Михаил Дубаков
www.bhv.ru www.bhv.kiev.ua

ВЕБ-МАСТЕРИНГ

Средствами CSS

- *Основы технологии CSS*
- *Отделение контента от дизайна*
- *Легкая верстка слоями*
- *Советы профессионала*

Санкт-Петербург, 2002

Содержание

Посвящается HTML-кодерам, чья жизнь все еще омрачена браузером Netscape Navigator 4

Введение

Вы держите в руках эту книгу и раздумываете, нужна она вам или нет? Давайте решим этот достаточно непростой вопрос вместе. Если вы обычный бухгалтер и ничего кроме своей бухгалтерии знать не хотите, то положите книгу обратно. Если вы инженер-строитель или просто строитель, и ничего кроме стройки знать не желаете, то остановите свой выбор на чем-нибудь другом. Если вы повар, плотник, художник или студент технического вуза и вам не нужно знать ничего, кроме кастрюль, топоров, красок или лекций, тоже можете книгу отложить. Если вы кто угодно по профессии, но по той или иной причине интересуетесь интернет-технологиями и непременно хотите сделать свой первый (n-й) сайт, то без этой книги вам будет труднее, чем с ней.

Если вы веб-дизайнер, то книга значительно расширит ваш кругозор и, вероятнее всего, поможет делать более грамотный дизайн с точки зрения HTML-верстальщика. Ну а если вы *HTML-кодер,* то эта книга способна перевернуть ваш взгляд на верстку в целом и на отдельные ее компоненты в частности.

Чего здесь нет

Почему-то плохих книг встречается гораздо больше, чем хороших. Этот факт не вызывает сомнений, но, к сожалению, многие совершают ошибку по той простой причине, что отличить хорошую книгу от плохой нелегко, особенно *непрофессионалам,* которых большинство.

Согласитесь, если вы уже знаете в некоторой мере C++, то вы без особого труда сможете выбрать лучшую книгу по C++ из имеющихся в наличии. Но если вы с C++ совершенно не знакомы и хотите купить книгу, чтобы начать изучать этот язык, сразу возникают проблемы. Представьте себе, что вы подходите к прилавку, не имея ничего, кроме горячего желания изучить C++. Скорее всего, вам не пригодится вон тот толстый фолиант на 1000 страниц, поскольку мелкие подробности на начальном этапе не нужны, да и черт его знает, захотите ли вы изучать этот язык программирования после знакомства с ним. Тоненький самоучитель на 200 страниц, лежащий в дальнем левом углу, тоже не особо привлекает, поскольку в такой объем всунуть основы с понятным и достаточно полным описанием практически невозможно. Вы начинаете искать взглядом книги средних размеров, страниц на 400—500.

Вот у этой обложка красивая, можно взять полистать. Вы открываете книгу с красивой обложкой и чрезвычайно интригующим названием «C++ для начинающих». Что дальше? По каким критериям вы будете оценивать книгу? Ну, введение, кажется, нормально написано, аннотация тоже ничего, иллюстрации имеются, значит, вроде наглядно будет все объяснено. Содержание вам ни о чем не говорит, фрагменты программного кода тоже. А внутри текст как текст, сухой и конкретный. В итоге, *вы покупаете книгу практическивслепую.*

Что же делать? Ответ прост: *советуйтесь.* Если вы хотите начать изучение C++, то спросите у профессионального программиста, с какой книги лучше начать, если вы хотите изучить HTML, то найдите HTML-кодера и вырвите у него страшную тайну названия лучшей книги по HTML, если вы хотите знать CSS, то никого не ищите и не спрашивайте, просто прочтите ту книгу, которую вы держите сейчас в руках.

Это все относится к тем, кто ничего не знает о каскадных таблицах стилей. Если вы уже знакомы с этой технологией, то сделать правильный выбор вам проще. Чем же все-таки отличается плохая книга от хорошей? Можно выделить несколько критериев.

- В плохой книге код листинга программ повторяется огромными объемами. Например, есть у нас документ, код которого занимает 50 строчек. Автор книги внес изменение в двух строчках, и вместо того, чтобы показать только эти строчки (возможно, вместе с самыми ближайшими для более точной ориентировки читателя), на следующей странице снова приводятся все 50 строк документа с выделенными полужирным шрифтом изменениями. Таким образом, из четырехсот страниц сто пятьдесят оказываются просто ненужными. *В той книге, что вы держите в руках, этого нет.*
- В плохой книге большинство примеров не представляют совершенно никакого интереса с точки зрения практического использования. Скажем, вы никогда не будете использовать C++ для вывода словосочетания "Hello World!", вы никогда не будете с помощью JavaScript делать кнопку, нажатие на которую изменяет титульную строку окна браузера, вы никогда не будете использовать CSS только с помощью атрибута STYLE внутри HTML-тегов. В этой книге все примеры имеют смысл и проверены на практике, так что *здесь нет совершенно бесполезных примеров.*
- В плохой книге гораздо больше листингов и иллюстраций, чем обычного текста. Это потому, что в плохой книге пояснения часто просто отвратительно короткие и совершенно ничего не объясняющие. Пролистайте эту книгу. В ней гораздо больше простого текста, *в ней нет неимоверного количества бесполезных иллюстраций и огромных кусков кода с объяснением в четырестрочки.*

Это объективные критерии, которые можно очень быстро оценить, если вы уже знакомы с предметом изложения книги. Если вы хотя бы немного знаете каскадные таблицы стилей, вы сможете выбирать осознанно. Листайте, не стесняйтесь! Все по честному!

Что здесь есть

Вопрос сложный, потому что хорошая книга всегда содержит больше того, что перечислено в оглавлении. Плохой учебник по C++ просто содержит описание синтаксиса языка, раскрывает некоторые приемы и расшифровывает листинги кода. Хороший учебник по C++ помогает осознать идеологию языка, его сильные и слабые стороны, его практическое значение в широком смысле, его историю и философию, если хотите. Если вы прочитаете плохой учебник, для вас C++ будет всего лишь языком программирования. Если вы прочитаете хороший учебник, для вас C++ будет языком разработчика. Вы будете думать на нем при написании программ, а не переводить на него алгоритмы. Плохой учебник по грамматике английского языка научит вас грамотно переводить фразы и предложения с русского на английский, хороший учебник поможет сразу строить фразы на английском. Эта тонкая разница и отличает хорошую книгу от плохой.

Итак, здесь есть:

3 Основы **HTML** и достаточно *подробный разбор табличной верстки,* которая была и пока остается главной для большинства HTML-верстальщиков. Это необходимо для четкого осознания *всех недостатков табличной верстки* и языка **HTML** при визуальном отображении контента.

3 История возникновения и внедрения технологии CSS. Это поможет вам тоньше чувствовать необходимость CSS и осознавать будущее.

3 Общее и достаточно беглое описание каскадных таблиц стилей для подготовки почвы.

• Подробный разбор отдельных областей CSS, включая правила построения таблиц стилей, выбор единиц измерения, грамотную работу со шрифтом и текстом. Это надо для того, чтобы помочь вам приобрести *мастерство.*

О Рассмотрение очень многих *проблем реализации поддержки CSS в различных браузерах.* Без этого совершенно невозможно грамотно использовать таблицы стилей на практике.

3 Очень подробное рассмотрение блоковой модели, модели визуального форматирования и всего того, что поможет вам переосмыслить верстку и *сделать правильный выбор в пользу таблиц стилей.* Без этого невозможен прогресс.

- Небольшой экскурс в будущее, чтобы вы смогли почувствовать тенденции современных технологий в области верстки.
- Совершенно реальный и достаточно сложный *пример CSS-верстки непростого макета* с адаптацией под браузеры Internet Explorer 5+, Netscape *6.x,* Mozilla 0.9.x, Opera 5+. Это облегчит для вас переход от табличной верстки к верстке CSS-блоками, потому что один практический пример вдохновляет сильнее, чем толстая книжка теоретических изысканий.

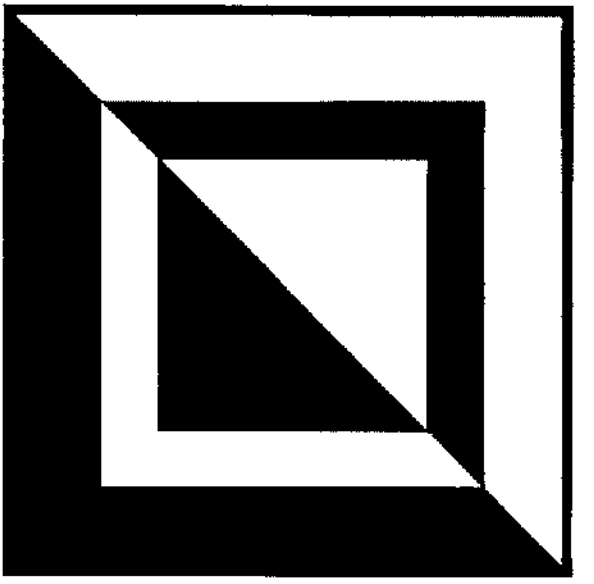
Прочитав эту книгу, вы не только будете знать синтаксис и свойства CSS. Вы научитесь мыслить связкой HTML + CSS, глядя на макет. Вы поймете внутреннюю логику таблиц стилей и совершенно осознанно сделаете выбор, в каком объеме и в каких случаях CSS нужно использовать. В общем, эта книга научит вас не *переводить язык HTML на язык CSS,* а *думать одновременно на двух языках: HTML и CSS.*

Благодарности

Спасибо моей жене Юле, которая терпеливо переносила многочасовое существование своего любимого мужа вохче компьютера в ущерб личной жизни. Спасибо ей же за моральную и физическую поддержку в трудные минуты отсутствия вдохновения. Спасибо моему сыну Максиму, который родился как раз вовремя.

Спасибо замечательному человеку Диме Болашеву, арт-директору студии Дизайн Артель, за систематическое повышение зарплаты и предоставление рабочих мощностей для создания сего труда. Спасибо не менее замечательному человеку Диме Драчкову, управляющему проектами студии Дизайн Артель, за умеренную нагрузку в рабочее время, вследствие чего у меня еще были силы для работы вечером.

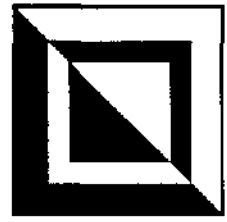
Заранее благодарен читателям этой книги, без которых она вообще не имела бы смысла.



ЧАСТЬ I

ВВЕДЕНИЕ В ТЕХНОЛОГИЮ CSS

Глава 1



Язык HTML: взгляд профессионального веб-мастера

Что такое HTML вы наверняка уже слышали. Возможно, вы его знаете очень поверхностно, возможно, очень глубоко. Эта глава предназначена тем, кто точно не помнит, есть ли у тега <TABLE> атрибут COOL3DBORDER, и что такое ключевое слово multiple в элементе <SELECT>. Если вы точно знаете, что у тега <TABLE> нет атрибута COOL3DBORDER, и что ключевое слово multiple обозначает возможность выбора одновременно нескольких пунктов в списке, то можете смело начинать читать книгу с *главы* 2. А с теми, кто остался, пожалуй, начнем.

Язык разметки гипертекста: вступление

Вы запускаете браузер, набираете в адресной строке адрес сайта и получаете желаемую страницу. Вот, вкратце, действия пользователя, который бродит в сети Интернет. Он не задумывается, что лежит за всем этим. Ему это не нужно. Но это нужно нам, профессиональным веб-разработчикам, чтобы максимально удовлетворить пользовательские запросы.

Мы не будем углубляться в историю и стряхивать пыль с архивных документов 90-х годов XX столетия, но базовые понятия рассмотрим. Итак, что же лежит в основе всемирной сети Интернет? Базовыми являются три технологии.

- URI (Universal Resource Identifier) — универсальный идентификатор ресурса. Именно это пользователи набирают в строке браузера.
- HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста. На основе этого протокола строятся взаимодействия клиент-сервер.
- HTML (HyperText Markup Language) — язык разметки гипертекста. Все веб-страницы в конечном итоге преобразуются в HTML-страницы.

Рассмотрим чуть подробнее каждую из этих технологий.

Для того чтобы найти в огромной сети какой-нибудь определенный документ, у него должно быть уникатьное, отличное от всех остальных документов имя. Понятие "имя" не совсем корректно, потому что мы привыкли применять его к файлам, здесь же ситуация несколько иная. В Сети сушест-

вует множество серверов, на которых хранятся миллиарды файлов. Естественно, что имена этих файлов могут совпадать. Например, на одном сервере может находиться сотня файлов index.html. По этой причине для того, чтобы их различать, существует универсальный идентификатор ресурса **(URI).** Вот так выглядит типичный URI:

```
http://www.artel.by/index.phtml
```

В нем дословно говорится, что по протоколу HTTP с сервера **www.artel.by** надо получить файл index.phtml, который находится в корневом каталоге этого сервера. Понятно, что в корневом каталоге может находиться только один файл с именем index.phiml. Зато в других — сколько угодно. Вот, например, URI внутренней папки сервера:

```
http://www.artel.by/folder/index.phtml
```

Протокол передачи гипертекста используется в Сети с 1990 года. Последний стандарт HTTP/1.1. Этот протокол работает по принципу запрос/ответ. Вначале браузер (или любой другой клиент) отправляет на сервер запрос, который включает в себя метод запроса, **URI,** версию протокола, MIME-подобное сообщение, содержащее управляющую информацию запроса, информацию о клиенте и тело сообщения (если оно есть). Сервер этот запрос обрабатывает и возвращает клиенту ответ, который содержит версию протокола, и код статуса (успех или ошибка), MIME-подобное сообщение с информацией о сервере, метаинформацию о содержании ответа, и само тело ответа (например, HTML-страничку).

Допустим, пользователь набр&т в адресной строке браузера LJRI, браузер отправил запрос, сервер его обработал и вернул браузеру страничку. Причем эта страничка обязательно будет размечена с помощью HTML И любой браузер поймет, что это именно HTML-страничка. Собственно, для этих целей и был разработан язык HTML. Он универсальный и платформонезависимый, *у.* е. правильно отображается на компьютерах с различными операционными системами. В настоящее время именно HTML является единственным языком в глобальной сети Интернет (возможно, в ближайшее время стандартом станет XML, однако это случится не раньше, чем через год-два). Язык HTML — это упрощенная версия стандартного общего языка разметки SGML (Standart Generalised Markup Language). Язык SGML достаточно сложный и запутанный, его спецификация занимает 600 страниц, и он предназначен для создания других языков разметки.

Сейчас достаточно сложно себе представить ситуацию 80-х годов, когда все начиналось. Сложно представить мир без Интернета, без тысяч серверов и миллионов веб-сайтов. Тем интереснее будет проследить эволюцию языка HTML.

Начало было положено Тимом Бернерс-Ли, выпускником Оксфордского университета. В 1989 году он выдвинул предложение *Системы Гипертекстовых До-*

кументов, которая должна была использоваться внутри CERN. В 1990 году он назвал эту систему World Wide Web (на русский язык это можно перевести как Всемирная Паутина). Одной из составляющих системы был язык разметки гипертекста. Его основы были заложены в 1990 году, когда Бернерс-Ли разрабатывал первый веб-браузер. Наконец, в 1993 году появилась первая версия языка — HTML 1.0. Но он не был стандартом. И только в 1995 году, когда закончилась разработка языка HTML 2.0, он стал таковым. К тому времени новую версию языка HTML полностью поддерживало большинство браузеров.

Что же было в этой версии? Во-первых, полностью утвердилась структура документа (она осталась неизменной до настоящего времени). Во-вторых, достаточно широко были представлены элементы разметки текста, такие как <P>, <H1-6>, <UL>, <LI>, <CITE>, <CODS>, <EM>, <B>, <I>,
 И Другие. В-третьих, появился тег <IMG>, ПОЗВОЛИВШИЙ добавлять рисунки. Конечно же, можно было вставлять гиперссылки, для этих целей предназначался элемент <A> С атрибутом HREF. БЫЛИ И формы, реализованные элементами <FORM>, <INPUT>, <SELECT>, <TEXTAREA>. Как МЫ ВИДИМ, ОЧеНЬ МНОГОв ИЗ сегодняшнего стандарта языка HTML 4.0 уже было реализовано в 1995 году в спецификации HTML 2.0. Вне всяких сомнений, именно этот язык стал базисом, который уже потом усложнялся и дополнялся.

В 1994 году было решено разбить язык HTML на уровни. Это было сделано для удобства, чтобы при реализации очередного уровня сохранялась обратная совместимость, иными словами, имея принципиальные отличия, новый уровень непременно включал в себя все предыдущие.

G *Уровень 0*— обязательный для поддержки всеми браузерами. Включает в себя заголовки, якоря, списки.

• *Уровень 1* — добавляются рисунки и элементы выделения текста (<EM>, <в>, <i> и др.).

3 Уровень 2 — добавляются формы.

3 *Уровень 3* — добавляются таблицы.

Очевидно, что язык HTML 2.0 является языком второго уровня. С его помощью нельзя верстать страницы со сложной структурой, потому что это невозможно сделать без таблиц.

Дальнейшим развитием языка стала версия HTML 3.0. В ней были реализозаны таблицы, обтекание текста вокруг фигур и многие другие идеи. Несмотря на обратную совместимость, отличия между версиями 2.0 и 3.0 были :фомными, по этой причине браузеры очень медленно и достаточно вяло включали поддержку новых тегов и новых возможностей Версия 3.0 так и не стала стандартом. Зато им стала версия HTML 3.2. Она создавалась с учетом мнений производителей браузеров (Microsoft и Netscape), что естественным образом привело к положительным результатам. Примечателен тот

факт, что еще до официального утверждения данного стандарта (это произошло в 1997 году), уже в 1996 году практически все браузеры полностью поддерживали его. В эту версию включили многие теги, которые до сих пор были специфичны для тех или иных браузеров. Фактически данный стандарт объединил и легализовал эти теги. Из нововведений надо отметить таблицы, возможность манипуляций со шрифтом (тег <FONT>), возможность использования на страницах Java-апплетов (тег <APPLET>).

Во многом благодаря стандарту HTML 3.2 веб-дизайн испытал небывалый взлет. Появилась возможность проектировать и отображать на экране сложные композиции, ничем не уступающие журналам, газетам и другим печатным изданиям (это стало возможным благодаря табличной верстке). Но из-за медленных каналов связи приходилось значительно ограничивать применение графики, а многие мониторы отображали не более 256 цветов, так что и богатство красок оставляло желать лучшего. Обобщая, можно сказать, что *инструментарий уже позволял делать красивые и сложные сайты, а техническая база — нет.*

Несмотря на то, что язык HTML 3.2 включал многие расширения, внедренные разработчиками браузеров, он все еще оставался достаточно ограниченным, и новый стандарт не заставил себя долго ждать. Уже в том же 1997 году появилась спецификация HTML 4.0. В нее включили фреймы (<FRAME>, <FRAMESET>, <IFRAME>, <NOFRAMES>), унифицировали Процедуру ВСТаВКИ рЭЗличных объектов в документ, будь то Flash, Java или Sound, добавив тег <OBJECT>, реализовали поддержку каскадных таблиц стилей (CSS), пожалуй, самую полезную новую технологию для верстки (тег <STYLE>, атрибуты class и id к элементам). Кроме того, были значительно усовершенствованы формы и таблицы, а некоторые теги помечены как *неж&ттельные для использования* (<APPLET>, <BASEFONT>, <CENTER>, <DIR>, <FONT>, <ISINDEX>, <MENU>, <S>, <STRIKE>, <U>).

Что нового дал язык HTML 4.0? Самое главное и принципиальное отличие — это возможность разделения визуального представления документов и их содержимого документов. Все визуальное представление должно было описываться средствами технологии CSS, а сама структура содержимого — средствами языка разметки HTML. Именно по этой причине в спецификации 4.0 был объявлен нежелательным для использования тег <FONT>, Т. К. ОН отвечал именно за визуальное представление, а не за структуру документа. Все, что можно было реализовать тегом <FONT>, реализовывалось средствами CSS. Единственное, что сдерживало такой глобальный переход — слабая поддержка стандартов CSS ведущими браузерами. Они поддерживали стандарты CSS далеко не полностью, к тому же реализация стандартов существенно отличалась у разных производителей (взять, хотя бы браузеры Microsoft Internet Explorer 4.0 и Netscape Navigator 4.0).

Надо отметить, что четвертая версия HTML отличается законченностью и полнотой. Фактически, это предел возможностей данного языка. Последняя версия HTML 4.01 стала стандартом в 1999 году, после чего разработка этого языка прекратилась. Производители браузеров перестали вводить новые теги, так что поддержка четвертой версии стала повсеместной и полной. На этом закончились споры и прения насчет HTML, а камнем преткновения стали каскадные таблицы стилей и будущий язык разметки гипертекста, который должен заменить HTML. На данный момент этим языком считается XHTML (extensible HyperText Markup Language), первая версия которого была рекомендована консорциумом W3C в январе 2000 года.

Надо сказать, что XHTML всего лишь промежуточный шаг, который должен упростить переход от HTML к XML (по замыслу его создателей). Почему же понадобился новый язык? Лая того чтобы понять это, надо критически взглянуть на HTML и рассмотреть все его недостатки.

- HTML состоит из ограниченного набора элементов и атрибутов, по этой причине он недостаточно гибкий и не совсем подходит для многих задач, которые предвидятся в ближайшем будущем.

3 Набор тегов — это не лучшее средство для разметки документа с точки зрения семантики. Например, если в тексте встречаются названия книг и названия фильмов, а нам надо как-то визуально отобразить их по-разному, то придется пользоваться конструкциями типа

```
<H6 class="film">На3ВаННе фильма</H6>
<H6 class="book">На3ВаННе книги</H6>
```

а затем с помощью CSS описывать эти визуальные представления. Гораздо логичнее с любой точки зрения было бы использовать такую конструкцию:

```
<К1ЬМ>название фильма</Р1Ш>
<BOOK>название книги</BOOK>
```

Причем данная конструкция, на самом деле, не намного проще, но зато логичнее, а в наш перегруженный информацией век это немаловажно, потому что найти какую-либо информацию станет легче.

Вот два глобальных недостатка, которые принципиально нельзя решить на базе языка HTML. Зато их лишен новый язык XML (extensible Markup Language). Но путь до него еще долог по той простой причине, что HTML слишком сильно укоренился в Сети и переход на другой язык будет трудным и постепенным, потому что люди крайне консервативны и не желают учиться новому языку без веских на то оснований. Так что пока не возникет крайняя необходимость, прогресс будет медленным. Для ускорения роцесса и был разработан стандарт XHTML, который должен безболезненно обеспечить плавный переход. Основным отличием от HTML является

изменение синтаксиса, который в XHTML строго соответствует синтаксису языка XML {строго соблюдается правильная вложенность тегов, элементы и атрибуты пишутся в нижнем регистре, значения атрибутов заключаются в кавычки).

О будущем языков разметки мы поговорим несколько позже, а пока вплотную займемся языком HTML, потому что де-факто он все еще остается стандартом во всемирной сети Интернет.

Структура HTML-документа

Как уже было сказано ранее, HTML— это язык разметки гипертекста. То есть он определяет структуру документа. Давайте рассмотрим простейшую HTML-страничку и на ее примере разберем назначение основных структурных элементов языка HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML>
  <HEAD>
    <T1TЬE>Домашняя страница 0ксаны</T1TЬE>
    <SCRIPT LANGUAGE="JavaScript">
      function opn()
      {
        window.open('popup.html','w','toolbar="no",width-2Q0,height=4Q0'j
      }
    </SCRIPT>
    <STYLE type="text/css">
      P (color: #000}
      BODY (background-color: #FFF}
    </STYLE>
  </HEAD>

  <BODY>
  <H1>Домашняя страница 0ксаны</H1>
  <IMG SRC="img/myphotol.jPg" WIDTH-"300" HEIGHT="286" BORDER-"0"
       ALT="MOR фотография" ALIGN="left" HSPACE="15">
  <P>Это моя самая лучшая фотография, которую я люблю больше всего.
     Вам, наверное, тоже понравится.</P>
  <P>Зовут меня Оксана. Мне <EM>18 лет</EM>. Я живу в городе Житомире,
     но скоро перееду в Москву, потому что там мне больше нравится.
```

```
    Вот только ьасобираю побольше денег. У меня есть любимая собака
    <B>Шарик</B>. Скоро я сделаю страничку в Интернете и для нее,
    чтобы вы смогли увидеть ее фотографию.</P>
  <P>А пока все. Это был мой первый опыт в изготовлении странички.</P>
  </BODY>
</HTML>
```

На рис. 1.1 показано, как эта страничка будет выглядеть в браузере.

Рис.1.1. Простейшая HTML-страничка

В глобальном плане HTML-код страницы можно разбить на три части:

- строка, которая содержит информацию о версии языка HTML;
- описательная секция HEAD (В ней содержится служебная информация, которая не отображается в браузере, эта секция находится между тегами <HEAD></HEAD>);
- тело документа, которое, собственно, и содержит всю информацию, которую браузер выведет пользователю. Эта секция заключена в теги <BODY></BODY>.

Давайте детально разберем всю нашу страничку, чтобы составить первое представление о структуре документа и о языке HTML в целом. Первая строка — это объявление версии и типа языка HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
```

По большому счету, до недавнего времени от этой строки ничего не зависело для конечного пользователя. Иными словами, эта строка никак не влияла на отображение HTML-страницы. Однако на данный момент есть три причины для включения версии и типа используемого языка HTML в страницу:

- это требуется стандартом HTML 4.0;
- существуют программы (так называемые HTML-валидаторы), которые позволяют проверять корректность HTML-документа, они используют для анализа именно ту версию языка HTML, которую вы указали;
- с помощью инструкции ODOCTYPE> в браузере Microsoft Explorer 6.0 включается полная поддержка стандарта CSS-1.

Надо сказать, что с выходом шестых версий браузеров возникла проблема. Новые браузеры лучше поддерживают рекомендации консорциума W3C, и появилась возможность писать корректный код, который будет корректно отображаться. Однако старые браузеры такой код могут отображать с ошибками. Для решения этой проблемы использовали DTD (Document Type Declaration), что можно перевести как объявление типа документа. Так как нас интересует только четвертая версия языка HTML, то для нее существуют следующие объявления типа документа.

- strict (строгое определение) включает все элементы и атрибуты, кроме "нежелательных" и не использующихся в документах с фреймами:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">
```

- Transitional (переходное определение) включает все, что включено в строгое определение, но, кроме того, и "нежелательные" элементы и атрибуты:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
        "http://www.w3.org/TR/REC-html40/loose.dtd">
```

3 Frameset (определение для фреймов) включает вес, что включено в переходное определение, но, кроме того, и фреймы:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/REC-html40/frameset.dtd">
```

URI в конце каждого объявления версии и типа языка указывает на файл, который содержит описание данного объявления. Браузер может загрузить этот файл и использовать его для корректного отображения документа.

3 <HTML>

Вторая и третья секции должны быть заключены в теги <HTMLX/HTML>. Открывающий тег находится непосредственно перед тегом <HEAD>, а закрывающий в самом конце страницы. Впрочем, и открывающий, и закрывающий теги не являются обязательными, но все же лучше их ставить.

3 <HEAD>

Этот тег начинает вторую секцию HTML-документа. В ней содержится информация, которая не отображается браузером (список ключевых слов для поисковых систем, описание документа, конструкции языка JavaScript, каскадные таблицы стилей). Тег также не является обязательным, однако лучше его ставить.

3 <TITLE>

Элемент <TITLE> определяет заголовок страницы.

```
<T1TbE>Домашняя страница 0KcaHbi</TITLE>
```

В браузерах заголовок страницы обычно выводится как заголовок окна. Этот элемент обязательный и очень важный, в частности, для поисковых систем. Обычно ключевые слова, заключенные в теги <TITLEX/TITLE>, имеют большой вес, т. е. от них в значительной мере зависит, на каком месте в результатах поиска окажется данная страничка. Так что лучше делать название страницы подробным и максимально информативным. Например, в данном случае лучше было бы такое название для страницы:

```
<T1TЬ£>0ксана Стройнова — профессиональная модель. Домашняя
страница.</TITLE>
```

Далее в коде следует включение скрипта:

```
<SCRIPT  LANGUAGE="JavaScript">
function opn()  {
  window.open(•popup.html','w','toolbar="no",width=200,height=400')
}
</SCRIPT>
```

Элемент <SCRIPT> служит для включения скриптов в HTML-документ (наиболее часто используемым скриптовым языком является JavaScript, кроме него иногда используют VBScript). Скрипт можно включать и в раздел <HEAD>, и в раздел <BODY>. В данном случае используется язык JavaScript, это задано в атрибуте LANGUAGE. Однако рекомендуется тип языка задавать через атрибут TYPE. В нашем случае конструкция выглядела бы так:

```
<SCRIPT TYPE="text/javascript">
```

Дело в том, что идентификаторы языка для атрибута LANGUAGE (типа JavaScript, JavaScript 1.3, VBScript) не являются стандартизированными. Поэтому атрибут LANGUAGE в спецификации языка HTML 4.01 помечен как "нежелательный".

Кстати, в нашем случае описана функция, которая при вызове открывает новое окно шириной в 200 и высотой 400 пикселов без панели управления. В это окно загружается файл popup.html, который должен лежать в том же каталоге на диске, что и страница, с которой вызывается функция.

- Далее в коде идет включение каскадной таблицы стилей:

```
<STYLE type="text/css">
  P {color: #000}
  BODY {background-color: #FFF}
</STYLE>
```

Элемент <STYI,E> используется для включения таблиц стилей в HTML-документ. Вообще способов включения несколько (подробнее обсудим этот вопрос позже). Элемент <STYLE> должен находиться в секции <HEAD>. Атрибут TYPE определяет язык таблиц стилей. Самым распространенным является CSS, но существуют и другие, которые, впрочем, практически не используются.

В данном случае используется именно CSS. В стилях задано, что все элементы <?> будут отображаться в браузере черным цветом, а фон страницы будет белым.

O </HEAD>

Этим тегом заканчивается второй блок.

П <BODY>

Этот тег обозначает начало тела документа, т. е. третьего блока. Именно отсюда начинается та часть документа, которая будет выводиться браузером. У элемента <BODY> есть несколько атрибутов, которые определяют цвет ссылок, цвет фона, но все они являются "нежелательными", т. к. все оформление можно реализовать с помощью CSS.

О <H1>

Это заголовок.

```
<H1>Домашняя страница Оксаны</H1>
```

Существует шесть уровней заголовков, которые отличаются цифрами. Самый верхний (самый важный) — <H1>. Он отображается шрифтом самого большого размера из всех заголовков. Самый нижний — <нб>. Он отображается шрифтом самого маленького размера из всех заголовков.

Далее по тексту идет вставка в документ фотографии Оксаны.

```
<IMG SRC="img/myphotol.jpg" WIDTH="300" HEIGHT="286" BORDER="0"
     АЬТ="Моя фотография" ALIGN="left" HSPACE="15">
```

Внедрение изображений в HTML-документ осуществляется с помощью тега <IMG>. Рассмотрим все его атрибуты. Атрибут SRC содержит путь к графическому файлу относительно текущего каталога сайта. В данном случае файл myphotol.jpg находится в каталоге img, тогда как сам документ находится на уровень выше графического файла.

Атрибуты WIDTH и HEIGHT определяют ширину и высоту изображения. Они не являются обязательными и в некоторых случаях их можно опускать для сокращения размера кода. Однако при верстке с помощью таблиц лучше ставить ширину и высоту, потому что в таком случае браузер быстрее сможет отобразить таблицу. Дело в том, что браузер начинает отображать содержимое таблицы только в том случае, когда точно знает размеры *всех* ячеек. Если вы не указали размер изображения, то до тех пор, пока это изображение не загрузится, браузер не сможет узнать его размер, а, следовательно, и размер ячейки, в которую оно вставлено.

Атрибут BORDER определяет ширину рамки вокруг рисунка Он тоже не является обязательным и по умолчанию равен нулю. Однако если картинка представляет собой ссылку

```
<A HREF="index.html"><IMG SRC="img/main.jpg" BORDER="0"
   ALT="На главную"x/A>
```

то этот атрибут надо указать явно, т. к. в противном случае вокруг рисунка появится рамка, которая будет иметь цвет ссылки. Следовательно, для изображения, которое является ссылкой, по умолчанию атрибут BORDER не равен нулю.

Атрибут AIT задает краткое описание изображения. Оно нужно в тех случаях, когда у пользователя в браузере отключено отображение графики. Кроме того, при просмотре страницы с помощью некоторых браузеров оно появляется, когда пользователь наводит курсор мыши на изображение. Этот атрибут с недавних пор стал обязательным. Для осмысленных изображений (фотографий, рисунков, схем) надо составить описание, а

для графических элементов, которые входят в художественное оформление страницы (то есть являются элементами дизайна), надо оставлять его пустым (ALT="")- Во-первых, поисковые машины при индексации страниц учитывают текст внутри атрибута ALT. Во-вторых, посетители сайта, использующие текстовые или голосовые браузеры, смогут понять, что это за картинка.

Атрибут ALIGN для тега <IMG> определяет положение рисунка относительно окружающего текста. В данном случае ALiGN="ieft" говорит о том, что рисунок выровнен по левому краю, а текст обтекает его справа.

Атрибут HSPACE задает горизонтальный отступ. У нас он равен 15 пикселам.

```
<P>Это моя самая лучшая фотография, которую я люблю больше всего. Вам,
наверное, тоже понравится.</P>
```

Элемент <p> соответствует абзацу. В тексте перед ним и после него добавляется символ перевода строки.

- </BODY>

Этот тег закрывает третью секцию. Он необязательный.

П </HTML>

Этим тегом заканчивается HTML-страница. Он тоже не является обязательным.

Как видите, ничего сверхсложного пока не было. Сделать простейшую страничку можно после часа изучения **HTML.** Со структурой мы разобрались, а теперь уделим внимание основным элементам и некоторым особенностям языка **HTML.** Эта глава не заменяет подробный учебник, так что для полного освоения вам, скорее всего, потребуется купить еще одну книгу. Но здесь содержится много полезного, что вы не всегда найдете в распространенных пособиях и документации. Даже если вы уже более-менее знакомы с HTML, не поленитесь прочитать эту главу, все-таки это не просто справочник по тегам, а взгляд на элементы языка профессионального веб-мастера. А начнем мы с текста.

Элементы языка HTML

Текст

Текст ~ основа информации. Грамотная подача текстовых блоков на странице чрезвычайно важна, потому что от этого напрямую зависит легкость ее восприятия. Верстальщик же должен уметь правильно форматировать текст. О правилах форматирования поговорим по ходу дела, а пока подробнее остановимся на средствах форматирования.

В HTML существует определенный набор элементов для выделения текстовых блоков, обозначения цитат и т. д. Вот эти элементы.

П3 EM

Выделение (в большинстве браузеров выводит текст курсивом). Часто вместо данного элемента используют элемент <1>, что логически неверно, хотя визуально дает тот же результат.

• STRONG

Более сильное выделение (в большинстве браузеров выводит текст полужирным). Почти всегда вместо данного элемента используют элемент <в>. Опять же логически это неверно, однако позволяет сэкономить пару десятков байтов.

О CITE

Так надо выделять цитату или ссылку на другие ресурсы (в большинстве браузеров выводит текст курсивом).

О DFN

Так надо выделять определение термина (в большинстве браузеров выводит текст курсивом).

Я CODE

Фрагмент компьютерного кода (в большинстве браузеров выводит текст моноширинным шрифтом, обычно таковым является Courier New).

П SAMP

Вывод примера программ, сценариев и т. д. (в большинстве браузеров выводит текст моноширинным шрифтом).

П VAR

Так надо выделять переменную или аргумент программы (в большинстве браузеров выводит текст курсивом).

О ABBR

Так выделяют сокращения и аббревиатуры (например, WWW, HTTP, URI; в большинстве браузеров никак не выделяется).

Если вам понадобится использовать длинную цитату, то надо воспользоваться элементом <BLOCKQUOTE>. Браузер обычно отображает этот элемент с отступом, поэтому его часто используют именно для отбивки текста. Такая практика порочна, потому что подобное форматирование лучше делать с помощью CSS. Да, отступ с помощью <BLOCKQUOTE> зачастую сделать проще, и со старыми браузерами проблем не возникает, но эта практика противоречит идеологии HTML. Для расстановки кавычек в HTML 4.0 предусмотрен тег <Q>, однако многие браузеры его не понимают, поэтому вместо этого тега надо пользоваться специальными символами. Для русского текста левой кавычке будет соответствовать символ siaquo; правой кавычке — sraquo;.

Ниже дан пример правильно отформатированного текста, а на рис. 1.2 показано, как этот фрагмент кода выглядит в браузере.

```
<H3>Размер шрифта</H3>

<P>Одним браузером Netscape дело не ограничивается. К великому сожалению
<ABBR>WmIE 4/5</ABBR> тоже некорректно поддерживает ключевые слова, но
их корректно поддерживают <EM>Netscape6/Mozilla, Opera, и
MacIE5/WinIE6</EM>. Отличие заключается в том, что для <ABBR>WinIE
4/5</ABBR> начальным значением является <VAR>smalK/VAR>, тогда как по
спецификации <ABBR>CSS-K/ABBR> должно быть <VAR>medium</VAR>. Подробнее
можно посмотреть в <STRONG>&laquo;Пособии по CSSSraquo;</STRONG></P>


<P>Что же нам, бедным, делать? К счастью есть фишка, которая позволяет
обойти этот баг Microsoft. Вот он, работающий пример.</P>


<P><CODE>body, div, p, th, td, li, dd   {<BR>
            <CITE>/* это все для Netscape 4-x */</CITE><BR>
    font-family:  Verdana, Lucida, Arial, Helvetica, sans-serif;<BR>
    font-size:    llpx;<BR>
    }
</CODE></P>
```

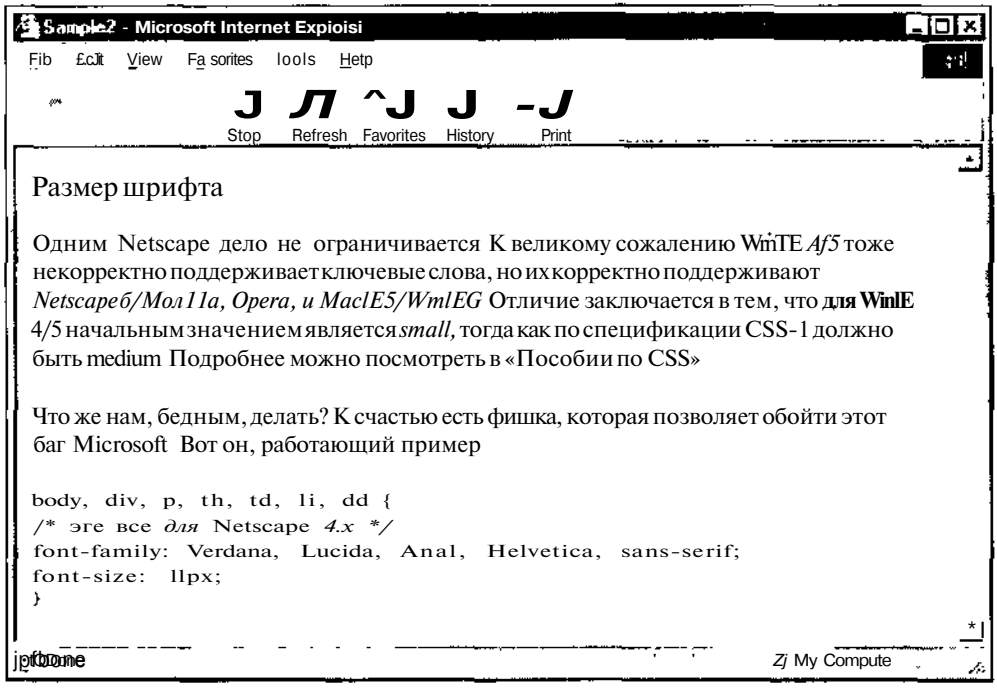


Рис. 1.2. Фрагмент правильно размеченного текста

Что касается форматирования абзацев, то в HTML оно отличается от привычного книжного форматирования. Если в книге первая строка каждого абзаца смешена, то в HTML-документе между абзацами существует промежуток. Вот пример книжного форматирования абзаца:

```
        Блоковые элементы начинаются с новой строки, то есть
они располагаются вертикально друг за другом. Расстояние
между блоковыми элементами определяется полями (margin).
        Строковые элементы располагаются непосредственно друг
за другом, то есть горизонтально. А вот с расстояниями все
гораздо сложнее.
```

Браузер отформатирует абзацы так, как показано на рис. 1.3.

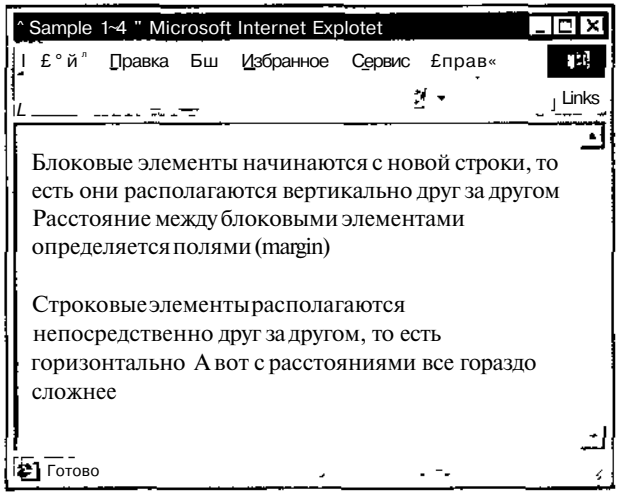


Рис. 1.3. Форматирование абзацев браузером

Кроме того, абзацы можно различным образом выравнивать. Делается это с помощью атрибута ALIGN, который имеет несколько значений:

- left — текст в абзаце выравнивается по левому краю;
- center — текст в абзаце выравнивается по центру;
- right — текст в абзаце выравнивается по правому краю;
- justify — текст в абзаце выравнивается по обоим краям (по ширине).

Данный код будет выравнивать абзац по правому краю, как на рис. 1.4.

```
<P ALIGN=right>CTpoKOBbie элементы располагаютея непосредственно друг за
другом, то есть горизонтально. А вот с расстояниями все гораздо
сложнее.</P>
```

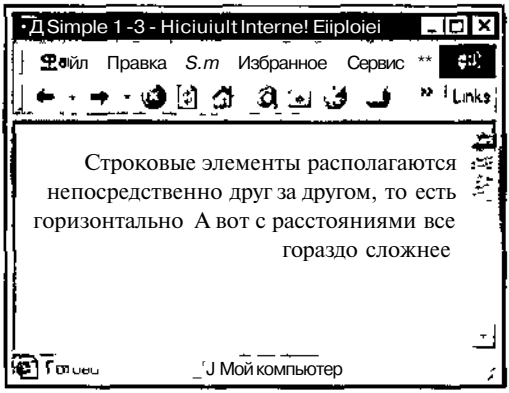


Рис. 1.4. Абзац, выровненный по правому краю

Использование такого способа выравнивания является нежелательным с точки зрения спецификации языка **HTML** 4.O. Выравнивание, как и многие атрибуты, относится к визуальной модели форматирования. Язык **HTML** должен структурировать документ, и не более того, а визуальное представление обеспечивается средствами CSS На самом деле, это идеальная модель, которая чрезвычайно редко воплощается на практике. Существует много преград, но основная — необходимость подстраиваться под старые браузеры типа Netscape Navigator 4.x, которые очень плохо поддерживают CSS. Так что довольно часто приходится смешивать содержание HTML-документа и его визуальное представление. Впрочем, скоро пятое поколение браузеров окончательно сменит четвертое, и тогда задача разделения структуры документа и визуального представления станет решаемой.

Есть еще несколько элементов для работы с текстом, однако основные мы с вами разобрали, а остальные вы сможете освоить самостоятельно. В дополнение отмечу, что для принудительного перевода строки существует тег
.

Ссылки

Гиперссылки есть основа сети Интернет, потому что именно они обеспечивают перемещение от сайта к сайту, в котором и заключается основное отличие среды Интернет от других сред (прессы, телевидения, радио). Давайте подробнее остановимся на том, какие бывают ссылки и как они задаются с помощью HTML.

Допустим, мы взяли с сайта **www.lib.ru** информацию, скажем, рассказ, переверстали его, разместили на своем собственном сайте и хотим сделать ссылку на источник данной информации, т. е. на сайт **www.lib.ru.** Тогда нам понадобится после основного текста вставить такую строку:

```
<P>Рассказ взят с сайта <A HREF="http://www.lib.ru">www. lib. ru</AX/P>
```

В браузере эта строка будет выглядеть так, как показано на рис. 1.5.

Рис. 1.5. Пример ссылки на сайт

Щелкнув мышкой по данной ссылке, посетитель попадет на главную страницу сайта **www.Hb.ru.**

Ясно, что ссылка формируется тегами <AX/A>. Рассмотрим наиболее важные атрибуты элемента <A>.

Я NAME

Обозначает имя якоря.

Определение

Якорь — это метка на странице, куда попадет посетитель, нажав на ссылку.

Звучит достаточно непонятно, так что рассмотрим реальный пример. Допустим, на вашей странице размещена повесть, состоящая из трех глав. Для быстрого перехода по главам надо написать такой код:

```
<A HREF="#chapterl">r\naBa 1</A>
<A HREF="#chapter2">maBa 2</A>
<A HREF="#chapter3">I\naBa 3</A>
<H3><A NAME="chapterr'>I\naBa 1. Название первой главы</A></H3>
... текст первой главы
<H3><A NAME="chapter2">rnaBa 2. Название второй главы</AX/H3>
... текст второй главы
<H3><A NAME="chapter3">r\naBa 3. Название третьей главы</AX/H3>
... текст третьей главы
```

Тогда нажимая на ссылку **Глава 2,** посетитель попадет точно на начало второй главы, потому что именно там расположен якорь с именем chapter2. Обратите внимание, что ссылка на якорь формируется из знака #, вслед за которым следует имя якоря.

Вообще, многие веб-разработчики считают использование якоря дурным тоном, потому что посетитель может и сам прокрутить страницу вниз до нужного ему места. К тому же при нажатии кнопки **Назад** в браузере посетитель попадет не на предыдущую страницу, а на предыдущий якорь, что сбивает с толку. Надо признать, что для небольших документов использование якорей действительно неоправданно, однако для объемных и хорошо структурированных документов якоря необходимы.

- HREF

Данный атрибут служит для формирования ссылки, т. е. он содержит путь к документу, на который необходимо сделать ссылку. Надо сказать, что кроме HTML-страниц можно делать ссылки и на другие объекты, в частности на графические файлы. Например:

```
<A HREF="img/myphoto.jpg">МОН фотография</A>
```

При нажатии на эту ссылку в окно браузера загрузится файл myphoto.jpg.

П TITLE

Этот атрибут устанавливает заголовок ссылки, что-то типа атрибута ALT для элемента **<IMG>**. В атрибуте TITLE должна указываться информация о природе ссылки. Делается это для того, чтобы посетитель ясно себе представлял, куда попадет, нажав на ссылку. Вообще информация подобного рола очень важна, потому что посетители сайта неохотно нажимают на ссылку, если слабо представляют, куда она ведет. Так что делать описания ссылок с помощью атрибута TITLE является хорошей практикой. Вот типичный пример:

```
Посмотреть на наши работы можно <A HREF="http://www.artel.by"
title="официальный сайт компании Дизайн Артель">здесь</A>.
```

- TARGET

Указывает, как будет открываться ресурс по данной ссылке. Можно сделать, чтобы он открывался в новом окне браузера. Для этого надо установить данному атрибуту значение _blank:

```
<A HREF="http://www.w3.org" TARGET="_blank">сайт консорциума W3C</A>
```

Это все атрибуты элемента <A>, так что двинемся дальше.

Все ссылки можно разделить на две группы: *относительные* и *абсолютные*. Пример абсолютной ссылки:

```
<A HREF="http://www.w3.org/index.htmi">W3</A>,
```

т. е. в атрибуте HREF указывается полный URI, так что данная ссылка будет работать на любой странице любого сайта.

Пример относительной ссылки:

```
<A  HREF="index.html">W3</A>
```

т. е. в атрибуте HREF указывается сокращенный URI, который задается относительно той страницы, на которой находится данная ссылка. В приведенном примере страница index.html должна находиться в том же каталоге на том же сервере, что и страница, содержащая эту относительную ссылку на файл index.html.

Таблицы

Полученные знания уже позволяют нам создать простейший HTML-документ, но не более того. Для создания сложных страниц понадобится знание таблиц, потому что в современной верстке все строится именно на их основе. Если вы с таблицами не знакомы вовсе, то советую вам купить хорошую книжку по HTML. Я, конечно, постараюсь кое-что рассказать и объяснить, но на детали не рассчитывайте. Итак, создадим таблицу, состоящую из двух строк и трех столбцов. В языке HTML это делается следующим образом:

```
<TABLE WIDTH="100%" CELLSPACING="5" CELLPADDING="10" BORDER="1">
  <TR>
    <TD>1 ячейка (1 строка)</TD>
    <TD>2 ячейка (1 строка)</TO>
    <TD>3 ячейка (1 строка)</TO>
  </TR>
  <TR>
    <TD>1 ячейка (2 строка)</TD>
    <TD>2 ячейка (2 строка)</TD>
    <TD>3 ячейка (2 строка)</TD>
  </TR>
</TABLE>
```

Как данная таблица выглядит в браузере, ясно из рис. 1.6.

Начало таблицы обозначает тег <TABLE>, а конец — тег </TABLE>. Элемент <TR> обозначает строку таблицы, а элемент <TD> — ячейку. Обратите внимание на вложенность элементов. Таблица может содержать несколько строк (несколько элементов <TR>), а строка может содержать несколько ячеек (несколько элементов <TD>). В нашем примере таблица состоит из двух строк, а каждая строка состоит из трех ячеек.

Рассмотрим атрибуты элемента <TABLE>. В примере указаны лишь основные, но вы редко будете пользоваться другими атрибутами. Итак.

- WIDTH

  Обозначает ширину таблицы. В приведенном примере ширина равна 100%, т. е. таблица займет все доступное окно браузера по ширине. При установке ширины таблицы можно использовать проценты (%) и пикселы (px).

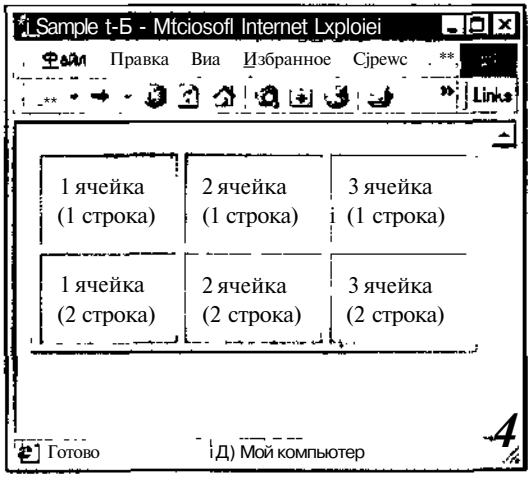


Рис. 1.6. Простейшая таблица

Ширину таблицы задают в пикселах, когда хотят задать ее жестко. В этом случае она не будет растягиваться на все окно браузера и не будет зависеть от разрешения экрана, а всегда будет иметь заданное значение. Например, если написать <TABLE WIDTH="300">, то таблица будет иметь ширину ровно 300 пикселов (если, конечно, содержимое физически сможет уместиться в эту ширину, потому что в противном случае *таблица растянется, чтобы вместить в себя содержимое).*

Ширину таблицы задают в процентах, если нужна гибкость. Тогда таблица будет иметь разную ширину при разных разрешениях, и будет изменяться при изменении размеров окна браузера. Например, если написать <TABLE WIDTH=IOO%>, то при разрешении 800x600 (и окне браузера, развернутом на полный экран) ширина таблицы будет около 800 пикселов (немного меньше, потому что элементы интерфейса браузера отнимут некоторое пространство), а при разрешении 1024x768 ширина таблицы будет около 1024 пикселов.

П BORDER

Обозначает ширину рамки таблицы. Каждая ячейка таблицы имеет рамку, однако нельзя установить рамку для какой-то определенной ячейки, рамки устанавливаются сразу для всех ячеек в теге <TABLE>. ЕСЛИ установить BORDER="0", то рамок не будет вовсе, при любом другом значении рамки будут. На фактическую ширину рамки влияет еще и атрибут CELLSPACING.

О CELLSPACING

Обозначает ширину пространства между ячейками. Для него значения устанавливаются в пикселах. Обратите внимание, что в нашем примере ширина рамки вроде бы должна быть равной одному пикселу, потому что

атрибут BORDER="I". Но фактическая ширина рамки равна 5 пикселам. Так получается из-за того, что значение атрибута CELLSPACING равно 5.

- CELLPADDING

  Обозначает ширину отступов вокруг содержимого каждой ячейки. В нашем примере атрибут CELLPADCING="IC", Т. е. ширина полей равна l0 пикселам. Ятя более глубокого понимания особенностей данных атрибутов давайте рассмотрим еше один пример, но с другими значениями.

```
<TABLE WIDTH="200" CELLSPACING="0" CELLPADDING="O" BORDER="1">
  <TR>
    <TD>1 ячейка (1 строка)</TD>
    <TD>2 ячейка (1 строка)</TD>
    <TD>3 ячейка (1 строка)</TD>
  </TR>
  <TR>
    <TD>1 ячейка (2 строка)</TD>
    <TD>2 ячейка (2 строка)</TD>
    <TD>3 ячейка (2 строка)</TD>
  </TR>
</TABLE>
```

В браузере эта таблица будет выглядеть так, как представлено на рис. 1.7.

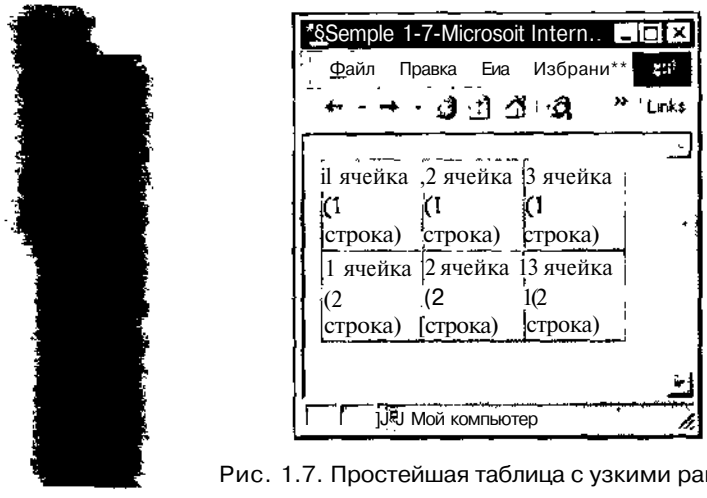


Рис. 1.7. Простейшая таблица с узкими рамками

Обратите внимание на отличия от предыдущего примера. Во-первых, ширина таблицы равна точно 200 пикселов, во-вторых, ширина рамки равна одному пикселу (потому что атрибут BORDER«"1", а атрибут CELLSPACING="O"),

в-третьих, содержимое каждой ячейки вплотную прилегает к рамкам, потому что ширина отступов вокруг содержимого равна нулю (атрибут CELLPADDING="0")-

В HTML на таблицы накладывается одно любопытное ограничение: количество ячеек в каждой строке одной таблицы должно быть одинаковым. Исходя из этого, на первый взгляд кажется, что нельзя сделать таблицу, где в первой строке будет три ячейки, а во второй строке — две ячейки. На самом деле можно. Для этого существует атрибут COLSPAN, который указывает браузеру, сколько ячеек следует объединить в одну. Например:

```
<TABLE WIDTH="200" CELLSPACING="O" CELLPADDING="O" BORDER="1">
  <TR>
    <TD>1 ячейка (1 строка)</TD>
    <TD>2 ячейка (1 строка)</TD>
    <TD>3 ячейка (1 строка)</TD>
  </TR>
  <TR>
    <TD>1 ячейка (2 строка)</TD>
    <TD COLSPAN="2">2 и 3 ячейка (2 строка)</ТБ>
  </TR>
</TABLE>
```

Результат выполнения этого кода продемонстрирован на рис. 1.8.

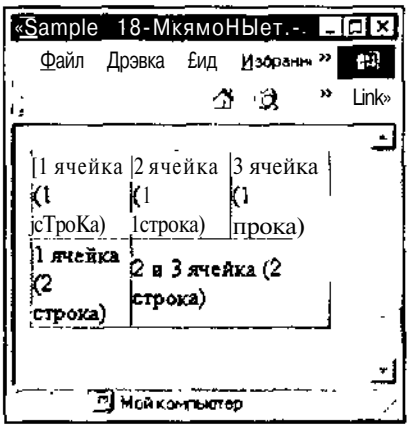


Рис. 1.8. Таблица с объединенными ячейками

Вы видите, что во второй строке вторая и третья ячейки объединились. На самом деле во второй строке по-прежнему три ячейки, но две из них объединены в одну, так что получается одна маленькая первая ячейка и одна

большая *спаренная* вторая. Таким образом можно добиваться любого необходимого количества ячеек в разных строках таблицы.

Для объединения ячеек по вертикали существует атрибут ROWSPAN. ПО механизму действия он аналогичен атрибуту COLSPAN, так что подробно останавливаться на нем не будем.

В HTML выравнивание содержимого ячеек таблицы по горизонтали осуществляется с помощью атрибута ALIGN. ОН может принимать четыре значения:

- left — содержимое выравнивается по левому краю;
- right — содержимое выравнивается по правому краю;
- center — содержимое выравнивается по центру;
- justify — текст выравнивается по ширине путем увеличения расстояния между словами.

Как использовать таблицы по их прямому назначению, надеюсь, понятно. А об использовании таблиц для сложной верстки страниц поговорим в следующей главе.

Фреймы

Фреймы представляют собой достаточно необычный, на первый взгляд, механизм. Если говорить кратко, то с помощью данного механизма можно разбить окно браузера на несколько областей (именно эти области и называются фреймами), а в каждый фрейм загружать свою HTML-страницу.

Что это дает? Первое, что приходит на ум, это разделить окно браузера на две части. В одном фрейме разместить навигацию по сайту, а во второй фрейм загружать информационную страницу. Таким образом уменьшается время загрузки страниц при перемещении по сайту (потому что навигация загружается сразу и не перегружается впоследствии), а также несколько упрощается кодирование сайта, хотя это уже спорный вопрос.

Если обобщить данный пример, то можно вывести правило использования фреймов.

> ***Правило***
>
> В отдельный фрейм надо заключать ту часть страницы сайта, которая не изменяется при переходе на другие страницы.

Самый распространенный случай — это разбиение страницы на три фрейма. В первом фрейме содержится панель навигации. Во втором — "шапка" страницы (обычно там размещается, скажем, логотип и название фирмы, которые остаются неизменными на всех страницах). В третьем — основная информация, которая, собственно, и будет изменяться при перехоле посетителя по страницам сайта. При этом посетитель нажимает на ссылки в первом фрейме, а новая страница загружается в третьем фрейме.

Вот код файла index.html, который создает документ из трех фреймов:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/REC-html40/frameset.dtd">
<HTML>
  <HEAD>
    <T1TЬЕЖоьтания "Страйк Лефт"</TITLE>
  </HEAD>
  <FRAMESET ROWS="2G%, 80%" FRAMEBORDER="0" BORDER="1">
    <FRAME SRO"top.html" SCROLLING="no">
    <FRAMESET COLS="20%, 80%" FRAMEBORDER="0" BORDER="1">
      <FRftME SRC-"menu.html" SCROLLING="no">
      <FRAME SRC="content.html" SCROLLING="no" NAME="content">
    </FRAMESET>
  </FRAMESET>
</HTML>
```

В верхний фрейм загружается документ top.html, в левый — menu.html, а в правый — content.html. Внешний вид такой страницы показан на рис. 1.9.

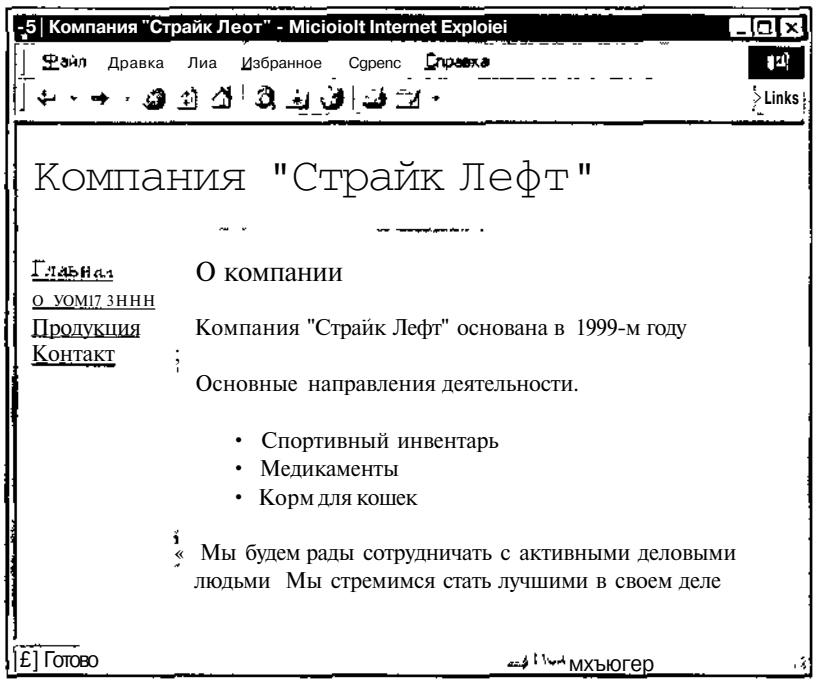


Рис. 1.9. Простейший документ с фреймами

Давайте сначала рассмотрим коды всех трех документов, которые загружаются в разные фреймы.

Код файла top.html:

```
<BODY BGCOLOR="#FFFFFF">
  <H1>Коыпания "Страйк Лефт"</H1>
</BODY>
```

Как видите, здесь опущены необязательные элементы <HTML> И <HEAD>, потому что никакой надобности в таком маленьком документе в них нет. Вообще, в этом документе на белом фоне выводится название компании и ничего более, что мы и наблюдаем в верхнем фрейме,

Код файла menu.html:

```
<BODY BGC0LOR»"#FFFFFF">
  <A HREF-"index.html" TARGET="content">Главная</A><ВЯ>
  <A HREF="about.html" TARGET="content">0 компании</A><ВЮ>
  <A HREF="product.html" TAGRET="content">npoflyKimH</A><BR>
  <A HREF="contact.html" TARGET="content">Контакт</AXBK>
</BODY>
```

Этот документ формирует панель навигации. На белом фоне друг за другом следуют четыре ссылки. После каждой стоит тег
, который обеспечивает перевод строки, вследствие чего ссылки идут столбцом.

Код файла content.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <T1TЬEЖompания "Страйк Лефт"</T1TЬE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
  <H3>0 компании</H3>
  <РЖompания "Страйк Лефт" основана в 1999 году.</P>
  <P>Основные направления деятельности:
  <UL>
    <LI>Спортивный инвентарь</LI>
    <LI>Медикаменты</LI>
    <Ы>Корм для KomeK</LI>
  </UL>
  </P>
  <P>Мы будем рады сотрудничать с активными деловыми людьми:
```

```
    Мы стремимся стать лучшими в своем деле.</P>
</BODY>
</HTML>
```

Это и есть информация главной страницы сайта компании. На белом фоне выводится заголовок раздела. Затем идет краткое описание и перечисление основных направлений в виде списка.

А теперь построчно разберем код файла index.html. Обратите внимание, что в коде отсутствует элемент <BODY>, а на его месте стоит <FRAMESET>.

```
<FRAMESET ROWS="2Q%, 80%" FRAMEBORDER="0" BORDER="1">
```

В этой строке окно браузера разбивается на два горизонтальных фрейма (на тот факт, что фреймы именно горизонтальные, указывает атрибут ROWS). Первый фрейм занимает 20% от высоты всего окна браузера, а второй — 80% от высоты окна браузера. Атрибут FRAMEBORDER="0" указывает на то, что объемной рамки у этого фрейма нет, а атрибут BORDER="1" устанавливает ширину рамки в один пиксел (на самом деле в браузере ширина границы будет больше).

```
<FRAME SRC="top.html" SCROLLING"no">
```

В этой строке мы задаем содержимое верхнего фрейма (у которого высота 20%). Атрибут SRC содержит URI файла, который будет загружаться в этот фрейм (в данном случае это файл top.html), а атрибут scROLLiNG="no" указывает на то, что фрейм не будет иметь полос прокрутки.

Далее по коду должно идти содержимое второго фрейма (у которого высота 80% от высоты всего окна). Вот оно:

```
<FRAMESET COLS-"20%, Э0%" FRAMEBORDER="0" BORDER-"1">
  <FRAME SRC="menu.html" SCROLLING="no">
  <FRAME SRC="content.html" SCROLLING="no" NAME-"content">
</FRAMESET>
```

Таким образом, второй фрейм мы разбиваем еще на два фрейма, на этот раз вертикальных. Первый из них будет занимать 20% от всей ширины окна браузера, а второй — 80%. Далее мы формируем содержимое каждого из этих двух фреймов. В левый фрейм будет загружаться файл menu.html, а в правый — content.html.

Таким образом, у нас получилось три фрейма и три HTML-страницы, которые загружаются в соответствующий фрейм. В левом фрейме находится меню, в правом — основной контент страницы. Обратите внимание, что правый фрейм имеет имя NAME="content" Оно необходимо для организации навигации по сайту. Ссылка в левом фрейме имеет вид:

```
<A HREF="product.htmL" TARGET="content">npo,qyKUMfl</A>
```

Атрибут TARGET указывает, в каком окне откроется файл product.html. В данном случае этот файл откроется во фрейме, который имеет имя rontent, т. е. в правом большом фрейме.

На первый взгляд кажется, что фреймы вещь полезная и нужная, но у них немало недостатков.

Имеется несколько проблем глобального плана.

3 Если вы во фрейме сделаете ссылку на другой сайт, то он откроется в этом же фрейме. Во избежание подобных проблем надо для каждой внешней ссылки прописывать атрибут TARGET="_top", тогда страница откроется в целом окне.

- Зачастую возникает проблема с поисковыми машинами. Для браузеров, которые не поддерживают фреймы (большая редкость в Сети, надо сказать), существует элемент <NOFRAMES>. В нем пишут текст для таких браузеров. Что-то типа "Этот сайт сделан с использованием фреймов, так что обновите ваш браузер". Часто поисковая машина индексирует именно эту фразу, потому что не понимает фреймовой структуры, как и очень древние версии браузеров. Как вариант, можно в элемент <NOFRAMES> вставлять небольшое описание сайта.
- Раньше браузеры плохо распечатывали страницы с фреймами. Можно было распечатать только отдельный фрейм, но не страницу целиком. Последние пятые-шестые версии браузеров с этой задачей справляются без проблем.

О Кроме того, часто можно услышать, что для пользователя фреймы непривычны, однако в настоящее время обращать внимание на этот аргумент не следует, потому что фреймы используются уже давно и привыкнуть к ним вполне успели.

На самом деле без использования фреймов верстать проще и лучше во многих случаях. Окончательный ответ на вопрос об использовании фреймов можно дать, только детально разобравшись со структурой страницы. Я бы советовал их избегать.

Формы

Вся прелесть сети Интернет в том, что она обладает интерактивностью. Это значит, что сайт может реагировать на действия пользователя и осуществлять (с помощью программных элементов) определенные действия. Например, можно послать письмо создателю сайта, поместить свое сообщение на форум, осуществить покупку на сайте электронного магазина.

Все эти действия невозможны без HTML-форм.

Определение

Форма — это элемент интерфейса HTML-страницы, с помощью которого организуется передача данных, введенных пользователем, на сервер для последующей обработки.

Если не углубляться в детали, то процесс взаимодействия посетителя с формой выглядит следующим образом. Пользователь заполняет поля формы, нажимает на кнопку **Submit,** после чего содержимое формы отсылается на сервер. На сервере есть определенная программа, написанная на языке Perl или PHP (конечно, можно писать и на C. и на других языках, но это труднее и встречается реже), которая обрабатывает полученную информацию и далее в зависимости от задачи совершает некоторые действия.

- Отсылает письмо (если это форма обратной связи на сайте).
- П Помещает сообщение в базу данных (если это форум или система комментариев).
- О Оформляет заказ (если это корзина online-магазина).

Результат работы программы может быть показан пользователю (на форуме появилось его сообщение; появилась надпись, что письмо успешно отправлено).

Вот пример типичной формы обратной связи на сайте:

```
<FORM NAME="feedback" ACTION="mail.php" METHOD="post">
  Имя:<BR>
  <INPUT TYPE="text" MAXLENGTH="100" NAME="name"><BR>
  E-mail:<BR>
  <INPUT TYPE="text" MAXLENGTH="100" NAME="email"><BR>
  Сообщение<BR>
  <TEXTAREA COLS="30" ROWS="5" NAME="txt"X/TEXTAREA><BR>
  <INPUT TYPE="Submit" VALUE="Отправить" NAME="ok">
</FORM>
```

В браузере эта форма выглядит так, как показано на рис. 1.10.

Обратите внимание, что код формы начинается и заканчивается тегами <FORM></FORM>. На странице допускается размещение нескольких форм, но они не могут быть вложенными. Этот тег с помощью атрибутов определяет, как будет обрабатываться введенная пользователем информация.

П ACTION

Задает обработчик информации. Это может быть программа на PHP, Perl или другом языке. В данном примере для обработки информации из формы используется программа, которая находится в файле mail.php (причем этот файл должен лежать в том же каталоге, что и документ, который содержит данную форму).

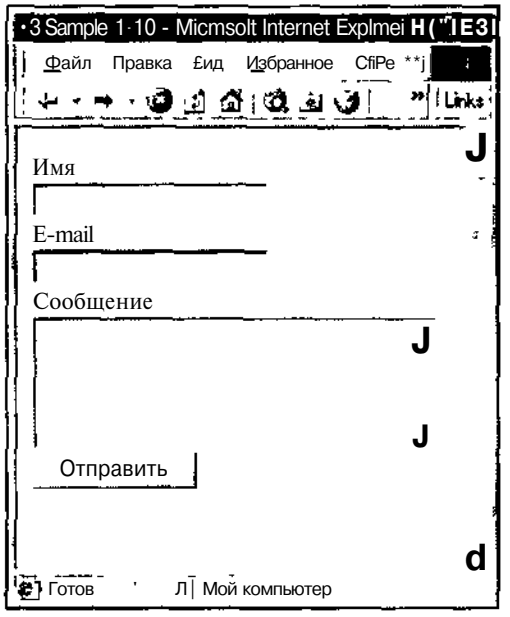


Рис. 1.10. Простейшая форма

D METHOD

Задает метод HTTP, который будет использоваться для передачи данных из формы. Существуют два основных метода: GET И POST. Основное отличие в том, что при методе GET данные из формы добавляются к **URI** в таком виде:

URL?name=valuesnarne=value£name=value,

где name — имя поля (указывается в атрибуте NAME ПОЛЯ формы), а value — значение поля, т. е. введенная пользователем информация. Пример запроса **GET:**

```
http://www.name.com/feedback.php?name=michael5txt=message
```

При методе POST все данные помешаются в переменную среды **HTTP** и пересылаются в теле запроса.

При пересылке большого количества данных необходимо пользоваться методом POST, потому, что длина URI ограничена.

Элемент <FORM> не влияет на отображение содержимого в браузере, он лишь объединяет все компоненты формы. Это нужно для того, чтобы браузер смог определить, какую информациго отсылать на сервер.

Элементы, которые позволяют осуществлять ввод информации, называются управляющими. К ним относятся <INPUT>, <TEXTAREA>, <SELECT> и др. Рассмотрим основные из них.

- INPUT

Является основным управляющим элементом. Имеет важный атрибут TYPE, который определяет назначение элемента <INPUT>. Атрибут TYPE может принимать следующие значения: text, password, checkbox, radio, submit, reset, hidden, image, button. Кратко рассмотрим все значения:

- text — создается стандартное текстовое поле, которое обычно используется для ввода имен, адреса, и другой короткой информации (именно такое поле есть в приведенном выше примере);
- password — создается текстовое поле для ввода пароля (от text отличается тем, что при вводе информации символы заменяются на звездочки (*), так что никто не сможет подсмотреть пароль на мониторе, но клавиатуру старайтесь закрывать или набирать быстро);
- checkbox — создается переключатель, который может иметь два значения (выбран/не выбран). Нужен для организации ответов с вариантами "да/нет";
- radio — создается переключатель как и в случае checkbox, но с помощью radio можно организовать выбор из нескольких вариантов. Для этого надо создать несколько элементов <INPUT> С одинаковым атрибутом NAME, но разным атрибутом VALUE. Например так:

```
<INPUT type="radio" NAME="radname" VALUE="first_variant">
<INPUT type-"radio" NAME-"radname" VALUE="second_variant">
<INPUT type="radio" NAME="radname" VALUE="third_variant">
```

- submit — создается кнопка, нажатие на которую отсылает содержимое формы на сервер. В нашем примере она имеет значение "отправить";
- reset — создается кнопка, нажатие на которую очищает всю ранее введенную в поля формы информацию. Ее использовать большого смысла не имеет, так что не стоит загромождать ею пользовательский интерфейс;
- hidden — создается поле, которое не отображается браузером. Его обычно используют для пересылки на сервер какой-нибудь служебной информации, необходимой для правильной работы программы-обработчика данных из формы;
- image — создается кнопка типа submit, но вместо обычной кнопки будет рисунок, который вы укажете через атрибут SRC;

- button — создается кнопка, действие на которую назначается с помощью скриптового языка (обычно это JavaScript). Это делается для предварительной проверки корректности введенной пользователем информации.

Кроме атрибута TYPE элемент <INPUT> имеет еще несколько атрибутов.

- MAXLENGTH

  Определяет максимальное число символов, которое может ввести пользователь. Имеет смысл ограничивать число символов в полях для имени, фамилии и в прочих коротких полях.

- VALUE

  Начальное значение поля. Обязательный атрибут для типа radio.

- SIZE

  Задает ширину поля на экране. Если атрибут TYPE имеет значения text или password, то эта ширина задается в символах, в любом другом случае — в пикселах.

Есть еще несколько управляющих элементов, которые широко используются при создании форм.

П TEXTAREA

Создается текстовое поле из нескольких строк для ввода больших объемов текста (например, сообщений на форум). Обязателен закрывающий тег. Имеет следующие атрибуты:

- ROWS — определяет число отображаемых в браузере строк (фактически высоту элемента <TEXTAREA>). Является обязательным атрибутом;
- COLS — определяет ширину элемента <TEXTAREA> в символах. Тоже обязательный атрибут.

О SELECT

Создает выпадающее меню из нескольких пунктов. Делается это так:

```
<SELECT size="8" name="menu">
  <OPTION value-"index">rjtaBHaH</OPTION>
  <OPTION value-"product">npOflyKUHH</0PTI0N>
  <OPTION value="contact">KoHTaKT</OPTION>
</SELECT>
```

Этот код создает выпадающее меню из трех пунктов, которое можно использовать для перехода по страницам. Для этого надо написать на скриптовом языке обработчик события на изменение в элементе <SELECT>.

Если понадобится сделать возможным выбор одновременно нескольких пунктов, то в элементе <SELECT> надо прописать ключевое слово multiple:

```
<SELECT multiple size="5" name="selector">
```

Вот и все, что вам совершенно необходимо знать о формах.

На этом краткое введение в HTML заканчивается. Естественно, если вы не были до сих пор знакомы с языком HTML, я в очередной раз настоятельно вам советую приобрести какой-нибудь справочник и разобраться в основах HTML-верстки, потому что следующие главы читать будет сложнее и вы, без всякого сомнения, упустите массу тонкостей, способствующих мастерскому владению CSS.

Глава 2

Как верстают сайты

Для того чтобы прилично сверстать сайт, мало знания языка HTML, надо уметь применять глубокие знания на практике. Любая теория без утилитарного применения остается всего лишь теорией. Возможно занимательной и интересной, но в конечном счете совершенно бесполезной. Нет смысла изучать какой-либо язык программирования и не писать на нем программ. Нет смысла изучать веб-дизайн и не создавать сайты. Точно так же нет смысла изучать HTML и не сверстать ни одной странички. В предыдущей главе вы познакомились с теорией, эта глава целиком посвящена практике.

Визуальное представление информации средствами HTML

Для того чтобы понять, зачем, собственно, разрабатывали и внедряли технологию CSS, надо детально разобраться в языке HTML, знать все его слабые и сильные места, прочувствовать его возможности на реальных примерах. Только тогда необходимость использования CSS станет для вас очевидной. Все познается в сравнении. Поэтому в этой главе я попытаюсь выявить проблемы языка HTML и детально рассказать о верстке только на основе этого языка.

Технология CSS используется для предстаачения информации на странице (мы будем говорить только о визуальном представлении, потому что для русскоязычного Интернета проблемы общедоступности не настолько актуальны, как для англоязычного, т. е. голосовые браузеры несомненная экзотика). Язык HTML также имеет средства для визуального представления информации. Давайте их рассмотрим и хорошенько изучим, чтобы впоследствии было с чем сравнить мощь и элегантность CSS.

Элемент <*FONT*>

Итак, самым ярким и неоднозначным представителем элементов семейства визуализации информации является тег <FONT>. Он позволяет задавать **цвет** шрифта, размер шрифта и гарнитуру. Приведем определение гарнитуры шрифта Николая Дубины.

Определение

Гарнитура— это совокупность шрифтов, объединенных общими стилевыми признаками, отличными от других шрифтов.

За короткое время элемент <FONT> стал чрезвычайно популярным, потому что до него вообще не было возможности менять параметры шрифта, кроме как в настройках браузера. Изначально этот элемент был введен в оборот браузером Netscape уже с самой первой версии и имел только один атрибут SIZE, который, собственно, позволял изменять размер шрифта. Вслед за Netscape поддержку этого элемента реализовали и в браузере Microsoft Internet Explorer 1.0, добавив атрибуты FACE (ПОЗВОЛЯЮЩИЙ изменять гарнитуру) и COLOR (позволяющий изменять цвет шрифта). В Netscape поддержка этих атрибутов появилась позже: COLOR СО второй версии, а FACE — с третьей.

Веб-сайты не замедлили раскраситься в богатую цветовую гамму, а остальным разработчикам браузеров поневоле пришлось вводить поддержку элемента <FONT>, хотя он никогда не был рекомендован консорциумом W3C. Вот типичный пример применения тега <FONT>:

```
<FONT SIZE="5" COLOR="#0000FF" FACE="Verdana">
  <B>Заголовок статьи</B>
</FONT>
```

Этот код выводит текст "Заголовок статьи", который написан полужирным шрифтом **Verdana** пятого размера и синего цвета. Да, гораздо логичнее было бы просто написать в коде:

```
<H2>Заголовок статьи</н2>
```

Но этот заголовок раньше невозможно было сделать синего цвета и изменить шрифт на Verdana. Конечно, первый пример можно несколько улучшить, добавив структурности и логичности. Вот так:

```
<H2XFONT COLOR="#0000FF" FACE="Verdana">3аголоаок статьи</Г0НТХ/Н2>
```

Результат будет тот же. Надо заметить, что указание в атрибуте FACE ТОЛЬКО одного шрифта является плохой практикой, потому что на компьютере пользователя может его не оказаться (например, на платформе Linux нет по умолчанию шрифта Verdana, там его функцию выполняет Tahoma). Для решения этой проблемы надо указывать шрифты через запятую в порядке убывания приоритета, а заканчивать список шрифтов ключевым словом, обозначающим семейство, к которому принадлежат данные шрифты, например так:

```
<H2><FONT COLOR="#0000FF" FACE="Verdana,Tahoma,sans-serif">3аголовок
статьи</Г0ЫТХ/Н2>
```

В данном случае Verdana и Tahoma являются рублеными шрифтами, и такое семейство обозначается ключевым словом *sans-serif*. Вообще имеются следующие основные семейства шрифтов:

О serif — шрифты с засечками, такие как Times New Roman и Garamond;

П sans-serif — рубленые, такие как Verdana, Arial и Tahoma;

• monospace — моноширинные, такие как Courier New;

• cursive — курсивные, такие как Zapf-Chancery;

П fantasy — декоративные.

Как видим, по сравнению с простым описанием заголовка с помощью единственного элемента <H2>, при описании с использованием элемента <FONT> код увеличивается с 24 до 84 байтов.

Можно делать выводы. Использование элемента <FONT>:

Я *значительно увеличивает размер страниц,* а, следовательно, увеличивается время загрузки страницы. Как известно, быстрая загрузка — один из параметров успешного веб-сайта;

Я *явным образом нарушает рекомендации консорциума W3C,* смешивая визуальное и структурное представления информации.

Решить все эти проблемы можно с помощью CSS. Тогда в коде страницы мы напишем заголовок как надо:

```
<H2>Заголовок статьи</H2>
```

А визуальное представление заголовков <H2> зададим в таблице стилей. Хотя бы так:

```
<STYLE type="text/css">
  H2 {
    color: #00F;
    font-family: Verdana, Tahoma, sans-serif}
</STYLE>
```

Причем это правило распространится на ВСЕ элементы <H2>, тогда как при использовании элемента <FONT> пришлось бы прописывать всю сложную конструкцию внутри каждого заголовка. Подробно к полноценной замене элемента <FONT> МЫ еще вернемся, здесь я просто хотел показать на конкретном примере компактность и логичность использования CSS вместо <FONT>.

Надо сказать, элемент <FONT> является единственным столь нежелательным хая использования. Однако существуют атрибуты, которые в спецификации HTML 4.0 тоже помечены как "нежелательные" именно по причине их визуллльности. Например, BGCOLOR.

Атрибут *BGCOLOR*

Этот атрибут определяет фоновый цвет элемента. Применять его можно к тегам <BODY>, <TABLE>, <TD>, <TH>, <TR>. Допустим, код

```
<BODYBGCOLOR="#FFFFFF">
```

сделает фон странички белым. Если применить его к тегу <TABLE>, то он установит фоновый цвет для всей таблицы. Если — к тегу <TR>, то фон будет установлен только для этой строки, а если — к тегу <TD>, то фон будет белым только для данной ячейки.

Вместо атрибута BGCOLOR рекомендуется использовать аналогичное свойство CSS. Например, вот такой код в таблице стилей даст **аналогичный** результат:

```
BODY {
  background-color: #FFF}
```

Если говорить начистоту, то на данный момент использовать CSS для установки фонового цвета не всегда целесообразно по причине обратной совместимости. Например, браузер Netscape 4.x плохо поддерживает стандарт **CSS-1** и вообще не понимает стилей, которые написаны для элемента <BODY>, так что для него придется прописывать атрибут BGCOLOR. А тогда зачем еще и в стилях указывать то же самое?

Конечно, когда пользователей Netscape *4.x* станет меньше одного процента (к моменту выхода книги скорее всего именно так и будет), то можно не обращать на них внимания и писать стили для элемента <BODY>. По крайней мере, для улучшения разделения структуры и визуального представления.

Атрибут *ALIGN*

Кроме BGCOLOR есть атрибут **ALIGN**, который отвечает за выравнивание содержимого блочных элементов.

> ***Определение***
>
> Элемент является блочным, если он визуально форматируется в виде структурной единицы, т. е. представляет собой параграф, заголовок и т. п. (например, <P>, <TABLE>, <UL>).

Для того чтобы выровнять, скажем, текст в параграфе по правому краю, в HTML надо написать следующий код:

```
<HTML>
  <BODY>
    <P ALIGN="i:ight">B этом абзаце текст выровнен по правому краю</P>
  </BODY>
</HTML>
```

То же самое с помощью CSS делается так:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      P.rig |
        text-align: right}
    </STYLE>
  </HEAD>
  <BODY>
    <P CLASS="rig">В этом абзаце текст выровнен по правому краю </P>
  </BODY>
</HTML>
```

Скажу сразу, что часто лучше пользоваться по старинке атрибутом ALIGN, чем вводить **классы** на каждый вид выравнивания (конечно, если вы хотите, чтобы ВСЕ элементы <p> выравнивались однотипно, то классов не понадобится). В данном случае при использовании CSS нет экономии в размере кода, да и атрибут CLASS принципиально мало чем отличается от ALIGN. Конечно, если вы хотите строго следовать стандартам HTML 4.0, то используйте таблицы стилей. Однако, повторяю, с практической точки зрения проще (и в некоторых случаях надежнее) пользоваться атрибутом ALIGN.

Форматирование текста

Вот еще некоторые элементы, вместо которых рекомендуется использовать каскадные таблицы стилей:

- <i> — выводит текст курсивом;
- <в> — выводит текст полужирным шрифтом;
- <тт> — выводит текст моноширинным шрифтом (шрифт пишущей машинки, например, Courier New);
- <BIG> — выводит текст крупным кеглем;
- <SMALL> — выводит текст малым кеглем.

До сих пор понятие ***кегль шрифта*** нигде не фигурировало, так что дадим его определение.

> ***Определение***
>
> Кегль шрифта— это высота области, отведенной под букву. Фактически это есть размер шрифта.

А вот эти элементы помечены в спецификации HTML 4.0, как "нежелательные":

- <STRIKE> и <s> — выводит перечеркнутый текст;
- <и> — выводит подчеркнутый текст.

На мой взгляд, вообще отказываться от использования этих элементов не стоит. Практически всегда проще воспользоваться тегом <в> вместо того, чтобы вводить новый класс для выделения текста полужирным шрифтом вот так:

```
<HTML>
  <HEAD>
    <STYLE TYPE-"text/css">
      .bid {
        font-weight: right}
    </STYLE>
  </HEAD>
  <BODY>
    <P>В этом абзаце есть <SPAN CLASS="bld">cj:o3o</SPAN>, выделенное полужирным</P>
  </BODY>
</HTML>
```

Но если вам нужно выделить полужирным не слово в предложении, а, скажем, заголовок или блок текста, то для этого лучше подходит CSS.

Вообще для выделения рекомендуется использовать элементы <EM> и <STRONG>. На них можно написать какие угодно стили, и пользоваться ими, но если понадобится какой-то третий вид выделения, то снова потребуется вводить класс и писать так:

```
Это слово Судет <EM C1A3Б="вт2">выделено</EM>
```

В HTML набор тегов и так ограничен, так что без веских оснований нет смысла пренебрегать всеми не рекомендованными тегами. Ведь с помощью каскадных таблиц стилей из элемента <тт> можно сделать что угодно. При этом сама структура документа не пострадает, но, правда, *ухудшится ее "логичность"*, так что будьте осторожны.

Все элементы, вплотную связанные с визуальным отображением информации, мы рассмотрели, а теперь непосредственно перейдем к верстке, конкретнее, к верстке с помощью таблиц, потому что любой мало-мальски сложный сайт делается только на табличной основе.

Табличная верстка: достоинства и недостатки

Позиционирование с помощью таблиц

С появлением таблиц произошла глобальная перемена в сфере веб-разработок. У верстальщиков впервые появился инструмент точного позиционирования графических элементов и текста на веб-странице. С развитием браузеров и стандарта HTML таблицы обогатились многими свойствами, которые позволяли еще лучше контролировать визуальное представление информации на странице. В наше время табличная верстка является основой любого сайта, так что без досконального знания таблиц невозможно сверстать что-нибудь сложное.

Конечно, таблицы не для того задумывались, чтобы на их основе верстать сайты. По большому счету, это *противоречит* идее разделения структуры и визуального представления. Однако в языке HTML больше НЕТ других механизмов для точного позиционирования элементов. Так что без использования CSS (или таблиц) просто невозможно сверстать сложный сайт. Хорошая поддержка стандарта CSS в браузерах реализована только начиная с пятых версий, которые появились не так давно, а пользователей четвертых версий браузеров не так уж и мало. Поэтому для обратной совместимости сайты пока приходится верстать с помощью таблиц. Как же это делается? Для начала, давайте рассмотрим процесс создания сайта. Конечно, это будет весьма грубое приближение, однако на данном этапе хватит и его. Надо сказать, что проектирование и разработка веб-сайта — достаточно сложная и нетривиальная задача, для решения которой необходимы знания из многих областей. Некоторые из них мы затронем в последней главе, когда будем с вами разрабатывать конкретный ресурс от самого начала и до конца. Но вкратце процесс выглядит следующим образом.

Вначале разрабатывается концепция сайта, определяются его цели и задачи, целевая аудитория и т. д. Все это делается с помощью информационной архитектуры. Затем разрабатывается структура сайта. Сложность этого процесса зависит от сложности сайта. Для простого сайта структурную схему можно написать за две минуты, а для сложного на разработку структуры отводится пара недель. Но вот структура готова. Затем надо придумать идею, которая бы выделяла этот сайт из числа себе подобных. На Западе такой процесс называется креативом. Подобные сайты лучше запоминаются посетителю, поэтому и приачекают его внимание. Конечно, хороший сайт можно сделать и без какой-то оригинальной идеи, но лучше, чтобы она присутствовала. Затем дизайнер рисует макет сайта, а уж потом этот макет попадает в руки верстальщика, у которого задача одна: *сделать так, чтобы HTML-страница выглядела точно так же, как и этот макет, во всех популяр-*

ных браузерах. Сложно это или просто? Для начала, перечислим популярные браузеры:

G Microsoft Internet Explorer 4.x;

• Microsoft Internet Explorer 5.x;

G Microsoft Internet Explorer 6.x;

• Netscape Navigator 4.x;

G Netscape Navigator *6.x;*

П Mozilla 0.9.x

П Opera 5.x;

G Opera *6.x.*

Фактически, на данный момент всеми этими браузерами пользуются (как это ни парадоксально, до сих пор есть даже пользователи совершенно устаревших третьих версий браузеров, которые отличаются очень слабой поддержкой CSS и значительными различиями в поддержке HTML). Если принять во внимание, что каждый браузер имеет свои особенности и свои недоработки, то поставленная задача кажется практически невыполнимой.

Допустим, к моменту выхода этой книги из печати, пользователей четвертых версий браузеров останется пренебрежимо мало. Тогда, учитывая схожесть MSIE 5 и MSIE 6, а так же NN 6 и Mozilla (они сделаны на основе одного и того же движка Gecko), останутся:

O Microsoft Internet Explorer версии 5JC И 6.X;

• Mozilla 0.9..r/Netscape 6.x;

G Opera версий 5.x и 6,x.

Таким образом, остаюсь три браузера (можно даже сказать, три класса браузеров со своими принципиальными отличиями). Тоже не так уж мало, потому что реализация HTML в этих браузерах по-прежнему разная (хотя различия уже не такие глобальные), а тем более в них разная реализация CSS, потому что стандарт CSS утвержден позже, к тому же эта технология еще развивается, тогда как язык HTML уже нет.

Хорошему веб-мастеру ничего не остается, как знать особенности всех этих браузеров. В частности досконально знать отличия в реализациях HTML и CSS, а также DOM (Document Object Model — объектная модель документа). Прочитав эту книгу, вы будете знать все основные отличия в CSS и кое-какие в HTML. Для изучения объектной модели документа, без знания которой невозможно писать скрипты на языках JavaScript или VBScript, вам понадобится другая книга. Технологии CSS будут посвящены все следующие главы, а пока мы попробуем сверстать страницу *только с помощью HTML.*

Итак, позиционирование с помощью таблиц. Почему таблицы **принципиально** можно использовать для расположения информации на странице? Благодаря атрибутам CELLPADDING, CELLSPACING и BORDER можно сделать таблицу с невидимыми рамками, точнее, рамки будут отсутствовать вовсе. Делается это установкой всех атрибутов в ноль:

```
<TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0">
```

Также в отдельную ячейку можно помешать что угодно: текст, картинку и лаже еще одну таблицу, что в некоторых **случаях** необходимо. Содержимое ячеек можно выравнивать по вертикали при помощи атрибута VALIGN, который может принимать следующие значения:

- top — выравнивание по верхнему краю ячейки;
- middle — выравнивание по среднему уровню ячейки;
- bottom — выравнивание по нижнему краю ячейки.

Также содержимое ячеек можно выравнивать и по **горизонтали** при помощи атрибута ALIGN, который принимает значения:

- left — выравнивание полевому краю ячейки:
- center — выравнивание по центру ячейки;
- right — выравнивание по правому краю ячейки.

Кроме того, можно жестко задавать ширину таблицы и каждой ячейки в пикселах при помощи атрибута WIDTH. И, в дополнение ко всему, задавать для каждой ячейки или для всей таблицы фоповый рисунок, используя атрибут BACKGROUND (в качестве параметра указывается путь к графическому файлу на сервере).

С помощью такой жесткой фиксации ширины и выравнивания можно позиционировать элементы на странице с точностью до пиксела. Если смотреть на вещи глобально, то *практически не существует макета веб-страницы, который нельзя сверстать на основе позиционирования с помощью таблиц.*

Давайте попробуем сверстать реальную страничку. На практике учиться гораздо легче, так что представьте себе, что вы работаете веб-мастером в какой-нибудь дизайнерской студии. И вот одним прекрасным утром (или днем) к вам подходит дизайнер, отдает макет и просит, чтобы вы сверстали HTML-страничку. Пусть этот макет выглядит так, как **показано** на рис. 2.1.

Что делает **сперва** хороший веб-мастер? Он внимательно смотрит на макет и прикидывает в уме, как его "порезать". Проследим за ходом его мыслей:

"Так, у нас есть три колонки. Первая содержит часть логотипа "Энигма" и главные ссылки на разделы ("О компании", "Продукты", "Решения" и т. д.). Вторая колонка содержит вторую часть логотипа и ***ссылка*** *на материалы в разделах ("открытая криптография", "цифровая подпись", "электронный документ" и т. д.). Третья ко-*

лонка содержит графические элементы оформления, название материала и информационную часть (собственно, сам материал). Ага. Логотип и слоган — это графика, а все остальные ***ссылки*** *надо давать обычным текстом, чтобы упростить процесс расширения сайта и уменьшить размер HTML-странички".*

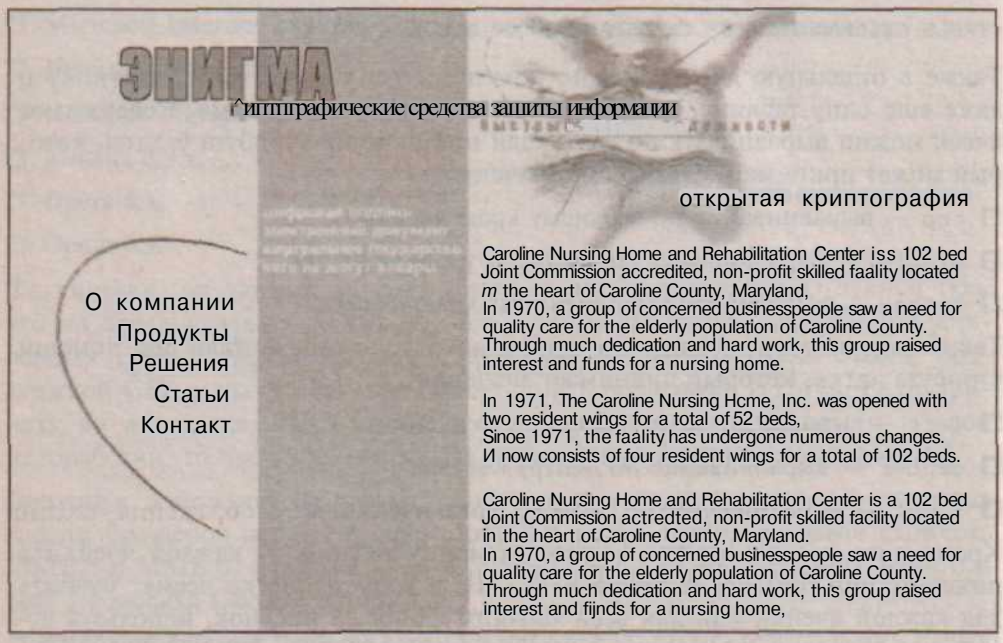


Рис. 2.1. Макет сайта

Здесь мы прервем ход его мыслей и дадим разъяснения. Прежде всего, надо сказать, что очень часто некоторые текстовые элементы на сайтах выполняются в виде графики, потому что в сети Интернет существует огромная проблема со шрифтами. Представьте, что вы написали, скажем, заголовок на странице шрифтом Lazursky. Средствами HTML делается это так:

```
<H1><PONT FACE="Lazursky">3arc^noBOK первого урОВНН</F0NT></Hl>
```

А у посетителя сайта на машине не установлен шрифт Lazursky, поэтому он увидит заголовок, написанный совсем другим шрифтом, скажем, Times New Roman. Но дизайнеру совершенно необходимо, чтобы заголовок был написан именно шрифтом Lazursky и никаким другим, потому что иначе у него композиция нарушается. Вот по этой причине очень часто меню и другие текстовые элементы делают в графическом редакторе, сохраняют в форматах GIF или JPEG и вставляют на страницу рисунком при помощи тега <IMG>. Относительно безопасными шрифтами являются те, которые установлены практически у всех. Это Verdana, Times New Roman, Tahoma, Courier New.

Если вам ну никак не обойтись без шрифта **Futuris,** к примеру, то лучше всего напишите необходимую фразу в Photoshop и вставьте на страницу этот рисунок. Хотя я настоятельно рекомендую поступать так как можно реже, поскольку текст должен быть текстом. На самом деле, это очень **принципиальный** вопрос. Текст — основной носитель информации в Сети со всеми вытекающими последствиями.

Следует отметить, что графические ссылки и заголовки сильно усложняют задачу обновления сайта. Подумайте сами, каждый раз, когда на сайте появляется новая статья, вам придется зайти в **графический** редактор, набрать **название** статьи нужным шрифтом, сохранить в режиме **оптимизации** для веб в формате GIF и вставить этот рисунок тегом <IMG>. Достаточно трудоемкий процесс, особенно если сайт обновляется часто.

Вернемся к нашему веб-мастеру. Он подходит к дизайнеру и говорит, что может быть лучше поменять шрифт *заголовков* и *ссылок на* ***разделы,*** потому что шрифт FreeSet есть далеко не у всех, а вот со шрифтом Verdana проблем не будет. Допустим, дизайнер соглашается и веб-мастер возвращается к своим размышлениям:

"Логотип с верхней картинкой лучше поместить в одну таблицу, а все остальное — в другую, потому что так посетитель сайта раньше увидит часть информации".

Здесь мы снова прервемся и рассмотрим механизм отображения таблиц браузером. Дело в том, что браузер покажет таблицу только тогда, когда она загрузится полностью, т. е. **загрузится** код самой таблицы и все содержимое таблицы (есть способ изменить этот алгоритм, но он поддерживается только браузерами фирм Microsoft и Netscape). Отсюда следует, что если заключить все содержимое документа в одну таблицу, то посетитель сайта не увидит ничего, пока не загрузится вся страница целиком. Если же разбить все содержимое на несколько таблиц, то посетитель будет видеть их по мере загрузки. Согласитесь, гораздо приятнее знать наверняка, что страница загружается, чем смотреть на пустой экран. К тому же в этих таблицах может содержаться информация, которой посетитель воспользуется еще до окончания загрузки всей страницы. Вы, как хороший веб-мастер, решили сделать две таблицы. Двигаемся дальше:

"Заголовки накладываются на рисунок, а это нехорошо, потому что возникают дополнительные трудности. Средствами HTML невозможно наложить текст и рисунок, если только этот рисунок не является фоновым изображением таблицы или ячейки таблицы. Надо бы сдвинуть заголовок немного вниз, думаю, дизайнер не будет возражать. Ну вот, в общем-то, и все. Графики немного, верстка неаожная".

После таких размышлений веб-мастер запускает Adobe ImageReady 3.0 и нарезает макет так, как представлено на рис. 2.2.

Большими цифрами в кружочках обозначены номера графических фрагментов макета, которые нужны для HTML-странички. Только их нельзя реализовать средствами HTML. Как видим, не так уж много. Фрагменты (1) и (2)

разместятся в первой таблице посредством тега <IMG>, фрагмент (3) будет фоном одной из ячеек второй таблицы, потому что поверх него надо сделать панель ссылок. Синюю полосу (на черно-белом рисунке она выглядит светло-серой) легко сделать при помощи атрибута BGCOLOR. Первая и вторая таблицы никак не связаны между собой, поэтому займемся пока первой.

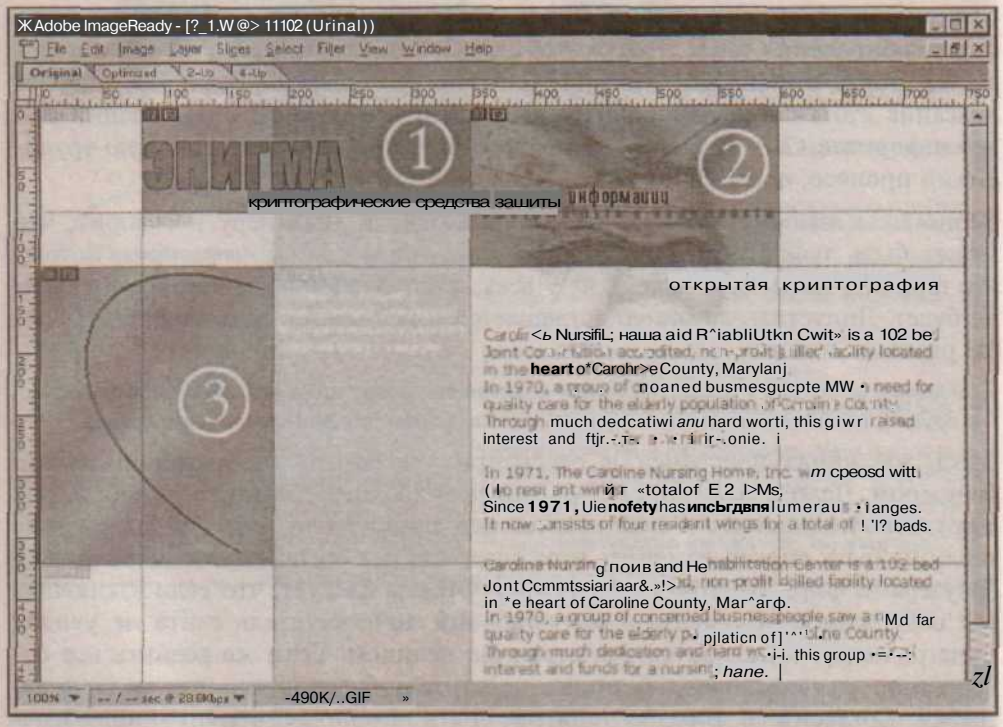


Рис. 2.2. Макет с направляющими в программе Adobe Image Ready 3.0

Общая ширина первой таблицы будет равна 750 пикселам, высота — 167 пикселам. Она будет состоять из двух строк. Глядя на рисунок, можно сказать, что первая строка будет состоять из четырех ячеек:

- первая шириной ровно 82 пиксела содержит только белый фон;
- вторая равна ширине фрагмента (1) (266 пикселов) и содержит как раз весь этот фрагмент;
- третья ячейка равна ширине фрагмента (2) (251 пиксел), содержит его полностью и ничего более;
- четвертая ячейка занимает все оставшееся пространство и содержит только белый фон.

Обратите внимание, что третья ячейка занимает обе строки, так что в ней надо прописать атрибут ROWSPAN="2" ДЛЯ расширения этой ячейки на две строки.

Теперь рассмотрим вторую строк)[7]. В ней должно быть ровно столько же ячеек, сколько и в первой строке. Причем ширина ячеек должна совпадать, т. е. ширина первой ячейки *первого ряда* должна быть равна ширине первой ячейки *второго ряда*. Глядя на второй ряд, сразу же видим несоответствие во второй ячейке. Там содержится одновременно и белый, и синий фон. Как известно, для ячейки можно с помощью атрибута BGCOLOR задать только один цвет. Вариантов решения **два.** Во-первых, можно в эту ячейку вставить таблицу с одним рядом и двумя ячейками, в каждой из которых задать свой лвет фона: в левой ячейке — белый, в правой — синий. Код будет такой:

```
<TABLE>
  <TR>
    СТО BGCOLOR="#FFFFFF" WIDTH="lO7">Snbsp;</TD>
    <TD BGCOLOR="#1D3D4E" WIDTH="159">Snbsp;</TD>
  </TR>
</TABLE>
```

Во-вторых, можно разбить эту большую вторую ячейку на две, но тогда общее число ячеек во втором ряду получится равным пяти, тогда как в первом ряду — четыре. Но эту проблему легко решить, если во второй ячейке первого ряда прописать COLSPAN="2", Т. е. вторая ячейка в первом ряду будет равна *сумме второй и третьей ячеек второго ряда*. Этот вариант мы и выберем, потому что он проще.

Итак, вот код, который соответствует вышеприведенному описанию:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE>3HHTMA — криптографические средства зашиты информэции</T1^E>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF" TOPMARGIN="0" LEFTMARGIN="0"
        MARGINHEIGHT="O" MARGINWIDTH="0">
    <TABLE CELLPADDING="0" CELLSPACING="O" WIDTH="750"
           BORDER="0" HEIGHT="167">
    <TR>
      <TDWIDTH="82"> </TD>
      <TD WIDTH="266" COLSPAN="2"XIMG SRC="img/fragment_l.gif"
                                    WIDTH="266"
                                    HEIGHT-"86"
```

```
                                        АЬТ="ЭНИГМА - криптографические
                                    средства защиты информации [логотип]">
      </TD>
      <TD WIDTH="251" ROWSPAN="2"XIMG SRO"img/fragment_2.gif"
                                        WIDTH="251" HEIGHT="167"
                                        ALT="3HHTMA - быстрый путь
                                            к надежности">
      </TD>
      <TD> </TD>
    </TR>
    <TR>
      <TD WIDTH="82"> </TD>
      <TD WIDTH="107" HEIGHT="81"> </TD>
      <TD WIDTH="159" BGCOLOR="#1D3D4E"> </TD>
      <TD> </TD>
    </TR>
    </TABLE>
  </BODY>
</HTML>
```

На рис. 2.3 для наглядности представлена визуализация данной таблицы.

Рис. 2,3. Визуальное представление таблицы

Разберем некоторые места, которые могут вызывать затруднения. Атрибутами TOPMARGIN и LEFTMARGIN для элемента <ВСЭУ> задаются отступы от верхнего и левого края соответственно. Однако браузер Netscape Navigator эти атрибуты не поддерживает, зато он поддерживает атрибуты MARGINHEIGHT И MARGINWIDTK, которые имеют аналогичное предназначение. В данном случае поля не нужны, по этой причине все значения атрибутов равны нулю. Вообще указывать все четыре атрибута до недавнего времени являлось хоро-

шей практикой, т. к. это обеспечивало универсальность кода. Однако, если забыть о браузере Netscape Navigator 4.v, то этого делать не нужно вовсе. Вообще говоря, правильно и хорошо задавать нулевые отступы для <BODY> С помощью CSS:

```
30DY {
  margin: Opx|
```

Однако мы договорились, что при верстке этой страницы вообще не будем **пользоваться** CSS, так что придется действовать по старинке.

Обратите внимание на элементы <IMG>. Все они имеют атрибуты WIDTH, HEIGHT и ALT, но ни один не имеет атрибута BORDER. ДЛЯ ускорения загрузки таблицы надо прописывать размер рисунка, поэтому без атрибутов WIDTH И HEIGHT не обойтись. Атрибут ALT нужен для того, чтобы посетители **сайта,** пользующиеся текстовыми или голосовыми браузерами, не теряли информации. В голосовых браузерах всегда произносится альтернативный текст, который задается в атрибуте ALT, если же атрибут ALT отсутствует, то вместо этого голосовой браузер будет просто произносить слово КАРТИНКА. Если атрибут ALT есть, но в нем ничего не указано, то голосовой браузер просто пропустит эту картинку, потому что посчитает ее неважной. Согласитесь, гораздо приятнее слышать, что па графическом элементе изображен логотип компании ЭНИГМА или не слышать ничего (если графическое изображение является элементом дизайна и не несет никакой смысловой нагрузки), чем десять раз просто слово **КАРТИНКА.**

Кроме того, некоторые пользователи для ускорения загрузки отключают картинки. Тогда на месте графического изображения браузер выведет прямоугольник с альтернативным текстом внутри. Если атрибута ALT нет, то браузер выпедет прямоугольник с надписью IMAGE.

Атрибут BORDER нужен только в том случае, когда рисунок является ссылкой. Чтобы убрать рамку вокруг рисунка, являющегося ссылкой, надо прописать 3ORDER="0". В данном случае картинка не является ссылкой, поэтому атрибут BORDER имеет значение ноль по умолчанию, так что для сокращения размера кода прописывать его явно не следует.

Вам может показаться странной следующая конструкция:

```
<TD WIDTH="82">£nbsp;</TD>
```

в которой символ snbsp; обозначает неразрывный пробел. Дело в том, что Netscape Navigator 4 просто игнорирует ячейки, которые ничего не содержат. **Так** что если написать:

```
<TDWIDTH="82"></TD>
```

то этот браузер просто не примет в расчет эту ячейку и сдвинет все остальные ячейки таблицы влево, невзирая на прописанную ширину для ячейки в 82 пиксела. Так что для браузера Netscape Navigator надо вставить в ячейку

хоть что-нибудь, лишь бы она не была пустой. Неразрывный пробел является неплохим кандидатом из-за своей невидимости. Конечно, если вам потребуется точно заданная ширина, скажем, в три пиксела, то придется вставить невидимый рисунок в формате GIF.

А сейчас перейдем ко второй таблице, которая гораздо проще, чем первая. Она состоит всего из одной строки и трех ячеек. Первая ячейка имеет ширину 189 пикселов. В ней находится главное меню и фрагмент (3) в качестве фонового рисунка. Вторая ячейка имеет ширину 159 пикселов и синий фон. В ней находится подменю. В третьей — основной текст. Попытаемся закодировать вторую таблицу в HTML.

```
<TABLE CELLPADDING="10" CELLSPACING="O" WIDTH="750" BORDER="0"
HEIGHT="400">
<TR VALIGN="top">
  <TD WIDTH="172" BACKGROUND="img/fragment_3.gif" ALIGN="right">
  <BRXBR>
  <A HREF="#"XFONT COLOR="#000000" FACE="Arial" SIZE="3">
    <B>0 KOMTiaHHH</Bx/F0NTX/AXBR>
  <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
     SIZE="3"><B>npOflyKTbi</B></FONT></A><BR>
  <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
     SIZE="3"xB>PemeHMfl</Bx/F0NT></A><BR>
  <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
     SIZE="3"XB>CTaTbM</B></F0NTX/A>
  </TD>
  <TD WIDTH="142" BGCOLOR="#1D3D4E">
  <A HREF="#"><FONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
    <B>открытая криптография</B></FONT></AXBR>
  <A HREF="#"><FONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
    <B>цифровая noflnHCb</BX/FONTX/A><BR>
  <A HREF="#"><FONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
    <B>электронный flOKyMeHT</BX/F0NTX/AXBR>
  <A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
    <B>чего не могут xaKepbi</BX/FONTX/AXBR>
 </TD>
  <TD WIDTH="392">
  <H3 ALIGN="right"><roNT COLOR="#000000" FACE="Arial">
    открытая криптография</ГОЫТх/Н3>
  <PXF0NT SIZE="2" COLOR="#000000" FACE="Verdana">Bbi запускаете браузер,
набираете в адресной строке URL сайта и получаете желаемую страницу. Вот,
вкратце, действия пользователя, который бродит в сети Интернет. Он не
```

```
„лльшается, что лежит за всем этим. Ему это не нужно. Но это нужно нам,
проф•2ссиональным веб-разработчикам, чтобы максимально удовлетворить поль-
зовательские запросы.</P>

<P>Мы не будем углубляться в историю и отряхивать пыль с архивных 90-х
_-:дов, но базовые понятия рассмотрим. Итак, что же лежит в основе всемир-
ной :ети Интернет? Базовыми являются три технологии:
</FO:rrx/p>
  </TD>
</TR>
«: 'TABLE>
```

То, что увидим в браузере, представлено на рис. 2.4.

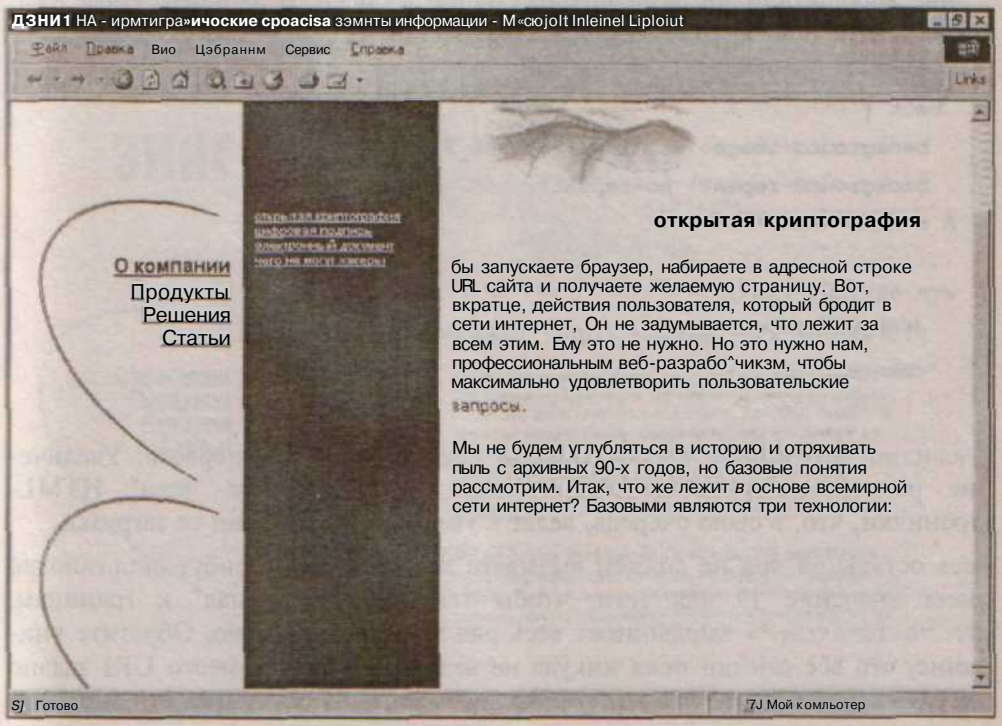


Рис. 2.4. Вид странички в браузере MSIE 5.0

Все бы хорошо, но фоновый фрагмент (3) продублировало^. Это естественно, потому что фоновый рисунок всегда занимает всю доступную ему площадь. В данном случае по горизонтали ему размножаться негде, зато по вертикали — есть.

От этого можно избавиться двумя способами.

- Первый способ: можно сделать фрагмент (3) очень большим по высоте, тогда на любых мониторах не будет заметно повторения (для этого надо сделать высоту фрагмента равной примерно 1600 пикселам). Делается это следующим образом. Берете графический редактор, к примеру, Photoshop 5.5, и открываете в нем файл fragment_3.gif Затем устанавливаете в качестве фонового цвета фоновый цвет файла, в данном случае белый. После этого в меню **Image** выбираете команду **Canvas Size,** в поле **Height** устанавливаете значение 1600, а в селекторе **Anchor** выбираете верхний средний квадрат, после чего нажимаете кнопку OK. В итоге получаем необходимое изображение размером 188x1600.

G Второй способ заключается в использовании CSS. С помощью CSS можно задать для данной ячейки фоновый рисунок и запретить его повторение вообще или же разрешить по одной из осей. В данном случае надо запретить повторение вообще. Делается это в блоке <STYLEX/STYLE> С помощью правила:

```
.back {
  background-image: url{img/fragment_3.gif);
  background-repeat: no-repeat}
```

А в HTML-коде:

```
...
<TR VALIGN="tOp">
  <TD WIDTH="172" ALIGN="right" CIASS="back">
  <BRXBR>
...
```

Очевидно, что второй способ гораздо предпочтительнее первого. Увеличение размера рисунка неизбежно ведет к увеличению "веса" HTML-странички, что, в свою очередь, ведет к увеличению времени ее загрузки.

Весь остальной код не должен вызывать затруднений. Атрибут CELLPADDING имеет значение 10 для того, чтобы текст не **"прилипал"** к границам, <TR VALiGN="top"> выравнивает весь ряд по верхнему краю. Обратите внимание, что все ссылки пока никуда не ведут, т. е. в них вместо URI задано HREF="#". Это потому, что мы еще не знаем адресов остальных страниц сайта, так что на первом этапе верстки страницы можно просто поставить такие заглушки.

Взгляните еще раз на макет, предостаатенный дизайнером, и на вид странички в браузере. Несоответствие заключается в том, что на экране все ссылки подчеркнуты, а на макете — нет. Надо сказать, что средствами **HTML** эту проблему не решить. Можно, конечно, сделать данные ссылки

графическиии, однако это очень неуклюжий выход. Зато избавить ссылки от подчеркивания можно с помощью CSS, делается это следующим образом:

```
<STYLE TYPE="text/css">
...
A {
  text-decoration: none}
...
</3TYLE>
```

Тогда все ссылки на странице будут без подчеркивания. Но мы договорились вначале не использовать CSS, так что конечный вид странички остается таким, как представлено на рис. 2.5.

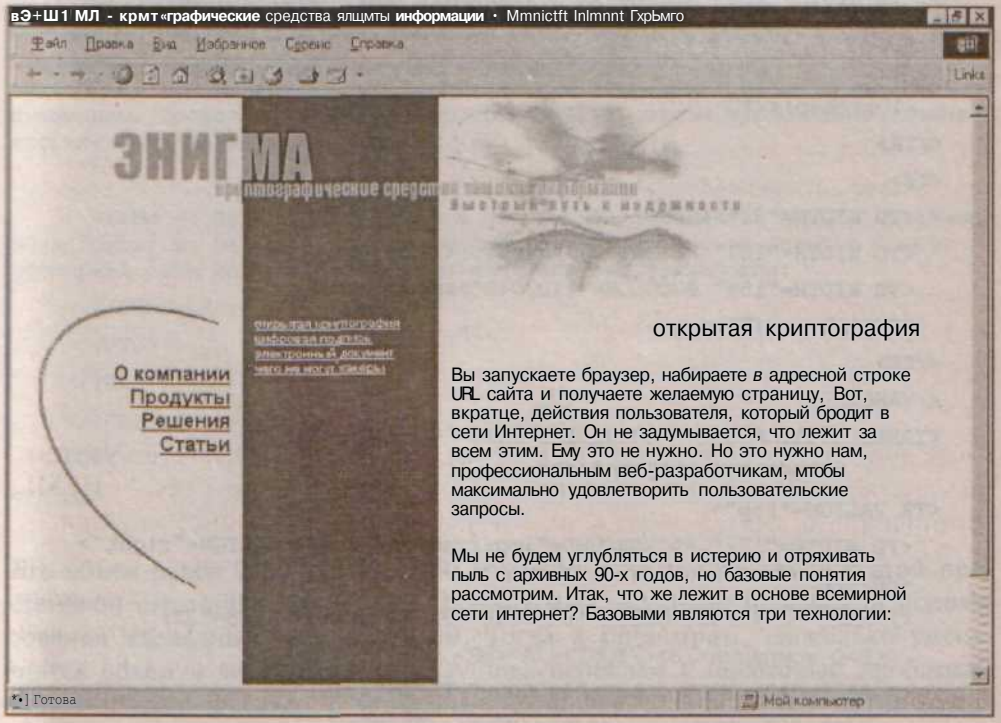


Рис. 2.5. Вид странички в браузере MSIE 5.0 (ссылки подчеркнуты)

Ниже приведем код целиком:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN'">
<HTML>
  <HEAD>
```

```
    <T1TЬE>ЭНИГМА — криптографические средства защиты информации</T1TЬE>
</H£AD>
<BODY BGCOLOR="#FFFFFF" TOPMARGIN="0" LEFTMARGIN="O" MARGINHEIGHT="O"
      MARGINWIDTH="O">
  <TABLE CELLPADDING="O" CELLSPACING="O" WIDTH="750" BORDER="0"
         HEIGHT="167">
  <TR>
    <TD WIDTH="82"> </TD>
    <TD WIDTH="266" COLSPAN="2"XIMG SRC="img/fragment_l .gif"
                                    WIDTH="266" HEIGHT="86"
                                    АЬТ="ЭНИГМА — криптографические
                        средства защиты информации [логотип)"></TD>
    <TD WIDTH="251" ROWSPAN="2"XIMG SRC="img/fragment_2 .gif"
                                    WIDTH="251" HEIGHT="167"
                          ALT="3HKTMA - быстрый путь к надежности"x/TO>
    <TD> </TD>
  </TR>
  <TR>
    <TD WIDTH="82">Snbsp;</TD>
    <TD WIDTH="107" HEIGHT="81"> </TD>
    <TD WIDTH="159" BGCOLOR="#1D3D4E"> </TD>
    <TD>finb3p;</TD>
  </TR>
  </TABLE>
  <TABLE CELLPADDING="10" CELLSPACING-"O" WIDTH="750"
         BORDER="0" HEIGHT="400">
  <TR VALIGN="top">
    <TD WIDTH="172" BACKGROUND="img/fragment_3.gif" ALIGN="right">
    <BRXBR>
    <A HREF="#"XFONT CO3,OR="#000000" FACE="Arial" SIZE="3">
      <B>0 KOMnaHMM</Bx/F0NTx/A><BR>
    <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
       S12E="3"><B>Продукты</B></FONT></A><BR>
    <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
       SIZE="3"><B>PemeHHfl</B></FONT></A><BR>
    <A HREF="#"><FONT COLOR="#000000" FACE="Arial"
       SI2E="3"XB>CTaTbM</Bx/FONTx/A>
    </TD>
    <TD WIDTH="142" BGCOLOR="#1D3D4E">
```

```
<A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
   <B>открытая криптография</B></ГОЫТ></A><BK>
<A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
   <B>цифровая noflnMCb</B></FONTX/AxBR>
<A HREF="#"><FONT COLOR="#FFFFFF" FACE="Arial"
   312E=-="1"xB>электронный    flOKyMeHT</Bx/F0NTX/AXBR>
<A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE="1">
  <B>чего не могут хаКерbi</B></FONTx/A><BR>
</TD>
<TD WIDTH="392">
<H3 ALIGN="right"><FONT COLOR="#000000" FACE="Arial">
   открытая криптография</ГОЮГх/нз>
<PXFONT SIZE="2" COLOR="#000000" FACE-"Verdana">Bbi запускаете
браузер, набираете в адресной строке URL сайта и получаете желаемую стра-
ницу. Вот, вкратце, действия пользователя, который бродит в сети Интер-
нет. Он не задумывается, что лежит за всем этим. Ему это не нужно. Но это
нужно нам, профессиональным веС-разработчикам, чтобы максимально удовле-
творить пользовательские запросы.</P>

<P>Мы не будем углубляться в историю, и отряхивать пыль с архивных
-^-х годов, но базовые понятия рассмотрим. Итак, что же лежит в основе
всемирной сети Интернет? Базовыми являются три технологии:
</FONTX/P>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Его объем равен 2500 байтам. При изучении CSS мы вернемся к этой простенькой страничке, чтобы вы в полной мере осознали полезность использования каскадных таблиц стилей. Тогда и посмотрим, насколько уменьшится объем и логичность кода. А пока перейдем к некоторым проблемам использования таблиц, с которыми частенько сталкиваются начинающие веб-мастера.

Хитрости использования таблиц

Таблицы — вещь непростая, так что с ними частенько возникают проблемы различного рода. Все мы здесь разбирать не будем, а остановимся на самых распространенных. К примеру, браузер Netscape Navigator весьма некор-

ректно обрабатывает фоновые изображения в таблицах. Давайте рассмотрим конкретный пример. Допустим, вам понадобилось вставить фоновый рисунок в таблицу. Вы пишите такой код:

```
<TABLE CELLSPACING="O" CELLPADDING="O" BORDER=*"0"
BACKGROUND^i img/table_back.gif" WIDTH*"196">
<TR>
  <TD>1</TD>
  <TD>2</TD>
  <TD>3</TD>
  <TD>4</TD>
</TR>
<TR>
  <TD>a</TD>
  <TD>a</TD>
  <TD>a</TD>
  <TD>a</TD>
</TR>
<TR>
  <TD>b</TD>
  <TD>b</TD>
  <TD>b</TD>
  <TD>b</TD>
</TR>
<TR>
  <TD>c</TD>
  <TD>c</TD>
  <TD>c</TD>
  <TD>c</TD>
</TR>
</TABLE>
```

Браузер Microsoft Internet Explorer делает все логично (рис. 2.6).

А вот некоторые браузеры, например Netscape Navigator 4.x, Netscape *6.x,* Mozilla 0.9.-Y, поступают с фоновым рисунком так, как представлено на рис. 2.7, т. е. пытаются поместить фоновый рисунок в каждую ячейку таблицы.

Нам это совершенно ни к чему. Однако надо что-то делать. Выход есть, хотя и совершенно не очевидный. Заключается он в использовании вложенных таблиц.

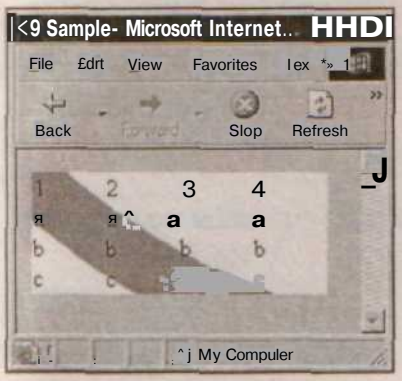


Рис. 2.6. Вид таблицы с фоном в браузере MSIE 5.x

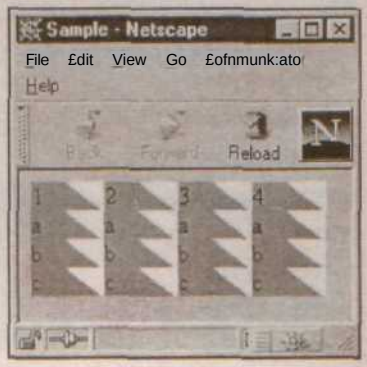


Рис. 2.7. Вид таблицы с фоном в браузере Netscape Navigator 4.x

Для корректного отображения фонового рисунка в данных браузерах надо создать таблицу из одной ячейки и прописать нужный нам фоновый рисунок в атрибуте BACKGROUND элемента <TABLE>. Затем в эту единственную ячейку первой таблицы вставить вторую таблицу с нужным нам числом ячеек и строк, но в атрибуте BACKGROUND ничего не прописывать (несмотря на такое нелогичное действие, наличие пустого атрибута BACKGROUND является необходимым). Вот рабочий кол, который устраняет проблему:

```
STABLE CELLSPACING="O" CELLPADDING="O" BORDER="0"
BACKGROUND-"ijng/table_back.gif"  WIDTH~"196">
<TR>
  <TD>
    <TABLE CELLSPACING»"!'1 BACKGROUND="" CELLPADDING="O" BORDER»"0"
           WIDTH="100%">
    <TR>
      <TD>K/TD>
      <TD>2</TD>
      <TD>3</TD>
      <TD>4</TD>
    </TR>
    <TR>
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
    </TR>
```

```
    <TR>
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
    </TR>
    <TR>
      <TD>C</TD>
      <TD>c</TD>
      <TD>C</TD>
      <TD>c</TD>
    </TR>
    </TABLE>
  </TD>
</TR>
</TABLE>
```

Согласитесь, что такое решение проблемы совершенно не очевидно и самостоятельно его найти вряд ли удастся. На самом деле в языке HTML много мелких и крупных особенностей, которые выявляются только на практике. Иными словами, пока вы не сверстаете л-ое количество сайтов, со всеми тонкостями не разберетесь. Хотя эта книга поможет во многом.

Кроме фоновых рисунков, иногда бывает необходимо сделать таблицу с тонкими рамками шириной в один пиксел, причем рамки эти могут быть произвольного цвета. Если прописать в элементе <TABLE> атрибут BORDER="1", то рамка получится псевдотрехмерной и достаточно убогой, так что не к любому дизайну она подойдет. Поэтому проблема остается актуальной.

Если рассуждать логически, то приходит на ум следующее решение. Берем таблицу с одной единственной ячейкой и устанавливаем там фоновый цвет, скажем, зеленый BGCOLOR="green", который в итоге будет цветом рамок. Далее в эту ячейку помещаем нужную нам таблицу, причем заметьте, что ширина первой и второй таблиц должна совпадать. Расстояние между ячейками второй таблицы указываем равным одному пикселу (в итоге это будет ширина рамок), прописав CELLSPACING="1". Также необходимо задать фоновый цвет таблицы, скажем, белый BGCOLOR="white". В итоге сквозь однопиксельные щели во внутренней таблице будет проглядывать внешняя таблица, которая имеет зеленый фон. Иными словами, мы как бы взяли зеленый прямоугольник, и положили на него белую решетку со щелями между белыми прямоугольниками толщиной в один пиксел. Так что мы должны получить таблицу с белым фоном и зелеными рамками.

Вот код:

```
<TABLE CELLSPACING="O" CELLPADDING="O" BORDER="0" BGCOLOR="green"
       WIDTH="196">
<TR>
  <TD>
    <TABLE CELLSPACING="1" CELLPADDING="O" BORDER="0" BGCOLOR="white"
           WIDTH="100%">
    <TR>
      <TD>K/TD>
      <TD>2</TD>
      <TD>3</TD>
      <TD>4</TD>
    </TR>
    <TR>
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
    </TR>
    <TR>
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
    </TR>
    <TR>
      <TD>C</TD>
      <TD>C</TD>
      <TD>c</TD>
      <TD>c</TD>
    </TR>
    </TABLE>
  </TD>
</TR>
</TABLE>
```

Однако HTML не всегда поддается логике. Вот что мы увидим в браузере Netscape Navigator 4.x (рис. 2.8).

А вид таблицы в браузере Microsoft Internet Explorer 5.x представлен на рис. 2.9.

Рис. 2.8. Вид таблицы в браузере Netscape Navigator 4.x

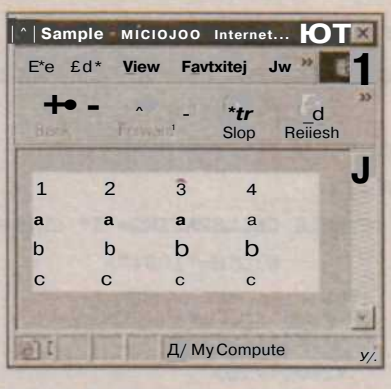


Рис. 2.9. Вид таблицы в браузере MSIE 5.x

В данном случае логично действует только Netscape Navigator, который закрашивает всю область вложенной таблицы, кроме пространства между ячейками. А Microsoft Internet Explorer закрашивает и сами ячейки, и пространство между ячейками. Однако этого не происходит в том случае, если фоновый цвет второй таблицы задавать отдельно для каждого ряда таблицы. т. е. для каждого элемента <TR> надо задать атрибут BGCOLOR="white". Размер кода слегка увеличится, но зато проблема решится:

```
<TABLE CELLSPACING="O" CELLPADDING="O" BORDER="0" BGCOLOR="green"
       WIDTH="196">
<TR>
  <TD>
    <TABLE CELLSPACING^'!." CELLPADDING="O" BORDER="0" BGCOLOR="white"
           WIDTH-"100%">
    <TR BGC0LOR»"whita">
      <TD>K/TD>
      <TD>2</TD>
      <TD>3</TD>
      <TD>4</TD>
    </TR>
    <TR BGCOLOR="white">
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
      <TD>a</TD>
    </TR>
    <TR BGCOLOR="white">
```

```
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
      <TD>b</TD>
    </TR>
    <TR BGCOLOR="white">
      <TD>c</TD>
      <TD>c</TD>
      <TD>c</TD>
      <TD>c</TD>
    </TR>
    </TABLE>
  </TD>
</TR>
</TABLE>
```

Этот код будет одинаково отображаться всеми браузерами. Пожалуй, это самые распространенные проблемы, связанные с таблицами, которыми интересуются начинающие веб-мастера, так что на этом и остановимся.

А. сейчас оставим таблицы и перейдем к остальным аспектам верстки. Например, частенько возникает вопрос.

Почему мои страницы такие "тяжелые"?

В среде веб-разработчиков есть сленговое выражение *"вес страницы"*. Имеется в виду размер HTML-файла **и** графики в килобайтах. Таким образом, проблему уменьшения веса документа можно разбить на две:

3 оптимизация кода;

3 оптимизация графики.

Как оптимизировать **[рафику,** мы рассматривать не будем, т. к. это выходит за рамки данной книги. Вообще надо сказать, что оптимизация графики обычно важнее, чем оптимизация кода, но ситуации бывают разные, а для больших посещаемых сайтов нельзя пренебрегать ничем. Особенно это важно для электронной коммерции типа B2C. Там скорость загрузки страниц — один из важнейших параметров успеха бизнеса. Так что займемся вплотную оптимизацией кода.

Для начала надо определить, какой вес является большим. Вообще это сильно зависит от предназначения сайта. Возьмем, к примеру, презентационный сайт. Его цель — эффектно подать материал, привлечь внимание, громко заявить о себе. Для этого максимально используется графика и флэш-технология, так что насыщенная графикой HTML-страница будет ве-

сить килобайт 60—80. Для презентационного сайта это нормально. Однако для информационного сайта или контент-сайта это непомерно много. Для них вес страницы не должен превышать 30—40 Кбайт, так что приходится минимизировать использование графики и важным является каждый лишний байт кода. Особенно остро встает проблема оптимизации кода для сайтов с большим числом посещений.

Чтобы улучшить восприятие информации об оптимизации сайта, я решил представить ее в виде советов. Итак, по порядку.

- Делайте максимально короткими имена файлов и каталогов. Например, для хранения картинок вместо распространенного названия каталога *images* лучше использовать просто /. Тогда все **пути** к графическим файлам вместо SRC="/images/picturel..gif" будут SRC="/i/picturel. gif". А если еще имя файла сократить, то получится SRC="/i/pl.gif". Таким способом можно уменьшить объем средней страницы байтов на 500.

G Нигде не прописывайте лишних атрибутов. В этом смысле код должен быть максимально компактным. Допустим, в элементе <IMG> атрибут BORDER нужно устанавливать в ноль только в том случае, если картинка является ссылкой, потому что в противном случае BORDER="0" ПО умолчанию. Вовсе не обязательно всегда и везде прописывать высоту и ширину ячеек в таблице, но тут надо смотреть по обстоятельствам. Общая стратегия такая: удаляете сомнительный атрибут и смотрите, как это повлияло на отображение страницы браузером. Если никак, то он не нужен.

O Согласно спецификации HTML, все атрибуты должны быть заключены в кавычки. Однако, как показывает опыт, браузеры прекрасно обрабатывают страницу и с атрибутами без кавычек. Так что вместо <A HREF="index.html">rviaBHaH<:/A> с легким сердцем можно писать <A HREF=»index.html>T\naBHafl</A>. Но надо быть крайне осторожным в различных скриптах, потому что синтаксис скриптовых языков не позволяет просто так не ставить кавычек. К вашему сведению, на кавычках можно сэкономить байтов 300—600.

- Нужно удалять все лишние пробелы и переводы строк. Каждый лишний пробел и каждый лишний символ перевода строки равен одному байту. Так что вместо:

```
<TR>
<TD>    </TD>
</TR>
```

лучше написать:

```
<TR><TDX/TDX/TR>
```

- Старайтесь поменьше использовать вложенные таблицы. Браузер быстрее обработает и отобразит код, в котором мало вложенных таблиц. Вообще,

чем проще таблицы, тем лучше. Идеальным решением было бы отказаться от таблиц вовсе, но исключительно средствами HTML этого добиться невозможно.

!1 Если вы используете JavaScript, который не является критичным при отображении страницы, то лучше поместить его в конце секции BODY. Потому что, если помешать его в секции HEAD, то документ не отобразится, пока не загрузится скрипт.

Здесь рассматривались основные способы уменьшения HTML-кода. Существует много других способов увеличить скорость загрузки страницы, но о них мы поговорим позже.

Как угодить поисковым машинам

В настоящее время найти необходимую информацию в Сети становится все сложнее, потому что критическим образом возрастает число сайтов. В каталогах накопилось столько ссылок, что разгребать неимоверное их количество модераторам не хватит ни времени, ни терпения. По этой причине основным инструментом поиска информации в Интернете стали поисковые системы типа Google, Yandex, AltaVista.

Механизм функционирования таких систем примерно следующий. Существует программный модуль, который путешествует по Сети и собирает информацию, из которой впоследствии формируется база данных ресурсов Интернета. Этот модуль называют *пауком,* от английского *spider.* На самом деле он представляет собой простейший браузер, который точно так же формирует запрос к серверу, сервер отсылает пауку HTML-документ, который анализируется и добавляется в базу данных системы, по которой и совершается поиск, когда пользователь вводит ключевые слова. Главная особенность поисковых систем — сортировка найденных по запросу страниц согласно их релевантности, т. е. адекватности критериям поиска. Алгоритмы оценки релевантности страниц зачастую держатся в секрете, но кое-что, конечно, известно.

Как определяется релевантность[17]. Параметров много. Вот некоторые из известных.

3 Большой вес имеет количество ключевых слов на странице. Скажем, если у вас на странице слово "дизайн" встречается 30 раз, то очень велика вероятность того, что при поиске по слову "дизайн" ваш сайт будет иметь достаточно большую релевантность.

3 В некоторых поисковых системах, например в Rambler, имеет значение то, насколько близко к началу документа находится ключевое слово. Чем оно ближе, тем больший вес имеет. Иными словами, если на вашей странице слово "дизайн" стоит первым, то релевантность при поиске по

этому слову будет выше, чем если бы слово "дизайн" находилось в конце страницы.

- Ключевые слова, заключенные в теги <TITLEX/TITLE>, <H#X/H#>. <вх/в>, <STRONG></STRONG>, <EMX/EM>, имеют больший вес. Значит, если страница связана с дизайном, то лучше слово "дизайн" поместить между тегами <TITLEX/TITLE>, а в тексте выделить его полужирным или сделать заголовком. Например так:

```
<TTTЬE>Дизайн информационных систем</T1TЬE>
. . .
<P>Когда говорят о <STRONG>jiH3aUHe</STRONG> информационных систем,
часто приводят неуклюжее сравнение</P>
```

- В таких поисковых системах, как Google и Yandex, одним из основных критериев оценки релевантности является *индекс цитирования.* Фактически, он представляет собой число ссылок, ведущих на ваш сайт. Индекс цитирования довольно логичен. Чем больше на сайт ведет ссылок, тем популярнее сайт у пользователей Интернета, а значит он **заслуживав!** внимания. Кроме того, имеет значение, какой именно сайт ссылается на ваш. Если ссылка находится на никому не известном и малопосещаемом сайте, то она имеет меньший вес, чем ссылка, находящаяся на сверхпопулярном сайте. На индекс цитирования оказывает влияние и длительность нахождения сайта в базе данных поисковой системы. Чем дольше находится в ней сайт, тем выше его вес.

О При запросе в несколько слов учитывается "расстояние" между ключевыми словами на странице. Таким образом, если вы ввели в поисковую форму два слова, скажем, *Netscape CSS,* то страница, на которой эти ключевые слова разделены двадцатью другими словами, будет обладать большей релевантностью, чем страница, на которой они разделены тридцатью словами.

Чем выше релевантность страницы, тем на более высоком месте ссылка на эту страницу будет стоять в результатах поиска. Следовательно, выше и вероятность того, что пользователь поисковой системы зайдет на данный сайт. Таким образом, владельцы сайтов должны стремиться обеспечить максимальную адаптацию своих страниц для поисковых систем, чтобы повысить релевантность. Иначе при прочих равных условиях их сайты лишатся значительного числа посетителей.

Нас, в первую очередь, интересует, что можно сделать средствами HTML. А сделать можно многое. В первую очередь, рассмотрим секцию <KEAD> И элемент <META>, С ПОМОЩЬЮ которого можно задавать описание страницы и набор ключевых слов для поисковых систем. Более важными являются ключевые слова и задаются они следующим образом:

```
<META NAME="keywords" CONTENT="Netscape Opera браузер design верстка">
```

Надо сказать, что данная конструкция задумывалась для того, чтобы помочь поисковым машинам индексировать **сайты.** И раньше действительно ключевые слова были важны. Однако впоследствии развилось и широко распространилось такое явление, как снам. Создатели сайтов задавали популярные ключевые слова, по которым часто осуществляется поиск в Интернете, например, sex, в надежде, что на их страницу придут посетители. Также использовалось повторение одного **и** того же слова несколько раз, потому что в этом случае искусственно увеличивалась релевантность страницы. Очевидно, что чем чаще встречается искомое слово на странице, тем важнее она хтя пользователя. Со спамом **стали** бороться следующими способами.

- Поисковые машины стали полнотекстовыми, т. с. они анализируют весь текст страницы, а не только ключевые слова. Причем частенько производится сравнение ключевых слов с текстом. По этой причине нет особого смысла указывать в качестве ключевых те слова, которых вообще нет на странице.
- Если вы употребили какое-либо ключевое слово более трех раз, то многими поисковыми системами это воспринимается как спам и данная страница не индексируется. Так что не стоит пользоваться таким приемом.

Как видите, сейчас ключевые слова не являются для поисковой машины приоритетными при индексации ресурса. Однако внимание им уделить все же стоит. Вот несколько советов по составлению списка ключевых слов.

- Старайтесь слова, используемые в элементе <TITLE>, включать в список ключевых слов.
- Некоторые поисковые системы различают прописные и строчные буквы, так что названия (типа Netscape) лучше прописывать дважды (netscape Netscape). Первое — полностью строчными буквами, второе — с первой заглавной буквой. Как показывает опыт, пользователи часто вводят поисковую фразу исключительно строчными буквами.
- Не используйте запятых для разделения ключевых слов. Многие поисковые системы читают только первые 200 символов, так что не стоит тратить полезное пространство на запятые, отделяйте слова одним пробелом.
- Используйте разные ключевые слова на разных страницах сайта. Чем выше специфичность, тем выше вероятность, что вашу страницу найдут по правильному запросу.

Что касается описания страницы, то задается оно так:

```
•'-МЕТА NAME="description" CONTENT="WEB-анатомия // HTML и CSS для
-рофессионалов">
```

Данный тег имеет некоторое значение. Поисковые системы часто выводят его в качестве описания ссылки в результатах поиска. Так что чем интерес-

нее и привлекательнее будет описание, тем с большей вероятностью пользователь зайдет по этой ссылке на ваш сайт.

Учтите, что почти все поисковые системы выводят только первые 170 символов описания, поэтому оно должно быть достаточно лаконичным. Часто в этот тег помещают то же самое, что и в тег <TITLE>, однако гораздо лучше хотя бы слегка изменить текст.

Все остальные теги <META> НИКОИМ образом не повысят релевантность вашей страницы, а некоторые серьезно помешают. Например, тег

```
<META HTTP-EQUIV="refresh" CONTENT=10;URL=http://www.newurl.by">
```

через 10 секунд перенаправит посетителя на сайт www.newurl.by. Многие поисковые системы рассматривают использование данного тега как спам и не индексируют такие страницы. Делают они так из-за того, что данный тег взят на вооружение многочисленными порносайтами. Первая страница с этим тегом содержит множество ключевых слов для максиматьного увеличения релевантности, а пользователь практически сразу попадает на другую страницу (ведь можно задать переход по ссылке и в течение одной секунды). Наличие данного тега многими поисковыми машинами воспринимается как показатель того, что страница содержит необъективную информацию и неверный список ключевых слов, что, по сути дела, является *спамом*.

С элементом <META> МЫ разобрались, но кроме него существует еще много способов повысить релевантность вашей страницы. Например, многие поисковые системы считают главными на странице фразы, находящиеся в тегах <TITLEX/TITLE>, <н#х/н#> и <в></в>, так что к заголовкам на странице нужно относиться очень внимательно. Лучше сделать информативный заголовок, чем интригующий и загадочный, но, конечно, лучшим решением было бы совмещение и того, и другого.

Некоторые поисковые системы в качестве описания сайта приводят первые 20—25 слов. Так что лучше в начале каждой страницы помещать самую важную информацию. Например, если это страница с какой-либо статьей, то можно вначале поместить краткое содержание статьи, тогда описание ссылки на эту статью в результате поиска будет информативным.

Большим препятствием для пауков при сканировании страницы являются таблицы. Происходит этого по той простой причине, что паук сканирует страницу линейно, тогда как человек видит ее уже отформатированной. Допустим, у нас есть таблица из одной строки и двух столбцов. В первом столбце располагается панель ссылок, а во втором — основной текст. Для нас в графическом браузере панель ссылок и текст будут находиться на одном уровне. А вот паук вначале прочитает ссылки, и только потом весь текст, который, собственно, и является информацией на дайной странице. Учтите этот момент при оптимизации сайта для поисковых систем.

И еще пара советов, которыми не следует пренебрегать.

- Во всех значимых с точки зрения информации рисунках прописывайте альтернативный текст в атрибуте ALT. ЭТО ОДИН ИЗ способов увеличить количество ключевых слов на странице. Надо сказать, что отсюда вытекает один из способов спама, когда для однопикселного невидимого изображения прописывают очень длинный ALT, содержащий большое количество ключевых слов, С точки зрения увеличения релевантности этот прием может помочь, но с точки зрения "чистоты кода" делать этого не следует. Правда, бывают случаи, когда страница содержит много **[рафики** и почти не содержит текста. Например, каталог фотографий. Вот тогда таким приемом можно воспользоваться. Однако далеко не все пауки читают альтернативный текст.

- При составлении списка ключевых слов старайтесь, чтобы они встречались в тексте.

Л уже упоминал о спаме, а теперь давайте подробнее рассмотрим, какие бы-,lют методы спама. Надо сказать, что в настоящее время поисковые машины научились различать многие из них, так что не стоит рисковать, иначе ваш сайт и вовсе могут не проиндексировать. Вот чего не надо делать.

- Не надо использовать на странице текст такого же цвета, как и фон страницы, чтобы спрятать ключевые слова от браузеров, но оставить их видными паукам. Почти все поисковые системы научились бороться с таким способом увеличения релевантности и не индексируют такие страницы.

- Не надо повторять одни и те же ключевые слова а теге <META>, а также делать заголовки типа *"Наши веб-дизайнеры лучшие веб-дизайнеры из всех веб-дизайнеров мира"*. Согласитесь, что воспринимается этот заголовок просто кошмарно, хотя и увеличивает релевантность.

- Не стоит размещать на странице популярные ключевые слова, которые не имеют отношения в тематике страницы. Так делают в надежде, что зашедший пользователь заинтересуется другой информацией. Однако это является очень плохим тоном. Посудите сами, если вы ищете в браузере информацию о группе Metallica, а попадаете на сайт о CSS, то это вызовет недоумение и раздражение за потраченное впустую время. Кроме того, распространение такого приема приведет к весьма негативным результатам, потому что найти в Сети что-либо станет несколько труднее.

- Не надо помещать большие объемы ключевых слов в отдельный слой (layer), а затем делать его невидимым с помощью CSS. Однако можно сделать невидимый слой, разместить его сразу после тега <BODY> и вставить в этот слой *небольшое* описание сайта. Тогда поисковые машины, которые в качестве описания выводят первые несколько строк, будут выводить именно это описание.

Вот основные и самые распространенные виды спама. Существуют гораази более сложные и изощренные, однако их рассмотрение выходит за рамки данной книги.

Надо сказать, что оптимизация страниц для поисковых систем — это лишь **одни** in этапов так называемой "раскрутки" или продвижения сайта. Все это входит в понятие интернет-маркетинг. Однако хороших книг на русском языке по интернет-маркетингу иы не найдете, так что заказывайте англоязычные издания через **Amazon.com,** либо ищите статьи и прочую информацию в Сети.

Что еще можно сказать о HTML? Вернее, что нам обязательно понадобится при изучении CSS? Сложный вопрос. О HTML говорить можно долго, обстоятельно объясняя каждую мелочь, однако это нам ни к чему. Но кое-чте впоследствии может вызвать вопросы с вашей стороны. Для предотвращения таковых коснемся еще одной темы.

Логические теги

Вообще с этим понятием вы можете нигде больше не столкнуться, но я выделил некоторые теги в отдельный класс. Вот их определение.

> ***Определение***
>
> Теги, которые в спецификации HTML определены, как формирующие структуру текста, назовем логическими.

В этом есть смысл. Данные теги сильно помогают правильно организовать логическую структуру страницы, а правильная структура страницы значительно упрощает использование CSS. Согласитесь, это не так уж мало. Итак. мы будем говорить о так называемых логических тегах. Дтя начала простс перечислим их вместе со смысловыми значениями:

О <н#> — заголовки разных уровней;

- <EM> — выразительность;
- <STRONG> — ударение;

П <CITE> — цитата;

- <ADDRESS> — адрес;
- <SAMP> — пример;

П <CODE> — код или листинг программы;

- <VAR> — переменная.

Вот они все. А сейчас займемся каждым в отдельности.

О <н#> обозначает заголовки в теле документа. Бывает шести видов <H1>, <H2>, <нз>, <H4>, <H5>, <нб>, которые отличаются степенью значимости

<FONT> с каким-либо классом, на который и пишется стиль. Конечно, это ухудшает логичность страницы.

Пример. Код:

```
<C1TE>Самоучитель сайтостроителя</C1TE>, Минск, 1989<BR>
<C1TE>Сайт кривыми ручками</C1TE>, Москва, 1999
```

Итог:

Самоучитыь сайтостроит&vt, Минск, 1989

Сайт кривыми ручками, Москва, 1999

- <ADDRESS> обозначает контактную информацию автора сайта или организации, которой принадлежит сайт. Во всем остальном аналогичен элементу <C1TE>.

  Пример. Код:

```
Наш адрес: <P>
<ADDRESS>
г. Минск, ул. Сторожевская 8, офис 142а<BR>
тел. 210-10-16
</ADDRESS></P>
```

  Итог:

  Наш адрес:

  г. Минск, ул. Сторожевская 8, офис 142а

  тел. 210-10-16

- <SAMP> обозначает пример. Обычно применяется для обозначения терминов или ключевых слов, а также примеров взятых откуда-либо. В браузере отображается моноширинным шрифтом, например, Courier New. В общем-то он похож на тег <CODE>.

  Пример. Код:

```
Что такое <ЗАМР>каскадные таблицы стилей</ЗАМР> знает все большее число людей
```

  Итог:

  Что такое каскадные таблицы стилей знает все большее число людей

- <CODE> обозначает программный код. В браузере отображается моноширинным шрифтом. Часто используется вместе с тегом <PRE>, который сохраняет все существующее форматирование текста, а потому лучше передает отступы и структуру. Обычно вместо элемента <CODE> чего только *m* используют. Но все это нарушает логичность структуры страницы.

Пример. Код:

```
<CODE>
function Opn(){<BR>
  newWindow=window.open (' ','NW','width','toolbar=0');<BR>
}<BR>
</CODE>
```

Итог:

```
function Opn(){
newWindow=window.open{'  ','NW,'width','toolbar=0');
}
```

- <VAR> обозначает переменную из компьютерной программы. По умолчанию браузеры отображают текст в тегах <VARX/VAR> наклонным шрифтом. Конечно, далеко не всегда вообще требуется выделять переменные, однако если захотите это сделать, то всегда пользуйтесь элементом <VAR>.

Пример. Код:

```
Переменная <VAR>width</VAR> содержит ширину нового окна
```

Итог:

Переменная *width* содержит ширину нового окна

Как вы заметили, большинство из этих тегов делают текст или курсивным, или полужирным, или моноширинным. Возникает вполне резонный вопрос, почему бы вместо этого не использовать просто теги <в> или <1>, ведь это гораздо проще? Так и делает подавляющее большинство веб-мастеров, однако есть причины поступать по-другому, и это не только упрощение использования CSS. Логические теги позволяют браузерам обрабатывать страницу в манере, наиболее свойственной данному браузеру, лаже если пользователь отключил поддержку CSS по какой-то причине. Допустим текст в тегах <EMX/EM> браузеры под Windows выделяют курсивом, а браузеры под Unix — полужирным. Кроме того, спустя полгода вернувшись к коду страницы, вы легко сможете определить, что тег <ADDRESS> в коде вашей странички обозначает адрес, тогда как, скажем, конструкция <SPAN CLASs="adr"> далеко не так интуитивна, да и в самой таблице стилей проще запутаться.

На мой взгляд, информации, содержащейся в предыдущих двух главах, вполне достаточно, чтобы прилично сверстать HTML-страничку без использования CSS. Однако даже сейчас, когда мы лишь слегка прикоснулись к каскадным таблицам стилей, вам, наверное, видны многие ограничения и недостатки языка HTML. Это должно служить хорошим стимулом для изучения CSS, потому что данная технология позволяет легко решать очень многие проблемы и обходить ограничения языка разметки гипертекста.

Изучением CSS мы вплотную займемся со следующей главы, а пока самое время поговорить о будущем HTML.

Перспективы развития HTML

Как вам уже известно, на данном этапе развития языков разметки для Интернета стандартом де-факто является HTML 4.01, а стандартом де-юре язык XHTML 1.0. По замыслу создателей языка XHTML, он должен стать переходным от языка HTML к языку XML. Мы рассмотрим все плюсы и минусы использования XHTML со стороны верстальщика. По сути дела, ознакомившись с данным материалом, вы сможете сделать осмысленный выбор в пользу того или иного языка.

Основной вопрос звучит так: *"Стоит ли сейчас использовать XHTML или же дождаться нормальной поддержки XML от производите. ^ браузеров?"* На самом деле это достаточно сложный вопрос, но мы все же попытаемся разобраться в нем и дать ответ. Итак, что принципиально нового в языке XHTML? Как это ни странно звучит, но практически ничего. Все тот же ограниченный набор тегов, все та же негибкость и плохая структурированность информации. Основные отличия касаются синтаксиса. Естественно, если создатели языка XHTML хотели упростить с помощью него миграции на XML, то они должны были унифицировать синтаксис, т. е. синтаксические правила XML и XHTML должны совпадать. Давайте рассмотрим, чем отличается синтаксис HTML и XHTML. Основных отличий четыре.

- Все элементы в XHTML должны иметь закрывающий тег, тогда как в HTML некоторые элементы не имеют закрывающих тегов вовсе, например,
 и <IMG>. И, кроме того, есть элементы, для которых закрывающий тег необязателен. Это, например, элементы , <TD>, <LI>.

  В XHTML *все* элементы должны иметь закрывающий тег. Поэтому нале обязательно закрывать тег <P> тегом . Для элементов, которые вообще не имеют закрывающего тега, предусмотрена сокращенная запись
 <IMG/>. Кстати, браузер не поддерживающий стандарт XHTML, не поймет тега
, он будет думать, что это есть неизвестный элемент
. Но если перед слэшем поставить пробел, то браузер решит что это элемент
 с неизвестным атрибутом / и корректно обработает эту конструкцию, просто проигнорировав знак /, т. е. для совместимости надо писать
 и <IMG />. Так что в этом плане не потребуется больших усилий для адаптации уже существующих страниц noz новый стандарт.

П Необходимо соблюдать правильную вложенность элементов. Например, в HTML можно написать:

```
<P><A HREF="index.html"XB>rnaBKaH</AX/BX/P>
```

и любой браузер правильно отобразит эту конструкцию. Однако синтаксис XHTML строже. Так что придется переместить тег </в> влево от тега </A> для сохранения корректной вложенности:

```
<P><A HREF^"index.html"><B>Главная</BX/AX/P>
```

Соблюдение этого пункта тоже не потребует больших усилий. К тому же в спецификации HTML 4.01 правильная вложенность также является обязательной.

- Все атрибуты без исключения должны быть заключены в кавычки. В HTML можно было написать:

```
<SPAN class=bfont>
```

А в XHTML придется написать

```
<span class="bfont">
```

Так что надо быть более внимательным, но, кроме того, кавычки занимают два байта в каждом атрибуте, а на приличной странице встречается порой до 300 атриб\тов, так что кавычки будут занимать 600 байтов, а это достаточно много, особенно если трафик на сайт большой.

На очень многих сайтах нет кавычек вообще, потому что их отсутствие позволяет увеличить скорость загрузки страниц без особых на то усилий. Многие веб-мастера привыкли кодировать без кавычек, так что придется изменить привычкам. Но, кроме того, понадобится исправить множество шаблонов и страниц, а это очень рутинная работа.

- Все элементы должны быть в нижнем регистре. Если в HTML можно было писать

```
<TABLE BORDER=0 WIDTH=100>
```

то в XHTML допустимо только

```
<table border="0" width="100">
```

И это, пожалуй, самая большая трудность для некоторых веб-мастеров, в частности для меня. Многие привыкли писать HTML-код прописными буквами, потому что в этом случае отличия элементов HTML от конструкций JavaScript и текста на странице становятся более заметными. В этом случае "читать" код страницы проще, а это существенный плюс.

Давайте подведем итог. Для перехода на XHTML нам придется закрывать ьсе теги, везде ставить кавычки и писать код в нижнем регистре. Это суще:твенные изменения привычек, так что должны быть веские доводы, чтобы 5еб-мастера решились на этот шаг.

Например, консорциум W3C в пользу XHTML выдвигает следующие доводы.

Разработчики документов и разработчики браузеров откроют новые пути выражения своих идей через новую разметку. В XML относительно легко

вводить новые элементы или новые атрибуты. Язык XHTML разработан для согласования этого расширения посредством специальных XHTML-модулей, которые можно разрабатывать совершенно самостоятельно. Эти модули позволят комбинировать существующие и новые возможности при разработке новых документов и новых браузеров.

В настоящее время активно развиваются новые альтернативные пути доступа в Интернет. Некоторые специалисты говорят, что в 2002 году 75% страниц будет просматриваться с альтернативных платформ (наладонники, мобильные телефоны и пр.). XHTML разрабатывался с учетом этого, и он должен обеспечить лучшее преобразование документов. В конечном счете. *можно будет разрабатывать* XHTML-конформные документы, которые будут доступны из любого XHTML-конформного браузера.

Основная идея — разбиение языка на отдельные модули. Согласитесь, что для мобильного телефона и обычного компьютера представление информации разное. *В перспективе* будет существовать множество модулей для тех или иных целей. Конечно, по умолчанию браузер будет поддерживать лишь несколько базовых модулей. Но ведь свобода в этом плане не ограничивается, так что, несомненно, появится множество самых различных модулей для описаний того или иного плана. Возникает проблема совместимости. Планируется, что браузеры смогут самостоятельно получать через Сеть необходимое программное обеспечение (что-то вроде плагинов) для отображении документов, которые используют неизвестные модули.

Но все это в будущем, Пока XHTML поддерживается очень слабо. Вообще говоря, кроме изменения синтаксиса ничего и нет. Так что пока не появятся новые браузеры с полной поддержкой XHTML, с реализацией *механизмов обновления модулей,* пока не появятся сами модули — нет смысла переходи i. на синтаксис XHTML. Да и тогда, я уверен, поддержка HTML останется для обратной совместимости.

Более того, на данный момент совершенно не ясно, станет ли язык XHTMI стандартом де-факто для сети Интернет. Очень велика вероятность, что BCL начнут переходить непосредственно на XML. Так что надо запастись терпением и ждать. А пока его использование лишено смысла, потому что переход к новому синтаксису (это единственное, что можно сделать сейчас) не дает совершенно ничего.

Кроме XHTML есть еще XML, который, собственно, и должен стать по идее тем самым стандартным языком разметки гипертекста в сети Интернет. Ка: это ни странно, его поддержка реализована в браузерах достаточно хорошо тогда как реализация поддержки XHTML самая примитивная на уровне внедрения синтаксиса. Это объясняется тем, что XML начал развиваться гораздо раньше: первая версия языка XML появилась на сайте консорциум. W3C 10 февраля 1998 года, а первая версия XHTML — 26 января 2000 года.

Все дело в том, что стандарт XML внедрялся уж очень вяло и медленно, для ускорения процесса и смягчения перехода был разработан XHTML. Но в последнее время ситуация несколько изменилась, язык XML стали в той или иной степени поддерживать практически все разработчики браузеров.

Перед нами стоит вопрос, нужно ли уже сейчас переходить на XML и что этот переход даст веб-разработчикам? Для ответа надо хотя бы кратко ознакомиться с новым языком и его возможностями.

Для начала, вот какие преимущества имеет XML перед HTML:

T Язык HTML описывает то. *как тот или иной фрагмент должен пониматься браузером,* а XML описывает то, *что этот фрагмент обозначает.* Это принципиальная разница. Если в HTML мы напишем:

```
<H1>Введение в HTML</H1>
<H4>Тим Бернерс-Ли</H4>
<P>Лучшая книга по языку HTML всех времен и народов!</P>
```

то в XML это же можно написать так:

```
<Ьоо1^дЛ1е>Введение в HTML</booktitle>
<bookauthor>TMM Бернерс-Ли</Ьоокаи^ог>
Obstract^ywmafl книга по языку HIML всех времен и народов!</atostract>
```

Как видите, в первом случае мы имеем набор тегов, документ размечен без всякой логики. Между тегами <HIX/HI> может находиться все, что угодно и тот факт, что фраза заключена в эти теги, нам ни о чем не говорит, кроме того, что это заголовок. Во втором случае мы четко и однозначно знаем, что между тегами <booktitie></booktitle> находится название книги.

Таким образом. XML-документ *гораздо логичнее структурирован,* чем HTML-документ.

- В языке HTML существует ограниченный набор тегов, что делает его совершенно негибким. В языке XML набор тегов не ограничен, вы можете придумывать какие вам заблагорассудится. Отсюда вытекает максимальная адаптируемость XML для очень многих видов задач. Фактически, на базе XML можно создавать свои собственные языки разметки для специфических нужд, например, для описания книг в книжном магазине или для расписаний рейсов в аэропорту.

Пока на этом остановимся и поговорим о том, как же браузер будет отображать неизвестные ему теги. Действительно, неограниченный набор тегов вызывает недоумение. На самом деле для форматирования XML-документа на стороне клиента необходима еше одна вещь — XSL, хотя в некоторых случаях вместо XSL можно использовать CSS. Кроме XSL, иногда бывает надо создавать еще и DTD, а для каких случаев это необходимо, рассмотрим чуть позже.

XSL — это язык стилей для XML, наподобие CSS для HTML. Надо сказать, что для форматирования XML-документа можно использовать и CSS, однако это несколько ограничивает возможности XML. Так, вы не сможете с помощью CSS преобразовать XML-документ в документ формата RTF или PDF, вы не сможете обеспечить взаимодействие CSS и структуры документа, т. к. в этом плане CSS ограничен и придется использовать какой-нибудь язык сценариев. Всех этих ограничений лишен язык XSL.

С помощью XSL можно преобразовывать XML-документы в формата HTML, RTF, PDF и др. С помощью XSL можно манипулировать данными например, сортировать, производить по ним поиск, добаатять данные и: браузера, минуя веб-сервер. Очевидно, это сильно разгрузит сервер и значительно уменьшит трафик.

Таким образом, для отображения XML-документа в обычном визуально\ веб-браузере надо проделать следующие действия:

1. Разметить документ в соответствии с синтаксисом XML, используя произвольные теги.
2. Написать XSL-шаблон, который преобразует XML-документ в HTML-документ для отображения в браузере.

Все HTML-документы имеют одно DTD (вернее, три модификации одно: и того же DTD), а XML-документы могут иметь разные DTD. опрелеленн: автором документа, отсюда и следует неограниченность набора тегов.

Написание DTD вовсе не является необходимым. По сути дела, если **нал** сать DTD, то это будет уже новый язык разметки гипертекста, базируюшп; ся на XML. Если вы используете XML исключительно для собственна нужд у себя на сайте, то в этом нет необходимости. Однако в случае, ее вы хотите обмениваться данными, вам придется документировать все ваи_ теги, чтобы другие люди знали, что значит тот или иной тег.

Файл DTD содержит набор правил о том, какие элементы с какими **атрибутами** могут встречаться в документе, какие данные может содержать тот и иной элемент. Фактически, именно в DTD мы описываем введенные на* теги, такие как **<booktltle>,** <abstract> и <bookauthor>. Синтаксис Г'TD достаточно сложный и нетривиальный, так что написать свой собствен: язык разметки очень непросто.

Кроме XSL и DTD есть еще несколько технологий, предназначенных .: использования вместе с XML. Например, Xlink (XML Linking Language) Namespaces (пространство имен).

Технология Xlink предназначена для расширения существующей модели : пересылок. В настоящее время в языке HTML ссылки только однонапр.- ленные, а в Xlink можно делать более сложные ссылки: между двумя ;;оку- ментами, ассоциировать метаданные со ссылкой и многое другое.

Технология Namespaces создана для устранения конфликтов, которые могут возникать при пересечении элементов с одинаковыми именами. Например, в одном документе элемент name может означать имя человека, а в другом — название какого-нибудь товара. Namespaces обеспечивает простой метод уникального определения имен элементов и атрибутов XML путем привязки их к URI, т. е. именно **URI** служат для различия одного элемента пакета от другого.

Пора рассмотреть еще один очень важный вопрос: в какой степени поддерживается XML различными браузерами. Итак...

- **Mozilla 0.9.6 (ноябрь 2001)**

  На данный момент поддерживает связку XML + CSS1, т. е. документ в формате XML можно вывести на экран с помощью каскадных таблиц стилей. Поддержка XSLT ожидается в ближайшее время.

- **Netscape** 6

  Ситуация аналогичная, поддерживается связка XML + CSS. Имеется некоторая поддержка технологии Xlink и поддержка Namespaces. Поддержка XSL обещается, но разработка двигается как-то вяло.

- **Opera 5-6**

  Поддерживает XML + CSS. Не поддерживает **XSLT,** и вроде бы разработчики не собираются включать его поддержку в ближайшее время. Поддерживает Namespaces в сокращенном варианте.

- **Internet Explorer 5-6**

  Это единственный браузер, который, уже начиная с пятой версии, поддерживает XSLT. В шестой версии реализована поддержка XSLT 1.0 и технологии Namespaces. Зато интеграция XML + CSS весьма слабая.

Как видите, полноценная обработка XML-документов на стороне клиента возможна только в браузерах фирмы Microsoft. Надо отметить, что это более 70% всех пользователей, но все же не 100%. Так что пока самой приемлемой считается связка XML + CSS, хотя и его можно рассматривать лишь как временное решение до появления хорошей поддержки XSLT.

Второй путь заключается в использовании XML/XSL на стороне сервера. При таком подходе можно определять браузер, с помощью которого посетитель пытается просмотреть сайт, и трансформировать XML-документ в подходящий формат. Например, для обычного веб-браузера сервер выдаст HTML-документ, для мобильного устройства— WML-документ, для речевого браузера — в свойственном ему формате. За счет этого достигается максимальная доступность сайта с любых интернет-устройств. Актуальность такого использования XML вызывает сомнения. По крайней мере, надо

тщательно проанализировать все плюсы и минусы, чтобы склониться к такому варианту.

Конечно, далеко не всегда использование XML вместо HTML оправдано У языка HTML есть свои достоинства:

О он достаточно прост в использовании и прост в обучении;

- он прекрасно поддерживается браузерами;

П он хорошо подходит для визуального отображения обычных документов, включающих текст и картинки.

Так что для некоторых задач HTML является идеальным решением. Например, для различного рода домашних страниц, электронных публикаций online-изданий.

Зато XML гораздо лучше подходит для интернет-магазинов, различного род; каталогов, регистрационных данных и всего остального, где информации четко структурирована.

Будущим языком разметки, скорее всего, станет XML, однако произойдет это не ранее, чем захотят производители браузеров. Надо отметить, чтс стандарт CSS-t принят в 1996 году, а полная его поддержка в большинстве браузеров появилась только в 2000 году. Стандарт XSL до сих пор активна разрабатывается и усовершенствуется, так что достаточно полной поддержк; в браузерах надо ожидать не ранее 2003—2004 года. А пока единственны безопасным решением для Интернета является язык HTML.

Резюме

Итак, надо бы подвести промежуточный итог и обобщить особенности языка HTML.

- Основной проблемой яачяется очень слабый контроль над визуальны представлением информации на странице:
  - мы с помощью HTML практически не можем контролировать параметры шрифта и текста;
  - мы не можем задавать возможные параметры сразу для нескольким элементов;
  - визуальные элементы, типа <FONT>, значительно ухудшают логичност и структурность документа, что приводит к дополнительным трудностям при редактировании кода.
- Вторая проблема заключается в ограниченности тегов языка HTML. Про стота изучения и внедрения HTML ведет к негибкости языка. Он хорошс подходит для online-публикаций, но плохо для интернет-магазинов ; прочих сложных систем.

Вторая проблема решается только переходом на XML, а вот первая проблема нам особенно интересна. Она совершенно естественна, потому что язык HTML разрабатывался для *разметки гипертекста,* но не для его *форматирования*. Именно отсюда **и** вытекают все проблемы, т. е. мы пытаемся использовать HTML не по **его** прямому назначению. Это все равно, что рисовать картину поленом, макая его в краску, или забивать декоративные гвозди **гирей.** *Идя* форматирования документа надо использовать инструмент, который специально для этого и создавался. *И этот инструмент называется "Каскадные таблицы стилей".*

Глава 3

Для чего вам CSS: как появились каскадные таблицы стилей

В предыдущих главах вы кратко познакомились с языком HTML и с ізерс кой сайтов посредством HTML. Вы уже имеете представление, что мо* н сделать с помощью языка разметки гипертекста, а чего нельзя. В этой гл рассматриваются причины возникновения и эволюция CSS. Таким обра, прочитав эту главу, вы поймете предназначение каскадных таблиц стиле:- также их возможности.

Необходимость разделения контента и дизайна

Для начала давайте определимся с терминологией, иначе будет неясно, от чего надо отделять. Итак...

> ***Определение***
>
> Контент — это информационная часть документа, в которую входят текст люстративные рисунки, таблицы, графики.

> ***Определение***
>
> Дизайн— это отображение информационной части документа на каком-- устройстве.

Например, на экране монитора это будет визуальное отображение, а в і совом браузере, соответственно, речевое отображение. Для простоты приятия мы будем рассматривать только визуальный дизайн, хотя все суждения можно без особого труда перенести на любые виды дизайна.

Давайте рассмотрим, каким образом на данный момент формиру- HTML-документы, и какие явные недостатки отсюда вытекают. Допусти нас есть страница, которая представляет собой каталог книг. Каждая ка имеет **следующие** атрибуты;

- П название;
- автор;

О краткое описание;

- фотография обложки;
- цена.

Именно так представляется анонс книг на сайтах большинства электронных книжных магазинов. Мы не станем рассматривать структуру всего интернет-магазина, для наших целей вполне достаточно будет рассмотреть структуру одного этого блока. Типичная организация анонса выглядит так: слева располагается фотография обложки, справа — название книги, автор, описание, а в самом низу — цена. Средствами HTML это реализуется с помощью таблицы, которая содержит одну строку и два столбца. Допустим, согласно дизайну сайта, название нам надо написать полужирным шрифтом Verdana черного цвета, фамилию автора — курсивным шрифтом Verdana черного цвета, описание — обычным шрифтом Verdana черного цвета, цену — полужирным шрифтом Verdana красного цвета. Ниже следует код, который в точности соответствует такому описанию:

```
<TABL£ BORDER="0" CELLSPACING="O" CELLPADDING="10" WIDTH="4Q0">
<TR VALIGN="top">
  <TDXIMG SRC="cover.jpg" WIDTH="100" HEIGHT="132" BORDER="0"
          АЬТ="Веб-дизайн. Книга Дмитрия Кирсанова"></TD>
  <TD><FONT FACE="Verdana" SIZE-"2"><B>BeC-BH3aftH</B></F0NT><BR>
  <FONT FACE="Verdana" SIZE*"l"XI>fl. KnpcaHOB</I><BR>
Книга написана замечательно. В каждой главе видна тщательная работа с
материалом и столь же тщательный подбор слов и метафор. Нигде Дмитрий не
позволяет себе обойтись штампом, общими словами и описанием того, в чем
сам не разобрался.<BRX/FONT>
  <FONT FACE="Verdana" SIZE="2" COLOR="red"><B>104 руб.</BX/FONT>
  </TD>
</TR>
</TABLE>
```

Вид страницы в браузере представлен на рис. 3.1.

Если рассматривать этот пример с точки зрения контента и дизайна, то сразу же становится видно, что в HTML-коде эти категории переплетаются, и с первого взгляда сложно сказать, что есть *информация,* а что есть *средство визуального представления информации.* Например, фотография обложки — это файл cover.jpg, однако в теге <IMG> данная информация теряется, потому что она находится в атрибуте SRC, а кроме него тег <IMG> содержит еще несколько атрибутов. Более того, если в коде присутствуют элементы графического оформления страницы, то отличить элемент оформления от графического элемента контента можно только по расположению элемента <IMG>. Обобщая эти рассуждения, можно сделать вывод, что при такой разметке

понятие контент размывается и отсутствует четкая структурировании информации. Давайте проанализируем, к чему это ведет.

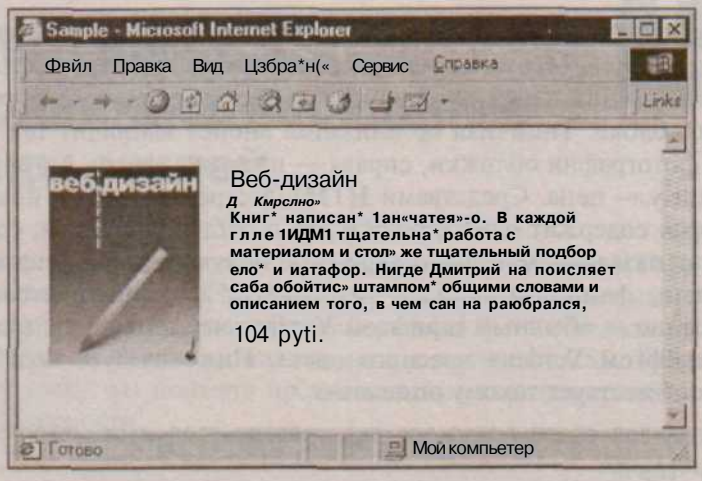


Рис. 3.1. Визуальное представление анонса книги

Самым очевидным следствием является трудность изменения дизайна страницы. Посмотрите на код. В нем жестко задано, что фотография находит^-слева, а описание книги — справа. В нем жестко заданы общая ширина фотографии и описания, начертание, pajMcp и цвет шрифта, а также выравшвание фотографии и текста по верхнему краю. А теперь давайте оставим век информацию прежней, но попытаемся изменить ее представление на стр^шще. Допустим, нам понадобилось:

О перенести название книги, автора и иену вправо от фотографин обложк и выровнять их по правому краю;

- описание книги выровнять не по верхнему краю, а по центру;
- изменить весь шрифт на Arial;
- увеличить шрифт названия книги на одну единицу;

Я увеличить общую ширину фотографии и описания с 400 до 500 пикселов

П сделать рамку шириной в один пиксел.

В этом случае нам надо будет переписать весь код. Он станет таким:

```
<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="O"
       WIDTH="500" BGCOLOR="black">
<TR>
  <TD>
    <TABLE BORDER="0" CELLSPACING^"!" CELLPADDING="10" WIDTH="100%">
```

```
    <TR VALIGN="top" BGCOLOR="white">
      <TD ALIGN="right" width="110">
      <FONT FACE="Arial" SIZE="3"><B>Веб-ОТЗаНН</B></F0NT><BR>
      <FONT FACE="Arial" SIZB-"1"XI>A. КНрсаНОВ</IX/F0NTXBR>
      <FONT FACE="Arial" SIZE="2" COLOR-"red"XB>104 руС.</BX/FONT>
      </TD>
      <TDXIMG SRC-"cover.jpg" WIDTH="100" HEIGHT="132" BORDER="0"
              ALT="Веб-дизайн. Книга Дмитрия Кирсанова"x/TO>
      <TD VALIGN="middle">"<FONT FACE="Arial" SIZE-"1">
Книга написана замечательно. В каждой главе видна тщательная работа с материалом и столь же тщательный подбор слов и метафор. Нигде Дмитрий не позволяет себе обойтись штампом, общими словами и описанием того, в чем сам не разобрался.<BRX/FONT>
      </TD>
    </TR>
    </TABLE>
  </TD>
</TR>
</TABLE>
```

Внешний вид страницы после преобразований представлен на рис. 3.2.

Рис. 3.2. Видоизмененный анонс книги

Как видите, понадобились коренные изменения в структуре кода страницы и добавление значительных фрагментов. Пришлось воспользоваться вложенными таблицами для создания рамки, потому что иначе средствами

HTML такую рамку не реализовать. Причем это совершенно простой и тр) виальный пример, но даже на нем ясно видно, что изменение визуально: представления информации неизбежно ведет к коренному изменени: структуры страницы. Иными словами, *при изменении дизайна изменяется структура информации.* Очевидно, что средствами HTML невозможно дс биться разделения *контента и дизайна.* Но чем же это плохо? Чтобы ОТІ тить на этот вопрос, надо рассмотреть принципиальные подходы к созд; нию сайтов.

Во-первых, весь **сайт** целиком можно сделать только на основе HTML.

ПЛЮСЫ:

О низкая себестоимость проекта;

О быстрота обработки запросов по сравнению со страничками, **генерир)** мым1! на основе баз данных (это совершенно ясно, потому что ког; страничка генерируется, то затрачивается время на обращение к ба: данных и на обработку программным модулем полученной информации

Минусы:

О сложность обновления веб-сайта;

• сложность создания больших сайтов с числом страниц более 30—40.

Во-вторых, аналогичный сайт можно сделать на основе языков программ* рованпя **Perl** или PHP, с помощью которых можно генерировать HTM страницы. Информация для наполнения страниц в таком случае содержит:! в реляционной базе данных, например, MySQL или MSSQL. Для генерак страниц достаточно нескольких шаблонов, а если сайт совсем простой, то одного. Проблема замедления загрузки сайта решается следующим образе Все необходимые страницы генерируются заранее, и только потом разм-. щаются на сервере. Фактически, пользователи сразу получают HTM страницу в ответ на запрос.

ПЛЮСЫ:

О относительная простота обноатения ресурса;

П относительная простота изгото&1ения больших сайтов, потому что в да ном случае число страниц практически не имеет значения.

Минусы:

О удорожание проекта, т. к. приходится привлекать программистов и у: личивается время, затраченное на разработку сайта;

О в том **случае,** если все страницы генерируются заранее, необходимы серпе:.], большой мощности, иначе на генерацию большого сайта (общим объем, около 600 Мбайт) понадобится целый день, а это сильно затрудняет устрш ние мелких недочетов в шаблонах и внесение небольших изменений.

'. -евидно, что простота обновления сайта, в случае реализации его вторым гсобом, следует из того, что:

- информация содержится в базе данных, поэтому она уже прилично структурирована;
- для полной смены дизайна необходимо переделать всего несколько шаблонов. Хотя если сайт очень большой, то число шаблонов доходит до 30, а это уже немало.

•'-як, для легко обновляемого сайта необходимы:

- HTML-шаблоны (зачастую с фрагментами PHP и Perl);
- отдельные скрипты на языках PHP или Perl;
- реляционная база данных.

~ги этом программисту совершенно необходимо знать хотя бы основы TML, а верстальщику хотя бы поверхностно ознакомиться с **PHP.** Но да- •.. несмотря на это, достаточно часто возникают ситуации, когда программу нужна помощь верстальщика, потому что он не знает некоторых -•ансов кода, а верстальщику — помощь программиста, потому что пра- ; льная организация шаблонов лучше известна именно программисту.

_ •: реальное положение **дел** в настоящий момент. Как мы уже установили :.нее, средствами HTML невозможно добиться приемлемого разделения _ иайна и контента. Что же в **идеале** принесет это пресловутое разделение, к оторому всеми силами стремятся многие разработчики стандартов консор- _ [ума W3C, и которого с нетерпением ждут простые веб-мастера. *В идеале нтент должен создаваться один и только один раз.* Если мы помещаем ин- : 'рмацию в какой-либо файл или в базу данных, или еще куда-нибудь, то -л изменении дизайна сайта мы никоим образом не должны менять со- :.-ржимое этого файла или базы данных. Заметьте, фактически комбинация -за данных + *шаблоны* + *скрипты* и есть разделение контента от ттредставле- я- Информация хранится в базе данных, она четко структурирована и осится неизменной при смене дизайна, но придется полностью переделать ;блоны и внести некоторые **исправления** в скрипты. Так что долгожданэе и совершенно необходимое разделение контента и отображения уже реал кзовано, к примеру, на основе связки MySQL + PHP + HTML. Почему же л этом не остановиться?

гернемся к нашему иде&ту. Кроме неприкосновенности контента нам необ- •лима максимальная простота действий для изменения дизайна. Более то- -\ нам нужна максимальная простота для внесения небольших исправлений дизайн сайта. Очевидно, что связка MySQL + PHP + HTML нам желаемой "ростоты не обеспечивает. Внесение **изменений** — это достаточно трудоемкий процесс. Допустим, нам понадобилось все заголовки сделать не темно- юными, а черными, тогда придется искать и заменять в шаблонах все

теги <FONT>. В чем же причина этой сложности? Давайте разложим наш\ связку на составляющие.

П В базе данных хранится "чистая" информация, которая сама по себе структурирована, но эта структура никоим образом *не влияет на структуру документа.*

П Скрипты PHP обеспечивают взаимодействие информации из базы данных с HTML-шаблонами.

• HTML-шаблоны определяют *структуру документа и визусыьное представлениедокумента.*

Причина заключается в третьем пункте. Во-первых, с помощью HTML вс многих случаях невозможно создать хорошо структурированный документ, во-вторых, с помощью HTML нельзя добиться разделения структуры документа н визуального представления.

Итак, мы значительно конкретизировали первоначальную задачу. Нам надс иметь язык разметки, который обеспечивает структурированность документа, а также отделить *структуру документа* от *визуального представления.*

В конце *главы 2 я* перечислял, в каких случаях HTML хорошо структурируе" документы. Это электронные публикации, не более и не менее. Что касаетс электронных магазинов, online-каталогов, научных баз данных, самых разнообразных реестров и списков, то их с помощью HTML хорошо структурировать совершенно невозможно. Зато это можно сделать с помощы XML. Итак, первая проблема решается следующим образом.

П Если мы создаем online-газету, домашнюю страничку пли любой друго; сайт, на котором будут в том или ином виде размещаться электронны-, публикации, то для структурирования документов будем использоват! HTML,

• Если же мы создаем электронный магазин, каталог, научную базу данных, то для структурирования документов будем использовать XML.

Нам осталось отделить структуру документа от визуального предстаааения И с недавних пор это стало возможным благодаря каскадным таблицам стилей. Очень часто в среде веб-разработчиков можно услышать мнение, что технология CSS обеспечивает разделение контента и дизайна. Однако, как *въ* могли убедиться, это не совсем верно. На самом деле *CSS обеспечивают всег. лишь отделение структуры документа от его визуального представления,* и н; более того. Так что контент в чистом виде здесь совершенно ни при чем.

Давайте вернемся к нашему примеру и попытаемся структурировать документ с помощью HTML, а его визуатьнос представление реализовать исключительно с помощью каскадных таблиц стилей. Вот HTML-код:

```
<DIV ID="brd">
  <DIV ID="title">
```

```
    <H2>BeO-дизайн</H2><BR>
    <H3>Д. Кирсанов</H3><ВИ>
    <SPAN CLASS="prise">104 руб.</SPAN>
  </DIV>
  <DIV ID="image">
    <IMG SRC="cover.jpg" WIDTH="100" HEIGHT="132" BORDER="0" ALT="Веб-
дизайн. Книга Дмитрия Кирсанова">
  </DIV>
  <DIV ID ="description">
    <C1TE>Книга написана замечательно. В каждой главе видна тщательная
работа с материалом и столь же тщательный подбор слов и метафор. Нигде
Дмитрий не позволяет себе обойтись штампом, общими словами и описанием
того, в чем сам не разобрался.</CITE><BR>
    </DIV>
</DIV>
```

Сравните новый вариант с предыдущим и отметьте, насколько он проще воспринимается. Здесь нет нагромождения таблиц и прочих тегов, которые отвечали за представление документа в браузере, а осталась только логическая структура. Все, что связано с визуализацией, делается с помощью CSS. Мы пока не знакомы ни с синтаксисом, ни со свойствами CSS, но короткие комментарии я дам, чтобы вы привыкали и затем легче воспринимали материал книги. Итак, таблица стилей будет такой:

```
<STYLE TYPE="text/css">
/* Задаем шрифт Arial размером 10 пикселов для описания книги */
CITE {
  font: 10px Arial, sans-serif}

/* Задаем полужирный шрифт Arial размером 16 пикселов для названия книги,
во второй строке задаем объявление display: inline, чтобы после назва-
ния не ставился перевод строки */
H2 {
  font: bold 16px Arial, sans-serif;
  display: inline}

/* Задаем наклонный шрифт Arial размером 10 пикселов для фамилии автора
книги;:, а во второй строке задаем объявление display: inline, чтобы после
автора не ставился перевод строки */
H3 {
  font: italic 10px Arial, sans-serif;
  display: inline}
```

```
I* Задаем красный цвет для вывода цены и полужирный шрифт Arial размеров
12 пикселов */
.prise {
  color: #F00;
  font: bold 12px Arial, sans-serif)

/* Задаем ширину общего блока в 500 пикселов, а затем указываем, что Ст:—
должен иметь рамку толщиной 1 пиксел, выведенную сплошной линией и чернь:
цветом */
#brd |
  width: 500px;
  border: lpx solid #000}

/* Блок для описания книги. Он смещается на 10% от верхнего края общего
блока и имеет отступ со всех сторон в 10 пикселов */
#description {
  position: relative;
  top: 10%;
  float: left;
  padding: 1Opx)

/* Слева и справа от изображения обложки выводится черная рамка сплошной
линией толщиной 1 пиксел. Причем рамка выводится на расстоянии 10 пиксе-
лов от картинки, т. е. за отступами */
#image {
  float: left;
  border-left: lpx solid #000;
  border-right: lpx solid #000;
  padding: lQpx}

/* Блок для основной информации (названия, автора, цены). Текст в нем
выровнен по правому краю, ширина блока — 130 пикселов, а отступы — 10
пикселов */
#title {
  text-align: right;
  float: left;
  width: 130px;
  padding: lOpxJ
</STYLE>
```

В итоге в браузере мы получим совершенно аналогичный вид анонса книп (рис. 3.2). Кроме того, если нам потребуется слегка изменить шрифт ил;

вет *всех* заголовков, то изменения надо будет внести только *один раз* в таблице стилей для элемента <H2>. Если нам потребуется немного изменить положение элементов на странице, то этого можно добиться изменением позиционирования блоков в таблице стилей.

Здесь надо отметить, что ни связка HTML+CSS, ни связка XML + CSS не обеспечивают полного разделения структуры документа от визуального отображения. Если нам потребуется коренным образом изменить вид страницы в браузере, то все равно придется перестраивать HTML-код, изменяя сложенность блоков, добавляя новые блоки. Но это намного *легче и быстрее,* чем полностью переписывать все таблицы,

возможно, полное разделение обеспечит связка XML + XSL, однако пока этот вопрос остается открытым. Если же говорить о каскадных таблицах стилей, то можно отметить, что они улучшают восприятие кода и существенно облегчают внесение новых элементов дизайна; в некоторой степени обеспечивают разделение структуры документа от визуального представления, но не более того.

Контент и дизайн — это не те вещи, которые можно разделить на HTML и CSS. Так что не стоит особо прислушиваться к радужным высказываниям по поводу CSS. Но не стоит и приуменьшать значение каскадных таблиц стилей, потому что их использование дает следующие преимущества;

- в большинстве случаев уменьшается объем кода;
- появляется возможность контролировать вывод страниц на принтер;
- увеличивается логичность кода;
- код становится более гибким, а изменять его становится проще;
- значительно увеличивается контроль над визуальным представлением документа.

Сообщая, можно сказать, что CSS предоставляет HTML-верстальщикам *новые возможности и контроль.* Согласитесь, не так уж мало для того, чтобы непременно взяться за изучение еще одного языка помимо HTML. По крайней мере, для профессиональной верстки каскадные таблицы стилей совершенно необходимы. Итак, вперед!

Первые попытки реализации таблиц стилей

Хотя чего вообще создавалась технология CSS? Ответ на этот вопрос достаточно прост. В начале 90-х годов язык HTML обогатился новым тегом, который стал причиной многочисленных споров. Это был тег <FONT>. Тег <FONT> не имел вообще никакого отношения к структуре документа, а отвечал исключительно за визуальное отображение текста в браузере. Это кричащим образом противоречило самой концепции HTML, как языка *раз-*

метки гипертекста. Было ясно, что если ничего не предпринять, то разработчики браузеров по собственной инициативе добавят еще несколько тегов, которые тоже будут служить исключительно для визуального отображения информации, потому что назрела необходимость в значительном улучшении представления документа. Веб-дизайнерам хотелось иметь больший контроль над **визуальным** отображением, иметь больше возможностей для работы с текстом, позиционированием, цветами. Средства HTML крайне ограничены, так что оставалось либо добавлять новые теги (которых все равно будет мало), либо создавать совершенно новую технологию, которая будет легко интегрироваться с языком HTML, и возложить всю ответственность за контроль над отображением документа на нее.

Можно сказать, что технология CSS была разработана именно для решения этой проблемы. На самом деле, таблицы стилей — не такая уж и новая идея. Разделение структуры документа и визуального отображения — это основная цель языка HTML. В 1992 году браузер Viola поддерживал простеньки. таблицы стилей, в 1993 похожие по синтаксису таблицы стилей поддерживал браузер Harmony. Однако, как это ни странно, новые браузеры сокращали возможности пользователей влиять на представление документа. В 1993 году вышел знаменитый браузер Mosaic, во многом благодаря которому Интернет стал популярным, и он разрешал пользователям только менять размер шрифта и некоторые цвета.

Предложение о каскадных таблицах стилей впервые было выдвинуто в 1994 году на конференции по вопросам Всемирной Сети. В данном предложении говорилось, в частности, что читатель документа должен иметь больший контроль над стилями, чем создатель документа. Синтаксис был похож на синтаксис объектно-ориентированных языков программирования. Например, шрифт Helvetica для элемента <н:> устанавливался таким объявлением:

```
hi.font.family = helvetica
```

а выравнивание по центру:

```
hi.align.style = center
```

Можно было устанавливать размер и гарнитуру шрифта, цвет фона и элементов, задавать выравнивание и отступы. В общем, простейшие свойств;: иизуалытого форматирования и не более того. Но для создания нормальных стилевых таблиц требовался более сложный синтаксис и более развитая концепция, по этой причине данное предложение решили доработать.

В 1995 году начал активно работать консорциум W3C и разработка стандарта CSS ускорилась. О будущей поддержке каскадных таблиц стилей заявили разработчики браузера Microsoft Internet Explorer. В 1996 году был утвержде): стандарт CSS-1, и в том же году вышел первый браузер, который частично поддерживал стандарт CSS-1. Это был Microsoft Internet Explorer 3.0.

С тех пор и началась пресловутая история поддержки CSS различными браузерами. Веб-мастерам приходилось учитывать различия HTML для Netscape и Explorer, писать кросс-браузерные скрипты на JavaScript, а сейчас приходится писать еще и кросс-браузерные таблицы стилей.

Нет ничего интереснее в истории CSS, чем хронология реализации поддержки этого стандарта в браузерах. Мы рассмотрим эволюцию браузеров трех производителей: Microsoft, Netscape и Opera Software, которые являются самыми распространенными и популярными за последние несколько лет.

Microsoft Internet Explorer

Первым браузером, который в некоторой степени поддерживал каскадные таблицы стилей, был Microsoft Internet Explorer 3.0, вышедший в 1996 году. Тогда еще спецификация CSS-1 не была стандартом, однако компания Microsoft все же решила включить поддержку CSS, будучи уверенной в дальнейшем успехе этой технологии. Как мы теперь знаем, они не ошиблись. Браузер Internet Explorer 3.0 поддерживал следующие свойства:

1. внешние и внутренние таблицы стилей;
2. четыре вида селекторов:
   - *простые селекторы элемента* (H2 {color: #000});
   - /i)(#soroeid {color: #000});
   - *классы* (.someclass {color: #000});
   - *контекстные селекторы* (нз EM {color: #000});
3. свойство font для изменения размера, насыщенности, начертания и стиля шрифта;
4. выравнивание и подчеркивание текста;
5. вставку фоновых рисунков и манипуляции с ними, например, запрещение повторения фоновых изображений;
6. регулировку высоты межстрочных интервалов;
7. установку левых, правых и верхних полей (но не отступов).

Как видите, для первой реализации поддержки CSS сделано достаточно много. Надо сказать, что очень многие веб-мастера по сей день довольствуются практически таким же набором свойств каскадных таблиц стилей. Во время чтения этой книги попробуйте походить по сайтам и посмотреть используемые там таблицы стилей. Примерно у 70% сайтов вы не найдете **ничего,** что выходило бы за рамки данного списка. И это очень печально, потому что на дворе 2002 год, т. е. прошло уже шесть лет, а ситуация изменилась слабо.

Надо сказать, что поддержка CSS в третьей версии браузера была реализована с достаточным количеством багов (порядка 190), так что применят: каскадные таблицы стилей в то время надо было очень осторожно. Но зар_анее отмечу, что по количеству багов вне конкуренции браузер Netscap Navigator 4.x Однако вернемся к фирме Microsoft.

В 1997 году вышел браузер Internet Explorer 4.0. Вот список нововведений:

О появилась возможность использовать несколько таблиц стилей на странице, а также подключать стили с помощью инструкции eimport;

- появился псевдокласс :hover, который позволяет менять свойства ссылок при наведении мышкой, например, менять цвет;
- появилось свойство text-align: justify, которое выравнивает текст пс обоим краям;
- появилась возможность изменять межбуквенное расстояние с помощью свойства letter-spacing;
- появилась возможность контролировать визуальное представление маркеров списков, например, внедрять изображение вместо цифр и кружков
- внедрены так называемая блоковая модель и позиционирование элементов на странице. С помощью JavaScript стало возможным делать выпадающие меню, создавать невидимые блоки и показывать их при наведении курсора, перемешать слои по экрану и многое другое. Фактически это обусловило зарождение DHTML;

П появилась возможность устанавливать рамки вокруг блоков, так что исчезла необходимость пользоваться итоженными таблицами для получения однопикселных рамок;

- реализовано управление некоторыми свойствами документа при распечатке;

О появилась возможность изменять вид курсора и применять к объектам визуальные фильтры, например, тень.

Как видите, поддержка CSS стала гораздо лучше, но количество багов не уменьшилось, их осталось порядка 200. Главное достижение четвертой версии браузера — это появление блоковой модели. Кстати, сама блоковая модель была реатизована не так, как в рекомендации консорциума W3C, да і ошибок в ней хватало, но все же это был большой шаг вперед. У пользователей появилась реальная возможность достаточно широко управлять визуальным представлением информации на странице. И самым известным применением блоковой модели и позиционирования является выпадающее меню. Оно реализуется следующим образом. Навигационные панели помешаются в отдельные блоки, которые представляют собой что-то вроде слоев Эти блоки при загрузке страницы не отображаются на экране. Когда пользователь подводит курсор к ссылке на раздел или нажимает на ссылку, с

пюшью JavaScript соответствующий невидимый слой меняет свое свойство становится видимым. Таким образом, создается впечатление вложенности. Подобный элемент интерфейса реализован, например, в ОС Windows: при нажатии на кнопку **Пуск** разворачивается меню.

В 1998 году вышла пятая версия популярного браузера, которая **стала** единоличным лидером. На момент написания этой книги браузером Internet **Explorer** 5.x пользовалось около 75% интернет-аудитории. Что же нового **принес** с собой пятый Explorer:

- появились псевдоэлементы first-letter: и first-line:, с помощью которых можно создавать буквицы и применять стили к первой строке текста;
- усилился контроль над фоном;
- появилась возможность раскрашивать полосу прокрутки браузера;
- усилился контроль над таблицами, в частности, таблицы с фиксированной шириной могли обсчитываться и отображаться браузером быстрее;
- незначительно улучшилась блоковая модель;
- появилась возможность увеличивать объекты;
- появились богатые возможности контролировать манеру письма данной страны. Например, письмо справа налево.

Как видите, нововведений стало меньше, и они не имеют определяющего значения. Они лишь привносят тонкости, которые позволяют улучшить облешний вид документа. Что касается самой реализации поддержки CSS, то она, без всякого сомнения, была лучшей на то время (около 150 багов, большинство из которых не влияли критическим образом на вид страницы в браузере, по крайней мере, существовали методы устранения этих багов). Но поддержка стандарта CSS-1 все же была неполной (правда, отличия были незначительными), а блоковая модель так и осталась с ошибкой. Несмотря на это, для браузера Internet Explorer *5.x* уже можно было в достаточной степени отделить структуру документа от его визуального отображения. Но этому очень сильно мешал браузер Netscape Navigator 4.x, который имел такое количество ошибок, что написать сложную таблицу стилей под этот браузер было совершенно невозможно.

И, наконец, в 2001 году вышел долгожданный Internet Explorer 6.0. Надо сказать, что, вопреки ожиданиям, ничего особо нового он не привнес. Самое главное — это полная поддержка стандарта CSS-1, а также исправление блоковой модели:

- добавлена поддержка двух новых свойств min-height и word-spacing, а также нескольких новых значений уже существующих свойств;
- увеличен контроль над таблицами;

- исправлены некоторые мелкие несоответствия для полной совместимости со стандартом CSS-1;
- в некоторой степени улучшено позиционирование элементов.

Вот, в общем-то, и все. Как видите, список улучшений и добавлений *m* сравнению с четвертой и пятой версиями значительно сократился, однакг качество самого браузера заметно улучшилось. На **мой** взгляд, Interne Explorer 6.x на сегодняшний день лучший браузер, хотя Netscape 6.2 тоже достойный конкурент. Настораживает потеря темпа в поддержке стандарта CSS. возможно, компания Microsoft считает, что будущее за связкой XML + XSL однако время покажет.

А теперь давайте рассмотрим хронологию развития браузеров главного, в недавнем прошлом, конкурента фирмы Microsoft на этом поприще — браузеров фирмы Netscape.

Netscape Navigator

Изначально разработчики браузера Netscape Navigator очень настороженно относились к технологии CSS и в третью версию своего браузера поддержку каскадных таблиц стилей не включили. Однако почти одновременно с браузеров Netscape Navigator 3.x вышел браузер Internet Explorer 3.x, который поддержива: некоторые свойства CSS. Компания Netscape не хотела отставать, у нее простс не было на это права, потому что к этому времени именно между этими компаниями разгорелась знаменитая "война браузеров". Так что вышедший в начато 1997 года браузер Netscape Navigator 4.x обеспечивал поддержку очень многих свойств CSS, но количество багов в реализации поддержки превышало все разумные пределы, их было порядка 500. Видимо, разработчики очень спешили, г времени на тестирование критически не **хватало.**

Главные отличия в реализации поддержки CSS от браузера Internet Explorer 4.x:

- не поддерживается псевдокласс : hover, так что невозможно изменять параметры ссылки при наведении курсора;
- невозможно подключать внешние таблицы стилей с помощью инструкции ^import;
- не поддерживаются свойства background-attachment И background-position, что ухудшает управление фоном;
- не поддерживаются свойства list-style-position **и list-atyle-image,** что сокращает контроль над списками:
- не поддерживается свойство height;
- поддерживаются значения display: block и display: list-item, тогда как в Explorer 4.x этого нет;
- менее развитая модель позиционирования;

- контроль над рамками значительно хуже;
- не поддерживается динамическое изменение вида курсора и фильтры;
- дополнительно поддерживается вставка какого-либо документа в текущий документ;
- нет совершенно никакого контроля над печатью страниц.

Как видите, почти во всем браузер Netscape Navigator 4.x уступал браузеру Internet Explorer 4.x. Именно начиная с четвертых версий, все больше и больше пользователей предпочитало браузер компании Microsoft. Появление Internet Explorer 5.x окончательно решило дело в пользу Microsoft, потому что Netscape затягивала с выходом новой версии браузера. Появилась она слишком поздно, и произошло это в 2000 году. Да, по сравнению с четвертой версией новая шестая очень хороша, но все же по ряду причин этой версии не удалось хоть сколько-нибудь поколебать позиции браузера № 1. Что касается каскадных таблиц стилей, то обеспечена полная поддержка стандарта CSS-1, без багов, конечно, не обошлось, пока насчитали порядка 100), но это не конечный результат.

Резюмируя, можно сказать, что браузер Netscape 6.x в достаточной мере удовлетворяет стандартам, чтобы особо не задумываться о различиях при написании каскадных таблиц стилей, однако им пользуется около одного процента всей аудитории, так что пока нет особого смысла обращать на него внимание.

Opera

Браузеры компании Opera Software всегда славились своей быстротой, малым размером и хорошей поддержкой веб-стандартов. Однако относительную популярность имеют только последние пятая и шестая версии браузера. Стандарт CSS начал поддерживаться браузером, начиная с версии 3.5, которая вышла в 1998 году, и реализация поддержки была очень хорошей. Отличительными ее особенностями на то время были:

- поддержка псевдоклассов, таких как first-line и first-letter;
- поддержка свойства word-spacing;
- очень хорошая реатизация контроля над рамками.

В то же время блоковая модель была не развита, а позиционировать элементы вообще было невозможно. Количество багов тоже было приличным, около 180.

В 2000 году вышла четвертая версия браузера, которая тоже не прижилась. Вот список наиболее значимых изменений:

- улучшился контроль над фоном;
- улучшилась блоковая модель и позиционирование;

П декларирована полная поддержка стандарта CSS-l;

О декларирована достаточно приличная поддержка стандарта CSS-2;

П введена поддержка связки XML + CSS.

Надо сказать, что четвертая версия отличалась очень хорошей поддержкою CSS и насчитывала только порядка 130 багов.

Вместо выпуска версии 4.03, разработчики почему-то решили присвоите своему браузеру версию 5.0. Надо сказать, что нововведений с точки зрения поддержки CSS было крайне мало, разве что устранили многие баги. В ноябре 2001 года вышел браузер Opera 6.x и он вообще не содержат ничей, нового в поддержке CSS. что крайне печально. Посудите сами, начиная с хорошо проработанной четвертой версии дальнейшая поддержка CSS развивалась очень слабо.

Что касается CSS-2, то браузеры Opera *6.x* и Netscape б.д: являются, несомненно, лучшими. Однако их слабое распространение не позволяет пользоваться многими преимуществами этого стандарта.

Итак, на сегодняшний день существует несколько браузеров, которые приемлемо поддерживают каскадные таблицы стилей:

О Microsoft Internet Explorer версий *5.x*—6.x

П Netscape *6.x;*

О Opera версий 4.x — 6.x

• Mozilla 0.9.x

Что касается остальных браузеров, то достаточно большую нишу в сети занимают браузеры четвертого поколения, это Netscape Navigator *4.x* и Internet Explorer 4.x в которых поддержка CSS недостаточная, а в случае с Netscape *4.x* еще и очень плохо реализованная. Полновесное применение CSS при верстке будет возможно только тогда, когда эти старые версии браузеров будут установлены у 1—2% пользователей, а не у 6—10%. Я надеюсь, что четвертое поколение браузеров сойдет со сиены где-то в середине 2002 года, по крайней мере, это случится в ближайшем будущем, так что каскадные таблицы стилей постепенно получат широкое распространение.

Прежде чем приступить непосредственно к изучению каскадных таблиц стилей, давайте кратко рассмотрим уже воплощенный стандарт CSS-1, а также заглянем в будущее и посмотрим, что нового принесет веб-разработчикам внедрение стандартов CSS-2 и CSS-3. Это позволит нам представить общую картину технологии CSS: понять, что можно сделать на ее основе, а чего нельзя, в каких случаях надо использовать каскадные таблицы стилей, а в каких не стоит. Кроме того, мы сможем оценить перспективность этой технологии и окончательно дать ответ на вопросы типа: "Для чего нужен стандарт CSS?", "Зачем CSS именно мне?", "Стоит ли мне изучать эту технологию?"

Стандарт CSS-1

Я уже достаточно долго говорю о каскадных таблицах стилей, однако до сих пор не дал определения этому понятию. Итак:

> ***Определение***
>
> Каскадные таблицы стилей — это набор правил или свойств, которые описывают, как тот или иной элемент или группа элементов будут отображаться на экране монитора в браузере.

С помощью первого уровня таблиц стилей можно выполнять следующее форматирование.

- Осуществлять выбор элемента или группы элементов, к которым будет применен тот или иной стиль. Это делается с помощью так называемых селекторов. Они в спецификации CSS-1 позволяют применять стили:
  - ко всем элементам данной группы, например, ко всем элементам <H1>;
  - к элементам с определенным классом или ID, которые указываются непосредственно в коде страницы, например, к элементам <H1 class="topHeader">;
  - к элементам, содержащимся внутри другого элемента, например, к элементу <EM> в таком контексте <h1>Заголовок с <EM>выделенным</EM> словом</H1>;
  - к первой букве элемента, например <p>, что позволяет создавать буквицы;
  - к первой строке элемента;
  - к посещенным, непосещенным и активным ссылкам.
- Осуществлять создание блоков; устанавливать размеры самих блоков, отступов, полей и рамок; устанавливать тип и цвет рамок.
- Осуществлять некоторое позиционирование блоков:
  - выравнивание по горизонтали;
  - обтекание блоков текстом.
- Управлять шрифтом на странице:
  - устанавливать начертание;
  - устанавливать насыщенность;
  - устанавливать размер и стиль шрифта.
- Управлять фоном на странице:
  - устанавливать цвет фона или фоновое изображение блока;
  - управлять повторением фонового изображения;
  - позиционировать фоновое изображение.

О Писать стили для текста:

- устанавливать расстояние между словами и буквами;
- устанавливать атрибуты оформления, например подчеркивание и перечеркивание текста;
- выравнивать текст по горизонтали и **вертикали;**
- устанавливать высоту строки:
- устанавливать цвет текста.

• Управлять списками:

- устанавливать тип маркера в начале каждого пункта списка;
- использовать в качестве маркера графические изображения;
- позиционировать маркеры.

Вот, собственно, и все, что можно сделать с помощью каскадных табли_ стилей первого уровня. (Сак вы уже знаете, стандарт CSS-1 был рекомендован в далеком 1996 году, а полная его поддержка в большинстве браузеров была реализована только в 2000—2001 годах, но кроме CSS-1 последние версии браузеров частично поддерживают и стандарт CSS-2. Давайте посмотрим, что нового принесет разработчикам внедрение этого стандарта.

Стандарт CSS-2

Этот стандарт базируется на CSS-1, но его функциональность значительна повысилась. Вот основные нововведения в каскадных таблицах стилей второго уровня.

3 Появилась возможность писать таблицы стилей для различных типоь устройств, таких как голосовые браузеры, тактильные браузеры для слепых на основе азбуки Брайля, проекторы, принтеры, телевизионные устройства, PDA.

О Добавилось множество селекторов, с помощью которых можно достаточно точно выбрать элементы, к которым применяется данный стиль. Например, можно применять стили:

- к элементу при наведении курсора и при получении элементом фокуса;
- к элементу, который является прямым потомком;
- к группе элементов, которые имеют одинаковые **атрибуты,** а также к группе элементов, которые имеют одинаковые значения атрибутов;
- к элементу, который следует непосредственно за родительским элементом;
- ко всем элементам сразу, это стало возможным благодаря появлению универсального селектора.

1 Значительно улучшилась блоковая модель. Появилась возможность точно позиционировать блоки на странице, делать их невидимыми, и многое другое.

1 Увеличился контроль за шрифтами на странице. Появилась возможность скачивать недостающие шрифты с удаленного сервера, что значительно расширяет перечень возможных для использования шрифтов.

3 Появилась возможность изменять вид курсора и внедрять графические изображения вместо стандартных курсоров операционной системы.

3 Появилась возможность изменять контуры элемента, получающего фокус.

~\ Внедрена поддержка различных направлений письма, например, справа налево.

Как видите, стандарт CSS-2 значительно отличается от CSS-1. Добавлено множество новых свойств и механизмов, что выводит стандарт CSS-2 на совершенно новый уровень функциональности. Этот стандарт в значительной мере позволяет отделить структуру документа от визуального и любого другого представления, чего не в состоянии обеспечить стандарт CSS-I.

Стандарт CSS-3

"ейчас в разработке находится совершенно новая спецификация каскадных :тилей — CSS-3. Принципиальные отличия и нововведения.

3 Введена модуляризация CSS, т. е. каскадные таблицы стилей могут быть разбиты на отдельные модули. Это делается для упрощения внедрения стандарта CSS-3, т. к. можно постепенно вводить поддержку отдельных модулей. Кроме того, обеспечивается дифференциация поддержки CSS. К примеру, голосовым браузерам ни к чему визуальная модель форматирования, а визуальным браузерам нет смысла поддерживать голосовые стили.

3 Добавлено очень много новых селекторов для наиболее точного выбора элементов, к которым применяется данный стиль. Например, можно применять стили:

- к элементам, значение атрибута в которых начинается или заканчивается с заданной текстовой строки, а также содержит заданную текстовую строку;
- к элементам, которые в данный момент выделены пользователем (например, курсором мыши);
- к самым разным потомкам и родительским элементам (в этом отношении система селекторов сильно развилась по сравнению со стандартом CSS-2);

- к элементам, которые являются якорями, т. е. можно определенным образом выделять якоря (это позволяет видеть, что ссылка ведет на элемент, расположенный на этой же странице, а не на **другую** страницу);
- к элементам пользовательского интерфейса, которые являются активными или неактивными;
- к элементам, которые содержат заданную строку в своем содержим (это будет позволять легко подсвечивать абзацы, в которых найден искомое слово при поиске).

О Значительно улучшилась блоковая модель.

П Появилась возможность объявлять любой элемент на странице ссылке; без помощи тега <A> И управлять поведением ссылок.

П Несколько увеличился контроль над фоновыми изображениями.

- Появилась возможность применять к шрифту различные эффекты, . также сглаживание, чего в настоящее время сильно недостает в веб.
- Значительно улучшился контроль над текстом, появились богатейши-. возможности для его **интернационализации.**
- Значительно расширились возможности контроля нал элементами пользовательского интерфейса.

О Появилась возможность верстки в несколько колонок без **применении** таблиц.

Как видите, отличий очень много и CSS-3 обеспечивает впечатляющий контроль над документом. К тому же CSS-3 очень сильно отличается от CSS-Z так что можно было бы констатировать достаточно динамичное развитие технологии каскадных таблиц стилей, если бы не задержки с ее внедрение[1]- производителями браузеров.

Надо сказать, что CSS-3 — дело недалекого будущего. Консорциумом W3C стандарт CSS-3 будет рекомендован где-то в 2003 году, а в браузерах реализация его поддержки начнется не ранее 2004.

Теперь вы имеете общее представление, что можно сделать с помощью каскадных таблиц стилей. На мой взгляд, уже сейчас стоит знать не толькс CSS-1, но и CSS-2, потому что поддержка этого стандарта уже частично реализована в новейших браузерах, а полная поддержка — лело ближайшеп будущего. Чтобы убедить вас окончательно, я в очередной раз приведу список того, то вам даст изучение CSS:

О в некоторой мере вы сможете достичь отделения структуры документа о~ его визуального представления, что делает код более гибким и логичным а изменять его становится проще;

- во многих случаях вам удастся значительно уменьшить время загрузкг документа, потому что CSS-файл можно присоединять к документу, ваи

браузер кэширует его и при следующих загрузках страницы будет брать CSS-файл из кэша; кроме того, уменьшается общий объем кода вследствие отсутствия таблиц и невидимых однопикселных GIF-рисунков, что также уменьшает время загрузки;

- благодаря логичности и уменьшению объема кода вы будете затрачивать меньше времени на разработку сайта;
- вы получите гораздо больший контроль над страницей, потому что в CSS реализовано значительно больше возможностей визуального форматирования, чем в HTML, тем более что стандарт CSS постоянно развивается, а в новых версиях браузеров практически всегда появляется поддержка новых свойств CSS.

Впрочем, о хорошем говорят часто и много, так что давайте рассмотрим причины, по которым не стоит использовать CSS:

- вам придется изучать еще один синтаксис и еше одну технологию, так что скорее всего на первых порах вы будете несколько путаться в коде и допускать глупые ошибки. Впрочем, все через это прошли;
- поддержка CSS в браузерах реализована по-разному, это несет дополнительные трудности, потому что в некоторых случаях придется минимизировать использование CSS или же писать кросс-браузерный код.

Это все. Больше нет ни одной объективной причины для того, чтобы закрыть эту книгу и забыть о существовании каскадных таблиц стилей. Если этих двух причин для вас достаточно, то приношу свои извинения, потому что вы зря купили книгу. Если же для вас этих причин недостаточно, давайте непосредственно приступим к изучению CSS. Кстати, деньги во втором случае вы потратили совсем не зря.

Глава 4

Основы CSS

В предыдущих главах мы достаточно подробно рассмотрели проблемы верстки, и теперь вы должны представлять себе, какие из них можно решить помощью каскадных таблиц **стилей.** Я намеренно не вдавался в подробное синтаксиса и не затрагивал прочих утилитарных вещей, но постарался **всесторонне** описать общую картину современных средств верстки. Надеюс вы теперь знаете, какая роль в них отведена CSS и какое будущее ждет э технологию, если, конечно, разработчики браузеров будут аккуратно прс должать ее внедрение.

Самое время перейти к непосредственному изучению технологии CSS. Дл наглядности возьмем несложную HTML-страницу **и** будем постепенно добавлять к ней различные свойства CSS. На примерах всегда учиться прош В *главе 2* мы верстали пробный сайт без использования CSS, именно его возьмем в качестве учебного пособия. В конце каждого подраздела мы будс возвращаться к нему и вносить необходимые изменения, так что постепенна код будет становиться более логичным, а структура документа все больш отделяться от визуального представления.

Включение CSS в HTML

Итак, что такое каскадные таблицы стилей — вы уже знаете, поэтому начнем того, каким образом они применяются. Допустим, у вас есть HTML-страничк к некоторым элементам надо применить стили. Код странички такой:

```
<HTML>
  <HEAD>
    <T1TЬE>Домашняя страница 0ксаны</T1TЬE>
  </HEAD>
  <BODY>
    <H1>Домашняя страница 0ксаны</H1>
    <P>Зовут меня Оксана. Мне <EM>18 лет</EM>. Я живу в городе Житомире,
но скоро перееду в Москву, потому что там мне 0больше нравится. Вот тольк
насобираю побольше денег. У меня есть любимая собака <B>Шарик</B>. Скорс
я сделаю страничку в Интернете и для нее, чтобы вы смогли увидеть ее фо
тографию.</P>
    </BODY>
</HTML>
```

Допустим, вы хотите сделать заголовок <H1> красным. Это можно реализовать как с помощью CSS, так и с помощью HTML. Средствами HTML это делается так:

```
<H1><FONT COLOR="#ET0000">Домашняя страница OKcaHbi</FONT></Hl>
```

Если использовать CSS, то применить стиль к элементу <H1> можно несколькими способами. Например, вписать в секцию HEAD такой код:

```
<STYLE type="text/css">
  H1 {
    color: #F00J
•/STYLE>
```

Таблицы стилей, включенные в страницу таким способом, называются *встроенными*. В данном случае текст во *всех* элементах <H1> на *этой* странице будет красного цвета.

Кроме встроенных есть еще и *внутренние* стили. Они вставляются непосредственно в нужный элемент с помощью атрибута STYLE (заметьте, что в HTML есть как элемент <STYLE>, так и атрибут STYLE):

```
<Hl STYLE="color: #Г00">Домапшяя страница 0ксаны</H1>
```

Текст будет красным только в *данном* элементе <H1> и нигде больше. Надо сказать, что внутренние стили имеют несколько принципиальных недостатков:

3 применяя данный стиль к какому-либо элементу, вы не используете селекторов, поэтому впоследствии можете не вспомнить, почему именно этот элемент <H1> имеет красный цвет, а другой элемент <H1> — **черный;**

П если вы **использовали** внутренние стили **и** при этом захотите поменять цвет заголовков на всех страницах сайта, то вам придется менять их вручную, как вы и поступали при изменении HTML-кода, т. е. исчезает одно из главных преимуществ CSS: разделение структуры документа и его визуального представления.

Я рекомендую не использовать внутренние стили, по крайней мере, свести их использование к минимуму.

Кроме того, таблицу стилей можно поместить в отдельный файл **и** подключать его с помощью элемента <LINK>. Такие таблицы стилей называются *внешними*. Допустим, мы поместили наш стиль, примененный к элементу <H1>, в файл style.ess, который размещен в том же каталоге, что и HTML-файл. Тогда к HTML-документу его можно подключить следующим образом:

```
•LINK REL="stylesheet" TYPE="text/css" HREF="style.css">
```

Тег <LINK> обязательно должен находиться в секции HEAD. Внешние таблицы стилей обладают очевидным и очень важным преимуществом: их можно

подключать к любому количеству HTML-документов. Поэтому для все сайта может быть всего одна внешняя таблица стилей, которая находится файле с расширением ess. В этот файл можно поместить все то, что встроенных таблицах стилей расположено между тегами <STYLE></STYL: т. е. в нашем случае файл style.ess содержит всего одну строку:

```
HI {
  color: #FO0}
```

Давайте подробнее рассмотрим атрибуты тега <LINK>.

- REL

  Описывает отношение внешней таблицы стилей и документа. Мож иметь значения stylesheet и alternate stylesheet. Если задано пере, из них, то внешняя таблица стилей используется браузером для формат рования документа по умолчанию, если же указано второе значение, эту таблицу стилей для форматирования документа может выбрать по.:, зователь. Альтернативные таблицы стилей поддерживают браузер Mozilla, Netscape 6.x и Opera 5+, в них реализована возможность пер ключения используемой таблицы стилей. Что касается браузеров фир.\ Microsoft, то они такой возможности не предоставляют.

- TYPE

  Используется для указания языка стилей. Кроме CSS существует е: XSL и некоторые другие языки, которые на данный момент практичен не используются. Для XSL атрибут TYPE должен иметь значен; text/xsl.

- HREF

  В этом атрибуте указывается путь к внешнему CSS-файлу. Путь мож: быть как относительным, так и абсолютным.

Все вышеприведенные атрибуты являются обязательными, но сущесть> еще один необязательный атрибут — MEDIA. ОН определяет тип устройст: для которого нужно применять данную таблицу стилей. Перечень возмо> ных значений этого атрибута представлен в табл. 4.1.

Таблица 4.1. Значения атрибута MEDI

Значение	Применение таблиц стилей
all	Для любых типов устройств
aural	Для устройств с голосовым выводом
braille	Для устройств с тактильным выводом с помощью азбуки Брайля
handheld	Для переносных устройств с маленьким экраном (например, Palm)
print	Для вывода страницы на печать

Таблица 4.1 (окончание)

Значение	Применение таблиц стилей
projection	Для вывода страницы на устройства типа проектора
screen	Для вывода страницы на цветной экран
tv	Для вывода страницы на интернет-устройства на основе телевизора (например, WebTV)

Атрибут MEDIA по умолчанию имеет значение all, так что применять его надо только в том случае, если вы используете данную таблицу стилей лля определенного типа устройств. Кроме того, некоторые браузеры (например, Netscape Navigator *4.x)* игнорируют таблицы стилей, которые подключены с использованием атрибута MEDIA, И ЭТО СВОЙСТВО МОЖНО использовать для сокрытия данной таблицы стилей от определенного браузера.

Есть еше один способ подключения внешних таблиц стилей с помощью инструкции @import:

```
<STYLE TYPE="text/css">
  HI {
    color: red)
  @import url(style.ess)
</STYLE>
```

В этом случае происходит интегрирование уже существующих стилей и стилей, находящихся в файле style.ess. Браузер Netscape Navigator 4.x не поддерживает данную инструкцию, поэтому ее можно использовать для простейшего кросс-браузерного кода. Делается это так: вы подключаете с помощью элемента <LINK> таблицу стилей, которая предназначается для браузера Netscape Navigator 4.x, а после этого подключаете внутри элемента <STYLE> с помощью инструкции @import таблицу стилей для всех остальных браузеров. Вся конструкция будет выглядеть следующим образом:

```
<HEAD>
  <LINK REL="stylesheet" TYPE="text/css" HREF="nn4,css">
  <STYLE TYPE="text/css">
    @import url(standart.ess)
  </STYLE>
</HEAD>
```

Тогда Netscape Navigator 4.x применит только те стили, которые находятся в файле nn4.css, подключаемом с помощью элемента <LINK>. Все остальные более современные браузеры сначала применят стили из этого же файла, но

потом переобозначат их стилями из файла, подключаемого с помощью инструкции @import.

Подводя итог, запомним, что стили к HTML-странице можно подключать с помощью тега <LINK> И инструкции @import, помещать их в раздел HEAD между тегами <STYLEX/STYLE> И вставлять непосредственно в элементы с помощью атрибута STYLE.

Селекторы

Вы уже знаете, как интегрируются таблицы стилей и HTML, и теперь у вас сразу должен возникнуть вопрос, *как выбирать элемент или группу элементов, к которым применяется тот иди иной стиль?*

Для осуществления подобного рода выборки и существуют так называемые селекторы.

> ***Определение***
>
> Селектором будем называть конструкцию, позволяющую выбрать элемент, к которому будут применены данные стили.

В CSS-1 есть сравнительно немного способов выбора элемента, к которому можно применить определенный стиль.

Селектор по элементу

В качестве селектора может выступать название самого элемента (p, EM, TD, BODY). Например, конструкция:

```
P (
  color: #CCC;
  font-size:  14px}
```

обозначает, что текст внутри всех тегов <PX/P> будет серого цвета и размером 14 пикселов.

Аналогично можно в качестве селекторов использовать любые элементы, имеющиеся в языке HTML. Но заметьте, что при этом примененные к данному селектору стили будут распространяться на все элементы данного вида, встречающиеся на странице. Например, в следующей таблице стилей черный цвет задается для *всех* элементов <TD> И <P>, а красный цвет для *всех* элементов <EM>:

```
TD {
  color: #000J
P 1
  color: #000}
```

```
  color: #F00)
```

Однако зачастую надо каким-то образом выделить элемент из себе подобных. Например, вы хотите, чтобы некоторые заголовки <Hl> были синими с размером шрифта 18 пикселов, а некоторые заголовки <Hl> черными с размером шрифта 16 пикселов. Совершенно очевидно, что если в качестве селектора брать название элемента H1, ТО нам не удастся добиться такой дифференциации. *Для* решения этой проблемы в CSS-1 ввели два механизма: классы и **уникальные** идентификаторы (или ID).

Сперва рассмотрим понятие класса.

Селектор по классу

Из названия можно сделать вывод, что этот механизм позволяет разбивать лпементы на классы, которые могут иметь совершенно разные стилевые решения. Делается это следующим образом:

```
<P>Здесь содержится какой-нибудь текст,</P>
<P CIASS="sinaller">A этот абзац будет выведен Солее мелким шрифтом. </P>
```

Как видите, прямо в теле документа мы присваиваем элементу <p> класс, который носит название smaller. А стили к данному классу применяются следующим образом:

```
P I
  color: #CCC;
  font-size: 14px}
?.smaller {
  font-size: 12px}
```

В этом случае первый абзац будет выведен шрифтом размером 14 пикселов, а второй — шрифтом размером 12 пикселов. Как видите, синтаксис похож на **применяемый** в объектно-ориентированных языках программирования, доступ к классу осуществляется через точку, словно класс smaller является свойством объекта <p>.

Надо сказать, что в CSS можно применять данный класс к совершенно раз**шчныш** элементам. Например:

```
<PRE CIASS="srtialler">3flecb содержится какой-нибудь текст, который сохранит свое форматирование.</PRE>
<P CLASS="smaller">A в этом абзаце будет просто текст.</P>
```

В данном случае класс smaller имеют и элемент <P>, и элемент <PRE>. В таблице стилей можно указать, что стиль для класса smaller применяется

ко всем элементам, а не только к элементам <p>, как было отмечено ранее. В этом случае перед точкой имя элемента не ставится:

```
.smaller {
  color: #CCC;
  font-size: 14px}
```

Фактически, селектор по классу, начинающийся с точки, применяется ко всем элементам с данным классом, т. е. пустота перед точкой обозначает *любой элемент*. Тогда текст, соответствующий этим элементам, будет выведен серым шрифтом размером 14 пикселов.

А можно дифференцировать элементы <P> и <?RE>, имеющие одинаковый класс smaller:

```
P.smaller {
  color: #CCC;
  font-size: 14pxf
PRE.smaller i
  color: #F00;
  font-size: 12px}
```

В этом случае текст в элементе <PRE> будет выведен красным шрифтом размером 12 пикселов, а текст в элементе <p> — серым шрифтом размером 14 пикселов.

Селектор по *ID*

Перейдем к рассмотрению уникальных идентификаторов, которые мы для краткости будем называть просто ID. С помощью ID можно применять стили к точно выбранному элементу, т. е. в HTML-коде может быть только один элемент с данным ID. Например:

```
<H1 ID="first">Уникальный заголовок</H1>
```

Селектор для 1D-"first" формируется, в общем-то, аналогично селекторам по классам, но вместо точки для адресации используется знак #:

```
H1#first {
  color: #00F;
  font-family: Verdana, sans-serif)
```

Однако ID является более специфичным, чем класс. Предположим, в коде есть элемент <HI> вида:

```
<H1 CLASS="redhead" ID="first">yHHKajibHbiu заголовок</H1>
```

а в таблице стилей класс и ID описаны следующим образом:

```
H1#first {
  color: #00F;
  font-family: Verdana, sans-serif}
H1.redhead |
  color: #F00;
  font-family: "Times New Roman", serif}
```

Тогда этот заголовок будет выведен на экран синим цветом и шрифтом Verdana. Таким образом, если понадобится выделить из класса какой-либо элемент, можно сделать это с помощью ID. Конечно, можно просто ввести новый класс, но иногда от этого страдает логичность кода, и сам код весит больше.

Кстати говоря, в HTML-документе может быть только один ID, Т. е. код, приведенный ниже, будет *некорректным:*

```
<P П)="рага">Абзац текста</P>
<CODE П>="рага">Пример кода</COOE>
```

Однако все браузеры позволяют применять стиль к нескольким элементам, содержащим одинаковый ID, ЧТО является ошибкой, хотя и не грубой.

В CSS-1 есть еще один вид селекторов — контекстные селекторы.

Контекстный селектор

Например, вы хотите, чтобы в заголовках <H2> фразы, выделенные с помощью элемента <STRONG>, были оранжевого цвета. Первое, что приходит на ум, написать такие стили:

```
H2 {
  color: #00F}
STRONG (
  color: orange}
. . .
<H2>Заголовок с <STRONG>opareKeBbJM</STRONG> выделением</H2>
```

Однако в этом случае *все* элементы <STRONG> на странице будут оранжевого цвета. Можно еще отвести под все элементы <STRONG>, находящиеся внутри элементов <H2>, отдельный класс. Тогда код придется написать так:

```
-H2>Заголовок с <STRONG CLASS-"inhead">BHfle«eHHbJM</STRC»JG> словом</H2>
```

А таблица стилей будет следующий:

```
-2 {
  color: #00F}
```

```
.inhead {
  color: orange}
```

Однако достаточно неудобно все время вставлять класс в элемент <STRONG>, потому что это увеличивает код. По этой причине и ввели контекстные селекторы, которые позволяют применить стиль к определенному виду элементов, находящихся внутри другого элемента. В нашем случае стиль будет выглядеть так:

```
H2 STRONG {
  color: orange}
```

Здесь в качестве селектора выступают *имена элементов через пробел.* Дословно это можно перевести на русский язык так: *"Элементы <STRONG>, которые находятся внутри элементов <H2>, будут иметь оранжевый цвет".* Аналогично можно применять стили к элементам любого уровня вложенности. Например, стиль

```
P CODE EM {
  color: green;
  font-weight: bold}
```

выводит элементы <EM> полужирным шрифтом зеленого цвета в коде:

```
<P>Здесь идет текст абзаца, а тут начинается <COOE>пример компьютерного
кода с <EM>выделенным</EM> словом</СОБЕ>, после чего абзац продолжает-
ся.</P>
```

Мы рассмотрели четыре способа выбора элемента, к которому будет применен стиль. Заметьте, что все они основываются на расположении элемента в дереве документа.

> ***Определение***
>
> Деревом документа называется структура всех элементов на странице с учетом их вложенности.

Однако есть достаточно много ситуаций, когда информации о расположении элемента в документе недостаточно для адресации. Например, в языке HTML нет элемента, который соответствовал бы первой букве параграфа (так называемой буквице). По этой причине данный элемент отсутствует в дереве документа и обычным способом невозможно применить стиль к первой букве параграфа.

Псевдоэлементы

Для решения проблемы с буквицей в CSS-1 ввели псевдоэлементы first-letter и first-line, которые соответствуют первой букве и первой строке параграфа. Приставка *псевдо* подчеркивает тот факт, что данные элементы но

являются элементами в обычном смысле этого слова, т. е. в самом теле документа их нет, поэтому и в дереве документа они не отображаются, но в селекторах их можно использовать. Наличие псевдоэлементов позволяет делать буквицы и выделять первые строки параграфов без каких-либо ухищрений.

На самом деле, буквицу можно реализовать и без применения псевдоэлемента first-letter. В документе надо написать такой код:

<PXSPAN CLASS»"fletter">n</SPAN>epBafl Оуква в данном абзаце будет большой и краснсх.</P>

И применить следующие стили;

```
P (
  font: 12px Verdana, Tahoma, sans-serif!
.flatter {
  color: #F00;
  font: bold Збрх Verdana, Tahoma, sans-serif)
```

Тогда в браузере мы увидим картину, показанную на рис. 4.1.

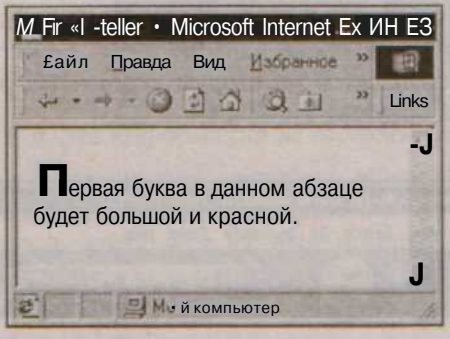


Рис. 4.1. Реализация буквицы без применения псевдоэлемента first-letter

Однако данный способ имеет существенный недостаток: нам придется *во всех* абзацах текста *вручную* выделять первую букву тегами <SPAN :LASS="fietter"></SPAN>. С применением псевдоэлемента first-letter **Зуквица** реализуется совсем просто, в таблице стилей надо написать:

```
P {
  font: 12px Verdana, Tahoma, sans-serif
P:first-letter {
  color: #F00;
  font: bold Збрх Verdana, Tahoma, sans-serif;
  float: left}
```

А в HTML-коде вообще не нужно будет выделять первую букву, т. е. можно просто написать:

```
<P>Первая буква в данном абзаце будет большой и красной.</?>
```

Селектор p:first-letter применять стили ко всем первым буквам текста в элементах <p>. Свойство float: left указывает на то, что текст должен обтекать буквицу справа. На рис. 4.2 предста&1ен вид подобной буквицы.

Рис. 4.2. Буквица, созданная с помощью CSS

Что касается выделения первых строк абзацев, то без применения псевдоэлемента first-line его вообще невозможно осуществить, потому что длина первой строки может изменяться. Например, если попробовать **заключить** некоторое количество символов в теги <SPAN></SPAN> по аналогии с попыткой реализации буквицы, то на больших экранах первая строка будет длинной и кроме символов, заключенных в теги <SPAN></SPAN>, в нее войдут символы, находящиеся после данных тегов. Если же экран маленький, то часть символов перенесется на вторую строку. На рис. 4.3 показано, как это будет выглядеть в браузере Netscape 6.x

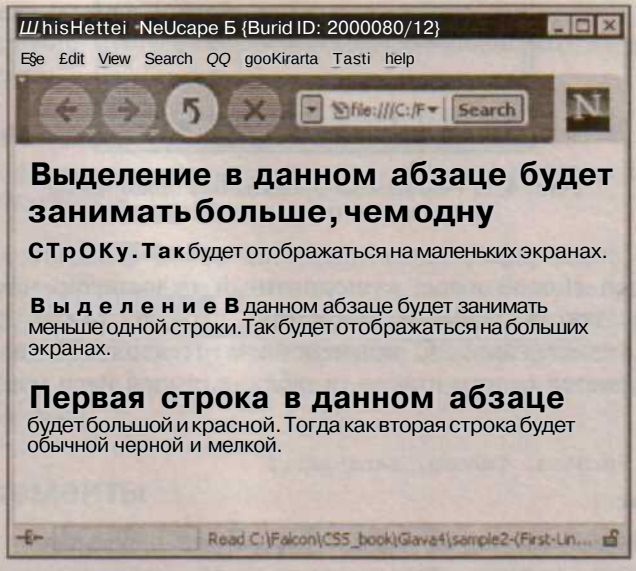


Рис. 4.3. Реализация выделения первой строки различными способами. С помощью псевдоэлемента first-line реализовано выделение в третьем абзаце

Выделение первой строки в третьем абзаце выполнено следующим образом:

```
P:first-line {
  color: #F00;
  font: bold 20px Verdana, Tahoma, sans-serif}
```

Надо сказать, что псевдоэлементы не поддерживаются браузерами Internet Explorer 5.0 и Netscape Navigator 4.x, что, впрочем, не является препятствием к их использованию. Просто эти браузеры будут игнорировать псевдоэлементы и выделения первой строки или первой буквы не произойдет, что само по себе некритично.

Кроме псевдоэлементов в CSS есть еще пссвдоклассы.

Псевдоклассы

Определение

Псевдоклассом называется класс, который основывается на информации, не являющейся частью HTML-документа.

Например, есть псевдокласс, который обозначает посещенные ссылки, однако в коде документа совершенно невозможно отличить посещенные ссылки от непосещенных. В данном случае такую информацию может предоставить браузер, на основе этих данных и определяется какая это ссылка: посещенная или нет, т. е. эта информация *не содержится в теле документа.*

В CSS-1 есть четыре псевдокласса, которые обозначают непосещенную ссылку, посещенную ссылку, ссылку при наведении курсора мыши и активную ссылку, т. е. такую на которую в данный момент нажимает посетитель сайта. Вот простейшая таблица стилей, которая содержит все четыре псевдокласса:

```
A:link {
  color: #00F}
A:visited [  color: #CCC}
A:active (
  color: #F00}
A:hover (
  color: #F90}
```

В данном случае непосещенные ссылки на странице будут синего цвета, посещенные — серого, активные — красного, а при наведении курсора мыши ссылки будут выделены оранжевым цветом. В HTML есть аналог трех из этих псевдоклассов — это атрибуты тега <BODY>:

- LINK — соответствует непосещенным ссылкам;

D VLINK — соответствует посещенным ссылкам;

O ALINK — соответствует активным ссылкам.

Однако таким способом можно задать один и тот же цвет для всех ссылок на странице, но никак нельзя их дифференцировать. А вот с помощью псевдоклассов это делается очень легко, потому что вместе с псевдоклассами можно использовать и обычные классы. Например, внизу страницы часто дублируют навигацию. Цвет таких ссылок может отличаться от цвета основных ссылок. Можно написать следующий CSS-код:

```
A.footer:link {
  color: #000}
A.footer:visited {
  color: #CCC}
A.footer:active (
  color: #00F)
A.footer:hover {
  color: #FFF}
```

Тогда в HTML-коде ссылка

```
<A HREF="/" CLASS="footer">На главнукХ/А>
```

в зависимости от статуса будет отображаться именно указанными выше цветами.

Кроме того, псевдоклассы можно использовать и с ID. Например, так:

```
A#someid:active I
  color: #F00}
```

В данном случае цвет активных ссылок с rD="someici" будет красным.

В спецификации CSS-1 не существует иных способов выбрать элемент, к которому будет применен стиль, но отдельные виды селекторов из спецификации CSS-2 уже поддерживаются некоторыми браузерами. Так что к ним мы еще вернемся несколько позже. А теперь детально разберем структуру правил, с помощью которых применяются стили.

Синтаксис и структура CSS

Для начала дадим определение понятию правила в CSS.

> ***Определение***
>
> Правилом называется структурная единица таблицы стилей, которая содержит описание стилей для данного элемента.

Правило имеет структуру, изображенную на рис. 4.4.

Рис. 4.4. Структура правила CSS

Таким образом, *праеiио* состоит из *селектора,* который всегда располагается :лева, и *блока объявлений сгтшй,* который заключается в фигурные скобки и :ледуст непосредственно за селектором. Каждое объявление в свою очередь :остоит из *свойства* и его *значения.* Именно свойства обозначают вид стиля, -.оторый будет применен к элементу, выбранному с помощью селектора. Свойством может быть цвет элемента (color), параметры шрифта (например, font-size), отступы (например, padding-left) и пр. Подробнее все .войства мы рассмотрим позже в этой главе.

Правило может содержать несколько объявлений. Впрочем, объявлений может и не быть вовсе, поэтому конструкция

```
ai {}
```

синтаксически корректна, хотя и никак не влияет на отображение элементов <H1>.

Если объявлений несколько, то все они отделяются друг от друга точкой с запятой. После последнего объявления точку с запятой можно не ставить. Поэтому стили:

```
H1 {
  font-family: Verdana;}
H1 {
  font-family: Verdana}
```

совершенно эквивалентны и корректны.

Свойство и его значение всегда разделяются двоеточием. В правиле

```
={
  color: #CCCr
```

свойством является цвет (color), а значением является задание этого цвета в RGB (#ccc).

3 некоторых случаях свойство может иметь несколько значений. Тогда они разделяются пробелами. Например, правило

```
ai{
  font: bold 18px Verdana}
```

указывает на то, что заголовки <H1> надо выводить полужирным шрифтом Verdana размером 18 пикселов.

Иногда есть необходимость задать ряд значений, которые применялись бы в случае невозможности применения предыдущих. Например, можно задать шрифтовой ряд:

```
H1 (
  font-family: Verdana, Tahoma, sans-serif}
```

В данном примере, если в операционной системе пользователя не установлен шрифт Verdana, то для отображения заголовков <H1> будет использован шрифт Tahoma. Если же и этот шрифт не установлен, то будет использован шрифт, являющийся для данной операционной системы шрифтом без засечек по умолчанию (он обозначается ключевым словом sans-serif).

Все пробелы и переводы строк в блоке объявлений игнорируются, так что для улучшения восприятия таблицы стилей можно использовать любое число этих символов. Например, правило

```
DIV {
  color: #000;
  background-color: #DDD;
  font-size: llpx}
```

совершенно аналогично **правилу**

```
DIV{color:#000,-background-color:#DDD;font-size:llpx}
```

Но первый способ организации кода значительно облегчает восприятие. Правда, если таблица достаточно большая, то при таком оформлении поиск нужного селектора может только затрудниться, но это уже дело привычки. Второй способ более экономный и помогает сократить код. Насколько велика экономия, можно легко выяснить на практике. В одном проекте таблица стилей, организованная первым способом, весила 2,82 Кбайт. После того, как я убрал все лишние пробелы **и** переводы строк, она стала весить 2,09 Кбайт, т. е. вес уменьшился на 26%. Конечно, экономия пусть даже одного килобайта дело хорошее, но в данном случае редактировать код практически без пробелов и переводов строк ужасно неудобно. Вот если бы была программа, которая сжимала исходный удобный код в экономичный, а при необходимости восстанавливала все обратно, то это было бы правильно. Но я такой программы не знаю. Вероятнее всего, оптимизатора CSS-кода вообще пока нет в природе, так что лучше пользоваться первым способом организации кода или его вариациями. Конечно, если вы владеете каким-либо языком программирования, то можете и сами написать оптимизатор для файлов CSS, но дело это непростое.

Группировка селекторов

Часто возникают ситуации, когда к разным элементам применяются одни и :е же стили. Например, нам надо, чтобы текст внутри элементов <P> И <TD> выглядел одинаково. Тогда мы создадим нечто похожее на это:

```
P (
  font: 12px Verdana, Tahoma, sans-serif}
TD {
  font: 12px Verdana, Tahoma, sans-serif}
```

Сразу же возникает вопрос, можно ли такую конструкцию оптимизировать? Оказывается, что можно: такая оптимизация называется *группировкой*.

> ***Определение***
>
> Группировкой называется объединение селекторов с одинаковыми объявлениями с целью уменьшений объема кода.

При группировке селекторы должны разделяться запятой:

```
?, TD {
  font: 12px Verdana, Tahoma, sans-serif}
```

Это правило совершенно аналогично двум правилам из предыдущего примера. Однако использовать группировку надо осторожно, потому что иногда может нарушаться логичность таблицы стилей. Например, селекторы в таблице стилей

```
нб {
  color: #F00;
  font: 12px Verdana, Tahoma, sans-serif}
P (
  font: 12px Verdana, Tahoma, sans-serif}
```

•южно сгруппировать следующим образом:

```
Нб, P |
  font: 12px Verdana, Tahoma, sans-serif}
Нб (
  color:ftFOO}
```

что сокращает размер кода на 32 байта. Однако логичность значительно ухудшается, и если вам потребуется изменить стиль элементов <нб>, то вы можете и вовсе не заметить расположенное в совершенно другом месте **правило**

```
нб {
  color: #F00}
```

определяющее цвет заголовков. Поэтому группировать нужно элементы только с *абсолютно одинаковыми* стилями, как в первом примере этого раздела.

Вот мы и подошли к одной весьма интересной проблеме. Итак.

Каскадирование

На протяжении трех глав вы только и слышите *"каскадные таблицы стилей"*. Что такое *"стили"* должно быть уже понятно. Слово *"таблицы"* появилось вследствие перевода на русский язык, в оригинале CSS расшифровывается как Cascade Style Sheets, т. е. дословно "Каскадные листы стилей". Однако слово "листы" как-то не прижилось, так что вместо слова "листы" употребляется слово *"таблицы"*. Остается еще один термин: *"каскадные"*. Почему именно *каскадные* таблицы стилей, а не какие-либо другие? На этот вопрос вы сейчас и получите ответ.

Как вы видели в предыдущем примере, к одному и тому же элементу можно применять несколько правил. Поэтому следующая таблица стилей совершенно корректна с точки зрения CSS:

```
EM {
  color: #F00}
EM {
  font: 12px Verdana, sans-serif}
```

В таком случае к элементу <EM> будут применены оба правила, т. е. текст внутри <EM> будет отображаться на экране красным цветом и шрифтом Verdana. Но очень часто возникают и более запутанные ситуации. Допустим, у нас есть такая таблица стилей:

```
P {
  color: #000;
  font: 12px Verdana, Tahoma, sans-serif}
P,smaller {
  color: #333;
  font-size: 10px}
CODE P {
  color: #00F;
  font-family: "Courier new", monospace}
```

А в коде страницы имеется конструкция:

```
<CODE>
  <P>Здесь идет код програмы</P>
```

```
<р CIASS="smaller">А здесь что-нибудь менее важное из кода мелким шриф-
том</P>
</CODE>
```

Получается, что тег <p CLASS="smaiier"> удовлетворяет всем трем правилам 5 таблице стилей:

^ он является элементом <P>;

3 он относится к классу smaller;

D он вложен в элемент <CODE>.

Зозникает вполне логичный вопрос, какие именно правила применять к данному элементу? Понятно, что он не может одновременно иметь три разных цвета и разный размер шрифта. Для решения этой проблемы ввели механизм *каскадирования,* который функционирует по следующему алгоритму.

I. Интерпретатор таблицы стилей (иными словами, веб-браузер) сначала находит в таблице *все* объявления стилей для данного элемента. Если селектор соответствует элементу, то объявление применяется. Допустим, если мы написали в таблице стилей H4 {color: green} и если в коде встретится элемент <H4>, то к нему будет применено объявление color: green. В нашем примере, однако, все селекторы соответствуют элементу <p>, так что на данном шаге алгоритма конфликт не устраняется. Если же на элемент нет стилей, то они наследуются от предыдущего элемента. Например, в коде:

```
<STYLE TYPE="text/css">
  P (
    color: #AAA}
</STYLE>
. . .
<P>Текст в <EM>выделенным</EM> словом</P>
```

элемент <EM> будет серого цвета, хотя явно это в стилях не указывалось. Тем не менее, элемент <EM> является потомком элемента <p>, т. с. он вложен в элемент <p>, и по этой причине унаследует родительский цвет.

Если же на родительский элемент тоже не написаны стили, то используются значения по умолчанию.

2. *Найденные объявления сортируются по степени важности.* В CSS существует ключевое слово : i-portant:, которое позволяет увеличивать важность данного стиля. Пример:

```
P {
  color: #CCC ! important;
  font: 12px Verdana, Tahoraa, sans-serif}
```

В некоторых браузерах можно настраивать свои собственные таблицы стилей, которые переписываются авторскими стилями. Однако пользователь может пометить некоторые объявления как важные, тогда они переобозначат авторские стили. Но уж если автор пометит объявление словом !important, то оно в любом случае переобозначит стиль, установленный пользователем.

3. Далее найденные объявления сортируются по параметру, который называется *специфичностью*. Специфичность вычисляется следующим образом: считается количество ID В селекторе (я), считается количество классов в селекторе *(Ь),* считается количество имен элементов в селекторе (с). Значение специфичности получается путем объединения трех значений, т. е. фактически это специфичность селектора. Например, селектор р {} имеет специфичность 1, потому что он не содержит ID *(а* = 0), он не содержит классов *(Ь* = 0) и содержит только одно имя элемента *(с* = 1). Получается 0-0-1, т. е. 1. Некоторые дополнительные примеры приведены в табл. 4.2.

Таблица 4.2. *Примеры вычисления специфичности различных селекторов*

Селектор	Вычисление	Специфичность
P.smaller {}	a = 0; Ь = 1; с - 1	11
CODE P {}	a = 0; Ь = 0; с = 2	2
Пауег U	a = 1; Ь = 0; с = 0	100
DIV#layer P.smaller {}	a = 1; Ъ = 1; с = 2	112
#layer #inlayer P	a = 2; Ь = 0; с = 1	201

Правила, относящиеся к более специфичному селектору, будут переобозначать правила, относящиеся к менее специфичному селектору. Если же специфичность будет одинакова, то к элементу будет применен тот стиль, который описан позже. Например, если в таблице стилей написано

```
HI (
  color: green}
HI {
  color: #F00}
```

то заголовки будут красными.

Вернемся к нашему примеру:

```
P {
  color: #000;
  font: 12px Verdana, Tahoma, sans-serif}
```

```
P.smaller {
  color: #333;
  font-size: 10px}
CCOE P {
  color: #00F;
  font-family: "Courier new", monospace}
```

В коде селектор p |} имеет специфичность 1, селектор p. smaller {f имеет специфичность И и селектор p CODE {} имеет специфичность 2. Следовательно, самыми важными будут яачяться объявления стилей в правиле с селектором p.smaller (}, затем в правиле с селектором p CODE {}, а затем уж в правиле с селектором p. Таким образом, в коде

```
<CODE>
<P CLASS«"smaller">A здесь что-нибудь менее важное из кола мелким шриф-
том</P>
</CODE>
```

тег <p CLASS="smaller"> будет отображен серым цветом #333 и шрифтом размером 10 пикселов. А начертание шрифта Courier New он унаследует от правила

```
CODE P {
  color: #00F;
  font-family: "Courier new", monospace}
```

Собственно, в этом и заключается механизм каскадирования. Я здесь уже упоминал термин *"наыедование"*, а теперь поговорим о нем подробнее. Каждый документ имеет структуру или дерево. Например, возьмем простейший документ:

```
<HTML>
<HEAD>
  <T1TbE>Простейшая страничка</T1TbE>
</HEAD>
<BODY>
  <H1>Заголовок</H1>
  <P>Здесь текст с <5TKOЫC>вышеленным</3TKOЫC> словом</P>
  <P>Здесь просто текст</P>
</BODY>
</HTML>
```

Он имеет следующее дерево:

```
HTML
  BODY
```

```
H1
p
  STRONG
P
```

Элемент <HTML> является самым верхним в иерархии, затем идет элемент <BODY>, элементы <H1> и <P> образуют третий уровень вложенности. Таким образом, элемент <HTML> является предком всех остальных элементов, а элемент <BODY> — предком всех отображаемых в браузере элементов.

В CSS реализован механизм наследования стилей. Это значит, что применение к элементу <BODY> стилей вида:

```
BODY {
  color: #000;
  font: 12px Verdana, Tahoma, sans-serif}
```

приведет к тому, что все элементы, вложенные в элемент <BODY> (элементы <p> и др.), будут выведены черным цветом и шрифтом Verdana размером 12 пикселов, но только в том случае, если к ним самим не применен какой-либо иной стиль.

Определение

Наследование — это механизм, который позволяет элементам-потомкам иметь те же стили, что и у элемента-предка.

Обобщая, можно дать четкое определение понятия "каскадирование".

Определение

Каскадирование— это метод определения веса или важности конкретного объявления, который устраняет конфликты, возникающие в случае нескольких объявлений для одного и того же элемента.

Единицы измерений

Без знания того, как задавать величины в каскадных таблицах стилей, дальнейший разговор лишен смысла. По этой причине подробно рассмотрим. как задаются значения цветов, линейных размеров и URL. Начнем, пожалуй, с линейных размеров.

Единицы длины

Все единицы длины служат для задания вертикальных или горизонтальных размеров. Например, правило

```
DIV#padd {
  padding-left: 10px}
```

устанавливает левый отступ элемента <DIV> С iD="padd" равным 10 пикселам. Само значение длины строится следующим образом: вначале идет знак - или +, причем знак - ставится по умолчанию и его можно опускать, затем следует численное значение, а в конце *идентификатор размерности,* представляющий собой двухбуквенную аббревиатуру (его, кстати, тоже можно опускать, если численное значение равно нулю). Надо сказать, что не все свойства могут иметь отрицательные значения, а какие именно — вы узнаете позже.

Все единицы длины делятся на два больших класса: *относительные и абсолютные.* Относительные единицы устанавливают размер относительно какой-то другой длины. Их всего три:

- px

  1px равен одной точке на экране монитора, которую обычно называют пикселом. Обычно мониторы имеют разрешение 800x600 пикселов или 1024x768 пикселов {прочие значения встречаются гораздо реже);

- em

  1em равен размеру (высоте) шрифта данного элемента, т. е. если элемент <H1> имеет размер шрифта 18 пикселов, то правило:

  ```
  H1 (padding-top: 2em}
  ```

  обозначает, что верхний отступ элементов <H1> равен 2x18=36 пикселам;

- ex

  1ex равен высоте строчной буквы "x" в шрифте. Именно по этой букве формируется оптическая высота строки. Обычно она соответствует половине высоты шрифта, однако в некоторых шрифтах это соотношение нарушается. Надо сказать, что особой разницы между em и ex нет, но в некоторых случаях пользоваться теми или иными единицами удобнее. Кроме того, для свойства font-size величина em или ex определяется по родительскому элементу. Например, в коде:

  ```
  P {
    font-size: 12px}
  EM {
    font-size: 1.5em}
  <P>Текст с <EM>выделением</EM></P>
  ```

  Текст в элементе <p> будет отображен шрифтом размером 12 пикселов, а текст в элементе <EM> шрифтом размером в 1,5 раза больше, т. е. 18 пикселов.

Абсолютные единицы длины имеют фиксированный размер. Их всего пять видов:

- in

  1in равен одному дюйму, который в свою очередь равен 2,54 см;

- pt

  1pt равен 1/72 дюйма, эта величина называется *типографским пунктом* и пришла в CSS из полиграфии, где очень часто кегль шрифта указывается в пунктах;

П pc

  1pc равна i2pt или 1/6 дюйма, эта величина называется *пикой* (тоже из **полиграфии);**

О mm

  1mm равен одному миллиметру метрической системы единиц;

П cm

  1cm равен одному сантиметру (тоже из метрической системы единиц).

Надо сказать, что использовать абсолютные величины надо только в том случае, когда точно известны параметры устройства вывода. Например, экраны мониторов могут иметь совершенно различные размеры, которые и определяют выставляемые разрешения. Так что шрифт размером 10 pt может хорошо читаться на маленьких разрешениях (640x480), но плохо на больших (1024x768). Поэтому в таблицах стилей, которые используются именно для вывода на экран, лучше употреблять относительные единицы длины. Абсолютные единицы длины можно применять, скажем, при выводе страницы на печать.

Кроме того, в CSS в качестве единицы измерения можно пользоваться процентами от какой-либо величины. Без процентов совершенно не обойтись при "резиновой" верстке. Как вам уже известно, используя CSS, теоретически можно обойтись без таблиц при верстке сайтов, а все таблицы заменять так называемыми блоками. Поэтому для задания ширины и высоты этих блоков при такой верстке необходимо использовать процентные соотношения. Например, в коде

```
<HTML>
<HEAD>
  <TITLE>Небольшой пример верстки в три колонки</T1TbE>
  <STYLE TYPE="text/css">
    DIV#leftpanel {
      width: 30%;
      float: left}
    Div#middlepanel (
      width: 40%;
      float: left)
    DIV#rightpanel {
      width: 30%;
```

```
      float: left}
  </STYLE>
</HEAD>
<BODY>
  <DIV ID="leftpanel">JleBafl колонка</DIV>
  <DIV lD="middlepanel'^Центральная колонка</DIV>
  <DIV ID="rightpanel">npaBa* колонка</ИУ>
</BODY>
</HTML>
```

устанавливаются процентные соотношения между шириной колонок 30—40, *л* проценты вычисляются относительно ширины окна браузера.

Подробнее о верстке с помощью блоков мы поговорим позднее, но что такое процентные соотношения в CSS вы. надеюсь, поняли.

Обозначение цвета

В CSS существует несколько способов задания цвета. В примерах вы уже сталкивались с тремя. Прежде всего, цвета можно задавать с помощью ключевых слов. Их 16 и пришли они из палитры VGA. Например, синий цвет задается так:

```
H1 {
  color: blue)
```

Я не рекомендую задавать цвета таким образом, потому что это вносит некоторую путаницу в код, да и ключевые слова обычно длиннее другого способа обозначения цвета.

Очень популярной является цветовая схема RGB, где все цвета описываются степенью насыщенности красного, зеленого и синего компонентов. Насыщенность компонентов указывается числами в шестнадцатеричном формате. Например, белый цвет обозначается `#FFFFFF`, черный — #000000, красный — `#FFOCOO`, серый — #666666. Кроме того, есть сокращенная запись цветов. Например, белый — `#FFF`, красный — `#FCO`. Как видим, вместо шести цифр можно записать три, причем они дублируются. Понятно, что такая сокращенная запись возможна только в том случае, когда цифры в каждом компоненте цвета одинаковые. Например, цвет `#CD00FF` невозможно записать сокращенно, потому что красный компонент выражен разными цифрами C и D. А вот цвет `#DDOOFF` МОЖНО записать в виде `#DOF`.

На практике почти всегда пользуются именно такими обозначениями цветов, но существует еще определение цвета в виде функции:

3 HI {color: rgb(0,0,255)} — заголовки будут выведены синим цветом;

3 HI (color: rgb(0, юо%, 0)}—заголовки будут зелеными.

Задание URL

В некоторых свойствах надо указывать пути к файлу. Например, в свойстве background-image надо указывать путь к графическому файлу, который будет являться фоном. Делается это следующим образом:

```
30DY {
  background-image:url{http://www.artel.by/i/backl.gif))
```

Формат универсального локатора ресурса следующий: сначала идет обозначение функции uri, а затем в круглых скобках указывается путь к файлу. Можно указывать абсолютные пути (как в примере), а можно и относлтельные, причем путь будет определяться *относительно таблицы стилей, а не относительно документа.* Так что, если вы подключаете к документу внешнюю таблицу стилей с помощью тега <ыкк>, то путь будет определяться относительно местонахождения на сервере данной таблицы стилей, а если вы используете в документах встроенные таблицы стилей между тегами <STYLE></STYLE>, то в этом случае путь будет определяться относительна документа, потому что физическое местонахождение таблицы стилей и документа совпадает (они находятся в одном и том же файле).

Итак, мы разобрались с базовыми понятиями CSS и ключевыми концепциями. Вы теперь знаете, как интегрировать стили с HTML-документом, как выбирать элемент, к которому применять стили, и что такое каскадирование. Иными словами, вы знаете, КАК работает CSS, но вы пока не знаете, ЧТО конкретно можно сделать с помощью CSS. Мы переходим к предметной части, т. е. к рассмотрению основных свойств спецификации CSS-I.

Свойства

Цвет и фон

Возможность изменять цвета на веб-страницах появилась уже достаточно давно. Очевидно, цвет элемента относится к визуальному представлению, так что в CSS есть способы его задавать. Можно устанавливать цвет текста, фона и рамок в элементе.

color

Задает цвет элемента. Можно применять ко всем элементам без исключения.

В качестве значения указывается цвет в одном из доступных форматов. Например, следующие правила совершенно идентичны и устанавливают красный цвет логического выделения в тексте:

```
EM (
  color: red}
EM i
  color: #F001
```

```
EM{
  color: rgb(255,0,0>)
```

В HTML тоже можно задавать цвет текста элемента. Делается это с помощью атрибута COLOR тега <FONT> (табл. 4.3).

Таблица 4.3. *Сравнение установки цвета текста с помощью HTML и CSS*

HTML	CSS
<PXPONT СОЪОК="#CCCCCC">Серый текст</FONT></P>	P { color: #CCC} . . . <P>Серый текст</P>

background-color

Задает цвет фона элемента.

В качестве значения указывается цвет в одном из доступных форматов или ключевое слово transparent, которым обозначается прозрачный фон. По умолчанию фон установлен как transparent.

В HTML цвет фона задается атрибутом BGCOLOR, однако он есть только у элементов <BODY>, <ТАЗЬЕ>, <TR>, <TH> И <TD>, поэтому в случае, если надо сделать абзац с каким-либо фоновым цветом, приходится использовать таблицу. Это продемонстрировано в табл. 4.4.

Таблица *4.4. Сравнение установки фонового цвета с помощью HTML и CSS*

HTML	CSS
<TABLE> <TRXTD BGCOLOR="green"> <FONT COLOR="#FFFFFF"> Белый текст на зеленом фоне</ЕШТ> </TDx/TR> </TABLE>	P { background-color: green; color: #FFF} . . . <P>Белый текст на зеленом фоне</P>

background-image

Задает графическое фоновое изображение элемента.

Значения:

3 URL графического файла;

3 ключевое слово попе, которое обозначает отсутствие фонового изображения.

Понятное дело, что нет смысла специально указывать background-image: попе, поскольку это значение у всех элементов установлено по умолчанию.

Вообще, если указывается фоновое изображение, то рекомендуется указывать и фоновый цвет, потому что рисунок может не загрузиться или пользователь отключит загрузку изображений. В этом случае текст может плохо читаться на фоне, заданном по умолчанию, так что лучше указывать приемлемый фон явно.

В HTML фоновое изображение задается с помощью атрибута BACKGROUND, который имеют только элементы <TABLE>, <BODY>, <TH> и <TD>. В табл. 4.5 фоновое изображение устанавливается для абзаца текста с помошью таблицы.

Таблица 4.5. *Сравнение установки фонового рисунка с помощью HTML и CSS*

HTML	CSS
<TABLE> <TRXTD BACKGBOUND="img/bg.gif"> <FONT COLCR="#FFFFFF">BeJibrfi текст на фоновом рисунке</FONT> </TD></TR> </TABLE>	P { background-image: url{img/bg.gif); color: #FFF) . . . <P>Белый текст на фоновом рисунке</P>

background-repeat

Для заданного фонового изображения это свойство определяет, будет ли это изображение повторяться, и если будет, то каким образом.

Значения:

О repeat — изображение повторяется по горизонтали и по вертикали;

П repeat-x — изображение повторяется только по горизонтали;

О repeat-y — изображение повторяется только по вертикали;

О no-repeat — изображение не повторяется.

В HTML такого атрибута нет вообще, а по умолчанию изображение повторяется и по горизонтали, и по вертикали, так что работать с фоном только средствами **HTML** очень сложно. К счастью, на вооружении есть CSS (табл. 4.6).

Таблица 4.6. *Пример установки неповторяющегося фонового рисунка с помощью CSS*

CSS
~T~{ background-image: url(img/bg.gif}; background-repeat: no-repeat} . . . <P>Текст на фоновом неповторяющемся рисунке</P>

background-attachment

Задает, будет ли перемещаться фон вместе со всем содержимым страницы при скроллировании или нет. Вообще когда фон страницы не перемещается, это несколько непривычно для посетителя сайта и иногда вызывает раздражение. Так что применяйте данное свойство очень осторожно.

Значения:

"3 fixed — фон будет оставаться неподвижным, а содержимое страницы будет перемешаться относительно него;

3 scroll — фон будет перемешаться вместе с остальным содержимым. Значение по умолчанию.

3 HTML нет атрибута, равнозначного данному свойству, а по умолчанию 1юбой фон перемещается при скроллировании, т. е. имеет значение scroll. В табл, 4.7 для элемента <p> устанавливается неповторяющийся нескроллируемый фоновый рисунок.

Таблица 4.7. *Пример установки нескроллируемого фонового рисунка с помощью CSS*

CSS

```
? {
  background-image:url(img/bg.gif);
  background-repeat: no-repeat;
  background-attachment: fixed}
. . .
<P>Текст на фоновом неповторягацемся и неподвижном рисунке</P>
```

background-position

Задает позиционирование фонового изображения. С помощью этого свойства можно смешать фоновое изображение относительно левого верхнего угла элемента.

Свойство имеет два параметра: первый определяет смешение по вертикали, второй — по горизонтали.

В качестве значений можно указывать как положительное, так и отрицательное смещение. Например, правило

```
P {
  background-image: url(img/bg.gif);
  background-position: -12px 50px>
```

смешает фоновое изображение на 12 пикселов влево и на 50 пикселов вниз от левого верхнего угла элемента <p>. Кроме того, можно указывать про-

центные соотношения. Проценты вычисляются относительно ширины и высоты блока элемента. Например, правило

```
P {
  background-position: 20% 40%}
```

смешает фоновое изображение на 20% вправо и на 40% вниз от левого верхнего угла блока элемента <?>. Значением по умолчанию является 0% 0%, что соответствует расположению изображения в верхнем левом углу блока.

Кроме того, можно вместо, **численных** значений указывать выравнивание относительно элемента. Так, **для** выравнивания по вертикали можно использовать три ключевых слова:

П top — выравнивание но верхнему краю;

О center — выравнивание по центру;

• bottom— выравнивание по нижнему краю.

Для выравнивания по горизонтали можно использовать ключевые слова:

• left — выравнивание по левому краю:

О center — выравнивание по центру;

• right — выравнивание по правому краю.

Таким образом, правило

```
P {
  background-position: 0% 0%)
```

эквивалентно правилу

```
P {
  background-position:  top  left}
```

В HTML нет атрибута, который бы соответствовал данному свойству, а значения по умолчанию для фона, заданного средствами HTML и CSS, совпадают и равны 0% 0%.

Весь набор свойств для работы с фоном представлен в табл. 4.8.

Таблица 4.8. Пример фонового рисунка, который отцентрирован с помощью CSS

CSS

```
P {
  background-image:  url{img/bg.gif);
  background-repeat:  no-repeat;
  background-attachment: fixed;
  background-position:  top  center}
```

```
<P>Текст на фоновом неповторяющемся и неподвижном рисунке, который отцентрирован по горизонтали</P>
```

Существует сокращенная форма записи свойств фона. Последний пример можно кратко записать следующим образом:

```
P {
  background: url(img/bg.gif) no-repeat fixed top center}
```

т. е. формат сокращенной записи такой:

```
background: <background-color> |\ <background-image> I I <background-
repeat> I I  <background-attachment> I I <background-position>}
```

В треугольных скобках <> содержатся свойства, а знак I i обозначает пробел. Причем порядок перечисления свойств в сокращенной форме записи не имеет значения, поэтому предыдущее правило вы можете написать и так:

```
P {
  background: fixed no-repeat top center url(img/bg.gif)(
```

Со свойствами цвета и фона мы познакомились, так что самое время сделать первое лирическое отступление.

Показательные выступления (Часть I)

Помните, я в начале главы говорил, что мы постепенно будем внедрять стили в сверстанную в *главе 2* HTML-страничку? Напомню ее кол, чтобы вы не листали зря страницы книги:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE>3HHTMA — криптографические средства зашиты информации</T1^E>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF" TOPMARGIN="0" LEFTMARGIN="0"
        MARGINHEIGHT="O" MARGINWIDTH="O">
    <TABLE CELLPADDING="O" CELLSPACING="O"
           WIDTH="750" BOBDER="0" HEIGHT="167">
    <TR>
      <TD  WIDTH="82">finbsp;</TD>
      <TD WIDTH="266" COLSPAN="2"><IMG SRC="img/fragment_l.gif"
WI[)TH="266" HEIGHT="96" АЬТ="ЭНИГМА - криптографические средства защиты
информации [логотип]"></TD>
      <TD WIDTH="251" ROWSPAN="2"XIMG SRC="img/fragment_2.gif"
WIDTH="251" HEIGHT="167" АЬТ="ЭНИГМА - быстрый путь к надежности"></TT)>
      <TD>finbsp;</TD>
    </TR>
    <TR>
```

```
    <TD WIDTH="82">inbsp;</TD>
    <TD WIDTH="107" HEIGHT="81">bnbsp;</TD>
    <TD WIDTH="159" BGCOLOR="#lD3D4E">£nbsp;</TD>
    <TD>finbsp;</TD>
  </TR>
  </TABLE>
  <TABLE CELLPADDING="10" CELLSPACING="O"
         WIDTH="750" BORDER="0" HEIGHT="400">
  <TR VALIGN="tOp">
    <TD WIDTH»"172" BACKGROUND="img/fragment_3.gif" ALIGN="right">
    <BRXBR>
    <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
       SIZE="3"><B>0 KOMnaHMM</Bx/FONTX/A><BR>
    <A HREF="#"XFONT COLOR="#000000" FACE="Arial"
       SIZE="3"XB>npojr/KTbi</Bx/FONTx/AxBR>
    <A HREF="#"xFONT COLOR="#000000" FACE="Arial"
       SIZE="3"XB>Pememifl</BX/FONTX/AXBR>
    <A HREF="M#"xFONT COLOR="#000000" FACE-"Arial"
       SIZE="3"XB>CTaTbM</B></FONTX/A>
    </TD>
    <TD WIDTH="142" BGCOLOR="#1D3D4E">
    <A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE="l"><B>OTKpbrrafl
криптография</B></FONTX/A><BR>
    <A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" 31rE="1"XB>цифровап
подпись</B></FONTX/AXBR>
    <A HREF="#"><FONT COLOR="#FFFFFF" FACE="Arial"
31rE="1"xB>электронный докуьлент</Bx/ГОЫТХ/A><BK>
    <A HREF="#"XFONT COLOR-"#FFFFFF" FACE="Arial" SIZE="l"><B>4ero не
могут xaKepbi</Bx/FONT></AXBR>
    </TD>
    <TD WIDTH="392">
    <H3 ALIGN="right"><FONT COLOR="#000000" EACE="Ar1a1">открьгтая крип-
тография</ГОШ'X/H3>
    <PXFONT SIZE="2" COLOR="#000000" FACE="Verdana">Bbi запускаете
браузер, набираете в адресной строке URL сайта и получаете желаемую стра-
ницу. Вот, вкратце, действия пользователя, который бродит в сети Интер-
нет. Он не задумывается, что лежит за всем этим. Ему это не нужно. Но это
нужно нам, профессиональным веб-разработчикам, чтобы максимально удовле-
творить пользовательские запросы.</P>
    <P>Мы не будем углубляться в историю, и отряхивать пыль с архивных
90-х голов, но базовые понятия рассмотрим. Итак, что же лежит в основе
всемирной сети Интернет? Базовыми являются три технологии:
    </FONTx/P>
```

```
      </TD>
    </TR>
    </TABLE>
  </BODY>
</HTML>
```

Мы уже можем с вами избавиться от всех цветов и обозначений фона в HTML-коде страницы, перенеся данную информацию в стили. Я буду делать это постепенно, комментируя каждую строчку. Итак, для начала зададим для элемента <=CEY> белый фоновый цвет. Пока это сделано так:

```
.BODY BGCOLOR="#FFFFFF" TOPMARGIN="0" LEFTMARGIN="O" MARGINHEIGHT="O"
1ARGINWIDTH«="O">
```

Перед ТеГОМ </HEAD> ВСТаВИМ теГИ <STYLE TYPE="text/css"x/STYLE>. В ЭТИ теги и будет заключена таблица стилей. Зададим в стилях фоновый цвет для ^сей страницы. Селектором в данном случае будет элемент <BODY>, а для установки фона возьмем сокращенное свойство background:

```
BODY {
  background: #FFP}
```

Тогда из HTML-кода можно убрать атрибут BGCOLOR, который задавал фон. Останется такая строка:

```
<3ODY TOPMARGIN-"0" LEFTMARGIN="O" MARGINHEIGHT="O" MARGINWIDTH="O">
```

Далее у нас есть столбец в таблице, где фоновый цвет должен быть темно-синий:

```
<TD WIDTH="159" BGCOLOR="#1D3D4E">inbsp;</TD>
```

Создадим для такого столбца класс с интуитивно понятным именем bluecol и в стилях пропишем для этого класса фоновый цвет по аналогии с предыдущим примером:

```
.bluecol {
  background: #1D3D4E}
```

В HTML-коде надо указать для данного столбца название класса с помощью атрибута CLASS и убрать атрибут BGCOLOR. Тогда строчка преобразуется так:

```
<TD WIDTH="159" CLASS="blu«col" > ir.r =p;</TD>
```

Далее мы вста&тяди фоновый рисунок в ячейку таблицы;

```
<7D WIDTH-"172" BACKGROUND="iiog/fragment_3.gif" ALIGN-"right">
```

Причем высота фонового изображения была очень большой, чтобы оно не повторялось несколько раз в ячейке. Сейчас можно вернуть изображению его нормальный размер, потому что в стилях мы запретим ему повторяться. Для ячейки, в которую вставляли фоновый рисунок, создадим класс back.

Для вставки фонового изображения воспользуемся сокращенным свойством background. В таблице стилей получится такое правило:

```
.back {
  background: url(img/fragment_3.gif) no-repeat}
```

В HTML-коле уберем атрибут **BACKGROUND** И вставим название класса:

```
<TD WIDTH="172" ALIGN="right" CLASS="back">
```

А теперь займемся ссылками. У нас их два блока. Сначала займемся первы.г блоком. Вот он:

```
<TD WIDTH="172" ALIGN=right CIASS="back">
  <BRXBR>
  <A HREF="#"XFONT COLOR="#000000" FACE="Arial" SIZE="3"><B>0 компа-
HHM</BX/FONT></AXBR>
  <A HREF="#"><FONT COLOR="#000000" FACE="Arial"
SIZE="3"><B>Продукты</B></FONTX/A><BR>
  <A HREF="#"xFONT C0LOR="#000000" FACE="Arial"
SIZE="3"XB>PemeHHH</3></FOHT></A><BR>
  <A HREF="#"><FONT COLOR="#000000" FACE="Arial"
SIZE="3"xB>CTaTbM</B></F0NTX/A>
</TD>
```

Заметьте, что этот блок располагается в отдельной ячейке таблицы <TDX/TD>. которая является для него *контейнером*. Так что если можно будет идентифицировать ячейки, то появится возможность воспользоваться *контекстными селекторами* для того, чтобы задать стили па элементы меню. Это естественно. потому что ссылки должны быть выведены полужирным шрифтом **Aria!** только в главном меню, но не во всем документе. Для идентификации введем и:. Ячейке, в которой располагается первое меню, присвоим iD="mainmenu". Тогда цвет ссылок можно задать с помощью свойства color, а контекстный селектор будет строиться из ID ячейки и названия элемента <A> через пробел.

```
ttmainmenuA(
  color: #000}
```

Тогда ссылки внутри **элемента** с lD="mainmenu" будут черного цвета.

Первый блок **ссылок** будет выглядеть так:

```
<TD WIDTH="172" CLASS="back" ALIGN=right ID="mainroenu">
  <BRXBR>
  <A HREF="#"XFONT FACE="Arial" SIZE="3"XB>0 ксмпа-
HHM</B></FONTX/AXBR>
  <A HREF="#"XFONT FACE="Arial" Г Г~-".'xB>npOflyKTU</BX/F0NTx/A><BR>
  <A HREF="#"XFONT FACE="Arial" SIZE="3"XB>PemeHMH</BX/F0NTx/AXBR>
  <A HREF="#"XFONT FACE="Arial" SIZE="3"XB>CTaTbM</BX/F0NTx/A>
</TD>
```

Второй блок ссылок тоже располагается в отдельной ячейке таблицы. Вот он:

```
<TD WIDTH="142" BGCOLOR="#1D3D4E">
  <A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" SIZE-"l"XB>OTKpbrra« криптографиж/B></FCWTX/A><BR>
  <A HREF="#"XFONT COLOR="#FFFFFF" FACE="Arial" 31гE="1"XB>цифрОВая -OflnMCb</BX/F0NTx/AXBR>
  <A HREF="#"XFONT C0LOR="#FFFFFF" FACE="Arial" 512E="1"xB>электрОННЫЙ r.cKyMeHT</BX/FOKT></A><BR>
  <A HREF="#"XFONT COLOR="#FFFFFF" ?ACE="Arial" SIZE="l"XB>4ero не мо-г/т xaKepbi</BX/FONTX/AXBR>
</TD>
```

Будем действовать по аналогии с первым блоком ссылок. Тоже введем уникальный идентификатор: ID-"aubmenu". Аналогично воспользуемся контекстным селектором и свойством color для задания цвета ссылок:

```
'submenu A {
  color: #FFF}
```

то есть ссылки внутри элемента с ID="submenu" будут белого цвета. А HTML-код второго блока станет таким:

```
<TD WIDTH="142" CLASS="bluecol" ID="suhmanu">
  <A HREF="#"XFONT FACE-"Arial" S:il -"-1"><B>открытая криптогра-фия•:/BX/FONTX/AxBR>
  <A HREF«"#"xFONT FACE="Arial" 312E="1"XB>цифровая под-пись </BX/FONT></A><BR>
  <A HREF="#"XFONT FACE="Arial" 31гE="1"XB>ЭЛектр0нныЙ доку-:-:ент</B></FONTX/A><BR>
  <A fiREF="#"XFONT FACE="Arial" SIZE="l"XB>4ero не могут хаке-zbt</BX/FONTX/AXBR>
</TD>
```

Далее по коду у нас идет заголовок черного цвета:

```
<H3 ALIGN="right"XFONT COLOR="#000000" EACE="Ar1a1">открытая криптогра-фия</FONT></H3>
```

На данном этапе мы не можем совсем избавиться от тега <FONT>, поскольку мы пока не умеем задавать без его помощи гарнитуру шрифта. Но атрибут :OLOR из него убрать уже можем. Допустим, что подавляющее большинство заголовков <нз> будет иметь черный цвет. Тогда в качестве селектора можно нзять название элемента <нз>. Получится такое правило:

```
НЗ {
  color: #000}
```

А из кода беспощадно выкинем атрибут COLOR, и он немного преобразуется:

<H3 ALIGN="right"XFONT ГАСЕ="Ar1a1">открытая криптография</5ШТХ/Н3>

Остался простой текст, который находится в элементе <?>:

<P><FONT SIZE="2" COLOR="#000000" FACE="Verdana">Bbt запускаете...

Стиль на элемент <p> пишется по аналогии с элементом <нз>. Он очень простой:

```
P {
  color: #000}
```

Из кода опять же убираем атрибут COLOR и он станет таким:

<P><FONT SIZE="2" FAC£-"Verdana">Bbi запускаете...

Пока мы больше ничего сделать не можем. Надо сказать, что исходный документ весил 2,54 Кбайт, а после наших преобразований он весит 2,41 Кбайт, но сама таблица стилей — 0,2 Кбайт, так что суммарный вес документа слегка увеличился. Однако мы перенесли в каскадные таблицы стилей только цвет и фон, так что не спешите удиапяться, к тому же документ сам по себе весьма мал, а значительной экономии можно добиться только для больших документов. Теперь перейдем к шрифту.

Шрифт

Наверное, контроль над шрифтом — одна из самых необходимых вешей для **веб-дизайнера.** Язык HTML очень ограничен с этой точки зрения, он позволяет лишь устанавливать начертание шрифта, кое-как определять размеры шрифта, и пару стилевых параметров, таких как насыщенность и наклон. Надо сказать, что стандарт CSS-1 недалеко ушел в этом направлении от HTML: значительно увеличился контроль над размером шрифта и насыщенностью, а также появилась возможность выводить текст милыми прописными буквами — и все.

Пока мы не будем касаться особенностей работы со шрифтом, я лишь перечислю имеющиеся свойства и объясню, что они из себя представляют. Вообще сейчас реачизация поддержки шрифтовых свойств в браузерах очень разная и несовершенная, так что проблем много. Некоторые из них мы решим, некоторые нет, но все это будет во второй части книги, а пока — свойства.

font-family

Задает собственно шрифт. Синонимом понятия *шрифт* является понятие *гарнитура* шрифта. Его я тоже буду иногда использовать, потому что словосочетание *"шрифт шрифта"лишено* смысла, а вот словосочетание *"гарнитуре, шрифта"построчно* благозвучно. Сразу определимся с терминологией, чтобы не возникало разночтений.

Определение

Шрифт (гарнитура) — это набор символов, которые объединены общими стилевыми признаками. Причем совокупность стилевых признаков является уникальной.

: качестве параметра задается имя шрифта или название семейства. Причем :жно указывать несколько имен через запятую, формируя список альтерт пвных шрифтов. Тогда в случае, если на компьютере пользователя не уст:жовлен шрифт, который стоит в списке первым, для отображения текста lei использоваться второй шрифт из списка. Если же и его нет. то **третий** м т. д. Что касается семейств, то они могут быть следующими:

^ serif — шрифты с засечками, такие как Times New Roman и Garamond;

3 sans-serif — шрифты рубленые, такие как Verdana, Arial и Tahoma;

^ monospace — шрифты моноширинные, такие как Courier New;

Э cursive — шрифты курсивные, такие как Zapf-Chancery;

П fantasy — декоративные шрифты.

Последние два семейства используются крайне редко. Надо сказать, что рекомендуется семейство шрифтов указывать в качестве последней альтерн тогда, если на компьютере пользователя не установлен ни один рифт из списка, то будет использован шрифт данного семейства, который данного браузера является шрифтом по умолчанию. Таким образом, текст будет выведен на экран, по **крайней** мере, похожим шрифтом, так что зайн страницы пострадает незначительно.

5 HTML аналогом свойства font-family является атрибут FACE элемента • (табл. 4.9).

Таблица 4.9. Сравнение установки шрифта с помощью HTML и CSS

HTML	CSS
<FXFONT FACE="Verdana, Tahoma, ins-serif">Текст выведенный шриф- ;м Verdana</FONTX/P>	P { font-family: Verdana, Tahoma, sans-serif} . . . <P>Текст выведенный шрифтом Verdana</P>

'ont-style

тает стиль шрифта.

учения:

~3 normal — обычный;

1 italic — курсивный;

3 oblique — наклонный.

Курсивный вариант шрифта должен быть установлен на компьютере пользователя, иначе он будет эмулироваться простым наклоном имеющегося шрифта, т. е. italic будет заменяться oblique.

С помощью данного свойства можно убрать наклон текста внутри элементов <i> и <EM>, так что можно делать совершенно разные виды выделений. Например, выделять текст оранжевым цветом, но без наклона:

```
EM {
  color: orange;
  font-style: normal}
. . .
<P>В данном тексте логическое <EM>вьшеление</EM> Судет обычньы шрифтом
без наклона оранжевого цвета</P>
```

В HTML есть теги <IX/I>, шрифт внутри которых отображается как italic, а по умолчанию шрифт имеет стиль normal (табл. 4.10).

Таблица 4.10. *Сравнение установки стиля i~=~**lie** с помощью **HTML** и CSS*

HTML	CSS
<P><FONT FACE="Verdana, Tahoma, sans-serif"><1>Текст выведенный курсивным шрифтом Verdana</I></PONT></P>	P { font-family: Verdana, Tahoma, sans-ser:f; font-style: italic} . . . <P>Текст выведенный **курсивньм** шрифтом Verdana</P>

font-variant

Позволяет выбирать из двух вариантов вывода шрифта: обычными или малыми прописными буквами.

Значения:

О normal — 06ЫЧНЫЙ Шрифт;

П small-caps — шрифт, в котором все строчные буквы заменены на малые прописные. Малые прописные отличаются от обычных прописных слегка измененными пропорциями и слегка уменьшенным размером.

Вообще малыми прописными можно выводить заголовки:

```
н :-
  font-variant:  small-caps}
```

но это уже зависит от дизайна сайта.

В **HTML** нет атрибута, который бы соответствовал данному свойству, а по умолчанию шрифт выводится как font-variant: **normal.**

Малые прописные можно устанавливать и хгя простого текста, но вообще делать этого не стоит (табл. **4.11).**

Таблица 4.11. Пример установки малых прописных букв

CSS

```
? {
  fant-vai-ant: small-caps}
. . .
<P>Текст, на известно зачем выведенный малыми прописными буквами</P>
```

font-weight

Определяет насыщенность шрифта.

Существует численный ряд для насыщенности шрифта: 100, 200, 300, 400, 500, 600, 700. 800, 900. Сразу скажу, что в браузерах такой богатой градации нет, **так** что приходится довольствоваться малым. Кроме того, двум числам лз данного ряда соответствуют ключевые слова:

П **400** — normal (нормальный);

3 700 — bold (полужирный).

Поэтому правила

```
P {
  font-weight: 700}
```

И

```
:-{
  font-weight: bold•
```

.овершенно эквивалентны.

Есть еще два ключевых слова, которые задают насыщенность относительно предка:

Т lighter — насыщенность шрифта будет меньше, чем у предка;

1 bolder — насыщенность шрифта будет больше, чем у предка.

3 следующем примере шрифт в абзаце будет полужирным, а шрифт между **гегамн** <вх/в> полужирным с еще большей насыщенностью:

```
= {
  font-weight: bold}
```

в I

```
  font-weight: bolder|
. . .
<P>Текст полужирным шрифтом с волшей <B>насыщенностью</B></P>
```

В HTML есть тег <в>, который соответствует значению font-weight: bold (табл. 4.12), По умолчанию шрифт имеет нормальную насыщенность.

Таблица 4.12. *Сравнение установки насыщенности в HTML и CSS*

HTML	CSS
<P><B>Текст полужирным шрифтом в абзаце</Вж/P>	P { font-weight: bold} . . . <P>Текст полужирньы шрифтом в аб-эаце</P>

font-size

Определяет размер (кегль) шрифта.

Существует множество способов задать размер шрифта. Первый — с помощью ключевых слов:

О xx-smali;

П x-small;

О small;

О medium',

• large;

• x-large;

П xx-large.

Какой величине соответствует каждое ключевое слово — зависит от браузера и от операционной системы. Нет однозначного стандарта, и по этой причине использовать ключевые слова достаточно сложно. Например, в Interne: Explorer 5.0 ключевое слово medium соответствует размеру 13,5 пунктов, т. е правила

```
P {
  font-size: medium)
И
P {
  font-size: 13.5pt)
```

-авнозначны. Подробнее о ключевых словах мы поговорим в *главе* 7. На с. 4.5 показано, как выглядит набор данных ключевых слов в браузере :ernet Explorer 5.0.

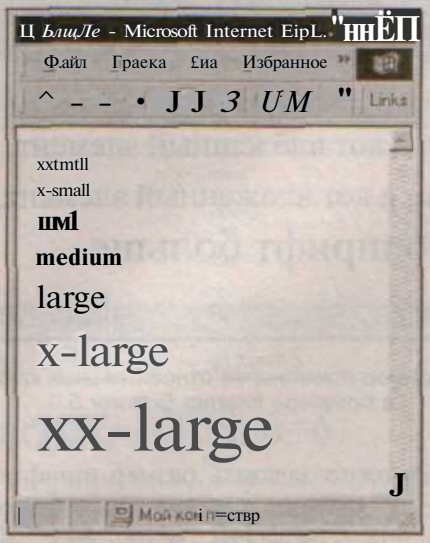


Рис. 4.5. Ключевые слова для задания размера шрифта в браузере Internet Explorer 5.0 при установленном размере шрифта в самом браузере **Средний**

х>ме того, есть два ключевых слова, которые позволяют указывать размер рифта относительно родительского элемента:

- ! smaller — шрифт будет меньше, чем у родительского элемента;
- larger — шрифт будет больше, чем у родительского элемента.

от пример применения относительных ключевых слов для установки раз- :ра шрифта:

```
{
  font-size:  large|
ипаП {
  font-size: smaller)
.arge {
  font-size I larger)
. .
">Данный абзац является родительским элементом, а вот вложенный элемент,
"PAN CLASS-"srt\all">у которого шрифт МеНbaie</SPAN>, а вот вложенный эле-
;нт, <SPAN CLASS="large">у которого шрифт Сольше</SPAN></P>
```

В браузере это будет выглядеть так, как показано на рис. 4.6.

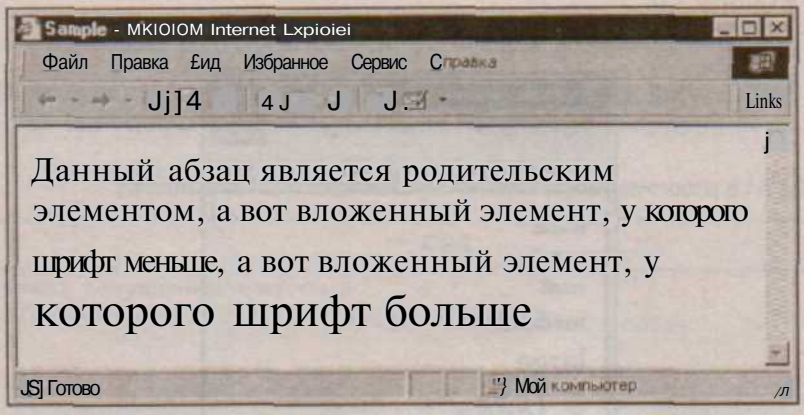


Рис. 4.6. Пример применения относительных ключевых слов в браузере Internet Explorer 5.0

Кроме ключевых слов можно задавать размер шрифта в любых доступных единицах длины: пунктах, дюймах, пикселах, сантиметрах и миллиметрах.

В HTML размер шрифта можно задавать с помощью атрибута SIZE элемента <FONT>, однако размерность величин несколько иная. В HTML размер шрифта задается цифрами от 1 до 7, а также относительно базового шрифта, например +1 или —2. Для сравнения взгляните на пример, показанный на рис. 4.7: в левой колонке размер шрифта задан с помощью CSS с использованием ключевых слов, а в правой колонке — с помощью атрибута SIZE элемента <FONT> в браузере Internet Explorer 5.0.

Как видите, размер шрифта, заданного с помощью ключевых слов CSS, совпадает с размером шрифта, заданного с помощью HTML. Сравнение размеров шрифтов, задаваемых с помощью HTML и CSS для браузера Interne: Explorer 5.0, приведено в табл. 4.13.

Таблица 4.13. *Сравнение размеров ключевых слов для размера шрифта в CSS со значениями атрибута SIZE в HTML*

CSS	xx-small	x-small	srr;all	medium	large	x-large	xx-large
HTML	1	2	3	4	5	6	7

Надо отметить, что в HTML базовым является размер 3, тогда как в CSS базовым является значение medium, а в таблице данному значению соответст-

лет размер 4, т. е. в HTML ось размеров шрифтов несимметрична, а в rSS — симметрична. Несоответствие совершенно очевидно и это приводит с некоторой нелогичности.

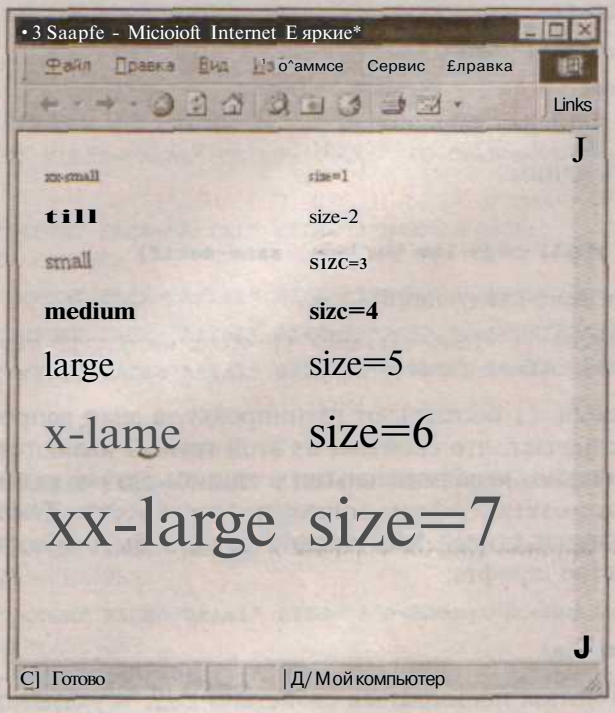


Рис. 4.7. Сравнение ключевых слов для задания размера шрифта в CSS с размерами шрифта в HTML в браузере Internet Explorer 5.0

~.сь я пока только обозначил проблему, а как ее решают и можно ли се .ижть вообще, расскажу несколько позже. В табл. 4.14 приведено сравне- ие установки размера шрифта средствами **HTML** и CSS.

Таблица 4.14. *Сравнение установки размера шрифта в HTML и CSS*

-TML	CSS
<P><FONT SIZE=4>Текст со стандарт-ным размером шрифта</FONT></P>	P { font-size: medium} . . . <P>Текст со стандартным размером шрифта</P>

Как и в случае с фоном, для всех свойств шрифта существует сокращенная запись, в которой можно объединить все вышеперечисленные свойства. Например, следующий набор объявлений

```
P {
  font-style: italic-
  font-variant: small-caps;
  font-size: lem;
  font-family: Verdana, sans-serif•
```

можно заменить одним:

```
P <
  font; italic small-caps lem Verdana, sans-serif)
```

Формат правила **font** следующий:

```
font: [<font-style> Ii <font-var:ant> || <font-weight>]? <font-size>
[/ <line-height>]? <font-faic:ly>
```

Квадратные скобки ;] обозначают группировку, а знак вопроса после данной группы обозначает, что свойства из этой группы являются необязательными. Следовательно, необязательными в данном случае являются свойства font-style, font-variant, font-weight и line-height. Таким образом, для сокращенной записи стилей хля шрифта должен быть обязательно указан размер и семейство шрифта:

```
P {
  font: lem Verdana)
```

Запомните, что нельзя пользоваться свойством **font** и указывать только размер или только гарнитуру шрифта. Это значит, что следующие правила некорректны:

```
P {
  font: Verdana, sans-serif-
P {
  font: lem)
```

Здесь мы впервые встретились со свойством line-height, которое позволяет задавать расстояние между строчками. Подробнее с данным свойством мы познакомимся позже. Вот и все, что пока надо знать о шрифте. Кстати, самое время сделать второе лирическое отступление.

Показательные выступления (Часть II)

А теперь вернемся к нашему HTML-локументу. для которого мы начали писать таблицу стилей. Сейчас мы можем перенести в каскадные таблицы стилей все, что связано со шрифтом. Сначала займемся ссылками. Ранее мы,

е перенесли в стили цвет ссылок, так что у нас на ссылки уже написаны .ис правила:

```
.inmenu A (
olor: #000)
.nmenu A {
rjlor: #FFF}
```

.од двух блоков со ссылками будет таким:

```
WIDTH="172" CIASS="back" ALIGN="right" :D-"mainmenu">
BRXBR>
A H?ZF="="XF0NT FACE="Arial" SIZE="3"XB>_" компа-
.</BX/F0NTX/A>-
A HREF-"=" <FONT FACE="Axial" SIZE="3"XB>";j;^y:--.r-::</BX/FONTX/A><BR>
A HREF="#"XPONT FACE="Araal" SIZE="3"XB>_ __-.</BX/PONTX/A><BR>
<A HREF="#"><FONT FACE="Axial" SIZE="3XB>:-^rr::</BX/F0NTX/A>
</TD>
WIDTH="142" CIASS="bluecol" ID="sutmenu">
Л HREF="-TXFONT FACE="Arial" SIZE="l"XB>iTKpbrraH криптогра-
</BX/F0NTX,AXBR>
A HREF="fTXFONT FACE-"Arial" 51rE-"1"XB>цнфровая под-
.i</BX/FOHTX/AXBR>
A HREF="="><FONT FACE="Arial" 312E="1"XB>электронный ДОКу-
r</BX:/F0NTX/A><3R>
-A HREF="#"XFONT FACE="Arial" SIZE="l"XB>4ero не могут хаке-
</BX/FONTX/AXBR>
</TD>
```

я главной панели ссылок нам нужен **полужирный** шрифт Arial размером _.:E="3", которому соответствует ключевое слово **small, в** для второй **пане-** ссылок— полужирный шрифт Arial размером **SIZE-"1",** которому соот- **гтетвуст** ключевое слово xx-small.

Для установки параметров шрифта воспользуемся сокращенным свойством -t. Согласно его формату, сначала надо указать насыщенность, затем раз- р шрифта, а в конце гарнитуру шрифта. Таким образом, правила допол- •тся объявлениями

```
font: bold small Arial
font: bold xx-small Arial
```

станут такими:

```
-ainmenu A {
;olor: #000;
font: bold small Arial'
```

```
#submenu A {
  color: #FFF;
  font: bold xx-small Arial}
```

А из HTML-кода наконец-то к всеобщему ликованию совершенно и окончательно исчезнут теги <FONT> И <B>:

```
<TD WIDTH="172" CLASS="back" ALIGN-"right" ID-"mainmenu">
  <BR><BR>
  <A HREF="#">0 KOMnaHMH</A><BR>
  <A HREF="#">npOflyKTbi</A><BR>
  <A HREF="#">PemeHMH</A><BR>
  <A HREF="#">CTaTbM</A>
</TD>
<TD WIDTH-"142" CLASS="bluecr_l" ID="submenu">
  <A HREF="#">OTKp*jrafl криптография</A><1.R>
  <A HP£P="#">цифровая подпись</A><BЙ>
  <A HP^="#">электронный документ</A><BR>
  <A HREF="#">4ero не могут xaKepbi</A><BR>
</TD>
```

Далее идет заголовок

```
<H3 ALIGN="right"XFONT ГАСЕ="Аг1а1">открытая криптография</ГСМТХ/Н3>
```

с таким стилем:

```
H3{
  color: #000)
```

Нам надо перенести в стиль только гарнитуру шрифта. Для этого воспользуемся свойством font-family:

```
H3 {
  color: #000;
  font-family: Arial)
```

А из кода снова уберем тег <F~NT>:

```
<H3 ALIGN="right">OTKpbiraH кркптография</H3>
```

Остался простой текст, который находится в элементе <p>:

```
<PXF0NT SIZE="2" FACE="Verdana">Вы запускаете...
```

Снова воспользуемся сокращенным свойством **font.** Размеру шрифта в **HTML** SIZE="2" в CSS соответствует значение **x-small.** С учетом этого правило на элемент <?> станет таким:

```
P (
  color: #000;
  font: x-small Verdana}
```

А код станет совсем простым и чистым:

<?>Вы запускаете...

На данном этапе можно полвести промежуточный итог. Код всей страницы вместе со стилями будет следующим:

```
^'DOCTYPE HTOL PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  • HEAП>
    <TITLE>ЭНИГМА — криптографические с;редства защиты информации</T1TXE>
    <STVLE TYTE •"text/ces">
      E :
        colo r: #000;
        font: x-small \ferdana)
      BODY {
        background: #FFF}
      H3 {
        color: #000,-
        font-family: Arial}
       -back {
        backgr: und: url(img/fragme Dt_3.gif) no-repeat}
      #mainmenu -- /
        color: #000;
        font: bold small Arial}
      #submenu A {
        color: IFFF;
        font: bold xx-small Arial}
       .bluecol {
        background: #1D3D4E}
    </STYLE>
      >
  <BODY TOPMARGIN-"0" LEFTMARGIN="0" MARGINHEIGHT="0"

    <TABLE CELLPAD:ING-"0" CE [_5PACING="0"
          WIDTH="750" BORDER="0" HEIGHT="157">
    <TR>
      <TD WIDTK="82"> </TD>
      <TD WIDTH"M266" COLSPAN="2"XIMG SRC="img/f ragment_1 .gif"
WIDTH-"266" HEIGHT="86" ALT="3HHTWA - криптографические средства защиты информации [логотип] "x/TD>
```

```
<TD WIDTH="251" ROWSPAN="2"XIMG SRC="iing/fragment_2.gii"
WIDTH="251" HEIGHT="167" АЬТ="ЭНИГМА - быстрый путь к надежности'-х/ТО
  <TD> </TD>
</TR>
<TR>
  <TD WIDTH-"82">Snbsp;</TD>
  <TD WIDTH="107" HEIGHT="81">Snbsp;</TD>
  <TD WIDTH-"159" CLASS="bluecol">Snbsp;</TD>
  <TD>4nbsp;</TD>
</TR>
</TABLE>
<TABLE CELLPADDING="10" CELLSPACING="O"
      WIDTH="750" BORDER="0" HEIGHT="400">
<TR VALIGN="top">
  <TD WIDTH="172" CLASS="back" ALIGN=-"right" ID="mainmenu">
    <BR><BR>
    <A HREF="#">0 KOMnaHMM</AXBR>
    <A HREF="#">npoflyKTbi</AXBR>
    <A HREF="#">PemeHHfl</A><BR>
    <A HREF="#">OraTbii</A>
  </TD>
  <TD WIDTH="142" CLASS="bluecol" ID="submenu">
    <A HREF="#">OTKpbrrafl криптография</A><BK>
    <A HP£E="#">цифровая подпись</A><BR>
    <A HREF="#">электронный документ</A><BK>
    <A HREF="#">4ero не могут хаКерbi</A><BR>
  </TD>
  <TD WIDTH="392">
    <H3 ALIGN="right">OTKpbrrafl криптогр=:ия</H3>
    <P>Вы запускаете браузер, набираете в адрес.-:ой строке URL сайта и
получаете желаемую страницу. Вот, вку атце, действия пользователя, котэрый
Ородит в сети Интернет. Он не задумывается, что лежит за всем этим. Ему
это не нужно. Но это нужно нам, профессиональным веб-разработчик ш, чтобы
максимально удовлетворить пользовательские запросы.*/P>
    <P>Мы не будем углубляться в историю, и отряхивать r^-ъ с =;:жив-
ных 90-х годов, но базовые понятия рассмотрим. И:ак, что же лежит в :сно-
ве всемирной сети Интернет? Базовыми являются три тех;.олоiи :</P>
  </TD>
</TR>
```

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

Обший вес документа стал равен 2,28 Кбайт, а в начале работы он составлял 2,54 Кбайт. Как видите, уже сейчас лаже для такого маленького документа мы добились существенного сокращения кода (на 10%). Надо сказать, что для дальнейшего сокращения кола нам понадобится отойти от табличной верстки, т. е. перейти к позиционированию элементов с помощью CSS, а это достаточно сложный и объемный вопрос. Так что мы его отложим до *главы* 9, а код пока оставим таким, какой он есть.

Свойства текста

Информация на HTML-страницах обычно представлена в текстовом виде, так что контроль над текстовыми блоками чрезвычайно важен для удобного представления информации. В HTML текст можно выравнивать по горизонтали и вертикали с помощью атрибутов ALIGN и VALIGN, а также применять некоторые элементы оформления, такие как перечеркнутый и подчеркнутый текст. В HTML нельзя рейдировать трекинг. расстояние между словами, высоту строки и абзацный отступ, но все это можно делать при помощи каскадных таблиц стилей.

word-spacing

Позволяет устанавливать интерваты между словами.

Любое значение с размерностью пины, как положительное, так и отрицательное. При отрицательном значении слова могут накладываться друг на друга, так что при его использовании надо быть осторожным.

В HTML аналогов данного свойства нет (табл. 4.15).

Таблица 4.15. Пример установки интервала между словами

CSS

```
p {
                em}
  word-spacing:  3
}
```

```
<?>В этом абзаце расстояние между словами будет большое.</P>
```

Пример использования свойства word-spacing показан на рис. 4.8.

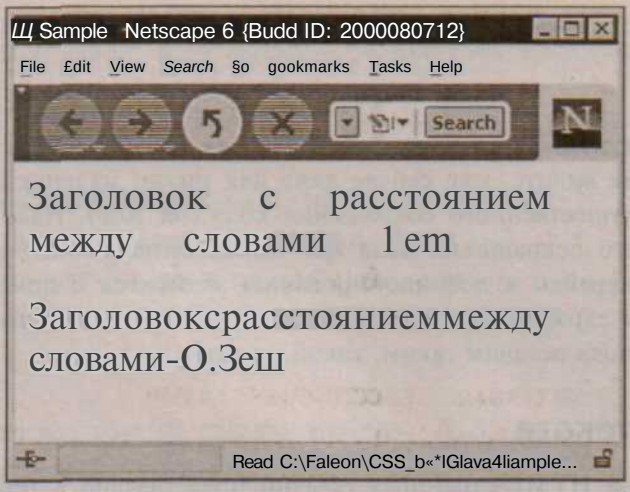


Рис. 4.8. Пример использования свойства **word-spacing**. Ранняя версия браузера Netscape 6.x

letter-spacing

Позволяет устанавливать величину трекинга (иначе говоря, расстояние между буквами). Подробно с трекингом мы познакомимся в *главе 7,* а пока только описание свойства.

Любое значение с размерностью длины, как положительное, так и отрицательное. При отрицательном значении буквы **могут** накладываться друг на друга, так что при его использовании надо быть крайне осторожным, а то текст может стать совершенно "нечитабельным".

В HTML аналогов данного свойства нет (табл. 4.16).

Таблица 4.16. Пример установки трекинга

CSS
p {
letter-spacing: 3ет}
<P>В этом абзаце расстояние между буквами будет больше</?>

Пример использования свойства **letter-spacing** показан на рис. 4.9.

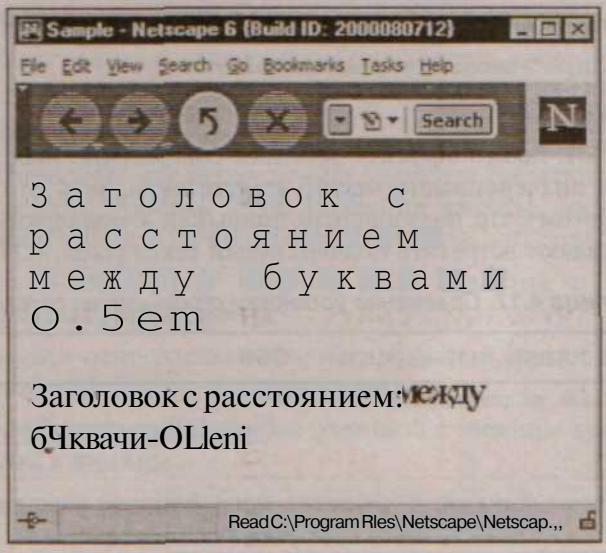


Рис. 4.9. Пример использования свойства letter-spacing. Ранняя версия браузера Netscape 6.x

text'decoration

Позволяет оформлять текст.

Значения:

- none — выводит обычный текст без всякого оформления;
- underline — выводит подчеркнутый текст;
- overline — выводит "надчеркнутый" текст (то есть с чертой сверху);
- line-through — выводит перечеркнутый текст;
- blink — выводит мерцающий текст.

В HTML есть теги с аналогичным эффектом:

- underline — <U>;
- line-through - <S>.

Есть еще тег <BLINK>, но он поддерживается только браузером Netscape Navigator. Свойство text-decoration обычно используют для того, чтобы сделать ссылки неподчеркнутыми. Делается это совсем просто. Берем селектор ссылки с псевдоклассом и прописываем ему объявление text-decoration: none:

```
A:link, A:visited, A:active {
  text-decoration: none)
```

В HTML-коде тогда ничего не надо менять. Например, ссылка

```
<A HREF="inciex.html">HenoOTepKHyTafl ссьшка</A>
```

будет неподчеркнутой.

Средствами HTML такого эффекта добиться совершенно невозможно. А для простого текста подчеркивание можно осуществить, хотя это очень порочная практика, потому что пользователи привыкли к подчеркнутым ссылкам, но никак не ожидают встретить подчеркнутый текст (табл. 4.17).

***Таблица** 4.17. Сравнение установки оформления текста в HTML и CSS*

HTML	CSS
<P><и>Непривычнкй подчеркнутый текст</и></P>	P { text-decoration: underline} . . . <P>Непривычный подчеркнутый текст</P>

Пример различным образом оформленного текста приведен на рис. 4.10.

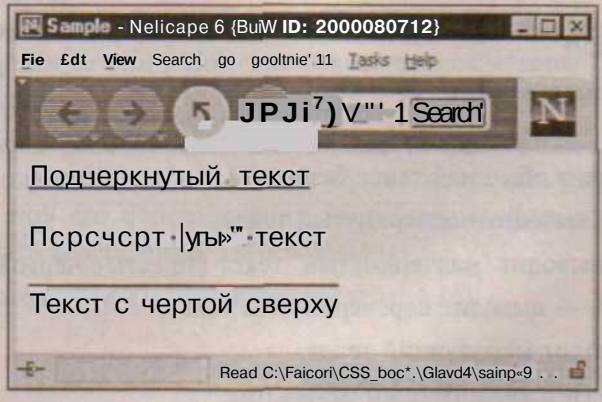


Рис. 4.10. Пример различного оформления текста. Ранняя версия браузера Netscape 6_x

vertical-align

Позволяет устана&тивать вертикальное выравнивание текста.

Вертикальное выравнивание может быть относительно родительского элемента и относительно форматируемой линии. При выравнивании относительно родительского элемента могут быть следующие значения:

П baseline — совмещает среднюю линию элемента со средней линией родительского элемента;

O sub — делает элемент подстрочным (аналог тега <SUB>);

0 super — делает элемент надстрочным (аналог тега <SUP>);

1 t.xt-top — совмещает верхний край элемента с верхним краем шрифта родительского элемента;

"I middlr, — совмещает среднюю линию элемента (обычно изображения) с)ровнем "средний уровень плюс половина высоты x-height родительского элемента";

G t:.xt-bottom — совмещает нижний край элемента с нижним краем шрифта родительского элемента.

При выравнивании относительно форматированной линии могут быть следующие значения:

O **boti** .om— совмещает нижний край элемента с нижним краем самого низкого элемента в линии;

• **top** — совмещает верхний край элемента с верхним краем самого высокого элемента в линии.

Кроме того, в качестве значения можно использовать процентные соотношения.

В HTML кроме тегов <SOB> и <SOP> нет способов для вертикального форматирования текста, однако для подстрочных и надстрочных элементов лучше пользоваться именно ими, а не средствами CSS (табл. 4.18).

Таблица 4.18. *Сравнение установки нижнего индекса в HTML и CSS*

HTML	CSS
<P>*Ъорыуам* i оды: H<SUB>2</SU3>O</P>	.sub t vertical-align: sub} <P>Формула волы: H<SPAN CLASS="sub">2</SPAN>O</P>

text-align

Позволяет устанавливать горизонтальное выравнивание текста внутри элемента.

Значения:

• left — выравнивание по левому краю **элемента;**

П center — выравнивание по центру элемента;

O right — выравнивание по правому краю элемента;

O justify — выравнивание по ширине элемента.

В **HTML** аналогом данного свойства является атрибут ALIGN, который может принимать совершенно аналогичные значения. Так что в данном случае использование CSS особого преимущества не дает. Но в некоторых случаях оно есть и достаточно ощутимое. Если у вас на странице несколько абзацев текста, и вы хотите выровнять их все по ширине, то гораздо лучше использовать CSS. В табл. 4.19 приведен пример подобного форматирования текста.

Таблица 4.19. *Сравнение установки выравнивания текста в HTML и CSS*

HTML	CSS

`<P ALIGN="ju5tify">AC3au текста, выровненный по ширине элемента P.</P>`	`I {`
	`text-align: justify}`
	
	`<P>ACзац текста, выровненный по ширине элемента P.</P>`

text-transform

Позволяет трансформировать имеющийся текст.

Значения:

- capitalize — первая буква каждого слова в элементе станет прописной;
- uppercase — все слова в элементе будут выведены прописными буквами;
- lowercase — все слова в элементе **будут** выведены строчными буквами;

О попе — текст в элементе выводится без трансформаций (значение по умолчанию).

В **HTML** аналогов данному свойству нет (табл. 4.20).

Таблица 4.20. *Пример установки прописных букв*

CSS
`P {` `text-transform: capitalize}`
`<P>первые буквы в словах будут прописными</P>`

text-indent

Позволяет устанавливать величину отступа перед первой строкой абзаца.

Любое значение с размерностью длины, как положительное, так и отрицательное. Кроме того, можно использовать проценты, которые вычисляются относительно ширины родительского элемента.

В HTML аналогов данного свойства нет (табл. 4.21).

Таблица 4.21. *Пример установки абзацного отступа*

CSS

```
P {
  text-ir.isrt: 20px)
. . .
<P>Отступ перед первой строкой будс равен 20 пикселам.</P>
```

line-height

Позволяет устанавливать величину расстояния между строками.

Любое значение с размерностью длины, но только положительное. Можно использовать проценты, которые вычисляются относительно размера шрифта данного элемента. Кроме того, можно использовать безразмерные числа — в этом случае высота строки определяется как высота шрифта, умноженная на данное число. Кстати говоря, безразмерные числа являются оптимальным вариантом лля установки высоты строки.

В HTML аналогов нет (табл. 4.22).

Таблица 4.22. *Пример установки высоты строки*

CSS

```
p (
  font-size: 14px;
  line-height: 1.5)
. . .
<P>В этом абзаце расстояние межяу строками будет равно
14px*1.5=21px</P>
```

Вообше в спецификацию CSS-1 входят еще свойства, касающиеся блоков. Блоковая модель будет подробно рассмотрена в *главе 8,* однако некоторые свойства я приведу уже сейчас. Для понимания этих свойств необходимо знать, что такое блок. Вообще любой элемент на странице представляет собой блок. Структура блока показана на рис. 4.11.

Блок состоит из так называемой контентной части, которая *может быть* окружена отступами (padding), рамками (border) и полями (margin). Причем и отступы, и рамки, и поля могут быть разными с четырех сторон (сверху, справа, снизу и слева). Каждый блок может иметь фиксированную ширину (width) и высоту (height).

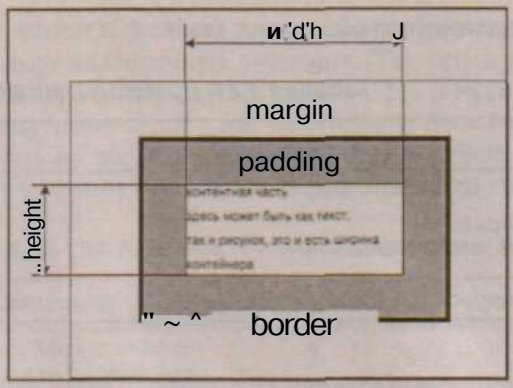


Рис. 4.11. Структура блока в CSS

Ниже приведены свойства, которые задают параметры отступов, рамок и полей. Подробнее о них мы будем говорить в *главе 8,* а пока краткое описание с небольшими примерами.

margin-top, margin-right, margin-bottom и margin-left

Позволяют установить величину *верхнего, правого, нижнего* или *левого* поля элемента соответственно.

Значения:

П любое значение с размерностью пины, как положительное, так и отрицательное;

- проценты, которые вычисляются относительно ширины родительского элемента;
- ключевое слово auto.

В HTML аналогов данного свойства нет (табл. 4.23).

Таблица 4.23. Пример установки полей

CSS

```
? '
  margin-top: Юpx;
  margin-bottom: Юpx;
  margin-left: 20px;
  margin-right: 5px}
. . .
<P>Блоковый элемент с верхним и нижними полями в 10 пикселов, левым
полем Б 20 пикселов, а правым полем в 5 пикселов.</?>
```

margin

Позволяет в сокращенной форме задавать величины всех полей элемента. Значения совершенно аналогичны возможным значениям у предыдущих свойств.

Формат у данного свойства достаточно разнообразный. Можно устанавливать все четыре значения, можно только одно, тогда данное значение будет применено ко всем четырем полям. Можно два значения, тогда первое из них будет применено к верхнему и нижнему полям, а второе — к левому и правому полям. Можно три. тогда первое значение будет применено к верхнему полю, второе — к правому и левому полям, а третье — к нижнему. Если какое-либо значение пропущено, то оно берется равным значению с противоположной стороны. Звучит совершенно непонятно и крайне запутанно, так что в табл. 4.24 приводятся варианты сокращенной записи.

Таблица 4.24. Примеры сокращенной записи стилей для полей элемента <p>

Полная форма	Сокращенная форма
P { margin-top: 10px; margin-bottom: 10px; margin-left: 20px; margin-right: 5px}	P { margin: 10px 5px 10px 20px}
P { margin-top: 1em; margin-bottom: 1em; margin-left: 1em; margin-right: 1em}	margin: 1em}
P { margin-top: 20%; margin-bottom: 20%; margin-left: 10%; margin-right: 10%}	P { margin: 20% 10%}
P { margin-top: 0.5em; margin-bottom: 2em; margin-left: 3em; margin-right: 3em}	P { margin: 0.5em 3em 2em}

padding-top, padding-right, padding-bottom и padding-left

Позволяют установить величину *верхнего, правого, нижнего* или *левого* отступа элемента соответственно.

Любое значение с размерностью длины, но только положительное, а также проценты, которые вычисляются относительно ширины родительского элемента.

В HTML аналогов данного свойства нет (табл. 4.25).

Таблица 4.25. *Пример установки отступов*

CSS

```
P {
  padding-top: 5px;
  padding-bottom: 5px;
  padding-left: 20px}
. . .
<P>Блоковый элемент с верхним и нижними отступами 5 пикселов, левым
отступом в 20 пикселов и правым отступом в 0 пикселов.</P>
```

padding

Позволяет в сокращенной форме задавать величины всех отступов элемента. По своему формату совершенно аналогично свойству margin.

border-top-width, border-right-width, border-bottom-width и border-ieft-width

Позволяют устанавливать ширину *верхней, правой, нижней* или *левой* рамки соответственно.

Любое положительное значение с размерностью **длины.** Кроме того, можно использовать ключевые слова:

- thin — тонкая рамка;
- medium — средняя по ширине рамка;
- thick — широкая рамка.

Численные значения ключевых слов зависят от конкретного браузера, так что лучше указывать ширину рамки непосредственно в единицах длины.

В HTML рамки есть только у рисунков и таблиц, где их ширина задается с помощью атрибута BORDER, тогда как с помошью CSS рамку можно создать вокруг любого элемента. Однако чтобы создать рамку, недостаточно указать только ее ширину, надо также указать хотя бы стиль рамки с помошью свойства border-style. В табл. 4.26 показано, как можно сделать рамку вокруг рисунка.

*Таблица **4.26.** Сравнение установки рамок для рисунка в HTML и CSS*

HTML	CSS

`<IMG SRC="i/logo.gif" WIDTH=200 HEIGHT»!00 BORDER=1>`	`IMG {`
	`  border-style: solid;`
	`  border-top-width: 1px;`
	`  border-bottom-width: 1px;`
	`  border-right-width: 1px;`
	`  border-left-width: 1px}`
	`. . .`
	`<IMG SRC="i/logo.gif" WIDTH=200 HEIGHT=100>`

border-width

Позволяет в сокращенной форме задавать ширину всех рамок. Значения соиершенно аналогичны возможным значениям у несокращенной формы записи ширины рамок. Формат такой же, как у свойств margin и padding, т. е. правила, приведенные в левом и правом столбцах табл. 4.27, одинаковые.

***Таблица 4.27.** Формы записи ширины рамок для элемента <p>*

Полная	Сокращенная

`P (`
`  border-top-width: 1px;`
`  border-bottom-width: 2px;`
`  border-right-width: 3px;`
`  border-left-width: 4px}`
	`P {`
	`  border-width: 1px 3px 2px 4px>`

border-color

Позволяет устанавливать цвет рамок.

Любые цветовые значения.

Заметьте, что если цвет рамки не указан явно с помощью свойства border-color, то рамка будет иметь цвет, установленный с помощью свойства color. В следующем примере рамка будет зеленого цвета:

```
P {
  color: green;
  border-width: lpx;
  border-style: dotted}
```

В HTML аналогичного данному свойству атрибута нет. А в CSS можно задавать как цвет всей рамки в целом, так и цвет отдельных частей рамки (табл. 4.28). Формат аналогичен формату свойства border-width.

Таблица 4.28. Пример установки рамок

CSS

```
p I
   border-width: lpx;
   border-color: #000 #FFF
. . .
<P>Правая и левая рамка будут белыми, а верхняя и нижняя - черными</P>
```

border-style

Позволяет устанавливать стиль рамки.

Значения:

- none — рамки отсутствуют вовсе (является значением по умолчанию);
- dotted — рамка в виде пунктирной линии;
- dashed — рамка в виде прерывистой линии;
- solid — рамка в виде сплошной линии;
- double — рамка в виде двойной линии (ширина линий и расстояния между ними напрямую зависят от значения свойства border-width);
- groove — трехмерная углубленная рамка;
- ridge — трехмерная выпуклая рамка;
- inset — рамка в виде углубления;
- outset — рамка в виде выпуклости.

Как видим, возможных значений много. Однако многие из них не поддерживаются некоторыми браузерами. В частности браузер Internet Explorer 5.Y отображает все рамки так, словно у них border-style: solid.

На рис. 4.12 показано, как выглядят разные стили рамок в браузере Netscape 6.x

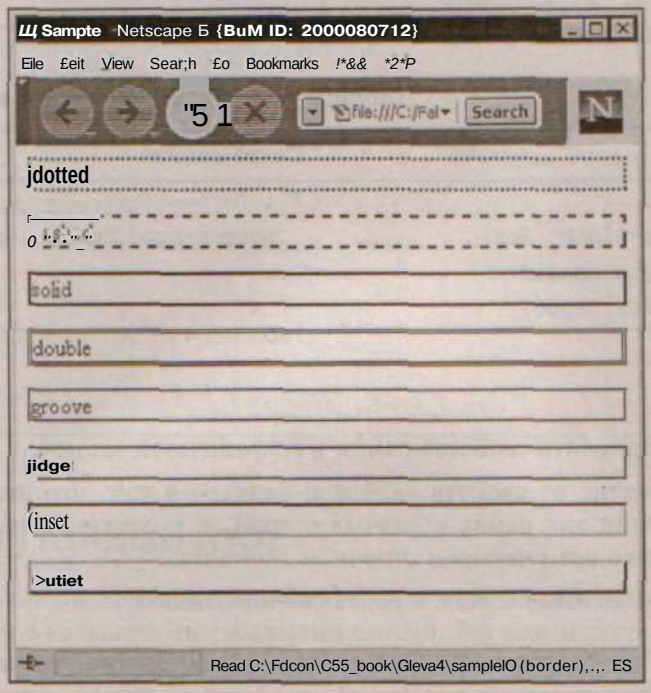


Рис. 4.12. Различные виды рамок в ранней версии браузера Netscape 6.x

В HTML аналогов данному свойству нет (табл. 4.29).

Таблица 4.29. Пример установки стиля рамок

CSS

```
P {
  border-color: #000;
  border-width: 3px;
  border-style: double1
. . .
<P>АСзац с двойной черной рамкой, каждая из которых шириной 1 пиксел
и расстояние между рамками тоже 1 пиксел.</P>
```

border-top, border-right, border-bottom и *border-left*

Позволяют установить свойства *верхней, правой, нижней* или *левой* рамки соответственно в сокращенной записи.

Формат свойств следующий:

border-top: <border-top-width> I| <border-style> I I <border-color>

Правила, приведенные в табл. 4.30, совершенно идентичны.

*Таблица **4.30.** Формы записи для верхней рамки*

Полная	Сокращенная
P { border-top-width: 2px; border-style: solid; border-color: #000}	P { border-top: 2px solid #000}

border

Позволяет установить свойства рамки в сокращенной записи.

Основное отличие от свойств margin и padding в том, что нельзя задавать разные значения для рамок с разных сторон, т. е. устаноаленные значения применяются ко всем четырем сторонам элемента.

Правила, приведенные в табл. 4.31. также идентичны.

***Таблица 4.31.** Формы записи установки рамок для элемента <p>*

Полная	Сокращенная
P { border-top: 2px solid #CCC; border-right: 2px solid #CCC; border-bottom: 2px solid #CCC; border-left: 2px solid #CCC}	P { border: 2px solid #CCC}

Как вы уже знаете, в HTML рамки можно указывать только в таблицах и рисунках **с** помощью атрибута BORDER. Поэтому если возникала потребность сделать рамку вокруг элемента, то элемент неизбежно заключался в таблицу. Причем если надо было сделать рамку шириной в I пиксел, то приходилось делать вложенные таблицы. Цвет рамки устанавливался с помощью фонового цвета первой таблицы, который проглядывал через щели между ячейками второй таблицы. Очевидно, что аналогом свойства border-width в HTML **являлся** атрибут CELLSPACING, а аналогом свойства border-color — атрибут BGCOLOR. Естественно, это приводило к значительному увеличению размера кода. При использовании CSS рамку можно легко создать вокруг любого элемента. Примеры представлены в табл. 4.32.

Таблица 4.32. *Сравнение установки рамок для абзаца текста в HTML и CSS*

HTML	CSS
<TABLE BGCOLOR='#000000" CELLSPACING="0" CSLLPADDING-"0" WIDTH="10O%"> <TRXTD> <TABLE CELLSPACING-"]." CELLPADDING="O" WIDTH="100%"> <TR><TD BGCOLOR="#FFFFFF"> Абзац взят в черную рамку шириной 1 пик-сел </TD></TR> </TABLE> </TD></TR> </TABLE>	P { border: lpx solid #000} . . . <P>A0зац взят в черную рамку шириной 1 пиксел</P>

Промежуточный итог

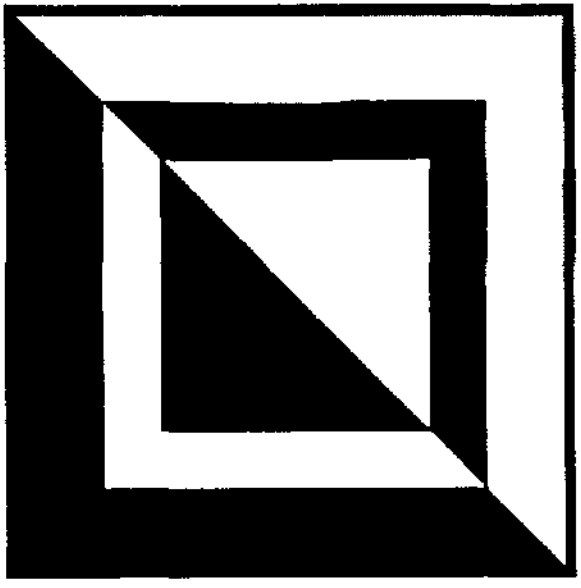
На данном этапе вы должны быть уже хорошо знакомы с языком HTML и в какой-то мере с каскадными таблицами стилей. Вы уже можете в некоторой степени манипулировать цветами, фоном, шрифтом и текстом на странице. Это совершенно необходимая база, без которой невозможно двигаться дальше: углубляться в детали, исследовать различия в реализации поддержки CSS браузерами, выявлять скрытые особенности CSS.

Вы уже должны были понять некоторые преимущества использования CSS: 1егкость кода и простоту' внесения изменений. Однако это только начало. Изучая следующую часть книги, вы почувствуете еще и мощь использования каскадных таблиц стилей. Язык HTML и близко не имеет такого контроля над визуальным представлением страницы, его возможности чрезвычайно узки и ограничены.

Итак, во второй части книги:

- вы научитесь правильно использовать CSS, максимально оптимизируя код;
- вы узнаете все о различиях поддержки CSS в браузерах;
- вы научитесь верстать сайты без применения таблиц;
- вы заглянете в будущее веб.

Первая часть есть *Основа,* во второй части вы приобретете *Мастерство.*



ЧАСТЬ II

CSS: НА ПУТИ К МАСТЕРСТВУ

Глава 5

CSS: правильное использование — залог успеха

Чачинаем свой путь к мастерству. Он будет достаточно долгим и непро-
гым, но вы справитесь, если у вас есть цель. По крайней мере, к этому мо-
ну вы уже обладаете определенным багажом знаний, необходимым для
рименения на практике. Однако использовать CSS можно по-разному.
ель данной главы — научить вас применять CSS *правильно*. Поверьте, это
е так просто, как кажется на первый взгляд. Ведь можно просто особо не
щумываясь написать таблицу стилей с множеством классов и ненужных
явлений, которая будет корректно работать. Но такая таблица стилей бу-
г *неоптшальной* и *неудобной*. Неоптимальной потому, что ее можно значи-
тельно сократить и уменьшить объем, что немаловажно. Неудобной потому,
о разбираться в неупорядоченном нагромождении классов достаточно
ожно. Вообще жестких правил при написании таблицы стилей нет, пото-
у что написать стиль можно по-разному и это будет работать, однако есть
ножество рекомендаций. Кроме того, можно все же выделить одно главное
хшило, которое всегда надо применять при использовании CSS.

> *Правило*
>
> Прежде чем написать какой-нибудь стиль или ввести новый класс, задумайтесь, нельзя ли оптимизировать и упорядочить код другим способом. Сделать так, чтобы не вводить этот класс или ввести максимально широко. Логика — незаменимый помощник любого профессионального веб-мастера Запомните это.

так, переходим непосредственно к рассмотрению способов, благодаря ко-
лрым вы существенно облегчите код и собственную жизнь.

Форматирование таблиц стилей

идвное требование — в таблице стилей должен быть порядок. Возьмем
ример из предыдущей главы. Вот эта таблица:

```
  tcolor: #000; font: x-sroall Verdana)
ilDY (background: #FFF}
ainmenu A {color: #000; font: bold small Arial}
```

```
TD {
  color: #000;
  font-style: italic;
  font-size: 14px}
```

А теперь внимательно сравните их и ответьте на вопрос, какая структурирована лучше и почему? Подумайте две минуты, не читайте дальше.

Нашли ответ? Если да, то вы, конечно, согласитесь со мной, что вторая таблица выглядит более логичной. В правилах второй таблицы стилей свойство цвета идет на первом месте, затем следуют свойства шрифта, а затем свойства текста, тогда как в первой таблице стилей *нет никакой системы*. Особенно это важно для больших таблиц стилей и в том случае, если вы записываете правило в одну строку.

Для себя вы можете выработать какой угодно порядок расположения свойств, однако при написании таблиц стилей вы должны строго его придерживаться. Например, я всегда на первое место ставлю цвет, на второе — свойства шрифта, на третье — свойства текста, а затем поля, отступы и рамки. Повторяю, изначально порядок может быть любым, главное, чтобы вы его строго соблюдали и постепенно привыкали к нему.

> ***Правило***
>
> Сформулируйте раз и навсегда свои собственные правила структуры таблицы стилей и соблюдайте их неукоснительно. Привычка— мощный инструмент увеличения производительности.

Имена классов и *ID*

При верстке вам частенько придется вводить новые классы и как-то их называть. Естественным образом возникает желание, чтобы названия были *короткими* и *логичными*. Если класс для заголовков раздела назвать headi, то это будет достаточно логично и коротко, а если bigbiacktext — нелогично и длинно. Конечно, часто встречаются ситуации, когда не получается придумать подходящее короткое имя, тогда приходится чем-то жертвовать: либо длиной, либо логичностью. Лично я предпочитаю жертвовать логичностью, потому что в своем коде я разберусь всегда. Однако если над проектом работает несколько HTML-верстальщиков и каждый из них создает какую-либо часть сайта, то лучше жертвовать длиной, потому что в этом случае логичность гораздо важнее. Согласитесь.' далеко не каждый поймет, что класс аа обозначает, к примеру, отступ справа на 10 пикселов, тогда как класс rightpadio об этом свидетельствует достаточно однозначно. Вообще вам все время придется балансировать на грани между объемом и логикой, важно сохранять баланс в большинстве случаев и четко представлять себе, когда его можно нарушить. Нарушить его можно в двух случаях.

- Если вы профессионал и работаете над HTML-кодом сайта в одиночку.

1 Если вам до зарезу надо максимально уменьшить объем кода и важным становится каждый лишний байт.

Больше о правилах форматирования таблиц стилей ничего сказать нельзя. Перейдем к тому, как создавать оптимизированные таблицы.

Оптимизация таблиц стилей

В любом деле оптимизация есть процесс желательный, а часто необходимый. Это естественно, потому что он несет всяческие выгоды, начиная с величения скорости работы приложения и заканчивая уменьшением затрат. :!сли взять локальную область, такую как верстка сайтов и написание таб-.|ц стилей в частности, то под оптимизацией понимается *уменьшение объема* • >*да,* что приносит следующие выгоды:

3 загрузка страниц немного ускоряется;

=\ писать и редактировать таблицу стилей становится несколько проще по причине уменьшения числа селекторов и свойств, что в конечном итоге ведет к экономии времени, пусть и небольшой.

\ак видите, не стоит ждать от оптимизации каких-то чудес, но и пренебречь ей не нужно, потому что это характеризует вас как профессионала и оказывает ваше отношение к делу. Собственно говоря, стремление *к* оптимизации кода— одна из отличительных черт профессиональных HTML-.фсталыщиков.

уществует несколько правил, которые могут помочь вам написать оптимилрованную таблицу стилей.

Стили по умолчанию

Че нужно писать ничего лишнего. Практически все свойства CSS имеют • лкое-либо значение по умолчанию. Не стоит явно их указывать. Например, ."иль на элемент <p>, приведенный ниже

```
i
font-style: normal;
letter-spacing: normal;
text-decoration: none1
```

:х>бще не имеет смысла, потому что все эти значения для элементов <p> .тановлены по умолчанию. Зачем же их указывать явно? Конечно, возможы случаи, когда надо переписать установленный ранее стиль. Например, -ы хотите, чтобы все элементы <H1> были подчеркнутыми и пишете стиль:

```
{
text-decoration: underline}
```

Но вдруг вам понадобилось, чтобы некоторые из заголовков первого уровня выводились неподчеркнутыми. Для этого вам придется ввести какой-нибудь дополнительный класс. Например, такой:

```
H1.nounderline {
  text-decoration: none}
```

То есть вы первым правилом переписали значение по умолчанию для элементов <KI>, поэтому потом пришлось явным образом восстанавливать это значение для заголовков с классом nounderline. В этом случае другого выхода нет, но вообще если вам нет необходимости изменять стили элементов, то и не надо прописывать значения по умолчанию явным образом.

Правило

Не указывайте в таблице стилей значений, которые являются для данного элемента значениями по умолчанию

Сокращенные формы записи

Второй способ сократить объем таблицы стилей — использование сокращенных форм записи некоторых свойств. В табл. 5.1 они систематизированы.

Таблица 5.1. *Сокращенные формы записи* ***в*** *CSS*

Сокращенная форма	Входящие свойства
font: italic s:r,all-caps 12px/140* Tahoma, sans-serif	font-style: italic **font-variant**: small-caps **font-weight**: normal font-size: 12px **line-height:** 140% **font-family:** Tahoma, sans-serif
margin: lem 2om **0 lem**	**margin-top**: lem **margin-right:** 2em **margin-bottom: 0** **margin-left:** lem
padding: ЮpX 20px	**padding-top:** lOpx padding-right: 20px padding-bottom: lOpx padding-left: 20px

Таблица 5.1 (окончание)

Сокращенная форма	Входящие свойства
border: ipx sonaffUuO	**border-width:** Ipx
border-top: Ipx solid #000	**border-style:** solid
border-right: Ipx solid #000	**color:** #000
border-bottom: Ipx solid #000	
border-left: Ipx solid #000	
background: urlCbackgr.gif") #fff repeat-x	**background-color:** #FFF
	background-image: url ("backgr.gif")
	background-repeat: repeat-x
	background-attachment: scroll
	background-position: Opx Opx

Использование сокращенных форм записи является делом весьма полезным нужным, потому что объем таблицы стилей при этом значительно умень-ается. Это видно уже из сравнения правого и левого столбцов табл. 5.1, но чобы окончательно вас убедить, я приведу еще один несложный пример. Пусть нам надо создать блок rule, который имеет черную рамку толщиной щи пиксел и внутренние отступы величиной 10 пикселов. Ширина блока Ч'дет 300 пикселов, а текст в нем будет выводиться полужирным шрифтом • erdana размером 12 пикселов. Если не пользоваться сокращенными фор-мами записи вообще, то код будет такой:

```
=rule {
  font-family: Verdana, sans-serif;
  font-weight: bold;
  font-size: 12px;
 width: 3OOpx;
 border-top: Ipx solid #000;
 border-right: Ipx solid #000;
 border-bottom: Ipx solid #000;
 oorder-left: Ipx solid #000;
 padding-top: lOpx;
 padding-right: lOpx;
 padding-bottom: lOpx;
 padding-left: lOpx}
```

Воспользуемся сокращенными формами записи, и правило для блока у нас получится такое:

```
#rule {
  font: bold 12px Verdana, sans-serif;
  width: 300px;
  border: 1px solid #000;
  padding: 10px}
```

Первое правило занимало 290 байтов, а второе всего 96. Вес уменьшился в три раза, т. е. оптимизация достигла 300%! Конечно, этот пример показывает что будет, если не использовать сокращений вообще, но тем не менее он совершенно реальный и вам частенько придется создавать очень похожие блоки, так что примите к сведению эти 300%.

Группировка

Для оптимизации таблиц стилей надо с умом применять группировку. Как вы уже знаете, группировка — это объединение селекторов с одинаковыми объявлениями. Однако из этого определения не вытекает достаточно неочевидное утверждение, что группировать селекторы можно только в том случае, *если они содержат совершенно одинаковые объявления.* Например, у нас есть такая таблица стилей:

```
P I
  color: #000;
  font: 14px Tahoraa, Verdana, sans-serif)
TD {
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif)
P i
  text-indent: 2em}
BODY {
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif}
```

Ее можно оптимизировать так:

```
p, то, BODY i
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif)
P {
  text-indent: 2em}
```

Получилось очень компактно, но зато логика таблицы стилей сильно пострадала. Если вам понадобится изменить какое-либо объявление в элементе <p>, то в большой таблице вы его можете искать очень долго, потому что селекторы на этот элемент будут располагаться в разных местах. Запомните правило.

Правило

Все объявления для одного и того же элемента должны находиться в одном месте.

Это означает, что в нашем примере мы можем объединить только селекторы ria элементы <BODY> И <TD>, а селекторы на элемент <P> надо свести в один, как того требует правило:

```
BODY, TD {
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif)
P (
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif;
  text-indent: 2em!
```

Кроме того, если вы захотите, скажем, добавить в таблицу стилей фон стра- lицы, то придется разбить группу селекторов <BODY> И <TD>, потому что .шли к ним уже не будут *совершенно одинаковыми*. Вот нам, к примеру, по- (адобилось добавить фоновый цвет на элемент <BODY>. Тогда придется разбить группу, и таблица стилей будет выглядеть так:

```
3ODY {
  color: #000;
  background-color: #CCC;
  iont: 14px Tahoma, Verdana, sans-serif)
"D {
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif)
= {
  color: #000;
  font: 14px Tahoma, Verdana, sans-serif;
  text-indent: 2em}
```

Строчные и прописные

Что касается манеры записи селекторов, то селекторы элементов я рекомен- \ю писать заглавными буквами, тогда как все остальные — строчными. На -JMOM деле это не играет большой роли, потом) что CSS *регистронезависи-*

мый язык, т. е. разницы между прописными и строчными буквами нет, поэтому все решает привычка. Объявления я тоже рекомендую писать строчными буквами, хотя это опять же дело привычки.

Вот практически все основные правила написания компактных и корректных таблиц стилей. Давайте сейчас оптимизируем отдельно взятую таблицу стилей, чтобы окончательно на практике закрепить полученные знания.

Оптимизация на практике

Исходная таблица у нас страшно неорганизованная и запутанная:

```
Ailink {font-weight: bold; COLOR: black)
A:visited (font-weight: bold; COLOR: #555555}
A:active {font-weight: bold; COLOR: #FF0000}
A:hover {font-weight: bold; COLOR: #FF0000}
#lnk A:link {color: #666666}
#lnk A:visited {color: #666666}
#lnk A:active (color: #ff99OO}
#lnk a:hover {color: #ff9900}
BODY (MARGIN: Opx; PADDING-BOTTOM: Opx;
      PADDING-LEFT: Opx; PADDING-RIGHT: Opx; PADDING-TOP: Opx}
.imgnews {margin-right: 12px}
CODE {color: #00C; font-family: courier new}
em {font-weight: bold; font-style: normal; color: black}
.norm {color: #555; FONT: 12px verdana, helvetica, Sans-serif}
p (text-align: justify; color: #555;
   FONT: 12px verdana, sans-serif}
input {border: lpx solid black; margin: lpx 2px}
TEXTAREA (border: lpx solid black; margin-top: lpx; margin-left: 2px;
          margin-bottom: lpx; margin-letf: 2px}
.EXAMPLE (background-color: tteeeeee; padding: 20px;
          BORDER: lpx dotted black}
```

Разобраться в ней непросто. Как видите, здесь нарушены все правила, описанные в этой главе. Начнем все исправлять и упорядочивать. Прежде всего разобьем таблицу стилей на три блока по селекторам: элементы, классы, ID, а в каждом блоке расположим селекторы в алфавитном порядке. Цифры обозначают порядковый номер строки и к самой таблице стилей не относятся. Получится так:

```
1. A:link {font-weight: bold; COLOR: black}
2. A;visited (font-weight: bold; COLOR: #555555}
```

```
1. A:active (font-weight: bold; COLOR: #FF0000}
4. A:hover {font-weight: bold; COLOR: #FF0000}
Ь. BODY (MARGIN: Opx; PADDING-BOTTOM: Opx; PADDING-LEFT: Opx;
   PADDING-RIGHT: Opx; PADDING-TOP: Opx}
•3. CODE {color: #00C; font-family: courier new)
7. em {font-weight: bold; font-style: normal; color: black}
3. input {border: lpx solid black; margin: lpx 2px}
9. p {text-align: justify; color: #555; FONT: 12px verdana, sans-serif)
10. TEXTAREA {border: lpx solid black; margin-top; lpx; margin-right:
    2px; margin-bottom: lpx; margin-letf: 2px}

:1 . .EXAMPLE {background-color: jfeeeeee; padding: 20px; BORDER: lpx
    dotted black}
12. .imgnews {margin-right: 12px}
13. .norm {color: #555; FONT: 12px verdana, helvetica. Sans-serif}

14. #lnk A:link {color: #666666}
15. #lnk A:visited {color: #666666}
16. #lnk A:active {color: Ш9900}
17. #lnk a:hover (color: #ff9900}
```

Пойдем дальше. Для начала займемся правилами для ссылок, которые находятся в строках 1—4. Заранее договоримся, что в объявлениях стилей сначала будет идти цвет и фон, затем шрифт, затем поля и отступы. Итак:

1. Перенесем в начало объявления цвета.
2. Свойство COLOR запишем строчными буквами.
3. Вместо обозначения черного цвета словом black запишем #000.
4. Остальные значения сократим так: #555555 поменяем на #555, #FFOOOO поменяем на #FOO.

Получится вот что:

```
1. A:link {
      color: #000;
-Щ    font-weight: bold}
2. A:visited {
      color: #555;
      font-weight: bold}
3. A:active /.
      color: #F00;
      font-weight: bold}
```

```
4. A:hover {
      color: #F00;
      font-weight: bold}
```

Заметьте, что последние два селектора имеют совершенно одинаковые объявления, так что можно смело их сгруппировать:

```
1. A:link {
      color: #000;
      font-weight:bold)
2. A:visite {
      color: #555;
      font-weight:bold}
3. A:active, A:hover {
      color: #F00;
      font-weight:bold}
```

Займемся строками 5—7:

```
5. BODY {MARGIN: Opx; PADDING-BOTTOM: Opx; PADDING-LEFT: Opx;
         PADDING-RIGHT: Opx; PADDING-TOP: Opx}
6. CODE {color: #00C; font-family: courier new}
1. em (font-weight: bold; font-style: normal; color: black}
```

Объявление стилей на элемент <BODY> содержит четыре отступа, причем все они имеют значение 0, так что можно смело вместо всех четырех отступов использовать сокращенную форму записи, кроме того, надо переписать все свойства строчными буквами:

```
5. BODY {
      margin: Opx;
      padding: Opx>
```

В строке 6 на первый взгляд все правильно, однако в свойстве font-family указан только один шрифт. Укажем кроме него еше и шрифтовое семейство, в данном случае это monospace. Кроме того, названия шрифтов из двух и более слов рекомендуется заключать в кавычки:

```
6. CODE {
      color: #00C;
      font-family: "Courier new", monospace}
```

В строке 7:

1. Запишем название селектора em большими буквами.
2. Переместим в начало свойство color и заменим значение black на #000.

```
1. EM {
      color: #000;
      font-weight: bold;
      font-style: normal}
```

Теперь рассмотрим строки 8—10:

```
8. input {border: 1px solid black; margin: 1px 2px}
Э. p {text-align: justify; color: #555; FONT: 12px verdana, sans-3erif}
10. TEXTAREA (border: 1px solid black; margin-top: 1px; margin-right:
    2px; margin-bottom: 1px; margin-letf: 2px}
```

Строка 8 практически правильная, надо только переписать селектор input заглавными буквами. В строке 9 надо:

1. Селектор p заменить на p.
2. Перенести в конец правила свойство text-align.
3. Переписать свойство FONT строчными буквами.

Получится так:

```
9. P {
      color: #555;
      font: 12px Verdana, sans-serif;
      text-align: justify)
```

В строке 10 можно кое-что сократить. Заметьте, что верхнее и нижнее поля имеют значение 1 пиксел, а левое и правое — 2 пиксела. Это можно записать сокращенно с помощью объявления margin: 1px 2px. Кроме того, надо в объявлении рамок поменять значение цвета black на #ооэ. Тогда строка 10 будет такой:

```
10. TEXTAREA {
      border: 1px solid #000;
      margin: 1px 2px;
```

А сейчас сравните исправленные строки 8 и 10. Они совершенно одинаковые, т. е. можно сгруппировать селекторы INPUT И TEXTAREA:

```
8. INPUT, TEXTAREA (
      border: 1px solid #000;
      margin: 1px 2px)
```

С блоком селекторов по элементам мы разобрались, так что перейдем к классам:

```
11. .EXAMPLE {background-color: #eeeeee; padding: 20px; BORDER: 1px
dotted black}
12. .imgnews {margin-right: 12px}
13. .norm {color: #555; FONT: 12px verdana, helvetica, Sans-serif}
```

Здесь надо в строке И переписать название масса EXAMPLE И СВОЙСТВО BORDER строчными буквами, а также написать значение цвета в сокращенной форме. Строка 12 теперь полностью корректна. А в строке 13 надо переписать свойство FONT строчными буквами:

```
11. .example {
      background-color: #EEE;
      border: Ipx dotted #000;
      padding: 20px}
12. .imgnews [
      margin-right: 12px}
13. .norm (
      color: #555;
      font: 12px Verdana, Helvetica, sans-serif I
```

Остался блок контекстных селекторов:

```
14. #lnk A:link {color: #666666}
15. #lnk A:visited {color: #666666}
16. #lnk A:active {color: #ff9900}
17. #lnk aihover {color: #ff9900J
```

Здесь надо только записать цвета в сокращенной форме и сгруппировать селекторы 14 и 15, а также 16 и 17:

```
14. #lnk A:IinJt, #lnk A:visited {
      color: #666}
15. #lnk Aractive, #lnk A:hov*r {
      color: #F90)
```

Вся обновленная таблица стилей будет выглядеть следующим образом:

```
1. A:link {
      color: #000;
      font-weight: bold}
2. A:visited {
      color: #555;
      font-weight: bold}
3. A:active, Aihover {
      color: #F00;
      font-weight: bold}
5. BODY {
      margin: Opx;
      padding: Opx}
```

```
6. CODE {
      color: #00C;
      font-family: "Courier new", monospace}
7. EM <
      color: #000;
      font-weight: bold;
      font-style: normal)
8. INPUT, TEXTAREA {
      border: Ipx solid #000;
      margin: Ipx 2px}
'. P (
      color: #555;
      font: 12px Verdana, sans-serif;
      text-align: justify}

11. .example {
      background-color: #EEE;
      border: Ipx dotted #000;
      padding: 20px}
12. .imgnews {
      margin-right: 12px}
13. .norm {
      color: #555;
      font: 12px Verdana, Helvetica, sans-serif}

14. #lnk A:link, #lnk A:visited {
      color: #666}
15. #lnk A:active, #lnk A:hover (
      color: #F90}
```

Как видите, таблица стилей стала значительно более структурированной и читаемой Кроме того, первоначальная таблица весила 877 байтов, а оптимизированная таблица весит 677 байтов (если убрать переводы строк и лишние пробелы перед каждым объяыеннем). Мы смогли уменьшить вес таблицы стилей на 23%, что само но себе не так уж мало. К тому же она стала более логичной, так что теперь найти нужное объявление или селектор не составит труда. Это и есть цель данной главы — научиться создавать легкочитаемые и компактные таблицы стилей, а также оптимизировать старые, уже созданные ранее.

Надеюсь, вы возьмете на вооружение описанные здесь способы организации и оптимизации таблиц стилей, потому что в противном случае созданные вами таблицы будут выглядеть непрофессионально. Стремление к совершенству — хорошее качество. Эта глава является одной, пусть и маленькой, но все же ступенькой на пути к нему.

Глава 6

Величины

Что бы вы ни пытались сделать с помощью CSS, без единиц измерения вам не обойтись. В языке HTML существует всего две единицы измерения, с помощью которых можно задавать линейные размеры элементов — это пикселы и проценты. Например, ширину таблицы можно задать так:

<TABLE WIDTH="750">

что будет соответствовать 750-ти пикселам. А можно так:

<TABLE WIDTH="100%">

что будет соответствовать ста процентам от ширины контейнера данного элемента. Если эта таблица является непосредственным предком элемента <BODY>, то она будет занимать все окно браузера по ширине.

Давайте вспомним, где в HTML нужно применять единицы измерений. Итак, в HTML можно задавать размеры рисунков, размеры встраиваемых объектов, размеры таблиц и ячеек таблиц, а также ширину встроенных фреймов. Кроме того, можно задавать ширину отступов для рисунков и полей всего документа. А еще можно было устанавливать размер шрифта с помощью атрибута SIZE тега <FONT>. И все. Если рассматривать важность всех этих возможностей, то задание ширины таблиц и ячеек, а также задание размера шрифта перекрывает все остальное.

В каскадных таблицах стилей ситуация гораздо интереснее. Огромное число свойств имеет численные значения, так что грамотная система единиц жизненно важна. По этой причине в стандарте CSS-1 единиц длины гораздо больше. Однако сразу надо сказать, что использовать вы будете все те же пикселы, все те же проценты, и время от времени величину em.

Вообще происхождение единиц в CSS позволяет разделить их на три группы. Первая группа — это *величины, пришедшие из реального мира,* которые используются повсеместно для измерения длин реальных предметов. К ним относятся:

- in — дюймы (пришли из английской метрической системы; один дюйм равен приблизительно 2,54 см);
- cm — сантиметры;
- mm — миллиметры.

Данная группа величин привычна всем, в этом ее достоинство.

Ко второй группе можно отнести величины, которые пришли в CSS *из полиграфии*. Они используются для установки размеров шрифта, межстрочных интервалов и прочих типографских величин. К ним относятся:

3 pt — типографский пункт (один пункт равен 1/72 дюйма);

3 pc — пика (одна пика равна 1/6 дюйма или 12 пунктам);

C ex — названия не имеет. Высота строчной буквы "x" в шрифте.

Данная группа величин привычна полиграфистам. Очень часто в графических пакетах (например, Photoshop) размер шрифта устанаашвается в пунктах.

Третью группу составляют *величины, которые являются относительными,* т. е. реальный размер элемента вычисляется относительно какой-либо иной величины. К ним относятся:

П e m - названия не имеет. Вычисляется относительно размера шрифта элемента;

П px — пиксел. Вычисляется относительно устройства отображения;

G % — процент. Естественно, процентные соотношения не являются единицами измерения в привычном смысле слова. Они вычисляются относительно размеров элемента-предка.

Данная группа (за исключением em) пришла, в общем-то, из языка HTML. Так что очень многие пользуются именно этими единицами. Конечно, есть еще много причин, которые приводят к тому, что первые две группы неудобны в использовании. Поговорим об этом подробнее.

Реальные единицы измерения

Реальные единицы измерения великолепно работают в физическом мире, однако для использования на экране монитора они оказываются мало приюдны. Причина на редкость проста: *мониторы имеют совершенно разные физические параметры*. К ним относятся:

- размер монитора по диагонали (варьируется от 14 до 24 дюймов);

П разрешение монитора (варьируется от 640x480 до 1600x1200).

Кроме того, на различных платформах могут быть различные экранные разрешения. Так, например, на платформе Macintosh разрешение 72 пиксела на дюйм, тогда как на платформе Windows — 96 пикселов на дюйм.

К чему это приводит[9] Если на мониторе с диагональю 21 дюйм шрифт размером 0,5 дюйма будет смотреться нормально, то на мониторе с диагональю 14 дюймов он будет слишком велик.

Точно так же, если при разрешении 640x480 текст, выведенный шрифтом размером 3 мм, будет читаться нормально, то при разрешении 1600x1200 невозможно будет ничего разобрать.

Кроме того, браузер сам по себе не в состоянии корректно отобразить величину в реальных единицах. Давайте рассмотрим, почему это происходит. Как известно, изображение на экране монитора строится на основе пикселов. Пиксел является *минимальной* единицей измерения, т. е. не бывает величины 0,5 пиксела. Чтобы отобразить на экране, скажем, два дюйма, операционная система переводит реальную величину в пикселы. И тут вступают в действие все вышеперечисленные факторы. Во-первых, физические размеры монитора, во-вторых, разрешение монитора, в-третьих, экранное разрешение.

Возьмем конкретный пример. Монитор размером 13 дюймов по горизонтали и 9 дюймов по вертикали с разрешением 1024x768 в операционной системе Windows, т. е. с экранным разрешением 96 пикселов на дюйм. Тогда величина два дюйма будет соответствовать 96x2=192 пикселам. Однако реальная высота этих двух дюймов будет (768/9)x2=171 пиксел. Получается несоответствие: браузер считает, что два дюйма — это 192 пиксела, а реальный размер двух дюймов на мониторе — 171 пиксел. Таким образом, если вы устанавливаете размер элемента в два дюйма, то на этом *конкретном* мониторе *реальный* размер элемента будет равен 2,22 дюйма. В то же время на другом мониторе с другими параметрами реальный размер может быть совершенно иным.

Исходя из всего этого, можно сделать простой вывод: использование в CSS реальных единиц измерения крайне ограничено. Принципиально использовать реальные единицы измерения можно только в том случае, когда *известны параметры устройства вывода.* Таким устройством может быть принтер, потому что для него известны геометрические размеры страницы и разрешение (например, 300 точек на дюйм и страница формата A4).

Давайте подумаем, для чего вообще нужны абсолютные единицы? Если мы знаем параметры устройства, то абсолютные единицы обеспечивают наиболее точный контроль. Именно так происходит в полиграфии, где верстальщик твердо знает, что его макет будет напечатан, скажем, на бумаге формата A4 и с разрешением 1200 точек на дюйм.

Теперь обратим свои взоры на HTML-страничку. Для чего ее вообще распечатывают? Делается это для того, чтобы в более комфортных условиях прочитать статью, обзор и любой другой текст. Значит пользователю нужно, чтобы распечатанный текст легко читался и был нормального размера. Обратите внимание на слово *нормального,* это значит, что шрифт должен быть не слишком большим, но и не совсем маленьким. Большинство принтеров имеет формат A4, так что размеры листа бумаги мы знаем точно. Осталось точно задать размер шрифта в абсолютных единицах для принтера. Вот здесь нам поможет атрибут MEDIA элемента <LINK>. Алгоритм действий такой.

1. Вы создаете отдельную внешнюю таблицу стилей для принтера. Называете ее как вам угодно, скажем, print.css.

2. Подключаете этот файл с помощью конструкции:

```
<LINK TVPE="text/css" REL-"atylesheet" MEDIA="print" HREF="print,css">
```

Тогда при выводе страницы на печать будет использоваться таблица стилей из файла print.ess, так что пользователю даже не придется заботиться о том, как страница будет напечатана, это за него должен сделать верстальщик сайта.

Типографские единицы

Типографские единицы измерения для дизайнера даже привычнее, чем рехтные единицы, потому что он сталкивается с ними повсеместно. В любом графическом пакете размер шрифта задается с помощью пунктов (pt). Так что использовать типографские единицы в CSS хочется по привычке (естественно, если вы до этого не занимались графическим дизайном, то этой вредной привычки у вас к счастью нет).

Однако придется изменить свои привычки по той простой причине, что пункты (pt) и пики (pc) принципиально *ничем не отличаются от реальных единиц измерения.* Это становится понятно если вспомнить, что такое пункт. Так вот один пункт равен 1/72 дюйма, а пика равна 12 пунктам, т. е. 1/6 дюйма. Таким образом, на использование данных типографских величин накладывается абсолютно такое же ограничение, как и на использование реальных вешчин. Получается, что использовать пункты и пики надо только в той таблице стилей, которая предназначается для вывода страницы на печать.

Надо сказать, что для принтера действительно лучше использовать типографские единицы, потому что они для него являются родными. Дело в том, что некоторые достаточно старые браузеры некорректно переводят пикселы в пункты. Например, если вы укажете размер шрифта 16 пикселов, то некоторые браузеры и распечатают его как шестнадцать точек. Но экранное разрешение значительно отличается от разрешения принтера. Например, принтер с разрешением 600 dpi выведет 16 пикселов как 0,03 дюйма (приблизительно 0,08 см), что читаться не будет совершенно.

Все вышесказанное ни в коей мере не относится к браузерам пятых и тем более шестых версий. Они совершенно корректно трансформируют пикселы при печати, так что можно не беспокоиться за неправильную распечатку страниц.

Кроме пунктов и пик есть еще одна достаточно интересная типографская единица — ex, которая соответствует высоте строчной букве "x". В отличие от всех предыдущих единиц, эта единица является *относительной.* Относительные единицы просто незаменимы в "резиновой" верстке, когда главным является сохранение пропорций. Что касается высоты буквы "x", то она может значительно варьироваться от шрифта к шрифту. У большинства шрифтов высота буквы "x" равна половине высоты шрифта, однако у некоторых декоративных шрифтов может состаа™ть всего одну треть от высоты шрифта.

Для чего же можно использовать единицу измерения, которая базируется на высоте буквы "x"? На самом деле, область применения данной единицы достаточно узка. Ее предпочтительно использовать для выравнивания по вертикали, чтобы точно выровнять какой-либо блок по линии текста. Кроме того, можно делать рамки, отступы и поля шириной по высоте текста. Вот, в общем-то, и все.

Для установки размера шрифта ex подходит слабо, потому что она обладает узким спектром значений. Например, невозможно задать размер шрифта на 25% больше, чем у родительского элемента. Для установки высоты строки (line-height) единица ex тоже не подходит, потому что надо знать полную высоту шрифта, иначе расстояние между строками может быть совсем не таким, каким задумывалось.

Что касается поддержки единицы ex в различных браузерах, то можно сказать следующее. В четвертых версиях браузеров Microsoft и Netscape данная единица измерения не поддерживается вовсе. В браузере Internet Explorer 5.0 1 ex равен 0,5 em, что абсолютно неверно, потому что у разных шрифтов соотношение высоты буквы "x" и высоты шрифта может быть и 0,3 em, и 0,6 em. Так что использованию данной единицы измерения мешает еще и плохая реализация ее поддержки браузерами.

Относительные величины

Единица em берется относительно размера шрифта элемента. Вообще возможны следующие ситуации. Если для элемента явно указан размер шрифта, то величина em вычисляется относительно этого размера. Пример:

```
P {
  font-size: 14px;
  line-height: 2em}
```

В этом случае размер шрифта равен 14 пикселам, а высота расстояния между строками будет равна 14x2=28 пикселам. Другой пример:

```
P i
  font-size: 14px;
  padding: 1.2em}
```

В этом случае ширина всех отступов в блоке будет равна 14,2=17 пикселам.

Но размер шрифта для элемента может и не указываться явно. Тогда em вычисляется относительно размера шрифта предка. Пример:

```
P {
  font-size:  16px}
CODE {
```

```
  font-size: O.Seau)
. , .
<P>Пример <ССЮЕ>кода програмыы</CO0Ex/P>
```

В этом случае размер шрифта, которым будет выводиться текст в элементе <CODE>, равен 16x0,8=13 пикселам.

Если же и у элемента-предка не указан размер шрифта, то em вычисляется относительно того размера шрифта, который выбран пользователем в браузере. Таким образом, можно делать таблицу стилей, *которая полностью адаптируется к наиболее предпочтительному для пользователя размеру шрифта.* Если все размеры в таблице стилей будут указаны в em, то при изменении размера шрифта они изменятся адекватно, т. е. пропорции сохранятся, а это очень важно. Обратите внимание на это утверждение. При такой таблице стилей доступность сайта возрастает, т. е. люди со слабым зрением смогут легко подстроить вид страницы под себя при сохранении всех пропорций. С другой стороны, люди с очень хорошим зрением любят мелкий шрифт на страницах, потому что на экране помешается больше информации, а это удобно. Так что они обычно делают шрифт мелким, Но и в этом случае вил страницы будет соответствующий, т. е. все пропорции сохранятся! Не правда ли, очень привлекательные возможности открываются перед разработчиками. Однако надо заметить, что сделать такую страницу достаточно сложно.

Как уже было сказано, для создания таблицы стилей, подстраивающейся под предпочтения пользователя, надо просто не указывать размер шрифта основных элементов, а размер шрифта тех элементов, которые должны отличаться от основного, указывать в em:

```
HI {
  color: #00C;
  font-size: 2em}
P {
  color: #000;
  font-family: Verdana, sans-serif}
.big (
  font-size: 1.2em}
.small t
  font-size: 0.3em}
```

Тогда шрифт в классе . small будет составлять 80% от выбранного пользователем шрифта, шрифт в классе .big — 120%, а шрифт в заголовках <HI> будет в два раза больше. Такого эффекта невозможно добиться, используя реальные и типографские единицы измерения.

Однако, если вы во что бы то ни стало хотите поддерживать старые браузеры, то от использования em и ex придется отказаться. В браузере Netscape 4.x поддержка данных единиц реализована настолько некорректно, что зачастую просто не поддается логике. Вообще говоря, для этого браузера можно безопасно использовать только px и %. Естественно, можно написать для старых браузеров свою таблицу стилей. Как это сделать, будет рассказано в главе, посвященной кросс-браузерным каскадным таблицам стилей.

Единица измерения px имеет принципиальное отличие от em, потому что она меняется в зависимости от устройства вывода, т. е. для конкретного монитора с конкретным разрешением I px есть величина постоянная, однако для разных разрешений она разная. Так, например, при разрешении монитора 800x600 один пиксел будет больше, чем при разрешении монитора 1024x768.

Надо сказать, что единица измерения px, в отличие от всех предыдущих единиц, была реализована в языке HTML. Это одна из причин, благодаря которой она совершенно корректно поддерживается всеми браузерами, даже самыми старыми.

Задавать все размеры в пикселах совершенно необходимо при жесткой верстке, когда надо позиционировать элементы с точностью до одного пиксела. Что касается "резиновой" верстки, то в ней лучше использовать em.

Чем же пиксел принципиально отличается от абсолютных единиц измерения, таких как in, mm, pt? На первый взгляд может показаться, что они очень похожи, но на самом деле это не так. Давайте для примера рассмотрим 15-дюймовый монитор. Допустим, мы задали размер шрифта 14 пикселов:

```
P {
  font-size: 14px}
```

При разрешении 1024x768 шрифт такого размера будет смотреться нормально, однако при разрешении 640x480 на этом же мониторе он будет чрезмерно крупным по той простой причине, что размер одного пиксела при этом разрешении гораздо больше. Однако вся прелесть в том, что пользователи устанавливают для себя наиболее приемлемое разрешение. Пользователи с 15-дюймовыми мониторами, как правило, устанавливают разрешение 1024x768 или 800x600, а пользователи с 17-дюймовыми мониторами устанавливают разрешение 1024x768 или 1280x1024. По этой причине шрифт размером 14 px будет нормально смотреться на всех распространенных мониторах.

Если принять в расчет различные разрешения в операционных системах Windows и MacOS, то проблем тоже не возникает. Если вы укажете размер шрифта, скажем, 16 пикселов, то это будет 16 пикселов и на макинтоше, и на обычной персоналке, не больше и не меньше.

Таким образом, пикселы избавлены от всех недостатков абсолютных единиц измерения. Поэтому они являются идеальным и единственно возможным решением для жесткого дизайна. Что касается распечатки страниц, то проблем тоже никаких не возникает, потому что современные драйверы принюров очень хорошо справляются с пересчетом размеров в пикселах.

Но все же пикселы имеют один существенный недостаток: пользователь не сможет изменить размер шрифта на страницы с помощью браузера (это не относится к браузерам Netscape 6.x и Opera 5+). В первую очередь отсутствие такой возможности сказывается на людях с ослабленным зрением. Вообще эту проблему можно решить с помощью переключаемых таблиц стилей. В этом случае вы подключаете к странице две таблицы стилей: первая обычная по умолчанию, а вторая альтернативная с размером шрифта, указанным в относительных единицах.

Например, первая таблица стилей будет храниться в файле default.css и содержать следующие правила:

```
·I{
  font: 18px Verdana, sans-serif)
P {
  font: 14px Verdana, sans-serif}
TD{
  font: 14px Verdana, sans-serif}
```

Вторая таблица стилей будет храниться в файле alter.css и будет содержать следующие правила:

```
HI l
  font: 1.4em Verdana, sans-serif}
? {
  font: lam Verdana, sans-serif}
TD{
  font: lem Verdana, sans-serif}
```

Подключаются к документу они с помощью тега <LINK>

```
<LINK REL="stylesheet" TYPE="text/css" HREF="default.ess"
?ITLE="Default">
<LINK REL="alternate stylesheet" TYPE="text/css" TITLE="Relative font"
HREF="alter.css">
```

Как видите, тот факт, что вторая таблица стилей является альтернативной, указывается с помощью атрибута REL.

При отображении страницы пользователь сможет выбирать одну из этих таблиц стилей. В браузере Netscape 6.x это делается через меню **View — Use stylesheet.** При выборе данного пункта выпадает список из таблиц стилей,

которые названы соответственно атрибутам TITLE элемента <LINK>. В данном конкретном случае список будет состоять из трех пунктов: **None** (то есть вообще отключить CSS), **Default** и **Relative font.** Тут надо заметить, что Netscape *6.x* великолепно может изменять размеры шрифта, заданного в пикселах. То же самое может делать и Opera 5+, а вот браузеры фирмы Microsoft — нет. Кроме того, в них нет механизма, который позволял бы пользователям выбирать таблицы стилей, а это уже серьезная проблема. Если обобщить, то ситуация такова:

- браузеры Netscape *6.x* и Opera 5+ позволяют выбирать таблицы стилей и могут увеличивать шрифт, размер которого задан в пикселах, т. е. фактически для этих браузеров альтернативные таблицы стилей не нужны;
- браузеры Microsoft Explorer 5+ не позволяют пользователям выбирать альтернативную таблицу стилей. Более того, они не могут изменять шрифт, размер которого задан в пикселах.

Фактически получается, что для пользователей Netscape *6.x* и Opera 5+ альтернативные таблицы стилей не нужны, а пользователи Internet Explorer 5+ не смогут ими воспользоваться, поскольку в браузере не реализован такой механизм. Эту проблему можно решить. С помощью Объектной Модели Документа (DOM) можно динамически подключать разные таблицы стилей при каком-либо действии пользователя. Например, можно поместить на странице кнопку, и при ее нажатии осуществлять переключение обычной таблицы стилей на альтернативную. Конечно, такое решение не является идеальным, но другого вообще не существует.

Осталось разобраться с процентами. Проценты обычно вычисляются от соответствующих размеров элемента-предка и применяются при "резиновой" верстке. Надо сказать, что проценты были уже и в языке HTML, так что они, как и пикселы, привычны. Чаще всего в процентах задают ширину и высоту блоков, а также высоту строки, реже отступы и поля, и совсем редко размер шрифта.

Однако старые браузеры, такие как Internet Explorer 4JV И Netscape Navigator 4.v имеют с процентами большие проблемы. Так Internet Explorer *4.x* корректно понимает значение в процентах только в том случае, если оно обозначает ширину или высоту строки, а в Netscape Navigator 4.x проценты реализованы с большим количеством ошибок. Так что, если для вас важна обратная совместимость, лучше от использования процентов воздержаться.

Вообще, если обобщить все знания о единицах измерений, то можно сказать, что для сохранения обратной совместимости надо пользоваться только пикселами, что и делали до сих пор почти все профессиональные HTML-верстальщики.

Если же вы ориентируетесь на браузеры пятых версий, то можно смело ввести в оборот единицы em и %. О пользе данных единиц можно особо не

распространяться, т. к. они напрямую связаны с преимуществами "резиновой" верстки:

- на любых мониторах страница будет занимать все окно браузера, что помогает максимально эффективно использовать столь драгоценное экранное пространство:
- можно разрешать пользователю изменять размер шрифта, тогда как при жесткой верстке изменять размеры шрифта совершенно нельзя, поскольку страница может просто развалиться. Так что при жестком дизайне размер шрифта нужно обязательно устанавливать в пикселах.

Эти преимущества достаточно серьезные. Кроме того, среди пользователей не без оснований бытует мнение, что "резиновая" верстка гораздо более сложна, чем жесткая, так что подобный сайт вызывает большее уважение, и он сам по себе является одним из признаков профессионализма. Это на самом деле так. В большинстве случаев "резиновая" верстка сложнее жесткой, а если постараться сделать так, чтобы пропорции сохранялись при изменении размера шрифта, то задача становится нетривиальной. Так что сложность является единственным минусом "резиновой" верстки.

Как бы то ни было, для каждого сайта подход к выбору схемы верстки должен быть строго индивидуальным.

Глава 7

Шрифт и текст

На данный момент текст является едва ли не единственным средством передачи информации во Всемирной сети. Графика в большинстве случаев служит исключительно для поддержки текста и для расстановки акцентов на странице. Звук вообще в Сети применяется редко, потому что даже небольшой трек весит достаточно много и требует либо высокоскоростного каната, либо значительного времени для загрузки. Что касается видео, то ситуация еще хуже. Даже совсем маленький ролик может загружаться несколько минут.

Текст обладает глобальным преимуществом с точки зрения скорости. Одна буква текста весит один байт, что по меркам сети Интернет очень мало. Исключительно по этой причине текст пока есть и в ближайшем будущем останется *самым распространенным источником информации* в Сети.

Без всяких сомнений, любой веб-разработчик должен уметь грамотно обращаться с главным носителем информации. CSS предоставляет достаточно обширные возможности по управлению визуальным представлением текста на странице, так что надо детально знать свой инструмент.

Для этого необходимо, во-первых, иметь хотя бы общее представление о шрифтах. Во-вторых, знать, как реализован контроль над шрифтом и текстом в HTML. В-третьих, знать соответствующие свойства CSS и способы их применения. Все это мы рассмотрим в данной главе и начнем с теории.

Краткое введение в типографику

В этом разделе мы будем иметь дело в основном со шрифтом, так что сразу надо дать определение понятия "шрифт", чтобы внести ясность.

> ***Определение***
>
> Шрифт— это набор символов, которые объединены общими стилевыми признаками. Причем совокупность стилевых признаков является уникальной.

Иными словами, по набору признаков можно *различать* шрифты, т. е. именно признаки делают шрифты непохожими друг на друга. Синонимом понятия "шрифт" является понятие "гарнитура".

На рис. 7.1 приведены различные шрифты.

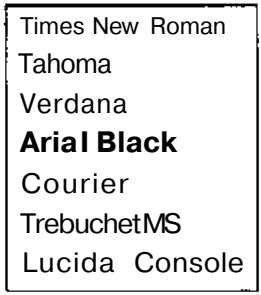


Рис. 7.1. Примеры различных шрифтов, которые установлены почти на всех компьютерах с ОС Windows

Как видите, мы достаточно легко можем идентифицировать шрифты. Каж-ЫЙ из них имеет уникальную форму отдельных букв (литер), межбуквенное расстояние, насыщенность и пр. Однако идентификация происходит имен-(о по форме литер, потому что другие атрибуты для любого шрифта можно вменять.

Давайте рассмотрим еще некоторые определения, относящиеся к шрифтам. Я сразу же буду показывать, чем отличается работа со шрифтом в полиграфии и в вебе, так что у вас должна сложиться ясная картина всех возможно-стей HTML-верстальшика.

Сначала рассмотрим возможность использования различных шрифтов. К сожалению, HTML-верстальщик весьма *ограничен* в выборе шрифта, поюму что на компьютере посетителя сайта обычно установлен какой-то базовый набор шрифтов, и, например, шрифта Georgia у него может не быть. Безопасными являются около десяти шрифтов, какие именно — вы узнаете чуть позже.

В полиграфии же верстальщик волен выбирать, какой шрифт использовать. Иными словами на выбор шрифта не накладывается никаких системных ограничений, и здесь имеется *полная свобода.*

Двигаемся дальше.

Определение

Высота шрифта (кегль) — это высота площадки, на которой располагается литера. Обычно кегль вычисляется от начала верхнего выносного элемента до конца нижнего выносного элемента, плюс небольшие отступы.

В полиграфии кегль обычно определяется в пунктах и пиках. Однако вы уже знаете недостатки подобных единиц применительно к экрану монитора. Давайте ненадолго вернемся к этому вопросу. В полиграфии верстальщик имеет глобальное преимущество — ему точно известны параметры страницы и в большинстве случаев качество печати. По этой причине он точно знает, что, скажем, шрифт в 12 пунктов будет выглядеть хорошо и легко читаться, Кроме того, печать происходит с большим разрешением, обычно 600 dpi и более.

В противоположность всему этому, HTML-верстальщик совершенно не знает параметров устройства вывода посетителя сайта. Даже если отбросить все альтернативные способы просмотра веб-страниц, WebTV и PDA, а оставить только мониторы, то все равно параметры у них могут быть очень разными. Так, например, разрешение мониторов варьируется от 640x480 до 1600x1200 пикселов, а линейные размеры по диагонали от 14 до 24 дюймов. Кроме того, как вы уже знаете, экранное разрешение может меняться в зависимости от операционной системы: на платформе Macintosh разрешение 72 пиксела на дюйм, а на платформе Windows — 96 пикселов на дюйм.

Можно подвести промежуточный итог и сравнить HTML-страницу с печатной. Такое сравнение приведено в табл. 7.1.

Таблица 7.1. *Сравнение параметров HTML-страницы и обычной печатной страницы*

Параметры	Веб-страница	Печатная страница
Линейный размер страницы	Заранее неизвестен	Заранее известен
Разрешение	Маленькое (72 — 96 dpi)	Большое (600 — 2400 dpi)
Наличие шрифта	Заранее неизвестно, поэтому выбор ограничен	Выбор не ограничен

Исходя из этого, получается, что в полиграфии верстальщик имеет *гораздо больший контроль над шрифтом.* Он может точно задавать размер шрифта и выбирать тот шрифт, который ему пришелся по душе. Тогда как веб-дизайнер такого контроля не имеет и к тому же очень ограничен в выборе самого шрифта.

По иронии судьбы, столь мощная в потенциале среда, как Интернет, до сих пор не может предоставить развитых технологий для работы со шрифтом. Надо сказать, что некоторые наметки уже есть. Так, например, в стандарте CSS-2 предусмотрен механизм, позволяющий устанавливать на компьютер пользователя необходимый шрифт. Однако в современных браузерах он пока не реализован.

Продолжим характеризовать шрифт. На рис. 7.2 показаны основные элементы шрифта. Приведем их определения.

Определение

Базовая линия — это гипотетическая линия, на которой и располагаются литеры. Причем нижние выноски и наплывы выходят за эту линию.

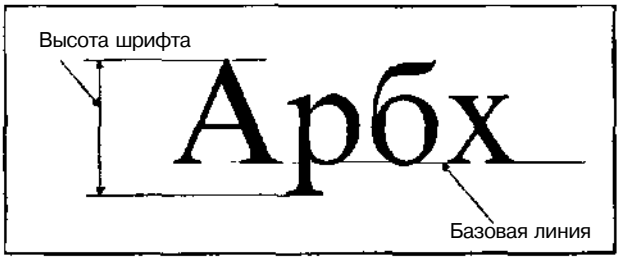


Рис. 7.2. Основные элементы шрифта

Определение

Наплыв— это выход округлых нижних элементов литер для устранения оптического обмана. Он делается для того, чтобы буквы выглядели лежащими на одной линии.

На рис. 7.2 наплыв есть у букв "р" и "б". Понятие базовой линии нам пригодится в дальнейшем, поскольку оно встретится в описании вертикального выравнивания и высоты строки.

Определение

Начертание — это комплект заглавных и строчных знаков, цифр и знаков препинания.

У одного и того же шрифта может быть несколько начертаний, которые отличаются друг от друга наклоном и насыщенностью:

- normal (обычный);
- bold (полужирный);
- *italic* (курсив);
- oblique (наклонный);
- bold italic (полужирный курсив);
- condensed (узкий);
- extended (широкий).

Что касается начертания, то в этом плане веб не отстает от полиграфии. Уже в HTML можно было устанавливать наклон и насыщенность шрифта, а CSS в принципе предоставляет еще бóльшие возможности, но они пока не реализованы производителями браузеров. Идем дальше.

Определение

Кернинг — это изменение величины пробела для *одной* пары букв (литер).

Кернинг очень часто применяется в полиграфии для того, чтобы текст выглядел более ровно. Его совершенно необходимо применять для шрифта большого размера, потому что между некоторыми парами литер (например ГА) получается слишком много пустоты и текст выыядит рваным. Поэтому пробел между такими литерами сокращают.

Средствами HTML нельзя осуществить кернинг, однако это принципиально можно сделать с помощью CSS Надо сказать, что решение громоздкое и неудобное, так что во многих случаях, если вам уж никак не обойтись без кернинга, лучше написать заголовок в графическом редакторе и вставить его на страницу рисунком.

Определение

Трекинг — это изменение величины пробела для *нескольких* букв (литер)

Очевидно, что принципиальное отличие трекиига от кернинга только в количестве букв, для которых изменяется величина пробела. При трекинге величина пробелов меняется одинаково для всей группы символов. В CSS ТрекИНГ Осуществляется С ПОМОЩЬЮ СВОЙСТВа letter-spacing.

Определение

Интерлиньяж — это расстояние между строками.

В среде веб-разработчиков этот термин не прижился, а вместо него употребляют выражение *высота строки*. Мы тоже не будем отходить от сложившейся традиции. Средствами HTML изменить высоту строки невозможно, однако ЭТО МОЖНО ОСУЩеСТВИТЬ С ПОМОЩЬЮ CSS, ИСПОЛЬЗУЯ СВОЙСТВО line-height.

В общем-то, этих определений достаточно для четкого понимания дальнейшего материала, так что перейдем непосредственно к особенностям использования шрифтов на веб-страницах.

Шрифты на веб-страницах

Гарнитуры

Сначала поговорим о гарнитурах. Вы никогда не задумывались, почему почти все книги напечатаны шрифтом с засечками, типа Times New Roman, а на подавляющем количестве сайтов для вывода текста используются шрифты без засечек, типа Verdana?

Вообще засечки предназначены для того, чтобы вести взгляд по строке, т. с они формируют достаточно устойчивую и цельную линию. Но, с другой стороны, засечки делают шрифт более декоративным и "сложным". Вот именно эта "сложность" играет решающую роль в том случае, если устройство вывода обладает маленьким разрешением. На экране с разрешением

% dpi шрифт с засечками воспринимается тяжелее, потому что для его вывода используется очень ограниченное число пикселов. Поэтому для маленьких разрешений лучше использовать как можно более простые шрифты, а таковыми и являются шрифты без засечек.

Основным параметром для определения удобства шрифта является скорость чтения. Так вот, для экрана монитора самым быстрочитаемым шрифтом является Tahoma. а на втором месте стоит Times New Roman. Надо сказать, что от кегля шрифта тоже зависит очень много. Так, в кегле 14 pt шрифты с засечками читаются быстрее, но это слишком большой кегль для вывода основного текста на странице. А вот в кегле 12 pt быстрее читается текст, набранный шрифтом без засечек. Кстати самым медленночитаемым является моноширинный шрифт Courier New, потому что он значительно шире остальных шрифтов, и строка, набранная им, содержит меньше слов, что существенно замедляет чтение.

На основе этих данных можно вывести простое правило.

Правило

Для основного текста на странице лучше использовать шрифт без засечек.

В полиграфии для заголовков традиционно используются шрифты без засечек, однако на веб это правило не распространяется, так что можно использовать любые шрифты, это уже дело вкуса и конкретного дизайнерского решения. Что касается моноширинных шрифтов, то они традиционно используются для вывода примеров программного кода.

Как вам уже известно, набор безопасных для использования шрифтов очень сильно ограничен шрифтами, установленными на компьютере пользователя. Около 98% пользователей Интернета предпочитают операционную систему Windows. Мы тоже будем ориентироваться на них, так что безопасными для использования можно считать шрифты, приведенные на рис. 7.3.

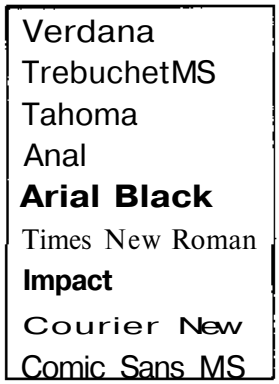


Рис. 7.3. Относительно безопасные шрифты для использования в веб

Как вы уже знаете, с помощью CSS первого уровня установить шрифт для элемента можно посредством свойства font-family:

```
P (
  font-family: Arial, sans-serif)
```

Существует правило, по которому название шрифта, состоящее более чем из одного слова, надо заключать в кавычки Однако делать это вовсе не обязательно (хотя желательно), потому что браузеры прекрасно понимают название и без кавычек, т е. следующие объявления совершенно одинаковы

```
P(
  font-familv "Tunes New Roman", serif}
P {
  font-family: Tunes New Roman, serif)
```

Вообще в последних версиях браузеров проблем с заданием начертания шрифта практически нет Однако некоторые ошибки в реализации все же имеются Так, например, браузеры Mozilla и Netscape 6 *x* адекватно воспринимают ключевое слово, обозначающее семейство шрифта, юлько в том случае, если оно прописано строчными буквами. Язык CSS является регистронезависимым, однако в этом случае имеет место тривиальный баг Почучается, что объявления

```
P {
  font-family: serif(
P f
  font-family: SERIF}
```

неравнозначны Так что во избежание ошибок лучше писать названия семейства строчными буквами, как в первом правиле

Возникает вполне логичный вопрос, почему до сих пор не был разработан механизм, который позволял бы автоматически устанавливать на компьютер пользователя шрифт, недостающий для корректного отображения вебстраниц" На самом деле подобные механизмы есть, но по ряду причин они так и не распространились

Практически в одно и то же время производители браузеров Netscape Navigator *4x* и Internet Explorer *4x* решили внедрить поддержку технологии, которая позволяла временно устана&чивать шрифты прямо с посещаемого сайта на компьютер пользователя Однако, к всеобщему сожалению, конк\ри рующие фирмы для этих целей решили использовать совершенно разные тех нологии Фирма Netscape назвала свою техночогию "Динамические шрифты (Dynamic Fonts), она базировалась на формате TrueDoc, и шрифтовой файл имел расширение pfr (Portable Font Resource) Фирма Microsoft назвала свою

технологию "Встроенные шрифты" (Embedded Font Technology), она базировалась на формате OpenType, и шрифтовой файл имел расширение eot. Давайте немного подробнее рассмотрим специфику обеих технологий.

Динамические шрифты (Netscape)

Обычно для отображения документа используются шрифты операционной системы, а они почти всегда имеют формат TrueType (файл с расширением ttf). Чтобы использовать динамические шрифты, надо вначале из TTF-файла создать PFR-файл. Для этих целей существуют специальные программы, такие как HexMac's Typograph и BitStream's Web FontMaker.

Созданный таким образом файл подключается к документу посредством элемента <LINK>, который может находиться только в секции HEAD, а выбор шрифта осуществляется с помощью атрибута FACE тега <FONT>. Например, если вы создали файл allfonts.pfr, в который включили шрифт Futuris, то вывести абзац текста этим шрифтом можно так:

```
<LINK REL="fontdef" SRC""http://mnr.youraite.com/fonte/allfont8.pfr">
. . .
<PXFONT FACE="Futuris">3TOT абзац будет выведен шрифтом Futuris, который является частью файла allfonts.pfr</FONT></P>
```

При первой загрузке страницы динамические шрифты кэшируются и при последующих загрузках уже не скачиваются с удаленного сервера, а берутся прямо их локального кэша, что удобно.

Встроенные шрифты (Microsoft)

Сразу скажу, что технология фирмы Microsoft куда более прогрессивна и, в обшем-то, опирается на стандарты. Чтобы реализовать HTML-страницу с использованием, скажем, шрифта Futuris, необходимо проделать следующие действия.

1. Создать HTML-страницу, прописав в стилях где это необходимо, шрифт Futuris. Например, вы хотите, чтобы этим шрифтом отображались только заголовки. Тогда НаДО НаПИСаТЬ Так: HI (font-family: **Futuris}.**
2. После того, как страница создана, надо использовать утилиту Microsoft's Web Embedding Fonts Tool (WEFT), которая проанализирует страницу и создаст файл с расширением eot. Этот файл яачяется *"шрифтовым объектом",* он отличается от обычных шрифтовых форматов тем, что является сжатым, содержит только те символы данного шрифта, которые используются на сайте или на странице, а также может использоваться только браузером, т. е. другие приложения не имеют к нему доступа.

3. После того, как создан шрифтовой объект, он подключается к странице посредством инструкции @font-face из спецификации CSS-2.

```
<STYLE TYPE="text/css">
6font-face!
  font-family: Futures;
  src: url(allfonts/futuris.eot)}
HI {
  font-family: Futuris, sans-serif}
P {
  font-family: Arial, sans-serif}
</STYLE>
. . .
<H1>Заголовок шрифтом Futuris</H1>
<P>00ычный текст шрифтом Arial</P>
```

Фактически компания Microsoft в то время использовала самые свежие рекомендации консорциума W3C по встраиваемым шрифтам. Однако ни технология компании Netscape, ни технология компании Microsoft не прижилась в среде веб-разработок. Причины, видимо следующие.

- В качестве шрифтового файла нельзя использовать стандарт TrueType, а приходится с помощью специальных утилит переводить шрифт или в объект EOT, или в формат PFR. На это надо время.
- И самое главное — технологии *разные*. Во время их появления процентное соотношение на рынке интернет-браузеров между Internet Explorer 4.Y и Netscape Navigator *4.x* было близким к 50/50, так что при необходимости использовать динамические шрифты надо было думать о совместимости А значит надо было создавать два файла; один *.eot, а второй *.pfr; а потом подключать их оба.

Если вернуться к нашему шрифту Futuris, то код страницы выглядел бы так:

```
<HEAD>
<LINK REL="fontdef" SRC="http://www.yoursite.com/fonts/allfonts.pfr">
<STYLE TYPE="text/css">
@font-face(
  font-family: Futuris;
  src: url(allfonts/futuris.eot)}
HI {
  font-family: Futuris, sans-serif }
P {
  font-family: Arial, sans-serif}
```

```
</STYLE>
</HEAD>
. . .
<H1>Заголовок шрифтом Futuris</H1>
<PX)бычныЙ текст шрифтом Arial</P>
```

Естественно, что мало кого прельщало таким образом поддерживать оба Зраузера, потому что процесс достаточно трудоемкий и вряд ли он себя оку- 1ает. Так что дизайнеры предпочитали использовать безопасные шрифты .1ля вывода основного текста, а заголовки и названия разделов частенько зыполняли в виде GIF-файлов.

Как бы то ни было, динамические шрифты на сайтах встречаются чрезвы- |<йшо редко, так что пока не появится четкая реализация, которая будет юлностью соответствовать спецификации CSS-2, дизайнеры по-прежнему будут делать заголовки в формате GIF, которые HTML-верстальщики будут зставлять на страницу с помощью элемента <IMG>, ХОТЬ ЭТО И абсолютно противоречит концепции HTML.

Размер

В полиграфии с размером шрифта нет никаких проблем. Верстальщик уста- 1авливает необходимый размер в пунктах, и он совершенно уверен, что на странице размер будет именно таким, какой он установил. Тут контроль над размером шрифта полный.

В среде Интернет ситуация далеко не такая радужная. Абсолютные единицы (К ним относятся те же пункты) использовать не рекомендуется по многим 1ричинам, перечисленным в предыдущей главе, а относительные единицы не обеспечивают такого жесткого контроля над размером шрифта, потому что его всегда может изменить сам пользователь Хорошо это или плохо? Для ответа на этот вопрос надо вернуться к отличиям печатной страницы от HTML-страницы.

Почему в полиграфии верстальщик может жестко задавать размер шрифта" По той простой причине, что он точно знает линейный размер страницы, на которой этот шрифт будет выводиться на печать. Он может не опасаться, что шрифт будет плохо читаться или его вдруг не окажется у наборщика. Это абсурдная ситуация.

Однако HTML-верстальшик не знает параметров устройства вывода Потенциально это может быть и портативный наладонник. и экран телевизора, и экран монитора В этом случае почти всегда лучше оставить выбор размера шрифта посетителю сайта, чем жестко устанаашвать его размеры самому. Конечно, если вы заведомо ориентируетесь на тех, кто посещает ваш сайт обычным способом, и вам совершенно все равно, смогут ли его читать все осталь-

ные, то можно задавать размер шрифта в пикселах (но не в метрических или типографических единицах измерения). Однако это далеко не лучший подход, потому что альтернативные способы просмотра сайтов получают все большее распространение, особенно в западноевропейских странах.

Итак, для задания размеров шрифта у нас есть две единицы — это пикселы (px) и em. Давайте разберемся, в каких случаях лучше использовать ту или иную единицу. Как вы уже знаете, существует два вида дизайна сайтов: "резиновый" (когда содержимое страницы растягивается или сжимается при изменении размеров окна браузера) и жесткий (когда ширина задана точно и не меняется при изменении размеров окна браузера).

Размер шрифта в жестком дизайне

При жестком дизайне элементы обычно позиционируются с точностью до одного пиксела, и для дизайнера часто чрезвычайно важно, чтобы шрифт имел совершенно определенный размер, иначе элементы могут сдвинуться друг относительно друга и дизайн страницы просто развалится. Иначе говоря, дизайнеру надо, чтобы пользователь не мог изменять размер шрифта. Мы уже говорили о минусах такого подхода, однако бывают случаи, когда это совершенно необходимо. Жесткий контроль над размером шрифта обеспечивает единица px. Например:

```
P {
  font-size: 14px}
```

В этом случае шрифт в абзацах будет иметь размер 14 пикселов на любом мониторе. Что касается изменения размера шрифта пользователем, то до недавнего времени ситуация была следующей: размер шрифта, заданного в пикселах, можно было изменять только в браузере Opera. Однако в последнее время ситуация изменилась, теперь изменять размер шрифта, заданного в пикселах, можно также в браузерах:

- Netscape 6.x;
- Opera 5+;
- Mozilla 0.9.x

Следовательно, нет никакой гарантии, что вместо величины шрифта 100% пользователь одного из этих браузеров не установит 120% и дизайн все равно может развалиться. Но вот браузеры компании Microsoft по-прежнему не позволяют изменять размер шрифта, заданного в пикселах. Учитывая тот факт, что этими браузерами пользуется около 90% аудитории Интернета, можно относительно спокойно задавать размер шрифта в пикселах, надеясь на то, что пользователи Netscape/Mozilia/Opera сами позаботятся об адекватном размере шрифта.

Это взгляд на проблему с точки зрения разработчика. Давайте посмотрим с точки зрения человека, зашедшего на сайт. Когда он заходит на сайт и испытывает дискомфорт при чтении текста, то у него возникает естественное желание избавиться от дискомфорта. Здесь возможны два варианта.

О Если человека очень заинтересует контент, то он постарается каким-либо образом облегчить восприятие текста Он попробует увеличить размер шрифта или отключить стили вообще, если его не устраивает контрастность. В браузерах Opera 5+ или Netscape 6.x у него это получится, потому что в них можно изменять размер шрифта, заданного в пикселах и отключать каскадные таблицы стилей. Однако в браузерах фирмы Microsoft у него ничего не выйдет. Естественно, это вызовет у посетителя раздражение, которое может стать причиной его ухода с сайта вообще. Так что с точки зрения посетителя размер шрифта в пикселах плох. Конечно, есть обходной путь решения этой проблемы. С помощью DOM и JavaScript можно динамически отключать стили или изменять размер шрифта, так что можно сделать в интерфейсе сайта две кнопки, одна из которых будет отключать стили, а другая увеличивать размер шрифта процентов на 30.

3 Если человека контент интересует постольку поскольку, то он уйдет сразу, даже не пытаясь ничего изменять.

На вторую категорию пользователей повлиять невозможно. Для нее надо просто угадать любимый размер шрифта, что практически невозможно. А вот для первой категории размер шрифта лучше задавать в em и соблюдать нормальную контрастность пары текст/фон.

Перейдем к "резиновому" дизайну.

Размер шрифта в "резиновом" дизайне

Как вы уже знаете, основное отличие "резинового" дизайна от жесткого заключается в том, что страница занимает все окно браузера. Учитывая тот факт, что практически все пользователи разворачивают окно браузера на весь экран, можно сказать, что для мониторов размер страницы будет варьироваться от 640x480 до 1600x1200 пикселов. Как видим, разница может быть больше чем в два раза. Естественно, очень сложно даже в таком дизайне сделать страницу, которая хорошо выглядела бы при любом разрешении. Однако при жестком дизайне этого вообще добиться нельзя.

Можно слегка упростить задачу, если учесть, что число пользователей с разрешением 640x480 составляет около 2%, а пользователей с разрешением 1600x1200 около 3%. Если не принимать во внимание эти 5% (что само по себе неправильно), то размер страницы будет варьироваться от 800x600 до 1280x1024 пикселов.

При "резиновом" дизайне позиционирование элементов с точностью до одного пиксела часто невозможно, но этого обычно и не требуется. Главным фактором является *сохранение пропорций.* Именно по этой причине здесь гораздо предпочтительнее пользоваться относительными единицами, поскольку именно они и только они обеспечивают сохранение пропорций. Что касается шрифта, то его размер лучше всего устанавливать в em. Причем размер в em надо указывать для *всех без исключения элементов и шрифтов* на странице, потому что только в этом случае будет обеспечено сохранение всех пропорций, при изменении размера шрифта посетителем. Давайте разберем это на конкретных примерах. Вот таблица стилей, где все размеры шрифта заданы величиной em:

```
BIG {
  font-size: 1.2em}
BODY {
  font-size: 1em}
HI (
  font-size: 1.8am}
P {
  font-size: 1em}
SMALL {
  font-size: 0.8em|
```

Допустим, пользователь просматривает страницу браузером Internet Explorer *5.x,* и у него в меню **Размер шрифта** выбрано значение **Средний**. Этот случай показан на рис. 7.4.

Если пользователь изменит размер шрифта на **Крупный,** то **шрифт** увеличится, однако все пропорции сохранятся (рис. 7.5).

А теперь давайте рассмотрим другую таблицу стилей:

```
BIG {
  font-size: 18px}
BODY {
  font-size: 1emf
HI {
  font-size: 24px}
P {
  font-size: lem}
SMALL {
  font-size: 12px}
```

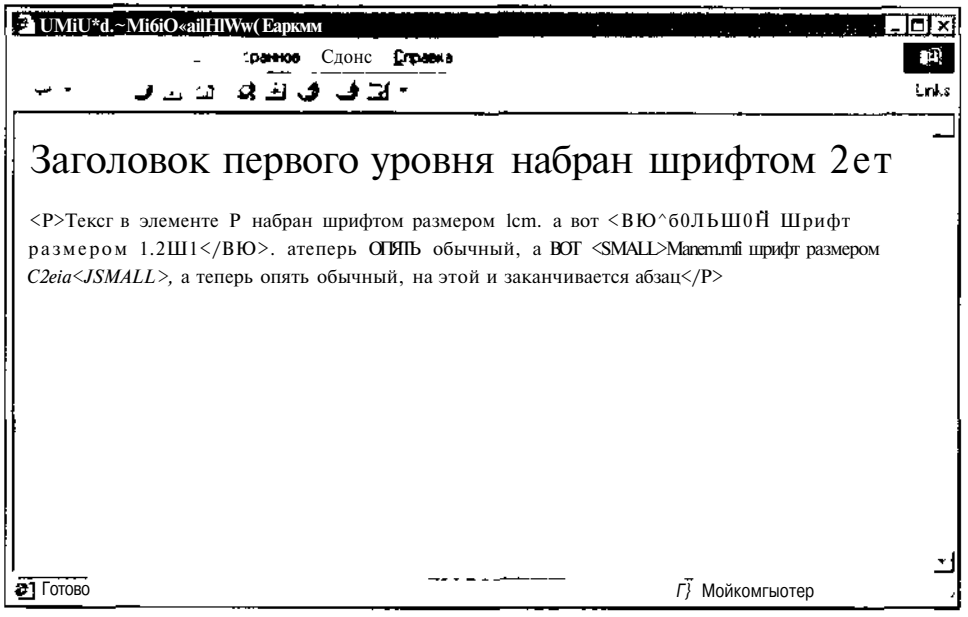


Рис. 7.4. Вид шрифта, размер которого задан в em, при пользовательском значении размера шрифта Средний

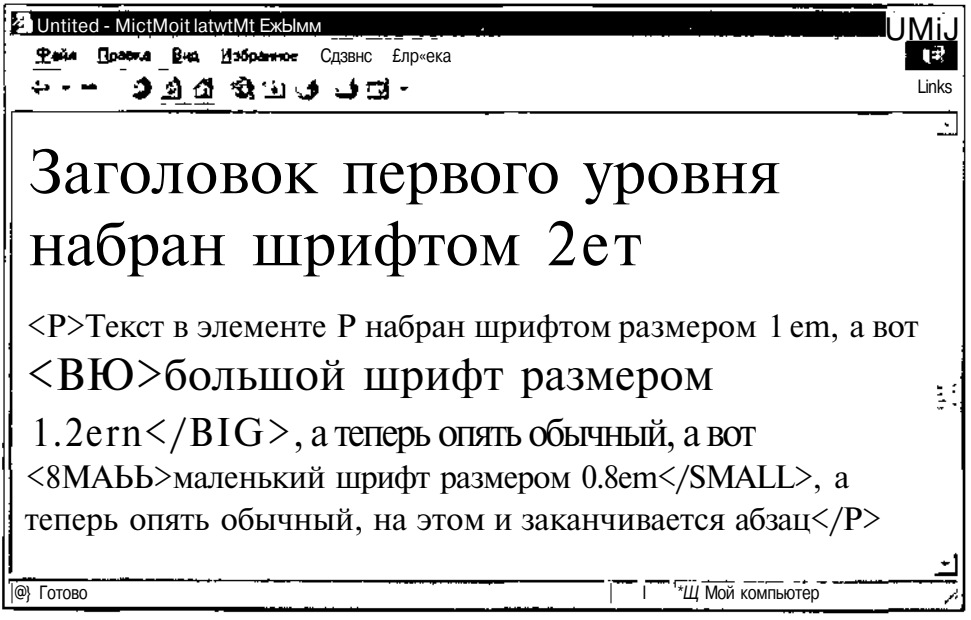


Рис. 7.5. Вид шрифта, размер которого задан в em, при пользовательском значении размера шрифта **Крупный**

Как видите, в ней только размер шрифта для элементов <P> И <BODY> задан с помощью относительных единиц em, тогда как размер шрифта остальных элементов задан с помощью пикселов. На рис. 7.6 показано, какой вид будет иметь тестовая страница в браузере Internet Explorer 5, со значением **Средний** в меню **Размер шрифта.**

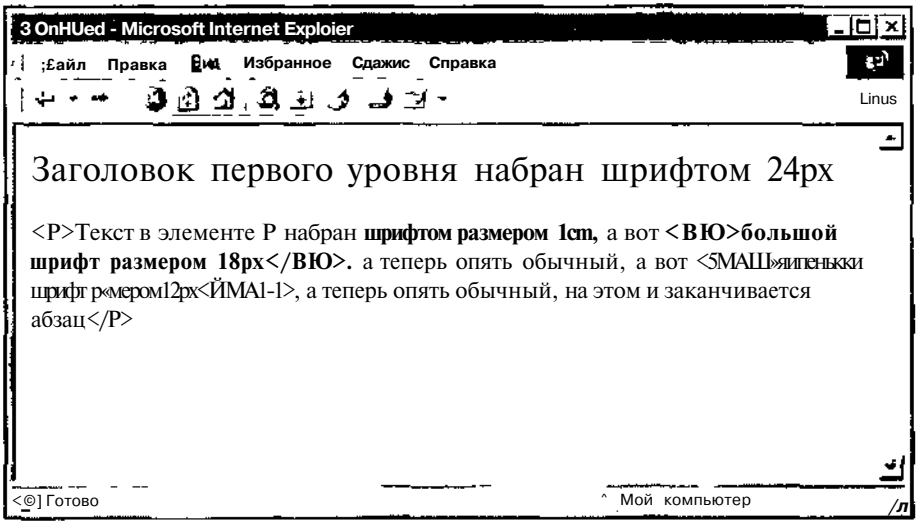


Рис. 7.6. Вид шрифтов, размер которых задан и в em, и в px, при пользовательском значении размера шрифта **Средний**

Как видите, пропорции адекватны, т. е. шрифт заголовка большой, в элементе <BIG> размер шрифта больше, а в элементе <SMALL> меньше, чем в элементе <P>. Теперь попробуем изменить значение в меню **Размер шрифта** на **Крупный.** То, что мы увидим, показано на рис. 7.7

Пропорции критически нарушились. Размер шрифта в заголовке и в основном тексте практически одинаков, а шрифт в элементе <BIG> стал значительно меньше шрифта в элементе <p>, что воспринимается совершенно неадекватно и нелогично.

На основе всего этого можно сделать следующий вывод.

Правило

Никогда не смешивайте при установке размеров шрифтов на странице единицы измерения em и px Все размеры шрифта на странице должны быть заданы или только в пикселах, или только в em

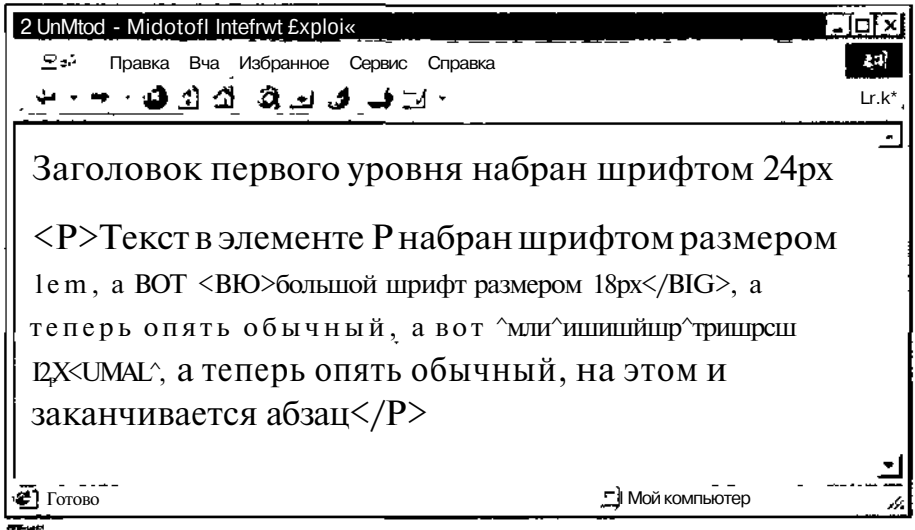


Рис. 7.7. Вид шрифтов, размер которых задан и в em, и в px, при пользовательском значении размера шрифта Крупный

Размер шрифта и ключевые слова

Кроме параметров em и px для установки размеров шрифта можно использовать ключевые слова xx-small, x-small, small, medium, large, x-large, x-large. Нечто похожее как раз и существовало в языке HTML: там атри-5ут SIZE элемента <гаыт> имел семь возможных значений (от 1 до 7). Ключевых слов тоже семь, но с их использованием есть определенные проблемы. Проблемы начинаются уже в спецификациях CSS-1 и CSS-2.

В спецификации CSS-1 говорится, что множитель для ключевых слов должен быть 1,5, т. е. если ключевому слову medium в браузере соответствует ирифт размером 16 px, то ключевому слову large соответствует шрифт 16x1,5=24 px. Именно такой множитель реализован в браузере Netscape Navigator 4.x.

В спецификации CSS-2 говорится, что множитель должен быть 1,2, т. е. есть ключевому слову medium в браузере соответствует шрифт размером 16 px, то ключевому слову large соответствует шрифт 16x1,2=19 px. Такой множитель реализован в браузерах Internet Explorer 5+, Netscape б.д:, Opera 5+.

Что ж, для нас это не имеет решающего значения, потому как браузером Netscape Navigator 4.jc пользуется весьма маленький процент интернет-аудитории. Однако есть еще и другая проблема. Из рис. 7.8 видно, что ключевые слова в различных браузерах выглядят по-разному.

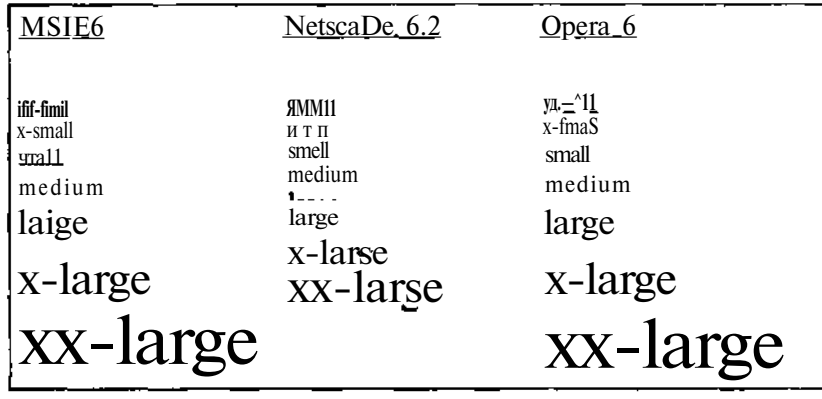


Рис. 7.8. Размер ключевых слов в различных браузерах

Помните, в *главе 4* я упоминал, что ось размеров шрифтов в **HTML** *асимметричная,* потому что базовым является размер <FONT SIZE="3">, а в CSS ось размеров шрифтов *симметричная,* потому что базовым является размер medium, находящийся точно посредине. Это ведет к проблеме синхронизации размеров шрифтов в HTML и CSS, а точнее возникает вопрос, что принять за базовый размер?

В спецификации CSS ясно сказано, что базовым является значение medium, чему и следуют браузеры Netscape 6.x, Mozilla и Internet Explorer 6.x (В режиме совместимости со стандартами).

А вот если напрямую сравнить оси размеров, то базовым будет являться значение small. Именно таким образом и поступают браузеры **Internet** Explorer *5.x* и Opera 5+. Эти разночтения, как видно из рис. 7.8, ведут к тому, что реальный размер шрифта, заданный в ключевых словах, будет неодинаков в разных браузерах. Точнее, будет происходить как бы сдвиг размеров. Посмотрите на рисунок внимательнее: размер medium в браузере Netscape 6.2 равен размеру small в браузерах Internet Explorer *5.x* и Opera 5+.

Вообще спорить, как на самом деле лучше, бесполезно. Но крайне неприятным является тот факт, что реализация поддержки ключевых слов *различается от браузера к браузеру,* и это значительно затрудняет использование их на практике.

На самом деле выход есть. Можно писать кросс-браузерные таблицы стилей, тогда, если в таблицах стилей для Internet Explorer *5.x* и Opera 5+ для элемента <p> будет задан размер шрифта small, то в таблице стилей для Netscape 6.x надо будет задать размер шрифта medium:

Файл ie5.css

```
P {
  font-size: email]
```

Файлпп6.css

```
P {
  font-size: medium}
```

После этого надо подключать файлы к странице в зависимости от используемого браузера. Как это сделать, рассказано в *гшве 10.*

Размер шрифта и элементы

С единицами измерения мы разобрались, а теперь перейдем к вопросу, для каких элементов надо устанавливать размер шрифта, а для каких это делать необязательно. Вообще существует простое правило. Есть набор элементов, is которых браузер так или иначе изменяет размер шрифта, по сравнению с размером шрифта для основного текста: <BIG>, <SMALL>, <SUB>, <SUP>, • CODE>, <PRE>, <тт>, <KBD>, <HI>—<нб>. Вот именно для всех этих элементов надо обязательно устанавливать размер шрифта. Для всех остальных элементов размер шрифта устанавливать совершенно не обязательно, хотя в некоторых случаях это тоже надо делать.

Все вышесказанное относится к браузерам, которые корректно поддерживают стандарт CSS-1. Однако до сих пор около 10% аудитории Интернета пользуются браузерами Netscape Navigator *4.x* и Internet Explorer 4.x. Я не буду заострять ваше внимание на этих браузерах, потому что через год процент их использования значительно уменьшится, но кратко описать их проблемы нужно.

Четвертые версии браузеров весьма некорректно поддерживают единицу em (в особенности это касается Netscape Navigator *4.x),* так что единственной единицей измерения для установки размера шрифта является пиксел. Если вы решите поддерживать старые браузеры, то вам или придется устанавливать все размеры шрифтов в пикселах, или писать кросс-браузерные таблицы стилей.

В браузере Netscape *4.x* есть еще один очень неприятный баг: шрифты приблизительно одинакового размера отображаются как одинаковые, т. е. следующие объявления равнозначны:

```
P {
  font-size: 13px}
P {
  font-size: 14px}
```

По этой причине размеры шрифтов в браузерах Internet Explorer и Netscape 4.x могут значительно отличаться, и добиться одинакового отображения некоторых кеглей совершенно невозможно.

N 3,1K г :

Увеличивать высоту строки можно для обычного текста, если плотность информации не яатяется критическим фактором. Если это лента новостей, то плотность информации важный фактор, если же это статья или иной материал, предназначенный для относительно долгого чтения, то плотность информации отступает на второй план, а главным фактором является комфортность чтения.

Что касается поддержки свойства line-height в браузерах, то ситуация достаточно хорошая, но и здесь без проблем не обошлось. Так, в браузере Internet Explorer *5.x,* если в строке встречается встроенный рисунок, то объявление line-height автоматически становится равным normal (то есть значению по умолчанию). Учитывая тот факт, что рисунки встраивают прямо в строку текста нечасто, этот баг не является принципиальным.

В браузере Netscape Navigator 4.x поддержка данного свойства реализована настолько некорректно, что практически любое его использование исключено. Очень осторожно line-height можно применять только к блоковым элементам типа <?>, в которых не содержится рисунков В противном случае может произойти наложение текста на рисунок или в лучшем случае высота строки останется прежней

Что касается всех остальных браузеров, то они поддерживают свойство line-height весьма корректно, и никаких значимых багов в них нет.

Выравнивание

В HTML для выравнивания абзацев текста предназначен атрибут ALIGN, который может принимать четыре значения: left, center, right, justify. В CSS есть совершенно аналогичное свойство text-align, которое может принимать такие же значения.

При выравнивании текста с помощью свойства text-align надо быть осторожным, потому что может возникнуть следующая ситуация. Вы решили выровнять все абзацы на странице по обоим краям и в таблице стилей пишете такое правило:

```
p{
  text-align: justify}
```

Все великолепно работает и выравнивается правильно. Но вот вам понадобилось один из абзацев, скажем, в блоке новостей, выровнять по правому краю. Вы, недолго думая, пропишете этому абзацу атрибут ALIGN прямо в HTML-коде:

```
<P ALIGN="right">A63au в блоке новостей...
```

Однако, вопреки вашим ожиданиям, абзац по-прежнему будет выровнен по обоим краям блока, а не только по правому краю. Это происходит из-за

того, что *объявление выравнивания в CSS важнее, чем объявление выравнивания в HTML,* поэтому атрибут ALIGN вам не поможет. Для того чтобы выровнять абзац по правому краю, вам придется вводить отдельный класс:

```
.alignright {
  text-align: right}
```

А в коде страницы встаачять этот класс во все элементы, в которых надо выровнять текст по правому краю:

```
r CLASS="alignright">A63au в блоке новостей...
```

Получается такая ситуация: если у вас на странице половина текстовых блоков выровнена по одному краю, а вторая — по другому, то использование выравнивания с помощью свойства CSS text-align не приносит практически никакой выгоды, логичность структуры не увеличивается, размер кода если и сокращается, то весьма несущественно. Так что в подобной ситуации лучше и не пользоваться каскадными таблицами стилей для выравнивания текста. Конечно, если какой-то вид выравнивания преобладает, то гораздо 1\чше его осуществлять с помощью CSS.

Пробелы

Пробелы в тексте бывают двух видов: межбуквенные и межсловные. Нетрудно догадаться, что межбуквенные пробелы находятся между отдельными буквами (в CSS их можно регулировать с помощью свойства letter-spacing), а межсловные пробелы находятся между отдельными словами (в CSS их можно регулировать с помощью свойства word-spacing). Пробелы очень сильно влияют на комфортность чтения и в идеапе строка должна представлять собой однородную линию, т. е. в строке не должно быть ни слишком больших, ни слишком маленьких пробелов. Сначала разберемся с пробелами между словами.

Межсловные пробелы

Вообще рекомендуется слегка увеличивать расстояние между словами в тексте, который отображается шрифтом в мелком кегле (рис. 7.И).

Расстояния между словами увеличиваются следующим образом:

```
? {
  word-spacing: 0.2em>
```

Надо сказать, что свойство это совсем недавно стали поддерживать браузеры, по этой причине без багов не обошлось. Свойство word-spacing вообще не поддерживается браузерами Internet Explorer 4.Y—5x и Netscape Navigator 4.x Что касается всех остальных браузеров, то у них есть небольшая ошибка в реализации. ЕСЛИ Элемент, К КОТОрОМу Применено СВОЙСТВО word-spacing, СО-

держит только одно слово, то пробел не будет увеличен за пределами этого элемента. Например, у нас есть такой код:

```
EM {
  word-spacing: 3em]
. . .
<P>Абзац текста с <EM>выделенным</EM> словом</P>
```

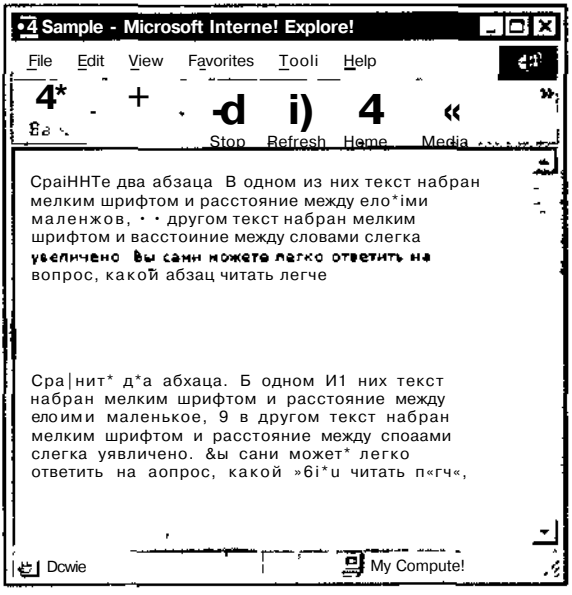


Рис. 7.11. Абзацы текста с разным расстоянием между словами. В первом абзаце стоит значение по умолчанию, а во втором абзаце оно слегка увеличено

Этот код ие лишен смысла, потому что, согласно спецификации CSS-1, перед и после слова "выделенным" должен добавляться пробел шириной I em, т. е. должно обеспечиваться как бы дополнительное выделение пустым пространством. Однако на самом деле никакого дополнительного пробела не будет, в этом и заключается ошибка.

Межбуквенные пробелы

Межб\квенные пробелы имеют гораздо большее значение и оказывают существенно большее влияние на восприятие текста, чем пробелы между словами. Сначала рассмотрим, в каких случаях межбуквенные пробелы надо увеличивать, а в каких уменьшать. Чаше всего трекинг (изменение величины пробела между несколькими буквами) применяется в заголовках.

Существует два диаметрально противоположных приема:

Э буквы в заголовках располагают очень тесно друг к другу (это достаточно естественное желание):

3 буквы разносят на достаточно большие расстояния (такой прием называется разрядкой).

>ба приема показаны на рис. 7.12.

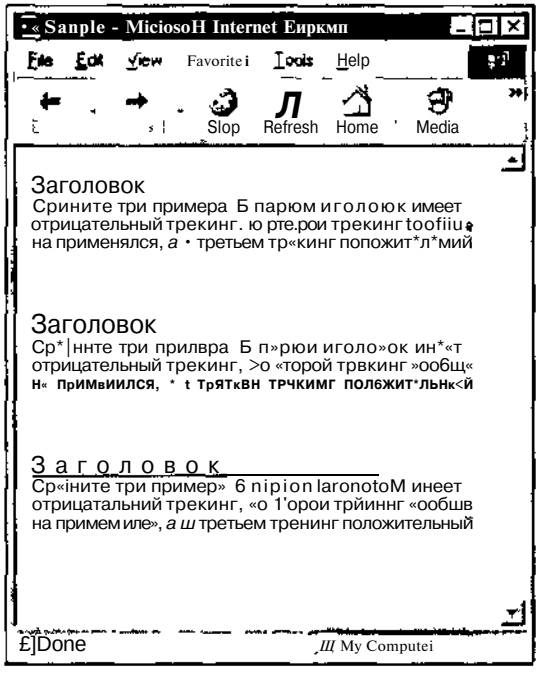


Рис. 7.12. Трекинг в заголовках

чак видите, первый и третий варианты смотрятся привлекательнее, тогда -ак второй вариант с межбуквенными пробелами по умолчанию, смотрится тривиально.

Кроме того, чем больше шрифт в заголовке, тем лучше смотрится заголовок : \меньшенными межбуквенными пробелами (рис. 7.13)

На рис. 7.13 в левой колонке заголовки без трекинга, а в правой — с тренингом. Вы сами можете убедиться, что для заголовков, выведенных боль-'lим шрифтом, гораздо лучше слегка уменьшить межбуквенные интервалы.

Часто в жизни верстальщика бывает такая ситуация: верстаешь шаблон для штернет-магазина, а в шаблоне перечень товаров, которые можно купить. Перечень, естественно, представляет собой таблицу. Порой в такой таблице

есть, скажем, шесть колонок, и при требуемом размере шрифта информация в эту таблицу ну никак не влезает. Вам может очень пригодиться такая строка в таблице стилей:

```
TD '
  letter-spacing: -lpx)
```

Рис. 7.13. Отрицательный трекинг в заголовках

В этом случае межбуквепиое расстояние в тексте, который находится в ячейках таблицы, уменьшится на один пиксел. Эффект может быть таким, как на рис. 7 14.

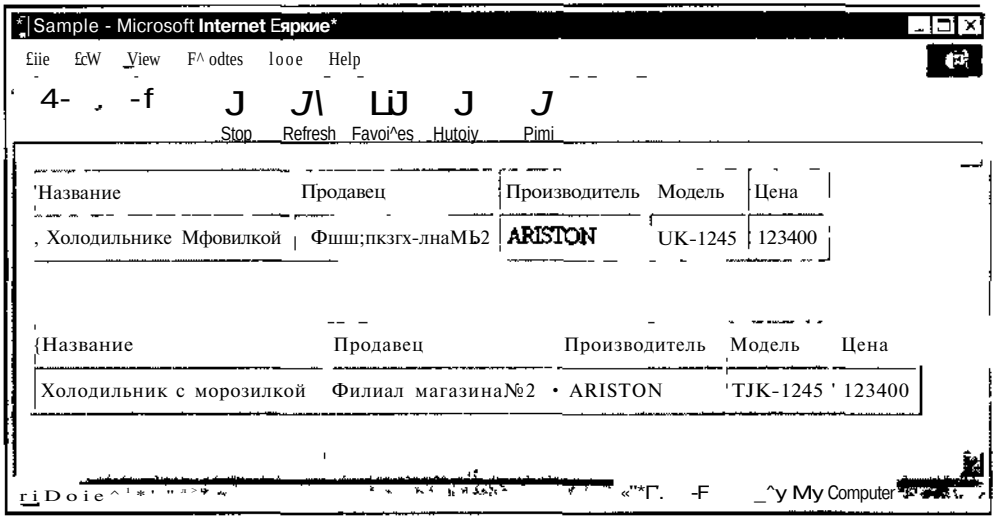


Рис. 7.14. Сравнение таблицы с отрицательным трекингом с таблицей без трекинга

Да, информация воспринимается несколько хуже, но трекинг позволил сэкономить место для целой колонки с ценой. В критических случаях это достаточно важно.

В полиграфии часто применяют разрядку для выделения в тексте. Разрядка действительно очень сильный прием, который может поспорить с выделением полужирным или наклонным шрифтом. Дело в том, что при чтении глаза скользят по строке и подсознательно на очень короткое время останавливаются перед пробелом, а когда в слове каждая буква отделена пробелом, то взгляд как бы притормаживает и такое слово читается труднее и медленнее, а значит на нем будет заострено внимание.

Однако разрядка требует увеличения пробелов вокруг выделенного слова, а из-за некорректной поддержки браузерами свойства word-spacing установить больший пробел до и после слова не удастся. Однако давайте разберемся в этом вопросе подробно. Итак, мы хотим сделать, чтобы в элементе <EM> текст выделялся не наклоном букв, а разрядкой. Пишем такой стиль:

```
EM {
  font-style: normal;
  letter-spacing: lent}
```

При этом мы переписали стиль шрифта с italic (который является для элемента <EM> стилем по умолчанию) на normal, и установили ширину межбуквенных расстояний 1 em. Выглядеть это будет примерно так, как показано на рис. 7.15.

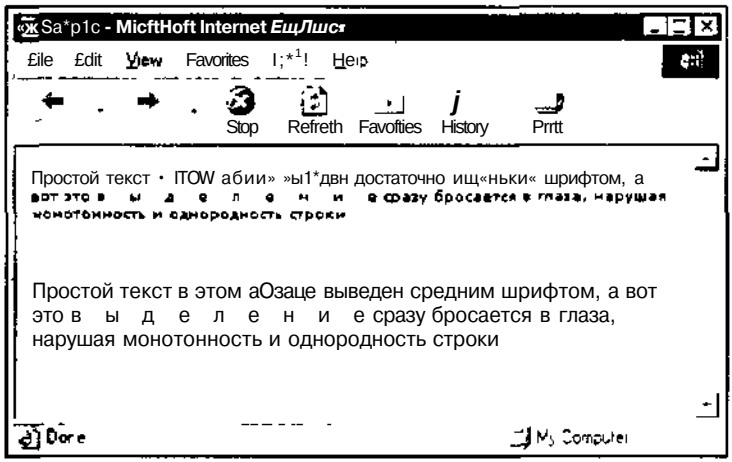


Рис. 7.15. Выделение слова разрядкой

Как видите, первая и последняя буквы в выделенном слове хочется отнести к прилегающим словам, потому что там пробел меньше. Тогда надо увели-

чить пробел перед первой и после последней буквы Согласно спецификации, делается это с помощью свойства word-spacing так:

```
EM {
  font-style: normal;
  letter-spacing: lem;
  word-spacing: 1.4em)
```

Но это работать не будет, так что надо искать обходные решения. Например, можно установить дополнительные отступы для элемента <EM>:

```
EM (
  font-style: normal;
  letter-spacing: 1.Oera;
  padding-left: 1.4am;
  paddang-right: 0.4em}
```

Тогда появятся дополнительные пробелы и разрядка действительно будет грамотно выделять слово (рис 7 16)

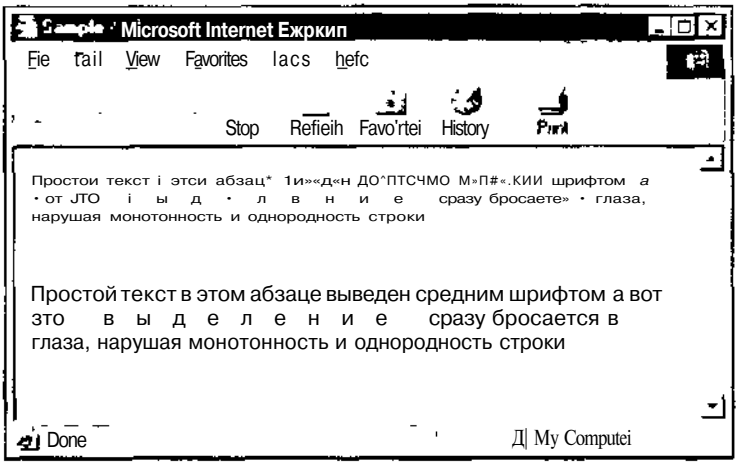


Рис. 7.16. Выделение слова разрядкой с дополнительными пробелами

К сожалению, в браузере Internet Explorer *5.x* некорректно реачизована блоковая модель для строчных блоков, так что для них нельзя устанавливать отступы, поля и рамки. По этой причине в данном бра\зере наше решение работать не будет

Что касается поддержки самого свойства letter-spacing, то сипапия очень хорошая Оно совершенно корректно поддерживается всеми последними версиями браузеров, кроме устаревшего Netscape Navigator 4.x

Все, что мы рассматривали до сих пор. есть настоящее положение дел в среде Интернет. Контроль над шрифтом и текстом на этом и ограничивается, потому что браузеры не поддерживают никаких иных свойств и механизмов. Однако разработчики стандартов на месте не стоят, и в спецификации CSS-2 описано много новых свойств. Если исходить из чисто практических соображений, то рассматривать нереализованные свойства нет смысла. Однако для того, чтобы видеть перспективу и быть готовым быстро изучить CSS-2, когда разработчики браузеров введут его поддержку, надо хотя бы познакомиться со спецификацией. Именно этим мы сейчас и займемся.

Будущеешрифтов:CSS-2

Самой большой проблемой до сих пор является ограниченный набор относительно безопасных шрифтов, тогда как множество остальных шрифтов совершенно невозможно использовать в HTML-документах. Так что же де-.тть. если на компьютере пользователя необходимый шрифт отсутствует? В спецификации CSS-2 этой проблеме уделяется очень много внимания. Как выбрать шрифт для отображения текста в элементе, если он не установлен на компьютере посетителя[1]^ Вот несколько возможностей.

- **Подбор шрифта по имени.** Что, собственно, сейчас и реализовано в браузерах. Теоретически возможна ситуация, когда шрифты на компьютере разработчика и пользователя будут от разных производителей, так что гарнитуры могут слегка не совпадать.
- Интеллектуальный подбор **шрифта по имени.** В этом случае браузер подбирает "наиболее близкий" по виду шрифт. Кроме того, можно описывать шрифт достаточно детально, чтобы приближение было как можно более точным

О **Синтез шрифта.** В этом случае браузер создает шрифт, который будет в точности соответствовать всем описанным параметрам. Здесь надо указывать достаточно большое количество параметров, чтобы созданный шрифт был именно таким, каким надо.

ГЗ Скачивание шрифта. Браузер может взять шрифт с удаленного сервера, наподобие того, как это сейчас происходит с flash-роликами. Единственный недостаток в том, что пользователю придется подождать чуть больше, прежде чем отобразится вся страница.

В спецификации CSS-2 описаны все эти возможности, причем реализуются они посредством правила @font-face, которое имеет следующий синтаксис:

```
@font-face (
  свойство: значение;
  свойство: значение
}
```

Разнообразные возможности выбора шрифта осуществляются с помошью различных свойств правила efont-face. Сейчас мы и рассмотрим эти свойства.

Подбор шрифта по имени

С таким способом выбора шрифта вы уже знакомы, потому что именно эту возможность на сегодняшний день используют браузеры, правда, вне правила @font-face. Для этого применяются свойства, с которыми вы уже хорошо знакомы, однако все же повторим их:

- font-family — задает семейство шрифта;
- O font-style — задает стиль шрифта, такой как italic или normal;
- O font-variant — задает вариант шрифта, выбор осуществляется межд\ small-caps (малыми прописными) и normal;
- O font-weight — задает насыщенность шрифта;
- O font-size — задает размер шрифта;
- font-stretch — это свойство появилось в спецификации CSS-2. С помошью него можно задавать вид шрифта, такой как Condensed или Expanded.

Если мы хотим, чтобы везде в качестве шрифта Futuris использовался шрифт Futuris Condensed с насыщенностью 800, то делается это так:

```
<STYLE TYPE-"text/css">
@font-face {
  font-family: Futuris;
  font-style: normal;
  font-weight: 800;
  font-stretch: condensed}
HI 1
  font-family: Futuris}
</STYLE>
. . .
<H1>Заголовок шрифтом Futuris Condensed</Hl>
<P>Обычный текст</P>
```

Интеллектуальныйподборшрифта

Для того чтобы браузер смог как можно более точно выбрать подходящий шрифт из уже установленных на компьютере пользователя, описание шрифга должно быть очень подробным.

- panose-i — задает числа panose-1. Panose-1 — это промышленный стандарт классификации шрифтов формата TrueType, весьма нетривиальный

и сложный. В этом свойстве шрифт описывается набором из десяти чисел через пробел, которые содержат ключевые атрибуты латинских литер. Начальное значение хтя всех чисел в наборе 0 Данное свойство рекомендуется использовать для латинских шрифтов.

3 stemv — задает ширину вертикальных засечек литер Если значение не указано, то это свойство не используется при подборе шрифта.

• stemh — задает ширину главных (горизонтальных) засечек литер. Если значение не указано, то это свойство не используется при подборе ^шрифта.

О slope — задает угол отклонения засечек от вертикали против часовой стрелки. У тех шрифтов, литеры которых наклонены вправо, величина этого утла отрицательная.

G cap-height ~ задает высоту заглавных букв в шрифте. Если значение не указано, то это свойство не используется при подборе шрифта.

3 x-height — задает высоту строчных букв в шрифте Если значение не указано, то это свойство не используется при подборе шрифта.

"Э ascent — задает высоту самой высокой литеры в шрифте с верхним выносным элементом. Высота вычисляется от базовой линии до крайней точки верхнего выносного элемента. Верхний выносной элемент есть, например, в литере "й". Если значение не указано, то это свойство не используется при подборе шрифта.

П descent — задает высоту самой высокой литеры в шрифте с нижним выносным элементом Высота вычисляется от базовой линии до крайней точки нижнего выносного элемента. Нижний выносной элемент есть, например, в литере "р". Если значение не указано, то это свойство не используется при подборе шрифта.

Пример правила @font-face для интеллектуального подбора шрифта:

```
<STYLE TYPE="text/css">
3font-face {
  panose-1: 2 11 3 2 6 0 3 2 4 0;
  slope: -15;
  x-height: 1;
  font-family: "Some Cool Font"}
HI {
  font-family: "Some Cool Font"}
</STYLE>
. . .
<H1>3аголовок шрифтом "Some Cool Font"</Hl>
<P>0бычный текст</P>
```

Синтез шрифта

Для синтеза шрифта необходимо знать размеры отдельных литер, чтобы не произошло наложение строк и букв друг на друга. Для этого существуют следующие свойства.

П widths — задает ширину литер. В качестве параметров выступают диапазон символов в стандарте Unicode и через пробел ширины соответствующих литер. Например'

```
widths: U+3000-3009 1730 1970 1822
```

обозначает, что в диапазоне U-3OO0-3OO9 (в котором содержится девять литер) первая литера будет иметь ширину 1730, вторая — 1970, а третья и все остальные литеры будут иметь ширину 1822.

- bbox — задает максимальный ограничивающий прямоугольник. Максимальный ограничивающий прямоугольник — это наименьший прямоугольник, который получится, если поместить все литеры шрифта рядом В качестве параметров для данного свойства через запятую указываются четыре координаты нижней левой и правой верхней вершин.
- defimtion-src — описание шрифта может находиться в таблице стилей, а может и в отдельном файле. Данное свойство подключает этот файл. Это может уменьшить трафик, если одно и то же описание шрифта используется разными таблицами стилей.

Вот, собственно, и все способы выбрать необходимый шрифт. Можно рассказать еще о многом, но пока это не имеет особого смысла, т. к. браузеры не поддерживают правило @font-face. На этом разговор о шрифтах можно закончить, и перейти к самой интересной части CSS — блоковой модели.

Глава 8

Блоковая модель

Вот мы и добрались до блоковой модели, которая является совершенно необходимым шагом на пути к верстке без использования таблиц. Если попробовать вывести алгоритм верстки, то можно ограничиться двумя действиями.

1. Создание блоков.
2. Позиционирование блоков,

Естественно, каждое из этих действий можно разбить еще на несколько. В этой главе мы с вами научимся создавать блоки с нужными нам параметрами, а позиционированием вплотную займемся в следующей главе.

Как вы уже знаете, любой HTML-элемент представляет собой так называемый блок. Структура отдельного блока вам тоже уже известна. Она наглядно показана на рис. 8.1.

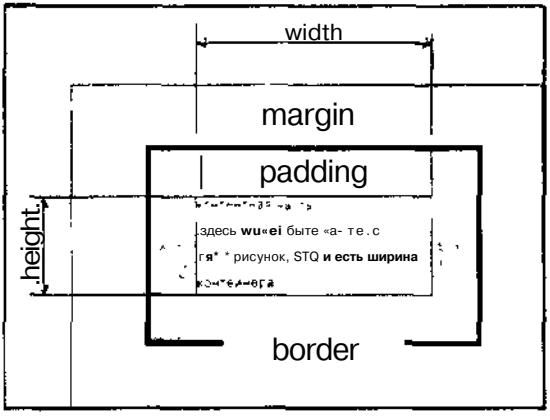


Рис. 8.1. Структура блока в CSS

Блок состоит из контентной части, которая *может быть* окружена отступами, рамками и полями. Каждый блок может иметь фиксированную ширину (width) и высоту (height). Как видно из рисунка, общая ширина контейнера складывается из ширины отдельных компонентов:

Ширина контейнера == margin-left + border-left + padding-left + width + padding-right + border-right + margin-left

Все параметры, представленные на рис. 8.1, можно задать с помощью CSS (причем совершенно аналогичными свойствами), а ширину контентной части можно явно задать с помошью свойства width. *По умолчанию и поля, и отступы, и рамки отсутствуют, т. е. имеют ее/шчину, равную нулю.* В этом случае ширина контентной части равна ширине всего блока. Если же и свойство «xdth не установлено, то ширина контентной части определяется контейнером. Любой элемент имеет контейнер.

Определение

Контейнер элемента — это элемент блокового уровня, являющийся ближайшим предком данного элемента

Например:

```
<BODY>
<P>Элемент P является потомком элемента BODY</P>
</BODY>
```

В данном случае для элемента <p> контейнером является элемент <BODY> Если не задавать явно ширину блока, образованного элементом <p>, и не задавать поля, отступы и рамки, то ширина блока, образованного элементом <p>, будет определяться шириной элемента <BODY>. Вам, наверное, известно. что ширина блока, образованного элементом <BODY>, равна размеру окна браузера Тогда получается, что если для <BODY> все поля установатены в ноль, то ширина блока элемента <p> будет равна ширине окна браузера как на рис. 8.2.

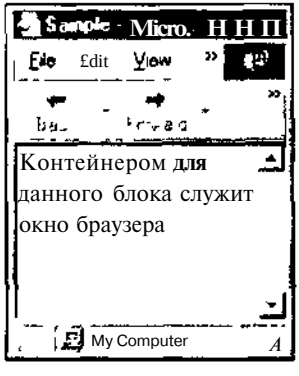


Рис. 8.2. Пример контейнера. Для блока, созданного элементом <P>, контейнером является блок, созданный элементом <BODY>

Для более ясного представления сути контейнера приведу еше один пример с шюженными блоками. Сначала создадим первый блок, который будет

иметь черную рамку толщиной один пиксел и ширину 40% от ширины окна браузера. Кроме того, в данном блоке правый и левый отступы будут равны 2 em. а верхний и нижний — 1 em. Код, который создает такой блок, будет следующим:

```
<STYLE TYPE="text/css">
#blockl (
  border: lpx solid #000;
  padding: lem 2em;
  width: 40%}
</STYLE>
<DIV id-"blockl">
это контент первого блока
</DIV>
```

А на рис. 8.3 показана визуализация этого примера в браузере.

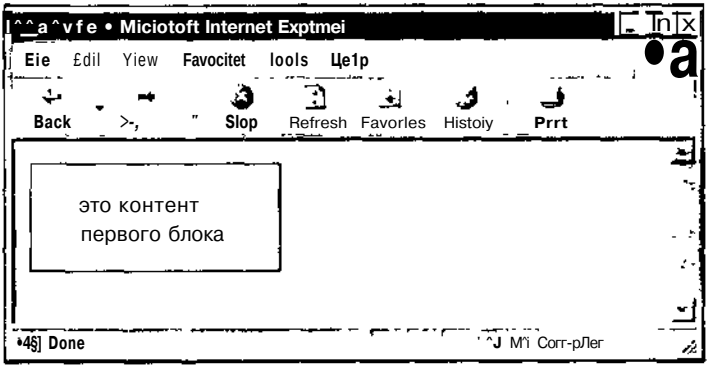


Рис. 8.3. Простейший блок с рамкой толщиной один пиксел и отступами

А теперь создадим второй блок, вложенный в первый. Этот блок будет без отступов, с рамкой толщиной два пиксела, причем ширина блока явно не задана:

```
<STYLETYPE="text/css">
#blockl {
  border: lpx solid #000;
  padding: lem 2em;
  width: 40%}
#Ыоск2 {
  border: 2px solid #000)
```

```
</STYLE>
<DIV id="block1">
это контент первого блока
  <DIV id="bloclc2">
  это второй блок, вложенный в первый
  </DIV>
</DIV>
```

На рис. 8.4 показано как эти два вложенных блока будут выглядеть в браузере.

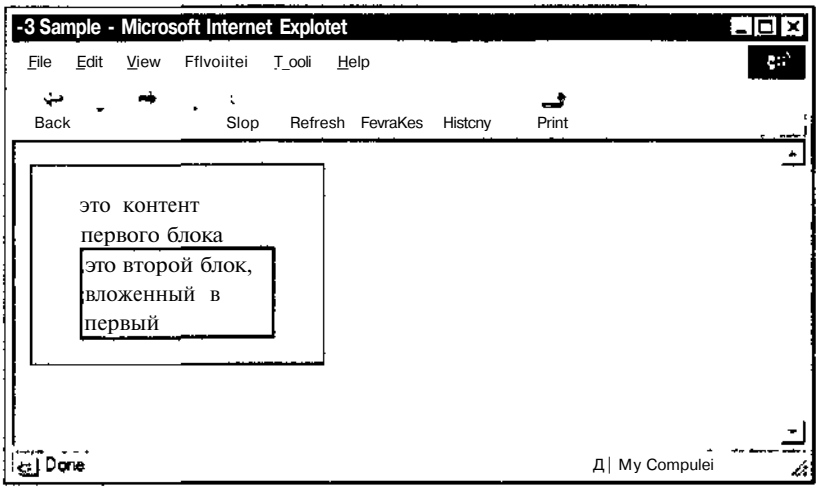


Рис. 8.4. Вложенные блоки. Второй блок имеет рамку толщиной два пиксела и не имеет отступов

Для наглядности на рис. 8.5 представлена схема расположения блоков.

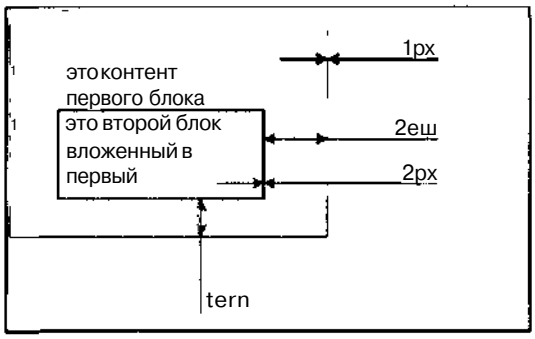


Рис. 8.5. Схема блоков с указанными размерами

Рамка первого блока имеет толщину один пиксел. Правый и левый отступы имеют ширину 2 em. а нижний и верхний — 1 em Содержимым первого блока является фраза "это контент первого блока" и весь второй блок. Во втором блоке отступов нет, потому что в таблице стилей явно они не указаны, а по умолчанию отступы равны нулю, по этой причине контент второго блока вплотную прилегает к рамке. Обратите внимание на то, что ширина второго блока не указана, а контейнером *х\я* него является первый блок. Поэтому *ширина второго блока равна ширине контентной части первого блока*.

Блоки в HTML

Давайте разберем причины, по которым вообще начали использовать таблицы при верстке страниц. Как я упоминал в начале главы, вся верстка сводится к двум операциям: созданию и позиционированию блоков.

G В HTML можно создать блок. Действительно, любой абзац или заголовок яа!яется блоком. Так что в следующей конструкции есть два блока:

```
<H1>эаголовок</H1>
<P>простой текст</P>
```

Первый блок — это заголовок, а второй блок — это простой текст.

П Кроме самого факта создания блока, нам надо задать ему определенные параметры, хотя бы ширину и высоту. А вот этого язык HTML делать не позволяет, т. е. вы не сможете создать блок ограниченной ширины с помощью элемента <p>. Единственными элементами, которые позволяют ограничивать ширину блока, являются элементы <TABLE> И <TD>. Очевидно, что *без использования CSS создать блок с ограниченной шириной можно только с помощью таблицы,* тогда как с использованием CSS это делается совершенно элементарно с помощью свойства width. В табл. 8.1 приведено сравнение способов создания блоков в HTML и CSS.

Таблица 8.1. Сравнение процесса создания блока с заданной шириной средствами HTML и CSS

HTML	CSS
<TABLE CELLSPACINGS" CELLPADDING-"0" BORDER="0" WIDTH="200"> <TRXTD> Блок шириной 200 пикселов. Почувствуйте разницу в коде. </TD></TR> </TABLE>	P { width: 200px} <P>Блок шириной 200 пиксе-лов. Почувствуйте разницу в коде.</P>

Итак, блок с ограниченной шириной в HTML можно создать только с использованием таблицы. Обратите внимание, чго с помощью одной таблицы можно создать несколько блоков, потому что для создания блока достаточно элемента <TD> С заданной шириной, а таблица может содержать сколько угодно ячеек.

Осталось рассмотреть проблему позиционирования блоков. Подробно CSS-позиционирование мы разберем в следующей главе, а здесь я приведу лишь небольшое сравнение. С помощью CSS мы можем свободно позиционировать блоки практически как нам угодно. В HTML нет механизма позиционирования, но выход нашли все в тех же в таблицах: *блоки позиционируют, размещая их в ячейках таблицы*. Создавая сложные таблицы, можно добиться того, чтобы ячейки находились в нужных местах экрана, и в эти ячейки помещать блоки, созданные посредством еще одних таблиц, или же самих ячеек главной таблицы. Пример позиционирования двух блоков (каждый из которых яатяется колонкой) средствами HTML и CSS представлен в табл. 8.2.

Таблица 8.2 *Сравнение процесса позиционирования блоков средствами HTML и CSS*

HTML	CSS

<TABLE BORDER--i" CELLPADDING="IO" CELLSPACING="Q" WIDTH="60%" HEIGHT="40%">
<TR VALIGN-"top">
<TD \^ЮТН="20%">левая колонка используется для навига-
ции</TD>
<TD И1ПТН="40%">здесь находится основной контент</TO>
</TR>
</TABLE>
	ttleft {
	float: left;
	border: lpx solid #000;
	width: 20%;
	height: 40%;
	padding: 10px)
	#right {
	float: left;
	border: lpx solid #000;
	width: 40%;
	height: 40%;
	padding: 10px}
	
	<DIV id="left">
	левая колонка используется для навигации
	</DIV>
	<DIV id="right">
	здесь находится основной контент

На первый взгляд, позиционировать блоки с помощью CSS сложнее, и это действительно так для простейшего расположения блоков. Однако если структура более сложная, то превосходство CSS сразу становится заметно (в следующей главе вы сами в этом убедитесь).

Виды блоков

Мы оставим пока вопрос позиционирования и вернемся к самим блокам. Введем несколько определений.

Определение

Элемент будем называть блочным, если он визуально форматируется в виде структурной единицы, т е представляет собой абзац, заголовок или другую структурную единицу,

В HTML такие элементы обычно начинаются и заканчиваются символом перевода строки (например, элементы <p>, <HI>, <TABLE>). Следовательно, блочные элементы не могут располагаться в одной строке.

Определение

Элемент будем называть строчным, если он не формирует структурных единиц и выводится линейной строкой

В HTML это обычно элементы форматирования текста (например, элементы <в>, <EM>, <CODE>). Естественно, два строчных элемента могут располагаться в одной строке.

Заметьте, что *блок* и *блоковый элемент* это не одно и то же. На самом деле два вышеприведенных определения в равной мере относятся и к блокам. Иначе говоря, блоки бывают *структурными* (порождаемые блоковыми элементами) и *строчными* (порождаемые строчными элементами). Примеры блоков различного типа показаны на рис. 8.6.

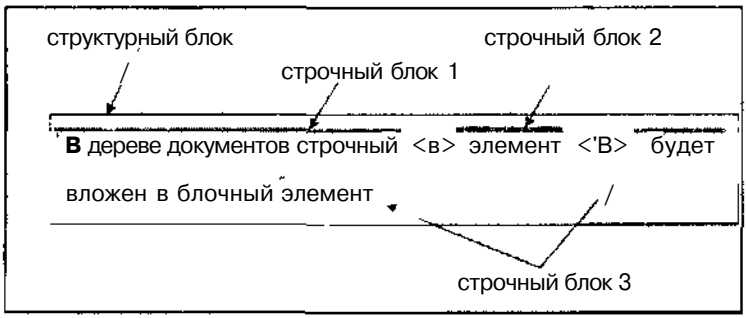


Рис. 8.6. Структурные и строчные блоки

Здесь есть один блочный элемент <?>, который порождает один структурный блок (на рисунке блок выделен рамкой), и три строчных блока. Причем два из них безымянные, т. е. они порождаются не строчными элементами, а самим блочным элементом <?>. Еше один строчный блок порождается строчным элементом <в>.

Структурные и строчные блоки должны различаться между собой только тем, что первые образуют структурные единицы, а вторые — нет Однако на практике в некоторых браузерах различий гораздо больше. Например, Е браузере Internet Explorer 5 некорректно реализованы строчные блоки. Это выражается в том, что для них невозможно задавать поля, рамки и отступы Если вы напишете на элемент <EM> СТИЛЬ

```
EM {
  border: Ipx solid #000}
```

то рамки вокруг текста, который находится в элементе <EM>, не появится, а согласно спецификации рамка должна быть.

Кстати, на основе рамок можно делать интересные выделения. Например, код

```
EM \
  border: Ipx dotted #000;
  font-style: normal;
  padding: Opx 3px)
. . .
<P>Обыгчный текст с <EM>выделенным</EM> словом</P>
```

обеспечит выделение текста внутри элемента <EM> пунктирной рамкой, как показано на рис. 8.7.

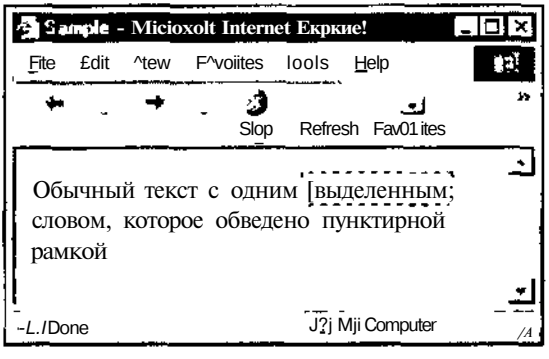


Рис. 8.7. Выделение текста пунктирной рамкой. Достаточно перспективный способ выделения в тексте

Вообще, веб представляет собой совершенно уникальную среду, в которой законы постоянно изменяются. Взять хотя бы логическое выделение текста. Первое время оно в основном осуществлялось курсивом с помощью тега <:>, потом стало популярным выделение полужирным с помощью тега <в>, сейчас входит в моду выделение фоном, т. е. изменением цвета фона выделяемого элемента **с помощью** свойства background-color. Возможно, и выделение с помощью рамок в скором времени займет **свое** место в книгах по веб-дизайну.

Ширина и высота блока

И снова вернемся к созданию блоков. Нам надо создавать не просто произвольные блоки, а блоки определенного размера с заданной шириной и высотой. В CSS общая ширина блока складывается из ширины контентной части и суммарной ширины полей, рамок и отступов. Ширина контентной части задается свойством width. Вот пример блока шириной 200 пикселов:

```
#leftb (
  width: 200px}
. . .
<DIV id="leftb">
Блок шириной 200 пикселов
</DIV>
```

Но в браузере Internet Explorer *5.x* свойство width трактуется несколько иначе. Оно определяет не только ширину контента, но включает в себя ширину отступов и рамок. Это четко видно на рис. 8.8.

Из-за такой неправильной реализации поддержки свойства width общая ширина блока в различных браузерах будет разной. Допустим, мы создали такой блок:

```
#box {
  border: 20px solid #000;
  padding: 40px;
  background: #CCC;
  width: 300pxf
```

Из рис. 8.9 ясно, что в различных браузерах этот блок выглядит по-разному.

Сверху на этом рисунке показано, как данный блок будет отображаться браузерами, которые корректно поддерживают блоковую модель стандарта CSS-1. К ним относятся Opera 5+, Netscape 6.v, Mozilla 0.9.x Как видите, ширина области контента составляет ровно 300 пикселов, как и было задано в таблице стилей, а *общая ширина блока* будет 20-40+300+40+20=420 пикселов.

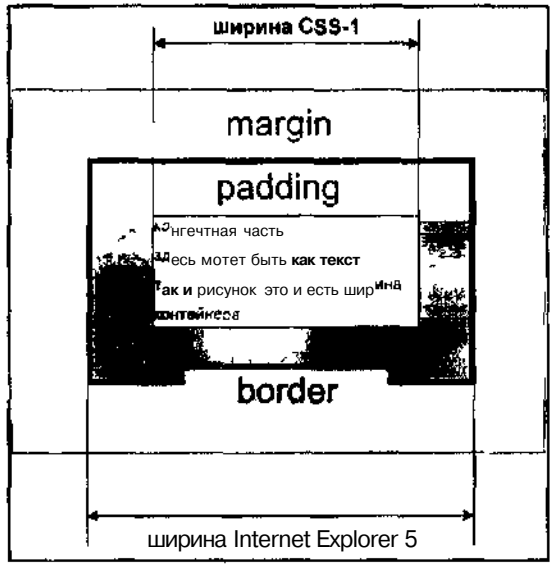


Рис. 8.8. Неверная трактовка свойства width браузером Internet Explorer 5.x

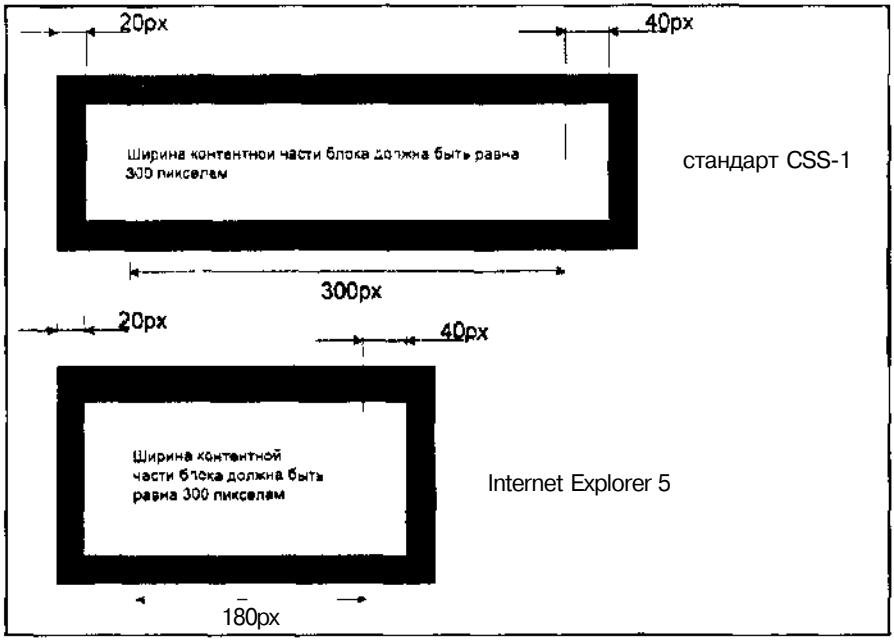


Рис. 8.9. Ширина блока в браузерах корректно поддерживающих стандарт CSS-1 (сверху) и в браузере Internet Explorer 5 x (снизу)

Внизу на рис. 8.9 показано, как данный блок будет отображаться в браузере Internet Explorer 5.v. В нем *общая ширина блока* б\дет 300 пикселов, а ширина области контента будет равна ISO пикселам, потому что указанная в таблице стилей ширина содержит еще отступы и рамки. Очевидно, что в этом случае под контент остается 300-20-40-40-20=180 пикселов.

Надо отметить, что если у блока нет ни рамок, ни отступов (то есть они имеют нулевую ширину), то такой блок будет одинаково отображаться всеми браузерами. Однако подобная ситуация возникает не всегда, и порой задать рамки или отступы совершенно необходимо Как же разрешить эту проблему? Вообще понятно, что для одних браузеров нам надо задать одну ширину, а для Internet Explorer 5.x другую ширину. Если вернуться к нашему примеру, то для браузера фирмы Microsoft надо задать значение ширины 420 px. То есть, нам просто надо разделить объявления для разных браузеров. Это можно сделать двумя способами:

- поместить таблицы стилей в разные файлы и подключать их в зависимости от браузера с помощью JavaScript (как это делается, будет подробно объяснено в *главе 10)*\
- *П* сделать так, чтобы браузеры "видели" *разные* объявления ширины в одной таблице стилей.

Устранение различий блоковых моделей

Мы с вами сейчас вплотную займемся вторым способом, потому как первый совершенно универсальный и подходит ко всем случаям жизни. Для выбора элемента, к которому применяются стили, используются селекторы. Селекюры из спецификации CSS-1 вам уже известны, однако новые версии браузеров поддерживают многие виды селекторов из спецификации CSS-2. Например, в CSS-2 есть селектор, позволяющий применять стили к элементам, которые являются прямыми потомками другого элемента. Пусть, например, у нас есть код:

```
<BODY>
  <P>Прямой потомок элемента BODY</P>
  <DIV id-"main">
    <P>Непрямой потомок элемента BODY</P>
  </DIV>
</BODY>
```

Нам надо сделать так, чтобы все элементы <p>, являющиеся прямыми потомками элемента <BODY>, отображались серым цветом. Можно, конечно, вручную расставить для них классы, но это непродуктивно. Проще такую операцию осуществить с помощью селектора:

```
BODY > P •
  color: #CCC}
```

Тогда первый абзац в примере будет выводиться серым цветом, а второй — нет.

Так вот, браузер Internet Explorer 5.x не понимает такого селектора, а браузеры Netscape 6.Y, Mozilla, Opera 5+ понимают. Поэтому мы можем для него задать одно значение ширины блока, а для остальных — другое. Код в таблице стилей будет таким:

```
#Box {
  border: 20px solid #000;
  padding: 40px;
  background: #CCC;
  width: 420px)
BODY > #box {
  width: 300px(
```

Сначала все браузеры установят ширину блока box равной 420 пикселам, а потом все браузеры, кроме Internet Explorer 5.v, установят ширину этого блока в 300 пикселов. Все бы хорошо, но возможна ситуация, когда браузер корректно поддерживает блоковую модель, но не поддерживает селекторы из спецификации CSS-2. Например, именно таким браузером является Internet Explorer 6.x, так что такое решение неприемлемо. Данный браузер установит значение ширины для некорректной блоковой модели, т. е. 420 пикселов, а переобозначить это значение на 300 пикселов |КЛ1]уже не сможет. Так как этот браузер корректно поддерживает блоковую модель (в режиме совместимости со стандартами), он отобразит блок совсем не таким, каким нам надо.

Еще одну реализацию второго способа, придумал достаточно известный разработчик стандарта CSS-3, сотрудник компании Microsoft *Таншек Целик (Tantek Celik)*. Решение такое:

```
#Box (
  border: 20px solid #000;
  padding: 40px;
  background: #CCC;
  width: 420px;
  voice-family: "\"}V";
  voice-family: inherit;
  width: 300px}

HTML>BODY #box {
  width: 300px}
```

Сейчас я объясню, почему это работает. В начале описания стилей данного блока мы задаем все объявления, кроме ширины, а ширину задаем в конце.

Тогда все браузеры сначала установят значение width для блока box в 420 пикселов. Потом следует конструкция

```
voice-family: "\"f\"";
```

Это свойство из звуковых таблиц стилей стандарта CSS-2, так что браузеры его вообще не должны воспринимать, т. е. просто игнорировать и обрабатывать объявления дальше. Однако браузер Internet Explorer *5.x* некорректно обрабатывает эту конструкцию. Он считает, что на этом месте блок объявлений для селектора #box заканчивается. А все остальные браузеры продолжат считывать объявления, и перепишут первоначальное значение ширины на новое, которое равно 300 пикселам.

Но не только браузер Internet Explorer 5.x некорректно обрабатывает свойство voice-family. Например. Opera 5+ тоже может иметь с ним проблемы. Но Opera 5+ прекрасно понимает многие селекторы из спецификации CSS-2. Вот именно пя подобных браузеров используется указание правильной ширины с помощью селектора CSS-2:

```
HTML>BODY #box {
  width: 300px}
```

Итак, обшая ситуация такая.

П Браузер Internet Explorer 5.x установит значение свойства width равным 420 пикселов, потому что он не понимает селекторов CSS-2 и некорректно обрабатывает свойство voice-family.

• Браузеры Mozilla и Netscape 6.x установят значение свойства width ранным 300 пикселов, потому что они игнорируют свойство voice-family, и к тому же понимают селектор CSS-2.

О Браузер Opera 5.x установит значение свойства width равным 300 пикселов, потому что он некорректно обрабатывает свойство voice-family, но понимает селектор CSS-2.

У нас остался неучтенным браузер Internet Explorer 6.x Он корректно обрабатывает свойство voice-family (то есть игнорирует его), и не понимает селектора CSS-2. Другими словами, если у него правильно реализована блоковая модель, то все будет нормально. Однако в этом и заключается трудность. Все дело в том, что в браузере Internet Explorer 6.x можно устанавливать модель поведения браузера. В одном случае блоковая модель будет точно такой же. как и в Internet Explorer 5.x (это так называемый режим обратной совместимости), а в другом — соответствующей стандарту CSS-1 (это режим совместимости со стандартами). Переключение между режимами осуществляется с помощью инструкции IDOCTYPE, О которой мы немного говорили в *главе 1.* Вообще формат данной инструкции нам не особенно интересен, а вот в каких случаях происходит переключение, знать важно.

В инструкции !DOCTYPE можно задавать URL объекта, в котором описывается тип документа, а можно и не задавать. Например:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<IDOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

В первом случае URL не определен, а во втором определен.

Ниже представлена табл. 8.3, в которой указано, какие значения IDOCTYPE переключают браузер Internet Explorer *6.x* в стандартный режим.

Таблица 8.3. *Значения !DOCTYPE для браузера Internet Explorer 6.x*

Значение !DOCTYPE	**Режим (URL задан)**	**Режим (URL не задан)**
Не установлен	Обратной совместимости	Обратной совместимости
HTML (без указания версии)	Обратной совместимости	Обратной совместимости
HTML 2.0	Обратной совместимости	Обратной совместимости
HTML 3.0	Обратной совместимости	Обратной совместимости
HTML 4.0	Стандартный	Обратной совместимости
HTML 4.0 Frameset	Стандартный	Обратной совместимости
HTML 4.0 Transitional	Стандартный	Обратной совместимости
HTML 4.0 Strict	Стандартный	Стандартный
XHTML	Стандартный	Стандартный

Таким образом, в приведенном выше примере только вторая инструкция переключит браузер в стандартный режим, потому что значение IDOCTYPE равно HTML 4.0 Transitional и задан URL. тогда как первая — нет, потому что в ней URL не задан.

С высотой блока ситуация совершенно аналогичная, и проблема устраняется точно так же с использованием свойства voice-family.

Когда ширина контента превышает ширину блока

Создавать блоки заданной ширины мы уже умеем, но остаются нерешенными еще некоторые вопросы. Один из них звучит примерно так: "Что будет, если ширина контента окажется больше, чем ширина блока, указанная посредством свойства `width`?"

Вообще варианта два:

- блок растянется, чтобы вместить в себя весь контент;
- ширина блока останется без изменения, но контент перекроет блок, т. е. надожится на другой блок, прилегающий к данному блоку.

В спецификации CSS по этому поводу ничего не сказано, так что будем рассматривать этот вопрос на практике. Создадим блок небольшой ширины, в котором есть достаточно длинное слово. Блок у нас будет с фоном серого цвета, рамкой из точек толщиной в два пиксела и шириной в 150 пикселов. Вот соответствующий код:

```
<STYLE TYPE="text/css">
  #small {
  background: #CCC;
  font: lem Verdana, sans-serif;
  border: 2px dotted black;
  padding: 0.5em;
  width: 150px}
</STYLE>
. . .
<DIV id="small">
Елок, в котором присутствует самоедлинноевмиреслово
</DIV>
```

Что нам покажет браузер Internet Explorer 6.v видно на рис. 8.10.

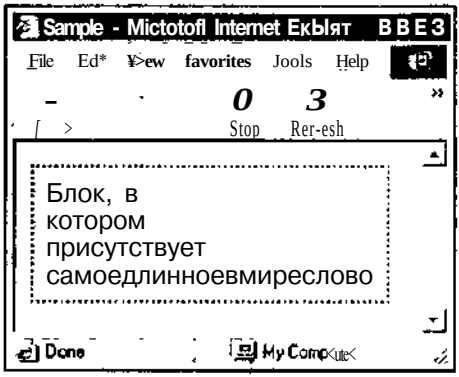


Рис. 8.10. Вид блока, в котором ширина контента превышает ширину самого блока (браузер Internet Explorer 6x)

Как видим, блок растянулся, чтобы вместить в себя весь контент. А на рис. 8.11 показан тот же код в браузере Netscape 6.2.

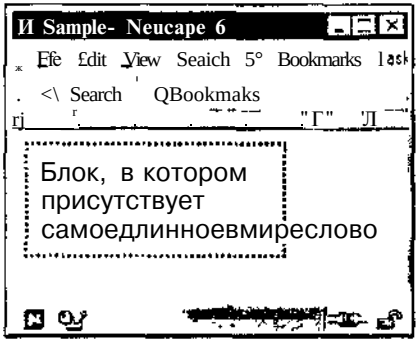


Рис. 8.11. Вид блока, в котором ширина контента превышает ширину самого блока (браузер Netscape 6.2)

Аналогично браузеру Netscape 6.x поступают браузеры Opera 5+ и Mozilla, так что имеет место проблема несовместимости. Причем приемлемого решения у этой проблемы нет. Здесь надо остановиться и подумать, насколько эта проблема серьезна. Для этого возьмем типичный сайт и посмотрим размеры сю основных блоков. Рассмотрим, к примеру, самый распространенный способ портальной верстки в три колонки. Тогда у нас фактически будет три блока:

- узкая левая колонка;
- достаточно широкая центральная колонка;

П узкая правая колонка.

Самое маленькое разрешение, для которого адаптируют сайт, 800x600, так что общая ширина страницы будет около 750 пикселов. Обычно под центральную колонку отводится 50%, так что на остальные остается по 25%. Получается, что ширина левой и правой колонок будет около 188 пикселов, *Максимальный* размер шрифта на сайтах примерно 16 пикселов, а ширина у буквы будет около 10 пикселов. Так что слово должно состоять их 19 букв, чтобы превысить ширину колонки. Заметьте, что это при очень большом размере шрифта (обычно он не превышает И пикселов) и при достаточно маленьком разрешении монитора. В русском языке не так уж и много слов из 19 букв. Конечно, если между словами вместо обычного пробела будет стоять символ неразрывного пробела snbsp;, то браузер будет воспринимать эти слова, как одно. Поэтому при верстке с помощью блоков паю очень осторожно позволять создателю контента пользоваться неразрывным пробелом. Напишет он вот такое предложение

КакЬпЬзр/з-то&пЬзр;чудесно, finbep;когдайпЬзр;можноблЬвр;пользоваться4пЬ8р;неразрывным&пЪзр;пробелом!

в колонку шириной 200 пикселов, растянется эта колонка в браузерах фирмы Microsoft пикселов на 600 и так искорежит дизайн сайта, что у руководителя проекта случится микроинфаркт.

Впрочем, с обычными таблицами ситуация совершенно аналогичная, так что все давно привыкли осторожно пользоваться неразрывным пробелом. Вообще, длинные слова встречаются достаточно редко, и с этой точки зрения поведение браузера Netscape *Ь.х* более приемлемо, чем поведение браузеров фирмы Microsoft. Очевидно, что когда одно слово залезет на другую колонку, это менее болезненно, чем когда вся колонка растянется на несколько процентов. Однако с точки зрения привычки все совершенно наоборот. Потому что в браузерах фирмы Microsoft блоки ведут себя точно так же, как и таблицы, а с таблицами все HTML-верстальщики знакомы очень хорошо. Короче говоря, проблема разного повеления не является критичной, однако ее всегда надо держать в уме, чтобы не пугаться, когда увидите, как браузер Netscape *6.x* отображает блок размером 150x100, в который вместо одной маленькой новости внесли объемный пресс-релиз.

Совершенно аналогичным образом поступают все вышеперечисленные браузеры и в том случае, когда контент превышает высоту блока.

Для манипулирования высотой и шириной блока есть еще несколько интересных свойств из спецификации CSS-2:

- rr.ir.-width — задает минимальную ширину блока;
- max-width — задает максимальную ширину блока;
- min-height — задает минимальную высоту блока;
- max-beight — задает максимальную высоту блока.

Сразу скажу, что поддерживаются они браузерами Netscape 6.x и Opera 5+, но не поддерживаются браузерами Internet Explorer любых версий, кроме шестой. Да и Internet Explorer *6.x* поддерживает только свойство min-height, причем только к элементам <TR>, <TH>, <TD>. Однако именно это свойство заслуживает особого внимания по той причине, что в браузере фирмы Microsoft без его использования иногда невозможно применять свойство table-layout: fixed.

Вообще, свойство table-layout описано в спецификации CSS-2. Оно устанавливает алгоритм, по которому будет обрабатываться таблица. Существует два таких алгоритма.

- Первый из них достаточно медленный и установлен по умолчанию (table-layout: auto). Медленный он по той простой причине, что пока *все содержимое таблицы* не будет загружено браузером, он не сможет просчитать ее параметры. Например, в таблице три столбца и четыре строки, т. е. 12 ячеек, а в каждой ячейке содержится рисунок. Получается 12 рисунков. Пока браузеру не станут известны размеры всех рисунков, он не сможет просчитать ширину и высоту таблицы.
- Второй алгоритм достаточно быстрый и устанавливается, например, так:

```
<TABLE STYLE="table-layout: fixed" WIDTH='M00">
```

Данный алгоритм не учитывает содержимое отдельных ячеек при вычислении ширины таблицы и ячеек. Он использует значения ширины таблицы, ширины столбцов, рамок и расстояний между ячейками. По этому алгоритму ширина столбцов таблицы вычисляется в следующем порядке:

- используя значение атрибута WIDTH элементов <COL>;
- используя значение атрибута WIDTH элементов <TD> первого ряда таблицы;
- если данные значения не установлены, то ширина столбцов вычисляется соответственно контенту ячеек.

Следовательно, для быстрой обработки таблиц надо указывать ширины столбцов с помощью элементов <COL>. В приведенном ниже примере таблица разбита на три колонки:

```
<TABLE STYLE="table-layout: fixed" WIDTH="100">
<COL WIDTH="100"><COL WIDTH="200"><COL WIDTH="100">
```

Подбираемся ближе к свойству min-height. Контент в столбце фиксированной таблицы будет переноситься, если это возможно, а если нет, то обрезаться. Практически если в стилях не прописывать table-layout: fixed, то ячейка таблицы будет растягиваться, как это было всегда во всех браузерах. Если же прописать это свойство, то ячейка будет строго заданного размера, а весь контент, который превышает этот размер, будет беспощадно обрезаться. Вот тут-то нам и пригодится свойство min-height. Ниже приведен код таблицы, у которой в первой ячейке задана высота 40 пикселов, а во второй ячейке задана минимальная высота 140 пикселов. В результате в первой ячейке содержимое обрезается, а во второй — нет:

```
<TABLE BORDER="1" STYLE="table-layout: fixed; width: 200px">
  <TR>
    <TD STYLE="height: 40px; background: #FC0">Высота данной ячейки равна
              40 пикселам. Так что контент в нее не влезает
              и оСрезается</TO>
  </TR>
  <TR>
    <TD STYLE="min-height: 140px; background-color: #09F">
       Минимальная высота данной ячейки равна 140 пикселам.
       Так что контент в нее влезает и не обрезается
    </TD>
  </TR>
</TABLE>
```

Этот код в браузере Internet Explorer 6.x выглядит так, как показано на рис. 8.12.

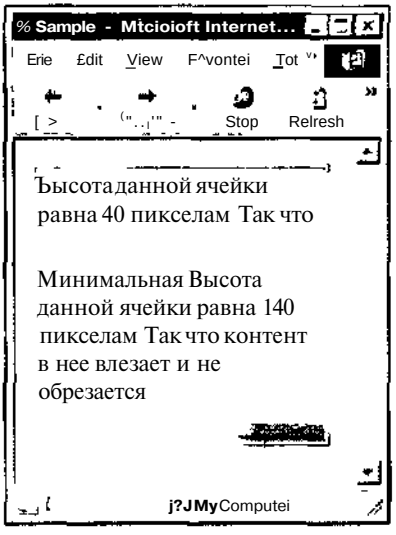


Рис. 8.12. Фиксированная таблица В первой ячейке свойство min-height не указано, а во второй оно присутствует

Вообще, свойство table-layout: fixed очень полезно при табличной верстке, потому как ускоряет обработку и вывод таблиц на экран монитора, что хорошо. Реальной пользы от него несколько меньше, т. к. работает оно только в бра\jера\ фирмы Microsoft (браузер Netscape 6.*x* тоже его поддерживает, но порой его реакция непредсказуема), да и табличная верстка в скором времени со сцены уйдет. Однако пока этого не произошло, обработку таблиц лучше ускорять.

Типы блоков

В глобальной системе классификаций блоки бывают только двух видов: структурные и строчные. Однако и те, и другие имеют множество типов, о которых мы с вами сейчас и поговорим.

В CSS тип блока определяется свойством display. Это свойство в спецификации CSS-1 может принимать четыре значения: none, inline, block, list-item. Однако в спецификации CSS-2 значений гораздо больше, и некоторые браузеры их в той или иной степени поддерживают. Тем не менее, поддержка эта часто некорректная и неполная, так что о них я скажу совсем немного, а пока подробно рассмотрим именно значения, прис\тств\юшне в CSS-1.

display:none

Отключает отображение элемента в браузере. В CSS существует свойство visibility, с помощью которого тоже можно скрыть элемент, однако раз-

ница между этими свойствами есть. Дело в том, что при использовании объявления visibility: hidden элемент просто становится невидимым, но присутствует на странице и занимает определенное место, тогда как при использовании объявления display: none элемент вообще не присутствует на странице, он удаляется из нормального потока, согласно которому браузер форматирует страницу.

Для чего может понадобиться это объявление? С помошью JavaScript можно динамически показывать и скрывать элементы, так что есть возможность организовать несложное иерархическое меню, подразделы которого будут разворачиваться при нажатии на ссылку раздела и сворачиваться обратно, если на ссылку нажать еше раз. Такое меню значительно ускоряет перемещение по сайту, да и пользователи к нему привыкли, потому что по своей сути оно ничем не отличается от дерева файлов R проводнике ОС Windows. Правда, такое меню нельзя реализовать для браузера Opera 5+, потому что у пего объектная модель документа неразвита, и динамически изменять значение свойства display нельзя. Что касается CSS, то никаких проблем у браузеров с объявлением display: none нет.

display: block

Элемент с таким объявлением будет форматироваться как структурный блок. Вообще многие элементы по умолчанию имеют объявление display: *block,* например <HI>, <P>, <TABLE>. Использовать это объявление целесообразно только вместе с объявлением display: none. Действительно, его можно применить для восстановления элемента на странице, который до этого был убран с помощью display: none. Конечно, можно делать строчные элементы блочными, но особого смысла в этом нет.

display: inline

Элемент с таким объявлением будет форматироваться как строчный блок. Точно так же многие элементы по умолчанию имеют объявление display: inline, например <i>, <CITE>, <EM>, <STRONG>. Использовать значение inline можно для того, чтобы превратить блочные по умолчанию элементы в строчные элементы. Например, можно использовать заголовки шестого уровня таким образом:

```
<STYLE>
  BODY {
    font: lem Verdana, sans-serif}
  P. in {
    display: inline•
  H6 {
    font: bold lem Verdana;
```

```
    display: inline}
-/STYLE>
  . .
Н6>Заголовок: </Н6>
P class=in>ТеКСТ подраздела начинается в той же строке</P>
```

Гогда в браузере мы увидим то, что на рис. 8.13.

Рис. 8.13. Строчный заголовок <Н6>. Делается с помощью объявления display: inline

Надо сказать, что данное свойство не поддерживается старыми браузерами, такими как Netscape Navigator 4.x и Internet Explorer 4.x, однако это некритично.

display: list-item

Делает элемент частью списка, т. е. вначале каждого абзаца добавляется кружочек или любой другой маркер, обозначающий пункт списка. С помощью такого объявления можно любой HTML-элемент превратить в список. Например, абзац:

```
P {
  display: list-item)
```

Однако никакой особой ценности это не имеет, потому что в HTML есть элементы <UL>, <OL>, <LI>, которые и задают списки. Так что гораздо логичнее делать список именно средствами HTML, а не средствами CSS, тем более данное значение свойства display не поддерживается браузером Internet Explorer 5.x и некорректно поддерживается браузером Netscape 6.x Другое де- [о, что в CSS есть несколько свойств, которые значительно увеличивают контроль нал представлением списков. Вот о них немного и поговорим.

list-style-type

Позволяет задавать маркер списка, т. е. то, что будет выводиться перед каждым пунктом списка. Может принимать следующие значения:

- disc — закрашенный кружочек (является значением по умолчанию);
- circle — незакрашенный кружочек;

П square — закрашенный квадратик;

O decimal — арабские цифры (1, 2, 3 и т. д.):

O lower-roman — римские цифры, обозначаемые маленькими буквами (i, ii. Hi, iv);

O upper-roir.an — римские цифры, обозначаемые большими буквами (I, 11, III, IV);

O lower-alpha — маленькие латинские буквы (a, Ь, c);

O upper-alpha — большие латинские буквы (A, B, C);

П попе — отключает маркеры.

Данное свойство достаточно корректно поддерживается браузерами. Однако некоторые проблемы есть. Так, например, браузер Internet Explorer 5.x не понимает значения попе, хотя список без маркеров можно сделать с помощью HTML элементами <DL> и <DD>. Кроме того, браузеры Netscape 6..v и Mozilla поддерживают еше несколько значений: lower-greek, armenia.-. georgian, cjk-ideographic, hiragana. hiragana-iroha, katakana. katakana-iroha.O, которые, впрочем, нам особо не пригодятся.

Средствами HTML тоже можно сделать список со всеми перечисленными выше пилами маркеров (табл. 8.4). Значения устанавливаются в атрибуте TYPS. однако в спецификации HTML 4.0 он помечен как нежелательный, так что лучше для установки типа маркера пользоваться свойством list-style-type.

Таблица 8.4. *Установка вида маркера средствами HTML и CSS*

HTML	CSS
<UL TYPE="circle"> <Ы>первый пункт</Ы> <Ы>второй пункт</Ы> </UL>	list-style-type: circle} . . . <UL> <Ы>первый пункт</lД> <Ь1>втсрой пункт</lЛ>

Но в качестве маркера можно использовать любой рисунок, и для этого служит еще одно свойство.

list-style-image

В качестве параметра используется URL к изображению, которое вы хотите сделать маркером. Например, вы нарисовали стрелочку, сохранили ее в файл arro\v.gif. и файл положили в каталог 1. Тогда для всех списков задать маркер в виде этой стрелочки можно так:

```
LI I
  list-style-image: url(i/arrow.gif)}
```

В браузере Internet Explorer *S.x* есть некоторая проблема с размером рисунков. Если вы захотите установить свойство line-height, то маркеры могут не отобразиться, если их высота больше 30 пикселов. Так что для безопасности больших маркеров не делайте. Кроме того, это свойство не понимает браузер Netscape Navigator 4.x

Есть еше свойство.

list-style-position

Оно позволяет устанавливать обтекание маркеров. Может принимать два значения: inside и outside (рис. 8.14).

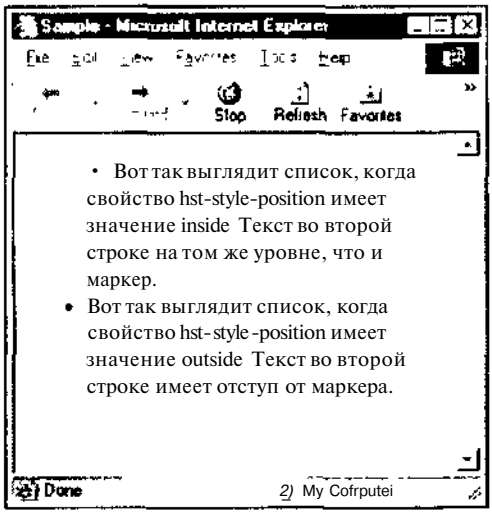


Рис. 8.14. Два списка с различным значением свойства list-style-position. Первый список имеет значение inside, а агорой список имеет значение outside

Проблем с этим свойством у браузеров нет, так что им можно пользоваться смело. Кроме того, есть свойство list-style, которое служит для сокращенной записи стилей. Формат у него такой:

```
list-style: <keyword> Ii <position> I I <url>
```

Например, список с маркерами disc и обтеканием inline можно записать так:

```
ы {
  list-style-type: circle;
  list-style-position: inlinel
```

А можно и так:

```
LI {
  list-style: circle inline}
```

В спецификации CSS-1 больше нет возможных значений свойства display, однако производители некоторых браузеров в этом вопросе продвинулись несколько дальше. Особенно радуют Opera 5+ и Netscape 6.x, а вот Internet Explorer несколько подкачал. Из более-менее значимых свойств можно выделить еще одно.

display: table

В стандарте CSS-2 реализовано визуальное представление таблиц с помощью стилей. Надо заметить сразу, что принципиальных отличий таких таблиц от таблиц в HTML 4.0 нет. Как видно из табл. 8.5, почти каждому тегу соответствует какой-нибудь атрибут свойства display.

Таблица 8.5. *Сравнение табличных свойств CSS с тегами HTML*

Значение свойства display в CSS	Аналогичный тег HTML	Что делает?
table	<TABLE>	Определяет элемент как таблицу блокового уровня со всеми вытекающими последствиями
inline-table	Нет	Определяет элемент как таблицу строкового уровня
table-row	<TR>	Определяет элемент, как строку **таблицы**
table-row-group	<TBODY>	Определяет элемент как группу нескольких строк (надо сказать, абсолютно не нужен)
table-column	<COL>	Определяет **элемент, как** столбец таблицы
table-cell	<TD>, <TK>	Определяет элемент, как ячейку таблицы

Я пропустил всякие малополезные свойства, типа tabie-focter-group (в **HTML** <TFOOT>) **и т. п.** Но это большого значения **не** имеет, потому **что в** браузере Internet Explorer все равно CSS-таблицы не поддерживаются и за-

острять на них внимания нет особого смысла. Реализуем таблицу с помощью CSS. Для наглядности все блоки, образованные элементами <DIV>, заключим в рамку толщиной один пиксел, а все свойства display укажем непосредственно в HTML-коде:

```
<STYLE>
DIV (
  border: 1px solid #000}
</STYXE>
. . .
<DIV STYLE="display: table; margin: 5px; padding: 15px">
display: table
  <DIV STYLE="ctisplay: table-row">
    <DIV STYLE="display: table-cell">
      display: table-cell
    </DIV>
    <DIV STYLEX'display: table-o*ll">
      display: table-cell
    </DIV>
  </DIV>
  <DIV STYLE="display: table-row">
    <DIV STYLE-"display: table-cell">
      display: table-ceil
    </DIV>
    <DIV STYLE="display: table-cell">
      display: table-cell
    </DIV>
  </DIV>
</DIV>
```

На рис. 8.15 показано, каким образом такая таблица отобразится в разных браузерах.

Полезность значения свойства table стремится к нулю при использовании HTML, потому что сделать таблицу средствами HTML гораздо проще. Но зато поддержка табличных свойств является критичной при использовании XML. Дело в том, что между этими языками существует принципиальная разница. В языке HTML реализованы таблицы, а в языке XML таблиц нет, т. е. назначение элемента должно определяться стилями. Как вы уже знаете, для визуализации XML-документа можно использовать XSL или CSS. Браузеры Opera 5+ и Netscape 6.x используют только CSS, так что для них поддержка свойства table очень важна.

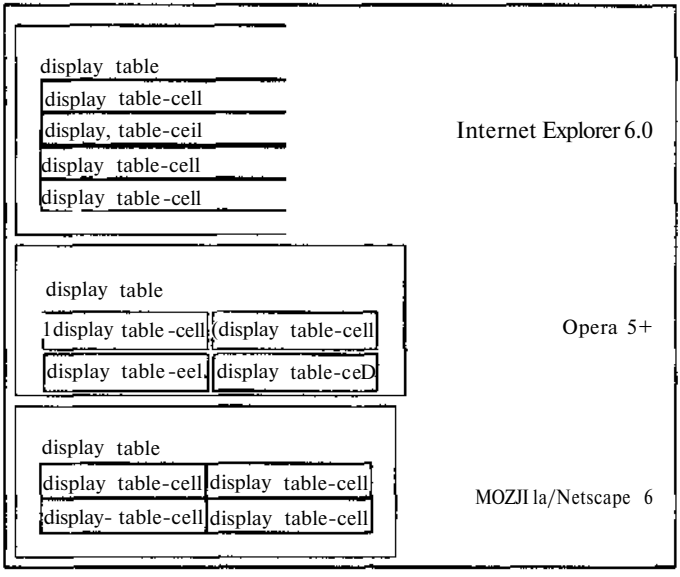


Рис. 8.15. Отображение таблиц, созданных с помощью CSS, различными браузерами

Свойство display имеет еще несколько значений, но они внедрены только браузером Opera 5+, так что рассматривать их не будем.

Visibility, overflow: видим и накладываем

Для полноты картины нам осталось рассмотреть полезные свойства из спецификации CSS-2, которые позволяют добиться некоторых эффектов при создании блоков.

visibility

С помощью данного свойства можно делать блоки видимыми и невидимыми. Причем делать это можно динамически, что и применяется успешно при изготоачешш так называемых выпадающих меню (drop-down menu). Делается оно следующим образом. На сайте организуется главное меню, а все подменю помещаются в отдельные слои.

Определение

Слоем называется блок, который имеет определенное положение на оси *z* Самая близкая аналогия — слои в графических пакетах, например, Photoshop. Координата z задается с помощью свойства z-index.

Все эти слои позиционируются о нужные области экрана, и всем задается свойство visibility: bidder., так что слои не видны. При подведении курсора мыши к пункту меню, с помощью скрипта свойство visibility у соответствующего слоя меняется с hidden на visible, так что он становится видимым и пользователь видит развернутый список подразделов. Пример выпадающего меню на сайте Microsoft продемонстрирован на рис. 8.16.

Рис. 8.16. Выпадающее меню на главной странице сайта www.microsoft.com

Еще раз ПОВТОрЮ главное ОТЛИЧИе между объявлениями visibility: hidden и display: none. Оно состоит в том, что при использовании свойства visibility блок физически остается на странице, тогда как при использовании display блок удаляется из потока форматирования, т. с. его вообще нет на странице.

overflow

Помните, у нас возникала проблема, когда размеры содержимого блока превышают размеры самого блока? Тогда мы с ней ничего не могли сделать, однако именно для решения этой проблемы служит свойство overflow. Оно может принимать четыре значения, вот и разберем их подробно. Сделаем блок шириной 200 пикселов, высотой 50 пикселов и наполним его текстом.

overflow: visible

В этом случае содержимое блока должно перекрывать блок и оставаться видимым. Но это работает именно так только в браузерах Opera 5+ и Netscape 6.x. А в браузере Internet Explorer 5+ блок будет растягиваться, чтобы вместить в себя весь контент. Для наглядности создадим такой блок:

```
<STYLE>
#block {
  border: lpx solid #000;
  width: 200px;
  height: 50px;
  padding: lem;
  overflow: visible}
</STYLE>
<DIV id="block">
```

```
Сделаем блок шириной 200 пикселов, высотой 50 пикселов и наполним его
текстом. При различных значениях свойства overflow будем получать различ-
ный результат.
</DIV>
```

Результат в различных браузерах будет таким, как на рис. 8.17.

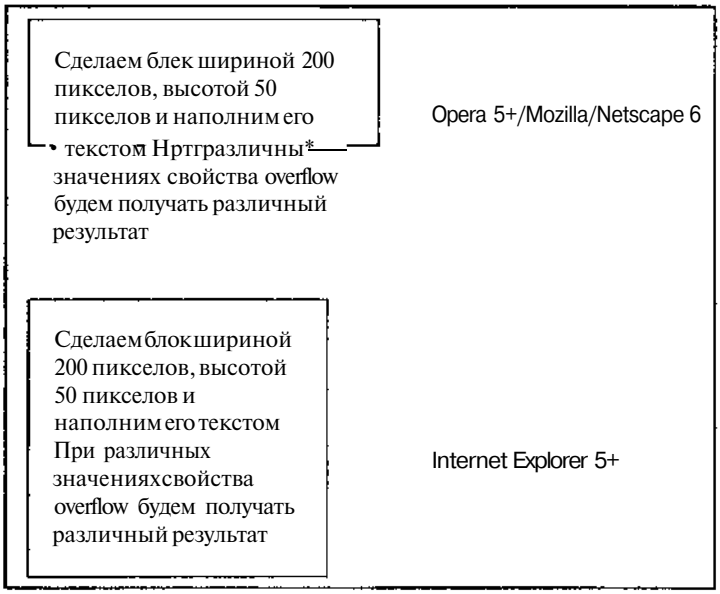


Рис. 8.17. Отображение блока с объявлением overflow: visible в различных браузерах

overflow: hidden

В этом случае весь контент, который превышает размеры блока, беспощадно обрезается и пользователь его не видит. Вообще применение данного объявления особого смысла не имеет. Если внутри блока находится текст, то он обязательно должен быть доступен для прочтения, иначе зачем же он вообще? Если внутри блока находится картинка, то она тоже должна быть видна. Так что полезность данного объявления весьма сомнительная, но оно одинаково отображается всеми браузерами, что хорошо (рис. 8.18).

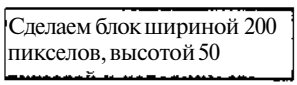


Рис. 8.18. Отображение блока с объявлением overflow: hidden

Гораздо более полезным является следующее свойство.

overflow: scroll

В этом случае блок должен снабжаться горизонтальными и вертикальными полосами прокрутки. При этом пользователь сможет просмотреть все содержимое блока. Как это выглядит в различных браузерах, видно из рис. 8.19.

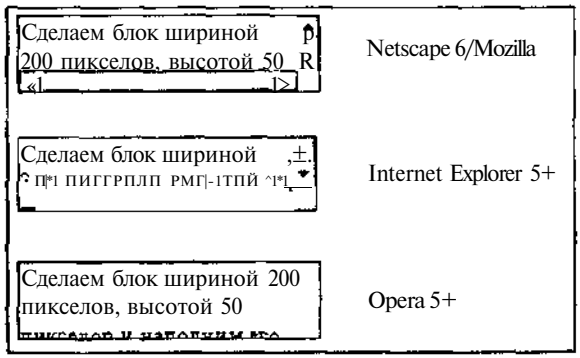


Рис. 8.19. Отображение блока с объявлением overflow: scroll

Как видите, такое объявление корректно работает в браузерах Netscape 6.v и Internet Explorer 5+, но вот в Opera 5+ оно эквивалентно значению hidden, что неправильно. Кроме того, горизонтальная полоса прокрутки в этом случае как бы и не нужна, потому что по горизонтали контент прекрасно умещается в ширину блока. И убрать ее в данном случае можно с помощью следующего объявления.

overflow: auto

Оно снабжает блок только той полосой прокрутки, которая необходима. В нашем примере высота контента превышает высоту блока и нужна только вертикальная полоса прокрутки. Этого можно достичь, установив значение auto в свойстве overflow (рис. 8.20).

Как видите, браузер Opera 5+ вновь некорректно отображает такое объявление. Вместо полосы прокрутки контент перекрывает блок, как это происходило в случае overflow: visible.

Применять значения scroll и auto можно в том случае, когда надо сэкономить место па странице. Например, так можно организовать подачу новостей. Каждая новость помещается в отдельный блок с объявлением overflow: auto. Высота блока подбирается таким образом, чтобы на виду был заголовок новости или же краткий анонс, а собственно весь остальной текст будет находиться ниже, и его можно будет прочитать при скроллировании.

В этой главе мы с вами научились создавать блоки с заданными параметрами. Мы можем помещать в эти блоки что угодно: текст, графику и произ-

вольную комбинацию Иными словами мы научились делать кирпичики, из которых можно создать сайт Действительно, любой сайт можно разобрать на составляющие Вот для примера разберем на составтяюшие сайт студии Дизайн Артель, который нахоштся по адрес\ www.artel.by (рис 8 21)

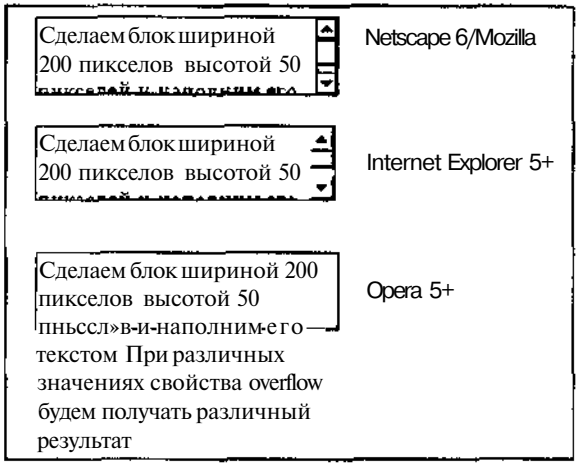
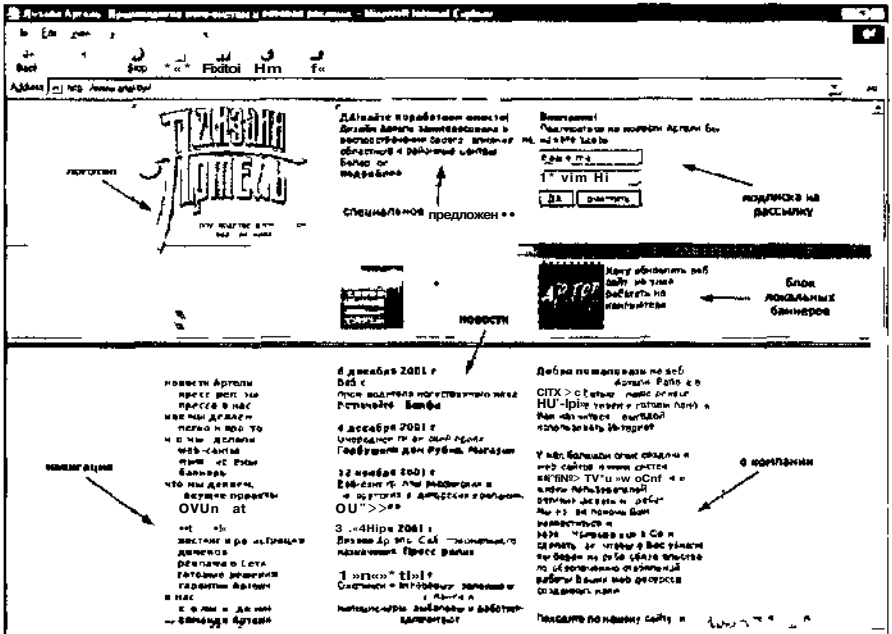


Рис 8 20. Отображение блока с объявлением overflow auto



РИС, 8 21 Структура сайта студии Дизайн Артель

На данном сайте используется верстка в три колонки. Что касается струк-\рны\ единиц, то можно выделить семь блоков. Верхний ряд составляют ри блока.

П логотип (по суш дела данный блок состоит из одного изображения логотипа);

3 специальное предложение (заполнен небольшим количеством текста);

П подписка на рассылку (состоит из текста и маленькой формы).

Средний ряд составляет один большой блок баннеров, которыи в свою очередь можно разбить на три маленьких блока, каждый из которых включает в себя рисунок и немного текста. А нижний ряд состоит из таких блоков:

П навигация (фактически это есть список);

О новости (состоит из маленьких отдельных текстовых блоков);

П презентационный текст о компании (достаточно большой текстовый блок).

Создать все эти блоки не составляет труда, и вы теперь это уже легко сможете сделать, а вот позиционировать на странице — нет В следующей главе именно этим мы и займемся Мы будем учиться позиционировать блоки.

Глава 9

Позиционирование

Позиционирование является самым нетривиальным разделом CSS, но и самым многообещающим. Все дело в том, что использование таблиц для позиционирования элементов, как это делается средствами HTML, искусственный прием. Изначально таблицы для этого не предназначались, но верстальщикам поневоле пришлось пользоваться именно ими, поскольку ь **HTML** больше нет никаких способов реализовать сложное форматирование страницы. Давайте выкинем таблицы из **HTML** и посмотрим, что останется (именно таким был язык HTML 2.0).

Без таблиц мы сможем разбивать текст на абзацы (этого у нас никто не отнимет), делать списки и различного рода выделения в тексте. Что еще/ В общем-то все. Фактически с помощью HTML без таблиц мы сможем верстать простейшие странички типа книжных. Причем заметьте, что даже п книгах очень часто верстка гораздо сложнее, если там используются различного рола врезки, обтекание текстом рисунков, выноски и прочие элементы, которые сильно разнообразят страницу и делают чтение более легким и интересным. Что касается газет и журналов, то о такой верстке приходилось только мечтать, поскольку без таблиц невозможно располагать текст в несколько колонок. Таблицы перевернули все. С их помощью можно делай невидимый каркас, который заполняется текстом и графикой, верстать п несколько колонок, позиционировать элементы на странице с точностью до одного пиксела, что открыло невиданные возможности совмещения частей порезанного на блоки изображения.

Да, все хорошо и здорово, но у таблиц есть свои недостатки. Это совершенно естественно, потому что они изначально не предназначались для позиционирования. Основные недостатки следующие.

О Во-первых, таблицы не слишком компактны с точки зрения кода.

П Во-втерыч, таблицы достаточно медленно отображаются даже самыми последними версиями браузеров, особенно медленно обрабатываются вложенные таблицы. Для ускорения обработки можно использовать CSS, тогда браузер сможет простую *фиксированную* таблицу обрабатывать быстрее. *Дя* фиксированных таблиц разработан специальный алгоритм обработки Включается он с помощью объявления table-layout: fixea. К сожалению, это работает только в браузерах Internet Explorer 5.5+ и Netscape 6.x

О В-третьих, содержимое таблицы располагается, как правило, достаточно нелогично Скажем, шапка страницы может быть закодирована с помо-

щью трех таблиц, потому что так на экране она будет появляться раньше, что часто очень важно. Гораздо логичнее было бы заключить шапку в отдельный блок. Другой пример. Меню может быть разбито на несколько строк и ячеек, что тоже затрудняет восприятие кода.

3 В-четвертых, при использовании таблиц затрудняется правка кода статичных страниц и шаблонов из-за смешения структуры документа и его содержимого.

О В-пятых, табличная верстка весьма нетривиальна и всем ее тонкостям приходится учиться достаточно долго, хотя источников знаний существует огромное количество.

Резюмируя, можно обобщить недостатки таблиц. Итак, таблицы *нелогичны, сюжны в восприятии и медленны при загрузке*.

Естественно, достоинства у них тоже есть:

- таблицы позволяют произвольным образом позиционировать элементы;

О табличная технология верстки хорошо освоена и по ней написано много книг, так что учиться можно эффективно, хотя сама табличная верстка не становится от этого проще;

G таблицы практически одинаково отображаются всеми браузерами (если говорить о браузерах пятых версий и выше, то различия крайне незначительны).

Здесь существенным является только третий пункт, поскольку совместимость и соответствие стандартам вообще вещь крайне полезная и необходимая. Гораздо лучше сравнивать свой код со спецификацией, чем проверять, как он выглядит в разных браузерах.

В целом можно сказать, что табличная верстка является основой любого более-менее серьезного сайта вот уже на протяжении нескольких лет. HTML-верстальщики к ней привыкли и изучили до тонкостей. Профессионалы знают мельчайшие несоответствия в различных браузерах и методы их устранения. Они, глядя на макет, уже *думают таблицами*.

Естественно, переход *к* любой другой верстке, пусть даже самой легкой и прекрасной, будет осуществлен только тогда, когда человек найдет в новой технологии достаточно много преимуществ для себя. Просто так никто переучиваться не будет, поэтому надо убедить человека и показать ему все достоинства нового типа верстки. Например, что она сэкономит ему 20% рабочего времени, так что за 8 часов работы можно выкроить лишний час на игру в Quake 3 в паре с арт-директором против программистов. Вот это будет действительно очень мощным стимулом. Экономия времени напрямую следует из упрощения кода. Следовательно, главным критерием, которым будет руководствоваться HTML-кодер, решая, изучать ли ему новый тип верстки или нет, будет именно упрощение кода и улучшение его логич-

ности. Вот мы вплотную и подобрались к тому, что часто называют CSS-позиционированием, а на английском — CSSP.

Для начала я приведу сводный список достоинств и недостатков (куда же без них) CSSP, а уж потом вы сами решите, соответствуют они действительности или нет. Достоинства верстки с помощью блоков и CSS-позиционирования.

- Значительно улучшается логичность кода. Это достигается благодаря нескольким причинам.
  - Каждый блок имеет уникальное имя. Естественно, имя можно дать максимально соответствующее самому блоку. Рассмотрим простой пример. В HTML отдельный блок, скажем, меню, заключается в свою таблицу, которая может начинаться так;

    <TABLE WIDTH="200">

    А если создавать блоки с помощью CSS, то блок меню может начинаться так:

    <DIV ID="raenu">

    Согласитесь, второй вариант проще и понятней.

    Конечно, в HTML перед каждым блоком можно вставлять комментарий. Например

    <!— Главное меню —>
    <TABLE WIDTH="200">

    Но это увеличивает объем кода, так что я, например, комментариями вообще не пользуюсь. Однако если над кодом трудятся несколько человек, то без них сложно.

  - За счет разделения визуального представления и структуры документа, непосредственно из HTML-кода исчезнут многие атрибуты, так что код станет прозрачней и чище. Для иллюстрации вполне подойдет слегка усложненный предыдущий пример:

    <TABLE CELLSPACING="O" CELLPADDING-"O" BORDER="0" WIDTH~"200">

    А если создавать блоки с помощью CSS. то все останется по-прежнему:

    <DIV ID="menu">

П Уменьшается общий объем кода. Это напрямую следует из предыдущего примера. Правда, для очень маленьких и простых страничек объем может и увеличиваться, но если учесть, что внешние CSS-файлы обычно кэшируются браузерами, то время повторной загрузки страницы все равно уменьшается. Собственно, время загрузки и есть определяющий фактор, тогда как объем кода вторичный.

О Значительно увеличивается контроль над блоками. Здесь объяснения несколько сложнее, потому что надо достаточно близко познакомиться с

CSS-позиционированием. Однако из чисто умозрительных соображений должно быть понятно, что изменять структуру таблицы сложнее, чем изменять стили для блоков. Конкретный пример приведу позже.

Что касается недостатков, то они, естественно, тоже имеются.

- Браузеры весьма неодинаково поддерживают CSS-позиииионирование. В предыдущей главе вы убедились, что даже создавать блоки с расчетом на различные браузеры не легко, а позиционирование гораздо более сложная вешь. К счастью, и здесь многие проблемы решаются.
- CSS-позиционирование — это совершенно новое направление. По этой причине учебных материалов крайне маю и практически все они на английском языке, так что осваивать данный тип верстки в некотором роде труднее, чем верстку табличную. Вам неизменно понадобится больше времени на эксперименты с кодом.
- Не каждый макет можно сверстать на основе CSSP. Это очень редкие исключения и часто можно сделать небольшие правки в дизайне, но, тем не менее, факт остается фактом. Еще один небольшой камешек в огород CSS-позиционирования.

В табл. 9.1 приведено сравнение двух схем позиционирования.

***Таблица** 9.1. Сравнение позиционирования элементов с помощью HTML-таблиц и с помощью CSSP*

Свойство	HTML-позиционирование	CSS-позиционирование
Логичность кода	Низкая	Высокая
Поддержка браузерами	Хорошая	Плохая
Внесение изменений	Сложно	Легко
Объем кода	Большой	Маленький
Быстрота обучения	Средняя	Низкая
Учебные материалы	Очень много	Мало

Можете сами сравнить две схемы и решить, какая из них лучше.

Скажу сразу, что в стандарте CSS-1 позиционирование крайне неразвито, так что для полноценного CSSP необходима хотя бы частичная поддержка браузерами стандарта CSS-2. К счастью, разработчики браузеров посчитали, что CSS-позиционирование вещь чрезвычайно полезная и необходимая, и по этой причине начали потихоньку внедрять его уже начиная с четвертых версий браузеров. Так, браузеры Internet Explorer 4.x и Netscape Navigator 4.x поддерживапи абсолютное позиционирование, хотя и с определенным числом критических багов. В пятых версиях поддержка была несколько улуч-

шена, главным образом исправлением багов. В общем, рассматривая позиционирование, мы будем отталкиваться от спецификации CSS-2.

Итак, для начала систематизируем все то, что может влиять на расположение блоков на странице. Оказывается, всего три вещи.

- *Схема позиционирования.* Она бывает четырех видов:
  - нормальный поток;
  - относительное позиционирование;
  - абсолютное позиционирование;
  - плавающая модель.
- *Расположение блоков в теле документа.* Если конкретизировать, то в некоторых случаях большое значение имеет относительное положение блоков друг относительно друга. Наименьшее алияние это оказывает при абсолютном позиционировании, а при нормальном потоке это фактически определяет вид документа.

П *Внешние параметры,* такие как размер окна браузера. Они критическим образом влияют на "резиновую" верстку, так что дизайн страницы может сильно пострадать при слишком маленьком размере окна. На самом деле ничего странного и страшного в этом нет, надо просто изначально планировать, как страница смотрится при разрешении 800x600.

Все это мы постепенно рассмотрим. Естественно, самым интересным и новым пунктом являются схемы позиционирования. Именно они и обеспечивают принципиальное различие между HTML-позиционированием и CSS-позиционированием. Так что займемся ими вплотную.

Нормальный поток

Давайте вспомним, как браузер обрабатывает дерево документа. Есть два типа блоков: структурные и строчные. Правила форматирования для них различаются.

С1 Структурные блоки форматируются следующим образом: браузер располагает их непосредственно друг за другом по вертикали, т. е. два структурных блока не могут находиться на одной горизонтальной линии. С помощью полей (margins) можно задавать расстояние между блоками по вертикали.

О Строчные блоки форматируются так: браузер располагает их горизонтально друг за другом. Одна строка называется *линейным блоком,* если строчные блоки не могут уместиться в одном линейном, то они распределяются по нескольким линейным блокам, которые следуют друг за другом по вертикали, т. е. в несколько строк.

Таким образом, ничего сложнее одной колонки с помощью нормального потока сделать невозможно.

Простейший пример — это несколько абзацев со строчными блоками. Для дополнительной иллюстрации свойства margin я сделал два блока-контейнера, в которые вложены по три структурных блока. Все блоки будут выделены рамками:

```
<STYLE TYPE="text/css">
  P {
    border: lpx dashed #AAA}
  P.marg {
    margin: 40px}
  #main {
    border: lpx solid #000;
    width: 90%}
</STYLE>
. . .
<DIV id="main">
  <P clазз="тагд">Первый абзац не слишком длинный. Он имеет нижнее поле, равное <VAR>40 пикселам</УАВ>. Это <3Т1Ю№3>увеличение</3ТКОЫС> расстояния совершенно бесполезно и служит исключительно для примера</P>
  <P class="marg">ВТорои абзац очень короткий</P>
  <P class="marg">ТреТМu абзац, как и все остальные, имеет левое и правое поля величиной <VAR>40 пикселов</УАК>. Иногда так бывает полезно делать отОивку.</P>
</DIV>
<BR><BR>
<DIV id="main">
  <P>Первый абзац не слишком длинный. Он не имеет никакой отбивки.</P>
  <P>Второй абзац очень короткий</P>
  <P>Третий абзац, как и все остальные, имеет левое и правое поля величиной <VAR>0 пикселов</УАК>. Иногда без отбивки текст выглядит ужасно</P>
</DIV>
```

В браузере этот код будет выглядеть так, как показано на рис. 9.1.

Как видите, *все* структурные блоки следуют строго друг за другом по вертикали. Даже если они вложены, все равно это правило не нарушается. Итак, с помощью нормального потока мы можем сверстать страницу только в одну колонку. Но давайте подумаем, где это нам может пригодиться. Вообще, сайт может быть сверстан и в одну колонку, например, если это online-

дневник или один из приобретающих популярность веб-логов (weblog). Тогда нет нужды в сложной навигации и в сложном интерфейсе, так что одна колонка именно то, что надо.

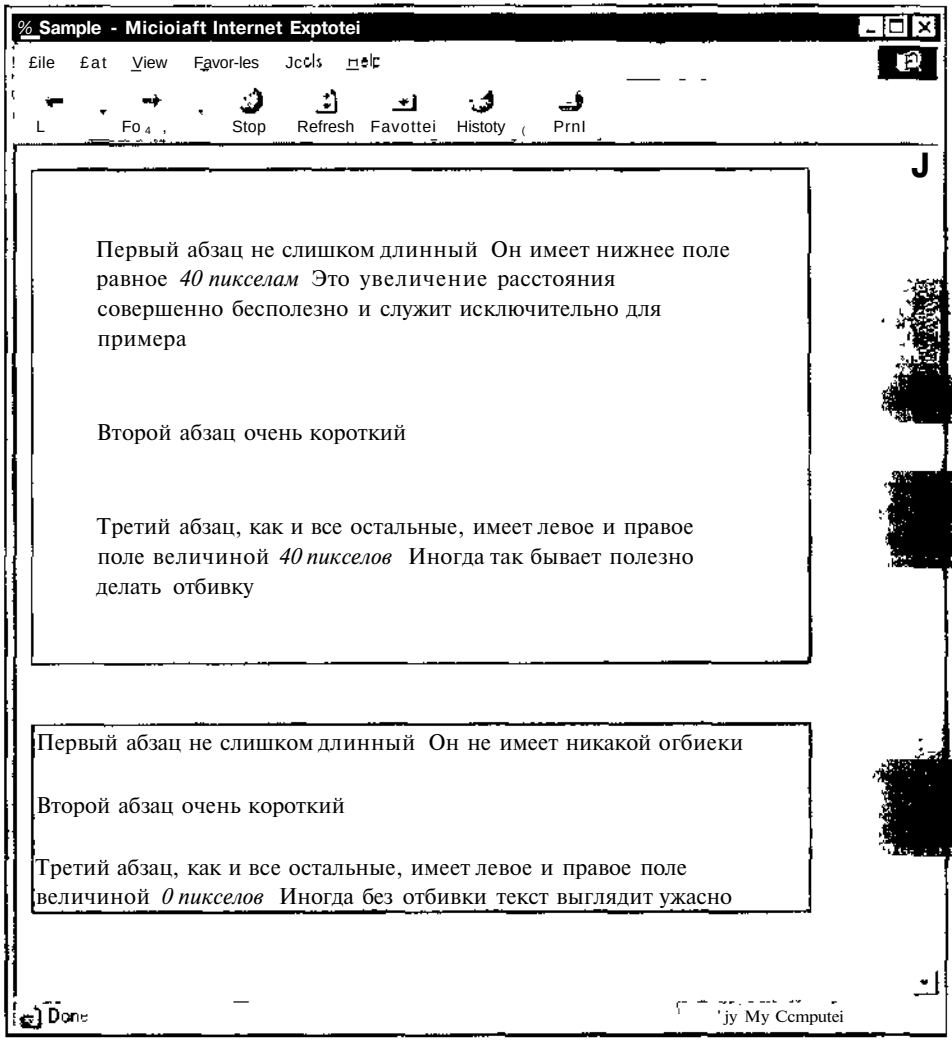


Рис. 9.1. Обработка блоков браузером в нормальном потоке

При верстке в одну колонку ее практически всегда центрируют Так действительно страница воспринимается гораздо лучше, и читать удобнее. Центрированием главного блока мы сейчас и займемся

Центрирование

Структура у нас такая. Контейнером яатяется элемент <BCDY>, а в него вложен блок, который надо центрировать. У элемента <BODY> есть один-единственный прямой наследник. По этой причине сразу возникает соображение центрировать все содержимое элемента <BODY> С ПОМОЩЬЮ объявления text-align: center. Вот такой кол по идее должен центрировать блок main:

```
BODY f
  text-align: center}
ttmain {
  border: Ipx dashed #000;
  width: 50%)
. . .
<BODY>
<DIV ID="main">
Вот этот блок мы и центрируем. Однако центрирование не такая тривиальная
задача, какой кажется на первый взгляд.
</DIV>
</BODY>
```

Однако если мы протестируем его в разных браузерах, то к нашему безмерному удивлению обнаружим, что центрируется блок только в браузере Internet Explorer, тогда как в Opera 5+ и Netscape *6.x* блок остается выровненным по левому краю. Вы еще больше удивитесь, если я скажу, что Internet Explorer ведет себя неправильно, потому что согласно спецификации CSS свойство text-align должно применяться к блоку текста, т. е. *центрировать только текст внутри блока,* но не сам блок. Браузеры Opera 5+ и Netscape 6.x именно так и поступают, они центрируют текст внутри блока, но сам блок располагается по левому краю окна. А браузер Internet Explorer, кроме текста, выравнивает еще и сам блок. Как будет выглядеть вышеприведенный код в разных браузерах, показано на рис. 9.2.

Возникает весьма насущный вопрос, как же нам центрировать блок? Оказывается, в спецификации CSS-2 для "незамешаемых элементов уровня блока в нормальном потоке" существует совершенно четкое правило.

> ***Правило***
>
> Если оба свойства: margin-left и margin-right, принимают значение auto, то их вычисляемые значения совпадают.

Задумайтесь, когда могут совпадать значения правого и левого полей? Значения могут совпадать только в том случае, если блок центрирован.

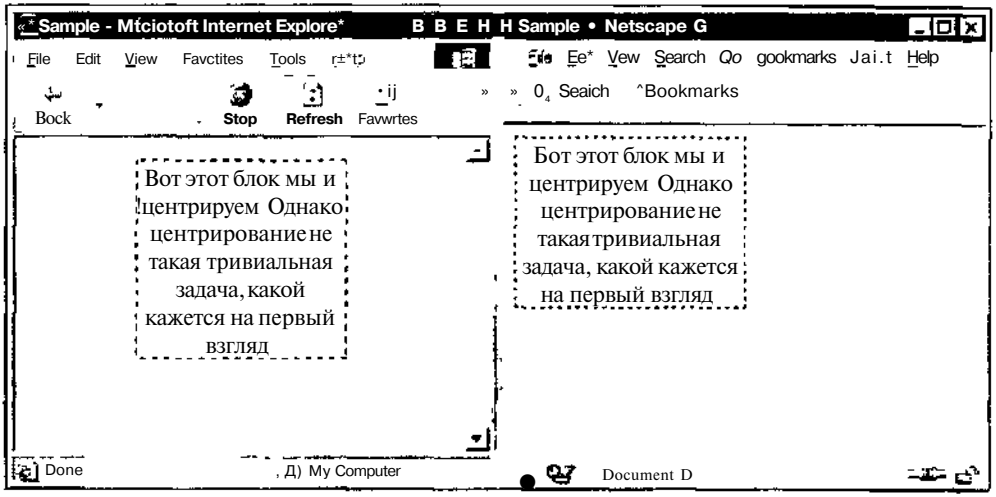


Рис. 9.2. Центрирование блока с помощью свойства text-align в браузерах Internet Explorer 6.x и Netscape 6.x

Таким образом, если добавить в правило для блока main объявления:

```
margin-left: auto;
margin-right: auto
```

то блок должен центрироваться. *К* нашей радости именно так и происходит, но не в браузере Internet Explorer 5+. Чтобы не запутаться окончательно, давайте систематизируем сведения:

- браузер Internet Explorer 5+ некорректно поддерживает свойство text-align, за счет чего можно центрировать блоки. Однако он не поддерживает центрирование с помошью полей;
- О браузеры Opera 5+ и Netscape 6_v корректно поддерживают свойство text-align, за счет чего блоки так центрировать нельзя. Однако они правильно поддерживают центрирование с помощью установки левого и ПравОГО ПОЛеЙ В auto.

Таким образом, для того, чтобы блок центрировался во всех браузерах, нам надо пользоваться обоими способами. Результирующий код будет выглядеть так:

```
BODY (
  text-align: centeri
#main {
  text-align: left;
  border: Ipx dashed #000;
  margin-left: auto;
```

```
  margin-right: auto;
  width: 50%}
. . .
<BODY>
<DIV id="main">
Вот этот Олок мл и центрируем. Однако центрирование не такая тривиальная
задача, какой кажется на первый взгляд.
</DIV>
</BODY>
```

Здесь еще в селекторе #main добавлено объявление text-align: left. Оно необходимо для устранения действия text-align: center, потому что текст в блоке нам по центру выравнивать не надо, а надо центрировать только сам блок.

Нормальный поток на примере weblog

Для закрепления полученных знаний давайте создадим простенький weblog. Вообще weblog представляет собой что-то вроде online-дневника. Такой тип сайтов очень популярен на Западе, и постепенно он вытесняет обычные домашние странички с фотографиями любимых собак и кошек, что хорошо. Мы сделаем только одну страничку. Она будет состоять из:

О заголовка (при желании можно сделать его графическим);

О блока ссылок;

О непосредственно сообщений, разбитых по дням.

Все это мы заключим в блок с рамкой толщиной один пиксел. Начнем именно с блока. Центрировать его мы умеем.

```
BODY i
  background: #FFF;
  font: lem Verdana, sans-serif;
  text-align: center}
#maan (
  text-align: left;
  border: lpx solid #000;
  margin-left: auto;
  margin-right: auto;
  padding: lam;
  width: 60%}
```

Для <BODY> прописываем размер и гарнитуру шрифта, чтобы это все правильно наследовалось, а также белый фон страницы. Кратко пробежимся по

стилям для блока mam. Черная рамка шириной в один пиксел создается объявлением border: ipx solid #oco. Ширина блока будет равняться 60 процентам от ширины окна браузера. Кроме того, установлены отступы padding: lem для того, чтобы текст не прилипал к рамке. Без отступов читать неудобно и плохо. В HTML-коде пока будет следующее:

```
<BODY>
  <DIV id="main">
  </DIV>
</BODY>
```

Как это все выглядит, пока демонстрировать нет смысла, просто черная рамка на белом фоне.

Далее сделаем заголовок. Поместим его в отдельный блок, который назовем logo. Эго будет просто название сайта, выведенное шрифтом Times New Roman размером в 2,6 раза большим, чем базовый шрифт страницы. Стиль будет такой:

```
«logo I
  font: 2.6ет "Times New Roman", serif;
  margin-bottom: 0.5em}
```

Кроме того, заголовок имеет нижнее поле 0,5 em, чтобы с ним не соприкасался следующий блок. У нас все блоки будут вложены в главный блок, так что HTML-код станет таким*

```
<BODY>
  <DIV id="main">
    <DIV id="logo">
      Falcons Weblog
    </DIV>
  </DIV>
</BODY>
```

Визуализация этого кода в браузере показана на рис. 9.3.

Сейчас создадим блок ссылок. Допустим, у нас будет гри страницы: weblog, about, links. Фактически в этих разделах будет находиться собственно само содержимое weblog, информация о сайте или о его владельце и список ссылок на сашы, которые любит автор данной странички. Мы эти три ссылки расположим горизонтально и заключим в серенький блок с пунктирной рамкой. Сами ссылки сделаем синими без подчеркивания.

Вначале создаем блок ссылок. Стиль у него будет такой:

```
#links {
  background: #EEE;
```

```
  font-size: 0.8em;
  font-weight: bold;
  border: Ipx dashed #000;
  padding: Q.3em(
```

Рис. 9.3. Первый промежуточный этап создания weblog
Общий центрированный блок и заголовок Вид в браузере Netscape 6.2

Итак, светло-серый фон задается объявлением background: #EEE, черная пунктирная рамка толщиной в один пиксел — объявлением border: ipx dashed #000, кроме того, есть еще небольшие отступы, чтобы сами ссылки не соприкасались с рамкой.

Для ссылок внутри этого блока напишем такой стиль:

```
#links A {
  color: #039;
  text-decoration: none}
```

Здесь используется контекстный селектор. Цвет ссылок синий (#039), а подчеркивание устраняется объявлением text-decoration: none. Обратите внимание, что здесь не используются псевдоклассы для ссылок. В этом случае все ссылки будут выглядеть одинаково. Это в большинстве случаев плохо. Однако конкретно здесь у нас есть всего три страницы, так что посетитель сайта без груда запомнит, где он уже был, а где еще нет. Именно по причине очень малого числа ссылок я и не делаю отдельных стилей для псевдоклассов.

В HTML-коде у нас добавится один блок со ссылками:

```
<BODY>
  <DIV id="main">
    <DIV id="logo">
```

```
      Falcons Weblog
    </DIV>
    <DIV id«"links">
    weblogfinbsp; ЬпЬэр; <A
 HREF="about.html">about</A>bnbsp; 6nbsp;bnbsp;<A
HREF="link3.html">links</A>
    </DIV>
  </DIV>
</BODY>
```

А в браузере это будет выглядеть так, как на рис. 9.4.

Рис. 9.4. Второй промежуточный этап создания weblog. Добавилась панель ссылок Вид в браузере Netscape 6 2

Собственно, осталось добавить только текстовые блоки. Структура у них будет такая. Каждый блок будет заключаться в теги <P></P>, сначала будет стоять дата сообщения, выделенная полужирным, а само сообщение будет начинаться с новой строки. Стиль на элемент <p> напишем такой:

```
P {
  font-size: Q.8em}
```

Кроме того, снизу страницы вставим копирайт с таким стилем:

```
.copy {
  font-size: О.бепп;
  text-align: center)
```

Общий код будет выглядеть так:

```
<STYLE TYPE="text/css">
A {
```

```
  color: #039}
BODY {
  background: #FFF;
  font: lem Verdana, sans-serif;
  text-align: center}
P {
  font-size: 0.8em}

•copy {
  font-size: O.6ет;
  text-align: center}

#links A {
  color: #039;
  text-decoration: none}
#links {
  background: #EEE;
  font: bold 0.8em Verdana, sans-serif;
  border: Ipx dashed #000;
  padding: O.3ет)
#logo {
  font: 2.6ет "Times New Roman", serif;
  margin-bottom: 0.5em}
#main {
  text-align: left;
  border: Ipx solid #000;
  margin-left: auto;
  margin-right: auto;
  padding: lem;
  width: 60%}
</STYLE>
. . .
<BODY>
  <DIV id="main">
    <DIV id="logo">
      Falcons Weblog
    </DIV>
    <DIV id="links">
```

```
    weblogsnbsp;Snbsp;finbsp; <A
HREF="about.html">about</A> Snbsp; Snbsp;<A
HREF="links.html">links</A>
    </DIV>
    <PXB>10.07.200K/BXBR>
Ухожу в отпуск. Поеду в Крым, ну их, эти Канары. Мне там не понравилось.
То ли дело Коктебейль' Как приеду, обязательно поделюсь впечатлениями. До
встречи :)</P>
    <PXB>01.06.2001</BXBR>
Сегодня нашел великолепный сайт, посвященный CSS-верстке. Называется <A
HREH^="http://www.bluerobot.com">www.bluerobot.com</A>. Много интересных
статей и примеров, так что посмотрите обязательно.</P>
    <PXB>23.05.200K/BXBPO
Прекрасная стоит погода' За окном непрерывно идет дождь... Красота! Люблю
дождь. Люблю дождливым майским вечером устроиться на мансарде с бутылкой
кальвадоса и смотреть на пелену воды. Это прекрасно.</P>
    <PXB>19.01.2001</BXBR>
Сегодня я открыл свой собственный weblog. Здесь будут мои мысли о жизни и
вообще обо всем, что меня интересует.</P>

  <P c1a83="copy">Диэайн Ьу <A
HREF="mailto:firefalcon@tut.by">FireFalcon</AXBR>bcopy;200K/P>
</BODY>
```

В браузере вся эта неземная красота будет выглядеть тик. как на рис. 9.5.

Естественно, дизайн без всяких изысков, но страница выглядит очень аккуратно и спокойно. Именно этих качеств часто не хватает начинающим веб-дизайнерам. По неизвестным причинам они стараются сделать сайт максимально ярким (это было бы еще ничего, поскольку яркость сама по себе вещь приемлемая, но буйство красок иногда просто шокирует), а элементы разбросать по странице в полнейшем беспорядке. Я, на всякий случай, дам совет. Если вы не сильны в дизайне, но хотите во что бы то ни стало сделать свой собственный сайт, то используйте крайне ограниченное количество цветов и старайтесь размешать блоки на странице логично. По крайней мере, тогда никто не скажет, что ваша страница вопиюще кошмарна (хотя ругать все равно будут, впрочем, ругают всех без исключения, включая известного веб-дизайнера А. Лебедева).

Итак, нормальный поток — это именно то, что использует браузер по умолчанию для форматирования документа. Вот как раз пришла пора познакомиться с новым свойством, которое появилось в спецификации CSS-2.

Свойство position определяет схему позиционирования. Правда, для плавающей модели используется свойство float, так что свойство position охватывает не все возможные схемы.

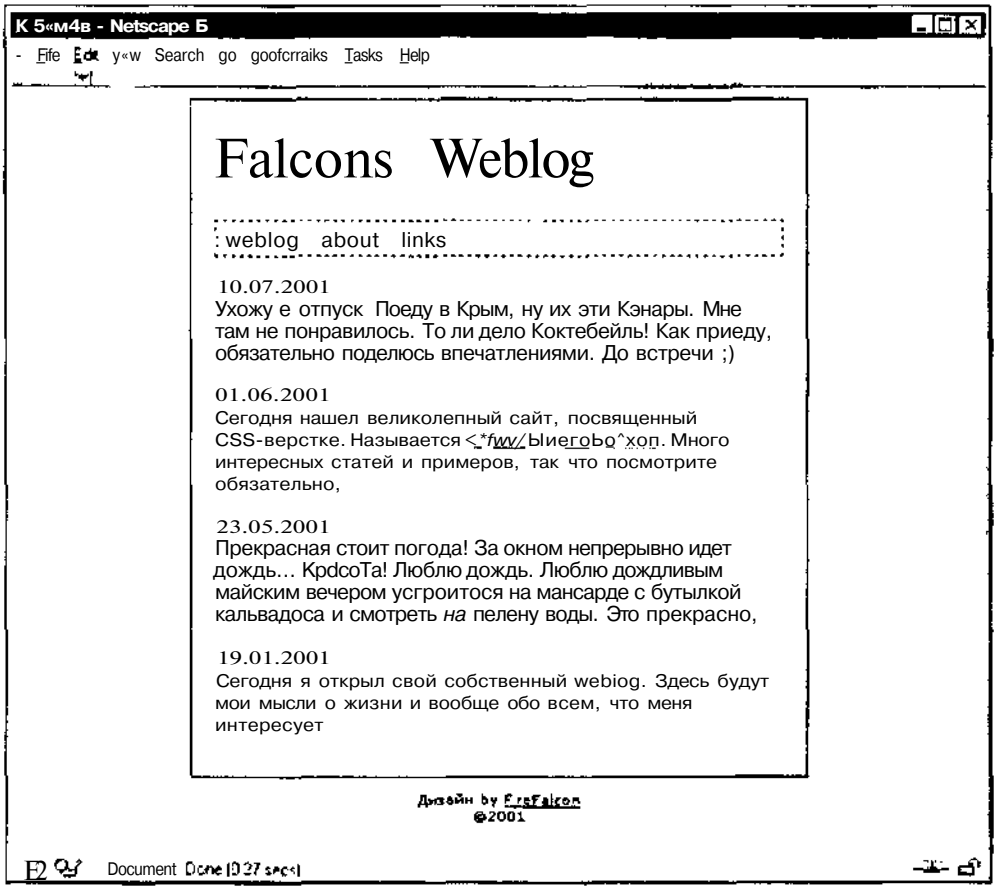


Рис. 9.S. Окончательный вид webiog в браузере Netscape 6.2

Свойство position может принимать следующие значения:

- *O* static — нормальный поток;
- *O* relative — относительное позиционирование (подвид нормального потока);
- O absolute — абсолютное позиционирование;
- O fixed — фиксированное позиционирование (подвид абсолютного).

Значение static устанавливает нормальный поток, т. е. является значением по умолчанию. К разновидности нормального потока можно отнести и *относительное позиционирование,* которое устанавливается с помощью *объявления* position: relative.

Относительное позиционирование

Относительное позиционирование функционирует по следующему алгоритму. Вначале положение элемента рассчитывается согласно нормальному потоку (оно называется *нормсньным положением)* Потом элемент смешается относительно нормального положения. Величина смешения задается с помощью следующих свойств:

- left — задает смешение от левого края контейнера;
- right — задает смешение от правого края контейнера;
- top — задает смещение от верхнего края контейнера;

D bottom — задает смешение от нижнего края контейнера.

Значения для этих свойств можно указывать в любых единицах длины и в процентах. Как работает относительное позиционирование, хорошо видно из следующего примера. Создадим три совершенно одинаковых блока:

```
P {
  border: Ipx solid #000}
. . .
<P>первый Олок</P>
<P>второй блок</P>
<P>третий Олок</P>
```

Внешний вид блоков в браузере показан на рис. 9.6.

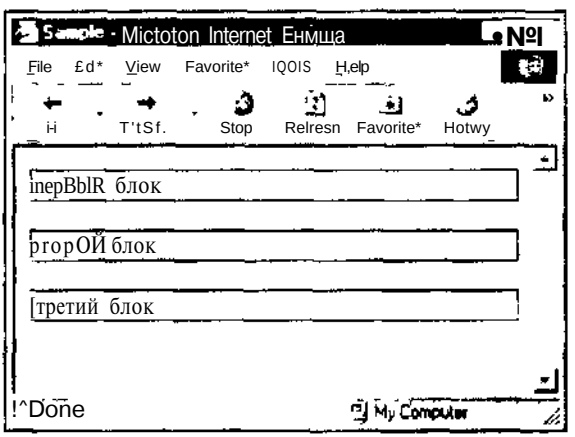


Рис. 9.6. Три блока в нормальном потоке

А сейчас второй блок у нас будет относительно позиционирован[1] мы сдвинем его на 23 пиксела относительно верхнего края.

Для этого присвоим второму блоку iD="rel", и код будет такой:

```
p
   {border: 1px solid #000}
#rel {
  position: relative;
  top- 23px
. . .
<P>первый Олок</P>
<P 10="rel">второй блок</p>
<P>третий Олок</P>
```

Из рис. 9-7 видно, как изменится картина в браузере.

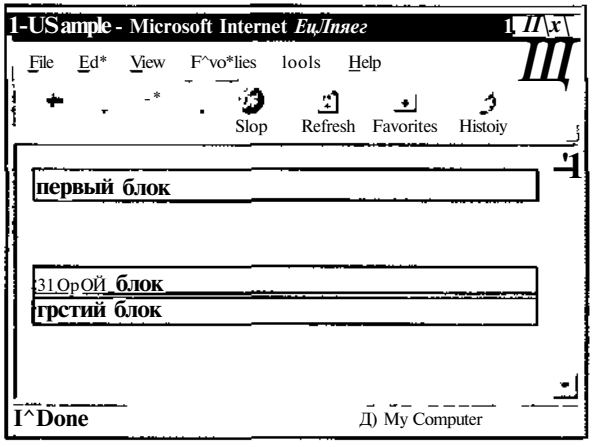


Рис. 9.7. Три блока в нормальном потоке. Второй блок сдвинут на 23 пиксела вниз, т е он позиционирован относительно

Как видите, второй блок просто сдвинулся на 23 пиксела вниз, вследствие чего произошло наложение второго и третьего блоков, но положение третьего блока на странице *не изменилось*. Следовательно, относительное позиционирование непосредственным образом влияет только на сам позиционируемый блок, но *не влияет на все остальные блоки* Это важное правило, которое надо помнить.

Полезность относительного позиционирования весьма сомнительна Оно позволяет сдвигать блоки, и не более того. Теоретически мы можем расположить блоки как захотим, если точно заданы их высота и ширина

Однако есть две причины, которые этому препятствуют.

- Фиксировать высоту блока практически нереально. Например, у нас есть блок новостей. Естественно, новости могут быть длинными или короткими, так что высота блока будет варьироваться. При этом расположенные ниже блоки просто опускаются вниз. Дизайн должен быть сделан так, чтобы не рассыпаться при разумной высоте блока.
- Браузеры совершенно по-разному интерпретируют сложное относительное позиционирование. Когда мы возьмем два блока и сдвинем один относительно друюю, то вес будет одинаково. Однако если мы возьмем три блока, то картины будут критическим образом отличаться.

Чтобы не быть голословным, покажу это на примере. Создадим три блока (one, two, three) и попытаемся с помощью относительного позиционирования расположить их в три колонки Нам придется строго задать размеры всех блоков. Для наглядности вокруг всех блоков будет однопикселная рамка.

```
DIV {
  border: lpx solid #000}
#one I
  width: 30%;
  height: 20%}
#two {
  width: 30%;
  height: 20%;
  position: relative;
  left: 30%;
  top: -20%i
#three {width: 30%;
  height: 20*;
  position: relative;
  left: 60%;
  top:-40%''
' ' '
<DIV id="one">nepBb& CJIOK</DIV>
<DIV id="two">BTopou UnoK</DIV>
<DIV id="three">TpeTH# блок</DIV>
```

Первый блок имеет высоту 20% и ширину 30%. Второй блок имеет точно такие же размеры, но он сдвинут на 30% влево и на 20% вверх, т. е. он должен стать на одном уровне с первым блоком. Третий блок сдвинут на 60% влево и на 40% вверх. У нас в идеале должно получиться три одинаковых

блока, стоящих на одной горизонтали А реальная картина для разных браузеров показана на рис 9.8.

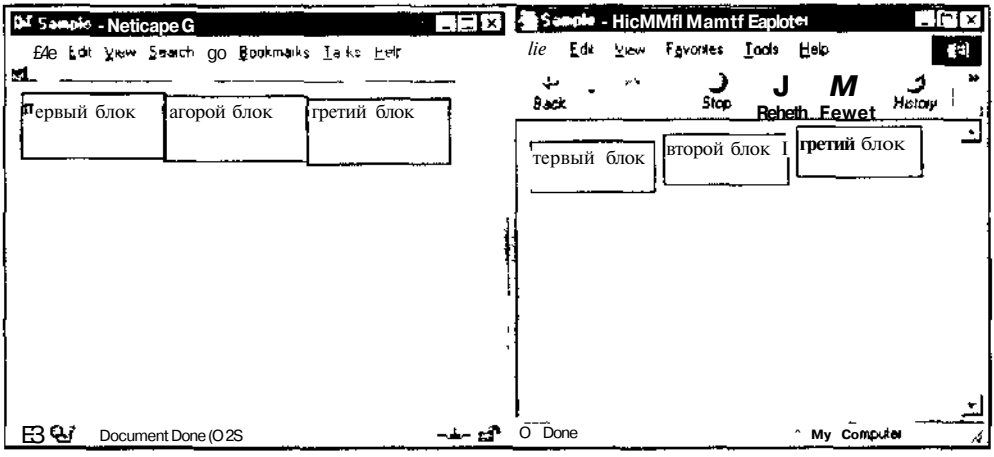


Рис. 9.8. Вид трех относительно позиционированных блоков в браузерах Netscape 6 2 и Internet Explorer 6 *x* соответственно

Как видите, чуть более сложный код приводит к совершенно разным картинам в бра>зсрах. Вообще разобраться в причинах, по которым это происходит, практически невозможно. Впрочем, по своей сути относительное позиционирование мало пригодно для сложной верстки, а использовать его можно только для смешения блоков Следующий пример демонстрирует, какое форматирование текста можно выполнить па основе относительного позиционирования Создадим три блока, все шириной 50% относительно ширины окна браузера Второй блок сдвинем на те же 50% от левого края Код будет такой:

```
DZV {
  border: lpx solid #000}
ttone {
  width: 50%}
#two {
  width: 50%;
  position relative,
  left: 50%.
#thiee
  width: 50%}
. . .
<DIV id=-"one"><B>01.06.200K/B><BR>
m i,ik r:
```

```
Сегодня нашел великолепный сайт, посвященный CSS-верстке. Называется <A
HREF="http://www.bluerobot.com">www.bluerobot.cora</A>.
</DIV>
<DIV id="two"><B>23.05.200K/B><BR>
Прекрасная стоит погода' За окном непрерывно идет дождь... Красота'
</DIV>
<DIV id«="three"><B>19.01.2001</BXBR>
```

Сегодня я открыл свой собственный weblog. Здесь будут мои мысли о жизни и вообще обо всем, что меня интересует.

```
</DIV>
```

А в браузере он будет выглядеть так, как на рис. 9.9.

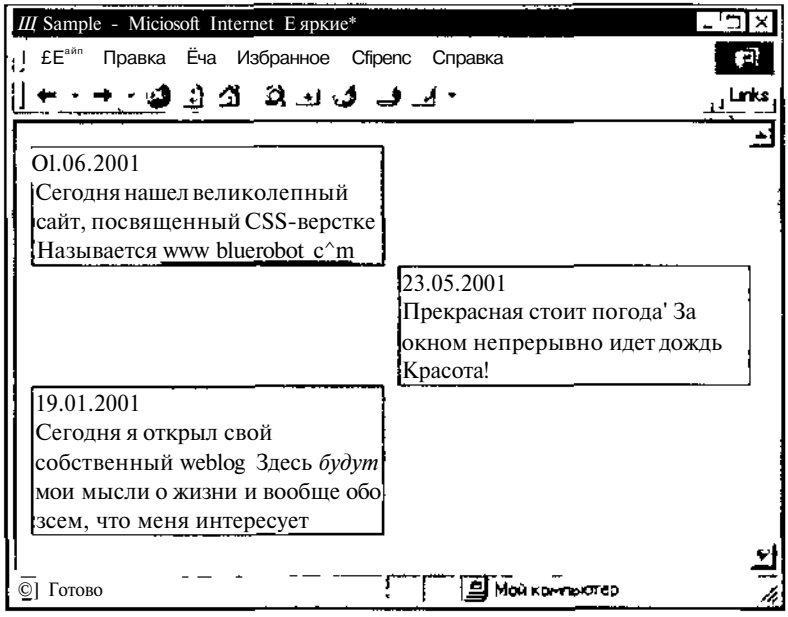


Рис. 9.9. Верстка в две псевдоколонки с помощью относительного позиционирования

Иногда текст на сайтах действительно размещают такой лесенкой. Конечно, она оформлена гораздо более привлекательно, чем в этом простом примере С помощью относительного позиционирования добиться эффекта лесенки гораздо проще, чем с помощью таблиц.

Пожалуй, мы исчерпали все возможности нормального потока в целом и относительного позиционирования в частности. Сейчас самое время остановиться и еще раз посмотреть на проблему позиционирования.

Верстальщики газет и журналов, должно быть, счастливые люди. Там с позиционированием проблем гораздо меньше. Взял текстовый блок и разместил его на странице где угодно. Возникает естественное желание перенести этот механизм в веб. Однако ни HTML, ни CSS-1 этого делать не позволяли. Создавать блоки было уже можно, более того, с помощью свойства float эти блоки можно было располагать в несколько колонок, но точно позиционировать — нет.

Откровенно радует тот факт, что разработчики стандарта CSS-2 учли опыт газетных верстальщиков и создали достаточно простую и надежную схему позиционирования. Именно ее мы сейчас и рассмотрим во всех подробностях.

Абсолютное позиционирование

Если в относительном позиционировании элемент лишь смешался относительно нормального положения, но оставался в нормальном потоке, то при абсолютном позиционировании элемент полностью вырывается из нормального потока и уже затем позиционируется в соответствии с заданными объявлениями в таблице стилей. Это очень принципиальное отличие и из него напрямую вытекают два следствия.

- Если блок абсолютно позиционируется, то в HTML-коде он может располагаться в любом месте (если не принимать в расчет вложенные блоки, с ними несколько иначе), потому что он как бы находится вне нормального потока. При относительном позиционировании положение блока напрямую влияло на место блока на экране.
- Очевидно, свойства top, left, right, bottom должны функционировать как-то иначе, чем при относительном позиционировании. Действительно, тогда они задавали величину смещения относительно положения элемента в нормальном потоке, сейчас же элемент вырывается из нормального потока, так что контейнером для него *всегда* будет *или окно браузера, иди позиционированный блок* (то есть блок с объявлением position: relative **ИЛИ** position: absolute).

Сделать блок абсолютно позиционированным можно с помошью объявления position: absolute. Для того чтобы вы ясно прочувствовали различия между относительным и абсолютным позиционированием, приведу пример. Возьмем два блока шириной 30% и высотой 20% от ширины экрана. Оба будут позиционированы относительно. Первый блок (one) будет смещен вниз на 10%, второй (two) тоже будет смешен вниз на 10%, но, кроме того, еще и влево на 30%.

```
DIV (
  border: 1px solid #000}
#one {
  width: 30%;
```

```
  height: 20%;
  position. relative,
left  0;
  top. 10%}
#two t
  width:  30%;
  height   2C*;
  position   relative,
  left   30%,
  top 10%}
.    .
<DIV   1c1="one"первый блок</DIV>
<DIV   id="two">BTopOu   блок</01У>
```

Естественно, в этом случае положение блоков в коде существенно, так что блок one идет первым, а блок two — вторым В итоге блоки б>дут располагайся лесенкой, как показано на рис 9 10

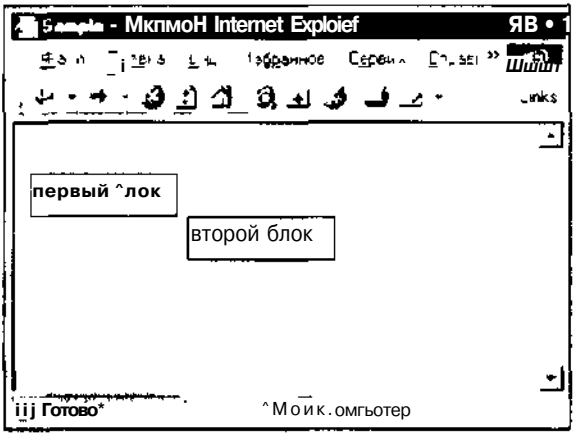


Рис. 9.10. Два блока, позиционированные относительно

А сейчас мы просто заменим в коде объявление position- relative на position: absolute и поменяем местами блоки one И two в коде Все остальное оставим без изменений Кол станет таким'

```
DIV {
  border:  lpx solid #000}
#one )
  width: 30%;
```

```
  height: 20%;
  position: absolute;
  left: 0;
  top:10%}
#tWo {
  width: 30%;
  height: 20%;
  position: absolute;
  left: 30%;
  top: 10%}
. . .
<DIV id="two">BTopoii блок</ГЛУ>
<DI" id="one" 'первый блок</01У>
```

Из рис. 9.11 ясно, что в браузере картина кардинально изменится.

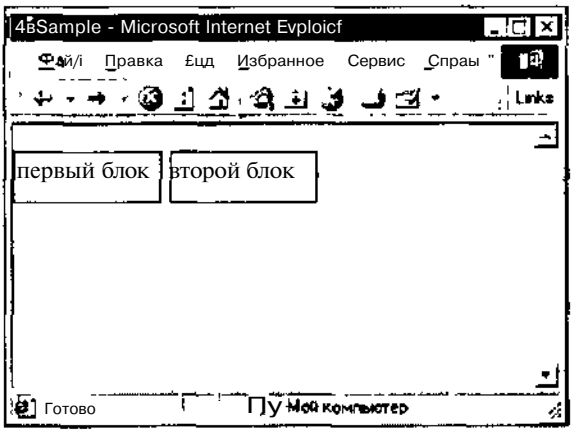


Рис. 9.11. Два абсолютно позиционированных блока

Как видим, перестановка в HTML-коле блоков, созданных с помощью элементов <DIV>, совершенно не повлияла на расположение блоков в окне браузера. Иначе говоря, вы получили наглядное подтверждение, что при абсолютном позиционировании элементы полностью вырываются из нормального потока, а точное расположение блока в окне браузера задается непосредственно в стилях. На основе этого можно вывести очевидное правило.

Правило

При абсолютном позиционировании расположение блока, который является прямым потомком элемента <BODY>, В HTML-коде документа не влияет на положение этого блока в окне браузера

Кроме того, в данном случае все значения смещений берутся относительно окна браузера, потому что в данном случае оно является контейнером для абсолютно позиционированных элементов. По этой причине объявление top: 10% будет смещать любой абсолютно позиционированный блок на 10% вниз от верхнего края окна браузера. Именно из-за этого два блока находятся на одной линии.

Благодаря этим отличиям, абсолютное позиционирование представляется неплохим кандидатом на роль схемы для верстки сайтов. Если оперировать самыми общими понятиями, то с помощью абсолютного позиционирования можно легко размещать блоки на одной горизонтальной линии и контролировать их позицию с точностью до одного пиксела. Это именно то, чего не хватало позиционированию относительному.

Вообще надо коснуться понятия *контейнер для абсолютно позиционированного элемента,* потому что оно немного отличается от обычного представления контейнера. Это существенно для вложенных блоков. Надо сказать, что для вложенных блоков абсолютное позиционирование редко применяется сразу к ним обоим.

Определение

Контейнером для абсолютно позиционированного элемента может быть только абсолютно или относительно позиционированный элемент или же непосредственно окно браузера.

Таким образом, абсолютно позиционированные элементы первого уровня вложенности имеют один и тот же контейнер — окно браузера.

Абсолютное позиционирование оптимально подходит для жесткой верстки, т. е. для тех случаев, в которых раньше мы пользовались таблицами фиксированной ширины. Помните сайт из *главы* .?? Там мы пользовались именно фиксированными таблицами, так что попробуем заменить их на блоки с абсолютным позиционированием. Это не так сложно, как кажется на первый взгляд. Нам надо всего лишь забыть вообще о табличной верстке и перейти на другой тип мышления. Глядя на макет (рис. 9.12), надо мыслить блоками

Вообще мыслить блоками несложно, но сложно *перестроиться* с таблиц на блоки. Достаточно трудно переход дастся профессиональным HTML-верстальщикам, не особенно интересующимся CSS Наиболее легко верстку блоками воспримут начинающие, потому что они еше не имеют привычки мыслить таблицами Конечно, им будет сложнее в том плане, что верстка блоками — область новая и недостаточно исследована и описана. Иными словами, разбить макет на правильные блоки после небольшой тренировки смогут все, а вот применять CSS на практике будет сложнее начинающим.

Итак, надо описать такой способ мышления на конкретном примере. Для начала надо найти на макете главные блоки, т. е. те. которые будут являться

прямыми потомками элемента <BODY>. В данном случае можно главные блоки определить по-разному. Если посмотреть на колонки, то их всего три: первая содержит основные ссылки, вторая содержит ссылки второго уровня, а третья широкая колонка содержит основной текст раздела Таким образом, можно выделить три блока, каждый из которых соответствовал бы своей колонке. Но заметьте, что тогда пришлось бы рисунок с названием фирмы Энигма разделять на два рисунка. По некоторым соображениям мы этого делать не будем. Вообще, предпочтительно, чтобы название фирмы располагалось на одном рисунке, потому что при загрузке страницы может загрузиться сперва только одна часть названия, например, «Энига», так что это будет смотреться нереспектабельно. Таким образом, нам схема трех блоков в три колонки не подходит.

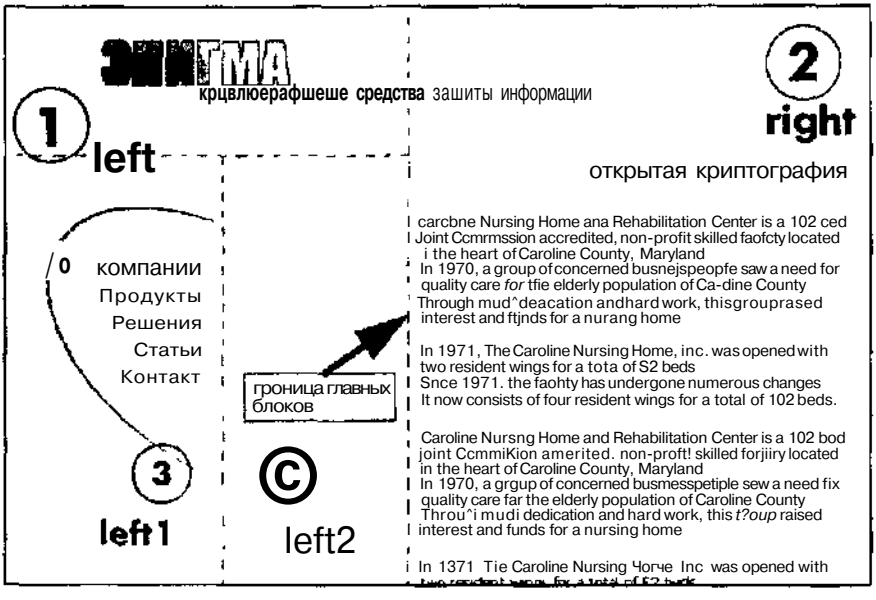


Рис. 9.12. Макет сайга, разбитый на логические блоки для последующего абсолютного позиционирования. Границы блоков отмечены пунктиром. Стрелка указывает на границу двух главных блоков

Тогда мы просто соединим первую и вторую колонки, так что останется два главных блока. На рисунке они обозначены цифрами 1 и 2 в больших кружках. Левый блок мы назовем left, а правый right, что достаточно логично. Граница главных блоков обозначена толстой пунктирной линией, на которую указывает еще более толстая стрелка.

Внутри левого главного блока после картинки с логотипом будет сше два блока, которые на рис. 9.12 обозначены цифрами 3 и 4. Назовем мы их lefti

и ieft2. Это разделение напрашивается само собой, потому что колонки-то все равно две и объединить их в одну невозможно.

С блоками мы определились, так что начнем их кодировать. Причем начнем практически с нуля, поскольку таблицы нам уже совершенно не нужны, т. е. придется полностью переписать HTML-код и достаточно много чего добавить в CSS-код.

Для начала создадим два главных блока.

```
<BODY>
  <DIV id="left">
  </DIV>
  <DIV id="right">
  </DIV>
</BODY>
```

Как мы уже договорились, главные блоки являются прямыми потомками элемента <BODY>. А сейчас надо задать для элементов <DIV> СТИЛИ, чтобы они, собственно, и стати блоками. Ширина *левого б.юка* будет 348 пикселов, он будет иметь нулевые сдвиги слева и сверху, следовательно, код будет такой:

```
#left {
  width: 348px;
  position: absolute;
  left: Opx;
  top: Opx}
```

Правый блок будет иметь ширину 402 пиксела. Нам надо, чтобы он был как бы второй колонкой. Тогда придется его отодвинуть от левого края документа настолько, чтобы в это расстояние как раз уместился *левый блок*. Значит, нам надо его отодвинуть на ширину левого блока, которая равна 348 пикселам. Код второго блока будет такой:

```
#right {
  width: 402px;
  position: absolute;
  left: 348px;
  top: Opx}
```

Теперь поместим в первый блок соответствующий рисунок. Заметьте, что рисунок меньше ширины блока, так что надо выравнивать его по правом) краю. Для этих целей создадим еще один блок. Назовем его logo, и стиль для него будет очень простым:

```
#logo {
  text-align: right}
```

Второй рисунок вставляется в блок right без каких-либо дополнительных трудозатрат, потому что нам надо его выровнять по левому краю, а это делается по умолчанию. После вставки рисунков код станет таким:

```
<BODY>
  <DIV id»"left">
    <DIV id="logo">
      <IMG SRC='ᵗimg/fragment_l.gifᴹ WIDTH="266" HEIGHT="86" ALT="ЗНИТМА -
криптографические средства защиты информации [лохютип]">
    </DIV>
  </DIV>
  <DIV id="right">
    <IMG SRC«"in»g/fragment_2.gif" WIDTH="251" HEIGHT="167" АЬТ="ЭНИГМА -
быстрый путь к надежности">
  </DIV>
</BODY>
```

Осталось создать два внутренних блока 3 и 4. Ширина блока 3 будет 188 пикселов. Высоту можно задавать по ситуации. Вообще высота блоков — это одна из *самых больших* проблем в CSS на сегодняшний день. Вплотную с ней мы столкнемся позже, когда будем верстать большой сложный сайт. Теперь же для блока 3 зададим высоту явно, и равна она будет 600 пикселам. Как видно из рис. 9.12, ссылки должны быть выровнены по правому краю, так что код блока lefti пока будет таким:

```
#leftl (
  text-align: right;
  width: 188px;
  height: 600px}
```

Осталось блок позиционировать. Слева смешение будет нулевым. А вот со смешением сверху ситуация сложнее. Если мы зададим top: Opx, то смещения не будет и блок 3 будет находиться в самом верху окна браузера, т. е. произойдет наложение блоков lefti и lego. Нам же надо, чтобы блок lefi.1 начинался непосредственно после рисунка, т. е. после блока lego. Для этого можно воспользоваться до сих пор нам незнакомым значением auto свойства top.

Значение auto размешает элемент там, где он должен был бы стоять в нормальном потоке. Получается, что для относительно позиционированного элемента объявление top: auto эквивалентно объявлению top: Cpx. Для абсолютно позиционированного элемента все несколько сложнее.

Правило

Величина auto равна такому значению, которое необходимо для размещения элемента **в** нормальной позиции, но учитывая абсолютное позиционирование.

Звучит крайне запутанно, но одним предложением проще не скажешь. Сейчас все объясню. Пока отвлечемся от нашего сайта и рассмотрим простейший пример.

Пусть у нас имеется нормальный поток. В нормальном потоке два блока: первый (first) содержит рисунок, а второй (second) текст, т. е. текст следует сразу после рисунка. Затем мы взяли и позиционировали первый блок абсолютно, но точно на то же место. Таким образом, мы задали:

```
«first {
  width: 200px;
  position: absolute;
  left: Opx;
  top: Opx)
```

При этом первый блок из нормального потока вырывается, и его там как бы нет, так что второй блок в нормальном потоке становится единственным, т. е. помещается максимально сверху *и* накладывается на первый блок. Это наложение нам совершенно ни к чему. Значит, нам надо сместить второй блок ровно настолько, чтобы он встал сразу за первым. Вот именно это и осуществляется объявлениями

```
*second 1
  position: absolute;
  top: auto}
```

Очевидно, что второй блок тоже вырывается из нормального потока и располагается непосредственно за первым блоком.

У нас возникала точно такая же ситуация, так что решается она совершенно аналогично. Итак, стилевое правило на блок 3 будет таким:

```
«leftl (
  text-align: right;
  width: 188px;
  height: 600px;
  position: absolute;
  left: Opx;
  top: auto}
```

Вообще, для свойства top значение auto является значением по умолчанию, так что его можно не прописывать явно. Здесь это сделано хля наглядности.

Кроме того, нам надо добавить фоновый рисунок (дуга, которая окружает ссылки). Для этого уже есть класс back, так что общий код первого главного блока left с вложенным блоком lefti будет таким:

```
<DIV id="left">
  <DIV id="logo">
```

```
    <IMG SRC="img/fragment_1.gif" WIDTH=*"266" HEIGHT="86" АЬТ="ЭНИГМА -
криптографические средства защиты информации [логотип]">
  </DIV>
  <DIV id="1eft1" CIASS="back">
    <ERXBR>
    <A HPEF="tt"X) компании</AXB1r>
    <A HREF="#">npoflyKTbr</AXBR>
    <A HREF="#">Pemero«i</AXBR>
    <A HH£P="#">Статьи</A>
  </DXV>
</DIV>
```

Остался блок 4. Он имеет ширину 160 пикселов, и текст в нем выровнен по левому краю. Кроме того, он должен быть смешен от левого края па ширину блока 3, т. е. на 188 пикселов. Стиль этого блока будет таким:

```
Heft2 {
  width: 160px;
  height: 600px;
  position: absolute;
  left: 188px;
  top: auto}
```

Весь блок должен иметь синий фоновый цвет. Для этого у нас был когда-то введен класс bluecol, так что им и воспользуемся.

```
.bluecol {
  background: #1D3D4E}
```

Собственно, осталось только вставить основной текст в блок right, а это делается совершенно элементарно После этого можно взглянуть на страничку в браузере. И выглядеть она будет так, как на рис. 9.13

Обратите внимание, что отсутствуют отступы, и текст вплотную "прилипает" к границам контейнеров. Впечатление это остаатяет неважное, так что проблему надо решать. Усугубляется она тем, что в браузере Internet Explorer *5.x* некорректно реализована блоковая модель, о чем мы подробно говорили в предыдущей главе. Впрочем, там же мы рассматривали и способы решения.

Нам надо добавить правый отступ для блока lefti, левый отступ для блока ieft2 и левый отступ для основного текста в блоке right. Пусть все отступы будут шириной 10 пикселов. Если добавлять отступ, то надо изменять значение свойства width для всех корректных браузеров, потому что в них ширина блока определяется суммой значения свойства width и свойств

padding Однако для браузера Internet Evplorer 5 \ ничею менять не надо, потому что в нем ширина блока полностью определяется значением свойства width, тогда как отступы на ширину никакого влияния не оказывают

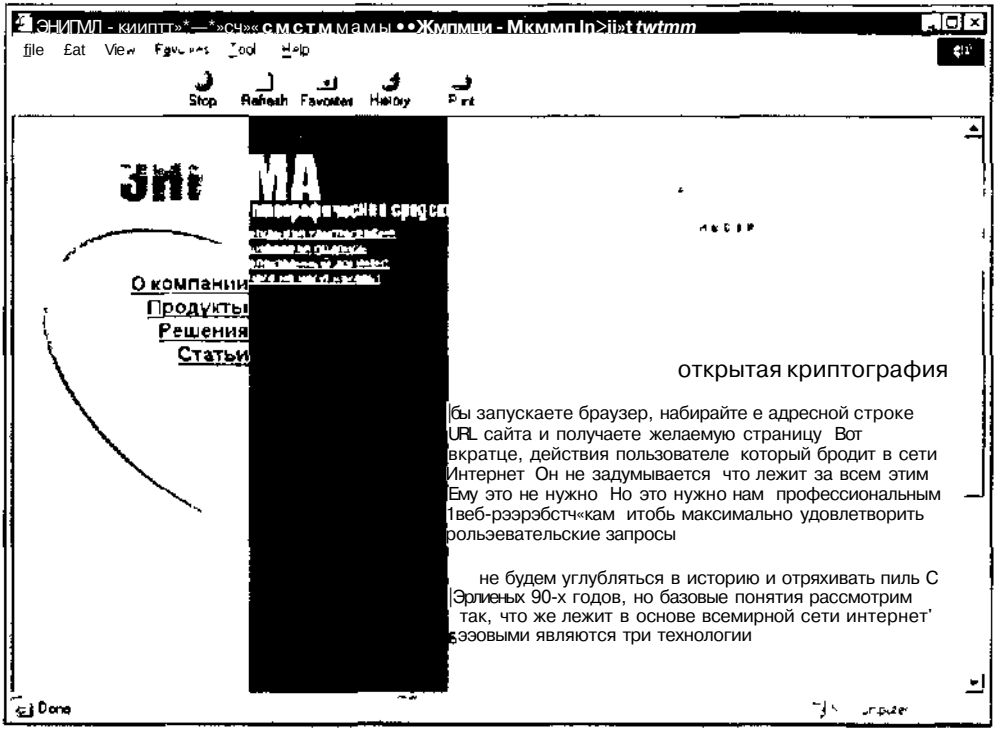


Рис. 9.13. Вид странички, сверстанной блоками, в браузере Internet Explorer 6x

Итак, *нам* надо для браузера Internet Explorer 5 v сохранить текущее значение свойства width и добавить объявление отступа, а для остальныч браузеров уменьшить на 10 пикселов значение ширины и добавить объя&тенис отступа Все вышеописанное применяется для блоков left: и left2. Для начала займемся блоком left1.

```
#leftl {
  text-align: right/
  width: 188px;
  height: 6Q0px;
  position: absolute;
  padding-right  lOpx
```

Вот мы добавили отступ, а сейчас с помощью метода Целика установим для браузеров с корректной блоковой моделью значение свойства width 178 пикселов:

```
#leftl (
  text-align: right;
  width: 188px;
  height: 600px;
  position: absolute;
  padding-right: lOpx;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 178px•
H1ML>BODY ifleftl {
  width: 178pxJ
```

Я совершенно недавно обнаружил, что браузер Internet Explorer *5.x* игнорирует правило, следующее непосредственно за правилом, в котором применяется метод Целика. Если в таблице стилей будет такой код

```
Ueftl (
  width: 188px;
  padding-right: Юpx;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 178px}
P t
  color: red}
VAR {
  color: orange}
```

то браузер проигнорирует правило P (color: red} и не выведет текст в абзацах шрифтом красного цвета, но текст в элементах <VAR> он выведет правильно, т. е. шрифтом оранжевого цвета, Поэтому в нашем случае правило

```
HTML>BODY #leftl (
  width: 178px}
```

надо ставить непосредственно после правила с применением метода Целика. Браузер Internet Explorer *5.x* его проигнорирует по двум причинам:

П он имеет баг, благодаря которому игнорирует первое правило после применения метода Целика;

О он не понимает селектор > из спецификации CSS-2.

С блоком ieft2 все делается по аналогии:

```
#left2 {
  width: 160px;
  height: 6OOpx;
  position: absolute;
  left: 188px;
  padding-laft: lOpx;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 150px)
HTML>BODY #left2 {
  width: 150px}
```

С двумя блоками мы уже разобрались, так что остался последний блок right. Отбить текст тут проще, но особенности тоже есть. Если прописать отступ для всего блока right, то будет нестыковка рисунков, потому что между ними появится 10-пикселная шель. Поэтому весь текст, включая заголовок, нужно взять в отдельный блок. Назовем его *zy.t* и напишем такой стиль:

```
#txt {
  padding-left: Юpx;
```

Итак, HTML-код у нас будет выглядеть следующим образом:

```
<BODY>
  <DIV id="left"
    <DIV id="logo">
      <IMG SRC="img/fragment_1.gif" WIDTH="266" HEIGHT»"86" ALT="3HMTMA —
криптографические средства защиты информации [логотип]">
    </DIV>
    <DIV id="left1" CLASS="back">
      <BR><BR>
      <A HREF="#">0 KOMnaHMM</A><BR>
      <A H&EГ="#">Продукты</A><Ba>
      <A HREF="#">PemeHHH</A><BR>
      <A HREF="#">CTaTbH</A>
    </DIV>
    <DIV id=left2 CLASS="bluecol">
      <A HREF="#">QTKpbrrafl криптография</A><BP>
      <A HREF="#">цифровая noflTlHCb</A><BR>
      <A HKEP»"#">электронный flOKyMeHT</A><BR>
      <A HREF="tt">4ero не могут xaKepbi</A><BR>
```

```
    </DIV>
  </DIV>
  <DIV id="righf>
    <IMG SRC="img/fragment_2.gif" WIDTH="251" HEIGHT="167" АЬТ="ЭНИГМА -
 быстрый путь к надежности">
    <DIV id-"txt">
      <BR><H3 ALIGN="right">ОТКрbiТаН криптография</Н3>
      <P>Вы запускаете браузер, набираете в адресной строке URL сайта и
получаете желаемую страницу. Вот, вкратце, действия пользователя, который
бродит в сети Интернет. Он не задумывается, что лежит за всем этим. Ему
это не нужно. Но это нужно нам, профессиональным веб-разработчикам, чтобы
максимально удовлетворить пользовательские запросы. </P>
      <P>Мы не будем углубляться в историю, и отряхивать пыль с архивных
90-х годов, но базовые понятия рассмотрим. Итак, что же лежит в основе
всемирной сети Интернет? Базовыми являются три технологии:</P>
    </DIV>
  </DIV>
</BODY>
```

Сравните его с кодом из *главы 2* и почувствуйте разницу. Логичность и четкость структуры не идет ни в какое сравнение с табличной разметкой страницы.

Что касается таблицы стилей, то из нес исключены все необязательные объявления, которые присутствовали для наглядности. Все правила снабжены комментариями. Полный и несколько оптимизированный CSS-код представлен ниже.

```
<STYLE TYPE-"text/css">
/* Устанавливаем черный цвет основного текста, белый цвет фона на страни-
це и нулевые отступы от краев окна браузера. Задаем базовый размер шрифта
V
BODY {
  color: #000;
  background: #FFF;
  font-size: lem;
  margin: Opx}
/* Устанавливаем шрифт Arial для заголовков */
H3 {
  font-family: Arial, sans-serif}
/* Устанавливаем размер и гарнитуру шрифта для текста */
P (
  font: 0.8em Verdana, sans-serif)
```

```
/* Задаем фоновое изображение для блока ссылок и запрещаем его повторение
*/
.back (
  background: url(img/fragment_3.gif) no-repeat}
/* Задаем синий фон для колонки со ссылками второго уровня */
.bluecol (
  background: #1D3D4E}

/* Задаем левый главный блок шириной 348 пикселов */
#left 1
  width: 348px;
  position: absolute}
/* Формируем первый левый блок. Задаем выравнивание ссылок по правому
краю, устанавливаем :ширину блока с помощью метода Целика */
#leftl {
  text-align: right;
  width: 188px;
  height: бООpx;
  position; absolute;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 178px}
HTML>BODY #leftl (
  width: 178px}
/* Задаем черный цвет главных ссылок и устанавливаем для них полужирный
шрифт Arial  V
ttleftlA{
  color: #000;
  font: bold lem Arial, sans-serif}
/* Формируем второй левый блок. Задаем смещение блока от левого края на
188 пикселов, устанавливаем ширину блока с помощью метода Целика */
#left2 {
  width: 160px;
  height: бООpx;
  posltion: absolute;
  left: 188px;
  voice-family: "\"}\"";
  voice-family: inherit;
  width: 150px}
HTML>BODY #leftl {
```

```
  width: 150px}
'* Задаем белый цвет ссылок второго уровня и полужирный шрифт Arial */
Heft2 A (
  color: #FFF;
  font: bold O.бет Arial, sans-serif}
'* Задаем отступ для основного текста в правом главном блоке */
-logo (
  text-align: right}
'* Задаем правый главный блок шириной 402 пиксела и сдвигаем его на 349
-1икселов вправо */
=right (
  width: 4Q2px;
  position: absolute;
  left: 348px}
'• Задаем выравнивание по правому краю для логотипа */
*txt {
  padding-left: 10px|
</STYLE>
```

Окончательный вид странички представлен на рис. 9.14.

Помните, у нас вес кода до использования CSS был 2,54 Кбайт. После частичной оптимизации — 2,28 Кбайт. А сейчас, когда CSS используется по максимуму, код весит 2.36. Вес кода увеличился из-за того, что пришлось использовать метод Целика для решения проблемы с блоковой моделью в браузере Internet Explorer 5..r. Если бы блоковая модель была корректной, то код весил бы 2,17 Кбайт. Тогда экономия составила бы 15%. Естественно, для каждого конкретного случая будет свое процентное соотношение, и чем сложнее код, тем большую выгоду даст использование CSS. Даже если взять те же 15%, то можно получить интересные цифры. Вес средней странички составляет 60 Кбайт, а при использовании CSS он уменьшится на 15%, т. е. на 9 Кбайт. Согласитесь, что 9 Кбайт — это очень значительная экономия, особенно для часто обновляющихся ресурсов с большой аудиторией.

Но уменьшение веса документа — это не главное достоинство верстки блоками. Главным является *логичность кода,* которая несет с собой легкость внесения изменений и быстрый поиск ошибок. Значительно сокращается время, затрачиваемое на верстку страницы. Ценность рабочего времени знают все, так что я не буду но этому поводу особо распространяться, но, как вам уже известно, лишний час для Quake 3 никому не помешает. Заметьте, что все эти рассуждения относятся только к легким сайтам с относительно простой структурой. Если же брать сложные сайты, то возникают серьезные проблемы. Дело в том, что при сложной структуре браузеры иногда начинают вести себя совершенно непредсказуемо, потому что сама

блоковая модель и схемы позиционирования являются достаточно новым! технологиями, так что в их реализации встречаются ошибки Без всякой сомнения, они практически исчеэн\т в самом ближайшем будущем, так чк через год-полтора верстка блоками будет столь же предсказуема, как и вер стка таблицами Надо сказать, что даже сейчас у браузеров Netscape 6 2 i Opera 5+ проблем практически нет, что откровенно рад>ет

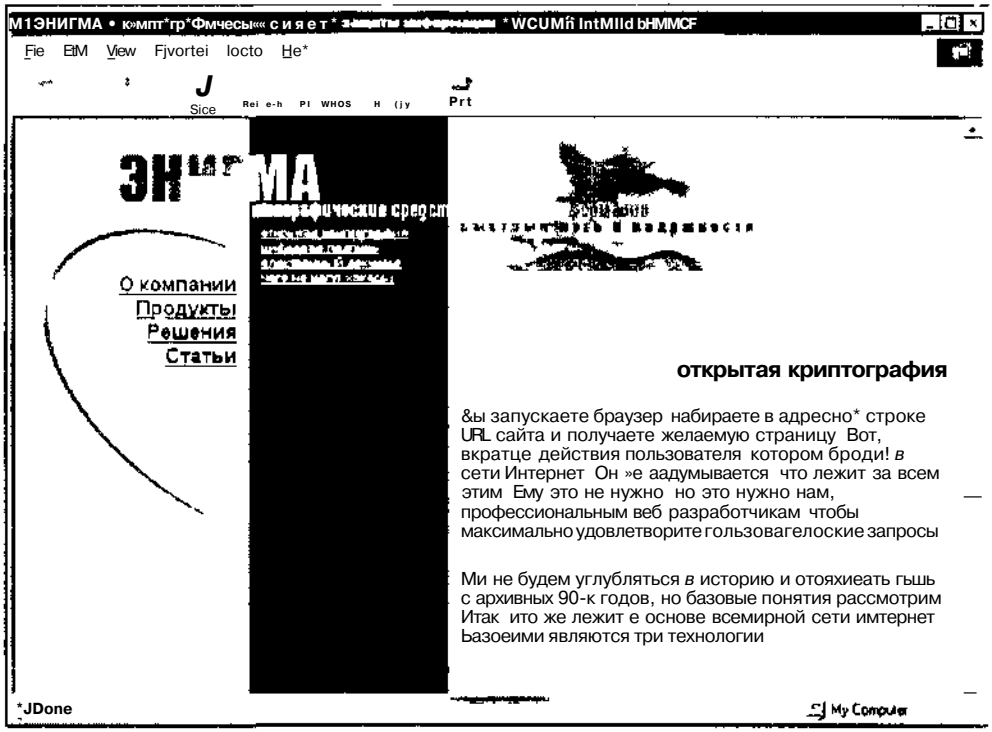


Рис. 9.14. Окончательный вид странички сверстанной блоками, в браузере Internet Explorer 6x Добавлены отступы

Есть в абсонотном позиционировании еще одна особенность Помни к свойство ngnr, которое смещает элемент относительно правого края[9] Па пример, если указать в стилях для элемента <P> правило

```
P {
  position: absolute;
  right- ЮОpx}
```

то абзац текста будет смешен относительно правого края окна браузера н 100 пикселов Заметьте, что смещение формируется именно *относшпе.гьн< правого края,* т е можно делать сайты выровненные как по левому, так и m

правому краю. Сайт, выровненный но правому краю окна браузера, пока смотрится достаточно свежо и непривычно.

Фиксированное позиционирование

Кроме всего прочего, есть интересный подвид абсолютного позиционирования. Назовем его фиксированным позиционированием. Особенность его в том, что фиксированный элемент *всегда* будет присутствовать в области просмотра, т. е. при скроллировании он все равно будет виден. Это очень похоже на фреймы, при использовании которых мы тоже можем прокручивать содержимое одного фрейма, тогда как содержимое второго фрейма будет на виду. Но у фреймов множество недостатков, тогда как у фиксированного позиционирования их гораздо меньше.

Самой большой проблемой является то, что фиксированное позиционирование поддерживается только браузерами Opera 5+ и Netscape 6.1+. Браузеры фирмы Microsoft рассматривают фиксированные блоки так, словно они находятся в нормальном потоке. По этой причине вы вряд ли будете пользоваться фиксированным позиционированием, но немного поговорить о нем стоит.

Итак, эта схема включается с помощью объявления position: fixed. Вот простейший пример. Есть у нас колонка текста, и мы хотим, чтобы на экране слева всегда присутствовало меню. Делается это так:

```
#menu i
  line-height: 2;
  width: 200px;
  position: fixed;
  left: 10px;
  border: lpx solid #CCC}
#txt 4
  width: 3OOpx;
  position: absolute;
  left: 240px}
. . .
<BODY>
  <DIV icfc="menu">
    <A HREF="#">nepBbiu раздел</A><BK>
    <A HREF="#">BTopoU раздел</A><BK>
    <A HKEP="#">Третий раздел</A><BK>
  </DIV>
  <DIV id="txt">
```

```
    <P>Здесь идет текст, который и будет скроллироваться, но меню все
равно останется на своем месте.</P>
  </DIV>
</BODY>
```

Я немного прибавлю текста, и, глядя на рис. 9.15, вы сможете представить внешний вид странички в браузере Mozilla 0.9 7.

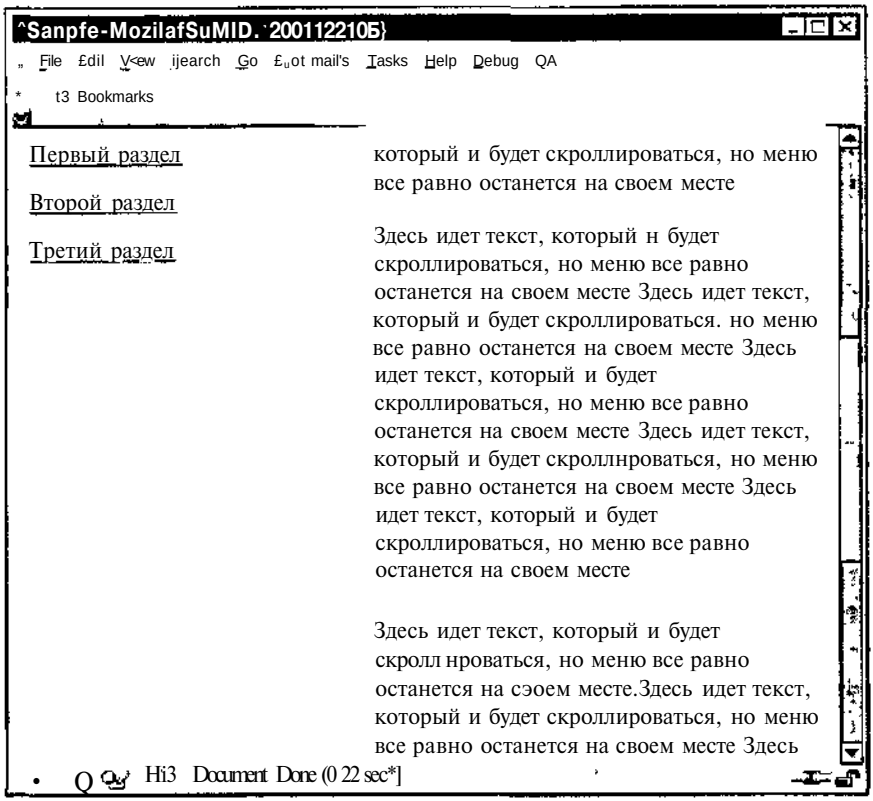


Рис. 9.15. Вид в браузере простейшей странички с меню, фиксированным слева

Обратите внимание, что полоса прокрутки находится посредине, т. е. некоторая часть текста уже проскроллнрована. а меню осталось видным. Как бы мы не скроллировали текст, все равно меню будет присутствовать в левом верхнем углу браузера, даже если будет перекрывать контент. Например, если мы не станем позиционировать блок с текстом, а оставим его в нормальном потоке, то блок ссылок на него будет наложен, и текст читать в этом месте станет невозможно. Так что надо внимательно следить, не может

ли принципиально фиксированный блок перекрывать какие-либо важные элементы при скроллировании. Таких ситуаций желательно избегать.

Что касается стабильности и адекватности, то абсолютное позиционирование однозначный лидер по этим параметрам. Оно очень простое, его легко изучить и серьезных багов оно не имеет. Так что для несложных сайтов с фиксированной шириной эта схема является наиболее подходящей. Точнее говоря, она является единственно подходящей.

Нам осталось рассмотреть последнюю схему позиционирования, которая называется...

Плавающая модель

Она принципиально отличается от абсолютного позиционирования тем, что здесь нет такой точности и нельзя до пиксела позиционировать блоки. Однако без нее не обойтись в том случае, если необходима "резиновая" верстка сайта. Кроме того, плавающая модель является очень подходящей для верстки в несколько колонок. Собственно говоря, как раз для этого она и задумывалась, так что попробуем в ней разобраться.

Итак, если элемент обозначен как плавающий, то он вырывается из нормального потока (как и при абсолютном позиционировании) и "прилипает" к правому или левому краю контейнера, а текст и графика обтекают его. Если элемент "прилипает" к левому краю, то текст обтекает его справа, и наоборот. Плавающая модель для элемента устанавливается с помощью свойства float, которое может принимать три значения: left, right и none. При объявлении float: left элемент "прилипает" к левому краю контейнера, а при объявлении float: right, соответственно, к правому. Если же указано float: none, то элемент не перемещается.

Для начата простейший пример. Создадим блок, который будет "прилипать" к правому краю контейнера, а текст — обтекать его слева. Блок будет иметь серый фон, толстую черную рамку и отступы со всех сторон для лучшего восприятия текста. Код блока будет такой:

```
#right {
  background: #CCC;
  width: 200px;
  border: 5px solid #000;
  padding: 1em;
  float: right)
```

А HTML-код вообще простой:

```
<DIV id»"right">
  Это блок, который будет "прилипать" к правому краю контейнера.
```

```
</DIV>
<P>Здесь идет текст, который будет обтекать блок. Вообще говоря, плаваю-
щая модель достаточно сложная вещь, которая имеет множество нюансов и
багов в реализации, так что крови она попортит нам немало. Однако девать-
ся некуда..,</P>
```

Тогда в браузере нам откроется картина, представленная на рис. 9.16.

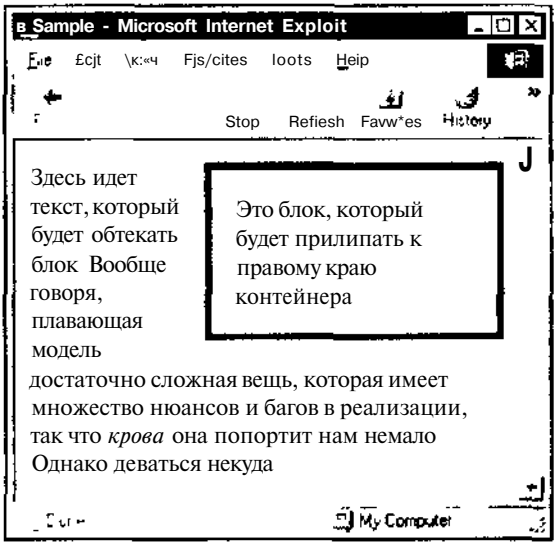


Рис. 9.16. Элементарный плавающий блок

Обратите внимание на следующее правило.

> ***Правило***
>
> Для плавающих блоков надо всегда указывать ширину с помощью свойств w-dt:., потому что в противном случае блок будет растягиваться, чтобы BV-стить в себя весь контент.

В HTML был атрибут ALIGN у элемента <IMG>, так что можно было застав лять текст обтекать рисунки. С помощью CSS можно легко сделать так, что бы обтекались блоки текста. На основе этого можно делать врезки, в кок рые помещать подсказки, отступления от текста, полезные советы — тог~ страница будет выглядеть гораздо интереснее и разнообразнее, а в конечно итоге привлекательнее. Читать текст на экране монитора вообще труднес чем текст, напечатанный на бумаге, так что любое усовершенствование, of легчающее процесс чтения, очень ценно само по себе. По этой причине текст для веб пало максимально структурировать, и везде, где это возможно, использовать списки и подобного рода врезки.

Вот уже нашли одно достаточно явное, но, вместе с тем, полезное применение для плавающей модели.

Для дальнейшего изучения плавающей модели надо рассмотреть одну проблему. На рис 9 16 текст обтекает блок. Иногда совершенно необходимо, чтобы текст начинался непосредственно *под* блоком.

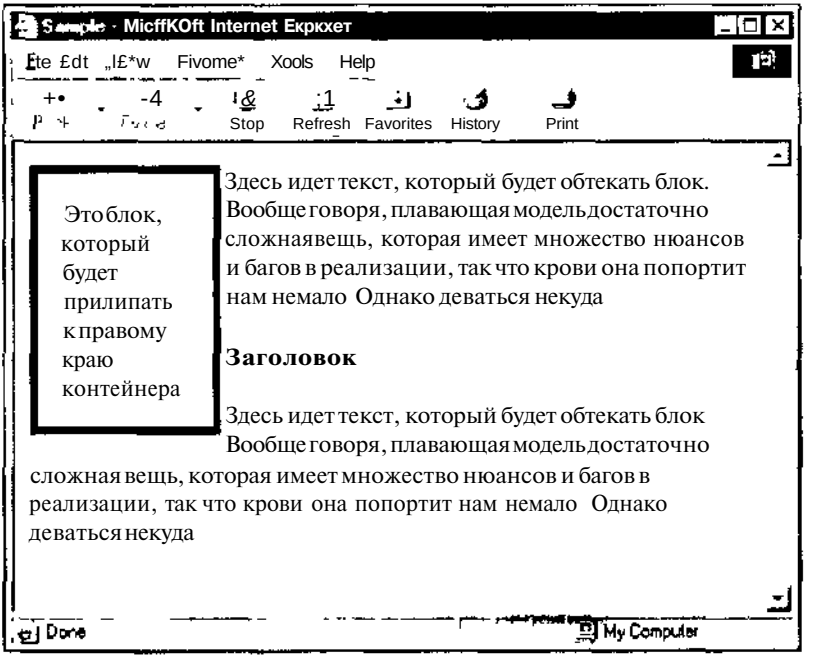


Рис. 9.17. Плавающий блок и обтекающий его заголовок

Например, идет у вас текст, обтекающий врезку, и вдруг начинается новый раздел с заголовком (рис. 9.17). Код такой:

```
#right {
  background: #CCC;
  width: 120px;
  border: 5px solid #000;
  padding: 1em;
  float: left'-
. . .
<DIV id="right">
  Это блок, который Судет "прилипать" к правому краю контейнера„
</DIV>
```

<P>Здесь идет текст, который Судет обтекать Олок. ВооОще говоря, плавающая модель достаточно сложная вещь, которая имеет множество нюансов и багов в реализации, так что крови она попортит нам немало. Однако девать-ся некуда...</P>

<H3>Заголовок</H3>

<P>Здесь идет текст, который будет обтекать блок. ВооОще говоря, плавающая модель достаточно сложная вещь, которая имеет множество нюансов и Оагов в реализации, так что крови она попортит нам немало. Однако деватч ся некуда...</P>

Заголовок должен располагаться непосредственно после рисунка, а не слсь и не справа от него. И это можно сделать с помощью свойства clear, которое принимает следующие значения:

- О left — блок перемещается ниже все\ плавающих блоков с объявление float: left;
- П right — блок перемешается ниже всех плавающих блоков с объявление[1] float: right;
- П both — блок перемещается ниже всех плавающих блоков;
- О попе — ограничений на обтекание нет.

Поэтому для того, чтобы заголовок начинался после врезки, нам надо прс писать для него объявление clear: left:

```
H3 {
  clear: left}
```

И картина волшебным образом примет необходимый вид, показанным \ рис. 9.18.

Что же произойдет в том случае, если мы создадим два блока и в стилях • обоим напишем объявление float: left? По этому поводу в спенификаии, CSS-2 сказано следующее.

Правило

Если текущий плавающий блок является левосторонним (то есть имеет объяв ление float: left) и существует еще хотя бы один левосторонний плаваы щий блок, который был сгенерирован в исходном документе раньше него, т левый внешний край текущего блока должен находиться справа от право: внешнего края предыдущего блока, или верхний его край должен быть ниже чем нижний край предыдущего блока

Получается, что блоки будут располагаться на одной юрпзонталп в том случае. если их суммарная ширина меньше или равна ширине контейнера. Поясню на примере. Если у нас окно браузера имеет ширину 500 пикселов, и мы создал - два блока шириной по 200 пикселов, то они будут расположены на одной гори зонтади, образуя две колонки. Если же ширина каждого блока будет 300 пиксс лов, то второй блок будет находиться под первым. Исходя из этого, сразу же

можно сделать вывод, что указывать ширину блоков в абсолютных единицах при плавающей модели плохо. Посудите сами, если вы рассчитывали на ширину окна браузера 800 пикселов и сделати три блока по 260 пикселов, то, если пользователь уменьшит размеры окна, блоки будут располагаться один под одним, так что дизайн развалится окончательно и бесповоротно. Для жесткой версгки лучше пользоваться абсолютным позиционированием. Так что при плавающей модели ширину блоков надо указывать в процентах.

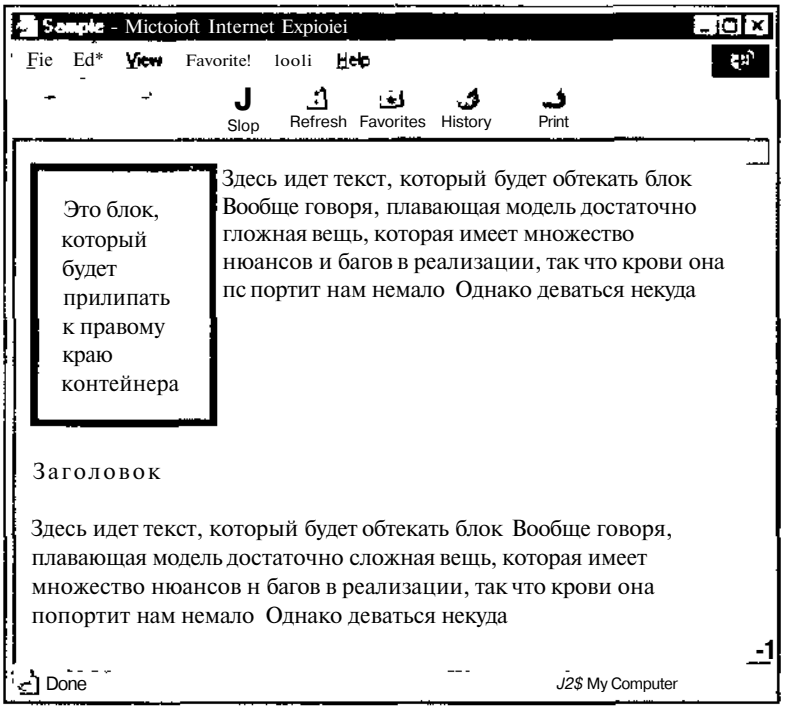


Рис. 9.18. Иллюстрация работы свойства clear

Давайте для начала создадим три плавающих блока. Несколько предварительны)! замечаний. При использовании плавающей модели обязательно задавайте поля для элемента <BCDY>, потому что в противном случае браузер Internet Explorer 5.v увеличивает в два раза отступы плавающих блоков. Кроме того, надо очень осторожно использовать одновременно проценты и :шксслы, потому что может возникнуть ситуация, когда суммарная ширина блоков превысит из-за этого 100% и последний блок будет располагаться под остальными. Пока создадим псе три блока совершенно одинаковыми:

```
BODY I
  margin: Opx}
```

```
#left {
  background: #CCC;
  width: 26%;
  height: 90%;
  padding: 2%;
  border: lpx solid #000;
  float: left}
#center {
  background: #CCC;
  width: 28%;
  height: 90%;
  padding: 2%;
  border: lpx solid #000;
  float: left}
#right {
  background: %CCC;
  width: 28%;
  height: 90%;
  padding: 2%;
  border: lpx solid #000;
  float: left}
. . .
<DIV id="left">
  <B>Левая колонка</B>, хоть и не очень широкая, но нам хватит.
</DIV>
<DIV id="center">
  <B>Центральная колонка</B>. Тоже не слишком широкая. Но нам больше и .-.
надо. Зато в нее можно поместить очень много текста, так что она станет
выше всех остальных колонок.
</DIV>
<DIV id«"righf>
  <B>Правая колонка</B>. Про нее и сказать нечего.
</DIV>
```

Что интересно, в браузерах картина будет разная. На рис. 9.19 показано, кабудут выглядеть наши три колонки в Internet Explorer 6.

А вот в браузере Mozilla 0.9.7 нам откроется совсем другая картина (рис. 9.201

Видим одну принципиальную разницу: несмотря на то, что высота для блс ка задана явно (height: 90%), работает это только в Mozilla, тогда к;— Internet Explorer устанавливает высоту блока по величине контента, что корне неверно. Однако если задавать высоту блока в пикселах, то такс

проблемы не будет, в этом случае высота задается корректно и правильно. Можно рекомендовать задавать ширину плавающих блоков в процентах, а высоту — в пикселах. Однако здесь возникает проблема другого плана, которая связана с привычками пользователя.

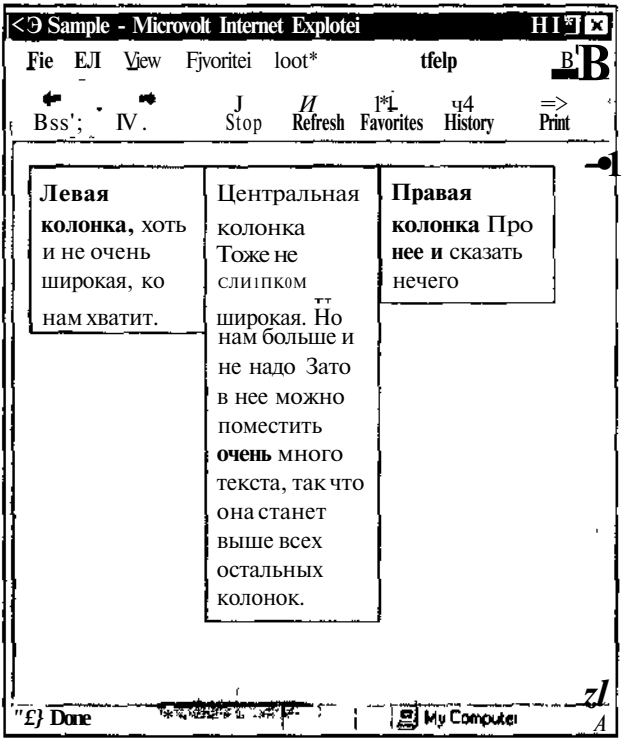


Рис. 9.19. Три плавающих блока в браузере Internet Explorer 6 *x*

Обратите внимание, что сайты практически всегда сделаны так, чтобы не было горизонтальной полосы прокрутки. Естественно, что страницы бывают достаточно большими, так что не вмешаются на одном экране (тем более, .кранные разрешения бывают разными). Поэтому подавляющее большинство сайтов имеют вертикальную полосу прокрутки. Получается, что ширина страницы имеет предсказуемое значение, по крайней мере, можно с уверенностью сказать, что она не превышает ширины окна браузера. Тогда как высота страницы может варьироваться в зависимости от объема контента: на одной странице сайта может быть короткий пресс-релиз, который занимает 50% от высоты окна браузера, а на другой странице большая статья, которая занимает 400^ от высоты окна браузера. С этой точки зрения жестко задавать высоту блоков нельзя по той причине, что контент, превышающий эту высоту, в браузерах Opera 5+ и Netscape 6+ будет выступать за пределы бло-

ка, не изменяя высоту самого блока, а в браузере Internet Explorer 5+ высота блока будет увеличена настолько, чтобы блок вместил в себя весь контент. Вообще, поведение браузера Internet Explorer 5+ некорректно с точки зрения спецификации CSS, однако на вопрос, какое поведение лучше, дать однозначный ответ сложно. Если говорить о ширине, то поведение браузеров Opera 5+ и Netscape 6+ лучше, потому что ширина должна задаваться достаточно жестко и не зависеть от контента блока. Если же говорить о высоте, то, наоборот, поведение браузера Internet Explorer 5+ лучше, потому что жестко задавать высоту практически всегда невозможно (за исключением редких случаев, когда есть возможность полностью контролировать контент), так что нет смысла и жестко ограничивать высоту блока.

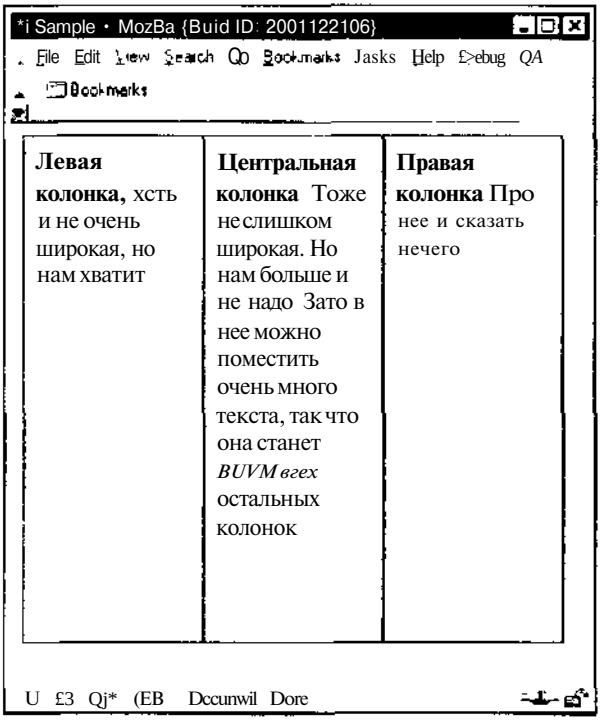


Рис. 9.20. Три плавающих блока в браузере Mozilla 0.9.7

На мой взгляд, идеальным решением было бы совместить модели поведения обоих браузеров, однако в ближайшем будущем этого не произойдет наверняка, да и в отдаленном будущем вряд ли.

Сейчас я вам открою тайну. На самом деле проблема с высотой блоков — это самая большая проблема при верстке без таблиц, которая в некоторых случаях (к счастью, достаточно редких) *не имеет решения*. Какие именно эк

случаи, мы рассмотрим в *главе* Д посвященной верстке сайта от начала и до конца.

Нам осталось познакомиться с еще одним понятием, которое может приголмться при верстке блоками. Это слои.

Слои

Любой блок имеет координаты на плоскости экрана, однако в CSS-2 есть и третья координата, которая описывает положение блока в пространстве. Что такое координаты x и y на экране монитора — объяснять не надо, а вот что такое координата z пожалуй стоит пояснить. Представьте, что у вас есть несколько прозрачных пленок размером с экран. Вы взяли, нарисовали чго-то на каждой пленке и прикрепили их к экрану. Каждая пленка представляет .обой отдельный слой, а удаленность ее от поверхности монитора и будет характеризоваться координатой z. Как видите, она достаточно специфичная, поэтому ее бессмысленно выражать в пикселах или любых других единицах [лины, т. е. имеет значение только относительное положение слоев: какой .и них ближе к поверхности, а какой дальше. Именно так и реализована [чоординашая ось z в CSS-2, а положение слоя задается свойством ?.-index, которое может принимать только целочисленные значения. Чем выше зна- 1сние. тем дальше от поверхности экрана располагается слой. Например

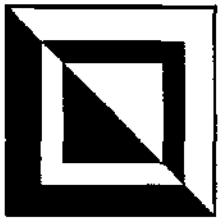
```
*deep {
  position: absolute;
  left: 100px;
  z-index: 1
ahigh i
  position; absolute;
  top: 220px;
  z-index: 2'•
```

Слой high будет дальше от поверхности, т. е. ближе к пользователю. При лом содержимое этого слоя будет перекрывать содержимое слоя deep.

Где может пригодиться использование слоев? Во-первых, если вы хотите создать анимацию на основе DHTML, то без слоев не обойтись. Во-вторых, при создании различного рода выпадающих меню тоже надо указывать -mdex, чтобы меню распола(алось максимально ближе к пользователю, 1. е. поверх всех остальных слоев, иначе его просто не будет видно.

Итак, в этой главе вы познакомились с теорией позиционирования в целом, по некоторые особенности можно выявить только на практике. Поэтому одна из следующих глав будет полностью посвящена верстке сайта, причем особое внимание будет уделено именно позиционированию. Подробное обоснование выбора схемы позиционирования, выявление слабых сторон всех остальных схем, создание блоков и многое другое ждет вас впереди.

Глава 10



Браузеры: несовместимость и непримиримая борьба

Пожалуй, именно по причине рыночной борьбы и конкуренции верстать сайты *правгиьно* до сих пор так сложно Вернее, в последнее время ситуация стала изменяться к лучшем) и достаточно быстрыми темпами, но все же проблема существует по сей день.

Нас с вами в отдельно взятом браузере особенно интересуют две веши:

О насколько он поддерживает стандарты CSS-1 и CSS-2;

П какие особенности имеет этот браузер в реализации поддержки CSS.

Эти два вопроса нас интересуют как разработчиков. Если же посмотреть на браузер с точки зрения пользователя, то вопросы кардинально изменятся. Пользователей интересует:

О быстрота загрузки страниц и собственно самого браузера, как приложения;

П правильность отображения страниц (здесь надо заметить, что причины неправильного отображения его не интересуют совершенно, а только сам факт, чего не скажешь о разработчиках);

- безопасность данного браузера (многие обращают на нее особое внимание, потому что от этого в некоторой степени зависит конфиденциальность пользователя в Сети);
- стабильность работы (естественно, никому не нравится, когда браузер выдает внутреннюю ошибку, и операционная система закрывает 14 ранее открытых окон с найденной неимоверными усилиями информацией).

Это основные объективные критерии, по которым пользователи выбирают себе браузер. Кроме них существует большое количество субъективных обстоятельств, таких как удобство интерфейса, простота установки, мнение окружающих, ненависть к фирме Microsoft и тысяча других. Субъективные причины нас не интересуют, а объективные мы оценим.

Исходя из интересов пользователей, можно выделить критерии, по которым характеризуют браузеры. Общих критериев получается несколько.

П *Быстрота и производительность.* Это понятие включает в себя, насколько быстро браузер может загрузиться сам, с какой скоростью он форматирую

ет HTML-страницы со сложной табличной верстской, насколько быстро он прокачивает рисунки и работает с кэшем.

- *Поддержка стандартов.* Основная функция браузера — корректно отображать все страницы. Уже прошли те времена, когда каждый браузер имел множество собственных особенностей, касающихся верстки, потому что не было единого органа, который разрабатывал стандарты. С появлением консорциума W3C поддержка стандартов браузерами стала желательной, а с недавних пор и обязательной. Именно стандарты сейчас определяют корректность отображения страниц, т. е. их поддержка является главным и необходимым атрибутом любого браузера. Естественно, п контексте данной книги нас больше всего интересует поддержка CSS, так что ей будет уделено достаточно много внимания.

- *Интерфейс.* Продуманность и логичность интерфейса необходимы для любого программного продукта и для браузера в частности. Чю касается привлекательности интерфейса, то она в большей степени основана на субъективном восприятии пользователя, поэтому ее мы оценивать не будем.

O *Фишки.** Наиболее краткая и емкая формулировка понятия "Специфические особенности браузера". Наряду с интерфейсом, набор фишек является основным признаком отличия данного браузера от всех остальных.

G *Безопасность.* В связи с распространением электронной коммерции многие стараются заботиться о безопасности номеров своей кредитной карточки. Для стран СНГ это пока не очень актуально, но вот на Западе вопрос безопасности стоит чрезвычайно остро, так что многие просто панически боятся хакеров и прочих сетевых страшилок. Однако плохо то, что в основном мнение о безопасности того или иного браузера у пользователя складывается на основе рекламы и пиара. Так, например, очень многие считают, что самые безопасные браузеры выпускаются фирмой Microsoft, но это совершенно не так.

Четвертое поколение браузеров постепенно сходит с арены, так что целесообразно начать рассмотрение сразу с пятых версий. На момент написания этой книги самым распространенным браузером в Сети был Microsoft Internet Explorer 5.x, однако новая версия Internet Explorer 6.0 постепенно вытесняла предыдущую, но к моменту выхода книги из печати ситуация кардинально не изменится.

Далее изложение строится следующим образом. Каждый браузер характеризуется по всем упомянутым выше пунктам, но, естественно, особое внимание уделяется каскадным таблицам стилей. В конце главы будет представлена сводная таблица, содержащая все значимые особенности браузеров.

Для чего это надо? Во-первых, это поможет вам легко ориентироваться в браузерах и отличать плохое от хорошего. Во-вторых, вы сможете сделать

---

* Сленг. — *Ред.*

однозначный выбор для себя пли, по меньшей мере, переосмыслить свое отношение к тому или иному браузеру. Возможно, именно эта глава послужит стимулом для перехода на другой браузер. Что касается меня, то я постараюсь дать максимально объективную информацию. Итак, начнем с лидера рынка.

Microsoft Internet Explorer 5.x

Первая бета-версия вышла в 1998 году и, по сути дела, именно она послу жила причиной массового перехода пользователей на браузеры фнрмь Microsoft. Причины этой миграции станут вам вскоре ясны и прозрачны Для начала рассмотрим производительность.

Быстрота и производительность

На то время эго был, вне всяких сомнений, один из самых производительна браузеров. Специалисты из CNET Labs сравнили Internet Explorer 5.v с браузерами Internet Explorer 4.v и Netscape Navigator 4.5. Почти по всем параметрам *v* превосходил их. Пройдемся по пунктам тестирования и убедимся в этом сами

ПроизводительностьJava

- Explorer 5.л - 100%.
- Explorer 4л- - 98%.

CI Netscape 4.5 — 64%.

Как видите, весьма небольшое улучшение по сравнению с четвертой верен ей, однако ближайший конкурент отстает на 36%, что достаточно много.

Форматирование сложных таблиц

O Explorer 5.V - 100%.

П Explorer 4.x - 95%.

П Netscape 4.5 — 22%.

Вообще браузеры фирмы Netscape всегда отличались медленной обработка вложенных таблиц. Вероятно, алгоритмы были крайне несовершенны. Toi, как уже в браузере Internet Explorer 4.x таблицы форматировались достато[1] но быстро, и в пятой версии улучшение весьма несущественное.

Скорость загрузки графики и текста

- Explorer 5.x — 88%.

a Explorer 4.x — 67%.

- Netscape 4.5 - 100%.

Это единственный (хотя и очень важный) параметр, по которому Netscape быстрее Microsoft. Сам контент страниц загружается намного быстрее, так что на слабых каналах именно этот браузер гораздо предпочтительнее. Однако прогресс налицо и 12% не являются критическим отрывом.

Загрузка кэшированных страниц

- Explorer *5.x* - 100%.
- *П* Explorer 4.x — 67%.
- а Netscape 4.5 — 34%.

Как видите, работа с кэшем в Internet Explorer тоже сделана лучше, причем намного. Причина в том. что разработчики пересмотрели механизм кэширования. Браузер стат кэшировать страницы, объединяя разметку страницы с графикой и текстом. Особенно это чувствуется при нажатии кнопки **Назад** в браузере. Тогда преимущество Internet Explorer 5.Y перед Netscape 4.5 может достигать 85%!

Вообще если подводить итог, то можно сказать, что разработчики смогли значительно улучшить кэш и скорость загрузки контента, вес остальное осталось без особых изменений. Однако на фоне и без того отстающего браузера Netscape отрыв серьезный

Интерфейс и фишки

Браузеры фирмы Microsoft отличаются тем, что внешний вил интерфейса в них остается практически неизменным начиная с четвертой версии. И это по большому счету очень хорошо, потому что пользователи привыкают к интерфейсу и переход, скажем, с браузера Internet Explorer 5.v на Internet Explorer 6.v значительно облегчается, тогда как браузер Netscape 6.v концептуально отличается от браузера Netscape 4.л\

Возможно, при таком подходе теряется ощущение новизны, однако это не столь важно. По сравнению с большинством других браузеров, интерфейс Internet Explorer очень лаконичен и ненавязчив. Стандартный серый фон с неброскими серыми кнопочками, которые подсвечиваются **при** наведении мышкой. Стандартное меню и привычная всем адресная строка. Ничего лишнего вы на панели не найдете, а если и найдете, то сможете удалить с экрана, благо есть настройки интерфейса, которые позволяют это сделать. Эта возможность настроить браузер под себя вызывает у пользователя субъективное удовлетворение, что хорошо.

Браузер Internet Explorer 5.v имеет следующие фишки.

- *О* Сохранение страниц целиком. Это первый браузер, где была реализована такая опция. Она означает, что вы можете сохранить HTML-страницу и все файлы, которые с ней связаны, включая графику, CSS-файлы, скрип-

товые файлы. Достаточно удобно, если надо внимательно изучить дизайн или код, а времени и денег в режиме online на это жалко. Можно смело сохранять страницу целиком и в дальнейшем ее изучать.

- Вид Во весь экран. При нажатии на клавишу <F11> браузер переключается в полноэкранный режим. При этом убираются титульная, адресная и статусная строки, убирается также все меню, за исключением узкой навигационной панели. Данная опция полезна для обладателей небольших мониторов с разрешением 800x600 и ниже, поскольку значительно экономит экранную площадь, сокращая элементы интерфейса до минимума.

П Автоматическое исправление типичных ошибок при вводе адреса. Например, http:/ автоматически заменяется на hup://.

- Автоматическое заполнение форм и адресов. Ели вы когда-то вводили *в* поле, скажем, E-mail какую-то информацию, то браузер предложит вам ее для ввода в следующий раз, так что набирать адреса, электронные адреса, имена и фамилии становится проще

D Встроенный механизм поиска позволяет проще начодить нужную информацию.

- Интеграция с сервисами компании Microsoft. По большому счету, она нужна только западным пользователям.

П В браузере версии 5.5 появилась очень полезная функция предварительного просмотра страниц перед печатью.

Конечно, сам по себе набор характерных свойств мало о чем говорит, потому что надо с чем-то сравнивать. В конце главы помешена сводная таблица, а пока можно сказать, что по сравнению с четвертой версией браузера улучшения достаточно значительные.

Безопасность

Как это ни странно, но браузеры Microsoft большинством пользователей считаются самыми безопасными. Странно это по той простой причине, что но самом деле Internet Explorer *самый уязвимый браузер.* Он содержит самое большое количество дыр в защите, начиная от мелких и незначительных и заканчивая критическими, когда хакер может получить полный доступ к вашем> компьютеру. Так что безопасность браузеров Microsoft не более чем миф.

Вы можете установить один из четырех уровней безопасности, но кроме этого есть и более детальные настройки. Например, можно запрещать загрузку файлов в элемент <IFRAME> (С ЭТИМ элементом связан один очень серьезный баг, благодаря которому можно запускать исполняемые файлы на компьютере пользователя), отключать поддержку ActiveX и Java, отключать cookies и др. Несмотря на это, диапазон настроек достаточно узок и ограничен по сравнению с остальными популярными браузерами.

С выходом шестой версии ситуация практически не изменилась, так что браузеры Internet Explorer 5лг—6JC по прежнему вчистую проигрывают по этому критерию браузерам Netscape и Opera.

Стандарты

Пожалуй, это самый значимый пункт для веб-разработчиков и самый неинтересный для пользователей. Их совершенно не волнует, поддерживает ли браузер XML и в какой степени. Пользователя интересует только вид страницы в браузере, так что адаптировать свой код под популярные браузеры обязан любой хороший HTML-верстальщик.

Стандарты хороши именно тем, что *не надо подстраиваться под отдельно взятый браузер,* надо просто соблюдать правила, изложенные в спецификации, и ваш код будет корректно отображаться в любом браузере, полностью поддерживающем стандарты. Естественно, полная поддержка не является гарантом полной одинаковости отображения страниц, потому что в спецификациях тоже есть свои слабые места, которые можно трактовать поразному. Однако это уже не является критичным, потому что небольшие различия в реализации поддержки HTML приводят к небольшим отличиям при отображении страниц браузерами (это практически всегда так).

Разработчиком стандартов является консорциум W3C. Так что у нас есть возможность протестировать каждый браузер и выявить несоответствия.

Сразу скажу, что на то время браузер Internet Explorer 5_r соответствовал требованиям стандартов достаточно хорошо.

HTML

В целом обеспечивается хорошая поддержка HTML 4.О. Однако крайне негативным является то, что по сравнению с Internet Explorer 4.v исправили только три бага и на этом остановились. Что осталось плохого и неправильного.

- Не поддерживается элемент <ABBR>, которым должны обозначаться аббревиатуры.
- Не поддерживается элемент <Q>, который должен обеспечивать выделение кавычками, что удобно для цитат, потому что сейчас приходится выделять цитаты знаками « И firaquo;.
- Неполная поддержка элемента <OBJECT> И серьезные проблемы с ним. Это одна из причин (возможно главная), по которой данным элементом пользуются весьма ограниченно и крайне осторожно. Кстати, именно с этим элементом связана одна из самых опасных дыр в безопасности, используя которую, можно запускать программы на машине пользователя.
- Несколько некорректная работа с таблицами, хотя проблемы незначительные и легко решаемые.

П Есть проблемы с вложенными элементами.

• Есть проблемы с формами, многие из которых были решены в версии 5.5.

В общем-то, странно, что разработчики семейства браузеров Internet Explorer за полтора года не смогли улучшить поддержку HTML браузером Однако факт остается фактом.

XMUXSL

Браузер достаточно хорошо поддерживает XML, однако сама по себе полдсржка XML не дает никаких выгод по сравнению с HTML, потому чю XML-документ надо каким-то образом форматировать. Форматировать ни стороне клиента его можно с помошью каскадных таблиц стилей или же XSL. Исходя их эти возможностей, существует два принципиальных подход.) к XML-документам. Первый подход заключается в использовании CSS для форматирования. Главные достоинства.

О Этот стандарт давно рекомендован консорциумом W3C и в той или иное степени уже реализован в браузерах, так что дальнейшее его внедрение не потребует больших усилий.

П Каскадные таблицы стилей достаточно привычны разработчикам, и пере ход будет менее болезненным.

Недостаток, естественно, тоже имеется. Вот он:

Каскадные таблицы стилей в принципе недостаточно функциональны, чтобы максимально использовать все преимущества XML, что уменьшает полезность перехода.

Как видите, у сторонников связки CSS + XML есть серьезные доводы. Они считают, что на данном этапе гораздо проще и лучше ре&1изовать хороший уровень поддержки CSS, чем внедрять абсолютно новую технологию. Однако такая позиция противоречит некоторым принципам XML.

Второй подход заключается в использовании XSL для форматирования XML-документов. Его достоинство.

XSL изначально разрабатывался под XML, так что возможности такой связки намного шире.

Недостатки.

П Стандарт XSL 1.0 был рекомендован только в октябре 2001 года, так что он является сравнительно молодым.

• Язык XSL достаточно сложен и переход со связки HTML + CSS на связку XML + XSL верстальщикам дастся нелегко.

Сторонники второго подхода считают, что чем раньше начнется внедрение XSL, тем лучше будет для всех.

Принципиальным отличием браузеров фирмы Microsoft от всех остальных является поддержка XSL. Ни один другой браузер не поддерживает эту многообещающую технологию. Сам факт ее поддержки радует, однако на момент выхода браузера Internet Explorer *5.x* спецификация XSL еще находилась в процессе доработки и усовершенствования

Получилась следующая ситуация. Фирма Microsoft сделала упор на XSL в ущерб CSS, но все остальные разработчики браузеров поступили совершенно иначе. Они просто не стали внедрять поддержку XSL, потому что не было рекомендованного единого стандарта, а сосредоточились на CSS и ввели поддержку связки XML + CSS.

На мой взгляд, именно такая позиция правильная. Это доказывает и тот факт, что связка XML + XSL используется только в интранст-решениях, но никогда в Интернете. Причина проста Принцип общедоступности информации ведет к тому, что используются только проверенные и надежные средства, которые работают везде. К ним относятся HTML и CSS, но не XSL.

CSS

Вот эта технология нас интересует особенно, так что внимания ей заслуженно уделим больше.

Проблемы.

О Одной из самых значимых проблем является неправильная реализация блоковой модели. Вы уже знаете, что согласно стандарту CSS-1 ширина отступов и рамок не должна входить в значение ширины контента блока, которая устанавливается свойством width. Однако Internet Explorer 5.x включает ширину рамок и отступов в ширину контента. Для создания блоков это достаточно критично, потому что приходится прибегать к кросс-браузер]юму CSS, что усложняет код и значительно ухудшает логичность таблицы стилей.

- Кроме того, в браузере Internet Explorer 5JC невозможно устанавливать отступы, рамки и поля на строчные элементы, такие как <EM>, <B>, <STRONG>. Такое неравноправие строчных и структурных блоков в некотором роде оправдано, потому что для структурных блоков поля, рамки и отступы совершенно необходимы, тогда как для строчных блоков их отсутствие некритично. Естественно, такая точка зрения преступна по своей сути, но, вероятнее всего, именно так рассуждали разработчики браузера Internet Exploier 5.v, когда хотели реализовать как можно больше свойств CSS, по не полностью. Конечно, в этом случае можно написать в пресс-релизе, что блоковая модель реализована, но умолчать, что реализована она не до конца.
- Существенные проблемы возникают при использовании свойств float и clear. Так, например, возьмем строчные блоки. Если элемент с объявле-

нием float: left не является первым в строке, то он все равно должен переноситься максимально влево, т. к. он вырывается из нормального потока. Однако Internet Explorer *5.x* переносит его ниже, Вот в таком коде

```
<P>ВТОрой<SPAN STYLE="float: left">nepsblu<SPAN></P>
```

Слово "Первый" и должно стоять первым, однако оно оказывается ниже слова "Второй".

Если взять структурные блоки, то проблемы тоже есть. Допустим, у вас есть плавающий блок шириной 100%. После него идет абзац с текстом.

```
#main t
  width: 100%;
  float: left}
. . .
<DIV id="main">
</DIV>
<P>АСзац текста</p>
```

Браузер должен расположить текст снизу вслед за блоком main, потом\ что ширина блока main 100%, и ширина блока, образованного элементом <p>, тоже 100% (так как он является прямым наследником элемент.; <BODY> и ширина явно не указана). Однако Internet Explorer 5дг размещает некоторое количество контента справа от блока и ш , а некоторое количество под блоком main, вследствие чего появляется горизонтальная полоса прокрутки. Решение есть, надо просто написать в стилях правило:

```
P {
  clear: left}
```

Что будет принуждать браузер начинать выводить текст после блока.

G Есть проблемы с наследованием. Оно нарушается при использовании таблиц. Например, если вы указали размер шрифта для документа так

```
BODY {
  font-size: 12px}
```

то такой размер должен принимать весь шрифт на странице. Однако если в коде присутствует таблица, то размер шрифта внутри таблицы будет совершенно не таким. Вот типичный пример:

```
<BODY>
  <P>Текст, который будет выводиться шрифтом размером 12 пикселов.</?>
  <TABLE>
    <TR>
      <T0>Текст, который будет выводиться шрифтом, установленным в
браузере, в данном случае он установлен в значение "средний".
```

```
        </TD>
      </TR>
    </TABLE>
</BODY>
```

Как этот код будет выведен браузером видно из рис. 10.1.

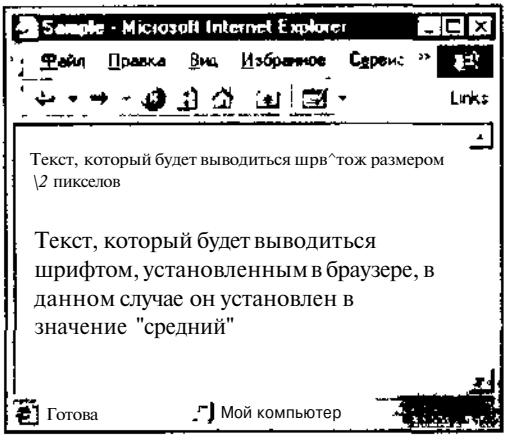


Рис. 10.1. Нарушение механизма наследования в браузере Internet Explorer 5.x при использовании таблиц

Эта проблема стоит особенно остро, потому что табличная верстка применяется повсеместно. Решение есть, надо указывать размер шрифта и для элемента <TD>, Т. е. правило станет таким:

```
BODY, TD {
  font-Size:  12px}
```

- Еще в браузере Internet Explorer 5.v не реализован достаточно интересный и полезный механизм альтернативных таблиц стилей. Мы уже немного говорили о нем. Вообще, для чего он нужен? Одним из главных принципов сети Интернет является *общедоступность информации*. Это означает, что сайт должен быть доступен для максимально возможного числа пользователей. Давайте рассмотрим простой пример. Вы сделали черный фон на странице и текст выводите серым цветом шрифтом размером 10 пикселов. Это может быть приемлемо для большинства пользователей, однако есть достаточно большая группа людей с ослабленным зрением, так что для них такой неконтрастный текст мелким шрифтом читать либо очень сложно, либо совершенно невозможно. Вот как раз для них можно написать альтернативную таблицу стилей с белым фоном и черным текстом, выведенным шрифтом 18 пикселов. В браузере должна быть реали-

зована возможность переключать основную таблицу стилей на альтернативную, что сделает информацию более доступной и расширит аудиторию сайта.

Как видите, проблем достаточно много Некоторые из них устраняются леио (наследование), некоторые очень неочевидно и сложно (ширина блоков), ,\ некоторые не устраняются принципиально (альтернативные таблицы стилей)

Microsoft Internet Explorer 6.x

Первая версия вышла в конце 2001 года, так что можно было бы надеяться на значительные сдвиги к лучшем) в целом и в поддержке стандартов в частности. Вопреки ожиданиям, этого не произошло. Да, браузер достаточно стабильный и быстрый, да, появились некоторые интересные функции, да. исправили приличное количество багов. Но это, фактически, и все, потом\ что поддержка новых стандартов улучшилась весьма незначительно. Однако по порядку.

Быстрота и производительность

Производительность тестировалась все в той же лаборатории CNET Labs так что результаты должны быть сопоставимыми. Что особенно интересно Удивляет тот факт, что результаты тестирования под операционной системой Windows 2000 и Windows 98 *принципиально* отличаются. Если в первом случае Netscape 6.v по производительности не уступает браузеру Internet Explorer 6.v. то во втором случае Explorer просто наголову превосходит своего конкурента. Исходя из этого, можно сделать вывод, что относительная произволительность сильно зависит от конфигурации системы.

ПроизводительностьJava

Win 98

- Explorer 5.5 - 100%.

П *Explorer 6.x~ 99%.*

O Netscape 6 I v — 44%

Как видим, никаких улучшений в данном случае нет. Что касается браузера Netscape 6.1, то он на фоне браузеров фирмы Microsoft выглядит просто ужасно.

Форматирование сложных таблиц

Win 98

3 EXplorer 5 5 — 100%.

3 Explorer 6.x - 99%.

3 Netscape 6.1 — 54%.

Win 2000

3 Explorer 6 л - 100%.

3 Netscape 6.1 -94%.

Фирма Netscape значительно улучшила механизм обработки таблиц. Помните, раньше было 22%, а теперь 54%. Однако это все равно почти в два раза медленнее.

Но в операционной системе Win 2000 браузеры практически сравнялись.

Что касается последней версии, то никаких улучшений в браузере Explorer 6.v в этом плане вновь не наблюдается.

Скорость загрузки графики и текста

Win 98

0 Explorer 5.5 -100%.

O *Explorer 6-x — /769?J*

O Netscape **6.1** — 53%.

Win 2000

O Explorer 6.x — 124%.

П Netscape 6.1 - 100%.

В прошлом это было самой слабой стороной браузера Internet Explorer. Однако, как мы видим, разработчики это учли и существенно переработан! механизм. Вследствие чего скорость возросла очень сильно. В то же время, Netscape начал сдавать позиции.

Загрузка котированных страниц

Win 98

3 Explorer 5.5 - 100%.

O Explorer 6-Y— 83%.

O *Netscape 6.1 — 34%.*

Win 2000

- Explorer 6.v — 80%.
- Netscape 6.1 - 100%.

Если в Win 98 с кэшем гораздо лучше работает браузер Explorer, то в Win 2000 лучше работает Netscape. Это особенно удивительно, потому что разработчики фирмы Microsoft просто обязаны были максимально оптимизиро-

вать браузер под свои операционные системы. Возможно, они решили сосредоточиться на оптимизации браузера под Win XP, однако все равно непонятно, по какой причине Win 98 вчистую обыгрывает Win 2000.

Вообще, если подводить итог, то можно сказать, что разработчики смогли очень сильно улучшить скорость загрузки контента. Именно это и было самой слабой стороной браузера Internet Explorer 5.x.

Интерфейс и фишки

Интерфейс рассматривать не будем, потому что изменений в нем по сравнению с пятой версией практически никаких нет. А вот некоторые интересные особенности появились.

О Специальное контекстное меню, которое всплывает, когда пользователь наводит мышку на рисунок. Меню содержит опции **Сохранить. Распечатать, Отправить по E-mail,** что достаточно удобно.

- Появилась возможность выбирать и проигрывать видео- и аудиофайлы непосредственно из браузера.

П Значительно улучшили функцию печати страниц, но все равно остались некоторые принципиальные проблемы, из-за которых страница часто просто не умещается по ширине на печатный лист.

Собственно, из новых возможностей это и есть главные, остальные незначительны. Как видите, разработчики не ставили себе цель внедрить множество фишек. Браузер Internet Explorer сам по себе продукт устоявшийся, так что ничего принципиально менять не стали.

Безопасность

Что нового? Internet Explorer 6_* поддерживает новый стандарт P3P. который позволяет управлять своими персональными данными. Следовательно, пользователь сам определяет, какую информацию может получать о нем вебсайт. Например, можно разрешать или запрещать определение вашего браузера и выбор просматриваемых страниц. Но пока этот стандарт не распространен. Дело в том, что сайт должен поддерживать его, т. е. получить лицензию, тогда схема будет работать. Если же сайт не поддерживает стандарт P3P, то Internet Explorer 6.v на некоторых уровнях безопасности просто не будет принимать определенного рода информацию с этого сайта, включая cookies. Это может помешать нормальному функционированию сайта в целом. Например, чтобы сделать заказ на сайте интернет-магазина, надо включить в браузере поддержку cookies, потому что большинство так называемых "виртуальных корзин" функционирует именно на их основе. Получается, что устанавливать в браузере высокий уровень безопасности нецелесо-

образно, потому что значительно ухудшается функциональность. Иначе говоря, механизм защиты, основанный на P3P, пока не работает.

Соответственно не работают и уровни системы безопасности. Однако вы можете устанавливать фильтр для cookies, хотя он достаточно несовершенный.

Кроме того, появился интересный и полезный фильтр, который выдает вам предупреждающее сообщение, если какой-нибудь скрипт пытается заставить Outlook отправить письмо. Это помогает избежать заражения некоторыми вирусами или, по крайней мере, обнаружить заражение на ранней стадии.

В общем, несмотря на усилия фирмы Microsoft, с безопасностью браузера по-прежнему есть большие проблемы.

Стандарты

HTML

Что касается HTML, то улучшения есть, хотя и немного.

- Улучшили функционирование элемента <OBJECT>. ОН сейчас поддерживает атрибуты ALT И BORDER.
- Добавили поддержку декларации IDOCTYPE, ЧТО само по себе очень хорошо и полезно.

О У элемента <HTML> ПОЯВИЛСЯ атрибут SCROLL, который, впрочем, не входит в спецификацию HTML 4.O.

О У элемента <A> ПОЯВИЛСЯ атрибут SHAPE, который позволяет устанавливать координаты и форму области, которая и будет являться ссылкой.

CSS

K. великому сожалению многих веб-разработчиков, Internet Explorer 6.v очень недалеко продвинулся вперед по сравнению с Internet Explorer 5.5. Вероятнее всего, это из-за упора на связку XML + XSL. Как бы то ни было, но улучшения все же есть.

- Самым главным, без всяких сомнений, является исправление блоковой модели. Так что в скором времени можно будет, особо не задумываясь, устанаативать отступы, рамки и задавать ширину блоков. Правда, есть одна особенность. Дело в том, что в браузере можно переключаться между блоковыми моделями с помощью инструкции IDOCTYPE. *Как это делается,* мы уже с вами обсуждали, а вот *для чего это сделано,* не совсем понятно. Представители фирмы Microsoft утверждают, что это сделано для обратной совместимости с браузером Internet Explorer 5x Однако давайте рассмотрим вопрос подробнее.

  В настоящее время при создании блоков с отступами и рамками обычно поступают следующим образом. Для Internet Explorer 5_v устанавливают

ширину до специальной фишки voice-family, после которой он не воспринимает объявления, а для всех нормальных браузеров >стапавливаю1 ширину согласно спецификации CSS-1 после нее. Но Internet Explorer 6 л корректно обрабатывает свойство voice-family (он его игнорирует), так что для нею естественным образом тоже будет установыена ширина согласно спецификации CSS-1. Получается, что включение неправильной блоковой модели лишено смысла. Если включить режим обратной совместимости, то страница будет отображаться неправильно, несмотря на ухищрения для браузера Internet Explorer 5_v, но если включить режим совместимости со стандартом, то все будет нормально. Обратная совместимость была бы в том случае, если бы *все* полыовалнсь браузерами фирмы Microsoft. Однако это не так.

Кроме того, нужно внимательно следить и всегда в начале страницы ставить правильную инструкцию IDOCTYEE. которая будет переключать браузер в совместимый со стандартами режим.

- Наконец-то браузер Internet Explorer *полностью* поддерживает стандар] CSS-1. Разработчики удосужились исправить некоторое количество багов и добавить два новых свойства. Сначала о багах'
  - в предыдущих версиях на элемент <IMG> нельзя было устанавливать отступы с помощью свойства padding, в шестой версии это исправили;
  - исправили механизм наследования для таблиц, что достаточно важно;
  - стало возможным устанавливать ширину элемента <BODY>. ЭТО достаточно удобно для сайтов с простейшей разметкой, потому что не надо делать лишних блоков;
  - при установке cable-layout: fixed вычислялась ширина ячеек таблицы, но не вычислялась высота Это тоже исправлено.

  Исправили также обработку неизвестных объявлений и прочих ошибок и таблицах стилей, и еще некоторые мелкие баги.
- Что касается новых свойств и значений, то их больше:
  - добавили свойство min-height, но только на элемента гн> и <TR>, так что применять его можно для таблиц с tapie-iayout: fixed, но нельзя для всех остальных блоков, что плохо;
  - добавили свойство word-spacing;
  - добавили корректную поддержку свойства auto для свойств widtr. и margin, что позволяет, в частности, центрировать блоки и многое другое. Это действительно очень полезное дополнение;
  - свойство display теперь может принимать значение list-item, что позволяет, скажем, нумеровать заголовки;
  - добавили значение small-caps ДЛЯ СВОЙСТВа font-va:...

- Из спецификации CSS-2 реализовано очень интересное свойство, которое позволяет разработчикам изготавливать и подключать к веб-сайту свои собственные курсоры. Это свойство cursor. Оно было реализовано в пятой версии браузера, а в шестой стало функциональнее. Для подключения своего курсора надо задать URL его фафического изображения:

```
A (
  cursor: url(newcursor.cur)}
```

  Подробнее об этом свойстве мы поговорим в *главе П.*

Подведем итог. В браузере Internet Explorer *6.x* устранены две проблемы: исправлена блоковая модель и наследование в таблицах. Однако ничего не сделали с плавающей моделью, не ввели альтернативные таблицы стилей, не исправили строчные блоки. Учитывая долгий срок между выходами браузеров Internet Explorer 5.x и 6.Y, это достаточно странно.

Кроме того, разработчики убрали поддержку плагинов, которая осталась в браузере Netscape. Наверное, это сделали для продвижения технологии ActiveX, но шаг достаточно спорный и, на мой взгляд, неразумный. Из базового инсталляционного пакета убрали поддержку Virtual Java Machine, но ее отдельно можно взять с сайта Microsoft, что тоже нехорошо.

Если делать выводы, то можно сказать, что браузер Internet Explorer 6..r разочаровывает. Это касается поддержки стандартов и безопасности, что является главным. Есть увеличение производительности, но этот отрадный факт не очень-то сглаживает обшее впечатление. Професс недостаточно большой, и это. пожалуй, главный вывод. Возможно, это является основной причиной, по которой пользователи не торопятся переходить с пятой версии браузера на шестую.

Netscape 6.x

Быстрота и производительность

Мы уже сравнивали скорость работы браузера Netscape 6..v с браузером Internet Explorer 6.x Она сильно отличается для различных операционных систем, но одно можно сказать с полной уверенностью: стартует Netscape 6.Y гораздо медленнее, чем Internet Explorer. Этому есть тривиальное объяснение. Просто Explorer резидентно находится в оперативной памяти на всех машинах с операционной системой Windows, а Netscape — нет. С этим можно бороться, и есть возможность сделать резидентным и Netscape, но это требует значительного объема оперативной памяти.

Что касается быстродействия, то оно улучшилось по сравнению с четвертой версией, но все же браузер остался достаточно медленным.

Интерфейс и фишки

Шестая версия имеет очень хороший интерфейс. Он наконец-то стал не хуже интерфейса браузера Internet Explorer 6.x Что есть интересного, по сравнению с браузером фирмы Microsoft?

П Очень хороший менеджер cookies. Действительно удобный и полезный Вы можете добавлять сайты в стоп-лист, т. е. с этих сайтов cookies не будут приниматься.

П Есть менеджер паролей. Тоже очень полезная утилита.

П Поддерживаются так называемые темы (Themes). Теперь вы можете менять внешний вид браузера, просто меняя темы. Это создает эффек: большęго контроля над браузером, а, кроме того, позволяет получать эстетическое наслаждение при просмотре веб-странии.

П Возможность блокировать всплывающие окна без отключения JavaScript Вы, наверное, сталкивались с проблемой надоедливых всплывающих окон. Так вот, в браузере Netscape 6.x это можно легко устранить.

О Автоматическая проверка обновлений на сайтах. Очень полезная вещь.

Если рассматривать слабости в реализации интерфейса, то в Netscape блс нет:

- полноэкранного режима, что достаточно плохо для обладателей небольших мониторов;
- возможности конфигурации панели инструментов, что некритично, но несколько неудобно;
- поддержки иконок (favicon.ico). что ухудшает быстрый поиск нужной ссылки в Избранном. Надо сказать, что поддержка иконок появилась и браузере Mozilla 0.9.7, так что она будет и в новых версиях браузера Netscape.

Согласитесь, что уникальных особенностей больше именно у Netscape 6.x и они интереснее сами по себе. Так что с точки зрения удобства пользования Netscape 6.x в некотором отношении лучше.

Безопасность

Netscape 6.x на сегодняшний день является одним из самых безопасных браузеров. У него есть множество настроек, типа менеджера cookies и шифрования паролей. Кроме того, пока неизвестно ни одного вируса и ни одной дыры в безопасности, тогда как браузер Internet Explorer 6.x имеет как минимум 5 вирусов и 10 дыр в защите. Конечно, это может быть связано с тем, что Netscape 6.x не очень популярный браузер, так что и дыры в нем искать нет особого смысла.

Стандарты

HTML

В браузере Netscape 6.v стандарт HTML 4.0 поддерживается очень хорошо. По сравнению с четвертой версией изменения огромные, однако достаточно перечислить основные.

- Реализовали поддержку элемента <IFRAME>.

П Основательно поработали над элементом <OBJECT>. ЕСЛИ раньше можно было вставлять только Java-апплеты, то теперь можно вставлять изображения, карты и делать вложенные объекты. Конечно, кое-какие мелкие недостатки остались.

П Реализовали поддержку элемента <q_>, теперь текст между тегами <QX/Q> заключается в двойные кавычки, но пока нет возможности изменять вид кавычек. Кроме того, вложенные элементы <Q> ДОЛЖНЫ заключать текст в одинарные кавычки, однако этого не происходит, и кавычки остаются двойными.

З Поддерживаются элементы <ABBR> И <ACRONYM>. А также добавили полную поддержку элементов <INS> И <:DEL> (ОНИ ПОЗВОЛЯЮТ помечать исправления и добавления в документе).

О Значительно улучшилась реализация форм. Реализована поддержка элементов <BUTTON>, <FIELDSET>, <LEGEND> И МНОГИХ атрибутов ДругИХ ЭЛементов.

О Полностью переработана поддержка таблиц. Начиная с шестой версии, стали поддерживаться элементы <THEAD>, <TFOOT>, <TBODY>, <COLGROUP> И <COL>.

Вообще-то на данном этапе браузеры уже достаточно одинаково поддерживают стандарт HTML, чтобы особо не задумываться о совместимости при верстке страниц. Так что в этом плане Netscape 6.x мало чем отличается от Internet Explorer *6.x,* но если систематизировать все мелочи, то можно сделать однозначный вывод: стандарт HTML 4.0 лучше поддерживается браузером Netscape *6.x.*

XML

Netscape *6.x* поддерживает стандарт XML и связку XML + CSS. Не поддерживается XSL и в ближайшем будущем его поддержка не планируется. Ввиду того, что реализация CSS очень хорошая, можно делать XML-документы и форматировать их с помощью каскадных таблиц стилей, но особого смысла в этом нет.

CSS

Спецификация CSS поддерживается браузером прекрасно. Естественно, есть некоторые баги, но н\ не так уж много.

П Имеются достаточно серьезные проблемы с плавающей моделью, такие как неправильное выравнивание плавающих блоков и некорректное обтекание текстом.

О Серьезные проблемы с подвидом абсолютного позиционирования — фиксированным позиционированием.

- Неправильное вычисление высоты строки
- Имеются определенные проблемы с текстом, а именно некорректно реализованы свойства vertical-align И word-spacing.

Остальные баги в большинстве своем мелкие и не заслуживают особого внимания. Если сконцентрироваться на поддержке CSS-2, то Netscape 6_v превосходит Internet Explorer *6.x,* но немного уступает Opera 6jr

Opera 6.x

Быстрота и производительность

Традиционно Opera считается самым быарым браузером. Именно на быстродействие делается упор во всех рекламных кампаниях, а главный рекламный слоган звучит так: "The fastest browser on earth'", что переводится как "Самый быстрый браузер на Земле'" Кроме того. Opera — самый нетребовательный к ресурсам браузер. В табл. *\0Л* дана сравнительная оценка некоторых системных требований для разных браузеров

Таблица 10.1. *Системные требования для основных браузеров*

Параметры	Explorer 6.x	Netscape 6.x	Opera 6.x
Тип процессора	Pentium	Pentium	486
Минимальная частота процессора	**233 МГц**	**233 МГц**	133 МГц
Рекомендуемая частота процессора	400+ МГц	**700+ МГц**	233+ МГц
Объем оперативной памяти	**64-96 Мбайт**	126-256 Мбайт	48-64 Мбайт
Объем дискового пространства	75 Мбайт	**40 Мбайт**	23 Мбайт

Как видите, Opera бд: может совершенно нормально работать на машине с процессором частотой 133 МГц и объемом оперативной памяти 64 Мбайт, тогда как для остальных браузеров нужны машины посерьезней. Самым требовательным к ресурсам является браузер Netscape 6 .v, так что на слабых

машинах даже не пытайтесь им пользоваться, потому что производительность будет очень низкая.

Интерфейс и фишки

Если Internet Explorer 6.Y и Netscape 6_x имеют сходный интерфейс, то Opera 6JC достаточно сильно от них отличается. Например, вы можете выбрать, как будут открываться окна. Можно для каждого сайта открывать отдельное окно, как это делается в Internet Explorer и Netscape. А можно открыть одно главное окно браузера, внутри которого будут открываться все остальные окна. Как удобнее конкретно для вас, вы можете решить сами.

Список уникальных фишек достаточно внушительный.

- Присутствует интересная функция Zoom, которая позволяет увеличивать как шрифт, так и рисунки пропорционально. Это делает страницы более доступными для просмотра пользователями со слабым зрением.
- Можно сохранять окна перед закрытием браузера. Это позволяет легко возвращаться к прерванной по какой-либо причине работе. Представьте такую ситуацию. Вы пишете статью и с огромным трудом нашли пять сайтов сопряженной тематики, а в это время произошел сбой операционной системы (что бывает, хоть и нечасто). Естественно, система перегружается и, если вы пользовались браузером Internet Explorer или Netscape, вся информация теряется и вам придется искать эти сайты вновь. Если же вы пользовались Opera, то после загрузки браузера откроется пять окон и волшебным образом в них загрузятся найденные вами сайты. Так что фишка очень полезная.

3 Поддерживаются так называемые Skins (в Netscape нечто похожее называется Themes). Это позволяет настраивать и изменять внешний вид браузера.

3 Есть полноэкранный режим работы.

<*3* Есть очень приятная утилита поиска, которая позволяет легко производить поиск по словам и фразам на различных поисковых серверах, а также сохранять результаты поиска в закладках.

П Отключение загрузки изображений одной кнопкой, что чрезвычайно удобно и полезно в условиях плохого соединения и медленного канала. Ничего подобного больше нет нигде.

- Отключение каскадных таблиц стилей одной кнопкой, что бывает хорошо для сайтов с неприятными цветами и мелкими шрифтами.
- Предварительный просмотр страниц перед печатью.

Можно сказать, что Opera *Ь.х* вобрала в себя практически все свойства браузеров Internet Explorer 6.v и Netscape *6.х.* Из недостатков можно выделить следующие

О Плохая реализация автозаполнения различного рода форм на сайтах, тогда как в остальных браузерах она реализована очень хорошо

О Не поддерживаются иконки (favicon.ico), что ухудшает быстрый поиск нужных ссылок в Избранном.

О Нет редактора cookies.

Как видите, интересных и необычных свойств гораздо больше, чем упущений. Так что с точки зрения удобства пользования браузер Opera *6.х* должен быть лучшим. Однако проблема в том, что интерфейс достаточно сильно отличается от интерфейса остальных браузеров, к которым пользоватечи привыкли Привычка вещь совершенно уникальная и субъективная, так чте даже очевидные выгоды не всегда могут ее изменить Вообще, распространению браузера Opera *6.х* мешают три вещи.

• Он является условно бесплатным. В этом конкретном случае вы будете вынуждены любоваться баннером вохпе навигационной панели или же заплатить 30 $ фирме Opera Software, после чего баннер исчезает.

О Достаточно непривычный интерфейс. Большинство пользователей начинают с браузера Internet Explorer, потому что он устанавливается сраз\ с операционной системой Windows. Кстати, в последней версии интерфейс претерпел значительные изменения именно в сторону сближения с Internet Exploiei, так что популярность Opera должна немного увеличиться

П Слабая маркетинговая поддержка браузера. Реклама в конечном итоге является главным фактором, определяющим популярность любого продукта (если качество не кошмарное).

Безопасность

Некоторые специалисты считают, что браузер Opera является на сегодняшний день самым безопасным. Он имеет только две дыры в системе безопасности, которые, впрочем, исправлены в шестой версии А зашита информации в браузере достаточно сильная

О Возможно 128-битное шифрование отправляемой на сервер информации

О Возможно запрещение принятия cookies, но таких гибких настроек, как в Netscape 6 \\ нет.

Стандарты

С точки зрения стандартов, браузер Opera 6.Y весьма противоречивый. Он превосходно поддерживает CSS-1 и лучше всех CSS-2, но хуже всех браузе-

ров поддерживает DOM и JavaScript. Кроме того, есть некоторые отличия в поддержке HTML 4.O. что может приводить к неправильному отображению веб-страниц.

HTML

Спецификация HTML 4.0 поддерживается не полностью. Если уточнять, то хуже, чем браузерами Internet Explorer 6.x и Netscape 6.x. Однако различия не критичные, так что подавляющее большинство страниц должны отображаться корректно.

П Если рассматривать поддержку таблиц, то Opera не поддерживает элементы <*CO'~>,* <COLGROUP>. <THEAD>, <TFOOT> И <TBCDY>. Не ТО чтобы ЭТИ элементы были очень важны, но определенный выигрыш на практике они обеспечивают. Элемент <COL> удобно использовать для задания стилей на определенный столбец таблицы, однако в Opera 6 это не работает.

- Если говорить о формах, то в Opera их поддержка тоже развита недостаточно. Не поддерживаются элементы <BUTTON>, <OPTGROUP>, <LABEL>.

O Не поддерживается атрибут FRAMEBORDER.

Остальные отличия совсем мелкие, причем большинством атрибутов, которые не поддерживаются браузером Opera 6.x, HTML-верстальшики не пользуются.

XML

Opera 6.x поддерживает стандарт XML, пространство имен XML и связку XML + CSS. Так что вообше-то можно создавать XML-документы и подключать к ним таблицу стилей таким образом:

```
<?xml version'-"!..0"?>
<?)cml-stylesbeet href="xmlstyle,css" type="text/css"?>
```

Но, как вы уже знаете. Opera 6.x не поддерживает XSL, а в браузере Internet Explorer слабо развита связка XML + CSS, так что на практике создавать XML-документы пока очень тяжело. Разве что для сетей интранет.

CSS

С каскадными таблицами стилей ситуация диаметрально противоположная. Браузер Opera 6.x на сегодняшний день лучший браузер по поддержке спецификации CSS-2, и он полностью поддерживает спецификацию CSS-1. Естественно, есть отличия от других браузеров. Если говорить о CSS-2, то проще перечислить, что не поддерживается, чем то, что поддерживается. Итак. Opera 6.x не поддерживает:

- свойства:
  - clip -— позволяет устанавливать область видимости при значении СВОЙСТВа overflow, ОТЛИЧНОГО ОТ visible;

- cursor — позволяет задавать вид курсора или же задавать собственные курсоры;
- direction — позволяет задавать расположение блоков слева-направо или справа-налево;
- font-size-adjust — позволяет подбирать шрифт с заданным соотношением высоты шрифта к высоте буквы "x". Можно использовать для более точного подбора шрифта в том случае, если на машине пользователя нужного шрифта не оказалось;
- font-stretch — позволяет задавать тип шрифта, такой как Condensed или Expanded;
- text-shadow — позволяет применять к тексту различного рода эффекты, такие как тени;

П объявления:

- `visibility: collapse;`
- `overflow: scroll;`
- `overflow: auto.`

- `правило dfont-face;`

О селекторы, псевдоклассы:

- :first-child — применяется к первому потомку элемента;
- : focus — стили будут применяться, если элемент получил фокус (например, стало активным поле формы);
- : lang — позволяет применять те или иные стили в зависимости оі языка. Это можно использовать, например, для установки разных кавычек в русскоязычном и англоязычном тексте.

Обратите внимание, что Opera 6.v поддерживает почти все виды селекторов спецификации CSS-2, за исключением трех псевдоклассов, тогда как практически все селекторы из спецификации CSS-2 не поддерживаются браузером Internet Exlplorer 6JT, так что ими можно активно пользоваться для сокрытия объявления от этого браузера, т. е. для создания кросс-браузерного кода.

Но Opera 6.v имеет очень большой недостаток. Он плохо поддерживает стили для элементов форм, таких как <INPUT>. <TEXTAREA> И др. Недостаток действительно серьезный, потому что разработчики уже привыкли устанавливать для элементов форм однопикселные рамочки.

Если сравнивать поддержку CSS браузера Opera 6.x с остальными браузерами, то различия очень существенные. В конце книги в *пршюженшг 4* есть сводная таблица поддержки CSS различными браузерами, которой весьма полезно пользоваться при верстке.

Кросс-браузерный CSS

Для начала разберемся, что обозначает понятие "кросс-браузерный" в целом. Вы. наверное, слышали о термине кросс-платформенный. Это одна из характеристик некоторых языков программирования и обозначает она то, что программа на данном языке будет работать одинаково на любой платформе, будь то Windows, Macintosh, Unix или Sun Solaris. Подобным свойством обладает язык Java.

По аналогии термин *кросс-браузерный* обозначает, что данный кол будет работать корректно во всех браузерах. Причем заметьте, что здесь отсутствует слово *одинаково.* Дело в том, что многие браузеры отображают кол неодинаково, но корректно, т. е. согласно спецификации. Именно по этой причине слово *одинаково* для браузеров неприменимо, зато очень подходит слово *корректно.* Если же рассматривать каскадные таблицы стилей, то термин *кросс-браузерный CSS* обозначает, что стили будут корректно отображаться разными браузерами.

Итак, для чего же нам кросс-браузерный CSS? Ответ достаточно тривиален. Если бы все браузеры *полностью и одинаково* поддерживали спецификации CSS-1 и CSS-2. то он бы нам не понадобился, однако это совершенно идеалистическая картина. Таким образом, можно выделить две причины, которые оправдывают применение кросс-браузерного CSS.

3 Неодинаковая поддержка браузерами селекторов, свойств, значений и прочих понятий из спецификаций CSS-1 и CSS-2.

"3 Неодинаковый набор багов браузеров в реализации поддержки одинаковых селекторов, свойств и прочих элементов из спецификаций CSS-1 и CSS-2

Главная цель HTML-версталыяика — *сделать так, чтобы сайт одинаково смотрелся в максимальном количестве браузеров.* Задача сложная и нетривиальная. Раньше проблемы возникали из-за неодинаковой поддержки HTML, однако на данный момент ситуация улучшилась. Во-первых, нашли очень много способов устранения различий, во-вторых, поддержка HTML стала практически одинаковой, начиная с шестых версий браузеров. Сейчас проблемы возникают из-за CSS, потому что стандарт достаточно молод (по меркам производителей браузеров), так что реализация его поддержки весьма неодинакова и сильно варьируется от браузера к браузеру.

Если отталкиваться от главной цели HTML-верстальшнка. то возможно несколько вариантов действий.

- Отказаться от использования каскадных таблиц стилей вообще. Вы, конечно, понимаете, что в этом случае работы у HTML-программиста прибавится и немало, потому что, по меньшей мере, придется повсеместно

вставлять теги <FONT>. Кроме тою, невозможно будет делать некоторые эффекты, наподобие hover, которые осуществляются только с помощью каскадных таблиц стилей.

- I Использовать CSS минимально, т. е. задавать цвета и параметры шрифта на странице, потому что эти свойства реализованы корректно во всех браузерах. К сожалению, именно так поступает большинство HTML-программистов, потому что для этого не надо детально знать каскадные таблицы стилей и не надо беспокоиться о совместимости. Такой подход может объясняться либо тотальным недостатком времени на изучение новой технологии, либо строгим указанием равняться на браузеры третьих и четвертых версий, либо природной ленью.

П Отбросить браузеры всех версий ниже пятой. Тогда можно не опасаться использовать практически весь набор из CSS-1, потому что в пятых версиях реализация данного стандарта достаточно хорошая. Но в таком подходе все же есть слабые места. Например, вы никогда не сможете таким образом сверстать сайт на основе блоков, потому что блоковая модель реализована по-разному в разных браузерах. Так что одно из главных преимуществ CSS остается за бортом. К слову сказать, чтобы сверстан сайт на основе блоков, все равно придется отказаться от поддержки старых браузеров.

О Можно детально изучить нюансы поддержки каскадных таблиц стиле!: браузерами и писать кросс-браузерный код на основе различных хитростей и уловок. Этот способ будет работать во многих случаях, однако ь последних версиях браузеров из-за корректности реализации поддержкг CSS таких уловок осталось очень мало и пользоваться ими сложно. Кроме того, писать такой код при верстке больших сайтов блоками неимоверно сложно и времени отнимает гораздо больше, чем кодирование і помощью таблиц.

О Можно создать несколько таблиц стилей, каждая из которых будет адаптирована строго под определенный браузер. Затем с помощью JavaScript надо детектировать браузер и его версию, и, в зависимости от типа браузера, подключать к странице нужный файл с расширением ess. Этот способ однозначно лучший для сложных таблиц стилей, потому что не надо думать о методах сокрытия тех или иных частей кода от того или иного браузера. Надо просто создавать для каждого браузера свою таблицу стилей, которая учитывает все его особенности.

Мы по определению себя ограничивать не хотим, поэтому без кросс-браузерного кода нам не обойтись. Два последних способа в принципе хорошие и подходящие, но когда использовать тот или иной способ, надо решать в каждом конкретном случае отдельно.

В первую очерель мы займемся рассмотрением методов сокрытия кусков кода от различных браузеров. Принципиально можно разделить все эти методы на несколько.

П Баги браузера, которые приводят к тому, что он не видит определенный кусок кода.

• Пробелы в реализации поддержки CSS. Браузер может не поддерживать некоторые типы селекторов или способы подключения таблиц стилей к страницам.

Можно рассматривать методы сокрытия стилей от браузеров, а можно брать отдельный браузер и рассматривать все способы, которые могут скрыть код именно от него. Оба подхода имеют как положительные, так и отрицательные стороны Для понимания сути процесса лучше применить первый подход, потому что он логичнее, а для наглядности лучше второй подход, потому что он разбит по браузерам. Я вначале рассмотрю именно методы, т. с. воспользуюсь первым подходом, а затем сделаю общую таблицу, в которой будет показана вся картина целиком. Начнем со способов подключения CSS к странице.

Подключение таблицы стилей

Может осуществляться тремя способами:

• с помошью элемента <LINK>;

П с помощью инструкции ^import;

О с помощью элемента <STYLE> (естественно, таким образом можно подключать только внутренние таблицы стилей).

Начнем по порядку.

Атрибут *MEDIA*

Элемент <LINK> имеет один-единственный атрибут, который нам может помочь, это атрибут MEDIA. Дело в том, что данный атрибут плохо поддерживается браузером Netscape *4.x* (он понимает только значение ME01л="зсгсеп"), **так** что скрывать таблицу стилей от этого браузера можно с помощью такой строки:

<LINK HREF="none_nn4.css" TYPE="text/css" REl>"stylesheet" **MEDIA="all">**

Если вы хотите сделать таблицы стилей для разных устройств, например, для экрана монитора и для принтера, то такой способ сокрытия таблиц стилей от Netscape 4.x уже не пройдет. Для разграничения стилей на экран монитора и принтер надо подключать их так:

<LINK HREF="screen.css" TYPE="text/css" REL-"stylesheet" **MEDIA=e"screen">**
<LINK HREF="print.css" TYPE="text/css" REL="stylesheet" **MEDIAE"print'•>**

Однако Netscape 4 л прекрасно поничиим *только WLDIA--"bete*:.",* так чп надо нсмною подумать и найти выход. Браузер Netscape *4.x* не понимай если значения в атрибуте перечисляются через запятую. Например, если подключить таблицу стилей таким образом:

```
<LIKK HREF="screen_none_nn4.css" TYPE="text/ess" REL="styiesheet"
MEDIA="screen, projaction">
```

то она по-прежнему будет специфичной для экрана монитора, но Netscape 4 * ее уже не поймет. Так что разделение стилей для браузера Netscape 4.v и OL-тальныч браузеров будет реализовыва1ься следующим образом:

```
<LINK HREF="nn4.css" ?YPE="text/css" REL="stylesheet">
<LINK HREF»"screen.css" TYPE="text/css" REL="styleshee*:" MEDIA="screen,
projection">
<LINK HREF="print.css" TYPE="text/css" REI>"stylesheet'' MEDlA="print">
```

Здесь надо рассказать об одном из главных правил использования подобною кросс-браузерного кода. Например, вы хотите, чтобы для всех браузеров, кроме Netscape 4..г, ширина полей всего документа была 0. а хи Netscape *4.x* — 15% от ширины окна браузера. Для чего это может вообик понадобиться? Допустим, вы не хотите поддерживать браузер Netscape 4.л. но надо, чтобы контент был доступен всем, так что для этого браузера можно просто сделать большие поля, пусть текст будет выводиться шрифтом ш умолчанию, так что читать можно будет, хотя красоту дизайна посетитель сайта и не увидит. Тогда вы пишите для Netscape *4.x* таблицу стилей

```
BODY (
  margin-left:  15%;
  margin-right:  15%}
```

которую сохраните в отдельный файл nn4.css. Но все остальные браузеры тоже поймут эти объяыенмя, так что есть два правила, которые на.и неукоснительно выполнять:

- *все* объявления, указанные в таблице стилей для старых браузеров, должны переписываться в таблице стилей для новых браузеров;
- П таблица стилей хтя новых браузеров должна подключаться *последней.* чтобы объявления переписались.

Если вернуться к примеру, то для новых браузеров вы напишите стиль

```
BODY {
  margin: Opx}
```

и поместите его в файл ail.ess. А подключаться эти файлы будут *только и* таком порядке:

```
<LINK HREF="nn4.ess" TYPE="text/css" P£L="stylesheet">
<LINK HREF="all.css" ?YPE="text/css" REL="stylesheet" MEDIA="all">
```

Есть еше одна очень интересная возможность скрывать стили от браузера Opera всех версий. Ее обнаружил *Хуан Позо*. Для этого надо вместо MEDiA="screer.[M] написать MEDIA-"SC*#82;eeп", т. с. заменить букву "г" на ее код &#82. Что интересно, все остальные браузеры применяют стили, а вот Opera нет. Так что для сокрытия стилей от браузеров Opera 4+ надо написать такой код:

```
<LINK HREF="none_op.css" TYPE="text/css" REL="stylesheet"
MEDIA="scfi#82/een">
```

Инструкция *©import*

Эта инструкция нам интересна тем, что ее не понимает не только браузер Netscape 4..x, но и браузер Internet Explorer 4..v и ниже. Так что можно скрывать таблицу стилей от браузеров четвертого поколения. Делается это так. Вы пишете две таблицы: одну для четвертых версий, а вторую для пятых. Например, первую назовем ver4.css, а вторую — ver5.css. Тогда подключение стилей будет выглядеть так:

```
<LINK HREF="ver4.css" TYPE="text/css" REL="stylesheet">
<STYLE TY?E="text/css">
  @unporturl("ver5.css")
</STYLE>
```

Если вам надо скрыть от браузера Netscape 4..v внутреннюю таблицу стилей, то можно опять же воспользоваться атрибутом MEDIA ИЛИ же инструкцией Omedia. В следующем примере все объявления будут скрыты от Netscape 4.x

```
<STYLE TYPE="text/cs2" MEDIA="all">
P {
  color: #000}
</STVLE>
<STYLETYPE="text/css">
3media all {
  P {
    color: #000}}
</STYLE>
```

Больше нет способов скрыть таблицу стилей при ее подключении.

Селекторы

Выбор элемента, к которому будет применен данный стиль, осуществляется с помощью селектора. Логично предположить, что баги в ре&тизашш селекторов могут помочь скрыть *праеи.io или же отдельное объявление* от какого-либо браузера. И это на самом деле так.

Вообще, достаточно сложно классифицировать те или иные методы сокрытия таблиц стилей с помощью селекторов. Так что будет простое перечисление методов.

Селекторы по атрибутам

Эти виды селекторов появились в спецификации CSS-2. Но некоторые из них уже поддерживаются браузерами Opera 5+ и Netscape 6.x (еще их поддерживает достаточно популярный браузер под Linux, который называется Konqueror) На данный момент поддерживается три вида селекторов по атрибутам.

- **[attr]**

  Стили применяются к элементу, в котором имеется указанный атрибут Например, если в коде написать

  <P CLASS="first">AC3au текста</P>

  то сделать текст красным внутри этого элемента можно таким образом:

```
P[CLASS] {
  color: #F00}
```

  Но заметьте, что если у нас в коде будет присутствовать элемент <p> с другим классом, то он все равно станет красным (если, конечно, на данный класс уже не указан в стиле другой цвет). Например, в таком коде

```
P[CLASSj {
  color: #F00}
. . .
<P CLASS="first">A63au текста</P>
<P CLASS="second">AC3au текста</P>
```

  все абзацы будут красными. Но если задать цвет для класса явно

```
PtCLASS] {
  color: #F00}
P.second {
  color: #000}
. . .
<P CLASS="first">A63au текста</P>
<P CLASS«"second">AC3au текста</P>
```

  то второй абзац будет черным. Этот селектор корректно поддерживается браузерами Opera 5+ и Netscape 6.x.

- **[attr=val]**

  Этот селектор существует для более точной выборки. В этом случае стили применяются к элемент)', который имеет атрибут с определенным значением.

В нашем примере это делается так:

```
PtCIASS=first] {
  color: #F00)
. ' .
<P CLASS="first">A03au текста</P>
```

Браузер Netscape 6 корректно поддерживает этот селектор, а вот у браузера Opera 5+ есть некоторые проблемы. Дело в том, что он понимает этот селектор, если атрибутом является, например, CLASs="first'\ но не понимает, если атрибутом является ALiGN«"nght". Как видим, не все атрибуты могут использоваться для такого селектора. Какие именно, надо устанавливать опытным путем.

O [attr~=valj

В этом случае стили применяются к элементу, который имеет атрибут с некоторым набором значений, разделенных пробелами, и одно из значений, указанное в селекторе, имеется среди значений атрибута. Для чего это вообще надо? Вот возможный способ применения. Например, нам надо к одному из абзацев текста применить такие стили, чтобы текст выводился полужирным красным шрифтом. А к другому, чтобы текст выводился полужирным синим шрифтом. Конечно, можно просто написать такие классы:

```
P.first {
  color: #F00;
  font-weight: bold}
P.second {
  color: #00F;
  font-weight: bold}
. . .
<P  CLASS=*"first">KpacHbiu  абзац  текста</P>
<P  CLASS="second">Cnwoi  абзац  текста</P>
```

Получается, что у нас есть несколько разных классов, где применяется полужирный шрифт, но зато там разные цвета.

А можно поступить следующим образом: сделать отдельный класс для полужирного шрифта и совмещать затем его с другими классами, которые имеют другой цвет. Тогда таблица стилей для двух абзацев разного цвета будет выглядеть так:

```
.ыа {
  font-weight: bold)
P.first {
```

```
    color: #F00}
  P.second (
    color: #00F)
```

А в коде мы укажем классы через пробел:

```
<P CLASS="first ЫсГЖрасный абзац текста</P>
<P ciASS="second Ыс1">Синий абзац текста</P>
```

Произойдет как бы микширование классов. Если какое-то объявление часто кочует из класса в класс (как font-weight: bold в нашем случае), то создание для него отдельного класса с последующим микшированием сократит код. Кстати, такое микширование корректно поддерживается шестыми версиями всех браузеров. А селектор [attr—val] понимаю1 браузеры Netscape 6 и Opera 5+.

Итак, чтобы скрыть стили от браузеров Netscape 4.Y и Internet Explorer всс\ версий, надо к элементу обратиться через атрибут. Все нижеприведенные примеры скрывают стили от Netscape 4..г и Internet Explorer:

```
TDJclass] {
  border: lpx solid #0001
TD[class=-brd] {
  border: lpx solid #000}
TD[class~=brd] i
  border: lpx solid #000)
. . .
<TABLE>
  <TR>
    <TD CLASS="brd">H4e&Ka с рамкой</T0>
  </TR>
</TABLE>
```

В указанных выше браузерах таблица будет *без рамки,* а вот в Opera 5- и Netscape *6.х рамка будет.*

Селектор наследника

В CSS-2 есть достаточно удобный селектор, который позволяет применятL стили к потомкам элемента. Например, у нас в коде встречаются абзацы текста, заключенные в теги <px/p>. Встречаются они непосредственно внутри элемента <BODY>, а также вложенные в отдельные блоки, образованные элементами <DIV> (допустим, это может быть блок новостей).

```
<BGDY>
  <P>Текст на сайте</P>
  <DIV id="news">
```

```
    <P>Текст новости</P>
  </DiV>
</BODY>
```

Нам надо, чтобы внутри <BGDY> текст был черного цвета, а внутри блоков новостей — серого. Можно написать отдельный класс, но можно воспользоваться селектором наследника. Наша задача решается такой таблицей стилей:

```
BODY>P {
  color: #000}
DIV>P {
  Color: #CCC}
```

Но данные стили не увидят браузеры Netscape 4.v и Internet Explorer 4+, так что селектором наследника можно с успехом пользоваться для сокрытия стилей от этих браузеров (что, собственно, и делается в методе Целика).

Кроме селекторов, есть еше и баги браузеров, с помощью которых можно писать кросс-браузерный код.

Баги

На самом деле в пятых и шестых версиях браузеров маю багов, которые позволяют скрывать стили. Самым известным и наиболее часто применяемым является метод сокрытия объяатений от браузера Internet Explorer 5~v, который придумал *Тантек Целик*.

voice-family

Мы уже рассматривали этот способ в *главе* <?, так что я здесь лишь кратко повторю его. Например, мы имеем таблицу стилей:

```
#box {
  border: 20px solid #000;
  padding: 40px;
  width: 420px;
  voice-family: "\"}\"";
  voice-family: inherit;
  widtn: 3COpx}
```

В начале описания стилей данного блока мы задаем все объявления, которые будут общими для всех браузеров. Тогда все браузеры сначала установят значение width для блока box в 420 пикселов, а также указанные отступы и рамки. Потом следует конструкция

```
voice-family: "\"}V";
```

Это свойство из звуковых таблиц стилей стандарта CSS-2, так что браузеры его вообще не должны воспринимать, т. е. просто игнорировать и обрабатывать объяаения дальше Однако браузер Internet Explorer 5.x некорректно обрабатывает эту конструкцию. Он считает, что на этом месте блок объявлений для селектора «box заканчивается. Этот браузер по сути дела будет "видеть" вот такую таблицу стилей:

```
#box {
  border: 20px solid #000;
  padding: 40px;
  width: 420px}
```

А все остальные браузеры продолжат считывать объявления и перепишут первоначальное значение ширины на новое значение, которое равно 300 пикселам. Так что для всех остальных браузеров таблица стилей будет выглядеть следующим образом:

```
#box {
  border: 20px solid #000;
  padding: 40px;
  width: 300px)
```

Этот метод широко применяется для устранения проблем с блоковой моделью в браузере Internet Explorer 5.x Заметьте, что Internet Explorer 6.x такого бага не имеет. Если он находится в режиме, совместимом со стандартами, та все будет хорошо, однако если он находится в режиме обратной совместимости, то блоковая модель будет неправильная и этот метод не обеспечш решения проблемы. Так что надо следить за тем, чтобы в документах всегда правильно указывался тип с помощью IDOCTYPE.

Комментарии

В некоторых браузерах некорректно реализованы комментарии к стилям Конкретнее, страдают этим все браузеры фирмы Microsoft с третьей до пятой версии. Этот баг исправили только в браузере **Internet Explorer 5.5,** так что можно скрывать правило от браузеров Internet Explorer 4.v и Internet Explorer 5.0 таким образом:

```
РЛ */[
  line-height: 1.5)
```

Если вставить комментарий непосредственно после селектора, то браузеры Internet Explorer 4.x—5.x ЭТО правило просто проигнорируют. Тогда таблица стилей будет формироваться следующим образом. Например, вы хотите, чтобы во всех браузерах, кроме Netscape 4.x и Internet Explorer 4.x—5.x высота строки была 1.5.

Это реализуется так:

```
@media all {
  P/*  */{
    line-height: 1.5}}
```

Кавычки

Есть крайне любопытный баг, связанный с кавычками. Например, если заключать в них значения свойств, то браузеры Opera 5+ и Netscape 6.x проигнорируют это значение. Допустим, мы захотим создать элемент <INPUT> С рамкой черного цвета, но не хотим, чтобы этот стиль применялся в браузерах Netscape 6.x и Opera 5+. Тогда нам надо написать такой кол.

```
INPUT.blackborder {
  border: "1px solid #000")
. . .
<INPOT TYPE-"text" SIZE="12" ClASS«"blackborder">
```

Вообще это имеет смысл, потому что рамки в браузере Opera 5+ на элементы формы выглядят не так, как надо.

Пора подвести итог и собрать все методы в единую таблицу (табл. 10.2). Она будет вам ценным помощником в некоторых случаях. Например, вы верстаете страницу и внезапно обнаружили, что какая-то комбинация свойств-значений в таблице стилей приводит к зависанию браузера Netscape 6л\ Скорее всего, эта комбинация не будет критичной с точки зрения дизайна, так что ее можно просто убрать, но зачем, если она все же делает страницу удобнее или красивее? Можно отключить ее для пользователей браузера Netscape 6.x и только. Вы заглядываете в нижеприведенную таблицу, находите, какой метод скрывает стиль от браузера Netscape 6.г, и успешно его применяете.

Таблица 10.2. Таблица методов сокрытия стилей от основных браузеров

Метод	NN4	NN6	IE4	tES	IE6	Opera 5+
MEDIA-al.	Да					
MEDIA»scReen						Да
9import	Да		Да			
Селекторыпоатрибутам	Да		Да	Да	Да	
Селектор наследника	Да		Да	Да	Да	
Voice-family			Да	Да		
Комментарии			Да	Да		
Кавычки	Да	Да				Да

Приведу пару реальных примеров. Я совершенно случайно нашел занимательный и очень неочевидный баг в Netscape *6.x* (он касается и ранних версий Mozilla, iiciipaaien в браузере Mozilla 0.9.7J. У меня был такой стиль:

```
news:first-line •
  font-weight: bold)
```

И следующий код:

```
<TABLE WIDTH=252>
  <TR>
    <тп CLASS="news"><A HKEP="#">Если длина этой'строки Судет больше двух
строк, то браузер вылетит</A></TOx/TK>
</TABLE>
```

Так вот. Оказывается, эта комбинация приводит к зависанию браузера — выводится сообщение о недопустимой ошибке. Баг возникает тогда, когда используется псевдоэлемент first-line внутри ограниченного пространства (таблица шириной в 252 пиксела), причем контент внутри таблицы должен занимать не менее трех строк. Эта таблица я&тялась блоком новостей и первую строку надо было вывести полужирным шрифтом. Но браузер Netscape 6 '. выполнял недопустимую операцию, так что надо было этот стиль от нель скрыть. Решение простое, надо взять в кавычки значение свойства font-weight

```
news:first-line {
  font-weight: "bold"}
```

Еше один пример. Как вы уже знаете, Internet Explorer 5.v некорректно об рабатывает ключевые слова в размере шрифта. Так, если вы установи! font-size: small, то размер шрифта будет разный в браузерах фир\п Microsoft и в других браузерах. Чтобы размер совладал, надо для браузер фирмы Microsoft установить значение зт.а::. а для остальных браузеров - значение medium. Это исправлено в браузере Internet Explorer 6..v, так чп» нам надо скрыть стиль только от браузеров младших версий. Смотрим - таблицу и видим, что для этого можно воспользоваться комментариями.

Тогда таблица стилей будет выглядеть таким образом:

```
P (
  font-size: small}
P/* •/ {
  font-size: medium}
```

Получится, что первую строчку поймут все браузеры и установят размен- small, а вторую строчку поймут все браузеры, кроме Internet Explorer 4-v 5.Y. так что они перепишут первое значение small новым значением medium

В итоге получится, что Internet Explorer 4.x—5.* видит таблицу стилей:

```
P (
  fcnt-si^e: srT\5ll '>
```

А все остальные браузеры:

```
P {
  font-size: medium.}
```

Так что размер шрифта будет одинаков *во всех* браузерах.

Кроме самих каскадных таблиц стилей для кросс-браузерного кода можно пользоваться языком JavaScript.

Разделение таблиц стилей

Идея такая. С помощью JavaScript можно определить тип и версию браузера, а затем подключать к документу именно ту таблицу, которая написана *специально* для данного браузера. Есть много разных скриптов, но один из лучших опубликован на сайте www.richinstyle.com. Я его немного модифицировал, и в конечном виде выглядит он так:

```
ua=navigator.userAgent;
1-"<LINK rel^'stylesheet" type="text/css" href="';
c='.css">';
if (ua.indexOf{'IE 4')!=-l) document,write (1+ie4 ••*<;! ;
if (ua.indexOf('IE 5')!=-!) document.write(I+'ie5'+c);
if (ua.indexOf{'IE 6')!—1) document.write<l+'ie5'+c);
if (ua. indexOf ('Opera') !=-l) document.write(1+*op*+c);
if (ua.indexOf('Netscape6')!—1) document.write(l+'moz'+c);
if (ua.indexOf('Gecko'Y1—-1) document.write(l+'moz'+c);
  else if {ua.indexOf("compatible")™=-l) (
    if (ua.indexOf("/4")!=-l) document.write(l+'nn4'+c))
```

В общих чертах скрипт работае! следующим образом. Вначале для сокращения кода вводится переменная ua, которая содержит объект navigator.userAgent. Этот объект представляет собой строку, которую отсылает браузер серверу для своей идентификации. В ней содержится имя браузера, его версия и название его движка.

Д&lee опять же для сокращения кода вводится еще одна переменная 1, которая содержит первую часть тега <ыык> для подключения таблицы стилей, а переменная с содержит вторую часть тега <LINK>.

Далее идут сравнения. Функция indexOf позволяет выделить часть символов из строки. Так что код

```
if (ua.indexOf('IE 4')!»-l) document.write(l+'ie4 *+c) ;
```

обозначает следующее: в строке navigator.userAger.t ищется строка IE 4 Если она есть, то с помощью конструкции document.write к странице подключается файл ie4.css. Если же такая строка отсутствует, то файл не подключается и проверка продолжается.

Так работает скрипт. А принцип действий при верстке следующий. Вы создаете отдельные таблицы стилей для тех браузеров, которые ведут себя HL так, как вы планировали. А x™ тех браузеров, которые ведут себя одинакова с каким-либо другим, отдельных таблиц стилей создавать не нужно.

В данном скрипте предусмотрено подключение следующих таблиц стилей.

- Для браузера Opera любых версий — файл op.css.
- Для браузера Internet Explorer *4.x* — файл ie4.css.
- Для браузера Internet Explorer 5..v— файл ie5.css.
- Для браузера Internet Explorer 6.x — файл ie6.css.
- Для браузеров Netscape 6.v и Mozilla — файл тог.ess.
- Для браузера Netscape Navigator 4.x — файл nn4.css.

ЭТО. пожалуй, максимально возможное число отдельных таблиц стилей, которые вам вообще когда-либо могут понадобиться. Обычно же хватает двухтрех. Если вам не нужна отдельная таблица стилей, скажем, для браузер. Internet Explorer 4.x, то надо просто удалить соответствующую строчку :- скрипте, в данном случае такую:

```
if (ua.indexOfC IE 4')!=-l)document.write{l+'ie4'+c);
```

Вообще, делают так: создают простейшую таблицу стилей, которая будекорректно обрабатываться всеми брауэерами, и подключают ее как обычш при помощи элемента <LINK>; затем (обязательно после элемента <LINK>!> подключают скриптовый файл, который уже в свою очередь подключает нужную таблицу стилей, содержащею CSS-код для данного браузера.

Главное достоинство такого способа — простота Вы совершенно не задумываетесь над тем, какой метод использовать для сокрытия стилей от того или иного браузера. Вы просто берете и пишете для каждого браузера отдельную таблицу стилей. Для сложной верстки этот способ лучший Но у него есть один недостаток. Некоторые пользователи отключают JavaScript в браузере по тем или иным причинам (обычно в целях безопасности), в этом случае такой способ подключения стилей не будет работать.

В настоящее время самым распространенным является дифференциация браузеров Internet Explorer 5+, Netscape 6.v и Opera 5-". Для них создаются таблицы стилей ie.css, nn.css и op.css, а для всех остальных браузеров all.css Код для дифференциации будет таким: *Щ*

```
<LINK REL="stylesheet" TYPE="text/css" HPEF="all.css">
<SCRIPT TYPE="text/javascript">
```

```
ua=navigator.userAgent;
1-'<LINK rel-"stylesheet" type="text/css" href-"';
c='.css">';
if (ua.indexOf('IE 5*  --i> docuraent.write(l+'ie'+c);
if [ua.indexOf('IE 6')!=-l) document.write(l+'ie'+c);
if (ua.indexOf('Opera'''=-!) document.write(l+'op'+c);
if (ua.indexOf 'Netscape6')I°—!} document.write(l+'nn'+c);
if (ua.indexOf; 'Gecko') !=-l) document.writetl+'nn'+c);
</SCRIPT>
```

Собственно понимание необходимости применения кросс-браузерного CSS придет к вам по мере профессионального роста. Вы вряд ли станете сразу тестировать сайт в нескольких браузерах и помнить все их особенности, но со временем вам без этого будет не обойтись, если захотите достичь вершин мастерства.

В заключение привожу сводную таблицу (табл. 10.3), в которой есть некоторое сравнение браузеров Internet Explorer *6.x,* Netscape б.д: и Opera *6.x.* Конечно, невозможно свести в таблицу все свойства браузеров, но ряд достаточно важных — можно.

Таблица 10.3. *Сводная таблица особенностей браузеров шестых версий*

Свойство	Internet Explorer 6.x	Netscape 6.x	**Opera 6.x**	**Комментарий**
Фильтрация cookies	Очень сложно пользоваться	Есть и чрезвычайно простая	Есть, но несовершенная	
Редактор cookies	Нет	Есть	Нет	
Автозаполнение форм	Есть	Есть	Крайне ограниченное	В этом отношении Opera сильно проигрывает. А у IE данная опция лучшая
Менеджер паролей	Нет	Есть	Нет	
Плагины	Только ActiveX	Есть	Есть	IE 6 не поддерживает технологию плагинов, что является большим недостатком
Сохранение открытых окон	Нет	Нет	Есть	
Отключение графики/стилей одной кнопкой	Нет	Нет	Есть	Данная опция может серьезно поднять скорость загрузки на слабых каналах

Таблица 10.3 *(окончание*

Свойство	**Internet Explorer 6.x**	Netscape 6.x	Opera 6 x	Комментарий
Блокировка всплывающих окон	Нет	Есть	Есть	Полезная опция для отключения назойливых баннеров
Полноэкранный режим	Есть	Нет	Есть	
Увеличение страницы (zoom)	Только шрифты, заданные не в пикселах	Только шрифты. но **любые**	Всю страницу	В этом плане Opera лучше всех тогда как IE 6 слаб
Подсветка найденных слов	Нет	Нет	**Есть**	
Предварительный просмотр перед печатью	Есть	Есть	**Есть**	Самый удобный, на мой взгляд у IE 6 Самый слабый у Netscape 6
Поддержка иконок (favicon)	**Есть**	**Есть**	**Нет**	
HTML	2	1	3	
CSS	3	2	1	
DOM	**1**	2	3	
XSL	Есть	**Нет**	**Нет**	IE 6 вне конкуренции, но только он **один** и поддерживает XSL

Глава 11

Приемы и хитрости

Эта глава состоит из маленьких заметок, совершенно не связанных между собой. Их объединяет лишь одно обстоятельство. Ниже дано решение некоторых частных проблем, которые по разным причинам не рассмотрены в предыдущих главах или представлены там недостаточно подробно.

Здесь также собраны оригинальные свойства, которые не поддерживаются всеми браузерами, а являются достаточно специфичными в каждом конкретном случае.

Печать страниц

Зачем вообще страницы распечатывают? Вы, наверное, знаете, что информация с экрана монитора воспринимается гораздо хуже, чем информация, напечатанная на бумаге. В большинстве своем исследователи утверждают, что разница составляет 25—30%, что достаточно много. Это и служит основной причиной, по которой посетители сайтов распечатывают страницы. Особенно это касается сайтов с техническими статьями и обучающими матсриалами. Так что печать страниц является достаточно серьезным вопросом, которому надо уделить внимание.

До недавнего времени печать страниц была головной болью для вебразработчиков. Браузеры отвратительно выводили на печать сложные страшны, и не было никакой возможности предварительно просмотреть резуль*лт.* По этой причине для серьезных ресурсов всегда делали два варианта чаждой страницы: первый — для браузера, а второй — для принтера. В на-л.lс или в конце материала на видном месте размещали ссылку па страни-л, предназначенную для печати. Она не содержала навигации, сложной рафики и прочих ненужных вещей, потому что при распечатке страницы человека обычно интересует только информация, а не дизайн сайта.

С течением времени ситуация постепенно улучшалась. Так, в браузерах)рсга и Internet Explorer 5.5 появился предварительный просмотр страницы :ри выводе на печать, позже такую же функцию реализовали в браузере Mozilla. Так что пользователь при выводе страницы на печать мог посмотаь, как она будет выглядеть на бумаге.

Кроме того, в последних версиях браузеров можно контролировать вид -траницы, выводимой на печать, с помощью CSS! Для этого надо просто

создать отдельную таблицу стилей (например, в виде файла prinl.css) и подключить ее к странице единственной строчкой

```
<LINK REL="stylesheet" TYPE="text/css" HREF="print.ess" MEDIA="print">
```

т. е. в атрибуте MEDIA надо указать, что таблица стилей предназначена для принтера.

Давайте рассмотрим, какую таблицу стилей надо сделать, чтобы отпечатанная страница не содержала ничего лишнего, а текст читался хорошо. Сначала четко сформулируем задачу. Итак:

- страница для печати не должна содержать рисунков;
- страница для печати не должна содержать навигации по сайту;
- текст должен занимать всю ширину страницы, не считая полей;
- текст должен выводиться шрифтом Times New Roman, потому что на бумаге лучше читается шрифт с засечками.

Вот основные правила, которыми нужно руководствоваться при подготовке таблицы стилей для печати.

Для наглядного примера можно взять все ту же страницу, которую мы с вами модифицировали и улучшали на протяжении всей книги. Помните ее? Нам надо вспомнить HTML-кол, чтобы написать для него таблицу стилей.

```
<BODY>
  <DIV ID="left">
    <DIV ID="logo"><IMG SEU>"img/fragment_1.gif"
                    WIDTH="266" HEIGHT="86"
                    АЬТ="ЭНИГМА - криптографические средства защиты
                         информации [логотип]">
    </DIV>
    <DIV ID="leftl" CIASS="back">
      <BR><BR>
      <A HREF="#">Q KOMTiaHiiH</A><BR>
      <A HREF="#">ripOflyKTbi</AXBR>
      <A HREF="#">PeiueHH*</A><BR>
      <A HREF="#">CTaTbH</A>
    </DIV>
    <DIV ID="left2" CLASS="bluecol">
      <A HREF="#">oTKpbrraH криптографиЖ/AXЫO
      <AHPEГ="#">цифроваяnoflnnCb</A><BR>
      <AHREF="#">^eKTpOHHbMдокумент</A><B10
      <A HREF="#">4ero не могут хакеры</A><BE1>
    </DIV>
```

```
  </DIV>
  <DIV ID-"nght">
    <IMG SRC="img/fragment_2.gif" WIDTH="251" HEIGHT="167"
         АЬТ=-"ЭНИГМА — быстрый путь к надежности">
    <DIV ID="txt">
      <BR>
      <H3 ALIGN="right">оТКpuraH криптография</H3>

      <P>Вы запускаете браузер, набираете в адресной строке URL сайта и
получаете желаемую страницу. Вот, вкратце, действия пользователя, который
бродит в сети Интернет. Он не задумывается, что лежит за всем этим. Ему
это не нужно. Но это нужно нам, профессиональным веб-разработчикам, чтобы
максимально удовлетворить пользовательские запросы.</P>

      <P>Мы не будем углубляться в историю, и отряхивать пыль с архивных
90-х годов, но базовые понятия рассмотрим. Итак, что же лежит в основе
всемирной сети Интернет? Базовыми являются три технологии:</P>
    </DIV>
  </DIV>
</BODY>
```

Если мы попробуем распечатать эту страницу в браузере Internet Explorer 6.x, то получится результат, показанный на рис. 11.1.

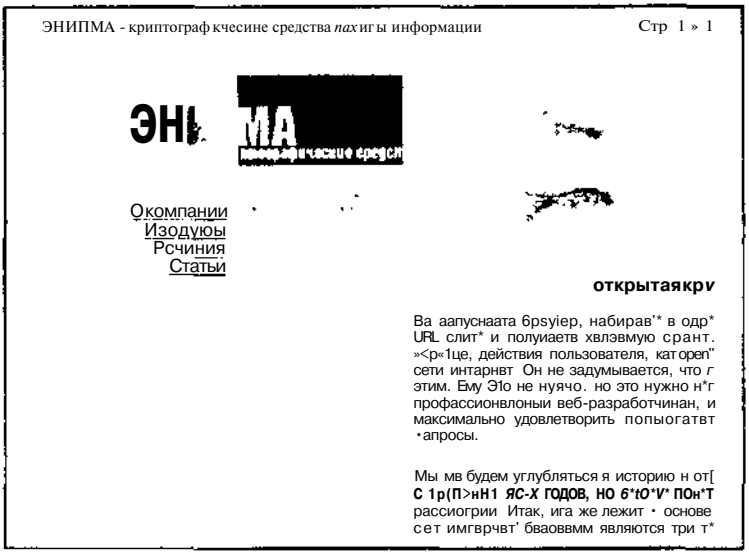


Рис. 11.1. Вид распечатанной страницы в браузере Internet Explorer *6.x*

Как видите, на странице не только присутствует графика и навигация, но сам текст обрезается по правому краю п не помешается на листе бумаги, так

что эта распечатанная страница совершенно бесполезна! Обратимся к коду. У нас все содержится в отдельных слоях. Надо оставить только текст и заголовок статьи, значит надо скрыть вес остальные слои, кроме слоя с iD="txt". Вся навигация у нас находится в блоках left: и ieft2. Для тою чтобы отключить навигацию, можно просто убрать эти блоки со страницы. Как вы помните, для этого существует объявление display: none. Оно как раз полностью удаляет элемент из нормального потока. Следовательно, и нашем случае навигация убирается так:

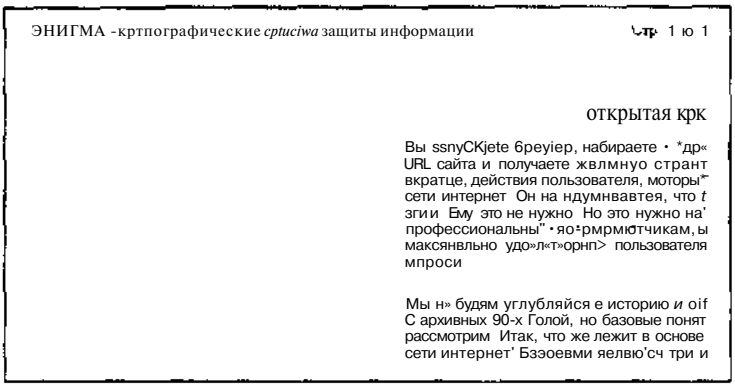
```
fUeftl
  display: none}
#lefC2 {
  display: none}
```

Далее нам надо отключить картинки. Все картинки вставляются в документ посредством элемента <JMG>, так что надо применить объявление display: none именно к этому элементу, что приведет к удалению iu норм&тъною потока всех картинок.

```
IMG 1
  display: noneI
```

Обратите внимание, что это правило отключает все изображения, но сама статья может содержать информативные рисунки, так что этот способ подходит не всегда. Если у вас именно такая ситуация, то все изображения, относящиеся к дизайну, надо постараться обьедипить в несколько блоков, которые в таблице стилей для принтера можно отключить.

Таким образом, JTJI три правила \бирают со страницы все ненужные нам элементы. После этою распечатанная страница будет выглядеть так, как па рис. 11.2.


ЭНИГМА -кртпографические *cpticiwa* защиты информации

открытая крк

Вы ssnyCKjete breyiep, набираете • *др« URL сайта и получаете жвлмнуо страит вкратце, действия пользователя, моторы" сети интернет Он на ндумнвавтея, что *t* зги и Ему это не нужно Но это нужно на' профессиональны" • яо:рмрмютчикам, ы максянвльно удо»л«т»орнп> пользователя мпроси

Мы н» будям углубляйся е историю *и* oif С архивных 90-х Голой, но базовые понят рассмотрим Итак, что же лежит в основе сети интернет' Бзэоевми яелвю'сч три и

Рис. 11.2. Вид распечатанной страницы после отключения отображения навигации и рисунков в браузере Internet Explorer 6.x

Как видите, ненужных элементов больше нет. по текст по-прежнему выводится шрифтом Verdana и обрезается справа. Попробуем решить эту проблему. Обратимся к коду. У нас есть два больших блока, которые называются left и right. Левый блок содержал навигацию, но сейчас ее нет, т. е. его тоже можно отключить.

```
Heft {
  display: none)
```

Правый блок содержит текст. Он имеет ширину 402 пиксела и отступ слева 348 пикселов. Сейчас нам отступ совершенно не нужен, а ширина должна быть 100%. Так что напишем такой стиль:

```
#right {
  width: 100%;
  position: absolute;
  left:Орж,
```

Осталось поменять шрифт. Делается это так:

```
? {
  font-family: "Times New Roman", serif}
```

Можно еще выровнять заголовок по левому краю, потому что в данном случае это будет выглядеть лучше:

```
ИЗ{
  text-align: left)
```

Полная таблица стилей для вывода страницы на принтер будет выглядеть так:

```
ИЗ {
  text-align: left}
IMG {
  display: none}
? {
  font-family: "Times New Roman", serif}

#left {
  display: none}
#leftl {
  display: none}
#left2 {
  display: none}
#right {
  width: 100%;
```

```
position: absolute;
left:0px}
```

Поместим ее в файл print.css и подключим к странице с помощью элемент^ <LINK> со значением атрибута MEDIA, равным print:

```
<LINK REL="stylesheet" TYPE="text/ess" HREF="print.ess" MEDIA="print">
```

Тогда распечатанная страница будет в точности соответствовать всем требованиям (рис. 11.3).

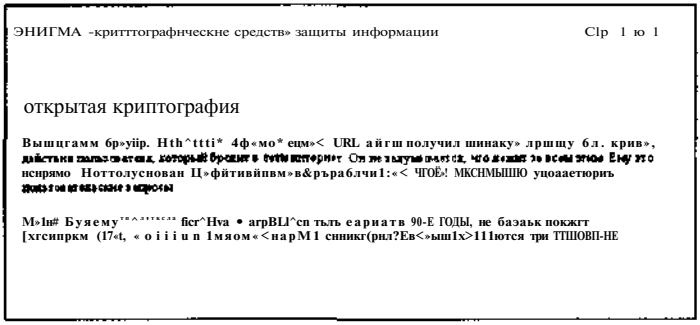


Рис. 11.3. Окончательный вид страницы, предназначенной для печати

Это будет корректно работать в браузерах Opera 5+, Internet Explorer 5+, Mozilla 0.9.7+, так что практически все посетители вашего сайта смогут в полной мере насладиться печатной версией без каких-либо дополнительных усилий со своей стороны.

Изменение курсора

В спецификации CSS-2 есть достаточно интересное свойство, которое позволяет устанавливать тип курсора. Это свойство cursor.

Свойство *cursor*

В качестве параметра задается одно из ключевых слов или же URL графического файла с расширением cur. Теперь вы можете нарисовать свой собственный курсор и подключить его к HTML-странице. Это хорошо тем, что ваш сайт может стать более самобытным {в различных играх оригинальные курсоры совершенно привычное дело) и интерактивным, а плохо тем, что нарисованные вами курсоры могут быть непонятными или просто некрасивыми, и, кроме того, пользователи уже привыкли к стандартным курсорам, так что удобство пользования сайтом однозначно уменьшится. Так или иначе, решение об использовании собственных курсоров принимать вам.

А стандартные курсоры могут быть такими:

- crosshair — крестик;
- default — тип курсора по умолчанию в данной операционной системе. Зависит от конкретной операционной системы;
- pointer — указатель на ссылку. В системе Windows это рука с указательным пальцем;
- move — обозначает, что объект может быть перемешен с помощью мыши. Полезен при использовании DHTML;
- text — обозначает, что текст может быть выделен. В системе Windows это курсор в виде буквы I;
- wait — обозначает, что система занята. Обычно отображается в виде песочных часов;
- help — обозначает, что в данном месте доступна помошь (подсказка). Обычно отображается стрелочкой со знаком вопроса. С помощью DHTML можно делать динамические подсказки, так что в этом случае изменять тип курсора на help будет правильно;
- e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize — обозначают, что объект может изменять свой размер (то есть край объекта может быть перемещен). Естественно, что для нижнего левого и для нижнего правого угла курсоры будут разными. Здесь, например, нижний левый угол обозначается nw-resize, а верхняя сторона объекта — n-resize.

На рис. **11.4** представлены все перечисленные выше курсоры в операционной системе Windows.

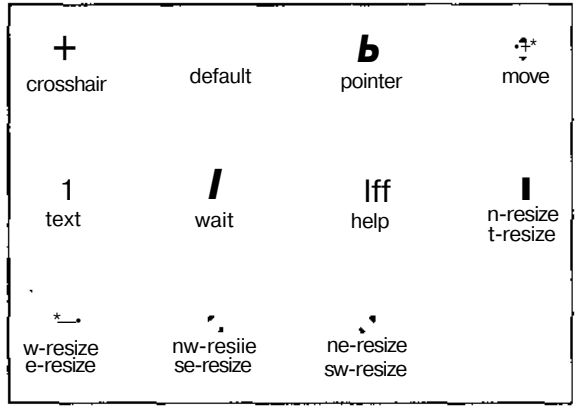


Рис. 11.4. Вид курсоров в операционной системе Windows

Свойство cursor поддерживается браузерами Internet E\ploier 5+ и Netscape 6jr. но не поддерживается браузером Opera.

Что касается своих собственных курсоров, то их поддерживает только браузер Internet Explorer 6д\ Впрочем, это не является большой проблемой. Вы можете задать для пользователей браузера Internet Explorer 6.v собственным курсор, а для всех остальных пользователей курсор стандартный, но подходящий по контексту. Например, вы хошге сделать свой собственный курсор при наведении на ссылку. Тогда в таблице стилей для элемента чл> надо указать:

```
A {
  cursor: url(link.cur), pointer}
```

Здесь через запятую последовательно перечисляются значения, которые будут приняты в случае невозможности приник предыдущее. Точно так же мы через запятую перечисляли возможные шрифты для элемента. Это должно работать, но пока браузер Mozilla игнорирует второе значение. Надеюсь, в скором будущем это исправят.

Полоса прокрутки

В браузерах фирмы Microsoft, начиная с версии 5.5, можно устанавливать цвета полос прокрутки в браузере. Вообще, полоса прокрутки состоит m нескольких элементов: стрелки, бегунка и поверхности для него Кроме того, она трехмерная. Для дополнительного разукрашивания полосы существует несколько свойств:

- scrollbar-3diight-coior — цвет для блика, который располагается на левой и верхней сторонах бегунка и стрелок (по умолчанию белый);
- O scroilbar-arrow-color — цвет для самих стрелок (по умолчанию черный);
- scrollbar-base-coior — базовый цвет всей полосы (по умолчанию серый);
- scroiibar-darkshadow-coicr — цвет дли тени, которая располагается на правой и нижней сторонах бегунка и стрелок (по умолчанию черный);
- O scrollbar-face-color — базовый цвет для стрелок и бегунка (по умолчанию серый);
- scrollbar-highlight-color — цвет блика на полосе прокрутки;
- O scrollbar-shadow-color — цвет тени на полосе прокрутки;
- scrollbar-track-color — цвет для фона полосы прокрутки.

Вообще, словами описать значение каждого свойства достаточно сложно, но на рис. 11.5 некоторые из указанных элементов оформления есть.

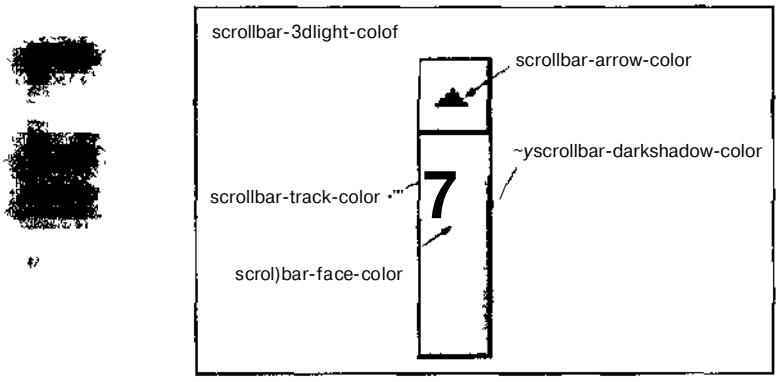


Рис. 11.5. Некоторые свойства для задания цвета полосы прокрутки

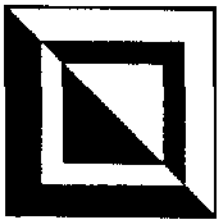
Я предлагаю вам поэкспериментировать самим, потому что приемлемого результата сразу добиться сложно. А пока несколько замечаний.

Полосы прокрутки могут быть как у всего документа, так и у отдельной его части. Например, с помощью объявления overflow: scroll можно сделать блок, который будет иметь полосы прокрутки, кроме того, полосы прокрутки могут быть реализованы для элемента <?F,XTAREA>. ДЛЯ ТОГО чтобы задать цвета полос для главного окна, надо применить данные свойства к элементу <BODY>. Следующее правило делает полосу прокрутки белой с черной рамкой:

```
BODY {
  scrollbar-3dlight-color: #000;
  scrollbar-base-color: #FFF}
```

Все указанные свойства не относятся к официальной спецификации CSS, так что они не иоддерживаю1ся никакими другими браузерами, которые их просто игнорируют. Так что, раскрашивая полосы прокрутки, вы ничем не рискуете с технической точки зрения. Конечно, остается еше один очень спорный вопрос С одной стороны, при раскрашивании полосы возможно улучшение визуального восприятия сайта, потому что даже полоса прокрутки будет соответствовать цветовой гамме сайта. С другой стороны, пользователи привыкли к серой полосе, и другой цвет может приводить их в замешательство, что ухудшает психологическое восприятие сайта Лично я предпочитаю остэдлять цвет полосы таким, какой он есть в операционной системе, потому что обилие красок отнюдь не помогает восприятию информации.

Глава 12



Будущее

Среда Интернет чрезвычайно динамичная и стремительно развивающаяся. Стандарты рождаются и умирают буквально в течение нескольких лет. Технологии иногда совершенно не успевают за стандартами, а иногда далеко обгоняют их. Разработчики не успевают ни за стандартами, ни за технологиями и частенько живут вчерашним днем. Почему так происходит? Без тщательного анализа положения дел в отрасли ответ на этот непростой вопрос дать невозможно. Давайте бросим взгляд хотя бы на последние пять лет. Что произошло за это время?

Появилось несколько достаточно популярных языков программирования серверных приложений. К ним относятся PHP, ASP, Python, ColdFusion и некоторые менее известные. Язык PHP (Personal Home Page) вообще развивается поразительными темпами и за пару лет приобрел огромное число поклонников. В основном за счет своей относительной простоты, логичности и функциональности. Также весьма популярным стал язык ASP (Active Server Page). Он, по мнению большинства, быстрее PHP, по работает только на серверах под управлением операционной системы Windows, в чем и заключается его огромный недостаток.

Появилось несколько реляционных баз данных, например MySQL и Postgrc SQL. Связки серверных скриптов и баз данных подняли на совершенно новый уровень функциональность сайтов. Веб-программистам пришлось ускоренными темпами осваивать хотя бы некоторые из этих технологий, потому что без этого они бы просто утратили квалификацию

Если взять верстку, то новинок было не меньше. Критическим образом изменился язык HTML, потому что только с появлением таблиц HTML-верстальщики собственно и обрели свое призвание и выделились в особую группу (до этого верстка сайтов была настолько тривиальной, что ей мог заниматься практически кто угодно). Появился стандарт CSS, который, впрочем, окончательно утвердился в среде веб-разработок лишь совсем недавно. К сожалению, компания Microsoft с некоторых пор не считает CSS технологией будущего и внедряет ее достаточно медленно. Именно по этой причине на данный момент времени многие свойства из стандарта CSS-2 не имеют практического значения. Браузеры Netscape 6.v и Opera 5+ поддерживают стандарт CSS-2 достаточно неплохо, а браузеры Internet Explorer *5JC—6.X* лишь на зачаточном уровне, но именно они и формируют среду для всб~ разработчиков. Какой смысл верстать сайт с использованием селекторов

CSS-2, если у 90% пользователей он будет выглядеть совершенно не так, как задумывал создатель?

Кроме CSS появился принципиально новый язык разметки XML, а также язык стилей XSL. Предоставляя широкие возможности, эти языки гораздо сложнее, чем HTML и CSS. Естественно, их сложнее изучать и на это пало потратить больше времени. Их распространение до последнего времени сдерживала плохая поддержка браузерами. Однако ситуация потихоньку испра&тяется, все новые браузеры поддерживают XML. но вот XSL поддерживают только браузеры фирмы Microsoft. Браузеры Opera 5-^ и Netscape 6.Y поддерживают связку XML + CSS, а браузеры Internet Explorer 5+ поддерживают связку XML + XSL и очень слабо связку XML + CSS. Такой разброд очевидно не способствует популяризации XML, и у HTML-программистов нет веской причины заняться изучением этой технологии.

Появился скриптовый язык JavaScript и Объектная Модель Документа (Document Object Model, DOM). Все вместе слилось в так называемый DHTML (Dynamic HTML). С этой технологией тоже проблем немало. Браузеры поддерживают объектную модель документа в разной степени. Так. в Opera 5+ объектная модель очень слабая, однако в остальных браузерах она достаточно развита. Порой приходится писать достаточно сложный кросс-браузерный код, потому что в браузерах Internet Explorer и Netscape по разному реализован доступ к тому или иному элементу. Технология DHTML сама по себе достаточно сложна, а если учитывать проблемы в совместимости, то сложность еше более возрастает.

Посудите сами, сколько надо всего знать, чтобы быть хорошим веб-мастером. Просто невозможно в совершенстве освоить все эти технологии, потому что каждая из них еше и постоянно прогрессирует (исключая HTML). На данный момент почти все HTML-верстальшики должны знать HTML, CSS, DOM, JavaScript, DHTML. Но это уже достаточно много. Вообще, чтобы на хорошем уровне освоить данные технологии, нужно года два при наличии определенного таланта. Это много. Так что в серьезных студиях уже намечается разделение труда: некоторые веб-мастера занимаются только версткой, т. е. им достаточно знать HTML и CSS. а некоторые специализируются на скриптах и сложных интерактивных страницах, т. с. им достаточно лишь поверхностно знать HTML и чуть глубже CSS, но зато досконально DOM, JavaScript и DHTML.

Вообше, подобное разделение существовало всегда. Мелкие студии предпочитают брать на работу специалистов широкого профиля, чтобы минимизировать затраты на заработную плату. Такие специалисты знают понемногу обо всем. Но и результат их труда зачастую оказывается среднего уровня, т. е. средний HTML-код дополнен средним скриптом. Большие и серьезные студии предпочитают узкоспециализированных профессионалов. Для подобных фирм определяющим фактором является качество, так что они луч-

ше возьмут на работу двоих, из которых один досконально знает верстку, а другой знает все о скриптах, чем одного, который в некоторой степени умеет верстать и писать скрипты.

Впрочем, такая тенденция не яшшется уникальной для среды Интернет. Она ьюб&тьна и распространяется на все области человеческой деятельности. Если в XIX веке в школе один учитель преподавал все предметы, то в XX — каждый преподает отдельный предмет (конечно, иногда встречаются случаи, когда учитель истории и математики это одно лицо, но они являются бесспорным исключением). Сужение специализации есть необратимый процесс, который лишь ускоряется с развитием цивилизации и накоплением знаний.

Таким образом, у вас есть два пути. Если вы хотите работать в небольшой студии, то лучше изучать все понемногу, но если вы хотите работать в известной профессиональной фирме, то лучше выбрать для себя две-три технологии и постараться изучить их досконально. Например, HTML-верстальщику нужно взяться за HTML, CSS, а по возможности еще за XML и XSL. Также надо в некоторой степени владеть ImageReady или другой программой для нарезки изображений, досконально знать графические форматы GIF и JPEG и ошпмизацию изображений. Конечно, надо немного интересоваться и другими технологиями, потому что полное зацикливание на чем-то одном ведет к ограниченности и косности мышления.

Перейдем к предмету данной книги. Что можно сказать о будущем CSS? Прежде всего, надо рассмотреть стандарты. Их разработкой занимается консорциум W3C. Занимается он этим хорошо и уверенно. В 1996 году окончательно утвержден стандарт CSS-I, в 1998 году — CSS-2, стандарт CSS-3 будет окончательно утвержден в 2002—2003 годах. Так что дело за разработчиками браузеров — все зависит от них.

Со стороны разработчиков ситуация отнюдь не такая радостная. Активно внедряют стандарт CSS производители браузеров Netscape и Opera. На данный момент эти браузеры поддерживают большинство свойств и селекторов из спецификации CSS-2. Но основным игроком на рынке браузеров является компания Microsoft, и она очень сдержанно и медленно внедряет поддержку CSS. Начиная с браузера Internet Explorer 4.x. темпы внедрения замедлялись, а браузер Internet Explorer 6.* по поддержке CSS практически не превосходит браузер Internet Explorer 5.5. Спецификация CSS-2 поддерживается достаточно слабо но сравнению с остальными браузерами. Иными словами, создается устойчивое впечатление, что компания Microsoft либо не верит в будущее этой технологии, либо хочет продвигать свою. Второе предположение ближе к истине.

Если оценивать сегодняшний лень, то связка HTML * CSS используется практически всеми. Если смотреть в будущее, то на смену HTML *может* прийти язык XML. Правда, этот переход достаточно туманный, и когда он

произойдет — неизвестно, но он скорее случится, нежели нет. Тогда возникает проблема выбора языка стилей для языка разметки гипертекста. CSS в принципе является достаточно подходящим для этих иелен, но XSL псе же лучше. В недалеком будущем (скажем, через пять лет) технологии CSS может просто не найтись места на рынке веб-разработок, т. к. подавляющее большинство веб-мастеров будет использовать связку XML + XSL. Но в ближайшие два-три года технологии CSS ничего не грозит, поскольку на это время HTML по-прежнему останется превалирующим языком. Получается, что Microsoft смотрит достаточно далеко в будущее, но самое ближайшее ее не интересует.

На мой взгляд, это неверный подход, поскольку на данный момент времени поддержка технологии XSL преждевременна. Очень небольшое количество людей готово отказаться от привычного и до боли знакомого HTML в пользу XML, и еще меньшее количество желает пользоваться XSL. Это и неудивительно, поскольку стандарт XSL 1.0 окончательно утвержден консорциумом W3C только в 2001 году! Внедрение поддержки стандарта CSS-1 осуществлялось производителями браузеров четыре года, частичная поддержка стандарта CSS-2 — три года. Так что поддержка стандарта XSL будет реализовываться не меньшее количество времени, т. с. к году 2003—2004. Новое всегда приходит с достаточно большим скрипом. До сих пор XSL вообще не поддерживается никакими браузерами, кроме браузеров Internet Explorer. Да, это около 80% рынка, но нельзя пренебрегать оставшимися 20%, поскольку это все же очень большое число пользователей.

Таким образом, в ближайшие три года технология CSS будет определять вил страниц подавляющего большинства сайтов. Исходя из этого, можно без опасений изучать CSS, — его просто необходимо знать любому профессиональному HTML-верстальщику.

Ближайшее будущее

Давайте рассмотрим те особенности, которые появятся с внедрением стандарта CSS-2. Среди них есть достаточно интересные и полезные.

Новые селекторы

Пожалуй, селекторы — это один из самых интересных разделов каскадных таблиц стилей Возможность наиболее точно выбрать элемент как раз обеспечивает гибкость CSS, позволяет сократить размер кода и реагировать на действия пользователя, что делает страницу более интерактивной без применения сложных скриптов. Мы уже рассматривали многие селекторы, появившиеся в спецификации CSS-2, однако не все. Так что здесь будут описаны только те селекторы, которые не рассматривались до сих пор. Сразу

отмечу, что большинство из перечисленных здесь селекторов поддерживаются весьма ограниченным числом браузеров или же не поддерживаются вообще, так что использовать их можно достаточно редко и для очень специфичных целей, по это только пока. Начнем с псевдокласса : focus.

Псевдокласс *:focu$*

Это один из самых интересных псевдоклассов, наряду с :hover. Позволяет применять стили к элементам, которые находятся в фокусе. Что такое фокус, лучше всего объяснять на примере формы. Допустим, па странице находится форма из трех полей: "Имя", "E-mail", "Сообщение". Когда вы подводите курсор мышки к любому из полей, и щелкаете на нем — оно становится активным, т. е. в него можно вводить текст. Говорят, что это поле находится в фокусе. Вы можете перемешаться по полям формы с помощью клавиши табуляции.

Сразу становится ясно, каким образом можно использовать псевдокласс : focus. Он идеально подходит для выделения активного элемента. Например, когда вы вводите данные в форму, можно менять фон текущего поля, что повышает удобство пользования.

```
INPUT:focus {
  background-color: yellow)
```

Можно менять цвет рамки или сам вид рамки, но это уже менее наглядно.

```
INPUT {
  border: lpx solid #000}
INPUT:focus {
  border-color: #F00}
```

Можно даже увеличивать ширину поля формы. Это имеет некоторый смысл. В обычном неактивном состоянии иоле может иметь небольшую ширину. Однако в узкое поле иногда неудобно вводить текст, потому что полностью строка не видна. Увеличение ширины строки может не только показать пользователю, что данное поле активно, но и облегчит ввод строки. Следующий код увеличивает все активные поля, образованные элементом <INPUT> С 10 до 16 em.

```
INPUT (
  border: lpx solid #000;
  width: 10em}
INPUT:focus (
  width: 16«nt}
```

Однако самым ненавязчивым и, *а* то же время, заметным является изменение цвета фона активного элемента.

Кстати, псевдокласс : focus поддерживается браузером Netscape 6.x

Непосредственный сосед

Достаточно интересный и полезный селектор, который позволяет применить стиль к элементу, который непосредственно следует за каким-либо другим элементом. Обозначается это знаком +. Например, правило

```
Hl+P \
  font-style: italic)
```

указывает на то, что абзац текста, который расположен непосредственно за элементом <н!>, будет выведен наклонным шрифтом. В книгах часто каким-то образом выделяют первый абзац, если он содержит общие заключения. Теперь это очень легко можно делать с помощью селектора +.

Кроме того, с его помощью можно располагать первый абзац непосредственно после заголовка. Часто так смотрится лучше.

```
H1+P{
  font-style: italic-
  margin-top: -lem}
```

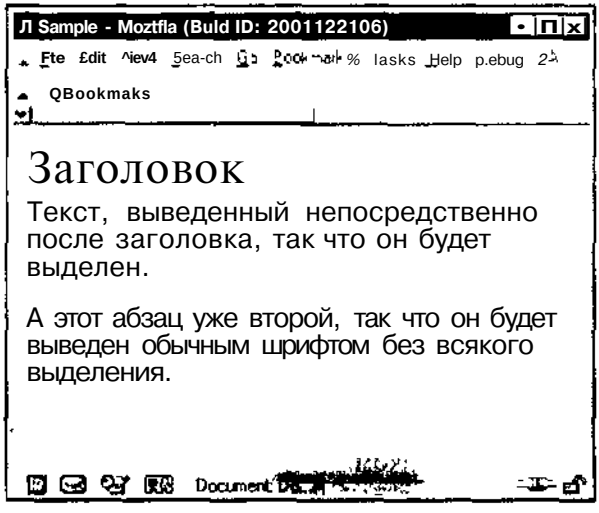


Рис. 12.1. Применение селектора непосредственного соседа. Первый абзац после заголовка HI будет выведен полужирным шрифтом и без отступа

Данный селектор поддерживается браузерами Opera 5+ и Netscape 6.x. Например, такой код:

```
P {
  font: lem Verdana, sans-serif}
Hl+P (
```

```
  font-weight: bold;
  margin-top: -lem}
. . .
<H1>Заголовок</H1>
<P>Текст, выведенный непосредственно после заголовка, так что он Судет вьщелен.</P>
<P>А этот абзац уже второй, так что он будет выведен обычным шрифтом без всякого выделения.</P>
```

будет отображаться браузером Netscape 6.v так, как показано на рис. 12.1.

Универсальный селектор

Это очень интересный селектор, обладающий уникальными свойствами. Он позволяет применять стили ко всем элементам, минуя наследование, т. е. фактически универсальный селектор представляет собой *любой элемент*. Обозначается универсальный селектор знаком звездочка — *. В следующем примере для всех элементов на странице будет установлен черный цвет шрифта:

```
* {
  color: #000}
```

Во многих случаях универсальный селектр как бы подразумевается, но его можно не указывать. В табл. I2.I записи слева и справа эквивалентны.

Таблица 12.1. Ситуации, когда универсальный селектор можно опускать

* [HREF]	[HREF)
*.classname	•classname
*#idname	#idname

Вообще, универсальный селектор — вещь достаточно полезная. С его помощью можно, например, применять стили к любому элементу, следующему непосредственно за заголовком.

```
H2 + * {
  margin-top: -lem}
```

Можно применять стили к любому элемент)', который вложен в элемент, например,<CODE>:

```
CODE * {
  background-color: #CCC}
```

Универсальный селектор поддерживается браузерами Internet Explorer 6.x, Opera 5+, Netscape *6.x*.

Селекторы по атрибутам

Мы уже встречались с этим типом селекторов в *главе !0,* однако там не рассмотрено несколько достаточно интересных способов его применения. Как вы помните, селектор вила [atti обеспечивает применение стилей ко всем элементам, которые имеют атрибут att. Попробуем с помощью этого решить весьма нетривиальную проблему из области удобства использования сайта. На HTML-странице обычно существует два типа ссылок:

- *внешние* — они ссылаются на локументы, находящиеся на другом сайте;
- *О внутренние* — они ссылаются на документы, находящиеся на этом же сайте.

При первом взгляде на ссылку нельзя сказать, какой она является: *внешней* или *внутренней.* Конечно, при наведении курсора на ссылку можно по URL в статусной строке определить это. но такой способ достаточно неудобный, так что лучше бито бы каким-либо образом *выделить внешние ссылки.* Например, внутренние ссылки можно сделать синего цвета, а внешние — красною (естественно, выбор цветов может быть произвольным и зависеть от дизайна сайта, но в данном случае сами цвета нас не интересуют, нам важен только сам факт, что они *разные).* Как же это сделать? Можно написать отдельный класс и строго встааоять его в каждую внешнюю ссылку, но это достаточно громоздкий способ. Давайте попробуем воспользоваться селектором по атрибутам.

Для этого надо найти атрибут, который будет присутствовать во внешних ссылках и отсутствовать во внутренних (или наоборот). Атрибут HREF присутствует во всех ссылках, так что он не подходит, а вот атрибут TTTLE ДЛЯ нас представляет интерес. В нем обычно расшифровывается сама ссылка, т. е. кратко объясняется, кула она ведет. Для внутренних ссылок он совершенно не обязателен, тогда как для внешних — крайне желателен. Поэтому во всех внешних ссылках нам надо ставить атрибут TITLE, а во внутренних — нет. В этом случае стили будут такими:

```
A (
  color: #00F}
AtTITLE) {
  color: #F00}
```

Тогда в коде:

```
<P>В этом предложении есть ссылка, на <A HREF="/index.html'^главную страницу</A> данного сайта. А в этом предложении есть ссылка на внешний сайт, которым является сайт студии <A HREF="http://www.artel.by" T1TbE="сайт ведущей белорусской студии дизайна">Дизайн Артель</A></P>
```

Ссылка http://www.artel.by будет красного цвета, потому что в ней присутствует атрибут TITLE (причем нам совершенно не важно, какой текст он будет

содержать). А ссылка /index.html будет синего цвета, потому что в ней атрибута TITLE нет.

Вообще, селекторы CSS-2 достаточно новая область, которая практически не освоена. До тех пор. пока большинство пользователей не перейдет на браузеры, поддерживающие селекторы CSS-2, она и не будет осваиваться. Однако, с точки зрения теории, новые селекторы могут быть очень полезны.

Генерируемое содержимое

Генерируемое содержимое не является совершенно новым видом контента на веб-страшшах. Что такое генерируемое содержимое вообще? По сути дела, это контент, который не содержится в дереве документа. В языке HTML генерируемым содержимым являются маркеры списка. Действительно, написав такой код:

```
<OL>
  <Ы>Первый пункт</ы>
  <Ы>Второй пункт</Ы>
</OL>
```

мы явно нигде не указываем, что перед каждым пунктом списка надо ставить цифру, тем не менее, в браузере она присутствует. В HTML и CSS-1 контроль над списком достаточно хороший: можно устанавливать разные типы маркеров, вплоть до произвольного изображения. Но зато нет никаких других механизмов контроля за генерируемым содержимым. Стандарт CSS-2 предоставляет разработчику достаточно широкие возможности. Например, можно сделать, чтобы в начале каждого замечания выводилось слово "Замечание:", чтобы перед описанием рисунка выводилось слово "Рис." и номер рисунка, и многое другое. Делается это с помощью свойства content и псевдоэлементов :before и :after. Свойство content совершенно уникально для CSS, поскольку в качестве параметра в нем может указываться строка текста.

Простейший пример. Если вам надо сделать на странице врезки, которые будут содержать очень важную информацию, то можно поступить следующим образом. Скажем, каждая врезка должна быть заключена в черную рамку шириной два пиксела, иметь серый фон и начинаться со слова "Внимание!". Это реализуется простым кодом.

```
P.attention {
  background: #CCC;
  border: 2px solid #000}
P.attention:before {
  content: "Внимание! ")
```

В результате перед всеми элементами <p> с классом attention будет вставляться слово "Внимание" с восклицательным знаком и пробелом. В HTML-коде врезка будет задаваться совершенно обычным образом:

```
<P ClASS="attention">3flecb содержится что-то очень важное</P>
```

Кстати говоря, генерируемое содержимое поддерживается браузерами Netscape 6.v и Opera 5+! На рис. 12.2 показано, как будет выглядеть данный код в браузере Netscape 6.x

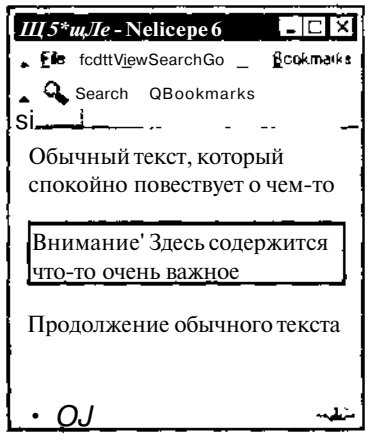


Рис. 12.2. Один из способов применения генерируемого содержимого. В данном случае генерируемым содержимым является слово "Внимание!"

А теперь подробно рассмотрим следующее свойство.

content

Оно может применяться только к селекторам с псевдоэлементами :before и :after. Значения для него существенно различные.

- *Строка*. Может содержать строку произвольного текста. Это значение мы уже рассмотрели на примере. Спектр его применения достаточно широкий: замечания, заголовки, ремарки и т. п.
- *URL*. Посредством задания URL можно вставлять изображения, звуковые файлы и прочие объекты. Например, вам надо, чтобы каждое замечание начиналось не со слова "Замечание", а с картинки. Это может быть стрелочка или другой графический объект. Например, если файл называется note.gif, а класс note, тогда код будет таким:

```
P.note:before {
  content: url("nofce.gif")}
```

D *Счетчики.* Это совершенно новый вид контента. С их помощью можно создавать нумерованные разделы и подразделы, как это принято в различного рода руководствах и официальных документах. Например, не нужно будет писать номер 1.2.2. перед очередным пунктом.

D open-quote и close-quote. Эти значения заменяются соответствующей строкой из свойства quote.

О no-open-quote И no-close-quote. Нужны ДЛЯ ОПИСаНИЯ ВЛОЖенНЫХ кавычек.

П attrtwM* атрибута!. Эта функция возвращает строку, которая является значением указанного атрибута. Например, если написать код:

```
IMG:after {
  content: attr(alt) }
```

то после всех рисунков на странице будет выводиться текст, содержащийся в атрибуте ALT. Вообще, применение этой функции особого смысла не имеет. Разве что когда надо выводить название самого рисунка.

Кавычки

Вам, должно быть, известно, что в разных языках кавычки отличаются. Так, например, в английском языке кавычки первого уровня — это обычные лапки " ", а кавычки второго уровня — это одинарные кавычки ' '. Так что в английском языке цитата может выглядеть вот так: "Some color combinaiions generally frustrate users and make it virtually unreadable for color deficit or 'colorblind' users". Обратите внимание, что вся цитата взята в двойные кавычки, а слово colorblind в одинарные, поскольку оно часть цитаты и его уже нельзя брать в двойные кавычки. В русском языке кавычки первого уровня — это елочки « », а кавычки второго уровня — это лапки. Если придерживаться правил, то так и надо делать. В HTML елочки можно вставлять с помощью спецсимволов slaquo; и sraquo;, но достаточно хлопотно постоянно следить за тем, как вставить кавычку, кроме того, код несколько увеличивается в объеме, что неудивительно.

В спецификации CSS-2 появился очень удобный механизм для задания типа кавычек в зависимости от языка. Осуществляется это с помощью псевдоэлементов ibefore, :after, : lar.g и свойств content И quotes.

quotes

Это свойство залает тип кавычек. В качестве параметра указывается строка в таком формате:

```
"левая кавычка первого уровня" "правая кавычка первого уровня" "левая кавычка второго уровня" "правая кавычка второго уровня"
```

т. е. с помощью данного свойства можно задать тип кавычек для какого-либо элемента. Например, в HTML специально для коротких цитат предназначен элемент <c>. Для него тип кавычек можно задать таким способом:

```
Q<
  quotes: '"' "" "|" '" "}
```

Однако такого правила недостаточно, чтобы элемент <*Q*> заключал текст в кавычки. Мы пока просто задали тип кавычек, и не более того. Нам надо сделать так. чтобы левая кавычка отображалась *перед* элементом <Q>, а правая кавычка — *после* элемента <Q>. Это реализуется с помощью псевдоклассов :before И rafter И свойства content:

```
Q:before {
  content: open-quote}
Q:after (
  content: close-quote)
```

Если в коде встретится предложение

<P>Предложение с <0>неОольшой цитатой</0> в середине</P>

то оно должно отобразиться браузером так:

Предложение с "небольшой цитатой" в середине

А сейчас рассмотрим, как сделать для разных языков разные кавычки. Например, для русского и английского. Для этих целей подходит псевдокласс :lang. Итак, хтя английского языка указываем:

```
Q:lang(en) •
  quotes: '"' "" "|" "'"}
```

А для русского языка:

```
Q:lang(ru) {
      quotes: '<<' '»' '"' "" }
```

Далее надо собственно подключить кавычки к элементу <Q>:

```
Q:before {
  content: open-quote}
Q:after {
  content: close-quote)
```

Теперь язык можно задавать в атрибуте LANG элемента <HTML>, И на страницах будут отображаться именно те кавычки, которые вы указали для данного языка. Например, для русскоязычной страницы:

```
<HTML LANG="ru">
<HEAD>
```

```
  <STYLE TYPE="text/css">
    Q:lang(en) {
      quotes: •" ' "" " '" '" " }
    Q:lang(ru) {
      quotes: '«' '»' "" "" }
    Q:before f
      content: open-quote}
    Q:after I
      content: close-quote}
  </STYLE>
</HEAD>
<BODY>
    <P><O>Цитата, выделенная кавычками, с <0>еще одним уровнем</0> вло-
женности кавычек</O></P>
</BODY>
</HTML>
```

Если у пользователя выбран русский язык, то браузерами этот код должен выводиться так:

«Цитата выделенная кавычками, с "еще одним уровнем" вложенности кавычек»

Однако на данный момент времени ни олин браузер не поддерживает псевдокласс :iang, хотя Opera 5+ и Netscape 6.x поддерживают элемент <Q>, свойства content и quotes. Таким образом, для них уже можно задавать тип кавычек для элемента <Q>, НО нельзя задавать тип кавычек для каждого языка.

Вообще, на основе данного механизма можно реализовать весьма интересный способ выделения слов. Это вытекает из того факта, что вместо елочек можно поставить любые символы! Например, можно выделять слова вот так — <слово>, или вот так — /слово/. Подобное выделение (в данном случае звездочками) реализуется следующим образом:

```
EM {
  quotes: '*' '*' "" "" )
EM:before {
  content: open-quote}
EMrafter (
  content: close-quote}
• • •
<P>Небольиой текст с <£M>вьщеленным</EM> словом. Оно должно Оыть доста-
точно хорошо заметным</P>
```

В этом случае слово, заключенное в теги <EMX/EM>, будет выделяться символами * (рис. 12.3).

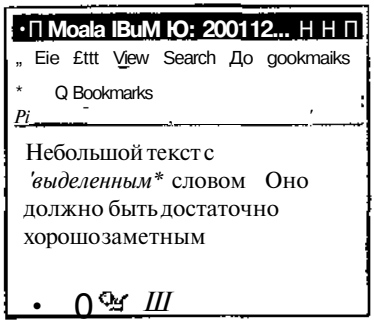


Рис. 12.3. Текст с выделением на основе псевдоклассов :before и :after

Счетчики

Счетчики предназначены для автоматической нумерации чего-либо. Это может быть нумерация разделов и подразделов, рисунков, глав и вообще любых элементов на странице.

Для создания счетчика надо использовать свойства counter-reset и counter-increment, а также функцию counter!) свойства content.

Свойство counter-increment задает величину, на которую увеличивается значение счетчика, когда в коле встречается очередной элемент с подключенным счетчиком. Если вы подключили счетчик к элементу <H1>, то первый элемент <HI> на странице будет иметь цифру 1, второй элемент <ш> — цифру 2 и т. д. По умолчанию величина приращения равна единице. В качестве значения для свойства counter-increment надо указывать название счетчика.

Свойство counter-reset сбрасывает указанный счетчик на начальное значение. В качестве параметра задается имя счетчика. Счетчик сбрасывается, когда в коде встречается очередной элемент, который в данном случае является селектором.

Давайте рассмотрим реатьный пример. Итак, нам надо, чтобы перед всеми заголовками <HI> выводилось слово "Глава" и номер главы, а после номера стояла точка. Перед всеми заголовками <H2> выводился номер главы, номер подраздела через точку, » все должно заканчиваться точкой. Вначале общий код:

```
HI:before (
  content: "Глава " counter(chapter) ". ";
  counter-increment: chapter;
  counter-reset: subpart}
H2:before {
  content: counter(chapter) "." counter(subpart) ".";
  counter-increment: subpart}
```

А теперь давайте подробно разберем каждое объявление.

- H1:before {

  content: "Глава " counter(chapter) ". ";

  Данное свойство указывает, что перед элементом <HI> будет выводиться слово "Глава" с пробелом, затем значение счетчика, который называется chapter, а затем точка.

- counter-increment: chapter;

  Данное свойство указывает на увеличение счетчика, т е. как бы включает счетчик chapter В качестве значения указывается имя счетчика.

- counter-reset: subpart}

  А здесь указывается, что если в коде обнаружен новый элемент <т>, то значение счетчика, который называется subpart, будет сбрасываться. Таким образом, реализуется правильная нумерация подразделов. Иначе говоря, если у нас был первый раздел {первый элемент <Ч1>), то при наличии второю раздела, все подразделы начинают нумероваться с единицы.

- H2:before (

  content: counter (chapter) "." counter{subpart) ".";

  По аналогии для подразделов выводится значение счетчика chapter, затем точка, затем значение счетчика subpart, затем опять точка.

- counter-increment: subpartI

  Здесь включается счетчик subpart.

Теперь, если у нас есть код:

```
<HTML>
<HEAD>
  <STYLE TYPE="text/css">
    Hi:before {
      content: "Глава " counter(chapter) ". ";
      counter-increment: chapter;, , <
      counter-reset: subpart}
    H2:before (
      content: counter(chapter) "." counter[subpart! ".";
      counter-increment: subpart}
  </STYLE>
</HEAD>
<BODY>
  <H1>Основы CSS</H1>
  <H2>Что надо знать о HTML</H2>
```

```
  <H2>Механиэмы таблиц стилей</H2>
  <H1>Позиционирование</H1>
  <H2>Табличная верстка</H2>
  <H2>Свойсггво float</H2>
</BODY>
</HTML>
```

то браузер должен отобразить его так, как показано на рис. 12.4.

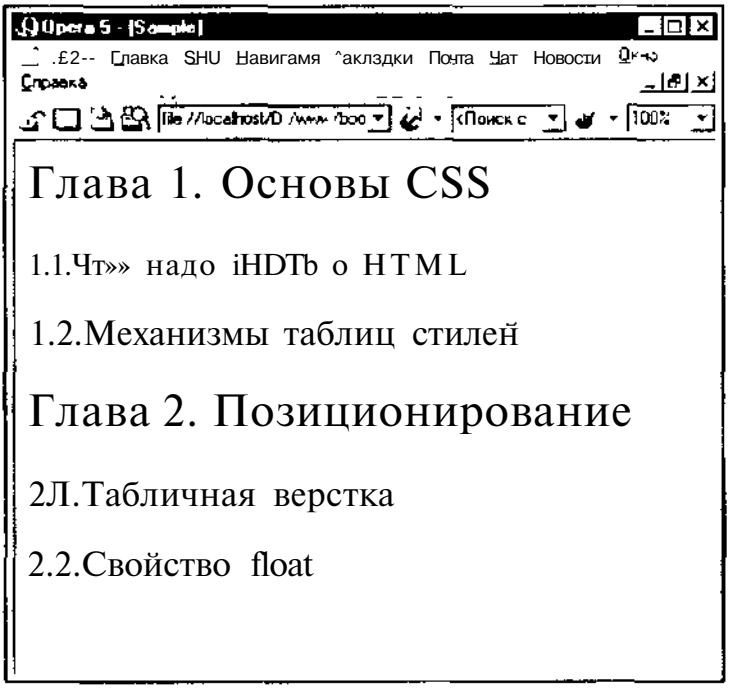


Рис. 12.4. Нумерация глав и подразделов с помощью счетчиков. Вид в браузере Opera 5 *x*

На данный момент счетчики поддерживают только браузеры компании Opera Software, гак что на практике их использовать невозможно.

Устройства с постраничной разбивкой

К таким >строис!вам относится все тот же принтер или проектор, кроме того, можно и на экране компьютера выводить документ постранично. В этом случае большой документ будет разбит на дискретное количество страниц. Конечно, целесообразность такого способа вывода весьма сомни-

тельна, но, скажем, для предварительного просмотра перед печатью он может пригодиться.

Что может понадобиться разработчикам для контроля над видом страницы? В первую очередь, это возможность изменять параметры блока на странице, а также регулировать сам процесс разбивки на страницы, т. е. запрещать или разрешать разбивку в определенных местах. В стандарте CSS-2 все это предусмотрено, так что давайте разберем возможности подробнее.

Страничный блок и *©раде*

Страничный блок представляет собой прямоугольник, состоящий из страничной области и области полей, которая окружает страничную область. В отличие от обычного структурного блока, для страничного блока нельзя устанавливать отступы и рамки, зато можно устанавливать высоту и ширину блока, а также поля.

Создание страничного блока осуществляется с помощью инструкции @раде, которая может содержать несколько объявлений. Например, блок

```
@раде (
  size: 10cm 16cm;
  margin: lcm}
```

будет иметь ширину 10 см, высоту 16 см и поля по 1 см. При этом ширина страничной области будет равна 10—1—1=8 см, а высота страничной области 16—1—1=14 см, т.е. размеры страничной области равны размерам страничного блока минус поля. Займемся свойствами.

size

Позволяет устанавливать размеры и ориентацию страничного блока. Если указывать численные значения, то первое значение задает ширину блока, а второе — высоту блока. Можно использовать как абсолютные, так и относительные величины. Однако ввиду того, что для страничного блока нельзя задавать шрифт и его размер, нельзя использовать единицы em и ex, но можно использовать проценты. В этом случае устройство вывода может само устанавливать оптимальный размер выводимой страницы.

Кроме того, есть три ключевых слова, влияюших на ориентацию страницы:

- auto — сохраняются размер и ориентация страницы, т. е. устройство вывода само подбирает нужный размер. K. примеру, если это принтер формата A4, то ширина страницы будет 21 см, а высота — 29,7 см;
- landscape — сохраняется размер, но горизонталью становится наибольшая сторона страницы, т. е. страница как бы поворачивается на 90°. Очевидно, подходит для широких фотографий;

- portrait — сохраняется размер, но горизонталью становится наименьшая сторона страницы. Подходит для фотографий портретного типа.

Например, если вам надо расположить фотографию пейзажа на странице формата A4, то можно задать такое правило:

```
@раде {
  size: landscape;
  margin: 10%)
```

Возможны ситуации, когда страничный блок будет превышать размеры страницы устройства вывода. В этом случае браузер должен либо повернуть страницу на 90°, если это позволит уместить блок на странице, либо пропорционально уменьшить размеры блока.

Псевдоклассы *:left, .right* и *'.first*

Бывают ситуации, когда надо распечатанные страницы сшивать наподобие книжных. Тогда левая и правая страницы должны отличаться. У левой страницы правое поле должно быть больше, а у правой страницы, наоборот, должно быть больше левое поле, чтобы страницы удобно было сшивать. В CSS-2 можно устанавливать различные блоки для различных видов страниц с помощью псевдоклассов :ieft и : right. Вот пример использования исевдоклассов:

```
@page:left (
  size: 19cm 25cm;
  margin-left: 2.5cm;
  margin-right: 4cm}

@page:right {
  size: 19cm 25cm;
  margin-left: 4 cm;
  margin-right: 2.5cm}
```

Кроме того, можно устанавливать параметры отдельно для первой страницы. Например, установить большое верхнее поле, чтобы блок выводился ближе к центру страницы, тогда получится титульный лист. Делается это так:

```
@page:first {
  size: 19cm 25cm;
  margin-top: 9cm}
```

Вообще устана&тивать верхний отступ надо с большой аккуратностью, потому что в случае, если установить его. скажем, в 10 000 in. то принтер, прежде чем напечатать страничный блок, выведет несколько десятков пустых страниц. Естественно, это абсолютно бесполезно и для вас, и для принтера.

раде

С помощью этого свойства можно давать название страничным блокам и потом обращаться к ним Например, если в документе встречаются широкие таблицы (то есть с большим числом столбцов), то лучше их выводить на печать таким образом, чтобы страница была повернута на 90°. Тогда вероятность того, что таблица просто не влезет на страницу, значительно уменьшится. Для всех широких таблиц можно написать класс wide:

```
TABLE.wide •
  page: fortable}
```

Страничный блок будет называться fortable, а обращаться к нему можно с помощью инструкции ©page, после которой через пробел указывается имя блока:

```
@page fottable (
  size: landscape)
```

В этом случае, если в коде у нас встретится такая таблица, то она будет распечатана на отдельном листе бумаги. Например:

```
<BODY>
  <P>Здесь идет текст, который будет выведен на первой страмице</P>
  <TABLE CLASS="wide">
    <TR>
      <TP>Эта таблица будет выведена на отдельной странице</T0>
    </TR>
  </TABLE>
</B0DY>
```

Разрыв страницы

В CSS-2 есть способ задавать, в каких местах браузер может или должен вставлять разрыв страницы. Для этого существует несколько свойств. Надо сказать, что эти свойства поддерживаются браузерами Internet Explorer 5+, так что ими можно пользоваться и без директивы @раде.

О page-break-before — позволяет устанавливать разрыв страницы перед элементом. Имеет следующие значения:

- auto — значение по умолчанию. Браузер руководствуется своим собственным алгоритмом;
- always — всегда устанавливается разрыв перед элементом;
- avoid — указывает браузеру на то, что перед элементом лучше всего разрыв не ставить;

- left — указывает браузеру на то, что надо вставить 1 или 2 разрыва подряд таким образом, чтобы следующая страница была левой (число разрывов зависит от того, какой является текущая страница: правой или левой);
- right — указывает браузеру на то. что надо вставить 1 или 2 разрыва подряд таким образом, чтобы следующая страница была правой (число разрывов зависит от того, какой является текущая страница: левой или правой).

Применяется данное свойство достаточно естественным образом. Например, вам надо в документе каждый новый раздел начинать печатать с новой страницы. Допустим, каждый раздел начинается элементом <H1>. Тогда надо задать, чтобы разрыв страницы стоял перед всеми элементами <H1>. Делается это так:

```
Hi {
  page-break-before: always
```

Как видите, ничего сложного нет.

П **page-brealc-after** — совершенно аналогично СВОЙСТВУ page-break-before, но только устанавливает разрыв страницы после элемента. Значения также совершенно аналогичные.

О page-break-inside — часто возникают ситуации, когда желательно, чтобы элемент распечатался на одной странице. Например, если таблица будет распечатана на двух страницах, то ее восприятие значительно ухудшится. В этом случае можно воспользоваться свойством page-break-inside. Оно может принимать два значения:

- auto — обозначает, что можно вставлять разрыв страницы внутри элемента;
- avoid — обозначает, что разрыв внутри элемента желательно не ставить

Конечно, если таблица очень большая, то она неизбежно булет разбита на несколько страниц, именно по этой причине формулировки достаточно мягкие. К сожалению, мы не можем быть уверены па 100%, чго браузер не вставит разрыв внутри таблицы, мы лишь смеем надеяться.

Если говорить о таблицах, то избежать переноса на другую страницу можно таким способом:

```
TABLE I
  page-break-inside: avoid'
```

Есть достаточно подробное описание правил, согласно которым браузер должен расставлять разрывы страниц. Однако в них нет ничего интересного, и знать их нужно будет только в том случае, когда дело дойдет до применения инструкции @page на практике. Пока ни один браузер не поддерживает данную инструкцию, так что особого смысла рассматривать эти правила нет.

Отдаленное будущее

К отдаленному будущему можно смело отнести стандарт CSS-3. Сейчас он находится на стадии разработки, так что консорциум W3C его рекомендует (то есть окончательно утвердит в качестве официального стандарта) не ранее конца 2002 начала 2003 года. Его внедрения (хотя бы частичного) производителями браузеров надо ждать не ранее 2004 года. Таким образом, если более-менее полное внедрение произойдет, то случится это где-то в году 2005. Причем такой прогноз можно назвать сверхоптимистичным, если принимать во внимание темпы внедрения стандартов CSS-I и CSS-2, а также все увеличивающуюся роль XML, следствием чего просто обязано стать >величение роли XSL.

Зачем же вообще заглядывать так далеко? Имеет ли это смысл на данный момент? Сразу скажу, что вопрос достаточно простой. Все зависит от того, чего вы хотите от каскадных таблиц стилей. Если они для вас всего лишь приятное дополнение к языку HTML, которое надо немного знать, то столь далекое будущее вас не должно интересовать. Если же для вас каскадные таблицы стилей это мощный инструмент, которым вы умеете пользоваться и любите его за относительную простоту и еще большую относительную логичность, то вам просто необходимо знать о том, что же будет дальше. Знание прошлого и знание будущего CSS позволит составить общее мнение о нем и понять, куда же движется прогресс в области верстки в целом:

- *О* вы сможете осознать многие существующие недостатки, которые сложно выявить, не зная направления прогресса;
- *П* вы сможете определить меру контроля за страницей на всех этапах истории верстки;
- *П* вы сможете взглянуть на верстку масштабно.

В конце концов, вы научитесь легче замечать ускользающие до этого мелкие детали, без понимания которых невозможно стать настоящим профессионалом.

Итак, экскурс в стандарт CSS-3. Самым знаковым отличием стандарта CSS-3 от стандарта CSS-2 я&пяется так называемая модуляризация, т. е каскадные таблицы стилей будут разбиты на отдельные модули. Это делается для упрощения внедрения поддержки стандарта CSS-3. т к. можно *постепенно*

вводить поддержку отдельных модулей. Развитие стандарта тоже значительно упрощается. Это видно из простого примера. Допустим, разработчикам самым важным кажется внедрение новых типов селекторов. Для осуществления этого не нужно ждать полной переработки стандарта CSS, достаточно переработать только один модуль, который отвечает за селекторы, и внести некоторые исправления в сопряженные модули, т. е. времени на улучшение существующего стандарта нужно значительно меньше, как и времени на его внедрение.

Кроме того, обеспечивается *дифференциация поддержки CSS.* К примеру, голосоиым браузерам ни к чему визуальная модель форматирования, а визуальным браузерам нет смысла поддерживать голосовые стили.

Селекторы

Ввиду того, что селекторы очень важны, разработчики стандарта CSS-3 уделили им максимум внимания. Новых селекторов очень много, так что все их рассматривать не будем, ограничимся самыми интересными и многообещающими.

Селекторы по атрибутам в CSS-3

Появились в спецификации CSS-2, а сейчас получили дальнейшее развитие. Реализованы очень нужные селекторы по части содержимого атрибута.

- E[attr^="begin"] — позволяет применять стили к элементу <E>, .который имеет атрибут at.tr и значение в нем *начинается* со строки begin. Что это дает? В первую очередь позволяет совершенно легко и просто реализовать выделение внешних ссылок. Помните, в спецификации CSS-2 для этого мы нспользов&ти выбор по наличию атрибута TITLE у элемента <A>? Сейчас же все элементарно. Любая внешняя ссылка начинается с nttp, так что код для выделения внешних ссылок будет таким:

```
  color: #00F}
AtHREF^="http"] {
  color: #FO0}
. . .
<P>Внешняя ссылка на сайт <A
HREF="http://www.artel.by">www.artel.by</A> будет выведена красным
цветом. А внутренняя <A HREF=Vindex, html ">ссылка на главную страницу</A> будет выведена синим</P>
```

Как видите, наличие атрибута TITLE уже не играет никакой роли, поэтому можно его прописывать и для внутренних ссылок.

D E[attr$="end"] — позволяет применять стили к элементу <E>, который имеет атрибут attr и значение в нем *оканчивается* на end. **Если у нас** есть в коде элемент <IMG> С атрибутом SRC:

<IMG SRC="i/picture.gif" WIDTH="200" HEIGHT="120" ALT="">

то можно применить стили ко всем элементам <IKG>, которые подключают к документу файлы в формате GIF:

```
IMG[SRC$="gif"]  {
  padding: 1px}
```

- E[attr*="middie"] — позволяет применять стили к элементу <E>, который имеет атрибут attr и значение в нем *содержит* сроку middle. Признаться, этот селектор выглядит достаточно бесполезным, однако время покажет.

Псевдоклассы

Появилось несколько полезных псевдоклассов.

- :target

  Кроме внешних и внутренних ссылок есть еше и якоря. Как вы помните, якорь — это ссылка на контент, который находится на текущей странице. Ссылка на якорь формируется с участием значка # и имени якоря. С помощью якорей очень удобно делать оглавление для больших документов, если они не разбиты на несколько страниц. Теоретически, якоря тоже неплохо бы было выделять. Именно для этого и предназначен псевдокласс : target.

  Однако если вы уже используете цветовое выделение для внутренних и внешних ссылок, то еще одно выделение для якорей может окончательно запутать пользователя. Поэтому можно до или после якоря ставить значок, который и будет обозначать якорь. Делается это так:

```
A:target::after  {
  content:  url(target-img.gif)}
```

  Обратите внимание, что в CSS-3 псевдоэлементы и пссвдоклассы применяются в селекторах с разным синтаксисом. Если раньше и перед псевдоэлментом, и перед псевдоклассом ставилось одно двоеточие, то сейчас перед псевдоклассом ставится одно двоеточие (A : hover), а перед псевдоэлементом Два ДВОЕТОЧИЯ (P.end: :before).

- **:enabled, :disabled И :checked**

  Данные псевдоклассы предназначены для контроля над интерфейсом странички. Появилась возможность применять стили к активным и неактивным элементам, а также к выбранным элементам типа checkbox. Это

позволит сделать формы более интерактивными и удобными для пользователя.

O :contains

Чрезвычайно интересный псевдокласс, который позволяет применять стили к элементам, содержащим заданную строку. Например, правило:

```
P:contains("ess")  |
  background-color: yellow}
```

выделит желтым фоном все абзацы, в которых содержится слово "ess". Таким образом, используя DOM, можно очень просто реализовать выделение абзацев, в которых найдено искомое слово или словосочетание, т. е. можно будет легко находить ключевые слова на текущей странице.

D :not

Позволяет применять стили к элементам, за исключением указанных. Например:

```
*:not(A) {
  color: #000}
```

установит черный цвет шрифта для всех элементов, кроме элемента <A>. А правило

```
BUTTON:not([DISABLED]) {
  background-color: #C00}
```

установит красный цвет фона для всех элементов <BUTTGN>, за исключением тех. где есть атрибут DISABLED, Т. е. неактивных.

Псевдоэлемент *'.'.selection*

Позволяет применять стили к части документа, которая выделена пользователем (например, с помощью курсора мыши). Так можно, например, устанавливать фон для выделенного текста и т. п.

```
?::selection {
  background-color: yellow}
```

Пользовательский интерфейс

В CSS-3 значительно переработан контроль над пользовательским интерфейсом. Работа над этим модулем еще не закончена, однако основные изменения уже можно перечислить.

П Появилась возможность с помощью CSS делать элементы форм, такие как <TEXTAREA>, <INPUT> и др. Таким способом из элемента <DIV> можно сделать поле для ввода текста.

- Новые псевдоклассы и псевдоэлементы. С ними мы уже познакомились в разделе о новых селекторах.
- Новые виды курсоров и цветов на странице.
- Появился механизм, который позволяет изменять размеры элемента.
- Появилась возможность ясно показывать пользователю, с какими элементами он может взаимодействовать.

А сейчас более детально рассмотрим некоторые свойства и возможности.

Курсоры

Как вы уже знаете, вид курсора для элемента устанавливается с помощью свойства curscr. Новые виды курсоров следующие:

- copy — показывает, что можно произвести операцию копирования. Обычно отображается стрелкой с маленьким значком "+";
- context-menu — показывает, что для данного элемента можно вызвать контекстное меню. Это действительно очень удобно, потому что реализовывать контекстные меню можно уже сейчас;
- grab — показывает, что элемент может быть перетащен в другое место. Обычно отображается открытой рукой. Вообще, технологию drag-and-diop уже можно реализовать средствами DHTML, хотя это и не очень просто. Так что тоже полезный вид курсора.

Остальные виды курсоров менее интересны, так что не станем на них останавливаться.

Изменение размера элемента

Есть очень любопытное свойство, которое позволяет изменять размер элемента. Называется оно resizer. Основное применение — изменение области просмотра блока. Например, у нас есть текстовый блок, который содержит новость. Допустим, весь текст не виден сразу, т. е его приходится скроллировать. С помощью свойства resizer можно позволить пользователю изменять размеры данного блока, чтобы вся новость стала видна Свойство может принимать следующие значения:

- both — пользователь может изменять и ширину, и высоту блока;
- horizontal — пользователь может изменять только ширину блока;
- vertical — пользователь может изменять только высоту блока.

Media queries

Это понятие невероятно сложно перевести на русский язык. Звучать будет примерно так: "Запросы для устройств просмотра". Если подробнее, то в

CSS-3, возможно, будет реализован интересный механизм, который позволит формировать запросы и достаточно детально определять специфичность устройства вывода. Например, можно будет делать различные таблицы стилей для различных экранных разрешений и подключать их с помощью CSS. Для этого предназначены специальные выражения. В следующем примере стили будут применены только в том случае, если у пользователя установлено разрешение 640x480.

```
5media screen and (min-width: 600px) and (max-width: 700px) {
  #main ;
    width: 500px}
  #bottom {
    height: 400px}
f
```

Аналогичным образом можно подключать определенные таблицы стилей:

```
<LINK REL="stylesheet" MEDIA^"screen and (min-width: 6OOpx) and (max-
width: 700px)" HREF-"smallscreen,css">
```

Существует набор свойств, которые могут быть применены к устройству вызода. К ним относятся цвета, разрешение, размеры, пропорции. Таким образом, можно применять разные таблицы стилей к принтерам разных форматов. Можно применять разные стили к цветным мониторам и монохромным устройствам.

Это обеспечивает гибкость CSS, хотя требует увеличения времени на разработку сайта. Однако это уже дело разработчиков, пользоваться таким механизмом или нет.

CSS-сценарии

Что такое сценарий?

> ***Определение***
>
> Сценарий— это небольшой скрипт на языке JavaScript или VBScript, который выполняется после какого-либо события.

Например, наведение мышки на элемент, загрузка документа, движение мышки, щелчок мышкой. Главная цель сценариев — оживить страницу, сделать ее более интерактивной и удобной. Самым распространенным сценарием является так называемое "перекатывание", когда при наведении на картинку она заменяется другой картинкой, на основе чего можно делать простейшие интерактивные меню, в которых текущий пункт каким-то образом выделен.

Вообще-то сами кеб-разработчики не пользуются термином "сценарий", вместо этого они юворят "скрипт". Я не будут отступать от традиций. Игак, в CSS-3, возможно, появится модуль, который будет позволять писать скрипты, т. е. реагировать на действия пользователя.

Необходимость такого модуля достаточно спорна. С одной стороны, писать простейшие скрипты *возможно* станет проще, с другой стороны, для сложных скриптов все равно придется пользоваться языками типа JavaScript.

Дли тех, кто знаком со скриптовым языком, привел} пример, опубликованный на сайте консорциума W3C **(http://www.w3.org/TR/I999AVD-becss-19990804).** Эффект перекатывания будет создаваться примерно так:

```
<IMG ClASS="rollover" SRO"yes .gif"
     OVERSRC="hiliteyes.gif"
     0OTSRC="ye3.gif" STATUS="press yes!">
<IMG CLASS-"rollovcr" SRO"no.gif"
     OVERSRO"hiliteno. gif"
     OUTSRO"no. gif"
     STATUS-"press no'">
```

Как видите, здесь есть два новых атрибута, в которых указывается основное изображение и изображение, появляющееся при наведении курсора. Сам CSS-скршп будет находиться в селекторе . rollover и выглядеть так'

```
.rollover \
  border: lpx solid tlOOF;
  onmouseover: "this.src = this.getAttribute('oversrc');
       this.style.borderColor = '#F00';
       stzatusText.data = this.getAttributei'status');"
  onmouseout: "this.src = this.getAttribute('outsrc');
       this.style.borderColor = '#00F';
       statusText.data = '';"}
```

Как видим, для описания событий используются два новых свойства каскадных таблиц стилей: onmouseover и or.mousecut. А затем идет стандартный DOM. Вообще-то этот скрипт ничуть не проще того, который можно написать с помощью JavaScript.

На мой взгляд, это расширение не приживется, хотя оно достаточно интересное, но на сегодняшний день не может дать ничего нового. Да, возможно, некоторое упрощение кодирования, но не более того, т. е. качественно нового уровня не будет. Веб-разработчики просто не захотят обучаться новой технологии по причине ее весьма сомнительной простоты.

Вообще, над CSS-сценариями наао очень хорошо поработать и вывести их на новый уровень, только в этом случае их могут внедрить разработчики браузеров.

Многоколоночная разметка

Возьмите любую газету и вы увидите, что текст в ней разбит на несколько колонок. Это н называется многоколоночной разметкой. В настоящее время ее можно реализовать с помощью таблиц, но с некоторыми недостатками. Именно для того, чтобы недостатки таблиц устранить, разработчики стандарта CSS-3 включили в него модуль, который и отвечает за многоколоночную разметку.

Какие же недостатки есть у таблиц?

0 Текст не перетекает из колонки в колонку, как в газетной статье. Следовательно, мы не можем взять и поместить произвольное количество контента в таблицу, чтобы браузер сам распределил его по ячейкам. Нам приходится в каждую ячейку помешать контент отдельно, что отнимает дополнительное время, и все равно не обеспечивает плавного переноса контента из колонки в колонку.

П Нельзя менять число колонок в зависимости от размеров устройства вывода. При использовании таблиц всегда жестко и однозначно задается число колонок, тогда как неплохо было бы задавать только ширину одной колонки, и сделать так. чтобы браузер сам разбиват страницу на столько колонок, сколько помешается на экране.

Это основные недостатки. Есть еше громоздкость и нелогичность табличного кода. Как вы догадываетесь, в CSS-3 все эти недостатки учли и сделали модуль, который их лишен.

Для контроля над колонками ввели три группы новых свойств.

1 Установка ширины и числа колонок.

- column-count — число колонок (можно указывать только целое число);
- column-width — ширина отдельной колонки:
- column-min-width — минимальная ширина отдельной колонки;
- column-width-policy — это свойство показывает браузеру, каким образом интерпретировать значение свойства column-width. Значение `flexible` указывает на то, что ширина колонок может быть увеличена для заполнения всего свободного пространства. Значение `strict` указывает на то, что ширина колонок должна точно соответствовать той, что указана в свойстве column-width;

3 Установка расстояний между колонками.

- column-gap — отступы между колонками;

- coiumn-rule-coior — цвет рамки между колонками;
- column-rule-style — стиль рамки между колонками (solid, dotted и т. п.);
- coiumr.-ruie-width — ширина рамки между колонками;
- column-rule — сокращенная форма записи. Формат практически такой же, как у свойства border;

П Установка слияния колонок.

- column-span — количество колонок, которые должны быть соединены в одну. Чем-то похож на атрибут COLSPAN элемента <TD>.

Следующий код создает документ с тремя колонками и заголовком, который будет находиться в объединенной ячейке всех трех колонок.

```
DIV.article {
  column-count: 3;
  column-rule: solid #000 O.lem;
  column-gap: 0.5em}
DIV.article HI f
  column-span: all}
. . .
<DIV CLASS="article">
  <H1>Заголовок статьи.</H1>
  <P>OСычный текст, которого достаточно много</P>
</DIV>
```

Схематично документ в итоге будет выглядеть так, как показано на рис. 12.5.

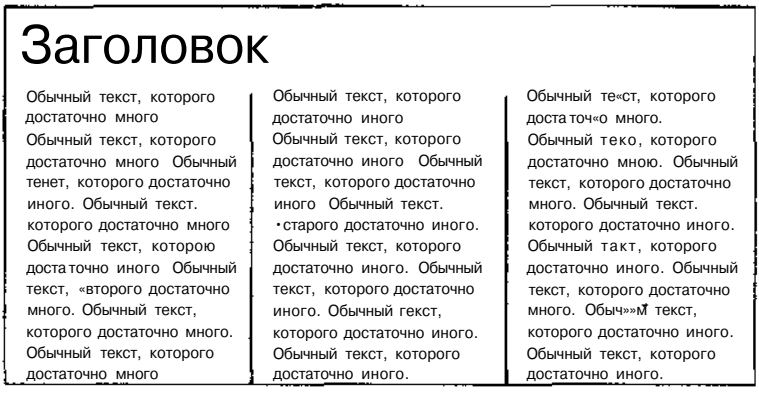


Рис. 12.5. Пример многоколоночной разметки текста

Как видите, создавать подобие газетной разметки будет очень просто. Но важно понимать, что веб как среда значительно отличается от печатных изданий. Поэтому комфортность чтения текста в нескольких колонках на экране монитора вызывает сомнения.

Тенденции развития

Сейчас можно проанализировать развитие стандарта CSS и выделить основные тенденции.

Вначале каскадные таблицы стилей обеспечивали контроль в основном над визуальным представлением шрифта и текста. Селекторы были простейшими, контроль над блоками был минимальный. В то время табличная верстка только зарождалась. Еше не было видно множества ее недостатков и слабых мест, потому что все обращали внимание в основном на достоинства. Это естественно, поскольку табличная верстка обеспечивала небывалый контроль над визуальным представлением страницы. Она еще была слабо исследована, так что мало кто заглядывал дальше. От стилей в то время требовался именно контроль над шрифтом и текстом, а все остальное было менее важно и не востребовано.

С течением времени табличная верстка становилась все привычнее, раскрывались ее недостатки, включая громоздкость кода и трудности по его редактированию. Прогресс неумолимо двигался дальше, что привело к появлению верстки блоками. Именно в этом заключается основное и принципиальное отличие стандарта CSS-2 от CSS-1. Появились детально проработанные модели форматирования, такие как абсолютное позиционирование и плавающая модель Появилась возможность жесткого контроля над блоками. Это стало новым этапом верстки. Табличная верстка постепенно будет заменяться версткой блоками.

Вторым важным отличием является развитие доступности документов (accessibility). Появились звуковые таблицы стилей, стили для устройств с постраничной разбивкой, а также поддержка различных типов устройств.

Кроме того, значительно развились селекторы.

Если рассматривать рабочие материалы по CSS-3, то можно сказать, что наибольшее развитие получили селекторы, т. е. наблюдается устойчивая тенденция роста способов доступа к элементу документа.

Все остальное (медиа-запросы, многоколоночная разметка и пр.) еще очень недоработано и особой выгоды не дает, но все же можно отметить усилия разработчиков по увеличению доступности.

Так что, если рассматривать каскадные таблицы стилей вообще, то основные тенденции развития следующие:

- Селекторы

О Доступность документов

- Размещение информации, т. е позиционирование

Очевидно, это самые важные свойства таблиц стилей. Так что неудивительно, что именно на их развитие делается упор.

Глава 13

Верстаем сайт

Все полученные знания могут вам пригодиться только в одном случае, — когда вы начнете верстать сайт Неважно, какой именно сайг это будет, важен сам факт Конечно, вы б\детс начинать с простых страничек и почти наверняка с жесткой верстки таблицами фиксированной ширины. По ходу книги уже был достаточно подробно описан такой тип верстки на примере нирту&1ьного (то есть несуществующего) сайта компании Энигма Как вы сами убедились, здесь нет ничего сложного.

Самым интересным является процесс мышления HTML-верстальщика, когда он смотрит на шаблон, встречается с какоѝм-либо трудностью, или когда он пытается оптимизировав код и сделать все как можно лучше и быстрее

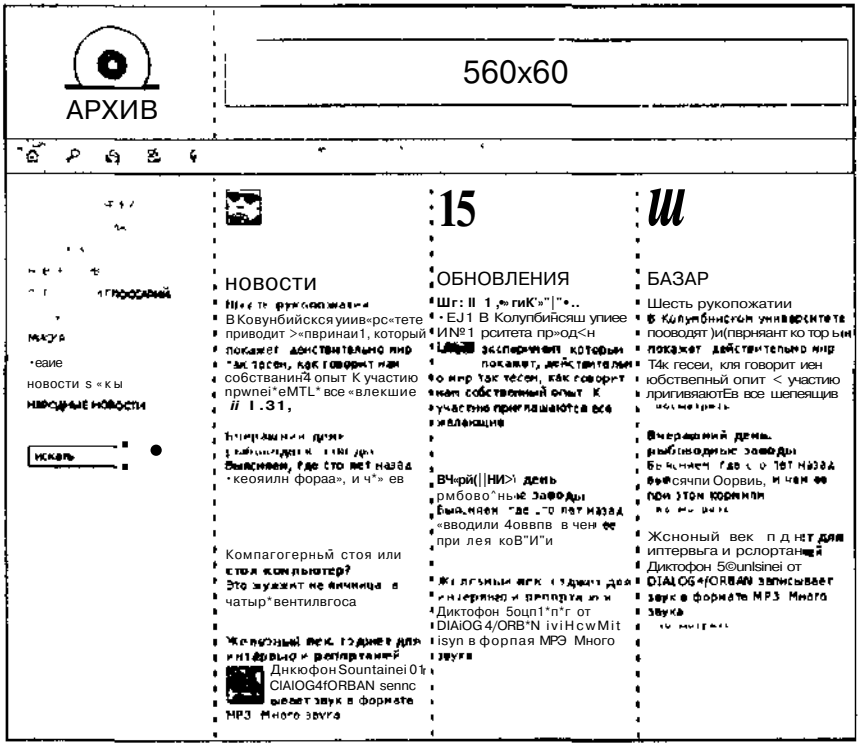


Рис. 13.1. Макет сайта

Как я к обещал, сейчас мы будем верстать достаточно сложный сайт. На рис. 13.1 он представлен в черно-белой гамме, так что все богатство красок и красота дизайна от вас ускользает, однако структуру вы оценить можете. Кстати говоря, к HTML-всрсталыиикам обычно попадает макет в формате PSD, т. к. подавляющее большинство веб-дизайнеров для создания макета пользуются графическим пакетом Photoshop.

Однако вернемся к структуре. Она примерно следующая. Вверху идет шапка, которая состоит из двух блоков:

- логотип и баннер размером 560x60;

G зеленая полоска с пятью пиктограммами, названием сайта и текущей датой.

На этом шапка заканчивается, и начинается основное содержимое. Оно сосюит из четырех блоков. Самый левый — это служебный блок меню, а три остальных — это столбцы под контент. Условно макет можно разбить на шесть больших блоков, как это показано на рис. 13.2

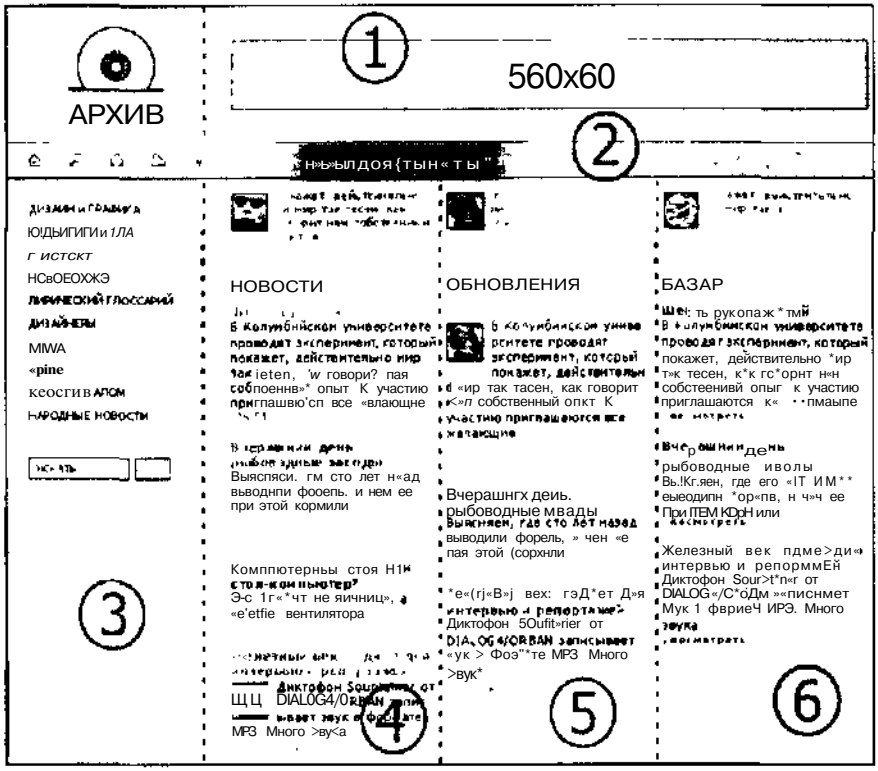


Рис. 13.2. Основные блоки, на которые можно разбить макет сайта

Принципы верстки

До того, как двигаться дальше, определимся с принципами верстки.

D Ресурс у нас информационный, т. е. важен каждый байт. По этой причине будем оптимизировать все, что возможно[1]

- не ставить кавычки там. где это необязательно;
- беспощадно убирать из кода все лишние атрибуты;
- все изображения поместим в каталог с максимально коротким названием /:
- не будем вводить ненужных элементов.

П Код будем оптимизировать под браузеры Internet Explorer 5+, Opera 5+, Netscape 6.2. Вы, наверное, сразу спросите, почему не будет поддержки браузеров Netscape Navigator *4.x* и Internet Explorer 4-х? Причин, вообще говоря, несколько. Во-первых, для них невозможно сверстать такой макет без использования таблиц, потому что CSS они поддерживают очень слабо. Во-вторых, этими браузерами в совокупности пользуется не более 7% аудитории Интернета и около 3% целевой аудитории данного сайта, тем более что этот процент постоянно и неуклонно уменьшается. Можно пожертвовать этими тремя процентами во благо новым технологиям и всем преимуществам, которые дает верстка CSS-блоками.

Остальными же браузерами и вовсе пользуется менее одного процента пользователей, так что специально оптимизировать страницы для них совершенно невыгодно.

Выбор схемы позиционирования

Разбили мы макет на блоки и думаем, как к нему подступиться и с чего, собственно, начать. Если бы мы верстали сайт с помощью таблиц, то начали бы верстать особо не задумываясь. Однако мы твердо решили уйти от табличной верстки и пользоваться только CSS. Значит, нам надо выбрать схему позиционирования. Вспомним, какие схемы вообще есть. Понятно, что нормальный поток никак не подойдет, потому что большинство блоков располагается на одной горизонтали. Относительное позиционирование тоже можно отбросить, потому что мы, во-первых, убедились, насколько это непредсказуемая вещь, а во-вторых, нам неизвестна высота некоторых блоков, т. е. их высота есть величина изменяемая и зависит от контента Остались две схемы: абсолютное позиционирование и плавающая модель. Вообще-то абсолютное позиционирование гораздо лучше подходит для жесткой верстки (не наш случай), но и для "резиновой" верстки его теоретически использовать можно Давайте подумаем, почему. Ятя лого обратимся к определениям. Верстка называется "резиновой", если размеры страницы изменяются

при изменении размеров окна браузера. Но мы ведь можем это обеспечить при абсолютном позиционировании. Для этого надо указывать ширины блоков и величины смешения в процентах. Например, создадим три блока, которые будут образовывать три колонки. Каждый блок будет обладать рамкой и занимать по ширине одну треть экрана. Высота всех блоков будет 50% от высоты окна браузера. Итак, код первого блока:

```
#first {
  width: 33%;
  height: 50%;
  position: absolute;
  left: 0%;
  top: 0%)
```

Включаем для него схему абсолютного позиционирования и задаем нулевое смещение от левого и верхнего краев. Вообще говоря, нулевое смещение является значением по умолчанию, но для наглядности мы его укажем. Код второго блока:

```
#second {
  width: 33%;
  height: 50%;
  position: absolute;
  left: 33%;
  top: 0%)
```

Как видите, мы задали смещение как раз на ширину первого блока, так что второй блок будет располагаться точно после первого блока и на одной с ним горизонтали, потому что смещения от верхнего края у блоков одинаковы и равны нулю. Код третьего блока:

```
tfthird  {
  width: 33%;
  height: 50%;
  position: absolute;
  left: 66%;
  top: 0%)
```

А для третьего блока мы задаем смещение, равное суммарной ширине первого и второго блоков, т. е. 33+33=66%. Поэтому третий блок будет располагаться непосредственно после второго на одной с ним горизонтали. Код самой страницы будет таким:

```
<BODY>
  <DIV ID="first">
```

```
    Первая колонка
  </DIV>
  <DIV ID="second">
    Вторая колонка
  </DIV>
  <DIV iD-"third">
    Третья колонка
  </DIV>
</BODY>
```

Получается, что при изменении ширины окна браузера размеры блоков по горизонтали будут изменяться, т. е. колонки будут *"резиновыми"*. Однако у такого подхода к "резиновой" верстке есть определенные недостатки. Дело в том, что очень сложно точно позиционировать блоки, если смещения задаются в процентах. На экране все равно не существует такой величины как процент, а все размеры переводятся в пикселы. Поэтому 66% в некоторых случаях может быть меньше, чем 33%+33%, как это ни странно звучит. Как это может быть? Давайте посчитаем. Например, ширина окна браузера 793 пиксела. Тогда 33% это 793x0,33=261.69. Но пикселы есть исключительно целые числа, так что браузер округлит это число до 262 пикселов. В этом случае 33%+33% будет равняться 262+262=524 пиксела. А если посчитать напрямую, то 66% это 793x0,66=523.38, т. е. с учетом округления 523 пиксела! В нашем коде ширина первых двух блоков как раз и есть 33%+33%, а величина смешения третьего блока 66%, т. е. он будет накладываться на второй блок из-за разницы в один пиксел! Вот к чему приводит задание величины смешения в процентах.

Вообще в некоторых случаях можно и для "резиновой" верстки пользоваться абсолютным позиционированием, но обычно это сложнее и труднее. Мы не пионеры и трудностей сами себе создавать не любим, поэтому будем пользоваться плавающей моделью, которая как раз и предназначается для "резинового" дизайна.

Начало верстки: шапка

Со схемой мы определились, так что нужно решить, что делать дальше. Собственно, дальше надо приступать к верстке, но следует решить, с чего начинать. Вариантов, надо сказать, мало. Практически всегда начинают верстать сайт сверху, т. е. с шапки. Это правильно — надо быть последовательным. Однако прежде чем начать верстать, нужно определить, какие графические части макета нам могут принципиально понадобиться. Нам будут необходимы только те элементы, которые невозможно реализовать средст-

вами HTML и CSS. Так что давайте внимательно рассмотрим каждый блок в отдельности и решим, что мы из него возьмем в виде графики.

О *Блок 1* — содержит всего три графических элемента: логотип "Архив", баннер и пунктирную линию. Логотип нужно вырезать и сохранить в формате GIF (так как это рисованная фафика). Баннер тоже можно вырезать и сохранить. Вообще-то баннер будет вставляться в шаблон специальным кодом, и делать это будет веб-программист, но на стадии верстки это нас не касается, поэтому пока можно вставить его простой картинкой. С пунктирной линией все сложнее. Можно ее тоже вырезать и вставить на страницу с помощью тега <IMG>, а можно *попробовать* реализовать с помощью CSS (получится или нет — изначально неизвестно).

G *Блок 2* — содержит пять пиктограмм, которые можно вставить только графикой. Все надписи, идущие после пиктограмм, будут текстовыми, так что их вырезать не надо. Осталось разобраться со светло-зеленой полоской (на рисунке она серого цвета). Структура полоски следующая. Сначала идет черная полоска шириной 1 пиксел, затем светло-зеленая шириной 30 пикселов, затем снова черная шириной I пиксел, а затем серая (это тень) шириной 3 пиксела. Реализовать такую полоску можно средствами HTML и CSS. но это достаточно сложно. Гораздо проще вырезать из нее кусочек высотой 35 пикселов и шириной 1 пиксел (рис. 13.3), а затем этот кусочек вставить как фоновое изображение, повторяющееся по оси x.

Рис. 13.3. Маленькая часть полоски из блока 2, которая станет фоновым рисунком

Пока мы разобрались с двумя блоками. На этом временно остановимся и сверстаем их. Итак, для верстки двух первых блоков у нас должно быть 8 рисунков.

- Логотип "Архив" (archive.gif).

G Баннер (banner.gif)-

П Пиктофамма "Домой" (home.gif).

П Пиктофамма "Поиск" (search.gif).

- Пиктограмма "Письмо" (letter.gif).

- Пиктограмма "Редакция" (about.gif)
- Пиктограмма "Карта сайта" (sitemap.gif).

3 Фоновый рисунок (bgl.giO-

Начнем, пожатуй. Для начала зададим для всего документа DTD, чтобы браузер Internet Explorer 6.x переключился в режим совместимости со стандартами:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Кстати говоря, не факт, что нам понадобится такой режим. Ситуация следующая[1] браузер Internet Explorer 6.x в режиме обратной совместимости ведет себя практически так же, как и браузер Internet Explorer 5.x. Так что если оставить именно этот режим, то нам фактически придется адаптировать код только под три браузера, а не под четыре (как будет, если включить режим совместимости со стандартами). Тем не менее, сначала мы попытаемся сверстать страницу именно так. чтобы использовать корректную блоковую модель браузера Internet Explorer 6.x. В этом есть смысл. Во-первых, это принципиальный вопрос, поскольку корректность браузера надо поощрять по мере сил и возможностей. Во-вторых, с прицелом на будущее лучше верстать именно так, все-таки браузеры двигаются по направлению к стандартам.

Итак, вначале зададим фоновый цвет #FCF9ED и нулевые отступы для всего документа. Это делается просто: надо применить соответствующие свойства CSS к селектору <BODY>:

```
BODY i
  background-color: #FCF9£D;
  margin: Opx[
```

Кроме того, сразу установим гарнитуру шрифта и его размер для всего документа. Шрифт у нас Verdana, а размер согласно дизайну достаточно мал, ввиду большой информационной плотности первой страницы:

```
BODY {
  background-color: #FCF9ED;
  font: 0.7em Verdana, Tahoroa, sans-serif;
  margin: Opx}
```

А теперь перейдем к блокам.

Блок 1

Создадим первый блок. Он будет у нас иметь ширину 100% и называться top. Весь HTML-код будет пока очень простым:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<HTML>
<HEAD>
<TITLE>APXHB</TITLE>
  <STYLE  TYPE»"text/css">
    BODY {
      background-color:  #FCF9ED;
      margin:  Opx}
    #top {
      width:  100%;
      height:  118pxJ
  </STYLE>
</HEAD>

<BODY>
<DXV  id=top>

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

До этою момента все прекрасно и жизнь кажется безоблачной. Но это достаточно иллюзорно.

Итак, блок top должен включать в себя два блока. Первый содержит логотип, а второй — баннер. Сначала займемся логотипом. Создадим блок topleit. Ширина блока будет 25% от ширины окна браузера, и логотип должен быть центрирован внутри блока. Для центрирования воспользуемся СВОЙСТВОМ text-align!

```
#topieft {
  text-align: center;
  width: 25%>
```

Пока никаких проблем. Далее нам надо отбить логотип от верхнего края на 15 пикселов. Это легко делается с помошью объявления padding-top: ispx. Раз мы задали отступ, то надо указать высоту. Общая высота контейнера у нас 118 пикселов, значит высота контента должна быть 118-15=103 пиксела. Здесь мы совершенно случайно вспомним, что браузер Internet Explorer 5.x имеет *некорректную* блоковую модель. Некорректность заключается в том, что *высота и ширина блока вычисляются без учета отступов.* Сразу становится понятно, что для этого браузера надо задавать высоту не 103, а 118 пикселов. Пока отложим этот вопрос на некоторое время, и создадим пунктирную линию по правому краю блока.

Делается она так:

```
ttopleft {
  text-align: center;
  width: 25*;
  height: 118px;
  border-right: lpx dotted #000;
  padding-top: 15px}
```

Здесь мы сталкиваемся с первой принципиальной проблемой. Дело в том, что браузер Internet Explorer *5.x* не поддерживает стиля рамки dotted, и вместо пунктира он рисует сплошную линию. В нашем случае вместо необходимого пунктира браузер отобразит совершенно ненужную сплошную черную линию толщиной 1 пиксел. Таким образом, если включать поддержку браузера Internet Explorer 5.v, то надо искать другой способ.

Очевидно, что можно попробовать решить проблему с помощью фонового рисунка. Точки в пунктирной линии повторяются через равные отрезки. В данном случае идет темно-зеленая точка, затем четыре точки фонового цвета, затем снова темно-зеленая — так образуется пунктир. Значит можно вырезать лишь один экземпляр последовательности, т. е. одну зеленую и четыре фоновых точки, после чего задать повторение по оси y.

Саму линию таким образом мы сможем создать, но надо поместить ее строго в определенное место, а именно на 25% вправо от левого края окна браузера. Делается это с помощью свойства background-position. Мы будем привязывать фон к блоку topieft, а у него ширина задана 25%. Поэтому смешать фоновое изображение надо вправо до упора, т. е. на 100% относительно блока topieft, что и будет 25% по отношению к окну браузера. Рамку убираем, значит правило для блока topieft пока будет таким:

```
fttopleft {
  background-image: url(i/dotl gif);
  background-repeat: repeat-y;
  background-position: 100% 0%
  text-align: cent-tr;
  width: 25%;
  height: 118px;
  padding-top: 15px}
```

Объявления для фона можно записать в краткой форме, так что код немного сократится:

```
#topieft (
  background: url(i/dotl.gif) repeat-y 100% 0%;
  text-align: center;
```

```
  width: 25%;
  height: 118px;
  padding-top: 15px}
```

В HTML-код вставим изображение логотипа.

```
<BODY>
<DIV id=top>
  <DIV id«topleft>
    <IM3 SRC=i/archive.gif WIDTH=87 HEIGHT=89 A1/Г="Архив">
  </DIV>
</DIV>
</BODY>
```

И уже в этом немудреном коде найдем отличие, показанное на рис. 13.4.

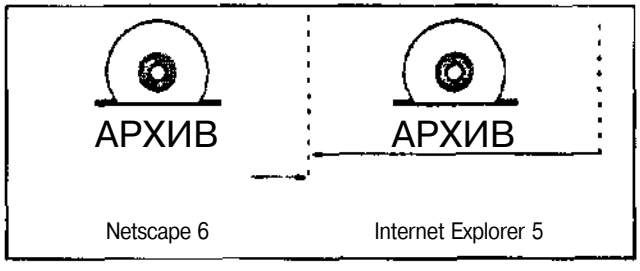


Рис. 13.4. Несовпадение высот блоков в разных браузерах

Как видите, в браузере Internet Explorer 5.v высота пунктирной линии, а, следовательно, и блока, меньше, чем в браузерах Netscape 6.v и Opera 5+. Вог и всплыла та самая разница в 15 пикселов. Будем ошибку исправлять.

Для Internet Explorer *5.x* надо задать высоту блока 118 пикселов, потому что он включает в это значение и отступы, а для всех остальных браузеров надо указать 103 пиксела. Делается это с помощью метода Целика, и код для блока станет таким:

```
#topleft {
  background: urlfi/dotl.gif) repeat-y 100% 0%;
  text-align: center;
  width: 25%;
  height: П3px;
  padding-top: 15px
  voice-family: "\"}\"";
  voice-family: inherit;
```

```
  height: 118px•
HTML>BODY ftopleft f
  height: I03px)
```

После этого проблема исчезает. Кстати, обращаю ваше внимание на одну особенность. Ширина блоков у нас относительная, тогда как высота задается жестко. Такая ситуация встречается достаточно часто при верстке совершенно различных сайтов. Почему так происходит? Вообще, ответ на этот вопрос дать сложно, но причин существует несколько.

G Высота сама по себе очень дорогостоящая площадь. Чем больше важных элементов находится на первом экране (то есть их видно без скроллирования страницы), тем лучше, потому что *повышается информационная плотность.* Так как ширина фактически ограничена 750 пикселами, на ней не сэкономишь, но высота ограничена только разумными пределами, так что экономить на ней можно и часто нужно.

3 Очень сложно сверстать сайт таким образом, чтобы он пропорционально растягивался и сжимачся по обеим осям. Отчасти потому, что в CSS плохо с выравниванием по вертикали, и к огромному сожалению *нет ничего подобного* атрибуту VALIGN.

Однако вернемся к верстке сайта. Нам еше надо вставить баннер в первый блок. Как видите, баннер находится справа, т. е. как бы во второй колонке. Его можно позиционировать с помощью плавающей модели. Для этого нужно создать ешс один блок, который будет вложен в блок top и иметь тот же уровень, что и блок topleft. Назовем вновь созданный блок topright. Ширина блока top у нас 100%, ширина блока topiieft 25%. Значит, ширина блока topnght должна быть 75%.

```
#topriaht (
  width: 75%;
  float: left}
```

Кроме того, надо не забыть включить для блока topleft плавающую модель, потому что в противном случае блок topright будет находиться слева, что нам совершенно не нужно:

```
#topleft (
  - - -
  float: left}
```

Все будет хорошо, но вот баннер нам надо выровнять по вертикали, т. е. чтобы и сверху и снизу он был отбит на равное число пикселов. В HTML для вертикального выравнивания элементов в ячейках таблицы предназначен атрибут VALIGN, но в CSS ничего подобного нет. В данном случае это непринципиально, потому что высота задается жестко, поэтому можно от-

бить баннер сверху на нужное количество пикселов. В нашем случае высота баннера 60 пикселов, а высота блока 118 пикселов. Значит надо отбить баннер на (118-60)/2 – 29 пикселов.

```
#topright {
  width: 75%;
  margin-top: 29px;
  float: left)
```

Код станет таким:

```
<DIV id=top>
  <DIV id-topleft>
    <IMG SROi/archive.gif WIDTH-87 HEIGHT-89 AЬT-"Архив">
  </DIV>
  <DIV id=topnght>
    <IMG SRC=i/banner.gif WIDTN=560 HEIGHT=60 BORDER=0 ALT»"">
  </DIV>
</DIV>
</BODY>
```

Но в данном случае баннер у нас будет "прилипать" к пунктирной линии, как показано на рис. 13.5.

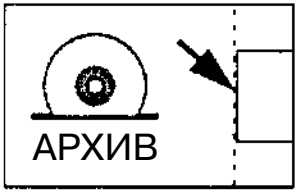


Рис. 13.5. "Прилипание" баннера к разделительной пунктирной линии

Можно задать отступ, скажем, 1%, и уменьшить ширину блока с 75% до 74%.

```
#tcpright {
  width: 74%;
  margin-top: 29px;
  padding-left: 1%;
  float: left}
```

В браузере Internet Explorer 5.v общая ширина блока будет 74%, а надо 75%, но в данном конкретном случае это не имеет значения.

Блок 2

Первый блок у нас готов, так что примемся за второй. У нас должна быть зеленая полоска, растянутая на весь экран. Слева на ней располагается блок пиктограмм, затем название сайта, а справа — текущая дата.

Сперва займемся самой полосой Ее высота составляет 35 пикселов, а ширина — 100?с. Создадим блок с такими размерами и назовем его tcpgreen.

```
#topgreen (
  width: 100%;
  height: 35px>
```

Мы ранее уже заготовили рисунок bgl.gif, который будет фоновым и. собственно, заполнит бюк Вот и сделаем его фоновым изображением:

```
#topgreen <
  background-image:url(i/bgl.gif);
  background-repeat; repeat-x;
  width: 100%;
  height: 35px}
```

Рисунок bgl.gif имеет высоту 35 пикселов, т. е. он соответствует высоте блока. Он будет размножаться по оси \, так что заполнит весь блок. Объявления для фона можно записать и короче:

```
#topgreen {
  background: url(i/bgl.gif) repeat-x;
  width: 100%;
  height: 35px}
```

В коде страницы добавится всего один элемент <DIV>, обозначающий блок topgreen:

```
<DIV id=top>
  <DIV id=topleft>
    <IMG SROi/archive.gif WIDTH=87 HEIGHT=89 BORDER=0 АЬТ="Архив">
  </DIV>
  <DIV id=topright>
    <IMG SRO"i/banner.gif" WIDTH="560" HEIGHT="60" BORDER="0" ALT-"">
  </DIV>
</DIV>
<DIV xd=topgreen>

</DIV>
</BODY>
```

А страница будет выглядеть так, как на рис. 13.6.

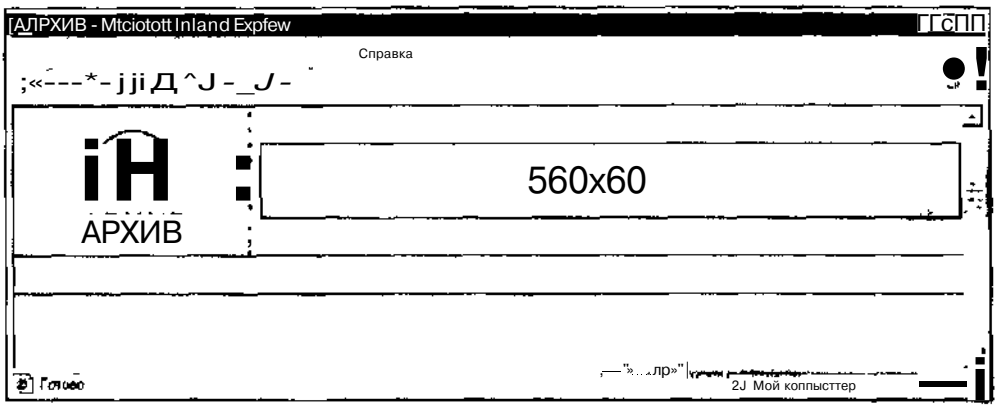


РИС. 13.6. ВИД страницы с добавленной светло-зеленой полосой

Итак, блок с полосой создан. А сейчас начинается самое интересное: нам надо сделать блоки с пиктограммами и надписями. На первый взгляд кажется, что все достаточно просто, однако это далеко не так.

Для начала бросим первый взгляд. Нам надо создать три блока. Первый должен быть шириной 25%, ширина остальных блоков достаточно произвольна, но в сумме не должна превышать 75%, так что выберем ширину второго и третьего блоков 55% и 20% соответственно. Далее эти блоки должны располагаться в один ряд, чего можно добиться с помощью плавающей модели. Блоки назовем достаточно логично:

- *O* topgreenief-ˌ— самый левый блок с пиктограммами;
- O topgreencenter — центральный блок с названием сайта;
- O topgreenright — правый блок с текущей датой.

Код у трех блоков пока будет такой:

```
#topgreenleft {
  width: 25%;
  float: left!
#topgreencenter {
  width: 55%;
  float: left)
#topgreenright (
  width: 20%;
  float: left)
```

В исходном HTML-коде добавятся три блока, которые будут вложены в блок topgreen.

```
<DIV id=top>
. . .
</DIV>
<DIV id=topgreen>
  <DIV id=topgreenleft>
    Пиктограммы
  </DIV>
  <DIV id=topgreencenter>
    6laquo;ApxMB£raquo; bndash; журнал для умных
  </DIV>
  <DIV id=topgreenright>
    12 февраля 2002
  </DIV>
</DIV>
```

Обратите внимание на то, что в блоке topgreenief= будут располагаться пиктограммы, но мы их пока не вводим, а заменяем словом "Пиктограммы". Далее в блоке topgreencenter слово "Архив" взято в кавычки-елочки, а после него стоит длинное тире, которое обозначается спецсимволом smdash.

На рис. 13.7 показано как станет выглядеть полоса в браузере.

Рис. 13.7. Вид светло-зеленой полосы с тремя вложенными блоками

Как видите, положение блоков правильное. Но надо изменить шрифт на Verdana, поменять его цвет на белый и сделать полужирным. Кроме того, *надписи не центрированы по вертикаш,* так что придется для общего блока topgreen установить верхний отступ в 8 пикселов. А это автоматически приводит к тому, что надо пользоваться методом Целика для компенсации некорректности блоковой модели в браузере Internet Explorer 5.x

```
#topgreen {
  background: url(i/bgl.gif) repeat-x;
  width: 100%;
  height: 35px;
  padding-top: 8px}
HTML>BODY #topgreen {
  height: 27px(
```

Шрифт мы установили для всего документа в селекторе <BODV>, а вот цвет и насыщенность — для каждого блока отдельно.

```
frtopgreencenter (
  color: #FFF;
  font-weight: bold;
  width: 55%;
  float: left}
#topgreenright {
  color: #FFF;
  font-weight: bold;
  width: 20'*;
  float: left)
```

После внесения этих испраатений полоса будет выглядеть так, как на рис. 13.8.

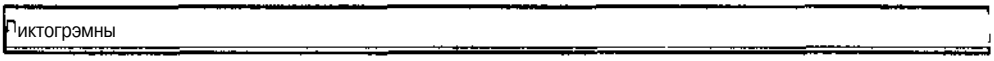


Рис. 13.S. Вид светло-зеленой полосы после изменения цвета, насыщенности и гарнитуры шрифта, а также добавления верхнего отступа в родительский элемент

Нам осталось только заменить слово "Пиктограммы" на сами пиктограммы. Вот здесь-то мы и столкнемся со страшной проблемой. Задача следующая: сделать так, чтобы пиктограммы были отделены друг от друга отступами, которые зависят от размера ширины окна, т. е. заданы в процентах. Кроме того, было бы неплохо, если бы сами пиктограммы были центрированы, т. е. располагались симметрично относительно логотипа.

Сначала просто вставим все пять картинок в код страницы, не изменяя стилей для блока:

```
<DIV id^topqreen>
  <Div id=topgreenleft>
  <IMG SRC=i/home.gif WIDTH=l2 HEIGHT=12 BORDER=0 ALT=""> <IMG
SRC=i/search.gif WIDTH=12 HEIGHT=12 BORDER=0 ALT=""> <IMG
SROi/letter.gif WIDTH=12 HEIGHT=12 BORDER=0 ALT=""> <IMG SRC=i/about.gif
WIX»TH=l2 HEIGHT=12 BORDER=0 ALT=""> <IMG SRC=i/sitemap.gif WIDTH=12
HEIGHT=12 BORDER=0 ALT="">
  </DIV>
  <DIV id=topgreencenter>
    &laquo;ApXHB&raquo; Smdash; журнал для умных
  </DIV>
  <DIV id*topgreenright>
```

```
· 12 февраля 2002
  </DIV>
</DIV>
```

В браузере пиктограммы будут расположены так, как показано на рис. 13 9.

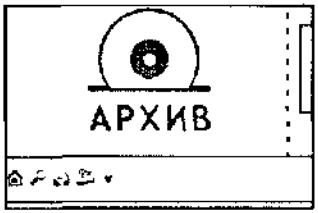


Рис. 13.9. В.[1].-: -пикограмм в браузере
Они разделены пробелами и выровнены по левому краю

Попробуем центрировать пиктограммы. Это можно сделать с помощью объявления text-align: center для блока topgreenleft:

```
tttopgreenleft {
  text-align: center;
  width: 25%;
  float: left)
```

Пиктограммы действительно выровняются по центру (рис. 13.10).

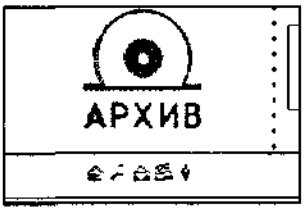


Рис. 13.10. [illegible] эммы. выровненные
по центру с [illegible] свойства text-align

Сейчас нам осталось всего лишь отбить пиктограммы друг от друга, скажем, отступами шириной 10%. Это делается с помощью свойств paadmg-ieft и padaing-nght. Задавать объявления нужно для рисунков, которые находятся внутри блока topgreenieft. Для этого прекрасно подойдет *контекстный селектор,* так что правило получится следующее:

```
fttcpgreenlcft IMG {
  padding-left: 5%;
  padding-right: 5%}
```

т. е. мы установили и левый, и правый отступы по 5%, так что в сумме между рисунками должны быть **10%-ные** отступы Вроде бы все логично и должно работать. Однако в разных браузерах картина будет различаться (рис 13.11).

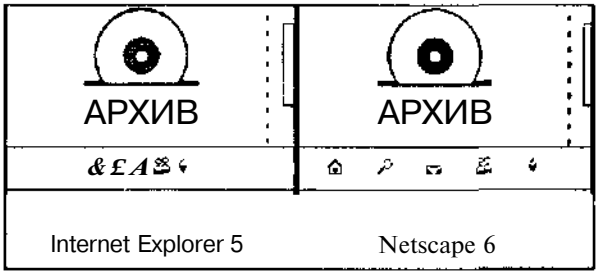


Рис. 13.11. Разница в отображении пиктограмм между браузерами Internet Explorer 5 и Netscape 6

Как видите, в браузере Netscape 6 отступы есть. А вот в браузере Internet Explorer 5 отступов нет. Все из-за того, что в пятой версии браузера фирмы Microsoft имеются проблемы с блоковой моделью применительно к изображениям, т. е. к элементам <:ко Для них нельзя устанавливать отступы. Но зато для них можно устанаштивать поля! Значит, нам надо заменить свойства padding-left И padding-right на margin-left И margin-right:

```
#topgreenleft IMG (
  margin-left: 5%;
  margxn-right: 5%}
```

У нас ширина блока topgreenieft 25% от ширины окна браузера. Ширина нолей для каждого рисунка должна вычисляться относительно этой ширины, т. е. 5% берется относительно 25%. Так и делают браузеры Internet Explorer 5.x, Netscape 6.* и Opera 5+. Но, к огромному удивлению, браузер Internet Explorer 6.v вычисляет ширину полей относительно ширины окна браузера' В данном случае это будет 5% по отношению к 100%, а не к 25%. В итоге содержимое будет искажаться страшным образом (рис. 13.12)

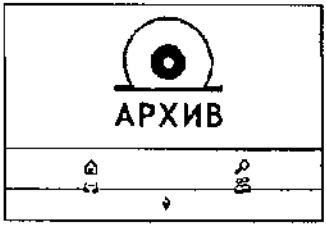


Рис. 13.12. К чему приводит неправильная трактовка ширины полей для элемента <IKG> в браузере Internet Explorer 6 *x*

Для того чтобы поля были одинаковыми во всех браузерах, надо для internet Explorer 5.v, Netscape 6.v и Opera 5+ установить их значение равным 5%, а для Internet Explorer 6.v — 1,25%. Делается это чрезвычайно хитро с помощью метода Тантек Целика. Для начала дополним правило:

```
#topgreenleft IMG f
  margin-left: 5%;
  margin-right: 5%;
  voice-family:"\"}\"".'
  voice-family:inherit;
  margin-left:1.25%;
  margin-right:1.25%}
```

После чего получится, что для Internet Explorer 5.v ширина полей будет 5%, как и надо, но для всех остальных браузеров ширина будет 1.25%. Далее надо воспользоваться селектором наследника, чтобы для браузеров Opera 5+ и Netscape *6.x* переписать неверную ширину полей. Это делается так:

```
HTML > BODY #topgreenleft IMS (
  margin-left: 5%;
  margin-right: 5%}
```

После этого наши пиктограммы наконец-то будут отображаться всеми браузерами практически правильно. Можно сказать, что с шапкой мы разобрались. Остаюсь только добавить ссылки на пиктограммы — и все. На данном этапе общий код страницы будет такой:

```
BODY f
  background-color: #FCF9ED;
  font: 0.7em Verdana, Tahoma, sans-serif;
  margin: Opx}

#top {
  width: 100%;
  height: 118px}

#topleft {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  text-align: center;
  width: 25%;
  height: 118px;
  padding-top: 15px;
  float: left;
```

```
  voice-family: "\"Л"";
  voice-family: inherit;
  height: Ю3px}

HTML>BODY #topleft {
  height: Ю3px}

#topright {
  width: 74%;
  margin-top: 29px;
  padding-left: 1%;
  float: left;}

#topgreen (
  background: url(i/bgl.gif) repeat-x;
  width: 100%;
  height: 35px;
  padding-top.: 8px}

HTML>BGDY fttopgreen {
  height: 27px}

#topgreenleft {
  text-align: center;
  width: 25%;
  float: left}

#topgreenleft IMG {
  margin-left: 5%;
  margin-right: 5%;
  voice-family: "\"}\"";
  voice-family: inherit;
  margin-left: 1.25%;
  margin-right: 1.25%}

HTML>BODY #topgreenleft IMG {
  margin-left: 5%;
  margin-right: 5%}
```

```
#topgreencenter (
  color: #FFF;
  font-weight: bold;
  width: 50%;
  padding-left: 1%;
  float: left}

#topgreenright {
  color: #FFF;
  font-weight: bold;
  width: 20%;
  float: left}
. . .
<BODY>
<DIV id=top>
  <DIV id=topleft>
    <IMG SRC=i/archive.gif [illegible] ЖКИТ-89 АЬТ="Архив">
  </DIV>
  <DIV id=topright>
    <IMG SRC=i/banner.gif [illegible] miGHT=60 BORDER[illegible] O'ALT="">
  </DIV>
</DIV>
<DIV id=topgreen>
  <DIV id=topgreenleft>
    <A HREF=#><IMG SRC=i[illegible] -
<A HREF=#><IMG SRC=i/searci
<A HREF-#><IMG SRC=i/lette[illegible]
<A HREF=#XIMG SRC-i/abO-[illegible]
HREF=#><IMG SRC=i/5itemap.[illegible]
  </DIV>
  <DIV id=topgreencenter>
    &laquo;ApxHBiraquo; iEndaaffii щ^адп для умных
  </DIV>
  <DIV id=topgreennght>
    12 февраля 2002 
  </DIV>
</DIV>
</BODY>
```

-12 HEIGHT=12 BORDER=0 ALT=""X/A>
.i HEIGHT-12 BORDER=0 ALT=""></A>
«12 HEIGHT»12 BORDER=0 ALT=""></A>
= 12 HEIGKT=12 BORDER=0 ALT=""X/A> <A
^ HEIGHT=12 BORDER-0 ALT=""></A>

И это только кол первыж **Споков,** т. е. шапки. Нам осталось закодировать четыре центральных [illegible]. Эгим и займемся.

Центральная часть

Каждая колонка предста&1яет собой отдельный блок шириной 25% от ширины окна браузера. Блоки можно отобразить в виде колонок с помощью плавающей модели. Вот и создадим четыре блока maini, main2, main3, rcain4. Пока стили для них будут совершенно одинаковыми:

```
#main1 {
  width: 25%;
  float: left)
#main2 {
  width: 25%;
  float: left}
#main3 {
  width: 25%;
  float: left)
#main4 {
  width: 25%;
  float: left}
```

В HTML-коде блоки будут идти сразу после шапки, и все они будут одного уровня, т. е. все будут являться непосредственными потомками элемента <BODY>:

```
<BODY>
  <DIV id=top>
  . . .
  </DIV>
  <DIV id=topgreen>
  . . .
  </DIV>
  <DIV id=4main1>
  </DIV>
  <DIV id=roain2>
  </DIV>
  <DIV id=main3>
  </DIV>
  <DIV id=main4>
  </DIV>
</BODY>
```

Никакого изменения внешнего вида пока не произойдет. Между блоками у нас есть разделительные пунктирные линии. Помните, как мы делали их для отделения логотипа от баннера в шапке? Вот точно так же с помощью фонового изображения мы сделаем пунктирные линии и для трех центральных колонок. Обратите внимание, что вертикальной пунктирной линии нет у последнего блока таз.п4, так что он будет отличаться от первых трех блоков:

```
#mainl {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  width: 25%;
  float: left}
#main2 {
  background: urlU/dotl.gif) repeat-y 100% 0%;
  width: 25%;
  float: left)
#main3 {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  width: 25%;
  float: left)
#main4 1
  width: 25%;
  float: left}
```

Опять же, никаких изменений в браузере не будет. Почему? Потому что блоки имеют нулевую высоту, т. е. они не заполнены контентом, и в стилях высота явно не задана. Если высота не задана, то она будет определяться имеющимся в блоке контентом. Например, если вставить рисунок высотой 120 пикселов, то высота блока будет 120 пикселов. Для решения проблем подобного рода подходящим было бы свойство min-height, но, к сожалению, оно поддерживается далеко не всеми браузерами. В частности оно не поддерживается в браузерах Internet Explorer 5лс—*6.x,* значит его использование лишено смысла.

Тогда получается, что надо задавать высоту явно с помощью свойства height Но здесь есть другая проблема. Если указать слишком большую высоту, то при маленьком количестве контента под ним может остаться неоправданно много пустого места. Если указать высоту маленькую, то один из блоков может оказаться выше всех и будет выбиваться, что несколько испортит композицию. Следовательно, нам надо подобрать высоту таким образом, чтобы весь контент ровно укладывался по высоте самого высокого блока, а это чрезвычайно нетривиальная задача. Более того, в некоторых случаях это вообще невозможно сделать. Например, если мы не знаем, сколько будет контента в блоках. Может, там будет находиться одна или две новости, а может десять новостей.

Мы пока не будем заострять внимания на этой проблеме и вернемся к ней позже, а высоту всех блоков пока укажем в 70% от высоты окна браузера:

```
#mainl {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  width: 25%;
  height: 70%;
  float: left}
#main2 {
  background: urlЦ/dOtl.gif) repeat-y 100% 0%;
  width: 25%;
  height: 70%;
  float: left}
#main3 {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  width: 25%;
  height: 70%;
  float: left}
#main4 {
  width: 25%;
  height: 70%;
  float: left)
```

А сейчас самое интересное. Браузеры Internet Explorer *5.x.* Netscape 6.x и Opera 5+ установят высоту блоков 70% и разделительные линии в них будут. А вот браузер Internet Explorer *6.x* не сделает этого, так что в нем высота блоков по-прежнему будет определяться высотой контента, которого пока нет. Так что в этом браузере разделительных линий не будет. Получается, что у браузера Internet Explorer 6..Y блоковая модель имеет достаточно серьезный баг. которого нет у браузера Internet Explorer *5.x.* Получается, что мы просто не сможем установить корректную высоту блоков, причем бороться с этим багом невозможно. Нам остается одно единственное решение — использовагь режим обратной совместимости. Тогда Internet Explorer 6..v будет вести себя практически аналогично браузеру Internet Explorer 5..v, за исключением обработки ошибок. Так, например метод Целика для него не действует. Значит, нам придется пересмотреть код и внести некоторые изменения. Что поделаешь, такова реальная жизнь HTML-версталыдика.

Во-первых, нам придется изменить объявление типа документа на такое:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

т. е. включить режим обратной совместимости в браузере Internet Explorer 6.x После этого вид уже сверстанной шапки в этом браузере изменится (рис. 13.13).

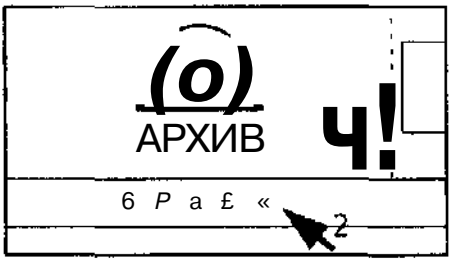


Рис. 13.13. Неточности, которые стали видны в браузере Internet Explorer 6.x после включения режима обратной совместимости

Как видите, пиктограммы опять встали рядом, т. е. между ними нет нужных промежутков (2). Кроме того, образовалась щель между пунктирной линий и зеленой полосой (I). Сначала устраним шель. Для блока topieft у нас были такие правила:

tttopleft (

```
  background: url(i/dotl.gif) repeat-y 100% 0%;
  text-align: center;
  width: 25%;
  height: 118px;
  padding-top: 15px;
  float: left;
  voice-fajnily: "\"}\"";
  voice-family: inherit;
  height: 103px}

HTML>BODY #topleft {
  height: 103px}
```

Раньше браузер Internet Explorer 6.Y *не включал* размер отступов в значение свойства height, что соответствует спецификации. Сейчас, наоборот, он включает отступы. Если раньше ему надо было указывать высоту 103 пиксела, то теперь — 118 пикселов. Получается, что для браузеров Internet Explorer 5.x—6.x надо указать высоту 118 пикселов, а для браузеров Opera 5+ и Netscape 6.x надо указать высоту 103 пиксела. Если мы заглянем в табл. 10.2 *главы Ю,* где собраны методы сокрытия стилей от браузеров, то увидим, что для этой задачи отлично подойдет селектор наследника, т. е. >. Значит, нам надо просто не использовать метод Целика, т. е. правила станут такими:

```
#topieft {
  background: url(i/dotl.gif) repeat-y 100% 0%;
  border-right: lpx dotted #000;
```

```
  text-align: center;
  width: 25%;
  height: 118px;
  padding-top: 15px}

HTML>BODY tttopleft {
  height: 103px}
```

Для селектора tttopieft мы указываем высоту 118 пикселов, что поймут все браузеры, а затем в селекторе HTML>BODY #topleft мы указываем значение высоты 103 пиксела, что поймут браузеры Opera 5+, Netscape 6.v и перепишут его.

Осталось решить проблему с пиктограммами. У нас правила были такими:

```
#topgreenleft IMG {
  margin-left: 5%;
  margin-right: 5%;
  voice-family: "\"}\"";
  voice-family: inherit;
  margin-left: 1.25%;
  margin-right: 1.25%}
HTML > BODY tttopgreenleft IMG {
  margin-left: 5%;
  margin-right: 5%}
```

Напомню ситуацию. Неверно вел себя только браузер Internet Explorer 6.v, который высчитывал величину полей относительно окна браузера, а не относительно ширины контейнера. После переключения его в режим обратной совместимости проблема отпала сама собой, потому что он начал вести себя как браузер Internet Explorer 5.x, т. е. корректно. Поэтому нам нет необходимости искать обходные пути, так что просто уберем все лишние объявления:

```
#topgreenleft IMG {
  margin-left: 5%;
  margin-right: 5%}
```

В итоге общий код лаже уменьшился, что не может не радовать. Итак, текущие проблемы решили, надо двигаться д&аьше.

Блок3

Займемся третьим блоком и сделаем меню. В блок maim вложим блок menu.

```
<DIV id=mainl>
  <DIV id=menu>
```

```
    <A HREF=#>,IIW3AЙH И ГРАФИКА</AXBK>
    <A HREF=#>O3AEWIHTH</AXBR>
    <A HREF=#>K HCTOKAM</AXBR>
    <A HREF=tt>НОВОЕ СЛОВСК / AXBR>
    <A HKEP=#>ДИЗАЙНЕРЬК/АХВЮ-
    <A HREF=#>MHCKA</A><BR>
    <A HREF=#>НОВОСТН BWICfrK / AXBR>
    <A HREF=#>НАРОJIНЫЕ НОВОСТИ</A>
  </DIV>
</DIV>
```

Пока он будет выглядеть так, как показано на рис. 13.14.

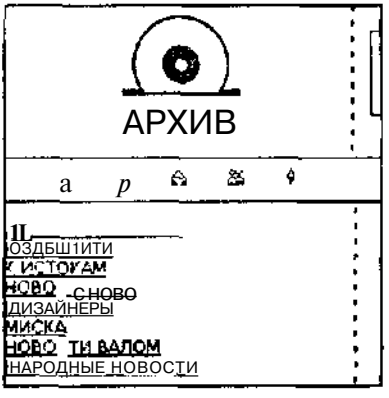


Рис. 13.14. Вид блока меню без стилей

Нам надо отбить блок слева на 5% и сверху на *5%,* сделать ссылки черным цветом и убрать подчеркивание Кроме того, надо увеличить высоту строки.

```
#menu 1
  line-height: 2;
  margin-left: 5%;
  padding-top: 5%}
```

Псевдоклассы для ссылок в меню нам не нужны, потому что не надо обозначать посещенные и активные ссылки, так что можно обойтись контекстным селектором'

```
#menu A {
  color: #000;
  text-decoration: none}
```

После этого блок меню будет выглядеть следующим образом (рис. 13.15).

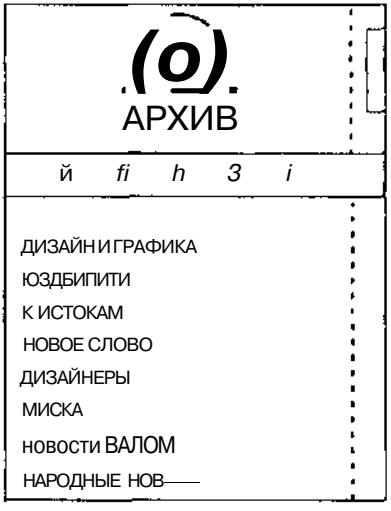


Рис. 13.15. Вид меню после применения стилей

Теперь надо бы сделать форму поиска. Для этого нам понадобится одна картинка с кнопкой отправки Вырежем ее в ImageReady и сохраним в файл go.gif Поместим форму в отдельный блок, который назовем search. Блок будет располаться непосредственно после блока меню.

```
<DIV id=raainl>
  <DIV id=menu>
  . . .
  </DIV>
  <DXV id=search>
    <FORM>
      <INPUT TYPE=text VALUE="noncK" SIZE=12> <INPUT TYPE=image
                                                    SRC=i/go.gif>
    </FOPM>
  </DIV>
</DIV>
```

Нам надо отбить этот блок от левого края на 5% и от блока меню на 10%:

```
#search {
  margin-left: 5%;
  margin-top: 10%}
```

Далее надо сделать черную рамку вокруг поля ввода толщиной I пиксел, а также задать белый фон. Для этого введем класс sfieid, который будем подключать к элементу <INPUT> С атрибутом TYPE=text:

```
.sfieid {
  background:  #FFF;
  border:  Ipx 'Solid #000}
```

Осталось только сделать так, чтобы и форма ввода, и кнопка поиска были на одном уровне. Если бы эти элементы были в ячейках таблицы, то можно было бы использовать атрибут VALIGN, однако они располагаются в одном блоке search, а не в ячейках таблицы. Пока форма поиска в различных браузерах будет выглядеть так, как на рис. 13.16.

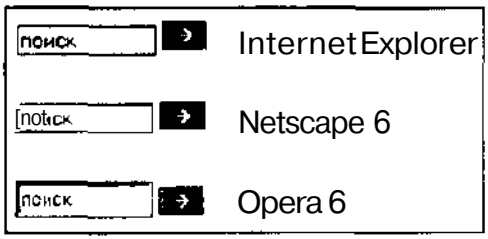


Рис. 13.16. Вид формы поиска в различных браузерах

Как видите, в браузерах **Internet** Explorer и Netscape 6.v элементы формы не выровнены по вертикали, а в браузере Opera выровнены. Но зато Opera очень странным образом применяет рамку к элементу <INFUT>. Вообще-то выравнивание можно было бы осуществить с помощью свойства vertical-align, но браузер Internet Explorer *5.x* не поддерживает нужные значения, так что этот метод отпадает. Но других методов тоже нет! Следовательно, нам *придется* воспользоваться таблицей для вертикального выравнивания элементов.

```
<DIV  xd=search>
<TABLE>
  <FORM>
    <TR>
      <TDXINPUT  TYPE=text  УАШЕ="поиск"5  SIZE=12  class=sfieldX/TD>
      <TDXINPUT  TYPE=image  SRC="i/go ,gif"X/TD>
    </TR>
  </PORM>
</TABLE>
</DIV>
```

Обратите внимание, что тег <FORM> находится перед тегом <TR>, а тег </FORM> после </TR>. Это нарушает вложенность элементов. Но если внести их внутрь элементов <TD>, ТО браузер Internet Explorer добавит лишний перевод строки, так что поле ввода и кнопка снова будут не выровнены. Такая хитрость проблему решает.

Кроме того, у нас элемент с классом sfieid на один пиксел выше, чем кнопка. Высота кнопки 19 пикселов, так что надо для этого класса указать такую же:

```
.sfieid {
  background: #FFF
  height: 19px;
  border: Ipx solid #000}
```

С блоком 3 тоже покончено, так что перейдем к четвертому блоку, в котором находится колонка новостей. Вообще, следующие блоки практически идентичны, так что, создав один из них, мы легко перенесем результат на все остальные.

Блоки 4—5

В самом верху всех трех блоков находится анонс, который состоит из маленькой картинки, текста и стрелочки. Стрелочку вырежем и сохраним в файл arrow.gif. Создадим класс, который назовем anons. В этом классе и будут определены все параметры блока анонса. Сам блок будет формироваться следующим образом:

```
<DIV id=main2>
  <DIV class=anons>
    <IMG SRC=i/anons.gif WIDTH=34 HEIGHT=35 BORDER=0 ALT="">
      Действительно ли мир так тесен, как говорит собственный опыт
    <IM3 SROi/arrow.gif WIDTH=6 HEIGHT=5 BORDER^O АЬТ="далее">
  </DIV>
</DIV>
```

Без стилей этот блок будет выглядеть так, как показано на рис. 13.17.

Нам надо сделать отбивку сверху, слева и справа, уменьшить размер шрифта и сделать его темно-зеленым. Вначале делается отбивка со всех сторон сразу. Пусть она будет одинаковой:

```
.anons {
  padding: 5%}
```

Затем зададим цвет и размер шрифта:

```
.anons {
  color: #277D1E;
```

```
  font-size• 0.9era;
padding: 5%}
```

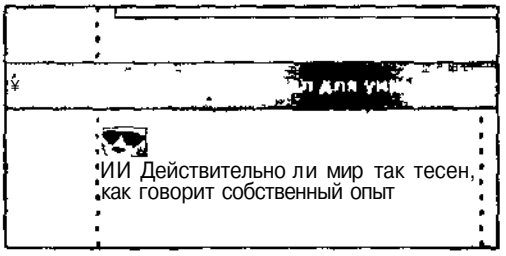


Рис. 13.17. Вид блока анонсов без стилей

На рис 13 18 показано как будет выглядеть блок после внесенных изменений

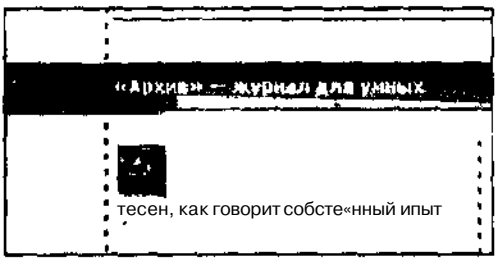


Рис. 13.18. Вид блока анонсов после установки полей и параметров шрифта

Как видите, блок смотрится не совсем правильно Надо, чтобы текст был выровнен по верхнем) краю картинки. Это произойдет, если задать картинке обтекание с помощью объявления float: left. На первый взгляд, можно воспользоваться контекстным селектором .ar.ons <IMG>, однако у нас есть еще одна картинка — это стрелочка По этой причине надо для фотографии задать класс, скажем, thumb Кроме обтекания надо задать небольшой отступ, чтобы текст анонса не прилипал к картинке:

```
.anons IMG.thumb {
  float: left;
  margin-right: 2%}
```

Окончательный вид блока anons будет таким, как на рис 13 19.

A\, да Мы совсем забыли, что и фотография, и текст, и стрелка должны быть ссылкой! Ну, это не беда

```
.anons A {
  color: #277D1E;
```

```
  text-decoration: none
. . .
<DIV id=-main2>
  <DIV class=anons>
    <A HREF=#XIKG SROi/anons.gif HIDTH=34 HEIGHT-=35 BORDER=0 ALT="">
Действительно ли мир так тесен, как говорит собственный опыт <IMG
SRC-i/arrOw.gif WIDTtt=6 HEIGHT=5 BORDER=0 ALT-"далее"></A>
  </DIV>
</DIV>
```

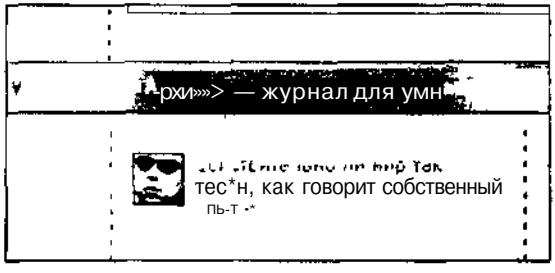


Рис. 13.19. Окончательный вид блока анонсов

Далее идет название блока НОВОСТИ Оно должно быть выведено полужирным шрифтом Arial на светло-зеленом фоне. Кроме того, блок должен быть отбит на 1% от левого края. Для этих целей можно было бы написать стили для заголовка <HI>, но заголовок может использоваться в других местах, так что создадим класс tit, который будет определять блок, содержащий название

```
Hl.t:t (
  background-  #C4DF9B;
  font: bold 1 4ею Arial;
  margin-left: 1%}
```

В коде этот блок будет располагаться непосредственно после блока anons:

```
<DIV id-main2>
  <DIV class=anons>
    . . .
  </DIV>
  <H1 class=tat>
    НОВОСТИ
  </H1>
</DIV>
```

Блок в браузере будет выглядеть так, как на рис. 13.20.

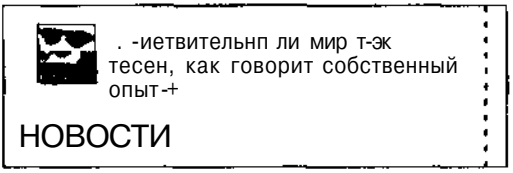


Рис. 13.20. Вид блока НОВОСТИ

Как видите, блок имеет ширину 100%, т. е. равен по ширине блоку main2. Это естественно, потому что блок структурный. Но мы не можем задать ширину, потому что она должна напрямую зависеть от ширины контента (класс мы делаем универсальный, так что слово ОБНОВЛЕНИЯ тоже должно влезать), т. е. нам нужен строчный блок. Но для строчного блока в браузере Internet Explorer *5.x* невозможно установить поля, поэтому нам придется делать вложенные блоки. Для этого уберем фон из класса tit, заключим слово НОВОСТИ в теги <SPANX/SPAN> И зададим для него зеленый фон с помощью контекстного селектора.

```
H1.tit {
  font: bold 1.4em Arial;
  margin-left: 1%}
H1.tit SPAN >
  background: #C4DF9B-
. . .
<H1 class=tit>
  <SPAN> H0BOCTM </SPAN>
</H1>
```

При внимательном рассмотрении получившейся на данный момент картины выясняется, что отступы для блока анонсов заданы не совсем так, как надо. Это нормально, потому что ошибки возникают часто и не сразу их можно заметить, особенно при "резиновой" верстке. У нас для блока анонсов пока задан единый отступ для всех четырех сторон, а именно 5%. Немного поработав с кодом, можно выяснить, что оптимальными являются такие значения:

```
.anons {
  color: #277D1E;
  font-size: 0.9em;
  padding-top: 3%;
  padding-right: 3%;
  padding-bottom: 15%;
  padding-left: 2%)
```

Записать компактно это можно так:

```
.anons {
  color: #277D1E;
  font-size: 0.9em;
  paddxng: 3% 3% 15% 2%i
```

после чего картина станет правильной (рис. 13.21).

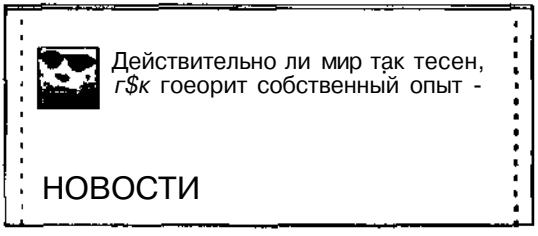


Рис. 13.21. Вид блоков анонса и заголовка новостей с исправленными отступами

Далее должны следовать сами блоки новостей. Структура отдельно взятого блока, содержащего новость, следующая: вначале идет заголовок, который является ссылкой, он выведен красным цветом и полужирным шрифтом; затем идет небольшой текст, а под текстом находится дата новости.

Сначала создадим сам блок, назовем его newsbiock. **Отступы** у нею будут практически такие же, как у блока anons, отличаться будет только нижний отступ, он будет 5% вместо 15%.

```
.newsbiock i
  font-sxze: lem;
  padding: 3% 3% 5% 2%}
```

В коде блок будет стоять после слова НОВОСТИ, т. е. после элемента <HI> с классом tit:

```
<DIV id=main2>
  <DIV class=anons>
    . . .
  </DIV>
  <H1 class=tit>
    <SPAN>4nbsp;HOBOCTH </SPAN>
  </Hi>
  <DXV class=newsblock>
      <A HREF=#>CaMaH важная новость</AXBK>
```

```
Нет важнее новости для нас, чем сам факт наличия новости Без новостей мы
не можем <BR>
      <SPAN>/12.01.2002/</SPAN>
  </DIV>
</DIV>
```

Название является ссылкой, далее идет перевод строки, затем текст новости, потом еще один перевод строки, а в конце дата, которая должна выводиться зеленым цветом, поэтому для нее надо написать отдельный стиль. А пока блок будет выглядеть так, как показано на рис. 13.22.

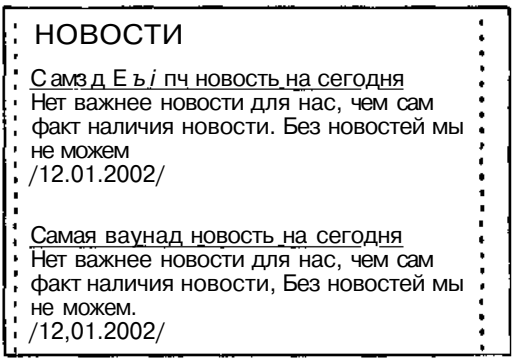


Рис. 13.22. Два блока новостей

Для ссылок надо \брать подчеркивание, сделать их красными и полужирными.

```
.newsblock A {
  font-weight: bold}
.newsblock A:link {
  color: #EC2B20,
  text-decoration: none'
```

При наведении курсора должно появляться подчеркивание, так что уберем свойство text-decoration:

```
.newsblock A:hover (
  color: #EC2B20>
```

А посещенная ссылка должна быть бледно-красная, но подчеркивания быть не должно:

```
.newsblock A:visited {
  color; #FA6F67,
  text-decoration: none}
```

Для даты надо только изменить цвет. Обратите внимание, что мы воспользуемся контекстным селектором для применения стилей к элементу <SPAN>, который находится внутри блока новостей:

```
-newsblock SPAN {
  color: #277D1E}
```

После этого блоки будут выглядеть так, как на рис. 13.23.

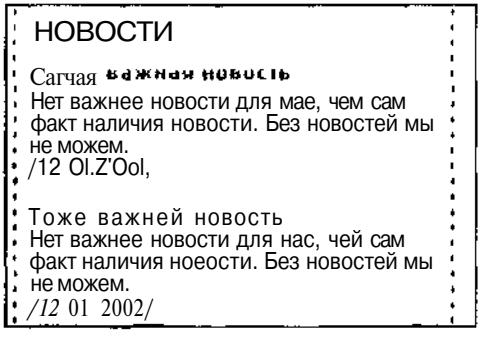


Рис. 13.23. Окончательный вид блоков новостей

Теперь мы можем создавать сколько угодно блоков с новостями, поскольку они все совершенно однотипные. Далее нам надо сделать рскламн>ю врезку. Она вплотную прилегает к пунктирным линиям и имеет светло-фиолетовый цвет фона. Весь текст в этом текстовом рекламном блоке является ссылкой, но подчеркивания быть не должно.

Сначала сделаем сам блок, который назовем advi. Размешаться он будет где-то между блоками новостей:

```
<DIV class=newsblock>
  <A НКЕГ=#>Тоже важная новость</A><BR>
Нет важнее новости для нас, чем сам факт наличия новости. Без новостей мы не можем.<BR>
  <SPAN>/12.01.2002/</SPAN>
</DIV>
<DIV class=advl>
  <A НКЕЕ"=#XB>Рекламный блок.</B> Немного текста о чем-кибудь важном или не очень важном. Главное, что рекламный.</A>
</DXV>
```

Ну а теперь напишем стили. Сделаем фиолетовый фон и отступы:

```
.advl (
  background: #C4C8E3;
  padding: 5%}
```

А теперь уберем подчеркивание ссылок внутри блока и сделаем их черными:

```
.advl A {
  color: #000;
  text-decoration: none}
```

Вид рекламного блока после внесенных дополнений показан на рис. 13.24.

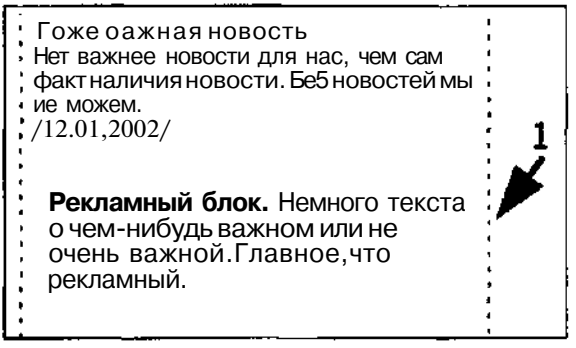


Рис. 13.24. Вид рекламного блока

Обратите внимание на правый край блока. Фон закрывает собой пунктирную линию, а нам нужно, чтобы она была видна. Кроме того, нам надо, чтобы и слева, и справа между линией и блоком была щель в 1 пиксел. Решение на ум приходит сразу: следует просто установить левое поле для блока шириной в 1 пиксел, а правое поле шириной в 2 пиксела, тогда линия будет видна и появится нужная шель:

```
.advl {
  background: #C4C8E3;
  padding: 5%;
  margin-right: 2px;
  margin-left: lpx)
```

После этого блок станет таким, как на рис. 13.25.

Мы создали все необходимые блоки для заполнения второго и третьего столбцов. Так что размножим их и посмотрим на результат (рис. 13.26). По ходу вырежем еше две маленькие картинки для анонсов.

На первый взгляд все нормально, но это только на первый взгляд. Впрочем, давайте сначала доделаем последнюю колонку, а уж потом рассмотрим еще одну проблему.

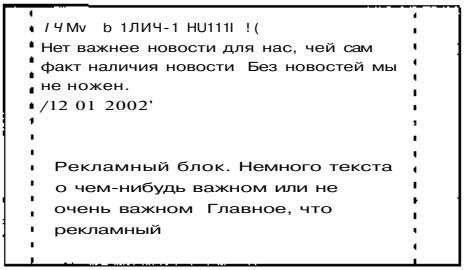


Рис. 13.25. Вид рекламного блока с добавленными полями

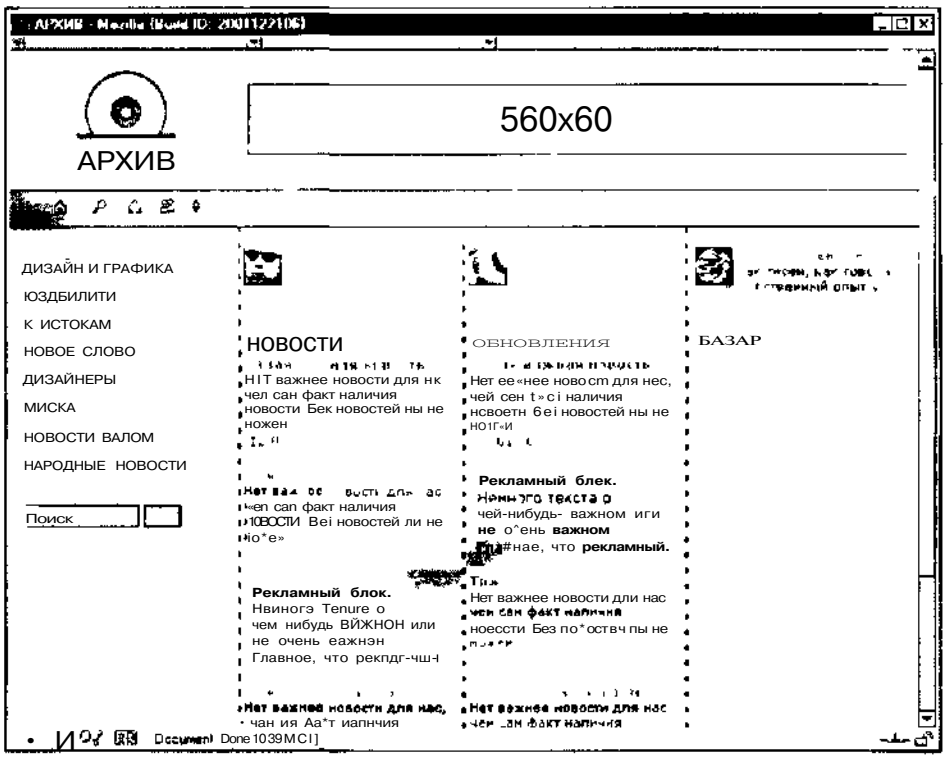


Рис. 13.26. Вид сайта на промежуточном этапе верстки

Блок6

Итак, в четвертой колонке у нас блоки очень nросiые В начале идет выделенный полужирным заголовок, затем небольшой текст, а в коние — красная ссылка, тоже выделенная полужирным Блок назовем bazar

```
<H1 class=tit>
  <SPAN> БАЗАР </SPAN>
```

```
</H1>
<Drv class=bazar>
  «ЗХГообщвние, добавленное посетителем сайта</ВХВЙ>
  Я тут сегодня нашел очень интересный сайт по верстке, так что идите и
смотрите<ВН>
  <А НКЕГ=#>/посмотреть/</А>
</DIV>
```

Стиль у этого блока будет совершенно такой же, какой и у блока новостей newsblock:

```
.bazar {
  font-size: lem;
  padding: 3% 3% 5% 2%)
```

А для ссылки напишем такой стиль:

```
.bazar A t
  color: #EC2B20;
  font-si ze: 0.9em;
  font-weight: bold;
  text-decoration: none}
```

После этого блок будет выглядеть так, как на рис. 13.27.

Рис. 13.27. Блок сообщения посетителя сайта

Размножим несколько блоков, чтобы представить общую картину, и тут же наткнемся на ту самую последнюю проблему (рис. 13.28).

Как видите, высота контента у нас превысила высоту блоков, поэтому пунктирная линия обрывается (1), (2). Но это только в браузерах Opera 5+ и Netscape *6.x*. Браузеры Internet Explorer ведут себя по-другому: если содержимое превышает высоту блока, то его высота увеличивается, так что пунктирная линия идет вровень с контентом. Что же делать?

Помните, мы указывали высоту блоков 70°£? Так вот, этого оказалось недостаточно. Нам надо подобрать высоту так, чтобы она лишь немного превышала высоту контента. Но в этом и заключается *практически нерешаемая проблема*. Мы не можем знать точно, будет в колонке три новости или семь новостей. Для семи новостей надо указать высоту где-то 140%, а для трех

новостей около 80%. Таким образом, решение отсутствует принципиально. В данном случае я знаю, что новостей будет не больше шести, но обычно это неизвестно, так что проблема достаточно важная.

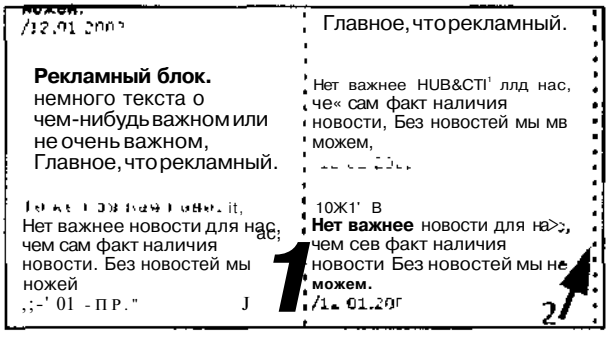


Рис. 13.28. Проблема с высотой блоков

Отсюда вытекает, что с помощью стилей невозможно сверстать сайт, у которого есть шапка, центральная часть и так называемый футер*. Причем в центральной части должен быть элемент, который согласно дизайну *должен соприкасаться и с шапкой, и с футером,* например, пунктирная линия, сплошная линия или какой-нибудь хитрый фон. Структурная схема такого сайта показана на рис. 13.29.

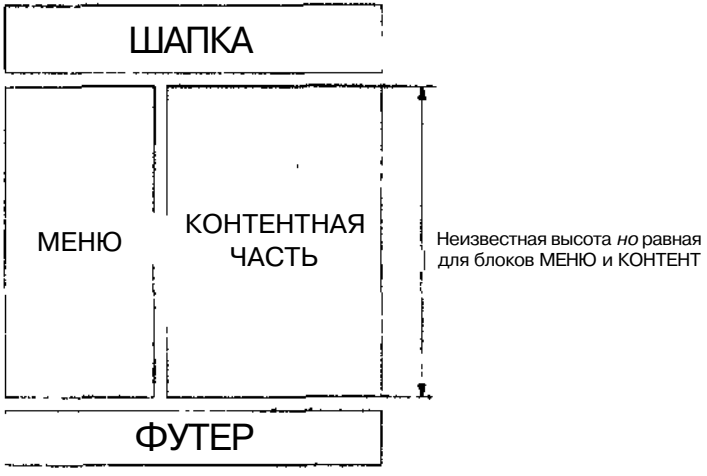


Рис. 13.29. Макет сайта, который принципиально нельзя корректно сверстать с помощью блоков

* Нижняя часть HTML-странички, содержащая копирайт и прочую служебную информацию (сленг). — *Ред.*

Однако если такого требования нет, то сайт точно можно сверстать с помощью стилей.

Почему возникает такая проблема? Вспомните табличную перетку. Допустим, мы задали таблицу с двумя колонками.

```
<TABLE>
  <TR>
    <TDX/TD>
    <TDX/TD>
  </TR>
</TABLE>
```

В первой колонке контент будет высотой 600 пикселов, а во второй — 300 пикселов. Естественно, как бы мы ни задавали высоту таблицы, все равно в этом случае она будет равна высоте наибольшей ячейки, т. е. 600 пикселов. Но высота второй колонки не может быть меньше высоты самой таблицы, т. с. ее высота тоже будет 600 пикселов. Получается, что высота обоих колонок будет одинаковой! Но в CSS невозможно сделать так, чтобы высота одного блока была привязана к высоте другого блока, если высота контента изменяемая величина. Поэтому мы не можем выровнять блоки и по нижнему, и по верхнему краю. т. е. мы не можем сделать блоки одинаковых размеров и по высоте, и по ширине, подчеркиваю, *если высота контента изменяется.*

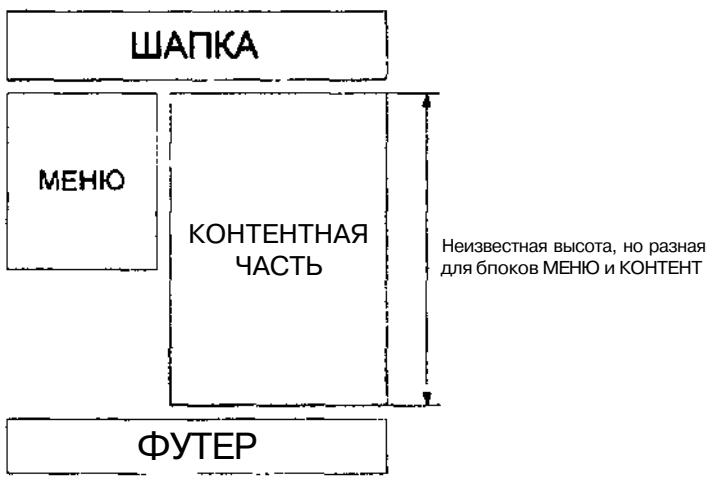


Рис. 13.30. Макет сайта, который можно сверстать с помощью блоков

На первый взгляд, проблему можно решить с помощью вложенных блоков. ДОПУСТИМ, МЫ заключим четыре блока rr.ainl, main2. гааш3 И main4 В какой-ТО Общий блок mainall, НО Какую ВЫСОТУ уКЭЗЭТЬ ДЛЯ блОК3 mainall? Вот В

чем вопрос. Фактически нам псе равно надо знать выс-оiу самого высокого блока, но если бы мы обладали этим знанием, то вложенные блоки нам были бы ни к чему.

Вообще говоря, если нет элемента, который *должен соприкасаться и с шапкой, и с футером,* то сайт можно сверстать с помощью стилей даже в том случае, если высота контента есть величина неизвестная. Схема такого сайта показана на рис. 13.30.

Полный CSS-код

Ну и в заключение приведу полный код страницы с комментариями. Этот код будет уже организованный и оптимизированный, поэтому могут быть некоторые отличия от того, что было но чоду главы. Но они все будут объяснены в комментариях.

```
BODY { /* тело документа */
  background: #FCF9ED; /* бледно-охровый цвет фона всей страницы */
  font: 0.7em Verdana, Tahoma, sans-serif; /* методом подбора установили,
что шри^т 0.™ет. является оптимальным. Tahoma для платформы Linux,
a sans-serif для страховки */
  margin: Opx /* обнуляем поля, чтобы не было неприятных дополнительных
отступов от границ окна браузера */}

.advl ( /* текстовый рекламный блок */
  background: ЯС4С8Е3; /* светлб-фиолетовый цвет фона */
  padding: 5%; /* отступы, чтобы текст не "прилипал" к границам блока.
Отступы нужны для комфортности чтения и для улучшения визуального восприятия */
  margin-right: 2px; /* устраняет наложение фона на пунктирную линию и
устанавливает отступ от нее справа в 1 пиксел */
  margin-left: lpx /* устанавливает отступ от пунктирной линии слева в 1
пиксел */|

.advl A { /* ссыпки в текстовом рекламном блоке */
  color: #000; /* черный цвет ссылок, находящихся в рекламном блоке */
  text-decoration: none /* устраняет подчеркивание ссылок */}

.anons ( /* блок анонса */
  font-size: O.Sem; /* размер шрифта чуть меньше, чем основной •/
  padding: 3% 3% 15% 2% /* отступы разной величины */}
```

```
.anons A | /* ссылки в блоке анонса *,
  color: #277D1E; /* зеленый цвет ссылок */
  text-decoration: none  * устраняет подчеркивание ссылок */}

.anons IMG.thumb ( /* маленькие фотографии в блоке анонса */
  margin-right: 2%: /* поля вокруг фотографии, чтобы текст не прилипал */
  float: left; /• текст будет обтекать фотографию справа */}

.bazar, .newsblock { /* так как блоки новостей и рубрики "базар" одинаковы, их можно сгруппировать */
  padding: 3% 3% 5% 2% /* отступы снизу чуть больше, а слева самый маленький */

.bazar A { /* ссылки в блоке рубрики "базар" */
  color- #EC2B20; /* красный цвет ссылок */
  font-weight: bold; /* устанавливает полужирный шрифт для ссылок */
  font-size; 0.9em; /* размер шрифта чуть меньше, чем основной */
  text-decoration: none  ' устраняет подчеркивание ссылок */}

.newsblock A { /* ссылки в блоке новостей */
  font-weight: bold /* устанавливает полужирный шрифт для всех ссылок */}

.newsblock A:link { /* непосещенные ссылки в блоке новостей *7
  color: #ЕС2В20; /* красный цвет ссылок */
  text-decoration: none /* устраняет подчеркивание ссылок */)

.newsblock A:visited { /* посещенные ссылки в блоке новостей */
  color: #FA6F67; /* бледно-красный цвет ссылок */
  text-decoration: none /* устраняет подчеркивание ссылок */}

.newsblock A:hover { /* ссылки при наведении курсора в блоке новостей */
  color: #EC2B20 /* красный цвет ссыпок. Так как нет объявления text-decoration: none, то ссылки будут подчеркиваться при наведении курсора */>

.newsblock SPAN { /* дата в блоке новостей */
  color: #277D1E /* устанавливает зеленый цвет */}

.sfield { /* голе в форме поиска */
  background: #FFF; /* белый цвет фона поля */
```

```
  border: 1px solid #000; /* черная рамка толщиной 1 пиксел */
  height: 19px /* высота поля должна быть 19 пикселов. Это для согласования с высотой рисунка go.gif, который тоже 19 пикселов высотой V)

HI.tit ( /* блок заголовка раздела */
  font: bold 1.4emArial, sans-serif; /* заголовок будет выводиться полужирным шрифто:.: Anal в 1,4 раза большим, чем основной */
  margin-left. 1% /* установка поля для маленькой отбивки блока от пунктирной разделительной линии */}

HI.tit SPAN | /* строчный блок в заголовке, чтобы фон не распространялся дальше букв */
  background: #C4DF9B /* светло-зеленый фон для заголовка */}

#mainl, #mam2, #main3 { /* первые три центральных блока совершенно одинаковы, поэтому их можно сгруппировать */
  background: url{i/dotl.gif) repeat-y 100% 0%; /* устанавливает фоновое изображение dotl.gif, которое Судет повторяться по оси Y. Кроме того, оно будет с правого края блока, поскольку сдвинуто на 100% относительно левого края */
  width: 25%; /* ширина блока будет 1/4 от ширины окна браузера */
  height: 100%; /* высота блока будет равна высоте окна браузера */
  rain-height: 90%; /* минимальная высота будет чуть меньше высоты окна браузера * '
  float: left; /* включает для данного блока плавающую модель, причем остальной контент должен обтекать этот блок справа. Необходимо для формирования колонок */}

HTML > BODY #mainl, HTML > BODY #raain2, HTML > BODY #raair.3 { /* так как в Netscape б и Opera 5+ при высоте контента, превышающей высоту блока, блок не растягивается, надо явно это указать */
  height: auto /* указывает, что блок должен растягиваться, чтобы вместить в себя весь контент */}

#main4 { /* четвертый центральный блок. Совершенно аналогичен предыдущим за исключением фонового рисунка */
  width: 25%;
  height: 100%;
  min-height: 90%;
  float: left}

HTML > BODY #main4 {
  height: auto*
fmenu: - -оьной навигации */
```

```
  line-height: 2; /* высота межстрочных интервалов Судет равна удвоенной
высоте шрифта, так меню смотрится лучше */
  margin-left: 5%; /* поле для отбивки слева */
  padding-top: 5% /* отступ от верхнегр края, чтобы меню к нему не
"прилипало" */)

#menu A { /* ссылки в меню */
  color: #000; /* черный цвет ссылок */
  text-decorataon: none /* устраняет подчеркивание ссыпок */}

if search ( /* блок для формы поиска */
  margin-left: 5%; /* левое поле такое же, как и у блока навигации, чтобы
они Сыпи выровнены по одной линии */
  margin-top: 10% /* отбивка сверху */}

#top { /* верхний блок, содержащий логотип и баннер */
  width: 100%; /* ширина блока равна ширине окна браузера */
  height: 118px /* точно задаем высоту в пикселах */}

#topleft { /* Слок с логотипом */
  background: url{i/dotl.gif) repeat-y 100% 0%; /* фоновый рисунок, кото-
рый прсдстар-.гярт гоСпй разделительную пунктирную линию */
  text-align: center; /* выравнивание логотипа по центру */
  width: 25%; /* ширина блока равна ширине первого главного блока. Это
необходимо для совпадения разделительных линий */
  height: 118px; /* без этого объявления браузер Internet Explorer не
доводит до конца блока пунктирную линию, так что остается пустой промежу-
ток */
  padding-top: 15px; /* точно устанавливаем верхний отступ ~/
  float: left; /* включение плавающей модели. Необходимо для того, чтобы
и логотип, и баннер находились на одной горизонтальной линии */}

HTML>BODY #topleft ( /* устранение конфликта между неверными блоковыми
моделями в браузерах Internet Explorer и всех остальных */
  height: Ю3px /* если прибавить 15 пикселов верхнего отступа, то полу-
чится как раз 118 пикселов */}

#topright ! /* блок для баннера */
  margin-top: 29px; /* по сути дела, это центрирование баннера по верти-
кали */
  width: 74%; /* для Internet Explorer вообще-то надо бы указать 75%, но
небольшая разница в 1% в данном случае не играет роли */
```

```
  padding-left: 1%; /- небольшая отбивка от разделительной пунктирной
линии */
  float: left; /* включение плавающей модели. Необходимо для того, чтобы
и логотип, и баннер находились на одной горизонтальной линии */)

#topgreen { /* Олок, формирующий зеленую полосу */
  background: url(i/bgl.gif) repeat-x; /* собственно, сама полоса. Форми-
руется повторением фонового рисунка bgl.gif по оси X */
  width: 100%; /* полоса растягивается на всю ширину окна браузера */
  height: 35px; /* высота задается точно в пикселах */
  padding-top: 8px; /* отступ сверху. Фактически, для выравнивания по
вертикали пиктограмм */}

#topgreenleft { /* блок пиктограмм */
  text-align: center; /* выравнивает пиктограммы по центру */
  width: 25% /* ширина блока равна ширине первого главного блока и блока
с логотипом */
  float: left; /* включение плавающей модели. Необходимо для того, чтобы
пиктограммы и надписи находились на одной горизонтальной линии */)

#topgreenleft IMG { /* отдельно взятая пиктограмма. Это нужно для того,
чтобы пиктограммы были разделены пространством, которое зависит от ширины
окна браузера */
  margin-left: 5%; /* левое поле */
  margin-right: 5% /* правое поле */}

#topgreencenter { /* блок с названием журнала */
  color: #FFF; /* белый цвет шрифта */
  font-weight: bold; /* полужирный шрифт */
  width: 50%;
  padding-left: 1%; /* отступ точно такой же, как и у блока баннера */
  float: left; /* включение плавающей модели. Необходимо для того, чтобы
пиктограммы и надписи находились на одной горизонтальной линии */}

tttopgreenright { /* блок с текущей датой. Практически аналогичен блоку с
названием */
  color: #FFF;
  font-weight: bold;
  «dtth: 2T\-
```

Полный HTML-код

А вот так выглядит HTML-код.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <T1T1£>АРХИВ - журнал для умных</T1TbE>
</HEAD>
<BODY MARGINHEIGHT-0 MARGINWIDTH«0>
<.• — Блок 1 (Шапка) —>
<DIV id=top>
  <!— Логотип —>
  <DIV id=topleft>
    <IMG SRC-i/archive.gif WIDTH=87 HEIGHT=89 BORDER=0 ALT="Архив">
  </DIV>
  <!— Баннор —>
  <DIV id=topright>
    <IMG SRC=i/banner.gif WIDTH*560 HEIGHT=60 BORDER=0 АЬТ="Баннер">
  </DIV>
</DIV>
<!— Блок 2 (Шапка) —>
<DIV id*topgreen>
  <!— Пиктограммы —>
  <DIV ia=-copgreenieft>
  <A HREF=#><IMG SRC=i/home.gif WIDTH=12 HEIGHT=12 BORDER=0
                ALT="На главную"x/A>
  <A HREF-#><IMG SRC=i/search.gif WIDTH=12 HEIGHT=12 BORDER=0
                АЬТ="Поиск"X/A> <A HREF=#>
            <IMG SRC-i/letter.gif WIDTH=12 HEIGHT=12 ЮRDER=0
                ALT="Отправить письмо"></A>
  <A HREFb#xIMG SRC=i/about.gif WIDTH=12 HEIGHT=12 ЮRDER=0
                АЬТ="Редакция"x/A>
  <A HREP«#><IMG SRC=i/sitemap.gif WIDTH=12 HEIGHT=12 BORDER^O
                ALT="КарТа сайта"></A>
  </DIV>
  <!— Название журнала —>
  <DIV id=tcpgreencenter>
    &laquo;ApxMBSraquo; fimdash; журнал для умных
  </DIV>
```

```
  <!— Текущая дата —>
  <DIV id«topgreenright>
    12 февраля 2002 
  </DIV>
</DIV>
<!-- Основная контентная часть -->
<DIV id=rrainl>
  <•-- Елок 3 (Панель навигации) -->
  <D1V ia-ir.cr.u>
    <A HREF-^ДИЗАЙН И ГРАФИКА</A><BK>
    <A HREF=#>D3AEMJ|HTM</A><BR>
    <A HREF=#>K НСТОКАМ</A><BR>
    <A HREF=#>НОВОЕ СЛОЮ</A><BR>
    <A HREF=#>ДИЗАЙНЕРЫ</A><BR>
    <A HREF=#>MMCKA</AXBR>
    <A HREF=#>НОВОСТН ВАЛСМ</A><ВЯ>
    <A HREF=#>HAPOflHUE НОВОСТИ</A>
  </DIV>
  <•-- Форма поиска -->
  <DIV id=search>
  <TABLE>
    <FORM>
      <TR>
        <TD><INPUT TYPE=text VALUE="noMCK" SI2E=12 class=sfieldx/TD>
        <TDXINPDT TYPE=iinage SRC-i/go.gifx/TD>
      </TR>
    </FORM>
  </TABLE>
  </DIV>
</DIV>
<!— Блок 4 (Новости) -->
<DIV id=main2>
  <! — Анонс —>
  <DIV class=anons>
    <A HREF=#><IMG SRC=i/anons.gif WIDTH=34 HEIGHT=35 BORDER=0
                  ALT-"" class-thumb>
      Действительно ли мир так тесен, как говорит собственный опыт
      <IMG SRC=i/arrow.gif WIDTH«6 HEIGHT=5 BORDER=0 ALT=-далее"></A>
```

```
  </DIV>
  <'— Название колонки (Новости) —>
  <л1 CxdSb*Llt>
    <SPAN> HOBOCTM </SPAN>
  </H1>
  <t— Первый блок новостей —>
  <DIV class=new^n^?-- >
    <A HREF=#>CdMa.s важная новость</AxBK> *
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SPAN>
  </DIV>
  <•-- Второй блок новостей -->
  <DIV class=newsblock>
    <A ШгЕГ=#>Тоже важная новость</A><BR>
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SEAN>
</DIV>
    <•— Текстовый рекламный блок -->
  <DIV class=advl>
    <A HREF-#xB>Рекламный блок.</B>
              Немного текста о чем-нибудь важном или не счень важном.
              Главное, что рекламный.</A>
  </DIV>
  <'-- Третий блок новостей -->
  <DIV  class=newsD_ock>
    <A НЙЕГ=#>Тоже важная НОВОСТЬ</A><BR>
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SPAN>
  </DIV>
</DIV>
<'— Блок 5 (Обновления) —>
<DIV ad=inain3>
  <•-- Анонс ™>
  <DIV class=anons>
    <A HREF=#XIMG SRC=i/anons2.gif WIDTH'34 HEIGHT=35 BORDER=0 ALT=""
                  class=thumb>
```

```
      Действительно ли мир так тесен, как говорит собственный опыт
      <IMG SROi7arrow.gif WIDTH-6 HEIGHT=5 BORDER-0 ALT="далее"></A>
  </DIV>
  < ! — Название колонки (Обновления) —>
  <H1 class=tit>
    <SPAN>&nb3p;0EH0BJlEHMflSnbsp;</SPAN>
  </DIV>
< ! — Первая новость об обновлении на сайте —>
  <DIV clas5=newsblock>
    <A HR£F=#>CaMaa важная новость</A><BR>
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SPAN>
</DIV>
  < i — Текстовый рекламный блок -->
  <DIV ciass=advl>
    <A НРЕГ=#><B>Рекламный Олок.</B> Немного текста о чем-нибудь
              важном или не очень важном. Главное, что рекламный.</A>
  </DIV>
  < ! — Вторая новость об обновлении на сайте -->
  <DIV class=newsblcc:k>
    <A HREF=#>To:«e важная НОВОСТЬ</A><BR>
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SPAN>
  </DIV>
  <!-- Третья новость об обновлении на сайте -->
  <DIV class*newsblock>
    <A HREF=#>T^e важная новость</A><BЯ>
    Нет важнее новости для нас, чем сам факт наличия новости.
    Без новостей мы не можем.<BR>
    <SPAN>/12.01.2002/</SPAN>
  </DIV>
</DIV>
<!-Блок 6 (Базар) —>
<DIV id=-main4>
  < i — Анонс —>
  <DIV class=anons>
```

```
    <A HREF=#XIMG SRC=i/anons3,gif WIDTK=34 HEIGHT=35 BORDER=0 ALT=""
                  class=thumb> Действительно ли мир так тесен,
                               как говорит собственный опыт
      <IMG SROi/arrow.gif WIDTH»6 HEIGHT=5 BORDER»0 ALT-"далее"></A>
  </DIV>
  <!— Название колонки (Базар) —>
  <H1 class-tit>
    <SPAN> EA3AP </SPAN>
  </H1>
  <!— Первое сообщение -->
  <DIV class=bazar>
    <B>Сообщение, добавленное посетителем сайта</BxBP>
    Я тут сегодня нашел счень интересный сайт по верстке,
    так что идите и смотрите<BK>
    <A HREF=#>/nocMOTpeTb/</A>
  </DIV>
  <i— Второе сообщение —>
  <DIV class-bazar>
    <B>Сообщение, добавленное посетителем cauTa</B><BR>
    Я тут сегодня нашел очень интересный сайт по верстке,
    так что идите и смотрите<BЯ>
    <A HREF=#>/noCMOTpeTb/</A>
  </DIV>
  <!— Третье сообщение -->
  <DTV class=tazar>
    <B>Сообщение, добавленное посетителем caftTa</BXBR>
    Я тут сегодня нашел очень интересный сайт по верстке,
    так что ияите и смотрите<BЯ>
    <A HREF=#>/nocMOTpeTb/</A>
  </DIV>
</DIV>
<•-- Нижняя зеленая полоса (Футер) -->
<DIV id=bottomgreen>
  <DIV id=bot:tomgreenleft>
    <A HREF=#XIMG SRC=i/home.gif WIDTH=12 HEIGHT~12 BORDER=0
                  ALT="Ка главную"x/A>
    <A HREF-#XIMG SRC=i/search.gif WIDTH=12 HEIGHT-12 BORDER=0
                  АЬТ="Поиск"X/A>
```

```
    <A HREF"#><IMG SROi/letter.gif WIDTH-12 HEIGHT=12 BORDER=0
                ALT*"Отправить гшсьмо"X/A>
    <A HREF=#XIMG SRC=i/about.gif WIDTH=12 HEIGHT=12'BORDER=0
                ALT="Редакция"X/A>
    <A HREP=#XIMG SRC=i/sitemap.gif WIDTH=12 HEIGHT=12 BORDER=0
                АЬТ="Карта сайта"></A>
  </DIV>
  <DIV id=bottomgreencenter>
    &laquo;ApxHB&raquo; smdash; журнал для умных
  </DIV>
  <DIV id=bottomgreenright>
    12 февраля 2002snbsp;
  </DIV>
</DIV>
</BODY>
</HTML>
```

Вот такой у нас получился код. А теперь не поленитесь, зайдите на какой-нибудь достаточно сложный сайт, например. www.Icnta.ru или www.tut.by, посмотрите исходный кол и сравните с тем, что напечатан выше Ч\вствиуете разницу?

Выводы

Мы уже рассматривали преимущества и недостатки каскадных таблиц стилей, однако мы не говорили конкретно о модели верстки. Чем же CSS-позипионирование лучше табличною позиционирования? Давайте для начала выделим достоинства.

- О При использовании таблиц поисковые машины гораздо хуже индексируют страницы. Использование стилей эту проблему устраняет полностью, так что по нужным запросам страница будет находиться чаще
- G Сама структура документа при использовании CSS-блоков становится гораздо логичнее, что облегчает восприятие.
- *О* Уменьшается объем HTML-кода. Учитывая тот факт, что внешний CSS-фаил кэш и рус гея браузерами, то при последующих посещениях сайта страницы будут открываться быстрее. Тогда как при табличной верстке этого преимущества нет
- G Таблицы обрабатываются и отображаются браузерами медленнее, чем CSS-блоки, так что субъективное время открытия страницы уменьшается, что хорошо.

- Использование CSS-блоков обеспечивает относительную простоту внесения изменений в содержимое и дизайн сайта, тогда как при использовании таблиц о простоте вообще нельзя вести разговор.

Недостатки, естественно, тоже есть.

- Верстать сложные сайты с помощью CSS-блоков пока гораздо труднее, чем с помощью таблиц. Это связано с неодинаковом реализацией поддержки каскадных таблиц стилей в различных браузерах. Кросс-браузерный код в абсолютном большинстве случаев проблему решает, но он достаточно сложен.

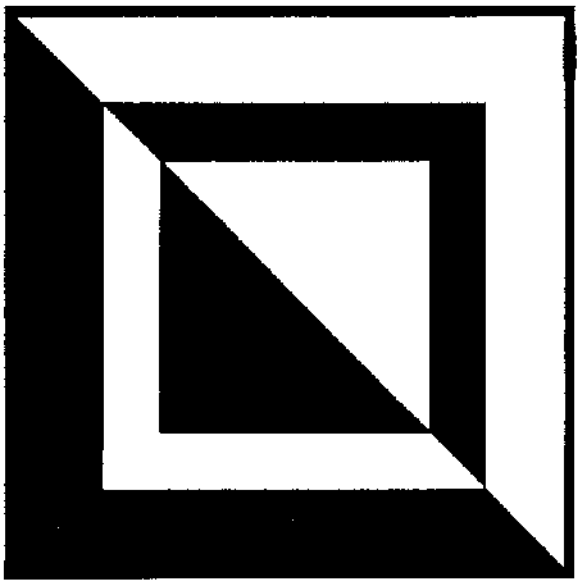
О Не любой макет можно сверстать с помощью CSS-блоков. но таких макетов весьма мало, так что проблема встает далеко не всегда

О Переход от таблиц к CSS-блокам непрост. Это объективная трудность, связанная с привычкой HTML-верстальщиков к таблицам. Практически все нюансы табличной верстки уже изучены и давно известны, тогда как нюансы верстки CSS-блоками только начинают широко изучаться.

Что можно сказать в заключение? Да, пока верстка CSS-блоками нова и непривычна. Да, у нее есть свои проблемы, лежащие, в основном, на плечах производителей браузеров. Да, сложные макеты верстать CSS-блоками пока сложнее, чем таблицами. Но все это временные проблемы, которые решатся в ближайшие год-два. Уже сейчас верстать простые и средней сложности сайты па основе CSS-позиционированпя легче и лучше, а это является первым признаком того, что прогресс неумолимо движется вперед. Чтобы успеть за ним, надо постоянно \читься.

Прочитав эту книгу, вы, скорее всего, не узнали всего, что хотели. Что-то осталось пока непонятным, что-то показалось странным и нелогичным, что-то вы просто пропустили, посчитав неважным. Вы не стали профессионалом, прочитав книгу Но профессионалами в одночасье не становятся, так что я и не ставил перед собой такую цель. Возможно, книга вам и не понравилась, но я надеюсь только на одно, что книга помогла вам, по крайней мере, *почувствовать идею каскадных таблиц стилей* и проникнуться духом CSS. Все остальное придет со временем, если есть заинтересованное^. Если она у вас появилась, то я считаю свою задачу полностью выполненной.

И Эаь Г?



ЧАСТЬ III

ПРИЛОЖЕНИЯ

Приложение 1

Элементы HTML

Данное приложение содержит описание всех основных элементов HTML, атрибутов и их возможные значения.

A

Служит для объявления элемента гиперссылкой или якорем.

Атрибуты:

- NAME — служит для установки якоря. Ссылка на якорь должна включать в себя имя элемента <A>;
- HREF — содержит ссылку на ресурс, i e. UR1;
- HREFLANG — содержит идентификатор языка документа, на который ведет гиперссылка;
- TYPE — содержит тип документа, на который ведет ссылка;
- REL — описывает отношения документа с гиперссылкой;
- REV — описывает обратную ссылку;
- TITLE — содержит описание гиперссылки;
- TABINDEX —• содержит позицию элемента в списке, согласно которому происходит передача фокуса при нажатии клавиши <ТаЬ>;
- ACCESSKEY — содержит клавишу, при нажатии которой элемент получает фокус.

Пример:

```
<A HREP="http://www.artel.by" TYPE="text/html" HREFLANG="rus" T1TЪE="Сайт студии дизайна">Д-:зайк Артель</A>
```

ABBR

Служит для обозначения аббревиатур, таких как WWW, CSS, HTML и др.

Атрибут:

- TITLE — содержит описание аббревиатуры. Например, можно указывать ее расшифровку.

BASEFONT

Задает базовый шрифт документа, В спецификации помечен как нежелательный. Гораздо удобнее задавать базовый шрифт с помошыо каскадных таблиц стилей для селектора <BODY>.

BDO

Задает направление письма (справа налево или слева направо).

Атрибут:

П DIR — задает направление письма. Может принимать два значения itr и ГСГ.

Пример:

```
<BDO DlR="ltr">B русском языке пишут слева направо</BBO>
```

BIG

Увеличивает размер шрифта. Можно использовать для логического выделения текста.

Пример:

<P>Мы любим только <B10большие</B10 автомоОили</P>

BLOCKQUOTE

Служит для выделения больших цитат.

Атрибут:

П CITE — в этом атрибуте указывается адрес документа, с которого была взята цитата.

Пример'

```
<BLOCKQUOTE CITE="http://www.firefalcon.com/silver/silver.html">
Серебряный бакс, растаявший прямо на ладони у Оармена, Был пераьм тревожным звоночком. Хрен его знает, что на него нашло, но он с ревом выхватил из-под стойки дробовик с явным намерением прикончить меня на месте. К егс огромному сожалению заряд прошёл мимо, разбив только вчера вставленное окно. До сих пор не понимаю, как это он промахнулся с двух метров?
</BLOCKQUOTE>
```

BODY

Служит для обозначения тела документа.

Атрибуты:

O BGCOLOR — задает цвет фона документа;

O BACKGROUND, TEXT, MINK, LiiiK, VLIKK — задают фоновый рисунок, цвет текста, цвет активных ссылок, цвет непосещенных ссылок, цвет посещенных ссылок соответственно. Все они помечены как нежелательные, а вместо них предлагается использовать каскадные таблицы стилей.

Пример:

```
<BODY BGCOLOR="#FFFFFF">
</BODY>
```

BR

Вставляет принудительный перевод строки.

Атрибут:

D CLEAR — прекращает обтекание объекта. Практически аналогичен свойству CSS clear. Может принимать значения left, right и all.

Пример:

```
Иногда надо, чтобы слово начиналось<вк>с новой строки.
```

BUTTON

Создает кнопку, которая служит для расширения функциональности форм. Например, с помощью JavaScript можно при нажатии на кнопку проверять заполнение полей формы на корректность. Кроме того, можно создавать кнопки с текстом и изображением.

Атрибуты:

- NAME — задает имя кнопки;
- VALUE — задает начальное значение кнопки;

D TYPE — задает тип кнопки:

- submit — создается кнопка, отсылающая информацию па сервер;
- rese-_ — создается кнопка, очищающая содержимое полей формы;
- button — создается кнопка, функциональность которой надо задать с помощью JavaScript или VBScript;

- DISABLED — ключевое слово, которое делает кнопку недоступной для нажатия;
- TITLE — поясняющий текст для кнопки;
- ACCESSKEY, TABINDEX.

Пример:

```
<BUTTON NAME="check" VALUE="0" TYPE="button" T1TЬE="Эта кнопка служит для проверки данных" OnClicJc="chec]cform() ">
  Send <IMG src="/i/check.gif">
</BUTTON>
```

CAPTION

Служит для задания заголовков таблицы. Должен следовать непосредственно после тега <TABLE>. Имеет атрибут ALIGN, который помечен как нежелательный.

Пример:

```
<TABLE BORDER="1" WIDTH="400">
<CAPT10Ы>Таблица 1. Список членов рабочей 1^>уплькК/САРТ1<Ж>
<TR>
  <TO>Имя</TO>
  <TO>Фамилия</TO>
</TR>
<TR>
  <TD>HBaH</TD>
  <TD>HenoceflOB</TD>
</TR>
</TABLE>
```

CENTER

Центрирует содержимое. Помечен как нежелательный, а вместо него рекомендуется центрировать содержимое с помошью каскадных таблиц стилей (text-align: center).

CITE

Служит для обозначения небольших цитат или источников, из которых взята информация.

Пример:

```
Я точно не помню, но вроде бы <C1TE>B. И. Ленин</C1TE> сказал:
<0>Учиться, учиться и учиться</0>.
```

CODE

Служит для обозначения кода программы. По умолчанию выводит текст моноширинным шрифтом, таким как Courier New.

Атрибут:

- TITLE — служит для пояснений.

Пример:

```
<CODE TITLE="функция на JavaScript">
<PRE>
function hi(imgDocID,imgObjName)
{
  if (loaded)
    {document,images[imgDocID].src = eval (imgObjName + ".src"))
1
</PRE>
</CODE>
```

COL

Служит для обозначения столбца таблицы.

Атрибуты:

- SPAN — указывает, к какому числу столбцов относится данный элемент <COL>. Может принимать целочисленные значения, а по умолчанию этот атрибут имеет значение 1;
- WIDTH — задает ширину столбца. Этот атрибут необходим, если применяется усовершенствованная модель обработки таблиц table-layout: **fixed.**

Пример:

```
<TABLE BORDER*"1" WIDTH="400">
<COL WIDTH="150"XCOL WTDTH«"250">
<САРТЮЫ>ТаСлица 1. Список членов рабочей группj</CAPTION>
<TR>
  <TD>Имя</TD>
  <TO>Фамилия</TD>
```

```
</TR>
<TR>
  <TD>HBan</TD>
  <TD>HenoceflOB</TD>
</TR>
</TABLE>
```

COLGROUP

Обозначает группу столбцов. Есть смысл использовать, если таблица состоит из большого числа столбцов одинаковой ширины.

Атрибуты.

П SPAN — задает число столбцов в группе;

- WIDTH — залает значение ширины по умолчанию для каждого столбца данной группы.

Пример:

```
<TABLE BORDER^! WIDTH=4 00>
<COLGROUP SPAN="2" WIDTH="200">
</COLGROUP>
<САРТЮМ>Таблица 1. Список членов рабочей грухшы</САРТЮЫ>
<TR>
  <TD>Hb«</TD>
  <ТП>Фамилия</ТО>
</TR>
<TR>
  <TD>HBaH</TD>
  <TD>HenoceflOB</TD>
</TR>
</TABLE>
```

DD

Обозначает отдельный элемент списка определений (definition list). Его особенность в том. что он состоит из названия (термина), который впоследствии описывается Описание как раз и содержится в элементе <сг>. Перед элементами определяющего списка не ставится никаких маркеров, так что подобным образом можно оформлять глоссарий.

Пример:

```
<DL>
  <ГЛ"Жонтейнер</ОТ>
```

<Ш>Блоковый элемент, который содержит в себе текущий элемент</M)>
</DL>

DEL

Обозначает информацию, которая уже устарела в текущей перси и документа и будет удалена в последующих версиях. Вообще, его использовать нет особого смысла.

Атрибуты:

О CITE — обозначение URL документа, с которого взята информация;

П DATETIME — содержит дату и время удаления данной части документа.

Пример:

```
<DEL datetime="2002-04-02T11;00:00">3Та часть текста будет удалена ровно
в II часок. Так что читайте, пока есть возможность<Д)Е1.>
```

DFN

Этот элемент обозначает определение какого-либо термина.

Пример:

```
<DFNxt>M3MKa</DFTI> - это наука о природе.
```

DIR

Применяется для создания многоколоночных списков. В спецификации HTML 4.0 помечен как нежелательный, потому что практически аналогичен нетипизированному списку <UL>.

DIV

Обозначает структурный блок Создан для того, чтобы использоваться совместно с каскадными таблицами стилей.

Атрибут:

О ALIGN — служит для выравнивания содержимого блока по горизонтали.

Пример:

```
<DXV class="newsblock">
<A HREF=/news.phtml?id-765>Ba*HaH новость</A><BR>
Нет важнее новости для нас, чем сам факт наличия новости. Без новостей мы
не можем.
</DIV>
```

DL

Обозначает список определений.

Пример:

```
<DL>
  <0TЖонтейнер</0T>
  <00>Блоковый элемент, который содержит в себе текущий элеыент</00>
</DL>
```

DT

Обозначает термин в писке определений.

Пример:

```
<DL>
  •ОЭТ>Контейнер</ОТ>
  <ООБлоковый элемент, который содержит в себе текущий элемент</00>
</DL>
```

EM

Обозначает логическое выделение в тексте Часто вместо него используют элемент <i>, но это логическая ошибка.

Пример:

```
<P>Я подъял пистолет. Это был обычный кольт 32-го калибра, а вот пули
были <£M>серебряными</ЕЬ^</P>
```

FIELDSET

Позволяет визуально объединять отдельные поля формы посредством рамки вокруг них.

Пример:

```
<FIELDSET>
  <FOBM ACTION="/sender.php">
    Имя -<BR>
    <INPUT NAME="name" TYPE="text"><BR>
    Сообщение ~<BR>
    <tEXTAREA Ш*Е«"тезз" COLS-"20" R0WS»"6"></TEXTAREAx3R>
```

```
    <INPUT TYPE="5ubmit" УАЬПЕ="Отослать">
  <FORM>
</FIELDSET>
```

FONT

Служит для задания начертания шрифта, его цвета и размера. Помечен как *неже. lam& lьный,* причем его использование *крайне нежелательно.* Весь контроль за шрифтом *необходимо* осуществлять с помощью CSS.

FORM

Служит для создания различного рода форм.

Атрибуты:

- ACTION — содержит URL скрипта, который будет обрабатывать данные, введенные в поля формы;
- METHOD — определяет метод, с помошью которого данные будут отправляться на сервер. Возможные значения GET И POST;
- ENCTYPE — определяет тип данных, которые будут отсылаться на сервер. Нужен только при использовании метода POST. Значение по умолчанию application/x-www-form-urlencoded. При использовании <INPUT TYPE="file"> надо указать ENCTYPE="multipart/form-data";

О ACCEPT — с помошью этого атрибута задается список типов контента, который может приниматься сервером.

Пример:

```
<FORM ACTXON="/sender.php" METHOD="POST">
  Имя -<BR>
  <INPUT NAME="name" TYPE="text"><BR>
  Сообщение —CBR>
  <TEXTAREA NAME="mess" COLS="20" ROWS="6"></TEXTAREA><BR>
  <INPUT TYPE="submit" VALUE="OTOcnaTb">
<FORM>
```

FRAME

Определяет содержимое и внешний вид отдельного фрейма.

Атрибуты:

П NAME — задает имя фрейма. Оно необходимо для задания ссылок на документы таким образом, чтобы документы открывались в данном фрейме;

- SRC — задет URL документа, который будет загружаться в данном фрейме первоначально;
- NORESIZE — запрещает пользователю изменять размеры фрейма;

П SCROLLING — устанавливает состояние полос прокрутки:

- auto — полосы прокрутки будут пояа!яться только в том случае, если размеры контента превышают размеры самого фрейма;
- уеь — полосы прокрутки будут присутствовать всегда;
- по — полос прокрутки не будет ни при каких обстоятельствах;

П FRAMEBORDER — устанавливает или убирает разделительную линию между соседними фреймами. Если значение 0, то линии не будет, если 1, то ли-ния будет;

О MARGINWIDTH — задает левое и правое поля для фрейма;

- MARGINHEIGHT — задает верхнее и нижнее поля для фрейма.

Пример:

```
<FRAMESET  COLS^"50?,50«">
  <FRAME  src="left.html"  FRAMEBORDER="0"  SCROLLING="no">
  <FRAME  src="right.html"  FRAMEBORDER="0"  SCROLLING="no">
</FRAMESET>
```

FRAMESET

Разбивает окно браузера на фреймы, а уже затем каждый отдельный фрейм описывается элементом <FRAME>.

Атрибуты:

О COLS — разбивает окно на колонки. В качестве значений выступает список ширин колонок, разделенных запятыми;

- ROWS — разбивает окно на горизонтальные блоки. В качестве значений выступает список высот блоков, разделенных запятыми.

Пример:

```
<FRAMESET  ROWS="30%,30%, *">
   •^FRAME SRC="top.html">
  <FRAME  SRC-"center.html">
  <FRAME  SRO"bottom.htral">
</FRAMESET>
```

H1, H2, H3, H4, H5, H6

Служат для обозначения заголовков разного уровня. Самый важный (самый большой) заголовок обозначается элементом <HI>, а самый маленький — элементом <нб>.

Атрибут:

О ALIGN — задает горизонтальное выравнивание заголовка.

Пример:

```
<H1>Как верстают сайтьК/H1>
<P>Для того чтобы прилично сверстать сайт, мало знания языка HTML, надо уметь применять глубокие знания на практике. Любая теория без утилитарного применения остается всего лишь теорией.</P>
<H2>Визуальное представление информации средствами HTML</H2>
<P>Дпя того чтобы понять, зачем, собственно, разрабатывали и внедряли технологию CSS, надо детально разобраться в языке HTML, знать все его слабые и сильные места, прочувствовать его на реальных примерах</P>
```

HEAD

Обозначает секцию документа, в котором находятся метаданные и прочая служебная информация, которая не отображается браузером явно.

Пример:

```
<HTML>
<HEAD>
  <T1T1£>Профессиональная верстка</ТПЪЕ>
  <LINK REL*"stylesheet" HREF="/main.css">
  <5CRIPT TYPE="text/javascript" SRC="/script.js"x/script>
</HEAD>
```

HR

Визуальный элемент, который отображается браузером в виде горизонтальной полоски. Имеет атрибуты ALIGN, NOSHADE, SIZE, WIDTH, которые все помечены как нежелательные для использования. Вместо них внешний вид элемента <HR> надо задавать с помощью CSS

Пример:

```
<STYLE TYPE="text/css">
  HR (color: #000; width: 50%; text-align: left; height: lpx}
</STYLE>
. . .
<P>Вчера все было хорошо.</P>
<HR>
<P>Позавчера было пасмурно, так что дворник улицу не подметал. Предлагаю объявить ему строгий выговор</P>
```

HTML

Обозначает начало HTML-документа.

Пример:

```
<HTML>
. . .
</HTML>
```

I

Текст выводится стилем italic.

Пример:

Признаться, я удивился <1>ничуть не меньше</1> того бармена: серебряные доллары не каждый день растворяются в воздухе, но все же это не повод палить в людей!

IFRAME

Позволяет встраивать в документ другой документ.

Атрибуты-

- NAME — задаст имя фрейма, которое необходимо для задания ссылок на документы таким образом, чтобы док>менты открывались в данном фрейме;
- SRC — задает URL документа, который будет зафужаться в данном фрейме первоначально;

П WIDTH — задает ширину фрейма;

- HEIGHT — задает высоту фрейма;
- SCROLLING, FRAMEBORDER, MARGINWIDTH, MARGINHEIGHT — СОВСршеННО **ЗНа-**

логичны атрибутам элемента <FRAME>.

Пример:

```
<IFRAME SRC="inc.html" WIDTH="350" HEIGHT="400" SCROLLING-="no"
FRAMEBORDER="0">
Ваш браузер-то IFRAMR не поддерживает. Уж не Netscape 4 ли это? В утиль
его[1]
</IFRAME>
```

IMG

Позволяет встраивать в документ изображения.

Атрибуты:

- SRC — задает URL графического файла, который должен быть встроен в документ;
- ALT — задает текст, который будет выводиться на месте изображения в том случае, если рисунки отключены в браузере;
- TITLE — содержит краткое описание изображения;

П WIDTH — задает ширину изображения. Обязательно надо указывать при табличной верстке;

П HEIGHT — залает высоту изображения. Обязательно надо указывать при табличной верстке;

- BORDER — устанавливает ширину рамки вокруг изображения. Если изображение является ссылкой, то для того, чтобы рамки не было, надо обязательно прописать BORDER="0". Если изображение ссылкой не является, то рамки отсутствуют по умолчанию;

П ALIGN — указывает, как текст должен обтекать рисунок. Данный атрибут является нежелательным, а вместо него лучше использовать плавающую модель CSS;

О HSPACE — указывает величину верхнего и нижнего полей изображения. Помечен как нежелательный, а вместо него рекомендуется использовать свойство каскадных таблиц стилей margin;

- WSPACE — указывает величину левою и правого полей изображения. Помечен как нежелательный, а вместо него рекомендуется использовать свойство каскадных таблиц стилей margin.

Пример.

```
<IMG SRO"A/anons.gif'' WIDTH="34" HEIGHT="35" BORDER="0" ALT="aHOHC горячей новости">
```

INPUT

С помощью данного элемента можно рсатизовать большое количество различных элементов форм.

Атрибуты:

О TYPE — фактически значение этого атрибута определяет элемент формы:

- text — стандартное текстовое поле;
- password — текстовое поле для ввода пароля;

- checkbox — переключатель, который может иметь два значения (выбран — не выбран);
- radio -~ переключатель, как и в случае checkbox, но с помощью radio можно организовать выбор из нескольких вариантов;
- subr.it — кнопка, нажатие на которую отсылает содержимое формы на сервер;
- reset — кнопка, нажатие на которую уничтожает всю ранее введенную в ноля формы информацию;
- hidden — служебное поле, которое не отображается браузером;
- image — кнопка типа submit, но вместо обычной кнопки будет рисунок;
- button — кнопка, действие на которую назначается с помощью скриптового языка;

D MAXLENGTH — задает максимальное число символов, которое может ввести пользователь;

O VALUE — начальное значение поля;

O SIZE — задает ширину поля на экране;

D READONLY — указывает, что из поля данные могут только читаться, но не изменяться;

O ACCESSKEY, TABINDEX.

Пример:

```
<ЪХШ ACTION="/sender.php" METHOD="POST">
  Имя -<BR>
  <INPUT NAME="name" TYPE="text" SIZE="12"XBR>
  Вариант ответа -<BR>
  <INPUT TYPE="radio" NAME="answer" VALUE="yes">fla<BR>
  <INPUT TYPE="radio" NftME="ans\u'r" VALUE="no">???<BR>
  <INPUT TYPE="sufcmit" УАШЕ="Ответить">
<FORM>
```

INS

Обозначает текст, который появился в текущей новой версии документа.

Атрибуты:

П CITE— обозначение URL документа, с которого взята информация;

- DATETIME — содержит дату и время добавления данной части документа.

Пример:

```
<INS DATXTIME="2001-02-01T12:30:00">Согласно новым данным, прошлая теория совершенно неверна!</INS>
```

ISINDEX

С помошыо данного элемента можно сделать строку для ввода информации. Помечен как нежелательный, так что вместо него нужно использовать элемент <INPUT> TYPE="text".

KBD

Обозначает текст, который нужно ввести пользователю.

Пример:

```
Для входа в рабочую область введите <KBD>user</KBD> в поле "Пользователь"
```

LABEL

Служит для ассоциации элемента, например, текста, с полем формы.

Атрибут:

О FOR — содержит ссылку на ID элемента формы, с которым надо ассоциировать текст.

Пример;

```
<FORM>
  <LABEL for="yourname">???: </LABEL>
  <INPUT TYPE="text" NAME*"firstname" ID="yourname">
</FORM>
```

LEGEND

Формирует название формы, которая обведена рамкой с помощью элемента <FIELDSET>. Имеет атрибут ALIGN, который помечен как нежелательный.

Пример:

```
<FIELDSET>
  <LEGEND>Bame сообвдение</ЬЕ(ЗЕМЭ>
  <FCRM ACTION-'Vsendcr.php" METHOD="POST">
    Имя: <BR>
    <INPUT NAME-"name" TYPE-"text"><BR>
```

```
    Сообщение: <BR>
    <TEXTAREA name="mess" COLS="20" ROMS="6"X/TEXTAREA><BR>
    <INPUT TYPE="submit" VALUE="OrocnaTb">
  <FORM>
</FIELDSET>
```

LI

Элемент списка.

Атрибут:

О TYPE, VALUE, COMPACT — все эти атрибуты помечены как нежелательные, а контроль над визуальным представлением элементов списка должен осуществляться с помощью CSS.

Пример.

```
<P>Черты разносторонне развитой личности —</P>
<UL>
 <Ы>Эрудированност ь</Ы>
 <Ы>Образованность</Ы>
 <LI>Ум</LI>
</UL>
```

LINK

Формирует связь (соотношение) данного документа с каким-либо другим. Может находиться только в секции <HEAC>. Например, с помощью этого элемента подключается файл с таблицами стилей.

Атрибуты:

П MEDIA — определяет тип устройства, которое может обрабатывать связанное содержимое:

- screen — экран монитора;
- tty — устройства с фиксированной шириной букв. Такие как терминалы;
- tv — устройства телевизионного типа;
- projection — проекторы;
- handheld — портативные устройства, такие как Palm;
- print — печатающие устройства, такие как принтер;
- braille — устройства вывода на основе азбуки Брайля (для слепых);

- aural — звуковые устройства вывода;
- all — все возможные устройства;

• HREF — содержит URL связанного документа;

П REL, REV, HREFLANG.

Пример:

<LINK TITLE="Англоязычная версия списка оборудования" TYPE="text/html" REL="alternate" HREFIANG="eng" HR£F="http.//www.firm,com/eng/lxst.html">

MAP

Служит для создания изображений-карт.

Пример:

```
<MAP NAME="main">
<AREA АЬТ="Главиал страница" TITLE="На главную" COORDS="20,16,76,67"
HREF="/index.phtml">
</MAP>
```

MENU

Служит для создания одноколоночного меню, но помечен как нежелательный для использования, т. к. аналогичного эффекта можно добиться с помощью элемента <UL>.

META

Задает метаданные документа, такие как ключевые слова, описание и др.

Атрибуты:

П NAME — содержит имя метаданных;

• CONTENT — содержит сами метаданные;

О HTTP-EQUIV — может использоваться вместо атрибута NAME. Содержит информацию о HTTP-запросе.

Примеры:

• `<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1251">`

О `<META NAME="keywords" CONTENT="ALT CSS DHTML HTML IE JavaScript MSIE Netscape Opera tutorial">`

NOFRAMES

Залает контент, который должен отображаться теми браузерами, которые не поддерживают фреймы. Ввиду того, что таких практически не осталось, использовать этот элемент нецелесообразно.

NOSCRIPT

Задает контент, который должен отображаться браузерами, не поддерживающими скриптовый язык.

Пример:

```
<SCRIPT TYPE="text/javascript" SRC-"/js . js"x/SCRIPT>
<NOSCRIPT>
Скрипт по какой-то причине не отработал.<BR>
<A HRF,F="/index.html">BXQfl</A>
</NOSCRIPT>
```

OBJECT

Служит для встраивания в документы различных данных, начиная от изображений и заканчивая другим документом.

Атрибуты:

- CLASSID — содержит URL описания типа вставляемого объекта или может служить заменой атрибута DATA;
- CODETYPE — содержит MIME-тип объекта;
- DATA — можно использовать для задания URL объекта;
- STANDBY — содержит текст, который может отображаться браузером, пока загружается объект;
- WIDTH, **HEIGHT, BORDER.**

Пример:

```
<OBJECT DATA="insert.gif' TYPE="±mage/gif" WIDTH="300" HEIGHT="100">
```

OL

Служит для создания нумерованных списков. Имеет атрибуты COMPACT, START и TYPE, которые помечены как нежелательные. Атрибуты с успехом могут быть заменены каскадными таблицами стилей.

Пример:

```
<P>Последовательность действий приготовления курицы </P>
<OL>
  <LI>Поймать курицу</LI>
  <LI>Ощипать</LI>
  <LI>Приготовить</LI>
</OL>
```

OPTGROUP

Позволяет разбить элементы выбора в селекторе на логические группы, что облегчает поиск нужного поля.

Атрибуты:

D DISABLED — при наличии этого атрибута выбор из данной группы осуществить будет невозможно;

O LABEL — задает название группы, которое будет отображаться в селекторе.

Пример:

```
<SELECT NAME="car">
  <C?T:ON VATXT, ^"::cne">None</OPTION>
  <OPTGROUP LAB£L="BA3">
    <OPTION VALUE »"v2110">2110</OPTION>
    <OPTION VALUE ~"v2109">2109</OPTION>
  </OPTGROUP>
  <OPTGROUP LABEL="BMW">
    <OPTION •.•AUi.-'1r;rr.wz3">2-3</OPTION>
    <OPTION VALUE ="bnw514">514</OPTION>
  </OPTGROUP>
</SELECT>
```

OPTION

Устанавливает поле в селекторе.

Атрибуты:

O SELECTED — если этот атрибут задан, то данное поле будет выбрано по умолчанию;

O DISABLED — при наличии этого атрибута выбор данного поля невозможен;

П LABEL — задает название поля. Браузер должен в селекторе выводить **именно его,** а не содержимое элемента;

- VALUE — задает начальное значение данного поля.

Пример:

```
<SELECT NAME="car">
  <OPTION VALUE="v2110" SELECTED>21I0</OPTION>
  <OPTION VALUE="v2109">21C9</OPTION>
</SELECT>
```

P

Обозначает абзац текста. Имеет атрибут ALIGN, который помечен как нежелательный для использования.

Пример:

```
<P>Прогуливаясь по улице, я всё никак не мог увязать пропавший бакс с
серебряными пулями. Впрочем, мало кто на свете способен это сделать,
только это меня и успокаивало.</P>

<P>Пошёл мелкий противный дождь, и я уединился в маленьком уютном кафе по
соседству с большим отелем. Подкидывая на ладони последний доллар, я за-
казал кофе и кусок орехового пирога, который оказался даже вкуснее, чем я
надеялся, хотя кофе был отвратительным.</P>
```

PARAM

Служит для задания значений, которые могут быть нужны во время выполнения встроенного объекта.

Атрибуты:

О NAME — задает имя выполняемого параметра, которое должен понимать встроенный объект;

- VALUE — задает соответствующее атрибуту NAME **значение;**

П VALUETYPE — задает тип атрибута VALUE:

- data — стандартный тип данных, которые будут переливаться объекту как строка;
- **ref** — указывает на то, что значение атрибута VALUE представляет собой URL;
- object: — указывает на то, что значение атрибута VALUE представляет собой ID объекта, который уже включен в данный документ;

- TYPE — определяет MIME-тип ресурса только в том случае, если VALUE="ref".

Пример:

```
OBJECT  DATA="insert.gif"  type»"image/gif">
  <PARAM NAME="heaght" VALUE="100" VALUETYPE="data">
  <PARfIM 1WME="width" VALUE="3000" VALUETYPE="data">
</OBJECT>
```

PRE

Указывает браузеру, чго текст внутри элемента уже отформатирован, т. е. надо сохранить имеющиеся переводы строк и пробелы. Имеет атрибут WIDTH, который помечен как нежелательный.

Пример:

```
<PRE>
.bg  f
   background-image — url(iZback2.gif);
   background-repeat  -  repeat-y
 l
</PR£>
```

Q

Служит для выделения коротких цитат.

Атрибут:

П CITE — в этом атрибуте указывается адрес документа, с которого была взята цитата.

Пример:

```
<Q cite="http: //www. f iref alcon . com/silver/silver. html">Хорошо, начало
интересное,  я уже с нетерпением ждал продолжения,  поглаживая в кармане
рукоять трофейного кольта с пулями из злополучного серебра.</Q>
```

S, STRIKE

Текст внутри этих элементов выводится перечеркнутым. Помечены как нежелательные для использования, а вместо них рекомендуется использовать CSS.

SAMP

Обозначает результат (вывод) работы программы, скрипта и пр.

Пример;

```
Если все отработает нормально, то на экране появится надпись <SAMP>OK.
Press any key<:/SAMP>
```

SCRIPT

С помощью данного элемента в документе размещаются скрипты.

Атрибуты:

- SRC — задает URL внешнего подключаемого скрпптового файла;
- TYPE — задает скриптовый язык;
- D DE:TEP — указывает браузеру, что скрипт не генерирует контента, т. е. не содержит, например, document.write (j, вследствие чего браузер может продолжить загрузку остальной страницы, а скрипт загрузиib в конце.

Пример-

<SCRIPT TYPE="text/}avascript" SRC*"script. 3s[11] DEFERX/scnpt>

SELECT

Создает селектор.

Атрибуты:

- NAME — имя селектора, когорое используется обрабатываюшим скриптом на сервере;
- SIZE — можно указывать, сколько строк текста будет видно По умолчанию SIZE="1";
- O MULTIPLE — указывает, что можно производить выбор нескольких значений одновременно.

Пример:

и-

```
<SELECT NAME="car" SI2E="2">
  <OPTION VALUt:="v2110" SELECTED>2110</OPTION>
                                          c/o, ,.
  «DPTTON VALUE="v2109">2109</OPTION>
</SELECT>
```

SMALL

Выводит текст мелким шрифтом.

Пример:

```
<P>Мы любим только <SMALL>ManeHbfoie</SMALL> автомобили</P>
```

SPAN

Обозначает строчным блок. Создан для того, чтобы использоваться совместно с каскадными таблицами стилей.

Пример:

```
<DIV class="newsblock">
Нет важнее новости для нас, чем сам факт наличия новости. Без новостей мы не можем.<BR>
<SPAlJ> 12.01.2002/</SPAN>
</DIV>
```

STRONG

Обозначает сильное логическое выделение в тексте.

Пример:

```
<P>Если нельзя, но <STRONG>"-:-::-</STRONG> хочется, то можно</P>
```

STYLE

Служит для задания таблицы стилей. Может находиться только в секции <HEAD>.

Атрибуты:

П TYPE — задает язык стилей;

- MEDIA — задает тип устройства, для которого предназначена таблица стилей.

Пример:

```
<STYLE TYP£="text/ess" MEDIA="screen">
  BODY {
    background-color -r #FFF;
    margin ~ Opx;
    font — 0.7em Verdana, Tahoma, sans-serif}
</STYLE>
```

SUB

Формирует подстрочный символ.

Пример:

```
<P>Формула этилового спирта: <BR>C<SUB>2</SUB>H<SUB>5</SUB>OH</P>
```

SUP

Формирует надстрочный символ.

Пример:

```
<P>Знаменитая формула Эйнштейна: <BR>E=mc<SUP>2</SUP></P>
```

TABLE

Служит для создания таблицы.

Атрибуты:

П SUMMARY — содержит краткое описание таблицы. Может использоваться визуальными браузерами для предоставления дополнительной информации;

О ALIGN — служит для задания обтекания таблицы. Помечен как нежелательный, а вместо него лучше использовать плавающую модель CSS;

- WIDTH — задает ширину таблицы;
- RULES — задает, какие линейки будут между ячейками таблицы:
  - пог.е — никаких линеек не будет. Это значение по умолчанию;
  - groups — линейки будут между элементами <THEAD>, <TBODY>, <COLGROUP>, <COL>;
  - rows — линейки будут между рядами таблицы;
  - cols — линейки будут между столбцами таблицы;
  - аи — линейки будут всюду;

П FRAME — задает, какая часть области, окружающая таблицу, будет видна. По сути, делает невидимыми линейки (но только окаймляющие таблицу), заданные с помощью атрибута RULES:

- void — все стороны будут видны;
- above — видимой будет только верхняя линейка;
- below — видимой будет только нижняя линейка;
- hsides — видимыми будут только верхняя и нижняя линейки;

- vsides — видимыми будут только левая и правая линейки;
- lhs — видимой будет только левая линейка;
- rhs — видимой будет только правая линейка;
- box — видимыми будут все четыре линейки;

- BORDER — задает ширину рамок между ячейками таблицы. Если значение атрибута равно нулю, то рамок не будет;

O CELLSPACING — задает ширину пространства между ячейками;

- CELLPADDING — задает ширину отступов внутри ячеек,

O BACKGROUND, BGCOLOR — служат для задания фонового рисунка и цвета фона. Оба помечены как нежелательные, а вместо пи\ рекомендуется использовать соответствующие свойства CSS.

Пример.

```
<TABLE WIDTH=209 BORDER=0 CELLPADDING=O CELLSPACING=0>
  <TR>
    <TD>
      Первый столбец
    </TD>
    <TD>
      Второй столбец
    </TD>
  </TR>
</TABLE>
```

TBODY

Обозначает фуппу строк, которые образуют основную часть таблицы. Его использование особого смысла не имеет.

Пример:

```
<TABLE WIDTH-50%>
  <CAPTION>CnMCOK украденных вещей у гражданина Простофилева</CAPTЮM>
  <TR>
    <TH>Номер</TH>
    <TH>Украденная вешь</TH>
  </TP>
  <TBODY>
  <TR>
    <TD>K/TD>
```

```
    <TD>ribinecoc</TD>
  </TR>
  <TR>
    <TD>2</TD>
    <T0>ХОЛОДИЛЬНИК</T0>
  </TR>
  </TBODY>
</TABLE->
```

TD

Обозначает отдельную ячейку таблицы.

Атрибуты:

П AXES — служит для отнесения ячейки к определенной категории, посредством чего можно формировать оси в /7-мерном пространстве;

О ROWSPAN — служит для объединения ячеек по вертикали;

- COLSPAN — служит для объединения ячеек по горизонтали;
- WIDTH, HEIGHT — задают ширину и высоту ячейки соответственно. Помечены как нежелательные для использования;

П HEADRES — содержит список 1С ячеек, предоставляющих заголовочную информацию для текущей ячейки;

- SCOPE — определяет ячейки, для которых заголовочная информация задается текущим заголовком. Принимает значения row, col, rowgroup, colgroup.

Пример:

```
<TABLE WIDTH=50%>
<TBODY>
  <TR>
    <TD C0LSPAN=2>06niaH ячейка</TO>
  </TR>
  <TR>
    <TO>Имя</TT)>
    <TO>Фамилия</ГО>
  </TR>
</TBODY>
</TABLE>
```

TEXTAREA

Создает многострочное поле ввода. Оптимально подходит для использования в тех случаях, когда надо ввести достаточно большой объем текста.

Атрибуты:

- Rows — задает число отображаемых в браузере строк (фактически высоту элемента <TEXTAREA>);
- COLS — задает ширину элемента <TEXTARSA> в символах;
- DISABLED, READONLY, TABINDEX.

Пример:

```
<FORM ACTION=*ysender.php" METHOD="PO5T">
  Имя -<BR>
  <INPUT NAME="name" TYPE="text"><BR>
  Сообщение —<BR>
  <TEXTAREA NAME="iress" COLS*"20" ROWS="6"X/TEXTAFEA><BR>
  <INPUT TYPE="submit" УАЩЕ=»"0тослать">
<F0RM>
```

TFOOT

Обозначает группу строк, которые образуют нижнюю часть таблицы. Его использование особого смысла не имеет.

TH

Обозначает, что ячейка содержит заголовок. Например, заголовок столбца. Атрибуты совершенно аналогичны атрибутам элемента <TD>.

Пример:

```
<TABLE WIDTH=50%>
  <CAPTION>CnwcoK украденных вещей у гражданина Простофилева</CAPTTOЫ>
  <TR>
    <TH>Номер</TH>
    <TH>Украденная вещь</TH>
  </TR>
  <TR>
    <TD>K/TD>
    <TD>ribUiecoc</TD>
  </TR>
</TABLE>
```

THEAD

Обозначает группу строк, которые образуют заголовок таблицы. Его использование особого смысла не имеет.

TITLE

Заголовок HTML-документа. Очень важный элемент с точки зрения поисковых систем. Каждый документ *обязан* содержать элемент <TITLE>.

Пример:

```
<TITLE>WEB-aHaTOMMfl [ HTML I CSS I JavaScript I DHTML ]</TITLE>
```

TR

Обозначает строку таблицы.

Атрибут:

П BGCOLOR, ALIGN — помечены как нежелательные, а вместо них рекомендуется пользоваться таблицами стилей.

Пример:

```
<TABLE>
  <TR>
    <T0>Первая ячейка</T0>
    <T0>Вторая ячейка</TЭ>
  </TR>
  <TR>
    <T0>Третья ячейка</TO>
    <TO>Четвертая ячейка</T0>
  </TR>
</TABLE>
```

TT

Отображает текст моноширинным шрифтом, как пишущая машинка.

Пример:

```
<P>Вот так раньше <TT>печатали текст на печатных машинках</TTX/P>
```

U

Отображает текст с подчеркиванием. Помечен как нежелательный для использования, а вместо него рекомендуется использовать свойство CSS text-decoration: underline.

UL

Ненумерованный список. Имеет те же атрибуты, что и <OL>, И все они являются нежелательными для использования.

Пример:

```
<P>Без добра не бывает: </P>
<UL>
  <Ы>худа</Ы>
  <Ы>ничего не бывает</Ы>
</UL>
```

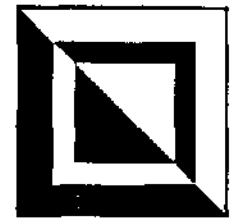
VAR

Обозначает переменную, например, компьютерной программы.

Пример:

```
<P>А вот про переменную <VAR>$item</VAR> мы совсем забыши</P>
```

Приложение 2



Поддержка браузерами элементов HTML

В приложении приведены данные о поддержке наиболее важных элементов HTML основными браузерами.

Таблица П2.1. *Аббревиатуры*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<ABBR>	**Нет**	Нет	**Да**	**Да**	По совершенно непонятной причине поддержку этого элемента в браузеры IE так и не ввели
<ACRONYM>	Да	Да	Да	**Да**	

Таблица П2.2. Визуальные элементы

Элемент	IE 5	IE6	Netscape 6	Opera 6	Комментарий
<SPACER>	Нет	Нет	Нет	Нет	
<HR>	**Да**	**Да**	**Да**	Да	
<HR> COLOR	**Да**	**Да**	**Нет**	Нет	
<HR> NOSHADE	**Да**	**Да**	**Да**	Нет	

Таблица П2.3. *Элементы для редактирования текста*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<INS>	**Да**	**Да**	**Да**	**Да**	
<INS> CITE	**Нет**	**Нет**	Да	**Нет**	Чтобы узнать URL, откуда взята цитата, надо на элементе <.;Ni> щелкнуть правой кнопкой мыши и выбрать в контекстном меню пункт **Свойства**

Таблица П2.3 (окончание)

Элемент	IE5	IE 6	Netscape 6	Opera 6	Комментарий
<INS> DATETIME	Нет	Нет	Да	Нет	
<DEL>	Да	Да	Да	Да	
<DEL> CITE	Нет	Нет	Да	Нет	Чтобы узнать URL, откуда взята цитата, надо на элементе **O E L** > щелкнуть правой кнопкой мыши и выбрать в контекстном меню пункт Свойства
<DEL> DATETIME	**Нет**	Нет	Да	**Нет**	

ТаблицаП2.4. *Цитаты*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<Q>	Нет	Нет	Частично	Частично	Netscape и Opera просто ставят двойные кавычки, хотя это не всегда верно Например, при вложенности элементов •Q> одинарные кавычки не ставятся
<Q> CITE	Нет	Нет	Да	**Нет**	Чтобы узнать URL, откуда взята цитата, надо на элементе <Q> щелкнуть правой кнопкой мыши и выбрать в контекстном меню пункт **Свойства**
<BLOCKQUOTE>	Да	Да	Да	Да	
<BLOCKQUOTE> CITE	Нет	Нет	Да	Нет	Чтобы узнать URL, откуда взята цитата, надо на элементе <3L0CKQU0TE> щелкнуть правой кнопкой мыши и выбрать в контекстном меню пункт **Свойства**

Таблица П2.5. *Формы*

Элемент	IE 5	IE6	Netscape 6	Opera 6	Комментарий
<BUTTON>	Да	Да	Да	Нет	Opera много теряет в функциональности из-за того, что не поддерживает элемент <BUTTON>
<FIELDSET>	Да	Да	Да	Нет	
<INPUT> DISABLED	Да	Да	Да	Да	
<INPUT> READONLY	Частично	Частично	Частично	Да	Все браузеры, кроме Opera, запрещают изменять содержимое текстовых полей, но не запрещают изменять состояние checkbox и переключателей
<INPUT> TABINDEX	Да	Да	Да	Нет	
<INPUT> ACCESSKEY	Да	Да	Да	Нет	
<INPUT> ALT	**Нет**	**Нет**	**Нет**		
<INPUT> TITLE	Да	Да	Да		По сути дела является заменителем <TNr";: Я1Т-[iM]>, так что поддержка предыдущего атрибута непринципиальна
<LABEL>	Частично	Частично	Частично	Нет	Если щелкнуть на элементе, то Netscape передаст фокус только в том случае, если метка указывает на checkbox или переключатель
<LABEL> FOR	Да	Да	Частично	Нет	Internet Explorer вообще не ассоциирует метку с элементом формы, если не задан атрибут **гс?,**
<LABEL> ACCESSKEY	Нет	Нет	Частично	Нет	Фокус не передается для элемента <SELECT>
<LEGEND>	Да	Да	Да	Нет	

ТаблицаП2.5(окончание)

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<OPTGROUP>	Нет	Частично	Да	Нет	Internet Explorer 6 **не** поддерживает атрибут DISABLED
<OPTGROUP> DISABLED	Нет	Нет	Да	Да	
<OPTION> LABEL	Нет	Нет	Нет	Нет	
<OPTION> DISABLED	Нет	Да	Да	Да	
<SELECT> DISABLED	Да	Да	Да	Да	
<TEXTAREA> DISABLED	Да	Да	Да	Да	
<TEXTAREA> READONLY	Да	Да	Да	Да	
<TEXTAREA> TABINDEX	Да	Да	Да	Нет	

ТаблицаП2.6. Таблицы

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<TABLE> FRAME	Да	Да	Да	**Нет**	Internet Explorer не использует стандартные значения, если установлен один из атрибутов FRAME ИЛИ RULES, а не оба
<TABLE> RULES	Да	Да	Нет	Нет	Internet Explorer не использует стандартные значения, если установлен один из атрибутов FRAME **или** RULES, **а** не оба
<TABLE> SUMMARY	Нет	Нет	Да	Нет	
<THEAD>	Да	Да	Да	**Да**	
<TFOOT>	Да	Да	Да	**Да**	
<TBODY>	Да	Да	Да	**Да**	

*Таблица **П2.6** (окончание)*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<COLGROUP>	Да	Да	Да	Нет	
<COLGROUP> SPAN	Да	Да	Да	**Нет**	
<COLGROUP> WIDTH	Да	Да	Да	**Нет**	
<COL>	Да	Да	Частично	Нет	Netscape не позволяет применять стили к столбцу таблицы, что сильно ухудшает функциональность этого элемента
<TD> ABBR	Нет	Нет	Нет	Нет	
<TD> AXIS	Нет	Нет	Нет	Нет	
<TD> HEADERS	Нет	Нет	Нет	Нет	
<TD> SCOPE	Нет	Нет	Нет	Нет	

*Таблица **П2.7.** Ссылки*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<A> TYPE	Нет	Нет	Да	Нет	
<A> REL	Нет	Нет	Да	Нет	
<A> REV	Нет	Нет	Да	Нет	
<A> HREFLANG	Нет	Нет	Да	Нет	
<IMG> LONGDESC	Нет	Да	Да	Нет	Если навести курсор на рисунок вызвать контекстное меню правой кнопкой и выбрать пункт **Свойства** то там будет содержаться URL с подробным описанием рисунка

Таблица П2.8. *Объекты*

Элемент	IE 5	IE 6	Netscape 6	Opera 6	Комментарий
<OBJECT> TYPE	Нет	Нет	Да	Да	
<OBJECT> (Рисунки)					Internet Explorer некорректно отображает страницы, в которых рисунки вставлены посредством элемента OBJECT. Если размеры рисунка не заданы, то он вообще не отображается
<OBJECT> (Текст)	Частично	Частично	Нет	Да	
<OBJECT> (HTML-документ)	Частично	Частично	Нет	Да	
<OBJECT> (Java-апплеты)	Частично	Частично	Да	Да	
<OBJECT> STANDBY	Да	Да	Нет	Нет	
<A> SHAPE	Нет	Нет	Да	Нет	
<A> COORDS	Нет	Нет	Да	Нет	

Приложение 3

Свойства CSS

Данное приложение содержит краткое описание большинства свойств из спецификации CSS-2, за исключением звуковых таблиц стилей и некоторых других, важность которых весьма сомнительна.

background

Краткая форма записи свойств фона. Включает в себя свойства background-color, background-image, background-repeat, background-attachment, background-position.

Пример:

```
BODY (                                                    /
  background: url{i/dos.gif) repeat-y 33% 0%)
```

background-attachment

Задает, является фоновое изображение фиксированным или нет. Если оно фиксировано, то будет оставаться всегда в одном и том же положении, и не будет скроллироваться вместе с остальным контентом страницы.

Значения:

П scroll — фоновое изображение не фиксированное. Значение по умолчанию;

• fixed -— фоновое изображение фиксированное.

Пример:

```
BODY (
  background-image: url(i/bg.gif);
  background-attachment: fixed}
```

background-color

Устанавливает фоновый цвет элемента.

Пример:

```
BODY {
  background-color:  #FFF}
```

background-image

Устанавливает фоновый рисунок для элемента.

Пример:

```
BODY {
  background-image: url(i/bg.gif) ^
```

background-position

Устанавливает смещение фонового изображения относительно контейнера. Первое значение залает смешение относительно левого края, а второе значение задает смещение относительно верхнего края.

Пример:

```
BODY {
  background-image:  url(i/bg.gif);
  background-position: 50% 0%}
```

background-repeat

Устанавливает условия размножения фонового изображения для элемента.

Значения:

O repeat — фоновое изображение размножается по обеим осям. Значение по умолчанию;

- repeat-x — фоновое изображение размножается только по оси x;
- repeat-y — фоновое изображение размножается только по оси y;

D no-repeat — фоновое изображение не размножается вообще.

Пример:

```
BODY {
  background-image: url(i/bg.gif);
  background-repeat: no-repeat!
```

border

Сокращенная форма записи для установки свойств рамок. Включает в себя свойства border-width, border-style, border-color.

Пример:

```
#news {
  border: lpx solid #0001
```

border-collapse

Задает модель отображения рамок в таблице.

Значения:

- separate — рамки лля каждой ячейки отображаются отдельно, т. е. если задать рамку толщиной 1 пиксел для всех ячеек, то при нулевом значении атрибута CELLSPACING толщина рамок вокруг ячеек б\дет 2 пиксела. Значение по умолчанию;
- collapse — рамки между отдельными ячейками будут сливаться, т. е. ее-ли задать толщину рамки для ячейки 1 пиксел, то такой она и будет.

Пример:

```
TABLE {
  border: lpx solid #000;
  border-collapse: collapse}
TD {
  border: lpx solid #000}
```

border-color

Задает цвет четырех сторон рамки. Можно указывать как одно значение, которое будет применено ко всем четырем сторонам, так и разные значения через пробел, которые будя применяться для каждой стороны в отдельности.

Пример:

```
DIV {
  border: 2px dotted #0001
DIV.addon {
  border-color: #F00 #000)
```

border-spacing

Задает величину расстояния между ячейками таблицы. Имеет смысл только В ТОМ случае, если СВОЙСТВО border-collapse имеет значение separate.

Пример:

```
TABLE {
  border: lpx solid #000;
  border-spacing: 5px}
TD (
  border: lpx solid #000)
```

border-style

Задает стиль четырех сторон рамки. Можно указывать как одно значение, которое будет применено ко всем четырем сторонам, так и разные значения через пробел, которые будут применяться для каждой стороны в отдельности.

Значения:

- none — рамки не будет;
- dotted — линия из точек;
- dashed — пунктирная линия;
- solid — сплошная линия;
- double — две сплошные линии;
- groove, ridge, inset, outset — различного рода псевдотрехмерные рамки.

Пример:

```
DIV {
  border: 1px solid #000;
DIV.note {
  border-style: double}
```

border-top, border-right, border-bottom, border-left

Сокращенные формы записи для верхней, правом, нижней и левой рамок соответственно. Формат совершенно аналогичен формату свойства border, но вообще использование этих свойств в большинстве случаев нецелесообразно.

border-top-color, border-right-color, border-bottom-color, border-left-color

Задают цвет верхней, правой, нижней и левой рамок соответственно.

Пример:

```
DIV {
  border: 2px dashed #000;
  border-top-color: #666}
```

border-top-style, border-right-style, border-bottom-style, border-left-style

Задают стиль верхней, правой, нижней и левой рамок соответственно.

Пример:

```
DIV {
  border: 2px solid #000;
  border-left-style: dotted)
```

border-top-width, border-right-width, border-bottom width, border-left-width

Задают ширину верхней, правой, нижней и левой рамок соответственно.

Пример:

```
DIV 1
  border: 2px solid #000;
  border-top-width: 1%}
```

border-width

Задает ширину четырех сторон рамки. Можно указывать как одно значение, которое будет применено ко всем четырем сторонам, так и разные значения через пробел, которые будут применяться для каждой стороны в отдельности.

```
DIV {
  border: solid #000;
  border-width: 2px lpx 3px lpx}
```

bottom

Задает смешение позиционированного блока относительно **нижнего края** контейнера.

Пример:

```
#footer {
  position: absolute;
  bottom: 10%;
  width: 100%}
```

caption-side

Задает, с какой стороны таблицы будет выводиться ее заголовок.

Значения:

О top — заголовок выводится сверху. Значение по умолчанию;

• right — заголовок выводится справа;

СЗ bottom — заголовок выводится снизу;

• left — заголовок выводится слева.

Пример:

```
CAPTION {
  caption-side: left
```

clear

Запрещает обтекание для блока с определенной стороны. Применяется для плавающей модели.

Значения:

О left — блок перемешается ниже всех плавающих блоков с объявлением **float: left;**

• right — блок перемешается ниже всех плавающих блоков с объявлением **float: right;**

• both — блок перемещается ниже всех плавающих блоков;

О попе — ограничений на обтекание нет. Значение по умолчанию.

Пример:

```
•main (
  cl%ar: left;
  width: 50%)
```

clip

Задает область просмотра для элемента, у которого установлено свойство overflow, В качестве значения указывается прямоугольник с отступами от четырех сторон элемента. Формат такой — rect (сверху, справа, снизу, слева'•.

Пример:

```
#first {
  clip: rect(0%, 5%, 30%, 5%)}
```

color

Задает цвет текста в элементе.

Пример:

```
BODY {
  Color: #000-
```

content

Служит для формирования генерируемого содержимого. Может применяться только вместе с псевдоэлементами :before и :auto.

Значения:

- строка — можно задавать строку произвольного текста, т. е. делать замечания, заголовки, ремарки и т. п.;
- url — посредством задания URL можно вставлять изображения, звуковые файлы и прочие объекты;
- счетчики — с помощью счетчиков можно создавать нумерованные разделы и подразделы, как это принято в различного рода руководствах и официальных документах;
- open-quote и close-quote — эти значения заменяются соответствующей строкой из свойства quote;
- no-open-quote и no-close-quote — нужны для описания вложенных кавычек;
- attr (имя атрибута) — эта функция возвращает строку, которая является значением указанного атрибута.

Примеры:

```
P.textnote:before {
    content: "Замечание: ")

P.imgnoterbefore {
    content: url("note.gif")f

HI:before {
    content: "Глава " counter(chapter) ". ";
    counter-increment: chapter)
```

counter-increment

Включает счетчик. Задаст, какой счетчик и на какое значение должен увеличиться при очередном обнаружении данного элемента в документе. В качестве значения передается имя счетчика и целое число через пробел. i. с. формат следующий:

```
counter-increment: имя_счетчика целое_число
```

По умолчанию счетчик увеличивается на единицу.

Пример:

```
HI:before {
  content: "глава " counter(chapter) ". ";
  counter-increment: chapter 1}
```

counter-reset

Сбрасывает значение указанного счетчика в начальное состояние. Необходимо для нумерации подразделов. В качестве значения указывается имя счетчика.

Пример:

```
Hl:before (
  content: "Глава " counter(chapter) ". ";
  counter-increment: chapter 1;
  counter-reset: subchapter}
```

cursor

Задает тип курсора, который будет применяться при наведении мышки на элемент.

Значения:

П crosshair — крестик:

- default — тип курсора по умолчанию в данной операционной системе. Зависит от конкретной операционной системы;
- pointer — указатель на ссылку. В системе Windows это рука с указательным пальцем;
- move — обозначает, что объект может быть перемещен с помошыо мыши;
- text — обозначает, что текст может быть выделен. В системе Windows это курсор в виде буквы I;

D wait — обозначает, что система занята. Обычно обозначается в виде песочных часов;

O help — обозначает, что в ланжш месте доступна помошь (подсказка). Обычно обозначается стрелочкой со знаком вопроса;

П e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize — обозначает, что объект может изменять свой размер (то есть край объекта может быть перемешен);

G uri — можно задать свой собственный курсор, указав его местонахождение па сервере.

Примеры:

```
O BUTTON (
    cursor: pointer}

• #tip <
    cursor: url(/i/tip.cur)}
```

direction

Задает направление вывода блоков и текста на странице. Так. например, в арабском языке TCKCI пишется справа налево.

Значения:

П itr — слева направо;

• rti — справа налево.

Пример;

```
BODY {
  direction: rtl;
  unicode-bidi: embed)
```

display

Задает тип блока.

Значения:

D block — структурный блок;

G inline — строчный блок;

П list-item — элемент списка;

П marker — тип генерируемого содержимого, которое создает маркер. Можно использовать только с псевдоэлементами :before и :after;

O none — блок отсутствует в дереве документа;

- run-in и compact — создается или структурный блок, или строчный блок в зависимости от контента;

П table, inline-table, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell и tabie-capticr. — служат для создания таблиц средствами CSS, Нужны только для форматирования XML-документов.

Примеры:

```
O P.list {
     display, list-item-
     list-style-type: circle}

П EM.important:before {
    display; marker;
    content: """j
```

empty-cells

Задает отображение рамок вокруг ячеек таблицы, которые не содержат видимого содержимого, если border-collapse: separate.

Значения:

O show — рамки вокруг пустых ячеек отображаются;

- hide — рамки вокруг пустых ячеек не отображаются.

Пример:

```
TABLE {
  border: Ipx solid #000;
  border-collapse: separate;
  empty-cells: show•
TD {
  border: Ipx solid #000}
```

float

Включает плавающую модель для блока.

Значения:

- left — элемент "прилипает" к левому краю контейнера;
- right — элемент "прилипает" к правому краю контейнера;

O попе — плавающая модель не применяется.

Пример:

```
#main {
  float: left;
  width: 50%}
#right {
  float: left;
  width: 30%}
```

font

Сокращенная форма записи свойств шрифта. Включает в себя свойства fonc-style, iont-variant, font-weight, font-size, line-height, font-family.

Пример:

```
BODY {
  color: #000;
  font: lem/1.4 Verdana, sans-serif}
```

font-family

Задает гарнитуру шрифта. Можно указывать значения через запятую.

Пример:

```
PRE {
  font-family: "Courier New", monospacej
```

font-size

Задает размер шрифта.

Значения:

О размер .можно задать с помощью единиц измерений и ключевых слов (xx-small, x-small, small, medium, large, x-large, xx-large);

- можно задать относительный размер'
  - larger — больше, чем размер шрифта у родительского элемента;
  - smaller — меньше, чем размер шрифта у родительского элемента.

Пример:

```
PRE 1
  font-family: "Courier New", monospace;
  font-size: smaller}
```

font-size-adjust

Задает корректирующую величину, значение которой основывается на отношении величины x-height шрифта к величине шрифта (чем больше это значение, тем лучше читается шрифт в мелких кеглях). То есть если это отношение XlЯ шрифта, который заменяет основной (отсутствующий) шрифт меньше, то сам шрифт будет иметь больший кегль, чем основной.

Пример:

```
#news {
  font-family: Verdana, sans-serif;
  fent-size: 12px;
  font-size-adjust: 0.6j
```

font-stretch

Задает тип штифта, такой как condensed или expanded.

*Значения**

П ultra-condensed;

O extra-condensed;

O condensed;

O semi-condensed;

O normal;

• semi-expanded;

П expanded;

O extra-expanded;

O ultra-expanded.

Пример:

```
HI{
  font-stretch; condensed)
```

font-style

Задает стиль шрифта.

Значения:

П normal — обычный шрифт;

• oblique — наклонный шрифт;

O italic — Курсив.

Пример:

```
EM {
  color: #C00;
  font-style: normal}
```

font-variant

Задает вид шрифта.

Значения:

- normal — обычный шрифт;
- small-caps — шрифт выводится малыми прописными.

Пример:

```
H1, H2, H3 {
  font-variant: small-caps}
```

font-weight

Задает насыщенность шрифта.

Значения:

- численные (от 100 до 900 с промежутком в 100, т. е. 200, 300 и т. д.);
- normal (соответствует 400) — обычный шрифт;
- bold (соответствует 700) — полужирный шрифт;
- bolder — насыщенность выше, чем у шрифта родительского элемента;
- lighter — насыщенность ниже, чем у шрифта родительского элемента.

Пример:

```
B {
  font-weight: 900}
```

height

Задает высоту блока.

Пример:

```
#main {
  width: 50%;
  height: 90%}
```

left

Залает смешение блока относительно левого крал родительского элемента.

Пример:

```
#top {
  position: absolute;
  width: 3OOpx;
  left: KOOpx)
```

letter-spacing

Задает расстояние между буквами.

Пример:

```
EM (
  letter-spacing: 2em}
```

line-height

Задает высоту строки. В качестве значений можно указывать безразмерные числа, проценты и другие единицы измерения.

Пример:

```
P K
  font-family: Verdana, sans-serif;
  line-height: 1.5}
```

list-style

Сокращенная форма записи свойств списка. Содержит свойства list-style-type, list-style-position, list-style-image.

Пример:

```
UL {
  list-style: circle inside[1]
```

list-style-image

Задает изображение, которое будет служить маркером для списка.

Пример:

```
UL {
  list-style-iroage:url{/i/marker,gif)}
```

list-style-position

Задает позицию маркера по отношению к содержимому элементов списка.

Значения:

П outside — маркер будет выводиться за пределами блока, создаваемого элементом списка, т е пол маркером не будет никакого контента, если строк в элементе списка больше, чем одна. Значение по умолчанию,

- inside — маркер будет выводиться в пределах блока, создаваемого элементом списка, г е под маркером будет расползаться вторая и последующие строки элемента списка

Пример

Ш

```
list-styla-position  insidei
```

list-style-type

Задает тип маркера

Значения

О для нумерованного списка

- decimal — номера из десятичных цифр (1, 2 и т д);
- decimal-leading-zero — номера из десятичных цифр, но с нулем в начаае первого десятка (01, 02, 03,. . , 98, 99);
- lower-roman — строчные римские цифры (i. п. iii, iv, v и т. д.);
- upper-roman — прописные римские цифры (I. II. III. IV, V и т. д);
- hebrew — еврейская нумерация,
- georgian — грузинская нумерация (an, ban. gan. , he, tan. in, in-an,..);
- armeman — армянская нумерация,
- hiragana — a, l, u, e. o, ka, kl, . ;
- касакача - A, I. U, E, O, KA, KI, .;
- hiragana-iroha — i, ГО, ha. m. ho, he, to,.. ,
- katakana-iroha- I. RO, HA. N1, HO, HE, TO, ,

- для ненумерованного списка
  - circle — незакрашенный кружок;
  - disc — закрашенный кружок;
  - square — квадратик

Примеры:

```
a OL {
    list-style-type: upper-roman
a иь (
    list-style-type: square
```

margin

Сокращенная форма записи для установки полей. Включает в себя свойства margin-top, margin-right, margin-bottom И margin-left.

Пример:

```
ttmenu {
  margxn: 5% 3%}
```

margin-top, margin-right, margin-bottom и margin-left

Задают значения величины верхнего, правого, нижнего и левого поля соответственно.

Пример:

```
#top {
  margin-left: 10%;
  margin-right: 10%}
```

marks

Служит для задания разного рода меток на странице при печати. Метка печатается вне страничного блока.

Значения:

- crop — обозначает место отреза;
- cross — служит для выравнивания страниц.

max-height

Задает максимально допустимую высоту блока. Если высота контента превышает эту высоту, то контент должен обрезаться.

Пример:

```
#top {
  height: 100px;
  max-height: 120px}
```

max-width

Задает максимально допустимую ширину блока. Если ширина контента превышает эту ширину, то контент должен обрезаться.

Пример:

```
#top {
  width: 25%;
  max-width: 30%}
```

min-height

Задает минимальную высоту блока. Она не будет меньше ни при каких обстоятельствах.

Пример:

```
#leftmenu {
  height: auto;
  min-height: 90%}
```

min-widht

Задает минимальную ширину блока.

Пример:

```
#topmenu (
  width: auto;
  min-width: 40%}
```

orphans

Задает минимальное число строк, которые должны быть оставлены внизу страницы.

Пример:

```
@page ordinary (
  size: 21cm 28cm;
  orphans: 4}
```

outline

Сокращенная форма записи для установки контуров. Вообще контуры от рамок отличаются тем, что не входят в блоковую модель и могут быть непрямоугольными. Содержит свойства outline-color, outline-style и outline-widht.

Пример:

```
IMG <
  outline: #000 solid lpx'-
```

outline-color

Задает цвет контура.

Пример:

```
#note (
  outline-color: #000;
  outline-width: 2px}
```

outline-style

Задает стиль контура. Значения совершенно аналогичны тем, которые имеются у свойства border-style.

Пример;

```
#note {
  outline-color: #000;
  outline-width: 2px;
  outline-style: dotted}
```

outline-widht

Задает ширину контура.

Пример:

```
#note {
  outline-color: #000;
  outline-width: 2px;
  outline-style: dotted)
```

overflow

Описывает повеление содержимого блока в случаях, когда размеры содержимого превышают размеры блока.

Значения:

- visible — часть содержимого блока, выходящая за пределы блока, остается видимой. Значение по умолчанию;
- hidden — часть содержимого блока, выходящая за пределы блока, не видна;
- scroll — блок в обязательном порядке снабжается полосами прокрутки, чтобы пользователь мог просмотреть все содержимое.
- auto — блок снабжается полосами прокрутки только в том случае, когда в них есть необходимость. Если размеры содержимого не превышают размеры б.юка, то полос прокрутки не будет.

Пример:

```
#newsblock 1
  width: 15%;
  height: 10%;
  border: lpx solid #000;
  overflow: auto}
```

padding

Сокращенная форма записи для установки отступов. Включает в себя свойства padamg-top, padding-right, padding-bottom И padding-left.

Пример:

```
#menu {
  padding: 10% 5% 5% 10%}
```

padding-top, padding-right, padding-bottom и padding-left

Залают значения величины верхнего, правого, нижнего и левого отступа соответственно.

Пример:

```
#top {
  padding-bottom: 10%;
  padding-top: 10%}
```

page

Служит для обращения к заданному страничному блоку. В качестве значения >казывается имя блока.

Пример:

```
@page hor (
  size: landscape}
IMG.bigimage {
  page: hor}
```

page-break-after, page-break-before и page-break-inside

Служат для установки разрыва страницы после, перед и внутри элемента соответственно.

Значения:

- auto — разрывы при печати ставит сам браузер;
- always — разрыв ставится в любом случае;
- avoid — по возможности браузер должен не ставить разрыв страницы;
- left — вставляет один или два разрыва, чтобы следующая страница была левой;
- right: — вставляет один пли два разрыва, чтобы следующая страница была правой.

Примеры:

```
TABLE I
  page-break-inside: avoid}
IMG.biq {
  page-break-before: always}
```

position

Задает схему позиционирования.

Значения:

- static — блок будет позиционироваться в соответствии с нормальным потоком;
- relative — включает относительное позиционирование;
- absolute — включает абсолютное позиционирование;
- fixed — включает абсолютное фиксированное позиционирование.

Примеры:

```
G #menu (
    position: fixed;
    top; Юрх;
    left: 10px}
• #top (
    position: absolute;
    right: 30%;
    top: 5px}
```

quotes

Задает тип кавычек.

Примеры:

```
D Q:lang(ru) {
    quotes: *«' '»' "" 11" }
O Q:lar.g(eng) {
    quotes:. . . . . '. . . . . .)
```

right

Задает отступ от правого края контейнера для позиционированного блока.

Пример:

```
#rbiock {
  position: absolute;
  right: 13%;
  width: 67%}
```

size

Задает размеры и ориентацию страничного блока.

Значения:

- если указывать численные значения, то первое значение задает ширину блока, а второе — высоту блока;
- если использовать ключевые слова, то возможны три значения:
  - auto — сохраняется размер и ориентация страницы, т. е. устройство вывода само подбирает нужный размер.

- landscape — сохраняется размер, но горизонталью становится наибольшая сторона страницы, т. е. страница как бы поворачивается на 90 градусов;
- portrait — сохраняется размер, но горизонталью становится наименьшая сторона страницы.

Примеры:

```
O браде {
    size: landscape •
G Эраде (
    size: 20cm 28cm}
```

table-layout

Задает алгоритм обработки таблиц.

Значения:

П fixed — алгоритм фиксированной таблицы, значительно ускоряет обработку таблицы браузером;

O auto — автоматический алгоритм. Значение по умолчанию.

Пример:

```
TABLE.fix <
  table-layout: fixed}
```

text-align

Задает горизонтальное выравнивание содержимого блока.

Значения'

- left — содержимое выравнивается по левом} краю. Значение по умолчанию;
- right — содержимое выравнивается по правому краю;

O center — содержимое выравнивается по центру;

П justify — содержимое выравнивается и по левому, и по правому краю. Фактически влияет только на текст.

Пример:

```
p<
  text-align: [justify}
CITE.first {
  text-align: right}
```

top

Задает отступ от верхнего края контейнера.

Пример;

```
#news {
  position: absolute;
  top: l0px
```

vertical-align

Задает вертикальное выравнивание вн>три строчного блока

Значения:

- baseline — совмещает базовую линию строчного блока с базовой линией родительского строчного блока;
- middle — совмещает базовую линию блока с уровнем "средний уровень плюс половина высоты x-height родительского элемента";
- sub — делает элемент подстрочным;
- super — делает элемент надстрочным;
- text-bottom — совмещает нижний край строчного блока с нижним краем шрифта родительского блока;
- bottom — совмещает нижний край строчного блока с нижним краем самого низкого элемента в линии;
- top — совмещает верхний край строчного блока с верхним краем самого высокого элемента в линии.

Пример:

```
GrAN.super {
  vertical-align: sup;
```

visibility

Устанавливает, является элемент видимым или невидимым.

Значения:

- visible — элемент видимый;
- hidden — элемент невидимый;
- collapse — служит для сокрытия колонок и рядов в таблице.

Пример:

```
#menu-level2 {
  position: absolute;
```

```
  top: 140px;
  left: 23px;
  visibility: hidden}
```

white-space

Задает модель обработки браузером пробелов и переводов строк в тексте.

Значения:

- normal — все пробелы преобразуются в один. То есть если между словами в тексте встав/юны три пробела, в браузере пробел будет один. Кроме того, браузер расставляет переволы строк так, чтобы точно вместить контент в размеры блока;
- pre — браузер полностью сохранит имеющееся форматирование;
- nowrap — пробелы будут преобразовываться в один, однако перевод строки будет только там, где это указано явно.

Пример:

```
H1{
  color: #C00;
  white-space: nowrap}
```

widows

Задает минимальное число строк, которые должны быть оставлены вверху страницы.

Пример:

```
@page ordinary (
  size: 21cm 28cm;
  widows: 2;
  orphans: 4}
```

width

Задает ширину блока.

Пример:

```
#main {
  float: left;
  width: 60%;
  neight: auto}
```

word-spacing

Задает изменение ширины пробела между отдельными словами в тексте.

Пример:

```
P {
  word-spacing: -lpx}
```

z-index

Задает индекс слоя по оси *z*. В качестве значений указываются целые числа.

Пример:

```
#dropdown2 {
  position: absolute;
  top: 100px;
  left: 20px;
  z-index: 2}
```

Приложение 4

Поддержка браузерами элементов CSS

В данном приложении представлены сведения о поддержке наиболее важных свойств CSS основными браузерами.

CSS-1

Таблица П4.1. *Поддержка способов включений каскадных таблиц стилей в документ*

Способ включения	IE5	IE 6	N6	Opera 5	Комментарий
STYLE	Да	Да	Да	**Да**	
STYLE MEDIA	**Частично**	Частично	Частично	**Да**	Поддерживаются далеко не все типы устройств. Не поддерживаются handheld, tv, projection **и** др.
LINK	Да	Да	Да	**Да**	
LINK MEDIA	Частично	Частично	Частично	**Да**	Поддерживаются далеко не все типы устройств. Не поддерживаются handheld, tv, projection **и** др.
@import	Мелкий баг	Да	Да	**Да**	IE 5 позволяет включать файлы даже тогда, когда инструкция @import находится 8 конце таблицы стилей. Это неверно с точки зрения спецификации CSS-1

Таблица П4.2. *Поддержка селекторов, псевдоклассов и псевдоэлементов*

Селектор	IE 5	IE 6	N6	Opera 5	Комментарий
Элемента E {}	Да	Да	Да	Да	

Таблица П4.2 *(окончание)*

Селектор	IE 5	IE 6	N6	Opera 5	Комментарий
Класс ,classname t}	Мелкий баг	Мелкий баг	Да	Да	В IE 5—6 допускается, чтобы имя класса начиналось с цифры, однако это противоречит спецификации CSS-1
ID Hdname { }	**Мелкий баг**	**Мелкий баг**	**Да**	Да	В IE 5—6 допускается, **чтобы** имя :э начиналось с цифры, однако это противоречит спецификации CSS-1
Контекстный E EE П	Да	Да	Да	Да	
:link	Да	Да	Да	Да	
:active	Частично	Частично	Частично	Да	Все браузеры, кроме Netscape 6, поддерживают данный псевдокласс только для элемента <A>
::visited	Да	Да	Да	Да	
:• first-letter	Нет	Да	Да	Да	В браузере IE 5 5 :firsc-lei.i~r поддерживается
:first-line	Нет	Да	Баг	Да	В браузере IE 5.5 :first-line поддерживается. В некоторых версиях Netscape использование данного псевдоэлемента может приводить к выполнению браузером недопустимой операции

Таблица П4.3. *Поддержка базовых концепций* ***каскадных таблиц стилей***

Концепция	IE 5	IE 6	N6	Opera 5	Комментарий
Группировка E, EE, ЬEE<}	Да	Да	Да	Да	
Каскадирование	Да	Да	Да	Да	
Каскадирование !important	Да	Да	Да	Да	
Наследование	Средний баг	Да	Да	Да	В IE 5 для установки размера шрифта элемент <TABLE> нарушает наследование

Таблица П4.3 (окончание)

Концепция	IE5	IE6	N6	Opera 5	Комментарий
Комментарии	Мелкий баг	Да	Да	Да	В IE 5, если комментарий ставится внутри правила, некоторые объявления могут игнорироваться

Таблица П4.4. Поддержка единиц измерения длины

Единица измерения	IE5	IE6	N6	Opera 5	Комментарий
px	Да	Да	Да	Да	
em	Да	Да	Да	Да	
ex	Мелкий баг	Мелкий баг	Да	Да	В браузерах IE 5—6 1 ex равен 1/2 em, что неверно, хотя почти всегда является хорошим приближением
in	Да	Да	Да	Да	
pt	Да	Да	Да	Да	
pc	Да	Да	Да	Да	
cm	Да	Да	Да	Да	
mm	Да	Да	Да	Да	
%	Да	Да	Да	Да	

Таблица П4.5. Поддержка различных форматов цвета

Форма задания цвета	IE 5	IE 6	N6	Opera 5	Комментарий
Сокращенная RGB ttFCO	Да	Да	Да	Да	
Полная RGB #FF0000	Да	Да	Да	Да	
Функциональная rgb(255,0,0)	Да	Да	Да	Да	
Функциональная процентная rgb(255j»,P*¡/6%)	Да	Да	Да	Да	
Ключевые слова	Да	Да	Да	Да	

Таблица П4.6. *Поддержка свойств фона*

Свойство	IE5	IE 6	N 6	Op 5	Комментарий
back.grouna-co-ì.or	Да	Да	Да	Да	
background-image	Да	Да	Да	Да	
background-repeat: no-repeat	Да	Да	Да	Да	
background-repeat: repeat-x	Баг	Да	Да	Да	В IE 5 фон размножается только вправо. Поэтому если применяется свойство background-position, ОН **не размножится влево** и не закроет всю ось **x**
background-repeat: repeat-y	Баг	Да	Да	Да	В IE 5 фон размножается только вправо, но не размножается вверх
background-attachment	Да	Да	Да	Да	
background-position	Да	Да	Да	Да	
background	Да	Да	Да	Да	

Таблица П4.7. *Поддержка свойств шрифта*

Свойство	IE 5	IE 6	N 6	Op 5	Комментарий
font-family	Да	Да	Да	Да	
font-style	Частично	Частично	Частично	Частично	У **всех** браузеров возникают проблемы с italic. Они все отображают italic **как** oblique, что **неверно**
font-variant	Баг	Да	Да	Да	Если размер шрифта задан в пикселах, то IE 5 выводит и прописные буквы, и small-caps одинакового размера, что неверно
font-weight	Частично	Частично	Частично	Частично	Если задавать насыщенность в цифрах, то в IE 5-6 работают три значения (100, 600, 900), а *в* N6 только два (100, 600)
font-size	Баг	Да	Да	Да	В IE 5 ключевые слова реализованы со сдвигом. То есть нормальный размер шрифта соответствует ключевому слову srr.all, а не medium

Таблица /74.7 (окончание)

Свойство	IE 5	IE6	N6	Op5	Комментарий
font	Баг	Да	Да	Да	В IE 5 можно задать два произвольных свойства {например, font: bold lem), тогда как в CSS-1 нужно обязательное присутствие семейства шрифта, т е font: bold iem Verdana

Таблица П4.8. Поддержка свойств текста

Свойство	IE5	IE 6	N6	Op 5	Комментарий
text-alig-	Да	Да	Да	Да	
text-decoration	Частично	Частично	Да	Да	IE не поддерживает значение blink, которое обеспечивает мигание текста
text-indent	Да	Да	Да	Да	
text-transform	Да	Да	Да	Да	
word-spacing	Нет	Да	Да	Да	
letter-spacing	Да	Да	Да	Да	
line-height	Да	Да	Да	Да	
vertical-align	Частично	Частично	Да	Да	IE 5 поддерживает только значения super, sub и baseline IE 6 не поддерживает значение в процентах

Таблица П4.9. Поддержка блоковых свойств

Свойство	IE5	ЕЕ6	N6	0p5	Комментарий
widtn	Да	Да	**Да**	**Да**	
height	Да	Баг	**Да**	**Да**	6 IE 6 в некоторых случаях высоту блока задать невозможно, что чрезвычайно странно
display	Частично	**Да**	**Да**	**Да**	IE 5 не поддерживает значение list.-it.em

Таблица П4.9 (продолжение)

Свойство	IE 5	IE 6	N6	Op 5	Комментарий
wnite-space	Нет	Да	Да	Да	IE 5 вообще не поддерживает это свойство, но IE 5 5 поддерживает значения normal И nowrap
float	Баг	Да	Да	Да	Вообще это свойство достаточно сложно для реализации, однако некоторые проблемы есть только в браузере IE 5
clear	Да	Да	Да	Да	
margin-top margin-right margin-bottom margin-left	Частично	Баг	Да	Да	IE 5 не позволяет задавать поля для строчных блоков, а IE 6 имеет странный баг, из-за которого в некоторых случаях ширина полей, заданных в относительных единицах, вычисляется относительно окна браузера, а не относительно контейнера
margin	Частично	Баг	Да	Да	
padding-top padding-right padding-bottom padding-left	Частично	Да	Да	Да	В IE 5 нельзя устанавливать отступы для строчных блоков
padding	Частично	Да	Да	Да	
border-color	Да	Да	Да	Да	
border-style	Частично	Да	Да	Да	IE 5 не поддерживает значения *c.-.tze-J* и casr.ea, но их поддерживает IE 5.5
border-top-width border-right-width border-bottora-width border-left-width	Частично	Да	Да	Да	

Таблица П4.9 *(окончание)*

Свойство	IE 5	IE 6	N6	Op 5	Комментарий
border-width	**Частично**	Да	Да	Да	
border-top border-right border-bottom border-left	Частично	Да	Да	Да	
border	Частично	Да	Да	Да	

Таблица П4.10. *Поддержка свойств списков*

Свойство	IE 5	IE6	N6	Op5	Комментарий
list-style--,ype	Частично	Да	Да	Да	IE 5 не поддерживает значение попе
list-style-Image	Баг	Да	Да	Да	IE 5 имеет небольшой баг. Если установить свойство line-height, то маркеры могут не отобразиться, если их ширина больше 30 пикселов
list-style-position	Да	Да	Да	Да	
list-style	Да	Да	Да	Да	

CSS-2

Таблица П4.11. *Поддержка селекторов, псевдоклассов и псевдоэлементов*

Селектор	IE 5	IE 6	N6	Op 5	Комментарий
Универсальный * { }	Да	Да	Да	Да	
Наследника E > EE {}	Нет	Нет	Да	Да	
Предка E + EE {}	Нет	Нет	Да	Да	

Таблица П4.11 *(окончание)*

Селектор	IE 5	IE6	N6	Op5	Комментарий
Атрибута E[attr3	Нет	Нет	Да	Да	
По значению атрибута E[attr=value] E[attr~=value]	Нет	Нет	Да	Да	
:hover	Частично	Частично	Частично	Да	Все браузеры, кроме Netscape 6, поддерживают данный псевдокласс только для элемента <A>
:focus	**Нет**	Нет	Да	Нет	
:before :after	Нет	Нет	Да	Частично	Opera 5 не позволяет внедрять изображения, тогда как Netscape позволяет
:first-child	Нет	Нет	Да	Нет	
: lang	**Нет**	Нет	**Нет**	**Нет**	

Таблица *П4.12. Поддержка визуальных эффектов*

Свойство	IE5	IE6	N6	Op5	Комментарий
overflow: visible	**Баг**	Баг	**Да**	Да	
overflow: hidden	**Да**	**Да**	**Да**	Да	
overflow: scroll	**Да**	**Да**	**Да**	**Нет**	
overflow: auto	**Да**	**Да**	**Да**	**Нет**	
clip	**Нет**	**Да**	**Да**	**Нет**	
visibility	**Да**	**Да**	**Да**	Частично	Opera 5 не поддерживает значение collapse

Таблица *П4. 13. Поддержка генерируемого содержимого*

Свойство	IE 5	IE6	N6	Op5	Комментарий
content: string	Нет	Нет	Да	**Да**	

Таблица П4.13 (окончание)

Свойство	IE 5	IE 6	N6	Op 5	Комментарий
content: url	Нет	Нет	Да	Нет	
content: counter	Нет	Нет	Нет	Да	
content: open-quote	Нет	Нет	Да	Да	
content: close-quote					
quotes	Нет	Нет	**Да**	Да	

Таблица П4.14. Поддержка страничных блоков

Свойство	IE5	IE 6	N6	Op 5	Комментарий
(lpage	Нет	Нет	Нет	Час-тично	Opera 5 не поддерживает именованные страницы
size	Нет	Нет	Нет	**Да**	
margin	**Нет**	Нет	Нет	**Да**	
:first	Нет	Нет	Нет	**Да**	
:left					
:right					
page-break-before page-break-after	Нет	Да	Нет	Да	Данные свойства поддерживаются в браузере IE 5 5
page-break-inside	Нет	Нет	Нет	Да	

Таблица П4.15. Поддержка свойств шрифта и текста

Свойство	IE 5	IE 6	N6	0p 5	Комментарий
font-stretch	Нет	Нет	**Нет**	Нет	
font-size-adjust	Нет	Нет	Нет	Нет	
Системные шрифты	Да	Да	Да	Нет	

ТаблицаП4.15*(окончание)*

Свойство	IE 5	IE 6	N6	Op 5	Комментарий
tfionn-race	Нет	Нет	Нет	**Нет**	
cext-shadow	Нет	**Нет**	**Нет**	**Нет**	
direction	Да	Да	**Нет**	Нет	

Таблица П4.16. *Поддержка свойств контроля за пользовательским интерфейсом*

Свойство	IE5	IE 6	N6	Op5	Комментарий
cursor	Частично	Да	Частично	Нет	IE 5 и Netscape не позволяют подключать собственные курсоры с **помощью объявлений** cutsor: url()
outline	Нет	**Нет**	**Нет**	**Нет**	
outline-width	**Нет**	**Нет**	**Нет**	**Нет**	
outline-style	**Нет**	Нет	Нет	Нет	
outline-color	Нет	Нет	Нет	**Нет**	

Таблица П4.17*. Поддержка блоковых свойств*

Свойство	IE 5	IE6	N6	Op 5	Комментарий
min-width	**Нет**	**Нет**	Да	Да	
max-width	**Нет**	**Нет**	Да	Да	
min-height	Нет	Частично	Да	Да	Браузер IE 6 поддерживает данное свойство только для ячеек таблицы
max-height	Нет	Частично	Да	Да	Браузер IE 6 поддерживает данное свойство только для ячеек таблицы
display: inline-block	**Нет**	**Нет**	Да	**Нет**	
display: run-in	**Нет**	**Нет**	**Нет**	Да	
display: compact	Нет	Нет	Нет	Баг	Opera 5 делает небольшой отрицательный отступ
display: table	Нет	Нет	Да	Да	

Таблица П4.17 (окончание)

Свойство	IE 5	IE 6	N6	Op5	Комментарий
display: ma: -.e-	Нет	Нет	Да	Нет	
display: in-line-table	Нет	**Нет**	Баг	Да	**Браузер Netscape воспринимает данное объявление как** display: table, **что неверно**

Таблица П4.18. Поддержка позиционирования

Свойство	IE 5	IE 6	N6	Op5	Комментарий
position	**Частично**	Частично	Да	Да	Браузеры **!E** не поддерживают объявление position: fixed
top	**Да**	**Да**	Да	Да	
right	**Нет**	Да	Да	Да	
bottom	Нет	Да	Да	Да	
left	**Да**	Да	Да	Да	
z-indey	Да	Да	Да	Да	

Таблица П4.19. *Поддержка блоковых свойств*

Свойство	IE 5	IE6	N6	Op5	Комментарий
caption-side	Нет	Нет	Нет	Нет	
table-layout	**Да**	**Да**	**Да**	Баг	По неизвестной причине Opera 5 иногда делает **таблицы \же, чем надо, если указано объявление** table-layout: fixed
border-spacing	**Нет**	Нет	**Да**	**Да**	
border-collapse	Да	**Да**	Нет	**Да**	
empty-cells	Нет	Нет	**Да**	**Да**	

Глоссарий

ASP (Active Server Pages)

Среда для создания серверных скриптов. ASP-страницы обрабатываются на стороне сервера, и браузер получает только HTML-код. Функционирует только на серверах фирмы Microsoft.

ColdFusion

Среда для создания веб-приложений. Включает в себя ColdFusion Markup Language (CFML) и язык сценариев CFScript. От **PHP** и ASP отличается простотой.

Cookies

файл с информацией, хранящийся на машине пользователя. Информация может использоваться браузером для различных целей (например, для авторизации).

DHTML (Dynamic HyperText Markup Language)

Технология, основанная на слиянии HTML, CSS и JavaScript. Позволяет динамически изменять вид страниц в ответ на те или иные действия события. С помощью DHTML можно делать выпадающие меню, анимацию на страницах, динамические подсказки и т. д.

DTD (Document Type Declaration)

Объявление типа документа. Фактически **DTD** представляет собой синтаксис для описания языков разметки. Файл DTD содержит спецификацию языка разметки гипертекста, т. е. все возможные элементы, атрибуты и их значения. Достаточно сложен для изучения. Необходим при использовании XML

DOM (Document Object Model)

Объектная модель документа — это платформонезависимая среда, позволяющая создавать скрипты для динамического доступа к элементам документа, что позволяет динамически менять структуру, контент и представление элемента на странице.

HTML (HyperText Markup Language)

Язык разметки гипертекста, базирующийся на SGML. Все веб-страницы в конечном итоге преобразуются в HTML-страницы.

HTTP (HyperText Transfer Protocol)

Протокол передачи гипертекста. На основе этого протокола строятся взаимодействия клиент-сервер.

JavaScript

Язык программирования скриптов, которые выполняются на стороне клиента. Основан на объектной модели документа. В основном применяется для создания динамических эффектов на странице.

MySQL

Реляционная база данных с открытым кодом. Отличается быстротой функционирования. Идеально подходит для создания несложных систем.

P3P (Platform for Privacy Preferences)

Стандарт, позволяющий пользователю контролировать информацию, которую собирает сайт при посещении пользователем. Например, можно запретить определять версию браузера и т. д.

Perl

Язык программирования, который специально разрабатывался для работы со строками текста. Отличается мощностью и универсальностью, но достаточно сложен.

PHP (Personal Home Page)

Язык программирования серверных скриптов, который разрабатывался специально для веб. Отличается простотой и ясностью. Код, как и в ASP, можно встраивать в HTML-страницы.

Python

Язык программирования серверных скриптов. Отличается сбалансированностью, хорошей объектно-ориентированной семантикой и модуляризацией.

SGML (Standard Generalized Markup Language)

Стандартный обобщенный язык разметки, на базе которого и создан язык HTML. Фактически HTML является крайне упрошенной версией SGML.

URI (Universal Resource Identifier)

Универсальный идентификатор ресурса Строка, состоящая из текста и цифр, которая однозначно нденшфицпр\е1 ресурс в сети Интернет.

Weblog

Персональный некоммерческий сайт, который имеет вид дневника и обновчяется достаточно часто (в идеале, каждый день).

XHTML (Extensible Hypertext Markup Language)

Модифицированный язык HTML. Синтаксис приведен в соответствие с XML-синтаксисом, добавлена модуляризация. но принципиальных преимуществ по сравнению с HTML нет.

XML (Extensible Markup Language)

Расширенный язык разметки гипертекста Обеспечивает совершенно иной уровень структурированности документов. На основе XML можно создавать языки разметки, наподобие HTML

XSL (Extensible Stylesheet Language)

Язык создания таблиц стилей для форматирования XML-документов. Гораздо более мощный, чем CSS. но и более сложный. Позволяет трансформировать документы в разные форматы и манипулировать данными на стороне клиента.

Абсолютное позиционирование

Вид CSS-позиционирования, при котором положение блока задается жестко.

Атрибут

Атрибут языка HTML Является дополнительным носителем информации о разметке документа.

Баг

Ошибка, возникающая в результате некорректной реализации чего-либо в программном продукте. Например, в браузерах есть баги, вызванные некорректной реализацией CSS.

Блочный элемент

Элемент документа, который визуально форматируется в виде структурной единицы, т. е. представляет собой абзац, заголовок и т. п.

Браузер

Приложение, которое позволяет просматривать информацию во Всемирной сети и взаимодействовать с веб-сайтами.

Веб-мастер **(HTML-верстальщик)**

Человек, который занимается кодированием макетов в HTML с применением каскадных таблиц стилей и других технологий верстки.

Гарнитура

Совокупность шрифтов, объединенных общими стилевыми признаками, отличными от других шрифтов.

Гиперссылка

Объект, который образует одностороннюю связь одного документа с другим.

Гипертекст

Ассоциативное пространство, отдельные части которого связаны между собой гиперссылками.

Группировка

Объединение селекторов с одинаковыми объявлениями с целью уменьшения объема кода.

Дерево документа

Структура всех элементов на странице с учетом их вложенности.

Динамические шрифты

Технология, которая обеспечивает скачивание недостающих шрифтов с удаленных серверов и установку их на компьютер клиента.

Индекс цитирования

Величина, обозначающая число ссылок с других сайтов на данный сайт. Фактически определяет меру популярности данного сайта.

Каскадирование

Метод определения веса или важности конкретного объявления, который устраняет конфликты, возникающие в случае нескольких объявлений для одного и того же элемента.

Кегль

Высота площадки, на которой располагается литера. Обычно кегль вычисляется от начала верхнего выносного элемента до конца нижнего выносного элемента, плюс небольшие отступы.

Кернинг

Изменение величины пробела для *одной* пары букв (литер).

Контейнер

Контейнер элемента — это элемент блокового уровня, являющийся ближайшим предком данного элемента.

Контент

Содержимое интернет-сайта. Информационная часть документа, в которую входят текст, иллюстративные рисунки, таблицы, графики.

Кросс-браузерный CSS

Код таблицы стилей, обеспечивающий корректность отображения документа в разных браузерах.

Наследование

Механизм, который позволяет элементам-потомкам иметь те же стили, что и у элемента-предка.

Начертание

Комплект заглавных и строчных знаков, цифр и знаков препинания.

Нормальный поток

Простейшая схема позиционирования, которую по умолчанию использует браузер для форматирования документа.

Объявление

Представляет собой пару свойство/значение. Фактически задает определенный стиль для элемента.

Относительное позиционирование

Вид CSS-позишюнирования, при котором положение блока задается относительно нормального потока.

Паук (Spider)

Программный модуль поисковой системы, который посещает сайты и составляет их индекс. Чем-то напоминает простейший веб-браузер.

Плавающая модель

Вид CSS-позиционирования, при котором положение блока жестко не фиксируется.

Позиционирование

Механизм, который обеспечивает размещение блоков на странице.

Поисковая система

Состоит из паука и каталога индексов. Обеспечивает поиск по каталогу, находит совпадения, оценивает их релевантность и выдает результат пользователю.

Правило

Структурная единица таблицы стилей, которая содержит описание стилей для данного элемента.

Псевдоэлемент

Используется в селекторах для доступа к элементам, коюрые не включены в дерево документа. Например, первая буква или первая строка абзаца.

Псевдокласс

Используется в селекторах для ассоциаций с внешней но отношению к документу информацией. Например, информацию о том, что ссылка уже посещена, предоставляет; браузер.

Селектор

Конструкция, позволяющая выбрать элемент, к которому будут применены данные стили.

Слой

Блок, который имеет определенное положение на оси z Самая близкая аналогия — слои в графических пакетах, например, Photoshop. Координата *z* задасшя с помощью свойства z-mdex.

Спам

Массовая рассылка писем рекламного характера, а также некорректные способы увеличения посещаемости сайта.

Строчный элемент

Элемент документа, который не формирует структурных единиц и выводится линейной строкой.

Таблица стилей

Набор правил или свойств, которые описывают, как тот или иной элемент или группа элементов будут отображаться на экране монитора в браузере.

Табличная верстка

Тип верстки, при котором элементы на странице позиционируются с помощью таблиц.

Трекинг

Изменение величины пробела для *нескольких* букв (литер).

Устройства с постраничной разбивкой

Устройства, которые каким-либо образом разбивают контент на отдельные страницы. К ним относятся принтеры, проекторы ш . п

Фишка

Кра|кая формулировка понятия "Специфические особенности чего-нибудь" Например, совокупность фишек является основным признаком отличия данного браузера от всех остальных

Форма

Элемент интерфейса HTML-сграницы, с помощью которого организуется передача данных, введенных пользователем, на сервер для последующей обработки.

Шрифт

Набор символов, коюрые обьединены общими сіплевыми признаками Причем совокупность стилевых признаков является уникачьнои

Элемент

Элемент языка HTML. Является основным носителем информации о разметке документа.

Якорь

Ссылка на контент, который находится на текущей странице.