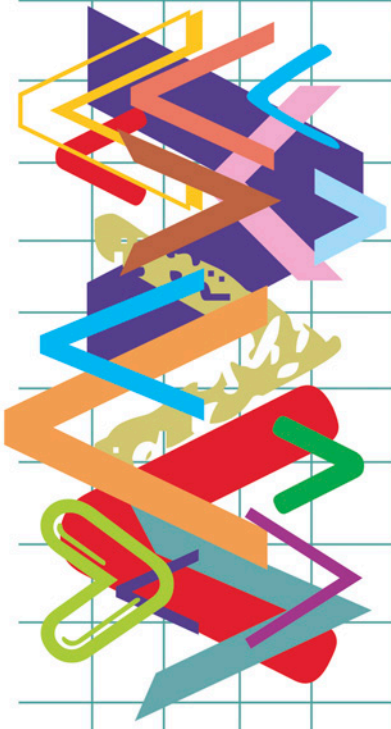
bhv
www.bhv.ru
www.bhv.kiev.ua

Игорь Шапошников

HTML 4

САМОУЧИТЕЛЬ



Стилевое оформление страниц средствами CSS

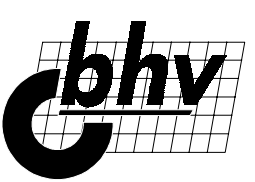
Создание сценариев с помощью JavaScript

Использование DHTML для управления Web-страницами

От Web-страницы к полноценному сайту

Игорь Шапошников

САМОУЧИТЕЛЬ HTML 4



Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Книга знакомит читателей с современными технологиями разработки Web-страниц и Web-сайтов. Подробно описан язык HTML версии 4.1. Рассматриваются способы стилевого оформления Web-страниц с помощью каскадных стилевых таблиц. Обсуждаются приемы создания динамических элементов в HTML-документах средствами технологии DHTML и языка JavaScript. Большое количество примеров, листингов и иллюстраций делают книгу особенно полезной для начинающих Web-дизайнеров. Профессионалы также смогут найти интересную информацию.

Для широкого круга пользователей

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Татьяна Коротяева*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Шапошников И. В.
Самоучитель HTML 4. — СПб.: БХВ-Петербург, 2001. — 288 с.

ISBN 5-94157-123-2

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 21.09.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 23,22.
Тираж 4000 экз. Заказ
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН.
199034, Санкт-Петербург, 9-я линия, 12.

Содержание

Благодарности

Любая книга — это плод деятельности многих людей, которые часто остаются читателю неизвестны. Мне хотелось бы поблагодарить всех, кто помог в работе над этой книгой.

Прежде всего, весь редакторский состав издательства "БХВ-Петербург" и всю группу подготовки издания.

Спасибо моей Ольге — за все. Компании "Петерлинк" — за все, что эти парни сделали для нас. Особая благодарность Сергею Торопу и Юрию Степанову. Моим друзьям в Сети. Финисту, Буке, Лайке, Панку, Умке и многим-многим другим. Мой почтовый ящик всегда открыт для вас.

Огромное спасибо всем читателям этой книги. Ваши вопросы и отзывы показывают, что работа была проделана не зря.

Ольге

Твоя улыбка стоит целого мира.

Введение

Самая распространенная и известная часть Интернета — так называемая "паутина", World Wide Web. Она состоит из Web-сайтов и отдельных Web-страниц, а Web-страницы создаются с помощью языка HTML. Этот язык и является предметом нашего интереса. Мы рассмотрим новейший стандарт языка HTML, его четвертую версию. Если быть совсем точным, то версию под номером 4.1.

Язык HTML довольно прост, и пользоваться его возможностями может каждый, для этого совсем необязательно быть программистом. Достаточно иметь некоторый опыт работы в Интернете и быть обычным пользователем компьютера. Никаких специальных навыков не нужно. Это *действительно* просто. Мы вместе в этом убедимся. С помощью языка HTML каждый в состоянии создать собственную Web-страничку или целый Web-сайт и представить себя таким образом во всемирном информационном пространстве. Впрочем, для чего создавать свой Web-сайт — каждый решает самостоятельно. Задача этой книги — лишь дать читателю подходящий инструмент.

Книга разбита на три главы. В первой главе мы тщательно и подробно рассмотрим полный стандарт языка HTML версии 4.1. с примерами и иллюстрациями. Вторая глава посвящена вопросам единого стилевого оформления Web-страниц с помощью так называемых каскадных стилевых таблиц. Эта технология носит название CSS (Cascading Style Sheets). В заключительной главе мы коснемся вопросов использования динамических элементов. Технология, которую мы будем для этого применять, так и называется — динамический HTML (DHTML). Таким образом, мы научимся создавать полноценные информационные Web-сайты.

Конечно, только знания языка HTML недостаточно для того, чтобы сделать интерактивный торговый сайт. Для таких серьезных целей требуется хоро-

шее знание программирования и различных технологий Интернета. Но в том-то и преимущество языка HTML, что, пользуясь этим несложным инструментом, можно создавать полноценные информационные Web-сайты. Средств HTML с лихвой хватит для нужд большинства посетителей паутины, за исключением пяти или десяти процентов. Если же возникнет надобность использовать на своем сайте какую-либо специализированную технологию, то придется изучить что-то еще, благо литературы сейчас достаточно. Но перед тем как обращаться к новейшим средствам сетевого программирования, следует научиться использовать обширные возможности HTML. Из этой книги мы узнаем о них практически все.

Несмотря на то, что язык HTML достаточно прост по сравнению с прочими современными технологиями сетевого программирования, Всемирная паутина — самая обширная и популярная часть Интернета — целиком базируется на огромной массе частных и корпоративных сайтов, которые созданы только с помощью этого языка. Технология Интернета, а вместе с ней и язык HTML, относятся к тем немногим достижениям компьютерной индустрии, которые повлияли на распространение и развитие гражданских свобод. Любой человек, имеющий доступ в Интернет, пусть и нерегулярный, может получать и сообщать информацию, не прошедшую государственный или корпоративный контроль. Таким образом, свобода слова и легко реализуемое право на получение правдивой информации стали самыми важными достижениями общества, обеспечиваемыми технологиями Интернета. Если прибавить к этому легко реализуемую анонимность или, другими словами, приватность работы в Сети, то становится ясно, что с помощью Интернета человек может выйти из-под информационного контроля, который в той или иной форме обязательно проводится любым современным государством. Плату же за предоставление доступа в Интернет можно считать налогом на свободу, который следует выплачивать сполна. Интернет — это, прежде всего, свобода и право на самовыражение, но для того, чтобы воспользоваться этими преимуществами, следует освоить соответствующие технологии. Мы уже говорили, что для Всемирной паутины база и основа — язык HTML.

ГЛАВА 1

Язык HTML

Основы

Интернет — это огромное объединение компьютерных сетей в планетарном масштабе. Если учесть, что и обитатели международной космической станции пользуются услугами электронной почты, то становится ясно, что Интернет уже шагнул за пределы планеты. Очень часто Интернет ошибочно отождествляют с самой популярной и масштабной его частью — Всемирной паутиной, которая в английском языке получила наименование WWW (World Wide Web). По сути дела, паутина — это огромное количество взаимосвязанных документов. Ключевое слово — *взаимосвязанных*. В текст Web-страницы органично вставляются гиперссылки, которые обеспечивают соединения с другими Web-страницами. Описать несколькими словами механизм гиперссылок трудно, но тот, кто хоть раз посетил какой-либо Web-сайт, сразу поймет их значение.

Именно гиперссылки, позволяющие связывать друг с другом самые различные документы из Сети, создали ту удивительную общность, которая составляет теперь главную отличительную черту Всемирной паутины. Гиперссылки используют для поиска документа его уникальный адрес во Всемирной паутине, который также называется URL (Universal Resource Locator).

Как мы знаем, основное назначение Web-страниц — *отображать* информацию и делать ее доступной для пользователя. При этом существует ряд функциональных ограничений: заранее неизвестно, какой именно компьютер у пользователя, просматривающего Web-страницу, какое разрешение у его монитора, или какие размеры окна просмотра он установил. Мы даже не можем заранее знать, какая используется операционная система или платформа. Web-страницы должны практически одинаково отображаться и на Intel-машинах, и на Макинтошах, и на телевизионных Web-приставках. Неизвестно, какие шрифты установлены и используются в операционной системе пользователя, или какая глубина цвета поддерживается его видеокар-

той. Отсутствие или недостаток информации должны были бы стать непреодолимым барьером на пути создания общего языка, но этого не случилось.

Дело в том, что еще в 1986 году Международной организацией по стандартизации ISO (International Organization for Standardization) был создан язык разметки документов SGML (Standard Generalized Markup Language), который предусматривал практически все допустимые варианты отображения информации как на бумаге, так и на мониторах. Чтобы учесть все возможные случаи, была разработана мощная система. Казалось бы, для Web-страниц — это идеальный вариант. Но только описание правил этого языка занимает сотни страниц. На разработку программ, которые могли бы отображать страницы, созданные на основе такого языка, ушло бы очень много времени, поэтому для нужд Интернета было выбрано некоторое подмножество языка SGML, которое получило самостоятельное наименование — HTML (HyperText Markup Language), т. е. язык разметки гипертекстовых документов.

В файле описания Web-страницы на языке HTML основная информация чередуется с инструкциями по ее отображению. По сути, это обычный текстовый файл, но читать его без применения соответствующих специализированных программ-браузеров трудно, т. к. инструкции по отображению информации затрудняют чтение текста. А графику тем более нельзя увидеть, потому что в самом HTML-файле вместо графики стоит тэг, указывающий браузеру, что именно в это место надо вставить некое изображение.

Web-страницы, написанные на HTML, просматриваются с помощью специализированных программ, которые обычно называют *браузерами*. Это калька англоязычного термина. Прямой перевод на русский язык, программы-обозреватели, почему-то не прижился. Что ж, будем называть их устоявшимся термином. Основная задача браузера — по запросу пользователя найти требуемый документ в Интернете и без искажений отобразить его. Сначала браузер анализирует инструкции, написанные на языке HTML, а затем, пользуясь этими инструкциями, отображает информацию, находящуюся на Web-странице. Отсюда вывод — если мы хотим создавать собственные Web-страницы, то жизненно необходимо знать язык HTML.

Конечно, можно писать код Web-страницы вручную, пользуясь каким-нибудь простеньким текстовым редактором, таким, например, как тривиальный "Блокнот", но это не самое приятное времяпрепровождение. Сейчас существует огромное множество визуальных редакторов Web-страниц, которые позволяют простым и естественным образом размещать информацию, не заботясь о ее переводе в HTML. Казалось бы, если все так замечательно, то зачем изучать HTML самостоятельно? Оказывается, при создании HTML-кода эти редакторы в некоторых случаях пишут неверный, частично неверный или избыточный код. Иногда просто не удается добиться именно того результата, который необходим, и надо знать язык HTML, чтобы понять, как преодолеть барьеры, встроенные в эту технологию.

Также следует упомянуть о последствиях так называемой "браузерной войны". Дело в том, что в самом начале развития WWW лидерство на рынке захватил браузер Netscape Navigator от Netscape. Фирма Microsoft изначально не смогла правильно оценить будущего потенциала WWW, и браузер Netscape Navigator благополучно завоевал почти весь рынок. Но когда менеджмент Microsoft понял, что они упустили, в ход пошла вся тяжелая артиллерия. Срочно был создан браузер Internet Explorer, и началась браузерная война. В целях продвижения собственного браузера каждая из конкурирующих фирм немного "улучшала" стандарт HTML, т. е. добавляла в него конструкции, которые мог правильно обрабатывать только собственный браузер. Конечно, WWW Consortium, организация, курирующая развитие интернет-технологий, некоторые из этих новшеств включала в последующие версии стандартов, но конкуренты не останавливались на достигнутом. Более того, в рамках все той же браузерной войны Microsoft включила свой браузер в операционную систему Windows 9x. Закончилось вся эта эпопея, как мы помним, судебным разбирательством, в котором ставший в одночасье известным всему миру судья Джонсон обязал Microsoft вывести браузер Internet Explorer из состава операционных систем.

К тому времени задача-минимум уже была решена. Браузер Internet Explorer занял около половины объема рынка браузеров. Так закончилась "браузерная война". Или, если быть точным, перешла из "горячей" фазы в "холодную". Последствия этого противостояния до сих пор ощущаются разработчиками Web-страниц, т. к. им приходится проверять, как выглядит разрабатываемая страница в каждом браузере. Более того, некоторые визуальные HTML-редакторы прямо ориентированы на тот или иной браузер, и, следовательно, используя их, разработчик тоже будет ориентироваться только на один браузер или ограничивать функциональность и наполнение Web-страниц, упрощая их структуру до предела.

Но выход есть. Код, генерируемый визуальным редактором, практически всегда необходимо исправлять вручную, а для этого, повторюсь, стоит знать язык HTML. Без знания HTML просто нельзя создавать Web-страницы хорошего качества. В любой технологии есть свои подводные камни, и если мы не знаем основы этой технологии, мы обязательно на них наткнемся. Для того чтобы добиться максимального соответствия создаваемой Web-страницы первоначальному замыслу, действительно необходимо изучать язык HTML. Чему, собственно, и посвящена эта книга — замечание для тех, кто еще не догадался...

Структура HTML-документа

Конструкции HTML называются *тэгами*. Для того чтобы браузер мог отличить их от обычного текста, они заключаются в угловые скобки. Тэг обозначает начало действия какой-либо инструкции отображения. Если эта инст-

рукция применяется ко всему документу, то такой тэг не имеет своего закрывающего двойника. Большинство тэгов все-таки парные, и второй тэг прекращает действие первого. Например, каждая Web-страница должна начинаться с тэга <html>, а заканчиваться его закрывающим двойником </html>. Обратите внимание, закрывающий тэг отличается от открывающего лишь наличием косой черты после первой угловой скобки.

Некоторые тэги обладают параметрами, которые уточняют правила отображения содержимого. Немного позже мы на примере увидим, как применяются эти параметры, а сейчас лишь отметим, что параметры могут указываться только в открывающем тэге.

Наименования тэгов и их параметров могут быть написаны в любом регистре как заглавными символами, так и строчными. Анализаторы HTML, встроенные в каждый браузер, не обращают внимания на регистр символов, которыми набраны все служебные конструкции HTML-документов.

В HTML, как и в любом компьютерном языке, нельзя обойтись без комментариев, содержимое которых не обрабатывается браузером и не отображается. Они служат лишь для удобства разработчика, для внутреннего документирования структуры текста. Комментарии располагаются между фрагментами <-- и -->. Вот пример создания комментария:

```
<-- Это комментарий -->
```

Любая Web-страница структурно разбивается на две части: заголовок и тело. В заголовке указывается служебная информация обо всей Web-странице, а в теле Web-страницы мы описываем ее содержимое вместе с правилами его отображения. При этом заголовок Web-страницы ограничивается тэгами <head> и </head>, а тело документа обозначается тэгами <body> и </body>. По правилам хорошего стиля программирования перед заголовком ставится идентификатор применяемого стандарта HTML. Структура любой Web-страницы выглядит следующим образом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  Заголовок документа
  </head>
  <body>
  Тело документа
  </body>
</html>
```

Первый тэг <!DOCTYPE> со всеми его параметрами — это идентификатор, который сообщает браузеру, какая именно версия HTML была использована для создания данной Web-страницы. Эта достаточно громоздкая и непонятная конструкция на самом деле является пришельцем из языка более высо-

кого уровня, чем HTML. Подразумевается, что в будущем браузеры смогут работать одновременно как с обычными Web-страницами, написанными на языке HTML, так и с HTML-документами. С расчетом на это светлое будущее и используется данный тэг-идентификатор. Точная дата наступления этого светлого будущего, как обычно, неизвестна, поэтому очень часто этим идентификатором пренебрегают без каких-либо последствий. Но предусмотрительность, как известно, лучше, чем недальновидность, поэтому идентификатор лучше все-таки использовать.

Теперь рассмотрим заголовок. В него могут входить тэг, отображающий наименование Web-страницы, тэг стилевого оформления Web-страницы, тэг выполняемого сценария и так называемые метаданные. Стилевое оформление Web-страниц будет рассматриваться во второй главе, а выполняемые сценарии — в третьей. О метаданных мы поговорим чуть позже, а сейчас узнаем, как использовать наименование Web-страницы.

Вы наверняка замечали, что при загрузке Web-страницы в самой верхней строке браузера появлялось краткое наименование загружаемого документа. Для создания такого заголовка используется тэг `<TITLE>` с соответствующей закрывающей конструкцией. Начальный блок Web-страницы с обозначением подобного заголовка может выглядеть следующим образом:

```
<head>
<title> Заголовок Web-страницы</title>
</head>
```

Заголовком Web-страницы никогда не следует пренебрегать, т. к. это самое первое, что видит посетитель Web-сайта. Заголовок отображается еще до того, как произойдет окончательная загрузка содержимого страницы, поэтому выбирать его следует тщательно.

С первой частью структуры Web-страницы мы разобрались и теперь можем переходить к рассмотрению тела HTML-документа, его основной части. Как мы уже знаем, содержимое Web-страницы располагается между тэгами `<body>` и `</body>`. В простейшем случае это может быть обыкновенный текст. Браузер правильно интерпретирует его и отобразит. Попробуем увидеть это на примере.

Для создания нашей первой Web-страницы нам потребуется обычный текстовый редактор. Стандартный "Блокнот" вполне подойдет. Нам достаточно будет создать текстовый файл, содержимое которого приведено в листинге 1.1.

Листинг 1.1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
```

```
  <head>
  <title> Моя первая Web-страница</title>
  </head>
  <body>
  Доброго времени суток всем посетившим мой скромный сайт.
  </body>
</html>
```

Не забудьте, сохраняя файл, установить для него расширение htm или html. Если после этого один или два раза (в зависимости от настроек операционной системы) щелкнуть кнопкой мыши по имени файла в Проводнике Windows, то автоматически будет запущен браузер, установленный по умолчанию, и в него уже будет загружен выбранный HTML-документ. Как выглядит наша первая Web-страница в браузере Internet Explorer, видно на рис. 1.1.

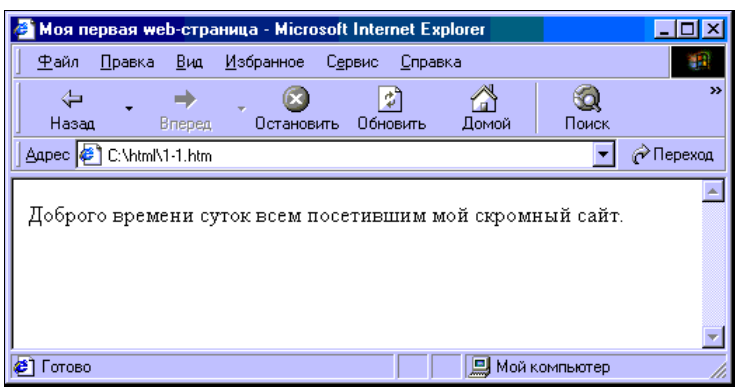


Рис. 1.1. Результат просмотра в браузере файла, приведенного в листинге 1.1

Следует отметить, что тэг `<body>` может содержать дополнительные параметры. Мы уже говорили немного ранее, что параметры включаются в состав стартового тэга конструкции. Теперь пришло время увидеть, как это происходит на самом деле.

Чаще всего параметр представляет собой пару "наименование-значение". Рассмотрим простой пример. Параметр `bgcolor` позволяет задавать цвет фона, на котором будет отображаться содержимое Web-страницы. Например, если мы хотим использовать зеленый фон, то мы должны применить следующую конструкцию:

```
<body bgcolor="green">
```

Все текстовые значения параметров обычно заключаются в кавычки. О том, как задаются цвета, мы узнаем в следующих разделах этой главы, а сейчас вернемся к параметрам тэга `<body>`.

О параметре `bgcolor`, который позволяет устанавливать цвет фона Web-страницы, мы уже знаем. Рассмотрим остальные параметры.

- Параметр `background` дает возможность использовать в качестве фона какое-либо графическое изображение. Значением параметра является адрес этого изображения, т. е. его URL (Universal Resource Locator).
- Параметр `text` задает цвет шрифта, которым будет отображаться текстовое содержимое Web-страницы.
- Параметр `link` позволяет устанавливать цвет, которым будут отображаться в окне просмотра браузера текстовые гиперссылки, внедренные в содержимое Web-страницы.
- Параметр `vlink` задает цвет гиперссылок, которые пользователь уже просматривал в текущем сеансе работы.
- Параметр `alink` определяет, какой цвет будет использоваться для отображения гиперссылок, выделенных пользователем.
- Параметр `lang` указывает, на каком языке написано текстовое содержимое Web-страницы. В качестве значения используются кодовые двухбуквенные обозначения языков, приведенные в документе RFS 1766. Все обозначения нам знать не нужно. В подавляющем большинстве случаев мы будем использовать русский или английский язык. Их коды — "`ru`" и "`en`", соответственно.

Помимо вышеперечисленных параметров тэг `<body>` может обладать двумя идентифицирующими параметрами `id` и `class`, но на практике они в этом тэге почти никогда не задаются.

Как видно, все просто и незатейливо. Теперь самое время узнать, что же такое метаданные. Метаданные можно определить как неотображаемую информацию о документе. Она используется для идентификации документа и указания режима отображения Web-страницы. Для внедрения метаданных в Web-страницу применяется тэг `<meta>`. Чаще всего он имеет следующий вид:

```
<meta name="имя переменной" content="значение переменной">
```

Таким образом, если мы хотим указать авторство какой-либо Web-страницы, достаточно вставить в блок ее заголовка следующую конструкцию:

```
<meta name="Author" content="It's me!!!">
```

Установка собственных переменных в метаданных необходима только в том случае, если Web-страницы обрабатываются с помощью какого-либо специализированного интернет-приложения. Метаданные могут понадобиться и для просмотра Web-страницы браузером.

Как мы все знаем, недостаточно просто поместить сайт во Всемирную паутину, надо еще сделать так, чтобы он попал в списки поисковых машин. Мы не будем сейчас рассматривать полностью процедуру регистрации сайта

на поисковых машинах, тем более что для каждой такой машины процедура регистрации своя. Нас интересует кое-что другое.

Откуда поисковые машины берут информацию о содержимом той или иной Web-страницы? Как раз из метапеременных. Чаще всего используются метапеременные с именами `keywords` и `description`. Переменная `keywords` содержит список ключевых слов Web-страницы. А переменная `description` предназначена для хранения краткой аннотации Web-страницы. Приведем пример использования подобных метаданных. Предположим, что наша Web-страница посвящена сложному и деликатному вопросу правильного кормления хомячков, тогда ее структура должна выглядеть приблизительно следующим образом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Все о кормлении хомячков</title>
  <meta name="keywords" lang="ru" content="хомяк, грызун, кормление,
↳животное, уход">
  <meta name="description" content="Web-страница о кормлении мелких
↳грызунов, в частности хомяков, в условиях домашнего содержания">
  </head>
  <body>
  О, эти маленькие симпатичные животные — хомячки...
  </body>
</html>
```

Зная, что идентификация содержимого Web-страниц поисковыми машинами осуществляется с помощью ключевых слов, указываемых разработчиком, вы можете ввести в их состав слова, не отражающие суть документа, но часто запрашиваемые посетителями поисковых машин. Должен заметить, что этот фокус, вероятно, не получится. Дело в том, что поисковые машины чаще всего проверяют еще и текст самой Web-страницы, и если какое-либо ключевое слово не встречается в нем, то оно просто не учитывается.

Следует также обратить внимание на то, что при указании ключевых слов мы добавили в тэг `<meta>` дополнительный параметр `lang`. Этот параметр предназначен для указания языка, на котором написан тот или иной текст. Мы можем задать наборы ключевых слов на разных языках, используя для этого несколько тэгов `<meta>`. В нашем примере мы указали, что перечисленные ключевые слова написаны на русском языке.

Кроме того, метаданные позволяют описывать так называемые заголовки HTTP. Здесь необходимо сделать маленькое техническое отступление. Все HTML-документы передаются с помощью специализированных программ, называемых Web-серверами, по определенным правилам. Набор правил

приема и передачи информации в компьютерной индустрии называется протоколом. Для передачи Web-страниц и данных от удаленных пользователей применяют протокол HTTP (HyperText Transfer Protocol). Он обладает набором директив и переменных, которые часто называют заголовками HTTP-протокола.

Мы не ставим перед собой задачу изучить все переменные протокола HTTP, нам достаточно будет узнать о наиболее часто применяемых его заголовках. Прежде всего, стоит упомянуть о переменной `Expires`, которая позволяет устанавливать так называемый "срок годности" Web-страницы. Дело в том, что браузеры и некоторые другие коммуникационные программы сохраняют посещенные пользователем Web-страницы в кэше, а затем, когда пользователь запрашивает их снова, предлагают ему эти копии, экономя, таким образом, время получения. Web-страницы все-таки достаточно часто обновляют, поэтому пользователь может получить устаревшую копию страницы. Конечно, существуют способы настройки правил работы с кэшем, но далеко не все их используют. Лучше подстраховаться и указать "срок годности" Web-страницы. Если он прошел, то браузер вместо использования копии из кэша запросит документ из Сети.

Тэг `<meta>`, приспособленный для указания срока годности Web-страницы, выглядит приблизительно следующим образом:

```
<META http-equiv="Expires" content="Tue, 20 Aug 1996 14:25:27 GMT">
```

Из примера видно, что для указания наименования стандартной переменной HTTP-протокола используется параметр `http-equiv`, а для установки значения этой переменной — уже знакомый нам параметр `content`. Легко заметить, что установка срока последнего использования документа производится с помощью переменной `Expires`, ее значение должно быть записано в определенном текстовом формате с указанием времени по гринвичскому меридиану.

Информация на страничке может меняться настолько быстро, что ее необходимо несколько раз перезагружать в процессе одного сеанса работы. Это достаточно частое явление встречается в чатах и на Web-страницах с информацией, обновляемой в реальном времени, например, при отображении изменений котировок ценных бумаг во время операционного дня на фондовой бирже. В этом случае необходимо использовать переменную с наименованием `Refresh`. Значение этой переменной указывается в секундах. Рассматриваемый нами тэг `<meta>` приобретет следующий вид.

```
<META http-equiv="Refresh" content=10>
```

Страница с подобной конструкцией в блоке заголовка будет автоматически перезагружаться каждые десять секунд.

На этом заканчивается рассмотрение структуры заголовка HTML-документа. Мы переходим к изучению структуры основного раздела Web-страницы.

Как мы помним, вся отображаемая в окне просмотра браузера информация размещается между тэгами <body> и </body>. О том, какие возможности отображения содержимого Web-страницы нам предоставляет язык HTML, мы узнаем в следующих разделах этой главы. Здесь мы рассматриваем лишь общую структуру HTML-документа.

HTML позволяет для каждого применяемого тэга задать уникальный идентификатор. Скажем, если наш текст разбит на абзацы, то каждому абзацу мы можем присвоить специфичное наименование, а затем, с помощью некоторых дополнительных средств языка HTML, управлять отображением этих абзацев. Мы можем делать некоторые из них невидимыми, менять цвет шрифта, т. е. изменять правила их отображения. Это относится не только к абзацам, а ко всем частям содержимого Web-страницы, которые заключены в те или иные тэги.

Для идентификации какого-либо тэга применяется параметр id. Вернемся к примеру с абзацами текста. Забегая немного вперед, можно сказать, что абзацы указываются с помощью пары тэгов и . Таким образом, описание абзацев, которые мы сможем потом отличать, выглядит так:

```
<p id="p1">Первый абзац</p>
<p id="p2">Второй абзац</p>
```

Значения всех параметров id в HTML-документе должны быть уникальными. Если встречается пара идентификаторов, имеющих одинаковые значения, то эти идентификаторы просто игнорируются. Естественно, применение параметра id не является обязательным. Имеет смысл использовать его только в тех случаях, когда конструкция с идентифицируемым тэгом будет подвергнута стилевой обработке (о которой мы поговорим во второй главе), или этот тэг будет закладкой в документе, на которую указывает какая-либо гиперссылка, либо предполагается динамическая обработка тэга с помощью инструкций DHTML, о которых мы узнаем в третьей главе. Идентификаторы применяются и в тех случаях, когда HTML-документ обрабатывается специализированными приложениями, но это уже для совсем серьезных программистов. Нам это пока не нужно.

Если параметр id позволяет отличать тэг от всех остальных, то с помощью параметра class мы можем относить тэг к той или иной группе. Этот параметр используется только для стилевого оформления. Мы разбиваем некоторые элементы Web-страницы на классы, а затем достаточно в одном месте изменить описание правил отображения класса, и это изменение автоматически распространится на все тэги, которые вошли в искомый класс.

Нам доступны методы объединения соседних элементов Web-страницы в единые блоки. Все элементы оформления HTML-документов разделяются на два типа: Inline-элементы, которые чаще всего являются просто элементами текста, и блочные элементы. Inline-элементы могут быть частью стро-

ки, а блочные элементы всегда занимают обособленное место на Web-странице и обязаны начинаться постоянно с новой строки. Блочные элементы могут включать в себя другие блочные элементы и inline-элементы. По вполне понятным причинам inline-элементы не могут содержать блочные элементы.

Объединение элементов Web-страницы в блоки позволяет применять к ним единое оформление, осуществлять некое подобие верстки. Можно изменить расположение блока, переопределив один объединяющий тэг. Естественно, это удобнее, чем менять расположение каждого элемента Web-страницы по отдельности.

Для объединения элементов блочного типа используется тэг `<div>` с его закрывающим двойником`</div>`. Inline-элементы объединяются парой тэгов `<span>` и `</span>`. Учитывая вышесказанное, ясно, что блок с тэгом `<div>` не может располагаться внутри блока с тэгом `<span>`, т. к. блочные элементы не могут входить в состав inline-элементов.

Также следует отметить, что браузеры обрамляют div-блоки разрывами строки. Проще всего это показать на примере.

Листинг 1.2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Отображение div-блоков</title>
  </head>
  <body>
  <div>
  Доброго времени <div>суток всем посетившим </div>мой скромный сайт.
  </div>
  </body>
</html>
```

Результат отображения этого HTML-файла браузером Internet Explorer показан на рис. 1.2.

Тэги `<span>` и `<div>` могут также иметь дополнительные параметры. Помимо уже знакомых нам идентифицирующих параметров `id` и `class`, используются параметры `style` и `align`. Параметр `style` применяется для установки стиля отображения содержимого блока, а параметр `align` позволяет устанавливать выравнивание данного блока относительно других элементов содержимого Web-страницы. Более подробно применение этих параметров мы рассмотрим в следующих разделах этой главы.

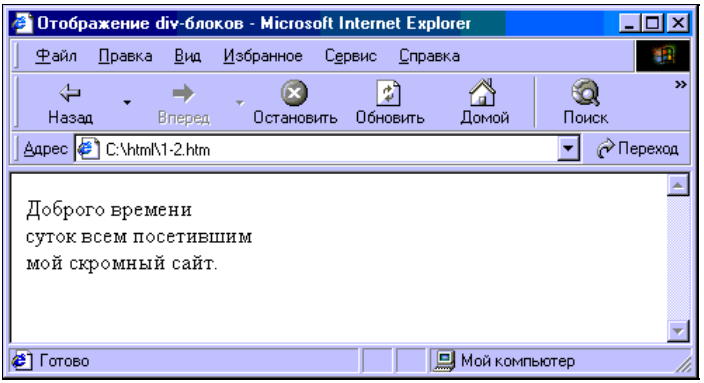


Рис. 1.2. Окно браузера с результатом отображения файла, приведенного в листинге 1.2

К структурным элементам HTML-документа можно отнести различные заголовки в тексте. Для заголовков в HTML существуют собственные тэги. Всего в HTML-документах применяется шесть уровней текстовых заголовков. Самый старший уровень — первый. Для каждого заголовка есть свой тэг и свои правила отображения.

Тэги для обозначения заголовков чрезвычайно просты. Для заголовка первого уровня применяется тэг `<h1>` с его закрывающим двойником `</h1>`, заголовок второго уровня описывается парой `<h2>` — `</h2>`, и т. д., вплоть до заголовка шестого уровня с тэгом `<h6>`. Ниже, в листинге 1.3 приведен пример использования заголовков в HTML-документе.

Листинг 1.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Отображение заголовков</title>
  </head>
  <body>
  <h1> Заголовок первого уровня </h1>
  <h2> Заголовок второго уровня </h2>
  <h3> Заголовок третьего уровня </h3>
  <h4> Заголовок четвертого уровня </h4>
  <h5> Заголовок пятого уровня </h5>
  <h6> Заголовок шестого уровня </h6>
  <p>Обычный текст</p>
  </body>
</html>
```

А как это все выглядит, хорошо видно на рис. 1.3.

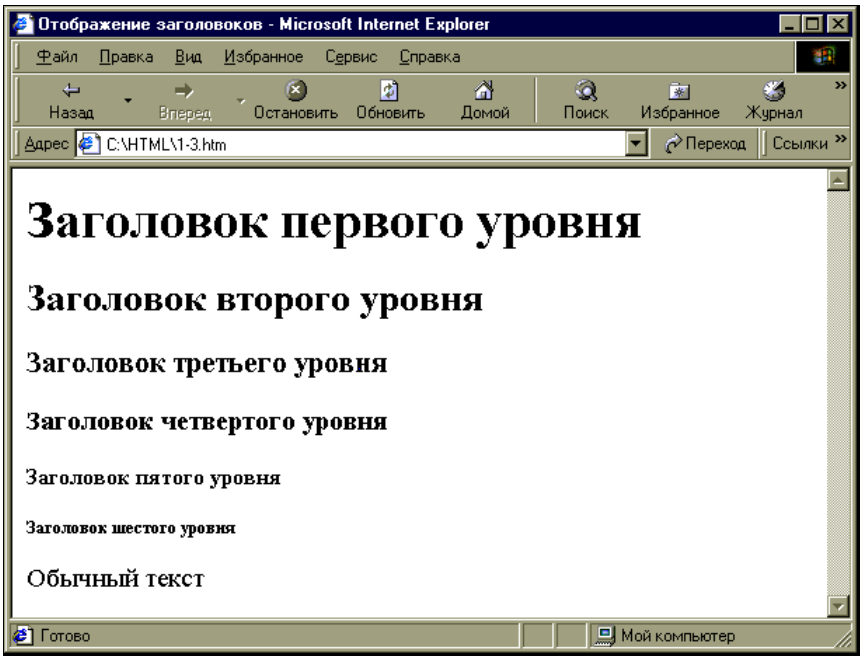


Рис. 1.3. Окно браузера с результатом отображения файла, приведенного в листинге 1.3

Тэги заголовков обладают тем же набором параметров, что и недавно рассмотренные тэги `<span>` и `<div>`, а именно идентификационные `id` и `class`, параметр общего оформления `style` и параметр выравнивания `align`.

На этом заканчивается рассмотрение структуры типичного HTML-документа. Как видно, никаких особых сложностей в этом нет. Все стройно и логично.

Используемые символы

Теперь обратимся к правилам использования символов в HTML. В компьютерах каждый символ — это некое число. Операционная система при отображении текста выводит символ, соответствующий какому-либо числу. Таблица соответствия чисел и символов называется *кодировкой*. Сейчас существует не менее пяти кодировок только для русскоязычных символов. В каждый браузер встроена функция смены кодировки отображаемой Web-страницы. Если браузер не распознает, какая кодировка использована при создании Web-страницы, то вместо текста пользователь увидит мешанину из

непонятных символов. Каждый, наверное, с этим встречался. Язык HTML позволяет указать используемую кодировку, чтобы браузер не пытался распознать ее самостоятельно. Для этого применяется уже знакомый нам тэг <meta>. Среди предопределенных переменных протокола HTTP есть переменная с именем Content-Type. Она задает тип содержимого Web-страницы и дополнительно позволяет указывать наименование применяемой кодировки. Полностью соответствующая конструкция выглядит так:

```
<META http-equiv="Content-Type" content="text/HTML;
↳charset=ISO-8858-5">
```

В приведенном примере значение переменной состоит из двух частей, разделенных знаком точки с запятой. Первая часть говорит о том, что данный документ — это обычный текст с тэгами HTML, а вторая часть указывает используемую кодировку. При этом применение слова charset является обязательным. После знака равенства приводится само название кодировки. В примере использована стандартная кодировка, утвержденная Международной организацией по стандартизации (ISO), с поддержкой кириллицы. Вместо нее можно применять стандартную кириллическую кодировку Windows или КОИ-8.

К сожалению, браузеры не могут обычным способом отображать некоторые символы, встречающиеся в тексте. Если браузер в тексте встретит знак неравенства "меньше", то он интерпретирует его как открывающую скобку для тэга. А так как стандартного тэга за этим знаком не последует, то некоторая часть текста будет просто проигнорирована и не отображена. Более того, некоторые буквы из алфавитов языков на основе латиницы отсутствуют на клавиатуре, и их трудно вставить в текст содержимого Web-страницы. Выход был найден.

Вместо самих символов в текст подставляются последовательности, которые можно правильно интерпретировать. Знак неравенства "меньше", он же открывающая угловая скобка, заменяется последовательностью "<". Вся последовательность заключается в кавычки, начинается она со знака амперсанда, а заканчивается знаком точки с запятой. Подобные последовательности перекочевали в язык следующего поколения — XML (eXtensible Marckup Language), и получили наименование "entities", что в русскоязычной литературе переводят как "сущности". Перевод, конечно, правильный, но, к сожалению, ничуть не разъясняет сути дела. Проще и, наверное, правильней называть эти сущности символьными подстановками. Чаще всего используют четыре символьные подстановки, которые приведены в табл. 1.1.

Таблица 1.1. *Символьные подстановки*

<	Знак неравенства "меньше" "<"
>	Знак неравенства "больше" ">"

Таблица 1.1 *(окончание)*

`&`	Знак амперсанда "&"
`&quot;`	Знак кавычек

Этими четырьмя сочетаниями список символьных подстановок не заканчивается. Оставшаяся часть списка приведена в *приложении 1.*

Подстановки бывают не только символьные. Мы можем с помощью подстановки поместить в текст любой символ из текущей кодировки, если нам известен его числовой код. Для этого применяется конструкция "`&#числовой_код;`". В подобном формате численных подстановок указывают десятичную запись числового кода символа. Если используется шестнадцатеричная запись кода, то подстановка принимает следующий вид: "`&#xчисловой_код;`". После знака решетки добавляется латинский символ "икс".

Цвета и единицы измерения

В коде HTML-документа нам всегда придется указывать размеры тех или иных объектов оформления Web-страницы, а также цветовые свойства этих объектов. В HTML предусмотрены стандартные правила для обозначения цветов и единиц измерения. Начнем мы с цветовых обозначений.

Существует два способа задания цвета. Чаще всего используется способ с указанием RGB-кода требуемого цвета. Как известно, любой цвет разлагается на три основных: красный, зеленый и синий. Браузеры позволяют нам отображать более шестнадцати миллионов цветов. Такое многообразие обеспечивается за счет того, что доля каждого из трех основных цветов может меняться от нуля до двухсот пятидесяти пяти. Любой цвет задается сочетанием трех чисел, каждое из которых определяет долю одного из трех основных цветов. Для удобства обработки в HTML цвет задается в виде группы из шести шестнадцатеричных цифр в следующей форме:

```
color="#FF0000"
```

Перед последовательностью шестнадцатеричных цифр ставится знак решетки. Порядок чисел, указывающих доли основных цветов, должен строго соблюдаться. Сначала красный, затем зеленый, и в самом конце — синий. Легко догадаться, что в примере мы установили красный цвет.

Есть и альтернативный вариант установки цвета. Для шестнадцати наиболее популярных цветов были установлены символьные обозначения, приведенные в табл. 1.2.

Таблица 1.2. *Цветовые обозначения*

Цвет	Шестнадцатеричный код	Буквенное обозначение
Черный	#000000	Black
Серебряный	#C0C0C0	Silver
Серый	#808080	Gray
Белый	#FFFFFF	White
Темно-красный	#800000	Maroon
Красный	#FF0000	Red
Пурпурный	#800080	Purple
Малиновый	#FF00FF	Fuchsia
Зеленый	#008000	Green
Ярко-зеленый	#00FF00	Lime
Оливковый	#808000	Olive
Желтый	#FFFF00	Yellow
Темно-синий	#000080	Navy
Голубой	#0000FF	Blue
Темная морская волна	#008080	Teal
Морская волна	#00FFFF	Aqua

С учетом этих обозначений, наш пример с красным цветом мы могли бы написать таким образом:

```
color="Red"
```

Теперь переходим к рассмотрению единиц измерения длины. Согласно спецификации языка HTML, мы можем указывать размеры каких-либо объектов оформления Web-страниц только двумя способами. Либо задать их размер в пикселах, либо в процентах от размера "родительского" объекта, внутри которого находится описываемый элемент оформления. Если мы на Web-странице размещаем таблицу и указываем, что ее ширина должна составлять пятьдесят процентов, то имеется в виду, что это пятьдесят процентов от ширины окна просмотра браузера. Ширина ячейки таблицы будет рассчитываться в процентах от общей ширины таблицы, в которой находится эта ячейка. Если пользователь изменит размеры окна браузера, соответствующим образом изменится и компоновка содержимого Web-страницы. При создании Web-страницы всегда следует изыскивать такие способы размещения содержимого, чтобы их компоновка не менялась кардинально при изменении размеров окна браузера. Кроме того, следует учитывать, что мони-

торы удаленных пользователей имеют различные разрешения. О том, как определить, какое именно разрешение монитора установлено у посетителя Web-страницы, мы узнаем в третьей главе, а пока вернемся к единицам измерения размеров объектов Web-страниц.

Для того чтобы указать размер некоего элемента в пикселах, достаточно в качестве значения соответствующего параметра задать необходимое число. Если мы захотим установить ширину некоего объекта равной тридцати пикселам, следует использовать следующую конструкцию:

```
width="30"
```

Если ширина должна составлять тридцать процентов от "родительского" объекта, то мы запишем такой код:

```
width="30%"
```

Еще раз обращаю внимание на то, что все значения параметров обрамляются двойными кавычками.

Помимо приведенных двух способов указания размеров, есть и еще один. Это нечто среднее между процентным соотношением и прямым определением размера в пикселах. Мы можем указать размер, кратный некоторому количеству пикселов. Например, есть у нас таблица, состоящая из трех строк. Высота таблицы задана, причем, неважно, каким способом. Если мы хотим, чтобы высота каждой строки была кратна тридцати пикселам, то в каждый тэг, создающий одну из этих строк, мы должны внести следующий фрагмент кода:

```
height="3*"
```

Признаком задания кратного размера служит символ "звездочка" (*). При расчете коэффициента кратности число, стоящее слева от звездочки, увеличивается в десять раз. Браузер же попытается отобразить такие объекты с максимально возможным размером. Если таблица имеет высоту сто восемьдесят пикселов, то высота каждой строки будет шестьдесят пикселов. То же самое случится, если мы создадим таблицу высотой в двести пикселов. Двадцать лишних пикселов просто пропадут. Если необходимо, чтобы наши строки заняли равную высоту, следует использовать следующий вариант задания параметра:

```
height="*"
```

Такой способ задания размера действует по умолчанию. Если для группы одинаковых объектов размеры не указаны, они размещаются максимально равномерно в пространстве "родительского" объекта.

Это все, что мы можем сказать о единицах измерения, принятых в HTML. На самом деле, мы имеем возможность задавать абсолютные размеры элементов оформления Web-страниц не только в пикселах. Но для этого необходимо использовать возможности технологии стилевого оформления CSS,

которые мы рассмотрим в следующей главе. Теперь пора переходить к рассмотрению возможностей отображения текста.

Оформление текста

Как известно, основное наполнение Web-страниц — это текст за редким исключением специальных сайтов — галерей. Неудивительно, что в HTML введено столько различных средств управления отображением текста.

Для того чтобы в окне просмотра браузера отобразить текстовую строку, никаких тэгов применять не требуется. Достаточно написать текст. Но когда нужно разбить его хотя бы на абзацы, тут уже без тэгов не обойтись. В различных компьютерных системах используются разные символы для разбиения текста на абзацы, а HTML-документы должны отображаться по возможности одинаково на любых компьютерных платформах, поэтому и пришлось ввести тэги, обозначающие абзацы.

В начале каждого абзаца ставится тэг `<p>`, а в конце — закрывающий тэг `</p>`. При этом тэг обладает некоторыми параметрами. В их число входят уже известные нам общие параметры идентификации `class` и `id`, параметр стилевого оформления `style`, который нам предстоит рассмотреть во второй главе, и параметр выравнивания `align`. О последнем параметре нам следует поговорить несколько поподробнее.

В HTML термин "выравнивание" означает как горизонтальное, так и вертикальное позиционирование элемента. Когда речь идет об абзацах текста, имеет смысл говорить только о горизонтальном выравнивании, или, как его еще называют, "выключке".

Выключка позволяет прижимать абзац к левому или правому краю окна просмотра браузера, центрировать его или растягивать слова таким образом, чтобы текст равномерно занимал всю ширину отведенного ему места. Для этих целей используются значения `left`, `right`, `center` и `justify`, соответственно. Рассмотрим их использование на примере еще одного HTML-документа.

Листинг 1.4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Горизонтальное выравнивание абзацев</title>
  </head>
  <body>
  <p align="left">Абзац, прижатый к левому краю</p>
```

```
  <p align="right">Абзац, прижатый к правому краю</p>
  <p align="center">Центрированный абзац</p>
  <p align="justify">Абзац, растянутый по ширине</p>
  </body>
</html>
```

Результат просмотра файла с таким кодом браузером Internet Explorer показан на рис. 1.4. Кстати, для отображения этого файла применялась одна из устаревших версий браузера Internet Explorer, которая не обрабатывала растяжение абзаца по ширине поля просмотра. В подобных случаях, браузеры просто игнорируют неизвестные тэги или значения параметров и используют стандартный вариант отображения.

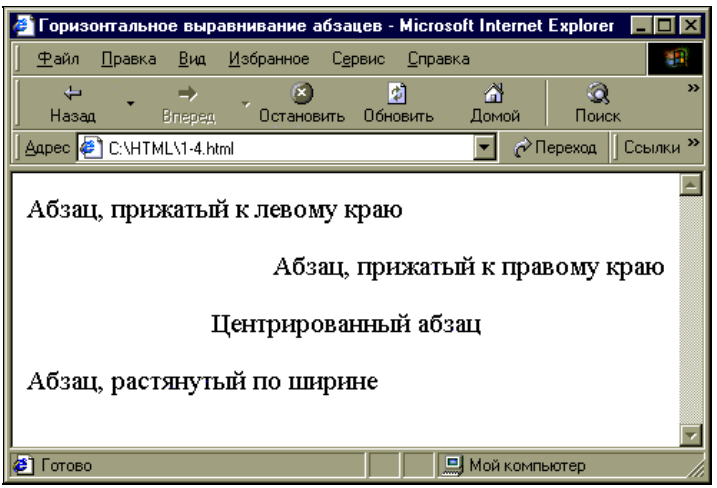


Рис. 1.4. Окно браузера с результатом отображения файла, приведенного в листинге 1.4

Иногда для того, чтобы увеличить расстояние между абзацами, создатели Web-страниц пытаются использовать пустые абзацы, т. е. абзацы, у которых между начальным и конечным тэгом ничего нет. Согласно спецификациям, браузеры должны игнорировать подобные конструкции, поэтому для разделения абзацев или принудительного обрыва строки внутри одного абзаца следует применять тэг <br>. Это директивный тэг. Он обозначает то место, где надо перенести текст на другую строку. Пример использования этого тэга для достижения обеих целей приведен в листинге 1.5.

Листинг 1.5

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
```

```
<html>
  <head>
  <title> Принудительный разрыв строк</title>
  </head>
  <body>
  <p>Абзац, который мы <br>принудительно разорвали</p>
  <p>Следующий абзац</p>
  <br>
  <p>Абзац после принудительного разрыва</p>
  </body>
</html>
```

Как выглядит этот файл при просмотре его с помощью браузера, показано на рис. 1.5.

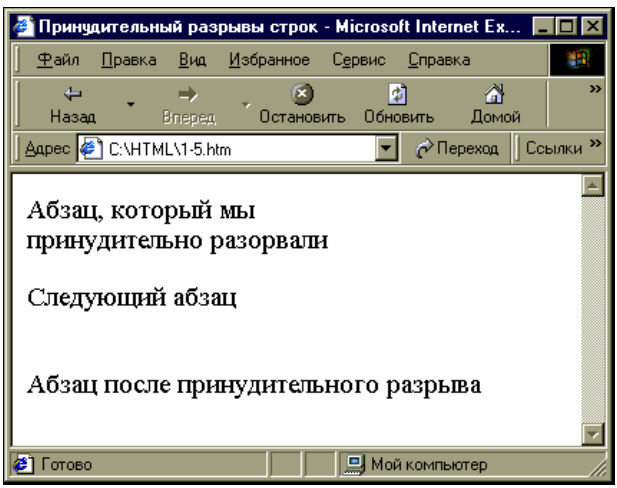


Рис. 1.5. Окно браузера с результатом отображения файла, приведенного в листинге 1.5

Тэг `<br>` помимо обычных параметров идентификации обладает еще параметром `clear`, который применяется для более точного выравнивания текста, когда тот обтекает какой-либо объект, например графическое изображение, внедренное в состав Web-страницы. В качестве значения этого параметра может использоваться одно из четырех предопределенных ключевых слов: `none`, `left`, `right`, `all`. Значение `none` применяется по умолчанию, и указывает, что следующая строка начнется с того места, где она начиналась бы и без использования данного параметра. Значение `left` говорит о том, что следующая строка начнется у левого поля объекта, обтекаемого текстом. Если же необходимо использовать правое поле для этих целей, то следует задать значение `right`. Значение `all` указывает браузеру, что воспользоваться

можно как левым, так и правым полем объекта, лишь бы текст был максимально компактно отображен.

Теперь перейдем к рассмотрению шрифтового оформления текста. В любом месте абзаца мы можем использовать тэг `<font>` с набором параметров, которые и будут определять внешний вид шрифта, применяемого для отображения текста, находящегося после этого тэга. Прекращение действия тэга `<font>` задается тэгом `</font>`.

Тэг `<font>` обладает следующими, присущими именно ему параметрами: `size`, использующийся для указания размера применяемого шрифта, `color` — для установки цвета символов шрифта, и `face`, указывающий, какой именно шрифт будет применяться для отображения текста.

Как мы только что говорили, параметр `size` применяется для указания размера символов используемого шрифта. Значениями этого параметра могут быть числа от единицы до семи. Они обозначают некий относительный размер символов. Дело в том, что в HTML нельзя установить абсолютный размер символов в пунктах, как мы это привыкли делать в обычных офисных приложениях. Пользователь будет просматривать Web-страницу на своем компьютере, и нам заранее неизвестно, какие шрифты могут быть у него установлены и какие их размеры доступны. Мы можем лишь указать относительный размер шрифта, а браузер пользователя сам подберет максимально подходящий размер.

В качестве значения параметра `size` мы можем задать изменение размера шрифта. Например, для того, чтобы увеличить размер шрифта на один уровень, следует использовать такую конструкцию:

```
<font size="+1">
```

Для уменьшения размера символов на два уровня применяется следующий код:

```
<font size="-2">
```

Для использования подобных конструкций необходимо отталкиваться от некоего базового размера. Для установки такого размера применяется тэг `<basefont>` с все тем же параметром `size`. Если же этот тэг не использовать, базовым размером символов будет третий уровень. Приведем пример использования этих тэгов в листинге 1.6.

Листинг 1.6

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Размер символов</title>
  </head>
```

```
  <body>
  <p><font size="7">Седьмой размер</font></p>
  <p><font size="6">Шестой размер</font></p>
  <p><font size="5">Пятый размер</font></p>
  <p><font size="4">Четвертый размер</font></p>
  <p><font size="3">Третий размер</font></p>
  <p><font size="2">Второй размер</font></p>
  <p><font size="1">Первый размер</font></p>
  <p><basefont size=2><font size="+2">Смещение размера</font></p>
  </body>
</html>
```

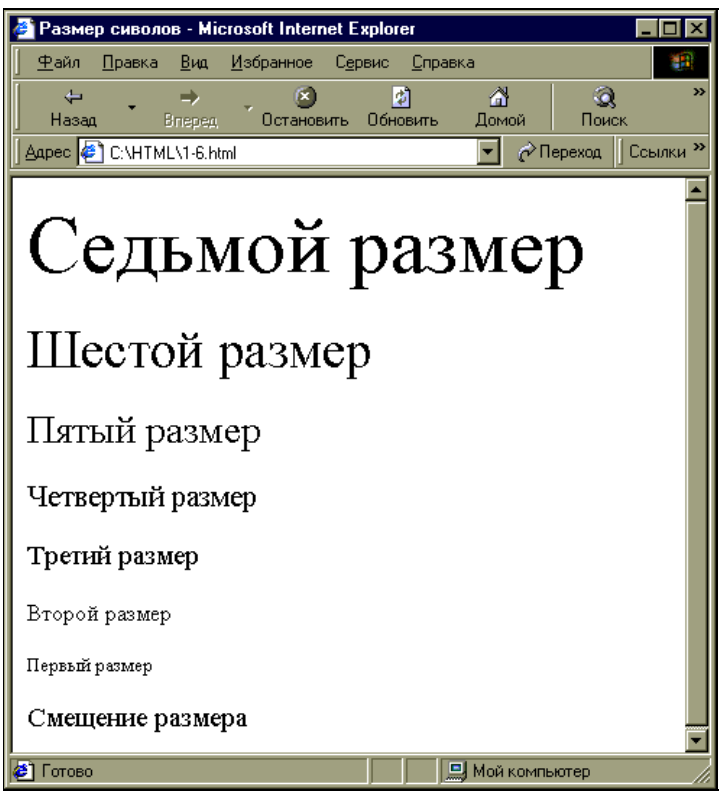


Рис. 1.6. Окно браузера с результатом отображения файла, приведенного в листинге 1.6

Мы рассмотрели применение только одного атрибута тэга `<font>`. На очереди — процедура установки цвета символов применяемого шрифта. Мы уже знаем, что для этого предназначен параметр `color`. О том, как именно записывать обозначение того или иного цвета, говорилось в предыдущем разде-

ле. Нам осталось лишь привести пример. Для установки зеленого цвета символов применяемого шрифта используется следующая конструкция:

```
<font color="green">
```

Последний параметр `face` позволяет устанавливать вид применяемого шрифта. Мы можем указать, что текст надо выводить с помощью шрифта Times New Roman или Copperplate Gothic. Однако следует понимать, что браузер пользователя будет отображать текст посредством шрифтов, установленных в операционной системе удаленного пользователя, и если мы применим некий редкий шрифт для придания большей выразительности текстовому наполнению Web-страницы, то его может не оказаться в системе пользователя. В этом случае браузер будет пользоваться собственными правилами шрифтового оформления. В каждом браузере есть раздел настроек, в котором пользователь указывает, какие именно шрифты применять для текстового содержимого загружаемых Web-страниц. В качестве значения параметра `face` часто применяется список из названий шрифтов, разделенных запятой. Браузер пытается отыскать их в своей системе в том порядке, в котором они перечислены, и первый найденный шрифт используется для отображения текста. Теперь, когда мы знаем правила задания всех параметров тэга `<font>`, приведем пример их общего использования.

```
<font size=4 color="black" face="Courier New, Arial Black">
```

В этом тэге мы объявляем, что текст, расположенный после него, нужно отображать четвертым размером и символами черного цвета, а применяться должен шрифт Courier New или, если он в системе не установлен, Arial Black.

Создавая текстовые документы, мы не ограничиваемся указанием размера, цвета и вида шрифта. Есть ведь еще различные начертания одного шрифта. Мы можем делать символы курсивными, полужирными, подчеркнутыми и перечеркнутыми. HTML предоставляет нам эти возможности. Итак, все по порядку.

- Тэг `<b>` с закрывающим тэгом `</b>` задает полужирное начертание символов выбранного шрифта.
- Тэги `<i>` и `</i>` обрамляют курсивные символы.
- Тэги `<u>` и `</u>` заставляют браузер подчеркивать текст, расположенный между ними.
- Тэг `<strike>` с закрывающим тэгом `</strike>` описывает перечеркнутый текст.
- Тэги `<tt>` и `</tt>` обрамляют символы, которые браузер отображает моноширинным шрифтом.
- Тэги `<small>` и `</small>` указывают, что текст, заключенный между ними, необходимо отображать шрифтом уменьшенного размера.

- Тэги <big> и </big>, наоборот, увеличивают размер символов, находящихся между ними.

Естественно, без примера нам никак не обойтись.

Листинг 1.7

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title> Начертания символов</title>
  </head>
  <body>
  <p><font face="Times New Roman">Текст бывает <b>полужирным</b> или
<i>курсивным</i><br>
  А может быть одновременно и <b><i>полужирным и курсивным</i></b>.</p>
  <p>Бывает <u>подчеркнутым</u> и <strike>перечеркнутым</strike>.</p>
  <p>Или <tt>моноширинным</tt>.</p>
  <p>Мы можем <big>увеличивать</big> и <small>уменьшать</small> размер
символов.</p>
  </body>
</html>
```

Как видно из примера, мы можем сочетать различные начертания, но следует быть аккуратным при расстановке тэгов: открывающий и закрывающий тэги должны оба вкладываться как в футляр в другие тэги. Пример этого можно увидеть в десятой строке листинга. Как выглядит данный HTML-документ при просмотре его с помощью браузера, показано на рис. 1.7.

Мы рассмотрели способы, с помощью которых задаются различные параметры отображения текста, однако в HTML есть несколько базовых способов отображения текста, которые не обязательно задавать целым набором тэгов. Для них выделены собственные тэги, т. е. определены некоторые категории текста, которые отображаются предопределенным способом. Эти категории чаще всего применяются в научной и компьютерной отрасли. Перечислим их.

- Выделение термина в тексте производится парой тэгов и .
- "Усиленное" выделение, призванное привлечь внимание, создается с помощью тэга и его закрывающего двойника .
- Тэги <cite> и </cite> указывают на то, что текст, расположенный между ними, — цитата.
- Определение некоего термина выделяется тэгами <dfn> и </dfn>.
- Пара тэгов <code> и </code> применяется для выделения исходного кода на каком-либо языке программирования.

- Текстовая информация, выводимая программой, оформляется с помощью тэгов <samp> и </samp>.
- Текст, вводимый пользователем, обозначается тэгами <kbd> и </kbd>.
- Переменные в исходном коде программ выделяются с помощью пары тэгов <var> и </var>.
- Тэги <abbr> и </abbr> обозначают аббревиатуры.
- Устоявшиеся буквосочетания, не являющиеся аббревиатурами, т. е. акронимы, выделяются тэгами <acronym> и </acronym>.

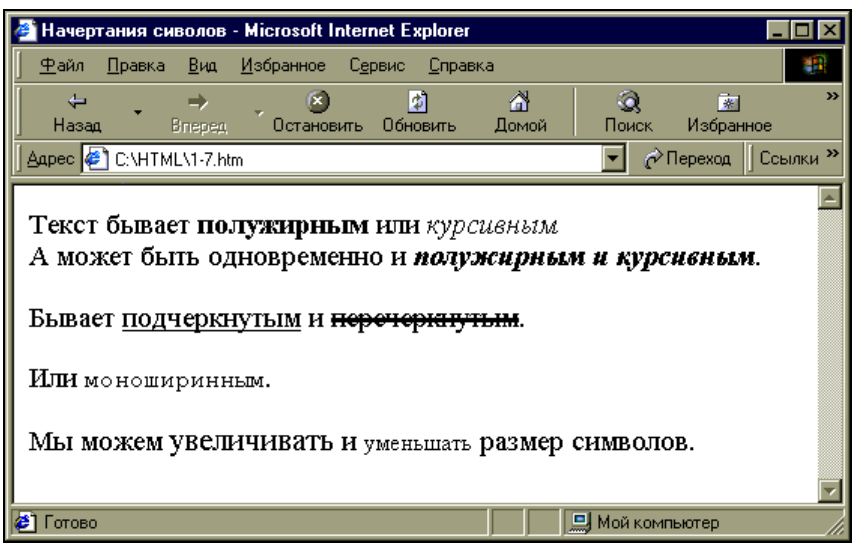


Рис. 1.7. Окно браузера с результатом отображения файла, приведенного в листинге 1.7

Однако знать наименования тэгов мало, надо еще отчетливо себе представлять, как именно они оформляют текст. Здесь без примера никак не обойтись.

Листинг 1.8

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Стандартные варианты отображения</title>
  </head>
  <body>
  <p><font face="Times New Roman">Это обычный текст</p>
```

```
  <p>Это <em>выделение</em>. И <strong>более заметное
↳выделение</strong></p>
  <p>Это <cite>цитата</cite>. А это <dfn>определение</dfn>.</p>
  <p>Мы можем писать <code>код программы</code>, текст, <kbd>вводимый
↳пользователем</kbd>,
  <samp>выводимый текст</samp> и <var>переменные</var> программы.</p>
  <p>Так отображаются <abbr>аббревиатуры</abbr>. А так —
↳<acronym>акронимы</acronym>.</p>
  </body>
</html>
```

Результат отображения подобного HTML-документа показан на рис. 1.8.

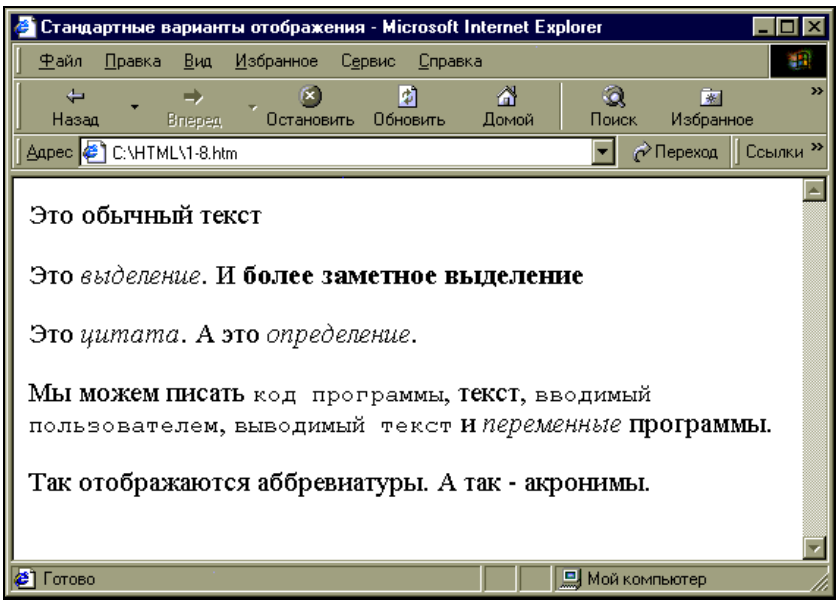


Рис. 1.8. Окно браузера с результатом отображения файла, приведенного в листинге 1.8

Очень часто возникает необходимость разместить на Web-странице текст, заранее приготовленный с помощью какого-либо текстового редактора, который уже самостоятельно отформатировал текст. В этом случае формат текста, его переносы и размещение напрямую зависят от длины строки в символах, которая была установлена в этом текстовом редакторе. Если текст разместить на Web-странице с помощью обычных средств, это форматирование будет нарушено. Для таких предварительно отформатированных фрагментов был введен специальный тэг. Этот тэг `<pre>` обладает параметром `width`, в качестве значения которого указывается длина строки в символах. Желательно перед применением этого тэга принудительно установить

именно тот шрифт, который использовался при форматировании текста. Итак, для вставки предварительно отформатированного текста мы можем применить следующий фрагмент кода:

```
<pre width=60>Текст...</pre>
```

В данном примере мы указываем, что строка для отображения текста имеет длину шестьдесят символов. При этом в предварительно отформатированном тексте не действуют правила "сворачивания пробелов". Когда браузер встречает в обычном тексте Web-страницы несколько идущих подряд пробелов, он их сворачивает в один пробел. Заранее отформатированный текст подобному преобразованию не подвергается, что и позволяет сохранять его форматирование, которое в простейших текстовых редакторах часто осуществляется пробельными символами. В HTML символ табуляции тоже считается пробельным символом.

HTML предоставляет возможность с помощью тэгов отображать верхние кавычки, которыми обычно выделяется прямая речь и цитаты. Для цитат предусмотрен тэг `<blockquote>` с его завершающим двойником `</blockquote>`. Для обрамления прямой речи обычно используются тэги `<q>` и `</q>`. При этом тэг `<blockquote>` обладает параметром `cite`, в качестве значения которого используется сетевой адрес, называемый также URL того интернет-ресурса, на котором изначально находился цитируемый текст. Вместе с тэгом `<q>` обычно применяется параметр `lang`, задающий кодовое обозначение языка, согласно правилам пунктуации которого и ставятся ограничивающие кавычки. Не секрет ведь, что в разных языках принято использовать различные символы кавычек.

Несколько слов о проблеме переносов. Обычно создатели Web-страниц даже не задумываются об этой проблеме, и браузеры не занимаются переносом слов. Если слово целиком не помещается в строке, оно просто переносится на другую строку. Но ведь это не единственно правильный способ отображения текста. Может возникнуть ситуация, когда понадобится отображать текст, пользуясь переносами слов. В HTML предусмотрено два вида переносов — явный и так называемый "мягкий". Явный перенос задается с помощью символа короткого тире, заменяемого обычно численными текстовыми подстановками "`&#45;`" или "`&#x2D;`". Такой перенос неудобен тем, что при изменении размеров окна просмотра браузера может измениться и длина строки, а знак переноса все равно останется виден, даже если он будет находиться в середине строки. Чаще пользуются "мягкими" переносами. Они создаются с помощью текстовой подстановки "`&shy;`". При этом символы мягких переносов в тексте не видны, и лишь в том случае, когда какое-либо слово, в которое были вставлены эти символы, не помещается целиком в строке, оно переносится с разбиением на слоги, согласно вставленным "мягким" переносам. Лишь один из символов мягкого переноса в этом слове становится виден.

Кроме того, HTML позволяет отображать верхние и нижние индексы. Для создания верхнего индекса используется пара тэгов ^и, а нижний индекс должен быть обрамлен тэгами _и. Рассмотрим пример использования этих тэгов.

Листинг 1.9

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Верхние и нижние индексы</title>
  <body>
  <p>Вода это H<sub>2</sub>O</p>
  <p>Соотношение массы и энергии -   E = mc<sup>2</sup></p>
  </body>
</html>
```

Как именно отображает индексы браузер, показано на рис. 1.9.

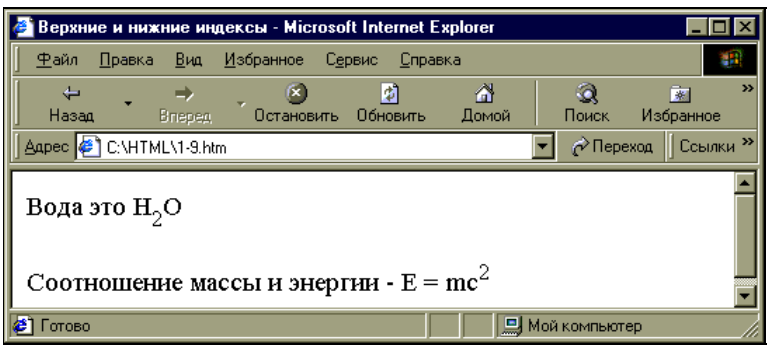


Рис. 1.9. Окно браузера с результатом отображения файла, приведенного в листинге 1.9

К текстовому оформлению можно отнести горизонтальные линии, которыми часто отделяют значимые части текстового содержимого Web-страниц. Линия вставляется в текст HTML-документа с помощью тэга <hr>. Тэг обладает несколькими параметрами, которые позволяют достаточно детально определять внешний вид горизонтальной линии.

Параметр align задает горизонтальное выравнивание линии. Он может принимать одно из трех предустановленных значений — left, right и center, которые прижимают горизонтальную линию к левому или правому краю окна просмотра, или центрируют ее, соответственно. По умолчанию используется значение center. Если в состав тэга <hr> входит параметр noshade, то отображаемая горизонтальная линия не будет иметь тени. Последний пара-

метр `width` устанавливает длину горизонтальной линии. По умолчанию используется значение "`100%`". Высота линии в пикселах задается с помощью параметра `size`. Рассмотрим использование всех этих параметров на примере.

Листинг 1.10

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Горизонтальные линии</title>
  <body>
  <p>Это обычная линия, отображаемая по умолчанию <hr></p>
  <p>Это укороченная линия, прижатая влево <hr align="left" width="70%"
size=5></p>
  <p>Это укороченная линия, расположенная по центру <hr align="center"
width="70%" size=5></p>
  <p>Это укороченная линия, прижатая вправо <hr align="right" width="70%"
size=5></p>
  <p>Это утолщенная линия без тени <hr align="center" width="70%" noshade
size=10></p>
  </body>
</html>
```

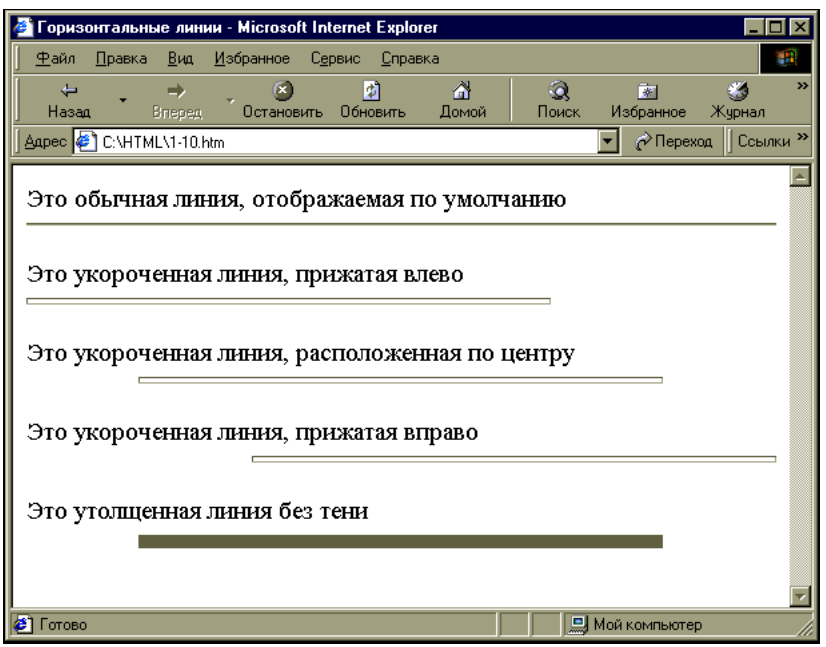


Рис. 1.10. Окно браузера с результатом отображения файла, приведенного в листинге 1.10

Как выглядит подобный HTML-документ при просмотре его с помощью браузера, показано на рис. 1.10.

На этом, пожалуй, можно закончить рассмотрение возможностей оформления текста, присущих HTML.

Графика и мультимедиа

Графический объект, несомненно, — второй по значимости после текста компонент наполнения Web-страниц. К графическим объектам относят различного рода рисунки, фотографии и видеоклипы. Часто используют звуковое сопровождение отображения Web-страниц.

Начнем мы с размещения графических изображений. Браузеры в состоянии отображать только три вида графических файлов: файлы форматов GIF, JPEG и PNG. Файлы формата GIF позволяют создавать анимированные изображения. JPEG-файлы обычно применяются для сохранения фотографических изображений. Недавно появившийся формат PNG обеспечивает хорошее качество изображения и маленький объем графического файла. После того как изображение было упаковано в графический файл, его необходимо каким-либо образом внедрить в состав Web-страницы.

Для этого применяется тэг `<img>` со множеством параметров. Этот тэг не имеет закрывающего двойника, т. к. он не задает какую-либо область действия правила отображения, а лишь внедряет в содержимое Web-страницы графическое изображение. Графическое изображение может быть еще и гиперссылкой, или даже скрывать несколько гиперссылок, но о гиперссылках мы поговорим в следующем разделе этой главы, а пока разберемся с правилами применения тэга `<img>`.

Основным и обязательным атрибутом тэга `<img>` является параметр `src`. В качестве значения этого атрибута используется адрес вставляемого графического файла, или, если быть точным, его URL. Если графический файл находится на том же Web-сервере, то достаточно записать полное наименование файла, включая путь к нему по вложенным каталогам. Допустим, что каталог `images` с рисунками расположен в той же папке, что и HTML-файлы с Web-страницами, тогда тэг вставки графического изображения приобретет следующий вид:

```
<img src="images/pict1.gif">
```

В этом примере мы ссылаемся на рисунок формата GIF, находящийся в файле с именем pict1.gif, который расположен в каталоге с именем `images`. Следует обратить внимание на то, что в тэге используется прямой слэш (наклонная черта) в отличие от обратного, применяемого для указания путей к файлам в операционных системам семейств DOS и Windows. Дело в том, что изначально Web-серверы базировались на операционной системе

Unix, которая и поддерживает файловую систему с подобными слэшами. Сейчас абсолютно неважно, какая операционная система поддерживает сервер с Web-сайтом, все пути записываются одинаковым способом и правильно обрабатываются программным обеспечением сервера.

Первые появившиеся браузеры WWW отображали только текстовую информацию, никакая графика не поддерживалась. Сейчас такие обозреватели практически не встречаются, но каждый браузер имеет возможность отключения загрузки графики, поэтому всегда следует использовать альтернативное текстовое представление рисунка. Попросту, необходимо приготовить текст, который будет отображаться вместо рисунка, если тот не может быть по каким-либо причинам загружен браузером. Этот текст добавляется к тэгу `<img>` с помощью параметра `alt`, значением которого является текстовая строка. То есть, получится приблизительно следующая конструкция:

```
<img src="images/pict1.gif" alt="Это я на отдыхе">
```

В том случае, если графическое изображение все-таки показывается браузером, текст альтернативного текстового представления отображается в виде *хинта*, короткой текстовой подсказки, когда пользователь наводит курсор мыши на это графическое изображение.

Существует и более развернутый вариант создания подобных текстовых подсказок. Параметр `longdesc` задает адрес интернет-ресурса, на котором находится полное описание данного графического изображения. В качестве значения этого параметра указывается URL ресурса с описанием изображения.

Параметр `name` позволяет задавать уникальное имя изображения, которое идентифицирует описываемый элемент оформления Web-страницы. Этот параметр оставлен для обратной совместимости с предыдущими версиями стандарта HTML. В современных версиях языка для этих целей все тэги используют параметр `id`.

По умолчанию графическое изображение показывается именно в том виде, в каком оно было создано, с сохранением размеров по вертикали и горизонтали. Однако мы имеем возможность явно задавать размеры рисунка по своему усмотрению. Для этого используются параметры `height` и `width`. Как задавать размеры в пикселах или процентах, мы уже знаем. Необходимо отметить лишь, что браузеры стремятся сохранять пропорции рисунка, поэтому явное задание размеров, меняющее пропорции, может быть проигнорировано браузером, и он выберет такие размеры, которые были бы максимально близки к указанным пользователем, но не нарушали пропорций изображения. Обычно для Web-страниц готовят рисунки тех размеров, которые будут применяться при их отображении в составе Web-страниц. Если одно изображение должно выводиться несколько раз с различными размерами, то проще приготовить несколько графических файлов, чем отдавать свои рисунки для бесконтрольного отображения браузеру, который сможет нарушить всю верстку Web-страниц.

Существуют параметры hspace и vspace, позволяющие указывать величину чистого пространства, которое будет отделять графическое изображение от окружающих его других элементов оформления Web-страницы, другими словами, задавать отступ рисунка. Параметр hspace устанавливает отступ по горизонтали в пикселах, а vspace — по вертикали. Обратите внимание, эти параметры могут задаваться только числовыми значениями, указывающими расстояния в пикселах. Нулевого значения для этих параметров не предусмотрено, но обычно каждый браузер использует малое ненулевое значение.

С помощью параметра border мы можем устанавливать толщину границы, окружающей рисунок. Как обычно, значением параметра является число, указывающее толщину в пикселах. По умолчанию используется нулевое значение, делающее границу невидимой.

Необходимо упомянуть о выравнивании графического объекта относительно обтекающего его текста. Для этого используется параметр align. Его значением может быть одно ключевое слово из предопределенного набора. Значения bottom, middle и top применяются для позиционирования первой строки текста, обтекающего рисунок по вертикали. Значение top смещает ее вверх, bottom — вниз, а middle позволяет центрировать строку по вертикали. Для выравнивания по горизонтали графического изображения применяются значения left и right. Первое значение left, как нетрудно догадаться, смещает рисунок к левому края блока, в котором тот отображается, а right — к правому.

Теперь пришло время рассмотреть на примерах, как мы можем позиционировать рисунок и комбинировать его с текстом, который должен обтекать заданное графическое изображение.

Листинг 1.11

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Рисунки и текст</title>
  <body>
  <p>Это текст, который обтекает рисунок.<img src="redhood.gif">
  Это текст, который обтекает рисунок. Это текст, который обтекает
↳рисунок.
  Это текст, который обтекает рисунок.</p>
  </body>
</html>
```

Внешний вид этого HTML-документа при отображении его с помощью браузера показан на рис. 1.11.

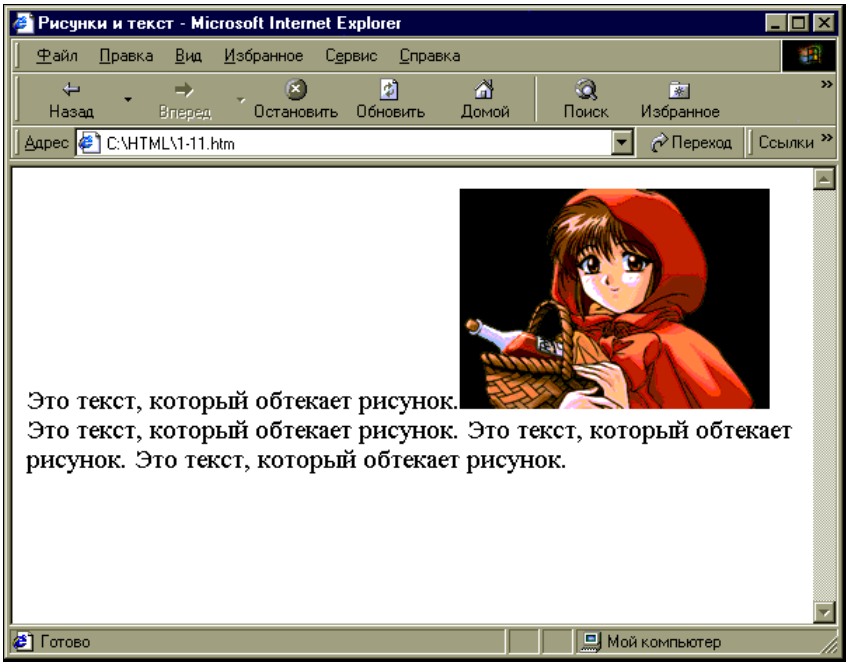


Рис. 1.11. Окно браузера с результатом отображения файла, приведенного в листинге 1.11

В этом примере мы использовали чистый тэг вставки изображения без каких-либо дополнительных параметров. Как мы можем видеть, изображение вставляется сразу после первого предложения, там, где мы разместили тэг `<img>`. Если мы уменьшим размеры окна просмотра браузера по горизонтали так, чтобы первое предложение и рисунок не смогли поместиться на одной строке, то сначала будет отображено предложение, а уже под ним рисунок, прижатый к левому краю окна просмотра. Справа от него начнется отображение следующего за ним текста таким образом, что базовая линия строки совпадет с нижним краем рисунка.

Теперь добавим к тэгу отображения рисунка параметр горизонтального выравнивания. Получившийся код приведен в листинге 1.12.

Листинг 1.12

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Рисунки и текст</title>
```

```
  <body>
  <p>Это текст, который обтекает рисунок.<img src="redhood.gif"
↳align="left">
  Это текст, который обтекает рисунок. Это текст, который обтекает
↳рисунок.
  Это текст, который обтекает рисунок.</p>
  </body>
</html>
```

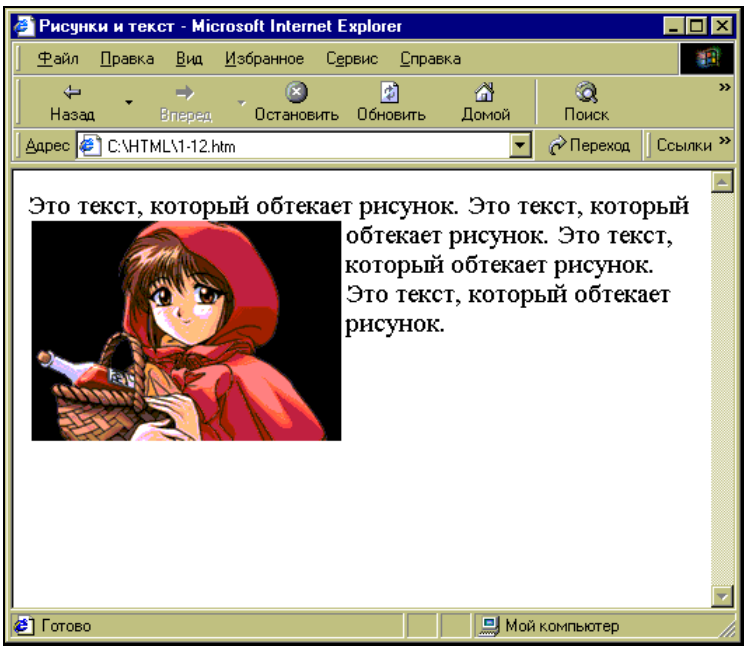


Рис. 1.12. Окно браузера с результатом отображения файла, приведенного в листинге 1.12

В этом случае сначала начинает отображаться текст, затем графическое изображение, прижатое к левому краю, согласно директиве, заданной с помощью параметра `align`, а справа от рисунка размещается остальной текст. Изображение не может занять первую строку, т. к. текст начинается раньше него. То же самое произойдет, если применить параметр `align` со значением `right`, но рисунок окажется прижатым к правому краю окна просмотра, а текст будет обтекать его с левой стороны. Вот и все изменения.

Теперь рассмотрим, как действует выравнивание по вертикали. Хотя параметр выравнивания вставляется в тэг рисунка, наибольшие изменения заметны в расположении текста, окружающего рисунок. Рассмотрим действие параметра вертикального выравнивания на примере.

Листинг 1.13

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Рисунки и текст</title>
  <body>
  <p>Это текст, который обтекает рисунок.<img src="redhood.gif"
align="middle">
  Это текст, который обтекает рисунок. Это текст, который обтекает
рисунок.
  Это текст, который обтекает рисунок.</p>
  </body>
</html>
```

Результат отображения этого HTML-документа показан на рис. 1.13.

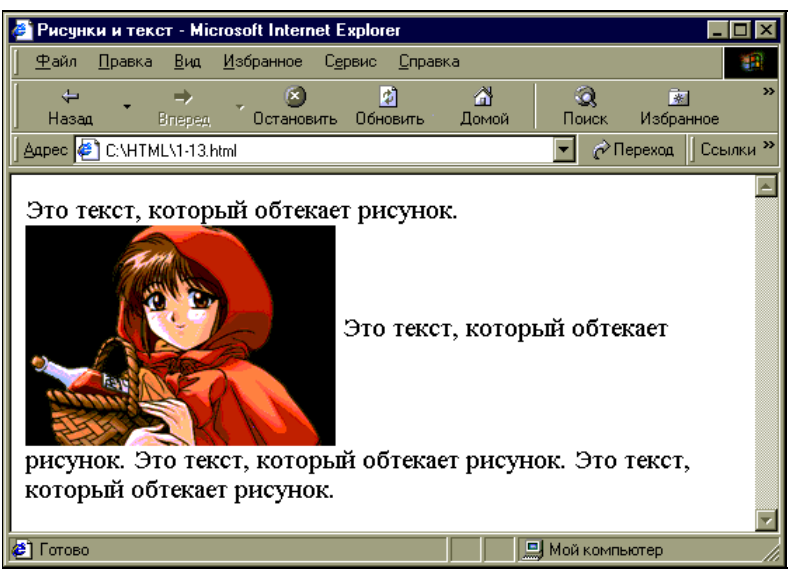


Рис. 1.13. Окно браузера с результатом отображения файла, приведенного в листинге 1.13

В данном примере мы установили вертикальное выравнивание посередине. Первая строка текста, находящегося после тэга вставки изображения, отображается по вертикали в центре свободного пространства справа от рисунка. Остальной текст располагается под рисунком. Если бы мы присвоили параметру `align` значение `top`, то первая строка появилась бы рядом с верхним обрезом рисунка. При использовании значения `bottom` первая строка

текста выводится рядом с нижней границей рисунка. Следует обратить внимание, что браузер распознает только один параметр `align`, т. е. мы можем указать либо вертикальное, либо горизонтальное выравнивание. Если же встроенных средств позиционирования и выравнивания рисунка не хватает, можно обратиться к процессу верстки с использованием таблиц. Применение таблиц мы рассмотрим в одном из следующих разделов этой главы.

У нас есть возможность использовать в оформлении Web-страниц видеоролики, но применять их следует аккуратно, т. к. файлы, содержащие эти видеоресурсы, обычно имеют достаточно большой объем. Для того чтобы удаленный пользователь мог просмотреть их в своем браузере, ему понадобится полностью загрузить этот файл на свою машину. У большинства пользователей Интернета из-за характеристик используемых каналов подобная процедура может занять достаточно много времени, а ведь никто из нас не любит ждать загрузки Web-страницы.

Браузеры обычно в состоянии воспроизводить видеофайлы форматов AVI, Real Video и Windows Media. Внедрение их в состав Web-страницы выполняется посредством все того же тэга `<img>`. Для указания местонахождения видеофайла используется параметр `dunscr`. По умолчанию, внедренный в Web-страницу видеоклип проигрывается один раз, сразу после полной загрузки страницы, но у нас есть возможность регулировать процесс проигрывания клипа или позволить самому пользователю управлять его воспроизведением.

Включив в состав тэга `<img>` параметр `start`, мы сможем явно указывать событие, после которого браузер должен будет воспроизвести загруженный видеоклип. Ключевые слова `mouseover` и `fileopen` используются как значения данного параметра. Первое из них указывает, что клип необходимо воспроизвести после того, как пользователь поместит курсор мыши на пространство, отведенное под видеовставку, а второе — что воспроизведение начнется сразу после полной загрузки HTML-файла. Впрочем, мы можем сочетать эти два варианта. В этом случае тэг вставки видеоролика будет выглядеть следующим образом:

```
<img dynsrc="movie.avi" start="mouseover, fileopen">
```

Мы можем указать, сколько раз необходимо воспроизвести видеоролик, установив значение параметра `loop` равным числу повторений видеоклипа. Если мы хотим, чтобы видео воспроизводилось постоянно, без какого-либо фиксированного количества повторений, то следует присвоить параметру `loop` значение `infinite`. Параметр `loopdelay` устанавливает временную задержку между повторами воспроизведения видеоклипа. Значением данного параметра должно быть число, указывающее промежуток времени в миллисекундах. Рассмотрим использование этих параметров на примере следующего фрагмента кода:

```
<img dynscr="movie.avi" loop="2" loopdelay="5000">
```

Здесь мы явно указываем, что видеоклип будет воспроизведен два раза сразу после загрузки Web-страницы с пятисекундной задержкой между воспроизведениями.

В данном случае мы сами управляем отображением видеофайла. Эту прерогативу можно передать и удаленному пользователю, который будет загружать Web-страницу. Для этого необходимо ввести в состав тэга `<img>` параметр `controls` без какого-либо значения. Наличие этого параметра в тэге явно указывает, что вместе с видеоклипом на Web-странице будут отображены и элементы управления, например кнопки перемотки, запуска и остановки воспроизведения, ползунок ускоренной перемотки и др.

Рассмотренных нами параметров вполне хватает для правильного внедрения видеофайлов в состав содержимого Web-страницы. Рассмотрим детальный пример (листинг 1.14 и рис. 1.14).

Листинг 1.14

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Видео</title>
  <body>
  <p>Обычный текст.<img dynsrc="intro.avi" start="mouseover" loop="2"
↳loopdelay="5000">
 </p>
  </body>
</html>
```

Процедура вставки видеофрагмента в тело Web-страницы — это частный случай вставки графики, следовательно, в тэге вставки видеофрагмента используются и все остальные параметры, применимые к тэгу `<img>`, такие как параметры выравнивания, явного указания размера и пр.

К мультимедийным возможностям относится и звуковое оформление Web-страницы, но звук мы можем использовать лишь как фон. Звуковой файл будет воспроизводиться на компьютере удаленного пользователя только после загрузки Web-страницы, если, конечно, в его компьютере установлена звуковая плата. Сначала на компьютер удаленного пользователя загружается HTML-файл с Web-страницей, а затем подгружаются все остальные файлы, используемые в оформлении этой Web-страницы, такие как графические файлы, видео- и аудиоклипы.

Итак, для того чтобы просмотр Web-страницы сопровождался звуковым воспроизведением, необходимо в код страницы вставить тэг `<bgsound>`. Тэг

помещается не в тело Web-страницы, а в его заголовок, т. е. между тэгами <head> и </head>. У этого тэга существует обязательный параметр src, в качестве значения которого используется URL подключаемого звукового файла. Необходимо отметить, что браузеры гарантированно распознают и воспроизводят аудиофайлы форматов MIDI и WAV. Воспроизведение иных форматов, таких как MP3 и Real Audio, возможно только при подключении дополнительных модулей к браузерам.

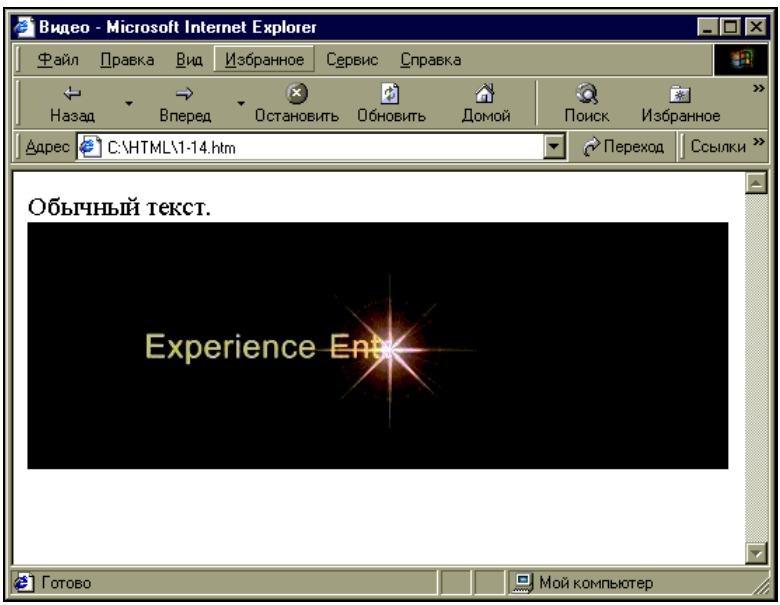


Рис. 1.14. Окно браузера с результатом отображения файла, приведенного в листинге 1.14

Как и в случае с видеоклипами, мы можем задавать количество воспроизведений звукового фала с помощью параметра loop. В качестве значения этого параметра указывается число повторений. Если необходимо воспроизводить аудиофрагмент постоянно, следует использовать значение "-1". Непрерывное проигрывание некоего аудиофайла при загрузке Web-страницы описывается следующей конструкцией:

```
<bgsound src="song.mid" loop="-1">
```

На этом мы заканчиваем рассмотрение вопросов использования мультимедийных и графических объектов в оформлении Web-страниц и переходим к следующему разделу этой главы.

Гиперссылки

Гиперссылки являются основным достоинством Web-страниц. Они — ядро Всемирной паутины, то, без чего она так и осталась бы еще одним средством отображения документов. Гиперссылки — видимый объект той технологии связи самых различных интернет-ресурсов, которая и создает уникальную интегрированность Сети.

Мы все прекрасно знаем, что если при просмотре Web-страницы навести курсор мыши на гиперссылку, внедренную в состав содержимого Web-страницы, то курсор примет форму кисти руки с вытянутым указательным пальцем, а единичный щелчок кнопкой мыши по этой гиперссылке заставит браузер отыскать в Сети тот ресурс, на который гиперссылка указывает, и загрузить его.

Гиперссылкой может стать любой фрагмент видимого содержимого Web-страницы: и текст, и графические изображения. Для этого применяется тэг `<a>` со своим закрывающим двойником `</a>`. Рассмотрим самый простой пример.

Листинг 1.15

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Гиперссылки</title>
  <body>
  <p>Обычный текст. <a href="other_doc.htm">Гиперссылка на другую Web-
↳страницу</a></p>
  </body>
</html>
```

Как легко увидеть на рис. 1.15, гиперссылка выделяется цветом, чтобы ее можно было отличить от обычного текста. Цвет выделения устанавливается удаленным пользователем в своем браузере, но мы имеем возможность явно указывать, какой цвет использовать для выделения гиперссылок с помощью стилевых таблиц. Их применение рассмотрим в следующей главе. Если мы посмотрим на код листинга 1.15, то увидим, что текст, с которым связана гиперссылка, обрамляется тэгами `<a>` и `</a>`. При этом открывающий тэг обладает параметром `href`, значение которого определяется URL того интернет-ресурса, на который указывает создаваемая гиперссылка. URL может быть как полным, т. е. включающим в себя наименование протокола доступа к ресурсу, наименования сайта и страницы, как, например, "`http://www.mysite.com/mypage.htm`", так и относительным, указывающим

путь к документу, находящемуся на том же сайте, где и исходная Web-страница, с кодом приблизительно следующего вида: "`../chap2/page1.htm`"

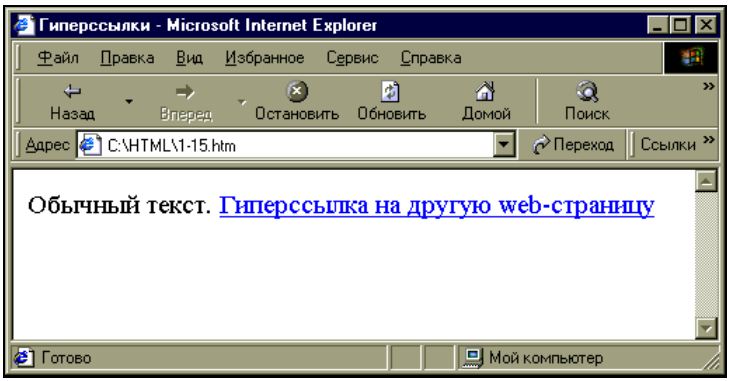


Рис. 1.15. Окно браузера с результатом отображения файла, приведенного в листинге 1.15

Здесь мы ссылаемся на Web-страницу, находящуюся в файле с наименованием page1.htm, который размещается в каталоге `chap2`.

Об URL следует рассказать несколько подробнее, это единственное и абсолютно точное средство идентификации любых ресурсов, размещенных в Сети. Его можно описать, например, следующим образом:

```
http://www.mysite.com/folder1/file1.htm
```

URL делится на три логические части. В самом начале URL мы задаем протокол, по которому производится доступ к интернет-ресурсу. Для Web-страниц используется протокол HTTP, как в нашем примере. Если необходимо установить гиперссылку на некий файл, размещенный на каком-либо FTP-сервере, то наименование протокола записывается в начальной части URL. Существует механизм, позволяющий одновременно с активизацией гиперссылки запускать почтовую программу, установленную пользователем в системе по умолчанию, которая автоматически открывает окно создания электронного письма с уже заданным адресом получателя. Тогда URL записывается следующим образом:

```
mailto://address@host.com
```

В этом случае для обозначения протокола используется ключевое слово `mailto`, а после двоеточия и пары слэшей, которые всегда отделяют наименование протокола от основной части адреса, записывается адрес электронной почты, на который будет отсылаться созданное пользователем электронное письмо.

Вторая часть URL описывает адрес ресурса. Для Web-страницы это доменное имя сайта. Обычно применяется доменное имя второго или третьего

уровня. Доменные имена первого уровня записываются в самом конце доменного имени. Внутри каждого домена первого уровня контролирующими организациями выделяются доменные имена второго уровня. А владельцы доменных имен второго уровня самостоятельно создают доменные имена третьего уровня. Так, практически, каждый владелец доменного имени второго уровня выделяет себе доменное имя третьего уровня "www", и из выделенного имени mysite.com получается уже более привычное www.mysite.com.

Третья часть URL представляет собой путь к конкретному файлу во внутреннем файловом пространстве Web-сервера. Здесь необходимо сделать маленькое отступление, разъясняющее механизм действия Web-серверов.

Для того чтобы любой удаленный пользователь мог получить доступ к какому-либо Web-сайту, представляющему собой, как известно, коллекцию отдельных Web-страниц и специализированных выполняемых приложений-скриптов, необходимо, чтобы все эти Web-страницы находились на отдельном компьютере, на котором действует специализированное приложение, называемое Web-сервером. В обязанности подобного Web-сервера входит получение запросов удаленных пользователей, обработка этих запросов и передача удаленным пользователям содержимого Web-страниц по протоколу HTTP. Для Web-сервера на диске компьютера выделяется отдельный каталог, в котором и размещаются HTML-файлы, содержащие Web-страницы, и приложения-скрипты, обеспечивающие интерактивные функции сайта. Естественно, внутри общего каталога, отведенного для Web-страниц, мы можем создавать свои структуры папок для более четкого разделения ресурсов сайта. Обычно в отдельные папки выделяются графические изображения, используемые в оформлении Web-страниц, сами Web-страницы группируются в каталогах по признаку принадлежности к тому или иному разделу сайта. Полный путь к некоему HTML-файлу или иному ресурсу, который используется в оформлении Web-страниц, и является третьей частью URL. Рассмотрим маленький пример.

```
http://www.mysite.com/content/about.HTML
```

Этот URL указывает на файл с наименованием about.html, который располагается в каталоге content, находящемся в файловом пространстве Web-сервера с доменным именем www.mysite.com. При этом передача пользователю содержимого искомого файла будет проходить с использованием протокола HTTP.

Впрочем, в поле ввода адреса интернет-ресурса любого браузера можно ввести только доменное имя сайта, и мы уже получим доступ к основной Web-странице данного сайта. Дело в том, что если не указывать в поле ввода протокол, то браузер автоматически использует протокол HTTP. Если не указывать полное наименование HTML-файла с Web-страницей, то Web-сервер, принимающий запрос удаленного пользователя, выдаст главную страницу сайта, называемую часто "домашней страницей", которая обяза-

тельно хранится в файле с наименованием index.html. Иными словами, каждый сайт обязан содержать файл с таким наименованием, который является стартовой страницей Web-сайта.

При этом необходимо учитывать, что если в браузере мы можем позволить себе написать адрес не полностью, понадеявшись на правильную его интерпретацию самим браузером, то в гиперссылках мы обязаны писать URL полностью, за исключением тех случаев, когда документы, на которые указывает гиперссылка, находятся на том же сайте, что и исходная Web-страница. В подобных случаях мы можем не указывать доменное имя сайта, ограничившись указанием протокола и полного наименования ресурса, включающего в себя путь в файловой системе Web-сервера, на который указывает гиперссылка.

Тэг `<a>` обладает достаточно большим количеством параметров, которые позволяют задавать самые различные свойства гиперссылки. По мере изучения материала мы рассмотрим их все.

Гиперссылки принято разделять на глобальные и локальные. Глобальные гиперссылки указывают на интернет-ресурсы, размещенные вне исходной Web-страницы, а локальные — на некоторые ресурсы, расположенные на текущей странице. Их часто используют в тех случаях, когда страница содержит достаточно большой объем информации, который не умещается полностью в окне просмотра. Например, мы размещаем на Web-странице некую повесть, или текстовый документ, разбитый на разделы. При этом в начале этого документа мы можем создать его оглавление, действующее на основе гиперссылок, каждая из которых будет указывать на какой-либо раздел документа. Тогда пользователь, загрузивший Web-страницу, сможет перемещаться по документу с использованием этих гиперссылок, а не только с помощью вертикальной полосы прокрутки, что, несомненно, облегчает навигацию.

Для того чтобы в документе мы могли использовать локальные гиперссылки, необходимо сначала пометить те фрагменты документа, на которые они будут указывать. Например, если мы хотим, чтобы какая-либо локальная гиперссылка перемещала удаленного пользователя к некоему определенному абзацу, то в этот абзац мы должны поместить тэг `<a>` с параметром `name`. При этом сама гиперссылка будет создаваться с использованием несколько измененного значения параметра `href`. Но проще все это рассмотреть на примере.

Листинг 1.16

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
```

```
  <title>Гиперссылки</title>
  <body>
  <p><a href="#anch1">Локальная гиперссылка</a></p>
  <p>Обычный текст</p>
  <p><a name="anch1">Текст, на который мы ссылаемся в начале
↳страницы</a></p>
  </body>
</html>
```

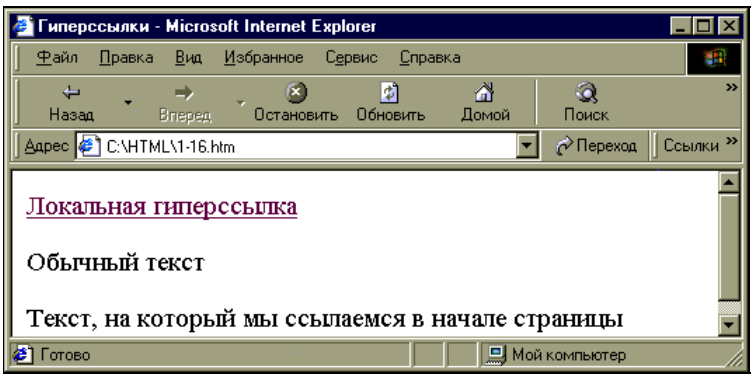


Рис. 1.16. Окно браузера с результатом отображения файла, приведенного в листинге 1.16

Из текста листинга видно, что при ссылке на идентификатор, располагающийся в теле Web-страницы, мы в качестве значения параметра `href` указываем наименование этого идентификатора со знаком решетки перед ним. Подобным образом мы можем использовать гиперссылки не только на фрагменты содержимого исходной Web-страницы, но и на фрагменты других Web-страниц, помеченные тем же способом. Такая гиперссылка будет иметь приблизительно следующий вид:

```
<a href="http://www.mysite.com/doc2.htm#anch3">
```

В этом примере мы совмещаем использование полного URL документа и ссылку на некий его фрагмент, помеченный как "`anch3`".

Следует отметить, что если текст, на который указывает гиперссылка, уже отображен в окне просмотра, как это случилось в нашем последнем примере, то никаких изменений не произойдет. Но стоит лишь изменить размеры окна просмотра браузера по вертикали так, чтобы скрыть последнюю строку содержимого, на которую указывает гиперссылка, и результат ее действия можно будет уже наглядно продемонстрировать.

Все наименования подобных маркеров-закладок, которые определяются значением параметра `name`, должны быть уникальными. При этом следует учитывать, что HTML не делает различий между заглавными и строчными

символами. Следовательно, если в нашем HTML-документе определены две закладки, на которые будут впоследствии указывать гиперссылки, а имена этих закладок различаются лишь регистром символов, то для HTML-анализатора, встроенного в браузер, эти идентификаторы будут считаться одинаковыми, и он не будет обрабатывать ни один из них.

Впрочем, мы уже знаем, что идентифицировать какой-либо элемент HTML-документа можно и с помощью параметра `id`, включенного в состав любого тэга. При этом гиперссылки, указывающие на фрагменты документов, могут использовать и эти параметры. Таким образом, для установки маркера-закладки на какой-либо тэг не обязательно использовать тэг `<a>` с параметром `name`, достаточно лишь применить параметр `id`.

Значения параметров `name` и `id` — это уникальные идентификаторы элементов HTML-документа, поэтому ни одно значение параметра `name` не должно совпадать ни с одним значением параметра `id`.

Какой же вариант следует выбрать для своих Web-страниц, какой именно параметр стоит использовать? Здесь необходимо учитывать, что, несмотря на то, что параметр `id` может служить для выполнения нескольких действий, таких как идентификация маркера-закладки, создание уникальных идентификаторов для выполняемых сценариев DHTML и стилевого оформления, некоторые устаревшие браузеры могут не воспринимать эти идентификаторы для определения гиперссылок.

Совместно с тэгом `<a>` часто используется параметр `title`, который помогает удаленному пользователю идентифицировать гиперссылку. Значением этого параметра является текстовая строка, которая отображается в виде маленькой подсказки (хинта), когда пользователь наводит курсор мыши на гиперссылку. Выглядит объявление подобной гиперссылки приблизительно следующим образом:

```
<a href="www.site.com" title="Очень симпатичный сайт">
```

С помощью параметра `hreflang` мы можем задать язык, на котором написано текстовое содержимое того интернет-ресурса, на который указывает данная гиперссылка. В качестве значения параметра используется одно из стандартных обозначений языка, которые мы рассматривали в этой главе несколько ранее.

Одного указания языка, на котором написано текстовое содержимое Web-страницы, бывает недостаточно. Необходимо задавать еще и применяемую кодировку. В этой ситуации может помочь уже знакомый нам параметр `charset`, значением которого является стандартное обозначение кодировки, применяемой для отображения текстового содержимого того интернет-ресурса, на который указывает данная гиперссылка.

Параметр `rel` позволяет описать назначение документа, на который указывает гиперссылка. Значение этого параметра определяет, как связаны между собой исходный документ и документ, на который мы ссылаемся. Исполь-

зование данного параметра никак не влияет на способ отображения гиперссылки или механизм получения ресурса, но может быть полезным тогда, когда Web-страницы предназначены не только для просмотра браузером, но и для обработки некоторыми специализированными приложениями. К таким приложениям относятся, например, мощные поисковые системы, которые в состоянии правильно распознавать и обрабатывать подобные отношения между документами, обозначаемыми с помощью гиперссылок.

Значением параметра `rel` может быть одно из предопределенных ключевых слов, которые мы сейчас рассмотрим.

- `Alternate`. Указывает на то, что документ является альтернативным представлением исходного документа. Если при этом в гиперссылке используется параметр `lang` со значением, отличным от языка исходного документа, то подобная гиперссылка обычно рассматривается как ссылка на копию исходного документа на другом языке.
- `Stylesheet`. Обозначает, что документ, на который указывает гиперссылка, является стилевой таблицей. О механизме применения стилевых таблиц подробно рассказано во второй главе.
- `Start`. Используется для обозначения начального, стартового документа некоего множества документов. Применительно к Web-сайту это, очевидно, будет домашняя страница.
- `Next`. Используется, если стартовый документ и документ, на который указывает гиперссылка, входят в некую линейную упорядоченную последовательность документов, и последний является следующим в последовательности по отношению к исходной Web-странице.
- `Prev`. Употребляется в том же случае, что и предыдущее значение, но теперь указывает на то, что документ в цепочке является не следующим, а предыдущим по отношению к стартовому документу.
- `Index`. Используется в гиперссылках, которые указывают на документ, включающий в себя индексированное содержание исходной Web-страницы.
- `Glossary`. Указывает на то, что документ содержит словарь терминов, использующихся в исходном документе.
- `Copyright`. Обозначает, что в документе, на который указывает гиперссылка, хранится уведомление об авторских правах на содержимое исходного документа.
- `Chapter`. Применяется в гиперссылках, указывающих на документы, содержащие отдельные главы из некоего множества документов.
- `Section`. Гиперссылка указывает на документ, который является разделом в общем множестве документов, образующих единое целое.

- `Subsection`. Употребляется в том же случае, что и предыдущее, но содержимое документа рассматривается как *подраздел*.
- `Appendix`. Используется тогда, когда документ, на который указывает гиперссылка, содержит приложение к исходному документу.
- `Help`. Употребляется для ссылок на документы, которые предоставляют дополнительную справочную информацию о содержимом исходного документа.
- `Bookmark`. Применяется для ссылок на HTML-документы, содержащие ссылки на некоторые выделенные ключевые фрагменты исходного документа.

При использовании этих значений необходимо учитывать, что HTML-анализаторы чувствительны к регистру символов, поэтому нужно полностью придерживаться того написания, которое приведено в нашем списке.

На самом деле в HTML есть возможность создавать свои значения для параметра `rel`, но для использования этого параметра в полной мере необходимо применять специализированные средства описания этих новых значений и программное обеспечение, нацеленное на распознавание этих типов ссылок.

Данная технология, описывающая взаимосвязи документов, указываемых в подобных ссылках, — это паллиатив, который призван обеспечить хотя бы часть возможностей, предоставляемых новым языком описания документов в Интернете — XML (eXtensible Markup Language), которому пророчат роль преемника и "убийцы" HTML. Но пока не существует распространенных XML-браузеров и приложений, обрабатывающих XML-документы. Следовательно, технология HTML имеет полное право на жизнь и будет использоваться в качестве основы для Web-страниц еще достаточно долго.

Вернемся к рассмотрению атрибутов тэга `<a>`. Если атрибут `rel` описывает тип документа, на который указывает гиперссылка, то атрибут `rev` задает тип исходного документа. В качестве значений этого атрибута применяется все тот же набор ключевых слов, который использовался и для атрибута `rel`.

С помощью параметра `target` мы указываем, в каком фрейме необходимо отобразить документ, на который указывает гиперссылка. Дело в том, что обычно в одном окне просмотра браузера отображается один документ. Но в HTML существует возможность разделить окно просмотра на несколько областей, называемых фреймами, в каждой из которых будет отображаться свой HTML-документ. Технологию использования фреймов мы будем рассматривать в одном из следующих разделов этой главы, а пока лишь отметим, что параметр `target` позволяет явно указывать, в каком фрейме необходимо отобразить Web-страницу. Установка гиперссылки с использованием этого параметра будет выглядеть приблизительно следующим образом:

```
<a href="http://www.mysite.com/doc1.html" target="_top">
```

Подобная гиперссылка заставит браузер загрузить Web-страницу, URL которой указан в качестве значения параметра `href`, в верхний фрейм с именем, заданным параметром `target`. Значения этого параметра выбираются из списка ключевых слов, определенных в спецификации HTML. Мы рассмотрим их в разделе, посвященном фреймам.

Некоторые управляющие элементы и гиперссылки способны передавать друг другу с помощью клавиши табуляции фокус ввода, т. е. готовность к приему информации. Каждое нажатие этой клавиши активизирует следующий по порядку управляющий элемент, входящий в общую последовательность. Порядок перемещения между элементами управления, входящими в общую последовательность, задается с помощью параметра `tabindex`. В качестве значения этого параметра используется целое положительное число, и чем больше это число у какого-либо элемента управления или гиперссылки, тем позже дойдет до него очередь при перемещении фокуса ввода. Естественно, ни у какой пары элементов оформления Web-страницы значения этого параметра не должны совпадать, иначе браузер просто не будет включать их в последовательность элементов, получающих фокус ввода.

Помимо доступа к гиперссылкам с помощью последовательных нажатий клавиши табуляции, мы можем использовать параметр `accesskey`, в качестве значения которого указывается символ. Как только пользователь нажимает на клавишу, которая вводит заданный символ, фокус ввода автоматически передается той гиперссылке, в объявление которой и встроен параметр `accesskey`, и пользователь может активизировать ее без использования мыши, одним лишь нажатием клавиши <Enter>. Рассмотрим маленький пример.

```
<a href="http://www.mysite.com/doc1.html" tabindex="2" accesskey="U">
```

Этим тэгом мы объявляем гиперссылку, доступ к которой можно получить либо с помощью последовательных нажатий клавиши табуляции, либо нажатием клавиши с символом "U".

Носителем гиперссылки может выступать не только текст, но и графическое изображение. Для этого необходимо тэг, объявляющий вставку графики в состав содержимого Web-страницы, поместить между тэгами `<a>` и `</a>`. С графическим изображением мы можем связать не одну, а несколько гиперссылок. Для этого в пределах картинки выделяется несколько активных областей, нажатие кнопкой мыши на каждую из которых активизирует соответствующую гиперссылку. Подобная технология называется *сегментированной графикой*.

Для создания таких графических изображений, связанных с несколькими гиперссылками, применяются специализированные тэги. Сначала объявляются активные области рисунка, называемые также сегментами. Вся их совокупность составляет карту активных сегментов рисунка, которой присваивается некое имя. Затем это имя карты связывается с самим изображением, объявляемым с помощью тэга `<img>`. Рассмотрим типичный пример.

Листинг 1.17

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Сегментированная графика</title>
  <body>
  <p>Это  обычный текст.<img src="redhood.gif" usemap="#map1"
↳alt="docs"></p>
  <map name="map1">
  <area href="1.htm" shape="rect" coords="1,1,10,10" alt="1">
  <area href="2.htm" shape="circle" coords="20,20,5" alt="2">
  <area href="1.htm" shape="poly" coords="1,30,10,35,1,50,1,30" alt="3">
  </map>
  </body>
</html>
```

В этом примере видно, что при объявлении рисунка в тэг `<img>` мы помещаем параметр `usemap`, значением которого является наименование карты активных сегментов рисунка, связанных с гиперссылками. Перед наименованием используемой карты активных сегментов вставляется символ решетки. Описание этой карты располагается между тэгами `<map>` и `</map>`. При этом у открывающего тэга `<map>` существует обязательный параметр `name`, значением которого является наименование карты.

Карта состоит из описания сегментов. Каждый сегмент описывается посредством одного тэга `<area>`. Мы можем применять активные сегменты трех различных форм: прямоугольники, круги и многоугольники. Форма задается с помощью обязательного параметра `shape`. В качестве значения данного параметра используется одно из трех предустановленных значений.

- Значение `rect` используется для создания прямоугольных активных областей.
- Значение `circle` применяется для создания кругового сегмента.
- Значение `poly` позволяет создавать активные сегменты в виде выпуклых многоугольников.

После того как мы задали тип формы, следует точно определить размеры сегментов и их расположение на нашем графическом изображении. Для этого используется параметр `coords`, в качестве значения которого записывается перечень координат, определяющих активный сегмент. В листинге 1.17 мы можем увидеть, что координаты в общем списке разделяются обычной запятой. Отсчет координат ведется от верхнего левого угла рисунка, который имеет координаты (0;0).

Для прямоугольных сегментов задаются координаты верхнего левого и правого нижнего угла. А для многоугольников последовательно перечисляются координаты всех точек, в порядке соединения их линиями. Естественно, первая и последняя пара координат должны совпадать, иначе многоугольник окажется незамкнутым, и активный сегмент не будет обрабатываться.

В тэг `<area>` входит и параметр `href`, значение которого задает URL ресурса, на который указывает гиперссылка данного сегмента. Но этот параметр, как ни удивительно, не является обязательным. В том случае, когда сегмент создается, но не должен соединяться с гиперссылкой, следует использовать параметр `nohref`, который не имеет значения.

У тэга `<area>` есть и обязательный параметр. Это параметр `alt`, значением которого является текстовая строка, задающая альтернативное представление рисунка. Как мы знаем, эта строка отображается в виде подсказки (хинта), когда пользователь наводит курсор мыши на объект. В нашем случае подобным объектом является активный сегмент, внедренный в графическое изображение.

В тэге `<area>`, как и в тэгах обычных гиперссылок, применяются параметры `tabindex` и `accesskey`, которые позволяют активизировать гиперссылку без использования мыши, с помощью клавиатуры. Структуру значений этих параметров мы рассмотрели несколько ранее. Немного отступая от темы, следует заметить, что любой *правильный* Web-мастер непременно должен использовать эти параметры, т. к. только их применение может гарантировать активизацию гиперссылок без помощи мыши.

Итак, мы разобрались с применением технологии сегментированной графики, но о ссылках мы узнали далеко не все. В спецификации HTML определен тэг `<link>`, который создает непривычную для нас гиперссылку, а некоторую связь между отображаемым документом и каким-либо дополнительным файлом. Тэги `<link>` могут размещаться только в разделе заголовка HTML-документа, между тэгами `<head>` и `</head>`. В качестве примера использования рассматриваемого нами тэга `<link>` можно привести следующий фрагмент кода:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/HTML4/strict.dtd">
<HTML>
<HEAD>
  <TITLE>Chapter 2</TITLE>
  <LINK rel="Index" href="../index.HTML">
  <LINK rel="Next"  href="Chapter3.HTML">
  <LINK rel="Prev"  href="Chapter1.HTML">
</HEAD>
```

В этом фрагменте мы указываем, что данная Web-страница содержит вторую главу некоего обширного документа, и с помощью тэгов `<link>` описываем

связи с Web-страницами, содержащими оглавление, а также предыдущую и следующую главу. Тип документа, на который указывает ссылка, определяет знакомый нам параметр `rel`. Обычный браузер, конечно, не сможет каким-либо способом проанализировать и отобразить подобные ссылки. Они были введены в HTML для создания документов, которые обрабатывались бы специализированными приложениями, позволяющими на основе данного формата создавать системы документооборота. Но встроенных скудных возможностей HTML для создания полноценных систем документооборота было явно недостаточно, и когда был разработан стандарт XML, разработчики с радостью сменили фаворита.

Так что же получается, что в обычных Web-страницах тэг `<link>` абсолютно бесполезен? На самом деле нет. Только с помощью этого тэга мы можем соединять Web-страницы с внешними стилевыми таблицами, мощнейшим средством управления отображением документа в браузере. О стилевых таблицах мы узнаем чуть позже, во второй главе. Там же мы рассмотрим пример использования тэга `<link>`. Пока нам осталось лишь перечислить параметры данного тэга. Все эти параметры применяются и в тэгах `<a>` с теми же возможными значениями, поэтому мы не будем детально рассматривать каждый параметр, а просто вкратце их назовем.

- `charset` — задает кодировку документа, на который указывает ссылка.
- `href` — описывает URL документа, на который указывает ссылка.
- `hreflang` — равен кодовому обозначению языка, на котором написан связываемый документ.
- `type` — устанавливает тип документа, на который указывает ссылка.
- `rel` — задает статус документа, на который указывает ссылка, по отношению к исходному документу.
- `rev` — определяет статус исходного документа по отношению к тому, на который указывает данная ссылка.
- `media` — указывает тип устройства, на котором будет отображаться связываемый документ. В настоящее время поддерживаются документы, отображаемые на обычных мониторах, на брайлевских мониторах, рассчитанных на людей с потерей зрения, а также предназначенные для печатающих устройств и устройств речевого воспроизведения.

Помимо перечисленных параметров используется набор общих для всех тэгов параметров дополнительной идентификации.

Нам осталось рассмотреть только один тэг, применяемый для обслуживания гиперссылок. Как мы знаем, можно не указывать полный URL для документов, на которые указывают гиперссылки, если они находятся на том же Web-сайте, что и исходная Web-страница. Оказывается, мы можем точно так же поступать и с гиперссылками на Web-страницы, входящие в состав иных сайтов.

С помощью тэга <base> мы задаем основание для гиперссылок с укороченными значениями параметра href. Попробуем продемонстрировать действие этого механизма на простом и наглядном примере.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<HTML>
 <HEAD>
   <TITLE>Web-страница</TITLE>
   <BASE href="http://www.mysite.com/tour.HTML">
 </HEAD>
 <BODY>
   <P>Текст<A href="../cages/birds.gif">Ссылка</A>
 </BODY>
</HTML>
```

В качестве значения обязательного параметра href тэга <base> мы задаем имя документа, служащего основой для содержимого Web-страниц нашего сайта. Теперь если в теле нашего HTML-документа мы применим гиперссылку с укороченным URL, то доменным именем для этого URL будет имя не исходного сайта, а того, который указан в тэге <base>. Таким образом, в примере гиперссылка указывает на документ с URL www.mysite.com/cages/birds.gif.

Следует уточнить, что тэг <base> размещается в заголовке документа, между тэгами <head> и </head>.

Это все, что мы можем рассказать о применении гиперссылок в HTML.

Списки

Правильнее было бы рассматривать вопросы оформления списков в *разд. "Оформление текста"* (ранее в этой главе), т. к. списки тоже состоят из текстовых строк и абзацев, но тема списков достаточно велика, поэтому ей посвящен отдельный раздел.

Как выглядят списки, мы все прекрасно знаем. Чаще всего они бывают нумерованными и маркированными. В нумерованных списках каждый пункт обозначен некоторым номером. У маркированных списков пункты выделяются графическими маркерами. Списки также делят на одноуровневые и многоуровневые. В одноуровневых списках все элементы равноценны. В многоуровневых каждый элемент может содержать еще несколько пунктов, которые организуют новый список, вложенный в исходный элемент. Итак, начнем мы с маркированных ненумерованных списков. Все содержимое такого списка обязано располагаться внутри пространства, ограниченного стартовым тэгом <ul> и его закрывающим двойником . Отдельные

элементы списка объявляются с помощью обозначающего тэга <li>, который не имеет закрывающего двойника. Точнее, закрывающий тэг может применяться, но он не обязателен.

Замечание

Использовать необязательные закрывающие тэги в HTML-документах имеет смысл, если в дальнейшем мы собираемся отображать их с помощью XML-браузеров. О новом стандарте XML, который со временем должен будет сменить HTML, в данной книге мы не рассказываем.

Приведем простейший пример использования маркированного списка.

Листинг 1.18

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Маркированный список</title>
  <body>
  <p>Это обычный текст.</p>
  <ul>
  <li>Первый пункт списка
  <li>Второй пункт списка
  </ul>
  </body>
</html>
```

Результат отображения данного кода в окне просмотра браузера показан на рис. 1.17.

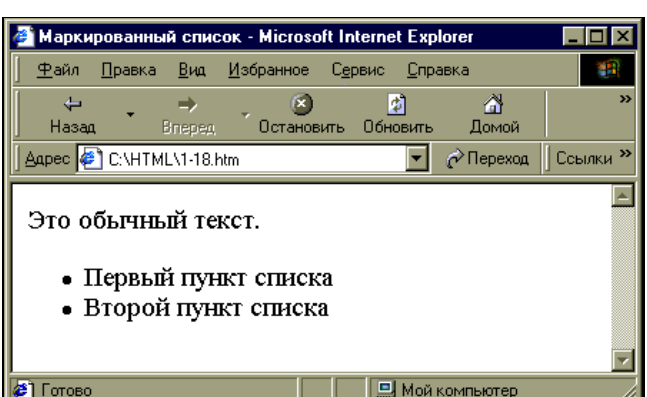


Рис. 1.17. Окно браузера с результатом отображения файла, приведенного в листинге 1.18

В нашем примере мы использовали тэги без параметров, поэтому браузер применил установки по умолчанию для отображения элементов списка. Но ведь у нас есть некоторые дополнительные возможности отображения, которые регулируются с помощью различных параметров тэга, объявляющего какой-либо элемент списка. Тэг `<li>` обладает некоторым набором параметров, часть из которых имеет смысл применять в случае использования нумерованных списков, а часть предназначена для маркированных списков. Раз уж мы начали свое рассмотрение с маркированных списков, то и параметры опишем именно те, которые применяются для элементов подобных списков.

С помощью параметра `type` мы можем явно указывать, какой тип маркера следует использовать для отображения элементов списка. В качестве значений данного параметра применяется одно из трех предустановленных ключевых слов.

- Значение `disc` указывает браузеру, что следует использовать маркеры в виде маленького заполненного круга.
- Значение `circle` задает маркер в виде окружности, используется по умолчанию.
- Значение `square` устанавливает маркер в виде маленького прямоугольника.

Все эти значения необходимо указывать с сохранением регистра так, как они приведены в книге, поскольку при распознавании ключевых слов HTML-анализаторы чувствительны к регистру. Желательно все предустановленные ключевые слова, используемые как значения каких-либо параметров, писать с сохранением регистра. Наименования параметров и тэги нечувствительны к регистру символов. Теперь, после этого отступления, узнаем, как именно выглядят эти маркеры.

Листинг 1.19

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Маркированный список</title>
  <body>
  <ul>
  <li type="circle">Первый пункт списка
  <li type="disc">Второй пункт списка
  <li type="square">Третий пункт списка
  </ul>
  </body>
</html>
```

Результат отображения данного кода в окне просмотра браузера показан на рис. 1.18.

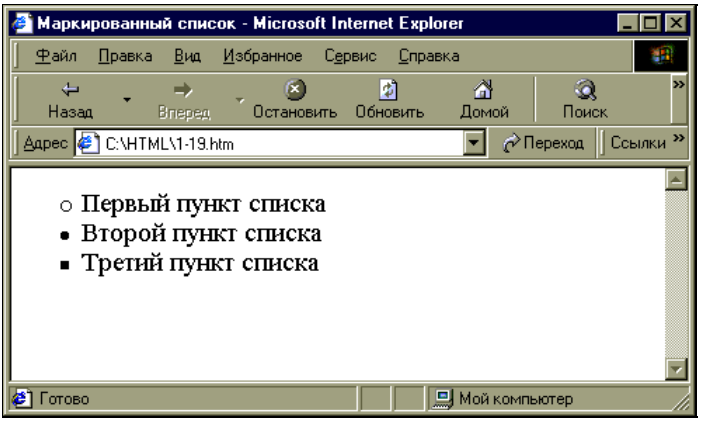


Рис. 1.18. Окно браузера с результатом отображения файла, приведенного в листинге 1.19

Необходимо отметить, что как именно будет отображен маркер того или иного типа зависит только от браузера, и маркеры не всегда одинаковы в различных браузерах. Однако общее подобие все же будет сохранено, и окружность всегда останется окружностью, а прямоугольник — прямоугольником.

Кроме того, в тэге `<li>` может использоваться параметр `compact`, применяемый без каких-либо значений. Если его ввести в состав искомого тэга, то браузер должен будет отобразить содержимое элемента списка более компактно. Параметр этот директивный, и выполнение его лежит только на совести производителей того или иного браузера. Мне не удалось заметить каких-либо ощутимых изменений во внешнем виде элементов списка при использовании данного параметра.

Теперь перейдем к рассмотрению упорядоченных нумерованных списков. Список подобного типа создается тэгами `<ol>` и `</ol>`. Элементы списка объявляются с помощью все того же тэга `<li>`. Рассмотрим пример создания простейшего нумерованного списка и увидим, как его обрабатывает и отображает браузер.

Листинг 1.20

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
```

```
  <title>Нумерованный список</title>
  <body>
  <ol>
  <li>Первый пункт списка
  <li>Второй пункт списка
  <li>Третий пункт списка
  </ol>
  </body>
</html>
```

Результат отображения данного кода в окне просмотра браузера показан на рис. 1.19.

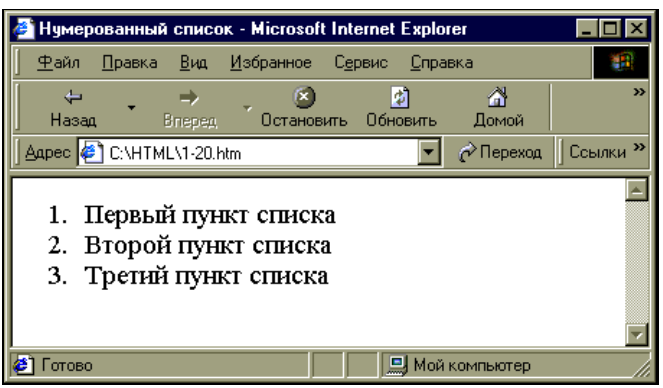


Рис. 1.19. Окно браузера с результатом отображения файла, приведенного в листинге 1.20

Если использовать только стандартные значения (по умолчанию), нумерованный список выглядит просто и правильно, но этого может быть недостаточно. Для изменения его параметров и внешнего вида у нас есть достаточно средств, т. е. параметров все того же тэга `<li>`.

Уже знакомый нам параметр `type` позволяет указывать, какие именно цифры могут использоваться для обозначения элементов списка. Значением этого параметра может быть одно из предустановленных ключевых слов, которые мы сейчас все и рассмотрим.

- Значение `1` применяется для нумерации обычными арабскими цифрами.
- Значение `a` задает обозначения в виде строчных букв латинского алфавита. Нумерация идет в алфавитном порядке, начиная от a и до z. Впрочем, при дальнейшем продолжении списка обозначения тоже будут продолжены с использованием комбинаций строчных латинских букв.
- Значение `A` так же, как и предыдущее, задает обозначение элементов списка с помощью букв латинского алфавита, но при этом будут уже использоваться заглавные буквы.

- Значение `i` создает нумерацию римскими цифрами, которые обозначаются строчными буквами латинского алфавита.
- Значение `I` устанавливает римские цифры в качестве основы нумерации, но для их отображения будут использоваться заглавные латинские буквы.

Теперь, когда мы теоретически знаем, какие бывают варианты нумерации, и как их устанавливать, пришло время привести пример их использования.

Листинг 1.21

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Нумерованный список</title>
  <body>
  <ol>
  <li type="1">Первый пункт списка
  <li type="a">Второй пункт списка
  <li type="A">Третий пункт списка
  <li type="i">Четвертый пункт списка
  <li type="I">Пятый пункт списка
  </ol>
  </body>
</html>
```

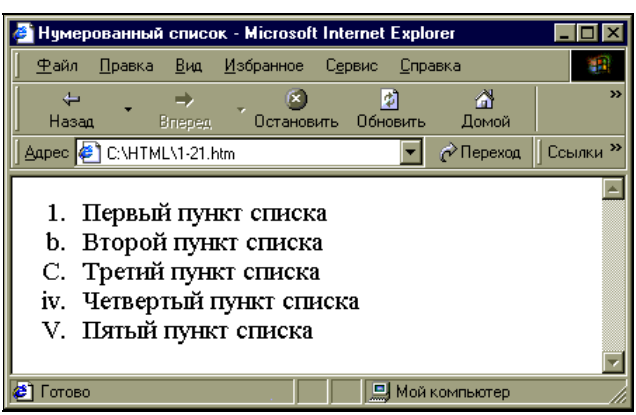


Рис. 1.20. Окно браузера с результатом отображения файла, приведенного в листинге 1.21

На рис. 1.20 видно, как браузер меняет внешний вид каждого элемента списка, сохраняя при этом их общую нумерацию. HTML предоставляет воз-

можность самостоятельно указывать стартовый номер элемента. Для этого в тэг <ol> вставляется параметр start. Значением этого параметра является натуральное число, которое и будет номером первого элемента списка. Внутри списка мы можем произвольно менять порядковые номера элементов с помощью параметра value, применяемого в составе тэга <li>. Для того чтобы увидеть механизм использования этих параметров, рассмотрим еще один пример.

Листинг 1.22

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Нумерованный список</title>
  <body>
  <ol start=5>
  <li>Пятый пункт списка
  <li>Шестой пункт списка
  <li value=10>Десятый пункт списка
  <li>Одиннадцатый пункт списка
  </ol>
  </body>
</html>
```

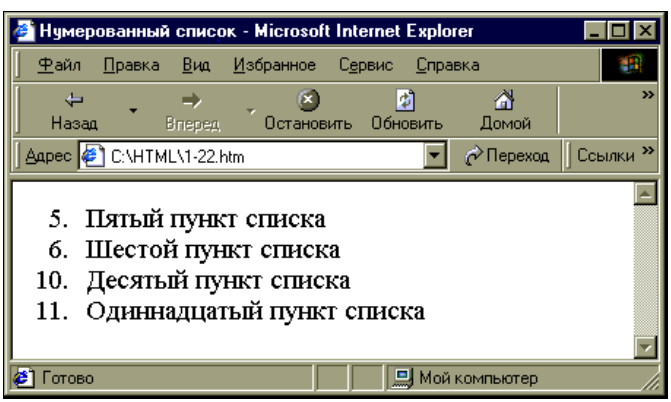


Рис. 1.21. Окно браузера с результатом отображения файла, приведенного в листинге 1.22

Мы можем также создавать многоуровневые вложенные списки, совмещая при этом маркированные и нумерованные элементы. Для этого достаточно вложить тэг, обозначающий начало нового вложенного списка в соответ-

ствующий элемент основного списка так, как показано в следующем примере.

Листинг 1.23

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Вложенные списки</title>
  <body>
  <ol>
  <li>Первый пункт списка
  <ol>
  <li>Первый вложенный пункт
  <li>Второй вложенный пункт
  </ol>
  <li>Второй пункт списка
  <ul>
  <li>Вложенный маркированный элемент
  <li>Вложенный маркированный элемент
  <ul>
  </ol>
  </body>
</html>
```

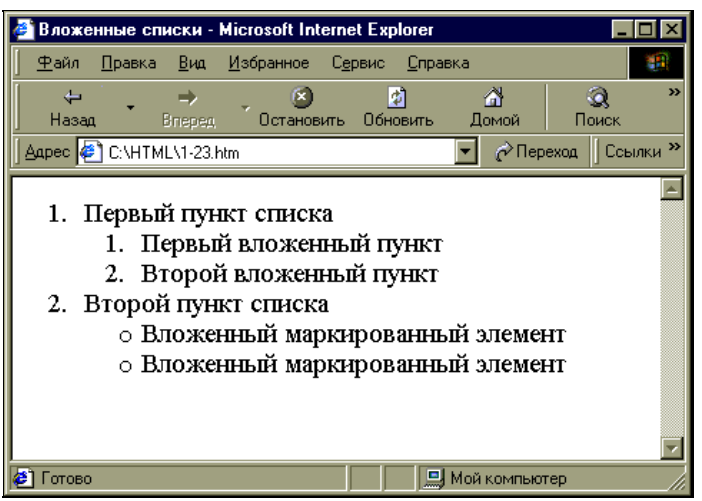


Рис. 1.22. Окно браузера с результатом отображения файла, приведенного в листинге 1.23

Итак, мы изучили основные варианты использования стандартных нумерованных и маркированных списков, но в HTML предусмотрены еще несколько видов специализированных списков, которые тоже необходимо рассмотреть.

Перед тем как перейти к непосредственному рассмотрению этих дополнительных типов списков, следует сделать маленький экскурс в историю создания HTML. Дело в том, что он был разработан в лаборатории ЦЕРН Тимом Бернерсом-Ли для публикации связанных научных отчетов и документов, и как следствие, изначально в него были вставлены средства текстового оформления, присущие техническим и научным документам. В их числе и списки определений и терминов.

Список определений создается с помощью тэга `<dl>` с его закрывающим двойником `</dl>`. В качестве элементов используются термины, обозначаемые тэгами `<dt>`, и определения, создаваемые с помощью тэгов `<dd>`. Техническое отличие этих двух элементов списка состоит в том, что термины являются inline-элементами, т. е. обязаны занимать не более одной строки, а определения — flow-элементами, т. е. не ограничены пределами одной строки. Рассмотрим пример использования подобного списка определений.

Листинг 1.24

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Список определений</title>
  </head>
  <body>
  <dl>
  <dt>Это термин
  <dd>А это определение.
  </dl>
  <p>А это обычный текст</p>
  </body>
</html>
```

Результат отображения данного кода в окне просмотра браузера показан на рис. 1.23.

Это все, что мы можем узнать о списках. Мы научились их использовать. Всегда следует помнить о различных списках, если необходимо перечислить на Web-страницах некие однотипные элементы. Это очень мощное и *полезное* средство структурирования информации.

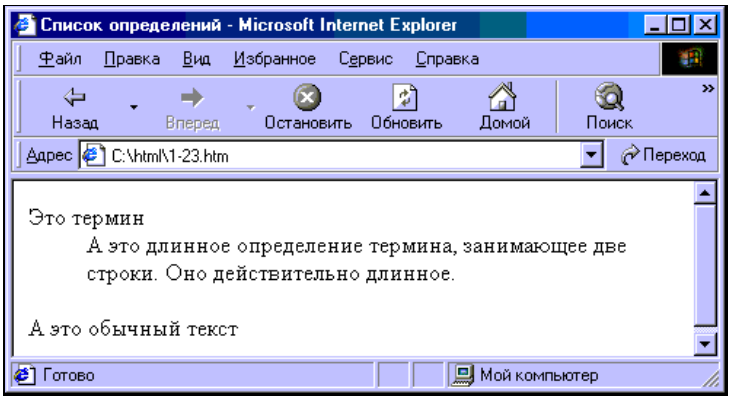


Рис. 1.23. Окно браузера с результатом отображения файла, приведенного в листинге 1.24

Таблицы

Таблицы — одна из важнейших форм визуальной организации информации, располагающейся на Web-страницах, а не просто средство отображения таблиц, включенных в состав Web-страниц. Это, пожалуй, единственное средство некоей верстки страниц. Мы знаем, что в HTML, а точнее, в его стандарт, не встроены средства точного позиционирования элементов оформления Web-страниц, поэтому таблицы пришлись как нельзя кстати. Их можно распространить на всю страницу, и уже в ячейках таблиц размещать элементы оформления Web-страниц. В качестве подобного средства верстки также могут применяться фреймы, которые мы будем рассматривать в следующем разделе этой главы, но в последнее время в среде Web-мастеров они теряют популярность. Тому есть свои причины, берущие начало в теории пользовательского интерфейса. Но вернемся к нашим таблицам.

Для полного понимания правил использования таблиц необходимо знать их структуру. В HTML таблица разбивается на строки, а те, в свою очередь, на ячейки. Так как для всех этих объектов существуют свои параметры, задающие их размеры, создается некоторая иерархия свойств ячеек, строк и самой таблицы. Основой этой иерархии являются ячейки и их содержимое. Если мы жестко задаем ширину таблицы, скажем, сто пикселов, а в каждой строке таблицы находится по четыре ячейки, каждая шириной в тридцать пикселов, то, несмотря на явную установку ширины всей таблицы, она все равно составит не сто, а сто двадцать пикселов. Если же в какой-либо из ячеек будет находиться графическое изображение шириной больше заданных тридцати пикселов, то оно "растянет" собой ячейку, а с ней и весь столбец. Следовательно, еще более увеличится ширина всей таблицы. Сделано это для того, чтобы максимально точно отображать содержимое табличных ячеек,

без полной или частичной потери информации. При этом жестко заданные размеры таблицы сохраняются, если есть такая возможность.

Обратите внимание, что явного объекта, обозначающего столбец таблицы, нет. Количество столбцов рассчитывается браузером на основе анализа строк таблицы, а затем уже отображается вся таблица целиком. Таким образом, Web-страница, содержащая таблицы, будет отображаться не постепенно, по мере ее загрузки, а только тогда, когда браузер получит ее всю полностью.

Пора переходить к рассмотрению тэгов, реализующих таблицы. Таблица и все элементы, ее составляющие, находятся между стартовым тэгом `<table>` и его закрывающим двойником `</table>`. Таблица состоит из заголовка, реализуемого тэгом `<caption>`, группы столбцов, объявляемых с помощью тэгов `<col>` и `<colgroup>`, шапки и подвала, создаваемых тэгами `<thead>` и `<tfoot>`, соответственно, и группы строк, реализуемых тэгом `<tbody>`. Все остальные, более мелкие элементы таблицы размещаются уже внутри этих объектов.

Тэг `<table>` обладает достаточно обширным набором параметров, позволяющим устанавливать самые различные свойства таблицы.

Так уж сложилось, что вопросам ширины каких-либо элементов оформления Web-страниц мы уделяем гораздо больше внимания, чем установке высоты. Иными словами, люди значительно спокойнее относятся к вертикальной полосе прокрутки, нежели к горизонтальной, поэтому у таблицы нет параметра, устанавливающего высоту. Она рассчитывается исходя из размеров содержимого ячеек таблицы. Ширину таблицы мы можем устанавливать явно. Выполняется эта установка с помощью необязательного параметра `width`. Этому параметру мы можем присвоить процентное соотношение, абсолютное значение количества пикселов или кратное.

Для описания таблицы часто применяется параметр `border`, с помощью которого явно указывается ширина границы таблицы. В качестве значения данного параметра используется целое неотрицательное число, задающее ширину границы в пикселах. Если мы установим нулевое значение этого параметра, то граница таблицы будет невидима. Это и позволяет создавать невидимые таблицы, в ячейках которых размещаются элементы содержимого Web-страницы.

Параметр `cellspacing` определяет размер в пикселах между отдельными ячейками страницы. Похожий параметр `cellpadding` устанавливает размер отступа содержимого ячейки от ее границы. Таким образом, `cellspacing` устанавливает отступ вне ячейки, а `cellpadding` — внутри ячейки.

С помощью уже знакомого нам параметра `align` у нас есть возможность задать горизонтальное выравнивание таблицы. В качестве значений этого параметра может применяться одно из трех предустановленных ключевых слов: `left`, `center` и `right`, которые прижимают таблицу к левому краю родительского объекта, центрируют ее, или прижимают вправо, соответственно.

В самом простом варианте создания таблицы мы объявляем саму таблицу с помощью тэга <table>, затем несколько строк, а внутри этих строк ячейки. Это, повторюсь, самый простой способ, который не требует применения встроенных табличных объектов, упомянутых несколько выше. До них еще дойдет очередь, а пока рассмотрим пример того, как создаются и отображаются таблицы в HTML-документах.

Листинг 1.25

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Простейшая таблица</title>
  </head>
  <body>
  <p>Это обычный текст</p>
  <table border=5 cellpadding=7 cellspacing=10 align=center>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td>4</td><td>5</td><td>6</td>
  </tr>
  </table>
  </body>
</html>
```

Результат отображения этого кода в окне просмотра браузера показан на рис. 1.24.

В данном примере мы намеренно применили достаточно большие значения для толщины границы и отступов снаружи и внутри ячеек для того, чтобы наглядно продемонстрировать влияние этих параметров (см. рис. 1.24). Из листинга 1.25 видно, что строки таблицы объявляются парой тэгов <tr> и </tr>. Уже внутри этих тэгов мы описываем ячейки с помощью тэга <td> и его закрывающего двойника </td>, между которыми находится содержимое каждой конкретной ячейки. Сколько задано ячеек в строке, столько столбцов в таблице и отобразит браузер. Размер ячеек определяется исходя из размеров содержимого и установленных отступов. При этом, если содержимое какой-либо одной ячейки столбца больше по размерам, чем содержимое иных ячеек этого столбца, ширина всего столбца будет установлена по ширине самой большой ячейки.

Возможна ситуация, когда создатель Web-страницы в различных строках объявит разное количество ячеек, но и тогда браузер сможет отобразить таблицу, как это показано в следующем примере (листинг 1.26 и рис. 1.25).

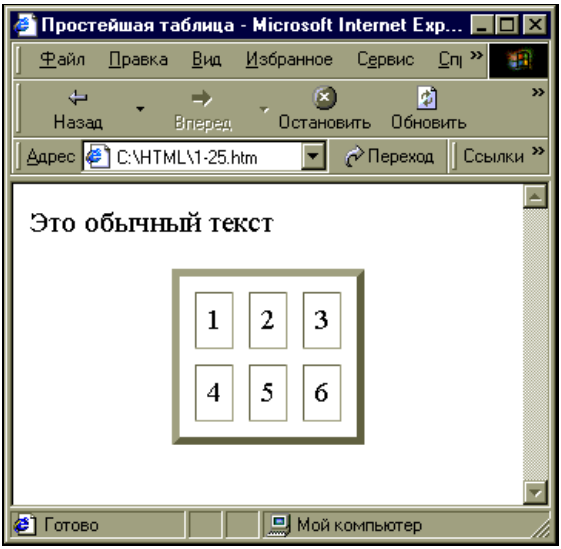


Рис. 1.24. Окно браузера с результатом отображения файла, приведенного в листинге 1.25

Листинг 1.26

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Простейшая таблица</title>
  </head>
  <body>
  <p>Это обычный текст</p>
  <table border=2 align=center>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td>4</td><td>5</td><td>6</td><td>7</td>
  </tr>
  <tr>
  <td>8</td><td>Длинная ячейка</td><td>10</td>
  </tr>
  </table>
  </body>
</html>
```

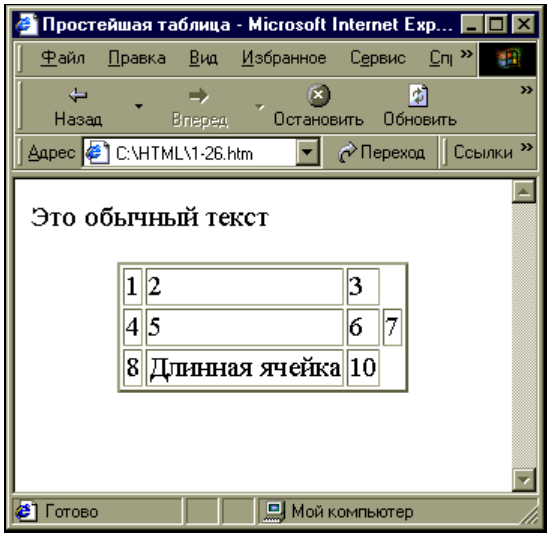


Рис. 1.25. Окно браузера с результатом отображения файла, приведенного в листинге 1.26

Мы описали самый простой способ создания таблиц, когда в ее состав входят только основные строки. Все можно сделать немного тоньше, если использовать различные дополнительные объекты, входящие в состав таблицы. Одним из таких объектов является заголовок, реализуемый с помощью тэга `<caption>`. Текст заголовка размещается между его стартовым тэгом и завершающим двойником `</caption>`. Тэг, объявляющий заголовок таблицы, обладает параметром `align`, позволяющим указывать расположение заголовка относительно самой таблицы. В качестве значения параметра может использоваться одно из четырех предустановленных ключевых слов: `top`, `bottom`, `left` и `right`. Значение `top` заставляет браузер отображать заголовок таблицы над ней самой. Значение `bottom` перемещает заголовок под таблицу. Значения `left` и `right` устанавливают, соответственно, левое и правое горизонтальное выравнивание заголовка, который при этом отображается над таблицей. По умолчанию, заголовок отображается сверху, а если не указывать явно выравнивание по горизонтали, то заголовок центрируется. Приведем пример использования этого тэга.

Листинг 1.27

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Таблица</title>
```

```
  </head>
  <body>
  <p>Это обычный текст</p>
  <table border=2 align=left>
  <caption align=right>Заголовок таблицы</caption>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td>4</td><td>5</td><td>6</td>
  </tr>
  </table>
  </body>
</html>
```

В данном случае мы использовали значение `right` для параметра `align`. Таким образом, заголовок размещается над таблицей и занимает по горизонтали пространство не большее, чем сама таблица; его текст прижат к правой границе этого пространства. Все это показано на рис. 1.26.

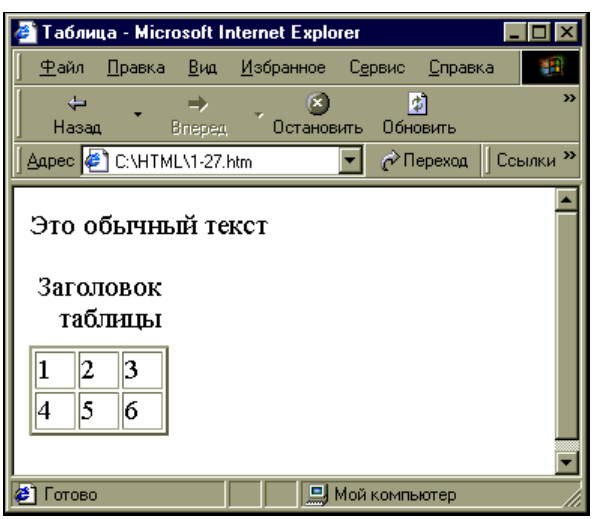


Рис. 1.26. Окно браузера с результатом отображения файла, приведенного в листинге 1.27

Как видно, заголовок таблицы, хоть и относится, строго говоря, к таблице, но отображается вне ее самой. Зато следующие три объекта, которые мы рассмотрим, являются составными частями таблицы.

В предыдущих примерах содержимое таблицы разбивалось на обычные равноценные строки. Как правило, таблицы содержат и явно выделенную верх-

нюю часть, именуемую шапкой, и четко обозначенную нижнюю часть, называемую подвалом, в которой обычно пишется сумма столбцов. Эти части обозначаются тэгами <thead> и <tfoot> соответственно. Если мы применяем эти тэги в объявлении таблицы, то нам необходимо явно описывать и ту часть таблицы, в которой размещаются обычные данные. Это описание выполняется с помощью тэга <tbody>. Естественно, все три только что упомянутых тэга применяются обязательно в паре со своими закрывающими двойниками. Рассмотрим на примере, как изменяется объявление таблицы в коде HTML-документа и отображение ее в браузере, благодаря использованию этих структурных элементов (листинг 1.28 и рис. 1.27).

Листинг 1.28

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Таблица</title>
  </head>
  <body>
  <p>Это обычный текст</p>
  <table border=2 align=left>
  <caption >Заголовок таблицы</caption>
  <thead>
  <tr>
  <td> Столбец 1</td><td>Столбец 2</td><td>Столбец 3</td>
  </tr>
  </thead>
  <tbody>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td>4</td><td>5</td><td>6</td>
  </tr>
  </tbody>
  <tfoot>
  <tr>
  <td>5</td><td>7</td><td>9</td>
  </tr>
  </tfoot>
  </table>
  </body>
</html>
```

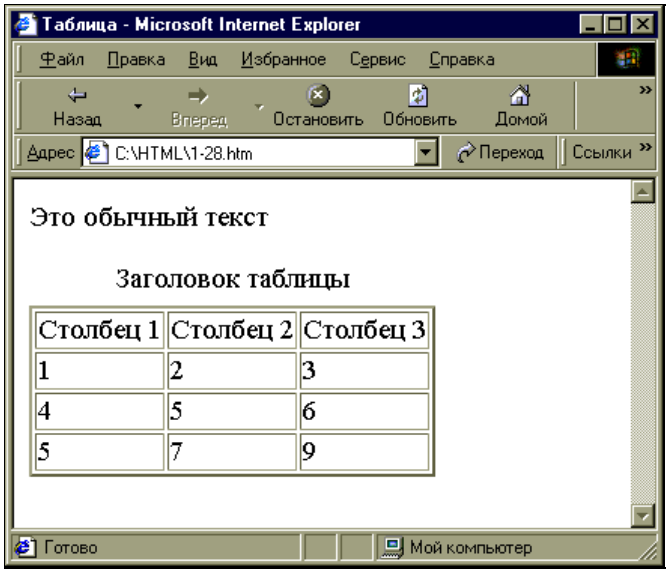


Рис. 1.27. Окно браузера с результатом отображения файла, приведенного в листинге 1.28

Из листинга видно, что содержимым всех рассматриваемых нами структурных частей таблицы по-прежнему являются строки, состоящие из отдельных ячеек.

Несмотря на кардинально изменившуюся структуру таблицы в коде HTML-документа, при отображении ничего принципиально нового не появилось. И это правильно, т. к. HTML старается передавать конкретные аспекты отображения на более низкий, детальный уровень иерархии объектов. Следовательно, если мы хотим, чтобы ячейки подвала, заголовка и тела таблицы отличались друг от друга по оформлению, мы должны либо самостоятельно установить эти правила отображения для каждой ячейки, либо воспользоваться стилевым оформлением.

Тем не менее, одно средство оформления все-таки встроено в эти тэги. С помощью параметров `align` и `valign` мы можем устанавливать выравнивание содержимого ячеек. Параметр `align` задает выравнивание содержимого ячеек данного раздела таблицы по горизонтали. В качестве значения этого параметра применяется одно из предустановленных ключевых слов. Рассмотрим их.

- Значение `left` прижимает текст и иное содержимое ячеек к левому краю ячейки. Применяется по умолчанию для основного раздела таблицы.
- Значение `right` выравнивает любое содержимое ячейки по правому краю.

- Значение `center` центрирует содержание ячеек, включенных в раздел, к которому применяется данный параметр. Используется по умолчанию для заголовков таблицы.
- Значение `justify` заставляет браузер отображать содержимое ячеек, равномерно растянув их по всей ширине ячейки.
- Значение `char` применяется тогда, когда в ячейках раздела отображается числовая информация в финансовом формате и необходимо выровнять ее не просто по левому или правому краю, а так, чтобы разделители целой и дробной части всех чисел находились на одном и том же уровне.

О последнем значении стоит поговорить особо. Мы сказали, что выравниваться будут не столько сами числа, сколько их десятичные разделители. А какие символы считаются десятичными разделителями? Если для всего документа явно установлен язык текстового содержимого, то браузер сам поймет, какой символ — десятичный разделитель. Но нет никакой гарантии, что мы для этих целей будем использовать именно тот символ, который распознается браузером. К счастью, у нас есть возможность указать, какой символ мы применяем для разделения целой и дробной части чисел. В этом случае используется параметр `char`, значением которого и является искомый символ, но данную возможность стандарта HTML поддерживают далеко не все браузеры.

Теперь перейдем к параметру, позволяющему явно устанавливать вертикальное выравнивание содержимого ячеек, входящих в состав какого-либо блока таблицы. Он носит наименование `valign`. Как и у параметра горизонтального выравнивания, его значением может быть только одно ключевое слово из предопределенного набора.

- Значение `top` прижимает содержимое ячеек к их верхним границам.
- Значение `middle` центрирует по вертикали содержимое ячеек, входящих в объявляемый блок. Это значение используется по умолчанию.
- Значение `bottom` прижимает содержимое ячеек к их нижним граням.
- Значение `baseline` имеет смысл применять только для ячеек с текстовым содержимым. В этом случае, базовые линии первых строк текстового содержимого ячеек, входящих в объявляемый раздел таблицы, выравниваются по базовой линии первой строки текста содержимого первой ячейки. Напомню, что базовой линией текста называется невидимая линия, на которой располагаются основания большинства символов применяемого алфавита. У некоторых букв, таких как "g" в латинском алфавите или "р" — в русском, существуют так называемые "хвостики", которые уходят вниз за базовую линию.

С помощью рассмотренных нами блоков таблицы мы можем структурировать ее содержимое и немного управлять отображением содержимого ячеек, входящих в эти блоки. Но что делать, если нам необходимо установить еди-

ные правила оформления для одного или нескольких столбцов таблицы? Как мы знаем, нет нужды объявлять их специально, т. к. столбцы формируются из ячеек, входящих в строки. Тем не менее, в стандарте HTML 4.1 существуют тэги, которые позволяют объявлять отдельные столбцы и их группы. Для объявления одного столбца применяется тэг `<col>`, а если нам необходимо несколько столбцов объединить в одну группу, лучше использовать тэг `<colgroup>`.

Следует учитывать, что если мы используем эти тэги для объявления столбцов, то количество столбцов в таблице будет рассчитываться именно на их основе. То есть, нам необходимо каждый столбец описать подобным образом, а не какую-либо их группу. Браузер при отображении таблицы сначала анализирует ее объявление для поиска тэгов, создающих столбцы. Если они есть, то количество столбцов в таблице рассчитывается на их основе. Если таких тэгов нет, то количество столбцов рассчитывается исходя из количества ячеек в строках.

С помощью тэга `<colgroup>` мы объявляем группу столбцов. Каждый столбец этой группы мы можем явно объявить, используя тэг `<col>`, но это делать необязательно, если столбцы имеют одинаковое оформление. Оформление группы столбцов задается с помощью параметров тэга `<colgroup>`. К этому тэгу применимы параметры `align` и `valign`, которые мы рассматривали чуть ранее, с теми же возможными значениями. Кроме того, часто используются еще два параметра.

- Параметр `span` показывает, какое количество столбцов находится в данной группе. Значением этого параметра может быть положительное целое число. По умолчанию используется единичное значение, т. е. в группу входит всего один столбец.
- Параметр `width` позволяет устанавливать единую ширину для всех столбцов, входящих в данную группу. Мы можем задавать точное абсолютное значение в пикселах, процентное соотношение или "кратные размеры", как мы описывали их в одном из предыдущих разделов данной главы. Так, если мы хотим, чтобы каждый столбец устанавливал ширину минимально необходимую для размещения содержимого ячеек, следует использовать конструкцию `width="0*"`.

Если мы хотим указать, что в таблице будет сорок столбцов, каждый шириной в двадцать пикселов, необходимо использовать следующий фрагмент кода:

```
<colgroup span="40" width="20">
</colgroup>
```

Никогда не следует забывать о том, что нам заранее неизвестно, какое разрешение экрана будет у пользователя, загружающего нашу Web-страницу.

Отдельный столбец, входящий в группу, объявляется с помощью тэга `<col>`. Этот тэг обладает тем же набором параметров, что и рассмотренный нами

тэг <colgroup>. Вот только параметр span здесь имеет несколько иное значение. Как мы знаем, некоторые ячейки могут объединять несколько ячеек из соседних столбцов. А тэги <col> могут создавать столбцы, которые состоят из нескольких столбцов, т. е. одна ячейка такого столбца может содержать несколько тэгов <td>. Значение параметра span как раз и указывает, сколько именно столбцов будут объединяться в одном. Приведем пример использования данного параметра.

Если нам необходимо создать таблицу с тремя столбцами, стоит в код HTML-документа вставить следующую конструкцию:

```
<TABLE>
<COLGROUP>
<COL>
<COL span="2">
</COLGROUP>
<TR><TD> ...
...Определение строк...
</TABLE>
```

Между тэгами <colgroup> и </colgroup> мы вставили два тэга <col>. Легко заметить, что это объявляющие тэги, которые не требуют использования закрывающего двойника для себя. В данном фрагменте мы создали таблицу с тремя столбцами. Первый столбец объявлен с помощью первого тэга <col>, а вторые два столбца объединены в единую группу вторым тэгом <col> с параметром span, значение которого установлено равным двум.

Мы знаем, как объявлять столбцы. Но после объявления столбцов мы все равно переходим к заполнению таблицы, и там уже без тэгов, создающих строки, не обойтись. В предыдущих примерах мы видели, что строки создаются с помощью пары тэгов <tr> и </tr>, между которыми находятся тэги, объявляющие отдельные ячейки. Тэг <tr> обладает целым рядом параметров, которые нам уже знакомы. Помимо всеобщих параметров идентификации, есть и параметры, регулирующие отображение информации, находящейся в ячейках описываемой строки. Перечислим их.

- Параметр align позволяет регулировать горизонтальное выравнивание содержимого ячеек строки.
- Параметр valign задает вертикальное выравнивание для содержимого ячеек строки.
- Параметр bgcolor позволяет задавать цвет фона для ячеек, входящих в объявляемую строку.

Ячейки таблицы мы можем создавать не только с помощью тэга <td>, но и применяя тэг <th>. Тэг <th> предназначен для создания ячеек верхних строк таблицы, которые содержат наименования столбцов. В листинге 1.29 показан пример их использования.

Листинг 1.29

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Таблица</title>
  </head>
  <body>
  <p>Это обычный текст</p>
  <table border=2 align=left>
  <caption >Заголовок таблицы</caption>
  <thead>
  <tr>
  <th> Столбец 1</th><th>Столбец 2</th><th>Столбец 3</th>
  </tr>
  </thead>
  <tbody>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td>4</td><td>5</td><td>6</td>
  </tr>
  </tbody>
  </table>
  </body>
</html>
```

Результат отображения данного кода в окне просмотра браузера показан на рис. 1.28.

В листинге, в так называемой "шапке" таблицы, мы объявили ячейки с помощью тэгов `<th>`, а на иллюстрации текстовое содержимое этих ячеек выделено полужирным шрифтом. Впрочем, из всех табличных объектов ячейки обладают, пожалуй, самым обширным набором параметров, который позволяет нам настраивать внешний вид каждой ячейки отдельно.

- Параметр `rowspan` указывает, на сколько строк по вертикали растянута искомая ячейка. Благодаря этому параметру, мы имеем возможность объединять соседние ячейки, находящиеся в одном столбце таблицы. Значением данного параметра является целое неотрицательное число. По умолчанию используется единичное значение. Если мы для этого параметра установим нулевое значение, то будут объединены ячейки в столбце, начиная с текущей до самой нижней ячейки.

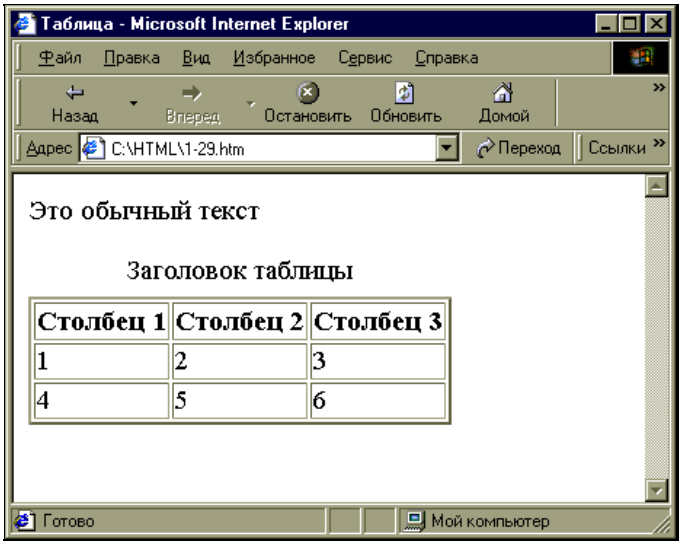


Рис. 1.28. Окно браузера с результатом отображения файла, приведенного в листинге 1.29

- Параметр `colspan` подобен только что рассмотренному параметру `rowspan`, но он определяет количество ячеек, объединяемых по горизонтали, в пределах одной строки. Как и прежде, значение по умолчанию — единица, а нулевое значение объединяет ячейки, начиная с текущей и до конца строки таблицы.
- Параметр `nowrap` применяется без каких-либо значений. Его присутствие в тэге, объявляющем ячейку, означает, что текстовое содержимое данной ячейки необходимо располагать в одной строке, при необходимости увеличивая ширину ячейки для того, чтобы содержимое было отображено полностью.
- Параметр `width` предназначен для указания рекомендованной ширины ячейки. Значением этого параметра является число, обозначающее ширину в пикселах, процентное соотношение или коэффициент кратности. Если содержимое ячейки, текстовое или графическое, по своим размерам превышает установленную ширину ячейки, то ячейка принудительно растягивается до необходимого размера.
- Параметр `height` используется для указания рекомендованной высоты ячейки. Возможные значения и порядок обработки этого параметра браузерами полностью совпадают с только что рассмотренным параметром `width`.
- Параметр `align` задает горизонтальное выравнивание содержимого ячеек. В качестве значения используется одно из предустановленных ключевых слов, список которых мы рассматривали ранее в этом разделе главы.

- Параметр `valign` предназначен для установки вертикального выравнивания содержимого ячейки. Возможные значения этого параметра мы также рассматривали несколько ранее в этом разделе.
- Параметр `dir` позволяет задавать направление отображения текста. В некоторых языках принято написание текстовой строки не слева направо, как мы привыкли, а справа налево, поэтому необходимо иметь возможность явно указывать направление вывода текста. Если отображаемый текст должен выводиться слева направо, следует использовать значение `LTR`, установленное по умолчанию, в противном случае применяется значение `RTL`.

После ознакомления со списком хотелось бы увидеть, как действуют эти параметры. Приведем пример, в котором это будет достаточно хорошо видно (листинг 1.30 и рис. 1.29).

Листинг 1.30

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Таблица</title>
  </head>
  <body>
  <table border=2 width="100%">
  <caption >Заголовок таблицы</caption>
  <thead>
  <tr>
  <th bgcolor=lime rowspan=2> </th><th> Столбец 1</th><th>Столбец
2</th><th>Столбец 3</th>
  </tr>
  <tr>
  <th colspan=2>Объединенная ячейка</th><th>Столбец 3</th>
  <tr>
  </thead>
  <tbody>
  <tr>
  <td width=100> </td><td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
  <td> </td><td>4</td><td>5</td><td>6</td>
  </tr>
  </tbody>
  </table>
  </body>
</html>
```

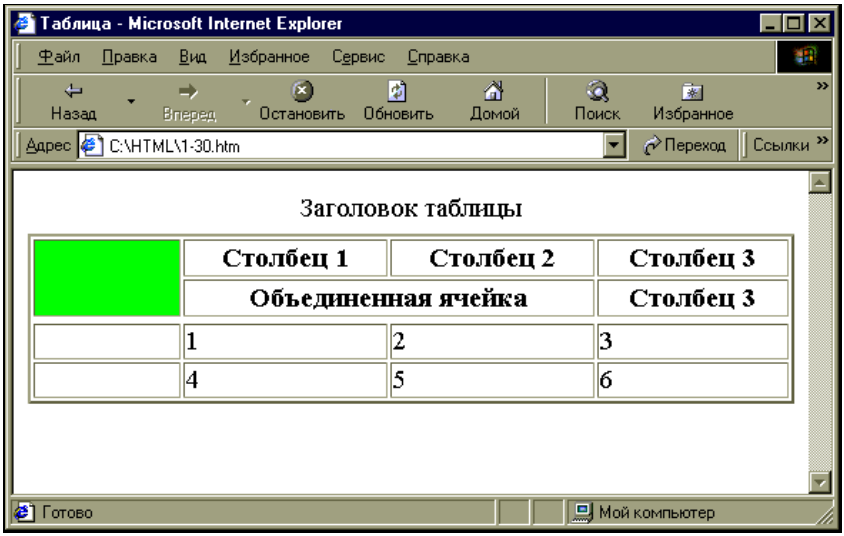


Рис. 1.29. Окно браузера с результатом отображения файла, приведенного в листинге 1.30

Теперь проанализируем, что у нас получилось. Самую первую ячейку мы объединили по вертикали, и явно задали для нее цвет фона. Этот цвет распространился и на присоединенную к ней ячейку из второй строки. Для всей таблицы мы задали ширину в процентах, поэтому все столбцы растянуты равномерно. Все, за исключением первого, т. к. для первой ячейки в третьей строке мы жестко установили ширину в сто пикселов, поэтому весь столбец у нас отображен с такой шириной. Во второй строке таблицы мы создали ячейку, объединяющую две соседние ячейки в текущей строке. На иллюстрации видно, что блок ячеек, сформированных с помощью тэга `<th>`, немного отделен по вертикали от основного блока таблицы, состоящего из ячеек, объявленных тэгом `<td>`. Кроме того, мы во всех ячейках, где нет текстового содержимого, вставили символ неразрывного пробела. Сделано это не случайно. Дело в том, что браузер Netscape Navigator ранних версий некорректно отображает столбцы таблиц, в которых нет содержимого, поэтому следует всегда вставлять неразрывные пробелы в пустые ячейки таблиц.

Вообще, прием установки ширины таблицы в процентах и жесткого задания ширины первого столбца широко используется в Web-дизайне. Обычно меню навигации по сайту размещается в левой части экрана, поэтому Web-дизайнер, как правило, заключает все содержимое страницы в одну большую таблицу с двумя ячейками. Ширина левой ячейки задается жестко, в пикселах, а все остальное пространство окна просмотра отводится под отображение основного содержимого страницы. При этом если это содержимое тоже имеет жесткую структуру, то его можно поместить в таблицу, которая вся будет размещаться в одной ячейке первичной таблицы. Следует

лишь внимательно относиться к созданию вложенных таблиц, т. к. ранние версии браузера Netscape Navigator при обилии таблиц, вложенных друг в друга, могут просто "зависать".

Фреймы

Как известно, в одно окно просмотра браузера мы не можем одновременно загрузить два HTML-документа. Таким образом, если у нас на всех страницах есть одинаковое меню навигации, то абсолютно одинаковый фрагмент кода пользователь будет загружать каждый раз при переходе с одной Web-страницы сайта на другую. Даже если меню невелико, это не самый удобный вариант, т. к. каналы связи российских пользователей не обладают большой пропускной способностью, и загрузка меню может занять как минимум несколько секунд. Есть ли у нас возможность сделать так, чтобы неизменяемые элементы сайта всегда оставались в окне просмотра пользователя без перезагрузки? Есть.

Мы можем создать документ, который разобьет одно окно просмотра на несколько прямоугольных областей, в каждой из которых будет отображаться один HTML-документ. Эти прямоугольные области, каждая из которых, по сути, отдельное окно просмотра, и называются *фреймами*. Таким образом, мы можем заставить левое и/или верхнее меню навигации постоянно находиться в окне просмотра, а перезагружать только ту часть сайта, которая необходима.

Как любое окно просмотра, фреймы могут обладать полосами прокрутки, которые позволяют пользователю увидеть все содержимое фрейма, если оно не умещается полностью в видимой зоне. Именно эта особенность фреймов и вызывает постоянные споры между Web-мастерами. Кто-то заявляет, что подобные элементы управления не должны находиться во внутреннем пространстве основного окна просмотра, т. к. они занимают бесценное пространство и рассеивают внимание пользователя. Кто-то утверждает, что эти недостатки — не слишком большая плата за облегчение перезагрузки Web-страниц в условиях медленных каналов связи. Так или иначе, вопрос о возможности применения фреймов каждый должен решать самостоятельно.

В HTML предусмотрено два вида фреймов: обычные и так называемые "плавающие". Если мы применяем обычные фреймы, то создается документ, который все окно просмотра разбивает на фреймы, и в них уже отображаются те или иные HTML-документы. Если же мы используем плавающий фрейм, то его можно включать в обычный HTML-документ без каких-либо особых ухищрений. Разницу между двумя этими типами фреймов мы увидим на примере.

Документ с фреймовой структурой создается с помощью тэга `<frameset>` и его закрывающего двойника `</frameset>`. Внутри этих двух тэгов размеща-

ются конструкции объявления отдельных фреймов и информация, отображаемая в окне просмотра браузера, если тот не распознает фреймы. Отдельные фреймы создаются с помощью пары тэгов <frame> и </frame>. Информация, отображаемая в окне просмотра браузера, если тот не поддерживает фреймы, задается в HTML-документе тэгом <noframes>. Но все по порядку.

Функцией тэга <frameset> является разбиение окна просмотра на несколько частей. Для этого у тэга есть два основных параметра: rows и cols, которые указывают количество и размеры фреймов по вертикали и горизонтали. При этом задается прямоугольная табличная структура. Если нам необходимо сделать более детальное разбиение, например, один фрейм в левой части окна по вертикали, и два тэга, отделенных друг от друга горизонтальной границей в правой части окна, то можно создать два тэга <frameset>, один внутри другого. Опишем параметры тэга <frameset> более детально.

- Параметр cols предназначен для указания количества и размеров фреймов по горизонтали в окне просмотра. В качестве значения параметра используется список размеров фреймов, разделенных запятыми. Как обычно, для указания размера мы можем использовать абсолютные и кратные величины, а также процентные соотношения. По умолчанию используется значение 100%, т. е. основное окно просмотра по вертикали не разбивается.
- Параметр rows позволяет указывать количество и размеры фреймов по вертикали в окне просмотра. Специфика задания значений полностью совпадает с параметром cols.

Рассмотрим на примере, как будет выглядеть создание HTML-документов с фреймовой структурой. Предположим, что нам необходимо один фрейм слева выделить под навигационное меню, а оставшееся пространство использовать для отображения информации с основных страниц Web-сайта. В этом случае мы разделим окно просмотра на две части. Для меню выделим фрейм шириной в сто пятьдесят пикселов, а оставшееся пространство должен занять второй фрейм. Подобная структура Web-страницы реализуется кодом, приведенным в листинге 1.31.

Листинг 1.31

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Фреймы</title>
  </head>
  <frameset cols="150,*">
  <frame src="1-27.htm">
  <frame src="1-30.htm">
```

```
  <noframes>
  <p>К сожалению, ваш браузер не поддерживает фреймы. Воспользуйтесь
↳более свежей версией ПО</p>
  </noframes>
  </frameset>
</html>
```

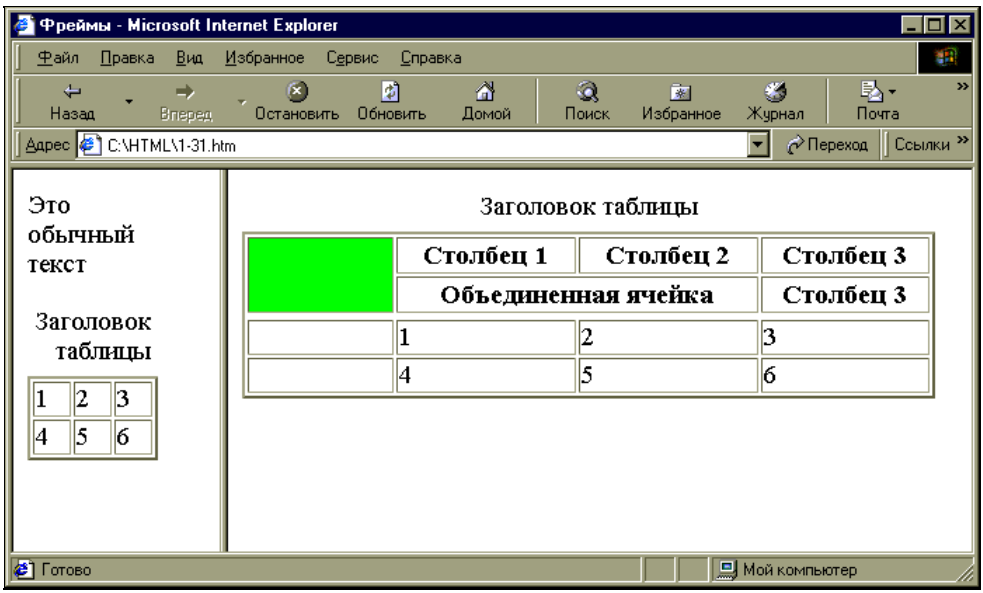


Рис. 1.30. Окно браузера с результатом отображения файла, приведенного в листинге 1.31

Проанализируем внимательно приведенный код HTML-документа и то, как был отображен браузером этот документ. Прежде всего, следует обратить внимание, что в листинге отсутствуют тэги `<body>` и `</body>`. Эти тэги говорят о том, что все находящееся между ними является отображаемым кодом, а тэг `<frameset>` сигнализирует браузеру, что в данном документе приведена лишь структура разбиения окна просмотра.

В тэге `<frameset>` мы использовали единственный параметр `cols`. Значением его был список из двух обозначений размеров. Из листинга видно, что первый, т. е. левый фрейм обладает шириной в сто пятьдесят пикселов. Для второго фрейма мы установили ширину в виде кратной величины. Поскольку мы не указали, какому именно числу кратна эта ширина, фрейм занял все свободное место окна, чего мы и добивались.

Между стартовым и закрывающим тэгами `<frameset>` мы разместили два тэга `<frame>`. Обстоятельный и подробный разговор о них еще впереди, а

пока ограничимся замечанием, что эти тэги объявляют каждый фрейм отдельно и задают их свойства. В нашем случае мы использовали лишь один обязательный параметр `src`, значением которого является URL того HTML-документа, который будет отображаться в данном фрейме.

Кроме того, мы использовали тэг `<noframes>`. Между ним и его закрывающим двойником `</noframes>` располагается HTML-код сообщения, которое будет отображаться в окне просмотра браузера, если он не поддерживает фреймовую технологию. Сейчас, конечно, встретить подобный браузер чрезвычайно трудно, тэг остался в спецификации HTML с давних времен текстовых браузеров, но использовать его все-таки стоит хотя бы из соображений вежливости по отношению к пользователям.

На иллюстрации видно, что окно просмотра действительно было разбито на две части, и в каждой из них отображался один из созданных нами ранее HTML-файлов. При этом пользователь может самостоятельно изменять размеры фреймов, т. к. граница между ними, называемая также *сплиттером*, подвижна. Достаточно навести на нее курсор мыши, нажать основную кнопку, и, не отпуская ее, перенести границу в необходимое место.

Единственным ограничением размещения фреймов служит четко выраженная табличная структура набора фреймов. А что делать, если нам захочется, чтобы меню в левой части окна просмотра было само разбито на два фрейма? Классическое использование фреймовой структуры не позволяет сделать это, но можно прибегнуть к помощи вложенных структур.

Для того чтобы создать разбиение основного окна просмотра на три фрейма, два из которых расположены в одной колонке, друг под другом, а третий занимает все остальное свободное пространство, следует использовать следующий фрагмент кода:

```
<frameset cols="20%,*">
<frameset rows="*,*">
<frame src="file1.htm">
<frame src="file2.htm">
</frameset>
<frame src="file3.htm">
</frameset>
```

В этом примере видно, как один блок `<frameset>` мы встраиваем внутрь другого такого же блока. И естественно, следует использовать тэги `<frame>`. Кстати, их мы еще не рассмотрели подробно. Самое время это сделать.

Тэг `<frame>` предназначен для установки свойств отдельного фрейма. Поскольку ширина и высота фрейма устанавливаются в конструкции `<frameset>`, нам остается описать остальные свойства.

- Параметр `name` позволяет устанавливать уникальное имя фрейма. Не следует путать его с параметром `id`. Имя, которое мы задаем с помощью па-

раметра name, используется в тэгах гиперссылок, когда необходимо загрузить документ не в родительский фрейм, а в какой-либо другой.

- Параметр src, как мы уже видели в предыдущем листинге, применяется для задания URL того HTML-документа, который должен быть отображен в описываемом фрейме.
- Параметр frameborder используется для того, чтобы указать, будет ли отображаться граница данного фрейма или нет. Дело в том, что создавать видимую границу не обязательно. Значениями данного параметра могут быть нуль или единица. Единичное значение, установленное по умолчанию, обозначает, что данный фрейм будет иметь видимую границу. При использовании нулевого значения граница фрейма остается невидимой.
- Параметр marginwidth позволяет задавать ширину полей данного фрейма в пикселах.
- Параметр marginheight предназначен для установки размеров полей по вертикали в пикселах для данного фрейма.
- Параметр noresize следует применять для запрета изменения размеров фрейма. Если мы включаем его в состав тэга <frame>, то пользователь не сможет изменять размеры данного фрейма, перемещая его границы. Параметр не имеет значения.
- Параметр scrolling позволяет создателю Web-страницы управлять отображением полос прокрутки для данного фрейма. В качестве значения используется одно из трех предустановленных ключевых слов. Значение auto, установленное по умолчанию, обозначает, что полосы прокрутки у фрейма будут появляться только в том случае, если содержимое данного фрейма не будет полностью помещаться в отображаемой области. Значение yes указывает браузеру, что для данного фрейма необходимо постоянно отображать полосы прокрутки, вне зависимости от того, насколько велика его отображаемая область и какая часть содержимого фрейма в ней помещается. Значение no заставляет браузер отображать фрейм вообще без полос прокрутки. Несмотря на то, что полосы прокрутки — это самый раздражающий пользователя компонент, не следует применять значение no без особых на то причин. Еще раз повторю, что нам заранее неизвестно, какое разрешение монитора установлено у пользователя, загрузившего нашу страницу, и каков размер окна просмотра браузера. Если на удаленном компьютере не окажется достаточно места для отображения содержимого фрейма полностью, и будут отсутствовать полосы прокрутки, пользователь вообще не сможет увидеть скрытую информацию.

Теперь, когда мы рассмотрели параметры тэга <frame>, следует сделать маленькое дополнение, объясняющее одно незначительное ограничение параметра src, тесно связанное с дополнительным тэгом <noframes>. Дело в том, что между этим тэгом и его завершающим двойником </noframes> мы раз-

мещаем некое содержимое, которое будет отображено в том случае, если браузер пользователя не поддерживает отображение фреймов. Это содержимое тоже может быть структурировано с помощью тэгов HTML. Следовательно, там могут быть использованы гиперссылки и закладки, называемые также "якорями". Так вот, если мы в этом фрагменте создадим такую закладку, то мы не можем для какого-либо из фреймов значением параметра `src` сделать URL данной закладки.

Кроме того, мы можем указать фрейм, в который загрузка HTML-документа будет происходить по умолчанию. Как мы знаем, тэг гиперссылки `<a>` обладает параметром `target`, в качестве значения которого записывается имя фрейма. В этом фрейме и будет отображаться содержимое HTML-документа, на который указывает гиперссылка.

Мы не рассматривали отдельно тэг `<noframes>`, но данный тэг не обладает какими-либо уникальными параметрами, кроме общераспространенных, а механизм его применения мы могли видеть в предыдущих примерах. Не будем на нем долго задерживаться.

До сих пор мы описывали обычные фреймы, которые полностью разбивают окно просмотра браузера на отдельные области, и для использования которых необходимо применять документы со специализированной структурой. Но есть и другой вид фреймов, которые можно просто вставлять в обычный HTML-документ как стандартный объект. Больше всего это похоже на вставку графического изображения. Для вставки подобного встроенного фрейма используется тэг `<iframe>`. В отличие от процедуры вставки графики и иных объектов, для встроенных фреймов необходимо применять и закрывающий тэг `</iframe>`. Приведем пример включения встроенного фрейма в обычный HTML-документ и посмотрим, как выглядит этот документ, при просмотре его в браузере.

Листинг 1.32

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Фреймы</title>
  </head>
  <body>
  <p>Текст со встроенным <iframe src="1-25.htm"> </iframe>фреймом</p>
  </body>
</html>
```

Как видно из листинга 1.32 и рис. 1.31, процедура использования встроенных тэгов проста. В листинге мы как всегда использовали простейший

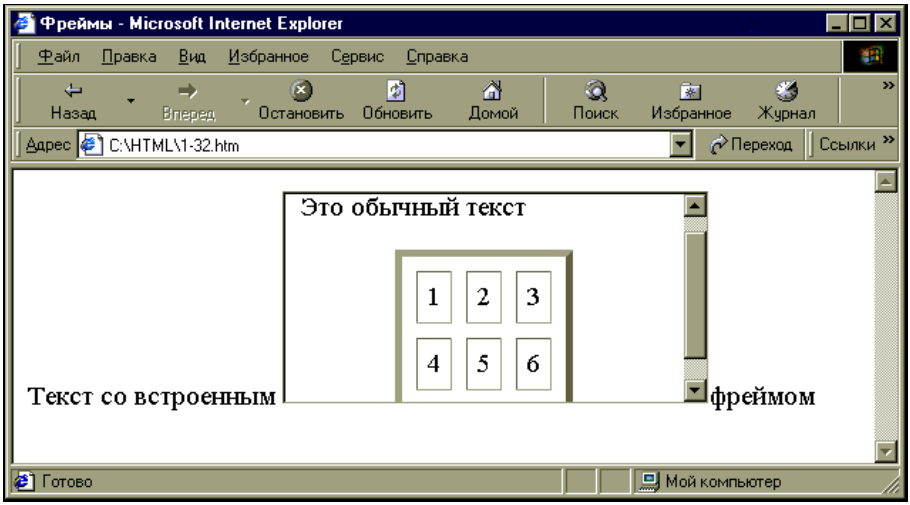


Рис. 1.31. Окно браузера с результатом отображения файла, приведенного в листинге 1.32

вариант по умолчанию, а ведь фрейм, пусть даже и встроенный, должен иметь достаточно обширный список свойств отображения, которые регулируются с помощью параметров. Их мы сейчас и рассмотрим.

- Параметр `name` позволяет задавать имя встроенного фрейма. Значением параметра является обычная текстовая строка. После этого на данный фрейм можно указывать в гиперссылках. Таким образом, мы получаем возможность динамического изменения содержимого встроенного фрейма.
- Параметр `src` предназначен для установки содержимого встроенного фрейма. Значением параметра является URL того HTML-документа, который будет загружен в этот встроенный фрейм.
- Параметр `frameborder` наравне со многими другими параметрами явно унаследован из обычных фреймов. Точно так же, как и там, он указывает браузеру, следует или нет отображать границу у фрейма. В качестве значений используются нуль и единица. Единичное значение, установленное по умолчанию, обозначает, что границу встроенного фрейма необходимо отображать. Нулевое значение делает границу невидимой.
- Параметры `marginwidh` и `marginheight` задают ширину и высоту полей встроенного фрейма соответственно. Значением параметров являются числа, выражающие размеры в пикселах.
- Параметр `scrolling` регулирует отображение полос прокрутки встроенного фрейма. В качестве значения используется одно из трех предустановленных ключевых слов. Значение `auto`, установленное по умолчанию, означает, что полосы прокрутки будут появляться только в том случае,

когда размеры содержимого фрейма больше, чем сам фрейм. Значение `yes` принудительно отображает полосы прокрутки в любом случае, а значение `no` запрещает браузеру отображать эти полосы прокрутки для данного фрейма вообще.

- Параметр `align` позволяет нам устанавливать выравнивание встроенного фрейма по вертикали или горизонтали относительно остального содержимого Web-страницы. Возможные значения точно такие же, как у одноименного параметра тэга `<img>`, поэтому не стоит их еще раз повторять. Результат действия этих значений тоже был нами рассмотрен ранее.
- Параметр `height` предназначен для явной установки размеров встроенного фрейма по вертикали. Размер может быть задан в пикселах, в процентах или кратной величиной.
- Параметр `width` позволяет задавать ширину встраиваемого фрейма. Значение его задается так же, как у предыдущего параметра.

На этом мы заканчиваем рассмотрение фреймов. Мы знаем о них все, что необходимо знать для эффективного их использования.

Встраиваемые объекты

Для того чтобы обогатить создаваемые Web-страницы дополнительными возможностями, мы можем вставить в их содержимое специализированные функциональные элементы. Подобными элементами могут быть Java-апплеты, элементы ActiveX, Flash-ролики и прочие дополнительные функциональные элементы. Как их обрабатывает браузер, нас заботить не должно. Он все равно сделает это правильно. Нас интересует, как их внедрять в Web-страницы.

До тех пор, пока мы не научились сами создавать функциональные объекты, пользуясь какой-либо достаточно серьезной технологией, мы можем использовать подобные компоненты сторонних производителей. Благо такие компоненты в изобилии представлены в Сети. Главное — их найти и правильно внедрить в Web-страницу.

Для вставки некоего функционального объекта следует воспользоваться тэгом `<object>`. У него есть и закрывающий двойник `</object>`. Между этими двумя тэгами обычно размещается список дополнительных данных, которые передаются объекту в качестве параметров, и некий текст, который выводится на экран, если браузер все-таки не сможет правильно обработать внедренный объект. Необходимо различать свойства объекта, которые мы задаем с помощью параметров тэга, и дополнительные данные, которые передаются как параметры самого объекта специализированными тэгами.

Например, если мы хотим вставить в свою Web-страницу некий элемент ActiveX, демонстрирующий изменение некоторой зависимости на графике,

то нам потребуется, помимо внедрения самого объекта, установить еще и начальные данные. Для этого придется воспользоваться приблизительно следующей конструкцией:

```
<object
classid="clsid:EB39F965-2374-11D3-85F1-F21B86A4754D"
codebase="http://www.mysite.com/scripts/myactx.ocx"#version=1,0,0,0
width=500 height=300 align=center>
<param name="start" value="0" valuetype="data">
<p>К сожалению, Ваша операционная система не поддерживает технологию
↳ActiveX</p>
</object>
```

В данном примере мы при объявлении внедренного объекта использовали в объявляющем тэге несколько параметров. Параметр `classid` применяется для установки идентификатора внедряемого элемента, а параметр `codebase` задает URL файла, в котором содержится внедряемый объект. Обычно для распространяемых объектов подобные параметры детально описываются в сопроводительном тексте. Чуть позже мы подробно рассмотрим все параметры этого тэга. После тэга `<object>` мы поместили тэг `<param>`, с помощью которого задали начальные данные для внедренного объекта. Проще всего рассматривать подобные передаваемые параметры как переменные. На каждую передаваемую переменную требуется один тэг `<param>`. Теперь перейдем к рассмотрению многочисленных параметров тэга `<object>`.

- Параметр `classid`, как мы уже знаем, применяется для установки уникального идентификатора внедряемого элемента. Часто его необходимо использовать, т. к. внедряемый элемент сохраняется в локальной системе пользователя, как, например, элементы ActiveX.
- Параметр `codebase` предназначен для указания URL, ссылающегося на некий файл или каталог, в котором располагаются все необходимые для функционирования внедряемого элемента файлы.
- Параметр `data` применяется для определения местоположения графических изображений или иных блоков данных, которые участвуют в функционировании встраиваемого объекта. В качестве значения параметра используется URL, указывающий на файл с требуемым блоком данных.
- Параметр `type` задает тип данных, на которые указывает предыдущий параметр. В качестве значения этого параметра используется одно из стандартных наименований типов данных, определенных в протоколе HTTP.
- Параметр `codetype` позволяет описать тип подключаемого объекта. Рекомендуется использовать данный параметр в тех случаях, когда информации, находящейся в параметре `classid`, недостаточно для того, чтобы четко определить тип внедряемого элемента.

- Параметр `archive` указывает местоположение архивных данных, которые имеют отношение к внедренному объекту, например, его обновления. В некоторых случаях это позволяет сократить общее время загрузки новой версии внедряемого объекта. В качестве значения данного параметра используется список URL, разделенных пробелами.
- Параметр `declare` не имеет значений. Если он входит в состав тэга `<object>`, то внедряемый объект просто объявляется, но не вставляется в Web-страницу. Обычно параметр применяется тогда, когда у нас есть несколько тэгов `<object>`, вставленных друг в друга.
- Параметр `standby` позволяет задавать текст, который будет отображаться на месте объекта, пока он сам загружается из Сети. В качестве значения параметра используется обычная текстовая строка.
- Параметр `height` предназначен для явного указания высоты объекта. В качестве значения параметра используется количество пикселов, абсолютное или кратное, или процентное соотношение.
- Параметр `width` позволяет явно задавать ширину внедряемого объекта. В качестве значения используется один из стандартных вариантов описания размеров.
- Параметр `usemap` применяется, если внедряемый объект — графическое изображение, которое предназначено для использования в качестве сегментированной карты. Применение сегментированной графики мы рассматривали в разделе, посвященном гиперссылкам. В качестве значения параметра указывается имя карты ссылок, заданное в параметре `name` тэга `<map>`, объявляющего подключаемую карту сегментов.
- Параметр `name` позволяет задавать уникальное идентифицирующее имя конкретного внедряемого объекта.
- Параметр `tabindex` указывает порядковый номер объекта в последовательности элементов управления, размещенных на Web-странице, переход между которыми производится с помощью клавиши табуляции.
- Параметр `hspace` задает размеры свободного пространства по горизонтали между встроенным объектом и остальным содержимым Web-страницы. В качестве значения используется число, обозначающее количество пикселов в искомом отступе.
- Параметр `vspace` определяет отступы по вертикали. В качестве значения используется все то же количество пикселов.
- Параметр `border` задает толщину границы, обрамляющей внедренный объект. В качестве значения используется количество пикселов.
- Параметр `align` позволяет указать вертикальное или горизонтальное выравнивание объекта относительно остального содержимого Web-страницы. В качестве значения используется одно из уже знакомых нам пре-

дустановленных ключевых слов: bottom, middle, top, left, right. Механизм их действия подробно описывался в разделе, посвященном использованию графики.

Если встраиваемому объекту необходимо передавать начальные данные для работы, то для этих целей применяется тэг <param>, который мы уже упоминали ранее. Он помещается между тэгами <object> и </object>. Если мы передаем данные объекту, то он воспринимает их как переменные. Следовательно, нам необходимо задать имя этой переменной, чтобы объект мог правильно ее распознать, и значение переменной. Это минимальные требования. Мы можем сделать и больше, воспользовавшись параметрами тэга <param>. Всего этих параметров — пять. Помимо общего идентифицирующего параметра id, есть и четыре специфичных.

- Параметр name является обязательным для тэга <param>. Значение этого параметра устанавливает имя передаваемой переменной. Значением параметра является текст. Так как существуют самые различные методы создания встраиваемых объектов, то может получиться так, что наименование переменной будет чувствительно к регистру символов. Следовательно, стоит придерживаться того написания, которое приводится в описании объекта.
- Параметр value предназначен для установки значения передаваемой переменной. Значением данного параметра является текст. Встраиваемый объект берет на себя его распознавание.
- Параметр valuetype позволяет задавать тип передаваемого значения. Это могут быть данные в каком-либо стандартизованном формате, ссылка на некий ресурс в Сети или другой объект. Да, некоторые встраиваемые объекты в качестве переменных могут принимать другие объекты, причем их тип может и не совпадать с типом встраиваемого объекта. В качестве значения параметра может использоваться одно из трех предустановленных ключевых слов. Значение data, установленное по умолчанию, указывает, что используется стандартная переменная, передающая данные какого-либо типа. Как уже упоминалось, мы передаем данные как строку, а объект сам интерпретирует их. Значение ref указывает, что в качестве переменной передается ссылка на какой-либо ресурс в Сети. Значение object сигнализирует, что мы передаем в качестве стартовых данных другой объект.
- Параметр type используется в тех случаях, когда параметр valuetype имеет значение ref, т. е. когда в качестве переменной мы передаем ссылку на некий ресурс в Сети. Данный параметр указывает, какой тип имеет ресурс, на который мы ссылаемся.

На этом перечень используемых параметров тэга <param> заканчивается. Еще раз напомню, что когда мы берем из Сети некий встраиваемый объект,

к нему всегда прилагается сопроводительный текст, в котором рассказывается, как подключать данный объект и какие стартовые данные ему нужны для работы. Внимательно читайте инструкцию, это может существенно сэкономить вам время.

Среди встраиваемых объектов HTML выделяет в особую группу Java-апплеты. Для того чтобы внедрить их в состав содержимого Web-страниц, предусмотрен специализированный тэг `<applet>`. Но прежде, чем мы разберем его использование, стоит все-таки узнать, что такое Java-апплеты.

Язык Java первоначально задумывался как язык для создания приложений, которые бы выполнялись на любой компьютерной платформе без изменения кода. Таким образом, если программное обеспечение пишется с использованием Java, то не нужно писать отдельные версии для компьютеров на базе Intel-процессоров и компьютеров семейства Macintosh, или для различных операционных систем.

Подобная "многоликость" достигается за счет очень остроумного решения. Java-приложения записываются не в кодах какого-либо процессора, как обычные исполняемые программы, а в специализированном формате, называемом байт-кодом. Этот байт-код распознается не процессором, а другим приложением, которое именуют "виртуальной Java-машиной". Виртуальная Java-машина создается для каждой компьютерной системы отдельно. Она переводит байт-код в команды процессора. Такие виртуальные Java-машины написаны уже почти для каждой операционной системы, поэтому Java-приложения медленно, но верно завоевывают популярность.

Возможность выполнения кода на любой компьютерной системе это именно то, чего не хватает WWW. Ведь если документы читают браузеры, то активные элементы должны обрабатываться непосредственно операционной системой, и если выбрать какое-либо решение, действующее только в одной системе, то тем самым мы делаем свой ресурс недоступным для пользователей остальных компьютерных платформ и операционных систем.

Для внедрения в Web-страницы был разработан дополнительный стандарт облегченных Java-приложений. Подобные облегченные Java-приложения называются Java-апплетами. Вставляются они в содержимое Web-страниц с помощью тэга `<applet>`. Специфика его использования ничем не отличается от правил применения тэга `<object>`. И наборы параметров этих двух тэгов практически не различаются. В тэге `<applet>` могут использоваться параметры `codebase`, `code`, `name`, `archive`, `width`, `height`, `alt`, `align`, `hspace`, `vspace`. Функциональность этих параметров также не отличается от функциональности их двойников, применяемых в тэге `<object>`. Между тэгами `<applet>` и `</applet>` могут размещаться тэги `<param>`, задающие стартовые данные для Java-апплета.

Следует обратить внимание, что в списке параметров отсутствует параметр `classid`. Дело в том, что тэг `<applet>` применяется для объектов строго фик-

сированного типа, апплетов, а они унаследовали пакетную структуру из своего прародителя — языка Java. Java-апплеты, если говорить более точно, представляют собой не просто какие-то файлы, а так называемые классы, хранимые в Java-пакетах, поэтому для их идентификации используется наименование класса, которое записывается в параметре `name`.

На этом мы заканчиваем рассмотрение внедряемых исполняемых объектов. Еще раз повторю: до тех пор, пока вы не научитесь создавать их самостоятельно, а для этого необходимо уметь программировать на достаточно высоком уровне, следует использовать общедоступные внедряемые элементы. Однако при их применять все-таки следует соблюдать определенную осторожность, т. к. исполняемые объекты от неизвестных производителей потенциально могут быть опасны для пользователей. Следует пользоваться элементами, которые уже прошли проверку временем и интернет-сообществом.

Формы

Не раз мы видели Web-страницы, на которых нам предлагалось внести некоторые данные в поля ввода. В HTML существует механизм получения данных от пользователя. Естественно, эти данные необходимо еще и обработать, но этим занимаются специализированные программы. Рассмотрим механизм взаимодействия подобных программ с Web-страницами.

Итак, пользователь открыл Web-страницу, на которой располагаются элементы управления для ввода информации. Все они объединены в общую совокупность, называемую *формой*. Каждая форма обладает кнопкой, щелчок по которой приводит к передаче введенных пользователем данных обрабатывающей программе. Эта программа размещается на Web-сервере, который обслуживает данную страницу. Подобные программы могут создаваться с помощью самых различных технологий программирования. Объединяет их лишь общий порядок получения данных от Web-страницы. Данные передаются шлюзовым интерфейсом CGI (Common Gateway Interface). Поэтому обрабатывающие программы часто называют CGI-приложениями или CGI-скриптами.

Специализированное приложение получает данные и обрабатывает их. Затем оно может либо послать некое электронное письмо, либо произвести некоторую операцию в базе данных, или передать пользователю новую Web-страницу. Возможные действия ограничиваются лишь возможностями применяемой технологии и фантазией программиста. Подобные программы обеспечивают работу всевозможных систем регистрации, обратной связи, гостевых книг, форумов, чатов. С их помощью создаются и более разветвленные и сложные системы, например интерактивные магазины.

Для того чтобы создавать подобные приложения, необходимо уметь программировать и знать соответствующие правила создания CGI-приложений.

До тех пор, пока мы этого не умеем, нам остается пользоваться опять-таки общедоступными CGI-скриптами. Даже имея такие приложения, форму для ввода пользователем данных мы должны разработать самостоятельно. В этом разделе мы научимся это делать.

Форма ограничивается тэгами <form> и </form>. Между этими тэгами располагаются тэги, создающие элементы управления, и тэги создания обычного содержимого Web-страницы. Элементы управления могут размещаться в таблице, которая в свою очередь полностью расположена в форме. Тэг <form> не создает какой-либо отображаемой структуры. Он, скорее, предназначен для внутренней группировки объектов.

Тэг <form> обладает целым рядом параметров, которые задают свойства создаваемой формы. Рассмотрим эти параметры.

- Параметр action является обязательным. Его значение — URL, указывающий на расположение того CGI-приложения, которое будет обрабатывать данные, введенные пользователем с помощью элементов управления искомой формы.
- Параметр method предназначен для указания способа, которым данные передаются обрабатывающему приложению. В качестве значения параметра используется одно из двух предустановленных ключевых слов: get или post. Сейчас нам нет нужды узнавать, какие механизмы реализуются тем или иным методом. В сопроводительной документации CGI-приложения указывается, какой метод передачи данных следует применить. По умолчанию используется значение get.
- Параметр enctype указывает тип передаваемых из формы данных. Обычно нет необходимости его использовать, т. к. значение application/x-www-form-urlencoded, применяемое по умолчанию, идеально подходит для подавляющего большинства CGI-приложений.
- Параметр accept-charset задается в тех случаях, когда пользователь передает из формы приложению не только информацию, но и файлы. В этом случае мы можем явно указать кодировки передаваемых файлов. В качестве значения данного параметра используется текстовая строка, в которой записывается одно или несколько названий кодировок. Если применяется несколько кодировок, их наименования разделяются пробелами или запятыми. По умолчанию используется значение UNKNOWN, которое указывает серверу, что он должен сам разобраться с применяемыми кодировками.
- Параметр accept описывает типы передаваемых файлов. Обычно не используется, т. к. сервер вполне в состоянии сам корректно распознать тип принимаемого файла.
- Параметр name определяет уникальное имя формы. Естественно, на одной Web-странице может находиться несколько форм. В этом случае, значения параметров name у них не должны совпадать.

Тэг <form> и его закрывающий двойник </form> создают контейнер для размещения элементов управления. Большая часть этих элементов управления реализуется с помощью тэга <input>. Продемонстрируем это на небольшом примере в листинге 1.33 и на рис. 1.32.

Листинг 1.33

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Формы</title>
  </head>
  <body>
  <form action="http://www.mysite.com/cgi-bin/test.exe" method="post">
  <p>Поле для ввода строки текста <input type="text"></p>
  <input type="submit" value="Отправить">
  </form>
  </body>
</html>
```

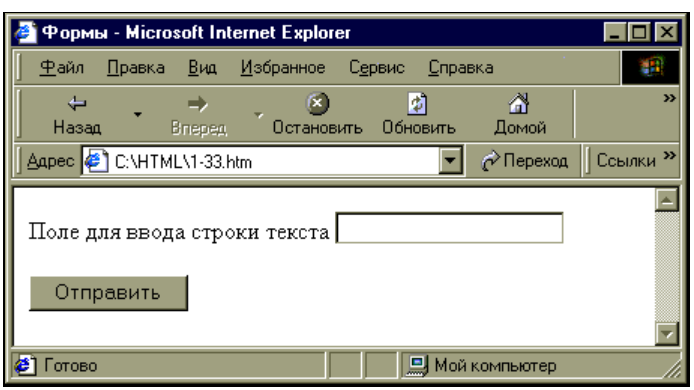


Рис. 1.32. Окно браузера с результатом отображения файла, приведенного в листинге 1.33

На иллюстрации видно, что мы смогли создать поле ввода текста и кнопку, при нажатии на которую введенная пользователем информация будет отправлена CGI-приложению для обработки. Если мы посмотрим на код листинга, то увидим, что и кнопка, и поле ввода создавались одним и тем же тэгом <input>. Регулировка свойств этого тэга производится с помощью его многочисленных параметров, которые мы сейчас и рассмотрим.

- Параметр type является, пожалуй, ключевым параметром. Его значение определяет тип создаваемого элемента управления. В качестве значения

используется одно из предустановленных ключевых слов: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `file`, `hidden`, `image`, `button`. По умолчанию используется значение `text`. Более подробно эти типы мы рассмотрим немного позже.

- Параметр `name` применяется для установки уникальных имен элементов управления. Несмотря на то, что этот параметр не является обязательным, настоятельно рекомендуется использовать его. В сопроводительной документации CGI-приложений обязательно указывается, какие имена должны быть у соответствующих элементов управления.
- Параметр `value` используется для указания значения, отображаемого по умолчанию для кнопок и полей ввода текста. Если мы используем радиокнопки и независимые переключатели, то значение параметра `value` не будет видно пользователю, но именно это значение получит обрабатывающее CGI-приложение, если пользователь выберет соответствующую радиокнопку или независимый переключатель.
- Параметр `cheked` применяется только для независимых переключателей и радиокнопок. Он устанавливает их начальное состояние. Если этот параметр будет введен в тэг `<input>`, то радиокнопка или независимый переключатель будут переведены во включенное состояние. Параметр не имеет значений.
- Параметр `disabled` делает элемент управления недоступным для пользователя. У параметра нет значений.
- Параметр `readonly` применяется только для полей ввода `text` и `password`. Использование этого параметра означает, что данные, отображаемые в этих полях, нельзя будет изменять.
- Параметр `size` обычно задает размеры элемента управления. Но для каждого отдельного типа элементов управления его действие специфично.
- Параметр `maxlength` позволяет устанавливать максимально возможное число символов, которые пользователь может ввести в поля текстового ввода. Значение параметра — целое положительное число.
- Параметр `src` используется в тех случаях, когда мы создаем элементы управления, связанные с графикой. Значением данного параметра является URL графического файла, который содержит отображаемый рисунок.
- Параметр `alt` позволяет вставлять описание создаваемого элемента управления. Это описание может отображаться в виде маленького хинта-подсказки, когда пользователь наводит курсор мыши на этот управляющий элемент.
- Параметр `tabindex` задает номер элемента управления в последовательности всех объектов, перемещение фокуса ввода между которыми осуществляется с помощью последовательных нажатий клавиши табуляции.

- Параметр `accesskey` позволяет указать "горячую клавишу", при нажатии на которую фокус ввода переходит к данному элементу управления.

Итак, мы рассмотрели параметры, применяемые в тэге `<input>`. Мы уже знаем, что с помощью данного тэга можем создавать самые различные объекты форм. Пришло время детально разобраться с ними.

Объекты, входящие в форму, разделяются на два типа — поля ввода данных и кнопки, инициирующие различные действия. Сначала посмотрим, как мы можем создавать поля ввода.

Одним из самых распространенных объектов форм является однострочное поле ввода. В листинге 1.33 мы видели, что оно создается с помощью параметра `type` со значением `text`. При этом достаточно часто нам надо задавать ограничения на максимально возможное количество символов, которое пользователь может ввести в это поле. Это ограничение устанавливается параметром `maxlength`.

Существует модификация однострочного поля ввода текста, которая предназначена для ввода секретной информации, например паролей. Такое поле ввода заменяет символы вводимого текста звездочками. Создание подобных полей ввода выполняется следующей конструкцией:

```
<input type="password">
```

Использование типа `checkbox` позволяет создавать элементы управления, называемые независимыми переключателями. Они представляют собой всем знакомые квадратики, в которых щелчком мыши мы можем устанавливать и снимать флажки в виде галочек, изменяя при этом значение параметра `value`. Значение этого параметра будет передано в обрабатывающее CGI-приложение, если пользователь установит флажок в данном независимом переключателе.

Следующий тип элемента управления, который мы можем создать, — группы радиокнопок, часто называемые зависимыми переключателями. В этой группе пользователь может выбирать и помечать только одну радиокнопку. Каждая радиокнопка описывается с помощью тэга `<input>` с параметром `type`, которому присвоено значение `radio`. Для того чтобы браузер понял, что несколько радиокнопок принадлежат к одной группе, необходимо задать одинаковое значение параметра `name` для всех радиокнопок, входящих в эту группу. При этом значения параметра `value` у них должны быть разными.

Рассмотрим на примере правила создания и отображения описанных элементов управления.

Листинг 1.34

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
```

```
  <head>
  <title>Формы</title>
  </head>
  <body>
  <form action="http://www.mysite.com/cgi-bin/test.exe" method="post">
  <p>Поле для ввода строки текста <input type="text"></p>
  <p>Поле для ввода пароля <input type="password"></p>
  <p>Независимый переключатель <input type="checkbox"
value="checked"></p>
  <p>Группа радиокнопок</p>
  <p>Вариант 1 <input type="radio" name="group1" value="check1"></p>
  <p>Вариант 2 <input type="radio" name="group1" value="check2"
checked></p>
  <p>Вариант 3 <input type="radio" name="group1" value="check3"></p>
  <input type="submit" value="Отправить">
  </form>
  </body>
</html>
```

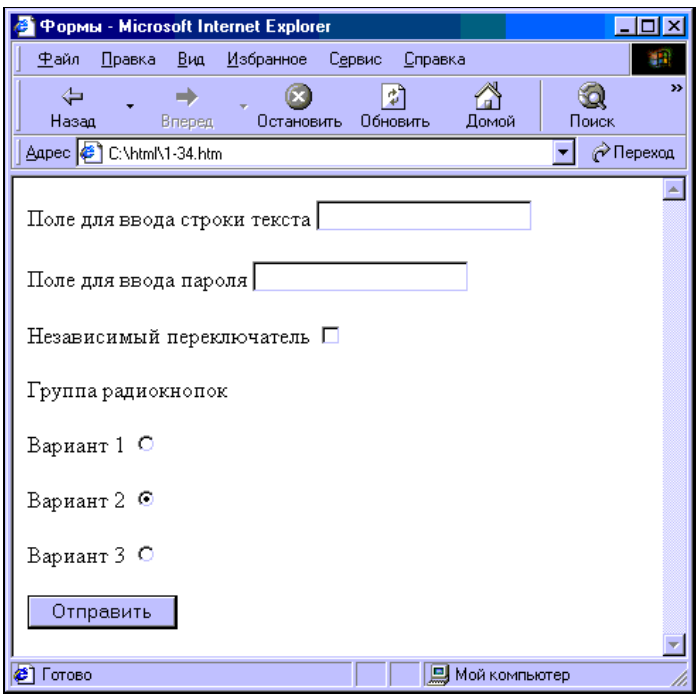


Рис. 1.33. Окно браузера с результатом отображения файла, приведенного в листинге 1.34

На рис. 1.33 видно, как отображается введенный текст в обычном поле ввода и в поле ввода конфиденциальной информации. В листинге 1.34 показа-

но, как мы можем создавать независимые переключатели и группы радиокнопок. При этом вторую по счету радиокнопку мы определили по умолчанию как выбранную. Пользователь, конечно, может выбрать другую.

С помощью параметра `type` со значением `hidden` мы можем создавать скрытые поля. Такое поле не только не позволяет пользователю вводить какую-либо информацию, но и вообще не отображается в окне просмотра. Данный тип объекта применяется обычно для служебных целей разработчиков.

Значение `file` параметра `type` описывает поле ввода имени файла. При этом рядом с полем ввода текста автоматически создается кнопка с наименованием **Обзор**, при нажатии на которую открывается стандартный диалог выбора файла.

Помимо полей ввода информации, мы можем размещать еще и элементы управления, называемые кнопками. В формах используется три вида кнопок. Кнопка типа `"submit"` отправляет введенные пользователем данные обрабатывающему CGI-приложению. Кнопка типа `"reset"` стирает из полей ввода информацию, внесенную пользователем, и отображает информацию, установленную по умолчанию. Существуют и простые кнопки, реакцию которых мы можем программировать самостоятельно с помощью скриптовых языков. Кроме того, допускается замена кнопки типа `"submit"` любым графическим изображением. Рассмотрим способы создания этих элементов управления.

Кнопка типа `"submit"` создается конструкцией следующего вида:

```
<input type="submit" value="Подтвердить">
```

Как видно, создание кнопки описывается с помощью параметра `type`, а надпись на ней задается значением параметра `value`.

Если же нам необходимо создать кнопку очистки полей ввода информации, мы можем применить следующий фрагмент кода:

```
<input type="reset" value="Очистить">
```

Для простой кнопки мы используем конструкцию следующего вида:

```
<input type="button" value="Кнопка">
```

Мы уже говорили, что вместо обычной кнопки типа `"submit"` мы можем использовать любое графическое изображение. Для этого мы применяем такой фрагмент кода:

```
<input type="image" src="http://www.mysite.com/image/pic1.gif">
```

В этом случае нет нужды использовать параметр `value`, т. к. не нужно задавать надпись на кнопке, но необходимо использовать параметр `src`, указывающий URL используемого графического файла.

Помимо тех элементов управления, которые мы уже рассмотрели, есть и дополнительные, которые не реализуются с помощью тэга `<input>`. К ним относится выпадающий список и многострочное поле текстового ввода.

Выпадающий список задается тэгом <select>. При этом используется и его закрывающий двойник </select>. Между этими тэгами мы объявляем элементы списка с помощью тэгов <option>. Тэг <select> обладает следующим набором параметров.

- Параметр name как обычно определяет уникальное имя данного поля ввода, т. е. выпадающего списка.
- Параметр size задает количество видимых строк, при раскрытии списка. Естественно, самих элементов в списке может быть больше, чем отображаемых строк. Но в этом случае всего-навсего будет отображаться вертикальная полоса прокрутки. Значением параметра является целое положительное число.
- Параметр multiple применяется, если мы хотим позволить пользователю выбирать из списка несколько значений сразу.
- Параметр disable заставляет браузер отображать данный выпадающий список, но при этом у пользователя нет возможности активизировать его и выбрать какое-либо значение.
- Параметр tabindex, как мы уже знаем, устанавливает порядковый номер элемента ввода данных в последовательности объектов, переход фокуса ввода между которыми производится с помощью клавиши табуляции.

Элементы выпадающего списка создаются тэгами <option>. Рассмотрим параметры этих тэгов.

- Параметр selected используется для тех элементов, которые при активизации выпадающего списка выделяются по умолчанию.
- Параметр value указывает значение, которое будет передано обрабатывающему CGI-приложению, если пользователь выделит именно этот элемент выпадающего списка.

Теперь рассмотрим несложную процедуру создания выпадающих списков в формах HTML-документов (см. листинг 1.35 и рис. 1.34).

Листинг 1.35

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Формы</title>
  </head>
  <body>
  <form action="http://www.mysite.com/cgi-bin/test.exe" method="post">
  <select name="menu1">
  <option value="sel1">1 пункт</option>
```

```
  <option value="sel2" selected>2 пункт</option>
  <option value="sel3">3 пункт</option>
  </select>
  <input type="submit" value="Отправить">
  </form>
  </body>
</html>
```

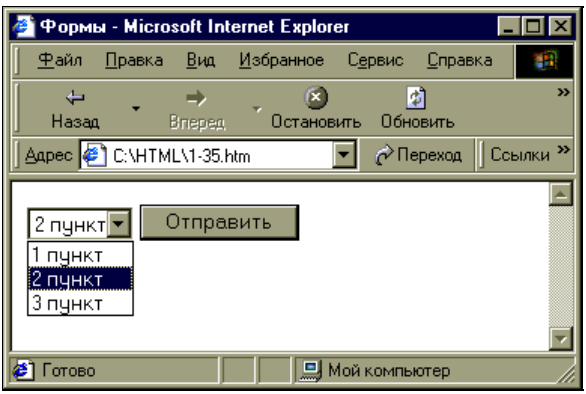


Рис. 1.34. Окно браузера с результатом отображения файла, приведенного в листинге 1.35

Нам осталось рассмотреть только один элемент ввода данных. Это многострочное поле текстового ввода. Оно реализуется с помощью тэга `<textarea>`. Этот тэг создает область ввода, которая может быть достаточно обширной, и применяется обычно для ввода средних и больших объемов текстовой информации. Тэг не может обойтись без параметров. И мы обязательно их рассмотрим.

- Параметр `name` позволяет задавать уникальное имя для данного элемента ввода данных.
- Параметр `rows` устанавливает количество отображаемых строк у создаваемого поля ввода. По сути дела, это его высота, но задается она в строках. Параметр является обязательным. В качестве значения используется целое положительное число.
- Параметр `cols` задает ширину в символах создаваемого поля текстового ввода. Параметр также является обязательным.
- Параметр `disabled` используется, если поле необходимо сделать недоступным для применения.
- Параметр `readonly` предназначен для создания полей с нередактируемым текстом. Текст, отображаемый в таких полях ввода, пользователь изменить не сможет.

- Параметр `tabindex` устанавливает порядковый номер элемента ввода данных в последовательности объектов, переход фокуса ввода между которыми производится с помощью клавиши табуляции.
- Параметр `accesskey` позволяет задавать клавишу, при нажатии на которую фокус ввода будет автоматически передан данному полю ввода.

Теперь посмотрим, как создаются подобные поля ввода.

Листинг 1.36

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Формы</title>
  </head>
  <body>
  <form action="http://www.mysite.com/cgi-bin/test.exe" method="post">
  <textarea rows=5 cols=20>
  Текст, который будет отображаться в многострочном поле ввода
  </textarea>
  <p><input type="submit" value="Отправить"></p>
  </form>
  </body>
</html>
```

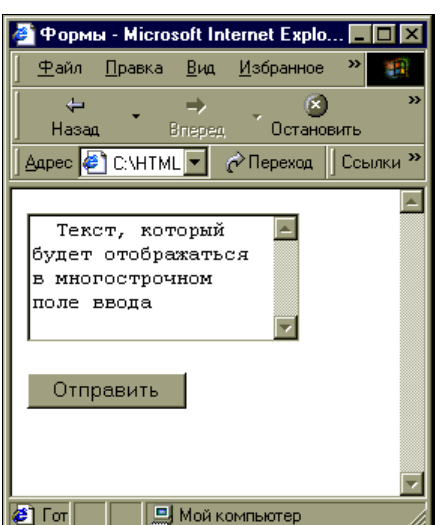


Рис. 1.35. Окно браузера с результатом отображения файла, приведенного в листинге 1.36

Помимо этих полей ввода есть еще один объект, который участвует в создании форм. Мы можем создавать так называемые текстовые метки для элементов управления.

Радиокнопки и независимые переключатели достаточно малы по своим размерам, и для их активизации необходимо точно попасть мышью в их активную область. Это не всегда бывает удобно, поэтому во многих программах с хорошо продуманным пользовательским интерфейсом можно активизировать подобные элементы управления еще и щелчком мыши на их текстовых наименованиях. Такую же возможность предоставляет нам HTML.

Для создания текстовых пометок и заголовков, связанных с конкретными элементами управления, мы можем использовать тэг `<label>`. Подобные метки мы сможем привязывать только к тем элементам управления, для которых явно задали идентифицирующий параметр `id`. Именно по нему и устанавливается соответствие между меткой и объектом формы. Основным параметром для тэга `<label>` служит параметр `for`, значение которого равно значению параметра `id` объекта, к которому данная текстовая метка привязывается. Приведем пример создания подобной связи. Сначала мы создадим независимый переключатель, а затем присоединим к нему связанный текстовый заголовок. Описывается подобная операция с помощью следующего фрагмента кода:

```
<label for="check1">Независимый переключатель</label>
<input type="checkbox" name="check" value="checked" id="check1">
```

Впрочем, можно обойтись и без параметра `for`, если тэг, объявляющий связанный элемент управления, разместить между тэгами `<label>` и `</label>`.

На этом мы заканчиваем рассмотрение процедуры создания форм для ввода данных в статических HTML-документах.

Использование сценариев

В рамках HTML мы обладаем возможностью использовать при создании Web-страниц специальный вид программ, называемых *сценариями*, или *скриптами*. Они очень тесно связаны с технологией динамического HTML (DHTML), которую мы будем рассматривать в третьей главе. Сейчас мы лишь познакомимся с возможностью их подключения к HTML-документу.

Для создания подобных программ-сценариев могут использоваться два языка программирования: JavaScript и VBScript. Эти программы просто встраиваются в Web-страницу, а браузер пользователя получает их и самостоятельно выполняет. В связи с тем, что VBScript недостаточно хорошо поддерживается всеми браузерами, чаще всего используется язык JavaScript.

Итак, мы создали или нашли на просторах Сети программу-сценарий, которую нам хотелось бы использовать на своей Web-странице, для придания ей

некоей динамичности и интерактивности. Теперь интересующую нас программу необходимо подключить к нашей странице. Для этих целей используется тэг <script> со своим закрывающим двойником </script>. Между этими тэгами обычно размещается текст программы-сценария. Впрочем, иногда просто указывается URL файла с этим текстом, и тогда браузер сам находит его, руководствуясь указанным URL. Однако, поскольку поиск проводится только тогда, когда необходимо запустить скрипт, то между действиями пользователя и реакцией программы возникает определенная задержка, поэтому чаще всего текст программы присоединяют к HTML-документу.

Тэг <script> с его содержимым обычно размещается в заголовке Web-страницы, между тэгами <head> и </head>. Связано это с тем, что подключаемый скрипт должен быть описан до момента его первого использования. Другими словами, мы можем описать его в теле документа после тэга <body>, но при этом мы должны быть уверены, что все элементы Web-страницы, использующие или активизирующие этот скрипт, будут описаны после него. Во избежание возможных ошибок скрипты описывают в заголовках. Это проявление корректного стиля создания документов, которого всегда следует придерживаться.

Как мы уже знаем, тэг <script> позволяет внедрять в создаваемые Web-страницы программы-сценарии. Этот тэг обладает рядом параметров, которые дают возможность браузеру наилучшим образом распознавать передаваемую информацию и выполнять инструкции программы.

- Параметр type предназначен для указания типа присоединяемого скрипта. Он заменил устаревший параметр language. Обычно используются значения text/javascript и text/vbscript. Первое значение указывает, что присоединяемый скрипт написан на языке JavaScript, а второе, соответственно, зарезервировано для VBScript. Данный параметр является обязательным.
- Параметр src применяется в тех случаях, когда код присоединяемого скрипта не внедряется непосредственно в HTML-документ, а вынесен в отдельный файл, который браузер должен сам найти и загрузить. Значением данного параметра является URL искомого файла, содержащего код присоединяемого скрипта.
- Параметр charset имеет смысл задавать в паре с предыдущим параметром, т. к. его значение определяет кодировку символов, которая использовалась при создании файла с подключаемым скриптом.

Теперь, когда мы знаем, какие параметры есть у тэга <script>, рассмотрим фрагмент кода, в котором показан порядок его использования.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
```

```
<head>
<title>Интерактивная страница</title>
 <script type="text/javascript">
     ...Код скрипта....
 </script>
</head>
```

В этом примере мы просто подключаем скрипт с вставкой его кода напрямую в создаваемую Web-страницу. Если необходимо присоединить отдельный файл со скриптом, следует использовать следующий фрагмент кода:

```
<script type="text/vbscript"
↳src="http://www.mysite.com/progs/vbcalc">
```

В этом фрагменте мы подключаем файл, который находится в каталоге `progs` на сайте `www.mysite.com`. Как видно, все достаточно просто и понятно.

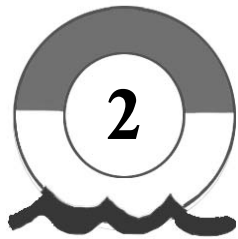
Очень часто с помощью подобных скриптов создатели Web-страниц динамически генерируют содержимое своих страничек. Это как раз одна из наиболее популярных возможностей DHTML, но всегда есть вероятность, что пользователь будет просматривать Web-страницу устаревшим браузером, который не сможет обработать скрипты. В этом случае было бы хорошо установить альтернативное оповещение, которое отображалось бы браузером, если тот не в состоянии использовать программы-сценарии. Подобное сообщение мы можем задать с помощью тэгов `<noscript>` и `</noscript>`. Обычно они располагаются сразу после тэгов `<script>`. В итоге получается конструкция следующего вида:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
<head>
<title>Интерактивная страница</title>
 <script type="text/javascript">
     ...Код скрипта....
 </script>
 <noscript>
 К сожалению, ваш браузер не поддерживает скрипты.
 </noscript>
</head>
```

На этом мы заканчиваем рассмотрение процедуры подключения программ-сценариев к HTML-документам, а вместе с ней и описание всего стандарта HTML. Он действительно не так уж велик, но при этом достаточно гибок.

Одного знания HTML недостаточно, если мы хотим сделать большой и профессиональный сайт. Нам необходимо рассмотреть вопросы единого стилевого оформления Web-страниц и механизмы придания им динамичности и интерактивности. Об этом мы узнаем в следующих главах.

ГЛАВА 2



Стилевое оформление

Стилевые таблицы

В первой главе книги мы узнали, как создавать Web-страницы с помощью тэгов. Кроме того, мы увидели, как при создании элементов Web-страниц мы можем задавать некоторые правила их отображения. Делается это, как мы помним, с помощью определенных параметров соответствующих тэгов. Но как быть, если этих параметров и реализуемых ими свойств явно недостаточно для адекватного воплощения замыслов разработчика Web-страниц?

При создании довольно объемного и разветвленного сайта может также возникнуть необходимость изменения дизайна. Если нужно произвести даже незначительную корректировку, например, всего лишь изменить цвет фона для всех Web-страниц, то придется редактировать каждую Web-страницу, входящую в состав сайта. При этом, естественно, резко возрастает вероятность ошибки, и, кроме того, все эти процедуры могут занять немало времени.

Было бы неплохо задавать параметры отображения каких-либо элементов для всего сайта сразу. Например, мы хотим, чтобы все заголовки первого уровня отображались шрифтом синего цвета на матово-серебристом поле. В популярном Microsoft Word и других текстовых процессорах подобные проблемы изящно решаются с помощью введения стилей, которые мы можем сохранять в шаблонах и применять к самым различным документам. Было бы хорошо иметь такую возможность и для Web-страниц.

Следовательно, нам необходима технология, которая была бы приспособлена для описания свойств отображения содержимого Web-страниц и позволяла бы нам задавать параметры отображения как для каждого отдельно взятого элемента какой-либо Web-страницы, так и для всех Web-страниц, вхо-

дящих в состав сайта. Такая технология есть. Называется она CSS (Cascading Style Sheets), что на русский язык традиционно переводят как "каскадные стилевые таблицы".

Первая версия стандарта CSS содержала правила отображения текста Web-страниц, его размещения на странице и некоторые детали отображения страниц и ячеек таблиц. Вторая версия, которая носит официальное наименование CSS level 2, добавила несколько новых возможностей, таких как визуальные эффекты или автоматическая нумерация элементов. Все это мы подробно рассмотрим.

Прежде всего, нам необходимо разобраться с механизмом создания и использования CSS. Стилевая таблица представляет собой обычный текстовый файл специализированного формата. В спецификациях данный формат имеет обозначение `text/css`. В CSS-файле записываются правила отображения для некоторых тэгов. Пример записи одного такого правила может выглядеть следующим образом:

```
a {color : navy; font-family : Arial}
```

Это правило указывает на то, что текстовое содержимое всех тэгов `<a>`, которыми в HTML обозначаются гиперссылки, будет отображаться с помощью любого доступного шрифта из семейства Arial, темно-синего цвета (navy).

Весь файл каскадной стилевой таблицы состоит из подобных определений. Помимо таких CSS-файлов мы можем внедрять определения стилей непосредственно в HTML-документы. Для этого используется тэг `<style>`, совместно со своим закрывающим двойником `</style>`. Между ними размещаются определения правил отображения тех или иных элементов Web-страницы, а вся конструкция в целом находится в разделе заголовка Web-страницы. В следующих разделах мы на примерах рассмотрим, как это делается. Определения стилей, внедренные в какой-либо HTML-документ, используются только при отображении этого документа. Если же мы создаем отдельный CSS-файл, то этот файл мы можем подключить к любой разрабатываемой Web-странице. Таким образом, если все Web-страницы сайта используют единую стилевую таблицу, то однотипные элементы в них будут отображаться одинаково. Следовательно, мы достаточно легко и просто добиваемся единообразного оформления всех Web-страниц, входящих в состав сайта. Такая технология облегчает изменение дизайна Web-страниц. Достаточно изменить одно-единственное определение отображения в стилевой таблице, и оно тут же отразится на всех Web-страницах, использующих эту стилевую таблицу.

Итак, резюмируем. Технология CSS позволяет нам задавать правила отображения тех или иных элементов разрабатываемых Web-страниц. Правила могут действовать в пределах одного HTML-документа или применяться сразу к нескольким. Параметры отображения, задаваемые правилами CSS,

обладают большими возможностями, чем параметры, описанные в HTML. При этом использовать стилевые таблицы легко: они правильно распознаются и обрабатываются всеми браузерами. Так что, если необходимо создать сайт с более чем десятью страничками, следует обязательно воспользоваться этой технологией.

Синтаксис CSS

Все правила CSS состоят из двух частей. В первой части записывается наименование тэга или класса, к содержимому которого будет применяться данное правило оформления. Такое объявление в спецификации CSS называют *selector*. Иногда это английское слово как кальку переносят в русскоязычную техническую литературу, и при описании технологии CSS тэги в стилевых таблицах именуют селекторами.

После наименования тэга или класса в фигурных скобках записывается правило оформления текстового содержимого. Посмотрим еще раз на наш пример.

```
a {color : navy; font-family : Arial}
```

Как видно, в фигурных скобках может записываться сразу несколько правил, разделенных символом точки с запятой. Само правило состоит из двух частей. Сначала записывается наименование параметра отображения, например вид применяемого шрифта, а затем, после двоеточия, мы пишем значение этого параметра.

Мы уже говорили, что правила отображения задаются для тэгов и классов, в чем же разница между этими двумя понятиями? Если мы задаем определения для тэга, то правило отображения будет действительно для всех элементов содержимого Web-страницы, которые реализуются с помощью такого тэга. Проще говоря, если мы установили стиль для тэга `<a>`, то все гиперссылки в документе, использующем данную стилевую таблицу, будут отображаться с учетом этого правила. Если мы задаем некий класс, то правило отображения будет применяться к тем тэгам, у которых есть параметр `class`, и его значение совпадает с наименованием заданного нами класса. Описанный порядок действий необходим, если правило отображения следует использовать не для всех однотипных объектов содержимого Web-страниц, а лишь для избранных. Например, мы хотим как-то по-особому выделять определения терминов в тексте, мы можем использовать либо стандартный тэг курсива, либо стилевое оформление. Чаще всего в таких случаях Web-дизайнеры выбирают именно стилевое оформление, т. к. это обеспечивает *большую гибкость управления* отображением и позволяет добиться более полного соответствия между замыслом и результатом.

Итак, в фигурных скобках мы можем записать сразу несколько правил отображения содержимого. Но иногда случается так, что сразу для нескольких

тэгов задаются одинаковые правила отображения. Мы можем не указывать одинаковые правила для каждого тэга отдельно, а объединить тэги в одну группу и задать правила отображения для всех тэгов сразу. При этом объединяемые наименования тэгов разделяются запятой. Это можно увидеть в описании следующей конструкции:

```
a, h1 { color : navy }
```

Приведенное правило указывает, что текстовое содержимое гиперссылок и заголовков первого уровня следует отображать темно-синим цветом.

В правила синтаксиса CSS входит способ оформления комментариев. Комментарии в CSS создаются в стиле языка C. Начинается комментарий с символа наклонной черты и звездочки (/*), а завершается обратным сочетанием (*/). Все это легко увидеть в следующем примере:

```
/*Это комментарий*/
```

Итак, мы рассмотрели правила группировки как наименований тэгов, так и правил отображения. Мы знаем, как использовать синтаксис CSS для создания каскадных таблиц стилей. Теперь мы можем перейти к рассмотрению последовательности применения правил оформления.

Порядок использования правил

Полное наименование CSS — каскадные таблицы стилей. Почему они называются таблицами стилей, мы уже поняли. Возникает вполне обоснованный вопрос: при чем тут каскадность, и что означает этот термин.

Компьютерная индустрия развивается очень быстрыми темпами. Следствием этого является возникновение самых различных тенденций и течений. Часть из них вскоре отмирает, а часть переходит из разряда тенденций в ранг идеологии. Одной из наиболее перспективных тенденций можно считать наследование ранее созданных решений. К CSS это имеет самое прямое отношение.

Любой HTML-документ может использовать одну стилевую таблицу или сразу несколько. Например, при добавлении новых документов к сайту придется добавлять новые правила отображения, создавать новые стили. Иногда это делается корректировкой уже существующей стилевой таблицы, но никто не запрещает нам создать дополнительный файл и подключить его к документу. Как мы только что говорили, мы можем подключать сразу несколько таблиц.

Иногда в нескольких CSS-файлах содержатся разные правила отображения для одного и того же тэга или, в случае использования XML, одного и того же элемента. В этом случае, браузером используется самое последнее объявление правила отображения.

Рассмотрим эту концепцию на примере. Предположим, у нас есть два файла стилевых таблиц. В файле style1.css мы размещаем объявления правил отображения для двух элементов HTML-документа с наименованиями классов `list` и `item`. Получаем следующий код:

```
list {font-family: Times New Roman}
item {color: blue; font-family: Arial}
```

А в дополнительном файле style2.css мы объявляем правила отображения для классов `item` и `part` следующим образом:

```
item {color: green; font-family: Arial}
part {color: black}
```

Теперь, если мы подключим к HTML-документу эти два стилевых файла, окажется, что для элемента `item` задано два различных правила отображения. В первом файле style1.css указано, что текст элемента `item` следует отображать синим цветом, а во втором файле правило задает для отображения содержимого этого же элемента зеленый цвет. Какой цвет использует браузер на самом деле, зависит от того, в каком порядке к HTML-документу будут подключаться файлы стилевых таблиц.

Используется всегда последнее правило. То есть, если сначала мы подключим файл style1.css, а потом файл style2.css, то текстовое содержимое элемента `item` будет отображаться зеленым цветом.

Мы можем устанавливать специальный модификатор для правил отображения, который заставит браузер игнорировать порядок подключения стилевых таблиц. Так, если мы припишем к какому-либо правилу оформления модификатор `important`, то для отображения текстового содержимого элемента будет использоваться именно это правило, вне зависимости от того, были ли заданы другие правила для отображения этого же элемента. Таким образом, использование ключевого слова `important` позволяет установить для какого-либо правила привилегированный уровень и нарушить обычный порядок выполнения правил каскадных таблиц стилей. Рассмотрим на примере. Возьмем знакомый нам файл style1.css и немного модифицируем его код. Получим следующую совокупность правил отображения:

```
.list {font-family: Times New Roman}
.item {color: blue ! important; font-family: Arial}
```

Теперь, при том порядке подключения стилевых файлов style1.css и style2.css, который мы рассматривали ранее, текстовое содержимое элемента `item` будет отображаться синим цветом.

Пример показывает, как использовать модификатор `important`. Если нам необходимо какое-либо правило сделать привилегированным, мы после него ставим символ восклицательного знака и записываем ключевое слово `important`, отделенное пробелом.

Вполне возможна ситуация, когда существует в нескольких CSS-файлах правила отображения для одного и того же элемента, и сразу несколько из них имеют модификатор `important`. В этом случае формируется дополнительная иерархия из этих привилегированных правил и используется для отображения самое последнее подключенное привилегированное правило.

Использование CSS в HTML-документах

Сначала мы узнаем, как включать определения стилей в HTML-документ без применения отдельных CSS-файлов. Как мы уже упоминали ранее, для этого предназначен тэг `<style>`. Рассмотрим его использование на примере.

Листинг 2.1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Встроенные стили</title>
  <style type="text/css">
    h1 { color: blue }
  </style>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    <p>Обычный текст </p>
  </body>
</html>
```

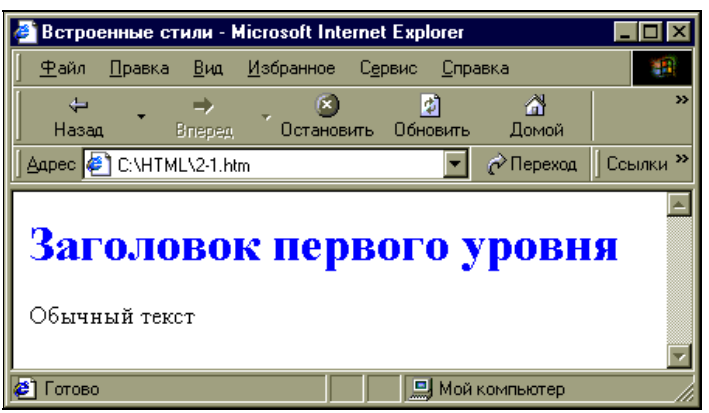


Рис. 2.1. Окно браузера с результатом отображения файла, приведенного в листинге 2.1

На рис. 2.1, из-за ограничений полиграфии, не видно, что текст заголовка первого уровня действительно отображен светло-синим цветом, но если самостоятельно создать HTML-документ, руководствуясь вышеприведенным листингом, и загрузить его в браузер, то в этом можно убедиться воочию.

В приведенном примере мы установили правило отображения, которое распространяется на все заголовки первого уровня. Теперь попробуем воспользоваться параметром `class`. Для создания селекторов, опирающихся на этот параметр, следует в определении правила перед наименованием класса ставить точку. Это хорошо видно в листинге 2.2 (результат отображения этого кода — на рис. 2.2).

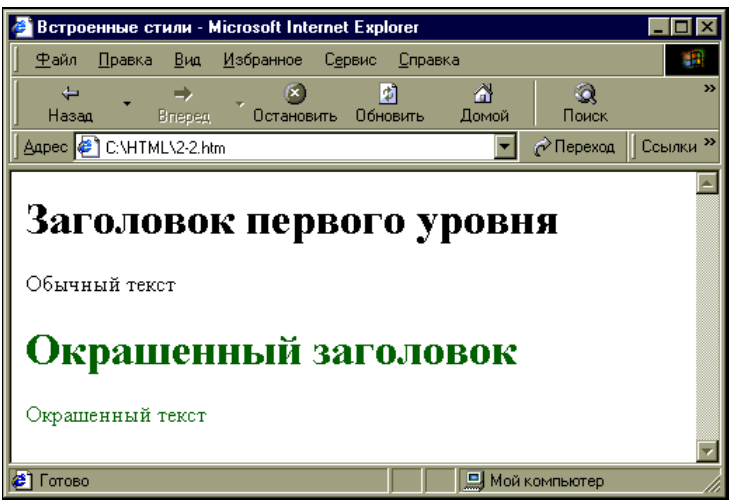


Рис. 2.2. Окно браузера с результатом отображения файла, приведенного в листинге 2.2

Листинг 2.2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Встроенные стили</title>
  <style type="text/css">
    .colored { color: green }
  </style>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    <p>Обычный текст </p>
```

```
    <h1 class="colored"> Окрашенный заголовок</h1>
    <p class="colored"> Окрашенный текст </p>
  </body>
</html>
```

После того как мы объявили в разделе стилей класс с наименованием `colored`, мы можем использовать это правило для любого тэга, лишь бы у него был описан параметр `class` со значением `colored`.

Впрочем, мы можем совместить в селекторе использование и тэга, и класса. Так, если нам надо окрашивать заголовки первого уровня синим цветом, а текст — зеленым, следует один и тот же класс `colored` объявить для каждого тэга в отдельности. Как это сделать — показано в листинге 2.3.

Листинг 2.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Встроенные стили</title>
  <style type="text/css">
    h1.colored { color: blue }
    p.colored { color: green }
  </style>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    <p>Обычный текст </p>
    <h1 class="colored"> Окрашенный заголовок</h1>
    <p class="colored"> Окрашенный текст </p>
  </body>
</html>
```

Как видно на рис. 2.3, использование одного и того же класса в разных тэгах привело к различным результатам.

Необходимо отметить, что в CSS существует несколько предопределенных классов. Такие классы в CSS называют псевдоклассами. Чаще всего используется четыре псевдокласса, связанных с тэгом `<a>`, реализующим гиперссылки. Поскольку эти псевдоклассы предназначены именно для этого тэга, то и селекторы используются составные, из наименования тэга и имени псевдокласса, разделенных знаком двоеточия.

- Селектор `a:link` позволяет задавать правила отображения для гиперссылки, к которой пользователь еще не обращался в текущем сеансе работы в Интернете.

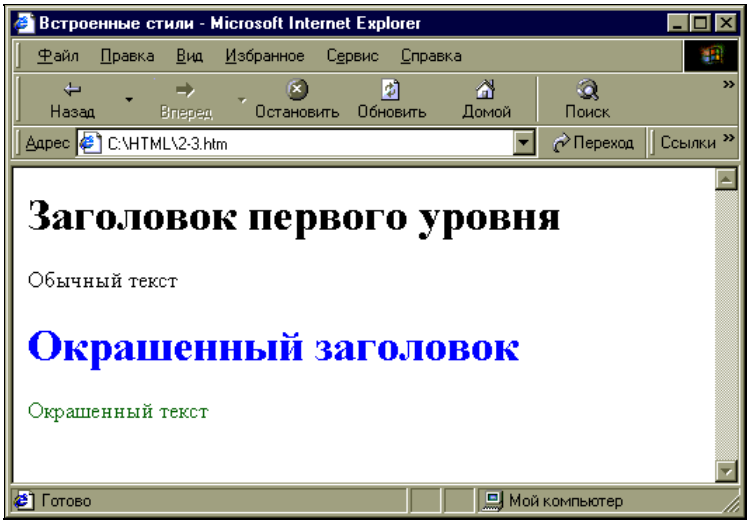


Рис. 2.3. Окно браузера с результатом отображения файла, приведенного в листинге 2.3

- Селектор `a:visited` предназначен для установки правил отображения гиперссылки, к которой пользователь уже обращался в текущем сеансе работы.
- Селектор `a:hover` позволяет задавать правила отображения гиперссылки, на которую пользователь указывает курсором мыши.
- Селектор `a:active` задает правила отображения гиперссылки в момент ее активизации, т. е. когда пользователь, указывая на нее курсором мыши, уже нажал кнопку, но еще ее не отпустил.

Последние два псевдокласса, т. е. `:hover` и `:active`, могут использоваться не только с гиперссылками, но и со многими другими тэгами. Есть еще один псевдокласс `:focus`, который определяет правила отображения элемента, получившего фокус ввода. Чаще всего он применяется для управляющих элементов форм.

Предусмотрена возможность в качестве селектора использовать идентификатор структурного элемента Web-страницы, т. е. значение параметра `id`. При этом совершенно неважно, к какому структурному элементу (заголовку, абзацу и т. п.) он относится. Поскольку все идентификаторы уникальны, то правило отображения, которое использует как селектор значение атрибута `id`, будет применяться всего один раз для конкретного структурного элемента с заданным `id`. Рассмотрим пример. В стилевой таблице мы можем разместить следующий фрагмент кода:

```
#z98y { letter-spacing: 0.3em }
```

Для того чтобы это правило выполнилось, в HTML-документе должна быть приблизительно следующая конструкция:

```
<p id=z98y>Растянутый текст</p>
```

Как видно из примера, в CSS перед значением идентификатора обязательно вставляется знак решетки (#), который указывает, что следующее за ним выражение будет восприниматься как селектор-идентификатор.

О селекторах следует поговорить особо. Существует несколько типов селекторов и, соответственно, несколько вариантов их обозначения. Мы пока что описали далеко не все варианты. Первый рассмотренный нами вариант позволяет создавать селекторы с помощью наименования класса, указывая тип структурного элемента. Их так и называют — селекторы типа (type selector). Приведем все остальные возможные варианты создания различных селекторов.

- *Универсальный селектор* (Universal selector). Позволяет устанавливать правило отображения сразу для всех элементов. В качестве универсального селектора используется символ звездочки (*).
- *Вложенный селектор* (Descendant selector). Применяется для установки правил отображения вложенных элементов.
- *Дочерний селектор* (Child selector). Позволяет создавать правила отображения для дочерних элементов. Стандартный синтаксис: `E > F`. То есть наименования родительского (E) и дочернего (F) элементов разделяются знаком "больше" (брекетом).

Замечание

Дочерние и вложенные селекторы CSS — разные вещи. Но для тэгов HTML мы можем сказать, что вложенный и дочерний элементы — это одно и то же, как и родительский элемент, содержащий в себе описываемый элемент (иногда такой элемент называют контейнером). Разница между понятиями "дочерний" и "вложенный", "родительский" и "элемент-контейнер" существует, но она достаточно тонкая, и в книге серии "Самоучитель" нет нужды рассматривать эти отличия.

Существует селектор, позволяющий устанавливать правило отображения для первого дочернего элемента. Так как в HTML нет механизма указания именно на первый дочерний элемент, для этой цели в CSS используется так называемый псевдокласс. Псевдоклассы являются механизмом CSS, позволяющим задавать элементы, которые нельзя идентифицировать стандартным образом, как в данном случае.

Итак, селектор, указывающий на первый дочерний элемент, специфицируется следующей конструкцией:

```
E:first-child
```

Таким образом, если мы хотим указать, что для отображения текстового содержимого первого дочернего элемента по отношению к элементу, реализуемому с помощью тэга <p>, будет применяться шрифт Arial, мы должны использовать следующую конструкцию:

```
p:first-child {font-family: Arial}
```

Мы рассмотрели все варианты селекторов, используемых в CSS. Теперь вернемся к способам применения правил отображения. Мы использовали включение стилевых правил непосредственно в HTML-документ с помощью тэга <style>. Этот тэг обладает параметром type, значение которого указывает тип содержимого этого тэга. Данный параметр редко используют, т. к. значение text/css, применяемое по умолчанию, характеризует именно объявление стилевых правил.

Еще несколько слов о задании стилей непосредственно в HTML-документе. Мы знаем, что в состав практически любого тэга входит параметр style. В качестве его значения мы указываем правило отображения содержимого тэга, но, естественно, без использования селекторов. В данном случае селекторы не нужны, т. к. правила отображения относятся только к содержимому данного тэга.

Помимо встраивания стилевых определений в HTML-документ, мы можем создавать стилевые таблицы и в виде отдельных файлов. Эти файлы обычно имеют расширение css. Для присоединения такого файла к какому-либо HTML-документу необходимо использовать тэг <link>. Как мы знаем, тэги <link> встраиваются в раздел заголовка HTML-документа. Для того чтобы правильно подключить стилевую таблицу, следует указать некоторые атрибуты. В общем случае, инструкция подключения CSS-файла выглядит следующим образом:

```
<LINK href="mystyle.css" rel="stylesheet" type="text/css">
```

Нам следует задать параметр href, значением которого является URL подключаемого CSS-файла, параметр rel со значением stylesheet и параметр type с уже упоминавшимся недавно значением text/css. Если мы хотим подключить сразу несколько файлов со стилевыми таблицами, нам придется использовать несколько тэгов <link>.

Еще раз перечислим способы подключения стилей в HTML-документах. Мы можем использовать правила отображения непосредственно в каком-либо тэге с помощью параметра style. Кроме того, мы можем создавать каскадные стилевые таблицы, внедряя их непосредственно в разрабатываемый HTML-документ с помощью тэга <style>. И самым универсальным и, кстати, рекомендуемым способом является подключение внешних стилевых таблиц, находящихся в отдельных CSS-файлах. Для связи HTML-файла и внешней стилевой таблицы мы используем тэг <link>.

Теперь, когда мы знаем синтаксис CSS, правила создания и использования стилевых таблиц, необходимо рассмотреть все применяемые в правилах CSS ключевые слова и единицы измерения. После этого мы сможем создавать свои стилевые таблицы.

Единицы измерения в CSS

Мы начинаем обзор не со стилевых свойств, а с единиц измерения, потому что без них не обойтись при описании параметров элементов отображения. С их помощью мы указываем расстояние на экране или задаем величину левого или правого поля. В CSS для этого предусмотрены различные единицы измерения. Их мы и рассмотрим в этом разделе.

В спецификацию CSS включены три вида единиц измерения (units). Тривиальные единицы измерения линейных размеров, единицы измерения в процентах и цветовые единицы. Да-да, цветовые обозначения тоже относятся к единицам измерения. Некоторые единицы измерения заимствованы из HTML, но есть и уникальные, что позволяет нам с помощью CSS задавать несколько более детальные правила отображения, по сравнению со средствами, встроенными в HTML. Но, все по порядку. Начинаем с линейных размеров.

Единицы измерения линейных размеров можно разделить на абсолютные и относительные. Мы увидим их различие.

- Сантиметр. Обозначается как `cm`. Абсолютная единица.
- Миллиметр. Стандартное обозначение — `mm`. Абсолютная единица.
- Дюйм. Обозначение `in` — сокращение его английского эквивалента (inch). Абсолютная единица.
- Пика. Иногда пишется как пайка. Имеет обозначение `pc`. Ранее применялась в типографской мере длин. Равна 4,23 миллиметра. Абсолютная единица.
- Ширина строчной буквы m применяемого шрифта. Обозначается как `em`. Традиционно буква m считается самым широким символом любого шрифта. Это относительная единица измерения, т. к. зависит от применяемого шрифта.
- Высота строчной буквы x применяемого шрифта. Стандартное обозначение — `ex`. Относительная единица измерения.
- Пиксел. Обозначается как `px`. Несмотря на то, что является абсолютной единицей, не имеет определенного размера, т. к. величина пиксела зависит от применяемого монитора и установленного графического разрешения.

Помимо рассмотренных нами единиц измерения линейных размеров, которые позволяют более или менее точно указать необходимую длину, мы мо-

жем задавать размеры в процентах. Например, если нужно указать, что величина левого поля равна одному проценту от ширины окна просмотра применяемого браузера. Проценты всегда отсчитываются от размера элемента, заданного по умолчанию. Горизонтальные величины отсчитываются от ширины, вертикальные — от высоты. При этом необходимо вместо наименования единицы измерения просто поставить знак процента (%).

Приведем пример. Создадим два правила отображения. Зададим размер правого поля для текста. В первом правиле используем абсолютную единицу измерения, во втором — размер в процентах. Это будет выглядеть следующим образом:

```
p.margins {margin-left: 12 mm}
h1 {margin-left: 2%}
```

Теперь нам осталось рассмотреть цветовые обозначения.

Для стандартных шестнадцати цветов есть зарезервированные словесные обозначения, которые приведены в табл. 1.2.

Таким образом, если мы хотим указать в стилевом файле, что для текста обычного абзаца мы собираемся использовать белый цвет фона, мы должны использовать следующую конструкцию:

```
p {background-color: white}
```

Если стандартных шестнадцати цветов не хватает, или просто нет желания применять уже надоевшие всем цвета, как это обычно бывает, можно указать практически любой цвет с помощью его RGB-кода.

В CSS предусмотрены четыре варианта задания RGB-кода требуемого цвета. Рассмотрим их на примере. Правило, реализующее первый вариант, выглядит следующим образом:

```
p.colored { color: #f00 }
```

Это усеченный вариант задания цвета. После знака решетки записываются три шестнадцатеричные цифры. Каждая из них указывает насыщенность соответствующего цвета. Первая слева указывает насыщенность красного цвета (Red), вторая — зеленого (Green), третья — синего (Blue). Так как мы используем всего по одной цифре для указания насыщенности, каждый цвет может иметь лишь шестнадцать градаций. Таким образом, на каждый цвет отводится по четыре бита. Поскольку основных цветов всего три, нетрудно подсчитать, что подобным образом мы можем задать всего лишь 4096 цветов.

Естественно, взыскательного пользователя (а кто из нас не считает себя таковым?) подобное количество цветов удовлетворить не могло, поэтому был создан вариант задания цвета 256 градациями насыщенности для каждого базового цвета.

Этот вариант показан в следующем примере:

```
p.colored { color: #ff0000 }
```

В примере для задания насыщенности базового цвета используется двузначное шестнадцатеричное число. Таким образом, мы можем определить уже 16 777 216 различных цветов, что соответствует стандарту TrueColor. Не думаю, что для интернет-приложений кому-то может понадобиться большее количество цветов.

Если же использование шестнадцатеричных чисел еще не вошло в привычку, можно задавать насыщенность базового цвета обычным десятичным числом. Естественно, все десятичные значения должны быть неотрицательными, целочисленными и не превышать числа 255. Правила применения легко увидеть в следующем примере:

```
p.colored { color: rgb(255,0,0) }
```

Если мы не используем знак решетки, который указывает на наличие шестнадцатеричных чисел, то следует применить оператор `rgb`, после которого в скобках указываются значения насыщенности базовых цветов, разделенные запятыми.

Четвертый вариант задания цвета с помощью RGB-кода предусматривает использование процентного определения насыщенности. Это видно в данном примере:

```
p.colored { color: rgb(100%, 0%, 0%) }
```

Легко заметить, что индикатором, указывающим обработчику стилевых таблиц на то, что в данном случае применяется процентное определение насыщенности, служит знак процента.

Последний способ позволяет использовать числовые значения с десятыми долями, т. е. с одним знаком после точки.

Что произойдет, если мы установим значение насыщенности более ста процентов? Например, мы зададим насыщенность какого-либо цвета равной 110 процентам. Исключительной ситуации не возникнет. Ошибочное значение будет всего лишь приведено к ближайшему допустимому, что представляется весьма разумным.

Кроме того, в CSS level 2 добавлен набор предустановленных цветовых значений, которые соответствуют системным установкам цветов для стандартных элементов интерфейса Windows. Все эти значения нечувствительны к регистру, поэтому использование заглавных букв в ключевых словах является необязательным. Но для удобства чтения стоит их писать в том виде, в каком они указаны. Полный перечень предустановленных значений цветов приведен в приложении 2.

Теперь мы уже имеем представление о синтаксисе CSS, мы знаем, как использовать стилевые таблицы в HTML-документах, мы познакомились с

единицами измерения, применяемыми для задания значений свойств. Но мы еще не знаем наименований всех свойств отображения элементов Web-страницы, входящих в состав CSS level 2. Следующие разделы этой главы будут посвящены рассмотрению свойств, используемых в каскадных стилевых таблицах.

Модели представления информации

CSS — это технология, нацеленная в будущее. Она предполагает, что для представления информации будут использоваться разные модели.

В спецификацию CSS level 2 включены следующие модели представления информации, именуемые `media types`.

- `all`. Применяется для обозначения модели представления информации, используемой на всех типах устройств воспроизведения информации.
- `aural`. Используется для описания модели информации, синтезирующей человеческую речь. Да, действительно, CSS level 2 предполагает возможность управления синтезом речи с помощью изменения свойств создаваемого речевого потока. Но к HTML это пока не имеет никакого отношения, поэтому рассматривать детально эту модель мы не будем.
- `braille`. Предназначена для указания модели информации, отображаемой на устройствах, генерирующих последовательность символов алфавита Брайля, применяемого для чтения слепыми людьми.
- `embossed`. Обозначает модель отображения информации на страницах, которые печатаются на специализированных брайлевых принтерах.
- `handheld`. Описывает модель представления информации, рассчитанную на ручные компьютеры, такие, как, например, Palm. Их характеризуют маленькие размеры экрана, монохромность и прочие ограничения.
- `print`. Используется для задания модели отображения информации, которая предназначена для печати на обычных принтерах и отображения в режиме предварительного просмотра.
- `projection`. Применяется для обозначения модели информации, которая будет показана с помощью какого-либо проекционного устройства.
- `screen`. Используется для указания модели представления информации на экране стандартного монитора.
- `tty`. Предназначена для указания модели информации для устройств с ограниченными возможностями отображения, таких как телетайпы, терминалы или переносные подключаемые устройства с ограниченным экраном.
- `tv`. Применяется для задания модели отображения информации на экране телевизора.

Нас же интересует особая модель представления информации, более всего подходящая для отображения документов XML. Мы можем искусственно разбивать такие документы на страницы и отображать их затем соответствующим образом. Для этого применяется страничная модель представления информации (paged media). Подобная страничная модель представления информации часто используется для вывода на печать и режимов предварительного просмотра страниц.

Для отображения информации, разбитой на страницы, применяется специализированное правило, селектором которого служит конструкция `@page`. Это правило следует размещать в самом начале описания страниц, до объявления иных правил, описывающих отображение отдельных элементов страницы. Вариант задания правила отображения отдельной страницы показан в следующем примере:

```
@page { size 8.5in 11in; margin: 2cm }
```

Теперь, когда мы знаем, какую модель представления данных нам надо использовать, мы можем рассмотреть свойства страниц, применяемые в технологии CSS level 2.

Свойство `size` позволяет определять размер страницы. Формальное определение возможных значений данного свойства выглядит следующим образом:

```
<length>{1,2} | auto | portrait | landscape
```

Следует сказать несколько слов о подобных формальных определениях. В них мы используем обозначения, которые ранее не применялись. В фигурных скобках указывается количество значений. В нашем случае мы можем задавать одно или два числовых значения. Если мы указываем два значения, то они разделяются запятой. Также мы можем использовать вместо числовых значений одно из ключевых слов. Все ключевые слова, которые допустимы в этом правиле, перечислены в формальном определении. Они разделены символом вертикальной черты, означающим логическое "ИЛИ". Следовательно, для своих целей мы можем выбрать лишь одно из перечисленных значений. Часто в записи формальных определений применяются угловые скобки, в которых указывается список или множество возможных значений для свойства. В квадратных скобках приводятся необязательные значения.

Как видно, мы можем прямо задавать абсолютный размер страницы по вертикали и горизонтали с помощью пары значений с указанными единицами измерения.

Значением данного свойства может также служить ключевое слово `auto`. При этом размеры страницы будут соответствовать размерам страниц, наиболее подходящих для данного принтера или монитора. Это значение устанавливается по умолчанию.

Значение `portrait` задает портретное или вертикальное расположение страницы. Значение `landscape` определяет горизонтальную ориентацию страницы.

Пример принудительной установки размеров страницы выглядит следующим образом:

```
@page { size: 8.5in 11in;}
```

Свойство `marks` позволяет задавать внешний вид меток верстки, которые могут размещаться в углах области отображения текста. Формальное определение возможных значений свойства выглядит следующим образом:

```
[ crop | cross ] | none
```

Значение по умолчанию `none` указывает на то, что метки верстки вообще не будут отображаться. Значение `crop` отображает метки в виде прямых углов, которые своими концами направлены от центра страницы. Значение `cross` выводит метки в виде обычных крестов.

Информация, представленная в виде страниц, может отображаться или печататься на принтере. При этом страницы делятся на четные и нечетные, и их можно отображать по-разному. Для четных и нечетных страниц можно устанавливать зеркальные поля, а первая страница может оформляться совершенно особым образом.

Для того чтобы различать страницы в CSS level 2 введено три псевдокласса, которые позволяют задавать оформление левых (четных) и правых (нечетных) страниц, а также первой страницы документа.

Псевдокласс `@page:left` предназначен для описания правил отображения левых страниц. Псевдокласс `@page:right` позволяет описывать правые страницы. Правила отображения первой страницы задаются с помощью псевдокласса `@page:first`.

Например, следующая конструкция позволяет нам установить левое и правое поле для левой страницы:

```
@page :left {
  margin-left: 4cm;
  margin-right: 3cm;}
```

Следующие три свойства относятся к размещению блоков на странице. Как мы уже упоминали, каждый элемент оформления вписывается в свой собственный блок-контейнер. Для текстовой информации в такие блоки вписываются абзацы текстового содержимого HTML-документа. При разбиении содержимого на страницы мы можем регулировать порядок размещения блоков на страницах. Свойства `page-break-before` и `page-break-after` позволяют начинать новую страницу до нового блока и после него соответственно. Формальное определение их возможных значений выглядит следующим образом:

```
auto | always | avoid | left | right
```

Значение по умолчанию `auto` перекладывает ответственность постраничного разбиения содержимого в зависимости от наполнения его блоками на отображающее приложение. Значение `always` принудительно устанавливает разрыв страницы до (`page-break-before`) или после каждого нового блока (`page-break-after`). Значение `avoid` принудительно убирает разрывы страниц до блока или после него. Значение `left` форматирует страницы так, что следующая страница после той, которая содержит блок, будет создана как левая (с четным номером). При этом разрыв страниц будет ставиться до (`page-break-before`) или после блока-контейнера (`page-break-after`). Действие значения `right` похоже на действие предыдущего значения, но при этом следующая страница будет создана как правая.

Свойство `page-break-inside` позволяет регулировать применение разрывов страницы внутри блока. Для него допустимы только значения `auto` или `avoid`.

Три описанных свойства устанавливают разрывы страниц, оставляя блоки целыми. Следующие два свойства регулируют разрывы внутри блоков, а если быть точным, внутри абзацев.

Свойство `orphans` указывает на то, какое минимальное количество строк абзаца может переноситься на новую страницу. Значение свойства является целым числом. По умолчанию, на новую страницу переносится две строки. По сути, использование этого свойства накладывает запрет на появление висячих строк.

Свойство `widows` указывает, какое минимальное количество строк из абзаца должно обязательно оставаться на странице. Значение свойства является целым числом. По умолчанию, на странице остается как минимум две строки.

Модели ячеек

Для содержимого каждого структурного элемента HTML-документа резервируется определенное место в окне просмотра браузера. Это место называют ячейкой (box). В CSS существует специальная концепция обработки подобных ячеек, называемая *box model*. Наиболее четко мы можем опознать ячейки в таблицах. Но вопреки расхожему мнению, ячейки могут находиться и вне таблиц. Каждый структурный элемент, отображаемый в окне просмотра браузера, заключается в подобную ячейку как в контейнер.

В CSS различается несколько видов ячеек-контейнеров.

- *Локальный контейнер* в спецификации CSS обозначается как inline box. Локальные контейнеры не могут содержать в себе других ячеек. Это основные, самые маленькие единицы форматирования содержимого HTML-документа. Они могут входить в состав других контейнеров, но сами иметь вложений не могут.

- *Ячейка-блок*, обозначаемая как block box, является базовым элементом разметки. Может содержать в себе иные ячейки.
- *Компактная ячейка*, называемая compact box, является частным случаем локальной ячейки. В компактных ячейках может содержаться только одна строка.
- *Начальная ячейка* (run-in box) размещается в самом начале ячейки-блока. Ближайшим аналогом является первая строка в абзаце.

На основе этой классификации ячеек действуют свойства визуального отображения элементов.

Свойство `display` позволяет идентифицировать вид блока, в котором будет размещаться содержимое того или иного элемента. Отобразится такой блок стилем, установленным по умолчанию для того элемента, наименование которого указано в виде свойства. Если мы укажем `display: table`, то содержимое будет отображаться так же, как таблица. Формальное определение возможных значений данного свойства выглядит так:

```
inline | block | list-item | run-in | compact | marker | table |
inline-table | table-row-group | table-header-group | table-footer-group |
table-row | table-column-group | table-column | table-cell |
table-caption | none | inherit
```

Таким образом, для свойства `display` предусмотрено восемнадцать (!) предустановленных значений. Рассмотрим их.

Значение по умолчанию `inline` означает, что данная ячейка будет квалифицироваться как локальный контейнер. `Block` специфицирует ячейку как ячейку-блок. `List-item` указывает уже не на тип ячейки, а на тип содержимого ячейки. При использовании этого значения свойства `display` считается, что содержимое — элемент списка. Значение `run-in` задает начальную ячейку. `Compact` создает компактную ячейку. `Marker` позволяет обозначать ячейки, содержимое которых находится сразу после или до значка маркера, устанавливаемого перед элементами списков. `Table` указывает, что в данной ячейке находится таблица. `Inline-table` создает таблицу, которая в своих ячейках не может содержать вложенных таблиц. `Table-row-group` обозначает элемент, содержащий несколько строк таблицы. `Table-header-group` задает верхнюю строку таблицы, которая содержит заголовки столбцов. `Table-footer-group` специфицирует нижнюю строку таблицы, которая также может содержать наименования столбцов или итоговые значения столбцов. `Table-row` указывает, что данный элемент содержит строку таблицы. `Table-column-group` специфицирует элемент, содержащий несколько столбцов таблицы. Если наш элемент содержит только один столбец таблицы, то следует использовать значение `table-column`. `Table-cell` говорит о том, что данный элемент является обычной ячейкой таблицы. Для обозначения элемента,

который содержит заголовок таблицы, используется значение table-caption. None применяется для обозначения элементов, которые не отображаются с помощью ячеек-контейнеров. При этом элементы, чье свойство display имеет значение none, принудительно передают это значение всем своим наследникам. Значение none не может быть перекрыто у дочерних элементов. Ключевое слово inherit означает, что свойство display наследуется от родительского элемента, т. е. используется значение свойства, заданное в элементе, содержащем описываемый. На этом разбор свойства display заканчивается, и мы переходим к следующим свойствам, регулирующим общие параметры расположения элементов в окне просмотра браузера.

Свойство position позволяет задавать способ позиционирования элемента в окне просмотра браузера. Формальное определение его возможных значений задается следующей конструкцией:

```
static | relative | absolute | fixed | inherit
```

Значение static, используемое по умолчанию, указывает на то, что данный элемент будет размещен в окне просмотра браузера по стандартным правилам размещения блоков, обрамляющих содержимое элемента. Специализированные свойства, позволяющие задавать координаты верхнего левого угла ячейки, при таком способе размещения не будут иметь силы.

Значение relative устанавливает способ относительного позиционирования для данного элемента и всех остальных дочерних элементов, содержимое которых размещается в ячейке, предназначенной для этого элемента. При этом координаты дочерних ячеек отсчитываются от верхнего левого угла родительской ячейки. С помощью этого типа позиционирования мы смещаем элемент относительно того положения, в котором он отображался бы по умолчанию.

Значение absolute определяет абсолютную систему позиционирования для элемента. Этот способ позволяет указывать все свойства, устанавливающие координаты и размеры ячеек-контейнеров.

Значение fixed задает фиксированную систему позиционирования. Фиксированная система позиционирования очень похожа на абсолютную, но при этом во внимание принимаются ограничения, накладываемые браузером. Если документ отображается в окне просмотра браузера, то учитываются размеры этого окна. При этом если размеры и координаты ячеек выводят их за пределы окна просмотра, то они игнорируются, и ячейки-контейнеры отображаются в пределах окна.

Ключевое слово inherit означает, что значение этого свойства наследуется от родительского элемента.

Если мы используем системы позиционирования, позволяющие устанавливать координаты и размеры ячеек-контейнеров, т. е. все системы, кроме за-

даваемой значением static, то мы можем применять четыре специализированных свойства.

Свойство top позволяет задавать вертикальное смещение элемента, т. е. указывать вертикальную координату верхней границы ячейки-контейнера, содержащей данный элемент. Формальное определение возможных значений данного свойства выглядит следующим образом:

```
<length> | <percentage> | auto | inherit
```

Ключевое слово length означает, что мы можем задать вертикальное смещение с помощью абсолютных или относительных единиц измерения. Либо можно указать его величину в процентах. Значение по умолчанию auto применяется в тех случаях, когда мы в вопросах размещения ячеек-контейнеров полагаемся на сам браузер. Ключевое слово inherit означает, что значение этого свойства наследуется от родительского элемента.

Свойство bottom позволяет задавать вертикальную координату нижней границы ячейки, в которой находится содержимое элемента. Способ задания этого свойства абсолютно идентичен способу задания только что рассмотренного нами свойства top.

Свойства left и right предназначены для установки горизонтальных координат левой и правой границы ячейки-контейнера, соответственно. Значения для них используются те же самые, что и для свойств top и bottom.

Свойство float позволяет сдвигать блок к левой или правой границе окна просмотра. Формальное определение значений выглядит следующим образом:

```
left | right | none | inherit
```

Применение значений left и right генерирует блочную ячейку, которая вне зависимости от ее позиционирования смещается влево или вправо, соответственно. Значение none не смещает принудительно ячейку к краю окна просмотра браузера. Ключевое слово inherit означает, что значение этого свойства наследуется от родительского элемента.

Свойство width позволяет принудительно задавать ширину ячейки-контейнера, в которой будет находиться содержимое элемента. Синтаксис его совпадает с синтаксисом только что рассмотренных нами четырех свойств позиционирования. Необходимо отметить, что свойство width не может применяться к элементам, размещаемым в локальных ячейках, т. к. ширина локальных ячеек определяется браузером автоматически, исходя из объема содержимого отображаемого элемента.

Помимо жесткого задания ширины ячейки-контейнера, мы можем задавать некие ограничения для ширины ячейки. В CSS level 2 предусмотрены свойства, позволяющие ограничивать минимальную и максимальную ширину ячейки.

Свойство min-width позволяет устанавливать минимальную ширину ячейки, в которой находится содержимое данного элемента. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
<length> | <percentage> | inherit
```

Все как обычно. Мы можем задавать минимальную ширину или прямым указанием размера, или в процентах. Ключевое слово inherit означает, что значение этого свойства наследуется от родительского элемента.

Свойство max-width устанавливает максимально возможную ширину ячейки-контейнера. Формальное определение возможных значений этого свойства описывается с помощью следующей конструкции:

```
<length> | <percentage> | none | inherit
```

Как видно, особых отличий от набора значений предыдущего свойства нет. Добавлено лишь еще одно значение none, используемое по умолчанию. Оно указывает, что максимальная ширина ячейки не ограничена.

Кроме ширины мы можем устанавливать и высоту ячеек-контейнеров. Для прямого указания высоты используется свойство height. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
<length> | <percentage> | auto | inherit
```

Значение по умолчанию auto означает, что реальная высота ячейки вычисляется, исходя из других свойств, таких как top и bottom. Механизм действия остальных значений мы прекрасно помним.

В CSS level 2 существуют свойства, задающие "мягкие условия" для ширины ячейки-контейнера. Для высоты, естественно, существуют их аналоги.

Свойство min-height позволяет задавать минимальную высоту ячейки-контейнера. Формальное определение возможных значений этого свойства выглядит так:

```
<length> | <percentage> | inherit
```

По умолчанию свойству min-height присваивается нулевое значение.

Свойство max-height позволяет установить максимальную высоту ячейки-контейнера, которая не может быть превышена. Формальное определение значений этого свойства выглядит следующим образом:

```
<length> | <percentage> | none | inherit
```

К набору значений предыдущего свойства здесь добавлено значение none, используемое по умолчанию, которое сообщает браузеру, что искусственных ограничений максимальной высоты у данной ячейки нет.

Вышеперечисленные свойства принудительной установки ширины и высоты ячейки не имеют силы, если содержимое элемента отображается в локальной ячейке. Ширина и высота локальных ячеек определяется их содержи-

мым. Но для указания высоты локальных ячеек все-таки существуют специальные свойства CSS level 2.

Поскольку чаще всего локальная ячейка является отдельной строкой, для установки ее высоты используется значение `line-height`. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
normal | <number> | <length> | <percentage> | inherit
```

Значение по умолчанию `normal` указывает, что высота ячейки будет вычисляться браузером, исходя из размера применяемого шрифта, полей и отступов. Конструкция `<number>` в синтаксическом определении означает, что для задания высоты локальной ячейки можно использовать коэффициент, на который умножается высота применяемого шрифта. Помимо этого способа задания высоты, мы можем применить стандартные единицы измерения и процентные соотношения. Ключевое слово `inherit`, как обычно, означает, что значение этого свойства наследуется от родительского элемента.

Вертикальное выравнивание локальной ячейки производится с помощью свойства `vertical-align`. Формальное определение возможных значений этого свойства описывается с помощью следующей конструкции:

```
baseline | sub | super | top | text-top | middle | bottom | text-bottom |
<percentage> | <length> | inherit
```

Значение по умолчанию `baseline` вертикально выравнивает локальную ячейку по уровню базовой линии родительского элемента. Если у родительского элемента нет базовой линии, то выбирается ближайший в объектной иерархии элемент-предок, у которого эта базовая линия есть. Базовой линией обычно называют нижний край стандартных символов применяемого шрифта, т. е. тех, у которых нет "нижних хвостиков".

Значение `sub` устанавливает данную локальную ячейку чуть ниже уровня базовой линии, позиционируя ее так же, как нижние индексы. Необходимо отметить, что данное свойство может применяться только в том случае, если размер ячейки родительского элемента больше, чем размер шрифта, которым отображается текстовое содержимое этого родительского элемента.

Значение `super` наоборот позиционирует локальную ячейку на уровне верхнего индекса, применяемого к базовой линии ячейки родительского элемента.

Значение `top` позволяет прижать локальную ячейку к самому верху родительской ячейки. Альтернативное значение `bottom` прижимает локальную ячейку к нижней границе ячейки, в которой базируется родительский элемент. А значение `middle` центрирует локальную ячейку по вертикали в пространстве родительской ячейки.

Значение `text-top` выравнивает локальную ячейку по уровню верхней линии используемого шрифта. При этом используется самая верхняя строка.

Значение `text-bottom` производит выравнивание по нижней линии используемого шрифта.

Конструкции `<percentage>` и `<length>` указывают, что мы можем задавать вертикальное выравнивание с использованием процентов или прямого указания его величины в стандартных единицах измерения. При этом смещение отсчитывается от базовой линии, т. е. от того уровня, который был бы установлен при использовании значения `baseline`. Положительные значения смещают ячейку вверх относительно базовой линии, отрицательные — вниз.

Ключевое слово `inherit` заставляет данную локальную ячейку наследовать выравнивание по вертикали у родительского элемента.

Помимо какого-либо конкретного содержимого ячеек-контейнеров, браузер обязан учитывать дополнительные элементы оформления, входящие в состав модели ячеек. Их называют *отступами* (padding), *границами* (border) и *полями* (margin). Взаимное их расположение показано на рис. 2.4.

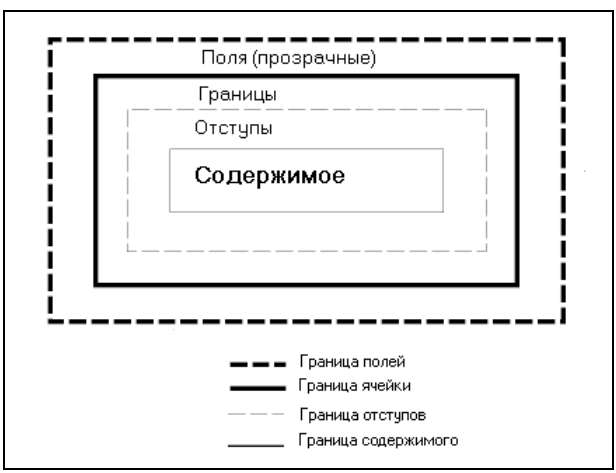


Рис. 2.4. Схема элементов обрамления ячеек-контейнеров

Для регулировки размеров полей применяются свойства `margin-left`, `margin-right`, `margin-top` и `margin-bottom`. Они позволяют явно задавать размер левого, правого, верхнего и нижнего поля, соответственно. Формальное определение их возможных значений одинаково и описывается следующей конструкцией:

```
<margin-width> | inherit
```

О действии ключевого слова `inherit` мы уже знаем. Оно вынуждает элемент унаследовать значение данного свойства у породившего элемента. Конструкция `<margin-width>` определена в спецификации CSS level 2. Она описывает три возможности. Мы можем либо явно указать размер полей с по-

мощью стандартных единиц измерения, либо использовать проценты. Кроме того, значением этого свойства может служить ключевое слово auto, и величина полей определяется браузером, исходя из его общих установок. По умолчанию для этих свойств устанавливается нулевое значение.

Существует также агрегатное свойство margin, позволяющее указывать размеры всех полей, не прибегая к помощи только что рассмотренных свойств. Формальное определение его значений задается с помощью следующей конструкции:

```
<margin-width>{1,4} | inherit
```

Как видно, мы можем устанавливать от одного до четырех размеров полей. При этом соблюдается такой порядок указания полей: верхнее поле, правое, нижнее, левое. Если указаны не все величины, то горизонтальные или вертикальные поля объединяются. Более подробно это можно увидеть в приведенной далее серии примеров.

Для того чтобы установить все поля величиной два миллиметра, используется следующая конструкция:

```
BODY { margin: 2mm }
```

Если же указаны всего два значения, то первое из них устанавливает размер вертикальных полей, а второе — горизонтальных. Проиллюстрировать этот случай можно так:

```
BODY { margin: 1mm 2mm }
```

При этом для вертикальных полей будет установлен размер, равный одному миллиметру, а для горизонтальных — двум миллиметрам.

Теперь определим значение этого свойства тремя величинами, в этом случае совпадут только горизонтальные поля. Проиллюстрировать такой вариант можно с помощью следующего фрагмента кода:

```
BODY { margin: 1mm 2mm 3mm }
```

При этом для верхнего поля будет установлено значение, равное одному миллиметру, для правого поля — двум миллиметрам, а для нижнего — трем миллиметрам. Поскольку величина левого поля не была задана явно, то левое поле будет иметь такой же размер, как и правое. То есть — два миллиметра.

Отступы задаются с помощью соответствующих свойств. Свойства padding-top, padding-bottom, padding-right и padding-left позволяют задавать верхний, нижний, правый и левый отступы, соответственно. Формальное определение возможных значений этих свойств описывается следующей конструкцией:

```
<padding-width> | inherit
```

Конструкция <padding-width> означает, что для указания размера отступа мы можем использовать проценты или обычные единицы измерения CSS level 2.

Автоматическое задание величины отступа с помощью ключевого слова auto не допускается. По умолчанию используется нулевой отступ. Ключевое слово inherit указывает, что значение данного свойства необходимо унаследовать от порождающего элемента.

Как и при указании размеров полей, мы можем использовать единое агрегатное свойство для задания размеров отступов. Оно носит наименование padding. Синтаксическое определение его значений выглядит следующим образом:

```
<padding-width>{1,4} | inherit
```

Как видно, механизм задания значений для этого свойства совпадает с порядком установки значений свойства margin.

С помощью свойств border-top-width, border-right-width, border-bottom-width и border-left-width мы можем задавать толщину границы ячейки, в которой отображается содержимое элемента. Они регулируют толщину верхней, правой, нижней и левой границы соответственно.

Возможные значения этих свойств определяются с помощью следующей конструкции:

```
<border-width> | inherit
```

Определение <border-width> в спецификации CSS level 2 для указания толщины позволяет использовать стандартные единицы измерения. Применение процентов или автоматической установки ширины не допускается. Помимо прямого задания толщины можно использовать три ключевых слова: thin, medium и thick, которые задают, соответственно, тонкую, среднюю и толстую границу. По умолчанию применяется значение medium.

Естественно, есть и агрегатное свойство border-width, позволяющее задавать толщину всех границ ячейки. Формальное определение значений данного свойства выглядит следующим образом:

```
<border-width>{1,4} | inherit
```

Стандартные правила задания значений агрегатных свойств действуют и в этом случае.

Конечно, границы не могут характеризоваться одной толщиной. В CSS level 2 описаны свойства border-top-color, border-right-color, border-bottom-color и border-left-color, которые позволяют устанавливать цвет верхней, правой, нижней или левой границы ячейки, соответственно. Формальное определение их значений описывается такой конструкцией:

```
<color> | inherit
```

Ключевое слово inherit, естественно, предназначено для наследования значения данного свойства от порождающего элемента. Способы задания цвета мы рассматривали ранее в *разд. "Единицы измерения в CSS"* этой главы.

Существует и обобщающее свойство border-color, которое позволяет устанавливать цвет для всех границ сразу. Формальное определение значений выглядит следующим образом:

```
<color>{1,4} | transparent | inherit
```

Помимо знакомого нам ключевого слова inherit и стандартного указания цвета, для установки этого свойства мы можем использовать значение transparent. При этом граница ячейки будет прозрачной. Естественно, для этого она должна иметь ненулевую толщину.

Кроме цвета, граница ячейки может (и должна) иметь определенный стиль. Стиль границы позволяет определить, как именно будет выглядеть линия, ограничивающая ячейку, в которой отображается какой-либо элемент. Применяются для этого свойства border-top-style, bottom-right-style, border-bottom-style и border-left-style. Они позволяют устанавливать внешний вид верхней, правой, нижней и левой границы ячейки, соответственно.

Значением этих свойств может быть одно из нижеперечисленных ключевых слов.

None просто убирает границу. Действие этого варианта идентично установке нулевого значения свойства border-width.

Ключевое слово hidden создает границу, но делает ее невидимой. Этот вариант отличается от предыдущего, т. к., несмотря на невидимость, граница все же существует, что позволяет более корректно размещать ячейки-контейнеры без наложения их друг на друга.

Ключевое слово dotted, используемое как значение этого свойства, создает границу в виде серии точек.

Значение dashed указывает, что границы ячеек-контейнеров будут проведены пунктирными линиями.

Для установки обычных границ в виде непрерывных тонких линий используется значение solid.

Значение double заставляет браузер отображать границу ячейки-контейнера в виде двух непрерывных линий. При этом расстояние между линиями определяется шириной границы, т. е. значением свойства border-width.

Ключевое слово groove позволяет отображать границу ячейки в виде некоего углубления, создавая иллюзию трехмерности.

Значение ridge создает эффект выпуклости границы. Оно заставляет браузер отображать границу ячейки-контейнера в виде трехмерного выступа.

Для рисования прямоугольной канавки используется значение `inset`. Отличие этого значения от рассмотренного нами ключевого слова `groove` заключается в том, что создаваемый желобок имеет прямоугольную форму.

Ключевое слово `outset`, применяемое как значение рассматриваемого свойства, создает границу в виде прямоугольного выступа.

Необходимо отметить, что последние четыре рассмотренных нами значения, создающие трехмерные границы, отображаются с помощью цвета, который установлен для самого содержимого элемента. То есть, цвет трехмерных границ зависит от значения свойства `color`, заданного для элемента в целом.

Рассматриваемые свойства могут иметь не только описанные нами значения, но и наследовать их от порождающих элементов. Для этого, как всегда, используется ключевое слово `inherit`.

Агрегатное свойство `border-style` позволяет задавать стили отображения для всех границ сразу. Формальное определение его возможных значений задается с помощью следующей конструкции:

```
<border-style>{1,4} | inherit
```

Как легко заметить, значения для всех агрегатных свойств задаются единообразно.

Помимо рассмотренных нами свойств существуют обобщающие свойства. Так свойства `border-top`, `border-right`, `border-bottom` и `border-left` позволяют сразу полностью устанавливать внешний вид верхней, правой, нижней и левой границы ячейки-контейнера, соответственно. Формальное определение значений этих свойств одинаково и выглядит так:

```
[ <'border-width'> || <'border-style'> || <color> ] | inherit
```

Таким образом, для каждой границы мы можем установить сразу ее ширину, стиль отображения и цвет. Например, следующее правило оформления для элемента `caption` задает параметры его нижней границы.

```
caption { border-bottom: thick solid }
```

В этом правиле мы устанавливаем толщину нижней границы и ее стиль. Нижняя граница будет отображена в виде толстой непрерывной линии.

Следует отметить, что если у рассматриваемых свойств какое-либо из трех значений не задано, то используется значение, установленное для него по умолчанию.

Если все четыре границы должны выглядеть одинаково, то имеет смысл для задания правила оформления использовать единое свойство `border`. Формальное определение его возможных значений описывается следующей конструкцией:

```
[ <'border-width'> || <'border-style'> || <color> ] | inherit
```

Формальное определение значений этого свойства повторяет определение значений предыдущих четырех свойств. Правила задания значений тоже совпадают. Следующее правило отображения позволяет устанавливать границу для всей ячейки-контейнера сразу:

```
text { border: solid red }
```

При этом вся граница ячейки, в которой находится содержимое элемента `text`, будет отображена в виде непрерывной красной линии.

Обратите внимание на то, что в отличие от агрегатных свойств `margin` или `padding`, свойство `border` не позволяет устанавливать внешний вид для каждой из четырех границ отдельно. В этом случае они отображаются одинаково.

Некоторые ячейки могут перекрывать друг друга. Для определения, какая именно ячейка будет отображаться сверху, используется свойство `z-index`. Это свойство позволяет создавать иерархию слоев. Формальное определение возможных значений этого свойства задается с помощью следующей конструкции:

```
auto | <integer> | inherit
```

Значение по умолчанию `auto` позволяет браузеру самостоятельно выстраивать иерархию слоев у пересекающихся ячеек. При этом элементы, которые расположены ближе к концу HTML-документа, отображаются "ближе" к пользователю, т. е., отображение элементов происходит по очереди, начиная с начала документа. Каждый раз, когда последующий элемент перекрывает предыдущий, он отображается сверху.

Мы можем также в качестве значения этого свойства использовать целое положительное число. Нуль — значение, обладающее наивысшим приоритетом. Элемент, у которого есть нулевое значение свойства `z-index`, будет всегда отображаться сверху. Таким образом, чем больше значение свойства `z-index`, тем ниже приоритет элемента. Ключевое слово `inherit` означает, что значение этого свойства наследуется от родительского элемента.

Текстовое содержимое элементов может отображаться в ячейках-контейнерах в различном направлении. Для указания этого направления используется свойство `direction`. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
ltr | rtl | inherit
```

Значение `ltr` заставляет браузер отображать текст слева направо, привычным для нас образом. Значение `rtl` вынуждает отображать текстовое содержимое данного элемента справа налево. Значение `inherit` означает, что направление отображения текста наследуется от родительского элемента.

Мы разобрались со стандартными вариантами размещения содержимого элементов в ячейках-контейнерах. Но в CSS level 2 предусмотрена возмож-

ность обработки ситуации, при которой объем отображаемого содержимого элемента превышает размеры отведенной ему ячейки. Подобная ситуация называется *переполнением* (overflow). Для того чтобы указать браузеру, какие действия необходимо произвести при переполнении, предусмотрено соответствующее свойство `overflow`. Формальное определение его возможных значений выглядит следующим образом:

```
visible | hidden | scroll | auto | inherit
```

Значение по умолчанию `visible` указывает браузеру, что содержимое данного элемента не прикреплено (clipped) к ячейке и может отображаться за ее пределами. Значение `hidden` означает, что содержимое данной ячейки прикреплено к ней, и если часть содержимого выходит за пределы ячейки, то эта часть остается невидимой, т. к. полосы прокрутки для этой ячейки не создаются. Значение `scroll` также прикрепляет содержимое к ячейке, но при этом позволяет браузеру создавать для ячейки полосы прокрутки, если ее содержимое выходит за пределы области просмотра. Использование значения `auto` оставляет выбор поведения за самим HTML-процессором. Ключевое слово `inherit` означает, что значение этого свойства наследуется от родительского элемента.

Мы можем устанавливать форму и размер области, в которой находится прикрепленное содержимое. Для этого используется свойство `clip`. Его возможные значения описываются с помощью следующей конструкции:

```
<shape> | auto | inherit
```

Конструкция `<shape>` указывает, что мы можем явно задать форму и размеры области прикрепления. К сожалению, в CSS level 2 можно задавать только прямоугольную форму с помощью ключевого слова `rect`. После этого ключевого слова в скобках следует указать координаты `top`, `right`, `bottom`, `left`. При этом необходимо учитывать, что мы можем задавать не только положительные, но и отрицательные значения. Например, для того чтобы задать область прикрепления, смещенную на пять пикселов вправо за границу ячейки-контейнера, следует использовать следующую конструкцию:

```
P { clip: rect(5px, -5px, 10px, 5px); }
```

Значение `auto`, используемое по умолчанию, распространяет область прикрепления на всю ячейку, в которой отображается содержимое элемента. Ключевое слово `inherit` заставляет элемент страницы унаследовать значение этого свойства от своего родителя.

Для управления видимостью ячейки-контейнера используется свойство `visibility`. Возможные его значения описываются следующей конструкцией:

```
visible | hidden | collapse | inherit
```

Значение `visible` заставляет браузер отображать данную ячейку. Значение `hidden` делает ее невидимой. Значение `collapse` сворачивает ячейку-кон-

тейнер. Этот вариант правильно работает только в том случае, когда ячейка является частью таблицы. Иначе, результат действия эквивалентен использованию значения `hidden`. По умолчанию используется значение `inherit`, принуждающее наследовать свойство `visibility` от родительского элемента.

Фон и цвета

В этом разделе мы рассмотрим свойства CSS level 2, которые регулируют цветовое оформление текстового содержимого элементов HTML-документа и управляют отображением фона элементов.

Свойство `color` задает цвет, которым будет отображаться содержимое элемента. Формальное определение возможных значений данного свойства обозначается следующим образом:

```
<color> | inherit
```

В качестве значения `color`, этого свойства, применяется обозначение цвета. Варианты этих обозначений мы рассматривали в *разд. "Единицы измерения в CSS"* настоящей главы. Значение по умолчанию отсутствует. Ключевое слово `inherit` означает, что для данного элемента значение этого свойства будет унаследовано от порождающего элемента.

Свойство `background-color` позволяет задавать цвет фона для любого элемента. Формальное определение возможных значений данного свойства обозначается следующим образом:

```
<color> | transparent | inherit
```

Значение `color`, как и в предыдущем свойстве, является обозначением какого-либо цвета и применяется для прямого назначения цвета фона элемента. Значение `transparent` используется по умолчанию и задает прозрачный цвет фона. Ключевое слово `inherit` означает, что для данного элемента цвет фона будет унаследован от порождающего элемента.

Свойство `background-image` позволяет задать в качестве фона для содержимого элемента графическое изображение. Формальное определение возможных значений данного свойства описывается следующим образом:

```
<uri> | none | inherit
```

Конструкция `<uri>` указывает, что мы можем использовать URI (Universal Resource Identifier) графического изображения как значение данного свойства. При этом оно будет являться фоном для содержимого нашего элемента. Значение `none`, применяемое по умолчанию, свидетельствует о том, что графического фона у данного элемента не будет. Ключевое слово `inherit` означает, что для данного элемента правила использования графики как фона будут унаследованы от порождающего элемента.

Замечание

URI является усложненным вариантом URL. Это понятие введено для более четкой идентификации различных ресурсов в Интернете. Отличается от URL добавлением усложненной адресации, позволяющей находить различные фрагменты в отдельных документах. Подобные продвинутые возможности будут применяться только в языке XML (eXtensible Markup Language), который, как ожидается, со временем заменит язык HTML. Однако это время настанет еще не скоро, и пока что URI полностью совпадает с URL.

В следующем примере показано, как для класса `caption` мы задаем в качестве фона графическое изображение.

```
.caption {background-image :url ("http://www.site.com/pict1.gif")}
```

Как видно из примера, для указания URL графического файла мы используем функцию `url`, которая текстовую строку преобразует в URL.

Свойство `background-repeat` позволяет более детально управлять фоном элемента, если в качестве фона применяется графическое изображение. Формальное определение возможных значений данного свойства выглядит следующим образом:

```
repeat | repeat-x | repeat-y | no-repeat | inherit
```

Напомню, что это свойство имеет смысл применять только в том случае, когда в качестве фона элемента явно установлено графическое изображение. Так как изображение не всегда полностью занимает рабочее пространство окна просмотра браузера, оно может повторяться в нем, закрывая его по вертикали и/или горизонтали. Свойство `background-repeat` как раз и управляет подобным повторением.

Значение по умолчанию `repeat` указывает, что если графическое изображение не полностью покрывает рабочее пространство окна просмотра браузера, оно будет повторяться по вертикали и горизонтали. Значение `repeat-x` позволяет дублировать фоновый рисунок только по горизонтали. Значение `repeat-y` позволяет повторять рисунок только по вертикали. Значение `no-repeat` явно запрещает повторение фонового рисунка как по вертикали, так и по горизонтали. Ключевое слово `inherit` означает, что для данного элемента значение этого свойства будет унаследовано от порождающего элемента.

Свойство `background-attachment` позволяет устанавливать "привязку" фонового рисунка к окну просмотра. Дело в том, что если фоновое изображение привязано к текстовому содержимому HTML-документа, то при прокрутке документа в окне просмотра браузера фоновый рисунок будет перемещаться вместе с текстовым содержимым элемента. А если фон привязан к окну просмотра, то при прокрутке смещаться будет только содержимое документа, а фоновый рисунок будет оставаться на месте. Данное свойство позволя-

ет управлять поведением фонового рисунка. Формальное определение значений этого свойства выглядит следующим образом:

```
scroll | fixed | inherit
```

Значение по умолчанию `scroll` указывает, что фоновое графическое изображение будет при прокрутке перемещаться вместе с содержимым документа. Значение `fixed` "привязывает" фоновый рисунок к окну просмотра браузера. Ключевое слово `inherit` означает, что для данного элемента привязка фонового рисунка будет унаследована от порождающего элемента.

Свойство `background-position` позволяет указывать позицию фонового рисунка в окне просмотра браузера. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
[ [<percentage> | <length> ]{1,2} | [ [top | center | bottom] || [left |
center | right] ] ] | inherit
```

Проанализируем внимательно это запутанное выражение. С помощью данного свойства мы фактически задаем координаты фонового рисунка в окне. Для этого мы можем использовать либо два процентных соотношения, либо два числа, напрямую указывающих отступы рисунка от границ окна просмотра, либо комбинацию из двух ключевых слов. Рассмотрим детально способы задания значений свойства.

Пара процентных выражений определяет координаты верхнего левого угла фонового рисунка. При этом задаются проценты от длины и ширины ячейки прямоугольника, в котором размещается содержимое элемента. Так, следующая конструкция помещает фоновый рисунок в левый верхний угол прямоугольника-контейнера, в котором размещается содержимое элемента `body`:

```
body {background-position : 0% 0%}
```

При этом первое процентное значение показывает смещение по горизонтали, второе — по вертикали от верхнего левого угла контейнера. В том случае, если мы задаем достаточно большие процентные величины, например `100% 100%`, то изображение прижимается к правому нижнему углу, но, тем не менее, отображается.

Помимо процентных значений мы можем указывать смещение фонового рисунка, используя абсолютные и относительные единицы измерения. Данный способ иллюстрирует следующее правило CSS:

```
body {background-position : 15 mm 20 mm}
```

В этом правиле мы указываем, что фоновое графическое изображение следует сместить относительно левого верхнего угла на 15 миллиметров по горизонтали и на 20 миллиметров по вертикали.

Также мы можем использовать ключевые слова, приведенные в синтаксическом определении возможных значений. Для позиционирования рисунка по горизонтали применяются ключевые слова `left`, `center` и `right`. Они позволяют прижимать рисунок к левому краю ячейки, центрировать его или прижимать к правому краю ячейки, соответственно. Ключевые слова `top`, `bottom` и `center` определяют вертикальное положение фонового рисунка и позволяют прижимать рисунок к верхнему или нижнему краю ячейки-контейнера, а также центрировать по вертикали, соответственно.

Все вышеперечисленные свойства фона можно объединить в одном агрегатном свойстве `background`. Формальное определение его возможных значений выглядит так:

```
[<'background-color'> || <'background-image'> || <'background-repeat'> ||
<'background-attachment'> || <'background-position'>] | inherit
```

Таким образом, мы можем в одном свойстве задать все параметры оформления фона. Проиллюстрировать это можно следующим примером:

```
P { background: url("chess.png") gray 50% repeat fixed }
```

В этом правиле оформления мы для элемента `P` устанавливаем фоновое изображение. Графический файл, в котором оно находится, имеет URL chess.png. В том случае, если графический файл не будет найден, то фон будет залит серым цветом. Фоновый рисунок сдвигается на пятьдесят процентов по горизонтали, может повторяться по вертикали или горизонтали, если ячейка-контейнер больше по размерам, нежели само графическое изображение. Также фоновый рисунок прикрепляется к окну просмотра, и не будет перемещаться вместе с текстовым содержимым при использовании полос прокрутки.

Итак, мы рассмотрели все свойства, регулирующие применение фона и установку цвета текстового содержания элемента. Можно идти дальше.

Шрифтовое оформление

В документах, предназначенных для размещения в Интернете, нет и не может быть максимально четкого шрифтового оформления. Связано это с тем, что заранее неизвестно, какая операционная система, набор шрифтов, разрешение монитора и размер окна просмотра браузера установлены удаленным пользователем, который собирается просмотреть наш документ. Исходя из опыта, мы можем предположить, что, скорее всего, у подавляющего большинства пользователей Интернета установлен в системе какой-либо шрифт семейства Times New Roman или Arial. Следовательно, мы можем регулировать хотя бы немного параметры шрифтового оформления текста. Даже если указанных нами шрифтов и вариантов их воспроизведения не окажется у удаленного пользователя, это не беда. Браузер проигнорирует данные нами инструкции и отобразит текст с установками, заданными по

умолчанию. Но вполне вероятно, что мы все-таки добьемся своего. Стало быть, использовать свойства шрифтового оформления текста в каскадных стилевых таблицах стоит. Приступим к их рассмотрению.

Свойство `font-family` задает наименование семейства шрифта, используемого для отображения текстового содержимого элемента. Формальное определение его значений имеет следующий вид:

```
[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>]
```

Из определения видно, что значение этого свойства может состоять из нескольких наименований семейств шрифтов. Вместо имени семейства шрифтов мы можем задать значение `generic-family`, которое позволяет управлять начертанием шрифта без указания конкретного семейства. Параметр `generic-family` может принимать значения: `serif`, `sans-serif`, `cursive`, `fantasy`, `monospace`.

Значение `serif` указывает, что для оформления применяется шрифт с засечками. Типичный представитель — Times New Roman. Значение `sans-serif` обозначает шрифт без засечек (Helvetica). Для обозначения наклонных шрифтов применяется значение `cursive`. Значение `fantasy` применяется для так называемых "фантазийных" шрифтов, например готических. Для обозначения моноширинных шрифтов, таких как Courier, используется значение `monospace`.

Как мы уже говорили, в значении свойства `font-family` мы можем указать сразу несколько семейств шрифтов. При этом браузер будет подбирать шрифты для отображения текста в том порядке, в каком они перечислены. Рассмотрим пример правила отображения, в котором устанавливается шрифт для текста.

```
element1 {font-family: Arial, fantasy, "Courier New"}
```

Браузер для отображения текстового содержимого элемента с наименованием `element1` сначала попытается использовать шрифт из семейства Arial. Затем, если ни один из шрифтов этого семейства не будет найден в системе пользователя, браузер попробует отобразить содержимое любым фантазийным шрифтом. Если же и их не будет, то браузер будет искать моноширинный шрифт Courier New.

Если ни один из перечисленных шрифтов не будет найден в системе, браузер использует шрифт, установленный по умолчанию.

Обратите внимание: если наименование семейства шрифта состоит из нескольких слов, оно заключается в двойные кавычки, как это показано в примере.

Свойство `font-style` может принимать только одно из четырех предустановленных значений. Формальное определение значений свойства `font-style` выглядит следующим образом:

```
normal | italic | oblique | inherit
```

Первые три значения позволяют принудительно задавать начертание шрифта. Последнее значение `inherit` указывает на то, что для данного элемента значение `font-style` будет наследоваться от родительского элемента.

Значение `normal`, применяемое по умолчанию, задает использование обычного начертания шрифта. Значение `italic` принудительно переводит выбранный шрифт в курсивное состояние. Значение `oblique` создает наклонное начертание выбранного шрифта.

Данное свойство является наследуемым. Может применяться ко всем типам элементов.

Свойство `font-variant` позволяет принудительно устанавливать регистр символов применяемого шрифта. Формальное определение возможных значений задается следующим образом:

```
normal | small-caps | inherit
```

Значение по умолчанию `normal` оставляет символы текста в том виде, как они были введены. Значение `small-caps` принудительно преобразует строчные символы в заглавные, но при этом их размер не превышает размера обычных строчных символов. Значение `inherit` означает, что истинное значение данного свойства необходимо унаследовать от родительского элемента.

Данное свойство может применяться к любому элементу.

Свойство `font-weight` позволяет задавать насыщенность символов применяемого шрифта для текстового содержания данного элемента. То есть с помощью этого свойства мы можем управлять толщиной символов. Формальное определение значений данного свойства выглядит следующим образом:

```
normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 |
700 | 800 | 900 | inherit
```

Числовые значения позволяют задавать толщину символов девяти градаций. Наименьшее значение соответствует наиболее тонкому начертанию символов. Помимо численных значений можно использовать символьные обозначения для четырех значений толщины символов.

Значение `normal` оставляет символы стандартной толщины и соответствует численному значению 400. Это значение используется по умолчанию. Значение `bold` соответствует численному значению 700, `bolder` — 900, `lighter` — 100.

Значение `inherit` используется, как обычно, для наследования значения этого свойства от родительского элемента.

Свойство `font-weight` применяется для любого типа элементов, имеющих текстовое наполнение.

Свойство font-stretch позволяет задавать межсимвольный интервал в словах. Формальное определение значений этого свойства выглядит так:

```
normal | wider | narrower | ultra-condensed | extra-condensed |
condensed | semi-condensed | semi-expanded | expanded | extra-expanded |
ultra-expanded | inherit
```

Значение по умолчанию normal оставляет нормальный межсимвольный интервал. Остальные значения позволяют принудительно устанавливать интервал, отличающийся от нормального. Если перечислять их в порядке задания интервала от самого узкого до максимально широкого, получится следующая последовательность:

```
ultra-condensed, extra-condensed, condensed, semi-condensed, normal,
semi-expanded, expanded, extra-expanded, ultra-expanded
```

В приведенной последовательности отсутствуют значения wider и narrower. В отличие от вышеперечисленных значений, эти два значения не абсолютные, а относительные. Значение wider изменяет унаследованное от родительского элемента значение свойства font-stretch на одну ступень в сторону более широкого межсимвольного интервала. Естественно, если унаследованное значение равно ultra-expanded, то дополнительного сдвига не происходит. Значение narrower изменяет текущее значение на один уровень градации в направлении к более узкому межсимвольному интервалу.

Значение inherit, как обычно, используется для наследования значения свойства от родительского элемента.

Свойство применяется ко всем элементам, имеющим текстовое содержимое.

Свойство font-size регулирует величину символов применяемого шрифта. Формальное определение возможных значений данного свойства достаточно сложно, и выглядит так:

```
<absolute-size> | <relative-size> | <length> | <percentage> | inherit
```

В приведенном определении только значение inherit задано явно. Все остальные типы значений указаны в виде множеств.

Множество absolute-size содержит семь предустановленных значений, которые задают абсолютный размер шрифта. Если перечислить их в порядке увеличения размера шрифта, получится следующая последовательность:

```
xx-small, x-small, small, medium, large, x-large, xx-large
```

Конкретное пропорциональное соотношение между размерами, которые задаются этими значениями, зависит от разрешения монитора, установленного удаленным пользователем. В описании правила используется одно значение из приведенных.

Множество relative-size состоит из двух возможных значений. Они используются для задания размера символов данного элемента относительно

размера символов шрифта родительского элемента. Значение `larger` увеличивает размер символов относительно размера символов родительского элемента, а значение `smaller`, естественно, — уменьшает.

Ключевое слово `length` указывает, что мы можем прямо задать размер шрифта с помощью абсолютных единиц измерения, таких как пункты или миллиметры.

Кроме того, мы можем задать размер шрифта в процентах от размера шрифта родительского элемента, о чем говорит ключевое слово `percentage`.

Значение `inherit`, как обычно, используется для наследования значения свойства от родительского элемента. По умолчанию используется значение `medium`. Свойство применяется ко всем элементам, имеющим текстовое содержимое.

Свойство `font-size-adjust` позволяет задавать коэффициент увеличения шрифта. Данное свойство применяется для более точной установки размера символов и напрямую связано с понятием коэффициента масштабирования шрифта. Этот коэффициент определяется отношением абсолютного размера символов шрифта к высоте стандартного символа 'x'. То есть, чем больше этот коэффициент, тем более вытянутыми становятся символы шрифта.

Формальное определение возможных значений данного свойства имеет следующий вид:

```
<number> | none | inherit
```

Таким образом, мы можем либо прямо указать коэффициент масштабирования, либо воспользоваться значением по умолчанию `none`, которое оставляет размеры шрифта без изменения.

Коэффициент масштабирования применяется к шрифту, установленному для родительского элемента. Значение `inherit` используется для наследования значения свойства от родительского элемента. Свойство применимо ко всем элементам, имеющим текстовое содержимое.

Свойство `font` является агрегатным свойством, которое позволяет объединить все рассмотренные параметры шрифта в одном объявлении правила отображения. Формальное определение возможных значений имеет следующий вид:

```
[ [ <'font-style'> || <'font-variant'> || <'font-weight'> ]? <'font-
size'> [ / <'line-height'> ]? <'font-family'> ] | caption | icon | menu |
message-box | small-caption | status-bar | inherit
```

Мы можем использовать несколько вариантов задания свойства `font`. Например, указать просто триаду значений свойств `font-style`, `font-variant` и `font-weight`. Причем если мы не укажем явно значение для одного из этих свойств, то анализатор CSS все равно это свойство распознает и подставит

значение, использующееся по умолчанию для данного свойства. Примером может служить следующая конструкция:

```
P { font: 80% sans-serif }
```

В этом правиле мы указываем, что для текстового содержимого элемента P мы используем шрифт, размер которого составляет 80% от размера шрифта родительского элемента, и применяем шрифт без засечек. Мы задаем только значения свойств font-size и font-family. Для остальных свойств используются значения по умолчанию.

Помимо известных нам свойств, чьи значения мы можем включать в агрегатное свойство font, существует еще несколько предустановленных значений, которые задают вид и начертание используемого шрифта.

Значение caption устанавливает шрифт, идентичный шрифту, применяемому для отображения наименований различных элементов управления. Значение icon ссылается на шрифт, которым отображаются подписи к пиктограммам. Значение menu создает шрифт аналогичный используемому в выпадающих списках и контекстных меню. Для указания шрифта, которым выводится информация в окнах сообщений, используется значение message-box. Значение small-caption задает шрифт, который применяется для заголовков маленьких элементов управления. Значение status указывает на шрифт, которым отображается информация в статусной строке.

Значение inherit используется для наследования значения свойства от родительского элемента. Свойство применимо ко всем элементам, имеющим текстовое содержимое.

Для того чтобы оценить выгоду от применения агрегатного свойства font, достаточно рассмотреть два равноценных правила отображения текстового содержимого элемента P.

```
P { font: 600 9pt Charcoal }
P {
font-style: normal;
font-variant: normal;
font-weight: 600;
font-size: 9pt;
line-height: normal;
font-family: Charcoal
}
```

Свойство text-decoration позволяет установить эффекты оформления текста. Формальное определение возможных значений описывается следующим образом:

```
none | [ underline || overline || line-through || blink ] | inherit
```

Значение по умолчанию `none` оставляет отображаемый текст без изменений. Если же мы хотим каким-либо образом декорировать текст, следует использовать комбинацию специализированных значений. Значение `underline` заставляет браузер подчеркивать каждую строку текста, входящего в состав элемента, к которому мы применяем данное правило оформления. Значение `overline` позволяет нам отобразить горизонтальную линию над строкой текста. Перечеркнуть текст можно с помощью значения `line-through`. Впрочем, декорирование текста не ограничивается использованием горизонтальных линий. Мы можем заставить текст мигать. Для этого следует использовать значение `blink`. Ключевое слово `inherit` указывает на то, что значение данного свойства должно наследоваться от родительского элемента.

С помощью свойства `text-shadow` мы можем задавать параметры тени для нашего текста. Возможные значения этого свойства описываются следующей конструкцией:

```
none | [<color> || <length> <length> <length>?,]* [<color> || <length>
<length> <length>?] | inherit
```

Что касается ключевых слов `none` и `inherit`, то у них обычная интерпретация. Значение по умолчанию `none` явно указывает, что тени у текста не будет. Значение `inherit` указывает, что значение данного свойства должно быть унаследовано от родительского элемента. Об остальных же значениях свойства следует поговорить особо.

Для указания свойств тени мы должны задать обязательно цвет тени и, как минимум, два значения, определяющих какое-то расстояние. Они задают, соответственно, горизонтальное и вертикальное смещение тени относительно текста. При этом положительные значения смещают тень влево и вниз. Отрицательное значение для горизонтального смещения переносит тень вправо от текста, а отрицательное вертикальное смещение — вверх от него. Третье числовое значение, как видно из записи, не является обязательным. Тем не менее оно достаточно часто используется. Это значение позволяет задавать степень размытости тени. То есть на тень накладывается размывающий фильтр, известный под названием *gaussian blur*, параметром которого служит третье значение.

Из определения возможных значений свойства видно, что мы можем задавать несколько вариантов теней. Они будут представлять собой наборы из указания цвета и смещения тени со значением степени размытости. Разделять эти определения следует запятой. Таким образом сформируется так называемый список определений тени. Эти определения будут порождать тени в том порядке, в котором они были записаны в правиле стилевого оформления. При этом порожденные множественные тени могут перекрывать друг друга, но они не могут перекрывать текст и выходить за пределы ячейки-контейнера, содержащей этот текст.

Рассмотрим пример, описывающий следующее правило оформления для элемента с наименованием `text`:

```
text { text-shadow: 3px 3px red, yellow -3px 3px 2px, 3px -3px }
```

В этом примере мы создаем три тени. Первая располагается справа снизу от текста со смещением по обеим осям на три пиксела. При этом цвет первой порожденной тени — красный, и на нее не воздействует размывающий фильтр. Вторая порожденная тень будет желтого цвета, и она будет смещена влево вниз от текста. При этом к ней будет применен размывающий эффект с задающим степень размытости параметром, равным двум пикселам. Третья тень будет смещена вправо и вверх относительно текста на три пиксела. Так как для нее не был явно указан цвет, она будет отображена тем же цветом, что и сам текст.

Ключевое слово `inherit` означает, что теневое оформление текста должно наследоваться от порождающего элемента.

Межсимвольный пробел в тексте задается с помощью свойства `letter-spacing`. Его возможные значения описываются с помощью следующей конструкции:

```
normal | <length> | inherit
```

Значение по умолчанию `normal` оставляет межсимвольный пробел таким, какой используется браузером для отображения шрифта текстового наполнения данного элемента.

Мы можем также прямо указать размер межсимвольного пробела, используя для этой цели стандартные единицы измерения CSS. Ключевое слово `inherit` означает, что значение для данного свойства такое же, как и у порождающего элемента.

Помимо межсимвольного пробела, можно явно указывать и интервал между словами отображаемого текста. Для этого используется свойство `word-spacing`. Множество его возможных значений описывается следующим формальным определением:

```
normal | <length> | inherit
```

Определение его значений полностью повторяет определение предыдущего свойства. Механизм их использования также одинаков, поэтому мы не станем его детально рассматривать.

С помощью свойства `text-transform` мы можем принудительно привести вид символов к одному из предустановленных типов. Формальное определение возможных значений описывается с помощью следующей конструкции:

```
capitalize | uppercase | lowercase | none | inherit
```

Значение `capitalize` заставляет браузер первую букву каждого слова принудительно превращать в заглавную. Значение `uppercase` преобразует в заглав-

ные все буквы. Значение `lowercase`, наоборот, все символы превращает в строчные. Используемое по умолчанию значение `none` оставляет текст без изменений. Если значение этого свойства — ключевое слово `inherit`, то оно будет унаследовано от порождающего элемента.

На этом мы заканчиваем рассмотрение всех свойств, описывающих шрифтовое оформление текстового содержимого элементов Web-страниц.

Стилевое оформление абзацев

Существуют и другие свойства, применимые к шрифтовому оформлению. Например, с помощью свойства `text-indent` можно задать отступ для первой строки каждого абзаца. Формальное определение возможных значений данного свойства описывается следующей конструкцией:

```
<length> | <percentage> | inherit
```

Отступ первой строки мы можем задавать как с помощью стандартных единиц измерения, так и с использованием процентов. Впрочем, отступ первой строки можно и унаследовать от родительского элемента, использовав для этого значение `inherit`. По умолчанию для данного свойства задается нулевое значение. С помощью следующей конструкции мы можем установить отступ для первой строки абзаца, равный одному сантиметру:

```
P { text-indent: 10mm }
```

Выключка строк текста регулируется с помощью свойства `text-align`. Возможные значения этого свойства описываются следующей конструкцией:

```
left | right | center | justify | <string> | inherit
```

Значение `left` заставляет браузер прижимать строки к левому краю ячейки-контейнера. Значение `right` прижимает текст к правому краю. Для центрирования строк используется значение `center`. Значение `justify` растягивает текст на всю длину строки. Помимо этого, мы можем использовать как значения этого свойства ключевые слова, устанавливающие выключку и выравнивание для столбцов таблицы, но они будут действовать только в ячейке-контейнере, содержащей ячейку таблицы. Ключевое слово `inherit` указывает, что значение этого свойства необходимо унаследовать от родительского элемента.

Тип используемого в тексте пробела может быть установлен с помощью свойства `white-space`. Его возможные значения описываются с помощью следующей синтаксической конструкции:

```
normal | pre | nowrap | inherit
```

Значение по умолчанию `normal` оставляет пробелы обычными. Значение `pre` сворачивает (collapse) пробелы и перенос текста на другую строку будет

произведен только тогда, когда в исходном коде документа тоже начнется новая строка. Значение `nowrap` преобразует пробелы в неразрывные пробелы. Пользоваться последними двумя значениями необходимо осторожно, т. к. они блокируют естественный перенос текста на другую строку при изменении размеров окна просмотра браузера. Ключевое слово `inherit` указывает, что значение данного свойства необходимо унаследовать от родительского элемента.

Таблицы в CSS

CSS level 2 содержит свою модель отображения таблиц. Но прежде чем перейти к обзору свойств стилевого оформления таблиц в CSS, необходимо рассмотреть объекты, к которым они применяются.

- Таблица (table) является специализированным элементом, который отображается в прямоугольной ячейке-контейнере и содержит специальным образом отформатированное содержимое.
- Строчная таблица (inline-table) — таблица с одной строкой.
- Строка таблицы (table-row) — совокупность ячеек, находящихся в одном горизонтальном ряду.
- Группа строк (table-row-group) — элемент, в который входит одна или несколько строк таблицы.
- Группа наименований столбцов (table-header-group) обозначает элемент, похожий на обычную строку таблицы или группу строк, но в этом элементе находятся только те строки, в которых отображаются заголовки столбцов. Другими словами, в этом элементе отображаются строки, которые находятся ниже заголовка всей таблицы и до основных данных таблицы.
- Группа подвала таблицы (table-footer-group) — также частный случай объекта "группа строк". Этот объект содержит в себе строки с итоговой информацией столбцов. Располагается после ячеек с обычными данными и до нижнего заголовка таблицы, если такой имеется.
- Колонка таблицы (table-column) — объект, содержащий совокупность ячеек, расположенных в одном вертикальном ряду.
- Группа колонок таблицы (table-column-group) объединяет одну или несколько смежных колонок.
- Ячейка таблицы (table-cell) — основной объект модели представления таблиц в CSS level 2. Содержит основной и минимальный элемент таблицы — ячейку.
- Заголовок таблицы (table-caption) — объект, содержащий заголовок таблицы. Может отображаться как до таблицы, так и после нее, или по бокам.

Теперь перейдем к обзору свойств, регулирующих отображение табличных объектов.

Свойство caption-side позволяет явно устанавливать положение заголовка таблицы относительно ее тела. Возможные значения свойства описываются следующей синтаксической конструкцией:

```
top | bottom | left | right | inherit
```

Значение по умолчанию top указывает, что название таблицы необходимо поместить сверху таблицы. Значение bottom смещает заголовок в нижнюю часть, располагая его после тела таблицы. Значения left и right заставляют браузер отображать наименование таблицы слева или справа от нее, соответственно. Ключевое слово inherit указывает, что значение данного свойства необходимо унаследовать от родительского элемента.

Данное свойство применимо лишь к объектам, которые содержат заголовок таблицы (table-caption).

Примером использования этого свойства может служить следующее правило оформления класса CAPTION:

```
.CAPTION { caption-side: bottom;
    width: auto;
    text-align: left }
```

Применение этого правила приводит к отображению заголовка таблицы после нее самой, при этом ширина ячейки-контейнера, содержащей заголовок, будет такой же, как ширина самой таблицы, что обеспечивается значением auto свойства width, а текст заголовка будет прижат к левому краю ячейки, на что указывает свойство text-align с установленным значением left.

Следует отметить, что при расположении заголовка таблицы сбоку от нее не следует использовать автоматический выбор ширины для него, т. к. браузер будет стремиться записать заголовок в одну строку, и таблица сместится, освобождая пространство для заголовка. Чтобы избежать подобной ситуации, можно использовать код, подобный приведенному ниже:

```
BODY {
 margin-left: 8mm
}
TABLE {
 margin-left: auto;
 margin-right: auto
}
.CAPTION {
 caption-side: left;
 margin-left: 8mm;
```

```
 width: 8mm;
 text-align: right;
 vertical-align: bottom
}
```

Совокупность этих правил оформления создает левое поле размером восемь миллиметров, и на нем размещается заголовок таблицы.

Свойство `table-layout` позволяет явно задавать алгоритм раскладки таблицы. Для того чтобы получить представление о его действии, необходимо рассмотреть сначала порядок создания таблиц.

Вся таблица разбивается на слои. Самым первым, самым нижним слоем является слой, в котором формируется контейнер для таблицы в целом (table). Затем формируется слой, в котором находятся группы столбцов (column groups). Сверху находится слой с одиночными столбцами (columns). На него накладывается слой с группами строк (row groups). А уже поверх него формируется слой, содержащий отдельные строки (rows). Самым последним, самым верхним слоем создается слой, содержащий отдельные ячейки.

Эта схема достаточно наглядно показана на рис. 2.5.

Формальное определение возможных значений свойства `table-layout` задается с помощью следующей конструкции:

```
auto | fixed | inherit
```

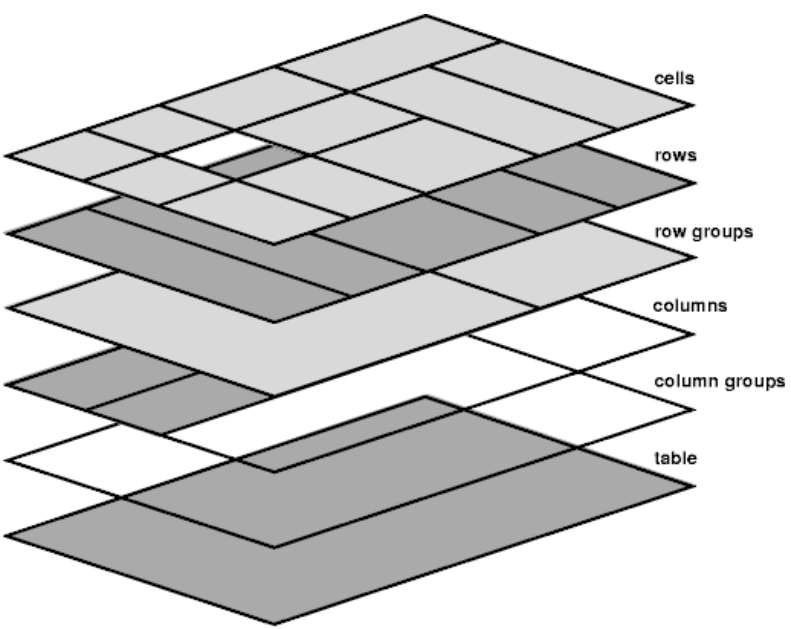


Рис. 2.5. Схема расположения слоев таблицы

Значение по умолчанию `auto` задает стандартный алгоритм размещения таблицы. Он обычно требует от анализатора браузера не больше двух проходов кода документа. Ширина таблицы задается как суммарная ширина всех столбцов.

Сначала вычисляется минимальная ширина содержимого каждой ячейки с учетом разбиения содержимого на максимальное количество строк. Если для какой-либо ячейки с помощью свойства `width` значение ширины указано явно, то это значение и принимается за минимальную ширину содержимого. Если же для свойства `width` было выбрано значение `auto`, то вычисленное значение минимальной ширины содержимого закрепляется за данной ячейкой. Также вычисляется максимальная ширина ячейки, которая получается из ширины содержимого ячейки без разбиения его на несколько строк.

Затем для каждого столбца определяется минимальная и максимальная ширина. Для этого берутся минимальная и максимальная ширина из массива всех ячеек, входящих в данный столбец. Или используется свойство `width`, заданное для этого столбца, если его значение превышает полученные величины.

Для каждой ячейки, которая растянута (span) на несколько столбцов, переопределяется максимальная и минимальная ширина столбцов, исходя из уже вычисленных значений ширины объединенной ячейки. Если общая ширина ячейки больше, чем суммарная ширина столбцов, то переопределяется максимальная ширина столбцов, включенных в объединенную ячейку.

После того как браузер вычислил ширину для всех столбцов таблицы, необходимо вычислить ширину самой таблицы. Если таблица имеет явно указанную ширину (использовано свойство `width`), то браузер должен сравнить это явно установленное значение с суммой минимальных значений ширины всех столбцов. При этом для реальной ширины таблицы выбирается максимальная из этих двух величин.

Если же для свойства `width`, примененного к таблице в целом, установлено свойство `auto`, напрямую используется сумма минимальных значений ширины всех столбцов, входящих в таблицу.

Кроме того, полная ширина таблицы зависит еще и от полей, отступов и толщины границ элементов таблицы.

Значение `fixed` свойства `table-layout` заставляет браузер применять более быстрый алгоритм. При этом ширина таблицы не зависит от ширины содержимого ячеек. При расчете используется ширина таблицы и столбцов, а также величина отступов и границ ячеек.

Сначала просматриваются все элементы, содержащие отдельные колонки таблицы. Если для них явно задано значение свойства `width`, то оно и используется для определения ширины колонки. Для тех колонок, у которых

значение этого свойства не задано явно, просматривается верхняя ячейка. Если она имеет явно заданное значение свойства `width`, то оно присваивается и самому столбцу. Если попадается растянутая (span) на несколько столбцов ячейка, то ее ширина делится на количество столбцов, которые она объединяет, и каждому столбцу присваивается соответствующее значение. Остальные столбцы поровну делят между собой оставшееся по горизонтали свободное пространство. При этом учитывается место, необходимое для отображения отступов и границ ячеек.

Для получения ширины всей таблицы сравниваются значение свойства `width`, если оно для нее задано, и суммарная ширина столбцов, включающая в себя отступы и границы ячеек. Выбирается максимальное значение.

Ключевое слово `inherit` означает, что вариант вычисления ширины таблицы наследуется от родительского элемента.

Теперь перейдем к рассмотрению свойств, регулирующих отображение границ.

Свойство `border-collapse` устанавливает модель отображения границ. Формальное определение возможных значений этого свойства определяется следующим образом:

```
collapse | separate | inherit
```

Значение по умолчанию `collapse` задает модель объединенных границ. При этом границы смежных ячеек фактически сливаются в одну общую границу, и, исходя из этой установки, браузер рассчитывает размеры таблицы или табличного объекта.

Значение `separate` не позволяет границам смежных ячеек сливаться, и каждая граница учитывается отдельно.

Ключевое слово `inherit` указывает, что значение этого свойства необходимо унаследовать от порождающего элемента.

Рассматриваемое свойство может применяться только к табличным объектам.

Свойство `border-spacing` позволяет задавать отступ между границами смежных ячеек таблицы. Естественно, это свойство принимается во внимание только в том случае, когда применяется раздельная модель границ табличных элементов (значение `separate` свойства `border-collapse`). Формальное определение его значений задается с помощью следующей конструкции:

```
<length> <length>? | inherit
```

Для задания размера пустого пространства между границами соседних ячеек мы можем задать одно или два значения. Если мы укажем только один отступ, то он будет использоваться для всех границ. Если же мы укажем два значения, то первое из них будет определять расстояние между границами по горизонтали, а второе — по вертикали. Эти значения не могут быть отрицательными.

Если значение этого свойства мы зададим ключевым словом `inherit`, то размеры отступов между границами смежных ячеек будут унаследованы от порождающего табличного элемента.

Отображением границ вокруг пустых ячеек управляет свойство `empty-cells`. Это свойство, как и предыдущее, имеет смысл использовать только в тех случаях, когда для табличного элемента установлена раздельная модель границ. Ячейки считаются пустыми, если для них свойство `visibility` имеет значение `hidden`, или если в них находится один из неотображаемых символов. Неотображаемыми символами считаются символы пробела и неразрывного пробела, табуляции, и два символа перевода каретки с кодами 10 и 13 (шестнадцатеричные коды — `0A` и `0D`, соответственно).

Формальное определение возможных значений этого свойства описывается следующим образом:

```
show | hide | inherit
```

Значение по умолчанию `show` заставляет браузер отображать границы вокруг пустых ячеек. Значение `hide` скрывает эти границы, делает их невидимыми. Однако в обоих случаях границы существуют. Ключевое слово `inherit` заставляет наследовать значение этого свойства от порождающего элемента.

Необходимо отметить, что рассматриваемое нами свойство может применяться только к элементам, которые являются ячейками таблиц (`table-cell`).

На этом мы заканчиваем рассмотрение табличной модели CSS level 2 и переходим к заключительной части обзора технологии каскадных стилевых таблиц.

Дополнительные свойства

Нам осталось рассмотреть всего несколько свойств CSS level 2, которые относятся к категории пользовательского интерфейса.

С помощью свойства `cursor` мы можем регулировать внешний вид курсора, при попадании его в ячейку-контейнер, в которой отображается содержимое элемента. Формальное определение его возможных значений задается следующей конструкцией:

```
[ [<uri>,]* [ auto | crosshair | default | pointer | move | e-resize |
ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-
resize| text | wait | help ] ] | inherit
```

Как видно из определения, мы можем указать несколько URI, ссылающихся на файлы с курсорами, или воспользоваться одним из предустановленных видов. Выбор уже установленного в системе курсора производится с помощью одного из нижеперечисленных ключевых слов.

Используемое по умолчанию значение `auto` указывает, что будет использоваться стандартный курсор, установленный для данного элемента. Конкретный его вид определяется браузером.

Значение `crosshair` при появлении курсора мыши в ячейке-контейнере придает ему вид обычного пересечения линий. В этом виде курсор больше всего похож на знак плюса.

Значение `default` заставляет браузер использовать курсор мыши, установленный в системе по умолчанию. Для Windows это обычная наклонная стрелка.

С помощью ключевого слова `pointer` мы имеем возможность при попадании курсора мыши на территорию элемента придавать ему форму указателя, как для обычных гиперссылок в HTML. Чаще всего этот курсор выглядит как рука с вытянутым пальцем, но конкретный вид курсора определяется установками операционной системы.

Значение `move` создает курсор, используемый обычно при перемещении каких-либо объектов или окон по экрану. Мы привыкли видеть его в виде перекрестия со стрелками, направленными от центра.

Ключевые слова `e-resize`, `ne-resize`, `nw-resize`, `n-resize`, `se-resize`, `sw-resize`, `s-resize` и `w-resize` создают курсоры, применяемые в тех случаях, когда изменяются какие-либо размеры окна. Для Windows это обычно самые различные двунаправленные стрелки. Так, например, значение `se-resize` создает курсор, возникающий при перетаскивании нижнего левого угла окна.

Значение `text` заставляет браузер использовать текстовый курсор мыши. Обычно это вертикальная черта с двумя засечками.

Для отображения курсора мыши, обозначающего процесс ожидания, т. е. период времени, когда система занята и не может оперативно реагировать на действия пользователя, применяется значение `wait`. Чаще всего подобный курсор выглядит как часы, обычные или песочные.

Ключевое слово `help`, применяемое как значение данного свойства, заставляет браузер отображать курсор мыши, используемый при запросе оперативной справки. Как правило, подобный курсор выглядит как стрелка, совмещенная с вопросительным знаком.

Ключевое слово `inherit` означает, что значение этого свойства будет таким же, как у элемента-родителя.

Примером использования этого свойства может служить следующее правило оформления:

```
P { cursor : url("mything.cur"), url("second.csr"), text; }
```

Для элемента с наименованием `P` мы задали три возможных типа курсора мыши. При этом доступ к ним осуществляется по очереди. Если не удается

загрузить курсор из файла mything.cur, то браузер пытается отыскать файл second.csr, а если и его нет, использует обычный текстовый курсор.

Иногда может возникнуть необходимость помимо границ или вместе с ними отобразить еще и обрамляющие линии элемента (outlines). Обрамляющие линии отличаются от границ тем, что не занимают лишнего пространства и могут отличаться от стандартной прямоугольной формы. Регулируется внешний вид обрамляющих линий с помощью нескольких свойств.

Свойство `outline-width` предназначено для регулировки толщины обрамляющих линий. Формальное определение его возможных значений задается следующим образом:

```
<border-width> | inherit
```

По формату это свойство идентично уже рассматривавшемуся нами свойству `border-width`.

Стиль обрамляющих линий задается с помощью значения свойства `outline-style`. Возможные значения этого свойства описываются следующей конструкцией:

```
<border-style> | inherit
```

Здесь значениями могут быть все ключевые слова, применяемые для задания стиля границы, за исключением ключевого слова `hidden`, т. к. нет никакого смысла создавать невидимые обрамляющие линии.

С помощью свойства `outline-color` у нас есть возможность задавать цвет обрамляющих линий. Формальное определение возможных значений этого свойства выглядит следующим образом:

```
<color> | invert | inherit
```

Из определения видно, что помимо обычных способов задания цвета мы можем использовать как значение свойства ключевое слово `invert`. Оно заставляет браузер отображать обрамляющие линии негативным цветом, по отношению к фону, на котором они прорисовываются.

Агрегатное свойство `outline` позволяет собрать все три только что рассмотренных нами свойства в единое целое. Формальное определение значений этого свойства описывается следующей конструкцией:

```
[ <'outline-color'> || <'outline-style'> || <'outline-width'> ] | inherit
```

Необходимо отметить, что все обрамляющие линии отображаются всегда "поверх" ячейки-контейнера, и отображение их обладает более высоким приоритетом. Другими словами, все границы и иные элементы оформления находятся на один слой ниже.

На этом мы заканчиваем обзор технологии CSS level 2 и переходим к технологии динамического управления содержимым Web-страницы.

ГЛАВА 3

Динамический HTML

Дополнительные возможности

Изучив первые две главы книги, мы научились разрабатывать HTML-документы, предназначенные для отображения в браузерах, и управлять стилевым оформлением содержимого этих документов. Теперь следует сделать еще один шаг вперед — изучить технологию динамического HTML, которую часто обозначают как *DHTML* (Dynamic HTML). Но прежде чем мы узнаем, в чем ее суть, следует сделать небольшое отступление.

В начале книги мы упоминали о языках сценариев, или, как их еще называют, скриптовых языках. Среди них есть такие языки, которые распознаются и обрабатываются непосредственно браузером. Это VBScript и JavaScript. Язык VBScript ведет свою родословную от языка Visual Basic, развиваемого и поддерживаемого корпорацией Microsoft. Язык JavaScript унаследовал свои конструкции от языка Java, который усиленно продвигает корпорация Sun. Как мы уже говорили, эти скриптовые языки распознаются и обрабатываются непосредственно браузерами. Другими словами, в тело HTML-документа мы внедряем код программы, написанной на одном из этих языков, а браузер выполняет все действия, предписанные этой программой.

Такая простота использования и обусловила столь высокую популярность скриптовых языков. Однако создатели программ-браузеров конкурируют друг с другом. И получилось так, что компания Netscape, изготовитель достаточно популярного браузера Netscape Navigator, не включила в состав своего браузера блок обработки языка VBScript. Это был серьезный удар по конкуренту, в результате подавляющее число разработчиков использует для своих нужд язык JavaScript. Несколько позже компания Microsoft нанесла "ответный удар", создав новую и очень интересную технологию, опирающуюся на язык VBScript, но эти события и технологии уже выходят за рамки нашей книги.

Помимо языков сценариев производители браузеров поддерживают так называемые "объектные модели", которые позволяют программам-скриптам управлять некоторыми свойствами браузеров. Причем перечень управляемых объектов достаточно обширен и позволяет разработчикам виртуозно оперировать огромным количеством свойств браузеров и выполнять некоторые действия, не осуществимые с помощью HTML.

Сплав HTML, CSS, скриптовых языков, объектных моделей браузеров и получил наименование DHTML. Эта не слишком сложная технология предоставила разработчикам поистине огромные возможности. Если попытаться дать точное определение, то DHTML — это технология динамического изменения содержимого Web-страниц с помощью скриптовых языков.

Как видно из приведенного определения, изменения содержимого Web-страниц мы можем производить только с помощью скриптовых языков. Следовательно, нам необходимо узнать, как ими пользоваться. Для того чтобы получить возможность применять самые заманчивые средства создания Web-страниц, все-таки придется немного попрограммировать. Это не так уж трудно! Для работы мы выберем JavaScript, т. к. сценарии, написанные на этом языке, будут гарантированно выполнены любым браузером. Прежде чем переходить к технологии DHTML, следует рассмотреть язык JavaScript. Знакомству с ним будут посвящены несколько следующих разделов этой главы.

Структура JavaScript

У любого языка программирования есть свои правила использования, свои типы переменных и свои ключевые слова. Это то, что отличает один язык от остальных его собратьев. Несмотря на указанные отличия, структура всех современных языков программирования практически одинакова.

Как, собственно, выполняется программа-сценарий, внедренная в Web-страницу? В этом сценарии мы задаем различные реакции Web-страницы на какие-либо действия пользователя, например: на нажатия кнопок, перемещение курсора мыши, работу с клавиатурой и т. д. Обрабатываемых событий достаточно много.

Как только произошло одно из указанных нами событий, браузер должен выполнить некоторую последовательность действий. Эта последовательность описывается с помощью блока ключевых слов и выражений. Подобные блоки кода оформляются как функции. Функции — это именованные блоки кода, которые проделывают определенные действия и возвращают некоторое значение. Например, вычисляют синус числа, являющегося вертикальной координатой верхнего левого угла графического изображения.

В языке JavaScript заранее определены некоторые простейшие управляющие конструкции. Предопределенные конструкции языка называются *операто-*

рами. Кроме того, в язык введены наиболее распространенные и часто используемые функции.

Для того чтобы нам оперировать какими-либо числами, строками и прочими данными используются переменные. *Переменная* — это просто контейнер для хранения какого-либо значения. При этом значение переменной мы в любой момент можем изменить. В JavaScript используется несколько различных типов переменных. Мы можем создавать переменные, содержащие числовые значения, строки и логические выражения. При этом явно обозначать эти типы не нужно. Интерпретатор программы сам правильно обработает их. Достаточно лишь присвоить переменной какое-либо значение.

Объявлять переменные мы можем в любом месте программы, но никто еще не отменял понятие "хорошего стиля программирования". Хороший стиль программирования предполагает объявление всех переменных, используемых в процедуре или функции, в самом начале этой процедуры или функции. При этом следует указывать ключевое слово `var`. Строго говоря, оно не обязательно, но если у нас появятся одинаковые имена переменных в различных процедурах, используемых совместно, то может возникнуть конфликт совпадения имен. Применение ключевого слова `var` позволяет браузеру правильно обрабатывать переменные в таких случаях.

Рассмотрим маленький пример.

```
var x=1;
var s="Строка";
var t=True;
```

В каждой строке мы объявляем одну переменную. Так как мы сразу присваиваем им некоторые значения, то переменные обретают соответствующий тип. Переменная `x` содержит число, переменная `s` — строку, а переменная `t` предназначена для хранения логических значений.

Числовые значения мы можем записывать в программе в десятичном виде, как целые, так и дробные. Кроме того, мы можем записывать числа в восьмеричной и шестнадцатеричной системе исчисления. Числа в восьмеричной записи должны начинаться с нуля, а в шестнадцатеричной — с сочетания `0x`. Если перечисленных способов нам не хватает, то можно воспользоваться экспоненциальным способом записи числа.

Если необходимо присвоить переменной строковое значение, то это значение следует заключить в кавычки или апострофы (одинарные кавычки). Строка может состоять из одного символа.

Переменные логического типа обычно используются в операторах выбора. Логических значений существует всего два: `True` — истина, и `False` — ложь.

Если мы объявляем переменную, но не хотим сразу присваивать ей какое-либо конкретное значение, можно в качестве пустого значения воспользоваться ключевым словом `null`. Таким образом, объявление переменной без

присваивания значения и типа выполняется с помощью следующего фрагмента кода:

```
var x=null;
```

Пользуясь языком JavaScript, всегда следует помнить, что он, в отличие от HTML, чувствителен к регистру символов. Интерпретатор языка, встроенный в браузер, не сможет выполнить всю программу, если в написании ключевого слова допущена хотя бы одна ошибка.

Кроме того, обратите внимание на то, что каждая инструкция в тексте программы заканчивается символом точки с запятой. С помощью этого символа мы отделяем операторы друг от друга, что позволяет браузеру правильно обрабатывать программу.

Язык JavaScript является языком со слабой типизацией. Это означает, что любая переменная в процессе работы программы может изменить свой тип. Так, например, с помощью следующего фрагмента кода мы превращаем переменную строкового типа в число:

```
var x="100";
x=x+1;
```

В первой строке этого фрагмента мы объявили переменную с наименованием `x`, которой присвоили текстовое значение, состоящее из трех символов. Во второй строке мы к этому строковому значению прибавили число. В результате этого действия в переменную `x` помещается числовое значение, равное ста одному.

Необходимо отметить, что в наименованиях переменных мы можем использовать только буквы латинского алфавита, цифры и символы подчеркивания. Начинаться наименование должно обязательно с буквы. Пробелы в наименованиях переменных использовать нельзя.

Естественно, JavaScript, являясь достаточно развитым языком программирования, позволяет применять в программах не только простые переменные, но и массивы. *Массив* — это тип данных, состоящий из фиксированного числа элементов одного и того же типа. Таким образом, переменная типа массив хранит не одно значение, а заданное количество значений, каждое из которых имеет свой номер, называемый *индексом*.

В языке JavaScript мы можем не только объявлять массив, но и динамически менять его размеры во время выполнения программы. Следовательно, для массива должен выделяться некий фрагмент адресного пространства памяти. Подобное выделение всегда производится с помощью ключевого слова `new`. Ключевое слово `Array` указывает интерпретатору программы, что область памяти выделяется именно под массив. Типичное объявление массива выглядит следующим образом:

```
my_array= new Array();
my_array(9)=0;
```

В первой строке мы объявляем массив с наименованием `my_array`. Во второй строке инициализируем десятый элемент только что объявленного массива, присваивая ему нулевое значение. Интерпретатор сочтет этот элемент последним, и наш массив первоначально будет содержать десять элементов, т. к. нумерация элементов массивов в JavaScript начинается с нуля. Для того чтобы обратиться к какому-либо элементу, после наименования массива в скобках нужно указать порядковый номер этого элемента, как это было показано в нашем примере.

Если это необходимо, можно указать размер массива при его объявлении. Для этого надо после ключевого слова `Array` в скобках указать предполагаемое количество элементов массива. В этом случае объявление нашего массива из десяти элементов будет выглядеть следующим образом:

```
my_array= new Array(9);
```

Во всех наших примерах объявления переменных мы использовали знак равенства, который на самом деле является одним из основных операторов языка. Это оператор присваивания. С его помощью мы можем переменным присваивать некие значения. Всего в языке JavaScript есть шесть операторов присваивания. Рассмотрим их.

- Оператор `=` выполняет стандартное присваивание. Пример его использования мы уже видели.
- Оператор `+=` осуществляет присваивание со сложением. Таким образом, выражение `x+=y` эквивалентно `x=x+y`.
- Оператор `-=` реализует присваивание с вычитанием. Таким образом, выражение `x-=y` эквивалентно `x=x-y`.
- Оператор `*=` позволяет производить присваивание с умножением. Выражение `x*=y` равносильно выражению `x=x*y`.
- Оператор `/=` выполняет присваивание с делением. Выражение `x/=y` равносильно выражению `x=x/y`.
- Оператор `%=` позволяет осуществлять присваивание остатка целочисленного деления. Таким образом, выражение `x%=y` эквивалентно `x=x%y`.

Большинство из рассмотренных операторов присваивания — это комбинация обычного оператора присваивания с каким-либо знаком арифметической операции. Язык JavaScript обладает достаточно обширным набором базовых операций, выполняющих стандартные действия над переменными, и нам никак нельзя обойтись без знакомства с ними.

- Операция `+` осуществляет сложение. 2+3 равно 5.
- Операция `-` реализует вычитание. 3−2 равно 1.
- Операция `*` производит умножение. 2*3 равно 6.
- Операция `/` выполняет деление. 6/3 равно 2.

- Операция `%` выделяет остаток от целочисленного деления. 9%4 равно 1.
- Унарная операция `++` увеличивает значение на единицу. 3++ равно 4.
- Унарная операция `--` уменьшает значение на единицу. 3-- равно 2.
- Операция `&` эквивалентна операции `AND`. Обе операции выполняют поразрядно логическую операцию "И". При этом числа, соединенные знаком этих операций, автоматически переводятся в двоичную систему. 2&5 равно 0.
- Операция `|` эквивалентна операции `OR`. Обе операции выполняют поразрядно логическую операцию "ИЛИ". 2|5 равно 7.
- Операция `^` эквивалентна операции `XOR`. Эти операции выполняют поразрядное сложение по модулю 2. 3^2 равно 1.
- Операция `<<` осуществляет поразрядный сдвиг двоичного представления числа влево на один разряд. 2<< равно 4.
- Операция `>>` выполняет поразрядный сдвиг двоичного представления числа вправо на один разряд. 2>> равно 1.
- Операция `&&` реализует операцию "И" для логических значений. `True && False` равно `False`.
- Операция `||` осуществляет операцию "ИЛИ" для логических значений. `True || False` равно `True`.
- Операция `!` выполняет операцию отрицания для логических значений. `!False` равно `True`.

Все перечисленные операции мы можем использовать в своих программах на языке JavaScript. Список не маленький, но чаще всего для нужд DHTML мы будем обходиться арифметическими и логическими операциями.

Любая программа, и наши скрипты — не исключение, практически никогда не действует без управляющих конструкций, которые позволяют реализовать алгоритм. К таким управляющим конструкциям относятся операторы циклов, условные операторы и операторы-переключатели. Все они достаточно просты, и применение их не вызовет каких-либо затруднений.

Условный оператор позволяет выполнять некие действия в том случае, если выполняется заданное разработчиком условие. Приведем маленький пример.

```
if (x==2) x*=7;
```

Данный фрагмент кода проверяет значение переменной `x`. Если это значение равно двум, то оно увеличивается в семь раз. Но умножение будет производиться тогда и только тогда, когда значение переменной равно двум. Проверка логического условия, как мы видим, задается в программе с помощью ключевого слова `if`.

Интуитивно конструкция условного оператора ясна, но его использование может быть и не таким простым, как в приведенном только что приме-

ре. В том случае, если нам надо проверить несколько логических условий сразу, каждое условие заключается в скобки, и к ним применяются логические операции, которые мы рассмотрели несколько ранее. Так, если нам необходимо проверить выполнение сразу двух условий, то наш пример будет выглядеть следующим образом:

```
if (x==2) && (y<0) x*=7;
```

В этом случае умножение производится только тогда, когда переменная `x` равна двум, а значение переменной `y` меньше нуля.

Условия проверки могут задаваться либо с помощью переменных логического типа, либо с помощью операций сравнения, список которых приведен в табл. 3.1.

Таблица 3.1

Операция	Значение
==	Точное равенство
!=	Неравенство
>	Больше
>=	Больше или равно (не меньше)
<	Меньше
<=	Меньше или равно (не больше)

Эти операции могут использоваться для сравнения как числовых значений, так и строковых. Если необходимо определить, какая из двух строк больше или меньше, то сравнение производится посимвольно, начиная с начала строки. Чем символ ближе к концу алфавита, тем он "больше".

При выполнении условия программа интерпретатор выполняет только одно действие. А что делать, если надо выполнить не один оператор, а несколько? В этом случае необходимо все эти операторы объединить в один блок. Для этого достаточно заключить все операторы в фигурные скобки, как это показано в следующем примере:

```
if (x==2) {x*=7; y=1};
```

Все рассмотренные нами примеры использования условного операторы построены на допущении, что лишь при истинности некоторого условия необходимо выполнить некий блок кода. Но может возникнуть ситуация, когда при истинности условия мы должны выполнить один блок кода, а если условие ложно — другой. В этом случае наш условный оператор приобретет следующий вид:

```
if (x==2) {x*=7; y=1}
else {y=x};
```

При этом каждый блок действия может, при необходимости, включать в себя и другие условные операторы. Подобным образом реализуются алгоритмы обработки сразу нескольких условий.

Если есть список условий, каждому из которых мы должны сопоставить некоторый программный блок, следует воспользоваться оператором-переключателем. В нем выполнения разных блоков программы увязываются со значением какой-либо переменной, и нельзя использовать логические условия и их комбинации. Вот как выглядит типичный пример оператора-переключателя:

```
switch (x) {
case 1: {Блок, выполняемый, если x=1}
case 2: {Блок, выполняемый, если x=2}
default: {Блок, выполняемый, во всех остальных случаях}
}
```

Алгоритм работы данного оператора ясен, но есть один подводный камень. Если управляющая переменная, значения которой проверяет оператор `switch`, равна значению, заданному после слова `case`, то будет выполнен блок кода, расположенный в текущей строке, и все следующие строки кода, до конца оператора `switch`. Таким образом, в нашем примере, если значение переменной `x` равно единице, то сначала будет выполнен блок кода для этого значения, затем блок для значения переменной, равного двум, а потом еще и блок, используемый для всех остальных значений. Для того чтобы избежать этого, следует в конец каждого блока кода, расположенного после ключевого слова `case`, вставлять оператор `break`. Оператор `break` принудительно прекращает выполнение любого оператора, в котором он применяется. Теперь пример использования оператора-переключателя выглядит приблизительно следующим образом:

```
switch (x) {
case 1: {y=x*2+z; break}
case 2: {y=x*3+z; break}
default: {y=0; z=0}
}
```

Как видно, в последнем блоке, который выполняется для значений переменной `x`, не перечисленных в операторе явно, мы не использовали оператор `break`, т. к. этот блок кода является последним, и после его выполнения управление все равно выйдет за пределы оператора-переключателя.

При написании скриптов нам часто придется использовать циклические конструкции, в которых один и тот же блок кода выполняется определенное количество раз, или до тех пор, пока некоторое логическое условие истинно. Обычно с помощью подобных циклов производится обработка массивов данных. О массивах мы узнаем немного позже, а сейчас рассмотрим применение циклов в качестве управляющих конструкций.

В JavaScript предусмотрено три вида циклов. Стандартный цикл, в котором некоторый блок кода выполняется определенное количество раз, реализуется с помощью оператора `for`. Наиболее типичный пример его использования выглядит следующим образом:

```
sum=0;
for (i=0; i<10; i++)
if my_array(i)>0 then sum=sum+my_array(i);
```

В этом примере мы с помощью цикла производим суммирование всех положительных элементов массива с наименованием `my_array`, содержащего десять элементов. Как видно, для оператора `for` мы указываем некоторую дополнительную информацию в скобках. В скобках мы обязательно должны использовать три оператора. Все они связаны с переменной, которая является счетчиком повторений цикла. В нашем случае мы использовали переменную с наименованием `i`. Первый оператор предназначен для присваивания переменной-счетчику стартового значения. Второй оператор является логическим условием. Цикл будет повторяться до тех пор, пока это условие истинно. Третий оператор обычно устанавливает операцию, которая выполняется каждый раз при завершении очередного прохода тела цикла. Чаще всего это операция изменения значения переменной-счетчика. Поскольку в нашем случае отсчет ведется от нуля до девяти включительно, то мы используем операцию увеличения, т. е. после каждого прохода цикла значение нашей переменной-счетчика увеличивается на единицу. При этом проходом цикла называют однократное выполнение тела цикла.

Тело цикла в нашем примере содержит один оператор. Если необходимо использовать не один оператор, а несколько, то их придется объединить в блок с помощью уже знакомых нам фигурных скобок.

Следует отметить, что для экстренного выхода из цикла до его нормального завершения можно использовать оператор `break`, который, как мы помним, прерывает выполнение текущего оператора и передает управление оператору, следующему за ним. Если нам необходимо при истинности или ложности каких-либо условий пропустить один проход цикла, а затем продолжить его выполнение, стоит воспользоваться оператором `continue`, который прекращает выполнение текущего прохода цикла, но не завершает выполнение оператора цикла. Приведем простейший пример.

```
for (i=0;i<10;i++)
{ if i=5 then continue;
my_array[i]=2;
}
```

С помощью данного цикла мы присваиваем всем элементам массива значение, равное двум. Точнее, всем, кроме элемента с индексом "пять", т. е. шестого по порядку. Для этого элемента операция присваивания не выполняется.

Как мы уже говорили ранее, в JavaScript существует три вида циклов. Один из них мы уже рассмотрели. Теперь мы узнаем, как действует цикл с предусловием. Если мы попробуем с помощью этой управляющей конструкции описать вычисление суммы положительных элементов массива, то получится следующий фрагмент кода:

```
sum=0;
i=0;
while (i<10)
{ if my_array(i)>0 then sum=sum+my_array(i);
i++;}
```

Как видно, данный вариант цикла реализуется с помощью оператора `while`, в котором указывается логическое условие. До тех пор, пока это условие истинно, тело цикла будет повторяться. Видно, что стартовое значение переменной-счетчика мы задаем самостоятельно, и позаботиться о ее дальнейшем изменении тоже придется нам самим.

Последний вариант циклической управляющей конструкции описывается также с помощью оператора `while`, но на этот раз с ним вместе должен использоваться оператор `do`. Наш цикл приобретет следующий вид:

```
sum=0;
i=0;
do {
if my_array(i)>0 then sum=sum+my_array(i);
i++;}
while (i<10);
```

Как и прежде, тело цикла будет выполняться раз за разом до тех пор, пока логическое условие, указанное в операторе `while`, остается истинным. В чем же тогда разница между циклами с предусловием и с постусловием? В первом из этих циклов логическое условие проверяется перед прохождением тела цикла, а во втором — после. Таким образом, даже если логическое условие еще до старта ложно, цикл с постусловием все равно будет выполнен как минимум один раз.

На этом список управляющих логических конструкций заканчивается. Но нам их будет вполне достаточно для того, чтобы реализовать любые замыслы.

Для создания программ-скриптов необходимо уметь разрабатывать свои функции. Мы уже говорили, что функции — это именованные, логически законченные блоки кода, возвращающие некое значение.

Вся программа-скрипт составлена из различных функций. Каждая функция выполняется интерпретатором в тот момент, когда наступает некоторое активизирующее событие. О событиях мы поговорим позже, а сейчас посмот-

рим, как объявляются функции в языке JavaScript. Простейший пример объявления функции выглядит следующим образом:

```
function my_first_function (name)
{alert("Hello, world! My name is "+name);
return("Your name is "+name);}
```

Итак, из приведенного примера видно, что функция объявляется с помощью ключевого слова `function`, затем указывается ее наименование, и в скобках перечисляются переменные, которые передаются этой функции для работы. После этого мы записываем код функции, обязательно заключая его в фигурные скобки. Наша функция с помощью оператора `alert` выводит окно сообщения, содержащее указанный текст.

Мы уже говорили, что любая функция должна возвращать какое-либо значение, даже если мы в нем не нуждаемся. Возврат значения осуществляется с помощью ключевого слова `return`, которое одновременно сообщает об окончании последовательности действий, заданных в нашей функции. В языке JavaScript можно обойтись и без этого ключевого слова, если функция не должна возвращать значение, которое мы будем использовать в работе скрипта позже.

Завершая знакомство с языком JavaScript, следует рассмотреть возможность работы с переменными составных типов и объектами. Что такое объект понять достаточно легко, но вот точно и просто объяснить, что это, собственно, такое, гораздо сложнее. Можно сказать, что объект — это некоторая сущность со своим набором свойств и списком действий, которые она может выполнять. Эти действия называются методами, их набор и механизм действия специфичны для каждого объекта. Также для каждого объекта определен набор возможных событий.

В языке JavaScript нельзя создавать собственные объекты, но мы можем пользоваться уже готовыми. Все доступные нам объекты описывают Web-страницу с внедренной в нее программой на языке JavaScript или элементы программы-браузера. Основным объектом является объект с наименованием `window`, который позволяет работать с окном просмотра Web-страницы в браузере. В этот объект в качестве вложенных частей входят другие объекты, такие как `screen`, связывающий нас с экраном монитора пользователя, или `document`, открывающий доступ к содержимому самой Web-страницы. Полностью эти объекты мы будем рассматривать несколько позже.

Повторю еще раз, мы не можем самостоятельно создавать свои объекты, но у нас есть возможность объявлять переменные составного типа. Например, мы можем создать свой тип для комплексных чисел, которые содержат мнимую и действительную части. Такие комплексные числа описываются с помощью пары обыкновенных чисел. Создать переменную подобного комплексного типа достаточно просто. Для этого необходимо использовать спе-

циализированную функцию-конструктор. Выглядит этот процесс следующим образом:

```
function complex(x, y)
{this.x=x;
this.y=y;
return this;
}
my_complex = new complex (5,2);
```

В этом примере мы видим, как создается функция-конструктор для комплексного типа чисел. После кода функции следует объявление и инициализация переменной с помощью оператора `new`, который используется для выделения памяти под создаваемые экземпляры объектов и составных переменных. Доступ к отдельным компонентам составной переменной осуществляется указанием составного имени:

```
<имя переменной>.<имя компонента>
```

В приведенном примере к компоненту `x` можно обратиться по имени `my_complex.x`.

И на этом мы заканчиваем рассмотрение структуры языка JavaScript. В этой главе мы постоянно будем пользоваться им для создания эффектов DHTML, поэтому все неясности будут устранены во время работы.

Объектная модель

В предыдущем разделе мы говорили, что язык JavaScript позволяет работать с объектами. Все доступные объекты языка образуют достаточно разветвленную иерархию. Они относятся именно к окну просмотра HTML-документа в браузере.

Каждый объект является совокупностью свойств, методов и событий. Методы, как мы уже говорили, — это некие действия, которые можно произвести с данным объектом, а события — это методы, описывающие реакцию объекта на различные ситуации, возникающие при работе с ним и адекватно отображающие изменения состояния. Для каждого объекта набор свойств, методов и событий уникален, поэтому придется все объекты рассматривать очень подробно.

Иерархия объектов для программ-скриптов является единственной возможностью динамически управлять отображением Web-страниц. Фактически, это интерфейс доступа к внутренним механизмам браузера. Для того чтобы наши скрипты, реализующие возможности DHTML, могли работать в любом браузере, необходимо, чтобы объектные модели для всех браузеров совпадали. К сожалению, полного соответствия нет, и каждый браузер пользу-

ется своей объектной моделью, но различаются эти модели только незначительными дополнениями. По сути дела, производители браузеров лишь попытались добавить нечто свое к общепринятой модели.

В этой книге мы не будем рассматривать эти дополнения и изменения, и на то есть достаточно веская причина. Еще несколько лет назад довольно часто можно было встретить в Сети Web-страницы с надписями: "Эта Web-страница лучше всего отображается в браузере...", и указывалось наименование того браузера, к которому благоволил разработчик Web-страницы. К счастью, в последнее время в среде разработчиков распространилось мнение о недопустимости подобных разработок, поэтому мы будем стараться создавать Web-страницы, которые бы одинаково отображались во всех основных браузерах. Исходя из этих же соображений, рассмотрим только стандартную объектную модель DHTML, которая одинаково функционирует в каждом браузере.

Прежде чем мы перейдем к непосредственному рассмотрению объектов, следует сделать уточнение. Некоторые объекты в Web-страницах могут существовать только в единственном экземпляре, как, скажем, объект `window`, олицетворяющий окно просмотра, а некоторых объектов может быть много. Так, например, графические изображения, внедренные в состав содержимого Web-страницы, создают множество однотипных объектов. Такие множества называются *коллекциями*, и больше всего они похожи на массивы однотипных объектов. Доступ к какому-либо конкретному элементу коллекции производится с помощью указания наименования коллекции и порядкового номера интересующего нас элемента.

Теперь настало время перейти к непосредственному рассмотрению всего множества объектов, которым мы можем пользоваться для придания некой динамики и интерактивности статичному содержимому наших Web-страниц.

Основой всего является объект `window`, который позволяет нам управлять окном просмотра браузера. Этот объект обладает набором свойств, методов и событий, который мы и рассмотрим. Следует отметить, что в этом разделе главы мы устроим лишь "обзорную экскурсию" по объектной модели, а примеры использования объектов мы рассмотрим немного позже. Вернемся к объекту `window`. Пользователь может открывать несколько копий программы-браузера, а значит, у нас будет несколько объектов `window`. Если содержимое этих окон — Web-страницы, ссылки на которые находятся на исходной страничке, то эти объекты `window` связываются друг с другом с помощью некоторых свойств. Также следует упомянуть, что по умолчанию все объекты, которые мы будем рассматривать, встроены в объект `window`. Но перейдем к рассмотрению обычных свойств этого объекта.

- Свойство `client` содержит информацию о браузере, используемом для просмотра данной Web-страницы. Свойство имеет статус "только для чтения", т. е. его значение не может быть изменено в процессе действия программы-скрипта.

- Свойство `status` содержит текстовую строку, которая отображается в статусной строке, в нижней части окна программы-браузера.
- Свойство `defaultStatus` содержит текст, отображаемый в статусной строке по умолчанию.
- Свойство `opener` ссылается на окно просмотра, которое было открыто из текущего окна, т. е. в данном свойстве находится ссылка на дочернее окно.
- Свойство `parent` является ссылкой на родительское окно, из которого открыли текущее.
- Свойство `name` хранит наименование текущего окна.
- Свойство `self` содержит ссылку на данное окно, т. е. с помощью этого свойства объект ссылается сам на себя.
- Свойство `top` содержит ссылку на самое первое окно в иерархии связанных друг с другом окон.
- Свойство `closed` имеет логический тип и сообщает, закрыто или открыто данное окно.
- Свойство `dialogArguments` — параметр, или массив параметров, передаваемых диалоговому окну. Диалоговые окна отображаются поверх основного окна просмотра и, помимо текстового сообщения, содержат кнопки **OK** и **Отмена**, при помощи которых пользователь подтверждает или отменяет предлагаемое программой-скриптом действие.
- Свойство `dialogHeight` — высота диалогового окна в пикселах.
- Свойство `dialogWidth` — ширина диалогового окна в пикселах.
- Свойство `dialogTop` — вертикальная координата верхнего левого угла диалогового окна в пикселах.
- Свойство `dialogLeft` — горизонтальная координата верхнего левого угла диалогового окна в пикселах.
- Свойство `returnValue` — содержит значение, возвращаемое в программу диалоговым окном.

Помимо свойств есть и список методов. Они реализованы в виде функций.

- Метод `alert(messagestring)` создает окно сообщения. Текст сообщения передается в параметре.
- Метод `confirm(messagestring)` отображает подтверждающее окно, текст которого передан в функцию в качестве параметра. Окно отличается от предыдущего наличием дополнительной кнопки **Отмена**.
- Метод `showModalDialog(url, header, attributes)` создает модальное диалоговое окно. В параметрах передается URL отображаемого файла, заголовок окна и атрибуты создаваемого окна.

- Метод `open(url, header, attributes)` открывает новое окно браузера. В параметрах передается URL HTML-документа, который будет отображаться в этом окне, заголовок нового окна просмотра и его параметры.
- Метод `close()` закрывает данное окно.
- Метод `setTimeout(expression, time, language)` создает таймер, который вычислит выражение или выполнит действие спустя определенное время. Выражение или код действия передается в параметре `expression`, язык, на котором оно записано, — в параметре `language`, а время, по истечении которого произойдет вычисление выражения или выполнение инструкций, — в параметре `time`. Функция возвращает идентификатор созданного таймера. Одновременно можно создавать несколько таймеров, и управление ими производится с помощью подобных идентификаторов.
- Метод `setInterval(expression, time)` по своему действию очень похож на предыдущий метод `settimeout`, но в отличие от него производит заданное действие не единожды, а многократно, каждый раз перед выполнением делая паузу, длительность которой указана в параметре `time`.
- Метод `clearTimeout(timerid)` ликвидирует созданный таймер. В качестве параметра передается идентификатор ликвидируемого таймера, который был получен как возвращаемое значение функции `settimeout`.
- Метод `clearInterval(timerid)` уничтожает повторяющийся таймер, созданный с помощью метода `setinterval`. В качестве параметра передается идентификатор ликвидируемого таймера.
- Метод `focus()` устанавливает фокус ввода на данное окно. Одновременно инициализируется событие `onfocus`.
- Метод `blur()` принудительно лишает окно фокуса ввода. Инициализируется событие `onblur`.
- Метод `prompt(message, default_value)` создает окно с полем ввода и текстом подсказки. Текст подсказки передается в параметре `message`, а значение, находящееся в поле ввода по умолчанию, — в параметре `default_value`.
- Метод `execScript(script, language)` выполняет блок кода, который передается в параметре `script`, а в параметре `language` — язык, на котором он написан.
- Метод `scroll(mode)` позволяет принудительно показывать и скрывать полосы прокрутки в окне просмотра браузера. Для отображения полос необходимо передать в параметре значение `yes`, для отказа — `no`.

Следует более тщательно рассмотреть метод `open`, с помощью которого создается новое окно просмотра браузера. Параметры `url` и `header` достаточно понятны, а параметр `attributes` мы не упомянули. Он представляет собой текстовую строку, в которой через запятую записываются пары

"свойство=значение". Подобным способом указывается внешний вид создаваемого окна. Перечислим атрибуты создаваемого окна просмотра.

- `fullscreen` — открыть окно в полноэкранном режиме или нет. Для использования полноэкранного режима необходимо использовать значение `1` или `yes`, в противном случае — `0` или `no`. Эти значения используются и во всех других параметрах, которые указывают на наличие или отсутствие некоего свойства у вновь создаваемого окна просмотра.
- `status` — регулирует наличие у окна строки состояния. Значения параметра полностью совпадают со значениями предыдущего параметра.
- `menubar` — указывает на использование строки меню в новом окне просмотра.
- `scrollbars` — устанавливает, будут ли присутствовать полосы прокрутки у окна просмотра.
- `resizable` — определяет, будет ли пользователь иметь возможность изменять размеры нового окна просмотра.
- `toolbar` — регулирует наличие инструментальной панели.
- `width` — устанавливает ширину окна просмотра в пикселах.
- `height` — задает высоту нового окна просмотра в пикселах.
- `top` — указывает вертикальную координату верхнего левого угла окна просмотра.
- `left` — задает горизонтальную координату верхнего левого угла окна просмотра.

Приведем пример использования метода `open`. Следующая конструкция предназначена для создания нового окна просмотра браузера, в котором будет отображаться основная страница поискового ресурса `www.yahoo.com`, в заголовке будет находиться строка "Поисковая система", а само окно будет иметь панель быстрых кнопок и строку меню, верхний левый угол его будет находиться в точке с координатами (100,100), а размеры — 400 пикселов по вертикали и горизонтали. Выглядеть этот вызов метода будет следующим образом:

```
open("http://www.yahoo.com", "Поисковая система", "menubar=yes,
↳toolbar=yes, top=100, left=100, height=400, width=400");
```

У метода `showModalDialog` в качестве третьего параметра тоже передается строка атрибутов создаваемого диалогового окна. Но они не совпадают с набором свойств окна просмотра браузера, поэтому их придется перечислять отдельно.

- `dialogTop` — вертикальная координата верхнего левого угла диалогового окна.

- `dialogLeft` — горизонтальная координата верхнего левого угла диалогового окна.
- `dialogHeight` — высота диалогового окна в пикселах.
- `dialogWidth` — ширина диалогового окна в пикселах.
- `center` — указывает на то, будет или нет диалоговое окно выводиться в центре экрана. Значения могут быть `yes` или `no`.
- `border` — позволяет выбрать внешний вид границы окна. На выбор предоставляется два варианта: "толстая" и "тонкая" граница. Указание конкретного варианта производится с помощью одного из двух возможных значений: `thin` или `thick`, соответственно.
- `font` — предназначен для установки шрифта, который будет использован в оформлении диалогового окна. В качестве значения параметра используется выражение CSS.
- `font-family` — менее общий, чем предыдущий, параметр. Устанавливает семейство применяемого шрифта.
- `font-style` — позволяет указывать стиль применяемого шрифта, т. е. используемое начертание: полужирный, курсив и т. д.
- `font-weight` — задает величину используемого шрифта.
- `font-variant` — модификатор, указывающий, какие буквы будут использоваться в шрифте: строчные или прописные, чей размер не будет превышать размер строчных букв.
- `help` — регулирует наличие кнопки помощи в верхнем правом углу окна, на строке заголовка. Могут быть использованы значения `yes` или `no`.
- `minimize` — указывает на наличие или отсутствие в верхнем правом углу диалогового окна кнопки минимизации окна.
- `maximize` — устанавливает наличие или отсутствие в верхнем правом углу диалогового окна кнопки распахивания окна во весь экран.

Формат указания атрибутов для диалогового окна немного отличается от правила описания атрибутов окна просмотра. В данном случае они записываются не со знаком равенства, а с двоеточием. Приведем пример.

```
ShowModalDialog("dial.HTML", "New dialog", "border:thin,
↳minimize:yes, center:yes, font-family:Courier, font-style: Bold");
```

С помощью этого метода мы создаем диалоговое окно, в котором отобразится содержимое HTML-файла dial.html, в заголовке будет находиться текстовая строка New dialog, диалоговое окно расположится в центре экрана, его можно будет свернуть, а использоваться в нем будет моноширинный шрифт Courier с полужирным начертанием.

Теперь перейдем к рассмотрению списка событий, которые возникают при работе с объектом `window`. В состав данного объекта входит девять событий.

- Событие `onblur` возникает в тот момент, когда текущее окно просмотра теряет фокус ввода, т. е. становится неактивным.
- Событие `onfocus` инициируется в момент получения окном фокуса ввода. Данное событие противоположно только что рассмотренному.
- Событие `onerror` возникает, если происходит ошибка при загрузке какого-либо ресурса, входящего в состав отображаемой Web-страницы. Чаще всего событие применяется для обработки ошибок загрузки графики и видеовставок.
- Событие `onload` инициируется сразу после того, как заканчивается загрузка Web-страницы в браузер.
- Событие `onhelp` инициируется, если пользователь нажал клавишу <F1> или кнопку оперативной справки.
- Событие `onscroll` возникает, когда пользователь использует прокрутку содержимого окна просмотра с помощью линеек прокрутки.
- Событие `onresize` возникает, когда пользователь изменяет размеры окна просмотра.
- Событие `onbeforeunload` инициируется перед выгрузкой Web-страницы из браузера с маленькой временной задержкой, что позволяет провести некоторые операции, такие как сохранение данных.
- Событие `onunload` инициируется уже непосредственно в момент выгрузки Web-страницы из браузера.

Мы полностью рассмотрели структуру объекта `window`. Пора перейти к другим объектам, которые детально описывают иные механизмы браузера и содержимое Web-страницы.

Как мы уже говорили, объект `window` является основой всей объектной иерархии, и все остальные объекты входят в его состав. Эти вложенные объекты мы можем разделить на два типа. Одни из них открывают доступ к отдельным частям отображаемой Web-страницы, а другие позволяют нам управлять отдельными элементами самого браузера. Вот с этих объектов мы и начнем.

Объект `location` предоставляет нам доступ к URL текущей Web-страницы. Прежде всего мы рассмотрим описание его свойств.

- Свойство `href` содержит URL текущего документа. Если в процессе работы программы-скрипта это свойство получает новое значение в виде URL, то браузер автоматически загружает в окно просмотра документ, находящийся по этому адресу.

- Свойство `hostname` — имя Web-сервера, указанное в URL, не включающее в себя явно указанный номер порта, на котором функционирует этот Web-сервер.
- Свойство `host` — имя Web-сервера, объединенное с номером порта, если тот указан явно.
- Свойство `port` — номер порта, на котором функционирует Web-сервер, чье имя указано в URL данного документа.
- Свойство `pathname` — путь к HTML-документу в файловой системе сервера.
- Свойство `hash` — имя локальной гиперссылки, не включающее в себя знак "#".

Как видно, с помощью этих свойств мы можем полностью разделить URL документа на составляющие. Что делать с ними потом, каждый решает самостоятельно.

У объекта `location` есть и методы. Их всего три.

- Метод `reload` позволяет осуществлять принудительную перезагрузку данного HTML-документа.
- Метод `replace(url)` загружает в окно просмотра браузера новый HTML-документ, URL которого передается в метод в качестве параметра. При этом изменяется и объект `history`.
- Метод `assign(url)` переназначает URL для данного окна просмотра, но принудительного перехода по новому адресу, который передается в качестве параметра, не производится.

Каких-либо специфичных событий объект `location` не имеет.

Адреса всех просмотренных Web-ресурсов записываются в один список, и перемещение по этому списку осуществляется с помощью кнопок **Back** и **Forward** любого браузера. Обращение к списку обеспечивает объект с наименованием `history`. Обычно применяется три метода этого объекта.

- Метод `forward()` загружает следующий документ из списка загруженных ранее документов. Фактически, аналог нажатия удаленным пользователем кнопки **Forward** в своем браузере.
- Метод `back()` загружает предыдущий документ из общего списка. Дублирует кнопку браузера **Back**.
- Метод `go(offset)` смещается по списку на величину, которая передается в параметре и загружает документ с данным URL. При положительном значении параметра смещение происходит к концу списка, при отрицательном — к началу.

Длина списка URL посещенных Web-страниц хранится в свойстве `length`.

Теперь рассмотрим объект navigator, предназначенный для получения данных о браузере, используемом удаленным пользователем. Применение этого объекта обусловлено тем, что в результате "браузерных войн" основные браузеры: Internet Explorer и Netscape Navigator по-разному интерпретируют некоторые тэги HTML. У них не совпадают иерархии объектов, используемые в сценариях. Следовательно, если мы хотим создать HTML-документ, который будет без искажений отображаться в обоих основных браузерах, мы должны получить информацию об используемом браузере, и на ее основании формировать страницу. Свойства объекта navigator перечислены в следующем списке.

- Свойство appName содержит наименование браузера.
- Свойство appVersion — номер версии браузера.
- Свойство appCodeName — наименование технологии, которую использует браузер, часто называемой "движком" (engine).
- Свойство cookieEnabled указывает на то, разрешено ли сохранение cookie в данном браузере.
- Свойство userAgent содержит наименование браузера в том виде, в котором оно передается с помощью HTTP-протокола

Мы упоминали свойство cookieEnabled, которое связано с так называемыми "cookies". Cookies — это текстовые строки специализированного формата, которые хранятся на локальной машине удаленного пользователя. Данные в этих строках редактируются программами, размещенными на сайтах. Именно при помощи этих строк разработчики сайтов могут узнавать пользователей без запроса пароля, хранить историю работы пользователя с сайтом и иные данные, связанные с этим пользователем. Несмотря на очевидную полезность, cookies могут быть источником потенциальной угрозы или средством сбора информации о пользователе без его ведома. Поэтому браузеры позволяют запретить сохранение cookies.

В состав объекта navigator входит еще два объекта: mimeTypes и plugins, которые на самом деле являются не просто объектами, а коллекциями. Коллекция mimeTypes содержит различные типы MIME (способы кодирования передаваемой информации), которые используются данным браузером. С помощью коллекции plugins мы можем получить доступ к любому расширению браузера, установленному пользователем в качестве дополнения к стандартной поставке. Одним из наиболее известных расширений является plug-in Flash.

Помимо сведений о браузере необходимо иметь информацию об установленном разрешении монитора удаленного пользователя. Для этого мы можем использовать свойства объекта screen.

- width — разрешение экрана по горизонтали.
- height — разрешение экрана по вертикали.

- `colorDepth` — количество битов, используемых для указания цвета каждого пиксела, другими словами, глубина цвета.
- `updateInterval` — частота обновления экрана.

Данный объект не имеет даже методов, т. к. вся работа с ним происходит путем изменения значений его свойств.

Мы видели, что большинство рассмотренных нами объектов не имеет обособленных событий. Связано это с тем, что большинство событий, происходящих во время работы браузера, обрабатывается с помощью специализированного объекта `event`, который и несет информацию о большинстве событий. В любой момент мы можем обратиться к одному из его свойств и получить информацию о каком-либо изменении статуса или совершенном действии пользователя. Список свойств данного объекта достаточно обширен по сравнению с другими объектами и выглядит следующим образом:

- Свойство `altKey` указывает, нажата или нет клавиша <Alt> в данный момент. Используются значения логического типа — `True` и `False`.
- Свойство `button` показывает состояние кнопки мыши в момент запроса.
- Свойство `cancelBubble` запрещает событиям проходить вверх по объектной иерархии. Дело в том, что у многих объектов существуют одинаковые события, и в момент возникновения такого события, оно сначала обрабатывается у активного объекта, а затем это же событие может обрабатывать по очереди все объекты, содержащие активный. То есть, обработка события происходит снизу вверх по объектной иерархии. Если же значение свойства `cancelBubble` установить в `True`, то это событие будет обработано лишь один раз и не будет передано по объектной иерархии вверх.
- Свойство `clientX` содержит координату текущего объекта по горизонтали.
- Свойство `clientY` содержит координату текущего объекта по вертикали.
- Свойство `ctrlKey` показывает, нажата или нет клавиша <Ctrl> в данный момент.
- Свойство `fromElement` указывает на элемент, с которого пользователь увел курсор мыши. Данное свойство обычно применяется при обработке событий `onmouseout` и `onmouseover`.
- Свойство `keyCode` содержит код ASCII клавиши, нажатой пользователем.
- Свойство `offsetX` хранит смещение по горизонтали в пикселах курсора мыши от элемента, при обработке которого было инициировано событие.
- Свойство `offsetY` содержит смещение по вертикали в пикселах курсора мыши от элемента, при обработке которого было инициировано событие.
- Свойство `reason` является индикатором успешной передачи данных. Если передача не удалась, в данном свойстве указывается причина.

- Свойство `returnValue` содержит значение, которое будет возвращаться данным событием.
- Свойство `screenX` хранит абсолютную горизонтальную координату курсора мыши в пикселах относительно самого экрана.
- Свойство `screenY` содержит абсолютную вертикальную координату курсора мыши в пикселах относительно самого экрана.
- Свойство `shiftKey` показывает, нажата или нет клавиша <Shift> в данный момент.
- Свойство `srcElement` ссылается на исходный, самый нижний объект в иерархии, при работе с которым было инициировано данное событие.
- Свойство `srcFilter` используется только вместе с событием `onfilterexchange`, и содержит указатель на графический фильтр, который породил это событие.
- Свойство `toElement` указывает на элемент, на территории которого находится в данный момент курсор мыши.
- Свойство `type` описывает тип инициированного события. Значением данного свойства служит наименование события без первых двух символов "`on`".
- Свойство `x` содержит горизонтальную координату курсора мыши.
- Свойство `y` хранит вертикальную координату курсора мыши.

Объект `event` — это, прежде всего, информационный объект, поэтому большинство его свойств имеют статус "только для чтения". Мы можем изменять значения только двух свойств: `keyCode` и `returnValue`.

Теперь перейдем к рассмотрению объектов, которые описывают отображаемую Web-страницу. Основой этой ветви объектной иерархии является объект с наименованием `document`, который открывает нам доступ к операциям с Web-страницей, отображаемой в окне просмотра браузера. В силу того, что данный объект — основа всей соответствующей ветви объектной иерархии, он обладает достаточно обширным списком свойств, методов и событий. Начнем рассмотрение объекта `document` с его свойств.

- Свойство `activeElement` содержит ссылку на тот элемент управления Web-страницы, который в данный момент обладает фокусом ввода.
- Свойство `alinkColor` хранит обозначение цвета, которым отображаются активные гиперссылки.
- Свойство `bgColor` позволяет определять цвет фона загруженной Web-страницы.
- Свойство `body` содержит все наполнение Web-страницы, находящееся между тэгами `<body>` и `</body>`. Это свойство имеет статус "только для чтения".

- Свойство `cookie` хранит строку cookie, которая при загрузке Web-страницы записывается на локальном компьютере удаленного пользователя. С помощью технологии "cookie" разработчики Web-страниц могут сохранять некоторую информацию о посетителе страницы на его же компьютере и таким образом персонализировать работу Web-сайта с пользователями.
- Свойство `domain` содержит доменное имя сайта, с которого была получена загруженная Web-страница.
- Свойство `fgColor` определяет цвет, применяемый по умолчанию для отображения текста.
- Свойство `lastModified` указывает дату последнего изменения данной Web-страницы.
- Свойство `linkColor` обозначает цвет, которым отображаются ссылки, еще не активизированные пользователем.
- Свойство `location` задает полный URL загруженной в браузер Web-страницы.
- Свойство `parentWindow` указывает на родительское окно, из которого было открыто окно с данной Web-страницей.
- Свойство `readyState` показывает текущий статус загрузки данного HTML-документа. Значением этого свойства может быть число в диапазоне от единицы до четырех. Значение "`1`" указывает, что документ еще не инициализирован, и загрузка его в браузер не началась. Значение "`2`" обозначает, что загрузка началась, но еще не закончилась. Значение "`3`" указывает, что загрузка закончена, но документ еще не отображен в окне просмотра. Значение "`4`" обозначает, что Web-страница полностью загружена, отображена в браузере, и готова к работе.
- Свойство `referrer` содержит URL страницы, которая ссылается на текущую Web-страницу, загруженную в браузер. То есть на ту, с которой пользователь и перешел на текущую страницу.
- Свойство `title` хранит заголовок данной Web-страницы, заключенный между тэгами `<title>` и `</title>`.
- Свойство `URL` содержит полный URL Web-страницы, загруженной в браузер. Фактически дублирует свойство `location`, но у некоторых объектов может отличаться от него.
- Свойство `vlinkColor` определяет цвет, которым отображаются ссылки, уже посещенные пользователем.

На этом список свойств объекта `document` заканчивается, и мы переходим к рассмотрению его методов.

- Метод `createElement(teg)` создает HTML-объект, наименование тэга которого задано в качестве параметра данного метода. Так, если мы хотим

создать на нашей Web-странице дополнительное графическое изображение, следует выполнить метод `document.createElement("IMG")`.

- Метод `clear()` очищает содержимое Web-страниц.
- Метод `close()` закрывает документ, а с ним и текущее окно просмотра.
- Метод `elementFromPoint(x,y)` возвращает HTML-объект, который находится в точке с координатами, переданными методу в качестве параметров.
- Метод `execCommand(command)` выполняет над выделенной областью Web-страницы некую операцию, код которой передан в качестве параметра,.
- Метод `open(type)` создает новый документ, mime-тип которого передается в качестве параметра, и открывает новое окно просмотра для отображения создаваемого документа. Метод обычно применяется для создания Web-страниц с динамически формируемым содержимым, поэтому в качестве параметра передается строка "`text/HTML`".
- Метод `queryCommandEnabled(command)` позволяет определять, можно ли выполнять данную команду над выделенной областью Web-страницы.
- Метод `queryCommandIndeterm(command)` сообщает, какой статус имеет данная команда.
- Метод `queryCommandState(command)` возвращает текущее состояние данной команды. Может использоваться для контроля за выполнением переданных исполняемых инструкций.
- Метод `queryCommandSupported(command)` указывает, поддерживается ли данная команда браузером пользователя.
- Метод `queryCommandText(command)` возвращает текстовое выделение, к которому применяется команда, переданная методу в качестве параметра.
- Метод `write(text)` записывает в тело документа новый HTML-код, который передается в качестве параметра. Таким образом, можно динамически формировать содержимое Web-страницы без использования специализированных приложений, действующих на Web-сервере.
- Метод `writeln(text)` позволяет записывать в тело документа HTML-код, но при этом дописывает в конце добавляемого блока символ перевода каретки.

Как видно, практически все методы объекта `document` предназначены для динамического формирования его содержимого. Немного позже мы увидим, как это делается, а пока перейдем к рассмотрению списка событий, которые могут возникать при работе с данным объектом.

- Событие `onafterupdate` возникает, когда пользователь при работе с формой, размещенной на Web-странице, отослал свои данные на сервер, а он успешно принял и обработал их.

- Событие `onbeforeupdate` инициируется в том случае, если пользователь ввел данные в форму, но вместо их отправки попытался выгрузить страницу из браузера.
- Событие `onclick` происходит в момент, когда пользователь производит на документе одиночный щелчок мышью.
- Событие `ondblclick` возникает в момент двойного щелчка мышью на одном из элементов загруженной Web-страницы.
- Событие `ondragstart` инициируется в тот момент, когда пользователь начинает перетаскивать с помощью мыши какой-либо элемент загруженной Web-страницы.
- Событие `onerror` происходит при наличии ошибки в сценарии или в случае возникновения ошибки при передаче каких-либо данных, например, графических изображений.
- Событие `onhelp` возникает в тот момент, когда пользователь нажимает клавишу <F1>.
- Событие `onkeydown` инициируется, когда пользователь нажимает какую-либо клавишу. Какая именно клавиша была нажата, мы можем узнать с помощью объекта `event`.
- Событие `onkeypress` постоянно генерируется в то время, когда пользователь нажал какую-либо клавишу и еще не отпустил ее.
- Событие `onkeyup` возникает, когда пользователь отпускает ранее нажатую клавишу.
- Событие `onload` происходит в тот момент, когда загрузка HTML-документа в браузер полностью заканчивается.
- Событие `onmousedown` возникает в тот момент, когда пользователь нажимает какую-либо кнопку мыши.
- Событие `onmousemove` постоянно генерируется при перемещении мыши пользователем.
- Событие `onmouseout` возникает, когда пользователь уводит курсор мыши с данного объекта.
- Событие `onmouseover` происходит при попадании курсора мыши на объект.
- Событие `onmouseup` инициируется в тот момент, когда пользователь отпускает заранее нажатую кнопку мыши.
- Событие `onreadystatechange` инициируется каждый раз, когда по каким-либо причинам изменяется значение свойства `readyState`.
- Событие `onselectstart` возникает в тот момент, когда пользователь начинает выделять мышью некую часть Web-страницы.

И на этом заканчивается список событий, которые могут возникать при работе с объектом document. Мы рассмотрели полностью структуру этого объекта. Но мы уже говорили, что он является "родителем" (или объектом-контейнером) для многих других объектов, которые более детально описывают элементы Web-страниц. Необходимо рассмотреть и их.

Мы уже видели, что у объекта document существует свойство body, которое содержит все тело Web-страницы. Но страницы с фреймовой структурой вообще не содержат тэг <body>. В этом случае мы можем работать с коллекцией фреймов, которая носит наименование frames и входит в состав объекта window. Мы уже говорили, что коллекции, по сути, являются массивами, состоящими из объектов. Если нам надо получить доступ к какому-либо фрейму, мы можем сделать это либо с помощью его порядкового номера, либо с помощью его наименования. Так, например, если самый верхний фрейм нашей Web-страницы носит наименование "topframe", то обратиться к нему мы можем с помощью двух эквивалентных конструкций:

```
window.frames(0)
window.frames("topframe")
```

Как мы помним, нумерация в массивах, а следовательно, и в коллекциях, начинается с нуля. Для циклической обработки коллекции нам необходимо знать не только стартовый номер, но и общее количество элементов, входящих в состав данной коллекции. Для каждой коллекции существует служебное свойство length, которое содержит количество элементов в ней. Следует помнить, что количество элементов и порядковый номер последнего элемента — разные вещи, т. к. нумерация начинается с нуля.

Из приведенного примера видно, что для доступа к элементу коллекции нужно сначала указать родительский объект. Наименования объектов разделяются точками.

Каждый элемент коллекции frames функционально эквивалентен объекту window, поэтому обладает всеми его свойствами, методами, событиями и встроенными объектами, исключая объекты, обеспечивающие доступ к внутренним механизмам браузера, такие как history и navigator. Следовательно, каждый элемент коллекции frames имеет встроенный объект document со всем его содержимым.

Объект document, как мы знаем, полностью описывает Web-страницу, отображаемую в окне просмотра браузера или фрейма. Следовательно, в состав этого объекта должны входить другие объекты и коллекции, которые смогут предоставить нам доступ к любому HTML-блоку, к любому тэгу. Рассмотрим список этих объектов и коллекций с кратким описанием каждого из них.

- Коллекция links содержит все гиперссылки и области, сформированные тэгами <area> на данной Web-странице.

- Коллекция `forms` обеспечивает доступ ко всем формам данного HTML-документа. Соответственно, каждый элемент этой коллекции содержит в своем составе объекты, позволяющие обращаться к различным элементам ввода данных, из которых состоит форма.
- Коллекция `anchors` хранит все закладки на странице, доступ к которым осуществляется с помощью гиперссылок.
- Коллекция `images` предназначена для хранения объектов графических изображений, внедренных в состав содержимого Web-страницы.
- Коллекция `scripts` обеспечивает доступ ко всем программам-сценариям, которые были вставлены в код HTML-документа с помощью тэга `<script>`.
- Коллекция `applets` содержит все внедренные объекты, такие как апплеты, элементы ActiveX, графические изображения, видеоклипы и любые объекты, обрабатываемые и отображаемые с помощью расширений (`plug-ins`) браузера.
- Коллекция `plugins` хранит объекты, связанные со всеми внедренными элементами, обрабатываемыми расширениями браузера.
- Коллекция `styleSheets` предоставляет нам доступ ко всем стилевым таблицам, внедренным в данный HTML-документ.
- Коллекция `filters` содержит все графические фильтры, примененные к содержимому данной Web-страницы. Графические фильтры не унаследованы из HTML, они являются прерогативой DHTML.
- Коллекция `all` хранит ссылки на все тэги и объекты, внедренные в состав содержимого данной Web-страницы.
- Объект `selection` позволяет работать с выделенной частью содержимого Web-страницы. Обычно это выделение производит пользователь.
- Объект `textRange` представляет собой блок текста, с помощью которого обычно разработчик динамически формирует текстовое содержимое HTML-документа.

Мы перечислили основные составные части объекта `document`. Конечно, одного их перечисления недостаточно для того, чтобы начать с ними работать. Несколько позже, в следующих разделах данной главы мы будем разбирать примеры задач, решаемых с помощью этих объектов, и нам придется тщательно рассмотреть их структуру.

Сейчас пора перейти к рассмотрению событийной модели DHTML и увидеть первые примеры использования программ-скриптов.

Обработка событий

В предыдущем разделе этой главы мы видели список событий, присущий объектам `document` и `window`. Уже, исходя из их наименований и кратких описаний, было ясно, что они инициируются при некоторых действиях пользователя или возникновении некоторых заранее определенных ситуаций. Любое действие, производимое с помощью программ-скриптов DHTML, может быть выполнено только в ответ на некоторое событие, благо их перечень охватывает все мыслимые ситуации, возникающие при работе с Web-страницей.

Рассмотрим простейший пример создания программы-скрипта, реагирующей на какое-либо событие. Например, программа изменит начертание символов текстового абзаца, когда пользователь щелкнет кнопкой мыши на тексте этого абзаца. Для этого мы должны создать HTML-документ, внешний вид которого приведен на рис 3.1, а код указан в листинге 3.1.

Листинг 3.1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Изменяемый шрифт</title>
  <script language="javascript">
  <!--
  function changeweight()
  {p1.style.fontWeight="bold";
  }
  //-->
  </script>
  </head>
  <body>
  <p id="p1" onClick="changeweight()"><font size=4>Строка
↳текста</font></p>
  <p id="p2"><font size=4>Другая строка</font></p>
  </body>
</html>
```

Этот маленький пример позволяет очень наглядно продемонстрировать основные принципы работы DHTML. Начнем мы с правил внедрения скриптов в HTML-документ.

Мы уже знаем, что скрипт должен размещаться в заголовке HTML-документа, между тэгами `<head>` и `</head>`. Сам код оформляется с помощью тэгов `<script>` и `</script>`. Пропуск закрывающего тэга является достаточно

распространенной ошибкой начинающих разработчиков, поэтому всегда следует проверять наличие закрывающих тэгов. В данном случае, если пропустить закрывающий тэг, браузер будет считать весь код, размещенный после тэга <script>, программой и, соответственно, не будет отображать его.

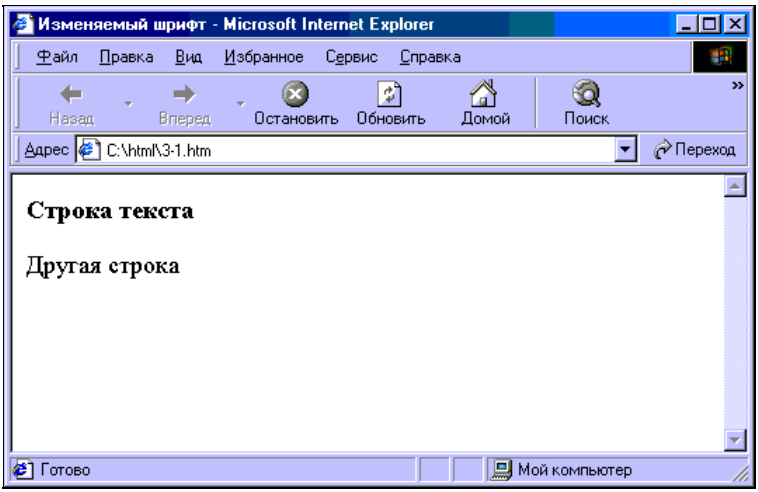


Рис. 3.1. Внешний вид HTML-документа после работы программы-скрипта

Между парой тэгов <script> размещен код самой программы. Для того чтобы устаревшие версии браузеров, не поддерживающие технологию программ-сценариев, не видели этот код, он обязательно оформляется как комментарии. В примере это видно.

Теперь ненадолго отвлечемся от кода программы и перейдем к рассмотрению тела HTML-документа. Основным его элементом является один текстовый абзац, объявленный с помощью тэга <p>. Из листинга видно, что помимо самого тэга мы использовали и дополнительные параметры. Одним из них является идентифицирующий параметр id, с помощью которого задаем уникальный идентификатор данного элемента Web-страницы. Благодаря этому уникальному идентификатору мы смогли избирательно применить динамическое оформление текста.

В состав открывающего тэга <p> входит дополнительный параметр, который мы не рассматривали в первой главе. Дело в том, что официально в HTML не существует подобного параметра. Как параметр мы задали наименование события, при возникновении которого управление перейдет к программе-скрипту. Подобные способы записи событий как параметров присущи именно технологии DHTML. В качестве значения этого параметра мы записали выполняемый код. В нашем случае это был лишь вызов функции, которую мы заранее определили в заголовке HTML-документа. Сама функция

в данном случае также чрезвычайно проста. Мы использовали всего один оператор:

```
p1.style.fontWeight="bold";
```

Видно, что левая часть оператора является обращением к некоему объекту, но в рассмотренной нами объектной иерархии не было объекта с наименованием "p1". Более того, очевидно, что он связан с текстовым абзацем, который имеет точно такой же идентификатор.

Дело в том, что в состав объекта document входит неявный объект с общим наименованием element. Объект мы называем "неявным" потому, что, создавая свою Web-страницу, мы можем заменить его любым, я подчеркиваю, любым элементом Web-страницы, который создается с помощью какого-либо тэга. При этом наименование объекта определяется идентификатором данного элемента, т. е. значением его идентифицирующего параметра id.

Для того чтобы изменить свойства отображения элемента p1, мы использовали объект style, входящий в состав объекта element. Этот объект позволяет применить всю мощь стилевых таблиц без прямого их создания. В коде Web-страницы мы не указали явно начертание шрифта, которым отображается текст абзаца. Но в программе-скрипте применили свойство fontWeight, которое позволяет задавать ширину символов текста, и при выполнении функции начертание шрифта было переопределено.

Следует обратить внимание, что скрипт-программа была выполнена в ответ на некое событие, порожденное действиями пользователя. Еще раз повторюсь, что идеология DHTML полностью ориентирована именно на события. Подобный подход в программировании обычно называют "событийно-ориентированным". Именно поэтому для овладения технологией DHTML в полной мере необходимо досконально разбираться в причинах возникновения событий и способах их обработки.

В DHTML важную роль играет концепция "конвейера событий". Суть этой концепции достаточно проста. Мы знаем, что в HTML одни элементы могут находиться внутри других. Например, абзац, порождаемый тэгами и , располагается внутри ячейки таблицы. Сами ячейки размещаются внутри отдельных строк таблицы, а те, в свою очередь, вставлены в единую таблицу, порожденную тэгами <table> и </table>. Теперь посмотрим, что произойдет, если пользователь произведет одиночный щелчок кнопкой мыши по этому абзацу. Абзац, естественно, инициирует событие onClick. Но ведь абзац находится в ячейке, следовательно, пользователь щелкнул и по ячейке. И сама ячейка тоже инициирует событие onClick, а за ней точно также поступит элемент строки таблицы и вся таблица. И так очередь дойдет до основного элемента body.

При этом, если для всех или некоторых элементов, которые участвуют в этом событии, были определены функции, реагирующие на щелчок кнопкой мыши, то они все будут выполнены. Если же разработчику необходимо

обрабатывать событие только в рамках того элемента, где оно произошло, не отправляя вверх по объектной иерархии, следует воспользоваться одним из свойств объекта event. Мы уже рассматривали этот объект и упоминали его свойство cancelBubble, которое отменяет перемещение события вверх по объектной иерархии. Следовательно, если бы мы в нашем скрипте захотели установить обработку события только для выбранного элемента, запретив передачу события вверх по объектной иерархии, наш скрипт выглядел бы следующим образом:

```
<script language="javascript">
<!--
function changecolor()
{p1.style.color="blue";
window.event.cancelBubble=True;
}
//-->
</script>
```

Как видно, в этом случае мы лишь присвоили логическое значение True описанному свойству. Этого достаточно для того, чтобы отменить обработку данного события всеми вышестоящими элементами. В предыдущем разделе мы уже рассматривали структуру объекта event. Мы знаем, что данный объект с помощью своих свойств предоставляет подробную информацию обо всех событиях, которые входят в состав событийной модели DHTML.

Следует также упомянуть, что различные элементы Web-страниц могут реагировать на разные события. Список возможных событий определяется функциональностью каждого элемента Web-страницы. Приведем таблицу соответствия тэгов и обрабатываемых этими тэгами событий.

Таблица 3.2

Тэг	Описание	Список поддерживаемых событий
<a>	Гиперссылка	onblur, onclick, ondblclick, ondragstart, onerrorupdate, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<address>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<applet>	Внедрение в состав Web-страницы java-апплета	onafterupdate, onbeforeupdate, onblur, onclick, ondataavailable, ondatasetchanged, ondatasetcomplete, ondblclick, ondragstart, onerrorupdate, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreadystatechange, onresize, onrowenter, onrowexit, onselectstart
<area>	Создание активной области-гиперссылки в сегментированной графике	onblur, onclick, ondblclick, ondragstart, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
<b>	Выделение текста полужирным шрифтом	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<big>	Увеличение размера шрифта на единицу	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
<blockquote>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<body>	Определение содержательной части Web-страницы	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<button>	Создание кнопки	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart
<caption>	Создание заголовка таблицы	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart
	Центрирование блока содержимого	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<cite>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<code>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<dd>	Элемент списка	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<dfn>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
	Объединение элементов содержимого Web-страницы	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart
<dl>	Создание списка	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<dt>	Создание элемента списка	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<embed>	Внедрение в состав содержимого Web-страницы объектов различных типов	onblur, onfocus
<font>	Определение шрифта для отображения текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<form>	Определение формы в HTML-документе	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreset, onselectstart, onsubmit
<h1> — <h6>	Создание заголовков	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<hr>	Отображение горизонтальной линии	onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onselectstart
<i>	Выделение текста курсивом	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<img>	Вставка графического изображения	onabort, onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onerror, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart
<input>	Создание элемента ввода информации	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onerrorupdate, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onselect, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<kbd>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<label>	Создание текстовой метки для некоторых управляющих элементов	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
	Создание элемента списка	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<link>	Объявление связи между различными HTML-документами	onerror, onload, onreadystatechange
<listing>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<map>	Создание карты сегментированной графики	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<object>	Внедрение какого-либо объекта в HTML-документ	onafterupdate, onbeforeupdate, onblur, onclick, ondataavailable, ondatasetchanged, ondatasetcomplete, ondblclick, ondragstart, oneror, onerrorupdate, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreadystatechange, onreset, onresize, onrowenter, onrowexit, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
	Создание нумерованного списка	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<p>	Задание текстового абзаца	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<plaintext>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<pre>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<s>	Отображение зачеркнутого текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<script>	Объявление программы-скрипта, включенной в состав HTML-документа	onerror, onload, onreadystatechange
<select>	Создание элемента управления	onafterupdate, onbeforeupdate, onblur, onchange, onclick, ondblclick, ondragstart, onerrorupdate, onfilterchange, onfocus, onhelp, onkeydown, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit

Таблица 3.2 (продолжение)

Тэг	Описание	Список поддерживаемых событий
<small>	Уменьшение размера используемого шрифта	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
	Объединение элементов Web-страницы	onblur, onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onscroll, onselectstart
<strike>	Отображение зачеркнутого текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
	Отображение текста полужирным начертанием шрифта	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<style>	Создание таблицы стилей отображения	onerror, onload, onreadystatechange
<sub>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<sup>	Специализированное форматирование текста	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<table>	Создание таблицы	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselectstart

Таблица 3.2 *(продолжение)*

Тэг	Описание	Список поддерживаемых событий
<tbody>	Описание основного раздела таблицы	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<td>	Объявление ячейки таблицы	onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onselectstart
<textarea>	Создание многострочного поля текстового ввода	onafterupdate, onbeforeupdate, onblur, onchange, onclick, ondblclick, ondragstart, onerrorupdate, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onscroll, onselect, onselectstart
<tfoot>	Создание нижней части таблицы	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<th>	Объявление строки с ячейками заголовка таблицы	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart
<thead>	Создание области заголовка таблицы	onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart

Таблица 3.2 *(окончание)*

Тэг	Описание	Список поддерживаемых событий
`<tr>`	Объявление строки таблицы	`onafterupdate, onbeforeupdate, onblur, onclick, ondblclick, ondragstart, onfilterchange, onfocus, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onresize, onrowenter, onrowexit, onselectstart`
`<tt>`	Специализированное шрифтовое оформление	`onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart`
`<u>`	Отображение подчеркнутого текста	`onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart`
`<ul>`	Создание маркированного списка	`onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart`
`<var>`	Специализированное форматирование текста	`onclick, ondblclick, ondragstart, onfilterchange, onhelp, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselectstart`

Этот список позволяет при создании программ-скриптов определять, какие события может порождать тот или иной тэг.

Иногда нам необходимо получить более подробную информацию о текущей ситуации. Стандартных событий может оказаться недостаточно. В этом случае разработчику поможет объект `event`. Его структуру мы уже обсуждали в предыдущем разделе этой главы. Теперь мы можем на примере рассмотреть механизм использования этого объекта. Мы уже говорили, что подавляющее большинство свойств объекта `event` носит сугубо информативный характер, и потому доступны в режиме "только для чтения". Возможный способ работы с ними показан в листинге 3.2 (результат отображения кода — рис. 3.2).

Листинг 3.2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Обработка событий</title>
  <script language="javascript">
  <!--
  function info()
  {
  s="Координаты курсора мыши " + window.event.screenX + "," +
↳window.event.screenY;
  alert(s);
  }
  window.document.ondblclick=info;
  //-->
  </script>
  </head>
  <body>
  <p><font size=5>Обычная Web-страница</font></p>
  </body>
</html>
```

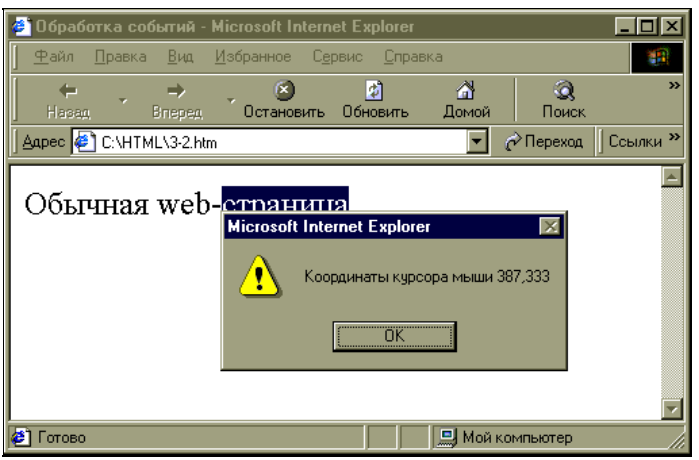


Рис. 3.2. Окно браузера с результатом отображения файла, приведенного в листинге 3.2

Разберем структуру использованного нами скрипта. В самом начале программы мы объявляем функцию с наименованием `info`. После кода этой функции мы вставили строку следующего вида:

```
window.document.ondblclick=info;
```

С помощью этой конструкции мы указываем, что функция `info` будет выполняться в том случае, если пользователь произведет двойной щелчок мышью в окне просмотра браузера. Так как мы не можем использовать для этих целей какой-либо тэг содержимого Web-страницы, нам приходится описывать событие объекта `document`.

Теперь рассмотрим механизм действия функции `info`. Функция состоит из двух операторов. Первый из них присваивает переменной некую строку, формируя ее из заданного текста и двух значений объекта `event`, которые указывают координаты курсора мыши в момент совершения пользователем двойного щелчка кнопкой мыши. Второй оператор с помощью ключевого слова `alert` отображает модальное окно с текстом, который содержится в переменной `s` из первого оператора (рис. 3.2).

Мы увидели в действии основные механизмы обработки событий в DHTML. Теперь можно создавать свои программы-скрипты, ориентированные на обработку событий, возникающих при работе пользователя с нашими HTML-документами. Пришло время привести перечень всех возможных событий, используемых в технологии DHTML. В табл. 3.3 перечислены эти события и ситуации, в которых они возникают.

Таблица 3.3

Событие	Описание
`onabort`	Инициируется, когда пользователь принудительно прерывает загрузку данных
`onafterupdate`	Возникает в момент окончания передачи данных
`onbeforeunload`	Инициируется перед выгрузкой страницы
`onblur`	Происходит, когда объект теряет фокус ввода
`onchange`	Возникает при изменении содержимого объекта
`onclick`	Происходит при одиночном щелчке кнопкой мыши на объекте
`ondataavailable`	Инициируется при получении данных из источника
`ondatasetchanged`	Возникает при изменении набора данных, на основе которого функционирует элемент
`ondatasetcomplete`	Инициируется в тот момент, когда исходный набор данных становится полностью доступным для документа
`ondblclick`	Происходит при выполнении пользователем двойного щелчка кнопкой мыши на элементе
`ondragstart`	Возникает в тот момент, когда пользователь начинает перетаскивать объект с помощью мыши

Таблица 3.3 *(продолжение)*

Событие	Описание
onerror	Инициируется в случае возникновения ошибки при передаче данных
onerrorupdate	Возникает при отмене изменения данных
onfilterchange	Происходит при изменении состояния графического фильтра
onfilterevent	Инициируется по окончании этапа действия графического фильтра
onfocus	Возникает при получении объектом фокуса ввода
onhelp	Происходит в тот момент, когда пользователь нажимает клавишу <F1>
onkeydown	Инициируется при нажатии пользователем какой-либо клавиши на клавиатуре
onkeypress	Происходит, когда пользователь нажал клавишу и удерживает ее в нажатом положении
onkeyup	Возникает, когда пользователь отпускает нажатую клавишу
onload	Инициируется, когда загрузка объекта полностью завершается
onmousedown	Возникает, когда пользователь нажимает кнопку мыши
onmousemove	Происходит, когда пользователь перемещает мышь
onmouseout	Инициируется в тот момент, когда пользователь уводит курсор мыши с пространства, занимаемого объектом
onmouseover	Происходит в тот момент, когда пользователь перемещает курсор мыши в пространство, занимаемое объектом
onmouseup	Возникает, когда пользователь отпускает ранее нажатую кнопку мыши
onreadystatechange	Происходит при изменении свойства readystate
onreset	Возникает при нажатии пользователем на кнопку **Reset**, расположенную на форме
onresize	Инициируется в тот момент, когда пользователь изменяет размеры окна просмотра
onrowenter	Происходит при изменении данных в строке, связанной с внешним источником данных
onrowexit	Инициируется перед тем, как данные в строке будут изменены источником данных

Таблица 3.3 *(окончание)*

Событие	Описание
onscroll	Возникает, когда пользователь прокручивает содержимое Web-страницы в окне просмотра браузера
onselect	Происходит при изменении текущей выделенной области
onselectstart	Инициируется, когда пользователь начинает выделять область содержимого страницы
onsubmit	Инициируется, когда пользователь нажимает на кнопку **Submit**, расположенную на форме, и отсылает данные из формы на сервер
onunload	Возникает непосредственно перед выгрузкой страницы из окна просмотра браузера

Теперь, когда мы знаем, какие события соответствуют различным элементам Web-страницы и в каких ситуациях эти события возникают, мы сможем создавать действительно разветвленные и гибкие программы-сценарии. Для решения этой задачи нам осталось познакомиться со свойствами и методами, присущими различным элементам Web-страниц. Об этом мы узнаем в следующем разделе.

Свойства и методы элементов Web-страниц

В предыдущем разделе главы мы узнали, что каждый элемент Web-страницы, объявляемый с помощью того или иного тэга, обладает собственным набором событий. Наличие событий свойственно, как мы знаем, объектам DHTML. А раз наши элементы являются объектами, то у них должны быть свои свойства и методы. И это действительно так.

Основные свойства и методы мы рассмотрели в этой главе несколько ранее. В данном разделе мы узнаем специфические свойства и методы каждого конкретного элемента Web-страницы. В табл. 3.4 приведен список методов для каждого элемента.

Таблица 3.4

Тэг	Описание	Список методов
<a>	Гиперссылка	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
<address>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<applet>	Внедрение в состав Web-страницы java-апплета	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<area>	Создание активной области-гиперссылки в сегментированной графике	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<b>	Выделение текста полужирным шрифтом	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<base>	Указание URL основного документа	contains, getAttribute, removeAttribute, setAttribute
<basefont>	Определение шрифта, используемого по умолчанию	contains, getAttribute, removeAttribute, setAttribute
<bgsound>	Установка фонового звука Web-страницы	contains, getAttribute, removeAttribute, setAttribute
<big>	Увеличение размера шрифта на единицу	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<blockquote>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<body>	Определение содержательной части Web-страницы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
 	Вставка принудительного разрыва строки	contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, setAttribute
<button>	Создание кнопки	blur, click, contains, createtextRange, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<caption>	Создание заголовка таблицы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Центрирование блока содержимого	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<cite>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<code>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<col>	Определение столбцов таблицы	contains, getAttribute, removeAttribute, setAttribute
<colgroup>	Создание группы столбцов	contains, getAttribute, removeAttribute, setAttribute
<dd>	Элемент списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<dfn>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
	Объединение элементов содержимого Web-страницы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<dl>	Создание списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<dt>	Создание элемента списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<embed>	Внедрение в состав содержимого Web-страницы объектов различных типов	blur, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<font>	Определение шрифта для отображения текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<form>	Задание формы в HTML-документе	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, reset, scrollIntoView, setAttribute, submit
<frame>	Описание отдельного фрейма	contains, getAttribute, removeAttribute, setAttribute
<frameset>	Определение фреймовой структуры	contains, getAttribute, removeAttribute, setAttribute
<head>	Создание заголовка документа	contains, getAttribute, removeAttribute, setAttribute

Таблица 3.4 (продолжение)

Тэг	Описание	Список методов
<h1> — <h6>	Создание заголовков в тексте	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<hr>	Отображение горизонтальной линии	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<html>	Объявление содержимого как HTML-документа	contains, getAttribute, removeAttribute, setAttribute
<i>	Выделение текста курсивом	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<img>	Вставка графического изображения	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<input>	Создание управляющего элемента	blur, click, contains, createTextRange, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, select, setAttribute t
<kbd>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<label>	Создание текстовой метки для некоторых элементов управления	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Создание элемента списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
<link>	Объявление связи между различными HTML-документами	contains, getAttribute, removeAttribute, setAttribute
<listing>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<map>	Создание карты сегментированной графики	click, contains, getAttribute, removeAttribute, setAttribute
<meta>	Установка meta-переменных HTML-документа	contains, getAttribute, removeAttribute, setAttribute
<object>	Внедрение какого-либо объекта в HTML-документ	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Создание нумерованного списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<option>	Создание элемента выпадающего списка	contains, getAttribute, removeAttribute, scrollIntoView, setAttribute
<p>	Определение текстового абзаца	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<plaintext>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<pre>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
<s>	Отображение зачеркнутого текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<script>	Описание программы-скрипта, включенной в состав HTML-документа	contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, setAttribute
<select>	Создание элемента управления	add, blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, item, remove, removeAttribute, scrollIntoView, setAttribute, tags
<small>	Уменьшение размера используемого шрифта	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Объединение элементов Web-страницы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<strike>	Отображение зачеркнутого текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Отображение текста полужирным начертанием шрифта	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<style>	Создание таблицы стилей отображения	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<sub>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 *(продолжение)*

Тэг	Описание	Список методов
<sup>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<table>	Создание таблицы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, nextPage, prevPage, refresh, removeAttribute, scrollIntoView, setAttribute
<tbody>	Обозначение основного раздела таблицы	click, contains, getAttribute, removeAttribute, scrollIntoView, setAttribute
<td>	Объявление ячейки таблицы	blur, click, contains, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<textarea>	Создание многострочного поля текстового ввода	blur, click, contains, createTextrange, focus, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, select, setAttribute
<tfoot>	Создание нижней части таблицы	click, contains, getAttribute, removeAttribute, scrollIntoView, setAttribute
<th>	Объявление строки заголовка таблицы	click, contains, getAttribute, removeAttribute, scrollIntoView, setAttribute
<thead>	Создание области заголовка таблицы	click, contains, getAttribute, removeAttribute, scrollIntoView, setAttribute
<title>	Создание заголовка HTML-документа	contains, getAttribute, removeAttribute, setAttribute
<tr>	Объявление строки таблицы	blur, click, contains, focus, getAttribute, removeAttribute, scrollIntoView, setAttribute
<tt>	Специализированное шрифтовое оформление	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

Таблица 3.4 (окончание)

Тэг	Описание	Список методов
<u>	Отображение подчеркнутого текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
	Создание маркированного списка	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute
<var>	Специализированное форматирование текста	click, contains, getAttribute, insertAdjacentHTML, insertAdjacentText, removeAttribute, scrollIntoView, setAttribute

В этой таблице приведены перечни методов, свойственных тем или иным элементам содержимого Web-страниц. Что означают некоторые из них, мы уже знаем, т. к. рассматривали их несколько ранее в данной главе. Но успели мы познакомиться далеко не со всеми, поэтому стоит обратить внимание на следующую табл. 3.5, в которой собрана вся информация о методах объектной иерархии DHTML.

Таблица 3.5

Метод	Описание
add	Добавляет элемент в список или коллекцию
addAmbient	Добавляет внешнее освещение в графический фильтр
addCone	Добавляет конический источник света в графический фильтр
addImport	Подключает стилевую таблицу из файла, URL которого передается в метод как параметр, к текущему HTML-документу
addPoint	Добавляет точечный источник света к действующему графическому фильтру
addRule	Добавляет еще одно правило отображения к таблице стилей
alert	Отображает окно сообщения
apply	Применяет графический фильтр к объекту

Таблица 3.5 *(продолжение)*

Метод	Описание
assign	Загружает HTML-документ, URL которого передан методу в качестве параметра
back	Загружает предыдущий HTML-документ из списка посещенных Web-страниц
blur	Удаляет фокус ввода с объекта
changeColor	Изменяет цвет объекта
changeStrength	Изменяет интенсивность освещения
clear	Очищает содержимое объекта
clear	Удаляет все источники света из графического фильтра
clearInterval	Сбрасывает таймер, заданный методом `setInterval`
clearTimeout	Сбрасывает таймер, заданный методом `setTimeout`
click	Имитирует щелчок кнопкой мыши на объекте и инициирует событие `onclick`
close	Закрывает окно просмотра браузера
collapse	Сворачивает текстовую область в точку
compareEndPoints	Сравнивает две текстовые области
confirm	Отображает окно с текстом и кнопками **OK** и **Cancel**
contains	Проверяет, включен ли в текущий объект другой объект, наименование которого передается методу в качестве параметра
createElement	Создает элемент, тэг для его объявления передается методу в качестве параметра
createRange	Создает копию выделенной части документа
createTextRange	Создает область текста
duplicate	Копирует текстовую область
elementFromPoint	Возвращает элемент, находящийся в точке, координаты которой переданы методу в качестве параметров
empty	Очищает выделение
execCommand	Выполняет заданную команду
execScript	Выполняет заданный скрипт
expand	Расширяет область текста, добавляя в нее новые фрагменты

Таблица 3.5 *(продолжение)*

Метод	Описание
`findText`	Ищет текст, переданный в качестве параметра, в содержимом документа
`focus`	Переводит фокус ввода к данному объекту
`forward`	Загружает в окно просмотра HTML-документ, который находится в списке посещенных страниц на одну позицию впереди текущего документа
`getAttribute`	Возвращает значение параметра-атрибута текущего тэга, наименование которого передано методу в качестве параметра
`getBookmark`	Возвращает строковое значение, определяющее закладку для локальной гиперссылки в данном объекте
`go`	Загружает в окно просмотра браузера HTML-документ из списка посещенных Web-страниц, номер его относительно текущего документа передается методу в качестве параметра
`inRange`	Возвращает булево значение, которое указывает, находится ли текстовая область, переданная методу в качестве параметра, внутри исходной области
`insertAdjacentHTML`	Вставляет внутрь текущего элемента HTML-код, переданный методу в качестве параметра. При этом все тэги адекватно обрабатываются браузером
`insertAdjacentText`	Вставляет внутрь текущего элемента текст, переданный методу в качестве параметра. Если в переданном тексте имеются тэги HTML, то они игнорируются браузером
`isEqual`	Возвращает логическое значение, которое указывает, эквивалентна ли текстовая область (`TextRange`), переданная методу в качестве параметра, исходной, к которой метод и применяется
`item`	Возвращает входящий в коллекцию объект с порядковым номером, который передается методу в качестве параметра
`javaEnabled`	Возвращает логическое значение, которое показывает, включена или нет поддержка Java в данном браузере
`move`	Перемещает границы текстовой области
`moveEnd`	Перемещает конечную границу текстовой области
`moveStart`	Перемещает начальную границу текстовой области

Таблица 3.5 *(продолжение)*

Метод	Описание
`moveLight`	Перемещает источник света в графическом фильтре
`moveToBookmark`	Перемещает границы текстовой области для включения в нее закладки, определенной с помощью метода `getBookmark`
`moveToElementText`	Перемещает границы текстовой области таким образом, чтобы в нее вошел текст, находящийся в элементе Web-страницы, ссылка на который передана методу в качестве параметра
`moveToPoint`	Перемещает текстовую область к точке, координаты которой переданы в качестве параметров, и сжимает область вокруг этой точки
`Open`	Открывает документ для записи в него нового содержимого с помощью методов `write` и `writeln`
`Open`	Открывает новое окно браузера и загружает в него документ, URL которого передается в качестве параметра
`ParentElement`	Возвращает элемент Web-страницы, который является родительским по отношению к текстовой области
`pasteHTML`	Вставляет HTML-код в текущую текстовую область
`play`	Накладывает графический фильтр на содержимое Web-страницы
`PrevPage`	Отображает предыдущую страницу в таблице
`prompt`	Отображает окно ввода информации
`queryCommandEnabled`	Возвращает логическое значение, показывающее, доступна ли для выполнения команда, переданная методу в качестве параметра
`queryCommandIndeterm`	Возвращает логическое значение `True`, если команда имеет неопределенный статус
`queryCommandState`	Возвращает текущее состояние команды в настоящий момент времени
`queryCommandSupported`	Возвращает логическое значение, указывающее, поддерживается данная команда браузером или нет
`queryCommandText`	Возвращает строковое значение команды
`queryCommandValue`	Возвращает значение команды
`refresh`	Обновляет содержимое таблицы
`reload`	Перезагружает текущую Web-страницу в окне просмотра

Таблица 3.5 *(продолжение)*

Метод	Описание
`remove`	Удаляет из коллекции элемент с порядковым номером, переданным методу в качестве параметра
`removeAttribut`	Удаляет из тэга параметр-атрибут, имя которого передано методу в качестве параметра
`reset`	Имитирует нажатие пользователем кнопки **Reset**, и порождает событие `onreset`
`scroll`	Разворачивает окно просмотра браузера на заданную ширину и высоту, которые передаются методу в качестве параметра
`scrollIntoView`	Прокручивает Web-страницу в окне просмотра браузера таким образом, чтобы элемент, к которому был применен данный метод, отображался в видимой зоне окна
`select`	Делает выделенный пользователем участок содержимого Web-страницы равным текстовой области, к которой метод и применяется
`setAttribut`	Добавляет в тэг дополнительный параметр-атрибут и устанавливает его значение
`setEndPoint`	Переносит одну из граничных точек текущей текстовой области в соответствующую граничную точку другой текстовой области, которая передана методу в качестве параметра
`setInterval`	Заставляет инструкцию, переданную методу в качестве параметра, периодически выполняться. Время периода повтора также передается методу как параметр
`setTimeout`	Выполняет инструкцию один раз по истечении заданного времени
`showHelp`	Отображает окно оперативной справки
`showModalDialog`	Отображает диалоговое окно
`stop`	Принудительно прекращает динамическое действие графического фильтра
`submit`	Имитирует нажатие пользователем кнопки **Submit**
`write`	Добавляет в текущий документ HTML-код, переданный методу в качестве параметра
`writeln`	Добавляет в текущий документ HTML-код, переданный методу в качестве параметра, и заканчивает его символом возврата каретки

Таблица 3.5 *(окончание)*

Метод	Описание
zOrder	Устанавливает z-индекс для вертикального псевдопозиционирования элементов

Кроме методов объекты обладают еще и свойствами. Различные объекты имеют разные списки присущих им свойств. Соответствие тэгов HTML и свойств, присущих объектам, порождаемым этими тэгами, приведено в табл. 3.6.

Таблица 3.6

Тэг	Описание	Список поддерживаемых свойств
<a>	Гиперссылка	accessKey, className, dataFld, dataSrc, document, event, hash, host, hostname, href, id, innerText, isTextEdit, lang, language, methods, name, offsetheight, offsetLeft, offsetParent, offsetTop, offsetWidth, outerText, parentElement, parentTextEdit, pathname, port, protocol, rel, rev, search, sourceIndex, style, tagName, target, title, urn
<address>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<applet>	Внедрение в состав Web-страницы java-апплета	accessKey, align, className, code, codeBase, dataFld, dataSrc, disabled, document, hspace, id, isTextEdit, lang, language, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, src, style, tagName, title, vspace

Таблица 3.6 (продолжение)

Тэг	Описание	Список поддерживаемых свойств
<area>	Создание активной области-гиперссылки в сегментированной графике	alt, className, coords, document, event, hash, host, hostname, href, id, isTextEdit, lang, language, noHref, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, pathname, port, protocol, search, shape, sourceIndex, style, tagName, target, title
<b>	Выделение текста полужирным шрифтом	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, sourceIndex, style, tagName, title
<base>	Указание URL основного документа	className, document, href, id, isTextEdit, lang, outerHTML, outerText, parentElement, parenttextEdit, sourceIndex, tagName, target, title
<basefont>	Установка шрифта, используемого по умолчанию	className, color, document, face, id, isTextEdit, outerHTML, outerText, parentElement, parentTextEdit, size, sourceIndex, tagName, title
<bgsound>	Задание фонового звука	balance, className, document, id, isTextEdit, loop, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, src, style, tagName, title, volume
<big>	Увеличение размера шрифта на единицу	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<blockquote>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<body>	Определение содержательной части Web-страницы	accessKey, alinkColor, background, bgColor, bgProperties, bottomMargin, className, clientHeight, clientWidth, document, event, id, innerHTML, innerText, isTextEdit, lang, language, leftMargin, linkColor, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, rightmargin, scroll, scrollHeight, scrollLeft, scrollTop, scrollWidth, sourceIndex, style, tagName, text, title, topMargin, vlinkColor
 	Принудительный обрыв строки	className, clear, document, id, isTextEdit, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<button>	Создание кнопки	accessKey, className, dataFld, dataFormatAs, dataSrc, disabled, document, event, form, id, innerHTML, innerText, isTextEdit, lang, language, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, status, style, tagName, title, type, value
<caption>	Создание заголовка таблицы	align, className, clientHeght, clientWidth, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title, vAlign
	Центрирование блока содержимого	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<cite>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<code>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<col>	Определение столбца таблицы	align, className, document, event, id, isTextEdit, parentElement, parentTextEdit, span, style, tagName, title, vAlign
<colgroup>	Объявление группы столбцов таблицы	align, className, document, id, isTextEdit, parentElement, parentTextEdit, span, style, tagName, title, vAlign
<dd>	Элемент списка	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<dfn>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Объединение элементов содержимого Web-страницы	align, className, clientHeght, clientWidth, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<dl>	Создание списка	className, compact, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<dt>	Создание элемента списка	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<embed>	Внедрение в состав содержимого Web-страницы объектов различных типов	accessKey, align, className, clientHeight, clientWidth, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, recordNumber, scrollHeight, scrollLeft, scrollTop, scrollWidth, sourceIndex, style, tabIndex, tagName, title
<font>	Определение шрифта для отображения текста	className, color, document, event, face, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, size, sourceIndex, style, tagName, title

Таблица 3.6 (продолжение)

Тэг	Описание	Список поддерживаемых свойств
<form>	Определение формы в HTML-документе	action, className, document, encoding, event, id, innerHTML, innerText, isTextEdit, lang, language, method, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, target, title
<frame>	Создание отдельного фрейма	borderColor, className, document, dataFld, dataSrc, event, frameBorder, height, id, isTextEdit, lang, language, marginHeight, marginWidth, name, noResize, parentElement, parentTextEdit, sourceIndex, src, style, tagName, title
<frameset>	Определение фреймовой структуры	border, borderColor, className, cols, document, frameBorder, frameSpacing, id, isTextEdit, lang, language, parentElement, parentTextEdit, rows, sourceIndex, style, tagName, title
<head>	Объявление заголовка документа	className, document, id, isTextEdit, parentElement, sourceIndex, tagName, title
<h1> — <h6>	Создание заголовков содержимого страницы	align, className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<hr>	Отображение горизонтальной линии	align, className, color, document, event, id, isTextEdit, lang, language, noShade, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, size, sourceIndex, style, tagName, title, width
<html>	Объявление содержимого как HTML-документа	className, document, id, isTextEdit, language, parentElement, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<i>	Выделение текста курсивом	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<iframe>	Вставка плавающего фрейма	align, border, borderColor, className, dataFld, dataSrc, document, event, frameBorder, frameSpacing, height, hspace, id, innerHTML, innerText, isTextEdit, lang, language, marginHeight, marginWidth, name, noResize, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, scrolling, sourceIndex, src, style, tagName, title, vspace, width
<img>	Вставка графического изображения	align, alt, border, className, dataFld, dataSrc, document, dynsrc, event, height, hspace, id, isMap, isTextEdit, lang, language, loop, lowsrc, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, readyState, scrollHeight, scrollLeft, scrollTop, scrollWidth, src, start, sourceIndex, style, tagName, title, useMap, vspace, width
<input>	Создание элемента ввода информации	accesskey, checked, className, dataFld, dataSrc, dataFormatAs, defaultChecked, defaultValue, disabled, document, event, form, id, indeterminate, innerHTML, innerText, isTextEdit, lang, language, maxLength, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, readOnly, recordNumber, size, sourceIndex, src, status, style, tabIndex, tagName, title, type, value

Таблица 3.6 (продолжение)

Тэг	Описание	Список поддерживаемых свойств
<kbd>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<label>	Создание текстовой метки для некоторых элементов управления	accessKey, className, document, event, HTMLFor, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Создание элемента списка	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title, type, value
<link>	Объявление связи между различными HTML-документами	className, disabled, document, href, id, parentElement, readyState, rel, sourceIndex, style, tagName, title, type
<listing>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<map>	Создание карты сегментированной графики	className, document, event, id, innerHTML, innerText, isTextEdit, lang, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<meta>	Объявление мета-переменной в HTML-документе	charset, className, content, document, httpEquiv, id, isTextEdit, lang, name, parentElement, parentTextEdit, sourceIndex, tagName, title, url
<object>	Внедрение какого-либо объекта в HTML-документ	accessKey, align, classid, className, code, codeBase, codeType, data, dataFld, dataSrc, disabled, document, event, height, id, isTextEdit, lang, language, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, readyState, sourceIndex, style, tabIndex, tagName, title, type, width
	Создание нумерованного списка	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, start, style, tagName, title, type
<option>	Создание элемента выпадающего списка	className, document, event, id, isTextEdit, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, selected, sourceIndex, style, tagName, text, title. value
<p>	Определение текстового абзаца	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<plaintext>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<pre>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<s>	Отображение зачеркнутого текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<script>	Объявление программы-скрипта, включенной в состав HTML-документа	className, document, event, HTMLFor, id, innerHTML, innerText, isTextEdit, outerHTML, outerText, parentElement, parentTextEdit, readyState, sourceIndex, src, style, tagName, text, title
<select>	Создание элемента управления	accessKey, className, dataFld, dataSrc, disabled, document, event, form, id, isTextEdit, lang, language, length, multiple, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, recordNumber, selectedIndex, size, sourceIndex, style, tabIndex, tagName, title, type, value
<small>	Уменьшение размера используемого шрифта	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Объединение элементов Web-страницы	className, dataFld, dataSrc, dataFormatAs, document, event, id, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent,

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
(прод.)		offsetTop, offsetWidth, outerText, parentElement, parentTextEdit, scrollHeight, scrollLeft, scrollTop, scrollWidth, sourceIndex, style, tagName, title
<strike>	Отображение зачеркнутого текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Отображение текста полужирным начертанием шрифта	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<style>	Создание таблицы стилей отображения	className, disabled, document, id, isTextEdit, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, readyState, sourceIndex, style, tagName, title
<sub>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
<sup>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<table>	Создание таблицы	align, background, bgColor, border, borderColor, borderColorDark, borderColorLight, cellPadding, cellSpacing, className, clientHeight, clientWidth, cols, dataFld, dataPageSize, dataSrc, document, event, frame, height, id, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerText, parentElement, parentTextEdit, rules, scrollHeight, scrollLeft, scrollTop, scrollWidth, sourceIndex, style, tagName, title, width
<tbody>	Обозначение основного раздела таблицы	align, bgColor, className, document, event, id, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, sourceIndex, style, tagName, title, vAlign
<td>	Объявление ячейки таблицы	align, background, bgColor, borderColor, borderColorDark, borderColorLight, className, clientheight, clientWidth, colSpan, document, event, height, id, innerText, isTextEdit, lang, language, noWrap, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, rowSpan, sourceIndex, style, tagName, title, vAlign, width
<textarea>	Создание многострочного поля текстового ввода	accesskey, className, clientHeight, clientWidth, cols, dataFld, dataSrc, disabled, document, event, form, id, innerText, isTextEdit, lang, language, name, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerText, parentElement, parentTextEdit, readOnly, rows, scrollHeight, scrollLeft, scrollTip, scrollWidth, sourceIndex, status, style, tabIndex, tagName, title, type, value, wrap

Таблица 3.6 *(продолжение)*

Тэг	Описание	Список поддерживаемых свойств
<tfoot>	Создание нижней части таблицы	align, bgColor, className, document, event, id, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, sourceIndex, style, tagName, title, vAlign
<th>	Объявление строки заголовка таблицы	align, background, bgcolor, borderColor, borderColorDark, borderColorLight, className, colSpan, document, event, id, isTextEdit, lang, language, noWrap, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, rowSpan, sourceIndex, style, tagName, title, vAlign
<thead>	Создание области заголовка таблицы	align, bgColor, className, document, event, id, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, sourceIndex, style, tagName, title, vAlign
<title>	Определение заголовка документа	className, document, id, isTextEdit, lang, parentElement, parentTextEdit, sourceIndex, tagName, title
<tr>	Объявление строки таблицы	align, bgColor, borderColor, borderColorDark, borderColorLight, className, document, event, id, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, parentElement, parentTextEdit, sourceIndex, style, tagName, title, vAlign
<tt>	Специализированное шрифтовое оформление	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

Таблица 3.6 (окончание)

Тэг	Описание	Список поддерживаемых свойств
<u>	Отображение подчеркнутого текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title
	Создание маркированного списка	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title, type
<var>	Специализированное форматирование текста	className, document, event, id, innerHTML, innerText, isTextEdit, lang, language, offsetHeight, offsetlLeft, offsetparent, offsetTop, offsetWidth, outerHTML, outerText, parentElement, parentTextEdit, sourceIndex, style, tagName, title

В этой таблице мы установили соответствие между тэгами и свойствами элементов оформления Web-страниц, которые создаются с помощью этих тэгов. Легко заметить, что часто наименования свойств повторяют наименования параметров-атрибутов, входящих в состав тэгов. Естественно, значениями этих свойств будут значения параметров тэгов, записанные в коде HTML-документа. Среди них встречаются аналоги элементов CSS. Но есть и иные свойства, с которыми мы не успели познакомиться при рассмотрении объектной модели браузера. В табл. 3.7 приведен список всех свойств, входящих в объектную модель DHTML с их краткой характеристикой.

Таблица 3.7

Свойство	Описание
accessKey	Задает "горячую клавишу" для быстрой установки фокуса ввода на элемент
action	Указывает адрес приложения или документа, обрабатывающего данные, передаваемые формой

Таблица 3.7 *(продолжение)*

Свойство	Описание
activeElement	Содержит идентификатор активного элемента
align	Задает выравнивание элемента
alinkColor	Указывает цвет отображения активных ссылок на странице
alt	Задает текст, заменяющий графическое изображение
altKey	Указывает состояние клавиши <Alt>
appCodeName	Содержит кодовое наименование движка браузера, в котором отображается текущая Web-страница
appName	Хранит краткое наименование браузера, в котором отображается текущая Web-страница
appVersion	Указывает номер версии используемого пользователем браузера
background	Задает графическое изображение или цвет, используемые в качестве фона для данного элемента
backgroundAttachment	Задает алгоритм отображения фоновой картинки при прокручивании содержимого Web-страницы в окне просмотра браузера
backgroundColor	Указывает фоновый цвет элемента
backgroundImage	Содержит URL графического файла с изображением, использующимся в качестве фона для элемента
backgroundPosition	Задает координаты расположения фонового изображения
backgroundPositionX	Содержит горизонтальную координату левого верхнего угла фонового изображения относительно левого верхнего угла элемента, содержащего его
backgroundPositionY	Содержит вертикальную координату левого верхнего угла фонового изображения относительно левого верхнего угла элемента, содержащего его
backgroundRepeat	Задает алгоритм повтора фоновой картинки
balance	Указывает соотношение громкости левого и правого каналов для фоновой музыки
bgColor	Задает цвет фона элемента
bgProperties	Предоставляет доступ к свойствам фонового изображения

Таблица 3.7 *(продолжение)*

Свойство	Описание
body	Содержит весь HTML-код, заключенный между тэгами <body> и </body>. Имеет статус "только для чтения"
border	Задает тип рамки вокруг элемента
borderBottom	Задает тип нижней границы элемента
borderBottomColor	Указывает цвет нижней границы элемента
borderBottomStyle	Устанавливает стиль отображения нижней границы элемента
borderBottomWidth	Задает ширину нижней границы элемента
borderColor	Определяет цвет рамки вокруг элемента
borderColorDark	Устанавливает цвет "темной половины" рамки, т. е. нижней и правой границы
borderColorLight	Содержит цвет "светлой половины" рамки, т. е. верхней и левой границы
borderLeft	Задает тип левой границы элемента
borderLeftColor	Указывает цвет левой границы элемента
borderLeftStyle	Устанавливает стиль отображения левой границы элемента
borderLeftWidth	Задает ширину левой границы элемента
borderRight	Устанавливает тип правой границы элемента
borderRightColor	Указывает цвет правой границы элемента
borderRightStyle	Устанавливает стиль отображения правой границы элемента
borderRightWidth	Задает ширину правой границы элемента
borderStyle	Устанавливает стиль отображения всех четырех границ элемента
borderTop	Задает тип верхней границы элемента
borderTopColor	Указывает цвет верхней границы элемента
borderTopStyle	Устанавливает стиль отображения верхней границы элемента
borderTopWidth	Задает ширину верхней границы элемента
borderWidth	Устанавливает ширину всех четырех границ элемента

Таблица 3.7 *(продолжение)*

Свойство	Описание
bottomMargin	Задает нижнее поле элемента
button	Описывает состояние кнопок мыши при возникновении соответствующих событий
cancelBubble	Регулирует передачу события вверх по объектной иерархии
cellPadding	Задает расстояние между содержимым ячейки и ее границей
cellSpacing	Устанавливает расстояние между ячейками в таблице
charset	Содержит наименование используемой кодировки символов
checked	Указывает, что тот или иной независимый переключатель или радиокнопка отмечены пользователем
classid	Содержит идентификатор класса объекта, применяемого для установки связи с одним из селекторов используемой стилевой таблицы
className	Задает класс (а не идентификатор) стилевой таблицы, связанной с данным тэгом
clear	Устанавливает более точное позиционирование текста после графического объекта
client	Возвращает объект navigator для браузера
clientHeight	Содержит высоту элемента без служебных элементов (поля, полосы прокрутки, границы и пр.)
clientWidth	Содержит ширину элемента, исключая служебные элементы
clientX	Возвращает горизонтальную координату элемента без служебных дополнений
clientY	Возвращает вертикальную координату элемента без служебных дополнений
clip	Определяет порядок отображения содержимого, выходящего за пределы отображаемой области элемента
closed	Указывает, закрыто ли текущее окно просмотра
codeBase	Содержит URL кода внедряемого объекта

Таблица 3.7 *(продолжение)*

Свойство	Описание
codeType	Указывает тип кода для внедряемого объекта
color	Задает цвет элемента
colorDepth	Указывает число бит на символ, которое регулирует количество отображаемых цветов
cols	Содержит число столбцов в таблице или наборе фреймов
colSpan	Определяет число столбцов, объединенных данной ячейкой таблицы
compact	Задает компактную модель отображения списков
complete	Сигнализирует о полной загрузке содержимого страницы
content	Хранит содержимое метапеременной пользователя или заголовка протокола HTTP в тэге <meta>
cookie	Содержит строку cookie-информации, хранящейся в локальной системе удаленного пользователя
cookieEnabled	Определяет, разрешает ли браузер пользователя использование cookie-информации
coords	Содержит координаты углов активной области сегментированной графики
cssText	Содержит значение параметра style для искомого тэга
ctrlKey	Определяет состояние клавиши <Ctrl> при обработке событий
cursor	Определяет внешний вид курсора мыши
data	Содержит URL объекта, содержащего данные, отображаемые элементом. Применяется в динамическом связывании данных. Очень часто используются элементы ActiveX, содержащие набор данных, который отображается таблицей. Этот метод возвращает URL источника данных
dataFld	Определяет поле источника данных, данные из которого отображаются в элементе
dataFormatAs	Указывает формат данных, получаемых из источника
dataSrc	Указывает источник данных, отображаемых элементом

Таблица 3.7 *(продолжение)*

Свойство	Описание
defaultChecked	Содержит логическое значение, указывающее, находится ли объект в исходном состоянии, определяемом по умолчанию
defaultStatus	Устанавливает строку, отображаемую в строке статуса браузера по умолчанию
defaultValue	Содержит текст, отображаемый в элементе по умолчанию
dialogArguments	Содержит список аргументов, передаваемых диалоговому окну
dialogHeight	Содержит высоту диалогового окна
dialogLeft	Задает горизонтальную координату верхнего левого угла диалогового окна
dialogTop	Задает вертикальную координату верхнего левого угла диалогового окна
dialogWidth	Содержит ширину диалогового окна
disabled	Содержит логическое значение, указывающее, является ли доступным данный элемент
display	Указывает, будет ли отображаться данный элемент
document	Свойство объекта window, ссылающееся на объект document
domain	Содержит доменное имя сайта, с которого был загружен текущий элемент
duration	Задает временной интервал смены графического фильтра
dynsrc	Задает URL видеоклипа, отображаемого в HTML-документе
encoding	Содержит тип кодировки, в которой отправляются данные из формы на сервер
event	Хранит ссылку на объект event
face	Задает вид шрифта, используемого для отображения текста
fgcolor	Определяет цвет, которым отображается содержимое элемента
filter	Задает применяемый к данному элементу графический фильтр

Таблица 3.7 *(продолжение)*

Свойство	Описание
font	Содержит перечень атрибутов применяемого шрифта
fontFamily	Устанавливает наименования используемых шрифтов
fontSize	Задает размер используемого шрифта
fontStyle	Указывает начертание применяемого шрифта
fontVariant	Задает способ отображения символов шрифта
fontWeight	Устанавливает ширину символов применяемого шрифта
form	Указывает на форму, в которой находится элемент
frame	Ссылается на одноименный объект
frameBorder	Устанавливает вид границ фрейма
frameSpacing	Задает размер отступа между фреймами
fromElement	Содержит имя элемента, с которого ушел курсор мыши при инициировании соответствующих событий
hash	Хранит часть URL, находящуюся после символа "#"
height	Содержит высоту элемента или разрешение экрана удаленного пользователя в пикселах
hidden	Указывает, является ли данный элемент скрытым или нет
history	Ссылается на одноименный объект
host	Содержит доменное имя и номер порта сервера, с которого был загружен данный документ
hostname	Содержит доменное имя сайта, с которого получен HTML-документ
href	Содержит полный URL HTML-документа, загруженного в данное окно просмотра
hspace	Задает величину горизонтального отступа между текущим и соседними элементами
htmlFor	Ссылается на элемент, с которым связан выполняемый скрипт
htmlText	Возвращает HTML-код содержимого объекта TextRange

Таблица 3.7 *(продолжение)*

Свойство	Описание
id	Содержит уникальный идентификатор элемента
indeterminate	Указывает, является ли недоступным для использования элемент ввода данных
innerHTML	Содержит HTML-код, находящийся между открывающим и закрывающим тэгами данного элемента
innerText	Содержит текст, находящийся между открывающим и закрывающим тэгами данного элемента
isTextEdit	Содержит логическое значение, указывающее, может ли данный элемент служить в качестве базы для создания на его основе объекта TextRange
keyCode	Содержит код нажатой клавиши
lang	Хранит наименование применяемого языка программирования скриптов по версии ISO
language	Содержит наименование применяемого языка программирования скриптов
lastModified	Содержит текстовую строку, в которой указывается дата последнего изменения содержимого страницы
left	Задает горизонтальную координату верхнего левого угла элемента
leftMargin	Задает величину левого поля для данного элемента
length	Указывает количество элементов в коллекции
letterSpacing	Задает величину межбуквенного интервала
lineHeight	Устанавливает межстрочный отступ
linkColor	Содержит цвет обычных гиперссылок, размещенных на Web-странице
listStyle	Описывает стиль отображения списка
listStyleImage	Задает графическое изображение, использующееся как маркер списка
listStylePosition	Задает расположение маркеров списка
listStyleType	Устанавливает вид маркеров или нумераторов списка
location	Содержит URL документа, отображаемого в окне просмотра

Таблица 3.7 *(продолжение)*

Свойство	Описание
`loop`	Задает число повторов воспроизведения фоновой музыки
`lowsrc`	Содержит URL альтернативного графического изображения с низким разрешением
`map`	Создает карту сегментированной графики
`margin`	Задает размеры полей для данного элемента
`marginBottom`	Устанавливает размеры нижнего поля для элемента
`marginHeight`	Задает величину верхнего и нижнего поля
`marginLeft`	Устанавливает величину левого поля элемента
`marginRight`	Указывает размер правого поля элемента
`marginTop`	Задает размер верхнего поля элемента
`marginWidth`	Устанавливает размеры левого и правого полей элемента
`maxLength`	Задает максимально возможное количество символов в поле текстового ввода
`method`	Определяет метод пересылки данных из формы в обрабатывающую программу
`mimeTypes`	Хранит ссылку на коллекцию типов MIME, которые поддерживает данный браузер
`multiple`	Регулирует возможность выбора пользователем в списке сразу нескольких элементов
`name`	Задает наименование элемента
`navigator`	Ссылается на объект `navigator`
`noResize`	Определяет, возможно ли изменение размеров объекта
`noShade`	Определяет, будет ли горизонтальная линия отображаться с тенью или без нее
`noWrap`	Указывает, разрешено ли браузеру переносить текст на другие строки, если тот в виде одной строки выходит за предопределенные размеры элемента
`object`	Содержит ссылку на объект, внедренный в состав содержимого Web-страницы
`offsetHeight`	Задает высоту содержимого элемента, включая и ту его часть, которая может быть не видна в данный момент

Таблица 3.7 *(продолжение)*

Свойство	Описание
offsetLeft	Указывает горизонтальную координату верхнего левого угла содержимого элемента даже в том случае, если в текущий момент этот угол не виден и доступ к нему может быть осуществлен с помощью полос прокрутки
offsetParent	Указывает координаты элемента, являющегося родительским, по отношению к текущему
offsetTop	Указывает вертикальную координату верхнего левого угла содержимого элемента даже в том случае, если в текущий момент этот угол не виден и доступ к нему может быть осуществлен с помощью полос прокрутки
offsetWidth	Содержит ширину содержимого элемента, включая и ту его часть, которая может быть не видна в данный момент
offsetX	Указывает горизонтальную координату курсора мыши относительно левого верхнего угла элемента, в котором он находился в момент возникновения обрабатываемого события
offsetY	Указывает вертикальную координату курсора мыши относительно левого верхнего угла элемента, в котором он находился в момент возникновения обрабатываемого события
opener	Содержит ссылку на окно, из которого было открыто текущее окно просмотра
outerHTML	Содержит HTML-код всего элемента, включая открывающий и закрывающий тэги
outerText	Предоставляет доступ к текстовому содержимому элемента, включая стартовый и закрывающий тэги
overflow	Определяет порядок отображения текста, выходящего за пределы элемента, в котором он содержится
padding	Задает расстояние между содержимым элемента и его рамкой
paddingBottom	Устанавливает расстояние между содержимым элемента и нижней его границей
paddingLeft	Устанавливает расстояние между содержимым элемента и левой его границей

Таблица 3.7 *(продолжение)*

Свойство	Описание
paddingRight	Устанавливает расстояние между содержимым элемента и правой его границей
paddingTop	Устанавливает расстояние между содержимым элемента и его верхней границей
pageBreakAfter	Создает разрыв текстового раздела на странице после текущего элемента
pageBreakBefore	Создает разрыв текстового раздела на странице перед текущим элементом
palette	Ссылается на цветовую палитру, установленную для отображения внедренного объекта
parent	Хранит ссылку на родительский фрейм или окно просмотра
parentStyleSheet	Указывает на родительскую таблицу стилевого оформления
parentTextEdit	Ссылается на родительский объект, который может быть использован в качестве базы для создания объекта TextRange
parentWindow	Ссылается на родительское окно просмотра
pathname	Задает имя файла, расположенное в URL после доменного имени сайта
pixelHeight	Содержит высоту элемента в пикселах
pixelLeft	Задает горизонтальную координату левого верхнего угла элемента
pixelTop	Содержит вертикальную координату левого верхнего угла элемента
pixelWidth	Содержит ширину элемента в пикселах
plugins	Задает ссылку на коллекцию подключенных к браузеру внешних модулей, предназначенных для отображения встроенных объектов
port	Хранит номер порта в URL документа, на котором функционирует Web-сервер, получивший данный документ
posHeight	Содержит высоту элемента в единицах измерения, которые использовались последний раз
position	Определяет тип позиционирования элемента

Таблица 3.7 *(продолжение)*

Свойство	Описание
posLeft	Содержит горизонтальную координату верхнего левого угла элемента в единицах измерения, которые использовались последний раз
posTop	Содержит вертикальную координату верхнего левого угла элемента в единицах измерения, которые использовались последний раз
posWidth	Содержит ширину элемента в единицах измерения, которые использовались последний раз
protocol	Хранит стартовую часть URL, в которой задан протокол, использовавшийся для получения данного документа
readOnly	Определяет, что содержимое элемента предназначается только для чтения и не может быть модифицировано
readyState	Указывает текущее состояние загружаемого объекта
reason	Содержит значение, сигнализирующее об успешности загрузки содержимого элемента
recordNumber	Содержит номер записи в таблице, связанной с источником данных
recordSet	Ссылается на набор записей в элементе, если тот связан с каким-либо источником данных
ref	Содержит значение, показывающее, является ли данный элемент URL-адресом
referrer	Хранит URL-адрес документа, по ссылке из которого был загружен текущий документ
rel	Определяет назначение документа, на который указывает элемент-гиперссылка
returnValue	Задает возвращаемое значение
rows	Задает число строк в таблице или наборе фреймов
rowSpan	Содержит число строк таблицы, объединяемых ячейкой
screen	Ссылается на объект screen
screenX	Содержит горизонтальную координату курсора мыши в пикселах, относительно самого экрана
screenY	Содержит вертикальную координату курсора мыши в пикселах, относительно самого экрана

Таблица 3.7 *(продолжение)*

Свойство	Описание
scroll	Управляет отображением полос прокрутки
scrollHeight	Задает высоту видимого содержимого элемента в пикселах
scrolling	Управляет возможностью прокручивания содержимого фрейма в окне просмотра
scrollLeft	Содержит расстояние в пикселах от левого края содержимого элемента до левого края видимой области элемента
scrollTop	Содержит расстояние в пикселах от верхней границы содержимого элемента до верхнего края видимой области элемента
scrollWidth	Задает ширину видимого содержимого элемента в пикселах
search	Хранит строку запроса в URL, расположенную после знака "?"
selected	Указывает, что данный элемент выпадающего списка является выбранным по умолчанию
selectedIndex	Содержит число, являющееся порядковым номером выбранного элемента в списке
selection	Ссылается на одноименный объект selection
self	Ссылается на текущее окно просмотра
shape	Устанавливает форму активной области сегментированной графики
shiftKey	Указывает состояние клавиши <Shift> в момент обработки события
size	Задает размер элемента
sourceIndex	Содержит число, являющееся порядковым номером элемента в коллекции all
span	Определяет количество столбцов таблицы, объединяемых элементом colgroup
src	Задает URL внешнего файла, в котором находится содержимое элемента
srcElement	Указывает элемент, в котором было изначально инициировано обрабатываемое событие
srcFilter	Показывает, какой фильтр инициировал событие onfilterchange

Таблица 3.7 *(продолжение)*

Свойство	Описание
start	Указывает номер первого элемента списка
status	Содержит текст, отображающийся в строке статуса браузера
style	Определяет стиль, применяемый к элементу
styleFloat	Задает порядок обтекания элементом иных объектов содержимого Web-страницы
tabIndex	Содержит порядковый номер элемента в последовательности элементов управления и ввода информации, перемещение между которыми осуществляется с помощью клавиши табуляции
tagName	Хранит наименование тэга, с помощью которого реализуется текущий элемент
target	Указывает наименование окна или фрейма, в котором будет отображаться HTML-документ
text	Устанавливает цвет, которым будет отображаться текст
textAlign	Задает выравнивание текста
textDecoration	Устанавливает эффект отображения текста
textDecorationLineThrough	Указывает механизм зачеркивания текста
textIndent	Задает отступ для первой строки абзаца
textTransform	Задает порядок отображения символов текста
title	Содержит краткое наименование или описание элемента, которое обычно отображается в виде подсказки-хинта
toElement	Указывает наименование элемента, на который попадает курсор мыши при возникновении событий onmouseover и onmouseout
top	Задает вертикальную координату верхней границы элемента
topMargin	Устанавливает размер верхнего поля элемента
type	Устанавливает тип создаваемого элемента ввода данных
updateInterval	Сообщает частоту обновлений экрана системы удаленного пользователя
url	Содержит URL документа

Таблица 3.7 *(окончание)*

Свойство	Описание
userAgent	Хранит HTTP-заголовок, который инициирует обмен данными между удаленным пользователем и сервером
vAlign	Определяет вертикальное выравнивание элемента
value	Содержит текст или значение, отображаемое по умолчанию в элементах управления и ввода информации
verticalAlign	Управляет вертикальным выравниванием элемента с помощью средств CSS
visibility	Определяет видимость элемента
vlinkColor	Задает цвет гиперссылок, посещенных данным удаленным пользователем
vspace	Устанавливает размер отступа по вертикали между соседними элементами
width	Задает ширину элемента
width	Указывает горизонтальное разрешение экрана системы удаленного пользователя
window	Содержит ссылку на окно просмотра, в котором отображается данный HTML-документ
wrap	Определяет порядок разрыва строк в случаях, когда те по своим размерам превосходят элемент, в котором они содержатся
x	Показывает горизонтальную координату курсора мыши в момент обработки какого-либо события
y	Содержит вертикальную координату курсора мыши в момент обработки какого-либо события
z-index	Хранит координату вертикального псевдопозиционирования элемента

Таким образом, все таблицы, которые были приведены в данном разделе главы, составляют единый блок, полностью описывающий возможности обработки любых тэгов HTML-документов с помощью технологии DHTML.

Использование стилей

Мы уже знаем, что использование каскадных таблиц стилей позволяет нам добиваться самых поразительных эффектов в оформлении содержимого Web-страниц. Когда это мощное средство оформления Web-страниц соеди-

няется с технологией DHTML, разработчик получает в свое распоряжение практически неограниченные возможности манипулирования содержимым Web-страницы. В этом разделе мы рассмотрим основные приемы применения правил отображения элементов Web-страниц с помощью стилей.

Существует два способа применения стилей. Можно создать стилевую таблицу, в данном случае не так уж важно, будет она встроенной или внешней, а затем обращаться к правилам отображения этой таблицы с помощью значений параметров `class`, которые будут совпадать с теми или иными селекторами стилевой таблицы. А можно воспользоваться параметром `style`, который встраивается практически во все тэги. Напоминаю, что создание отдельной стилевой таблицы оправдано в тех случаях, когда необходимо создать единообразное оформление. Если же нам требуется более гибко управлять параметрами отображения многих элементов, стоит использовать задание стиля для каждого отдельного тэга.

Для начала мы рассмотрим примеры операций с отдельными стилевыми таблицами. Существует два подхода к данной проблеме. Мы можем либо динамически изменять подключенную стилевую таблицу, либо менять идентификаторы классов у элементов оформления Web-страниц.

Изменение идентификаторов классов обычно используется в тех случаях, когда все необходимые правила отображения уже описаны в стилевой таблице, связанной с HTML-документом, а разработчику необходимо лишь гибко применять различные правила к одному и тому же элементу содержимого. Приведем один из примеров подобного подхода.

Мы можем создать документ, основным содержимым которого является таблица с тремя столбцами и одной строкой. В первой ячейке мы расположим два наименования коротких заметок, связанных с радиокнопками. Сами заметки мы разместим во второй и третьей ячейке. А пользователь, выбирая ту или иную радиокнопку, будет управлять выводом соответствующего текста. Таким образом, когда отображается содержимое второй ячейки, не отображается содержимое третьей ячейки, и наоборот. Код подобного документа приведен в листинге 3.3. Результат отображения этого HTML-документа показан на рис. 3.3.

Листинг 3.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Динамические стили</title>
  <script language="javascript">
  <!--
  function chan_cont(x)
```

```
  {
  if (x==1) {p1.className="visible"; p2.className="hidden"};
  if (x==2) {p2.className="visible"; p1.className="hidden"};

  }
  //-->
  </script>
  <style type="text/css">
   .visible {visibility:visible}
   .hidden {visibility:hidden}
  </style>
  </head>
  <body>
  <table border=2 id="table1">
  <tr>
   <td>
   <p>Первая заметка<input type="radio" name="group1"
onClick="chan_cont(1)" value="1" checked></p>
   <p>Вторая заметка<input type="radio" name="group1"
onClick="chan_cont(2)" value="2"></p>
   </td>
   <td>
   <p id="p1" class="visible">Текст первой заметки</p>
   </td>
   <td>
   <p id="p2" class="hidden">Текст второй заметки</p>
   </td>
  </tr>
  </table>
  </body>
</html>
```

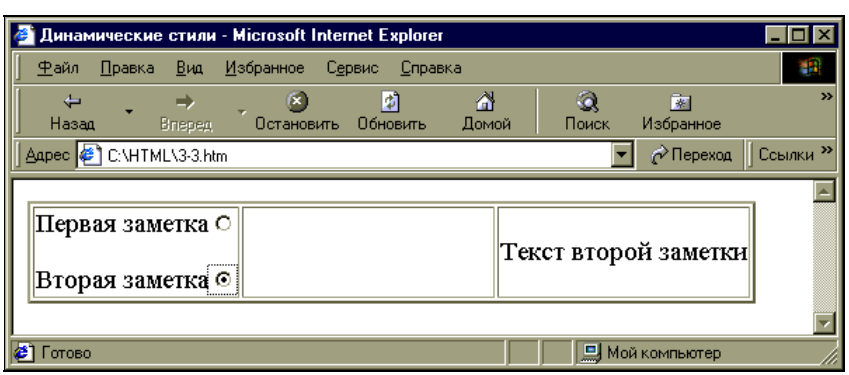


Рис. 3.3. Окно браузера с результатом отображения файла, приведенного в листинге 3.3, в тот момент, когда пользователь активизировал вторую радиокнопку

Теперь рассмотрим структуру HTML-кода созданной нами Web-страницы. Функционально код разбивается на три части: программа-скрипт, стилевая таблица и содержимое документа. Начнем мы со стилевой таблицы.

Легко заметить, что мы объявляем два класса с селекторами `visible` и `hidden`. Правила отображения для этих классов чрезвычайно просты. В обоих случаях мы явно устанавливаем видимость элементов, входящих в тот или иной класс с помощью свойства `visibility`, которое отвечает за видимость элемента данного класса. Соответственно, класс `visible` будет отображаться в окне просмотра, а класс `hidden` — нет.

Содержимое страницы полностью "упаковано" в таблицу (рис. 3.3). В первой ячейке размещены две радиокнопки. Так как нам не придется передавать данные об их состоянии обрабатывающей программе на сервер, мы с успехом обходимся без формы, создаваемой с помощью тэга `<form>`. Для каждой радиокнопки мы установили обработку события `onClick`, возникающего при щелчке кнопкой мыши по данной радиокнопке. Так как щелчок кнопкой мыши по радиокнопке вызывает изменение ее состояния, то, казалось бы, можно для этих целей воспользоваться и событием `onChange`. Здесь решающую роль играет тот факт, что событие `onClick` передается вверх по объектной иерархии до объекта `document`, а `onChange` — нет. Вверх по объектной иерархии передаются только те события, которые есть одновременно у всех объектов иерархии. Кроме того, событие `onClick` для объекта `document` инициирует его обновление в окне просмотра браузера, если содержимое данного объекта было изменено каким-либо образом. Следовательно, если бы мы воспользовались событием `onChange`, то видимые изменения содержимого документа произошли бы только после того, как пользователь дополнительно бы щелкал кнопкой мыши где-либо, кроме радиокнопок.

Для обработки щелчка кнопкой мыши на обеих радиокнопках мы используем одну и ту же функцию с наименованием `chan_cont`, которая объявлена в программе-скрипте. В качестве параметра для этой функции мы передаем порядковый номер заметки, которую необходимо сделать видимой. Затем в теле функции, в зависимости от значения переданного параметра, мы меняем имена классов у заметок, пользуясь их уникальными идентификаторами, созданными с помощью параметров `id`. Для того чтобы изменить наименование класса, мы пользуемся свойством `className`.

Впрочем, можно использовать более радикальный вариант динамического стилеобразования. У нас есть возможность создать несколько полноценных стилевых таблиц, внедренных в HTML-документ, а затем в зависимости от обстоятельств менять их. Очень часто этой возможностью пользуются для создания так называемых "скинов" сайта. То есть, для всех Web-страниц сайта создается одинаковый набор стилевых таблиц, которые полностью определяют их внешний вид. А затем удаленному пользователю предоставляется возможность выбрать именно тот вариант оформления сайта, который

ему больше всего нравится. Идентификатор соответствующей стилевой таблицы записывается в строку cookie-информации. Затем при следующем посещении сайта скрипт получает эту cookie-информацию, записанную в локальной системе удаленного пользователя, и на ее основе активизирует ту или иную стилевую таблицу. При создании подобного сценария следует учесть тот факт, что многие браузеры могут запретить создание cookie-записей на машине пользователя, следовательно, по умолчанию должна использоваться одна из стандартных стилевых таблиц.

Попробуем создать подобный документ. Мы не будем встраивать в него обработку cookie-информации, т. к. для ее создания и записи надо описывать специализированную форму. Мы ограничимся прямым выбором того или иного варианта оформления Web-страницы с помощью радиокнопок. Код подобного HTML-документа приведен в листинге 3.4, а внешний вид документа после выбора радиокнопки — на рис. 3.4.

Листинг 3.4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Динамические стили</title>
  <style id="style1" type="text/css">
   h1 {color:red}
   p {font-family:"Courier New"}
  </style>
  <style id="style2" type="text/css">
   h1 {color:blue}
   p {font-family:"Times New Roman"}
  </style>
  <script language="javascript">
  <!--
  function chan_style(x)
  {
  if (x==1) {style1.disabled=true; style2.disabled=false};
  if (x==2) {style2.disabled=true; style1.disabled=false};

  }
  //-->
  </script>

  </head>
  <body>
   <p>Первый стиль оформления<input type="radio" name="group1"
↳onClick="chan_style(1)" value="1" checked></p>
```

```
   <p>Второй стиль оформления<input type="radio" name="group1"
↳onClick="chan_style(2)" value="2"></p>
   <h1>Заголовок первого уровня</h1>
   <p>Пример обычной строки текста</p>
  </body>
</html>
```

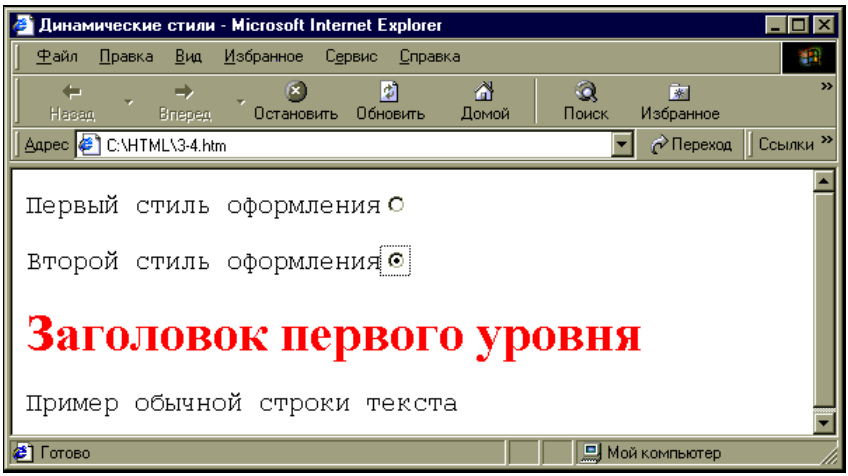


Рис. 3.4. Окно браузера с результатом отображения файла, приведенного в листинге 3.4, в тот момент, когда пользователь активизировал вторую радиокнопку

Структура документа практически не претерпела изменений по сравнению с предыдущим примером. Только теперь мы использовали два тэга `<style>` с двумя взаимоисключающими наборами правил отображения обычного текста и заголовков первого уровня. Для каждого такого тэга мы задали уникальный идентификатор. При загрузке HTML-документа браузер сначала использует вторую стилевую таблицу, т. к. она полностью перекрывает область действия первой стилевой таблицы, а загружена после нее.

После изменения состояния радиокнопок начинает действовать программа-скрипт. Для того чтобы активизировать одну стилевую таблицу и отключить другую, используется свойство `disabled`, присущее тэгам `<style>`. Его значением, как хорошо видно в тексте, является логическое выражение. В зависимости от переданного в функцию значения, мы отключаем одну стилевую таблицу и подключаем другую.

В самом начале данного раздела мы уже говорили, что можем воспользоваться объектом `style`, который входит в стандартную объектную иерархию DHTML. Этот объект присущ всем тэгам в качестве встроенного свойства. Рассмотрим простейший пример его использования (листинг 3.5 и рис. 3.5).

Листинг 3.5

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Динамические стили</title>
  <script language="javascript">
  <!--
  function chan_style()
  {
  p1.style.backgroundColor="blue";
  p1.style.color="white";
  }
  //-->
  </script>
  </head>
  <body>
   <p id="p1" onMouseover="chan_style()">Пример обычной строки текста</p>
  </body>
</html>
```

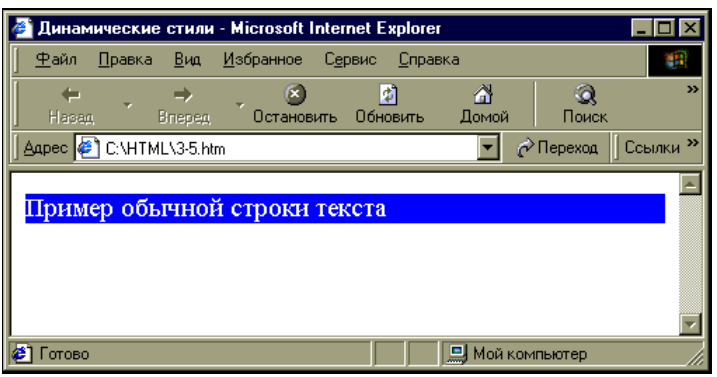


Рис. 3.5. Окно браузера с результатом отображения файла, приведенного в листинге 3.5, в тот момент, когда пользователь навел курсор мыши на текстовую строку

Механизм подключения скрипта к единственному абзацу, расположенному на нашей демонстрационной Web-странице (рис. 3.5), уже должен быть вполне понятен. Единственное место в данном HTML-документе, которое следует несколько внимательнее рассмотреть, — это тело функции, обрабатывающей событие `onMouseover`. В ее коде видно, что мы воспользовались объектом `style`, а точнее, его несколькими свойствами. По аналогии с технологией CSS легко догадаться, что свойство `backgroundColor` устанавливает

цвет фона, а свойство `color` — цвет шрифта. Однако хотелось бы точно знать, какими свойствами обладает объект `style` и какие параметры CSS они дублируют. В табл. 3.8 перечислены все свойства объекта `style`.

Таблица 3.8

Свойство	Описание
`background`	Задает URL графического изображения, использующегося как фон для элемента
`backgroundAttachment`	Определяет механизм прокручивания фонового изображения вместе с содержимым элемента
`backgroundColor`	Задает цвет фона
`backgroundImage`	Содержит URL графического изображения, используемого в качестве фона
`backgroundPosition`	Определяет координаты левого верхнего угла фонового изображения
`backgroundPositionX`	Устанавливает горизонтальную координату верхнего левого угла фонового рисунка
`backgroundPositionY`	Устанавливает вертикальную координату верхнего левого угла фонового рисунка
`backgroundRepeat`	Задает механизм повторения фонового изображения на пространстве, занимаемом элементом, если это пространство больше по размерам, чем применяемое графическое изображение
`border`	Задает стиль отображения границы вокруг элемента
`borderBottom`	Определяет параметры отображения нижней границы элемента
`borderBottomColor`	Задает цвет нижней границы элемента
`borderBottomStyle`	Задает стиль линии нижней границы элемента
`borderBottomWidth`	Устанавливает ширину нижней границы элемента
`borderColor`	Задает цвет границы элемента
`borderLeft`	Определяет параметры отображения левой границы элемента
`borderLeftColor`	Задает цвет левой границы элемента
`borderLeftStyle`	Задает стиль линии левой границы элемента
`borderLeftWidth`	Устанавливает ширину левой границы элемента
`borderRight`	Определяет параметры отображения правой границы элемента

Таблица 3.8 *(продолжение)*

Свойство	Описание
borderRightColor	Задает цвет правой границы элемента
borderRightStyle	Задает стиль линии правой границы элемента
borderRightWidth	Устанавливает ширину правой границы элемента
borderStyle	Указывает стиль линий границы элемента
borderTop	Определяет параметры отображения верхней границы элемента
borderTopColor	Задает цвет верхней границы элемента
borderTopStyle	Задает стиль линии верхней границы элемента
borderTopWidth	Устанавливает ширину верхней границы элемента
borderWidth	Указывает ширину границы элемента
clear	Указывает порядок позиционирования и выравнивания содержимого элемента
clip	Задает порядок отображения содержимого элемента, если оно по размерам превосходит пространство, отведенное элементу в окне просмотра браузера
color	Задает цвет элемента
cssText	Содержит текстовое значение атрибута `style`, внедренного в описываемый тэг
cursor	Задает вид курсора мыши, который будет отображаться в то время, когда мышь находится над элементом
display	Определяет, будет или нет отображаться данный элемент в окне просмотра браузера
filter	Указывает на набор всех графических фильтров, которые применялись к данному элементу
font	Задает свойства шрифта, которым будет отображаться текстовое содержимое элемента
fontFamily	Устанавливает шрифт, которым будет отображаться текст
fontSize	Задает размер используемого шрифта
fontStyle	Указывает начертание применяемого шрифта
fontVariant	Задает способ отображения строчных символов
fontWeight	Устанавливает ширину линий, которыми отображаются символы шрифта

Таблица 3.8 *(продолжение)*

Свойство	Описание
height	Задает высоту элемента
left	Устанавливает горизонтальную координату верхнего левого угла элемента
letterSpacing	Задает межсимвольное расстояние
lineHeight	Устанавливает межстрочный интервал
listStyle	Указывает способ отображения элементов списка
listStyleImage	Задает графическое изображение для создания маркера
listStylePosition	Устанавливает расположение маркеров элементов списка
listStyleType	Указывает, какой тип стандартных маркеров будет использоваться в данном элементе
margin	Определяет внешний вид полей элемента
marginBottom	Задает размер нижнего поля элемента
marginLeft	Задает размер левого поля элемента
marginRight	Задает размер правого поля элемента
marginTop	Задает размер верхнего поля элемента
overflow	Определяет правило отображения содержимого элемента, переполняющего выделенное для него пространство
paddingBottom	Определяет размер отступа между содержимым элемента и его нижней границей
paddingLeft	Определяет размер отступа между содержимым элемента и его левой границей
paddingRight	Определяет размер отступа между содержимым элемента и его правой границей
paddingTop	Определяет размер отступа между содержимым элемента и его верхней границей
pageBreakAfter	Устанавливает разрыв страницы после элемента
pageBreakBefore	Устанавливает разрыв страницы перед элементом
pixelHeight	Содержит высоту элемента в пикселах
pixelLeft	Содержит горизонтальную координату верхнего левого угла элемента в пикселах

Таблица 3.8 *(окончание)*

Свойство	Описание
pixelTop	Содержит вертикальную координату верхнего левого угла элемента в пикселах
pixelWidth	Содержит ширину элемента в пикселах
position	Определяет применяемую модель позиционирования элемента
posLeft	Содержит горизонтальную координату левого верхнего угла элемента в единицах измерения, заданных непосредственно перед использованием свойства
posTop	Содержит вертикальную координату левого верхнего угла элемента в единицах измерения, заданных непосредственно перед использованием свойства
posWidth	Содержит ширину элемента в единицах измерения, заданных непосредственно перед использованием свойства
styleFloat	Задает стиль расположения содержимого элемента
textAlign	Устанавливает выравнивание текстового содержимого элемента
textDecoration	Устанавливает способ отображения текста
textIndent	Устанавливает отступ первой строки абзаца
textTransform	Задает стиль отображения различных регистров шрифта
top	Содержит вертикальную координату верхнего левого угла элемента
verticalAlign	Задает вертикальное выравнивание элемента
visibility	Регулирует видимость данного элемента
width	Задает ширину элемента
zIndex	Устанавливает псевдовертикальное позиционирование элемента

Руководствуясь этой таблицей, мы получаем доступ ко всем свойствам стилевого оформления для каждого элемента содержимого Web-страницы. Таким образом, с помощью технологии DHTML мы управляем свойствами отображения любого элемента Web-страницы.

Позиционирование элементов Web-страницы

Одно из наиболее используемых свойств DHTML — возможность точно позиционировать элементы оформления Web-страницы. Как мы помним, в HTML у нас отсутствует возможность четкого позиционирования каких-либо элементов Web-страниц. Технология DHTML исправляет этот недостаток. Более того, DHTML позволяет не только точно устанавливать место отображения того или иного элемента, но и перемещать его в окне просмотра браузера. Подобная возможность обычно называется *"динамическим позиционированием"*.

Динамическое позиционирование обеспечивается взаимодействием программ-скриптов с элементами Web-страниц и их стилевым оформлением. Во второй главе мы уже рассматривали свойства CSS, позволяющие позиционировать элементы Web-страниц.

В DHTML применяются три типа позиционирования. Различают статическое позиционирование, относительное и абсолютное. Статическое позиционирование — это порядок размещения элементов Web-страниц самим браузером, исходя из обычных стандартов HTML. Относительное позиционирование позволяет изменять расположение элемента, учитывая его исходное расположение. Другими словами, смещение элемента отсчитывается от его исходных координат. На основе относительного позиционирования обычно создают различные эффекты анимации текста. Абсолютное позиционирование позволяет нам указывать координаты элементов относительно окна браузера. Следовательно, мы можем точно знать в каждый момент времени, в какой точке окна просмотра находится тот или иной элемент. Чаще всего для динамического позиционирования используется именно этот тип позиционирования.

Необходимо учитывать, что статическое позиционирование полностью опирается на порядок расположения элементов, который указан в HTML-коде, и параметры настройки браузера. Если же мы для какого-либо элемента установим абсолютное позиционирование, то он уже не будет учитываться браузером при размещении остальных элементов. Этот элемент исключается из процедуры статического позиционирования и не оказывает никакого влияния на соседние элементы.

Попробуем создать HTML-документ, в котором текстовый абзац мог бы изменять свое расположение в тот момент, когда пользователь наводит на него курсор мыши. Код подобного документа приведен в листинге 3.6. Результат отображения подобного HTML-документа в окне браузера показан на рис. 3.6.

Листинг 3.6

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Динамическое позиционирование</title>
  <script language="javascript">
  <!--
  function moving()
  {
  p1.style.position="relative";
  p1.style.top="10px";
  p1.style.left="10px";
  }
  function reverse()
  {
  p1.style.position="static";
  }
  //-->
  </script>
  </head>
  <body>
  <p><font size=5>Первый абзац</font></p>
  <p id="p1" onmouseover="moving()" onmouseout="reverse()"><font
size=5>Второй абзац</font></p>
  </body>
</html>
```

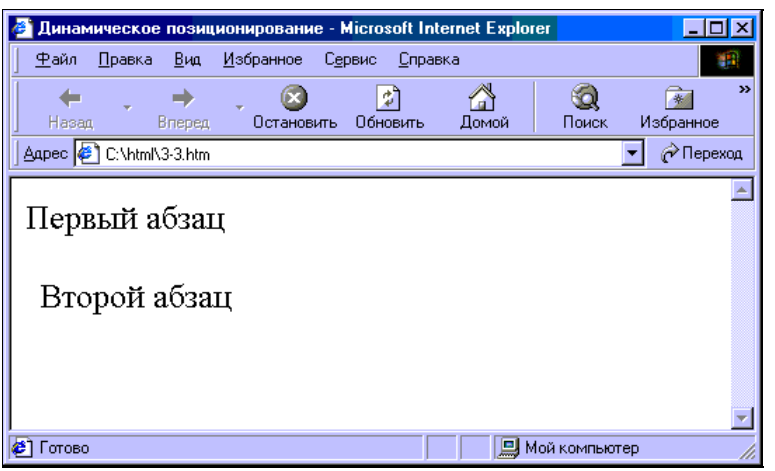


Рис. 3.6. Окно браузера с результатом отображения файла, приведенного в листинге 3.6, в тот момент, когда пользователь навел курсор мыши на второй абзац

При загрузке в браузер этого файла будет видно (рис. 3.6), что при наведении курсора мыши на второй абзац тот сдвигается влево и вниз, а после того, как курсор мыши убирается с него, абзац возвращается к своему исходному положению.

Теперь рассмотрим код, чтобы узнать, за счет чего мы можем добиться подобного эффекта. Мы установили идентификатор для второго абзаца и две функции, выполняемые при наступлении событий `onmouseover` и `onmouseout`.

Событие `onmouseover`, наступающее в тот момент, когда пользователь наводит курсор мыши на второй абзац, мы обрабатываем с помощью функции `moving`. В этой функции мы изменяем некоторые свойства объекта `style` для элемента с идентификатором `p1`. Использование объекта `style` мы обсуждали в предыдущем разделе этой главы. Сейчас нас интересует именно механизм перемещения содержимого абзаца. Мы использовали три свойства: `position`, `top` и `left`. Свойству `position` присвоено значение `relative`, которое позволяет нам менять расположение элемента относительно его начального размещения. Теперь достаточно задать значения свойств `top` и `left`, и абзац будет менять свое расположение. В нашем примере мы использовали для этих свойств одно и то же значение — `10px`, что позволяет смещать абзац вниз и влево на десять пикселов.

После смещения абзаца было бы неплохо вернуть его на место. Для этого достаточно убрать курсор мыши с территории, занимаемой искомым текстовым абзацем. Следовательно, необходимо обрабатывать событие `onmouseout`. Такую обработку в нашем скрипте выполняет функция `reverse`. Она также использует объект `style`. Для того чтобы вернуть абзац в его начальное положение, достаточно приписать свойству `position` значение `static`. Мы уже знаем, что с помощью этого значения определяется статическое позиционирование элемента. Следовательно, он будет отображаться именно так, как это было при начальной загрузке страницы. Проще говоря, он вернется на исходное место.

Мы рассмотрели динамическое позиционирование элементов, которое основано на применении относительного типа позиционирования, но есть и абсолютное позиционирование. Оно обычно применяется для позиционирования элементов, не ориентируясь на их расположение на Web-странице по умолчанию.

Примером может служить создание страницы, на которой некое слово будет постоянно отображаться в верхнем левом углу окна просмотра браузера, вне зависимости от того, какая часть содержимого документа видна в окне просмотра. Мы разместим это слово под основным содержимым Web-страницы, использовав для этого свойство `z-index`. Таким образом получится, что некий текст будет отображаться под основным содержимым страницы, как будто подложенный под него. Подобный документ можно создать с помощью кода, приведенного в листинге 3.7 (см. также рис. 3.7).

Листинг 3.7

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Абсолютное позиционирование</title>
  <script language="javascript">
  <!--
  function freezetext()
  {
  freeze.style.posTop=document.body.scrollTop;
  freeze.style.posLeft=document.body.scrollLeft;
  }
  //-->
  </script>
  </head>
  <body onScroll="freezetext()">
  <div id="freeze" style="position:absolute; z_index:-1; top:1px;
left:1px">
  <p>Замороженный текст</p>
  </div>
  <p><font size=5>Первый абзац</font></p>
  <p><font size=5>Второй абзац</font></p>
  <p><font size=5>Третий абзац</font></p>
  <p><font size=5>Четвертый абзац</font></p>
  <p><font size=5>Пятый абзац</font></p>
  </body>
</html>
```

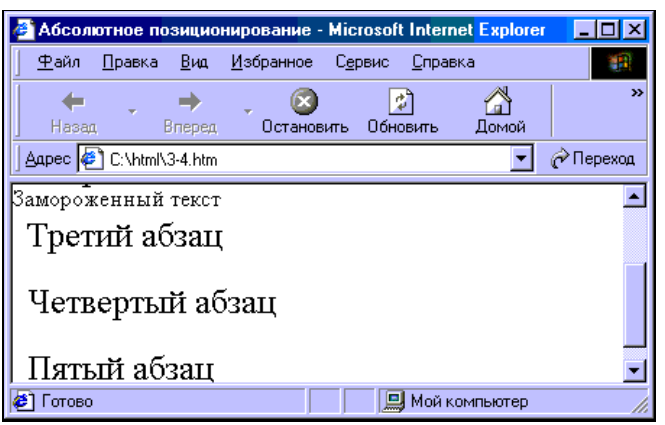


Рис. 3.7. Окно браузера с результатом отображения файла, приведенного в листинге 3.7, в тот момент, когда пользователь прокрутил содержимое документа вверх

На рис. 3.7 видно, что при прокрутке содержимого "замороженный текст" все равно остается прикрепленным к верхнему левому углу окна браузера. Попробуем разобраться, почему это происходит.

Из кода HTML-документа видно, что искомый текст мы заключили в контейнер, создаваемый тэгами `<div>` и `</div>`. Для этого блока-контейнера мы создали стилевое оформление, воспользовавшись параметром `style`. В объявлении стиля установили для него абсолютное позиционирование, задали его координаты и указали вертикальное псевдопозиционирование с помощью свойства `z-index`, которому придали значение "`-1`". Это заставит браузер при прокрутке содержимого Web-страницы в окне просмотра всегда отображать наш "замороженный текст" под основным содержимым, т. к. все стальные элементы оформления Web-страницы имеют по умолчанию нулевое значение свойства `z-index`, а следовательно, будут отображаться над "замороженным текстом".

Мы хотим, чтобы наш текстовый блок постоянно отображался в одном и том же месте — в левом верхнем углу окна просмотра. Координаты, которые мы установили для текстового блока в его определении, указывают его расположение на Web-странице без учета того, какая ее часть помещается в окне просмотра. Следовательно, когда посетитель страницы воспользуется полосами прокрутки, текстовый блок может выйти за пределы окна просмотра. Нам необходимо реагировать именно на использование полос прокрутки пользователем. Для этого мы будем обрабатывать событие `onScroll`, относящееся ко всему содержимому документа, т. е. нас будет интересовать тэг `<body>`. В коде документа мы использовали следующее объявление этого тэга:

```
<body onScroll="freezetext()">
```

Нам необходимо при прокрутке содержимого окна просмотра принудительно отображать текстовый блок в левом верхнем углу окна просмотра браузера, для этих целей мы пользуемся абсолютным позиционированием. Но для того, чтобы позиционировать блок текста, нам необходимо знать, какие координаты документа соответствуют верхнему левому углу окна просмотра. Здесь следует вспомнить о свойствах `scrollTop` и `scrollLeft` элемента `body`, которые содержат координаты этого верхнего левого угла окна просмотра в системе координат самого документа. Эти координаты мы присвоили значениям `posTop` и `posLeft`, соответственно.

Подобную технологию мы можем применять не только к текстовым блокам, но и практически к любым элементам наполнения Web-страниц, в том числе и к графическим изображениям и к их более совершенной версии — видеовставкам.

Обработка форм

Чаще всего формы с их управляющими элементами ввода данных используются для того, чтобы передавать информацию, введенную пользователем, обрабатывающей программе, которая расположена на каком-либо Web-сервере. Все скрипты, которые мы используем в DHTML, в силу своей природы выполняются в браузере удаленного пользователя, поэтому обработка форм средствами DHTML невозможна. На самом деле, не все так безнадежно.

В предыдущих разделах главы мы уже видели примеры работы с элементами ввода информации. При этом их даже необязательно было включать в формы. Более того, когда мы рассматривали объекты, создаваемые теми или иными элементами содержимого Web-страниц, то видели, что тэги объявления форм и создания элементов управления тоже входят в объектную модель DHTML. Следовательно, у нас есть возможность работать с ними с помощью программ-скриптов.

Очень часто скрипты используют для проверки соответствия некоторым условиям информации, вводимой пользователем. Например, одно из полей текстового ввода предназначено для указания возраста удаленного посетителя. Разработчик Web-страницы ожидает, что туда будут вноситься только цифры. Далеко не всегда так происходит. Обязательно найдется человек (и не один), который внесет в это поле ввода несоответствующую информацию просто из любопытства или по ошибке. Естественно, при отсылке введенных данных ничего произойти не должно. Если программа-анализатор, принимающая эти данные из формы, спроектирована правильно, то она проверит введенное значение и, если оно не входит в установленные разумные пределы (возраст не может быть отрицательным), сгенерирует Web-страницу с сообщением о некорректности введенной информации и отправит эту страницу в браузер удаленного пользователя. На все это нужно время: для передачи данных на сервер, для их обработки, и для передачи созданной Web-страницы от сервера обратно к удаленному пользователю. Если же процедуру проверки введенных данных мы встроим в саму Web-страницу, используя для этих целей технологию DHTML, то временной задержки можно избежать. В листинге 3.8 приведен пример кода подобного HTML-документа.

Листинг 3.8

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Обработка форм</title>
```

```
  <script language="javascript">
  <!--
  function checkdata()
  {
  if ((event.keyCode<48)||(event.keyCode>57)) event.returnValue=false
  }
  //-->
  </script>
  </head>
  <body>
  <form>
  <p> Возраст <input type="text" onKeyPress="checkdata()"></p>
  </form>
  </body>
</html>
```

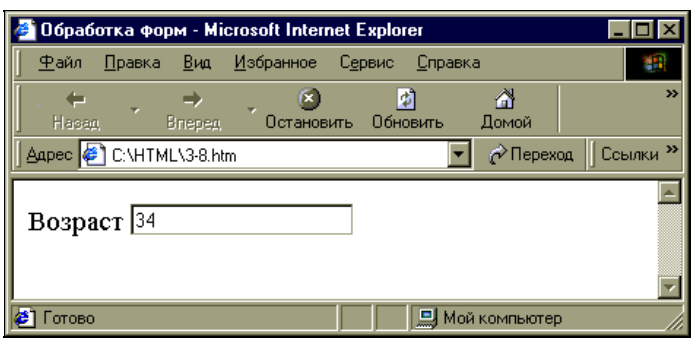


Рис. 3.8. Окно браузера с результатом отображения файла, приведенного в листинге 3.8

В этом HTML-документе мы создаем форму с одним полем текстового ввода (рис. 3.8). Данное поле предназначено для ввода возраста удаленного пользователя. Понятно, что возраст может быть лишь числовым значением. С помощью программы-скрипта мы блокируем ввод символов, не являющихся цифрами, используя тот факт, что коды цифровых символов лежат в промежутке между 48 и 57.

Мы будем обрабатывать событие `onKeyPress` для поля ввода текстовой информации. Таким образом, мы перехватываем нажатие каждой клавиши, когда фокус ввода передан полю ввода. Коды нажатых пользователем клавиш мы получаем с помощью свойства `keyCode` объекта `event`. После этого остается лишь проверить, что эти коды находятся в промежутке допустимых в данном случае значений. Если это не так, следовательно, пользователь нажал не цифровую клавишу. В этом случае мы используем свойство `returnValue` все того же объекта `event`. Введенный символ отображается в поле текстового ввода после того, как событие `onKeyPress` возвратит его код.

Если же мы присваиваем свойству `returnValue` значение `False`, то возврата значения не произойдет. Следовательно, и введенный символ не будет отображаться в поле ввода. Именно этот алгоритм проверки реализует скрипт, внедренный в HTML-документ, код которого приведен в листинге 3.8.

Одной проверки вводимых в соответствующее поле символов может оказаться недостаточно. Возраст человека должен быть правдоподобным. В нашем документе ничто не помешает человеку внести в поле ввода значение, состоящее, например, из пяти цифр. Понятно, что полученное число имеет весьма отдаленное отношение к реальному возрасту удаленного пользователя. Было бы неплохо, после ввода числа пользователем проверить, находится ли оно в промежутке между, скажем, десятью и девяноста. Код документа с подобным скриптом приведен в листинге 3.9, а результат его отображения — на рис. 3.9.

Листинг 3.9

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Обработка форм</title>
  <script language="javascript">
  <!--
  function checkdata()
  {
  if ((event.srcElement.value<10)||(event.srcElement.value>90))
↳alert("Введите правильный возраст");
  }
  //-->
  </script>
  </head>
  <body>
  <form>
  <p> Возраст <input type="text" name="f1" onChange="checkdata()"></p>
  <p> Имя и фамилия <input type="text" name="f2"></p>
  <p><input type="submit" value="Отправить"></p>
  </form>
  </body>
</html>
```

В этом документе для проверки значения, введенного пользователем, мы используем событие `onChange`, которое инициируется после того, как пользователь перейдет от искомого поля ввода к какому-либо другому элементу управления Web-страницы. Это событие отрабатывается непосредственно перед событием `onBlur`.

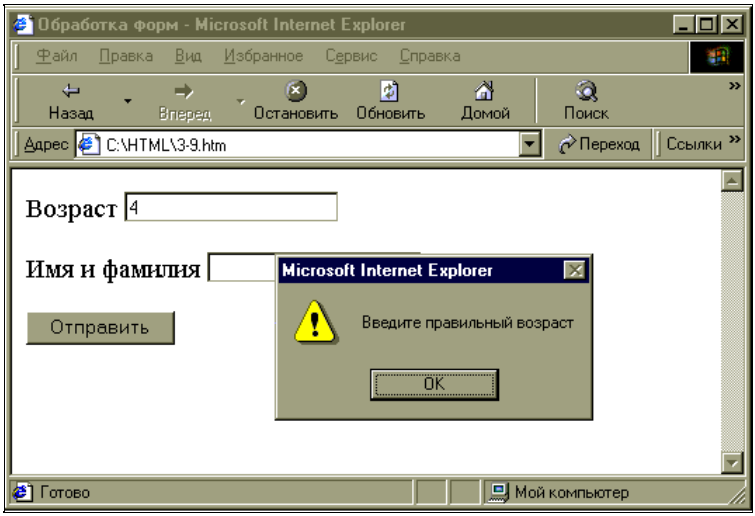


Рис. 3.9. Окно браузера с результатом отображения файла, приведенного в листинге 3.9, после того, как пользователь внес неправильное значение в поле ввода

Итак, пользователь ввел некоторое значение в интересующее нас поле. Теперь мы перехватываем событие `onChange` и проводим проверку введенного значения. Это значение хранит свойство `value`. Доступ к нему возможен с помощью свойства `srcElement` объекта `event`. Теперь остается с помощью условного оператора `if` проверить, удовлетворяет ли введенное значение заданным условиям. В том случае, если значение не удовлетворяет условиям, мы с помощью оператора `alert` отображаем информационное окно с соответствующим текстом (рис. 3.9).

Подобная технология позволяет проверять все вводимые пользователем данные на соответствие неким условиям, которые определяются разработчиком Web-страницы. Это помогает исключить досадные ошибки, которые обязательно будут возникать из-за невнимательности пользователей, без дополнительного обращения к приложению, обрабатывающему данные.

Рассмотренные нами способы работы с формами и элементами управления не исчерпывают весь спектр возможностей DHTML в этой области. Мы можем динамически изменять содержимое формы в зависимости от действий пользователя.

Предположим, что нам необходимо провести на сайте некий опрос. При этом для мужчин и женщин используются различные наборы вопросов. Мы можем либо создать две отдельные Web-страницы с наборами вопросов, одну для мужчин, а другую для женщин, и два приложения, обрабатывающих эти формы. Другой способ — с помощью DHTML модифицировать одну и ту же Web-страницу в зависимости от указанного пола удаленного пользователя. Код подобного HTML-документа приведен в листинге 3.10 (см. также рис. 3.10).

Листинг 3.10

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Обработка форм</title>
  <script language="javascript">
  <!--
  function changelist(x)
  {
  if (x==1)
   {document.all.select1.options[0].text="Ольга";
    document.all.select1.options[1].text="Елена";
    document.all.select1.options[2].text="Лариса";
    p3.innerText="Выберите Ваше любимое женское имя";
   }
  if (x==2)
   {document.all.select1.options[0].text="Игорь";
    document.all.select1.options[1].text="Алексей";
    document.all.select1.options[2].text="Олег";
    p3.innerText="Выберите Ваше любимое мужское имя";
   }
  }
  //-->
  </script>
  </head>
  <body>
  <form>
  <p>Ваш пол:  </p>
  <p>Мужской <input type="radio" name="group1" value="male"
onClick="changelist(1)" checked></p>
  <p>Женский <input type="radio" name="group1" value="female"
onClick="changelist(2)"></p>
  <p id="p3">Выберите Ваше любимое женское имя</p>
  <p><select name="menu" id="select1">
  <option value="1">Ольга</option>
  <option value="2">Елена</option>
  <option value="3">Лариса</option>
  </select>
  </p>
  <input type="submit" value="Подтвердить">
  </form>
  </body>
</html>
```

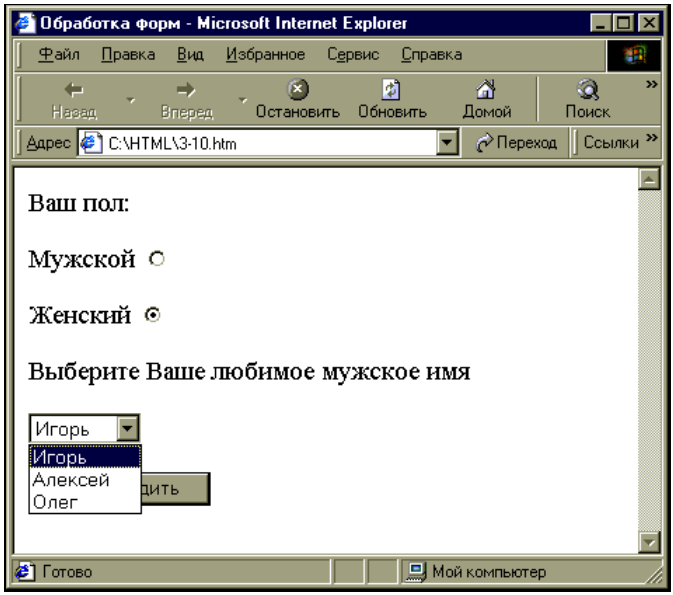


Рис. 3.10. Окно браузера с результатом отображения файла, приведенного в листинге 3.10, после того, как пользователь выбрал вторую радиокнопку

Итак, рассмотрим механизм изменения содержимого отдельных элементов оформления Web-страницы, инициируемого действиями пользователя. Изменения происходят после того, как пользователь активизирует одну из двух радиокнопок (рис. 3.10). Для этого мы в определении этих радиокнопок указали, что событие `onClick`, возникающее при одиночном щелчке мыши на радиокнопке, будет обрабатываться функцией `changelist`. В качестве параметра данной функции передается единица или двойка, в зависимости от выбранной удаленным пользователем радиокнопки.

Теперь опишем принцип действия искомой функции `changelist`. Она призвана изменять содержимое выпадающего списка имен. Так же, она изменяет текст, отображаемый перед этим списком. По умолчанию мы создали Web-страницу, ориентированную на мужчин. Как только пользователь с помощью соответствующей радиокнопки указывает, что он является женщиной, программа-скрипт изменяет список имен, вставляя туда мужские имена, и модифицирует текстовую строку перед этим списком. Необходимо учесть, что пользователь может несколько раз изменять свой выбор, активизируя ту или иную радиокнопку. Именно поэтому функция `changelist` состоит из двух блоков, динамически изменяющих часть формы в зависимости от выбора удаленного пользователя.

Для того чтобы изменить имена в выпадающем списке, мы используем конструкцию следующего вида:

```
document.all.select1.options[0].text
```

Мы знаем, что наш выпадающий список имеет идентификатор `select1`. В данном случае мы получаем доступ к нему через коллекцию `all` объекта `document`. В этом объекте выпадающего списка есть коллекция `options`, в которой содержатся все пункты выпадающего списка. Доступ к ним осуществляется с помощью указания их номера в квадратных скобках. Необходимо учитывать, что нумерация в коллекциях ведется с нуля.

В элементах списка мы меняем только текст, отображаемый в окне просмотра. Значение, передаваемое в обрабатывающую программу, мы не изменяем. Для того чтобы изменить текст элементов списка, мы воспользовались соответствующим свойством `text`.

Также изменению подвергался текст абзаца с идентификатором `p3`. Для того чтобы изменить его текстовое содержимое, мы воспользовались свойством `innerText`, присущим данному элементу. Это свойство содержит только текст, находящийся между граничными тэгами элемента.

Графические фильтры

Самой необычной возможностью технологии DHTML является использование так называемых "графических фильтров". Это, по сути, некоторые графические преобразования, которые могут быть применены избирательно к любому элементу оформления Web-страницы. Первоначально эти фильтры создавались на основе элементов ActiveX, и, соответственно, лучше всего они работают в браузере Microsoft Internet Explorer.

Обычно различают статические и динамические фильтры. Любой статический графический фильтр выполняет мгновенное преобразование элемента содержимого Web-страницы, к которому он применяется. Динамические фильтры преобразование выполняют в течение некоторого времени, что позволяет увидеть этот процесс "в динамике". Чаще всего при разработке Web-страниц используют статические фильтры, т. к. динамические графические фильтры поддерживаются далеко не всеми браузерами. И мы рассмотрим работу именно со статическими графическими фильтрами.

Доступ к объектам графических фильтров обычно производится с помощью коллекции `filters`, которая является свойством практически каждого объекта, соответствующего тому или иному элементу содержимого Web-страницы. Естественно, из самого наличия именно коллекции объектов следует тот факт, что к одному элементу можно применять несколько графических фильтров. Также можно воспользоваться свойством `style`, которое в свою очередь обладает подсвойством `filter`.

Приведем пример HTML-документа с использованием одного из статических фильтров, чтобы рассмотреть механизм их применения (листинг 3.11 и рис. 3.11).

Листинг 3.11

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="xray";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="redhood.gif" onClick="filtering()"></p>
  </form>
  </html>
```

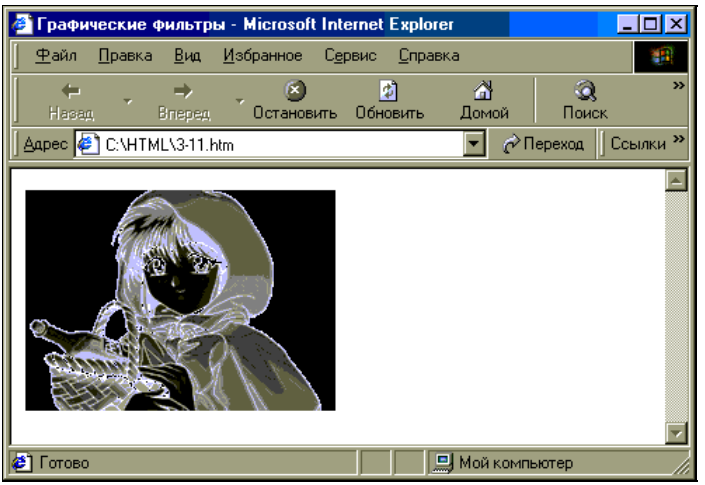


Рис. 3.11. Окно браузера с результатом отображения файла, приведенного в листинге 3.11, после того, как пользователь щелкнул кнопкой мыши на рисунке

На рис. 3.11 видно, во что превратился рисунок gif-формата из двухсот пятидесяти шести цветов после того, как пользователь щелкнул кнопкой мыши на графическом изображении. После этого щелчка к рисунку был применен статический графический фильтр с наименованием "`xray`", которое указывает, что данный графический эффект напоминает действие рентге-

новских лучей. Если посмотреть на рисунок, то станет понятно, что доля истины в этом названии есть, т. к. графическое изображение, в самом деле, похоже на рентгеновские снимки.

Активизация фильтра происходит с помощью свойства `filter` объекта `style`, являющегося составной частью объекта, содержащего графическое изображение. Доступ к этому объекту осуществляется с помощью уникального идентификатора элемента, через коллекцию `all`.

Итак, мы видели результат применения фильтра "`xray`" к обыкновенному графическому изображению. Осталось еще двенадцать статических фильтров, наименований которых мы не знаем. Нам неизвестны также преобразования, которые они производят с тем или иным элементом содержимого Web-страницы. Рассмотрим способы работы с этими статическими фильтрами.

Начнем мы с простых фильтров отражения. Фильтр с наименованием "`fliph`" задает горизонтальное отражение элемента. Код HTML-документа, в котором используется данный фильтр, приведен в листинге 3.12.

Листинг 3.12

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="fliph";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

Как видно, код документа не слишком отличается от предыдущего листинга. Основное отличие — в механизме действия статического фильтра. С помощью фильтра получено зеркальное отражение изображения относительно вертикальной оси, проходящей через его центр (рис. 3.12).

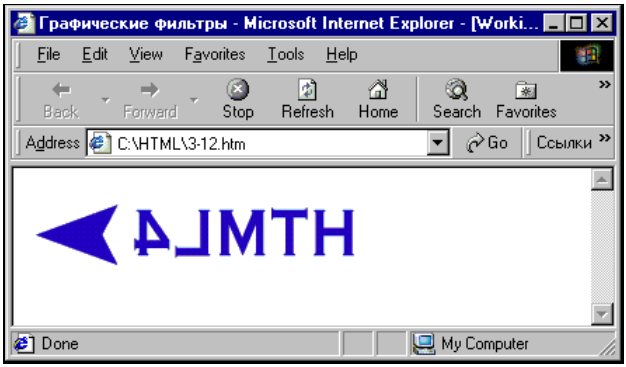


Рис. 3.12. Окно браузера с результатом отображения файла, приведенного в листинге 3.12, после обработки рисунка графическим фильтром

Существует также фильтр, отражающий содержимое элемента по вертикали. В листинге 3.13 приведен код HTML-документа, использующего данный фильтр.

Листинг 3.13

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="flipv";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

Из приведенного кода HTML-документа ясно, что этот статический фильтр носит наименование "`flipv`". Рис. 3.13 показывает, что данный фильтр отражает содержимое элемента Web-страницы по вертикали, относительно горизонтальной оси элемента, проходящей через его центр.

Список статических графических фильтров, не требующих передачи параметров, еще не исчерпан. Без параметров обходится статический фильтр

с наименованием "grayscale", который отображает элемент с помощью различных градаций серого цвета. Применение еще одного подобного фильтра показано в листинге 3.14.

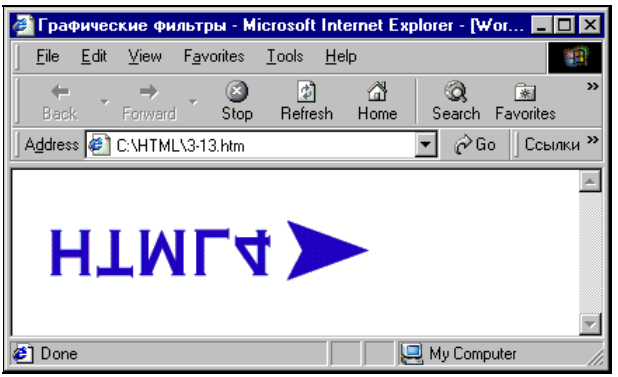


Рис. 3.13. Окно браузера с результатом отображения файла, приведенного в листинге 3.13, после обработки изображения графическим фильтром

Листинг 3.14

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="invert";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

К графическому изображению был применен фильтр с наименованием "invert". Данный фильтр инвертирует, т. е. меняет на противоположные, все цвета элемента, который он обрабатывает. На рис. 3.14 видно, что был изменен лишь цвет значимой части изображения, а фон остался прежним. Связано это с тем, что применяющееся в данном случае графическое изо-

бражение было сохранено в формате GIF, и фон его был обозначен как "прозрачный" цвет. Эта уникальная возможность есть только у данного формата. Применяя фильтр "invert", всегда следует помнить, что прозрачный цвет не поддается инвертированию.

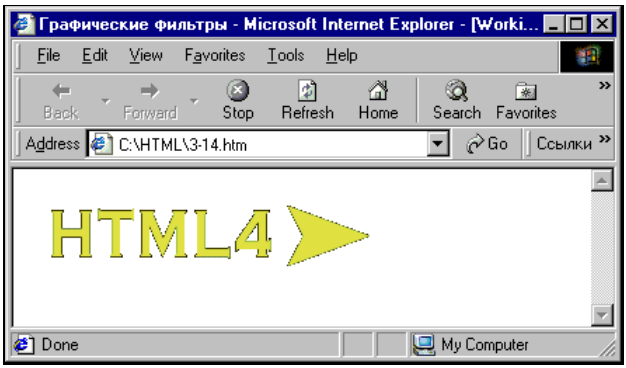


Рис. 3.14. Окно браузера с результатом отображения файла, приведенного в листинге 3.14, после обработки изображения графическим фильтром

Все пять рассмотренных нами фильтров обладают одной общей чертой — они не нуждаются в дополнительных параметрах. Далеко не все статические графические фильтры так нетребовательны. Большинству из них нужно задать дополнительные параметры. Пример использования подобного графического фильтра показан в листинге 3.15.

Листинг 3.15

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="shadow(color=black)";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

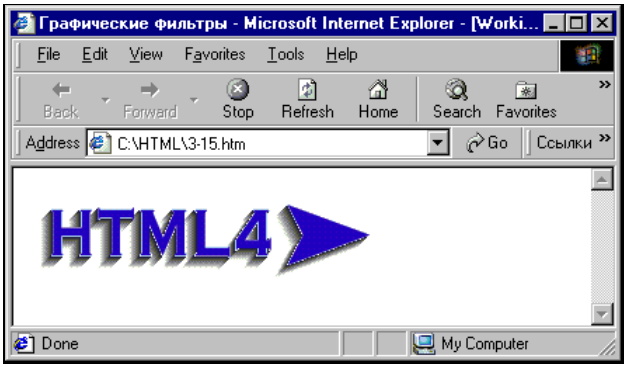


Рис. 3.15. Окно браузера с результатом отображения файла, приведенного в листинге 3.15, после обработки изображения графическим фильтром

В листинге хорошо видно, что при объявлении накладываемого графического фильтра мы добавили к его наименованию "shadow" параметр и его значение, заключенные в скобки. Результат применения данного фильтра показан на рис. 3.15. Этот графический фильтр создает тень у элемента оформления Web-страницы. Цвет тени мы можем задается параметром color. В качестве значения параметра используется одно из стандартных цветовых обозначений HTML.

Существует также фильтр с наименованием "mask", чье действие весьма похоже на только что рассмотренный фильтр. Но фильтр "mask" не инвертирует цвета элемента, а делает так называемую маску: фон изображения переводится в специально заданный разработчиком цвет, а основной план изображения переводится в белый цвет. Пример использования этого фильтра показан в листинге 3.16 (рис. 3.16).

Листинг 3.16

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="mask(color=green)";
  }
  //-->
  </script>
  </head>
```

```
<body>
  <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
</form>
</html>
```

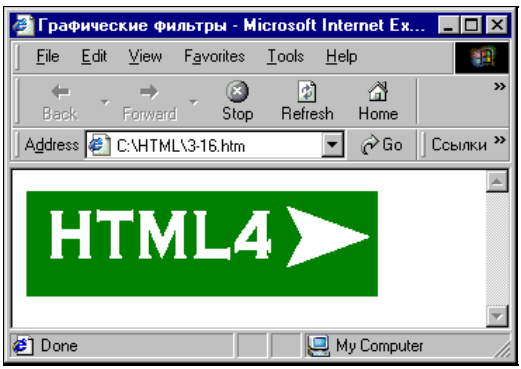


Рис. 3.16. Окно браузера с результатом отображения файла, приведенного в листинге 3.16, после обработки изображения графическим фильтром

Легко заметить, что для установки цвета, которым будет отображаться создаваемая маска, мы используем параметр с наименованием `color`. Значением параметра может быть любое принятое в HTML обозначение цвета. Если не указывать дополнительный параметр, то для создания маски будет использован черный цвет.

Способ применения еще одного графического фильтра показан в листинге 3.17. Фильтр с наименованием `"blur"` несколько "размазывает" отображаемый элемент, создавая эффект его движения.

Листинг 3.17

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="blur(add=1, direction=90, strength=20)";
  }
  //-->
  </script>
  </head>
```

```
<body>
  <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
</form>
</html>
```

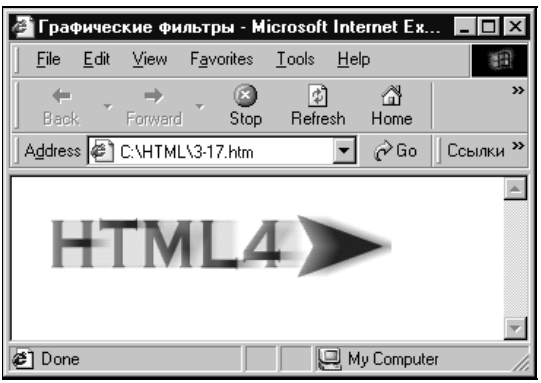


Рис. 3.17. Окно браузера с результатом отображения файла, приведенного в листинге 3.17, после обработки изображения графическим фильтром

К сожалению, ограниченные возможности полиграфии не позволяют показать результат применения данного фильтра в полном объеме (рис 3.17). Но если все-таки есть желание увидеть его, стоит создать HTML-файл с кодом, приведенным в листинге 3.17, и загрузить его в браузер.

Легко заметить, что данный эффект обладает тремя дополнительными параметрами. Параметр `direction` задает направление, в котором будет производиться смещение изображения элемента. Значение данного параметра — число, обозначающее количество градусов, отсчитываемых от вертикально направленной воображаемой линии. Параметр `strength` указывает, насколько "наполненным" будет смещенное отображение. Чем больше значение этого параметра, тем дальше будет смещаться изображение. Значение его — обычное целое число. Параметр `add` является логическим, и указывает, следует ли добавлять более слабые копии изображения в итоговый результат и создавать таким образом иллюзию движущегося элемента, или нет. Если мы используем нулевое значение, то ослабленные копии добавляться не будут, и мы получим достаточно сильно размазанное изображение, в котором трудно опознать оригинал. Обычно в качестве значения данного параметра используется любое целое число. При этом не имеет значения, какое число мы используем, лишь бы оно не было нулевым, т. к. параметр имеет логический тип.

Графический фильтр с наименованием "`dropshadow`" позволяет создавать тень у элемента, к которому он применен. Пример HTML-документа, использующего данный фильтр, показан в листинге 3.18.

Листинг 3.18

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="dropshadow(offx=10, offy=10, color=green,
positive=1)";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

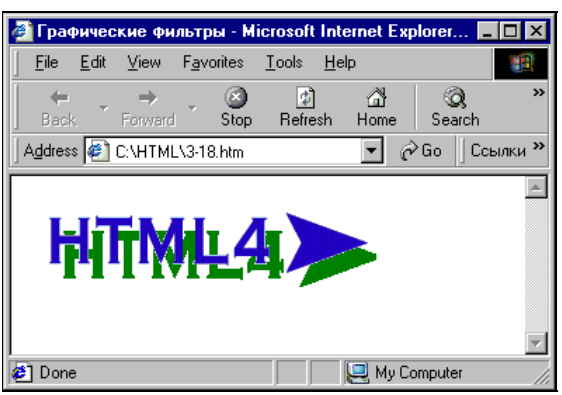


Рис. 3.18. Окно браузера с результатом отображения файла, приведенного в листинге 3.18, после обработки изображения графическим фильтром

С помощью параметров графического фильтра мы можем управлять внешним видом изображения элемента, к которому данный фильтр применяется. Параметры `offx` и `offy`, как нетрудно догадаться, позволяют задавать смещение тени относительно исходного элемента по горизонтали и вертикали соответственно. Для этих параметров используются числовые значения. Параметр `color` задает цвет тени. Параметр `positive` — логический. В том случае, если ему присвоено ненулевое значение, или он вообще не установлен, тень будет отображаться в стандартном виде, как это показано на рис. 3.18.

Иначе тень инвертируется, и ее фон будет отображаться цветом, заданным с помощью параметра `color`, а само теневое изображение элемента будет отображаться цветом фона.

Применение еще одного достаточно интересного статического графического фильтра показано в листинге 3.19 (рис. 3.19). Этот фильтр называется "`glow`", и позволяет создавать размазанный контур вокруг элемента, повторяя контур этого элемента.

Листинг 3.19

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="glow(color=green, strength=10)";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

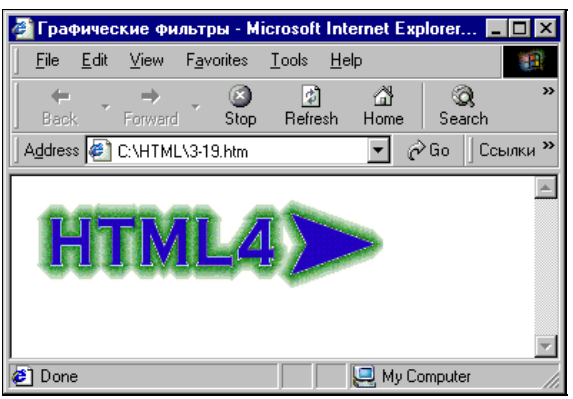


Рис. 3.19. Окно браузера с результатом отображения файла, приведенного в листинге 3.19, после обработки изображения графическим фильтром

Легко заметить, что данный фильтр обладает двумя параметрами. Параметр `color` позволяет указывать цвет, которым будет создаваться дополнительный контур элемента, а параметр `strength` задает размер этого нового контура и, косвенно, — его размытость. Чем больше значение этого параметра, тем больше дополнительный обтекающий контур у элемента.

С помощью фильтра "`alpha`" мы можем устанавливать уровень прозрачности для того или иного элемента. Пример использования данного фильтра показан в листинге 3.20 (см. также рис. 3.20).

Листинг 3.20

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="alpha(opacity=40)";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

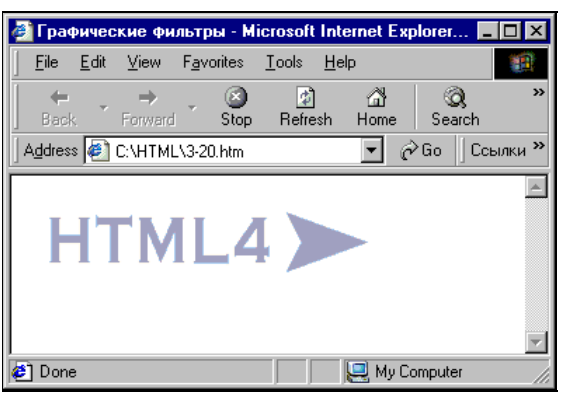


Рис. 3.20. Окно браузера с результатом отображения файла, приведенного в листинге 3.20, после обработки изображения графическим фильтром

Основной параметр `opacity` задается числом из диапазона от нуля до ста, указывающим процент прозрачности элемента. Если используется нулевое значение, то элемент становится полностью прозрачным и сливается с фоном, т. е. становится невидимым. Если же мы устанавливаем значение, равное ста процентам, то внешний вид элемента не меняется.

Одним из самых интересных фильтров является фильтр "`wave`", накладывающий на объект волнообразное преобразование. Его применение проиллюстрировано листингом 3.21 (рис. 3.21).

Листинг 3.21

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
  <head>
  <title>Графические фильтры</title>
  <script language="javascript">
  <!--
  function filtering()
  {
  document.all.p1.style.filter="wave(freq=3, strength=5, add=0)";
  }
  //-->
  </script>
  </head>
  <body>
    <p><img id="p1" src="htm.gif" onClick="filtering()"></p>
  </form>
  </html>
```

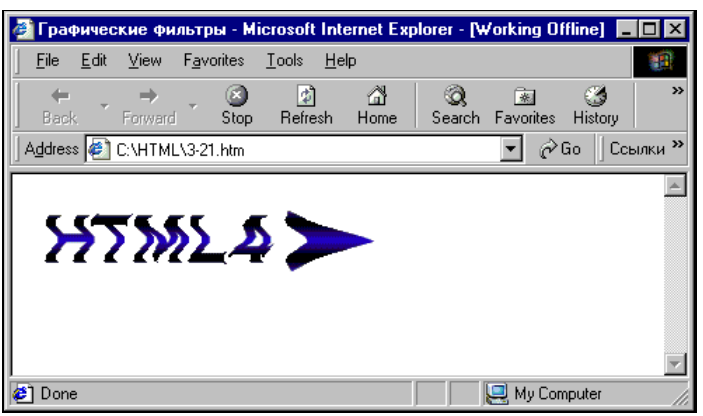


Рис. 3.21. Окно браузера с результатом отображения файла, приведенного в листинге 3.21, после обработки изображения графическим фильтром

С помощью числового параметра `freq` мы задаем количество "волн" в результирующем изображении. Чем больше значение этого параметра, тем более "волнистым" окажется результат. А "высота" волн, точнее, их вертикальный размер задается числовым параметром `strength`. С действием параметра `add` мы уже знакомы. Если мы укажем для него ненулевое значение, то вместе с результатом действия графического фильтра будет отображен и исходный элемент. Этот вариант изображения выглядит громоздко, поэтому чаще используют нулевое значение для параметра `add`.

Последний рассматриваемый нами статический фильтр носит наименование "`light`". Он позволяет создавать эффекты освещения элемента различными источниками света, и его стоит рассмотреть несколько подробнее, т. к. у него есть несколько методов. Мы создадим Web-страницу с одной кнопкой, расположенной по центру строки, и опишем реакцию на указание курсором мыши на кнопку. В этом случае мы создадим иллюзию возникновения точечного освещения, источник которого будет находиться почти точно над кнопкой. Таким образом, мы сможем привлечь внимание удаленного пользователя к этой кнопке. Вместо кнопки можно установить любой другой элемент Web-страницы.

Итак, рассмотрим листинг 3.22.

Листинг 3.22

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
↳"http://www.w3.org/TR/HTML4/strict.dtd">
<html>
 <head>
<title>Графические фильтры</title>
 <script language="javascript">
 <!--
 function lighting()
 {
  m.style.filter="light";
  l=m.filters[0];
  l.addPoint(250,50,150,60,230,60,10000);
 }
 //-->
 </script>
 </head>
 <body id="m">
 <p align="center"><input type="button" value="КНОПКА" name="BUT1"
↳onMouseOver="lighting()"></p>
 </body>
</html>
```

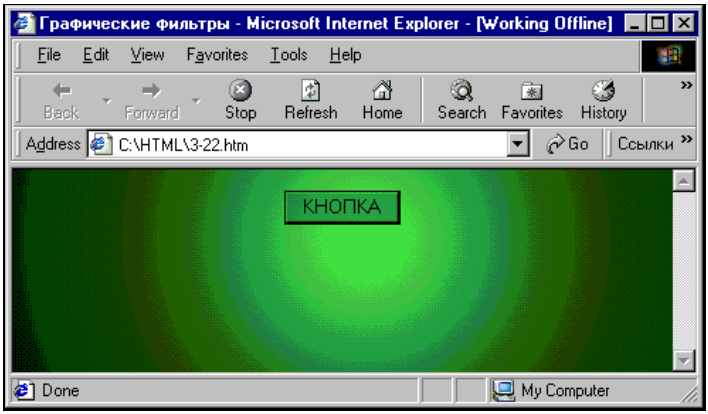


Рис. 3.22. Окно браузера с результатом отображения файла, приведенного в листинге 3.22, после обработки изображения графическим фильтром

В теле HTML-документа мы создаем одну кнопку, установленную по центру строки. Для нее устанавливаем отслеживаемое событие `onMouseOver` и приписываем ему выполнение функции `lighting()`. В самой функции мы работаем с объектом `m`, который является телом документа, и используем его подчиненный объект `style` с графическим фильтром `"light"`. С помощью этого графического фильтра можно создавать различные световые эффекты. Затем мы описываем переменную `l`, которая становится первым (и единственным) экземпляром коллекции `filters`. А для нее уже выполняем метод `addPoint`, создающий точечный источник цвета. Первые три параметра этого метода содержат в себе координаты `x`, `y` и `z` точечного источника света, следующие три — RGB-код используемого цвета, а последний параметр — интенсивность света (рис. 3.22).

Необходимо отметить, что при каждом выполнении этой функции будет добавляться новый источник освещения. Поэтому всякий раз при прохождении курсора мыши над кнопкой освещенность будет увеличиваться.

У фильтра `light` помимо метода `addPoint`, создающего точечный источник света, есть еще несколько специфических методов, каждый из которых предназначен для работы с освещением. Метод `addAmbient(R, G, B, strength)` создает эффект общего освещения. В первых трех параметрах передается RGB-код цвета освещения, а в последнем параметре — интенсивность освещения.

Существует метод, создающий эффект освещения фонариком, коническим пучком света. Этот метод вызывается конструкцией `addCone(x, y, z, x1, y1, R, G, B, strength, spread)`. В первых трех параметрах передаются координаты источника света. Следующие два параметра содержат координаты точки на Web-странице, на которую будет направлен свет. Затем передается

RGB-код цвета освещения, интенсивность освещения и величина освещаемого круга.

Для перемещения источника света применяется метод `MoveLight(id, x, y, z, absolute)`. Первым параметром служит идентификатор источника света. Следующие три параметра — новые координаты источника. Последний параметр имеет логический тип. Если передаваемое значение — `True`, значит, применяемая система координат станет абсолютной, если было передано значение `False`, то переданные координаты будут использоваться как смещение, и система координат станет относительной. Ее базовой точкой будет исходное положение источника света.

Метод `ChangeStrength(id, new_strength, absolute)` позволяет динамически менять интенсивность источника света без его явного переопределения. В первом параметре передается идентификатор источника света, во втором — его новая интенсивность, а в третьем — логическое значение, указывающее на характер применяемой системы отсчета. Если передано значение `True`, то интенсивность становится именно той, которая указана во втором параметре. Если использовано значение `False`, то к уже имеющейся интенсивности будет добавлено значение, переданное во втором параметре.

Для очистки всех фильтров типа `light` применяется метод `Clear()`.

На этом мы заканчиваем обзор графических фильтров, а вместе с ним и последнюю главу этой книги.

Послесловие

Теперь оглянемся назад. На протяжении всей книги мы знакомились с языком HTML и сопутствующими ему технологиями стилевого оформления и интерактивности. Используя все приобретенные навыки для создания своих Web-страниц и Web-сайтов, вы обязательно поймете, что возможности, предоставляемые языком HTML, несмотря на их разнообразие и многочисленность, всего лишь средства для выражения идей. Это всего лишь *возможности*. Основа любого хорошего сайта — прежде всего, идея, которая увлечет других людей и заставит их приходить на ваш сайт снова и снова. Многообразие выразительных средств может стать ловушкой. Все мы часто встречали в Сети страницы с разнообразным стилевым оформлением, множеством интерактивных элементов, встроенными анимациями, но абсолютно неинтересным содержанием. При реализации собственных проектов всегда следует помнить принцип создания сада камней: "Пока есть, что убрать — работа не завершена". Хорошая идея, прозрачная навигация, продуманное и единое оформление — вот основные слагаемые хорошего сайта.

Однако для того, чтобы идеи и представления превратились в Web-сайт, обязательно следует знать язык HTML, с помощью которого создаются Web-страницы. Искренне надеюсь, что эта книга помогла вам в его изучении.

Удачных вам проектов. Встретимся в Сети.

Шапошников Игорь
webmaster@bhv.spb.su

ПРИЛОЖЕНИЕ 1

Символьные подстановки

Подстановка	Описание
&Agrave	Прописная A, тупое ударение À
&Aacute	Прописная A, сильное ударение Á
&Acirc	Прописная A, диакритическое ударение Â
&Atilde	Прописная A, тильда Ã
&Auml	Прописная A, умляут Ä
&Aring	Прописная A, звонкое произношение Å
&agrave	Строчная a, тупое ударение à
&aacute	Строчная a, сильное ударение á
&acirc	Строчная a, диакритическое ударение â
&atilde	Строчная a, тильда ã
&auml	Строчная a, умляут ä
&aring	Строчная a, звонкое произношение å
&Aelig	Прописные AE, дифтонг Æ
&aelig	Строчные ae, дифтонг æ
&Ccedil	Прописная C, седиль Ç
&ccedil	Строчная c, седиль ç
&Egrave	Прописная E, тупое ударение È
&Eacute	Прописная E, сильное ударение É
&Ecirc	Прописная E, диакритическое ударение Ê

(продолжение)

Подстановка	Описание
&Euml	Прописная E, умляут Ë
&egrave	Строчная e, тупое ударение è
&eacute	Строчная e, сильное ударение é
&ecirc	Строчная e, диакритическое ударение ê
&euml	Строчная e, умляут ë
&Iacute	Прописная I, сильное ударение Í
&Igrave	Прописная I, тупое ударение Ì
&Icirc	Прописная I, диакритическое ударение Î
&Iuml	Прописная I, умляут Ï
&iacute	Строчная i, сильное ударение í
&igrave	Строчная i, тупое ударение ì
&icirc	Строчная i, диакритическое ударение î
&iuml	Строчная i, умляут ï
&ETH	Сочетание Eth
&eth	Строчные eth
&Ntilde	Прописная N, тильда Ñ
&ntilde	Строчная n, тильда ñ
&Ograve	Прописная O, тупое ударение Ò
&Oacute	Прописная O, сильное ударение Ó
&Ocirc	Прописная O, диакритическое ударение Ô
&Otilde	Прописная O, тильда Õ
&Ouml	Прописная O, умляут Ö
&Oslash	Прописная O, слэш Ø
&ograve	Строчная o, тупое ударение ò
&oacute	Строчная o, сильное ударение ó
&ocirc	Строчная o, диакритическое ударение ô
&otilde	Строчная o, тильда õ
&ouml	Строчная o, умляут ö

(окончание)

Подстановка	Описание
&oslash	Строчная o, слэш ø
&Ugrave	Прописная U, тупое ударение Ù
&Uacute	Прописная U, сильное ударение Ú
&Ucirc	Прописная U, диакритическое ударение Û
&Uuml	Прописная U, умляут Ü
&ugrave	Строчная u, тупое ударение ù
&uacute	Строчная u, сильное ударение ú
&ucirc	Строчная u, диакритическое ударение û
&uuml	Строчная u, умляут ü
&Yacute	Прописная Y, сильное ударение Ý
&yacute	Строчная y, сильное ударение ý
&reg	Зарегистрированная торговая марка — Trademark ™
&copy	Права собственности — Copyright ©
	Неразрывный пробел

ПРИЛОЖЕНИЕ 2

Предустановленные обозначения цветов

Обозначение	Описание
`ActiveBorder`	Обозначает цвет границы текущего активного окна
`ActiveCaption`	Идентифицирует цвет, которым отображается строка заголовка активного окна
`AppWorkspace`	Идентичен цвету фона, на котором отображаются документы в многодокументных приложениях (например, в MS Word)
`Background`	Обозначает цвет фона рабочего стола (Desktop)
`ButtonFace`	Ссылается на цвет передней поверхности любого трехмерного элемента управления, такого как обычная кнопка
`ButtonHighlight`	Идентифицирует цвет светлых границ трехмерных стандартных кнопок
`ButtonShadow`	Обозначает цвет, применяемый системой для отображения темной границы (обычно правой и нижней) трехмерной кнопки
`ButtonText`	Ссылается на цвет, применяемый для отображения надписей на обычных трехмерных кнопках
`CaptionText`	Определяет цвет, идентичный тому, который применяется для отображения текста заголовка активного окна
`GrayText`	Обозначает цвет, которым отображается недоступный (disabled) текст. Чаще всего мы видим этот цвет в оформлении неактивных пунктов меню
`Highlight`	Идентифицирует стандартный цвет для фона текста, который пользователь выделил в каком-либо элементе управления

(окончание)

Обозначение	Описание
HighlightText	Ссылается на цвет, которым отображается текст выделенного элемента
InactiveBorder	Обозначает цвет, которым отображается граница неактивного окна
InactiveCaption	Ссылается на цвет, которым заполняется строка заголовка неактивного окна
InactiveCaptionText	Идентифицирует цвет, которым отображается текст на заголовке неактивного окна
InfoBackground	Идентичен цвету, используемому для фона всплывающих информационных окон
InfoText	Обозначает цвет, которым отображается текст информационных окон
Menu	Идентифицирует цвет фона элементов меню
MenuText	Ссылается на цвет, которым отображается текст пунктов меню
Scrollbar	Обозначает цвет серой области полос прокрутки
ThreeDDarkShadow	Ссылается на цвет, которым отображается темная граница трехмерных элементов управления
ThreeDFace	Указывает на цвет, которым отображается передний план трехмерных элементов управления
ThreeDHighlight	Ссылается на цвет, используемый для отображения выбранных пользователем подсвеченных трехмерных элементов управления
ThreeDLightShadow	Обозначает цвет, которым отображается светлая граница трехмерных органов управления
ThreeDShadow	Идентичен цвету ThreeDDarkShadow
Window	Ссылается на цвет фона окна
WindowFrame	Указывает на цвет дочернего окна
WindowText	Обозначает цвет, которым отображается обычный текст в окне

Предметный указатель

К

М

О

П

Р

С

Т

Продолжение рубрики см. на с. 288

У

Ф