html`, первой позиции массива `group1_locations`. В оставшейся части данного кода информация об остальных товарах размещается по двум массивам в позициях, соответствующих их номерам. Как и при наименовании групп, мы используем метки-заполнители для обозначения товаров и адресов их страниц.

Вам часто придется работать над проектом, не имея окончательного информационного наполнения для страниц. Используя вместо него метки-заполнители, вы можете довести до конца разработку кода, чтобы после получения итогового варианта информационного наполнения осталось только постранично его разместить. Весь секрет заключается в использовании понятных и легко распознаваемых значений меток-заполнителей, хотя бы на тот случай, если не вы будете вставлять окончательное информационное наполнение.

Последняя строка кода заполнения массивов `group1` выглядит следующим образом:

```
var group1_length = group1_names.length;
```

Она присваивает длину массивов `group1` переменной `group1_length`. Мы выполняем это, обратившись к свойству `length` объекта `ARRAY`. Данная переменная будет применяться в последующем в функции, используемой для заполнения второго меню.

Приведем оставшуюся часть кода, служащую для заполнения массивов других групп товаров. Этот код имеет такую же структуру, как и код для первой группы.

```
// Заполнить массивы группы Group2
group2_names[0] = 'Group2_product one';
group2_locations[0] = 'group2/prod_1.html';

group2_names[1] = 'Group2_product two';
group2_locations[1] = 'group2/prod_2.html';

group2_names[2] = 'Group2_product three';
group2_locations[2] = 'group2/prod_3.html';

group2_names[3] = 'Group2_product four';
group2_locations[3] = 'group2/prod_4.html';

group2_names[4] = 'Group2_product five';
group2_locations[4] = 'group2/prod_5.html';
```

```
var group2_length = group2_names.length;

// Заполнить массивы группы Group3
group3_names[0] = 'Group3_product one';
group3_locations[0] = 'group3/prod_1.html';

group3_names[1] = 'Group3_product two';
group3_locations[1] = 'group3/prod_2.html';

group3_names[2] = 'Group3_product three';
group3_locations[2] = 'group3/prod_3.html';

group3_names[3] = 'Group3_product four';
group3_locations[3] = 'group3/prod_4.html';

group3_names[4] = 'Group3_product five';
group3_locations[4] = 'group3/prod_5.html';

var group3_length = group3_names.length;

// Заполнить массивы группы Group4
group4_names[0] = 'Group4_product one';
group4_locations[0] = 'group4/prod_1.html';

group4_names[1] = 'Group4_product two';
group4_locations[1] = 'group4/prod_2.html';

group4_names[2] = 'Group4_product three';
group4_locations[2] = 'group4/prod_3.html';

group4_names[3] = 'Group4_product four';
group4_locations[3] = 'group4/prod_4.html';

group4_names[4] = 'Group4_product five';
group4_locations[4] = 'group4/prod_5.html';

var group4_length = group4_names.length;
```

Теперь мы имеем по два массива для каждой группы товаров, заполненных данными о всех товарах в этих группах. У нас есть также четыре переменные, содержащие длину каждого из массивов. Выполнив эту работу, перейдем к следующему этапу — созданию функции, которая будет заполнять второе разворачивающееся меню.

Создание функции многоступенчатого меню

Перед тем, как приступить к написанию такой функции, необходимо понять ее смысл. Когда пользователь выбирает группу товаров из первого меню, сценарий должен убрать все значения, содержащиеся в настоящее время во втором меню, прежде чем вставить в него данные о новых товарах. Это важно, поскольку новая группа продуктов, выбранная пользователем, может иметь меньше товаров, чем предыдущая. Если такое произойдет, новая группа может не перекрыть все товары предыдущей. Чтобы иметь полную уверенность в том, что мы убрали все возможные товары из меню, нужно определить длину самого большого набора массивов. Это можно выполнить с помощью следующего кода:

```
// Найти длину самого большого набора массивов
var maxLength = group1_length;
if (group2_length > maxLength)
{
    maxLength = group2_length;
}
if (group3_length > maxLength)
{
    maxLength = group3_length;
}
if (group4_length > maxLength)
{
    maxLength = group4_length;
}
```

Здесь вначале происходит присваивание длины первого набора массивов переменной `maxLength`. Затем с помощью оператора `if` выполняется проверка того, превышает ли длина каждой из оставшихся групп массивов значение `maxLength`. При положительном ответе этой переменной присваивается новое значение. В конечном итоге переменная `maxLength` получает значение длины самого длинного массива группы.

Теперь можно перейти к самой функции. Первым делом следует воспользоваться переменной `maxLength` для очистки содержимого второго меню.

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
```

```
while (currentPosition < maxLength)
{
    ...
    ++currentPosition;
}
...
}
```

В первой строке приведенного кода создается переменная `currentPosition` и ей присваивается значение 0. Затем применяется структура, которая называется циклом `while`. Он еще не использовался, поэтому рассмотрим его синтаксис.

Мы уже работали в сценариях с операторами `if`: они выполняют фрагмент кода, если определенное условие `condition` является истинным. Цикл `while` действует аналогичным образом, однако он продолжает выполнять содержащийся в нем фрагмент кода `statements` до тех пор, пока определенное условие `condition` остается истинным. Вот синтаксис цикла `while`:

```
while (condition)
{
    statements
}
```

Данная структура очень удобна, когда нужно получить несколько значений или изменить значение многих свойств одинаковым образом. Тем не менее, циклы могут доставлять и неприятности. Например, если вы создадите цикл, условие которого будет всегда оставаться истинным, вы получите так называемый “бесконечный” цикл. Иными словами, ваш сценарий навсегда остановится в одной точке, не добравшись до своего завершения, а это обычно приводит к прекращению его работы и аварийному выходу из браузера.

Но как сделать условие цикла ложным, если вы не можете из него выйти, пока он выполняется? Хороший вопрос. В такой ситуации наиболее часто используется изменение условия внутри самого цикла.

Цикл `while` в приведенном выше коде продолжает выполняться до тех пор, пока переменная `currentPosition` не превышает значения `maxLength`. При каждом проходе по циклу очищается опция меню, позиция которой соответствует значению `currentPosition`, а затем значение переменной увеличивается на 1. Таким образом, `currentPosition` в конечном итоге станет равной значению `maxLength`, и мы выйдем из цикла.

Здесь вы также впервые встретились с оператором инкремента ++, прибавляющим 1 к значению его операнда (в данном случае — currentPosition). Поэтому строка

```
++currentPosition;
```

отвечает за увеличение значения данной переменной.

Цикл будет выполняться до тех пор, пока значение currentPosition не станет равным длине самого протяженного массива. Таким образом, мы можем быть уверены в том, что из меню удалены все возможные опции. Мы выполним фактическую очистку меню, поместив следующий код в цикл while:

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker.Products
            .options[currentPosition].text = ' ';
        document.ProductPicker.Products
            .options[test].value = ' ';
        ++currentPosition;
        ...
    }
}
```

Первая строка введенного кода устанавливает вместо текста опции меню, соответствующей текущему значению currentPosition, пустую строку. Вторая строка выполняет то же в отношении значения данной опции меню. Как только цикл пройдет по всем пунктам, мы получим пустое разворачивающееся меню, готовое к приему значений новой группы товаров.

Теперь, после очистки меню, настало время заполнить его товарами из вновь выбранной группы. Мы сможем узнать, какие продукты необходимо сюда поместить, получив информацию о том, какую группу товаров предпочел пользователь. Как и в предыдущих проектах, применим обработчик событий для передачи значения в функцию, с помощью которой мы можем проверить, что за опция была указана. Значение для этой функции станет храниться в переменной choice. Следующий добавленный нами фрагмент кода будет представлять собой ряд операторов if, предназначенный для определения выбранной группы:

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker.Products
            .options[currentPosition].text = ' ';
        document.ProductPicker.Products
            .options[test].value = ' ';
        ++currentPosition;
    }
    // Проверить, какая группа продуктов была выбрана
    if (choice == 0)
    {
        ...
    }
    else if (choice == 1)
    {
        ...
    }
    else if (choice == 2)
    {
        ...
    }
    else if (choice == 3)
    {
        ...
    }
    else if (choice == 4)
    {
        ...
    }
    ...
}
```

Изучив код HTML разворачивающихся меню, вы можете заметить, что значения, присвоенные группам товаров, представляют собой числа от 1 до 4, и каждое число соответствует номеру группы:

```
<SELECT NAME="ProductGroup">
  <OPTION VALUE="0">Choose a Group
  <OPTION VALUE="1">Group One
  <OPTION VALUE="2">Group Two
  <OPTION VALUE="3">Group Three
  <OPTION VALUE="4">Group Four
```

[illegible]

Поскольку не исключен выбор предусмотренного по умолчанию входа меню Choose a Group, мы присвоили этому варианту значение 0. Когда обработчик событий начнет действовать, он передаст выбранное значение функции, присвоит его переменной choice, и только что введенные нами операторы if выполнят команды, следующие за проверкой, результаты которой оказались истинными.

Теперь мы можем разместить в сценарии команды, что должны быть выполнены после каждой ветви оператора `if`. Эти команды предназначены для осуществления двух действий: присвоения переменной значения длины выбранного массива группы товаров `Products`, а затем — присвоения другой переменной имени выбранной группы.

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker.Products
        .options[currentPosition].text =
        document.ProductPicker.Products
        .options[test].value = '';
        ++currentPosition;
    }
    // Проверить, какая группа продуктов была выбрана
    if (choice == 0)
    {
        ...
    }
    else if (choice == 1)
    {
        OutputListSize = group1 length;
    }
}
```

```

        arrayName = "group1";
    }
    else if (choice == 2)
    {
        OutputListSize = group2_length;
        arrayName = "group2";
    }
    else if (choice == 3)
    {
        OutputListSize = group3_length;
        arrayName = "group3";
    }
    else if (choice == 4)
    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }
    ...
}

```

Ранее в этом сценарии мы уже определили значения длины отдельных массивов и присвоили их разным переменным, таким, как `group1_length`, `group2_length` и т.д. Для последующего использования данных значений необходимо иметь в нашей функции единственную переменную, которая хранила бы значение длины массива только для выбранной группы. Поэтому в приведенных выше строках кода мы присвоили длину массива для выбранной группы переменной `OutputListSize`.

Вторая строка, добавленная к каждой ветви оператора `if`, присваивает имя выбранной группы переменной `arrayName`. Эта переменная в дальнейшем поможет вызвать информацию из соответствующего массива.

До сих пор мы разрабатывали код всех ветвей оператора `if`, кроме первой, которая будет выполнена, если пользователь выберет предусмотренную по умолчанию опцию меню `Choose a Group`. Поскольку данная опция не соответствует ни одной группе товаров, то отсутствуют и массивы товаров, которые бы заполнили второе меню, и нам не надо думать о поиске длины массива или имени группы товаров. А именно это и нужно сделать, если будет выбрана названная опция. Поскольку после выполнения этой функции второе меню будет автоматически стерто, нам остается только заполнить предусмотренную по умолчанию опцию данного меню текстом `Choose a Group`. Поэтому добавим следующую команду, которая явно присваивает значение свойству `text` первой опции второго меню.

Сценарий дополнен также второй командой, выбирающей первую опцию меню.

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker.Products
        .options[currentPosition].text = '';
        document.ProductPicker.Products
        .options[currentPosition].value = '';
        ++currentPosition;
    }
    // Проверить, какая группа продуктов была выбрана
    if (choice == 0)
    {
        document.ProductPicker.Products
        .options[0].text = 'Choose a Group';
        document.ProductPicker.Products
        .options[0].selected=true;
    }
    else if (choice == 1)
    {
        OutputListSize = group1_length;
        arrayName = "group1";
    }
    else if (choice == 2)
    {
        OutputListSize = group2_length;
        arrayName = "group2";
    }
    else if (choice == 3)
    {
        OutputListSize = group3_length;
        arrayName = "group3";
    }
    else if (choice == 4)
    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }
}
```

У нас есть вся информация, необходимая для заполнения второго меню товарами выбранной группы. Перейдем к заключительному разделу функции, в котором фактически происходит это заполнение. Сначала надо переустановить в 0 значение переменной `currentPosition` и убедиться в том, что не была выбрана предусмотренная по умолчанию опция `Choose a Group`. Если бы она была выбрана, нам не пришлось бы повторно заполнять меню, поэтому используем оператор `if` и удостоверимся, что значение `choice` больше 0.

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
    var currentPosition = 0;
    while (currentPosition < maxLength)
    {
        document.ProductPicker.Products
        .options[currentPosition].text = '';
        document.ProductPicker.Products
        .options[currentPosition].value = '';
        ++currentPosition;
    }
    // Проверить, какая группа продуктов была выбрана
    if (choice == 0)
    {
        document.ProductPicker.Products
        .options[0].text = 'Choose a Group';
        document.ProductPicker.Products
        .options[0].selected=true;
    }
    else if (choice == 1)
    {
        OutputListSize = group1_length;
        arrayName = "group1";
    }
    else if (choice == 2)
    {
        OutputListSize = group2_length;
        arrayName = "group2";
    }
    else if (choice == 3)
    {
        OutputListSize = group3_length;
        arrayName = "group3";
    }
    else if (choice == 4)
    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }
}
```

```

    {
        OutputListSize = group4_length;
        arrayName = "group4";
    }
    var currentPosition = 0;
    if (choice > 0)
    {
        ...
    }
}

```

Теперь, установив, что действительно была выбрана новая группа товаров, мы приступаем к заполнению второго меню. Для этого можно воспользоваться таким же циклом while, как при очистке меню в начале функции. Он сверит значение переменной currentPosition с переменной OutputListSize, которая содержит длину выбранного массива товаров.

```

var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
    {
        ...
        ++currentPosition;
    }
    ...
}

```

Безусловно, операторы в цикле будут другими. Чтобы заполнить свойства text и value каждой опции меню, необходимо ввести для выполнения в цикле while две строки кода.

```

var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
    {
        document.ProductPicker.Products
        .options[currentPosition].text = eval(arrayName +
        '_names[currentPosition]');

        document.ProductPicker.Products
        .options[currentPosition].value = eval(arrayName +
        '_locations[currentPosition]');
    }
}

```

```

        ++currentPosition;
    }
    ...
}

```

В этих строках выполняется очень многое, поэтому приглядимся к ним более внимательно. В первой половине первой строки мы вызываем свойство text в опции меню, которая в настоящее время соответствует значению переменной currentPosition:

```
document.ProductPicker.Products.options[currentPosition].text
```

При каждом проходе по циклу данному свойству будет присвоено значение того, что находится во второй половине строки.

```
= eval(arrayName + '_names[currentPosition]');
```

Эта часть строки аналогична той, что использовалась в главе 2, когда мы работали со сменяющимися изображениями. Здесь мы применяем метод eval() для конкатенации значения переменной arrayName, хранящей имя выбранной группы, со значением такой строки, которая при ее добавлении позволит обратиться к необходимому массиву товаров и выбрать вход, находящийся в позиции, соответствующей текущему значению переменной currentPosition.

В то время как первая строка отвечает за заполнение свойства text опции разворачивающегося меню, вторая повторно заполняет свойство value каждой опции.

Последняя команда, которую требуется добавить к функции, призвана обеспечить установку выбранной по умолчанию позиции в качестве первой во вновь заполненном втором меню:

```

var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
    {
        document.ProductPicker.Products
        .options[currentPosition].text = eval(arrayName +
        '_names[currentPosition]');

        document.ProductPicker.Products
        .options[currentPosition].value = eval(arrayName +
        '_locations[currentPosition]');

        ++currentPosition;
    }
}

```

```

    }
    document.ProductPicker.Products
    .options[0].selected=true;
  }
}

```

Поздравляем с успешной разработкой функции! Это, безусловно, очень сложный фрагмент кода, поэтому вы вправе гордиться собой. Осталось пройти еще два этапа и проект будет завершен. Теперь нужно добавить функцию навигации, направляющую пользователя на страницу выбранного товара, а затем вставить обработчики событий, которые вызовут эту функцию.

Создание функции навигации

Итак, нам необходима функция для перевода пользователя на другую страницу в зависимости от того, какой пункт разворачивающегося меню Products он выбрал. Но — стоп, ведь мы это уже сделали! Мы разработали именно такую функцию в начале главы. Внеся небольшие изменения, можно применить ее и для быстрой навигации по ссылкам, и для разворачивающегося меню.

Когда страница только что загружена и пользователь еще не выбрал группу товаров, меню Products является пустым и останется таким, пока выбор не будет сделан. Дополнение, которое следует внести в функцию, состоит в обеспечении заполнения данными любой опции, выбранной из разворачивающегося меню Products. Мы должны исключить возможность того, что функция навигации отправит посетителя на несуществующую страницу.

Рассмотрим еще раз разработанную нами функцию, чтобы, так сказать, освежить ее в памяти.

```

// Функция навигации разворачивающегося меню
function PageChanger(page)
{
    document.location = page;
    document.NavForm.PullDown
    .options[0].selected = true;
}

```

Для выполнения проверки мы можем использовать значение опции меню, переданное в функцию. Вставим оператор if, который сообщит интерпретатору JavaScript, чтобы он выполнил оставшуюся часть функции только в том случае, если значение переменной page не равно (!=) пустой строке.

```

// Функция навигации разворачивающегося меню
function PageChanger(page)
{
    if (page != ' ')
    {
        document.location=page;
        document.NavForm.PullDown
        .options[0].selected=true;
    }
}

```

Таким образом, добавив один простой оператор if, мы получили возможность расширить область применения уже имеющейся функции. Включите этот прием в свой арсенал, используемый при разработке кода, — не стоит снова и снова изобретать колесо.

Теперь, когда у нас есть функция, позволяющая направить пользователя на выбранную им страницу, осталось только ввести обработчики событий.

Вставка обработчиков событий

Нам надо вставить два обработчика событий, по одному для каждого из двух разворачивающихся меню. Меню ProductGroup вызывает функцию MenuFiller(), а меню Products — функцию PageChanger(). Рассмотрим сначала меню ProductGroup:

```

<SELECT NAME="ProductGroup"
onChange="MenuFiller(this.options[this.selectedIndex]
.value)">
    <OPTION VALUE="0">Choose a Group
    <OPTION VALUE="1">Group One
    <OPTION VALUE="2">Group Two
    <OPTION VALUE="3">Group Three
    <OPTION VALUE="4">Group Four
</SELECT>

```

Этот обработчик событий должен показаться вам знакомым. В основном он совпадает с тем, что применялся в навигационном разворачивающемся меню в начале данной главы. Единственное отличие: теперь мы вызываем другую функцию. Поэтому можно использовать такой же код:

```

<SELECT NAME="Products"
onChange="PageChanger(this.options[this.selectedIndex]
.value)">
    <OPTION VALUE="">Choose a

```


[illegible]

Хотите верить, хотите нет, но работа завершена. Это, безусловно, самый длинный сценарий из тех, с которыми мы до сих пор имели дело, поэтому рассмотрим его в завершение как единое целое и перечислим все, что мы сделали.

Описание сценария

Первым этапом написания данного сценария стало определение массивов, которые будут хранить информацию о всех продуктах каждой группы товаров:

```
// Массивы для товаров
// Определить массивы
group1_names = new Array(5);
group1_locations = new Array(5);

group2_names = new Array(5);
group2_locations = new Array(5);

group3_names = new Array(5);
group3_locations = new Array (5);

group4_names = new Array(5);
group4_locations = new Array(5);
```

1. Мы создали по два массива для любой из групп товаров: один набор массивов хранит наименования продуктов каждой группы, а другой — URL, соответствующие отдельным товарам.

Следующим действием стало заполнение образованных массивов информацией о товарах.

```
// Заполнить массивы Group1
group1_names[0] = 'Group1_product one';
group1_locations[0] = 'group1/prod_1.html';

group1_names[1] = 'Group1_product two';
group1_locations[1] = 'group1/prod_2.html';

group1_names[2] = 'Group1 product three';
```

```
group1_locations[2] = 'group1/prod_3.html';

group1_names[3] = 'Group1_product four';
group1_locations[3] = 'group1/prod_4.html';

group1_names[4] = 'Group1_product five';
group1_locations[4] = 'group1/prod_5.html';

var group1_length = group1_names.length;

// Заполнить массивы Group2
group2_names[0] = 'Group2_product one';
group2_locations[0] = 'group2/prod_1.html';

group2_names[1] = 'Group2_product two';
group2_locations[1] = 'group2/prod_2.html';

group2_names[2] = 'Group2_product three';
group2_locations[2] = 'group2/prod_3.html';

group2_names[3] = 'Group2_product four';
group2_locations[3] = 'group2/prod_4.html';

group2_names[4] = 'Group2_product five';
group2_locations[4] = 'group2/prod_5.html';

var group2_length = group2_names.length;

// Заполнить массивы Group3
group3_names[0] = 'Group3_product one';
group3_locations[0] = 'group3/prod_1.html';

group3_names[1] = 'Group3_product two';
group3_locations[1] = 'group3/prod_2.html';

group3_names[2] = 'Group3_product three';
group3_locations[2] = 'group3/prod_3.html';

group3_names[3] = 'Group3_product four';
group3_locations[3] = 'group3/prod_4.html';

group3_names[4] = 'Group3_product five';
group3_locations[4] = 'group3/prod_5.html';

var group3_length = group3_names.length;
```

```
// Заполнить массивы Group4
group4_names[0] = 'Group4_product one';
group4_locations[0] = 'group4/prod_1.html';

group4_names[1] = 'Group4_product two';
group4_locations[1] = 'group4/prod_2.html';

group4_names[2] = 'Group4_product three';
group4_locations[2] = 'group4/prod_3.html';

group4_names[3] = 'Group4_product four';
group4_locations[3] = 'group4/prod_4.html';

group4_names[4] = 'Group4_product five';
group4_locations[4] = 'group4/prod_5.html';
```

```
var group4_length = group4_names.length;
```

- После заполнения всех массивов была создана переменная для каждой группы, хранящая длину массивов этих групп.

```
// Найти длину самого большого набора массивов
var maxLength = group1_length;
if (group2_length > maxLength)
{
    maxLength = group2_length;
}
if (group3_length > maxLength)
{
    maxLength = group3_length;
}
if (group4_length > maxLength)
{
    maxLength = group4_length;
}
```

- Получив хранящиеся в переменных данные о длине всех массивов, мы установили разветвленный оператор if для определения значения самой протяженной группы и присвоения этого значения переменной maxLength.

Затем была создана функция MenuFiller(), предназначенная для заполнения разворачивающихся меню.

```
// Создание функции многоступенчатого меню
function MenuFiller(choice)
{
    // Очистить второе меню
```

```
var currentPosition = 0;
while (currentPosition < maxLength)
{
    document.ProductPicker.Products
        .options[currentPosition].text = ' ';

    document.ProductPicker.Products
        .options[currentPosition].value = ' ';

    ++currentPosition;
}
// Проверить, какая группа продуктов была выбрана
if (choice == 0)
{
    document.ProductPicker.Products
        .options[0].text = 'Choose a Group';

    document.ProductPicker.Products
        .options[0].value = '';

    document.ProductPicker.Products
        .options[0].selected=true;
}
else if (choice == 1)
{
    OutputListSize = group1_length;
    arrayName = "group1";
}
else if (choice == 2)
{
    OutputListSize = group2_length;
    arrayName = "group2";
}
else if (choice == 3)
{
    OutputListSize = group3_length;
    arrayName = "group3";
}
else if (choice == 4)
{
    OutputListSize = group4_length;
    arrayName = "group4";
}
var currentPosition = 0;
if (choice > 0)
{
    while (currentPosition < OutputListSize)
```

```

        document.ProductPicker.Products
        .options[currentPosition].text = eval(arrayName +
        '_names[currentPosition]');

        document.ProductPicker.Products
        .options[currentPosition].value = eval(arrayName +
        '_locations[currentPosition]');

        ++currentPosition;
    }

    document.ProductPicker.Products
    .options[0].selected=true;
}

```

1. Вначале мы применяем функцию для очистки всех данных в разворачивающемся меню Products.
2. Затем функция принимает переменную, переданную в нее из обработчика событий, и определяет, какую группу товаров выбрал пользователь.
3. Теперь функция заполняет меню Products информацией из соответствующего массива.

Закончив разработку данной функции, мы внесли небольшое исправление в имеющуюся функцию навигации.

```
// Функция навигации разворачивающегося меню
function PageChanger(page)
```

```
{
    if (page != ' ')
    {
        document.location=page;
        document.NavForm.PullDown
            .options[0].selected=true;
    }
}
```

1. Здесь в функцию PageChanger() добавлен оператор if для проверки того, является ли действительной опция, выбранная из разворачивающегося меню.

И, наконец, в два разворачивающихся меню, ProductGroup и Products, введены обработчики событий.

```
<SELECT NAME="ProductGroup"
  onChange="MenuFiller(this.options[this.selectedIndex]
.value)">
  <OPTION VALUE="0">Choose a Group
  <OPTION VALUE="1">Group One
  <OPTION VALUE="2" >Group Two
  <OPTION VALUE="3">Group Three
  <OPTION VALUE="4">Group Four
</SELECT>
```

[illegible]

2. В эти меню вставлены обработчики onChange.

Обработчик событий в меню ProductGroup вызывает функцию MenuFiller().

Обработчик событий в меню Products вызывает функцию PageChanger().

Перечислим новые понятия, встретившиеся нам при работе над данным проектом:

- создание и заполнение массивов, определяемых пользователем;
- цикл while;
- оператор сравнения != ;
- доступ к информации и свойствам, содержащимся в массивах;
- доступ и изменение значений и свойств элементов формы.

Настало время показать выполненную работу руководителю, чтобы получить заслуженную похвалу. Вы становитесь настоящим специалистом по языку JavaScript! Безусловно, после того, как вы закончили данный проект, босс не даст вам покоя и быстро найдет другое задание. Что делать, для нас работа никогда не кончается!

◆ Проект III: Применение JavaScript на странице регистрации

Компания Shelley Biotech намерена вставить в свой узел новый раздел — Partners. Часть данного раздела не будет доступна для широкой публики: вход разрешен только для деловых партнеров. Программисты компании, используя язык Perl, создали страницу регистрации (рис. 3.4), написали сценарий и собираются поддерживать базу данных с именами пользователей и паролями. Но какое отношение это имеет к вам?

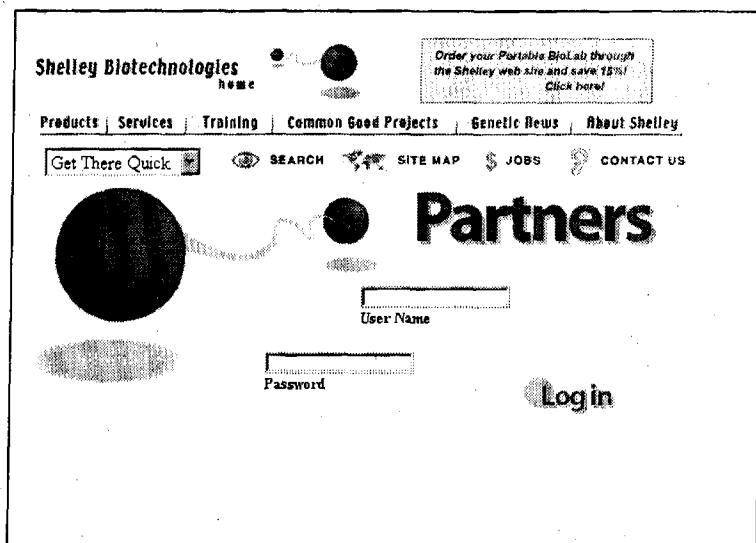


Рис. 3.4. Страница регистрации раздела Partners

А вот какое. Здесь нельзя обойтись без страницы регистрации, позволяющей ввести имя и пароль, а затем нажать клавишу Enter, чтобы вызвать сценарий регистрации. Страница, предложенная разработчиками на языке Perl, требует, чтобы пользователь нажимал вручную кнопку регистрации для вызова сценария. Наша задача — так откорректировать созданную страницу с помощью языка JavaScript, чтобы регистрация происходила при нажатии клавиши Enter.

После реализации последнего сценария это задание может показаться легкой прогулкой. Не нужно даже писать функцию. Достаточно вставить в код HTML соответствующий обработчик событий, и все будет готово.

Вставка обработчика событий

В двух последних проектах главы 3 обработчик событий on-Change применялся в разворачивающихся меню формы для вызова функций, необходимых для выполнения сценариев. В новом проекте мы опять используем onChange, только на этот раз посмотрим, как он работает с текстовым полем формы.

При размещении обработчика onChange в текстовом поле формы он активизируется, когда пользователь изменяет значение, находящееся внутри поля, а потом либо нажимает клавишу Tab или Enter, либо устанавливает курсор за пределами поля. В результате текстовое поле теряет фокус, что приводит затем к активизации обработчика событий. В данном сценарии мы поместим onChange в поле пароля. Когда пользователь посещает страницу, он сначала вводит свое имя, а потом переходит к полю пароля. Он вводит пароль, изменяя тем самым значение поля, после чего либо нажимает клавишу Enter, либо щелкает на кнопке регистрации Log-in. Вследствие этого поле пароля теряет фокус и происходит активизация события. Активизируемое нами событие состоит в вызове свойства SUBMIT формы, содержащей текстовые поля регистрации.

Рассмотрим код после вставки обработчика событий:

```
<INPUT TYPE="password" NAME="_User_Pass"
  onChange="document.login.submit();" >
```

Описание сценария

1. Мы вставили обработчик событий onChange в поле пароля формы.

Это все, что нужно было сделать. Превосходный пример очень короткого кода, который может оказаться действительно полезным. Ведь не все сценарии должны быть такими длинными или сложными, как второй из данной главы. Надо всегда стремиться найти наиболее эффективный способ создания сценария.

◆ Резюме

Теперь вы должны уже полностью представлять себе, как применять JavaScript для навигации, а также для управления формами в целях достижения максимальной эффективности и улучшения взаимодействия с пользователем. Вы также значительно расширили свои знания о массивах, научились их создавать и обращаться к хранящейся в них информации. В следующей главе мы оставим Web-узел Shelley Biotech и приступим к работе с узлом журнала Stitch. При этом мы мобилизуем все полученные знания о JavaScript и формах и изучим, как осуществлять проверку ошибок с помощью данного языка.

◆ Дополнительное задание

1. Создайте систему из трех разворачивающихся меню, в которой второе меню заполняется с учетом выбора, сделанного в первом, а третье — с учетом выбора, сделанного во втором меню.
2. Создайте систему из трех разворачивающихся меню, в которой выбор, сделанный в первом меню, приводит к динамическому заполнению двух других.



В этой главе

- Проект I: Контроль пустых полей формы
- Проект II: Контроль данных в полях формы
- Проект III: Уведомление пользователя об ошибках
- Резюме
- Дополнительное задание

Во второй части книги мы займемся разработкой Web-узла журнала *Stitch* — издания о современной моде. Это — красочный, популярный журнал, авторы которого анализируют основные тенденции развития моды, проникают в тайны творчества дизайнеров и выискивают сенсационные новости. Издание пользуется все большим спросом и, естественно, встал вопрос об улучшении доступа к информационному наполнению его Web-узла.

В двух последующих главах мы применим JavaScript для внесения изменений в кадры сетевых страниц *Stitch*, используя при этом наиболее эффективные средства языка для максимального привлечения внимания пользователя. В главе 5 будут рассмотрены способы проверки формы на предмет наличия в них ошибок, совершенных посетителем узла.

Маркетологи журнала решили провести розыгрыш призов среди Internet-читателей. Право включиться в соревнование получают те, кто, путешествуя по Сети, зашел на узел *Stitch* и ввел в форме некоторую информацию о себе (например, указал свое имя, адрес электронной почты и сообщил, что хотел бы узнать о моде).

Вероятность того, что большинство посетителей введут информацию правильно, весьма велика. Но некоторые могут неверно ввести ответы, другие не укажут информацию, которая крайне важна для специалистов по маркетингу, а третьи просто заполняют форму всякой чепухой. Если вы включите код JavaScript для контроля ошибок в форме перед ее передачей в сценарий CGI, маркетологам придется отсеивать меньше неполных или лишенных смысла данных. Проверка ошибок упростит и жизнь программиста CGI, поскольку поступающие данные будут гарантированно иметь определенную структуру.

Сценарий этот довольно длинный, поэтому разобьем его на три части (проекта). Прежде всего, проверим, нет ли в форме пустых полей. Затем установим, имеются ли ошибочные данные, введенные пользователем. И, наконец, опишем, каким образом дать пользователю знать, что он ошибся. Поскольку все три проекта фактически являются частями одного и того же сценария, мы еще раз просмотрим каждую эту часть в конце главы, когда сценарий будет закончен.

◆ Проект I: Контроль пустых полей формы

Специалисты по маркетингу хотят, чтобы посетители узла заполняли все поля в форме. Но в действительности для формирования

группы участников розыгрыша нужны только два элемента данных (рис. 4.1). Обычно следует настаивать на заполнении полей имени и адреса электронной почты как самых важных. Без крайней необходимости не требуйте от пользователя, чтобы он указал, например, номер своего телефона. Человек может подумать, что вы собираетесь напустить на него представителей фирм, занимающихся электронной торговлей, и, скорее всего, вместо настоящего номера введет вымышленный.

Рис. 4.1. Страница формы регистрации участников розыгрыша призов на узле Stitch

Перед вами форма, разработанная программистами HTML журнала Stitch:

```
<FORM ACTION="/scripts/promotion.cgi" NAME="promotion"
METHOD="post">
Fields with a <B><FONT COLOR="#FF0000">*</FONT></B> are
required fields.
<P>
<TABLE BORDER="0">
  <TR>
    <TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
    <TD><B>First Name</B></TD>
    <TD><INPUT TYPE="text" NAME="firstName"></TD>
  </TR>
  <TR>
    <TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
    <TD><B>Last Name</B></TD>
    <TD><INPUT TYPE="text" NAME="lastName"></TD>
  </TR>
</TABLE>
<TABLE BORDER="0">
  <TR>
    <TD COLSPAN="3">
      <B>Mailing Address</B>
    </TD>
  </TR>
  <TR>
    <TD>&nbsp;</TD>
    <TD><B>Street</B></TD>
    <TD><INPUT TYPE="text" SIZE="40" NAME="address"></TD>
  </TR>
  <TR>
    <TD>&nbsp;</TD>
    <TD><B>City</B></TD>
    <TD><INPUT TYPE="text" NAME="address">&nbsp;&nbsp;&nbsp;</TD>
  </TR>
  <TR>
    <TD><B>State</B></TD>
    <TD><INPUT TYPE="text" NAME="address"
      MAXLENGTH="2" SIZE="2">&nbsp;&nbsp;&nbsp;</TD>
    <TD><B>Zip
      Code</B></TD>
    <TD><INPUT TYPE="text" NAME="zipCode"
      MAXLENGTH="5" SIZE="5">
    </TD>
  </TR>
</TABLE>
<TABLE>
  <TR>
    <TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
    <TD><B>Email Address</B></TD>
    <TD><INPUT TYPE="text" NAME="email"></TD>
  </TR>
</TABLE>
```

```

<TD>&nbsp;</TD>
<TD><B>Phone Number</B></TD>
<TD>
  <INPUT TYPE="text" NAME="phonenummer" SIZE="12"
    MAXLENGTH="12">
</TD>
</TR>
<TR>
  <TD>&nbsp;</TD>
  <TD><B>Fax Number</B></TD>
  <TD>
    <INPUT TYPE="text" NAME="faxnumber" SIZE="12"
      MAXLENGTH="12">
  </TD>
</TR>
<TR>
  <TD>&nbsp;</TD>
  <TD><B>What interests you most<BR>
    about clothes and fashion?</B></TD>
  <TD>
    <TEXTAREA ROWS=6 COLS=40>Be honest.</TEXTAREA>
  </TD>
</TR>
<TR>
  <TD><B><FONT COLOR="#FF0000">*</FONT></B></TD>
  <TD>
    <B>What do you want to see in an<BR>
      online fasion magazine like Stitch?</B>
  </TD>
  <TD>
    <SELECT NAME="whatTheyWant">
      <OPTION VALUE="">Please choose a category
      <OPTION VALUE="trends">Upcoming trends
      <OPTION VALUE="models">Information about
        models
      <OPTION VALUE="designers">Information about
        designers
      <OPTION VALUE="rage">What the current rage is
      <OPTION VALUE="gossip">Gossip!
    </SELECT>
  </TD>
</TR>
<TR>
  <TD COLSPAN="2">Want to join?</TD>
  <TD><INPUT TYPE="checkbox" NAME="join" SELECTED></TD>
</TR>
<TR>
  <TD COLSPAN="2">
    <INPUT TYPE="button" VALUE="Click to Submit"
      onClick="errorCheck()">

```

```

</TD>
</TR>
</TABLE>
</FORM>

```

Обратите внимание, что в конце формы программисты уже вставили обработчик событий `onClick` в тег кнопки `Submit`, при нажатии которой вызывается функция `errorCheck()`. Поэтому используем данное имя в качестве функции, которая должна быть создана.

Проверка формы на отсутствие пустых текстовых полей

Приведем первый фрагмент кода JavaScript, который будет использоваться для проверки введенной посетителем информации:

```

<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Создать функцию errorCheck
function errorCheck()
{
  // Создать переменные для сценария
  var requiredFieldsErrorMessage = "";
  var firstNameEntered = "";
  var lastNameEntered = "";
  var emailEntered = "";
  var areaCodeEntered = "";
  var zipCodeEntered = "";
  var pullDownErrorMessage = "";
  var emailErrorMessage = "";
  var areaCodeErrorMessage = "";
  var zipCodeErrorMessage = "";
  ...
}
// Здесь сокращение кода прекращается -->
</SCRIPT>

```

В этом коде установлены некоторые переменные. Первые четыре из них будут применяться в текущем проекте, а остальные — в следующих. Переменные станут хранить конкретные сообщения об ошибках, допущенных пользователем. В конце сценария вся информация об ошибках будет собрана вместе и посетителя узла ждет одно сводное сообщение.

Начнем постепенно вводить функции контроля информации.

```

<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код

```



```
// Создать функцию errorCheck
function errorCheck()
{
    // Создать переменные для сценария
    var requiredFieldsErrorMessage = "";
    var firstNameEntered = "";
    var lastNameEntered = "";
    var emailEntered = "";
    var areaCodeEntered = "";
    var zipCodeEntered = "";
    var pullDownErrorMessage = "";
    var emailErrorMessage = "";
    var areaCodeErrorMessage = "";
    var zipCodeErrorMessage = "";

    // Обязательная проверка ошибок в полях
    firstNameEntered = document.forms['promotion']
        .elements['firstName'].value

    lastNameEntered = document.forms['promotion']
        .elements['lastName'].value

    emailEntered = document.forms['promotion']
        .elements['email'].value
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Мы требуем от посетителя, чтобы он обязательно ввел информацию в трех полях: полях имени, фамилии и адреса электронной почты. Из предыдущих глав мы узнали, как обращаться к значениям элементов тега FORM на HTML-странице. Теперь применим аналогичные способы для получения значений трех полей, которые необходимо проверить. Таким образом, строка

```
firstNameEntered =
document.forms['promotion'].elements['firstName'].value
```

контролирует, что за информацию ввел посетитель в текстовом поле имени, а затем помещает это значение в переменную firstNameEntered. За данной строкой следуют аналогичные операторы для двух других полей. Если пользователь не введет никакой информации в одно из этих полей, переменная получит пустое значение. Следующий шаг — применение оператора if для проверки значений.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Создать функцию errorCheck
```

```
function errorCheck()
{
    // Создать переменные для сценария
    var requiredFieldsErrorMessage = "";
    var firstNameEntered = "";
    var lastNameEntered = "";
    var emailEntered = "";
    var areaCodeEntered = "";
    var zipCodeEntered = "";
    var pullDownErrorMessage = "";
    var emailErrorMessage = "";
    var areaCodeErrorMessage = "";
    var zipCodeErrorMessage = "";

    // Обязательная проверка ошибок в полях
    firstNameEntered = document.forms['promotion']
        .elements['firstName'].value

    lastNameEntered = document.forms['promotion']
        .elements['lastName'].value

    emailEntered = document.forms['promotion']
        .elements['email'].value

    if ((!firstNameEntered) || (!lastNameEntered) ||
        (!emailEntered))
    {
        if (!firstNameEntered)
        {
            requiredFieldsErrorMessage = "- your
            first name\r"
        }
        if (!lastNameEntered)
        {
            requiredFieldsErrorMessage =
            requiredFieldsErrorMessage + "- your
            last name\r"
        }
        if (!emailEntered)
        {
            requiredFieldsErrorMessage =
            requiredFieldsErrorMessage + "- your
            email address\r"
        }
    }
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Первая строка этого оператора имеет такой вид:

```
if ((!firstNameEntered) || (!lastNameEntered) ||
    (!emailEntered))
```

Она содержит следующее сообщение для интерпретатора JavaScript: если посетитель не ввел имя, фамилию или адрес электронной почты, то условие оператора `if` принимает истинное значение.

Двойной символ канала (`||`) применяется в условных операторах в качестве логического оператора “или”. При его использовании, например, в таком выражении, как проверка требований оператора `if`, выполнение любого из условий, соединенных данным оператором, рассматривается как истина и реализуются операторы, следующие за этим условием. В приведенной выше строке применяется также оператор отрицания (`!`). В языке JavaScript (и в большинстве других языков программирования) это — символ “не”. Поэтому в данном случае интерпретатору указано, чтобы он вернул значение истина, если одна из переменных пуста. Затем интерпретатор перейдет к выполнению операторов, следующих за условием. Если все три поля будут заполнены, условие примет ложное значение и сценарий просто перейдет к реализации кода, находящегося за оператором `if`.

Вполне возможно, что не будет заполнено ни одно из полей, поэтому введем код, выполняемый при возникновении любых ошибок. Добавим еще три оператора `if`, проверяющих все три поля отдельно и, в случае необходимости, записывающих короткое сообщение в переменную `requiredFieldsErrorMessage`, которую мы затем вызовем, чтобы указать пользователю, какие поля он пропустил.

```
if (!firstNameEntered)
```

Приведенный оператор `if` означает, что если посетитель не ввел имя, то будет выполнена следующая строка:

```
requiredFieldsErrorMessage = "- your first name\r";
```

Она присваивает строковое значение “- your first name” переменной `requiredFieldsErrorMessage`. Обратите внимание на символ `\r` в конце строки. Как сообщалось в предыдущих главах, символ обратного слэша придает особое значение символу, следующему за ним, и указывает интерпретатору JavaScript, что тот должен его выполнить. Значение `r` указывает, что интерпретатору следует вставить символ перевода строки, аналогичный нажатию клавиши Enter.

Есть еще два оператора `if`, проверяющих наличие значения в двух других полях, которые нас интересуют, и добавляющих информацию к переменной `requiredFieldsErrorMessage`, если эти поля пусты.

```
requiredFieldsErrorMessage = requiredFieldsErrorMessage + "-
your last name\r";
```

Показанный выше оператор будет выполнен в том случае, если не заполнено поле фамилии. Надо добавить текст к существующему сообщению об ошибках посетителя, поскольку может быть допущено несколько ошибок. Поэтому в правой части выражения сценарий указывает интерпретатору, чтобы он добавил строку “- your last name\r” к существующему значению переменной `requiredFieldsErrorMessage`, а затем снова поместил полученную строку в эту переменную. Если ошибок несколько, необходимо все их перечислить, а не просто заменить первую ошибку второй.

В третьем операторе `if` применяется тот же синтаксис и выполняется проверка наличия значения в поле адреса электронной почты.

Проверка выбора опции из разворачивающегося меню

Теперь нужно определить, была ли выбрана опция из разворачивающегося меню с вопросом о том, что пользователь хочет видеть в Internet-версии журнала. К счастью, требующийся нам код очень похож на тот, который мы уже использовали для проверки значений текстовых полей, содержащих имя пользователя и адрес электронной почты. Мы вставим код JavaScript для проверки разворачивающегося меню вслед за созданным нами кодом определения текстовых полей.

```
// Контроль ошибок выбора из разворачивающегося меню
if (document.forms['promotion'].elements['whatTheyWant'].
    value == 'default')
{
    pullDownErrorMessage = "- you didn't tell us what
    you want to see in Stitch";
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Поскольку мы проверяем только одно поле формы, нам достаточно применить один оператор `if`. В приведенном выше коде мы сверяли значение поля `whatTheyWant` со значением, предусмотренным по умолчанию. Если вы рассмотрите код HTML для разворачивающегося меню, то увидите, что значением по умолчанию является

значение, присвоенное первой опции меню. Поэтому, если пользователь не изменил его по сравнению с первоначальным положением и не выбрал другую опцию, значение останется равным значению по умолчанию и сценарий присвоит короткое сообщение переменной `pullDownErrorMessage`.

До сих пор мы разрабатывали сценарий, который будет устанавливать наличие пустых полей в форме. Однако в некоторых случаях и заполненные поля не содержат достаточного объема информации. В определенных обстоятельствах может быть полезно проверять отсутствие неверных или ошибочных ответов пользователя. В следующем проекте представлено несколько примеров подобных случаев.

◆ Проект II: Контроль данных в полях формы

Проверка символов в адресах электронной почты

Получив адрес электронной почты пользователя, нельзя быть полностью уверенным в том, что он истинный. Не существует общей формулы или централизованной базы данных, позволяющей следить за всеми допустимыми адресами электронной почты в мире. Однако вы можете хотя бы проверить, что пользователь предоставил вам нечто похожее на такой адрес, если убедитесь в наличии символа `@` и точки. Несмотря на то, что в настоящем разделе рассматривается как бы новый проект, все фрагменты кода в данной главе фактически являются частью одного сценария. Мы добавим новый код после имеющегося, который все еще находится внутри функции `errorCheck()`.

```
// Контроль ошибок ввода адреса электронной почты
if (emailEntered)
{
    emailEntered = document.forms['promotion']
        .elements['email'].value;

    emailValue = new String(emailEntered);
    emailHasAt = emailValue.indexOf("@");
    emailHasPeriod = emailValue.indexOf(".");
    if ((emailHasAt == -1) || (emailHasPeriod == -1))
    {
        emailErrorMessage = "-your email address\r";
    }
}
// Здесь сокрытие кода прекращается --->
</SCRIPT>
```

Первым делом мы опять выбираем значение рассматриваемого поля и присваиваем его переменной. Затем создаем новый объект с именем `emailValue`. Это выполняется так же, как при построении объектов `IMAGE`, используемых в главе 2 для сменяемых изображений. Только вместо объекта `IMAGE` мы создаем объект `STRING`, хранящий значение, которое теперь находится в переменной `emailEntered`:

```
emailValue = new String(emailEntered);
```

Почему мы помещаем значение поля в объект вместо того, чтобы просто держать его в обычной переменной, как делалось до сих пор? Дело в том, что объекты являются более гибкими и позволяют обращаться к методам, которые нельзя использовать для переменных. Например, в создаваемом сейчас сценарии будет применяться метод `indexOf()`, что позволит вести поиск определенных символов в данной строке. В этом случае мы будем искать символ `@` и точку.

```
emailHasAt = emailValue.indexOf("@");
emailHasPeriod = emailValue.indexOf(".");
```

В первой строке осуществляется поиск символа `@` в объекте `emailValue` и происходит присвоение переменной значения позиции, в которой был обнаружен этот символ. Например, если значение `emailValue` составляет `dan@wire-man.com`, то символ `@` находится в позиции 3 (помните, что в JavaScript отсчет позиций символов начинается с 0) и переменная `emailHasAt` станет хранить значение 3. Если символ не будет найден, полученное значение составит -1.

Вторая строка, по сути дела, является аналогичной, но проверяет наличие точки и сохраняет полученное значение в переменной `emailHasPeriod`. Таким образом, строки:

```
if ((emailHasAt == -1) || (emailHasPeriod == -1))
{
    emailErrorMessage = "-your email address\r";
}
```

сообщают интерпретатору, чтобы он создал новое сообщение об ошибке с именем `emailErrorMessage`, если адрес электронной почты не содержит символа `@` или точки.

Следующий этап — проверка поля телефонного номера на наличие кода города.

Проверка кода города в телефонном номере

Здесь мы применим небольшую хитрость: переделаем код HTML в форме таким образом, чтобы из поля телефонного номера поступала

только нужная информация (рис. 4.2). Запомните этот прием на будущее, когда вам самому доведется писать сценарий, — довольно часто можно найти способ изменить код HTML так, чтобы сделать проект более эффективным. Ниже приведен новый код HTML для полей телефонного номера:

```
(<INPUT TYPE="text" NAME="areaCode" SIZE="3" MAXLENGTH="3">
<INPUT TYPE="text" NAME="phoneNumber1" SIZE="3"
MAXLENGTH="3">-
<INPUT TYPE="text" NAME="phoneNumber2" SIZE="4" MAXLENGTH="4">
```

Рис. 4.2. Страница формы регистрации участников розыгрыша призов журнала Stitch с новыми полями телефонного номера

Если мы разделим номер на три разных поля, то сможем проще выделить код города. Благодаря этому нам не придется анализировать весь номер, пытаясь определить, какая его часть обозначает этот код.

Подобная проблема может возникнуть, поскольку люди записывают код города и телефон по-разному, например: (925) 555-1212, 925-555-1212, 925/555.1212 и т.п. Помните об этом, разрабатывая сценарии: всегда можно отыскать более простые решения, чем те, что основаны исключительно на использовании JavaScript. Поэтому смело рассматривайте и другие варианты.

Поле телефонного номера не является обязательным, но если уж посетитель добровольно указал свой номер, следует убедиться, что он ввел и код города. Ведь вы во Всемирной Сети, и у вас нет информации о том, откуда к вам обратился пользователь. (Мы сознательно не касаемся международных телефонных номеров. Если на наш Web-узел будет поступать много обращений из-за границы, мы изменим поля телефонного номера в последующем.)

Поскольку решено использовать три разных поля для телефонного номера, то можно проверить значения этих полей во многом таким же способом, что применялся для проверки полей имени, фамилии и адреса электронной почты.

```
// Проверка кода города
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value

phoneNumber1Entered =
document.forms['promotion'].elements['phoneNumber1'].value
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В первых двух строках мы присвоили значения полей кода города и первой части телефонного номера переменным. В следующих строках мы используем оператор if для проверки того, содержится ли в них необходимая для нас информация.

```
// Проверка кода города
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value

phoneNumber1Entered =
document.forms['promotion'].elements['phoneNumber1'].value
```

```

if ((!areaCodeEntered) && (phoneNumberEntered))
{
    areaCodeErrorMessage = "-please enter your area code\r"
}
// Здесь соккрытие кода прекращается -->
</SCRIPT>

```

До сих пор вы не встречались с оператором &&. Это еще один логический оператор, подобный оператору ||, уже применявшемуся в данном сценарии. Оператор && (логическое “И”) показывает, что должны быть выполнены оба соединенные им условия, и тогда он вернет истинное значение выражения. Поэтому очередная строка:

```

if ((!areaCodeEntered) && (phoneNumberEntered))

```

сообщает интерпретатору JavaScript, что если переменная areaCodeEntered не имеет значения, а phoneNumberEntered — имеет, то он должен выполнить следующие строки. Таким образом, информация об ошибке появится только в том случае, если посетитель ввел телефонный номер без кода города.

Теперь перейдем к последнему полю, которое необходимо проверить.

Проверка наличия цифр в значении почтового индекса

Код JavaScript для установления того, что в почтовом коде посетителя содержатся только цифры, сложнее предыдущих и требует использования некоторых новых средств и функций. В целом задача состоит в просмотре по одному символу пятисимвольной строки, содержащейся в переменной zipCodeEntered, и проверке того, что каждый символ действительно является цифрой. При положительном ответе мы устанавливаем некоторую переменную в значение yes и переходим к следующему символу. Но если мы обнаруживаем символ, не являющийся цифрой, то получаем сообщение об ошибке. Данная часть сценария начинается, как и в большинстве случаев, с получения значения поля, которое нужно проверить, и присвоения его переменной.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
...
}
// Здесь соккрытие кода прекращается -->
</SCRIPT>

```

После этого, поскольку пользователь не обязан вводить адрес, необходимо проверить с помощью оператора if, введено ли вообще что-либо в поле почтового индекса.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    ...
}
// Здесь соккрытие кода прекращается -->
</SCRIPT>

```

Если в поле почтового индекса ничего не введено, сценарий игнорирует последнюю часть проверки и переходит к следующему этапу. Однако, если в поле имеется какое-то значение, надо создать требующиеся переменные и присвоить некоторым из них базовое значение, чтобы иметь возможность проверить, правильно ли введен почтовый индекс.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = '';
    ...
}
// Здесь соккрытие кода прекращается -->
</SCRIPT>

```

Мы образовали две переменные-счетчика, которые будут применяться в циклах while, и третью переменную, служащую для хранения результата проверки.

На очередном этапе создаются два цикла while.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;

```

```

zipCounter = 0;
foundNumber = "";
while (zipCounter < 5)
{
    ...
    zipCounter++;
}
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Приведенный выше сценарий закладывает основу для первого цикла `while`. Он указывает интерпретатору, чтобы тот выполнял следующие операторы до тех пор, пока значение переменной `zipCounter` не превышает 5. На данный момент в цикле находится только один оператор, но вскоре мы введем туда и другие. Однако оператор, который находится сейчас в цикле, является ключевым для предотвращения бесконечного цикла.

```
zipCounter++;
```

Эта строка берет значение `zipCounter` и прибавляет к нему 1, чтобы при новом проходе по циклу переменная `zipCounter` имела значение на 1 выше, чем при предыдущем. Такое действие будет повторяться до тех пор, пока значение переменной не достигнет 5 и условие выполнения цикла больше не будет соблюдаться, а сценарий перейдет к следующему этапу.

Первый цикл `while` применяется для последовательного просмотра всех пяти символов в поле почтового индекса. Далее введем второй цикл `while` в первый и станем проверять в нем последовательно все 10 цифр (0—9).

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;

if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = "";
    while (zipCounter < 5)
    {
        while (numberCounter<10)

```

```

{
    ...
    numberCounter++;
}
if (foundNumber != "yes")
{
    zipCounter = 6;
}
else
{
    zipCounter++;
    numberCounter = 0;
}
}
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Теперь второй цикл `while` будет определять наличие в текущей позиции цифр от 0 до 9 при каждом выполнении первого цикла. На данный момент не будем обращать внимания на оператор `if` в конце первого цикла и введем необходимые операторы во второй цикл.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
    numberCounter = 0;
    zipCounter = 0;
    foundNumber = "";

    while (zipCounter < 5)
    {
        while (numberCounter<10)
        {
            if (zipCodeEntered.substr(zipCounter, 1)
                == numberCounter)
            {
                foundNumber = "yes";
                numberCounter = 11;
            }
            else
            {
                foundNumber = "no";

```

```

    }
    numberCounter++;
  }
  if (foundNumber != "yes")
  {
    zipCounter = 6;
  }
  else
  {
    zipCounter++;
    numberCounter = 0;
  }
}
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Итак, во второй цикл while был введен оператор if. Он представляет собой самую важную часть сценария, так как фактически выполняет проверку значений в поле почтового индекса. Поэтому рассмотрим его более подробно.

Первым делом мы обеспечиваем проверку значения:

```
if (zipCodeEntered.substr(zipCounter, 1) == numberCounter)
```

Здесь метод `substr()` применяется для определения конкретного символа в поле почтового индекса. Используя его в главе 1, мы указывали диапазон символов, который необходимо было выделить. В данном случае мы проверяем единственный символ и указываем на него методу `substr()` с помощью значения переменной `zipCounter`.

Проходя по первому циклу и прибавляя 1 к этой переменной после каждого прохода, мы проверяем символы, находящиеся в позициях 0 — 4 переменной `zipCodeEntered`. То есть, мы тем самым обеспечиваем определение всех пяти символов почтового индекса.

Затем мы последовательно сравниваем эти значения со значением переменной `numberCounter`. Данная переменная является основой вычисления во втором цикле и после каждого прохода по второму циклу ее значение также увеличивается на 1. Поэтому в ходе выполнения второго цикла происходит проверка того, соответствует ли значение символа, находящегося в позиции `zipCounter`, значению переменной `numberCounter`, которая последовательно увеличивается от 0 до 9.

Если будет обнаружено соответствие, оператор `if` перейдет к присвоению переменной `foundNumber` значения `yes` и установит значение `numberCounter` в 11, чтобы при следующем вхождении во второй цикл while условие стало ложным и выполнение сценария продолжилось. Если в этом цикле соответствие не будет найдено, значение `foundNumber` установится в `on`, а `numberCounter` — увеличится на 1, и осуществится еще один проход по второму циклу для проверки соответствия.

После того, как либо второй цикл выполнит все 10 итераций, либо будет найдена цифра, отвечающая текущему символу, сценарий перейдет к реализации первого оператора `if`, введенного в первый цикл while ранее. Этот оператор проверяет значение переменной `foundNumber`. Если во втором цикле while не была обнаружена цифра в текущей позиции, данный оператор установит значение `zipCounter` в 6, в результате чего на следующем проходе выполнение первого цикла while будет прекращено и появится сообщение об ошибке. Если же во втором цикле обнаружится цифра в текущей позиции, оператор `if` увеличит значение `zipCounter` на 1 и разрешит продолжить выполнение цикла для проверки следующего символа в поле почтового индекса.

Последним добавлением к этому разделу сценария является код для передачи еще одного сообщения об ошибке в том случае, если в поле почтового индекса будет обнаружен нецифровой символ.

```

// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
if (zipCodeEntered)
{
  numberCounter = 0;
  zipCounter = 0;
  foundNumber = "";

  while (zipCounter < 5)
  {
    while (numberCounter < 10)
    {
      if (zipCodeEntered.substr(zipCounter, 1)
        == numberCounter)
      {
        foundNumber = "yes";
        numberCounter = 11;
      }
      else

```

```

        {
            foundNumber = "no";
        }
        numberCounter++;
    }
    if (foundNumber != "yes" )
    {
        zipCounter = 6;
    }
    else
    {
        zipCounter++;
        numberCounter = 0;
    }
}
if (foundNumber != "yes")
{
    zipCodeErrorMessage = "-zip code has non-numbers in it\r";
}
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Разработав этот довольно сложный и очень полезный для повышения квалификации код, мы почти закончили работу с данным разделом книги. Осталось лишь выполнить проверку того, были ли выработаны сообщения об ошибках, и показать их пользователю, а если ошибок нет — отправить форму на выполнение в сценарий CGI.

◆ Проект III: Уведомление пользователя об ошибках

В последнем фрагменте кода нет ничего особенно сложного. Здесь мы задействуем методы, которые уже были описаны в этой главе. Будь мир абсолютно идеальным, наверное и пользователи всегда заполняли бы форму безошибочно. Но поскольку дело обстоит несколько иначе, то теперь, когда мы установили все имеющиеся у нас неудовлетворительные данные, следует сообщить о полученных результатах посетителю узла. Для этого применяется простой оператор `if`.

```

// Уведомление пользователя о том, что пропущено
if ((requiredFieldsErrorMessage) ||

```

```

(whatTheyWantErrorMessage) || (emailErrorMessage) ||
(areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    ...
}
else
{
    ...
}
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

В первой строке осуществляется проверка того, имеют ли значение какие-либо переменные, содержащие сообщения об ошибках. Если имеют, значит пользователь в чем-то сплеховал и мы должны составить для него уведомление обо всех обнаруженных промахах. Мы выполним это, присвоив на первом этапе переменной `alertMessage` начало сообщения, а затем добавляя в его конце все сообщения об ошибках, выработанные во время проверок.

```

// Уведомление пользователя о том, что пропущено
if ((requiredFieldsErrorMessage) ||
(whatTheyWantErrorMessage) || (emailErrorMessage) ||
(areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
the information you've provided\r";

    // Построить остальную часть сообщения об ошибке
    alertMessage = alertMessage + requiredFieldsErrorMessage;
    alertMessage = alertMessage + whatTheyWantErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;
    ...
}
else
{
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```


После того, как мы добавим к общему заголовку все другие уведомления об ошибках, нам останется только показать итоговое сообщение с помощью функции `alert()`.

```
// Уведомление пользователя о том, что пропущено
if ((requiredFieldsErrorMessage) ||
    (whatTheyWantErrorMessage) || (emailErrorMessage) ||
    (areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
        the information you've provided\r";

    // Построить остальную часть сообщения об ошибке
    alertMessage = alertMessage + requiredFieldsErrorMessage;
    alertMessage = alertMessage + whatTheyWantErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;

    alert (alertMessage);
}
else
{
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Если же в ответах пользователя не было допущено никаких ошибок, вступает в действие оператор `else` и передает данные формы в сценарий CGI.

```
// Уведомление пользователя о том, что пропущено
if ((requiredFieldsErrorMessage) ||
    (whatTheyWantErrorMessage) || (emailErrorMessage) ||
    (areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
        the information you've provided\r";

    // Построить остальную часть сообщения об ошибке
    alertMessage = alertMessage + requiredFieldsErrorMessage;
    alertMessage = alertMessage + whatTheyWantErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;
```

```
// А теперь отобразить сообщение об ошибке
alert (alertMessage);
}
else
{
    document.forms['promotion'].submit();
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Описание сценария

Итак, чтобы добиться каких-то реальных результатов, нам пришлось заполнить несколько страниц кодом JavaScript! Зато теперь у нас есть сценарий, который специалисты по маркетингу наперебой расхваливают перед вашим руководителем. Рассмотрим его в целом и вспомним все, что он выполняет.

1. Первым делом были созданы переменные, хранящие значения, введенные пользователем в обязательные поля, а также переменные для хранения всех сообщений об ошибках, которые (сообщения) могут потребоваться.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
// Создать функцию errorCheck
function errorCheck()
{
    // Создать переменные для сценария
    var requiredFieldsErrorMessage = "";
    var firstNameEntered = "";
    var lastNameEntered = "";
    var emailEntered = "";
    var areaCodeEntered = "";
    var zipCodeEntered = "";
    var pullDownErrorMessage = "";
    var emailErrorMessage = "";
    var areaCodeErrorMessage = "";
    var zipCodeErrorMessage = "";
```

2. Затем была разработана часть функции, предназначенная для контроля всех упущений или ошибок в ответах и вывода сообщения о том, в чем ошибся пользователь.

Сначала значения ответов, которые введены в обязательных полях, присваиваются новым переменным.

```
// Обязательная проверка ошибок в полях
firstNameEntered =
document.forms['promotion'].elements['firstName'].value;
```

```
lastNameEntered =
document.forms['promotion'].elements['lastName'].value;
```

```
emailEntered =
document.forms['promotion'].elements['email'].value;
```

3. Потом применяется ряд операторов `if` для проверки того, есть ли в переменных данные, и если нет, то — для добавления сообщений об упущениях пользователя к переменной `requiredFieldsErrorMessage`.

```
if ((!firstNameEntered) || (!lastNameEntered) ||
    (!emailEntered))
{
    if (!firstNameEntered)
    {
        requiredFieldsErrorMessage = "- your first
name\r";
    }
    if (!lastNameEntered)
    {
        requiredFieldsErrorMessage =
requiredFieldsErrorMessage +
        "- your last name\r";
    }
    if (!emailEntered)
    {
        requiredFieldsErrorMessage =
requiredFieldsErrorMessage +
        "- your email address\r";
    }
}
```

4. Еще один оператор `if` устанавливает, выбрал ли пользователь из разворачивающегося меню необходимую опцию, отличную от той, что предусмотрена по умолчанию. Если другая опция не была выбрана, сценарий присваивает короткое сообщение переменной `pullDownErrorMessage`.

```
// Контроль ошибок выбора из разворачивающегося меню
if (document.forms['promotion']
.elements['whatTheyWant'].value == 'default')
{
```

```
pullDownErrorMessage = "- you didn't tell us what
you want to see in Stitch";
}
```

5. На следующем этапе выполнения функции `errorCheck()` используется ряд операторов `if`, подтверждающих наличие строки, введенной в поле формы для адреса электронной почты, и присутствие в ней символа `@` и точки.

```
// Контроль ошибок ввода адреса электронной почты
if (emailEntered)
{
    emailEntered =
document.forms['promotion'].elements['email'].value;

    emailValue = new String (emailEntered);
    emailHasAt = emailValue.indexOf("@");
    emailHasPeriod = emailValue.indexOf(".");

    if ((emailHasAt == -1) || (emailHasPeriod == -1))
    {
        emailErrorMessage = "-your email address\r";
    }
}
```

6. Если пользователь ввел телефонный номер, надо убедиться в том, что указан также код города. Поэтому опять применяется ряд операторов `if`, проверяющих значения, содержащиеся в обоих полях формы.

```
// Проверка кода города
areaCodeEntered =
document.forms['promotion'].elements['areaCode'].value;

phoneNumber1Entered =
document.forms['promotion'].elements['phoneNumber1'].value;
```

```
if ((!areaCodeEntered) && (phoneNumber1Entered))
{
    areaCodeErrorMessage = "-please enter your area
code\r";
}
```

7. Затем обеспечивается проверка того, содержит ли поле почтового индекса исключительно цифры. Выполняют задачу операторы `if` и два цикла `while`, которые выбирают отдельные символы в

строке, введенной в поле почтового индекса, и устанавливают, содержат ли они одни лишь цифры.

```
// Проверка почтового индекса
zipCodeEntered =
document.forms['promotion'].elements['zipCode'].value;
// Если посетитель ничего не ввел в поле почтового
// индекса, то нечего и проверять
if (zipCodeEntered)
{
    numberCounter = 0;
    zipLength = 5;
    zipCounter = 0;
    foundNumber = "";

    while (zipCounter < 5)
    {
        while (numberCounter < 10)
        {
            if (zipCodeEntered.substr(zipCounter, 1)
                == numberCounter)
            {
                foundNumber = "yes";
                numberCounter = 11;
            }
            else
            {
                foundNumber = "no";
            }
            numberCounter++;
        }
        if (foundNumber != "yes" )
        {
            zipCounter = 6;
        }
        else
        {
            zipCounter++;
            numberCounter = 0;
        }
    }
    if (foundNumber != "yes")
    {
        zipCodeErrorMessage = "-zip code has non-numbers
in it\r";
    }
}
```

8. И, наконец, с помощью операторов if выясняется, содержат ли значения различные переменные с сообщениями об ошибках. Если содержат, мы добавляем эти значения к переменной alertMessage. После проверки всей информации об ошибках содержание окончательного сообщения отображается на экране для просмотра пользователем. Если сообщения об ошибках не были обнаружены, выполняется вызов метода submit() формы.

```
// Уведомление пользователя о том, что пропущено
if ((requiredFieldsErrorMessage) ||
    (pullDownErrorMessage) || (emailErrorMessage) ||
    (areaCodeErrorMessage) || (zipCodeErrorMessage))
{
    alertMessage = "Oops! There's a little trouble with
the information you've provided\r";
    // Построить остальную часть сообщения об ошибке
    alertMessage = alertMessage +
requiredFieldsErrorMessage;
    alertMessage = alertMessage + pullDownErrorMessage;
    alertMessage = alertMessage + emailErrorMessage;
    alertMessage = alertMessage + areaCodeErrorMessage;
    alertMessage = alertMessage + zipCodeErrorMessage;

    // А теперь отобразить сообщение об ошибке
    alert ( alertMessage )
}
else
// В форме нет ошибок
{
    // Отправить данные этой формы в ее сценарий CGI
    // document.forms['promotion'].submit()
}
</SCRIPT>
```

В данной главе мы ознакомились с несколькими новыми понятиями, а также более подробно рассмотрели те, что уже описывались ранее. Вот они:

- иерархия JavaScript и средства доступа к ее объектам и манипуляции ими (в частности, речь шла об объектах формы);
- операторы if и циклы while;
- применение функции alert() как средства доставки информации пользователю;
- логические операторы ||, && и !;

- ввод жесткого символа перевода строки в виде символа `\r`;
- объект `STRING` и некоторые его свойства.

◆ Резюме

Глава 5 содержит очень значимую часть материала этой книги. Если вы как следует освоите описанные здесь свойства и структуры, то сделаете серьезный шаг в освоении JavaScript. Понимание и применение операторов `if` и циклов `while` является основой разработки сложных конструкций данного языка. По мере его дальнейшего развития это становится все более очевидным.

◆ Дополнительное задание

Попробуйте реализовать идеи, которые мы вам предлагаем, если хотите углубить свое понимание вопросов, рассмотренных в этой главе.

1. Попытайтесь выполнить проверку таких типов полей формы, которые не были описаны, например — переключателей.
2. Осуществите проверку наличия в текстовых полях конкретных ответов. Подобный код может применяться для оценки выполнения экзаменационного задания в оперативном режиме.



В этой главе

- Общее представление об объекте `WINDOW`
- Проект I: Создание, заполнение и закрытие окон
- Проект II: Обмен информацией между кадрами в коде JavaScript
- Резюме
- Дополнительное задание

Приобретенные знания уже позволили нам создать несколько интересных сценариев. В последних главах мы много внимания уделяли формам, а теперь настало время изучить кое-что новое. Если исключить начальную страницу, то весь Web-узел журнала *Stitch* создан с использованием кадров. В данной главе мы поработаем со вторичными страницами *Stitch* и рассмотрим, как в языке JavaScript применяются кадры и окна.

Однако прежде чем перейти к конкретным проектам, следует побольше узнать о предназначении этих средств и о том, какое место они занимают в иерархии объектов языка. Работа с кадрами и многочисленными окнами нередко оказывается очень сложной. Тем не менее, мы надеемся, что к концу главы у вас сложится четкое представление о том, как создать с их помощью и в сочетании с JavaScript необходимые вам конструкции.

◆ Общее представление об объекте WINDOW

Объект `WINDOW` в иерархии JavaScript занимает самое верхнее положение. Объект `DOCUMENT` и все другие, расположенные под ним, являются потомками `WINDOW`. Проекты, которые мы выполняли до сих пор, помещались в одном окне, поэтому нам никогда не приходилось иметь дело с `WINDOW` или его свойствами. Однако так бывает не всегда. Во многих обстоятельствах выгодно создать новое окно или обратиться к объектам, содержащимся в другом окне.

Поскольку окно является объектом, мы имеем возможность получить доступ к содержащимся в нем свойствам и объектам, а также к его собственным свойствам. С помощью JavaScript мы в состоянии изменить размер окна, настроить его панели инструментов, обратиться к его массиву `HISTORY` и сделать многое другое. Приступим к нашему первому проекту и посмотрим, как это все действует на практике.

◆ Проект I: Создание, заполнение и закрытие окон

На Web-узле *Stitch* имеется страница, на которой перечислены компании, разместившие свои рекламные объявления в последнем номере журнала. *Stitch* — это издание о моде, и среди его рекламодателей есть крупные компании. Их реклама порой сама определяет тенденции в развитии моды. Поскольку у журнала нет отбоя от

читателей, желающих непосредственно обратиться к компаниям-рекламодателям, редакция убедилась в том, что идея размещения рекламы на Web-узле себя оправдала. Поэтому решено расширить функциональные возможности данной страницы, превратив название каждой компании в ссылку, по которой пользователь может перейти на страницу, содержащую контактную информацию этой компании, а также другие ссылки — на ее Web-узел и на копию текущего рекламного объявления.

Но кто сумеет воплотить решение в жизнь? Конечно, все взгляды обратились в вашу сторону. Что ж, вам нечего бояться. Последние усовершенствования, внесенные вами в проект узла, создали вам репутацию ведущего специалиста по JavaScript. После короткого совещания со своим руководителем вы пришли к выводу, что имеется превосходный способ решения поставленной задачи. Единственная проблема, которая вас смущает, связана с тем, что в каждом номере журнала (а он выходит раз в месяц) присутствуют не менее 20 рекламодателей. Добавление такого большого количества страниц к узлу приведет к переполнению сервера и заставит ежемесячно обновлять весьма значительную группу файлов. Поэтому предлагается следующий выход: массивы JavaScript на имеющейся странице со списком использовать для хранения всей информации, нужной для динамического создания вводимой в новое окно страницы каждой компании. Это позволит сэкономить место на сервере, обновлять каждый месяц только один файл и заложить прекрасную основу для реализации намеченного проекта.

Создание нашего сценария включает три основных этапа. Сначала надо определить и создать массивы для информации о каждой компании. Затем — образовать функцию, которая станет открывать новое окно и заполнять его информационным содержанием. И, наконец, — вставить соответствующие обработчики событий в код HTML, чтобы сценарий начал работать.

Создание и определение массивов

Первым делом требуется создать массивы для хранения информации о компании. Эта информация состоит из девяти элементов данных: название компании, ее адрес, город, штат, почтовый индекс, телефонный номер, номер факса, адрес Web-узла (если он есть) и, наконец, URL страницы, на которой размещено рекламное объявление. В главе 3 мы образовывали по два массива для каждой группы: в одном хранилось имя страницы, а в другом — ее URL. Для хранения информации о компаниях в основном можно использовать ту же идею, только вместо двух массивов для группы мы будем применять

по одному для каждой компании. Хотя до сих пор мы присваивали значение и ссылались на любой элемент в массиве по его позиции, в данном проекте мы присвоим каждому элементу имя, по которому станем на него ссылаться.

Начнем разработку сценария с определения массивов для компаний. Поскольку эта страница должна быть добавлена только ко времени выхода следующего номера журнала, нам неизвестно, какие фирмы разместят свои рекламные объявления. Поэтому пока, чтобы не терять времени, просто создадим массивы для двух фиктивных компаний.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Массивы для информации о компаниях
// Определить массивы
Company1_info = new Array(9);
Company2_info = new Array(9);
...
// Здесь соккрытие кода прекращается -->
</SCRIPT>
```

Итак, мы определили два массива — один для Company1, другой для Company2 — и предусмотрели в обоих по девять элементов. Наш следующий шаг — размещение в каждом массиве контактной информации компании. Расположим информацию о Company1 в ее массиве.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
// Массивы для информации о компаниях
// Определить массивы
Company1_info = new Array(9);
Company2_info = new Array(9);

// Заполнить массив Company1
Company1_info['name'] = 'Company 1';
Company1_info['street'] = '1235 company way #1';
Company1_info['city'] = 'Smallsville';
Company1_info['state'] = 'California';
Company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';
// Здесь соккрытие кода прекращается -->
</SCRIPT>
```

При заполнении массивов таким большим количеством элементов, как в приведенном коде, часто гораздо проще присвоить каждому элементу имя, а не использовать его числовую позицию. Если вы применяете для ссылок только номер позиции, то можете легко забыть, где находится конкретный элемент.

Теперь массив `Company1_info` содержит всю информацию, необходимую для создания страницы `Company1`. Заполним массив для `Company2`.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
// Массивы для информации о компаниях
// Определить массивы
Company1_info = new Array(9);
Company2_info = new Array(9);

// Заполнить массив Company1
Company1_info['name'] = 'Company 1';
Company1_info['street'] = '1235 company way #1';
Company1_info['city'] = 'Smallsville';
Company1_info['state'] = 'California';
Company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';

// Заполнить массив Company2
Company2_info['name'] = 'Company 2';
Company2_info['street'] = '4321 company way #2';
Company2_info['city'] = 'Bigsville';
Company2_info['state'] = 'California';
Company2_info['zip'] = '91235';
Company2_info['phone'] = '818-555-2121';
Company2_info['fax'] = '818-555-2122';
Company2_info['website'] = 'http://www.company2.com';
Company2_info['ads'] = 'company2_ads.html';
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

У нас уже есть вся информация, которую нам нужно было надежно сохранить в двух массивах (если вам требуется освежить в памяти основные сведения о массивах, обратитесь к проекту II в главе 3). Пора переходить к следующему этапу.

Создание функции

Функция для этого сценария должна выполнять несколько действий. Прежде всего — определять, какая компания была выбрана, и получать информацию о ней из соответствующего массива. Затем она должна открывать новое окно и, наконец, заполнять данное окно добытой информацией. Сначала наметим каркас функции.

```
// Функция создания страницы
function PageCreator(selection)
{
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Таким образом, мы создали функцию `PageCreator()` и установили переменную `selection`, которая станет принимать значение, передаваемое в функцию из обработчика событий. Этим значением будет название компании, указанной пользователем. Следующий этап состоит в определении того, какая именно компания указана, и выборке информации о ней.

```
// Функция создания страницы
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Теперь мы знаем, что значением переменной `selection` должно служить название компании, и применили такие названия в именах массивов, хранящих информацию об этих фирмах. В приведенном выше коде с помощью метода `eval()` осуществляются соединения значения переменной со строкой и использование полученного значения для вызова информации из соответствующей позиции массива для компании. После выборки информации она включается

в переменную, которую мы будем использовать в дальнейшем для вывода данного информационного наполнения во всплывающее окно. Процесс повторяется для каждой из девяти установленных позиций.

Как только сведения о выбранной компании будут надежно записаны в переменных, следует открыть новое окно, предназначенное для размещения этой информации. Рассмотрим коротко синтаксис метода `open()` объекта `WINDOW`:

```
window.open(URL, windowName, windowFeatures)
```

В вызов `open()` надо включить три элемента: URL страницы, которую требуется поместить в новое окно; строку, содержащую имя, по которому можно будет ссылаться на данное окно, указывая его в качестве адресата `target` в теге `<A HREF>`; необязательный список стандартных настраиваемых свойств окна, таких, как ширина, высота или расположение различных панелей инструментов.

Окно не только способно нести имя, служащее адресатом, но и может быть связано с именем, применяемым для ссылки на объект `WINDOW` с использованием JavaScript. Для присвоения окну этого второго имени возможен следующий синтаксис (`Name` — имя, что должно быть присвоено окну):

```
Name = window.open(URL, windowName, windowFeatures)
```

Разместим в сценарии код, открывающий окно.

```
// Функция создания страницы
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");

    // Открыть новое окно
    infowin = window.open("blank.html", "Company_Info",
        "menubar=yes,width=250,height=200");
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В этом коде мы прежде всего присваиваем новому окну имя `infowin`. После чего сразу переходим к выполнению самого метода `open()`: в качестве URL страницы, которую мы хотим разместить в окне, указано `blank.html`. Мы намерены заполнять окно содержимым наших массивов динамически, поэтому HTML-страница является пустой, на что и указывает ее имя (`blank`). Следующий элемент кода — имя адресата, присваиваемое странице, — `Company_Info`. Мы не будем его использовать в нашем сценарии, но он является обязательным. Теперь очередь списка атрибутов, которые мы также хотим присвоить окну. Разместим в окне панель инструментов, содержащую такие меню, как `file`, `edit`, `view` и т.д., и назначим окну ширину в 250 пикселей и высоту в 200 пикселей.

Открыв новое окно, мы должны заполнить его информацией, хранящейся в переменных. Для этого будем использовать метод `document.write()`. Мы уже применяли его несколько раз в данной книге, но теперь он получит новое развитие. Если бы мы использовали стандартный синтаксис:

```
document.write('text to be written');
```

то могли бы просто перезаписать содержимое нашей страницы со списком компаний, а не выводить информацию в новое окно. Чтобы поместить сведения в новое окно, следует сообщить интерпретатору JavaScript, куда именно мы хотим выводить код HTML. Это можно выполнить, вставив имя окна перед словом `document` следующим образом:

```
infowin.document.write('text to be written');
```

Теперь, когда мы знаем, как записывать информацию в новое окно, введем код, выполняющий данное действие.

```
// Функция создания страницы
function PageCreator(selection)
{
    var company = eval(selection + "_info['name']");
    var street = eval(selection + "_info['street']");
    var city = eval(selection + "_info['city']");
    var state = eval(selection + "_info['state']");
    var zip = eval(selection + "_info['zip']");
    var phone = eval(selection + "_info['phone']");
    var fax = eval(selection + "_info['fax']");
    var website = eval(selection + "_info['website']");
    var ads = eval(selection + "_info['ads']");

    // Открыть новое окно
    infowin = window.open("blank.html", "Company_Info",
```



```

"menubar=yes,width=250,height=200");

// Записать информационное наполнение в новое окно
infowin.document.write
("<HTML><HEAD><TITLE>Company Information</TITLE>
</HEAD><BODY BGCOLOR='#FFFFFF'><CENTER>");

infowin.document.write
("<TABLE BORDER='0' CELSPACING='0'
CELLPADDING='0'><TR><TD><B>" + company + "</B>
</TD></TR>");

infowin.document.write
("<TR><TD>" + street + "</TD></TR>");

infowin.document.write
("<TR><TD>" + city + ", " + state + " " + zip +
"</TD></TR>");

infowin.document.write
("<TR><TD>" + "phone - " + phone + "</TD></TR>
<TR><TD>fax - " + fax + "</TD></TR>");

infowin.document.write
("<TR><TD><A HREF='" + website + "' TARGET= '_TOP' >" +
website + "</A></TD></TR>");

infowin.document.write
("<TR><TD ALIGN='CENTER'><BR><A HREF='" + ads +
" ' TARGET='Content'>View the Ad</A></TD></TR>
</TABLE>");

infowin.document.write
("<TABLE BORDER='0' CELSPACING='0'
CELLPADDING='0' WIDTH='249'><TR><TD WIDTH='249'
VALIGN='BOTTOM' ALIGN='RIGHT'><BR>
<A HREF='javascript:window.close()'><IMG
SRC='images/closer.gif' BORDER='0'></A></TD>
</TR>");

infowin.document.write
("<TR><TD WIDTH='249' BGCOLOR='#FFFF00'>&nbsp;  </TD>
</TR></TABLE>");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Рассмотрим, что происходит в приведенном блоке сценария. Может показаться, что здесь содержится очень большой объем информации, но на самом деле это не так. Для удобства чтения мы

использовали с целью вывода нового кода несколько вызовов метода `document.write()`. В первой строке мы создаем новый документ HTML и вставляем теги `HEAD` и `BODY`. В предыдущих сценариях в операторах `document.write()` применялись символы обратного слэша (`\`) для маскировки кавычек, которые надо было направить в вывод. В этом примере мы просто используем два типа кавычек. Двойные обозначают начало и окончание информационного наполнения, предназначенного для вывода на страницу, а внутри них используются одинарные — в том случае, когда действительно необходимо вывести их в коде. Оба способа — и с двумя видами кавычек, и с символами обратного слэша — работают одинаково хорошо, и предпочтение какого-либо из них — дело вкуса.

В остальных строках `document.write` для вывода в качестве информационного наполнения применяется сочетание образца таблицы и сведений, которые мы сохранили в переменных. Предпоследняя строка `document.write` содержит несколько непривычный код, поэтому рассмотрим ее подробнее. Здесь выводимое информационное наполнение является в общем-то довольно простым. Но кое-что необычное тут все-таки есть.

```
<A HREF='javascript:window.close()'>
```

Ссылку `javascript` в этом теге `HREF` можно рассматривать как обработчик событий `onClick`: когда пользователь на ней щелкает, выполняется код JavaScript, следующий за двоеточием. В данном случае команда позволяет пользователю закрыть новое окно после того, как он получит на странице всю необходимую информацию. Мы еще не изучали закрытие окон, поэтому обратимся к синтаксису метода `close()`:

```
window.close();
```

Эта строка по умолчанию закрывает окно, в котором находится упомянутая ссылка. Если бы в ваши намерения входило закрытие другого окна браузера, вы могли бы конкретно назвать его по имени следующим образом:

```
infowin.close();
```

Строки `document.write()` составляют последнюю часть создаваемой функции. Единственное, что осталось сделать в данном сценарии, — вставить обработчики событий в код HTML.

Вставка обработчиков событий

Как и при использовании чуть выше метода `close()`, разместим обработчики событий в информационном наполнении всплывающего окна прямо в тегах `HREF` кода `HTML`. Как только пользователь щелкнет на ссылке компании, будет вызвана функция `PageCreator()`, которая передаст в код название этой компании (рис. 5.1). Приводимый код `HTML` содержит ссылки с обработчиками, вставленными в теги `HREF`:

```
<TR>  
    <TD>  
        &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
    </TD>  
    <TD>  
        <FONT FACE="Helvetica, Arial" SIZE="-1">  
            <A HREF="javascript:PageCreator('Company1')">  
                Company 1</A></FONT>  
    </TD>  
    <TD>  
        <FONT FACE="Helvetica, Arial" SIZE="-1">  
            <A HREF="javascript:PageCreator('Company2')">  
                Company 2</A></FONT>  
    </TD>  
</TR>
```

На этом подготовка проекта практически завершена. Перечислим все, что мы сделали и проанализируем сценарий в целом.

Описание сценария

1. В первую очередь мы определили и заполнили массивы, хранящие контактную информацию по каждой из компаний. Вместо применения для хранения сведений числового обозначения позиции массива мы присвоили этой позиции имя. И, таким образом, получили возможность указывать не имя, а число.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Браузеры ранних версий игнорируют следующий код
// Массивы для информации о компаниях
// Определить массивы
Company1_info = new Array(9);
Company2_info = new Array(9);

// Заполнить массив Company1
Company1_info['name'] = 'Company 1';
Company1_info['street'] = '1235 company way #1';
Company1_info['city'] = 'Smallsville';
```

```
Company1_info['state'] = 'California';
Company1_info['zip'] = '91367';
Company1_info['phone'] = '818-555-1212';
Company1_info['fax'] = '818-555-1213';
Company1_info['website'] = 'http://www.company1.com';
Company1_info['ads'] = 'company1_ads.html';
```

```
// Заполнить массив Company2
Company2_info['name'] = 'Company 2';
Company2_info['street'] = '4321 company way #2';
Company2_info['city'] = 'Bigsville';
Company2_info['state'] = 'California';
Company2_info['zip'] = '91235';
Company2_info['phone'] = '818-555-2121';
Company2_info['fax'] = '818-555-2122';
Company2_info['website'] = 'http://www.company2.com';
Company2_info['ads'] = 'company2 ads.html';
```

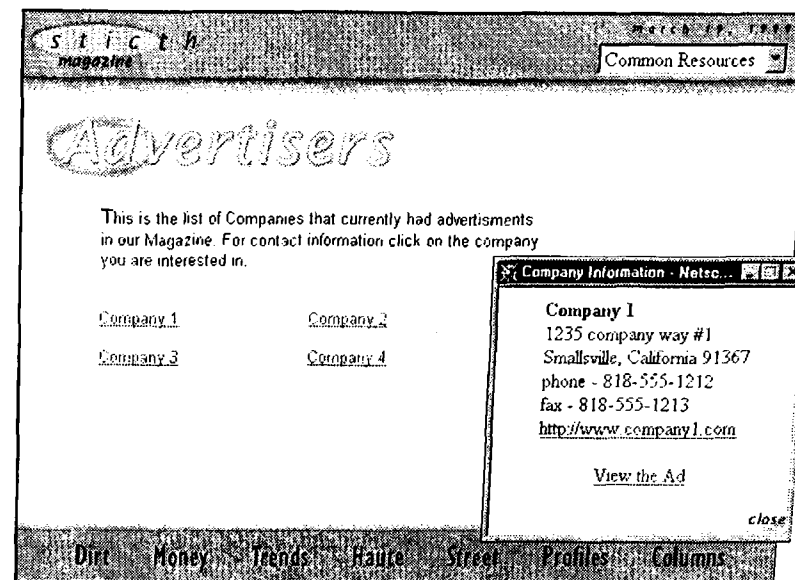


Рис. 5.1. Страница со списком рекламодателей

2. Следующий этап заключался в создании функции `PageCreator()`. Мы установили некоторые переменные для хранения сведений о компании и использовали значение, переданное в функцию из

обработчиков событий, для вызова информации из соответствующего массива.

```
// Функция создания страницы
function PageCreator(selection)
```

```
var company = eval(selection + "_info['name']");
var street = eval(selection + "_info['street']");
var city = eval(selection + "_info['city']");
var state = eval(selection + "_info['state']");
var zip = eval(selection + "_info['zip']");
var phone = eval(selection + "_info['phone']");
var fax = eval(selection + "_info['fax']");
var website = eval(selection + "_info['website']");
var ads = eval(selection + "_info['ads']");
```

3. После размещения кода для выборки информации из массива мы использовали метод `open()` объекта `WINDOW` для создания нового окна, содержащего сведения о компании.

```
infowin=window.open("blank.html" ,"Company_Info",
"menubar=yes,width=250,height=200");
```

4. Создав новое окно, мы применили метод `write()` объекта `DOCUMENT` для вывода информационного наполнения.

```
infowin.document.write
("<HTML><HEAD><TITLE>Company Information</TITLE>
</HEAD><BODY BGCOLOR=\"#FFFFFF\"><CENTER>");
```

```
infolwin.document.write
("<TABLE BORDER='0' CELSPACING='0' CELLPADDING='0'>
  <TR><TD><B>" + company + "</B></TD></TR>");
```

```
infowin.document.write
("<TR><TD>" + street + "</TD></TR>");
```

```
infolwin.document.write
("<TR><TD>" + city + ", " + state + " " + zip +
 "</TD></TR>");
```

```
infowin.document.write
("<TR><TD>" + "phone - " + phone + "</TD></TR>
<TR><TD>fax - " + fax + "</TD></TR>");
```

```
infolwin.document.write
("<TR><TD><A HREF='" + website + "' TARGET='_TOP'>"
+ website + "</A></TD></TR>");
```

```
infolwin.document.write
("<TR><TD ALIGN='CENTER'><BR><A HREF= '" + ads +
" ' TARGET='Content'>View the Ad</A></TD></TR>
</TABLE>");
```

```

inflowin.document.write
("<TABLE BORDER='0' CELLSPACING='0'
  CELLPADDING='0' WIDTH='249'><TR><TD WIDTH='249'
  VALIGN='BOTTOM' ALIGN='RIGHT'><BR>
  <A HREF='javascript:window.close()'><IMG
  SRC='images/closer.gif' BORDER='0'></A>
  </TD></TR>");

```

```
infowin.document.write  
("<TR><TD WIDTH='249' BGCOLOR='#FFFF00'>&nbsp;     
</TD></TR></TABLE>");
```

```
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

- Последний этап представлял собой вставку обработчиков событий в теги `<A HREF>` каждой компании.

[illegible]

При написании этого сценария мы использовали некоторые новые средства языка JavaScript. Назовем их:

- объект WINDOW: мы узнали, как открывать, закрывать и заполнять окна;
- заполнение массива и доступ к его содержанию по числовым позициям, занятым информацией, и по меткам, определяемым пользователем.

Отныне мы располагаем эффективным способом выполнения задания, которое иначе могло бы превратиться в довольно утомительный

труд. Одним из преимуществ языка JavaScript как раз и является то, что он позволяет разрабатывать решения, не только уменьшающие объем работы, но и использующие при этом всю мощь данной результативной технологии.

Однако перевести дыхание вам не суждено: у босса созрела идея нового проекта. А что если вам придется разрабатывать его самостоятельно, без нашей помощи?

◆ Проект II: Обмен информацией между кадрами в коде JavaScript

Впрочем, можно было не сомневаться, что руководитель потребует от вас решить очередную проблему, связанную с Web-узлом. Дело в том, что в настоящее время в нижнем кадре узла, содержащем главную навигационную панель, графическое изображение для раздела, в котором вы находитесь, является подсвеченным (рис. 5.2). Еще один из программистов написал код JavaScript, обеспечивающий смену изображений при выполнении на них щелчка. Но ваш руководитель хочет, чтобы именно вы занялись новой проблемой.

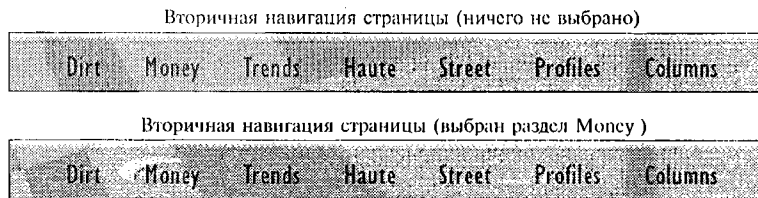


Рис. 5.2. Нижняя навигационная панель вторичной страницы Web-узла журнала Stitch

Если пользователь щелкает на кнопках Forward или Back браузера, вместо того, чтобы щелкнуть на навигационных изображениях, графика не обновляется. Поэтому требуется создать сценарий, в котором кадр с информационным наполнением обменивался бы содержанием с навигационным кадром. Это нужно для того, чтобы после загрузки новой страницы в кадр с информационным наполнением он сообщал навигационному кадру, какое графическое изображение должно быть подсвечено. Для решения данной задачи необходимо ознакомиться с тем, как в языке JavaScript осуществляется работа с кадрами.

Определение иерархии кадров

Кадр можно считать разновидностью объекта WINDOW. Как и окно, он содержит объект DOCUMENT, а также все производные объекты и свойства, которые могут быть созданы в HTML. Однако в иерархии JavaScript объект FRAME расположен ниже объекта WINDOW, являясь в действительности его свойством.

Каждый кадр в окне способен содержать отдельный URL. Такая особенность может быть очень полезной для узла Stitch, в котором применяются три кадра: один содержит информационное наполнение, а два других — навигационные ссылки. Это позволяет выполнять в кадре прокрутку информационного наполнения, оставляя на месте навигационные ссылки.

При загрузке в браузер документа HTML с тегом FRAMESET интерпретатор JavaScript образует массив frames, содержащий все созданные объекты FRAME. Это весьма напоминает способ обработки интерпретатором JavaScript обнаруженных на странице изображений. Способ ссылки на эти объекты также аналогичен. Рассмотрим тег FRAMESET для узла Stitch и определим, как можно ссылаться на различные созданные объекты FRAME.

```
<FRAMESET ROWS="65,*,50" FRAMEBORDER="0" BORDER="0"
FRAMESPACING="no">
```

```
<FRAME NAME="Top_Nav_Bar" SRC="top_nav.html"
MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no"
FRAMEBORDER="no" NORESIZE>
```

```
<FRAME NAME="Content" SRC="ad_index.html"
MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="auto"
FRAMEBORDER="no" NORESIZE>
```

```
<FRAME NAME="Bottom_Nav_Bar" SRC="bottom_nav.html"
MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="no"
FRAMEBORDER="no" NORESIZE>
```

```
</FRAMESET>
```

Перед вами код HTML, создающий документ FRAMESET. В теге FRAMESET описаны три кадра: Top_Nav_Bar, содержащий общие навигационные ссылки и главный логотип; Content, который включает в себя все информационное наполнение разделов узла; и Bottom_Nav_Bar, содержащий навигационные ссылки для главных разделов узла. Иерархия объектов тега FRAMESET показана на рис. 5.3.

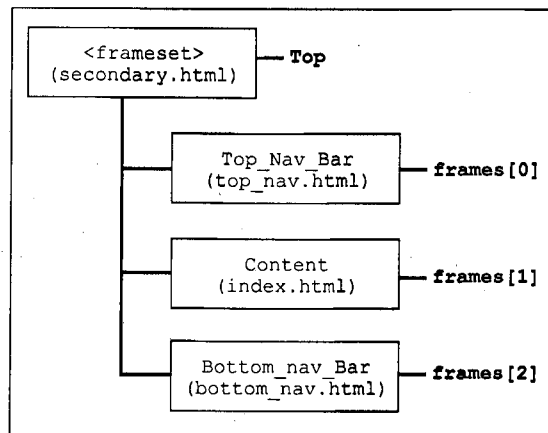


Рис. 5.3. Иерархия объектов FRAME в нашем примере

По мере загрузки страницы с тегом `FRAMESET` кадры помещаются в массив `frames` в том порядке, в каком их считывает интерпретатор. Кадр `Top_Nav_Bar` становится элементом массива `frames[0]`, кадр `Content` — `frames[1]` и `Bottom_Nav_Bar` — `frames[2]`. Для доступа к объектам FRAME можно использовать эти номера позиций, обращаясь к массиву `frames`. Например, чтобы обратиться к URL кадра `Content`, допустимо применить следующий синтаксис:

```
frames[1].src
```

Однако такую конструкцию правомерно использовать только при обращении к информации со страницы, содержащей данный тег `FRAMESET`. Если бы мы находились в одном из кадров другого набора, нам пришлось бы указать интерпретатору JavaScript, чтобы он сначала вернулся на верхний уровень окна, а затем перешел к кадру `Content`. Достигается это одним из двух способов. Во-первых, можно использовать строку

```
top.frames[1].src
```

Она сообщает интерпретатору, что нужно выйти на самый верхний уровень иерархии, а затем обратиться ко второму кадру в массиве `frames`. Во-вторых, вместо ключевого слова `top` допустимо указать `parent` для получения аналогичного эффекта:

```
parent.frames[1].src
```

Разница заключается в том, что ключевое слово `top` направляет интерпретатор к вершине иерархии, а `parent` перемещает указатель на один уровень выше — к объекту, являющемуся родительским по отношению к кадру, из которого его вызывают. Иногда применяются вложенные кадры. То есть, вызываемый в кадре документ HTML содержит еще один тег `FRAMESET`, а не просто код HTML. В результате происходит размещение в иерархии дополнительного уровня, и для перехода к вершине иерархии требуется вызвать свойство `parent` несколько раз:

```
parent.parent.frames[1].src
```

Применение вложенных кадров в языке JavaScript может оказаться довольно сложным. К нашему счастью, в узле `Stitch` их нет.

Пора переходить к проекту. Он состоит из двух разделов. В первую очередь надо создать в кадре `Bottom_Nav_Bar` функцию, сменяющую подсвеченное графическое изображение при изменении страницы `Content`. Затем нужно вставить обработчики событий во все страницы, которые заполняют кадр для информационного наполнения. Обработчики будут вызывать функцию, созданную в первой части проекта.

Создание функции для проверки того, в каком разделе мы находимся

Прежде чем перейти к описанию новой функции, посмотрим, нельзя ли использовать в нашем сценарии то, что уже существует. У нее два назначения. Во-первых, при ее вызове после щелчка пользователя на одном из навигационных изображений она проходит по всем навигационным изображениям и переводит их в неподсвеченное состояние. Далее она определяет, на каком графическом изображении щелкнул пользователь, и переводит его в подсвеченное состояние. Рассмотрим саму функцию, чтобы понять, как все это происходит.

```
function Highlighter(current)
{
    var test = 1;
    while (test < 8)
    {
        document.images[test].src = eval("Pic" +
            test + "Off.src");

        ++test;
    }
    document.images[current].src = eval(current
```

```

+ "On.src");
able = current;
}

```

В первой строке переменная `test` устанавливается в значение 1 (оно должно применяться в проверке условия следующего цикла `while`). За ней идет строка, служащая началом цикла `while`: пока значение `test` не превышает 8, цикл продолжает выполняться. Поскольку при входе в функцию значение `test` установлено в 1, цикл будет выполнен семь раз — по одному на каждое навигационное изображение. Мы начинаем со значения `test`, равного 1, а не 0, поскольку первую позицию в массиве изображений занимает фоновое изображение для этого кадра и его надо пропустить, переводя другие изображения в неподсвеченное состояние.

Внутри цикла мы устанавливаем в качестве источника изображения, находящегося в позиции, совпадающей со значением `test`, вариант графики с суффиксом `off`. По окончании цикла `while` все навигационные объекты будут переведены в предусмотренное по умолчанию неподсвеченное состояние.

Теперь можно сделать подсвеченным то изображение, на котором щелкнул пользователь. В следующей строке функции это выполняется путем установки в качестве источника изображения, соответствующего значению переменной `current`, версии изображения, над которой находится курсор.

В последней строке функции значением переменной `able` становится имя категории, на которой был совершен щелчок. Данное имя содержится в переменной `current`. Переменная `able` используется при проверке условия в функции `off()` — в одной из двух функций, выполняющих смену изображений на странице.

```

function off(pic)
{
    if (document.images)
    {
        if (pic != able)
        {
            document.images[pic].src= eval(pic + "Off.src");
        }
    }
}

```

Поэтому при удалении курсора с графики для раздела, в котором в настоящее время находится пользователь, данное изображение не переходит в неподсвеченное состояние. Эта переменная будет также применяться в нашей функции.

Теперь, четко представляя себе, что в настоящее время происходит на странице, мы можем перейти к построению новой функции. Ее разработка будет намного проще, по сравнению с написанием существующей функции. Создаваемая функция должна активизироваться при каждой загрузке новой HTML-страницы в кадр `Content`. Но нам нужно, чтобы она сменяла изображение только в том случае, если страница загружена не из того раздела, из которого загружалась предыдущая. Для этого надо передать в функцию при ее вызове определенное значение. Последним должно быть имя раздела, показанного в данный момент на странице. Затем необходимо сверить переданное значение со значением переменной `able`, содержащей имя раздела, на котором в последний раз щелкнул пользователь. Остальную часть этой функции нужно выполнить только при условии, что указанные значения являются различными, иными словами — при смене раздела.

Запишем начальный код функции.

```

function SectionChecker(content)
{
    if (content != able)
    {
        ...
    }
}

```

Итак, начав разработку, мы ввели оператор `if`, выполняющий сверку значения переменной `content` с переменной `able`. После этого осталось сделать единственное дополнение: вставить код, выполняемый, если данные значения являются разными.

```

function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}

```

Поскольку существующая функция уже содержит функцию `Highlighter()`, переводящую в состояние `off` все семь графических изображений, а затем — в состояние `on` одно изображение, имя которого передано функции, нам достаточно вызвать данную функцию и передать ей значение, отправленное со страницы с информационным наполнением.

В результате, если страница, загружаемая в кадр `Content`, передаст в нашу функцию значение `dirty`, то последняя вызовет функцию `Highlighter()` и передаст ей то же значение, по существу,

переводящее в неподсвеченное состояние все навигационные изображения, кроме dirt. Теперь надо только вставить обработчики событий во все страницы, которые станут поступать в кадр для информационного наполнения.

Вставка обработчика событий

До сих пор мы встречались с обработчиками событий, реагирующими на какие-то действия пользователя: перемещение курсора над изображением, щелчки кнопкой или изменение некоторых полей формы. Однако для данного проекта нам требуется обработчик событий, который будет активизироваться при загрузке HTML-страницы браузером. Нам повезло: именно это действие выполняет обработчик onLoad. Достаточно лишь поместить его в тег <BODY> документа HTML. Введем его в одну из страниц информационного наполнения для раздела Dirt.

```
<BODY onLoad="...">
```

Теперь вставим в него вызов функции SectionChecker(). Помните, что SectionChecker() находится в другом кадре, поэтому при ее вызове надо указать к ней путь.

```
<BODY onLoad="top.frames[2].SectionChecker('dirt')">
```

Хотите верьте, хотите нет, но работа почти закончена. Нужно лишь поместить этот обработчик во все HTML-страницы, предназначенные для кадра Content. Единственным отличающимся в них элементом явится значение, передаваемое в функцию SectionChecker(). А именно — имя раздела, в котором находится страница.

Впрочем, сделано еще не все. Один из программистов компании проверил новое дополнение к узлу и обнаружил, что в работе сценария может произойти некоторое нарушение. Вполне вероятно активизация обработчика onLoad до начала загрузки кадра Bottom_Nav_Bar, содержащего соответствующую функцию. В этом случае возникнет ошибка JavaScript.

Для предотвращения подобной ситуации надо обеспечить загрузку кадра Bottom_Nav_Bar перед вызовом функции SectionChecker() из кадра Content. Поэтому нужно добавить какой-то код к страницам FRAMESET, Bottom_Nav_Bar и ко всем страницам с информационным наполнением. Сначала вставим требуемый код в FRAMESET. Мы знаем, что если происходит загрузка кадра Content или Bottom_Nav_Bar, то предварительно должна быть загружена сама FRAMESET. Поэтому для выполнения проверки следует использовать переменную, находящуюся на странице FRAMESET.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
var capable = 'no';
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В приведенном коде мы всего лишь создаем переменную с именем capable и присваиваем ей по умолчанию значение no. Теперь эта переменная будет образовываться при загрузке страницы FRAMESET. Далее разместим необходимый код в кадре Bottom_Nav_Bar.

Нам надо вставить строку программы, которая станет вызывать переменную capable со страницы FRAMESET и присваивать ей значение yes. Такой код можно расположить сразу после функции SectionChecker().

```
function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}
top.capable = 'yes';
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Как и в обработчике onLoad, мы должны сообщить браузеру, чтобы он взял переменную на соответствующем уровне иерархии. При своей загрузке кадр Bottom_Nav_Bar изменит значение переменной capable на yes.

Остальной код следует поместить на страницы с информационным наполнением. Внесем коррективы в ту же самую страницу раздела Dirt, на которой мы уже разместили обработчик onLoad. Прежде всего нужно изменить этот обработчик.

```
<BODY BGCOLOR="White" onLoad="Checker()">
```

Вместо непосредственного вызова функции SectionChecker(), как мы это делали до сих пор, требуется создать новую функцию на самой странице с информационным наполнением и уже в ней активизировать SectionChecker(). В обработчике событий onLoad мы теперь будем вызывать эту новую функцию Checker(). После ее создания вся работа будет закончена.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
function Checker()
```

```

{
    if (top.capable='yes')
    {
        top.frames[2].SectionChecker('dirt');
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Данная функция очень проста. При обращении к ней первым делом происходит проверка значения переменной `capable` со страницы FRAMESET с помощью оператора `if`. Если кадр `Bottom_Nav_Bar` был загружен, она изменит значение `capable` на `yes`, чтобы мы могли вызвать `SectionChecker()`. Однако если `Bottom_Nav_Bar` еще не загружен, ничего не произойдет.

И на этом все. Нужно только вставить обработчик `onLoad` и функцию `Checker()` в другие страницы, которые могут быть загружены в кадр `Content`. Затем можно использовать кнопки `Back` и `Forward` для перехода к любому информационному наполнению, всегда зная при этом, в каком именно разделе вы находитесь.

Описание сценария

Рассмотрим все этапы создания нашего сценария.

1. Сначала мы включили в новый проект функциональные средства имеющегося сценария.

```

function Highlighter(current)
{
    var test = 1;
    while (test < 8)
    {
        document.images[test].src = eval('Pic"
            + test + "Off.src");

        ++test;
    }
    document.images[current].src = eval(current + "On.src");

    able = current;
}

```

2. Затем мы добавили функцию `SectionChecker()` к странице, находящейся в кадре `Bottom_Nav_Bar`. Она вызывает функцию `Highlighter()` и передает ей имя раздела, к которому принадлежит страница, загруженная в кадр для информационного наполнения.

```

function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}

```

3. После этого мы вставили обработчик событий `onLoad` во все HTML-страницы, способные появиться в кадре `Content`. Внутри обработчика мы поместили вызов функции `SectionChecker()` и передали ей имя раздела, к которому принадлежит страница.

```
<BODY onLoad="top.frames[2].SectionChecker('dirt')">
```

Наш сценарий стал работоспособным сразу после вставки обработчиков событий. Однако, чтобы сделать его более надежным, мы решили добавить последнюю составляющую, которая позволяет предотвратить появление ошибок в том случае, если кадр `Bottom_Nav_Bar` не был загружен до вызова функции `SectionChecker()`.

1. Первым делом мы добавили к HTML-странице переменную, содержащую тег `FRAMESET`. Она предназначена для проверки того, был ли загружен кадр `Bottom_Nav_Bar`.

```

<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
var capable = 'no';
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

2. Затем была введена дополнительная строка кода в HTML-страницу в кадр `Bottom_Nav_Bar`. Эта строка изменяет значение переменной на странице `FRAMESET`, сообщая, что нижняя навигационная страница уже загружена браузером.

```

function SectionChecker(content)
{
    if (content != able)
    {
        Highlighter(content);
    }
}
top.capable = 'yes';
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```


3. Наконец, мы добавили функцию ко всему коду HTML, который может проявляться в кадре Content, и изменили обработчик onLoad так, чтобы он вызывал эту новую функцию вместо непосредственного вызова функции SectionChecker().

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
function Checker()
{
    if (top.capable='yes')
    {
        top.frames[2].SectionChecker('dirt');
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
</HEAD>
</HEAD>
<BODY BGCOLOR="White" onLoad="Checker()">
```

◆ Резюме

Применение окон и кадров позволяет в значительной степени расширить функциональные горизонты Web-узла. При их использовании в сочетании с JavaScript ваши профессиональные возможности становятся буквально безграничными. В этой главе мы коснулись лишь незначительной части того, что может быть сделано с помощью данных средств.

Овладеть техникой работы с кадрами и разобраться в том, как действует иерархия этих объектов, нелегко. Но если вы освоите базовые положения, то сможете приступить к созданию исключительно широкого диапазона новых приложений.

◆ Дополнительное задание

1. Используйте методы, описанные в первом проекте главы 5, для создания Web-узла с вложенными кадрами, в котором применяются два набора навигационных изображений: для главной и для вторичной навигации.
2. Создайте Web-узел на основе вложенных кадров, в которых можно скрыть навигационную панель, сменив свойство location структуры, содержащей внутренние кадры.

6 Развитые средства JavaScript

В этой главе

- Проект I: Создание подменю для вторичных страниц Stitch
- Резюме
- Дополнительное задание

Теперь, после изучения основ, настало время перейти к созданию сложных разработок, которые непременно вызовут восторженные отзывы клиентов и коллег.

◆ Проект I: Создание подменю для вторичных страниц Stitch

Проведенные маркетинговые исследования показывают, что посетители многих Web-узлов часто не могут найти информацию, которая их интересует. В то же время большинство из них отправляются в Web именно за конкретной информацией. Вы скажете: “Ну кто читает отчеты об этих исследованиях!” Конечно, ваш руководитель. Именно поэтому он хочет предложить вам реализовать некоторые новые принципы построения средств навигации на начальной странице журнала Stitch. И вот вы вместе с другими программистами собрались, чтобы устроить “мозговой штурм” проблемы. Когда наступает ваша очередь высказать свежую идею, вы предлагаете применить систему графических подменю, с помощью которой пользователи смогут узнать, что находится в разделе, до того, как фактически в него перейдут. Не успеваете вы закончить свою мысль, как глаза вашего руководителя загораются... Догадываетесь, каким будет следующий проект? Точно: новая начальная страница Stitch с системой подменю, усовершенствованной с помощью JavaScript.

Постановка задачи

Поскольку проект довольно сложный, нужно точно описать, чего мы хотим добиться, прежде чем приступить к созданию кода. Обычно хочется пропустить этот этап и перейти сразу к работе, но постарайтесь себя сдерживать. Тщательно обдумав проект до начала его реализации, вы непременно сэкономите время и получите лучший конечный результат.

На рис. 6.1 вы видите имеющуюся начальную страницу Stitch. Дизайнеры хорошо над ней потрудились, и она вполне готова для подключения системы подменю. Сейчас в верхней ее части находятся логотип и слово Spring (весна), обозначающее сезон моды, а под ними справа размещено краткое меню общих ресурсов. Главные навигационные категории передвинуты влево, и в середине страницы расположена небольшая графическая реклама. И наконец, в правой нижней части страницы мы видим изображение женщины, сидящей в лодке.



Рис. 6.1. Текущая начальная страница Stitch

Надо сделать так, чтобы пользователь мог провести курсор над одной из главных опций навигации и вызвать появление другого всплывающего меню, которое заменило бы графическую рекламу. Например, при установке курсора над словом Haute (там, где находится реклама) должен появиться список некоторых статей раздела Haute. Затем желательно, чтобы посетитель мог перейти непосредственно к любой статье с помощью щелчка на одном из пунктов подменю, минуя главную страницу раздела.

Новое меню должно возникать каждый раз, когда курсор перемещается над пунктом главного меню узла. Так, когда пользователь устанавливает курсор над пунктом Street, показывается меню со

списком статей раздела Street. После установки курсора над пунктом Profiles появляется, соответственно, список статей этого раздела.

Каждый раз, когда курсор переходит на другое название раздела, вторичное меню данного раздела заменяет то, что находилось в центре страницы. Если курсор не находится ни над одним из главных навигационных изображений, графическая реклама возвращается на место.

И еще кое-что. Как только показывается вторичное меню, главное навигационное изображение, над которым пользователь установил курсор, должно быть каким-то образом отмечено, чтобы пользователь знал, что за вторичное меню он рассматривает. Пользователю также надо обеспечить возможность перемещать курсор над отдельными пунктами вторичного меню, которые тоже должны подсвечиваться.

Приступим к изучению конкретных компонентов страницы. После ввода вторичной навигации необходимо не только сменить изображение категории, над которой пользователь устанавливает курсор, но и выводить полностью новое изображение, содержащее возникающие вторичные категории (рис. 6.2). Где будет находиться это изображение? К счастью, данную задачу решить несложно. Как было сказано ранее, мы станем замещать графическую текстовую рекламу, расположенную справа от главного навигационного меню.

Вот задачи, которые должен выполнять новый сценарий:

1. Определение того, над какой основной навигационной категорией пользователь установил курсор (например, Dirt, Street и т.д.).
2. Подсвечивание первичной навигационной категории, над которой находится курсор.
3. Замена заданной по умолчанию текстовой рекламы соответствующим вторичным навигационным подменю.
4. Подсвечивание отдельных пунктов, над которыми проходит курсор, после того, как пользователь установит курсор над одним из элементов вторичного навигационного подменю.
5. Выполнение следующих действий: после ухода курсора со вторичного подменю оно должно исчезать и замещаться заданной по умолчанию текстовой рекламой; подсветка с категории главного навигационного меню убирается, а меню возвращается в состояние, заданное по умолчанию.



Рис. 6.2. Начальная страница со вторичной навигацией для раздела Dirt

Имеется два основных способа работы с графикой меню. Во-первых, — подготовка единственного изображения и применение карты изображения для установки соответствующих активных участков и замены всей графики при размещении курсора над каждой из категорий. Во-вторых, — такое разделение первичных навигационных изображений, чтобы они содержались в отдельных графических файлах, а затем — смена изображения, над которым в настоящее время появился курсор. Для наших целей более подходит второй способ. Это значит, что нужно предусмотреть две версии для каждой из основных навигационных категорий: нормальную и подсвеченную. Кроме того, должно быть несколько версий каждого подменю. Босс хочет, чтобы в любом из подменю содержались четыре ссылки.

Таким образом, требуется подготовить пять вариантов изображения: одно обычное и четыре — для разных подсвеченных вариантов. Поскольку для каждого изображения подменю имеется и четыре подсвеченных варианта, следует создать карту изображения.

<PLUG TYPE="БЕСПАРДОННАЯ РЕКЛАМА">

Метод навигации с помощью подменю намного проще реализуется в технологии DHTML. Если вы хотите узнать об этом побольше, прочтите книгу по DHTML из данной серии — “CSS и DHTML для Web-профессионалов”. Она написана с учетом того, что читатель уже немного знаком с HTML и JavaScript. Так что ее понимание не составит для вас труда после овладения материалом книги, которую вы держите в руках сейчас. Можете в этом не сомневаться.

</PLUG>

Для узла Stitch предусмотрено семь различных главных навигационных изображений. А при наличии одного нормального и одного подсвеченного варианта для каждого — 14 изображений. К тому же мы решили ограничить любое из подменю четырьмя вариантами выбора, поэтому для каждого из них нужно иметь пять отдельных график. В результате промежуточный итог подсчета числа изображений ($7 * 5$) составляет 35 изображений подменю, а общий — 49 ($35 + 14$).

Таким образом, вам придется управлять большим количеством изображений. Значит, надо либо строить громадные операторы if, позволяющие следить за соблюдением всех условий, либо создать систему, контролирующую необходимую информацию вместо вас. Ради упрощения работы выберем последний вариант. А так как мы имеем дело с журналом, содержание страниц которого регулярно обновляется, целесообразно построить код, легко дополняющийся новыми ссылками или даже новыми разделами.

Поскольку изображений будет много, надо также следить за тем, чтобы графические файлы для них имели предельно малые размеры, иначе продолжительность загрузки страницы может резко возрасти. Поэтому следует сообщить дизайнеру о необходимости придерживаться жестких требований к параметрам изображений.

Создание и заполнение массивов для хранения данных

Первый этап разработки нашего проекта предполагает создание двух глобальных переменных. Во-первых, — `whichMenu`, с помощью которой мы станем следить за тем, над какой главной навигационной категорией пользователь установил курсор в самую последнюю очередь, и какое вторичное подменю требуется отобразить. Во-вторых, — `overImage`, с помощью которой осуществляется проверка того, происходит ли перевод изображений в неподсвеченное состояние только в случае необходимости.

Для контроля за всеми изображениями надо создать многомерный массив. Назовем его `section`. До сих пор мы встречались только с одномерными массивами. Многомерные действуют практически аналогично, но предоставляют более широкие возможности. Если рассматривать одномерный массив как каталожный блок, а каждый вход массива — как ящик, в котором хранится отдельный фрагмент информации, то многомерный массив содержит по нескольку палок в каждом ящике картотеки. В результате любой ящик главного каталожного блока сам становится таким блоком.

Первая размерность в массиве `section` будет включать семь главных категорий, каждая из которых, в свою очередь — содержать вторую размерность, хранящую все отдельные варианты подменю категорий.

Прежде чем начать построение массива, определим две глобальные переменные:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
var whichMenu = " ";
var overImage = 'no';
...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Затем присвоим этим переменным значения по умолчанию, смысл которых станет ясен по мере дальнейшей разработки сценария. А теперь перейдем к созданию массива.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Броузеры ранних версий игнорируют следующий код
var whichMenu = " ";
var overImage = 'no';
section = new Array(4);
```

```
section['dirt'] = new Array (4)
  section['dirt'][0] = 'supermodel'
  section['dirt'][1] = 'affairnet'
  section['dirt'][2] = 'moss'
  section['dirt'][3] = 'thrift'

section['money'] = new Array (4)
  section['money'][0] = 'worldclique'
  section['money'][1] = 'germany'
  section['money'][2] = 'mergers'
  section['money'][3] = 'nmearnings'

section['trends'] = new Array (4)
  section['trends'][0] = 'tshirts'
  section['trends'][1] = 'sweatshirts'
  section['trends'][2] = 'hats'
  section['trends'][3] = 'mugs'

section['haute'] = new Array (4)
  section['haute'][0] = 'paris'
  section['haute'][1] = 'gaudy'
  section['haute'][2] = 'taffeta'
  section['haute'][3] = 'royalty'

section['street'] = new Array (4)
  section['street'][0] = 'angry'
  section['street'][1] = 'prison'
  section['street'][2] = 'skater'
  section['street'][3] = 'hip_hop'

section['profiles'] = new Array (4)
  section['profiles'][0] = 'moxon'
  section['profiles'][1] = 'west'
  section['profiles'][2] = 'trinidadaddy'
  section['profiles'][3] = 'spurn'

section['columns'] = new Array (4)
  section['columns'][0] = 'column1'
  section['columns'][1] = 'column2'
  section['columns'][2] = 'column3'
  section['columns'][3] = 'column4'

...
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Рассмотрим более внимательно, что происходит в приведенном выше коде. Обратимся к первой добавленной строке:

```
section = new Array(4);
```

Она должна показаться вам очень знакомой: с помощью точно такого же метода мы создали массивы для раздела Advertisers узла Stitch. Эта строка сообщает интерпретатору JavaScript о необходимости построить массив с именем section длиной в 4 элемента. В следующей строке мы уже вступаем на неизведанную для нас территорию:

```
section['dirt'] = new Array(4)
```

Здесь образуется новый массив, и массив section вставляется в позицию dirt. Тем самым первый этап создания многомерного массива выполнен. Можно приступить к заполнению массива dirt.

```
section['dirt'][0] = 'supermodel'
section['dirt'][1] = 'affairnet'
section['dirt'][2] = 'moss'
section['dirt'][3] = 'married'
```

Заполнив dirt, мы разрабатываем остальной код и заполняем массивы, содержащие информацию для других шести категорий.

Коснемся имен, которые мы присвоили информации, находящейся в массивах категорий. Они имеют смысл и найдут свое применение в этом сценарии далее. До сих пор мы всегда создавали новые объекты IMAGE для хранения изображений, применяемых в качестве сменяющихся. Для данного проекта мы собираемся использовать другой метод. При смене изображений мы станем просто устанавливать в качестве свойства источника первоначального изображения новый путь, и нам помогут в этом имена, хранящиеся в массивах. Например, в разделе Dirt есть статья с заголовком Supermodel Diet (“Чем питаются супермодели”), поэтому, когда пользователь подсветит раздел Dirt и появится соответствующее подменю, одной из его опций будет Supermodel Diet. Теперь, если пользователь установит курсор на данной опции, потребуется сменить изображение подменю, предусмотренное по умолчанию, изображением, на котором опция Supermodel подсвечена. Назвав этот графический файл supermodel.gif, мы можем просто использовать информацию, хранящуюся в массивах, для динамического создания имени файла gif в процессе изменения свойства источника подменю, предусмотренного по умолчанию. В настоящее время это, вероятно, кажется вам довольно сложным, но вскоре вы все поймете.

Создание функций управления системой подменю

После создания и заполнения массивов нужно разработать несколько функций, предназначенных для управления системой подменю. Начнем с функции, которая будет вызываться при появлении курсора над одним из главных навигационных изображений.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        ...
    }
    ...
}
// Здесь сокращение кода прекращается -->
</SCRIPT>
```

Есть много способов подсветки графики на начальной странице, и для нас особенно важно всегда иметь возможность узнать, установил ли пользователь курсор над изображением. Следить за этим нам поможет первая строка кода новой функции. Поскольку данная функция вызывается только при размещении курсора над одним из главных навигационных изображений, установим переменную overImage в значение yes.

С помощью следующего фрагмента кода осуществляется перевод в неподсвеченное состояние изображения, с которого только что был убран курсор. Для этого мы используем переменную whichMenu, объявленную в начале сценария. Немного позже в этой функции мы переустановим данную переменную так, чтобы она содержала имя раздела, над которым мгновением назад расположился курсор. Сразу после входа в функцию переменная whichMenu так или иначе должна содержать имя последней категории, отмеченной курсором. Если пользователь еще ни над чем не устанавливал курсор, эта переменная все равно будет содержать значение, присвоенное ей в начале сценария. В следующей строке находится оператор if, указывающий интерпретатору JavaScript, чтобы он выполнил очередной фрагмент кода при условии, что переменная whichMenu не содержит значение по умолчанию.

Введем код, который должен находиться в операторе if, и рассмотрим, что именно он будет выполнять.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        // Убрать подсветку с имени предыдущего раздела
        eval("document." + whichMenu + ".src =
        'images/nav/" + whichMenu + ".gif'");
    }
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Эта новая строка представляет собой первый реальный пример того, как мы будем осуществлять смену изображений в создаваемом сценарии. Применение данного метода требует особой пунктуальности, поэтому не мешает с большим вниманием отнестись к тому, что в нем происходит. Например, предположим, что пользователь убрал курсор с категории Dirt. Если бы это было действительно так, то переменная whichMenu содержала бы значение dirt. Посмотрим, как это значение будет выглядеть в новой строке кода.

```
document.dirt.src = 'images/nav/dirt.gif';
```

Здесь осуществляется попытка установить в качестве свойства источника объекта изображения dirt графический файл dirt.gif, находящийся в каталоге images/nav/. Чтобы метод мог работать, необходимо именовать все применяемые по умолчанию неподсвеченные версии главных категорий с использованием следующего синтаксиса:

```
"Name of the Category" + .gif
```

Для создания имен подсвеченных версий этих графических изображений должен применяться аналогичный синтаксис:

```
"Name of the Category" + _over.gif
```

Надеемся, вы уже полностью разобрались в новом методе смены изображений. Им удобно пользоваться, если есть возможность устанавливать соглашение об именовании график. Если же подобной возможности нет, то вы не сумеете обеспечить присвоение изображениям правильных имен, и в таком случае лучше уж придерживаться метода, который мы использовали до сих пор.

Как только прежнее изображение будет переведено в неподсвеченное состояние, нужно обновить переменную whichMenu таким образом, чтобы в ней хранилось имя категории, над которой пользователь в настоящее время установил курсор, и мы могли подсветить данное изображение.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        // Убрать подсветку с имени предыдущего раздела
        eval("document." + whichMenu + ".src =
        'images/nav/" + whichMenu + ".gif'");
    }
    // Установить переменную на новое сменяющееся изображение
    whichMenu = sectionName;

    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В приведенной строке мы устанавливаем в качестве значения переменной whichMenu значение, содержащееся в переменной sectionName, представляющее собой имя раздела, над которым пользователь только что установил курсор. Выполнив это, мы можем добавить новый код, позволяющий заменить изображение, над которым в настоящее время размещен курсор, его подсвеченной версией.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        eval("document." + whichMenu + ".src =
        'images/nav/" + whichMenu + ".gif'");
    }
    // Установить переменную на новое сменяющееся изображение
    whichMenu = sectionName;
```

```
// Подсветить новое изображение
eval("document." + whichMenu + ".src =
'images/nav/" + whichMenu + "_over.gif");
...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Эта новая строка должна выглядеть весьма похожей на ту, что применялась для перевода в неподсвеченное состояние изображения, над которым перед этим находился курсор. Единственное различие заключается в том, что мы добавили к имени файла gif суффикс `_over`. Теперь, при условии соблюдения нами правильного соглашения об именовании, изображение, над которым находится курсор, должно стать подсвеченным.

И наконец, последним этапом выполнения функции является вызов соответствующего изображения подменю. При первом выводе на экран подменю ни одна из вторичных категорий не будет подсвечена, поэтому нужно вызвать версию графического файла, применяемого по умолчанию.

```
function mainNavOn (sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        eval ("document." + whichMenu + ".src =
        'images/nav/" + whichMenu + ".gif");
    }
    // Установить переменную на новое сменяющееся изображение
    whichMenu = sectionName;

    // Подсветить новое изображение
    eval("document." + whichMenu + ".src =
    'images/nav/" + whichMenu + "_over.gif");

    // Вызвать вторичное навигационное изображение
    eval("document.submenu.src = 'images/nav/" + whichMenu
    + "_sec_blank.gif");
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Во всех применяемых по умолчанию изображениях подменю должен использоваться следующий синтаксис именования:

```
"Name of Category" + sec_blank.gif
```

Синтаксис именования подсвеченных версий изображений будет рассмотрен при описании функции `submenuOn()`, обеспечивающей подсветку соответствующей категории подменю.

Снова первым этапом является установка переменной `overImage` в значение `yes`. Это позволяет передать сценарию информацию о том, что пользователь в настоящее время установил курсор над изображением.

```
function submenuOn (sectionNum)
{
    overImage='yes';
    ...
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В дальнейшем, когда курсор будет перемещен в ту область, где находится карта изображения для подменю, важно убедиться, что он прежде прошел над одним из главных навигационных изображений, и подменю фактически стало видимым до того, как мы начали подсвечивать отдельные его элементы. Если пользователь просто проводит курсором над текстовой рекламой, никакие подменю всплывать не должны. Можно удостовериться в том, что подменю уже является видимым, проверив значение переменной `whichMenu`. Если она содержит имя раздела, а не заданное по умолчанию пустое значение, нет сомнений в том, что курсор прошел над одной из главных категорий, и появилось вторичное подменю. Для проверки применяется оператор `if` (как в начале первой функции).

```
function submenuOn (sectionNum)
{
    overImage='yes';
    if (whichMenu != "")
    {
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```


В операторе `if` в первую очередь нужно определить, над какой вторичной категорией установлен курсор в изображении подменю. Именно здесь нам понадобится наш многомерный массив.

```
function submenuOn (sectionNum)
{
    overImage='yes';
    if (whichMenu != "")
    {
        // Определить имя подраздела,
        // над которым пользователь установил курсор
        subsection = section[whichMenu][sectionNum];
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

В приведенном коде переменная `whichMenu` применялась для указания на то место в первом уровне массива, где хранится вложенный массив, который содержит информацию для вторичных категорий главной категории, выбранной в настоящее время. В переменной `sectionNum` находится числовое значение, соответствующее позиции вторичной категории, над которой пользователь установил курсор.

Если бы курсор был размещен над вторичной категорией Supermodel Diet раздела Dirt, правая часть данного оператора присваивания выглядела бы следующим образом:

```
section['dirt'][0]
```

Это, в свою очередь, инициирует обращение в данную позицию массива и вызов значения `supermodel`, которое затем помещается в переменную `subsection`.

Поместив переменную в соответствующее значение вторичной категории подменю, мы можем перейти к следующему этапу — замене графического изображения подменю таким, на котором подсвечена соответствующая категория.

```
function submenuOn (sectionNum)
{
    overImage='yes';
    if (whichMenu != "")
    {
        // Определить имя подраздела,
        // над которым пользователь установил курсор
        subsection = section[whichMenu][sectionNum];
```

```
        // Вызвать соответствующее изображение
        eval("document.submenu.src = 'images/nav/"
            + whichMenu + "_" + subsection + "_over.gif'");
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Как и во всех других случаях смены изображения, мы должны использовать конкретный синтаксис именования. Для подсвеченных вторичных категорий применяется следующий синтаксис:

```
"Name of Main Category" + "name of secondary category"
+ "_over.gif"
```

Теперь у нас есть обе функции, позволяющие переводить различные изображения в подсвеченное состояние. Следующий шаг связан с созданием функций, делающих подсвеченные изображения неподсвеченными при уходе с них курсора. Как и в функциях смены график на вторичных страницах, описанных в главе 2, прибегнем к помощи смекалки. Нам не нужно переводить в неподсвеченное состояние главное навигационное изображение только потому, что курсор переместился на подменю. Поэтому придется использовать две разные функции. Перейдем к описанию первой из них.

```
function mapOff()
{
    if (whichMenu != " ")
    {
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>
```

Опять прежде всего необходимо убедиться в том, что еще до вызова данной функции пользователь действительно установил курсор на графическое изображение. Поэтому введем оператор `if` для проверки значения переменной `whichMenu`. Как только будет определено, что можно продолжать работу, следующим этапом должна стать установка значения переменной `overImage` в `no`, поскольку курсор покинул позицию над изображением, что и привело к вызову функции.

```
function mapOff()
{
```

```

    if (whichMenu != " ")
    {
        overImage = 'no';
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Последний этап связан с вводом метода `setTimeout`, позволяющим сделать определенную паузу, а затем вызвать вторую функцию, которая нужна нам для перевода подсвеченных изображений в неподсвеченные.

```

function mapOff()
{
    if (whichMenu != " ")
    {
        overImage = 'no';
        setTimeout('mapOff2()', 1000);
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Данная новая строка сообщает интерпретатору, чтобы он выждал в течение 1000 миллисекунд (1 секунды), а затем вызвал вторую функцию перевода в неподсвеченное состояние — `mapOff2()`. Пауза дает пользователю возможность перевести курсор на иное графическое изображение, прежде чем текущая графика станет неподсвеченной. Если курсор просто перемещается с одной вторичной категории подменю на другую, нужно предотвратить мерцание главного навигационного изображения, вызванное его переходом из подсвеченного в неподсвеченное состояние и наоборот. Это станет для вас более понятным после изучения второй функции перевода в неподсвеченное состояние, так что приступим к ее разработке.

Каждый раз при уходе курсора с изображения мы устанавливаем значение переменной `overImage` в `yes`. Переменная будет использоваться для проверки того, было ли переведено подсвеченное изображение в неподсвеченное состояние. Рассмотрим первую часть функции `mapOff2()`.

```

function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {

```

```

        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Прежде чем продолжить работу и перевести в неподсвеченное состояние какие-либо изображения, следует убедиться в том, что пользователь не переместил курсор на другую графику за то время, что потребовалось для вызова новой функции. Для проверки применяется оператор `if`: если `overImage` имеет значение `no`, а `whichMenu` установлена в значение, отличное от значения по умолчанию, можно переводить в неподсвеченное состояние любые графические изображения. Прежде всего, сделаем неподсвеченной графику подменю и установим вместо нее заданную по умолчанию текстовую рекламу.

```

function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        // Возвратиться к изображению выдержки из статьи
        document.submenu.src = "images/nav/sec_blank.gif";
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

В этой строке кода применяется тот же метод, что использовался на протяжении всего данного сценария. Подменю теперь можно перевести в неподсвеченное состояние, вернув на место первоначальную текстовую рекламу. После этого мы должны сделать неподсвеченным главное навигационное изображение, которое в настоящее время имеет подсветку.

```

function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != " "))
    {
        // Возвратиться к изображению выдержки из статьи
        document.submenu.src = "images/nav/sec_blank.gif";

        // Перевести главное навигационное изображение
        // в неподсвеченное состояние
        eval ("document." + whichMenu + ".src = 'images/nav/"

```

```

        + whichMenu + ".gif");
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Последний этап выполнения функции — переустановка переменной `whichMenu` в значение по умолчанию.

```

function mapOff2()
{
    if ( (overImage == 'no') && (whichMenu != " ") )
    {
        // Возвратиться к изображению выдержки из статьи
        document.submenu.src = "images/nav/sec_blank.gif";

        // Переключить главное навигационное изображение
        // в неподсвеченное состояние
        eval("document." + whichMenu + ".src = 'images/nav/"
        + whichMenu + ".gif'");

        whichMenu = " ";
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Мы завершили разработку второй функции `off`. Теперь необходимо создать навигационную функцию, которая направит пользователя к правильному URL после щелчка на одной из вторичных категорий подменю. Поскольку существует единственная карта изображения, управляющая всеми семью подменю, следует создать функцию, определяющую, какое подменю развернуто, и выбирающую соответствующий URL.

```

function goThere (sectionNum)
{
    if (whichMenu)
    {
        ...
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Как и во всех наших функциях, прежде всего надо убедиться в том, что в настоящее время действительно выбрана какая-то категория.

Нельзя отправлять пользователя неизвестно куда, если он случайно щелкнул на текстовой рекламе. Получив положительный ответ, мы можем перейти к определению того, куда именно нужно перевести пользователя, а затем осуществить данную переадресовку.

```

function goThere (sectionNum)
{
    if (whichMenu)
    {
        subsection = section[whichMenu][sectionNum];
        location.href = whichMenu + "/" + subsection
        + ".html";
    }
}
// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

Первая новая строка кода обращается к массиву, выбирает имя страницы со статьей и помещает ее в переменную `subsection`. Затем в следующей строке эта переменная, наряду со значением, содержащимся в `whichMenu`, применяется для установки нового значения свойства `location` объекта `DOCUMENT`.

Как и при работе с изображениями, мы исходим из того, что информационное наполнение каждой категории находится в каталоге, имеющем такое же имя, как и раздел. Например, статья `Supermodel Diet` о том, как питаются супермодели, в разделе `Dirt` расположена по адресу `dirt/supermodel.html`. Опять-таки, это позволяет экономить время при разработке кода, но необходимо обязательно иметь полный контроль над соглашениями об именовании файлов.

Осталось пройти еще один этап и задача будет решена. Требуется вставить обработчики событий в теги `<A HREF>` главных навигационных изображений, а затем создать управляющую подменю карту изображения и поместить в нее обработчики событий.

Вставка обработчиков событий

Рассмотрим теги `<A HREF>` главных навигационных изображений и поместим в них обработчики событий. Ниже приведен фрагмент кода для этих изображений и изображения подменю/текстовой рекламы.

```

<TABLE>
<TR>
<TD>
<A HREF="dirt/index.html"

```

```

onMouseOver="mainNavOn('dirt');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/dirt.gif" NAME="dirt" BORDER="0"
width="135" height="30" alt="Dirt"></A><BR>

<A HREF="money/index.html"
onMouseOver="mainNavOn('money');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/money.gif" NAME="money" BORDER="0"
width="135" height="27" alt="Money"></A><BR>

<A HREF="trends/index.html"
onMouseOver="mainNavOn('trends');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/trends.gif" NAME="trends" BORDER="0"
width="135" height="29" alt="Trends"></A><BR>

<A HREF="haute/index.html"
onMouseOver="mainNavOn('haute');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/haute.gif" NAME="haute" BORDER="0"
width="135" height="26" alt="Haute"></A><BR>

<A HREF="street/index.html"
onMouseOver="mainNavOn('street');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/street.gif" NAME="street" BORDER="0"
width="135" height="27" alt="Street"></A><BR>

<A HREF="profiles/index.html"
onMouseOver="mainNavOn('profiles');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/profiles.gif" NAME="profiles" BORDER="0"
width="135" height="29" alt="Profiles"></A><BR>

<A HREF="columns/index.html"
onMouseOver="mainNavOn('columns');" onMouseOut="mapOff();" ">
<IMG SRC="images/nav/columns.gif" NAME="columns" BORDER="0"
width="135" height="26" alt="Columns"></A><BR>
</TD>

<TD>
<IMG SRC="images/nav/sec_blank.gif" NAME="submenu"
BORDER="0" USEMAP="#submenuMap" width="287" height="221"
alt="Submenu">
</TD>
</TR></TABLE>

```

Обработчики событий `onMouseOver` вам знакомы, поскольку применяются в такой же форме, как и в других сценариях смены изображения. Обработчики `onMouseOut` немного отличаются в том смысле, что мы не передаем значение в функцию, так как сменяющиеся изображения обрабатываются различным образом. Заметьте и

следующее: даже несмотря на то, что для графики подменю/текстовой рекламы применяется карта изображения, нам все равно необходимо присвоить ей свойство `NAME` в теге ``.

Теперь перейдем к рассмотрению последних включенных в карту изображения обработчиков событий. Данная карта должна иметь четыре активных области — по одной для каждой опции подменю.

```

<MAP NAME="submenuMap">
  <AREA COORDS="11,66,110,82" SHAPE="rect"
    HREF="javascript:goThere(3);" onMouseOver="submenuOn(3);"
    onMouseOut="mapOff();" ">
  <AREA COORDS="10,48,58,63" SHAPE="rect"
    HREF="javascript:goThere(2);" onMouseOver="submenuOn(2);"
    onMouseOut="mapOff();" ">
  <AREA COORDS="9,31,71,44" SHAPE="rect"
    HREF="javascript:goThere(1);" onMouseOver="submenuOn(1);"
    onMouseOut="mapOff();" ">
  <AREA COORDS="9,10,84,26" SHAPE="rect"
    HREF="javascript:goThere(0);" onMouseOver="submenuOn(0);"
    onMouseOut="mapOff();" ">
</MAP>

```

Нам потребуются три обработчика событий в каждой из активных областей. Обработчик `onClick` в поле `HREF` вызывает функцию `goThere()`, которая, при щелчке на этой опции, обеспечит направление пользователя на соответствующий адрес.

Обработчик `onMouseOver` вызывает функцию `submenuOn()`, обеспечивающую подсветку нужного изображения при перемещении курсора над какой-либо опцией. Значения, передаваемые в эту функцию, соответствуют позициям входов во второй размерности многомерного массива, который был заполнен ранее. И наконец, обработчик `onMouseOut` вызывает первую функцию перевода в не подсвеченное состояние — `mapOff()`.

Итак, вся работа выполнена! Массивы, правда, могут быть довольно большими и громоздкими, но они значительно упрощают разработку кода и позволяют без особого труда менять такие данные, как ссылки и имена разделов. В сценарии мы встретили очень много нового для себя, поэтому бросим на него традиционный последний взгляд и вспомним, что и как было сделано.

Описание сценария

1. Сначала мы создали и заполнили две переменные — `whichMenu` и `overImage` — которые в дальнейшем применяются в сценарии для контроля за тем, на каком изображении установлен курсор.

2. Затем мы определили и заполнили многомерный массив, предназначенный для хранения всей информации, необходимой для обеспечения работы системы подменю.

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Бrowsers ранних версий игнорируют следующий код
var whichMenu=" ";
var overImage = 'no';
section = new Array(4);

    section['dirt'] = new Array (4)
        section['dirt'][0] = 'supermodel'
        section['dirt'][1] = 'affairnet'
        section['dirt'][2] = 'moss'
        section['dirt'][3] = 'thrift'

    section['money'] = new Array (4)
        section['money'][0] = 'worldclique'
        section['money'][1] = 'germany'
        section['money'][2] = 'mergers'
        section['money'][3] = 'nmearnings'

    section['trends'] = new Array (4)
        section['trends'][0] = 'tshirts'
        section['trends'][1] = 'sweatshirts'
        section['trends'][2] = 'hats'
        section['trends'][3] = 'mugs'

    section['haute'] = new Array (4)
        section['haute'][0] = 'paris'
        section['haute'][1] = 'gaudy'
        section['haute'][2] = 'taffeta'
        section['haute'][3] = 'royalty'

    section['street'] = new Array (4)
        section['street'][0] = 'angry'
        section['street'][1] = 'prison'
        section['street'][2] = 'skater'
        section['street'][3] = 'hip_hop'

    section['profiles'] = new Array (4)
        section['profiles'][0] = 'moxon'
        section['profiles'][1] = 'west'
        section['profiles'][2] = 'trinidadaddy'
        section['profiles'][3] = 'spurn'

    section['columns'] = new Array (4)
```

```
    section['columns'][0] = 'column1'
    section['columns'][1] = 'column2'
    section['columns'][2] = 'column3'
    section['columns'][3] = 'column4'
```

3. После формирования массивов мы создали пять требующихся для сценария функций.

Первая — `mainNavOn()` — обеспечивает перевод в неподсвеченное состояние главных навигационных изображений, над которыми перед этим находился курсор, а также управляет подсветкой нового главного навигационного изображения и вызовом соответствующей графики вторичного подменю.

```
function mainNavOn(sectionName)
{
    overImage='yes';
    // Убедиться в том, что пользователь уже установил
    // курсор над главным навигационным изображением
    if (whichMenu != "")
    {
        // Убрать подсветку с имени предыдущего раздела
        eval("document." + whichMenu + ".src =
'images/nav/"
        + whichMenu + ".gif'");
    }
    // Установить переменную на новое сменяющееся
    изображение
    whichMenu = sectionName;

    // Подсветить новое изображение
    eval("document." + whichMenu + ".src = 'images/nav/"
    + whichMenu + "_over.gif'");

    // Вызвать вторичное навигационное изображение
    eval("document.submenu.src = 'images/nav/" + whichMenu
    + "_sec_blank.gif'");
}
```

Затем мы создали функцию `submenuOn()`, определяющую, какое подменю находится на экране, и выводящую соответствующую подсвеченную вторичную категорию для этого подменю.

```
function submenuOn (sectionNum)
{
    overImage='yes';
    if (whichMenu != "")
    {
```

```

// Определить имя подраздела,
// над которым пользователь установил курсор
subsection = section[whichMenu][sectionNum];

// Вызвать соответствующее изображение
eval("document.submenu.src = 'images/nav/"
+ whichMenu + "_" + subsection + "_over.gif'");
}
}

```

Третья и четвертая функции совместно обеспечивают перевод в неподсвеченное состояние различных изображений подменю после того, как пользователь убирает с них курсор. Первая из них — `mapOff()` — устанавливает переменную `overImage` в значение `on` и включает таймер для вызова второй функции через одну секунду. После этого `mapOff2()` проверяет, действительно ли курсор убран с графики, а затем переводит в неподсвеченное состояние все изображения с подсветкой.

```

function mapOff()
{
    if (whichMenu != " ")
    {
        overImage = 'no';
        setTimeout('mapOff2()', 1000);
    }
}

function mapOff2()
{
    if ((overImage == 'no') && (whichMenu != ""))
    {
        // Возвратиться к изображению выдержки из статьи
        document.submenu.src = "images/nav/sec_blank.gif";

        // Перевести главное навигационное изображение
        // в неподсвеченное состояние
        eval("document." + whichMenu + ".src =
'images/nav/"
+ whichMenu + ".gif'");

        whichMenu = " ";
    }
}

```

Функция `goThere()` направляет пользователя по нужному URL после щелчка на одной из активных областей карты изображения. Эта функция сначала проверяет, какая главная навигационная

категория выбрана, затем переходит в массив, выбирает имя вторичного раздела, в который должен быть переведен пользователь, и осуществляет данный перевод.

```

function goThere (sectionNum)
{
    if (whichMenu != "")
    {
        subsection = section[whichMenu][sectionNum]
        location.href = whichMenu + "/" + subsection
        + ".html"
    }
}

// Здесь сокрытие кода прекращается -->
</SCRIPT>

```

4. Заключительный этап состоял из добавления обработчиков событий к тегам `<A HREF>` главных навигационных изображений, а свойства `NAME` — к изображениям, которые могут участвовать в смене графики. Затем выполнялась вставка необходимых обработчиков событий в карту изображения, обеспечивающую доступ к этим подменю.

```

<TABLE>
<TR>
<TD>
<A HREF="dirt/index.html"
onMouseOver="mainNavOn('dirt');" onMouseOut="mapOff();">
<IMG SRC="images/nav/dirt.gif" NAME="dirt" BORDER="0"
width="135" height="30" alt="Dirt"></A><BR>

<A HREF="money/index.html"
onMouseOver="mainNavOn('money');" onMouseOut="mapOff();">
<IMG SRC="images/nav/money.gif" NAME="money" BORDER="0"
width="135" height="27" alt="Money"></A><BR>

<A HREF="trends/index.html"
onMouseOver="mainNavOn('trends');" onMouseOut="mapOff();">
<IMG SRC="images/nav/trends.gif" NAME="trends" BORDER="0"
width="135" height="29" alt="Trends"></A><BR>

<A HREF="haute/index.html"
onMouseOver="mainNavOn('haute');" onMouseOut="mapOff();">
<IMG SRC="images/nav/haute.gif" NAME="haute" BORDER="0"
width="135" height="26" alt="Haute"></A><BR>

<A HREF="street/index.html"

```

```
onMouseOver="mainNavOn('street');" onMouseOut="mapOff();" >
<IMG SRC="images/nav/street.gif" NAME="street" BORDER="0"
width="135" height="27" alt="Street"></A><BR>
```

```
<A HREF="profiles/index.html"
onMouseOver="mainNavOn('profiles');"
onMouseOut="mapOff();" >
<IMG SRC="images/nav/profiles.gif" NAME="profiles"
BORDER="0"
width="135" height="29" alt="Profiles"></A><BR>
```

```
<A HREF="columns/index.html"
onMouseOver="mainNavOn('columns');"
onMouseOut="mapOff();" >
<IMG SRC="images/nav/columns.gif" NAME="columns"
BORDER="0"
width="135" height="26" alt="Columns"></A><BR>
</TD>
```

```
<TD>
<IMG SRC="images/nav/sec_blank.gif" NAME="submenu"
BORDER="0"
USEMAP="#submenuMap" width="287" height="221" alt
="Submenu">
</TD>
</TR></TABLE>
```

```
<MAP NAME="submenuMap">
  <AREA COORDS="22,10,278,48" SHAPE="rect"
  HREF="javascript:goThere(0);"
onMouseOver="submenuOn(0);"
onMouseOut="mapOff();" >
  <AREA COORDS="22,48,278,79" SHAPE="rect"
  HREF="javascript:goThere(1);"
onMouseOver="submenuOn(1);"
onMouseOut="mapOff();" >
  <AREA COORDS="22,79,278,116" SHAPE="rect"
  HREF="javascript:goThere(2);"
onMouseOver="submenuOn(2);"
onMouseOut="mapOff();" >
  <AREA COORDS="22,116,278,157" SHAPE="rect"
  HREF="javascript:goThere(3);"
onMouseOver="submenuOn(3);"
onMouseOut="mapOff();" >
</MAP>
```

В последней главе мы ознакомились со многими новыми понятиями и расширили наши знания о средствах JavaScript, рассмотренных в предыдущих главах. Вот то, что мы изучили в данной главе:

- многомерные массивы и способы определения, заполнения и извлечения из них данных;
- новый способ создания сменяющихся изображений.

◆ Резюме

Пораженный тем, что вы сумели сделать для начальной страницы, ваш руководитель признал в вас Профессионала с большой буквы. Описанный в главе 6 сценарий — лишь один из примеров того, как много можно сделать с помощью JavaScript. Мы объединили большинство элементов, описанных в предыдущих разделах, для создания страницы, представляющей действительно высший уровень программирования Web. Тем не менее, это далеко не самое эффективное и сложное приложение JavaScript. По мере того, как популярность технологии DHTML и таблиц стилей растет, этот язык обогащается все новыми средствами, которые оказываются еще более мощными.

Мы прошли с вами нелегкий путь и создали несколько классных сценариев, пусть даже предназначенных для двух не существующих реально компаний. Настало время собрать изученное воедино и применить эти знания для работы над собственными проектами.

◆ Дополнительное задание

1. Создайте сценарий, который случайным образом сменяет баннерное рекламное изображение и обеспечивает переход с каждого из графических изображений по разным URL.
2. Создайте сценарий, обеспечивающий безопасную передачу информации, введенной в форме, шифруя данные полей формы.
3. Создайте сценарий, сохраняющий данные о пользователе в файле на Web-сервере, что даст возможность вывести для пользователя страницу, настроенную по его вкусу, после того, как он возвратится на ваш узел.
4. Для Web-узлов, содержащих большой объем графической информации на начальной странице, создайте экран заставки и напишите сценарий, который загружает всю необходимую для Home Page графику. При этом сама начальная страница должна загружаться быстро.

А Обработчики событий

Язык JavaScript основан на использовании событий. Это значит, что написанные на нем сценарии обычно выполняются в результате события, вызванного действиями пользователя или самого браузера. Данные события можно перехватывать с помощью обработчиков событий. Настоящее приложение содержит описание всех обработчиков событий, их синтаксиса и способа применения.

◆ onAbort

Описание:

Активизируется, когда пользователь прерывает загрузку изображения, либо выполнив щелчок на другой ссылке, либо нажав кнопку Stop.

Синтаксис:

```
onAbort="код, который должен быть выполнен"
```

Реализовано в версии:

Navigator 3.0

Используется в объектах:

Image

◆ onBlur

Описание:

Активизируется, когда окно, кадр или элемент формы теряют фокус.

Синтаксис:

```
onBlur="код, который должен быть выполнен"
```

Реализовано в версии:

Navigator 2.0/3.0

Используется в объектах:

Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window

◆ onChange

Описание:

Активизируется, когда поле формы Select, Text или Textarea теряет фокус и его значение изменяется.

Синтаксис:

onChange="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0/3.0

Используется в объектах:

FileUpload, Select, Text, Textarea

◆ **onClick**Описание:

Активируется при выполнении щелчка на объекте Form или когда обработчик событий находится в теге <HREF> и выполняется щелчок на изображении или текстовой ссылке.

Синтаксис:

onClick="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0/3.0

Используется в объектах:

Button, Checkbox, document, Link, Radio, Reset, Submit

◆ **onDbClick**Описание:

Активируется при выполнении двойного щелчка на объекте Form или когда обработчик событий находится в теге <HREF> и выполняется двойной щелчок на ссылке.

Синтаксис:

onDbClick="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

document, Link

◆ **onDragDrop**Описание:

Активируется, когда пользователь перетаскивает такой объект, как файл, в окно Navigator.

Синтаксис:

onDragDrop="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

Window

◆ **onError**Описание:

Активируется, когда загрузка документа или изображения вызывает ошибку.

Синтаксис:

onError="код, который должен быть выполнен"

Реализовано в версии:

Navigator 3.0

Используется в объектах:

Image, Window

◆ **onFocus**Описание:

Активируется, когда элемент Window, Frame, Frameset или Form получает фокус.

Синтаксис:

onFocus="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0/3.0/4.0

Используется в объектах:

Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window

◆ onKeyDown

Описание:

Активизируется при нажатии любой клавиши.

Синтаксис:

onKeyDown="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

document, Image, Link, Textarea

◆ onKeyPress

Описание:

Активизируется при нажатии любой клавиши.

Синтаксис:

onKeyPress="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

document, Image, Link, Textarea

◆ onKeyUp

Описание:

Активизируется, когда пользователь отпускает клавишу.

Синтаксис:

onKeyUp="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

document, Image, Link, Textarea

◆ onLoad

Описание:

Активизируется, когда завершается загрузка всего информационного наполнения в окно или всех кадров в набор кадров. Может быть также активизирован в результате загрузки изображения.

Синтаксис:

onLoad="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0/3.0

Используется в объектах:

Image, Layer, Window

◆ onMouseDown

Описание:

Активизируется при нажатии кнопки мыши.

Синтаксис:

onMouseDown="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

Button, document, Link

◆ onMouseMove

Описание:

Активизируется, когда пользователь перемещает курсор.

Синтаксис:

onMouseMove="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

Отсутствует

◆ onMouseOut

Описание:

Активизируется, когда пользователь перемещает курсор за пределы объекта.

Синтаксис:

onMouseOut="код, который должен быть выполнен"

Реализовано в версии:

Navigator 3.0

Используется в объектах:

Layer, Link

◆ onMouseOver

Описание:

Активизируется, когда пользователь перемещает курсор над объектом.

Синтаксис:

onMouseOver="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0/3.0

Используется в объектах:

Layer, Link

◆ onMouseUp

Описание:

Активизируется, когда пользователь отпускает кнопку мыши.

Синтаксис:

onMouseUp="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

Button, document, Link

◆ onMove

Описание:

Активизируется, когда сценарий или пользователь перемещает окно или кадр.

Синтаксис:

onMove="код, который должен быть выполнен"

Реализовано в версии:

Navigator 4.0

Используется в объектах:

Window

◆ onReset

Описание:

Активизируется, когда пользователь сбрасывает форму путем нажатия кнопки Reset.

Синтаксис:

onReset="код, который должен быть выполнен"

Реализовано в версии:

Navigator 3.0

Используется в объектах:

Form

◆ onSelect

Описание:

Активизируется, когда пользователь выбирает какой-то текст в поле формы Text или Textarea.

Синтаксис:

onSelect="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0

Используется в объектах:

Text, Textarea

◆ onSubmit

Описание:

Активизируется, когда пользователь подтверждает ввод формы, нажав кнопку Submit.

Синтаксис:

onSubmit="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0

Используется в объектах:

Form

◆ onUnload

Описание:

Активизируется при выходе из документа.

Синтаксис:

onUnload="код, который должен быть выполнен"

Реализовано в версии:

Navigator 2.0

Используется в объектах:

Window

Б Объекты JavaScript

Этот справочник включен в настоящую книгу, поскольку JavaScript — объектно-ориентированный язык и в нем сложно что-либо реализовать, не зная поддерживаемых им объектов. В настоящем приложении различные объекты, представленные в данном языке, разбиты на пять разных категорий и перечислены вместе с их свойствами и методами.

◆ Основные объекты

Основными являются объекты, которые не связаны с иерархией объектов JavaScript и доступны как в клиентских, так и в серверных приложениях.

Array

Описание:

Массив — это объект, позволяющий хранить наборы данных. В массиве каждый элемент набора данных хранится в уникальной позиции, что позволяет ссылаться на эти элементы или выбирать их.

Синтаксис создания:

```
new Array(arrayLength); или
new Array(element0, element1, ..., elementN);
```

Параметры:

arrayLength

Требуемая первоначальная длина массива.

element0, element1, ..., elementN

Первоначальный набор значений, которые будут храниться в массиве после его создания. Длина массива устанавливается равной числу параметров.

Реализовано в версии:

Navigator 3.0

Свойства:

`index`, `input`, `length`, `prototype`

Методы:

`concat`, `join`, `pop`, `push`, `reverse`, `shift`, `slice`, `splice`, `toString`, `unshift`

Boolean

Описание:

Применяется в качестве контейнера для логического значения.

Синтаксис создания:

```
new Boolean(value);
```

Параметры:

value. Начальное значение объекта Boolean.

Реализовано в версии:

Navigator 3.0

Свойства:

prototype

Методы:

toString

Date**Описание:**

Предоставляет возможность применять значения даты и времени.

Синтаксис создания:

```
new Date(); или
new Date("month day, year hours:minutes:seconds"); или
new Date(yr_num, mo_num, day_num, hr_num, min_num, sec_num);
```

Параметры:

day, hours, minutes, month, seconds, year.

При использовании в такой форме компоненты даты должны представлять собой строковые значения.

day_num, hr_num, min_num, mo_num, sec_num, yr_num.

При использовании в такой форме компоненты даты должны представлять собой целые числа.

Реализовано в версии:

Navigator 2.0/3.0

Свойства:

prototype

Методы:

getDate, getDay, getHours, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, parse, setDate, setHours, setMinutes, setMonth, setSeconds, setTime, setYear, toGMTString, toLocaleString, UTC

Function**Описание:**

Содержит строки кода JavaScript, выполняемые при обращении к объекту.

Синтаксис создания:

```
function name(arg1, arg2, ...argN)
{functionBody}
```

Параметры:

arg1, arg2, ..., argN

Набор строковых значений, которые могут применяться для хранения данных, передаваемых в функцию.

functionBody

Набор команд JavaScript, интерпретируемых данной функцией.

Реализовано в версии:

Navigator 3.0

Свойства:

arguments, arity, caller, prototype

Методы:

toString

Math**Описание:**

Содержит методы и свойства, позволяющие выполнять сложные математические расчеты.

Синтаксис создания:

Отсутствует. Объект Math встроен в интерпретатор JavaScript и его можно вызывать или ссылаться на него без его предварительного создания.

Реализовано в версии:

Navigator 2.0

Свойства:

E, LN10, LN2, LOG10E, LOG2E, PI, SQRT1, SQRT2

Методы:

abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan

Number**Описание:**

Содержит примитивные числовые значения, упрощает работу с числовыми значениями.

Синтаксис создания:

```
new Number(value);
```

Параметры:

value

Числовое значение, которое должно содержаться внутри объекта.

Реализовано в версии:

Navigator 3.0/4.0

Свойства:

MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY, prototype

Методы:

toString

Object**Описание:**

Встроенный объект JavaScript, от которого происходят все объекты данного языка.

Синтаксис создания:

```
new Object();
```

Параметры:

Отсутствуют

Реализовано в версии:

Navigator 2.0/3.0

Свойства:

constructor, prototype

Методы:

eval, toString, unwatch, valueOf, watch

RegExp**Описание:**

Содержит регулярное выражение, которое можно использовать для поиска, замены и манипуляции соответствиями, обнаруженными в строках.

Синтаксис создания:

```
new RegExp("pattern", "flags");
```

Параметры:

pattern

Текст, содержащийся в регулярном выражении.

flags

Флажок, который может иметь три возможных значения: глобальный поиск (g), без учета регистра (i), глобальный поиск без учета регистра (gi).

Реализовано в версии:

Navigator 4.0

Свойства:

\$n, \$, \$*, \$&, \$+, &', \$', global, ignoreCase, input, lastIndexOf, lastMatch, lastParen, leftContext, multiline, rightContext, source

Методы:

compile, exec, test

String*Описание:*

Содержит ряд символов, которые могут входить в состав строки.

Синтаксис создания:

```
new String(string);
```

Параметры:

string
Строка.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Свойства:

length, prototype

Методы:

anchor, big, blink, bold, charAt, charCodeAt, concat, fixed, fontcolor, fontsize, fromCharCode, indexOf, italics, lastIndexOf, link, match, replace, search, slice, small, split, strike, sub, substr, substring, sup, toLowerCase, toUpperCase

◆ **Объекты document**

В настоящем разделе рассматриваются объект document и все связанные с ним объекты.

Anchor*Описание:*

Содержит строку, которая является адресатом гипертекстовой ссылки, находящейся на HTML-странице.

Синтаксис создания:

```
theString.anchor(nameAttribute);
```

Параметры:

theString
Объект String.
nameAttribute
Строка с указанием имени.

Способ создания:

Создается либо с помощью тега <A> документа HTML, либо путем вызова метода String.anchor.

Реализовано в версии:

Navigator 1.0

Обработчики событий:

Отсутствуют

Свойства:

Отсутствуют

Методы:

watch, unwatch

Applet*Описание:*

Содержит все апплеты Java, находящиеся на HTML-странице.

Синтаксис создания:

Отсутствует

Способ создания:

Тег апплета HTML.

Реализовано в версии:

Navigator 3.0

Свойства:

С помощью этого объекта можно получить доступ ко всем общедоступным свойствам апплета.

Методы:

Все общедоступные методы апплета.

Area**Описание:**

Представляет область на карте изображения. Дополнительная информация о его свойствах представлена в описании объекта Link.

Реализовано в версии:

Navigator 3.0

document**Описание:**

Содержит свойства текущего документа.

Синтаксис создания:

Отсутствует

Способ создания:

Создается на основе тега <BODY> документа HTML по мере того, как интерпретатор считывает код страницы.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp

Свойства:

alinkColor, anchors, applets, bgColor, cookie, domain, embeds, fgColor, formName, forms, images, lastModified, layers, linkColor, links, plugins, referrer, title, URL, vlinkColor

Методы:

captureEvents, close, getSelection, handleEvent, open, releaseEvents, routeEvent, write, writeln

Image**Описание:**

Содержит изображение и обеспечивает доступ к свойствам изображения.

Синтаксис создания:

new Image(width, height);

Параметры:

width
Ширина изображения.
height
Высота изображения.

Способ создания:

Конструктор изображения или тег , находящийся в документе HTML.

Реализовано в версии:

Navigator 3.0/4.0

Обработчики событий:

onAbort, onError, onKeyDown, onKeyPress, onKeyUp, onLoad

Свойства:

border, complete, height, hspace, lowsrc, name, prototype, src, vspace, width

Методы:

handleEvent

Layer**Описание:**

Содержит слой, принадлежащий документу HTML, и обеспечивает доступ к свойствам слоя.

Синтаксис создания:

Отсутствует

Способ создания:

Создается с помощью тегов <LAYER> или <ILAYER> документа HTML.

Реализовано в версии:

Navigator 4.0

Обработчики событий:

onBlur, onFocus, onLoad, onMouseOut, onMouseOver

Свойства:

above, background, below, bgColor, clip.bottom, clip.height, clip.left, clip.right, clip.top, clip.width, document, left, name, pageX, pageY, parentLayer, siblingAbove, siblingBelow, src, top, visibility, zIndex

Методы:

captureEvents, handleEvent, load, moveAbove, moveBelow, moveBy, moveTo, moveToAbsolute, releaseEvents, resizeBy, resizeTo, routeEvent

Link**Описание:**

Содержит ссылку, принадлежащую документу HTML, и обеспечивает доступ к свойствам ссылки.

Синтаксис создания:

theString.link(href);

Параметры:

theString

Объект String.

href

Строка, содержащая URL.

Способ создания:

Создается либо с помощью тега <A HREF> или <AREA> документа HTML, либо путем вызова метода String.link.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseOut, onMouseOver, onMouseUp

Свойства:

hash, host, hostname, href, pathname, port, protocol, search, target, text

Методы:

handleEvent

♦ Объекты Window

В настоящем разделе рассматриваются объект Window и все связанные с ним объекты.

Frame**Описание:**

Содержит кадр, который указан в наборе кадров HTML. Каждый кадр в действительности является объектом Window; объект Frame используется просто для удобства.

Синтаксис создания:

Отсутствует

Способ создания:

Создается с помощью тега <FRAME> или <FRAMESET> документа HTML.

Реализовано в версии:

Navigator 2.0/3.0

Обработчики событий:

См. объект Window.

Свойства:

См. объект Window.

Методы:

См. объект Window.

History**Описание:**

Содержит массив, хранящий все URL, которые посетил пользователь в этом окне.

Синтаксис создания:

Отсутствует

Способ создания:

Встроенный объект JavaScript.

Реализовано в версии:

Navigator 2.0/3.0

Обработчики событий:

Отсутствуют

Свойства:

current, length, next, previous

Методы:

back, forward, go

Location**Описание:**

Содержит текущий URL.

Синтаксис создания:

Отсутствует

Способ создания:

Встроенный объект JavaScript.

Реализовано в версии:

Navigator 2.0/3.0

Обработчики событий:

Отсутствуют

Свойства:

hash, host, hostname, href, pathname, port, protocol, search

Методы:

reload, replace

screen**Описание:**

Содержит информацию о размерах и цветах экрана.

Синтаксис создания:

Отсутствует

Способ создания:

Встроенный объект JavaScript, создаваемый интерпретатором.

Реализовано в версии:

Navigator 4.0

Обработчики событий:

Отсутствуют

Свойства:

availHeight, availWidth, height, pixelDepth, width

Методы:

Отсутствуют

Window**Описание:**

Представляет броузер или кадр.

Синтаксис создания:

Отсутствует

Способ создания:

Создается каждым тегом <BODY>, <FRAMESET> и <FRAME> кода HTML или с помощью метода open объекта Window.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onDragDrop, onError, onFocus, onLoad, onMove, onResize, onUnload

Свойства:

closed, defaultStatus, document, frames, history, innerHeight, innerWidth, length, location, menubar, name, opener, outerHeight, outerWidth, pageXOffset, pageYOffset, parent, personalbar, scrollbars, self, status, statusbar, toolbar, top, window

Методы:

alert, back, blur, captureEvents, clearInterval, clearTimeout, close, confirm, disableExternalCapture, enableExternalCapture, find, focus, forward, handleEvent, home, moveBy, moveTo, open, print, prompt, releaseEvents, resizeBy, resizeTo, routeEvent, scroll, scrollBy, scrollTo, setInterval, setTimeout, stop

◆ **Объекты Form**

В настоящем разделе рассматриваются объект Form и все связанные с ним объекты.

Button**Описание:**

Содержит кнопку, принадлежащую форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "button".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onClick, onFocus, onMouseDown, onMouseUp

Свойства:

form, name, type, value

Методы:

blur, click, focus, handleEvent

Checkbox**Описание:**

Содержит флажок, принадлежащий форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "checkbox".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onClick, onFocus

Свойства:

checked, defaultChecked, form, name, type, value

Методы:

blur, click, focus, handleEvent

FileUpload**Описание:**

Содержит элемент загрузки файла, принадлежащий форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "file".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onChange, onFocus

Свойства:

form, name, type, value

Методы:

blur, focus, handleEvent, select

Form**Описание:**

Содержит форму HTML и все объекты, находящиеся в этой форме.

Синтаксис создания:

Отсутствует

Способ создания:

Интерпретатор создает этот объект, встретив тег <FORM> в документе HTML.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onReset, onSubmit

Свойства:

action, elements, encoding, length, method, name, target

Методы:

handleEvent, reset, submit

Hidden**Описание:**

Содержит объект скрытого текста, принадлежащий форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "hidden".

Реализовано в версии:

Navigator 2.0/3.0

Обработчики событий:

Отсутствуют

Свойства:

form, name, type, value

Методы:

Отсутствуют

Option**Описание:**

Содержит отдельную опцию из разворачивающегося меню выбора.

Синтаксис создания:

```
new Option(text, value, defaultSelected, selected);
```

Параметры:

text

Устанавливает текст, который должен быть отображен в списке меню.

value

Устанавливает значение, возвращаемое на сервер при подтверждении ввода формы с выбранной данной опцией.

defaultSelected

Устанавливает опцию, которая первоначально имеет истинное значение или выбрана.

selected

Устанавливает текущее состояние данной опции.

Способ создания:

С помощью тега <OPTION> или с использованием конструктора Option.

Реализовано в версии:

Navigator 2.0/3.0

Обработчики событий:

Отсутствуют

Свойства:

defaultSelected, selected, text, value

Методы:

Отсутствуют

Password**Описание:**

Содержит текстовое поле, принадлежащее форме HTML, которое скрывает свое значение.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "password".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onFocus

Свойства:

defaultValue, form, name, type, value

Методы:

blur, focus, handleEvent, select

Radio**Описание:**

Содержит отдельный переключатель из набора переключателей в форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "radio".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onClick, onFocus

Свойства:

checked, defaultChecked, form, name, type, value

Методы:

blur, click, focus, handleEvent

Reset**Описание:**

Содержит кнопку Reset, принадлежащую форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "reset".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onClick, onFocus

Свойства:

form, name, type, value

Методы:

blur, click, focus, handleEvent

Select**Описание:**

Содержит разворачивающееся меню выбора, принадлежащее форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

С помощью тега <SELECT>, вставленного в форму HTML.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onChange, onFocus

Свойства:

form, length, name, options, selectedIndex, type

Методы:

blur, focus, handleEvent

Submit**Описание:**

Содержит кнопку Submit, принадлежащую форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "submit".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onClick, onFocus

Свойства:

form, name, type, value

Методы:

blur, click, focus, handleEvent

Text*Описание:*

Содержит текстовое поле, принадлежащее форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <INPUT>, в котором в качестве атрибута TYPE указано "text".

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onChange, onFocus, onSelect

Свойства:

defaultValue, form, name, type, value

Методы:

blur, focus, handleEvent, select

Textarea*Описание:*

Содержит многострочное текстовое поле, принадлежащее форме HTML.

Синтаксис создания:

Отсутствует

Способ создания:

Тег <TEXTAREA>.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Обработчики событий:

onBlur, onChange, onFocus, onKeyDown, onKeyPress, onKeyUp, onSelect

Свойства:

defaultValue, form, name, type, value

Методы:

blur, focus, handleEvent, select

◆ **Объекты броузера**

В настоящем разделе рассматриваются объекты, которые содержат свойства, присущие конкретному браузеру.

MimeType*Описание:*

Содержит тип MIME, поддерживаемый браузером.

Синтаксис создания:

Отсутствует

Способ создания:

Создается автоматически при загрузке браузера.

Реализовано в версии:

Navigator 3.0

Свойства:

description, enabledPlugin, suffixes, type

Методы:

Отсутствуют

navigator*Описание:*

Содержит информацию о браузере пользователя.

Синтаксис создания:

Отсутствует

Способ создания:

Создается автоматически при загрузке броузера.

Реализовано в версии:

Navigator 2.0/3.0/4.0

Свойства:

appCodeName, appName, appVersion, language, mimeTypees, platform, plugins, userAgent

Методы:

javaEnabled, plugins.refresh, preference, taintEnabled

Plugin

Описание:

Содержит информацию о надстройке, установленной в броузере пользователя.

Синтаксис создания:

Отсутствует

Способ создания:

Создается автоматически при загрузке броузера.

Реализовано в версии:

Navigator 3.0

Свойства:

description, filename, length, name

Методы:

Отсутствуют

Предметный указатель

A

Атрибут:

ALT тег IMG	50
HEIGHT тег IMG	50
WIDTH тег IMG	50

B

Версия:

броузера	23
языка JavaScript	23

Вывод:

информации в документ HTML	30
нестроковых данных	36
строковых данных	36

Вызов:

обработчика событий	54
функции после паузы	68

З

Заголовок HTTP	34
----------------------	----

Загрузка:

изображения в кэш	50
предварительная	50

Закрывание окна	151
-----------------------	-----

Заполнение меню	92
-----------------------	----

И

Иерархия объектов JavaScript	24
------------------------------------	----

Изображение:

навигационное	46, 173
неподсвеченное	43

Имя функции	51
-------------------	----

Информация об авторских правах ..	33
-----------------------------------	----

K

Карта изображения	173
-------------------------	-----

Категория, навигационная	172
--------------------------------	-----

Кнопка Submit	117
---------------------	-----

Код HTML, динамическое создание	22
---------------------------------------	----

Комментарий:

HTML	23
JavaScript	26

Конкатенация	99
--------------------	----

Конструктор изображения	47
-------------------------------	----

Контроль ошибок в форме	113
-------------------------------	-----

Кэш броузера	50
--------------------	----

M

Массив	144
--------------	-----

многомерный	174
-------------------	-----

одномерный	174
------------------	-----

frames	157
--------------	-----

images	45
Меню:	
вторичное	171
разворачивающееся	77
Метка-заполнитель	85
Метод:	
close()	151
document.write()	30, 149
document.writeln()	30
eval()	52, 99, 147
indexOf()	123
open()	148
setTimeout()	68
substr()	28, 130

Н

Навигация с применением разворачивающегося меню	77, 84
--	--------

О

Область, активная	189
Обработчик событий	78, 144
JavaScript	53
onChange	81, 109
onClick	80, 117
onMouseOut	53, 188
onMouseOver	53, 188
Объект:	
DOCUMENT	143
FRAME	157
IMAGE	44, 176
NAVIGATOR	25
STRING	123
WINDOW	24
Окно, всплывающее	148
Оператор:	
инкремента	92
конкатенации	36
логическое И (&&)	126

логическое ИЛИ ()	120
логическое НЕ (!)	120
присваивания (=)	27
сравнения (==)	30
if	25, 28
Отображение: шрифта	21
Очистка меню	92

П

Панель:	
инструментов	149
навигационная	59
основная навигационная	43
Параметр функции	51
Переменная:	
глобальная	174
создание	27
языка JavaScript	27
Платформа:	
Macintosh	21
Windows	21
Подменю, графическое	169

Р

Раздел HEAD	25
Размерность массива	174
Размещение сценария	26

С

Сведения о платформе и браузере	22
Свойство:	
источника (src)	48
lastModified	34
LENGTH объекта ARRAY	88
LOCATION объекта DOCUMENT	79
navigator.platform	27
VALUE тега OPTION	78

Символ:

зарезервированный	31
кавычки одинарной	151
коммерческое at (@)	122
обратного слэша	31, 120, 151
перевода строки	120
управляющий	31
Скрытие кода от браузеров, не поддерживающих:	
JavaScript	23
IMAGE	45
Слово, ключевое:	
parent	159
top	159
Событие	53
Создание, динамическое:	
информационного наполнения	30
кода HTML	30
Создание объекта IMAGE	47
Сообщение об ошибке	117
Состояние изображения:	
неподсвеченное	59
подсвеченное	59, 60
Ссылка:	
на элемент массива	46
по имени	46
по индексу	46
Страница, начальная	58, 169
Строка состояния:	
браузера	54
содержащая пустое значение	54
Сценарий смены изображения	45

Т

Тег:

изображения	44
перевода строки HTML	36
якоря A HREF	44, 53, 148, 152
BODY	151
FONT	22
FORM	78
FRAMESET	157
HEAD	151
IMG	50
OPTION	78
SCRIPT	22, 78
Тело:	
функции	51
HTML-страницы	25

Ф

Файл JavaScript, внешний	39
Функция:	
многоступенчатого меню	90
навигации	78
предназначенная для всей страницы	51

Ц

Цикл while	91
------------------	----

Продукция соответствует требованиям
Министерства здравоохранения Российской Федерации.
Гигиеническое заключение 77.ФЦ.8.953.П.197.3.99 от 12.03.1999

Учебное пособие

Д. Баррет, Д. Ливингстон, М. Браун

JavaScript Web-профессионалам

Редактор Р.Н. Денисов
Технический редактор О.Н. Заплаткина

Лицензия на издательскую деятельность № 071405
от 28 февраля 1997 г.
ООО «Спарк».
123364, г. Москва, ул. Свободы, д. 28, корп. 2.

ООО «Издательская группа ВНУ»
Свидетельство о занесении в Государственный реестр
серия ДК №175 от 13.09.2000



Подписано в печать 20.02.2001. Формат 60×90^{1/16}.
Печать офсетная. Усл. печ. л. 15.
Тираж 6000 экз. Заказ № 225.

Отпечатано с готовых диапозитивов в ОАО «Типография «Новости»»
107005, Москва, ул. Фр. Энгельса, 46.

поддержка сервера: www.donkeyweb.ru

