Глава 2

Профессиональная верстка

Создать хороший сайт — дело нелегкое, а создать его профессионально и качественно — и совсем уж не простое. Но если вы хотите создать что-то действительно стоящее внимания, придется потрудиться.

Фреймы

С помощью фреймов можно разделить окно браузера на части. В результате пользователь, просматривающий страницу, сможет изучать каждую ее часть независимо от остальных частей.

Браузер, распознающий фреймы, загружает разные элементы веб-страницы в разные секции (или фреймы) своего окна. Например, можно организовать веб-страницу таким образом, что логотип будет зафиксирован в верхней части окна браузера, в то время как остальную ее часть пользователь сможет пролистывать обычным способом.

Чтобы лучше понять обсуждаемые вопросы, создадим пару простых страничек с фреймами. Это поможет нам понять внутреннее устройство HTML-страниц с фреймами и выяснить, для чего нужны основные дескрипторы и параметры, используемые при их описании.

```
<!-- Файл index.html -->
<HTML>
<HEAD>
<TITLE>Фреймы</TITLE>
</HEAD>
<FRAMESET COLS="20%, 75%">
<FRAME SRC="menu.html" NAME="main">
<FRAME SRC="base.html" NAME="bases">
</FRAMESET>
<NOFRAMES>
Установите себе браузер, который поддерживает фреймы!
</NOFRAMES>
</HTML>
```

В результате мы получим окно браузера, разделенное на две части. Левая занимает 20% окна браузера и содержит страницу с названием `menu.html`, а правая займет 75% окна браузера, там будет отображаться файл `base.html`. Поскольку указанные файлы пока отсутствуют, при запуске файла `index.html` мы увидим только страницу с двумя пустыми фреймами, как показано на рис. 2.1.

Обратите внимание, что правую страницу мы назвали *bases* (это важно). К тому же, как вы успели заметить, наш файл `index.html` не имеет дескриптора тела

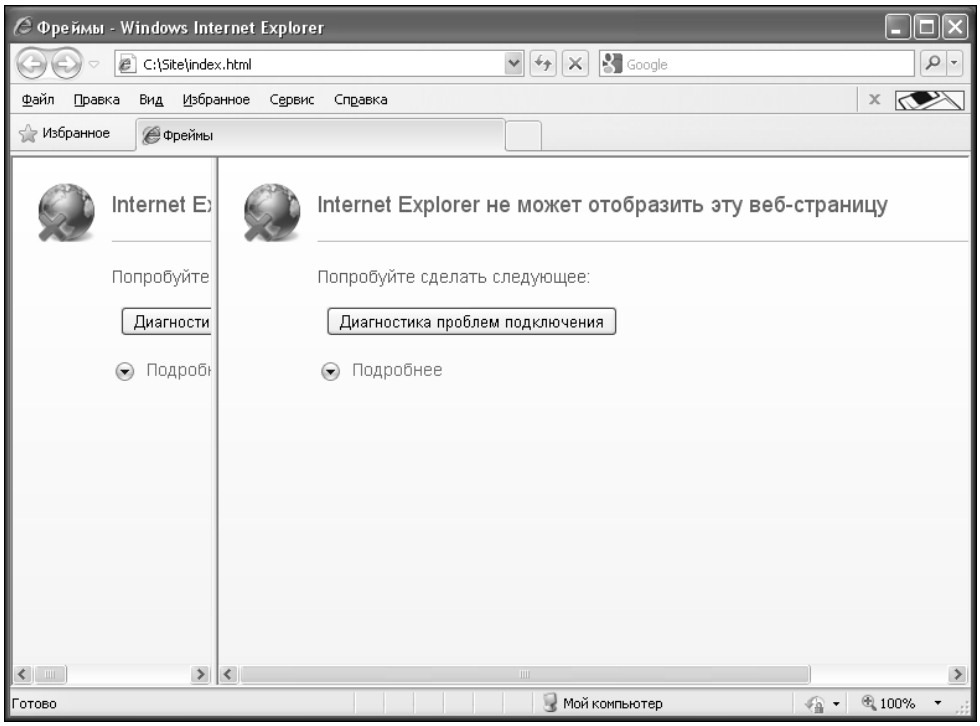


Рис. 2.1. *Исходный вид файла* `index.html` *в окне браузера*

<BODY>. Вся фреймовая структура находится между открывающим дескриптором <FRAMESET> и закрывающим дескриптором </FRAMESET>. Между дескрипторами <NOFRAMES>...</NOFRAMES> находится текст, который будет отображен пользователю, если его браузер не поддерживает фреймы. Имейте в виду, что поддерживающий фреймы браузер проигнорирует все, что находится между дескрипторами <NOFRAMES> и </NOFRAMES>. И наоборот, не поддерживающий фреймы браузер проигнорирует все, что находится между дескрипторами <FRAMESET> и </FRAMESET>. Код без фреймов можно поместить и в начало, и в конец страницы.

Теперь приступим к созданию меню — файла `menu.html`.

```
<!-- Файл menu.html -->
<HTML>
<HEAD>
<!-- Здесь мы определяем базовый элемент:
 присваиваем методу target значение bases.
 А это означает, что ссылки, находящиеся
 в этом документе, должны открываться в окне,
 которое имеет имя bases. Данное окно
 расположено в правой части фрейма -->
<base target="bases">
<TITLE>Меню</TITLE>
</HEAD>
<BODY>
<!-- Далее следуют названия меню и указываются ссылки -->
```

```
<H5 ALIGN="left">Наше меню:</H5>
<a href="base.html">Главная</a><p>
<a href="http://www.leninine.com/">Сайт автора</a><br>
<a href="http://www.dialektika.com/">Диалектика</a><br>
<a href="http://www.microsoft.com/">Майкрософт</a>
</BODY>
</HTML>
```

Осталось создать файл `base.html`, который может содержать любой текст, например:

```
<!-- Файл base.html -->
<HTML>
<HEAD>
<TITLE>Главная страница</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="CENTER">Новости сайта:</H1>
Ничего нового...
</BODY>
</HTML>
```

Вот теперь снова можно запустить файл `index.html` и полюбоваться своей работой. При этом все только что созданные HTML-документы должны находиться в одной папке. На рис. 2.2 представлен результат; то же самое должно было получиться и у вас. Если что-либо не работает, тщательно проверьте свой код. Теперь после щелчка

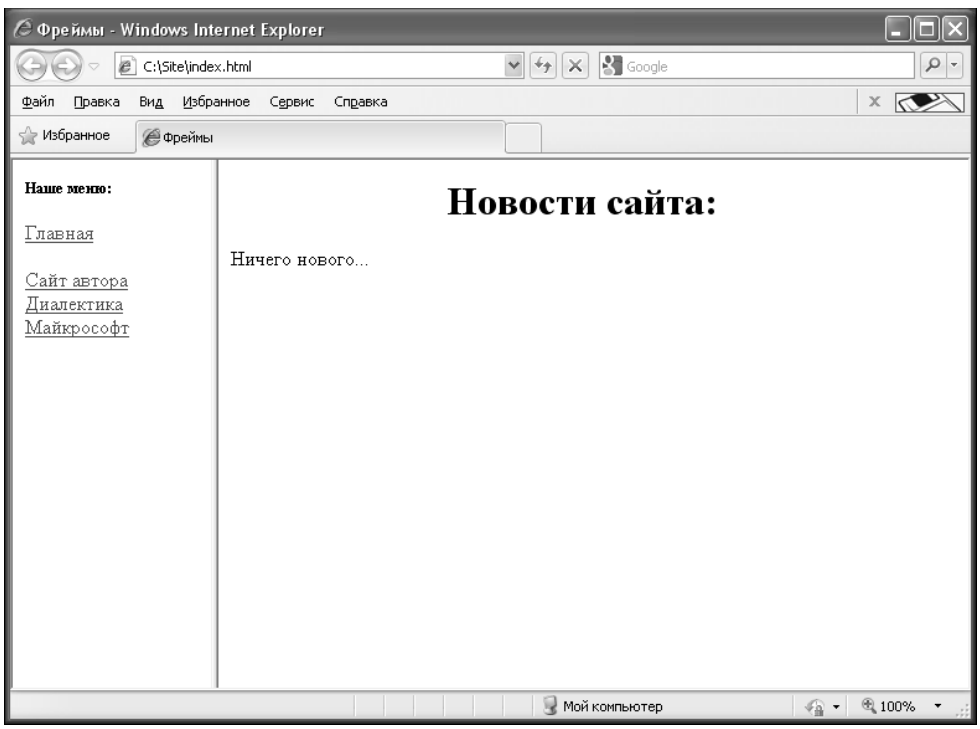


Рис. 2.2. *Вид окна браузера после создания файлов* `base.html` *и* `menu.html`

мыши на любом из пунктов меню в правой части браузера будет загружена соответствующая страница указанного сайта.

А теперь подробнее рассмотрим дескрипторы <FRAMESET> и <FRAME>.

В дескрипторе <FRAMESET> задается количество строк или столбцов (ROWS и COLS, соответственно), а также их размеры и расположение. Существует три способа задания размера строк и столбцов.

1. **Задание в пикселях**. Просто укажите высоту или ширину элемента в пикселях.
2. **Задание в процентах**. Укажите, сколько процентов от полного размера окна браузера следует отдать данному фрейму. После цифр обязательно поставьте знак "%". Также позаботьтесь о том, чтобы все указанные процентные значения в сумме составляли 100%.
3. **Использование символа звездочки**. С помощью символа звездочки элементу выделяется все оставшееся в окне браузера место. Например, можно написать 20%,20%,60% или 20%,20%,*, и никакой разницы в результатах не будет.

В этом же дескрипторе можно задать толщину разграничительной линии и окаймляющей рамки, для чего предназначены параметры FRAMEBORDER="*X*" и BORDER="*Y*", где *X* и *Y* — толщина соответствующих элементов в пикселях.

Дескриптор <FRAMESET> имеет следующий синтаксис.

```
<FRAMESET COLS="величины" ROWS="величины">
```

В дескрипторе <FRAME> задаются параметры для каждого фрейма в отдельности. Параметр SRC задает имя файла, который загрузится в данном фрейме. Параметр NAME задает имя данного фрейма. Это имя необходимо для того, чтобы впоследствии можно было указать, к какому именно фрейму относится ссылка. В этом дескрипторе можно также определить величину границы фрейма, за которую ничего, кроме фона, не может "заходить". Это осуществляется с помощью параметров MARGINWIDTH="*X*" и MARGINHEIGHT="*Y*", где *X* и *Y* — значения в пикселях.

Существует также очень полезный в некоторых случаях параметр SCROLLING, позволяющий потребовать наличие у фрейма полос прокрутки. Этот параметр может иметь значение YES, NO или AUTO. И конечно же, следует упомянуть о параметре NORESIZE. Этот параметр позволяет создавать фреймы, размеры которых в дальнейшем изменить будет невозможно. По умолчанию размер фрейма всегда можно изменить с помощью мыши так же просто, как и размер обычного окна Windows. Параметр NORESIZE отменяет подобную возможность. В целом, дескриптор <FRAME> имеет следующий синтаксис.

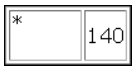
```
<FRAME SRC="URL" NAME="имя фрейма"
MARGINWIDTH="величина" MARGINHEIGHT="величина"
SCROLLING="yes|no|auto" NORESIZE>
```

В табл. 2.1 приведено несколько примеров создания фреймов.

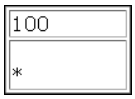
Фреймовая структура на первый взгляд кажется привлекательной, хотя бы из-за того, что определенную часть сайта, например, с логотипом или меню, можно всегда удерживать перед глазами пользователя. Тем не менее существует целый ряд недостатков сайта, построенного с применением фреймов. Вот некоторые из них.

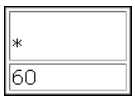
Таблица 2.1. Примеры описания фреймов различного типа

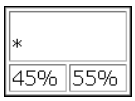
Вид фрейма	Код описания фрейма

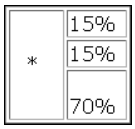
	`<FRAMESET cols="*,140">`
	`<FRAME SRC="homepage.htm" NAME="frame1">`
	`<FRAME SRC="menu.htm" NAME="frame2">`
	`</FRAMESET>`

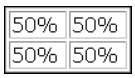
	`<FRAMESET cols="100,*">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAME SRC="menu.htm" NAME="Frame2">`
	`</FRAMESET>`

	`<FRAMESET rows="100,*">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAME SRC="menu.htm" NAME="Frame2">`
	`</FRAMESET>`

	`<FRAMESET rows="*,60">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAME SRC="menu.htm" NAME="Frame2">`
	`</FRAMESET>`

	`<FRAMESET rows="*,60">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAMESET cols="45%,55%">`
	`<FRAME SRC="menu.htm" NAME="Frame2">`
	`<FRAME SRC="menu2.htm" NAME="Frame3">`
	`</FRAMESET>`
	`</FRAMESET>`

	`<FRAMESET cols="*,55%">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAMESET rows="15%,15%,70%">`
	`<FRAME SRC="menu.htm" NAME="Frame2">`
	`<FRAME SRC="menu2.htm" NAME="Frame3">`
	`<FRAME SRC="menu3.htm" NAME="Frame4">`
	`</FRAMESET>`
	`</FRAMESET>`

	`<FRAMESET cols="50%,50%">`
	`<FRAMESET rows="50%,50%">`
	`<FRAME SRC="homepage.htm" NAME="Frame1">`
	`<FRAME SRC="homepage2.htm" NAME="Frame2">`
	`</FRAMESET>`
	`<FRAMESET rows="50%,50%">`
	`<FRAME SRC="menu.htm" NAME="Frame3">`
	`<FRAME SRC="menu2.htm" NAME="Frame4">`
	`</FRAMESET>`
	`</FRAMESET>`

- В случае попадания пользователя не на первую страничку сайта, например, через поисковый сервер, не существует “официального” способа перейти на его первую страничку — приходится вручную редактировать путь в адресной строке браузера.
- Ввиду того, что фреймовая структура сайта придает ему достаточно узнаваемый вид, большинство подобных страничек выглядит достаточно однообразно.

- Невозможно поставить закладку на внутреннюю страничку сайта. Представьте, что вы наткнулись на очень интересную статью и хотите, например, поместить ссылку на нее в свою коллекцию или послать ее адрес другу. Так вот: ни то, ни другое сделать не удастся — фреймы скрывают истинный адрес странички. Ради справедливости следует сказать, что этот адрес все же можно узнать с помощью обходного маневра — открывая ссылку в новом окне.

Хотя описанные выше проблемы являются довольно серьезными, их можно решить с помощью небольших и хитрых скриптов. В Интернете есть масса примеров того, как можно избежать подобных осложнений. Даже в этой книге есть некоторые из них, но рассмотрены они будут позже.

И все же есть один удачный способ применения фреймов — создание справочной системы для сложных сайтов. В этом случае очень удобно открывать новое окно, в котором можно использовать именно фреймовую структуру. В результате мы получим инструмент, очень похожий на встроенную справочную систему Windows формата `*.CHM`.

Еще одним главным недостатком фреймов всегда была невозможность выделения в произвольном месте страницы прямоугольной области заданных размеров `WIDTH` (ширина) и `HEIGHT` (высота). Ситуация изменилась с появлением дескриптора `IFRAME` — расширения стандарта языка HTML от корпорации Microsoft. Чаще всего данный дескриптор используют новостные сайты. Получается, что сайт состоит из скелета, в который подгружается информация с разных страниц, иногда даже какие-то части других сайтов, например, те, что показывают погоду, курсы валют.

Рассмотрим подробнее все особенности применения дескриптора `IFRAME`.

```
<IFRAME SRC="Имя файла" WIDTH="Ширина"
HEIGHT="Высота" NAME="Имя"
SCROLLING="yes/no/auto" [NORESIZE]>
</IFRAME>
```

Как видите, синтаксис дескриптора `IFRAME` близок к синтаксису обычного элемента `<FRAME>`. Например, можно задавать наличие/отсутствие полос прокрутки, запрещать/разрешать изменение размеров окна. Кроме того, поскольку данный фрейм есть ни что иное, как прямоугольная область со всеми вытекающими из этого последствиями, можно задать для него параметры размеров `WIDTH` и `HEIGHT`. Таким образом на своей странице можно вывести не только HTML-страницы и сторонние сайты, но и другие объекты, например текстовые файлы или изображения (рис. 2.3).

Применяя дескриптор `IFRAME`, можно часто менять содержимое отдельно взятого окна без изменения самой страницы. Конечно, для этих же целей можно воспользоваться, скажем, более продвинутой и сложной технологией SSI, речь о которой пойдет в следующем разделе.

Директивы SSI

Технология SSI (Server Side Includes — включения на стороне сервера) предусматривает использование специальных дескрипторов, а точнее директив, вставляемых непосредственно в HTML-код и предназначенных для передачи указаний веб-серверу. Встречая в тексте HTML-документа подобные директивы, называемые SSI-вставками, веб-сервер интерпретирует их и выполняет требующиеся от него дей-

ствия: вставляет фрагмент текста из другого HTML-файла, динамически формирует код веб-страницы в зависимости от значения определенных переменных (например, задающих тип браузера пользователя) и т.д. В целом, это технология несколько напоминает фреймы, не правда ли?

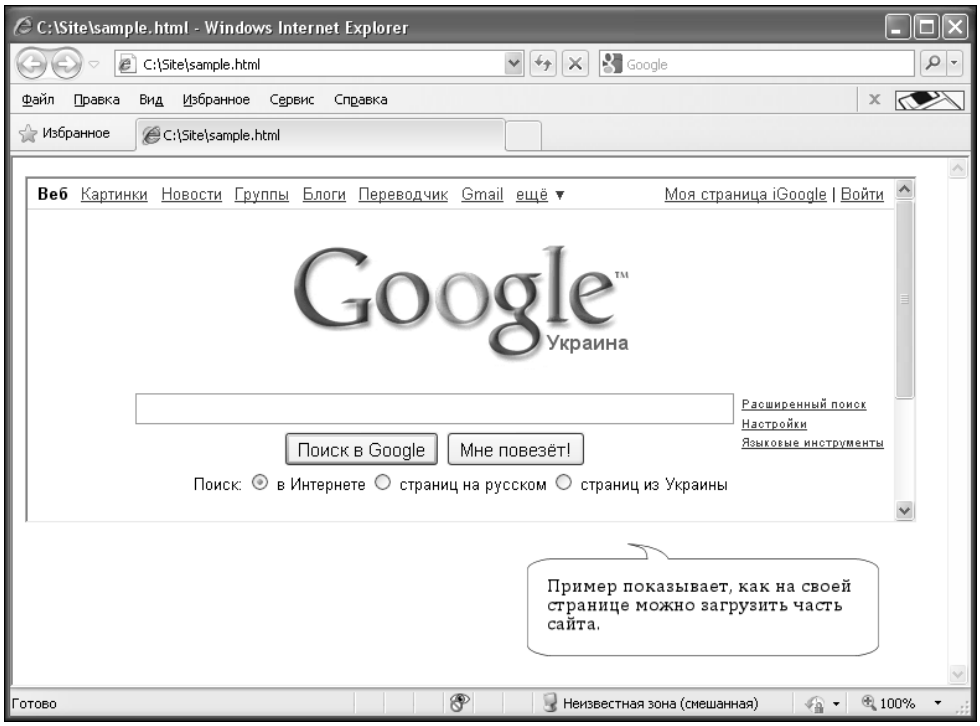


Рис. 2.3. *Один из вариантов применения дескриптора* `IFRAME`

Преимущества SSI особенно ощутимы, когда требуется поддерживать достаточно большой по объему сайт, имеющий определенную структуру с повторяющимися фрагментами HTML-кода на многих веб-страницах. Вообще говоря, при использовании серверных включений документы сайта удобно рассматривать как состоящие из отдельных блоков, каждый из которых включает определенную часть веб-страниц. Эти блоки практически неизменны и повторяются от страницы к странице. В подобные блоки можно вынести такие элементы, как главное меню, рекламные вставки, повторяющиеся элементы оформления и т.д. Физически эти блоки представляют собой просто HTML-файлы, содержащие ту часть кода, которая необходима для реализации поставленных задач.

Для того чтобы сервер знал, что данная веб-страница является не обычным HTML-документом, а содержит SSI-директивы, ее файлу присваивается специальное расширение имени: `*.shtml` или `*.shtm`. Наличие у файла такого расширения имени вынуждает веб-сервер выполнять предварительную обработку данной веб-страницы. Следует отметить, что расширение имени в принципе может быть любое — конкретное значение устанавливается при выполнении конфигурации данного веб-сервера, но чаще всего применяется именно `*.shtml`.

Полный текст страницы формируется веб-сервером "на лету", в процессе сборки ее кода из таких вот блоков. Для того чтобы указать серверу, какой именно блок нужно вставить и в каком месте результирующей веб-страницы, используется специальная форма записи, оформляемая в виде комментария.

```
<!--#command param="value" -->
```

Здесь знак "#" является признаком начала SSI-вставки; *command* задает SSI-команду, *param* определяет параметры SSI-команды, а *value* указывает значения этих параметров.

Всего различных SSI-команд насчитывается около десятка, однако в данной главе мы обсудим только самые используемые из них. Наиболее популярная SSI-команда — это команда включения содержимого одного файла в другой.

```
<!--#include virtual="/path/file.ssi" -->
```

Здесь include — код команды вставки; virtual — параметр, определяющий, каким образом трактовать указанный в команде путь — как абсолютный (значение file) или как относительный (значение virtual); /path/file.ssi — путь к включаемому файлу.

Результатом выполнения данной SSI-команды будет вставка содержимого файла file.ssi в то место исходного HTML-документа, где эта команда расположена. При просмотре сформированного HTML-файла мы не увидим никаких признаков SSI, так как данный механизм действует абсолютно прозрачно для браузеров, которые получают исключительно уже готовый и корректный HTML-код.

Следующий пример — это команда установки значения переменной.

```
<!--#set var="pic" value="picture.gif"-->
```

Здесь var — код команды присвоения значения переменной; pic — имя переменной, которой присваивается указанное значение; picture.gif — то значение, которое должно быть присвоено переменной pic.

В результате выполнения сервером приведенной выше SSI-команды создается переменная с именем pic, и ей присваивается строковое значение "picture.gif". Вновь созданная переменная становится доступной в пределах SSI-вставки, и ее можно использовать по собственному усмотрению. Например, применяя в разных документах одну и ту же SSI-вставку, но с разными значениями определенной в ней переменной, мы получим различные результаты. Прежде чем переходить к обсуждению реального примера использования переменных в SSI-включениях, рассмотрим некоторые команды, предназначенные для работы с переменными. Во-первых, это команда вставки значения переменной.

```
<!--#echo var="pic" -->
```

Выполнение этой команды приведет к тому, что в месте ее размещения будет вставлено значение переменной pic, т.е. picture.gif.

Переменная может участвовать в выражениях, в этом случае перед ней ставится знак $, показывающий, что это именно переменная, а не просто текст.

```
<!--#set var="A" value="123" -->
<!--#set var="B" value="$A456" -->
```

В результате выполнения этих двух команд присвоения переменная B будет содержать строку 123456. Если же в текст необходимо просто вставить собственно знак $

или какой-нибудь другой из специальных знаков, то перед ним следует поместить знак обратной косой черты (слеш): `\$`. В некоторых случаях для избежания двусмысленности значение переменной может быть заключено в фигурные скобки: `${A}`.

Более сложное применение механизма переменных возможно за счет использования условных операторов, имеющих следующий синтаксис.

```
<!--#if expr="condition" -->
<!--#elif expr="condition" -->
<!--#else -->
<!--#endif -->
```

Здесь значение *condition* представляет собой анализируемое условие.

В зависимости от результатов проверки условия, в генерируемый текст подставляется тот или иной фрагмент кода. Например, можно потребовать от сервера проанализировать тип браузера пользователя и в зависимости от полученного результата вставить в создаваемый HTML-документ специализированный код либо для браузера Opera, либо для браузера Internet Explorer. Такой подход может оказаться полезным в некоторых специфических случаях, когда невозможно создать такую вебстраницу, которая корректно отображалась бы в обоих браузерах. Вот пример использования условного оператора.

```
<!--#set var="Monday" -->
<!--#if expr="$Monday" -->
<p> Сегодня понедельник.</p>
<!--#else -->
<p> Что угодно, но не понедельник.</p>
<!--#endif -->
```

В данном случае условие состоит в проверке существования переменной `$Monday` и, в зависимости от этого, подстановка в создаваемый документ того или иного HTML-кода.

Теперь рассмотрим реальный пример применения технологии SSI для формирования сложного документа из нескольких SSI-вставок. Вначале напишем текст основного HTML-документа, полагая, что SSI-вставки будут находиться в папке `/ssi`.

```
<!--Файл index.shtml-->
<!--#set var="title" value="Что такое SSI?" -->
<!--#set var="keywords" value="SSI, SHTML, CGI,
 Apache" -->
<!--#set var="description" value="Пример
 использования SSI." -->
<!--#include virtual="ssi/header.shtml" -->
<!--Здесь находится текст нашей странички-->
<!--#include virtual="ssi/footer.shtml" -->
<!--Файл header.shtml-->
<html>
<head>
<title><!--#echo var="title" --></title>
<meta name="keywords" content="
<!--#echo var="keywords" -->">
<meta name="description" content="
<!--#echo var="description" -->">
</head>
<body>
```

```
<!--Файл footer.shtml-->
</body>
</html>
```

Как видите, основной документ предельно упрощен и состоит из директив, устанавливающих значения переменных `title`, `keywords` и `description`, которые и будут подставлены в HTML-код отправляемой пользователю веб-странички при обработке SSI-вставок, определяющих содержание для ее верхней и нижней частей. Реальный код SSI-вставок обычно гораздо сложнее и может включать в себя большее количество определяемых переменных и сложных условий, формирующих окончательный вид генерируемого документа.

Первое преимущество технологии SSI, с точки зрения дизайнера, состоит в том, что при ее использовании веб-мастеру, ответственному за поддержку сайта, можно не бояться случайно испортить дизайн документов. Благодаря использованию SSI, элементы сложной компоновки страниц оказываются скрытыми, и поддержка актуальности содержимого отдельных веб-страниц становится значительно более простым и приятным делом.

Второе, не менее важное, преимущество — это возможность очень быстрого, почти мгновенного изменения дизайна документов сайта, не требующая тотальной переделки всех страниц с информационным содержанием. Для смены дизайна всего сайта достаточно переписать SSI-вставки, формирующие внешний вид документов сайта. В нашем примере это файлы `header.shtml` и `footer.shtml`.

Серверы могут не поддерживать технологию SSI. Об этом обычно сообщается в описании условий хостинга. Но проще всего обратиться в службу технической поддержки за разъяснениями.

Таблицы

Таблицы в HTML являются одним из самых удачных и простых элементов форматирования различных объектов и текста. Используя таблицы, можно создавать в HTML-документе такие эффекты, как размещение текста в несколько колонок, состыковку картинки и фона, помещение тонких линий на всю ширину или высоту странички и т.д.

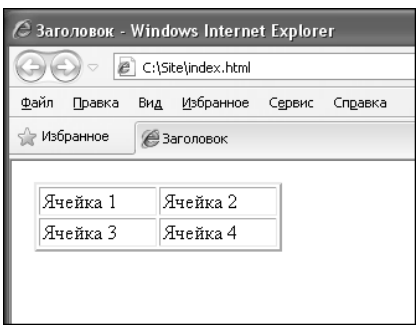


Рис. 2.4. *Пример таблицы с двумя столбцами и четырьмя ячейками*

Чтобы разобраться в построении HTML-таблицы, попробуем создать таблицу, которая показана на рис. 2.4. В данном случае это таблица с фиксированной шириной в 200 пикселей, однако иногда при задании размеров используются процентные значения, поскольку в этом случае размер таблицы будет изменяться в зависимости от размера окна браузера пользователя — растягиваться и сжиматься.

Этот пример реализуется следующим кодом.

```
<TABLE BORDER="2" WIDTH="200">
<TR>
<TD>Ячейка 1</TD>
```

```
<TD>Ячейка 2</TD>
</TR>
<TR>
<TD>Ячейка 3</TD>
<TD>Ячейка 4</TD>
</TR>
</TABLE>
```

Таблица начинается открывающим дескриптором <TABLE> и завершается закрывающим дескриптором </TABLE>. Дескриптор <TABLE> может включать параметры, описанные в табл. 2.2.

Таблица 2.2. Параметры дескриптора <TABLE>

Параметр	Описание
WIDTH	Ширина таблицы в пикселях или процентах. По умолчанию ширина таблицы определяется содержимым ячеек
BORDER	Толщина рамки таблицы. По умолчанию таблица рисуется без рамки – т.е. параметру BORDER присваивается значение 0
BORDERCOLOR	Цвет рамки
BGCOLOR	Цвет фона для всей таблицы
BACKGROUND	Заполнение фона таблицы изображением
CELLSPACING	Расстояние между рамками ячеек таблицы в пикселях
CELLPADDING	Расстояние в пикселях между рамкой ячейки и текстом
ALIGN	Определение расположения таблицы в документе. По умолчанию таблица прижата к левому краю страницы. Допустимые значения: LEFT (слева), CENTER (по центру страницы) и RIGHT (справа)
FRAME	Управление выводом внешней рамки таблицы. Допустимы следующие значения: VOID — рамки нет (значение по умолчанию); ABOVE — только линия сверху; BELOW — только линия снизу; HSIDES — линии сверху и снизу; VSIDES — только линии слева и справа; LHS — только линия слева; RHS — только линия справа; BOX — полная рамка; BORDER — также рисуется вся рамка
RULES	Управление отображением линий, разделяющих ячейки таблицы. Возможные значения: NONE — нет разделяющих линий (значение по умолчанию); GROUPS — линии будут только между группами рядов; ROWS — только между строками; COLS – только между столбцами; ALL — между всеми строками и столбцами

Таблица может включать заголовок, который располагается между дескрипторами <CAPTION>...</CAPTION>. Этот дескриптор должен следовать непосредственно после дескриптора <TABLE>. В дескрипторе заголовка допускается указание параметра ALIGN, определяющего его положение относительно таблицы:

- TOP — значение по умолчанию, заголовок над таблицей по центру;
- LEFT — заголовок над таблицей слева;
- RIGHT — заголовок над таблицей справа;
- BOTTOM — заголовок под таблицей по центру.

Теперь поговорим о строках и отдельных ячейках таблицы. Каждая строка таблицы начинается открывающим дескриптором <TR> и завершается закрывающим дескриптором </TR>, а каждая ячейка таблицы начинается дескриптором <TD> и завершается дескриптором </TD>. Данные дескрипторы могут иметь параметры, описанные в табл. 2.3.

Таблица 2.3. Параметры дескрипторов <TR> и <TD>

Параметр	Описание
Параметры, допустимые для дескрипторов <TR> и <TD>	
ALIGN	Горизонтальное выравнивание текста в ячейках строки. Может принимать следующие значения: LEFT — выравнивание влево; CENTER — выравнивание по центру; RIGHT — выравнивание вправо
VALIGN	Вертикальное выравнивание текста в ячейках строки. Допустимые значения: TOP — выравнивание по верхнему краю; CENTER — выравнивание по центру (это значение принимается по умолчанию); BOTTOM — выравнивание по нижнему краю
BGCOLOR	Цвет фона строки или ячейки
BACKGROUND	Заполнение фона строки или ячейки изображением
Параметры, применяемые только для дескриптора <TD>	
WIDTH	Ширина ячейки в пикселях
HEIGHT	Высота ячейки в пикселях
COLSPAN	Растягивание ячейки по горизонтали. Например, <TD COLSPAN="2"> означает, что ячейка будет растянута на 2 столбца таблицы
ROWSPAN	Растягивание ячейки по строке. Например, <TD ROWSPAN="2"> означает, что ячейка будет растянута на две строки таблицы
NOWRAP	Присутствие этого параметра показывает, что текст должен размещаться в одну строку
BACKGROUND	Заполнение фона ячейки изображением

Выравниванию объектов в таблицы следует уделить побольше внимания. Без этих знаний вам будет тяжело добиться нужного расположения объектов и текста. Обратите внимание на рис. 2.5. Здесь представлены различные варианты расположения, причем два нижних из них — это комбинация параметров.

Любая ячейка таблицы может быть определена не дескрипторами <TD>…</TD>, а дескрипторами <TH>...</TH> (Table Header — заголовок таблицы). В принципе это обычная ячейка, но текст, ограниченный этими дескрипторами, будет выделен полужирным шрифтом и отцентрирован. В табл. 2.4 приведено несколько примеров различных вариантов таблиц.

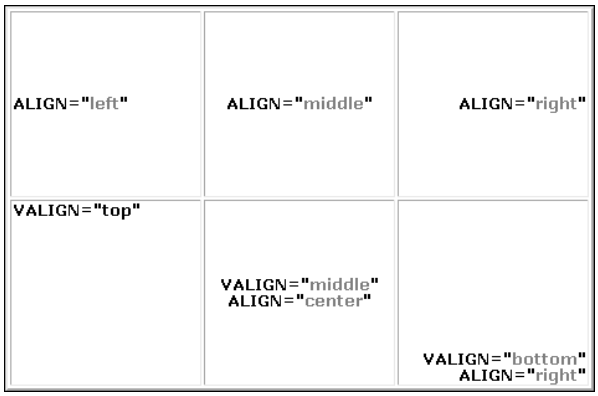


Рис. 2.5. *Примеры выравнивания в табличных ячейках*

Если ячейка пуста, то рамка вокруг нее не отображается. Если все же рамка требуется и вокруг пустой ячейки, то в нее надо ввести символьный объект ` ` (этот объект представляет собой non-breaking space — неразрывный пробел). Ячейка по-прежнему будет пуста, но рамка вокруг нее будет отображаться. (Значение ` ` обязательно должно набираться строчными буквами и завершаться точкой с запятой.)

Таблица 2.4. Примеры таблиц

Вид таблицы	Код таблицы

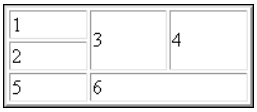
1 2	`<table width="200" border="2">`
	`<tr>`
	`<td>1</td>`
	`<td>2</td>`
	`</tr>`
	`</table>`

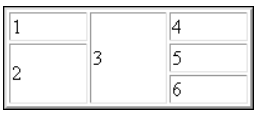
1
2
	`<table width="200" border="2">`
	`<tr>`
	`<td>1</td>`
	`</tr>`
	`<tr>`
	`<td>2</td>`
	`</tr>`
	`</table>`

1 2
3
4 5
	`<table width="200" border="2">`
	`<tr>`
	`<td rowspan="2">1</td>`
	`<td>2</td>`
	`</tr>`
	`<tr>`
	`<td>3</td>`
	`</tr>`
	`<tr>`
	`<td>4</td>`
	`<td>5</td>`
	`</tr>`
	`</table>`

Окончание табл. 2.4

Вид таблицы	Код таблицы

	`<table width="200" border="2">`
	`<tr>`
	`<td>1</td>`
	`<td rowspan="2">3</td>`
	`<td rowspan="2">4</td>`
	`</tr>`
	`<tr>`
	`<td>2</td>`
	`<tr>`
	`<td>5</td>`
	`<td colspan="2">6</td>`
	`</tr>`
	`</table>`

	`<table width="200" border="2">`
	`<tr>`
	`<td>1</td>`
	`<td rowspan="3">3</td>`
	`<td>4</td>`
	`</tr>`
	`<tr>`
	`<td rowspan="2">2</td>`
	`<td>5</td>`
	`</tr>`
	`<tr>`
	`<td>6</td>`
	`</tr>`
	`</table>`

	`<table width="200" border="2">`
	`<tr>`
	`<td>1</td>`
	`<td>2</td>`
	`<td>3</td>`
	`<td>4</td>`
	`</tr>`
	`<tr>`
	`<td colspan="4">5</td>`
	`</tr>`
	`</table>`

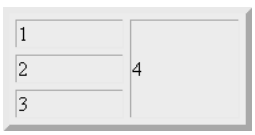
	`<table width="200" border="5" cellspacing="5"`
	`bgcolor="#FFF000">`
	`<tr>`
	`<td>1</td>`
	`<td rowspan="3">4</td>`
	`</tr>`
	`<tr>`
	`<td>2</td>`
	`</tr>`
	`<tr>`
	`<td>3</td>`
	`</tr>`
	`</table>`

Таблица — конструкция нежная. При потере хотя бы одного дескриптора <TR> она может повести себя непредсказуемо, а отсутствие завершающего элемента </TABLE> может привести к тому, что вся таблица вообще исчезнет с экрана.

И еще одно замечание — имейте в виду, что дескрипторы, задающие начертание шрифта (<B>, <I>, <FONT SIZE="*n*", FONT COLOR="*color*">), *необходимо повторять в каждой ячейке*.

При помощи таблиц можно осуществить самый замысловатый дизайн, расположить содержимое практически как угодно. На рис. 2.6 представлен пример стандартного дизайна окна сайта, которым просто "кишит" Интернет.

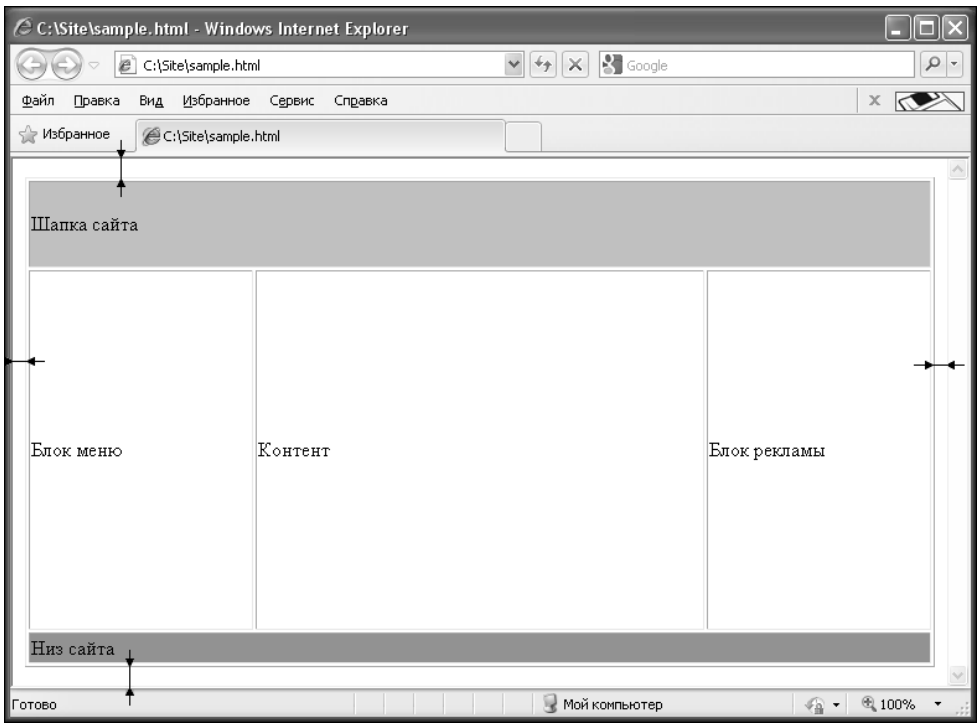


Рис. 2.6. *Пример стандартного табличного дизайна с полями*

Обратите внимание на пространства между таблицей и краями браузера. Для наглядности я отметил его стрелками. Устраняется данный эффект параметрами дескриптора <BODY>, которыми регулируются размеры полей во всем документе. Они задаются числовыми значениями и имеют следующие имена:

- `leftmargin` — левое поле;
- `rightmargin` — правое поле;
- `topmargin` — верхнее поле;
- `bottommargin` — нижнее поле.

Например, в приведенном ниже дескрипторе размеры полей устанавливаются равными нулю на всех ее краях.

```
<body leftmargin="0" rightmargin="0" topmargin="0" bottommargin="0">
```

Указанные параметры отлично работают в браузере Internet Explorer, а вот для других обозревателей надо использовать иные параметры:

- `marginwidth` — поле слева и справа;
- `marginheight` — поле сверху и снизу.

Для того чтобы создать одинаковое отображение полей в разных браузерах, будет достаточно воспользоваться такой конструкцией.

```
<body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
```

В этом случае Internet Explorer "видит" только свои параметры, а остальные браузеры — свои. Подобное написание кода в веб-мастеринге называется *кроссбраузерностью* — это одинаковый показ вашего сайта в различных браузерах, таких как Firefox, IE, Opera и др. Кроссбраузерность способствует популярности сайта, поскольку обеспечивает возможность работать с вашим ресурсом бо́льшему числу людей.

В дескрипторе `<BODY>` могут также присутствовать и другие параметры, например, базовые цвета текста и ссылок, которые задаются для всего документа; некоторые функции языка JavaScript.

Давайте реализуем табличную верстку на практике.

```
<!--Пример HTML-документа.-->
<html>
<head>
<title>Заголовок</title>
</head>
<!--Тело HTML-документа. Убираем поля со всех сторон.-->
<body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">

<!--Рисуем таблицу и растягиваем ее 100% во все стороны.-->
<table border="1" width="100%" height="100%">
<tr>
<td colspan="3" height="70" bgcolor="#c0c0c0">Шапка сайта</td>
</tr>
<tr>
<td width="25%">Блок меню</td>
<td>Контент</td>
<td width="25%">Блок рекламы</td>
</tr>
<tr>
<td colspan="3" height="25" bgcolor="#989090">Низ сайта</td>
</tr>
</table>

</body>
</html>
```

Таким образом мы получили шаблон сайта, который сверстан с применением таблицы, растянутой до краев окна браузера (рис. 2.7). Осталось только наполнить его необходимой информацией.

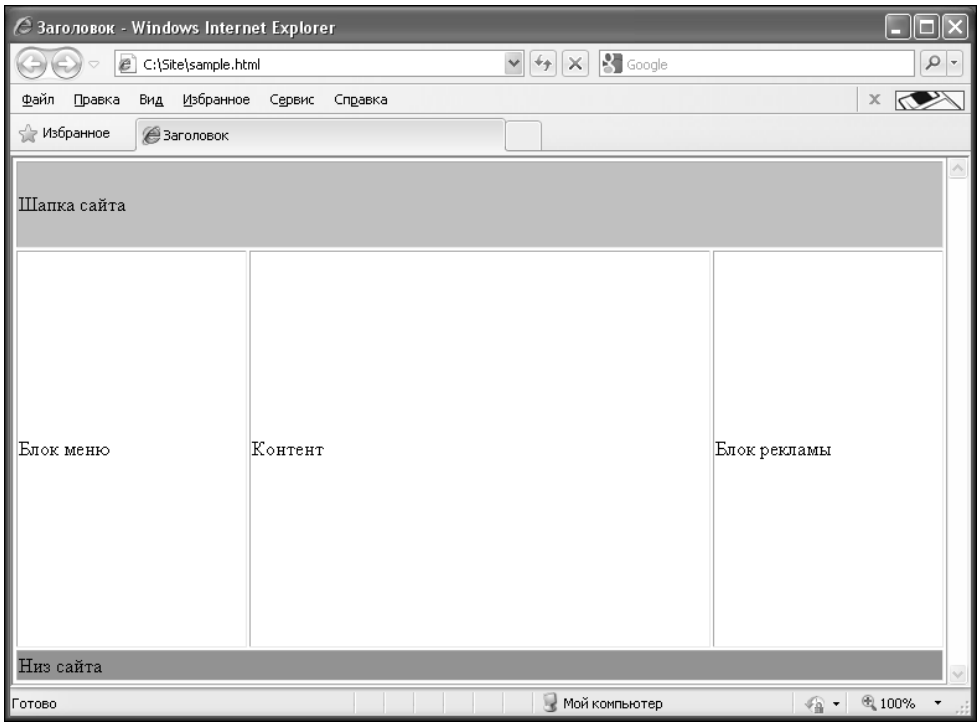


Рис. 2.7. *Пример стандартного табличного дизайна без полей*

Табличная верстка считается одной из самых удобных. Но последнее время большей популярностью пользуется *дивовый* дизайн — верстка при помощи дескриптора `<DIV>`. Об этом мы поговорим в следующем разделе.

Блоки CSS

Для чего нужен CSS?

Возможности обычного языка HTML позволяют задавать цвет и размер текста с помощью дескрипторов форматирования. Если понадобится изменить параметры однотипных элементов на сайте, придется просматривать все страницы, чтобы найти и поменять все соответствующие дескрипторы.

Для более простого изменения дизайна всего сайта была разработана технология **CSS** (Cascading Style Sheets, каскадные таблицы стилей) — в сущности, это набор параметров форматирования, который применяется к элементам веб-страницы для управления их видом и положением. Стили являются удобным, практичным и эффективным инструментом при верстке веб-страниц и оформлении текста, ссылок, изображений и других элементов.

Стили, как правило, хранятся в одном или нескольких специальных файлах, ссылка на которые указывается во всех документах сайта. Благодаря этому удобно вносить правки — стиль корректируется в одном месте, и при этом оформление эле-

ментов автоматически меняется на всех страницах, которые связаны с указанным файлом и используют данный стиль. При хранении стилей в отдельном файле последний кешируется браузером и при повторном обращении к нему извлекается из кеша, без обращения к серверу. За счет кеширования и того, что стили хранятся в отдельном файле, уменьшается размер кода веб-страниц и снижается время загрузки документов.

Кешем называется специальное место на локальном компьютере пользователя, куда браузер сохраняет файлы сайта при первом обращении к нему. При следующем обращении к тому же сайту такие файлы уже скачиваются не из сети, а с локального диска. Подобный подход позволяет существенно повысить скорость загрузки просматриваемых веб-страниц.

Синтаксис стилей

Стили, или, точнее, блок определения стилей, помещаются либо в заголовок страницы, между дескрипторами <HEAD>...</HEAD>, либо в отдельный файл. При размещении в разделе заголовка блок определения стилей начинается и заканчивается тегами <STYLE>...</STYLE>, а для исключения отображения его содержимого последнее заключается в знаки комментария <!-- и --> так же, как и в HTML.

```
<STYLE><!-- блок стилей --></STYLE>
```

Если вы размещаете стили в отдельном файле, тогда в разделе заголовка помещается ссылка на соответствующий CSS-файл.

```
<link rel="stylesheet" type="text/css" href="Файл_CSS">
```

В данном примере Файл_CSS — это имя файла каскадных стилей, который обязательно должен иметь расширение *.CSS и содержать описания применяемых стилей. При этом сам файл не должен содержать дескрипторы <STYLE>...</STYLE>! В общем-то, это обычный текстовый документ, только со своим специфическим расширением и определенным набором символов.

Поскольку определение стиля возможно на различных языках, в объявлении блока STYLE и ссылке на файл с определениями стилей следует указать в параметре TYPE язык определения. Для CSS, в частности, следует указать TYPE="text/css".

Выбор места размещения стилевого блока зависит от нескольких обстоятельств. Нет смысла выносить в отдельный файл определения стилей, если они предназначены для единственного документа. Вынесение определения стилей в отдельный файл имеет смысл только тогда, когда эти стили одинаковы для большего числа документов, например, всех страниц целого сайта — в результате отпадет необходимость вписывать определения в каждую страницу. Кроме того, ускорится загрузка: загрузив определение с первой страницей сайта, браузер не будет вновь обращаться за ним к серверу и таким образом сократит время загрузки следующих страниц. При этом если на какой-то из страниц определенный в этом файле стиль должен отображаться иначе, его можно переопределить, указав загрузку дополнительного файла определений, число которых не лимитируется, или указав новые параметры стиля в блоке STYLE после ссылки на файл определений.

Кроме того, стили можно указывать во всех дескрипторах и изменять только конкретные из них. Делается это при помощи параметра STYLE, который имеет следующий вид: внутри угловых скобок какого-нибудь дескриптора, отделенный от имени

тега пробелом, указывается параметр STYLE, затем знак равенства и в кавычках перечень свойств и их значений, разделенных точкой с запятой. Свойства отделяются от значений двоеточием, например:

```
<P style='color: green; border: solid red;'>
```

Определение стилей в заголовке страницы или отдельном файле происходит почти так же, только тег STYLE, указанный вначале определения, не повторяется, а роль кавычек выполняют фигурные скобки.

```
P {color: green; border: solid red;}
```

Поскольку для HTML-документа пробел и перенос строки равносильны, эту запись можно отобразить несколькими строками для визуального удобства.

```
P {
 color: green;
 border: solid red;
 }
```

Во внутрь определения стиля может быть помещен комментарий, заключенный между элементами /* и */.

Для дескрипторов, имеющих одинаковые определения, допускается группировка, при которой они записываются в одну строку через запятую, а затем указываются их свойства и значения свойств.

```
P, BR {margin-top: 0; margin-bottom: 0;}
```

В одном блоке STYLE может быть множество стилей. Допустимое количество самих блоков также неограничено. При этом, если определения стилей находятся в отдельных файлах, предпочтение отдается первому из них — следующий ищется только в случае невозможности загрузить или интерпретировать первый; если же одни и те же стили несколько раз определены в самом документе (что может быть вызвано только невнимательностью автора), предпочтение отдается последнему.

Рассмотрим пример определения стилей для нескольких дескрипторов.

```
<STYLE type="text/css">
<!--
A:link {color: #F30; background: transparent;}
A:visited {color: #363; background: transparent;}
A:active {color: #F30; background: transparent;}
A:hover {background: #FFA;}
-->
</STYLE>
```

В этом примере приведены специальные селекторы A:link, A:visited, A:active и A:hover, означающие: не посещенную гиперссылку, посещенную гиперссылку, открываемую в данный момент гиперссылку и гиперссылку, над которой находится указатель мыши. Таким образом эти дескрипторы не потребуется указывать в теле документа (следует указать лишь <A> с необходимыми ему параметрами). Браузер сам определит состояние ссылки и соответствующим образом оформит ее внешний вид.

Вышеприведенный пример для гиперссылок также демонстрирует, что CSS допускает наличие нескольких видов оформления одних и тех же дескрипторов, поэтому понятие дескриптора становится слишком узким для применения в CSS и заменяется

понятием *селектора*. Селектор — это имя, данное элементу страницы, для которого определяется стиль отображения. Оперирование селекторами имеет принципиальное значение для таблиц стилей, поскольку позволяет создавать различное оформления для одних и тех же дескрипторов.

Для того чтобы выделить дескриптор в отдельный класс оформления, после его имени, через точку, записывается произвольное название класса.

```
<STYLE type="text/css">
<!--
SPAN.MyClass {color: red; font: bold 14px/12px Arial;}
-->
</STYLE
```

При необходимости обращения к этому классу в объявлении дескриптора указывается класс. На дескрипторы, имеющие то же имя, но принадлежащие к другому классу, параметры отображения указанного класса не повлияют.

```
<SPAN>Здесь идет обычный текст, который <SPAN class="MyClass"> меняется на
текст с другими параметрами</SPAN>, а далее становится снова обычным.</SPAN>
```

В HTML-документе этот пример будет выглядеть так, как показано на рис. 2.8.

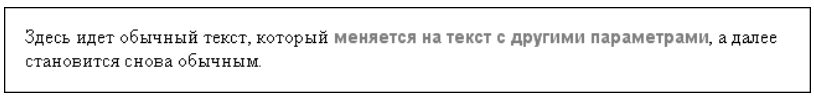


Рис. 2.8. *Пример изменения определенного фрагмента текста с помощью стилей*

В определении классов, так же как и с остальными дескрипторами, допустима их группировка через запятую.

```
SPAN.MyClass, DIV.MyClass {описание стилей}
```

Если же произвольное имя класса является уникальным и не входит в состав других селекторов, внутри объявления стилей на него можно ссылаться, опуская имя дескриптора, но обязательно начиная с точки.

```
.MyClass {описание стилей}
```

Кроме классов в CSS существует и такое понятие, как *идентификаторы*. Они при записи содержат знак решетки.

```
#MyClass {описание стилей}
```

А внутри дескрипторов указываются через параметр `ID`.

```
<SPAN id="MyClass">стилизированный текст</SPAN>
```

Вместо создания отдельного стиля представления содержимого, CSS позволяет определять его условное оформление, зависящее от контекста, для чего в определении стилей указывается дескриптор, в контексте которого элемент должен иметь особое оформление, а через пробел от него следующий дескриптор оформляемого элемента с перечислением параметров отображения.

```
H1 EM {color: red;}
```

Элемент `EM`, являющийся дескриптором языка HTML, будет оформлен по тексту как обычный `EM`, но внутри заголовка первого уровня примет указанное оформление.

Фактически контекстные селекторы мало отличаются от обычных. Единственное отличие заключается в отсутствии необходимости указывать класс элемента при его объявлении в теле документа. Вот пример оформления текста с помощью обычного селектора, идентификатора и контекстного селектора.

```
<!--Пример HTML-документа.-->
<html>
<head>
<title>Заголовок</title>
<!--Блок стилей.-->
<style type="text/css">
 <!--
  P.Red {color: red;}
  .Red small {background: #c0c0c0; color: black;}
  #Red {color: red;}
  H1 EM {color: red;}
 -->
</style>
</head>
<!--Тело HTML-документа.-->
<body>
<!--Текст, отформатированный с помощью стилей.-->
<H1>Заголовок, стиль которого не определялся, и <EM>красный элемент в нем</
EM></H1>
<p>Текст обычного абзаца, стиль которого не определялся, и <em>элемент</em>
стандартного оформления, рядом с которым <span id=Red>красный элемент</span>,
определенный с помощью идентификатора.
<p class=Red>Полностью красный абзац и <small>внутри него элемент SMALL
с определенным фоном и черным текстом</small>.</p></p>
</body>
</html>
```

В окне браузера этот пример будет выглядеть так, как показано на рис. 2.9.

Наиболее важные свойства элементов, которыми можно управлять с помощью технологии CSS, описаны в приложении Г "Элементы CSS".

Условные комментарии для IE

Как правило, разные версии браузера Internet Explorer по-разному понимают те или иные элементы CSS. Чтобы как-то исправить эту ситуацию, разработчики IE реализовали в нем довольно полезное свойство — поддержку условных комментариев. Условный комментарий — это конструкция особого вида, которая воспринимается всеми браузерами, *кроме IE*, как обычный комментарий. Таким образом, можно писать CSS-код для каждой версии браузера IE отдельно или для целой группы версий.

Синтаксис условных комментариев следующий.

```
<!--[if  "условие"]>
 /* здесь CSS-код или путь к файлу CSS
    здесь же может располагаться и HTML-код */
<![endif]-->
```

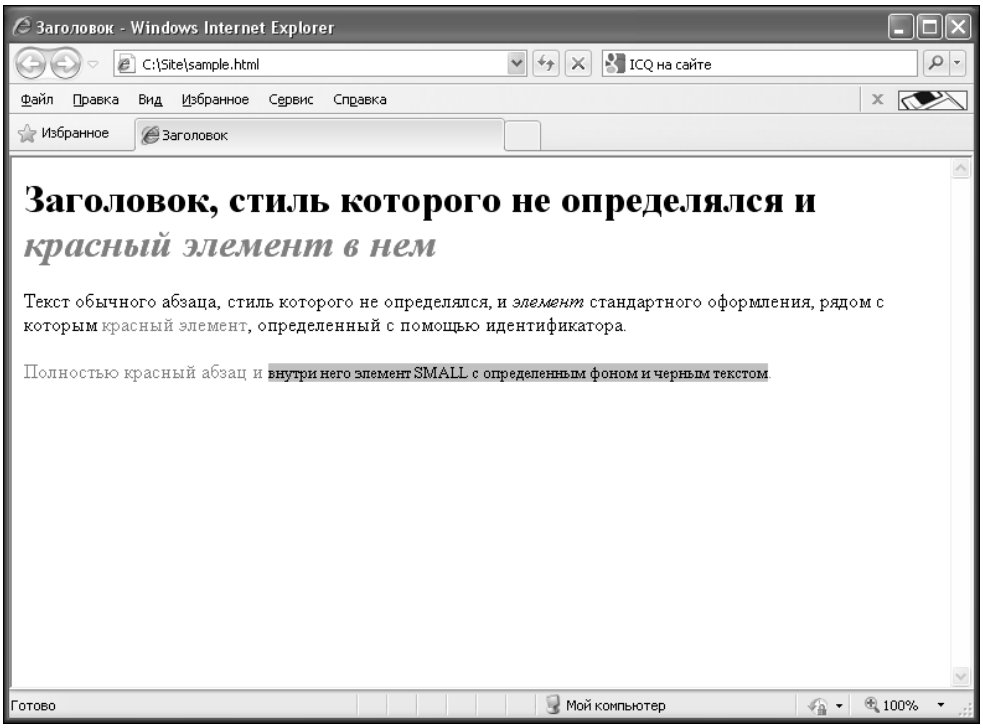


Рис. 2.9 .*Применение в документе разных стилевых селекторов*

В таблице приведены возможные варианты условных комментариев.

Переменная	Пример	Описание
`IE`	`[if IE]`	Условие выполняется во всех версиях браузера Internet Explorer
`value`	`[if IE 7]`	Условие выполняется только в Internet Explorer 7 (после IE стоит пробел!)
`!`	`[if !IE]`	Условие НЕ выполняется в Internet Explorer
`lt`	`[if lt IE 5.5]`	`less-than` – условие выполняется в браузерах IE версии ниже 5.5
`lte`	`[if lte IE 6]`	`less-than or equal` – условие выполняется в браузерах IE версии 6 и ниже
`gt`	`[if gt IE 5]`	`greater-than` – условие выполняется в браузерах IE версии выше 5
`gte`	`[if gte IE 7]`	`greater-than or equal` – условие выполняется в браузерах IE версии 7 и выше
`&`	`[if (gt IE 5)&(lt IE 7)]`	Оператор И (AND) – условие выполняется в браузерах IE версии выше 5, но ниже 7
`|`	`[if (IE 6)|(IE 7)]`	Операторы ИЛИ (OR) – условие выполняется в браузерах IE версии 6 или 7
`( )`	`[if !(IE 7)]`	Скобки позволяют выделить подвыражения в сложном выражении. Условие не выполняется в браузере IE версии 7

Рассмотрим несколько примеров. Вот так можно указать обозревателю IE шестой версии использовать файл стилей `ie6fix.css`.

```
<!--[if IE 6]>
  <link rel="stylesheet" type="text/css" href="ie6fix.css">
<![endif]-->
```

А в этом примере файл `ienewfix.css` будет загружен в браузерах IE версии выше пятой.

```
<!--[if gt IE 5]>
<link rel="stylesheet" type="text/css" href="ienewfix.css">
<![endif]-->
```

Верстка на основе дескрипторов DIV

Современный подход к созданию сайтов предполагает активное использование стилей для управления видом элементов веб-страниц и их верстки. Так называемая *блочная* верстка или верстка *с помощью слоев* (дескрипторы `<DIV>` или `<SPAN>`), в народе — *дивовая* (или дивная) верстка, приобретает все больше поклонников, а это в свою очередь предполагает знание свойств CSS и их правильное применение на сайте.

Дивовая верстка считается прогрессивным методом создания сайтов, причем выполненные таким способом сайты более "дружелюбно" воспринимаются поисковыми системами, так как имеют минимальный набор дескрипторов. Но и здесь присутствуют свои подводные камни: в разных браузерах, а то и в разных версиях одних и тех же браузеров сайт на дивах будет отображаться по-разному. Просто не все свойства CSS поддерживаются некоторыми браузерами. Такие нестыковки способны подкинуть огромную ложку дегтя в блочную верстку. Но мы все же рискнем.

Сайты на дивах — это HTML-шаблон и CSS-файл (или набор файлов стилей) для управления каждым элементом сайта: будь то меню или какой-либо другой блок. Попробуем создать страничку в DIV-верстке, похожую на ту, что мы делали, применяя таблицы. Для этого надо создать два файла. Первый, `index.html`, будет содержать следующий код.

```
<!--Пример HTML-документа.-->
<html>
<head>
<title>Заголовок</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<!--Тело HTML-документа.-->
<body>

<!--Рисуем таблицы на дивах.-->
<div id="container">
  <div id="header">Шапка страницы</div>
     <div id="wrapper">
      <div id="content">Контент</div>
     </div>
  <div id="navigation">Левый блок</div>
  <div id="extra">Правый блок</div>
```

```
  <div id="footer">Низ страницы</div>
</div>

</body>
</html>
```

Второй файл назовем `style.css` (название может быть любым, но чаще используется именно это, так как оно говорит само за себя) и поместим в него следующее содержимое.

```
/* Файл стилей style.css */
html, body {margin: 0; padding: 0; border: 0; text-align: left; height: 100%;}
#header {height: 70px; background: #ff0000; margin: 0; padding: 0;}
#container {margin: 0 auto; padding: 0; height: auto !important; height: 100%;
min-height: 100%; border: 1px solid blue;}
#wrapper {float: left; width: 100%;}
#content {margin: 0 25%; border: 1px solid red; height: 329px;}
#navigation {float: left; width: 25%; margin-left: -100%; border: 1px solid
red; height: 329px;}
#extra {float: left; width: 25%; margin-left: -25%; border: 1px solid red;
height: 329px;}
#footer {clear: left; width: 100%; height: 30px; background: #ffff00;
margin-bottom: 0px;}
```

За что отвечают те или иные директивы стилевого файла, хорошо объясняет рис. 2.10.

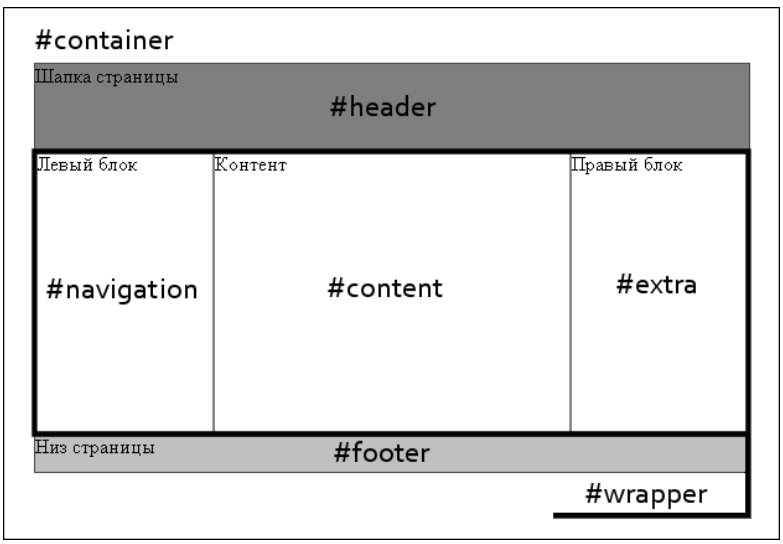


Рис. 2.10. *Пример DIV-верстки*

Данный пример демонстрирует правильно растягивающуюся по горизонтали таблицу, по вертикали она имеет фиксированную высоту в 329 пикселей. Но это не будет мешать таблице растягиваться по высоте по мере заполнения дизайна контентом.

Набор инструкций для CSS-файла далеко не ограничивается данным примером. Технология CSS — это мощный инструмент, обладающий всем необходимым для качественного и профессионального оформления страниц.

Псевдофреймы в CSS

Основной причиной использования фреймов является то, что они позволяют всегда иметь соответствующую информацию на экране, — например, навигационное меню, которое может быть на виду даже в то время, когда пользователь прокручивает страницу. В технологии CSS этого еще можно добиться простым способом, а вот для реализации более сложных фреймовых функций, таких как загрузка документа в одну часть экрана без перезагрузки другой — без сценариев уже не обойтись. Здесь мы рассмотрим пример реализации средствами CSS первой функции фреймов.

Допустим, мы хотим создать страницу из двух фреймов — фиксированного правого и обычного левого. По традиции создаем два файла, первый из которых называем `index.html`.

```
<!--Пример HTML-документа.-->
<html>
<head>
<title>Заголовок</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<!--Тело HTML-документа.-->
<body>

<!--Рисуем фиксированное окно.-->
<div id="framecontent">
  <div class="innertube">Неподвижное окно</div>
</div>
<!--Рисуем обычное окно.-->
<div id="maincontent">
  <div class="innertube">Контент</div>
</div>

</body>
</html>
```

Второй файл назовем `style.css`.

```
/* Файл стилей style.css */
body {margin: 0; padding: 0; border: 0; overflow: hidden; height: 100%;
max-height: 100%;}
#framecontent {position: absolute; top: 0; bottom: 0; left: 0; width: 200px;
height: 100%; overflow: hidden; background: navy; color: white;}
#maincontent {position: fixed; top: 0; left: 200px; right: 0; bottom: 0;
overflow: auto; background: #fff;}
.innertube {margin: 15px;}
/* Исправление отображения в IE6 */
*html body {padding: 0 0 0 200px;}
*html #maincontent {height: 100%; width: 100%;}
```

При прокрутке такой страницы у пользователя всегда будет на виду левая часть “фрейма” (рис. 2.11), а точнее псевдофрейма (ложного фрейма), ведь данный пример не реализует настоящую фреймовую структуру.

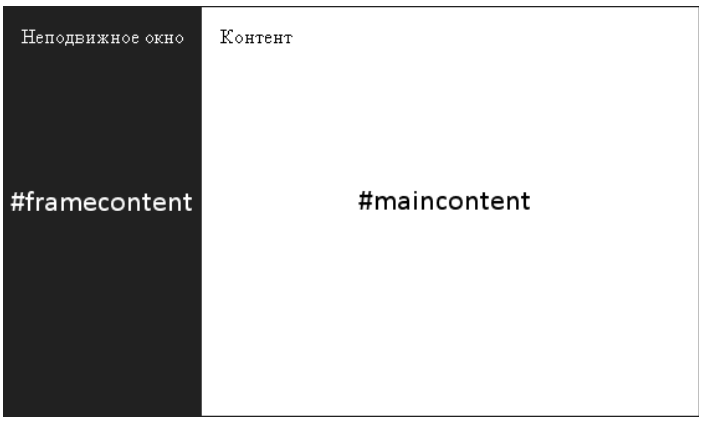


Рис. 2.10. *Пример создания пседофреймов в CSS*

Как вы уже поняли, в данном примере левое окно не имеет прокрутки. Поэтому текст, который вы в нем будете размещать, не должен превышать размеры окна, иначе пользователь не сможет прочитать его до конца.