ЭФФЕКТИВНЫЕ ТЕХНОЛОГИИ ДЛЯ ПОСТРОЕНИЯ СОВРЕМЕННЫХ ВЕБ-САЙТОВ

HTML и CSS
Путь к совершенству

Бен Хеник

O'REILLY®

ПИТЕР®

ББК 32.988.02-018
УДК 004.738.5
Х38

Хеник Б.
Х38 HTML и CSS: путь к совершенству. — СПб.: Питер, 2011. — 336 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-49807-864-9

HTML и CSS являются основными рабочими инструментами в современном веб-дизайне и веб-программировании. Параллельное использование этих технологий для создания качественных интернет-сайтов требует от разработчика не только знания самих языков программирования, но и «продвинутых» техник и приемов, позволяющих изящно решать поставленные задачи, тратя на это намного меньше времени и усилий.

С помощью данной книги вы узнаете, как эффективно использовать в своей работе подобные техники. Неважно, верстаете ли вы HTML-страницы вручную или используете в своей работе готовые шаблоны, — это издание поможет вам более продуктивно работать на каждом из этапов разработки сайта: от разметки страниц до использования типографики и работы с цветом.

ББК 32.988.02-018
УДК 004.738.5

ISBN 978-0-596-15760-9 (англ.)
ISBN 978-5-49807-864-9

Краткое оглавление

Оглавление

Предисловие

HTML и CSS применяются уже более десяти лет, и их развитие не стоит на месте. За пятнадцать лет работы разработчиками были реализованы всевозможные типы проектов, опробованы различные функции и выявлены их достоинства и недостатки.

Но разработчикам HTML и CSS, несмотря на все их усилия, не всегда удавалось применять правильно эти технологии. Ряд экспериментов оказался не особо успешным. С другой стороны, потенциал некоторых решений превзошел все ожидания. Для совершенствования этих технологий было необходимо выявить бесперспективные элементы и прекратить работу с ними, а также выбрать наиболее удачные элементы, заслуживающие самого широкого применения. Уделяя внимание передовым методам в HTML и CSS, вы не только создаете работоспособные сайты, но и эффективнее проектируете наиболее перспективные их образцы с минимальными материальными и временны́ми затратами.

Для кого и о чем эта книга?

Надеемся, вы держите эту книгу после прочтения ее великолепного обзора на одном из ваших любимых сайтов или по совету знакомого, настоятельно рекомендовавшего вам ее почитать. (Об этом автор может только мечтать.)

И все-таки вам необходима более подробная информация о том, нужна ли вам эта книга.

Если следующие абзацы затронут вас и в них будут представлены ваши цели, смело выходите из магазина с этой книгой под мышкой или хотя бы присядьте на ближайший свободный стул и начинайте читать.

Что значит «самое интересное»?

Длинные коды HTML и CSS однозначно утомляют. Такие строки просто вгоняют в спячку. В этом веб-технологии схожи с некоторыми фильмами, при просмотре которых зрителю не терпится промотать вступление и начать с самого интересного.

Эта книга нацелена на нетерпеливого читателя. Вводная часть, которую я настоятельно рекомендую освоить, представлена в главах 2 и 3. Если вы что-то упустили во вводной части, можно будет быстро проглядеть эти главы заново.

Практическое изложение материала приводит к потрясающим результатам: четкое освоение сложных приемов, контроль над появлением типичных

ошибок, разработка шаблонной разметки. Все это позволит избежать недоработок и множества других проблем.

Что нужно знать для чтения этой книги

Содержание этой книги предполагает, что вы знаете основы HTML 4.01, CSS-селекторы, пары свойство/значение. На сопутствующем сайте этой книги приведены ссылки на сайты, посвященные HTML и CSS и дающие исчерпывающую информацию по этим языкам. Но намного проще будет читать эту книгу тем, кто уже знаком с возможностями HTML и CSS.

Кроме того, эту книгу будет легче освоить, если у вас уже есть понимание таких терминов, как программное поведение, представление, контент и многослойная структура сайта или приложения.

Для тех, кто, возможно, высоко оценивает свои знания, базовые разделы о веб-документе, таблице стилей и структуре элементов изложены максимально кратко.

Идеальный читатель

Вы — просто идеальный читатель, если:

- вы уверены, что, когда дойдет дело до создания серверной части приложения, то переделка дизайна станет настоящей мукой, поскольку вы будете вынуждены заново пересматривать код на предмет проверки элементов разметки, расставленных по всему коду. Наиболее эффективным решением данной проблемы является техника «CSS-Дзен», проиллюстрированная на сайте Дэйва Ши www.csszengarden.com. Согласно этой книге, CSS-Дзен предлагает структурирующий метод создания разметки, позволяющий свести изменение дизайна к переработке таблиц стилей;
- вы хорошо работаете в интегрированной среде разработки (IDE) приложений, например Adobe Dreamweaver или Microsoft Visual Studio, но ваши задумки не реализуются вследствие ограниченных возможностей этих программ. Если создавать документы при помощи IDE-среды, то в них, как правило, начинает скапливаться программный хлам (например, «лишние символы, технический мусор»), усложняющий программный код. Это происходит из-за отсутствия гибкости у IDE-среды. В этой книге HTML и CSS объяснены на таком уровне, чтобы вы уже сами могли выбирать свои инструменты для выполнения специфических ежедневных задач;
- у вас почему-то масса вредных привычек, которые нужно заменить хорошими. Кто-то из вас до сих пор пользуется HTML для представления данных, а не их структурирования, мало используя при этом возможности CSS. В данной книге раскрывается весь потенциал CSS;
- вы опытный графический дизайнер, жаждущий понять плюсы и минусы веб-среды для получения нового опыта. Вам известно про HTML и CSS, и вы понимаете, что эти языки дополняют друг друга, но не умеете этим

пользоваться. В книге подробно объяснена связь между ними. Поэтому вы сможете освоить технологию применения этих языков там, где это нужно;

- вам в силу профессии желательно или даже необходимо разрабатывать требования к проектам или обеспечивать доступность проекта в других приложениях. Без продуманной CSS мало шансов на создание сайтов, оптимизированных под другие приложения. Еще меньше шансов оптимизировать такие сайты под начинающих пользователей. В этой книге объясняются принципы разработки сайта таким образом, чтобы требования к доступности можно было реализовывать, не разрабатывая при этом параллельно множество других сайтов;
- вы специалист по созданию проектов на уровне представлений и хотите упростить себе работу. Проще говоря, более узкая специализация ведет к утрате общих навыков, что, в свою очередь, создает препятствия для слаженной работы в команде. В этой книге представлены приоритеты для разработчиков, чья работа тесно связана с посещаемостью сайтов. Следование этим приоритетам даст вам информацию, необходимую для слаженной работы в команде;
- вы устали биться головой о кирпичную стену, именуемую Internet Explorer 6. Поисками решения проблемы, связанной с кошмарной реализацией таблиц стилей для существующих версий Internet Explorer, занимаются несколько сайтов, особенно стоит выделить сайт «Position Is Everything». Тем не менее большинство интернет-ресурсов содержит огромное количество разных ошибок. В главе 12 будет вкратце изложена суть проблемы, когда сайт оказывается «под колпаком», что приводит к нежелательным коллизиям и сбоям. Также в ней представлен набор методов, которые помогут вам избежать всех этих проблем.

Книга может показаться скучной (либо слишком сложной)

Есть вероятность того, что некоторые разделы этой книги вам уже знакомы, поскольку предназначена она для широкого круга специалистов. Иногда материал, предназначенный для программистов, может показаться слишком очевидным для дизайнеров, и наоборот. Иногда изложенный материал может показаться вам спорным. Творческие решения, а также методы разработки очень часто определяются популярностью, а не выгодой. Я надеюсь, что данная книга может применяться для развития рациональных идей, а не бесполезных.

Если все в этой книге покажется вам в новинку, значит, возможно, вы просто торопите события. Сопутствующий сайт, посвященный книге, создавался по большей части для новичков, чтобы те убедились, что данная книга будет полезна всем, кто ее приобрел.

Однако если вам кажется, что материал не по зубам, то, возможно, придется нелегко. Лучше всего преодолевать эти трудности, набравшись терпения и засыпая вопросами своих коллег и товарищей.

Цели этой книги

В этой книге на доступном языке объясняются хитросплетения HTML, CSS и структуры документа, которые нелегко даются начинающим без руководства. С помощью этой книги вы сможете:

- выбрать оптимальную версию HTML для вашего проекта;
- поднатореть в последовательной эффективной разметке;
- использовать HTML для структуры документа, а не его представления;
- маскировать пока еще полезные HTML-элементы;
- сделать так, чтобы встраиваемый контент, черт возьми, заработал;
- максимально эффективно и правильно применять таблицы;
- освоить метод управления CSS-селекторами, особенно селекторами дочерних элементов;
- освоить приоритетность CSS-селекторов;
- освоить блочную разметку с использованием CSS;
- разобраться в проблеме наложения полей элементов;
- ознакомиться с ошибками и другими проблемами браузера Internet Explorer 6;
- освоить различные формы представления;
- понять, почему в браузерах так много ошибок;
- понять, что делает HTTP-протокол без вашего ведома (и зачем это нужно).

Эта книга пытается осветить все аспекты, которые необходимо знать веб-разработчику. В книге представлено описание многочисленных связей между уровнями веб-приложений, с которыми работают дизайнеры и разработчики сайтов. Кроме того, книга освещает положительные аспекты HTML и CSS.

Для начинающего разработчика книга предлагает множество приемов CSS-разметки, необходимой для разработки представления на должном уровне, улучшения доступности и поисковой оптимизации, а именно:

- выравнивание контента по центру;
- работа с улучшенным приемом замены изображений по методу Фарнера для разработки растрового заголовка;
- создание ровных колонок одинаковой (или почти одинаковой) высоты;
- применение CSS-свойства `float` для оптимальной верстки колонок и разметки документа;
- организация разнообразной визуально интересной навигации;
- освоение приемов разработки сайтов с использованием технологии AJAX;
- максимально эффективное применение CSS-свойства `position`;
- разработка разнообразных сеток для ваших сайтов.

Прочтение этой книги должно вдохновить читателя с накопленным багажом знаний на реализацию доступных, удобных и оптимизированных под поисковые системы сайтов, несмотря на сложность поставленных задач.

Чего нет в этой книге

Эта книга в основном рассказывает о методах максимального увеличения эффективности разметки и таблиц стилей. Поэтому в ней не рассматривается ряд других тем.

- *Несовместимые с рядом платформ функции расширенных версий CSS.*

  С помощью CSS можно вытворять массу забавных вещей... но, к сожалению, некоторые из них основаны на плохо поддерживаемых CSS-селекторах и свойствах. Подобная проблема будет решаться в рамках *намеченных целей*: если фильтр ActiveX, поддерживаемый Internet Explorer, имеет аналог в браузере Firefox, то, возможно, он будет упомянут. Или наоборот, будут упоминаться свойства `-moz-*`, имеющие аналоги в исполняемой среде IE. *Минимальные* требования для обсуждения определенных приемов разработки в этой книге — это поддержка таких приемов браузерами Firefox 3 и Internet Explorer 8. А для освещения методов обработки скрытых элементов понадобится более широкая браузерная поддержка.

- *Свойства CSS, нацеленные на полускрытые типы элементов.*

  Здесь будут рассмотрены методы, оптимально подходящие для создания сайтов с хорошей доступностью, однако эта книга содержит лишь *вводную информацию*, касающуюся разработки сайтов для посетителей с ограниченными техническими возможностями.

- *Язык JavaScript и объектная модель документов (DOM).*

  Несмотря на то что в книге будет упоминаться язык JavaScript и иногда демонстрироваться программный код языка, основной упор в ней делается на HTML и CSS. И это не означает, что в книге будут представлены методы управления HTML и CSS посредством модели DOM или JavaScript.

- *Объединение с фреймворками типа jQuery и YUI.*

  Многие люди не устают петь дифирамбы фреймворкам JavaScript, но вы не увидите их в этой книге. Несмотря на эффективность применения этих фреймворков в различных средах, их рассмотрение не входит в задачу книги. О взаимодействии фреймворков, стилей и разметок JavaScript лучше всего прочитать в специальных книгах и на веб-ресурсах, посвященных исключительно фреймворкам.

- *Подробное освещение CSS-фреймворков, например YUI Grids и Blueprint.*

  Эта книга нацелена на оттачивание ваших навыков, чтобы ваше резюме поразило не только работников отдела кадров, но и менеджеров по набору сотрудников. Поэтому книга нацелена на освещение принципа построения CSS-фреймворка, который может вам понадобиться для работы, а не на инструктаж по применению какой-то конкретной оболочки.

- *Методы настройки веб-сервера.*

  Стандартные настройки веб-сервера не учитывают ряд настроек, которые могут помочь в обеспечении удобства, доступности и соответствия стандартам.

Однако с этими проблемами по большей части работают системные администраторы. Этот вопрос рассматривается в ряде других изданий O'Reilly, в частности *Webmaster in a Nutshell* и *Website Optimization*. Время от времени исследованием этой темы занимаются некоторые интернет-сообщества и блоги.

- *Разработка сайтов для мобильных телефонов.*
  К сожалению, эта книга написана человеком, всю жизнь прожившим в США, где над стабильностью выхода в сеть через мобильный телефон нужно еще очень много работать. Популярность айфонов улучшила ситуацию, но она по-прежнему далека от нормальной. Ситуация такова, что только очень немногие пользователи мобильных телефонов могут работать с Интернетом наравне с владельцами ПК. Однако затраты в США на предоплату к подключению устройств, а также отсутствие повсеместного доступа к свободным эмуляторам для платформ мобильных устройств усугубляют проблему разработки контента для мобильных телефонов, которыми пользуются жители США. Я надеюсь, что в следующее издание этой книги можно будет включить техники разработки для посетителей сайтов, выходящих в Интернет через мобильные устройства.
- *Информация про браузер Opera.*
  Отказаться от освещения браузера Opera было тяжело, в отличие от других тем. К сожалению, когда я сравнил рыночный оборот компании Opera с количеством тестирований этого браузера, необходимым для его освещения в этой книге, то результаты оказались очень пессимистичными. Поскольку я должен как-то отблагодарить Криса Миллза из компании Opera за помощь в пробивании контракта на эту книгу, будьте уверены, это решение далось мне нелегко. Если внимание к этому браузеру не ограничится простым интересом читателя, я с готовностью возьмусь за его подробное освещение на сайте этой книги.

Веб-стандарты

Последний по списку, но не по значимости, вопрос связан с соответствием коммерческих проектов, особенно тех, что были разработаны крупными корпорациями, правилам Консорциума Всемирной паутины (W3C).

Я всегда разграничивал «рекомендуемые» и «требуемые» стандарты. Рекомендуемые веб-стандарты следуют основным принципам построения веб-приложений, и их просто достичь на практике, а вот требуемые стандарты основаны на четком следовании Рекомендациям консорциума и могут оказаться недостижимыми.

Сайт становится гораздо более эффективным благодаря рекомендуемым, а не требуемым стандартам. Ну а максимальная эффективность достигается соответствием стандартам двух типов. Данная книга рассматривает компромиссы и обходные пути рекомендуемых стандартов, а не условия побочного развития.

Возможно, вы заметили, что ранее я говорил о «так называемых» веб-стандартах. Скрытая ирония заключается в том, что никаких веб-стандартов, по крайней мере, официально, не существует.

Стандартизация требует сознательного применения формально установленной системы, используемой веб-индустрией в целом. Эта система, как правило (если не всегда), разрабатывается специальным учреждением по разработке стандартов, чья деятельность косвенно или напрямую связана с политикой и публикациями Международной организации по стандартизации (ISO).

Признак настоящих стандартов — объективный набор критериев и процессов, благодаря которым предъявляются требования на соответствие стандартам. Именно этих критериев очень сильно не хватает Консорциуму W3C.

Поэтому популярное определение Рекомендаций W3C как *стандартов*, по сути, является верным, но фактически не имеет никаких оснований.

Это говорит о том, что практика развития веб-стандартов шагнула далеко вперед с началом активного периода 1990-х. Эта тема очень подробно освещена на сопутствующем сайте этой книги.

Photoshop

В главах 9 и 10 довольно подробно рассказывается о методах создания изображений на основе пользовательского интерфейса Adobe Photoshop. Я выбрал именно этот подход потому, что в любой достаточно большой группе специалистов вы всегда найдете огромное разнообразие инструментов и способов реализации, но только если речь *не идет* о работе с изображениями. Альтернативные варианты (особенно Fireworks — еще один продукт Adobe) тоже имеют своих поклонников, но даже они согласятся с тем, что знакомство с инструментарием и пользовательским интерфейсом Photoshop весьма полезно.

Причины моего выбора нельзя назвать объективными; с тех самых пор, как я был абсолютным новичком, я не пользовался для обработки веб-изображений ничем, кроме Photoshop. Я надеюсь, что посетители сопутствующего сайта книги поделятся с нами собственными способами реализации методов, описанных в этой книге.

Вопрос выбора Photoshop связан также с важностью правильного выбора инструмента для максимальной эффективности рабочей группы. В главе 4 говорится о значимости стандартов и библиотек, однако выбор однотипных инструментов обязан своими преимуществами также и стандартному программному обеспечению.

Чем интересен сопутствующий сайт

На сопутствующем сайте этой книги www.htmlcssgoodparts.net представлена масса полезной информации. В частности, вы найдете тут:

- статьи из блогов, в которых содержатся ответы на вопросы читателей, а также описания текущих технических разработок и оптимальных методов разработки;

- подробную демонстрацию методов, представленных в данной книге, с разметкой документа, правилами таблицы стилей и индексированием номеров страниц;
- библиотеки и/или шаблоны для многоколонной верстки и другие приемы;
- ссылки на материалы других сайтов по HTML и CSS;
- пользовательские обзоры книг и программ, которые могут заинтересовать людей, читающих эту книгу.

Условные обозначения

Названия различных элементов технологий меняются в зависимости от контекста употребления. Чтобы свести потенциальную путаницу к минимуму, расшифрованные ниже и выделенные курсивом термины будут использоваться на протяжении всей книги.

Файлы — дискретные узлы файловой системы сервера-хоста, а *ресурсы* — документы или части документов, обозначаемые дискретным универсальным идентификатором ресурса (URI). Не все файлы представляют собой URI-идентификаторы, и не все URI-идентификаторы представляют собой файлы. URI-идентификатор может содержать в себе несколько файлов, результаты запросов в базу данных или потоков данных, а файл может представлять собой просто логический элемент, определяющий контент многих URI-идентификаторов.

Страницы или *документы* содержат один или несколько ресурсов произвольной классификации и выводят данные в пользовательский интерфейс на запрос единичного URI-идентификатора (или, возможно, многочисленных URI-идентификаторов на сайтах с поддержкой системы Ajax). Наконец, в этой книге практически не проведено различие между терминами URI и URL. Отчасти потому, что значение самого термина «ресурс» настолько размыто, что оно просто перестает быть функционально значимым со стремительным развитием технологий.

Контент — содержание сайтов.

Теги HTML, XHTML и XML обозначают разметку.

Таблицы стилей — это содержимое CSS-файлов или элементов `style`. *Правила* таблицы стилей определяют внешний вид одного или нескольких элементов страницы. Правило таблицы стилей содержит *селектор*, определяющий элемент(ы) на странице, к которым можно применить одну или несколько пар свойство/значение.

Браузеры также называют пользовательскими интерфейсами (ПИ), или клиентами.

Элементы HTML и CSS *обрабатываются* последовательно, и в ходе этого процесса браузер *отображает* страницу.

JavaScript — зарегистрированная торговая марка компании Sun Microsystems, которая в данной книге обозначает язык программирования, используемый для

обработки скриптовых данных и интерактивности в рамках браузера. Многие дистрибьюторы обозначают этот язык другими названиями во избежание судебных проблем с правами, но там, где есть браузер, там есть и интерпретатор JavaScript.

Объектная модель документов (или модель DOM) представляет структуру веб-документа и определение того, как эту структуру организовывать, запрашивать и обрабатывать программными средствами. Существует несколько спецификаций модели DOM для веб-документов, хотя Консорциумом Всемирной паутины разработан и санкционирован только один вариант модели.

Веб-сервисы, как правило, включают в себя операционную систему, веб-приложение, реляционную базу данных, скриптовый язык для сервера, HTML, CSS и JavaScript. Платформы варьируются в зависимости от дистрибьютора. Первые четыре относятся к *серверной* части, а последние три — к *клиентской* среде приложения.

Клиентская часть приложения искусственно поделена на четыре подуровня: *структуру* (определяемую разметкой), *контент* (обрамляемый разметкой), *презентацию* (определяемую CSS) и *программное поведение* (определяемое языком JavaScript). Вместе все четыре подуровня образуют архитектуру модель-вид-управление (MVC), отображающую и взаимодействующую с MVC-архитектурой серверной части.

Ajax — сокращение от «асинхронный JavaScript+XML», удобная технология разработки благодаря повсеместному применению элемента интерфейса прикладного программирования (API) `GetXMLHttpRequest`.

HTML-элементы — основные составляющие пространства имен в HTML.

Теги — символьная разметка, которая может также содержать атрибуты со значениями, и, как правило, теги открывают и закрывают контент.

Копия и *иллюстрации* для контента — то же, что и *текст* с *изображениями* для данных.

Объявление типа документа может (и, как правило, должно) появляться в начале исходного веб-документа и определяет версию HTML, которой должен соответствовать этот документ. Определение типа документа (также именуемое DTD) — это машиночитаемая серия объявлений, определяющая достоверность используемой версии HTML. Значения, содержащиеся в объявлении типа документа, содержат прямую отсылку к определенному DTD.

Рекомендации WC3 представляют собой официальные документы, служащие спецификациями для веб-технологий и оптимальными методами применения этих технологий.

Менеджеры проектов минимизируют проблемы, возникающие между командой разработчиков и заказчиками. *Дизайнеры* создают внешний вид, стиль и пользовательский интерфейс сайтов. *Программисты* и *разработчики* приложения разрабатывают и пишут код, запускающий сайт в работу. Группа *разработчиков уровня внешнего вида* сайта разрабатывают все, что имеет непосредственное отношение

к посетителям сайта. *Разработчики стилей* создают шаблоны и таблицы стилей, а *контент-менеджеры* отвечают за содержимое. Большинство других людей в командах по разработке веб-проекта занимаются его маркетингом и рекламой.

Современные браузеры, или агенты пользователей, — это те браузеры, которые на момент публикации книги нацелены на массовый рынок: Internet Explorer 6–8, Firefox 3.x и Safari 3.x–4.x.

Некоторые из приведенных тут терминов указывают на скрытые процессы, влияющие на взаимодействие веб-интерфейсов с пользователем. Эти процессы будут обсуждаться более подробно в этой книге.

«Почувствуй силу, Люк!»

Когда я впервые начал работать с веб-технологиями в 1995 году, совет «Почувствуй силу, Люк!» был, пожалуй, самым популярным советом в списке рассылки почты для зеленых новичков. Это цитата из фильма «Звездные войны: Новая надежда», рекомендующая обратиться к разметке (а теперь, 13 лет спустя, к правилам таблицы стилей), дающей превосходные результаты.

И этот совет не просто глупая научно-фантастическая шутка. Понимание наиболее эффективной разметки и стилей приходит от их чтения без фильтров. Точно так же «чувствующие силу» из «Звездных войн» раскрывали *свои* таланты по полной через избавление от пагубных мыслей.

Если вы попытаетесь выяснить, как кто-то достиг финальной цели, не прочувствовав силу, то вы сильно разочаруетесь... и если вы никогда не читали код приложения, то вы так и не сможете понять принципы его разработки.

Однако перед тем как углубиться в детальное изучение исходной разметки и CSS-кода, лучше всего взглянуть на веб-приложения как на систему связей между методами и технологиями.

Примеры кода

Эта книга предназначена для того, чтобы помочь вам выполнить вашу работу. Вам разрешается использовать коды, представленные в этой книге, для ваших программ и документации. Вам не нужно связываться с нами, за исключением тех случаев, когда вы берете чересчур большой кусок кода. Например, для написания программы, использующей несколько кусков кода из этой книги, разрешение не требуется. А вот для продажи или распространения компакт-дисков с примерами из книги разрешение уже необходимо. К вопросу о цитировании книги и примеров кодов: разрешения не требуется. А вот для включения большого количества примеров кодов из этой книги в документацию вашего продукта разрешение необходимо.

Мы будем благодарны за ссылки на источник, но не требуем их. Ссылка на источник, как правило, состоит из названия, автора, издателя и международного стандартного номера книги (ISBN).

Слова благодарности

Когда я вспоминаю свой пятнадцатилетний опыт работы создателем сайтов, то больше всего поражает невежество. Оно повсюду, и, как и многие другие создатели сайтов, я часто порицаю невежество менее опытных... но не в этой книге.

Почему?

По большей части в силу своего собственного невежества, которое в не меньшей степени заслуживает критики. За ним следуют упрямство и волнение, которые также постоянно присутствовали в моем внутреннем диалоге в течение всего года, посвященного написанию этой книги.

Эта книга пытается показать, что нужно зажечь свечу и помочь другим выйти из мрака, вместо того чтобы проклинать их. Я успокаиваю себя тем, что описанные производственные методы доказали свою эффективность. И я как можно спокойнее продаю накопленный опыт, не идя вразрез со своими принципами.

В общем, я попытался сделать из этой книги набор рекомендаций, на которых можно было бы озолотиться лет восемь или девять назад. Тогда многие (включая меня) искали решение по крупицам, методом проб, ошибок и случайных озарений и тут же делились этими крупицами знаний.

Я надеюсь, что эта книга будет полезной для вас сейчас, как когда-то она была бы полезной для меня, осваивавшего CSS.

Я достиг определенного мастерства и написал эту книгу благодаря многим людям, с которыми меня связала жизнь. Наконец мне представился шанс поблагодарить их публично. Поэтому перечислю их поименно. Помимо родителей, я благодарю Кристиана Сепеля, Стивена Чэмпиона, Сумина Ху, Тэдди Депнера, Ника Финка, Дэвида Хэмфилла, Молли Холцшлаг, Бренду Хьюстон, Этана Маркотта, Дана Петерсена, Лэнса Тэйлора, Томаса Вандера Вала, Питера Зале и Джефри Зэлдмана. Эти люди внесли огромный вклад в мою жизнь и без них вряд ли эта книга была бы написана.

Несколько человек стоит упомянуть отдельно. Один из них — Крис Миллз из компании Opera Software. Ему особая благодарность. Крис всегда поддерживал меня с этим проектом. Именно он порекомендовал меня в качестве автора-кандидата издательству. Крис помог мне сделать первые шаги на этом пути, пригласив меня сделать вклад в курсы обучения по веб-стандартам браузера Opera.

Содержание и качество это книги — не только плоды моего труда. От провала затеи меня уберегло безграничное терпение Саймона Сэнт Лоурента, моего редактора из издательства O'Reilly. Хотя на страницах книги изложены мои мысли, а на обложке значится мое имя, постоянная поддержка Саймона по этому проекту сыграла существенную роль в реализации моих начинаний.

Майкл Смит — ответственный за содержание этой книги по теме HTML5. Его имя не упомянуто на обложке. И это очень несправедливо по отношению к тому, кто помогал мне не подорваться на минном поле словесности.

Мне удалось найти трех редакторов: Кимберли Блессинг, Геза Лемона и Криса Вэн Домелена. Все они внесли решающий вклад в обеспечение точности и современности материала. Если эти аспекты где-то остались недоработанными, то это стоит относить исключительно на мой счет.

Кимберли и Крис также были моими верными помощниками: на протяжении нескольких лет я часто обращался к ним с техническими вопросами; мне просто никаких слов не хватит (как и в ряде других случаев), чтобы выразить в полной мере всю благодарность за оказанную ими помощь.

И наконец, хочется поблагодарить Эрика Мейера, задавшего высокую планку для всех, кто собрался учить премудростям разработки.

И последнее. Я надеюсь, что знания, почерпнутые из этой книги, помогут вам достичь тех высот, которых удалось достичь упомянутым здесь людям.

1 Сущность гипертекста

Современные веб-сайты — это гораздо больше, чем совокупность разметки, стилей, скриптов и мультимедийных ресурсов. Они позволяют в полной мере воспользоваться преимуществами гипертекстовой среды и сделать данную технологию основным инструментом для получения информации. Всемирная паутина без активных гиперссылок была бы всего лишь плохо структурированной грудой документов.

Гибкость гипертекстового документа не отменяет поставленную перед разработчиками задачу помочь пользователям быстро найти необходимую информацию. Посетители могут попасть на сайт неожиданным способом — со страниц или закладок, которые не подконтрольны вам. Поэтому возможности гипертекста должны быть использованы таким образом, чтобы посетитель мог без труда сориентироваться на сайте.

Интернет без ссылок

Использование ссылок — главное отличие Интернета от прочих информационных сред. Сегодня, когда Всемирная паутина стала обыденным явлением, многие не задумываются об этом. Однако именно ссылки стали основой современных веб-сайтов. Давайте представим, что случится, если удалить с сайта все гиперссылки?

- Самым главным результатом удаления гипертекста станет *строго линейное структурирование содержания*. Пользователь, прежде чем добраться до интересующих его материалов, будет вынужден прочесть значительное количество ненужной информации. Если убрать из гипермедиа все ссылки, результат будет практически бесполезным (если, конечно, вы не зададите для данных внутренний порядок и структуру).
- При *создании линейных ресурсов* предполагается, что читатель ознакомился с предыдущими разделами, прежде чем перейти к последующим. Возьмем для примера эту книгу. Мы можем начать читать ее с любого места, но, поскольку главы расположены по мере усложнения материала, пользы от такого чтения

будет немного. Кроме того, если бы не было сайта, сопровождающего эту книгу (http://www.htmlcssgoodparts.net/), в нее пришлось бы включить множество длинных примеров разметки.

- *Местоположение пользователя должно быть указано с помощью стандартных средств.* В каждой книге, а также в других линейных ресурсах, такие сведения указываются в колонтитулах или в строке заголовка специальных приложений. В больших электронных документах, реализованных, например, в формате PDF, «прогресс» прочтения можно отследить с помощью полосы прокрутки.

Итак, гиперссылки выводят документы на качественно новый уровень. Но вместе с тем появляются и неожиданные проблемы. Гиперссылки затрудняют «навигацию по сайту». При чтении линейно организованных документов можно опираться на традиционные подсказки и свое собственное «чувство места», в то время как пользователи ресурсов с гиперссылками нуждаются в помощи разработчиков для «ориентации в пространстве», так как в подобных документах понятие «начало» и «конец» весьма условно.

URI

В идеале URI (Uniform Resource Identifiers, унифицированные идентификаторы ресурсов), более известные как URL (Uniform Resource Locators, унифицированные указатели ресурсов), должны быть скрыты от посетителей сайтов. Они не столь удобны для чтения, так как включают в себя маркеры протоколов, псевдонимы хостов, а также элементы, похожие на ссылки, типичные для файловых систем, но на деле ими не являющиеся. URI часто заканчиваются парами параметр/значение. Это делает их скорее машиночитаемыми, нежели удобными для пользователей.

Мы привыкли к простым URI вида http://www.example.com/, которые указывают на главную страницу сайта. Такие URI часто встречаются в рекламных объявлениях и на визитках. Однако грамотно созданный URI может содержать много дополнительной информации. Достаточно взглянуть на часто встречающиеся URI на популярных поисковых и новостных сайтах. Например, URI результата поисков в Google может содержать параметр start, который указывает, с какой позиции отображать результаты поискового запроса. Точно так же популярные CMS (Системы управления содержимым), платформы и каталоги электронной коммерции позволяют связывать один и тот же ресурс с несколькими URI, где более длинный URI повышает доступность ресурса для поиска или объединяет с основным ресурсом дополнительное содержание (например, список продуктов или сводку посещения).

Браузеры и другие программы используют для обработки URI и получения информации протокол HTTP. Если вы хотите подробнее узнать об этих процессах, а также о том, как они могут повлиять на вашу страницу, см. приложение.

Управление ссылками

Гипертекст впервые был применен в Стэнфордском университете в 1960 году, но широко использоваться стал лишь три десятилетия спустя после повсеместного распространения Интернета. Интернет-бум способствовал не только объединению множества сетей в глобальную паутину, но и привел к пониманию того, что гиперссылки должны быть простыми и отказоустойчивыми.

Ссылка в HTML предполагает, что ее создатель знает о том, куда она приведет. Однако он далеко не всегда контролирует то, что находится «на другом конце» ссылки. Способность соединять любой контент без предварительного согласия его создателя является одним из основных факторов успеха Всемирной паутины. Если контент имеет URI, вы можете дать на него ссылку. Если URI не работает, то сайт сообщит об ошибке (например, выдаст всем известное «Ошибка 404: Страница не найдена») и перенаправит посетителя на страницу, откуда он сможет продолжить поиск.

Возможности, которые предоставляют ссылки на веб-страницы, поднимают нравственные (а в некоторых случаях и правовые) вопросы о том, что означает возможность прямой ссылки на чьи-либо материалы. В то же время возникают проблемы неработающих ссылок. Создание ссылок на ресурсы, которые вы не контролируете, рано или поздно приведет к тому, что ссылка может стать «битой», поскольку информация меняется, а сайты появляются и исчезают. Точно так же посетители вашего сайта через некоторое время могут не найти интересующей их информации. В постоянно меняющемся Интернете контроль за достоверностью ссылок — задача практически невыполнимая. С ней не могут справиться даже автоматизированные поисковые системы. Даже если ссылка ведет на существующую страницу, информация, находящаяся там, с течением времени может сильно измениться. Если вы проводите масштабную реконструкцию своего сайта, велик риск того, что часть ссылок будет вести не туда. Минимизации подобных проблем способствует простая и ясная навигация на сайте, а также грамотно сделанные страницы ошибок.

Обычно посетители легко справляются с неправильными ссылками. Гораздо сложнее бывает восстановить потерянные изображения, стили, коды или другие компоненты, особенно если им были присвоены четкие значения `href` и `src`. Чем важнее информация, тем больше должно быть желание поместить ее в надежное, контролируемое вами место.

Усовершенствование взаимодействия с пользователем с помощью ссылок

Ссылки — это часть языка HTML, посредством которой URI обрабатывается сетевыми приложениями. Можно сказать, что они находятся как бы на

пересечении HTML и HTTP. На прикладном уровне нет разницы между следованием по ссылке и получением URI из адресной панели браузера. Ссылки предоставляют создателям сайтов многочисленные преимущества, большинство из которых, тем не менее, остается без внимания. Гиперссылки в документах не ограничены навигацией, стилевым оформлением и агрегированием. Они могут вести к огромному количеству дополнительных ресурсов. Ссылки, отвечающие за диалог пользователя с системой, могут быть размещены где угодно и ограничиваются лишь техническими характеристиками платформы, а также чувством меры и воображением создателя сайта.

Грамотно внедренный гипертекст, помимо всего прочего, дает информации следующие преимущества.

- *Увеличение доступности и контроля за информацией.*
  Гиперссылки могут связать сайт с любой частью Всемирной паутины за исключением зон контролируемого доступа. Таким образом, вы избавляетесь от необходимости приводить множество примеров и побочной информации, а пользователи сами решают, к каким дополнительным ресурсам им следует обратиться и как это лучше сделать.
- *Выбор последовательности восприятия информации.*
  Пользователь может сам определить, что и в какой последовательности ему читать.
- *Общественное внимание.*
  Внешние гиперссылки усиливают доверие к контенту вне зависимости от его содержания. Этот принцип учитывают многие системы, в том числе алгоритм PageRank, использующийся в Google. Конечно, подобный подход напоминает поговорку о том, что «сотни леммингов не могут ошибаться», но постепенно ситуация меняется в лучшую сторону, поскольку «оценка» содержимого сайтов становится все более профессиональной.

Проблемы применения гипертекста

Сетевые технологии позволяют пользователям приобретать новые знания способами, считавшимися до 1992 года фантастикой. В настоящий момент нет никакого контроля за тем, какую информацию получает пользователь из Сети. Стандартная интернет-сессия обычно представляет собой посещение множества сайтов, практически не связанных между собой, и выражается в произвольном количестве взаимодействий пользователя с системой.

Это подобие анархии предъявляет к разработчикам следующие требования.

1. Контекст (т. е. значки типа «вы здесь» и «вам сюда») является, наряду с содержимым, важнейшей частью качественного веб-сайта.
2. Непроверенные предположения о целях и знаниях посетителей сайта ведут кратчайшим путем к неприятностям.

3. Дублирование содержимого приводит к пустой трате времени для пользователя (как и для создателя сайта).
4. Отсутствие в Сети рамок и ограничений может нанести пользователю вред, о чем разработчики сайтов *должны* постоянно помнить. Открытость Сети требует специальных знаний о доступности и архитектуре сетевой информации.

Поскольку Сеть *разрушает* традиционную линейную структуру документов, разработчики должны всегда помнить о том, что создание *контекста* — их первейшая обязанность.

2 Работа с разметкой HTML

Создание ссылок является одной из основных задач при разработке сайтов. Однако возможности HTML (языка разметки гипертекста) не ограничиваются только работой со ссылками. Он позволяет работать со скриптами, изображениями, видеофайлами, звуком и многим другим.

С момента своего создания в 1992 году HTML постоянно совершенствуется параллельно с развитием сетевого программного обеспечения (браузеров и интегрированных сред обработки). Сейчас, несмотря на довольно почтенный возраст, он остается широко востребованным. Благодаря простому и понятному синтаксису HTML позволяет создать дерево документа, удобное для хранения контента, применения стилей и управления документом. Компоненты CSS, предназначенные для взаимодействия с деревом документа, подробно описаны в главе 5 этой книги.

Синтаксис HTML

HTML, как и родственный ему более строгий XHTML, не является языком программирования. Это скорее набор правил по разметке документов. Кроме того, они определяют структуру этой разметки. Парсеры HTML (но не XHTML) следуют закону Постела:

Будьте либеральны к тому, что принимаете, и консервативны к тому, что отправляете.

Пустые элементы и пропуски автоматически исключаются без сообщения об ошибке, что дает пользователю возможность прочитать документ, хотя, возможно, и не в том виде, в каком его задумал создатель страницы. XHTML, напротив, требует от создателя сайтов точности в разметке и использовании тегов.

Теги, элементы и атрибуты

HTML описывает множество *элементов*, каждый из которых принадлежит к определенной семантической группе и имеет имя на английском языке. Эти элементы и определяют структуру документа, его вид, а также возможность совершения с ним каких-либо действий.

Каждый из элементов документа обозначается с помощью одного или двух *тегов* – команд, представляющих собой заключенные в угловые скобки имена элементов. *Открывающий тег* всегда начинается с открывающей угловой скобки <, следующего за ней имени элемента и параметров, связанных с этим элементом. *Закрывающий тег* состоит из заключенной в угловые скобки косой черты, идущей перед названием элемента.

У элементов без закрывающих тегов есть следующие особенности.

- HTML-элементы с *необязательными* закрывающими тегами — в первую очередь `li` (элемент списка) и `p` (абзац) — могут вообще не иметь закрывающего тега.
- HTML-элементы, для которых использование закрывающего тега *запрещено*, ничем не отличаются от открывающих тегов остальных элементов.
- В XHTML понятие о необязательном закрывающем теге отсутствует. Все закрывающие теги являются либо обязательными, либо запрещенными.
- Элементы, для которых использование закрывающего тега запрещено, в XHTML должны иметь на конце не >, а />. При этом во избежание ошибок при обработке документа агентами пользователя перед этим сочетанием символов часто вставляется пробел.

Теги могут содержать любое количество пробелов. Параметры открытого тега можно писать в любом порядке.

Параметры тегов, как правило, имеют какие-либо значения и используются для изменения элементов. В отличие от HTML, где регистр символов не важен (кроме параметров `class` и `id`), в XHTML все элементы и параметры следует указывать строчными буквами. Также при создании XHTML-страницы следует помнить, что при использовании параметра ему обязательно следует присваивать какое-либо значение. В случае, если параметр не содержит никаких значений, принято вместо значения дублировать имя параметра (например, `checked="checked"`).

Листинг 2.1 показывает фрагмент грамотной разметки по стандарту XHTML 1.0 Transitional.

Листинг 2.1. Фрагмент разметки по стандарту XHTML 1.0 Transitional

```
<div id="header"><h1><a href="/">AcmeStore.com</a></h1></div>
<img src="/images/portrait.gif" width="144" height="180" alt="Это просто
чей-то портрет." />
```

Как мы видим, в первой строке примера представлены три элемента, как бы вложенные один в другой наподобие матрешки. Следует помнить, что вложенные

элементы следует закрывать в порядке, *обратном* их открытию (рис. 2.1). Несоблюдение этого принципа является одной из самых распространенных ошибок при создании страниц.

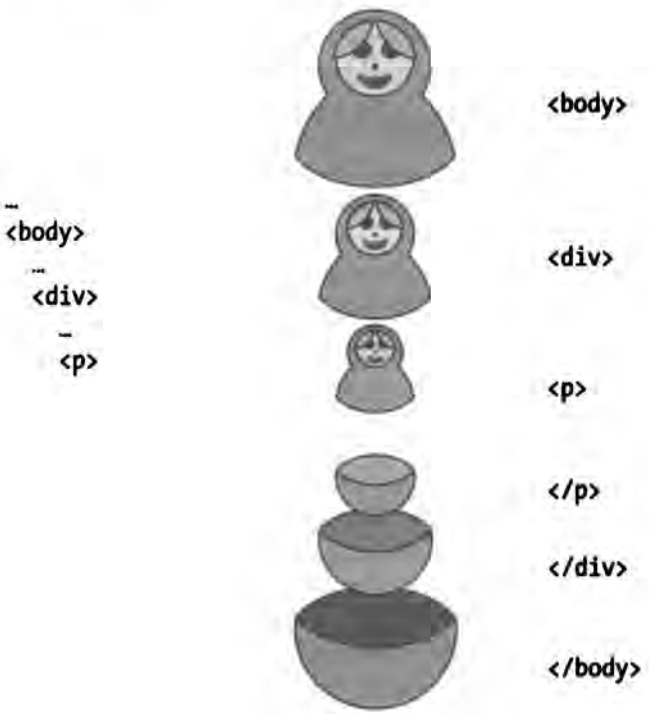


Рис. 2.1. Правильно организованные вложенные элементы XHTML напоминают матрешку

Обратите внимание: все параметры, как того требует XHTML, заключены в кавычки. Правила для параметров в HTML немного отличаются:

> В некоторых случаях создатели страниц могут не использовать кавычки. Значение параметров может содержать буквы (a–z и A–Z), цифры (0–9), дефисы (десятичный код ASCII 45), точки (десятичный код ASCII 46), нижние подчеркивания (десятичный код ASCII 95) и двоеточия (десятичный код ASCII 58). *Следует помнить, что спецификации HTML (например, 4.01) рекомендуют использовать кавычки даже в тех случаях, когда без них можно обойтись.*
>
> *Спецификация HTML 4.01, Консорциум Всемирной паутины*

Применение специальных символов в тексте и в URL подробно описано в разделе «Представления символов, не принадлежащих ASCII, в виде сущностей» на с. 258 и в разделе «Кодирование символов в URL: сущности ASCII» на с. 282.

Структура страницы

Если загрузить в браузер HTML-страницу, то он будет пытаться обработать ее содержимое, опираясь на разметку документа. Даже если какие-то теги отсутствуют или содержат ошибки, браузеру, скорее всего, удастся отобразить страницу, пусть и не совсем в том виде, в каком задумывал ее автор. А вот для того чтобы веб-документ прошел процедуру валидации (был правильно оформлен), необходимо, чтобы он обязательно содержал ряд расположенных в строгом порядке элементов.

1. Объявление типа документа.
2. Тег `<html>`.
3. Тег `<head>`, вложенный в тег `<html>`.
4. Вложенный в `<head>` тег `<title>` и при необходимости теги `<link>`, `<script>`, `<base>` и `<meta>`.
5. Тег `<body>`, который должен быть вложен в `<html>` и располагаться после `<head>`. В данном теге должно находиться все то, что будут видеть посетители вашей страницы.
6. Как минимум один тег, вложенный в `<body>`.

Режимы отображения, версии HTML и объявление типа документа (DTD)

Как уже упоминалось ранее, HTML постоянно совершенствуется на протяжении последних 17 лет. За это время было создано пять версий этого языка. Самая последняя из них, HTML5, находится в разработке. Кроме того, Консорциум Всемирной паутины (W3C) опубликовал рекомендации для XHTML, XML совместимой версии HTML 4.01.

Поскольку разработка HTML5 еще не закончена, в этой книге мы не будем подробно останавливаться на нем, а только вкратце опишем его функциональные отличия.

Начиная с версии 1.0 в HTML используется *объявление типа документа* (document type declaration). Например, объявление типа документа для версии HTML 4.01 Strict будет выглядеть следующим образом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Основное назначение объявления типа документа — определить корректный режим отображения.

Аббревиатура «DTD», которую вы можете видеть в примере, расшифровывается как «Document Type Definition» (определение типа документа). Именно «Document Type Definition» содержит информацию о синтаксических правилах для документа. Теоретически, все DTD может быть вставлено в объявление типа документа, но на практике используется лишь ссылка на него. Различные нюансы применения DTD и объявления типа документа подробно рассмотрены на сопутствующем сайте этой книги (http://www.htmlcssgoodparts.net/).

HTML или XHTML?

В настоящее время наиболее популярными «разновидностями» HTML являются вариации HTML 4.01. Часть из этих разновидностей следуют традиционному синтаксису HTML, другие более соответствуют требованиям XML. XHTML, особенно в совокупности с корректным MIME-типом (см. таблицу в приложении, с. 332), лучше всего отвечает жестким синтаксическим требованиям XML.

Традиционные версии HTML следуют весьма свободным правилам: в них разрешены незакрытые теги, не учитывается регистр букв и т. п. XHTML, как уже упоминалось выше, требует обязательного закрытия всех тегов и написания всех команд прописными буквами. Кроме того, существенным недостатком XHTML является тот факт, что его канонический MIME-тип не поддерживается браузером Internet Explorer (подробнее см. главу 14). Однако тщательно отформатированный XHTML (или HTML, созданный с соблюдением правил XHTML) гораздо проще читать. Вот почему в качестве примеров разметки в этой книге приводятся именно фрагменты XHTML.

HTML5 поддерживает синтаксис XML, но не требует его обязательного использования.

Strict, Transitional или Frameset?

Поскольку HTML постоянно совершенствуется и изменяется, некоторые теги устаревают и получают статус *нежелательных к применению*. Кроме того, многие теги имеют ограниченную область применения — они могут встречаться только внутри определенных элементов или обязаны содержать определенные элементы.

Если говорить кратко, эти разновидности отличаются степенью строгости в отношении ряда правил. Strict накладывает минимум ограничений на использование определенных элементов в качестве содержимого или контейнеров, но является строгим в отношении тегов, имеющих статус нежелательных, — их использование запрещено.

Frameset предназначен для создания страниц, содержащих фреймы. Более подробно о Frameset и фреймах можно прочитать в главе 14.

Наконец, следует помнить, что тег `<iframe>` применяется не в Frameset, а в Transitional.

Для HTML5 применим только один тип документа — Strict, дополненный некоторыми новыми возможностями, характерными для HTML 5.

Благодаря свойству `box-sizing` CSS3 позволяет менять эти характеристики вне зависимости от типа документа. Более подробно об этом свойстве, а также о значениях, которые оно может принимать (`content-box` и `border-box`), см. главу 6.

Два типа блочных моделей

Для современных браузеров объявление типа документа служит своего рода переключателем, который устанавливает блочную модель, определяющую параметры отображения страницы.

Некоторые параметры этих блоков, определенные HTML, можно изменить с помощью различных функций CSS. Каскадные таблицы стилей позволяют изменять такие параметры страницы, как внешние границы, отступы, поля, фон и т. д. В старых версиях браузеров эти параметры определялись субтрактивно — отступы и поля определялись исходя из параметров текста, заданных создателем страницы, а внешние границы обсчитывались отдельно.

Несмотря на то что, согласно спецификации CSS 2.1, параметры, определяющие границы, поля и отступы должны быть указаны создателем страницы отдельно для каждого элемента, в Сети еще осталось огромное количество веб-страниц, адаптированных для отображения в старых версиях браузеров.

Используя объявление типа документа, создатель сайта устанавливает модель отображения элементов страницы. При этом некоторые модели отвечают стандартам CSS 2.1, а некоторые основаны на старом подходе (часто называемым «режимом совместимости»). В документах, где тип не объявлен, по умолчанию применяется устаревшая блочная модель.

Приведенные в нашей книге и на сайте примеры выполнены с учетом требований к блочной модели, отраженных в спецификации CSS 2.1.

Подробный список вариантов `DOCTYPE` и моделей их отображения различными браузерами можно найти на сайте http://hsivonen.iki.fi/doctype/.

Выбор правильного типа документа

Для опытных создателей сайтов выбор типа документа, прежде всего, вопрос личных предпочтений. Я, например, предпочитаю работать в XHTML 1.0 Transitional, поскольку он наиболее близок к XML, а строгий синтаксис позволяет легче контролировать качество выполненной работы. Однако это лично мои предпочтения, и не стоит вам их навязывать. Чтобы выбрать наиболее подходящий для вашего проекта тип, я рекомендую ответить на несколько важных вопросов.

- Какой тип HTML используется в других проектах вашего заказчика или какой тип он бы предпочел использовать?

  Использование идентичных типов HTML упростит связь новой страницы с уже существующими.

- Будет ли контент доступен для различных систем обработки?

  В этом случае рекомендуется использовать XHTML по причине его хорошей межсистемной совместимости.

- Какова вероятность того, что ваша страница будет обрабатываться алгоритмами преобразования, такими как функция поиска и замены?

  Лучше всего с такими алгоритмами взаимодействуют типы документов Strict.

С другой стороны, основой для выбора типа могут служить используемые для создания страницы функции. Помните, что лучше всего пользоваться привычными инструментами и не прибегать к новым, если это не является действительно необходимым.

Универсальные атрибуты

Читая эту книгу, вы будете часто встречать атрибуты `class` и `id`. Это универсальные атрибуты, которые можно использовать с любыми элементами HTML. Кроме `class` и `id` в число универсальных атрибутов входят:

- `title`
- `lang/xml:lang`
- `dir`
- `style`

Атрибут `dir` определяет направление вывода текста. Атрибут `style` подробно описан в главе 14.

Создание стилевых крючков с помощью class and id

Атрибуты `class` и `id` могут быть применены ко всем элементам в HTML. Каждому конкретному элементу можно присвоить несколько значений `class`, но

только одно значение id. Множественные значения class записываются через пробел, например:

```
class="alternate callToAction"
```

Валидные значения id и class должны состоять только из букв, цифр, дефисов и нижних подчеркиваний и должны начинаться только с букв или цифр.

Более важнен вопрос о том, где следует применять class и id. Как правило, class используются в элементах, которые служат одновременно и конструктивным, и оформительским целям. На многих сайтах class также присваивается элементу body на страницах, выделяющихся из общего ряда архитектуры сайта (например, «О компании» или «Контакты»).

Общая структура шаблонов сайта дает представление о том, где и как применены атрибуты id и class. К каждому из элементов шаблона сайта я, как правило, присваиваю следующие значения id:

- main
- header
- primaryNav
- bodyCopy
- sidebar
- footer
- secondaryNav

Если внимательно взглянуть на этот список, то можно заметить полное отсутствие параметров, отвечающих за координаты (правая или левая колонки), цвет или особые размеры блоков. Похожую картину можно наблюдать и в отношении class — они практически не используются для определения абсолютных размеров или цвета объектов. Исключение делается разве что для стиля формы, где значения class short, medium и long используются для того, чтобы избежать многократного повторения пар label/field.

Описание контента с помощью title и lang

Помимо id и class в HTML 4.x и XHTML 1.x существует два универсальных атрибута, предназначенных для предоставления информации о языке, на котором написана страница, и о ее содержимом — lang и title. Атрибут title встречается почти повсеместно и представляет собой краткое описание содержимого элемента. Также он отображает заголовок ссылки (это свойство широко применяется в Википедии). Наконец, браузер может отобразить значение атрибута title в виде метаданных документа. Грамотно примененный атрибут title может оказать неоценимую помощь пользователю, пытающемуся найти крохи нужной информации среди терабайтов контента.

Атрибут title можно сравнить атрибутом alt, с той лишь разницей, что alt заменяет изображения, которые не могут быть отображены, а title служит дополнением к контенту.

В современных браузерах значение атрибута `title` отображается в окне всплывающей подсказки, если навести мышь на связанный с ним элемент (рис. 2.2). Некоторые браузеры убирают всплывающую подсказку, если она отображается слишком долго. Временной интервал варьируется в зависимости от браузера.

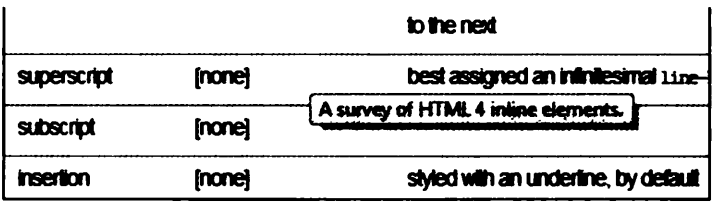


Рис. 2.2. Всплывающая подсказка в Internet Explorer 8 Vista/Aero

Основное назначение атрибута `lang` — это предоставление дополнительной информации об элементах контента, написанных на иностранных языках. Правила вежливости требуют использовать атрибут `lang`, если часть материала на странице написана на иностранном языке. Кроме того, программам, зачитывающим текст с экрана, необходимы корректно оформленные атрибуты `lang` или `content-language` для правильного произношения иностранных слов.

Также следует упомянуть атрибут `hreflang`, являющийся аналогом `lang` и применяемый в случае, если нужно сделать ссылку на документ, написанный на иностранном языке.

Наконец, следует помнить, что при использовании XHTML и MIME-типа `application/xhtml+xml` необходимо вместо `lang` использовать атрибут `xml:lang`.

Если вы хотите применить атрибут `lang` или `xml:lang` ко всему документу, то его следует добавлять к тегу `HTML`, а не к отдельным его элементам.

Значение атрибута `lang` (так же как и поля HTTP-заголовка ответа `Content-Language`) выбирается из списка, составленного из различных кодов, созданных в соответствии с требованиями ISO по правилам, установленным IETF (Специальная комиссия интернет-разработок).

Для более подробной информации об использовании атрибута `title` в ссылках см. раздел «Эффективные названия ссылок и значения заголовка» на с. 162. Значения атрибута `lang` приведены в табл. 2.1.

Таблица 2.1. Наиболее популярные значения атрибута Content-Language

Язык	Значение lang/Content-language
Английский	en
Американский английский	en-US
Британский английский	en-GB
Китайский (упрощенный)	zh-Hans
Китайский (традиционный)	zh-Hant
Китайский (тайваньский, нет специального написания)	zh-TW
Испанский	es
Японский	ja
Французский	fr
Португальский	pt
Португальский (Бразильский)	pt-BR
Немецкий	de
Арабский	ar
Русский	ru
Корейский	ko

Атрибут contenteditable в HTML5

В спецификацию HTML5 добавлено несколько новых атрибутов, в том числе `contenteditable`, поддерживаемый большинством современных браузеров. Этот атрибут предназначен для поддержки браузерных встроенных редакторов rich-text/WYSIWYG («What You See Is What You Get», «что посеешь, то и пожнешь»). Подобный интерфейс можно увидеть в различных блогах.

Атрибут `contenteditable` позволяет сделать отдельные части контента редактируемыми для пользователей. Посетители страницы могут производить различные манипуляции с текстом — выделять, копировать и вставлять, перемещать текст (в том числе и с помощью мыши), изменять шрифт на полужирный или курсив, менять цвет шрифта и даже добавлять гиперссылки.

Если просто добавить атрибут `contenteditable` к элементу текста, то это даст пользователям возможность применять функции редактирования, вызов которых доступен с помощью горячих клавиш (`Ctrl-X` — вырезать, `Ctrl-V` — вставить, `Ctrl-B` — для полужирного шрифта и `Ctrl-I` для курсива). Некоторые браузеры даже выдают текстовое меню при щелчке правой кнопкой мыши на элементе контента, к которому применен атрибут `contenteditable`. Благодаря этому посетители страницы могут воспользоваться функциями редактирования, для которых нет горячих клавиш (например, изменить размер и цвет шрифта).

Возможно, в скором времени во всех браузерах будут реализованы дополнительные функции редактирования, такие как, например, добавление пользователем гиперссылок с помощью контекстного меню. Однако в настоящее время

для создания пользовательского интерфейса редактирования, поддерживаемого всеми браузерами, необходимо использовать JavaScript. Можно запросто добавить кнопку, позволяющую сделать шрифт выделенного фрагмента полужирным, но для того, чтобы эта кнопка правильно работала, вам придется добавить несколько скриптов. (Спецификация HTML5 содержит большое количество API, которые предназначены для упрощения работы скриптов с атрибутом `contenteditable`, но мы не будем подробно на них останавливаться.)

Среди недостатков `contenteditable` следует отметить отсутствие функций, позволяющих пользователям сохранять отредактированные ими изменения. Интерфейс для подобных действий вам придется создавать самостоятельно.

Разделение контента, структуры, презентации и интерфейса

Создание абсолютно безопасного сайта

Представьте себе здание. Его основу составляет прочный каркас, стоящий на прочном фундаменте и покрытый различными панелями.

С точки зрения пользователя, каркас и панели — это и есть структурные оболочки сайта.

Когда вы начинаете украшать ваш дом, красите стены, добавляете колонны и флюгера — это соответствует созданию презентационной оболочки вашего сайта с помощью CSS. И если дальше продолжать аналогию, то, подобно тому, как стены и крыша развалятся, если каркас дома будет недостаточно прочен, использование CSS невозможно, если разметка страницы выполнена неграмотно.

В благоустроенном доме есть двери, окна, электричество, водопровод, системы кондиционирования и еще многие вещи, делающие жизнь комфортнее. Точно так же удобный интерфейс упрощает работу с сайтом. Однако интерфейс, даже очень хороший, окажется малоэффективным, если другие части архитектуры сайта не сконструированы должным образом.

Дом, каким бы он ни был, прежде всего предназначен для жизни людей, а сайт — для контента. Поэтому оболочка, интерфейс и разметка страницы — лишь вспомогательные инструменты, используемые для лучшего отображения контента.

Разделение на практике

Итогом внедрения принципа разделения на практике является то, что каждая из оболочек сайта, насколько это возможно, независима от остальных. Такая независимость, естественно, не может быть полной, поскольку в этом случае

потребовалось бы дублировать ресурсы. Как бы то ни было, следуя принципам разделения, необходимо учитывать следующие правила.

1. Интерфейс сайта, даже самый продвинутый, не спасет ситуацию, если отсутствует презентационная часть сайта.
2. Презентация сайта зависит от качества внутренней структуры.
3. Структура сайта не поможет без тщательно проработанного контента.

Однако достичь уровня независимости, при котором последствия изменений какого-либо элемента страницы сведены к минимуму, достаточно просто. Например, можно создать атрибут `class`, который назначается элементу только в случае, если пользователь взаимодействует с ним. Это позволит менять вид элемента, не затрагивая фрагменты JavaScript, определяющие его поведение. Точно так же можно полностью изменить внешний вид сайта, не меняя его структуру и контент. Подобные действия наиболее приближены к философии «CSS-Дзен» (более подробно о CSS-дзен можно прочитать на с. 82).

Работа с деревом документа

На начальных этапах разработки веб-сайта основной задачей является создание простой HTML-структуры, которую затем можно дополнить с помощью CSS и JavaScript. Работа сводится к созданию структуры, которую документы могут использовать в качестве базы и, возможно, в качестве шаблона. На этом этапе внимание уделяется не контенту, а его структуре. Листинг 2.2 демонстрирует простую структуру HTML-документа.

Листинг 2.2. Простая структура документа HTML.

```
<body>
    <h1>...</h1>
    <div id="main">
        <div id="priorityContent">
            <div id="bodyCopy">
                <h2>...</h2>
                <div class="section">
                    ...
                </div>
                <h2>...</h2>
                <div class="section">
                    ...
                </div>
            </div>
            <div id="sidebar">
                ...
            </div>
```

продолжение ⇩

Листинг 2.2 *(продолжение)*

```
            <ul id="primaryNav">
                ...
            </ul>
            <div id="footer"
                <ul id="secondaryNav">
                    ...
                </ul>
                <p id="colophon">...</p>
            </div>
        </div>
    </div>
</body>
```

Разметка и вложенные элементы в вышеприведенном примере образуют структуру, которая легко надстраивается с помощью CSS и JavaScript. Эти элементы и представляют собой дерево документа, которое можно схематично отобразить следующим образом (# заменяет `id`, а «.» — `class`):

Стандарты, принятые у многих разработчиков, принуждают пользоваться атрибутом `class` в шаблоне сайта гораздо чаще, чем указано в листинге 2.2 и нижеследующем дереве документа.

- `body`
  - `h1`
  - `div#main`
  - `div#priorityContent`
    - `div#bodyCopy`
      - `h2`
      - `div.section`
        - ...
    - `div#sidebar`
      - ...
  - `ul#primaryNav`
  - `div#footer`
    - `ul#secondaryNav`
    - `p#colophon`

Создание подобной схемы с последующей разметкой сильно упрощает работу. Конечно, приведенное здесь дерево документа относительно простое и не содержит никаких элементов контента. Если добавлять в документ контент, это приводит не только к расширению дерева документа, но и к увеличению материалов, которые должны обрабатывать ваши таблицы стилей и скрипты.

Браузеры, парсинг и отображение

Современные браузеры, как правило, анализируют и отображают контент по частям. Причем этот процесс начинается прежде, чем страница полностью загружена в браузер. Браузеры, или, в более широком смысле, пользовательские программные агенты, обрабатывают HTML- и XHTML-документы последовательно, с самого начала, рассматривая связи между различными элементами страницы и при необходимости заполняя пробелы для создания дерева документа. Кроме того, они обрабатывают CSS и соотносят таблицы стилей с элементами страницы (подробнее об этом в следующей главе).

Последовательная обработка страницы важна по следующим трем причинам.

- Пользователь может повлиять на процесс парсинга, только остановив его.

  Разметка, CSS, JavaScript, данные сессии и пользовательские данные, полученные браузером при загрузке отдельной страницы, имеют значение до тех пор, пока страница полностью не отображена.

- Пока страница и связанные с ней элементы полностью не загружены, обработаны и отображены, их внешний вид зависит от механизма визуализации используемого вами браузера.

  Медленная загрузка страницы, особенно в высокоскоростных средах, может привести к некорректному отображению страницы, например контент уже будет подгружен, а обрабатывающие его скрипты — еще нет. Следствием этого может стать появление FOUC (Flash of Unstyled Content) — «вспышки неоформленного содержания». Информацию по этой проблеме вы можете найти на посвященном этой книге сайте (http://www.htmlcssgoodparts.net/).

- Для браузеров не существует строгих правил парсинга, если входные данные могут быть обработаны.

  Браузеры, как правило, не имеют ограничений на скачивание и обработку. Подобная терпимость приводит к тому, что создателям сайтов приходится заботиться о сохранности ресурсов. Наилучший способ достижения этого — добиться точного соответствия таблиц стилей требованиям каждого конкретного документа. И это вызывает беспокойство у тех разработчиков, которые работают в малоэффективгых средах.

Ajax использует для обновления элементов страницы прикладной программный интерфейс Объектная модель документа (DOM API), рекомендуемый Консорциумом Всемирной паутины, поэтому жизненно необходимо, чтобы страница, предназначенная для отображения, имела корректный синтаксис *независимо* от того, является ли она постоянной, или обновляемой.

Синтаксические ошибки в разметке изменяют дерево документа и связи между элементами, поэтому найти причины ошибок JavaScript в Ajax-ориентированном коде очень сложно.

Динамический HTML, Ajax и отображение

Браузеры первого и второго поколения действовали по принципу «один запрос — одно отображение». Таким образом, отображение каких-либо дополнительных элементов было возможно только после отправки на сервер повторного запроса. В последующих поколениях браузеров эта проблема была решена. В них появилась возможность подгружать дополнительный контент после начальной «загрузки страницы». Это новшество получило название «Динамический HTML». Оно пользовалось большой популярностью вплоть до появления API XMLHttpRequest. Этот API позволил совершать асинхронные запросы и вставлять контент, не подгружая при этом новой страницы. Он стал важной частью технологии Asynchronous JavaScript And XML, которую по первым буквам называют AJAX.

3 Обзор CSS

Как и во многих других областях, «путь к совершенству» в понимании CSS становится гораздо более простым, если у вас есть базовое представление о том, что происходит. В этой главе речь пойдет о роли CSS в создании качественных веб-сайтов, а также будет дано краткое описание его основных компонентов.

Те, кто торопится перейти к изучению материала, могут пропустить эту главу, однако позволю себе заметить, что CSS — достаточно сложная тема, и поэтому небольшое повторение может быть полезным.

Связь таблиц стилей с HTML-документом

Для того чтобы применить к HTML-документу таблицу стилей, можно воспользоваться следующими функциями: `<link>`, `<style>` или директивой `import`.

На сайте http://www.htmlcssgoodparts.net/ можно ознакомиться с интерактивной презентацией, демонстрирующей связь между таблицами стилей и основными элементами интернет-страницы.

Связь с помощью тега link

Самым распространенным методом прикрепления таблицы стилей является использование тега `<link>`, вложенного в тег `<head>`. Ниже приведен один из вариантов использования этого тега.

```
<link rel="stylesheet" href="/styles.css" media="screen" title="Primary
Stylesheet" />
```

В данном примере также реализована функция выбора таблиц стилей. Дизайнер может создать несколько таблиц стилей, дать каждой название

с помощью атрибута `title` и добавить всем таблицам, кроме одной, ключевое слово `alternate` в атрибуте `rel`. Теперь пользователи могут самостоятельно выбирать наиболее удобный им стиль отображения страницы. Данная функция поддерживается браузерами Firefox, последними версиями Safari, а также Internet Explorer 8.

Определение версии Internet Explorer с помощью условных комментариев

Неполная поддержка CSS браузерами Internet Explorer создает для разработчиков сайтов дополнительные трудности. В то же время некоторые нестандартные возможности IE позволяют создавать таблицы стилей, предназначенные только для этих браузеров.

В конце 2009 года обновление Windows заблокировало поддержку условных комментариев для Internet Explorer 8 при работе в режиме «IE8 Standards».

Поскольку Internet Explorer определяет и обрабатывает комментарии в HTML не так, как другие браузеры, у разработчиков появляется возможность включить в страницу элементы, которые будет обрабатывать только он.

Вот пример, иллюстрирующий использование условных комментариев для подключения таблиц стилей.

```
<!-- [if lt IE 8]><link rel="stylesheet"
href="/styles.ie.css" media="screen" /><![endif]-->
```

Закрывающий тег `<![endif]-->` является обязательным элементом данной конструкции. Что же касается открывающего тега, то он имеет такой вид (параметр и версию Internet Explorer указывает пользователь):

```
<!-- [if соответствие_версии IE версия]>.
```

Устанавливаемые пользователями значения работают следующим образом:

- `version_constraint`

  Этот параметр необязателен, но если он применяется, то принимает одно из четырех значений:

  - `gt`: больше [>]
  - `gte`: больше или равно [⩾]
  - `lt`: меньше [<]
  - `lte`: меньше или равно [⩽]

- `version`

  Этот параметр также необязателен, но если он используется, то соответствует основным релизам 5, 5.5, 6, 7 или 8.

Также синтаксис условных комментариев поддерживает булевы операторы «И», «ИЛИ», «НЕТ», более подробно о которых можно прочитать на сайте разработчиков Microsoft (http://msdn.microsoft.com/en-us/library/ms537512.aspx).

Еще одним способом определения версии Internet Explorer является низкоуровневая и высокоуровневая фильтрация правил. Более подробно о таких фильтрах рассказывается в главе 14.

Замена тега link тегом style

Тег `style` может включать валидный код CSS.

Наиболее эффективным способом работы с CSS будет вынесение правил CSS в отдельный файл. Я регулярно пользуюсь этим методом при разработке веб-страниц. Многие разработчики используют для ссылки на таблицы стилей директиву `@import` (более подробно о ней см. ниже). Условные комментарии, описанные ранее, также могут содержать в себе стилевые элементы.

Рекомендуется делать содержание элементов `style` максимально коротким, поскольку их наличие влияет на соотношение контента, содержащего ключевые слова, что, в свою очередь, может негативно повлиять на работу поисковых движков.

Если для разметки страниц используется XHTML с корректным MIME-типом (`application/xhtml+xml`), тег `style` должен быть размещен внутри блока `<![CDATA[ ... ]]>`.

Применение @import

Директива `@import` стала популярной в конце 1990-х годов, когда разработчики обнаружили, что браузер Netscape 4 не обрабатывает ее. Это сделало возможным использование более продвинутых стилей, которые поддерживали все браузеры, кроме Netscape 4.

Сейчас @import используется в тех случаях, для которых она была предназначена изначально — в качестве аналога функции `include`, предназначенного специально для таблиц стилей. Корректную с точки зрения синтаксиса директиву `@import` требуется располагать вверху стилевого блока, после директивы `@charset`. Браузер обрабатывает и применяет таблицы стилей, объявленные с помощью `@import` так же, как если бы эти таблицы находились на месте директивы `@import`. Это может повлиять на правила приоритета.

Чтобы соответствовать синтаксическим правилам CSS, ссылка должна заключаться в скобки.

```
@import url(/form_styles.css);
```

Так же, с помощью функции `@import`, можно применять таблицы стилей к специфическим средам, о чем более подробно будет рассказано ниже.

Будьте осторожны с атрибутами style!

Первое правило дружественной для стандартов разработки (см. раздел «Правила дружественной к стандартам разработки» на с. 69) требует, чтобы функции, ответственные за внешний вид сайта, находились вне разметки страницы. Следовательно, рекомендуется при любой возможности избегать применения атрибутов `style`. В тех случаях, когда их использование необходимо (например, в Системах управления содержанием, которые блокируют таблицы стилей), они должны содержать необходимые пары «значение/характеристика», точно такие же, как если бы они были включены в стилевое правило, предназначенное только для этого элемента.

Более подробно атрибуты `style` будут обсуждаться в разделе «Ужасные стороны» на с. 324.

Определение правил для специфических сред

Благодаря HTML и CSS можно создать особые таблицы стилей для различных сред, например: одну для отображения на экране, а другую — для вывода на принтер. Каждая страница может иметь несколько стилей, каждый из которых предназначен для одной или нескольких сред. Существует три способа внедрения директивы `@media`.

- Добавление пары атрибут/значение `media` к соответствующему элементу `link` или `style`.

  При этом валидные правила, относящиеся к этому элементу, будут работать только для выбранной среды. Другими словами, если вы хотите, чтобы таблица стилей применялась только к печатным страницам, вы должны добавить `media="print"` в соответствующий тег `link`.

- Добавление блока `@media` в стилевой блок, которому до этого не было присвоено другого противоречащего значения `media`.

  Например, `@media print { body { font-size: 12pt; } }` сделает размер шрифта на странице для печати равным 12 пунктам.

- Добавление значения `media` в объявление `@import`, в которое ранее не было добавлено другого противоречащего значения `media`.

  Точно так же как селектор `@media` привязывается к именам одной или нескольких сред, файловая ссылка в объявлении `@import` может указывать на таблицу стилей, предназначенную для какой-либо среды, например `@import(/styles.print.css) print;`.

Помните, что нельзя помещать блок `@media` внутрь объявления `@import`.

Если вы назначаете несколько типов сред, то они должны быть указаны через запятую.

Ниже приведен список медиаустройств, описанный в спецификации CSS 2.1. Данный список поддерживается большинством браузеров.

- `all` – все устройства.
- `screen` – мониторы персональных компьютеров, жидкокристаллические или с электронно-лучевой трубкой, а также другие устройства отображения.
- `print` – листы бумаги различных размеров, надписи и рисунки на которые наносятся с помощью чернил, красок или тонера.
- `handheld` – карманные персональные компьютеры и прочие мобильные устройства. В настоящее время поддерживаются лишь частично.
- `projection` – настольные проекторы.
- `speech` – скринридеры, а также платформы text-to-phone.

Перечисленные далее медиа-устройства, описанные в CSS 2.1, функционально не поддерживаются.

- `braille` – тактильные устройства, предназначенные для работы с Интернетом для людей с ограниченным зрением
- `embossed` – принтеры для шрифта Брайля.
- `tty` –двумерные телетайпные дисплеи (обычно монохромные).
- `tv` – телевизионные устройства, браузеры для телевизоров.

Применение селекторов стилей

Типичная таблица стилей представляет собой серию правил, структурированных следующим образом:

```
selector { объект: значение; объект: значение; [ ... ] }.
```

Каждый из селекторов можно назначить любому количеству элементов на странице.

Настоящей головной болью всех разработчиков, особенно тех, которые недавно стали работать с CSS, стала лаконичность структуры каскадных таблиц, подчас доведенная до абсурда. Однако, с другой стороны, подобная краткость упрощает изучение синтаксиса CSS, а овладев этим языком разметки, можно решить практически любую задачу по оформлению внешнего вида страницы.

Более подробно см. «Применение таксономии к таблицам стилей страницы» на с. 93.

Родительские, дочерние и братские элементы: отношения узлов элементов

Раздел «Теги, элементы и атрибуты» на с. 33 рассказывает о вложенных элементах для того, чтобы объяснить, как правильно создавать тег внутри тега. Вложенные элементы являются ключевым аспектом применения HTML и CSS:

Не только можно, но и нужно объединять фрагменты контента в группы с помощью элементов, которые как раз для этого и предназначены, и приписывать этим элементам значения атрибутов `id` и `class`.

При включении в контент этих «семантически выделенных» элементов в дереве документа возникают новые отношения, и вследствие этого увеличивается количество CSS-селекторов, которые можно использовать в качестве дизайнерских решений для данной страницы.

Отношения между множественными вложенными элементами подразделяются на родительские, дочерние и братские.

Ниже приводится краткое описание этих и некоторых других отношений и их субъектов.

- *Дерево документа.*

  Воображаемая разветвленная структура элементов в документе. Понятие дерева документов синонимично понятию объектной модели документа (document object model).

- *Родительский элемент.*

  Элемент, непосредственно содержащий внутри себя рассматриваемый элемент.

- *Предок.*

  Элемент, находящийся на несколько уровней выше в дереве документов и содержащий в себе рассматриваемый элемент.

- *Дочерний элемент.*

  Элемент, содержащийся непосредственно внутри рассматриваемого элемента.

- *Потомок.*

  Элемент, находящийся внутри рассматриваемого элемента на несколько уровней ниже.

- *Братский элемент.*

  Элемент, имеющий общий родительский элемент с рассматриваемым.

В CSS можно четко проследить разницу между отношением родительский/дочерний элемент и отношением предок/потомок. Например, для элемента `li` родительскими элементами будут `ul` или `ol`, а элементов-предков у него будет несколько, например элемент `body`.

Простые селекторы

Как правило, селекторы взаимодействуют с разметкой с помощью названий элементов, значений атрибутов `class` и значений атрибутов `id`:

- элементы

```
p { ... }
```

- атрибуты `class`

```
.about { ... }
```

- атрибуты `id`

```
#corporatehistory { ... }
```

Нижеприведенный пример разметки иллюстрирует применение всех трех типов селекторов:

```
<body class="about" ... >
...
   <div id="corporatehistory" ... >
      ...
         <p>The 1990s were a time of drastic change
         throughout the industry.<p>
      ...
   </div>
...
</body>
```

Селектор `p { ... }` применяется ко всем элементам `p` (параграф) в разметке.

Селектор `.about { ... }` будет применяться ко всем элементам `selector body`, значение параметра `class` для которых равно «about». И наконец, селектор `#corporatehistory { ... }` применяется ко всем элементам `div`, чье значение `id` равно `"corporatehistory"`.

Помимо этих трех типов, в спецификации CSS 2.1 определены следующие типы селекторов: универсальный селектор (`*`), дочерний селектор (`div > p`), селектор потомков (`div p`), смежный селектор (`ol + p`), селектор атрибутов `p[lang]`, `p[lang="en"]`. Также следует упомянуть о селекторах псевдоэлементов, таких как `:first-line`, `:first-letter`, `:before`, `:after`, и псевдоклассов: `:firstchild`, `:link`, `:visited`, `:active`, `:hover`, `:focus` и `:lang`. В спецификации CSS3, еще находящейся в стадии разработки, упоминается большее количество псевдоклассов и псевдоэлементов.

Комбинации селекторов

Возможность объединить несколько селекторов в одном правиле — одно из главных преимуществ CSS. Если несколько селекторов имеют одно и то же значение, то их можно сгруппировать в одну строку и записать через запятую. Селекторы, записанные через пробел, применяются для элементов, являющихся потомками какого-либо элемента. Ограничений на количество и тип селекторов для одного правила CSS не существует, поэтому, соединяя их в цепочки,

опытный разработчик сможет решить практически любую задачу по оформлению своей страницы.

В табл. 3.1 приведены примеры селекторов, взятые с реально существующего сайта.

Таблица 3.1. Обзор селекторов CSS

Селектор	Область применения
`p`	Все параграфы документа
`.about`	Все элементы документа со значением атрибута `class`, равным `about`
`#corporatehistory`	Элементы документа со значением `id`, равным `corporatehistory`
`h1,h2,h3`	Заголовки документа первого, второго и третьего уровней
`.privacy..copyright`	Все элементы, у которых значение `class` равно `privacy` или `copyright`
`#header.#footer`	Все элементы, имеющие значение `id`, равное `footer` и `header`
`p.footnote`	Все параграфы, имеющие значение `class`, равное `footnote`
`#bodycopy.usergenerated`	Элемент, у которого значение `id` равно `bodycopy`, а значение `class` равно `usergenerated`
`.navigation a`	Все элементы, связанные с предком, имеющим значение `class`, равное `navigstion`
`#primarynavigation li.current`	Все элементы списка, имеющие значение `class`, равное `current`, и предка со значением `id`, равным `primarynavigation`
`.about #bodycopy`	Элементы сайта со значением `id`, равным `bodycopy`, и предком, имеющим значение `class`, равное `about`
`body#personalproducts, body#proproducts, body#enterpriseproducts`	Элементы `body`, имеющие значение `id`, равное `personalproducts`, `proproducts`, и `enterpriseproducts`
`body#personalproducts #bodycopy, body#proproducts #bodycopy, body#enterpriseproducts #bodycopy`	Элементы, имеющие значение `id`, равное `bodycopy`, и находящиеся внутри документов из предыдущего примера
`ol li ol li ol li`	Вложенные элементы списка, находящиеся на третьем уровне

Селектор дочерних элементов

В CSS существует возможность создать дочерние селекторы, которые применяются к любому элементу, находящемуся в дочерних отношениях с выбранным элементом. Например:

```
#bodycopy>p { ... }
```

относится к элементам параграфа:

```
<div id="bodycopy"><p>...</p></div>
```

но не относится к элементам:

```
<div id="bodycopy"> ... <blockquote><p> ... </p></blockquote> ...</div>
```

Селектор > не рекомендуется к применению, поскольку не поддерживается Internet Explorer 6.

Конфликты правил, приоритеты и очередность

Каскад позволяет применить селекторы к любому элементу документа, независимо от его положения в дереве документов. Однако они же могут стать причиной конфликта правил. Допустим, у нас есть два селектора: `p { ... }` и `#bodycopy p { ... }`. Какой из них будет обработан браузером?

Приоритет селекторов

Типы селекторов выбираются в зависимости от правила приоритетов. В порядке возрастания этот список выглядит так.

1. Селекторы таблицы стилей браузера.
2. Селекторы пользовательских таблиц стилей.
3. Универсальные селекторы (*).
4. Элементы и псевдоэлементы (например, `first-letter`).
5. Классы, псевдоклассы (например, `:hover`) и атрибуты (`[selected="selected"]`).
6. Атрибуты `id`.
7. Значения `inline` атрибутов в `style`, как описано в разделе «Ужасные стороны» на с. 324.

Из двух правил преимущество получает то, у которого выше приоритет. Если два правила содержат селекторы с одинаковым приоритетом, приоритетным считается то, в котором селекторов больше. Так, правило с двумя селекторами `id` окажется приоритетнее правила с одним `id` или с одним `id` и четырьмя селекторами `class`. В случае, если два селектора имеют одинаковый приоритет, в расчет принимается наличие в одном из них значения `!important`, а также относительное положение в исходной таблице стилей.

Важность порядка элементов в таблице объясняется в следующих параграфах, а также в разделе, посвященном псевдоклассам для организации ссылок (см. раздел «Псевдоклассы ссылок» на с. 164).

Конфликты правил

В случае, когда два конфликтующих правила имеют одинаковый приоритет, браузер применит правило, описанное последним. Порядок описания определяется последовательностью, в которой в документе расположены внешние и внутренние элементы оформления стиля. Рассмотрим следующий фрагмент разметки.

```
<link rel="stylesheet" type="text/css" href="/styles.css" media="all" />
<style type="text/css">@import (/styles.signup.css);</style>
```

Содержание `styles.signup.css` в данном примере является последним по отношению к `styles.css`. Однако если бы в `styles.css` было добавлено объявление `@import`, этот ресурс получил бы приоритет над `styles.signup.css`.

Кроме того, для повышения приоритета можно использовать значение `!important`, а также перегрузку селекторов. Первый способ предпочтителен, а второй рекомендуется применять только в крайнем случае.

Значение `!important` добавляется после пары свойство/значение, например:

```
color: #f00 !important;
```

Значение, снабженное `!important`, получает абсолютный приоритет в любых конфликтах правил, если, конечно, в другое правило также не добавлено `!important`.

В свою очередь, перегрузка селекторами основывается на том, что любой селектор, добавленный к правилу, влияет на его приоритет, даже если элемент, который он определяет, отсутствует в документе.

Рассмотрим следующие два правила, относящиеся к одной и той же таблице стилей:

```
a:visited { color: rgb(128,0,128); }
.sidebar a { color: rgb(0,0,255); }
```

Согласно требованиям спецификации, оба правила имеют одинаковый приоритет, поэтому применено будет то правило, которое описано последним. В нашем случае это правило `.sidebar a { color: rgb(0,0,255); }`. Однако, если необходимо, чтобы сначала было применено первое правило, и вам не хочется его переставлять, можно добавить в него фальшивый селектор, повысив тем самым его приоритет.

Подобный способ крайне нежелателен, поскольку усложняет разметку и увеличивает риск конфликта правил или ошибок обработки. Если же вам все равно пришлось воспользоваться перегрузкой селекторов, настоятельно рекомендуется по возможности снабдить их комментариями, чтобы они были более доступными для функции автоматического поиска и замены.

Наследование значений

Среди всех параметров наследуются только значения шрифта и цвета текста. Что же касается параметров фона (`background`), то технически они не наследуются. Однако благодаря тому, что элементы накладываются друг на друга

и перекрываются, создается иллюзия наследования значений, так как по правилам стека (более подробно см. главу 6) элемент, идущий позже в таблице стилей, автоматически ставится выше своего прямого родителя или предка.

Кроме того, существует два типа элементов, которые не наследуют значения `font\text` и `foreground color` от своих родительских элементов:

- Формы.

  Значения шрифта и цвета определяются исходя из значений, использующихся по умолчанию в операционной системе (если они не были предварительно сброшены в таблице стилей).

- Элементы `iframe`.

  Поскольку `iframe` содержит отдельный документ, параметры шрифта и цвета фона определяются собственной таблицей стилей, связанной непосредственно с этим документом.

Параметры CSS и обзор единиц измерения

Пожалуй, сложнее всего при изучении CSS и при написании таблиц стилей понять, как указать элемент, для которого вы хотите задать оформление. Но эти знания не помогут вам без понимания того, как и какие значения и параметры нужно применять к выбранным вами элементам.

Единицы измерения CSS

Несмотря на то что CSS поддерживает огромное количество параметров, схема присвоения параметрам значений весьма предсказуема. В табл. 3.2 указаны основные единицы измерения, используемые в CSS.

Таблица 3.2. Основные единицы измерения в CSS

Единица измерения	Тип	Пример
px (пикселы)	Длина	`width: 744px;`
em	Длина	`margin-left: 1.25em;`
% (проценты)	Длина	`left: 34%;`
in (дюймы)	Длина	`margin-top: .75in;`
cm (сантиметры)	Длина	`margin-top: 1.905cm;`
`xx-small` ... `xx-large`	Размер шрифта	`font-size: large;`
`rgb(r,g,b)`	Цвет (десятичная система)	`background-color: rgb(221,204,187);`
`#rrggbb`	Цвет (шестнадцатеричная система)	`background-color: #ddccbb;`
`#rgb`	Цвет (шестнадцатеричная система, уменьшенная глубина)	`background-color: #dcb;`

Единицы измерения в переменных средах

Для редактирования всего, что изображено на экране, чаще всего используются три единицы измерения:

- `px` (пикселы)

  Пикселы — относительные единицы измерения, зависящие от параметров дисплея пользователя. Пикселы всегда выражаются целым числом.

- `em`

  В цифровых средах верстки (в том числе и в CSS) em равняется максимально возможной высоте литеры (буквы) в применяемом шрифте. Современное определение несколько отличается от исторического, где em равнялось ширине заглавной буквы «M» в шрифте и кегле, принятом за единицу измерения. Значение часто выражается числом с плавающей запятой.

- `%` (проценты)

  Проценты, как нетрудно догадаться, относятся к относительным единицам измерения. Допускается использование значений с плавающей запятой.

`em` и `%` более подробно описаны в разделе «Типы верстки и сетка» на с. 131.

Зерно и размер пиксела

Несмотря на то, что типы сред, такие как `screen`, `handheld` и `projection`, поддерживают измерение в пикселах, в CSS не представлено никаких механизмов по определению размера зерна экрана. Пиксел является неделимой единицей отображения информации на экране. Поэтому все параметры, имеющие отношение к изображению, должны быть переведены в пикселы. Но благодаря этому изображение становится зависимым от размера зерна экрана: чем он меньше, тем меньше все, что отображается.

В табл. 3.3 показано отношение между стандартными разрешениями мониторов (как современных LCD, так и более старых электронно-лучевых (CRT)) и размером зерна экрана.

В настоящее время два типа мониторов (19″, 1440 × 900 и 22″, 1680 × 1050) – лидеры продаж на Amazon.com (справедливости ради стоит сказать, что обе модели вместе уступают по продажам нетбукам). Судя по таблице, размеры зерна этих двух типов мониторов различаются только на 2 нм (менее чем на 1 %), а разрешение составляет приблизительно 90 пикселов на дюйм.

Если учесть, что Windows допускает разрешение 96 пикселов на дюйм для вывода на экран документов для печати, то на первый взгляд разработчикам не стоит переживать о том, как их творение будет выглядеть на экране обычного пользователя.

Как правило, значения по умолчанию не доставляют хлопот дизайнерам. Однако следует помнить, что на экране нетбука пиксел занимает площадь, на 30 % меньшую, чем на 19- или 22-дюймовом мониторе — лидере продаж. Следовательно, физические размеры каждого слоя уменьшатся на экране нетбука почти на четверть.

Таблица 3.3. Разрешение экрана и размеры зерна

Размер	Разрешение	Соотношение сторон	Ширина	Высота	Размер зерна	Пикселов на дюйм
10.2"	**1024 × 600**	**≈17:10**	**224**	**130**	**0,218**	**116**
12" (CRT)	640 × 480	4:3	244	183	0,381	66
12"	1024 × 768	4:3	244	183	0,238	106
13"	**1280 × 800**	**16:10**	**280**	**175**	**0,219**	**116**
14" (CRT)	800 × 600	4:3	284	213	0,355	71
15"	1024 × 768	4:3	305	229	0,298	85
16" (CRT)	**1024 × 768**	**4:3**	**325**	**245**	**0,318**	**80**
17"	1280 × 1024	5:4	337	270	0,263	96
17"	1366 × 768	16:9	376	212	0,276	92
17"	1440 × 900	16:10	366	229	0,254	100
19"	**1440 × 900**	**16:10**	**409**	**256**	**0,284**	**89**
22"	1680 × 1050	16:10	474	296	0,282	90
23" (CRT)	1600 × 1200	4:3	447	335	0,279	91

Каждый дизайнер должен помнить об относительности измерения в пикселах и учитывать это при создании дизайна веб-страницы. Иначе он будет шокирован при виде своего детища, запущенного при неучтенных ими настройках монитора.

Единицы измерения для печатных форм

В главе 1 нашей книги отмечалось, что во Всемирной паутине, в отличие от ее предшественников, практически отсутствуют границы. Такая неограниченность относится не только к информации, но и к интерфейсам. Большинство аппаратных платформ позволяет уменьшать и увеличивать текст, прокручивать его и подвергать другим манипуляциям в окне браузера.

Если же речь идет о распечатке страницы, то здесь ограничения весьма заметны, поскольку большинство страниц распечатывается на бумаге формата Letter (8,5"×11"), использующегося в США, или аналогичного по размеру формата ISO A4.

Для таблицы стилей, предназначенной для распечатки документов, рекомендуется использовать дополнительные единицы измерения, которые могут задаваться с использованием плавающей запятой.

- pt (точка).

  Точка – это традиционная единица измерения размера шрифта. Она равна 1/72", или приблизительно 353 нм, если перевести в систему СИ; кроме того, в одном сантиметре помещается чуть меньше 28,35 точки.

- in (дюйм).

  Дюйм, согласно международным стандартам, примерно равен 2,54 см. Для сравнения: ширина четырех простых карандашей, уложенных рядом, равна примерно 1,125 дюйма.

- `cm` (сантиметры).

  Размер листа A4 составляет 21 × 29,7 см.

Более полный список дополнительных единиц измерения можно найти на сопутствующем сайте книги. Здесь также следует упомянуть о параметре `line-height`, значение которого можно задать с помощью числа с плавающей запятой, или же вообще не используя единицы измерения. В этом случае значение `1` будет эквивалентно `1em`.

Размер шрифта. Ключевые слова

Ключевые слова удобно применять в том случае, когда эстетика дизайна приносится в жертву доступности страницы для большинства браузеров и, как следствие, пользователей. Область определения ключевых слов включает семь значений, приведенных ниже в порядке убывания:

- `xx-large`
- `x-large`
- `large`
- `medium` (по умолчанию)
- `small`
- `x-small`
- `xx-small`

Ключевые слова для `font-size` также обсуждаются в разделе «Ключевые слова для размеров шрифтов» на с. 248.

Цвет. Единицы измерения

CSS поддерживает трехканальное (red, green, blue) цветовое пространство, с глубиной цвета 8 бит на канал. Таким образом, можно отобразить 16,7 миллионов (224) цветов.

Дизайнер может задать необходимый ему цвет тремя способами:

- `rgb(r,g,b)`

  Три канала, десятичная система; каждый канал принимает значения в диапазоне от `0` до `255`.

- `#rrggbb`

  Три канала, шестнадцатеричная система; каждый канал принимает значения в диапазоне от `00` до `ff`.

- `#rgb`

  Три канала, шестнадцатеричная система, уменьшенная глубина цвета; каждый канал принимает значение в диапазоне от `0` до `f`. Эквивалентный цвет в полноцветной палитре получается благодаря удвоению каждой шестнадцатеричной цифры. Таким образом, цвета `#6cf` и `#66ccff` идентичны.

При создании таблицы стиля важно выбрать наиболее подходящую классификацию цветов и пользоваться только ей для сохранения стилистического единства. Достоинства и недостатки классификаций приведены в табл. 3.4.

Таблица 3.4. Преимущества и недостатки трех способов назначения цветов в CSS

Стиль	Преимущества	Недостатки
Шестнадцатеричный, шестизначный	Точность, простота в наследовании	Сложность визуализации, трудность в прочтении
24-битный, десятичный	Удобство в прочтении. Простота обработки скриптами	Чувствительность к ошибкам ввода
Шестнадцатеричный, трехзначный	Простота. Удобство использования при создании макетов	Недостаток глубины цвета

Основные свойства визуализации в CSS

Чтобы отображение страницы на экране соответствовало замыслам дизайнера, ему придется применить множество свойств, изменяющих положение элементов в документе. Наиболее удобные и часто используемые функции приведены в табл. 3.5. Более подробно свойства и их значения описаны в главе 6.

Таблица 3.5. Функции визуализации

Параметр	Значение
Display	block
	inline
	inline-block
	none
width/height	[length]
	auto
Float	left
	none
	right
clear	both
	left
	none
	right
position	absolute
	fixed
	relative
	static
top/right/bottom/left	[length]

Функции параметров, приведенных в табл. 3.5, описаны ниже, а также показаны на рис. 3.1.

- `display`

  HTML подразумевает, что элемент может отображаться на экране одним из нескольких способов. Обычно положение элемента задается описанием в DTD, но оно может быть переопределено другими параметрами `display` в CSS. Применение параметра `inline` описывает элемент как строчный и приводит к его изображению без разрывов и переносов строки. К строчным элементам нельзя применить некоторые свойства CSS, предназначенные только для блочных элементов. `block` описывает элемент, который начинается и заканчивается разрывом строки и по умолчанию занимает всю ширину родительского контейнера. `inline-block` описывает строчные элементы, подобные `inline`, которые, однако, поддерживают все свойства CSS и свойства блоков. Значение `none` временно удаляет элемент из документа, а веб-страница формируется так, словно элемента на ней нет.

- `width` и `height`

  Описывают размеры блочных и строчных элементов Значение `auto` для параметра `width` означает, что элемент будет занимать всю ширину родительского контейнера, в то время как значение `auto` для `height` приведет к тому, что контент будет стремиться занять максимально возможную высоту, независимо от размера блока.

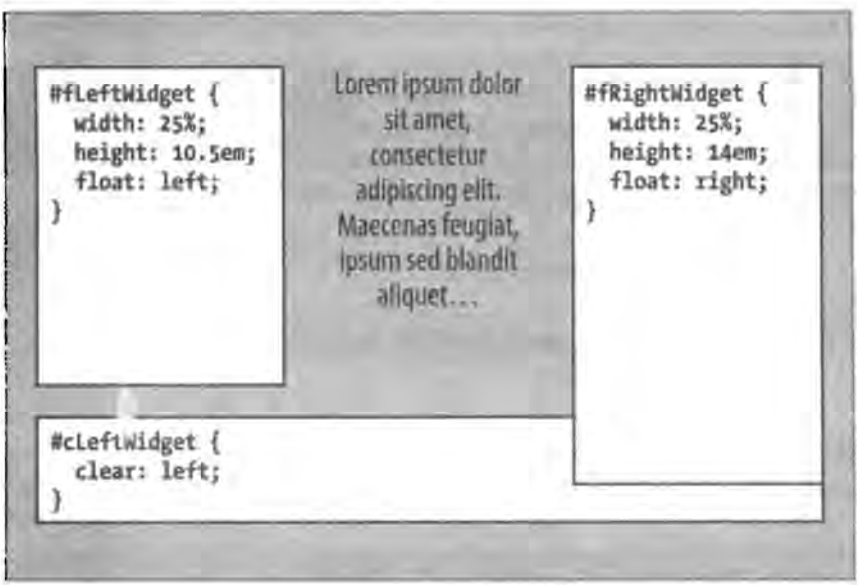


Рис. 3.1. Применение свойств `float` и `clear` к одному элементу изменяет отношения между этим элементом и последующими

- `float`

  Присвоение этого значения определяет особые границы элемента, в то время как другие элементы обтекают его слева или справа. Параметр `float` рекомендуется всегда использовать в паре с `width`, кроме случаев, когда элемент (например, изображение) имеет четко заданную ширину.

- `clear`

  Описывает, с какой стороны элемента запрещено его обтекание и располагает данный элемент ниже ближайшего элемента, имеющего параметр `float`.

- `position, top, right, bottom, left`

  Изменяют положение выбранного элемента, кроме случаев, когда ему присвоено значение параметра `position`, равное `static`. Данные параметры являются, пожалуй, наиболее эффективными для позиционирования контента. Более подробно они описаны в параграфе «Свойства позиционирования в CSS» на с. 121.

На рис. 3.2 показаны некоторые примеры применения этих параметров.

Описанные нами параметры составляют лишь малую часть из обширного арсенала CSS, о которых рассказывается подробно в следующих частях нашей книги.

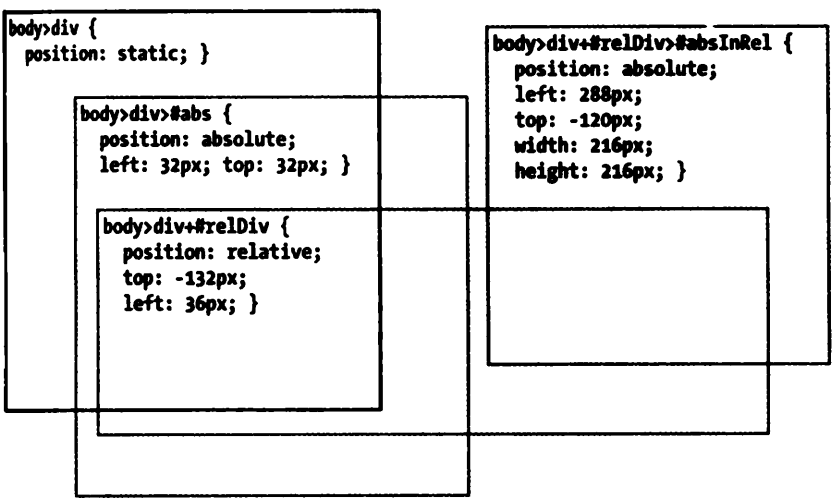


Рис. 3.2. Изменение положения контента в зависимости от изменения параметров `position` (кроме значения `static`)

Разработка в соответствии со стандартами

Всемирная паутина на практике очень сильно отличается от своих спецификаций. Несмотря на то, что в теории Интернет выстроен на стандартах и спецификациях, создатели программ обращают внимание на эти правила лишь эпизодически. Разработчики программного обеспечения меняют функциональные свойства или забывают о них, внедряют программное обеспечение некорректно. Создатели сайтов до некоторой степени стараются, чтобы их «творения» соответствовали спецификациям, но вместе с тем они должны добиваться того, чтобы их сайт нормально отображался на различных браузерах, а это часто идет вразрез со стандартами. Некоторые недобросовестные разработчики просто оставляют свои «творения» на милость рендеринговых движков браузеров.

Игнорирование или отказ от стандартов в пользу поддержки всеми браузерами может в итоге привести к постоянным проверкам работоспособности и корректности отображения страницы или даже к прекращению всяческих попыток по усовершенствованию сайта. С другой стороны, слепое следование духу и букве спецификаций может серьезно ограничить набор приемов и инструментов, с помощью которых вы делаете свой сайт привлекательным и доступным для пользователей. В данной ситуации мы рекомендуем вам держаться золотой середины.

Обзор стандартов Всемирной паутины

Помимо HTML и CSS — «главных героев» этой книги — существует множество стандартов, имеющих отношение к развитию Всемирной паутины. Ниже описываются некоторые из них.

- HTTP 1.x.

  Протокол передачи гипертекста (Hypertext Transfer Protocol), о котором было вкратце рассказано в предыдущих главах, поддерживается Специальной комиссией интернет-разработок (IETF) и Консорциумом Всемирной

паутины (W3C). Подобная «двойная опека» вполне оправдана, ведь IETF отвечает за разработку Интернет-протоколов в целом.

- Web Content Accessibility Guidelines (WCAG).

  На сегодняшний день были изданы две версии Рекомендаций по доступности содержимого веб-ресурсов. Первая версия появилась в виде рекомендаций Консорциума Всемирной паутины еще в мае 1999 года и была прямым следствием постановления правительства США. Данное постановление устанавливало требования к сайтам, предназначенным для людей с ограниченными возможностями. Последующая версия Рекомендаций появилась в декабре 2008 года. Она предназначалась не только для интернет-сайтов, но для веб-платформ в целом.

- ECMA-262.

  Данный стандарт определяет синтаксис, грамматику и основные объекты языка, известного под названием JavaScript. За аббревиатурой ECMA скрывается Европейская Ассоциация производителей компьютеров (European Computer Manufacturers Association).

- Document Object Model (DOM). Уровни 1–3.

  Объектная модель документа (DOM) представляет собой программный интерфейс для работы с веб-документами. С помощью него любой документ может быть представлен в виде древовидной структуры узлов. Попытки создания подобного интерфейса начались с момента первого применения Динамического HTML. Те варианты, которые появились до спецификаций Консорциума Всемирной паутины, принято называть Уровнем 0. Спецификации уровней 1–3 включены в Рекомендации W3C.

- ISO 639, 8859, и 10646.

  Главное назначение этих стандартов — корректное отображение знаков и символов. Стандарт 10646 также известен как Юникод. Для его обработки применяются различные вариации формата преобразования Юникода (UTF). Нелогографическое письмо часто кодируется с помощью UTF-8. Более подробно о кодировании символов рассказывается в разделе «Кратко о кодировке» на с. 255.

Также необходимо упомянуть о том, что Всемирная паутина опирается на стандарты, которые определяют формат изображений (JPEG, GIF, PNG, SVG). Впрочем, для любого контента, который можно включить в веб-страницу, существуют свои стандарты.

Для чего нужны стандарты?

С момента публикации Специальной комиссией интернет-разработок (IETF) первых спецификаций для HTML разработчики браузеров и создатели сайтов постоянно стараются эти спецификации игнорировать. В то же время

сообщество разработчиков, соблюдающих стандарты (к коим себя относит и автор этой книги), ограничивается только дебатами, так как фактически не может повлиять на своих менее дисциплинированных коллег. В последующих разделах мы постараемся рассказать об основных преимуществах и недостатках соблюдения стандартов.

Интероперабельность

Интероперабельность, то есть способность взаимодействовать с различными платформами, позволит создателям сайтов уменьшить количество различных проверок на совместимость и, как следствие, упростит работу над созданием сайта и уменьшит количество таблиц стилей, предназначенных только для одного вида браузеров. Добиться подобных результатов будет гораздо проще, если все разработчики будут следовать единым стандартам.

Рыночные силы

Все прелести межплатформенной совместимости не очень согласовываются с желанием создателей сайтов сделать свои страницы привлекательными для пользователей и задействовать как можно больше различных «примочек». С другой стороны, разработчики браузеров, чтобы сохранять конкурентоспособность своих детищ, должны постоянно их совершенствовать, а жизненные циклы на рынке гораздо короче, чем процессы утверждения тех или иных стандартов.

Законы рынка — вот основной фактор, препятствующий всеобщему переходу к единым стандартам. В начале 1995 года в Netscape 1.1 была введена поддержка элемента `table`, в то время как улучшения, представленные Mosaic 2.0, вышедшего на рынок раньше, все еще ожидали утверждения. В результате Netscape, несмотря на статус бета-версии, триумфально ворвался на рынок и оказался далеко впереди своих конкурентов как раз благодаря тому, что не тратил времени на согласование со стандартами. Результатом подобных действий стало двойственное отношение к стандартам, которое наблюдается и по сей день, почти пятнадцать лет спустя.

Совместимость снизу вверх

Считается, что следование стандартам гарантирует сайтам «долгую жизнь», поскольку различные свойства, поддерживаемые пользовательскими платформами, очень редко удаляются или отменяются (элемент `blink` исключение, только подтверждающее правило). С другой стороны, старые версии стандартов являются «наименьшими общими знаменателями» для свойств, поддерживаемых всеми платформами. Таким образом, следование стандартам дает сайтам шанс «остаться на плаву» при переходе на новую версию браузера.

Доступность

Следование стандартам делает материалы сайтов более доступными для пользователей с ограниченными возможностями, многие из которых используют для работы в Интернете стороннее оборудование и программное обеспечение. Для производителей подобной техники и программ стандарты полезны по целому ряду причин.

- Рекомендации W3C хоть и не обязательны для выполнения, но, тем не менее, заслуживают доверия в некоторых случаях (например, в США они включены в законы о доступности сетевых ресурсов).
- Рекомендации позволяют производителям специальных технологий предсказать, какие браузеры будут использовать покупатели. Это весьма полезно, даже если эта оценка основана на «наименьших общих знаменателях».
- Одной из основных целей HTML с момента его появления была межсредовая совместимость, упрощающая создание новых интернет-технологий.

Приоритеты поставщиков

После выпуска шестой версии Internet Explorer в 2001 году Microsoft практически перестала уделять внимание как своему браузеру, так и его пользователям. Подобная эпоха застоя закончилась лишь совсем недавно, с появлением Internet Explorer 8.

Выход IE6 пришелся на то время, когда другие игроки на рынке браузеров переживали упадок, и это дало возможность Microsoft «почивать на лаврах» до тех пор, пока сообщества разработчиков, следуя рекомендациям правительства США, фактически отвергли Internet Explorer. Этот, а также ряд других факторов, заставило компанию Microsoft возобновить масштабные разработки в области браузеров.

Похожая история связана с недостаточно полной поддержкой CSS в браузере Netscape 4. Когда спецификация CSS 1.0 была еще в разработке, Microsoft и Netscape направили W3C свои предложения, и предложение Netscape было полностью отвергнуто. Поскольку четвертая версия Netscape была уже практически готова, поспешные попытки переделать движок рендеринга с целью внедрения хотя бы некоторых из одобренных Консорциумом Всемирной паутины элементов CSS оказались неудачными, что в конечном итоге отразилось как на качестве этого некогда передового браузера, так и на его жизнеспособности.

Косность стандартов

В эпоху быстрого роста Всемирной паутины стандарты только начинали формироваться, а стоимость производства сайтов была очень высокой из-за несовершенства инструментов для их разработки. В результате огромные средства оказались вложенными в веб-сайты и программное обеспечение сомнительного качества. И подобная ситуация продолжает сохраняться, поскольку замены

обойдутся еще дороже. В результате страдают в основном независимые производители, например, разработчики новостных и рекламных платформ.

Факторы успеха (и их отсутствие)

В настоящий момент на рынке можно найти разработчиков сайтов всех мастей и размеров: фрилансеры-одиночки, магазины, рекламные агентства, специальные подразделения компаний, насчитывающие от одного-двух до нескольких десятков человек и даже отдельные мастера-самоучки, делающие сайты для своего удовольствия. Веб-сайты делают разные люди и с разными целями, и каждый человек имеет свое мнение по поводу того, какой веб-сайт считать хорошим, а какой — плохим. Подобная разница во мнениях о качестве сайтов и об инструментах для достижения этого качества приводит к спорам по вопросу о том, что считать основными факторами успеха. Для некоторых это хорошее образование и квалификация, для других — талант и креативность, для третьих — умение взаимодействовать с другими разработчиками, работать в коллективе или, как принято говорить, командный дух. Последнее качество кажется наиболее важным, особенно если учесть, что среди компьютерщиков в целом и веб-дизайнеров в частности насчитывается огромное количество интровертов, а нежелание многих работодателей брать на себя ответственность за постоянное обучение и совершенствование коммуникативных навыков своих сотрудников может в конечном итоге привести к полному развалу команды.

Таким образом, вопрос об определяющем факторе или комбинации факторов успеха остается открытым. Однако совершенно ясно, что любой фактор может оказать как положительное, так и отрицательное воздействие.

Жесткий конструктивизм

Самые жаркие споры по поводу соответствия стандартам вызваны тем, что здесь действует принцип «все или ничего». Наиболее значимым требованием соответствия стандартам является валидная разметка, которой, по причинам, обозначенным выше, практически невозможно следовать. Кроме того, соблюдающим стандарты профессионалам приходится помимо жестких требований мириться с довольно мощной группой противников стандартизации. В результате большая часть разработчиков совершенно не заботится о соблюдении стандартов.

Третий путь — дружественность к стандартам

Продукт разработки может содержать в себе полезные свойства, заложенные стандартами, даже если не полностью прошел тест на валидность. Главная цель дружественности к стандартам — это возможность постоянного совершенствования

продукта, а это, в свою очередь, важнейший аспект вертикальной совместимости. Дружественность к стандартам ведет к синтаксически корректной разметке «исходников» страницы, принуждает к отделению от структуры таблицы стилей, что, в свою очередь, приводит к уменьшению временных затрат, необходимых для анализа даже самых мельчайших деталей разметки. И хотя подобная практика допускает некоторые послабления, по сравнению с жесткими условиями, на которых настаивают ревнители стандартов, она может принести разработчикам гораздо больше пользы и, как следствие, прибыли.

Преимущества дружественной к стандартам разработки

В чем же заключаются преимущества дружественной к стандартам разработки?

- При такой разработке соблюдается правило Парето (80/20), помогающее правильно распределять время.

  Это позволяет избежать неконтролируемых расходов, делает проект более гибким. Сроки разработки существенно сокращаются.

- Повышается доступность ресурса для людей с ограниченными возможностями.

  Если контент правильно организован и прост для изучения, а разработчики используют надлежащие элементы, особенно списки, для упрощения навигации, то страница становится более удобной для людей с ограниченными возможностями.

- Повышение доступности сайта облегчает использование альтернативных сред.

  Дружественная к стандартам разработка упрощает возможность презентации контента в разных средах (экран, принтер, мобильные устройства и т. д.) без необходимости дублирования информации для каждой среды; экономит массу времени и других ресурсов.

- Информация становится более «портативной».

  Когда хранилища информации создаются в духе дружелюбности к стандартам, их содержимое гораздо проще переориентировать для использования в других информационных системах, даже если это не Интернет-системы. Этот аргумент особенно убедительно звучит для сторонников микроформатов.

Правила дружественной к стандартам разработки

Следование этим простым правилам разработки, дружественной к стандартам, упростит вам создание страницы и подарит сайту «долгую жизнь».

1. При любой возможности старайтесь избегать использования разметки, ориентированной на внешний вид сайта (например, встроенные атрибуты `style`).

2. Контент должен быть максимально удобочитаемым.
3. Не перегружайте страницу элементами разметки, всегда старайтесь свести разметку к минимуму.
4. Используйте таблицы стилей только для оформления страницы.
5. Убедитесь, что все элементы, независимо от их приоритета, надлежащим образом закрыты, вложены, заполнены и снабжены необходимыми атрибутами и значениями, даже если правилами разметки разрешается пропустить какие-либо элементы.
6. Старайтесь по возможности присваивать каждому элементу `body` отдельное значение `class` или `id`.
7. Присваивайте атрибутам `id` и `class` имена, обусловленные контекстом, а не внешним видом элементов. Если вы не можете этого сделать, значит, велика вероятность, что вы нарушаете правило 1. Если же вы соблюдаете данное правило, то вам будет проще соблюсти правила 2 и 3.
8. Всегда используйте заголовки и списки там, где это необходимо, даже если при этом вторые окажутся внутри первых.
9. Старайтесь подключать все изображения с помощью функции CSS `background-image`.
10. Для того чтобы предотвратить появление ошибок при отображении слоев, создавайте отдельные таблицы стилей для Internet Explorer.

Приведенные выше правила, впрочем, как и большинство правил, имеют свои исключения, например правило № 5 противоречит некоторым правилам полностью валидной разметки.

При следовании данным правилам следует опираться скорее на логику и соображения практичности, ведь зачастую очень сложно полностью контролировать весь сайт, а соблюдение правил даст возможность свести «проблемные места» к минимуму.

5 Создание эффективных стилей и структуры

Кто-то может читать разметку страницы, как ноты, и четко представлять конечный продукт. Большинству же людей для представления конечного результата приходится мыслить как бы параллельными потоками, особенно если таблицы стилей отделены от страницы. Многие разработчики применяют в своей повседневной работе именно такое параллельное мышление, каким бы оно ни было сложным.

Однако существует ряд методов и инструментов, упрощающих этот процесс. Среди инструментов, прежде всего, стоит выделить интерфейсы для разработки, существующие практически в каждом браузере. Эти интерфейсы отображают разметку и правила таблиц стилей *в контексте*.

Правила – это совсем другое дело. Следует всегда помнить, что Интернет – это многомерное пространство, поэтому если вы видите разметку или таблицы стилей, а ваш разум анализирует их с точки зрения двух (иногда даже трех) измерений, то *вы никогда не увидите всей полноты картины*.

Большая часть этого раздела содержит теоретическое описание методов разработки уровня внешнего оформления. Если вы хотите сразу перейти к самому интересному материалу, прочитайте раздел «Таксономия и номенклатура» на с. 92.

Четыре правила для эффективного создания стиля

Интернет, как Вселенная, не имеет ни начала, ни конца. Отдельные страницы могут иметь какие-либо границы, но чаще всего и они содержат ссылки на другие информационные ресурсы, куда (и откуда) могут перейти пользователи, а интернет-сеанс не заканчивается с последней буквой документа.

Работа в этой практически не имеющей ограничений среде требует особого внимания к *информации*. Все остальное имеет гораздо меньший приоритет: проблемы дизайнеров, брэндинг и даже создание графиков работы не будут иметь никакого значения на начальной стадии проекта. Разработчик стилей, который сосредоточен на информации, способен соблюдать приоритеты своей команды.

Успешный разработчик стилей должен следовать четырем правилам, приведенным ниже.

1. Будь проще.
2. Будь гибким.
3. Будь последовательным.
4. Придерживайся своего курса.

Эти правила упрощают как продукт, так и сам процесс работы и позволяют получить хороший результат, затратив меньше усилий. Это особенно полезно и актуально для создателей больших по объему сайтов, при работе над которыми вопрос об экономии времени и ресурсов стоит особенно остро.

Соблюдение этих правил в конечном итоге приводит к концепции «CSS-дзен» (более подробно об этом рассказывается в одноименном разделе на с. 82).

Правило № 1. Будь проще

Правило простоты дизайна, также известное как принцип KISS («Keep It Simple, Stupid», «будь проще, глупец») может принимать разные формы при работе над сайтом. Среди них: меньшее количество элементов разметки, более короткие запросы, сокращение количества нефункциональных элементов декора и т. д.

Чем больше элементов будет на странице, тем больше будет связей и взаимодействия между элементами, тем больше шансов, что что-нибудь будет работать неправильно и, наконец, больше вещей, которые могут сломаться.

Сторонники простоты дизайна удаляют со страницы все, что не является там необходимым.

И вот здесь мы сталкиваемся с первой проблемой: как определить критерии необходимости? Как правило, избыточная разметка используется либо для создания задела на будущее, либо для создания более аккуратного макета. Во втором случае избыточная разметка оправдана только тогда, когда ваша работа ориентирована на совсем невежественных пользователей. Это также относится к необязательным, но желательным элементам декора, таким, как закругленные углы, простые и сложные примеры реализации которых показаны ниже.

```
/* *** simple styles *** */

.someElementWithCorners { padding: 1em; background-color: #ccc;
-moz-border-radius:
.75em; -webkit-border-radius: .75em; }
.someElementWithCorners b { display: none; }

/* ***  ...but if EVERYBODY MUST have the same presentation,
        these rules will go in
        the IE conditional stylesheets... *** */

.someElementWithCorners { position: relative; }

.someElementWithCorners b.trc,
.someElementWithCorners b.brc,
.someElementWithCorners b.blc,
.someElementWithCorners b.tlc { display: block; position:
        absolute; z-index: 0; width:.75em; height: .75em;
        overflow: hidden; background-image:
        url(/images/corner_circle.png); }

.someElementWithCorners b.trc
        { top: 0; right: 0; background-position: 100% 0; }
.someElementWithCorners b.brc
        { bottom: 0; right: 0; background-position: 100% 100%; }
.someElementWithCorners b.blc
        { bottom: 0; left: 0; background-position: 0 100%; }
.someElementWithCorners b.tlc { top: 0; left: 0; }

.someElementWithCorners p,
.someElementWithCorners h4 { position: relative; z-index: 1; }

...

<!-- ...And now for the markup: -->

<div class="someElementWithCorners">
  <h4>Lorem Ipsum Dolor Sit Amet</h4>
  <p>This is filler content for a source example.
      In it the quick red fox jumps over
      the lazy brown dog.</p><b class="trc"> </b>
      <b class="brc">  </b><b class="blc"> </b>
      <b class="tlc"> </b>
</div>
```

В простом коде используется лишь первое правило таблицы стилей, четыре элемента `b` опускаются, нет необходимости в фоновых изображениях, а поскольку Internet Explorer не поддерживает закругление углов, пользователи этого браузера их и не увидят. (Они отображаются с помощью нестандартных функций `-webkit-border-radius` и `-moz-border-radius`.)

Более сложные способы добавления закругленных углов также работают, но при этом возникает ряд проблем.

- Разметка окончательно привязывается к внешнему виду; удаление закругленных углов из дизайна означает работу по удалению элементов, относящихся только к внешнему виду.
- Элементы `position`, `z-index` и др., используемые в данной разметке исключительно для отображения закругленных углов, занимают практически столько же места, сколько основная разметка. Результат всегда один и тот же: оформление сайта становится менее гибким.
- Дополнительная разметка и стили повышают вероятность ошибки при вводе данных или при отображении.

Занимаясь вопросами простоты (или ее отсутствия), всегда следует помнить, что и преимущества и проблемы имеют тенденцию многократно разрастаться. Если вы в своих шаблонах используете единую систему внедрения, то, возможно, проблем удастся избежать. Но при усложнении шаблоны начинают конфликтовать друг с другом, увеличивается количество ошибок отображения, возрастает необходимость в техническом обслуживании.

Некоторые вместо упрощения применяют как для новых проектов, так и для переработки старых, принцип под названием «CSS-дзен». Для этой дизайнерской философии характерно сведение всего к необходимому минимуму. Таким образом, самой важной задачей любого упрощения становится понимание того, какие элементы действительно *необходимы* для вашего проекта

Упрощение и большие сайты

Сущность простоты может меняться, особенно в зависимости от размера проектов. Командам разработчиков, ответственным за большие сайты, бывает порой очень трудно поддерживать огромное количество таблиц стилей, правил и т. п. Кроме того, для того чтобы обеспечить легкое управление сайтом, разработчики вынуждены значительно снижать количество элементов `id`, а оставшиеся подвергать жесткой стандартизации.

На практике, дизайнеры — единственные, кто проводит подобную «чистку» с одобрения разработчиков. Делается это для достижения трех основных целей: коммерции, удовлетворения ожиданий посетителей страницы и корректировки сроков разработки.

Результатом подобного сотрудничества, как правило, становится создание небольшого количества шаблонов, основанных в свою очередь на одном очень простом шаблоне, который можно назвать каркасным. Макет разделен на несколько отдельных компонентов, каждый из которых помещен в контейнер с присвоенным значением `id` и одним или несколькими значениями `class`, например:

- `#header`
- `ul#nav`

- #content
- #footer

Компоненты, вставляемые в страницу, заполняются автоматически. Образующийся при этом выбор контента имеет своей целью определение статуса пользователя, а также необходимость использования независимого контента.

Разметка страниц больших сайтов отличается от примеров, приведенных в этой книге, особенно в отношении навигации, которая в силу необходимости очень сильно упрощена. Например, на больших сайтах почти нельзя встретить смещение изображений: поскольку подобные сайты содержат очень много эксплуатационных требований, цена внедрения этой технологии недопустимо велика.

Подобные схемы дизайна используют также блог-платформы и Системы управления содержимым (CMS). Широкий спектр сценариев их использования вызван не нуждами посетителей сайтов, сконструированных на этих платформах, а скорее множеством целей, которые стремятся достичь их разработчики.

Правило № 2. Будь гибким

Гибкость страницы подразумевает правильный баланс между краткосрочными и долгосрочными нуждами. Данное правило скорее относится к людям, а не к способам разметки. Каждый проект в идеале составляет уникальную комбинацию организаторов, аудитории, требований, временных рамок, а также ожиданий по поводу срока существования этого проекта.

Понятие гибкости также меняется в зависимости от области применения. Гибкость в веб-приложении — совсем не то, что гибкость в архивах данных. А если мы будем рассматривать сайт, созданный в поддержку какого-либо события, то понятие гибкости для него будет кардинально отличаться от двух предыдущих примеров. Если в первых примерах гибкость касается организации многократных запросов и взаимодействия с пользователем, то для «скоропортящихся» сайтов гибкость заключается в возможности использования его ресурсов (включая разметку) для последующих проектов.

Наиболее важными факторами, для которых необходима гибкость, являются процесс создания сайта, цели сайта и культура труда. Успешные дизайнеры обязательно принимают это во внимание, прежде чем приступают к работе.

Продукт работы дизайнера будет тем гибче, чем точнее его значимость отражает настоящие причины того, почему он поставлен на первое место.

Применение гибкости на практике основывается на двух китах: прогрессивном улучшении и надстройке.

Эффективное применение прогрессивного улучшения в веб-приложениях очень часто требует того, чтобы разработчики параллельно писали два приложения: одно находилось на клиентской станции, чтобы уменьшить загрузку сервера, а другое запускалось на сервере для проверки сценариев, при которых пользователь выводит из строя клиентские скрипты.

Понятие *прогрессивное улучшение* неявно описано в разделе «Разделение контента, структуры, презентации и интерфейса» на с. 42. Контент находится внутри разметки, поверх которой расположена презентационная оболочка, основанная на CSS, которая в свою очередь «покрыта» оболочкой из скриптов, ответственных за обработку ошибок, интерактивность и т. п. Если эта многослойная структура внедрена корректно, то все слои находятся в однонаправленной зависимости — то есть внедрение презентационной оболочки или интерфейса повлияет на контент и разметку, но если эти оболочки убрать, то сайт останется пригодным для использования.

Сложность создания гибкого продукта на основе прогессивного улучшения повышают некоторые укоренившиеся привычки, а также конструкционные недостатки Интернета. Более подробно об этом можно прочесть в главе 14.

Вторая опора, на которую опирается гибкость, это так называемая надстройка. Существуют случаи, когда разметка может вставляться в шаблон, для того чтобы упростить применение таких вещей, как правила, основанные на элементе `class`, или абсолютное позиционирование. Необходимость в подобной разметке может возникнуть, если вы создаете ресурсы многократного использования или какие-либо долговременные проекты.

По нашим данным, значения `class` (такие как `section` и `postMetadata`) встречаются в Интернете очень часто, даже на таких страницах, где требования к дизайну позволяют их опустить. В основном их внедряют либо с эстетическими целями, либо для структурного развития сайта. Начальные условия могут не требовать присутствия этих элементов, но частенько случается, что подобные требования появляются по ходу работы над проектом.

Оборотной стороной надстройки является то, что подчас она прямо противоречит упрощению. Таким образом, в задачу дизайнера входит нахождение оптимального баланса между этими противоречащими друг другу приемами. Для этого необходимо ответить на три вопроса:

- Можно ли добавить в каждый из дополнительных элементов надлежащие значения `class` или `id`?
- Дополнительная разметка призвана обеспечивать *несколько* элементов презентации или только один?
- Есть ли вероятность, что сайт будет подвергаться постепенным изменениям с течением времени?

Если ответ на все три вопроса — «Да», то внедрение дополнительной разметки в ваш шаблон не только возможно, но и желательно. Однако даже в такой ситуации внедрение стоит проводить очень внимательно и осторожно, чтобы добавочные элементы не начали конфликтовать с уже существующими.

Гибкость, внутренние библиотеки и повторное использование кода

Когда разработчики начинают планировать долгосрочные проекты, создание гибких шаблонов, как мы уже писали выше, позволяет им сэкономить значительное количество ресурсов. Команды, работающие без существенной внешней поддержки, могут попытаться сэкономить за счет освоения готовых систем, с последующей модификацией их внутренних процессов, с целью улучшения производительности.

Фрилансеры, занимающиеся разработками долгосрочных проектов, как правило, придерживаются другой практики. Они создают уникальные библиотеки стилей, разметок и кодов. Со временем у каждого дизайнера накапливается большое количество подобных библиотек на все случаи жизни.

Еще одна группа, к которой принадлежит и автор этой книги, предпочитает пользоваться не очень распространенной, но весьма эффективной техникой, при которой каждый проект начинается (если такое возможно) с чистого листа, а процесс создания прототипа (см. раздел «Создание прототипа и макета» на с. 285) производится вручную, но с такими незначительными умственными усилиями, что ничего не мешает размышлять над условиями и требованиями к проекту.

Разработчики, занятые созданием больших сайтов и имеющие внешнюю поддержку, находятся в наиболее выигрышном положении. Они могут создать новые «модули», если в проект будут внесены новые требования, а вместе с тем перепрофилировать уже существующие продукты.

Однако внутренние стандарты продуктов и руководства по созданию стилей не одобряют подобного подхода. Если разработчик стилей вынужден соблюдать эти стандарты, то они влияют на его работу следующим образом:

- Большая часть поддержки стилей выполняется с помощью элементов `class`, что удобно для большого объема документации. Эти элементы могут быть изменены только после весьма сложного и долгого процесса согласования, специально созданного для того, чтобы отвратить дизайнеров, работающих в команде, от несогласованных индивидуальных действий.
- Индивидуальные, взаимозаменяемые компоненты сайта очень часто делаются «анонимными», а их сценарии использования практически не снабжаются документами. Данное требование сохраняется наполовину по необходимости, наполовину по инерции: до тех пор, пока CSS не получит широкого признания, использование вложенных элементов `table` (не дающее никаких преимуществ при использовании во встроенной документации) является необходимым для создания оболочек компонентов сайта.

- Неизбежные расхождения между требованиями ступенчатой поддержки (см. раздел «Ступенчатая поддержка» на с. 309) и возможностями браузера решаются с помощью JavaScript и встроенных таблиц стилей.
- Надстройки встречаются весьма редко и в основном во второстепенных элементах контента.

Все эти шаги требуют уже упоминавшегося выше сохранения баланса. Когда команда из 4–6 разработчиков занимается обработкой контента, созданного несколькими авторами, куда более сведущими в редакторском деле, чем в веб-технологиях, то цели последней группы вынужденно получают более низкий приоритет ради сохранения качества работы.

Правило № 3. Будь последовательным

В идеале разработчик стилей должен с легкостью адаптировать свои предыдущие разработки в новые проекты с минимальным изменением в разметке и в CSS. Подобные действия являются одним из примеров применения на практике принципа CSS-дзен. Для достижения этого стилист должен разработать методику создания шаблонов, пригодных на все случаи жизни. Однако на практике достичь подобного гораздо сложнее, нежели в теории, поскольку недостаточная дисциплинированность, вкупе с манипуляционным поведением некоторых менеджеров, заставляет разработчиков (а с ними и дизайнеров) каждый раз заново «изобретать колесо».

Таким образом, правило последовательности требует, чтобы разработчик стилей *узнавал* знакомые условия, был *готов* к ним и по возможности *стойко взаимодействовал* с ними.

Последовательность — это результат изучения, отражения и приготовления.

Последовательность особенно будет полезна вам при работе «внутри» сайта, а также при создании нескольких сайтов на основе одного шаблона.

Внутрисайтовая последовательность — одно из самых больших преимуществ каскадного использования таблиц стилей. Рассмотрим шаблон, в котором текст выводится в три колонки. Для создания подобного шаблона необходимо сравнительно небольшое количество правил. Если вам в дальнейшем понадобится создать на основе этого шаблона презентационную оболочку, содержащую всего две колонки, то для этого будет необходимо задать дополнительную таблицу стилей. Например:

```
                #main #priorityContent   { width: 42em; float: left; }
              #priorityContent #bodyCopy { float: right; width: 24em;
                                           padding: 0 1.5em 0 1.5em; }
               #priorityContent #sidebar { margin-right: 27em; }
                         #main #tertiary { margin-left: 42em; }
    body.mySpecialCase #main #priorityContent { width: auto; }
body mySpecialCase #priorityContent #bodyCopy { width: 34.5em; }
 body.mySpecialCase #priorityContent #sidebar { margin-right: 37.5em; }
          body mySpecialCase #main #tertiary { display: none; }
```

Селекторы в приведенном выше коде перегружены для того, чтобы дать читателям представление о родительско-дочерних отношениях элементов.

Необходимые нам три колонки описаны в первых четырех правилах. Другие удаляют из потока документа элемент `#tertiary` и приводят оставшиеся элементы в соответствие с отображением в двух колонках. Если бы не частные случаи, то для отображения были бы необходимы только первые три правила.

Дополнительные правила показывают преимущества простоты, которая все же присутствует в этом примере, особенно если сравнивать с вариантом создания отдельных таблиц стилей для двух и трех колонок. (Компромиссным, а также наиболее часто встречающимся вариантом является включение третьей колонки с помощью скрипта `include`, который активизируется только по необходимости.)

Разработчик стилей, написавший приведенный выше стиль, предполагал, что для отображения с двумя и тремя колонками будет использован только один шаблон, что говорит о его последовательности. Создание *дополнительных* шаблонов повышает риск возникновения ошибок, но в первую очередь добавляет требований по тестированию. Другой опасностью внедрения раздельных шаблонов является разветвление кода, которое возможно в случае любого изменения шаблонов. В результате через некоторое время оба шаблона могут оказаться настолько непохожими друг на друга, что усилия, потраченные на их приведение к общему виду, многократно превзойдут затраты на создание одного шаблона с несколькими видами презентации.

Другим выражением последовательности в этом сценарии является последовательность в дизайне. Как правило, количество стилевых правил, необходимых для реализации подчас несовместимых проектных решений значительно превышает количество элементов стиля, задействованных при соблюдении последовательности в дизайне. А чем больше различных элементов, тем выше вероятность ошибки, конфликта или неправильной обработки движком браузера.

В итоге, наилучшим выражением последовательности является многократное использование элементов. Если у вас есть шаблоны и таблицы стилей, пригодные на все случаи жизни, то разработка стиля в новых проектах не займет много времени. Достаточно открыть подходящий шаблон, при необходимости изменить значения `class` и `id`, а также добавить таблицы стилей и интерфейс, в соответствии с пожеланиями заказчика и вашими креативными идеями. Какой-то час работы, и вы можете приступать к верстке и различным улучшениям, вместо того чтобы тратить часы и дни на создание разметки и стиля с нуля.

Если вы с самого начала уверены в своем шаблоне и структуре документа, то это значит, что вы свободны от сиюминутных проблем и, следовательно, можете спокойно заниматься созданием элементов взаимодействия с пользователем, а также какими-либо акцентами дизайна.

Управление шаблонами для достижения последовательности

Обсуждение *шаблонов* в этом параграфе отсылает нас к идее «строительных лесов»: коллекции простых разметок, содержащих по четыре-пять секций страницы каждая. Этот внешний каркас должен вместить в себя другие шаблоны страниц или секций, используемых при создании сайта. Таким образом, дизайнер, который будет заниматься стилями вашего сайта, легко сможет понять и представить расположение и связи между любыми колонками и другими компонентами страницы.

Дизайнеры представляют наибольшую опасность для такого «телескопического» подхода, поскольку они зачастую подвержены ошибочному мнению, что вложения ресурсов в управления шаблонами измеряются *арифметически*. На практике же следует говорить скорее о *логарифмической* зависимости. Если два шаблона требуют в два раза больше времени и усилий, чем один, то затраты на третий шаблон будут превышать предыдущие гораздо больше, чем в два раза.

Такой резкий рост необходимых ресурсов возникает из-за того, что шаблоны, как правило, создаются для соответствия коммерческим целям, *частично или полностью уникальным*. В результате, каждый шаблон обладает уникальным набором элементов и их отношений, а также ошибок и требований. А если добавить к этому обычные трудности, которые возникают при внедрении нового шаблона, то возросшие ресурсозатраты становятся совершенно понятными.

Для борьбы с этим явлением помимо постоянного сокращения элементов `class` и `id`, лучше всего подходит изменение самого процесса создания новых шаблонов. Например, приступать к макетированию и тестированию только после того, как подавляющая часть акционеров проекта примет решение о необходимости нового шаблона. Иначе придется при каждом удобном случае создавать новые шаблоны, а это неизбежно приводит к описанному выше ветвлению кода.

Правило № 4. Придерживайся своего курса

Примеры разметки, структуры страниц и номенклатуры, приведенные в этом разделе, да и во всей книге, в основном противоречат общепринятым, используемым для создания больших сайтов. Причины этого будут описаны ниже.

Пожалуй, самым трудным в создании страницы является сохранение чувства места. Подобное чувство развивается только с годами, благодаря постоянной практике.

Следует постоянно помнить, что все веб-документы существуют в разных контекстах, в разных информационных пространствах, а ваша команда разработчиков способна контролировать только *некоторые* из них. Следует помнить, что чем проще пользователю будет ориентироваться у вас на странице (см. раздел «Навигация: ориентация и указатели» на с. 86), тем лучше он оценит ваш сайт. А если вы сами не способны представить расположение элементов контента на вашей странице, то вряд ли сможете донести это до посетителей.

Самое главное, понятие места — краеугольный камень, начало и конец последовательности: элементы лежат внутри элементов, которые лежат внутри документов, которые существуют на сайтах, которые в свою очередь являются частью среды сайта, находящейся в состоянии постоянного изменения. Чем точнее вы сможете определить место своего сайта в этой сайтовой вселенной, тем проще вам будет создать качественные таблицы стилей.

Опытные пользователи активно взаимодействуют с подсказками на локациях. Опытные разработчики должны твердо знать, какие подсказки и где они разместят, прежде чем начнут создавать страницу.

Рассмотрим нижеследующее дерево элементов:

```
○ body
  • h1
  • div#main
    • div#priorityContent
      • div#bodyCopy
        • h2
        • div.section
        • ...
      • div#sidebar
        • ...
    • div#tertiary
      • ...
    • ul#primaryNav
    • div#footer
      • ul#secondaryNav
      • p#colophon
```

Оно представляет минимум 16 простых контекстных элементов, к которым могут быть применены элементы презентации. Добавим к этому обозначение *границ контента*, выполненное с помощью элементов `id` и `class`, которые можно добавить в элементы `body` (а возможно, и другие).

Стилист может выделить какие-либо элементы контента, независимо от их местоположения в исходниках сайта, отмечая и комбинируя границы этих элементов или документов. Подобный прием часто используется для выделения важных частей контента.

Если элемент можно выделить, то его можно подвергнуть стилевой обработке, а если его можно обработать, то он может стать локационной подсказкой для

пользователя. (Следует помнить, что не все элементы *должны* становиться подобными подсказками.)

Документация на продукт в качестве компаса

Идеальные таблицы стилей — это целый сайт, основанный на контекстно-зависимых селекторах и тщательно созданном дереве документов. Но любой сайт, даже самый маленький, только выигрывает, если снабжен дополнительной документацией. Как правило, такая документация относится к дизайну, особенно к обработке шрифтов и спецификации системы координат. Ниже приведены еще три особенно полезных компонента документации:

- Описание таблиц стилей.

  Эту часть документации создать проще всего, однако она очень важна для разработчиков стилей. Любая достаточно полная коллекция таблиц стилей относится к нескольким правилам. Описание таблиц сводит все правила в одном месте и указывает на ресурсы сайта, к которым они относятся.

- Стандарты продукта/кода.

  Строится на описании таблицы стилей и описывает элементы разметки, а также типы селекторов, применяемые в данном типе контента. Например, кто-то может назвать мою привычку использовать `h1` (без `id`) с единственной целью идентификации сайта, а `h2` для первого заголовка — стандартами продукта.

- Руководства по стилю.

  Данная часть простым языком объясняет не только как, но и почему прием был применен к конкретному элементу в конкретном месте страницы.

Качественная внутренняя документация особенно полезна для разработчиков, которые приступают к переделке уже готового сайта или на каком-то довольно позднем этапе присоединяются к команде разработчиков. Она позволяет довольно быстро создать представление об архитектуре сайта и структуре его шаблонов. При отсутствии подобной документации новички, пришедшие в проект, вынуждены тратить много времени, чтобы разобраться во всех тонкостях и нюансах. Несмотря на огромное значение внутренней документации, в полном объеме она встречается довольно редко.

Чем эффективнее вы будете применять правила, описанные в этом разделе, тем меньше будет необходимости во внутренней документации.

CSS-дзен

Термин «CSS-дзен», по сути дела, неправильный. Дзен — одно из течений буддизма. Дзен-буддизм подчеркивает взаимосвязь всех вещей, особенно живых существ, среди многих аспектов веры. В применении к веб-разработкам этот термин пришел от традиционного английского названия японского сада

камней. Эти сады создаются с эстетической целью и символизируют стремление к точности и гармонии, несмотря на непредсказуемость и постоянную изменчивость природы.

Значимость сада камней для буддийской медитации (например, в качестве объекта созерцания) привела к появлению выражения «Zen garden» (сад дзен), которое стало названием популярного сайта Дейва Ши (http://www.csszengarden.com).

Этот сайт развивает очень важную идею:

> Один качественно сделанный шаблон разметки можно использовать практически для любых случаев. Подобные шаблоны позволяют менять дизайн, почти не затрагивая при этом разметку (разве что в случаях значительного изменения структуры информации, содержащейся в документе или на сайте).

Подобно саду камней, который помогает буддистам в медитации, CSS-дзен поможет в достижении основных целей проекта. Кроме того, подобные сайты сохраняют шаблоны внутренней разметки функционально статичными, точно так же, как камни в саду дзен, расположение которых кажется хаотичным, но на самом деле подчинено определенным законам и достаточно жестко определено.

Функциональные принципы CSS-дзен

Приемы и идеи, описанные в предыдущих разделах, весьма полезны, поскольку делают работу разработчиков стилей более эффективной и качественной. С другой стороны, идеал CSS-дзен построен на принципах, имеющих специфическую точку зрения на контент и структуру:

1. Информация и внешний вид отделены друг от друга, чтобы никто *не мог* утверждать, что природа одного зависит от другого.
2. Приоритеты в потоке информации определяются не *положением* информации, а *отношениями*.
3. Содержимое разделено по степени важности способом, понятным и видимым каждому посетителю страницы.
4. Для каждого пересечения содержания и оболочки нужно найти свое оптимальное решение.

Стоит разработчику стилей однажды применить эти принципы в своей работе – и его точка зрения изменится. В табл. 5.1 представлено диалектическое сравнение подходов дизайнеров к разметке и стилям в зависимости от соблюдения или несоблюдения ими принципа CSS-дзен.

Следование принципам CSS-дзен приводит к созданию сайтов, где форма вытекает из функции, на любых уровнях. А поскольку основная функция — это то,

что привлекает посетителей, то применение принципа CSS-дзен имеет право на существование.

Таблица 5.1. Сравнение подходов дизайнеров к разметке и стилям

Принцип	Несоблюдение дзен	Соблюдение дзен
Разделение	Структура и исходный ресурс разделены.	Структура диктуется приоритетами и (если необходимо) таксономией.
Взаимосвязь	Те или иные части контента неразделимы и расположены в строгом порядке.	Текст можно подразделять различными способами, преподносить в произвольном контексте, переконструировать, дробить. Дополнительная информация предоставляется с помощью гипертекстовых ссылок и метаданных.
Делимость	Разметка предназначена для презентации и разграничения контента. Часто используются вложенные элементы, но редко для придания дополнительного значения включенного в них контента.	Разметка и контент следуют логической таксономии и могут быть при необходимости присвоены даже мельчайшим элементам и иконкам.
Изменяемость	Структура документа либо формируется по принципу «всех под одну гребенку», либо подстраивается под требования проекта.	Контент и расположение элементов в дереве документов следуют афоризму «всему свое место и все на своем месте»; это может выражаться по-разному в зависимости от целей проекта, аудиторий и тем.

Информационная архитектура и удобство использования Интернета

Эта книга вносит несколько новых идей, касающихся веб-программирования:

- Гипертекстовые ссылки — это начало, конец и середина Интернета.
- Веб-ресурсы можно рассматривать как многомерные, а не как линейные.
- Бесконечное число способов, которыми контент может быть связан, скомбинирован или разбит на подразделы, иллюстрируют важность создания эффективных указателей на веб-сайтах.
- Каждый интернет-сайт или приложение являются многослойным ресурсом, который можно значительно улучшить.
- Не существует Единственно Верного Способа создания сайта, поскольку требования к интернет-страницам существенно меняются в зависимости от коммерческих целей или операционных сред пользователей.

Искусство *информационной архитектуры* пытается обосновать эти идеи, а также справиться со сложностями дизайна, созданными ими. Главной целью

данной дисциплины является повышение обнаруживаемости и простоты использования информации, ведь какая ценность может быть от информации, которую нельзя найти и использовать?

Это вводный раздел, поэтому, если вы хотите узнать подробнее об информационной архитектуре, я рекомендую вам посмотреть книгу Стива Круга «Веб-дизайн: книга Стива Круга, или "Не заставляйте меня думать!"» и книгу Луи Розенфельда и Питера Морвилля «Информационная архитектура в Интернете». Кроме того, вы можете найти полезную информацию на сайтах www.boxesandarrows.com и www.asis.org.

Информационные архитекторы, специализирующиеся на веб-контенте и интерфейсах, должны рассматривать предыдущие идеи как данность, иначе они рискуют прийти к нигилизму, поскольку без фундаментальных фактов или предположений становится невозможно эффективным образом организовать содержание веб-страниц и пользовательский интерфейс.

На практике большинство сайтов не нуждается в специалистах по информационной архитектуре. Причиной этого может быть ограниченность области или недостаточное финансирование. В результате разработчики вынуждены дополнительно осваивать этот весьма полезный, хотя и не основной навык.

Мультиразмерность

Как неоднократно указывалось в этой книге, в отличие от традиционных источников информации (печать, видео, аудио) Интернет имеет нелинейную структуру. Размерность веб-содержимого (не представление) может быть ограничена следующим образом:

- *Длина.*

  Аналогично линейной природе традиционных источников информации: начало и конец содержания документа.

- *Ширина.*

  Положение веб-документа в пределах логической области, включающей все непосредственно связанные с этим документом ресурсы. Зависит от «ситуации». Для отражения данного «измерения» сайта в последнее время стала часто применяться технология интеллект-карт.

- *Глубина.*

  Контент с точки зрения возможности воздействия на него пользователя. Глубина становится особенно заметной при применении прогрессивного улучшения.

- *Энтропия.*

  Документы, чувствительные ко времени, а также документы, отдельные элементы которых заполняются пользователями. За время своего существования могут измениться до неузнаваемости.

- *Ситуация.*
  Положение и контекст документа в истории интернет-сеанса какого-либо посетителя. «Ситуация» отличается от «ширины» и «энтропии» тем, что может быть осознана и изменена посетителем.
- *Цель использования.*
  Назначение данного контента и структуры. Например, формат RSS предназначен для того, чтобы публиковать постоянно меняющийся материал, а HTML — для нормального чтения. Также определяет пользовательский интерфейс в веб-приложениях и резюмирует данные работы поискового движка (SERP).
- *Степень детализации.*
  То, как меняется восприятие контента при добавлении, изъятии, экспорте или импорте каких-либо его частей.

Наше понимание веб-контента изменяется точно так же, как меняется представление о материальном объекте при изменении его длинны, ширины, глубины или энтропии.

Люди привыкли мыслить в двух измерениях, оперировать в трех и жить в четырех. А информационным архитекторам и создателям сайтов и стилей приходится решать, какое из семи описанных выше измерений использовать для создания подсказок о положении в локации и других навигационных указаний.

Даже такое, весьма упрощенное, описание веб-контента и структуры документа дает нам представление не только о четырех материальных измерениях, но также о трех, характерных только для Интернета. Поэтому нет ничего удивительного, что по сравнению с четко ограниченными традиционными информационными средами Интернет предоставляет людям гораздо больше возможностей для получения информации, даже несмотря на то что подчас ориентирование в нем сродни какому-то тайному знанию.

Навигация: ориентация и указатели

Советы по написанию эффективного текста ссылок вы найдете в разделе «Эффективные названия ссылок и значения заголовка» на с. 162.

Принимая во внимание бесконечную изменчивость веб-контента и интерфейсов, можно с уверенностью сделать вывод, что лучшим подспорьем для посетителей страниц будет четкое указание на их местоположение во время путешествия по сайту.

Приемы, применяемые для достижения этого, можно разделить на шесть групп:

- *Первичная и вторичная навигация.*

  Ссылки устанавливаются в конце (или в начале и конце) документа. Каждая из таких ссылок ведет к документу, который потенциально может быть интересен пользователю. Ссылки могут быть вложенными и составлять двух- или многоуровневые иерархии. Применение данного приема позволяет четко выделить и определить документ не только с помощью заголовка, но и благодаря активным ссылкам, обрамляющим текст.

- *Грамотно выстроенные навигационные цепочки.*

  Подобно вложенным элементам навигации, эти ссылки прикреплены к некоторым узлам иерархической организации сайта, однако представлены в зависимости от уровня значимости информации, от самого высокого к самому низкому. Такой прием не очень помогает определить местоположение документа в информационном пространстве, но бывает очень полезен, например, в качестве дополнения к контенту, выводимому на печать, так как позволяет человеку, читающему распечатку, четко понять, как найти в Интернете данную страницу.

- *Теги.*

  Для каждого документа назначаются определенные ключевые слова, а на каждой странице сайта приводится полный список этих слов. Если проследовать по ссылке от ключевого слова, то выводится список ссылок на соответствующие документы, которые могут быть отсортированы по нескольким критериям (например, по дате, популярности, частоте упоминания ключевого слова и т. п.).

- *Карты сайтов.*

  Принципиальная разница между вложенной навигацией и картой сайта заключается в том, что последняя представляет собой отдельную страницу, ссылки с которой ведут ко всем документам и приложениям данного сайта.

- *Встроенные ссылки.*

  Добросовестные разработчики контента часто вставляют ссылки на соответствующие документы в подходящие ключевые слова и фразы прямо в тексте документа.

- *Поиск по сайту.*

  Наподобие любой поисковой системы (Google, Яндекс и т. п.), только действующей в границах сайта.

Области применения указателей будут обсуждаться в последующих главах нашей книги: первичная и вторичная навигация — в материалах, посвященных структуре и спискам.

Обычно карты сайтов размечаются в виде списков или, иногда, в виде нескольких колонок. Списки тегов также размечаются в пределах списков с использованием значения свойства `display` для элементов `inline`. Для отражения частоты повторения ключевых слов разметка может производиться с помощью классов.

Стратегия посещения сайтов

Наиболее общие выводы, которые могут быть сделаны о посетителях сайтов, это:

- Цель посетителя сайта можно отнести к одной (или нескольким) из четырех основных категорий: информация, услуги, продаваемые продукты или развлечения.
- У посетителей есть два основных метода, которыми они могут достичь своей цели: просмотр или полнотекстовый поиск.

Чаще всего посетители предпочитают пользоваться сторонними поисковыми движками, поскольку они предоставляют точную информацию в пределах диапазона результатов, среди которых могут быть сделаны сравнения. Также большинству пользователей известно, что некоторые сайты, в частности большие социальные сети, могут удовлетворить множественные цели интернет-сессий.

Один из самых важных вопросов, которые необходимо решить во время процесса проектирования, звучит так: каким образом дизайн и внедрение пользовательского интерфейса будут служить основным целям посетителей?

Понятно, что в рамках одного сайта очень трудно (если не невозможно) применить все способы размещения указателей, обсуждаемые в нашей книге. Однако для разработчиков стиля очень важно знать, в какой части страницы обычно располагается тот или иной указательный элемент. Обладающим этим знанием дизайнерам будет гораздо проще разработать концепцию удобных и отвечающих концепции сайта локационных подсказок.

- *Первичная навигация.*

  Ориентирована горизонтально. Как правило, располагается сразу после верхнего колонтитула. Вложенные ссылки располагаются в ряд под главной «секцией» ссылок. Домашняя страница сайта обычно связана с логотипом сайта, располагающимся в левом верхнем углу страницы (в верхнем колонтитуле).

- *Ссылки нижнего колонтитула.*

  Располагаются в нижнем колонтитуле страницы, заданы с помощью `display: inline` или `inline-block`. Часто центрированы. Обычно применяется шрифт меньшего размера, чем в теле документа.

- *Навигационные цепочки.*

  Горизонтально ориентированы. Помещаются сразу под первичной навигацией или на странице, задающей форму вывода документа на печать.

- *Списки тегов.*

  Справа от основного контента при отображении в три колонки. Под второстепенным контентом при отображении в две колонки. Сразу под основным контентом в тех редких случаях, когда используется только одна колонка. Если при этом список тегов часто используется, то рекомендуется все же создать для него отдельную колонку.

- *Карты сайтов.*

  Связаны с нижним колонтитулом каждой страницы на сайте. Как правило, располагаются на отдельной странице в колонках.

- *Поиск.*

  В правом верхнем углу верхнего колонтитула, иногда дублируется в правом поле нижнего колонтитула. Результирующие страницы часто отображаются в одну колонку, даже если в самом сайте применяется несколько колонок. Причиной подобного вывода является, как правило, недостаточная гибкость модулей или приложений, предназначенных для генерации результатов поиска.

Рекомендации по созданию удобных интерфейсов

Для создания интерфейса мало объявить что «*x* ведет туда-то». Первейшая цель любого интерфейса — быть удобным для пользователей. Однако простого следования за вкусами толпы еще недостаточно для того, чтобы создать хороший интерфейс.

Первейшая и важнейшая цель, которую нужно преследовать, — это последовательность. Рассмотрим несколько различных интернет-страниц. Для пользователя ситуация будет идеальной, если на каждой из них элементы навигации и интерфейса (ссылки, теги, окно поиска, заголовки и т. д.) будут располагаться на одних и тех же местах.

Еще лучшим (хотя и гораздо более трудным) вариантом было бы сохранение этой последовательности внутри отдельных секций страницы, так, чтобы пользователь мог очень быстро найти как необходимую ему информацию, так и подсказку, иллюстрирующую его положение на странице.

Теперь перейдем к различным техникам создания указателей, которые, пусть и постепенно, упростят пользователям достижение их целей.

- Площадь, занимаемая навигационными ссылками или тегами, должна быть по возможности большой. Как минимум для каждого элемента свойству `display` следует присвоить значение `block` и добавить дополнительный отступ (см. главу 8). Также рекомендуется добавлять к поисковому окну кнопку подтверждения.
- Следует избегать использования всплывающих меню, выводящих внутренние ссылки, при наведении на них мышки (такие же, как например «Все программы» в меню «Пуск» в Windows), поскольку подобные меню требуют от пользователя очень хорошей моторики, что неудобно для людей с ограниченными возможностями, а подчас и для обычных пользователей. Microsoft применяет подобное меню, для того чтобы сохранить область отображения с целью приведения к наименьшему общему знаменателю аппаратной конфигурации. Так же в Windows реализована поддержка обращения к меню «Пуск» с клавиатуры и возможность помещать ссылки на программы прямо

на рабочий стол. Указатели на странице должны быть подобны иконкам на рабочем столе Windows. Они должны быть так же хорошо заметны и информативны.

- На многих страницах ссылки первичной навигации сделаны меньшими по размеру, чем основной текст, а некоторые дизайнеры еще и уменьшают их контрастность. На самом деле, все следует делать наоборот — ваши усилия необходимо направить на увеличение размера и контрастности этих элементов, помогающих пользователям ориентироваться и перемещаться по сайту.
- Делайте пометки и подсказки четкими и ясными. Иначе они рискуют превратиться в так называемую «мистическую навигацию» (термин предложен Винсентом Фландерсом), когда вы знаете, что на другом конце ссылки что-то есть, но вот что там — совершенно непонятно. Переход по такой ссылке чем-то сродни прогулке по неосвещенным аллеям парка глубокой ночью.
- Старайтесь не пользоваться миниатюрными полями `input type="text"` в поисковых формах. Уменьшенный размер полей приводит к тому, что поисковый запрос хуже читается, что сразу же повышает вероятность ошибки и потратить время впустую. Конечно, не стоит делать эти поля гигантскими, но их размер должен дать возможность пользователю легко прочитывать все, что он туда вводит.
- При отображении текущего документа в навигационном контексте, четко выделяйте его из окружения. Если возможно, удалите элемент `a`, сохранив его содержание.

Отключение ссылок на текущий документ

Отключение браузерного интерфейса, имеющего отношение к ссылкам, используется для того, чтобы одурачить пользователей и внушить им, что документ не имеет ссылок на себя и чтобы убрать все якорные теги. Для того чтобы убрать все ссылки, вам нужно совершить следующие действия.

Сначала нужно создать `class` для интерактивных. При этом необходимо задать для ссылки соответствующее значение свойства `cursor` (см. главу 8, а также сопутствующий сайт книги). В примере, приведенном ниже, этот класс назван `selfLink`.

```
for (var i = 0; i < document.getElementsByTagName("a").length; i++) {
   if (document.getElementsByTagName("a")[i].href) {
if (document.getElementsByTagName("a")[i].href == document.
   location.href) {
      document.getElementsByTagName("a").className = "selfLink";
     document.getElementsByTagName("a")[i].onclick = return false;
    }
   }
  }
```

Затем, если вы работаете с документом, MIME-тип которого равен `text/html`, добавьте JavaScript-код в функцию, которая инициирует `onload`.

Подобная техника может также применяться для сообщения пользователям об ошибках в формах ввода.

Наконец, следует отметить, что из соображений безопасности информация в строке состояния, показывающая URI, не может быть отключена. Поэтому способ, предложенный выше, подходит только для неискушенных пользователей Всемирной паутины.

Предсказание поведения пользователей с помощью сценариев и тестов

Различные формы навигации помогают ориентироваться в направлении путешествия. А что поможет определить, что ссылка уже была посещена?

Ниже приведены типичные способы, которыми пользуются большинство дизайнеров:

- выделяющийся заголовок или название статьи;
- навигационные пояснения, возможно, в комбинации с деактивированной ссылкой;
- особая расцветка фона и оформления страницы;
- особый стиль посещенных ссылок.

Подобные приемы не очень удачны, когда речь заходит о быстром и четком понимании пользователем своего местоположения на сайте. Как правило, пользователи рано или поздно приобретают чувство места и умение быстро ориентироваться в Интернет-пространстве. До этого перемещение по странице скорее напоминает игру «горячо — холодно». Такое отсутствие точности является прямым следствием безграничности Интернета. Знающие об этом опытные пользователи всегда с осторожностью подходят к интернет-серфингу.

Рассмотрим, как ориентирование работает в реальном мире. При наличии карты с достаточно маленьким масштабом и адекватным отображением окружающей местности, человек, обладающий навыками чтения карты и достаточными знаниями об окружающей местности, сможет довольно быстро определить свое местоположение с точностью до нескольких метров.

Обратите внимание на то, каким образом происходит ориентирование. Для того чтобы определить свое местоположение, необходимо знать рельеф местности, а также обладать еще рядом навыков, полезных для выживания в экстремальных условиях.

Конечно, для веб-сайтов, обсуждаемых в этой книге, понятие выживание не применимо, а случайный посетитель страницы вряд ли имеет представление о ее виртуальном рельефе, но, для того чтобы предсказать поведение пользователей, разработчики и дизайнеры используют такой же прием, какой применяют спасатели для поиска заблудившихся людей, — ставят себя на место потерявшегося.

«Искать» и «Найти» применительно к Интернету означает тестирование пользовательского поведения. Два самых распространенных способа тестирования — это применение сценария и пользовательское тестирование. Сценарии описывают условия, исходя из которых члены команды разработчиков моделируют принятие решения пользователями сайта, а затем анализируют их. При пользовательском тестировании обычные интернет-пользователи работают с прототипом сайта. Подобный тест дает более точные результаты, но требует значительно бóльших вложений.

Оба способа иллюстрируют важность применения таксономии на сайте. Хорошо организованная таксономия позволяет пользователю представить виртуальную карту сайта всего после нескольких запросов.

Таксономия и номенклатура

Таксономия — это создание таксонов (жестких иерархических форм классификации). Первое применение таксонов связано с именем шведского биолога Карла Линнея, который разработал классификацию всех форм жизни. Эта система применяется и по сей день и состоит из семи уровней. Все эти уровни, вплоть до человека, представлены в табл. 5.2.

Таблица 5.2. Пример таксономии в биологической классификации

Уровень	Наименование	Описание
Царство	Животные	Животные
Тип	Хордовые	Обладание позвоночником
Класс	Млекопитающие	Обладающие молочными железами
Порядок	Приматы	Первого порядка
Семейство	Гоминиды	Человекоподобные
Род	Хомо	Человек
Вид	Сапиенс	«Разумный»

Таксоны также могут быть применимы к информации вообще, а в частности к словарям, в которых упоминается какой-либо специфический контекст. Многие жители Соединенных Штатов знакомы с Классификацией Библиотеки Конгресса и классификацией североамериканской Системы Промышленности. Обе эти классификации широко используются для организации информации иерархическим способом.

Номенклатура, в свою очередь, слово практически жаргонное. Можно сказать, что это специализированный словарь, система наименования вещей, с которой согласны большинство пользователей.

Можно сказать, что таксономия организует номенклатуру, хотя вместе с тем она часто организовывает неконтролируемые словари (словосочетание «контролируемый словарь» синоним слова «номенклатура» и часто используется учеными, изучающими информацию).

Возьмем для примера следующие термины:

- Контент
- Сайд-бар
- Хироу шот
- Фонт
- Спейсер
- Клиент

В какой бы стране вы не употребили эти слова, большинство веб-разработчиков прекрасно поймут, о чем идет речь.

Эффективное использование таксономии и (в меньшей степени) номенклатуры позволит вам создать соответствующий контекст для вашего каскада, сохранив при этом гибкость.

Применение таксономии к таблицам стилей страницы

Применение таксонов на сайтах можно отнести к двум различным уровням: к телу контента и к структуре разметки шаблонов.

Применение таксонов к контенту отсылает нас к предыдущим дискуссиям о сценариях и пользовательских тестах. Создавая тесты, можно понять, как сайт будет использоваться и каким образом надо будет разместить информацию с точки зрения ее приоритета.

Представим себе небольшую компанию, занимающуюся розничной торговлей. Ее сайт будет главным образом использован для следующих целей:

- найти местоположение компании и часы ее работы;
- узнать номера телефонов, а также о других способах связи с представителями фирмы;
- получить информацию об акциях и распродажах;
- узнать о наличии товара и его стоимости.

В результате можно создать рекомендацию по поводу пунктов, которые должны присутствовать на сайте небольшой торговой организации:

- адреса магазинов;
- контакты;
- акции и распродажи;
- онлайн-магазин.

Кроме того, я бы рекомендовал представить часть информации из всех вышеперечисленных категорий на главной странице, а именно:

- адрес и часы работы главного магазина;
- телефонный номер, адрес электронной почты отдела по работе с клиентами, а также ссылку на группу в социальных сетях;

- информацию о самой значительной скидке, приуроченной к какой-либо дате;
- краткий список продукции, а также опцию «добавить в корзину» для трех-четырех наиболее популярных товаров, продающихся онлайн.

Если каталог продуктов для онлайн-магазина слишком велик, то рекомендуется сделать ссылки на категории продуктов в списке вторичной навигации.

Четыре основные страницы сайта должны содержать как можно более подробную информацию. Например, на странице «Контакты» должна содержаться подробная информация о телефонах и других способах связи с магазинами, а на странице «Онлайн-магазин» — ссылки на все категории товаров.

Каждая секция сайта должна содержать локационные подсказки. Для организации таких подсказок лучше всего использовать элементы `class` и `id` с названиями элементов аналогичных или похожих на названия элементов в вашем дереве шаблона, например:

```
<body class="stores" id="KCDetails">
```

Другие элементы `body`, имеющие отношение к `.stores` должны получить значения `id` наподобие `StLouisDetails`, `SpringfieldDetails`, `ColumbiaDetails`, и т. д. В нашем примере это помогает выделить информацию, относящуюся к самым большим городам, расположенным на шоссе Миссури. Для более крупных сайтов возможно предоставление не очень подробной информации, касающейся штатов или регионов, где находятся подразделения компании.

Подобный метод присвоения `class` и `id` отдельным страницам дает возможность использовать ваши селекторы в чрезвычайно ограниченных областях, что сокращает усилия, затрачиваемые на создание специальных акцентов на заголовках статей и т. п. Например:

```
#KCDetails h2 { background-image: url(/images/heading_kc_store.gif); }
```

Посмотрите, как можно реализовать навигацию, если учесть, что подуровни вряд ли будут иметь одинаковое количество элементов в каждой секции.

```
 /* #navOnlineStore actually holds two lines of links.
    #navStores only one */
.stores #nav #navStores { height: 1.5em; padding-bottom: 1.5em; }
/* make the unneeded sublevels GO AWAY */
.stores #nav #navContact,
  .stores #nav #navPromo,
 .stores #nav #navOnlineStore { display: none; }
```

Предыдущий пример заслуживает дополнительного внимания, поскольку иллюстрирует способы, которыми элементы могут быть вложены.

```
<body class="stores" id="StLouisStore">
...
<ul id="nav">
 <li id="navStores">Our Stores
  <ul id="subNavStores">
```

```
  <li id="#navKCDetails"><a href="/stores/kc/">Kansas City</a>
   </li>
  <li id="navStLouisDetails">St. Louis</li>
  ...
 </ul>
</li>
</li>
...
</body>
```

Вопрос заключается в том, стоит ли включать организацию контента вашего сайта в таблицы стилей? Таксономия и номенклатура на уровне страницы более субъективны: то, что для одного человека `#sidebar`, другой назовет `#highlights`.

Хорошей новостью можно считать то, что если вы соблюдали принцип последовательности при создании документов для сайта, то функциям, взаимоотношениям и типовым размещениям различных типов контента в шаблоне или на отдельной странице можно присвоить номенклатуру, взятую непосредственно из документов и свои собственные таксоны. Мой способ определения структуры документа показан во многих примерах, приведенных в этой книге, а также на многих действующих интернет-сайтах.

ДОСТИЖЕНИЕ ВЫСОКОЙ СТЕПЕНИ ДЕТАЛИЗАЦИИ В БОЛЬШИХ САЙТАХ

Особенно большие сайты или сайты крупных предприятий, как правило, не могут (по соображениям безопасности) использовать элементы `id` так, как представлено в большинстве примеров в этой книге. Во многих организациях можно заменять `id` на `class`, но в некоторых фирмах стилистам запрещено даже это. Менее сложные проекты могут практически без потерь принять это ограничение. Однако если вам все же необходимо использовать элементы дизайна, которые плохо согласуются с требованиям безопасности, то рекомендуется применять стили с помощью библиотек JavaScript, например `jQuery`.

Новые структурные элементы (HTML5)

Помимо элемента `nav`, про который будет подробнее рассказано в главе 7, в HTML5 вводится еще несколько новых структурных элементов:

- `Section`;
- `Article`;
- `Header`;
- `Footer`;
- `Aside`;
- `Figure`.

На момент написания этой книги ни один из браузеров не имел встроенной поддержки этих элементов. При этом для организации поддержки не требуется

вносить в браузеры существенные изменения, поскольку эти элементы не содержат никаких специальных требований к обработке или отображению. Впрочем, эти элементы не должны сильно повлиять на конечных пользователей вебстраниц, так как созданы скорее для удобства разработчиков.

Также до сих пор неясно, какие элементы будут отменены после того, как новая спецификация HTML пройдет процедуру стандартизации. Например, практически точно известно, что `section` будет использоваться в качестве дополнения к функциям определения языка, а вот по поводу `article` и `aside` полной информации пока нет. Также существует ряд принципиальных соглашений об использовании `header` и `footer` для разметки верхнего и нижнего колонтитулов страницы, но гораздо меньше ясности в вопросе об использовании их с элементом `section`. Последняя версия HTML5 позволяет элементу `section` иметь дочерние элементы `header` и `footer`, но в существующих текстах не часто можно встретить подразделы страницы с собственными верхними и нижними колонтитулами.

6 Создание макета в CSS

Когда вы создаете макет сайта, основанный на CSS, первейшей и труднейшей задачей для вас будет поместить отдельные элементы вашего макета именно туда, где они должны находиться. Для этих целей в CSS существует три основных инструмента: позиционирование, `float` и `width/margin`. К сожалению, методы применения этой техники весьма сложны для изучения.

Блочная модель CSS и контроль размера элементов

Когда браузер проводит рендеринг блочных элементов, таких как `div` или `p`, то при обработке каждого элемента учитываются четыре компонента: `content` (содержимое), `padding` (отступы), `borders` (рамки), и `margins` (поля) (рис. 6.1). В настоящее время размеры подобных блоков вычисляются путем *сложения* трех из четырех компонентов, то есть если вы зададите для элемента значения `width` или `height`, то его размеры на самом деле будут больше за счет рамки и отступов. Поля также оказывают влияние на этот процесс, но только если принимать во внимание смежные поля.

В этих правилах отображения существует два исключения: использование для блоков контента значения `auto`, а также сброс параметров.

Режим совместимости и строгий режим

Браузеры используют два основных типа рендеринга: режим совместимости (quirks mode) и строгий режим (strict mode). Выбор типа инициируется присутствием или отсутствием объявлений типа документа, которые подробно описаны в главе 2.

Вместо того чтобы обсчитывать размеры блока, *прибавляя* к длине и ширине значения полей, отступов и т. п., как это происходит в строгом режиме и реко-

мендуется в блочной модели CSS 2.1, режим совместимости рассчитывает размеры элементов из заданных значений `width` и `height` путем *вычитания* из них других значений блока. Подобное поведение аналогично свойству `box-sizing` в CSS3, имеющему два значения `content-box` и `border-box`.

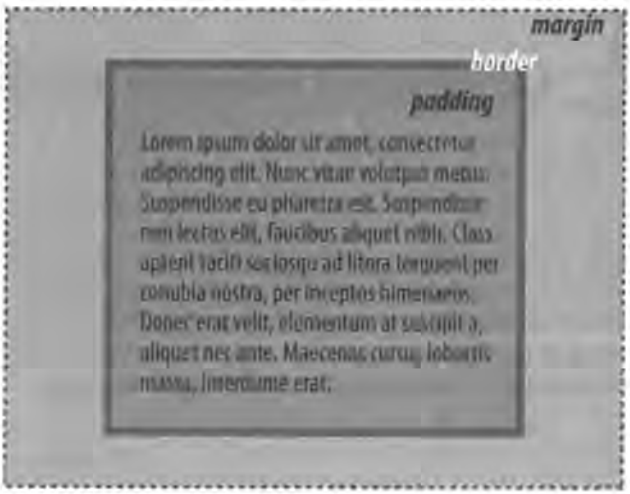


Рис. 6.1. Рассчитанная высота и ширина элементов состоит из четырех составных частей: поля, рамки, отступы и контент

Режим совместимости — это единственный возможный режим рендеринга для браузеров Internet Explorer версий 6 и ниже.

Значение auto

Параметры `width` и `height` имеют по умолчанию значение `auto`. Когда браузер использует это буквенное значение для вычисления ширины элемента, это значит, что элемент будет занимать всю ширину содержащего его контейнера. Для высоты же значение `auto` означает, что элемент будет изображаться в полную величину, независимо от высоты контейнера. Если выше будет находиться какой-либо другой блок, то произойдет наложение, но только в том случае, если у параметра `float` оставлено значение по умолчанию (`none`).

Если у элемента со значением `width`, равным `auto`, задать рамку и/или отступы, то они будут рассчитываться путем вычитания из `width`. То же самое справедливо и для соответствующих значений полей, не пересекающихся с полями других элементов.

Если вместо этого вы зададите дискретное значение `width` для вашего блочного элемента, а значения левого и правого полей будут равны `auto`, то элемент расположится в центре контейнера (рис. 6.2).

В этом разделе несколько раз упомянуты «вычисленные» значения `width` и `height`: размеры элементов после рендеринга страницы браузером пользователя. О них пойдет речь также в главе 14.

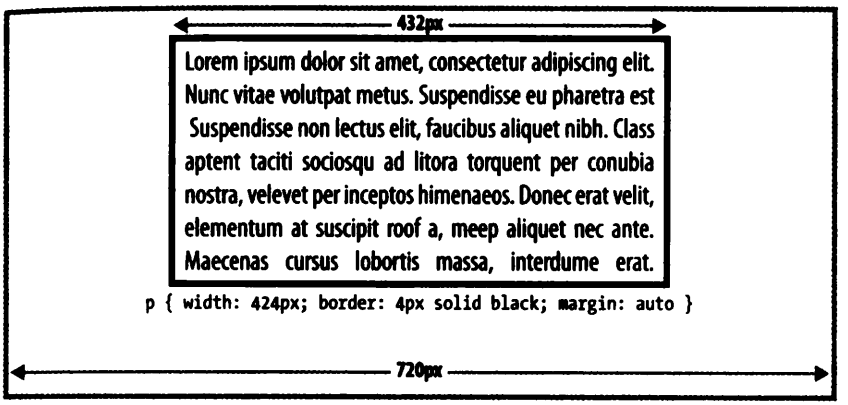


Рис. 6.2. Центрирование текстового блока. Поля автоматически устанавливаются вне рамок

Свойство overflow

Элемент, более узкий, чем его контейнер, может быть отцентрирован по горизонтали независимо от заданной или вычисленной ширины. Однако это правило не работает по отношению к высоте элемента. Значения как высоты элемента, так и высоты контейнера, должны быть известны до того, как CSS приступит к центрированию одного по отношению к другому.

Значение `height: auto` (выраженное или подразумеваемое) как бы сообщает браузеру, что элемент может распространяться вне зависимости от верхних границ контейнера. Поскольку текст, как правило, не имеет истинного размера, растяжение `margin: auto` по оси Y по умолчанию более ресурсоемкое, нежели полезное.

Заполнение элементом его контейнера по высоте достигается с помощью использования свойства `overflow`.

Это свойство описывает, как следует обрабатывать контент, если он превышает по высоте свой контейнер. Это первое свойство, описанное в данной книге, которое позволяет блочным элементам перекрывать друг друга, а дизайнерам нарушать правила потока блочных элементов, описанное в разделе «Поток элементов» на с. 107.

Четыре валидных значения для параметра `overflow` показаны на рис. 6.3. Вот их краткое описание:

- `visible` (по умолчанию)

  Весь контент становится видимым независимо от размеров контейнера. Если параметры элемента (контейнера) или его содержимого имеют значения, отличные от `auto`, то содержание, превышающее размеры контейнера, будет наплывать на соседние с контейнером блоки, при этом размеры контейнера и свойства соседних блочных элементов не будут изменяться.

- `hidden`

  Переполняющее контейнер содержимое будет скрыто от глаз пользователя, при этом размеры контейнера останутся без изменения.

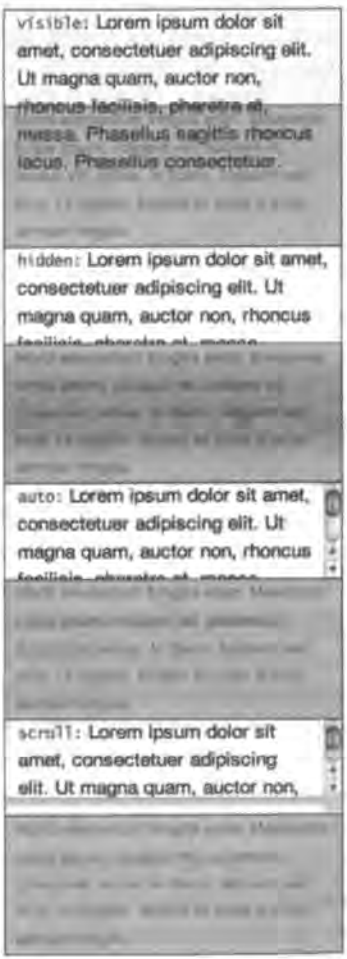


Рис. 6.3. Свойство `overflow` принимает одно из четырех значений

- `scroll`
  К элементу добавляются вертикальные и горизонтальные меню прокрутки и при необходимости элементы управлению скроллингом. Размеры контейнера остаются без изменений.
- `auto` (часто применяется в качестве значения пользовательского агента в `html` или `body`)
  Размеры элемента остаются неизменными, меню прокрутки добавляется при необходимости, причем появляется оно внутри заданных размеров элемента, не увеличивая его площадь.

Кроме того, значение `auto` придает элементу не очень понятное, но потенциально очень полезное свойство. Если элементу со значением `overflow`, равному `auto`, присвоить очень маленькое процентное значение `height` (например, 1%), то элемент расширится, охватывая вычисленную высоту контента, учитывая все отступы, установленные для этого контента. Обратите внимание, что подобные свойства не работают для параметра `width`.

Значения `overflow` также применяются и к размерам блочных элементов по оси X, но такое случается очень редко, поскольку значения `height: auto` почти всегда произвольное. Вот случаи, при которых значение `overflow` отражается на оси X:

- присутствие очень длинного текста, соединенного неразрывными пробелами и содержащего элемент `pre` или свойство `white-space`;
- вставка заменяемых элементов, особенно изображений;
- непредвиденная вставка дочернего элемента с вычисленными или собственными значениями `width`, превышающими ширину родительского элемента. Кроме того, к подобному эффекту могут привести очень сложные веб-приложения или некорректно написанные CSS.

Ограниченные, но не фиксированные размеры элементов

Иногда вам может потребоваться, чтобы элемент заполнил доступное место (в некотором смысле). А в других ситуациях вы, возможно, будете подгонять длину или ширину до тех пор, пока не будете уверены, что элемент соответствует контенту. Например:

```
.articleContainer { width: 80%; max-width: 50em; margin: auto; }
    .sidebarItem { float: right; width: 16.667%; min-width: 288px; }
```

Первое правило в этом примере создает сценарий, в котором справа или слева от элемента есть свободное место, но в то же время элемент может сильно растягиваться в ширину с целью сохранения читабельности (эта проблема обсуждается в главе 12). Второе правило создает блок, занимающий одну шестую ширины контейнера, который также может содержать изображение, размеры которого подчинены требованиям руководства по стилям. Согласно

этому руководству, изображение всегда должно занимать минимально возможное пространство по горизонтали, необходимое для нормального вывода изображения на экран. Обратите внимание, что оба правила используют свойство `max-*` *после* основной части, поэтому отношения между исходным ресурсом и приоритетом остаются актуальными не только в пределах документа, но и в пределах правила.

Управление непредсказуемым

Разработчики стилей предпочитают применять свойство `overflow` при отображении нескольких колонок в случаях, когда контейнер колонки (см. раздел «Создание макета с несколькими колонками» на с. 113) содержит свойства фона или блока (см. раздел «Отступы, поля и рамки» на с. 103), которые должны быть обработаны браузером.

Однако существует множество сайтов и приложений, где границы контента невозможно предсказать, а свойство `overflow` применяется для следующих практических целей.

- Предотвращение разрывов в статичных макетах.

Когда элементы имеют заданные, фиксированные размеры (обычно выраженные в пикселах, см. раздел «Единицы измерения в переменных средах» на с. 58), то изменение размеров текста пользователем может привести к разнообразным разрывам. Конечно, недостаточная грамотность интернет-пользователей в области изменения размеров текста, а также наличие на страницах специальных интерфейсов, делают подобные происшествия достаточно редкими, но опытные разработчики прекрасно знают о такой возможности и умеют ее быстро преодолевать. Для предотвращения разрывов предпочтительнее использовать макет, основанный на сетке, но если это невозможно, то лучшим выходом будет `overflow: hidden`.

- Системы управления содержанием с нечетко определенными границами контента.

  Рассмотрим возможность того, что элемент с четко заданными границами, например объявление или дополнительная панель, вставляется в контент, созданный в соответствии с каким-либо дизайнерским проектом. Для того чтобы избежать наложения информации, рекомендуется использовать `overflow: hidden`. Однако следует заметить, что подобные ситуации чаще всего возникают при несоблюдении культуры дизайна.

- Нормализация форм на странице.

  Близкое расположение элементов управления и связанных с ними ярлыков может привести к ситуации, при которой дочерний элемент выходит за границы своего контейнера. Для того чтобы размеры и формы страницы соответствовали установленной дизайном сайта сетке, рекомендуется использовать свойство `auto/1%`, описанное выше.

- Ограничение размеров контента в Ajax-приложениях.

  В случаях, если посетители страницы применяют функцию Обновить в веб-приложениях, возможны ситуации, когда созданная пользователями информация (особенно изображения) вызывает разрывы в интерфейсе приложения. Для борьбы с этим рекомендуется использовать `overflow: auto`.

Современные браузеры также поддерживают описанные в CSS3 свойства `overflow-x` и `overflow-y`, которые изменяют отображение контента по оси абсцисс и ординат соответственно. Однако следует обратить внимание, что присвоение `overflow-x` и `overflow-y` различных значений не всегда приводит к желаемому результату.

Наконец, дизайнерам стоит помнить, что когда применение значения `scroll` или `auto` создает меню прокрутки, то оно появляется *внутри* области отображения элемента. Среди прочих результатов этого возможно появление двух меню прокрутки вместо ожидаемого одного.

Отступы, поля и рамки

Дизайн сайта часто прорабатывается до мельчайших деталей, а условия контракта требуют переноса этого дизайна с бумаги на компьютер с точностью до пиксела или хотя бы с последовательностью до пиксела. Контроль размера элементов является только частью необходимых действий по достижению этого, ведь также важно учитывать пустые области и границы. О свойствах управления этими важными частями страницы и пойдет речь.

Отрицательные поля

Отрицательные значения можно присваивать свойствам элементов, ответственным за положение на странице. Кроме того, отрицательными могут быть поля элемента.

Если полю какого-либо элемента присваивается отрицательное значение, то его рамка оказывается обрезанной, и контент располагается впритык к рамке соседнего элемента.

Другими словами, отрицательные поля, вместо того чтобы добавлять, убирают пустые области. Эта способность на самом деле куда важнее, чем кажется на первый взгляд. Например, рассмотрим заголовок, разделенный пустым пространством, и правило, предположительно задающее `background-image`. Если дизайн сайта требует, чтобы заголовок появлялся вместе с метаданными, что обычно характерно для заголовков блогов, в разметке метаданные будут с большой долей вероятности располагаться сразу за заголовком, а следом за ними — контент (рис. 6.4).

Минимальная разметка для данного случая потребует отрицательного значения для `marginbottom` заголовка или отрицательного значения `margin-top` для

метаданных, для того чтобы удостовериться, что метаданные появятся в интервале между копией заголовка и созданным правилом.

Рис. 6.4. Соединение заголовка и метаданных с помощью отрицательных значений полей

Более удобным является применение отрицательных полей в нижних колонтитулах страниц. Если в двухколоночном сайте необходимо отцентрировать нижний колонтитул по телу документа, а не по его границам, необходимо, чтобы этот колонтитул находился в контейнере документа, но располагался в его конце. Кроме того, нижнему полю (`margin-bottom`) боковой колонки следует присвоить отрицательное значение, чтобы быть уверенным, что часть нижнего поля боковой колонки не наползает на верхнее поле колонтитула.

Практическая польза подобной техники кажется весьма ограниченной. Однако для достижения большего соответствия проектам и цельности изображения отрицательные поля используются чаще, чем вы себе можете представить.

Схлопывание полей

Элементы HTML появляются со значением `display` по умолчанию. Эти элементы в дальнейшем группируются по подтипам. Например, элементы `div` и заголовок относятся к блочным элементам по умолчанию, но шесть уровней

заголовков относятся непосредственно к классификации заголовков. Если два блочных элемента не имеют подкласса или относятся к одному подклассу, их соприкасающиеся поля будут схлопываться. Такое поведение особенно заметно в элементах параграфов с пользовательскими стилями, например:

```
          p { margin-top: 1.25em; margin-bottom: 1.25em; }
p:first-child { margin-top: 0; }
 p:last-child { margin-bottom: 0. }
        p+p { margin-top: 0; }
```

Последнее из четырех правил добавлено исключительно ради примера. На практике рендеринговый движок браузера определит два последовательных элемента-параграфа в качестве блочных элементов без подтипов и схлопнет их поля.

В описанном здесь примере поля между двумя последовательными параграфами составляют 1.25em (что приблизительно равно высоте одной строки). Если поля этих параграфов не схлопнутся, то отступы между ними увеличатся до 2.5em, что будет весьма заметно для пользователей и, возможно, ухудшит читабельность контента. Обратное поведение можно протестировать, вставляя между разделами другой блочный элемент, например `div`. Верхние и нижние поля блока, представленные в предыдущем примере, будут сохранены, и даже при простой перестановке элементов контент будет выглядеть точно так же, несмотря на то что элемент `div` по умолчанию имеет поля, равные нулю, со всех сторон.

Классификацию `display`, предлагаемую HTML 4, можно найти на сопутствующем сайте книги.

Рамки

CSS предлагает около двадцати свойств для описания рамок. Рамок практически невозможно избежать. Они имеют свои особенности. В этом параграфе мы остановимся не на внешнем виде рамок, а на их влиянии на макеты, особенно пропорциональные (так называемые «резиновые») макеты.

Проблемы с рамками начинаются, когда они встречаются с жесткими ограничениями так называемого strict рендеринга, создающего правила с фиксированной шириной, которые рассекают видимые поля элементов с *пропорциональными* размерами.

Рассмотрим три элемента, создаваемые один за другим:

```
#attractSection .promo { float: left; width: 32%; height: 11.417em;
      margin: 3.125%; }
```

При отсутствии ошибок рендеринга предыдущее правило создает три элемента одинаковой ширины, слева направо, с полями с каждой стороны, равными 1 % от ширины `#attractSection`.

Теперь предположим, что проект сайта требует создания рамки размером в 1 пиксел вокруг каждого из элементов. Добавление этих рамок приведет к тому, что третий элемент будет поставлен ниже двух предшествующих, поскольку принятые значения не могут допустить добавления 6 пикселов к `#attractSection`.

Решение подобной проблемы представляется весьма сложным. Если ширина #attractSection будет задана статичным количеством пикселов, то самым простым способом решения проблемы будет задать значения width и margin также в пикселах, а потом добавить рамки. А если значение #attractSection задано пропорционально и его ширина не статична, то дизайнеры могут пойти двумя путями:

- Опираясь на внутренние элементы.

  При таком сценарии элементы .promo, скорее всего, будут содержать заголовок и второй элемент для копии. Если аккуратно установить вертикальные поля и отступы этих парных элементов, то их блоки будут почти равны по высоте их контейнерам (меньше на 2 пиксела) и каждому элементу можно будет присвоить рамку в 1 пиксел с обеих сторон.
- Используя фоновые изображения.

  Вместо использования свойств CSS можно задать рамки с помощью фоновых изображений, с последующим позиционированием элемента .promo и его дочерних элементов.

Отступы

Самая большая ценность отступов для дизайна — это возможность создавать «канавки» — участки страницы, где виден только фоновый элемент (если он присутствует). Отступы редко доставляют проблемы дизайнерам, за исключением случаев смешения элементов, описанных выше, в применении к рамкам.

Кроме канавок, отступы имеют еще одно очень полезное свойство: они *не* схлопываются, если находятся в двух соприкасающихся блочных элементах. По этой причине объектам макета, например колонкам, часто присваиваются отступы, хотя на первый взгляд более верным решением кажется применение полей. При работе со сложными макетами проще визуализировать эффекты свойств, *не взаимодействующих* между собой, нежели тех, которые *взаимодействуют*.

Блочное поведение корневых элементов документа

Блочные свойства рамок браузера и элемента body проявляются несколько иначе, нежели аналогичные свойства других элементов, особенно выполняющих функции XML.

По умолчанию все браузеры при обработке text/html документов оставляют с каждой стороны по 10 пикселов свободного места — это значение задается для свойства margin элемента body.

В современных браузерах блочные свойства могут быть присвоены не только элементу body, но и html; большинство дизайнеров предпочитают обнулять их следующим образом:

```
html, body { margin: 0; padding: 0; }
```

В других случаях следует очень осторожно подходить к присвоению блочных параметров для body и html в силу целого ряда причин:

- блочные параметры элемента html не обрабатываются Internet Explorer 6;
- изменение блочных свойств элемента body не окажет никакого эффекта на фоновые цвета и рисунки документа, которые были применены к границам отображения браузера;
- изменение блочных свойств html изменит крайние координаты элемента body, что, в свою очередь, может привести к изменению положения всех элементов на странице.

Параметры блоков и процентное значение

Если вы зададите какие-либо размеры блока, кроме height, с помощью процентных значений, то результат будет вычислен относительно связанного элемента (или из родительского элемента, если речь идет о ширине).

Заданное в процентах значение height гораздо сложнее в применении, поскольку оно меняется на auto, за исключением тех случаев, когда один или несколько элементов выше в каскаде (как правило, непосредственный родительский элемент) имеют дискретное значение height.

Поток элементов

Вы можете изменить схему расположения элементов, задав для них один из трех типов поведения: inline, block и inline-block (рис. 6.5). Источником определения типов является спецификация HTML; она же устанавливает значения всех элементов по умолчанию.

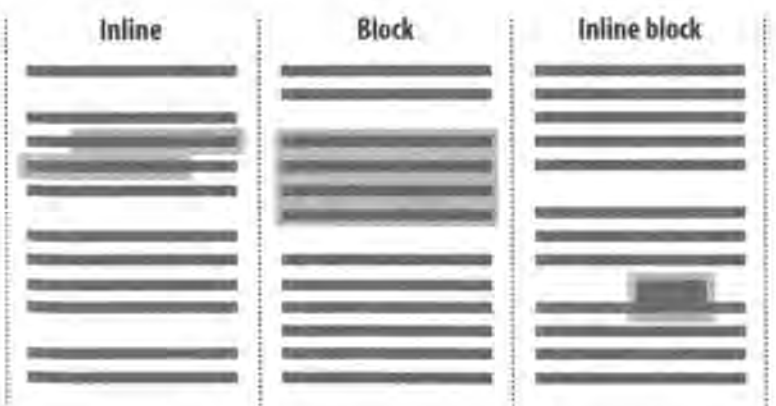


Рис. 6.5. Три основных типа потокового режима: inline, block и inline-block

В дальнейшем дизайнеры могут изменить потоковый режим или в некоторых случаях даже отключить его, используя свойства `float` или `position`.

Элементы inline

Строковые элементы, например `strong`, располагаются как обычный текст. Опорная линия является общей с соседним текстом, а разрывы строк применяются по умолчанию. В современных браузерах к строковым элементам можно добавлять поля, отступы и рамки. При этом они не оказывают никакого влияния на соседние элементы контента.

Самое главное, к строковым элементам нельзя добавлять никаких других свойств, влияющих на положение элемента, *кроме* перечисленных выше полей, отступов, рамок, а также позиционирования.

Текст, не включенный в `inline`-разметку, будет *вести себя* как строковый, но получит свойства стиля только своего родительского элемента.

Элементы block

Блочные элементы *по умолчанию* следуют четырем простым правилам.

1. Стремятся занять все доступное место по горизонтали в своих элементах — контейнерах.
2. Никогда не перекрывают и не перекрываются другими элементами, за исключением тех, которые содержат.
3. Содержимое блочного элемента должно целиком состоять из блочных, текстовых, строковых элементов или элементов inline-block. Смешение типов не допускается.
4. К блочным элементам можно применять любые свойства CSS.

Это последнее правило позволит вам при необходимости нарушить правила 1 и 2.

Третье правило дает возможность создавать так называемые «анонимные» ячейки контента, содержимое которых распространяется по ширине контейнера подобно блочным элементам, но при этом недоступно для CSS. По этой причине многие дизайнеры рекомендуют использовать нижеследующий код:

```
<div>
   <p>Lorem ipsum dolor sit amet.</p>
   Consectetur adipiscing elit.
</div>
```

Сущность «ремонта» заключается в том, что элемент помещается в простой блок, что делает его доступным для CSS.

Для строковых элементов нет необходимости проводить подобный «ремонт», кроме случаев, когда изменяется стиль на каком-либо участке внутри элемента.

Элементы inline-block

Многие элементы `inline-block` называют в технической литературе Консорциума Всемирной паутины «замещаемыми», поскольку они часто включают в себя изображения, элементы управления, а также другие объекты, которые обрабатываются при поддержке операционной системы, а не браузера.

Элементы `inline-block` приобретают потоковые характеристики строковых элементов. Это означает, что они будут выстраиваться по общей опорной линии. Стоит обратить внимание, что пустое пространство вокруг и внутри элементов `inline-block` отображается в границах браузера (а не в границах элемента `body`), так же как у строковых элементов. Несмотря на это, к элементам `inline-block`, как и к блочным элементам, можно применять все свойства CSS.

Особенности отображения потока элементов `inline-block` будут описаны далее вместе с обсуждением свойства `display`.

Использование display для изменения потока элементов

Свойство `display` принимает диапазон значений, соответствующих типам потоковых элементов, описанных выше, а также значение `none`. Результатом применения этого свойства могут быть многие полезные эффекты, подробно представленные на сопутствующем сайте книги.

- Первичные навигационные ссылки на сайтах-«брошюрах» чаще всего принимают горизонтальную ориентацию. Это достигается путем изменения значения `display` для списка ссылок на `block` (таким образом устраняется неясность в определениях типа документа в HTML), а также применением различных свойств CSS, особенно `float`. В случаях, когда навигационные ссылки следует изменить статически или пропорционально, подобное решение в сочетании с присвоением `display: block` гиперссылкам кажется более удобным, нежели применение `#nav li { display: inline; }`.
- После присвоения значения `display`, равного `block`, ссылкам можно будет назначить произвольные значения `width` и `height`, что упростит их компоновку в интерфейсе веб-приложений. Этот прием также позволит увеличить «зону обслуживания» ссылки, на практике применяя правило взаимодействия человек—компьютер, более известное как закон Фиттса. Этот закон гласит, что более крупный объект интерфейса удобнее для активации пользователем, нежели более мелкий.
- Способ, в соответствии с которым расположены упорядоченные и неупорядоченные списки, может быть изменен таким образом, чтобы списки отображались последовательно.
- Изменяя значение `display` для элементов управления на `block` можно добиться их расположения вдоль опорной сетки.

- Значение `display: none` можно применять для упорядочивания шаблонов в тех случаях, когда невозможно удалить все элементы, а также для сохранения читабельности разметки.
- Нескольким аналогичным текстовым фрагментам или строковым элементам, которые должны следовать общей базовой линии, можно присвоить значение `display: inline-block`. Это даст дизайнеру двойное преимущество: общую базовую линию и возможность применять свойства CSS.

Свойство display

Самые важные свойства `display` связаны со значением `none`.

- Элементы со значением `none` обрабатываются графическими агентами так, как будто их попросту не существует.
- Подобная «невидимость», создаваемая `display: none`, применяется в области вспомогательных технологий.

Применение значения `inline-block` также может привести к непредвиденным последствиям. Поскольку элементы `inline-block` имеют потоковые свойства `inline`, то пустое пространство между элементами `inline-block` и их «неблочными» соседями отображаются в границах браузера, а потому невозможно выровнять эти элементы без дополнительного регулирования исходного кода. А такое регулирование, в свою очередь, нарушает принципы разделения макетов, природа и преимущества которого обсуждались в главах 2 и 3. Вдобавок потоковые свойства `inline-block` элементов могут привести к появлению разрывов в контенте в случае, если эти элементы имеют различную ширину или если размер текста задан статично.

И наконец, многие старые версии браузеров просто не поддерживают (или поддерживают некорректно) значение `display`, равное `inline-block`.

Свойства float и clear

Эффект, вызываемый свойством `float`, не так-то просто описать. Элемент, к которому применено данное свойство, получает рамку, а последующий контент *обтекает* этот элемент, вместо того чтобы помещаться ниже. Свойство `clear`, наоборот, отменяет эффект обтекания. Визуально эти свойства отражены на рис. 6.6.

Но это только теория, а на практике все обстоит несколько иначе. Поскольку `float` — единственное свойство, позволяющее создать страницу с разным размером колонок в CSS 2.1, знание особенностей и тонкостей этого свойства необходимо для всех дизайнеров.

Рис. 6.6. Демонстрация свойств `float` и `clear`: 1 — значение `float`, равное `left`; 2 — значение `float`, равное `none`; 3 — свойство `clear` со значением `left`

Правила свойства float

Для того чтобы предсказать поведение элементов, к которым было применено свойство `float`, необходимо понимать правила, которым следуют рендеринговые движки.

Более подробно эти правила описаны в секциях 9 и 10 спецификации CSS 2.1. Здесь же мы вкратце опишем эти правила.

Элемент со значением `float:left` или `float: right` должен:

- иметь явно установленную ширину (выраженную или подразумевающуюся);
- находиться целиком в контенте, содержащемся в контейнере, если только он не шире и не выше этого контейнера (и в таком случае свойство будет влиять на расположение других элементов документа);
- не пересекаться более чем одной линией с другими элементами, не имеющими свойства `float` и предшествующими данному элементу в разметке. Также это справедливо, если соседним элементам присвоены дополнительные значения `float`;
- «прикрепляться» сначала к максимально возможной верхней линии, затем к самой левой (или правой);
- быть смежным с блоками элементов, не имеющих значения `float`, которые следуют в разметке после этого элемента, но *не* с их *контентом*. Это особенно важно при создании нескольких колонок.

Понятие элемента-контейнера является здесь важным, поскольку такие элементы могут иметь свое собственное значение `float`; об этом рассказывается в следующем разделе.

`Float: none` применяется, например, в случае, если нужно отменить значение, присвоенное ранее в таблице стилей. Таким способом можно избежать хаоса в макете, возникающего, если элементы с `float` слишком большие или слишком маленькие и их сложно подогнать к дизайну страницы.

Эти правила также поясняет рис. 11.7 в главе 11.

Более подробно об этих правилах можно прочитать на сопутствующем сайте книги.

Отмена значений float с помощью clear

При применении свойства `clear` элемент не обтекает своего предшественника, а помещается ниже него.

Наиболее эффективный способ применения данного свойства — выравнивание полей с полями предшествующего элемента (табл. 6.1).

Таблица 6.1. Значения свойства clear и результат их применения

Значение	Результат
`none`	Значение по умолчанию. Элемент обтекает предшествующий элемент, согласно заданным правилам
`left`	Элемент помещается после предшествующего элемента, имеющего значение `float: left`
`right`	Аналогично предыдущему, но для значения `float: right`
`both`	Элемент помещается ниже всех элементов, имеющих свойство `float`. Поля изменяются соответственно

Вскоре мы опять вернемся к свойствам `float` и `clear` и рассмотрим их применение на примере макета с несколькими колонками.

Контекст float

Как и многие другие свойства, `float` и `clear` применяются в диапазоне, определяемом присутствием значений `float` в каскаде.

Создание двух колонок, «обернутых» элементом со свойством `float`, будет также обсуждаться далее. Для достижения подобного эффекта можно поместить две «обернутые» колонки в окружение элемента со свойством `float`, заданным его родительским элементом, что приведет к визуальному изменению границ элементов, к которым применены свойства `float` и `clear`. При работе с тем же самым макетом, использующим три колонки и четыре контейнера, это приведет к тому, что присвоение `clear: both` элементу внутри элемента, «обернутого» вокруг двух колонок, сместит поля контента к полям «обертки», а не к контейнеру страницы.

Создание макета с несколькими колонками

Макеты с двумя и тремя колонками, которые мы часто встречаем в Интернете, а также можем наблюдать на рис. 6.7, — в известной степени результат инерции. До того момента, как браузеры стали поддерживать CSS, у дизайнеров практически не было средств для изменения внешнего облика страницы, за исключением табличной разметки, flash или изображений с локализованными ссылками.

Рис. 6.7. Исходный порядок этих элементов: `#main`–`#header`–`#bodyCopy`–`#sidebar`–`#footer`; к каждому из них применены свойства `float` или `clear`

Упрощенная разметка страницы выглядит приблизительно так:

```
<div align="center">
<table width="768">
  <tr>
    <td colspan="2">Это верхний колонтитул.</td>
  </tr>
  <tr>
    <td width="160">Это сайдбар.</td>
    <td>Это основное содержание.</td>
  </tr>
  <tr>
    <td colspan="2">Это нижний колонтитул.</td>
  </tr>
</table>
</div>
```

Конечно, на практике страницы имеют гораздо более сложную структуру. Они могут включать дополнительные строки и ячейки, заполняемые дополнительным контентом и предназначенные для создания отрицательного пространства в макете страницы. Например, в начале 2002 года я создал страницу, предназначенную для почтового клиента. Эта страница содержала около 40 колонок, при этом ни одна из секций макета не имела более четырех физических колонок. Подобное достигалось с помощью правил и изменения сетки макета, проводившегося последовательно от первой секции к последующим.

Создателям сайтов, работающим с почтовыми клиентами, до сих пор приходится создавать подобные сверхсложные конструкции, поскольку поддержка CCS в популярных почтовых клиентах остается на удивление слабой и улучшений в этом направлении не предвидится.

Конвертация двухколоночных макетов из разметки в CSS

Если вы попробуете самостоятельно создать многоколоночный макет, то быстро убедитесь, как много причин может привести к неудаче.

- Присутствие значений `float` и `width` во всех колонках инициирует связи между исходным порядком и порядком отображения. Также этот прием приводит к увеличению риска появления разрывов, особенно в Internet Explorer 6.
- Применение `position: absolute` уменьшает риск разрыва, но, чтобы этого добиться, вы должны заранее знать относительную длину ваших колонок, что на самом деле практически невозможно.

Для решения первой проблемы рекомендуется помещать значения `float` только в те колонки, которые *должны* обтекаться, а оставшиеся колонки контролировать с помощью свойств полей. Для них свойства позиционирования имеют весьма ограниченное применение, ведь вы скорее будете применять их к навигационным ссылкам, а не к колонкам, содержащим контент.

Переход к CSS в простых случаях, подобных приведенному ниже, на первый взгляд приводит к выполнению лишней работы: меняются имена элементов, вы используете меньшее их количество, а взамен добавляете целую кучу правил CSS!

Вот пример разметки, который также можно найти на сопутствующем сайте книги:

```
<div id="header">Это верхний колонтитул.</div>

<div id="main">

    <div id="bodycopy">Это основное содержание.</div>
    <div id="sidebar">Это сайдбар.</div>

</div>

<div id="footer">Это нижний колонтитул.</div>
```

а это таблица стилей:

```
                #main { height: 1%; overflow: auto; }
#main, #header, #footer { width: 768px; margin: auto; }
            #bodycopy { float: right; width: 598px; }
             #sidebar { margin-right: 608px; }
              #footer { clear: both; }
```

Схематическое представление этой разметки можно увидеть на рис. 6.8.

Как работают двухколоночные стили

Судя по первому правилу, в приведенной выше разметке допускается использование макета фиксированной ширины, поскольку большинство табличных макетов имеют фиксированную ширину. Применение свойства `margin: auto` к трем элементам-контейнерам гарантирует, что эти элементы будут отцентрированы по границам браузера.

Основному контенту присвоено свойство `id: bodycopy`. Поскольку это первый элемент в `#main`, то он получает необходимые значения `float` и `width`. Также вы можете заметить, что здесь применено описанное выше свойство `overflow: auto`, однако оно необходимо только в случае, если непосредственно к `#main` были применены какие-либо свойства фона.

Затем следует второстепенный контент (метаданные), которому присвоена метка `#sidebar`. Для определения его ширины использовано значение `auto`, поскольку к нему не применено свойство `float`, а контроль ширины достигается с помощью `margin-right`. Подобный прием более предпочтительный, в силу удобства, которое обеспечивают схлопывающиеся поля. Если вместо этого значения элементов макета будут заданы с помощью непостоянных единиц измерения, например em, то двоякое значение `width` сделает макет более чувствительным к округленным значениям в вычисленной ширине, что в свою очередь повышает риск разрывов. Установка же значения `margin-right` грозит незначительным изменением ширины контента, что также нежелательно, но менее неприятно.

Наконец, назначение `clear: both` для элемента `#footer` является излишним из-за применения `overflow: auto` к элементу `#main`, но такое положение дел мгновенно меняется после того, как значение `overflow` удаляется из таблицы стилей. В свою очередь, нижний колонтитул `#main` заканчивается на нижнем краю `#sidebar`. Применение `clear: both` к элементу `#footer` приводит к тому, что более поздние элементы будут отображаться ниже обеих колонок `#main`, *независимо от того, какая колонка выше*. Этот эффект вы можете наблюдать на рис. 6.8.

В стилях, примененных ранее, остаются нерешенными две проблемы. Первая из них касается навигации, которая, конечно, будет присутствовать на странице. По всей вероятности, она должна находиться в элементах `#header` или `#sidebar`, но более прогрессивным методом будет ее включение в исходном

порядке между #main и #footer. Однако это потребует применения дополнительных элементов позиционирования – вертикальных полей (или присвоения значения padding-top для #sidebar, если навигация должна находиться в верхней части этой колонки).

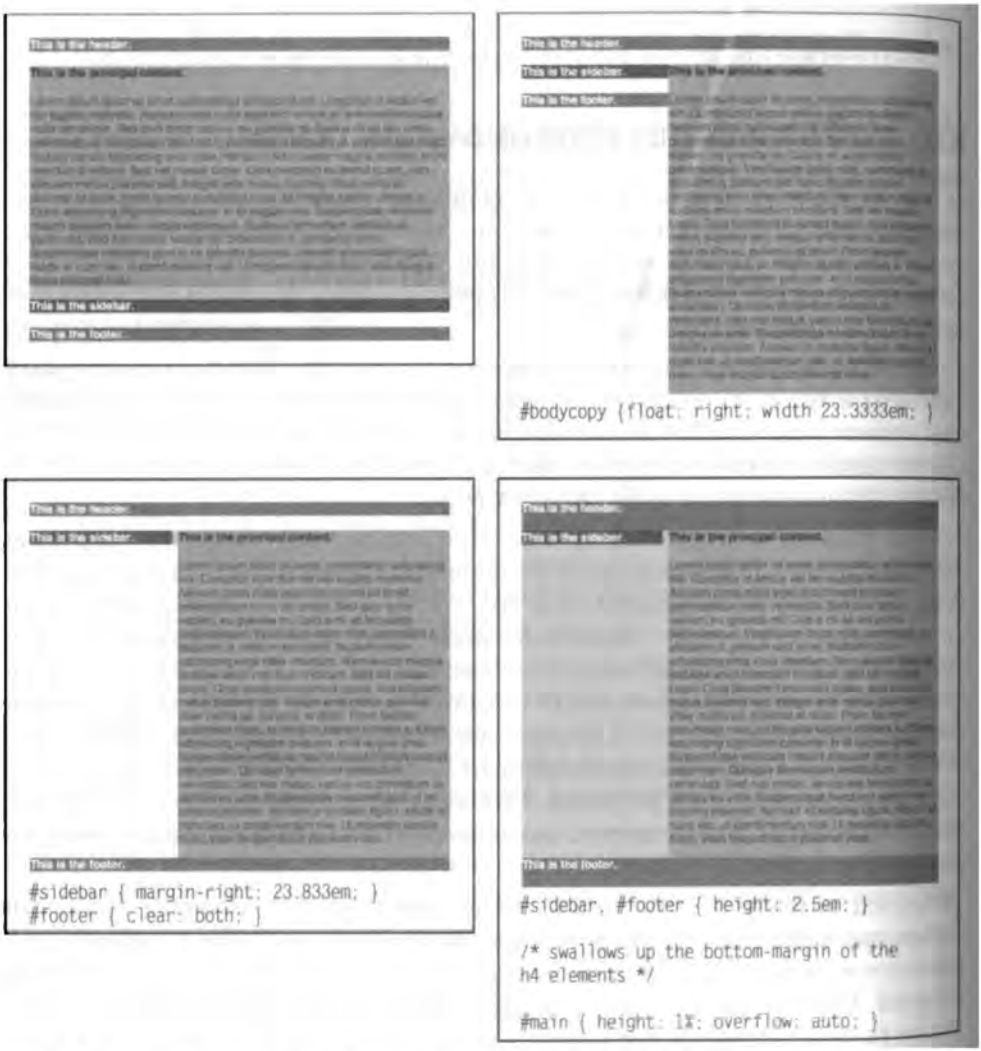


Рис. 6.8. Кумулятивный эффект стилевых правил, примененных к двухколоночной разметке. Каждый из фрагментов иллюстрирует применение одного правила

Вторая проблема касается наличия отрицательных полей поблизости от #footer, особенно если второстепенная навигация располагается снаружи элемента #footer. На практике области, граничащие с внутренними краями #header и #footer, часто переполнены различными полями, отступами и т. п.

Поэтому часто бывает необходимо проводить точную регулировку значений полей, связанных с этими областями макета. Подобный процесс чаще всего приводит к появлению как минимум одного отрицательного значения поля.

Еще одним вариантом будет применение противоположных значений `float` к двум колонкам, а также `overflow: auto` и `height: 1%` к `#bodycopy`. Однако подробно обсуждать этот вариант не будем, поскольку при его применении возможно возникновение ошибок в Internet Explorer 6.

Преимущества использования CSS для создания макетов

Описанные выше взаимосвязи и многочисленные проблемы, возникающие в процессе внедрения, могут вызвать у вас вопросы в целесообразности применения CSS для создания макетов.

Целесообразность применения заключается в упрощении процесса, как это ни парадоксально звучит. Усилия, затраченные на создание таблиц стилей, будут вознаграждены упрощением разметки, что, в свою очередь, создает ряд преимуществ.

- Контент расположен в логическом порядке.

  Из-за прямой взаимосвязи между исходным порядком контента в `table` и его положением на странице боковая панель и панель навигации (обычно визуально находящаяся в верхнем колонтитуле или в боковой панели) должны находиться в верхней части разметки. Следом за этим должно идти основное содержимое страницы — то, что кратко описывают поисковые движки, и то, что является основной целью посетителя страницы. Благодаря широкому распространению табличных макетов пользователи вспомогательных платформ ожидают, что панель навигации будет находиться вверху страницы, поскольку их программы отображают контент в исходном порядке.

- Элементы удаляются или добавляются из контента `body` в шаблонах, только если их действительно необходимо удалить или добавить.

  Чем реже вносятся изменения, тем меньше требуется проводить проверок на соответствие документов. Кроме того, это упрощает поддержку сайта при использовании его в течение длительного периода времени.

- Упрощенная разметка увеличивает совместимость и доступность ресурса.

  Самым очевидным подтверждением этого факта является вывод страницы на печать. Если вам требуется добавить графический элемент «Напечатать эту страницу», то документ на противоположном конце этой ссылки может использовать точно такой же шаблон, как и документ, предназначенный для вывода на экран. При этом нужно будет убрать некоторые элементы, относящиеся к навигации, что можно очень быстро сделать с помощью `display: none` (еще удобнее включить разметку навигации в отдельный шаблон, который можно при необходимости отключить).

- Хорошо это или плохо, но проблемы отображения можно решать с помощью введения дополнительных элементов `class` и `id` в шаблоны разметки и код.

  Благодаря использованию большого количества элементов `class` и `id` разработчики получат возможность создавать меньшее количество шаблонов или их фрагментов для каждого сайта.

- Отделение презентационного макета от шаблона многократно упростит проведение реконструкций сайта.

  Если макету сайта понадобится новая секция, то ее можно довольно быстро добавить в шаблон, а затем «навести красоту» с помощью таблиц стилей. Если же понадобится удалить какой-либо блок, то при наличии хорошо составленных стилевых правил вам будет достаточно просто закомментировать необходимый блок в шаблоне. Сравните эти простые действия с огромным объемом работы, который вам бы предстояло выполнить при каких-либо изменениях в структуре шаблонов, созданных на основе таблиц. Какой вариант вам кажется наиболее выгодным?

В дополнение к этим преимуществам можно с большой долей уверенности сказать, что уменьшение количества разметки по отношению к контенту (последствие внедрения грамотно написанных CSS) упрощает работу тем, кто использует поисковую оптимизацию (SEO). Конечно, в отсутствие возможности сравнить два одинаковых документа утверждать этого однозначно нельзя. Однако желание множества весьма искушенных в компьютерных технологиях людей верить в преимущества более простой разметки должны что-то значить, не так ли?

Переход с двух колонок на три

Двухколоночные макеты весьма просты в создании: при условии что у вас не очень сложно организован фон. Все, что вам нужно, — это задать значение `float` для одного элемента, поля для другого, а также поместить колонтитул в нижней части страницы. Есть и другой способ — применить значение `float` к обеим колонкам.

Примененную технологию можно охарактеризовать как комбинацию фальшивых колонок и значения `clear: both`, примененного к нижнему колонтитулу.

При переходе к трехколоночному макету количество возможных комбинаций `float/margin` возрастает до шести, а также добавляется необходимость поместить две из этих колонок в пустой контейнер, для того чтобы контролировать их ширину. Скорее всего, вы захотите поместить в контейнер две первые колонки (в соответствии с исходным порядком), однако в двух из шести возможных случаев это будет не просто сделать, поскольку эти колонки не будут смежными в макете. Существует шесть различных способов создания трехколоночного макета. Они показаны на рис. 6.9 и описаны в табл. 6.2.

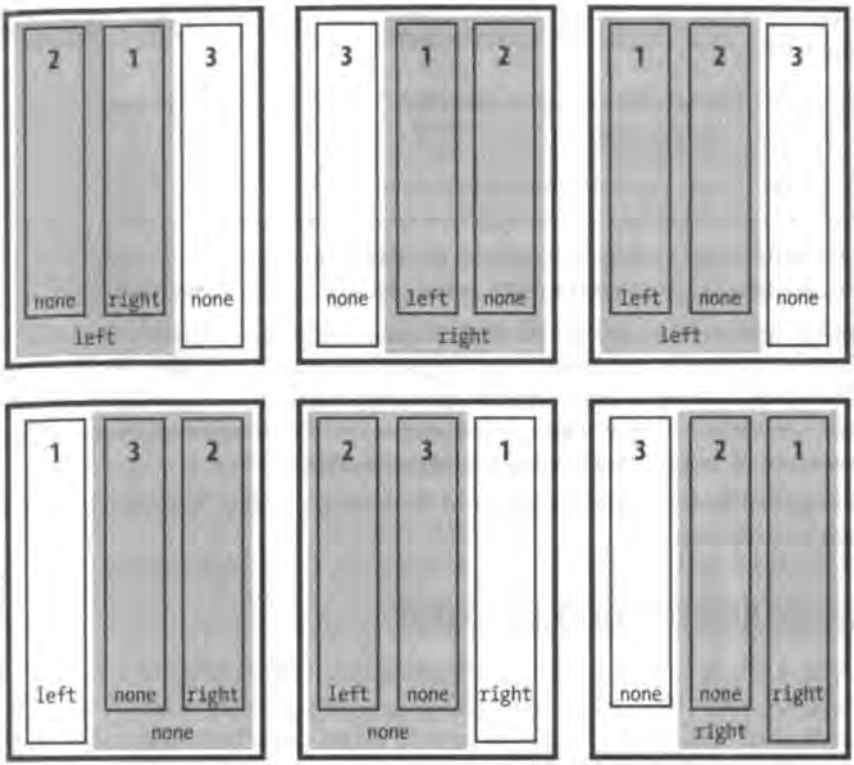


Рис. 6.9. Способы создания трехколоночного макета

Таблица 6.2. Порядок колонок (слева направо) и значения float/margin. Колонки пронумерованы в соответствии с исходным порядком. Квадратные скобки указывают на элемент-контейнер

Порядок	Параметры контейнера	Параметры колонок		
		Колонка 1	Колонка 2	Колонка 3
[[2, 1], 3]	float: left;	float: right;	margin-right: x;b	margin-left: x;
[3, [1, 2]]	float: right;	float: left;	margin-left: x;	margin-right: x;
[[1, 2], 3]	float: left;	float: left;	margin-left: x;	margin-left: x;
[1, [3, 2]]	margin-left: x;	float: left;	float: right;	margin-right: x;
[[2, 3], 1]	margin-right: x;	float: right;	float: left;	margin-left: x;
[3, [2, 1]]	float: right;	float: right;	margin-right: x;	margin-right: x;

Примечания:

а) помните, что элементы, которым присвоено значение `float`, должны иметь соответствующее значение `width`, а для элементов `margin-*` значение `width` можно опустить;

б) x здесь и далее указывает на переменную длину, а не на ту, которая остается неизменной в данном макете.

Для того чтобы создать контейнер для двух колонок, вам нужно выполнить следующие действия.

1. Переместите контейнер так, чтобы он охватывал эти две колонки.
2. Отмените свойство `width` для контейнера.
3. Установите соответствующие значения `float` и `width` для двух первых колонок.
4. Присвойте третьей колонке необходимые значения `margin` так, чтобы они были больше или равны сумме значений `width`, указанных в пункте 3.

Ниже представлены два способа, позволяющие избежать пересечения основного контента и нижних колонтитулов. Первый вариант — использовать `overflow: auto` в комбинации с `height: 1%` для элемента, содержащего колонки. Второй вариант — `clear: both` для нижнего колонтитула. Оба способа можно применять независимо от количества колонок в вашем шаблоне.

Стили и шаблоны для шести макетов, описанных в табл. 6.2, также приведены на сопутствующем сайте книги.

Если колонок больше трех

При росте числа колонок в макете удобнее всего использовать `overflow: auto;` для одного или нескольких контейнеров. Использование нескольких последовательных значений `float` — и вместе с тем риск возникновения проблем вследствие различных вариантов закругления или ошибок Internet Explorer — остается целесообразным, поскольку, когда количество колонок в макете становится больше трех, приемлемым считается *любой* удачный подход к решению проблем.

Семантически пустые контейнеры для многоколоночных макетов

Если вы проведете свое собственное независимое исследование того, каких результатов можно добиться, применяя множественные значения `float` и позиционирование, то увидите, что многие из описанных здесь макетов не требуют наличия контейнера для нескольких колонок. (Если вы не хотите сами этим заниматься, зайдите на сопутствующий сайт книги.)

Создание пустых контейнеров часто приводит к внедрению ненужной разметки, что, конечно, не добавляет плюсов странице. Это также справедливо, если вы используете один из трехколоночных шаблонов, описанных ранее. Конечно, вы можете провести необходимые замены в шаблоне, так чтобы основной контент переместился с края макета в центр (шаг, считающийся с точки зрения CSS-дзен совершенно излишним), однако контейнер обеспечивает лучшую защиту от разрывов и более удобен, если вы собираетесь устанавливать отдельный фон для всех трех колонок.

Дополнительные функции в CSS3

В текущую рабочую версию CSS3 для удобства дизайнеров добавлены специальные модули, с помощью которых можно легко создать макеты с несколькими колонками. Кроме того, эти модули снабжены инструментарием, например column-span, позволяющим разбить элемент на несколько колонок, имеющих произвольную ширину, высоту и информационное наполнение, а также ширину канавки.

Модуль шаблонов для макетов, как уже было написано, определяет расширение для свойств display и position, что позволяет дизайнерам создавать сетку макета, повторяющую поведение табличных макетов (таких, которые описаны в разделе «Создание макета с несколькими колонками» на с. 113).

Расширение display поддерживает переменные, закодированные в пределах буквенного подмножества ASCII, а также две константы. Первая константа «@» указывает на то, что элементы должны отображаться так же, как в контексте шаблона, если для них явно не указано расположение в сетке. Вторая константа «.» (точка; %2E) определяет канавки в макете.

JavaScript-фреймворк jQuery включает в себя модуль, позволяющий стилистам эмулировать поведение модуля, определяя метку-префикс для заданных пользователем свойств display и position (например, -mygrid-display), а также имя файла CSS, содержащего правила макета.

Свойства позиционирования в CSS

Свойство position принимает одно из четырех возможных значений (static — значение по умолчанию) и дает стилисту возможность поместить элемент в любое место макета. Более важно то, что присвоение элементу любого значения position, кроме static, изменяет позиционный контекст элемента, о чем кратко упоминалось в главе 3.

Как работает позиционирование

Рассмотрим пример:

```
#someDiv { ... left: 160px. top: 96px: }
```

Допустим, что это правило применено к нижеследующей разметке:

```
.. <body><div id="main"> .. <div id="someDiv">The quick
red fox jumps over the lazy brown dog.</div> ... </div></body>
```

Четыре значения position, которые вы можете применить к #someDiv, дадут следующие результаты:

- static

  Значения left и top будут проигнорированы; элемент сохранит свое место в потоке документа.

- `absolute`

  Верхний левый угол `#someDiv` сдвинется на 96 пикселов вниз и на 160 пикселов вправо от границы браузера. Поля, примененные к `body`, игнорируются. Элемент удаляется из потока документа.

- `fixed`

  Результат будет аналогичным применению `position: absolute`, за исключением того, что элемент сохранит свою позицию независимо от скроллинга контента. Элемент удаляется из потока документа.

- `relative`

  Вместо того чтобы сместиться от верхнего левого угла браузера, `someDiv` будет смещаться относительно своего положения в макете. Элемент сохраняет свое положение в потоке документа независимо от изменений `left` и `top`, инициированных таблицей стилей.

Давайте добавим еще одно правило:

```
#main {position: relative; margin: 20px;}
```

Значения `top`, `right`, `bottom` и `left` для `#main` по умолчанию равны нулю, но присвоение `position: relative` изменило позиционный контекст `#someDiv`, поэтому вместо того, чтобы смещаться от левого верхнего угла браузера, `#someDiv` будет смещаться от верхнего левого угла `#main` на 20 пикселов вниз и вправо по сравнению с первым примером, как показано на правом верхнем фрагменте рис. 6.10. Раскадровка рисунка основана на разметке и стилевых правилах, используемых для демонстрации двухколоночного макета.

Теперь немного о воздействии эффекта позиционирования на поток окружающих элементов. Элемент с присвоенным значением `display: none` имеет такие же эффекты, как удаленный элемент, за исключением того, что он остается в коде. Сохранение потокового поведения означает, что если положения элемента можно изменить (при применении `position: relative`), то это окажет воздействие и на соседние элементы потока, в соответствии с присвоенными им значениями `display` (см. раздел «Конвертация двухколоночных макетов из разметки в CSS» на с. 114).

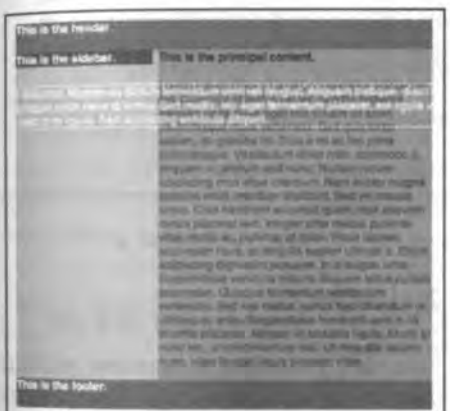
Наконец, следует помнить, что при использовании `position: relative` для восстановления позиционного контекста (как при создании панели навигации в примере, о котором говорится далее) версии Internet Explorer 6 и 7 обрабатывают поля не так, как остальные браузеры. Для предотвращения этого требуется восстановление значений в таблицах стилей.

Связанные элементы позиционирования

Свойства `top` и `left`, которые могут быть установлены с теми же самыми единицами измерения, как свойства блока, обычно связаны со свойствами `right` и `bottom`, применяемыми гораздо реже из-за непредсказуемости границ браузера.

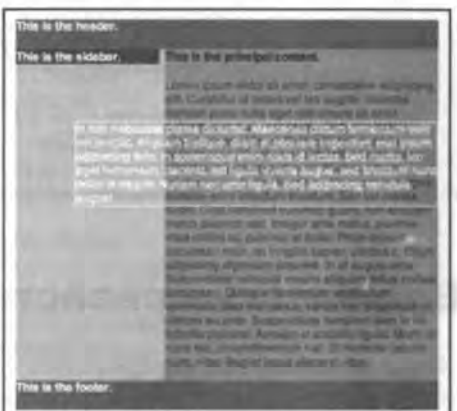

```
#sidebar p { position: static; }
```


```
#sidebar p { position: relative; }
```




```
#sidebar p { position: absolute; }

/* this value removes it from the 'normal'
flow, and it expands because it doesn't
have a parent with nondefault position and
width values */
```




```
#main { position: relative; left: 0; top: 0; }

/* changes the position context of all of
main's child elements */

#sidebar p { position: absolute; }

/* the origin of this element's positioning
grid has changed to the upper-left corner
of #main */
```

Рис. 6.10. Раскадровка, иллюстрирующая поведение элементов и свойств

Однако во всех современных браузерах можно присваивать дополнительные значения вместо назначений `width` и реже `height`. Рассмотрим следующий пример:

```
#someDiv { position: absolute; left: 25%; right: 25%; }
```

Результат которого будет аналогичен:

```
#someDiv { position: absolute; left: 25%; width: 50%; }
```

Подобная техника может быть полезной при создании макетов с переменной шириной, которые требуют установки полей в числовых, а не в процентных значениях.

Как и в случае с `top` и `left`. отрицательные значения `right` и `bottom` перемещают соответствующие поля вне полей элемента, устанавливающего позиционный контекст (см. рис. 6.11).

WIDTH: AUTO И ЗНАЧЕНИЯ POSITION/FLOAT

В отсутствие присвоенных значений `width`, элементы со значением `position`, равным `absolute` или `static`, будут расширяться по горизонтали и занимать все возможное место, точно так же как обычные элементы. Это означает, что отмена назначенного значения `left` или `right` будет иметь функциональное значение, такое же, как ноль.

Все становится гораздо интереснее, если элемент, имеющий абсолютную позицию или обладающий свойством `float`, содержит контент с собственными размерами (например, изображения или элементы управления). В этих случаях контейнер, имеющий значение `width` равное `auto`, будет подгоняться под контент.

Секция 10.3 спецификации CSS 2.1 приводит очень подробную информацию о вычислении ширины элементов.

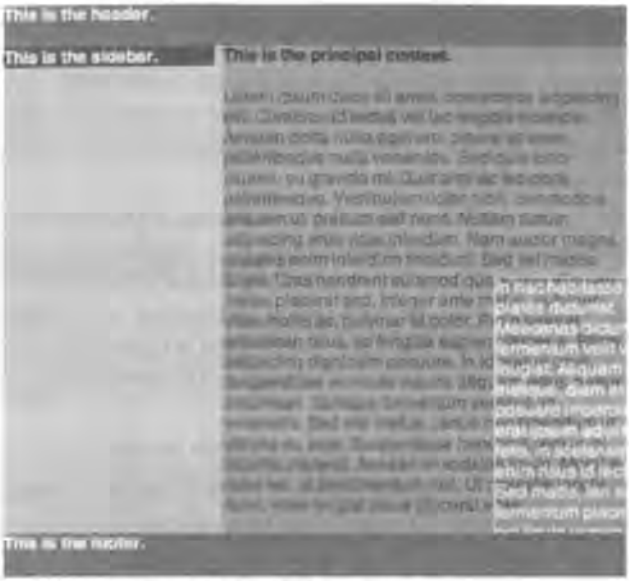
Видимость и свойство z-index

Помимо множества инструментов (для работы с блоками и задания позиционирования), позволяющих управлять макетом страницы в двух измерениях, существует два свойства, дающие возможность управлять макетом в трех измерениях.

Изменение видимости, не затрагивающее поток документов

Свойство `display: none` чрезвычайно полезно, но затрагивает поток документов, в которых используется. Свойство `visibility` и его значение `hidden` дает возможность убрать контент из вида, не затронув при этом поток документа. В то время как применение `display: none` заставляет обрабатывать контент так, будто элемента не существует, `visibility: hidden` просто убирает контент из вида, отображая пустое место между соседними блоками.

Применение `visibility: hidden` и `opacity: 0` дает одинаковые результаты. Разница заключается в поддержке данных свойств. Свойство `visibility` является частью оригинальной спецификации CSS, созданной в 1996 году, а поддержка `opacity`, входящей в спецификацию CSS3, появилась совсем недавно.




```
#main { position: relative; left: 0; top: 0;
  overflow: hidden;}

/* the height of #main is inherited from the auto height of body
   - changing overflow from auto to hidden removes scroll bars */

#sidebar p { position: absolute; right: -2.5em;
  bottom: -7.5em; width: 10em; }
```

Рис. 6.11. Отрицательные значения `right` и `bottom` перемещают элементы внутрь их заданной по умолчанию позиции

Стек

Если два элемента пересекаются — например, когда один из них содержит второй, — более поздний элемент в исходном порядке будет отображен поверх своего предшественника. Все бы ничего, но ситуация осложняется благодаря природе позиционирования и тому факту, что любой элемент содержит как минимум два своих макета.

Для стандартного блочного элемента стек отображения будет выглядеть следующим образом (снизу вверх):

1) цвет фона;
2) фоновое изображение;
3) рамки;
4) строковый контент;
5) обтекаемый контент.

Строковые элементы, а также элементы, обладающие свойством `float`, могут иметь свой собственный фон, границы и контент, и за счет этого свой собственный стек. Элементы, значение `position` которых отличается от `static`, имеют свой собственный стековый *контекст*. Если два элемента со свойствами позиционирования имеют один и тот же стековый контекст, то их порядок в стеке определяется исходным порядком.

Свойство `z-index` предназначено для управления порядком стека. Предположим, у вас есть несколько позиционированных элементов с одинаковым позиционным контекстом. Если эти элементы будут пересекаться, то элемент, видимый в месте пересечения элементов, будет по умолчанию последним в исходном порядке. И если порядок элементов в стеке надо поменять, то на помощь приходит свойство `z-index`. Согласно его значениям, элементы будут отображаться в следующем порядке.

- Элементы, которым присвоены отрицательные значения `z-index`.
- Элементы со значением `z-index` равным нулю или `auto`. Порядок элементов с одинаковыми значениями определяется исходным порядком.
- Элементы со значением `z-index` больше нуля.

Элементы с идентичным позиционным контекстом и идентичными значениями `z-index` образуют стек согласно исходному порядку; тот элемент, который идет раньше согласно исходному порядку, находится ниже в стеке.

Наиболее значительным ограничением свойства `z-index` является то, что элементы должны получить общий позиционный контекст, прежде чем их можно будет разместить согласно воображаемой оси Z документа.

Помните, что если вам понадобится поместить в стеке «нормальный» контент поверх своего соседа, имеющего свойство `float`, то вы можете дать ему самое высокое положение в стеке, применив к элементу свойство `position` со значением `relative`.

Создание точного кода и макета для навигации

В разделе «Оформление навигации» на с. 146 описывается разметка, стилевые правила, а также приводится список шагов, необходимых для создания элементов основной панели навигации сайта. Шаг 5 не содежит большого количества подробностей, однако легко предположить, что создание элементов навигации потребует использования множества разнообразных свойств.

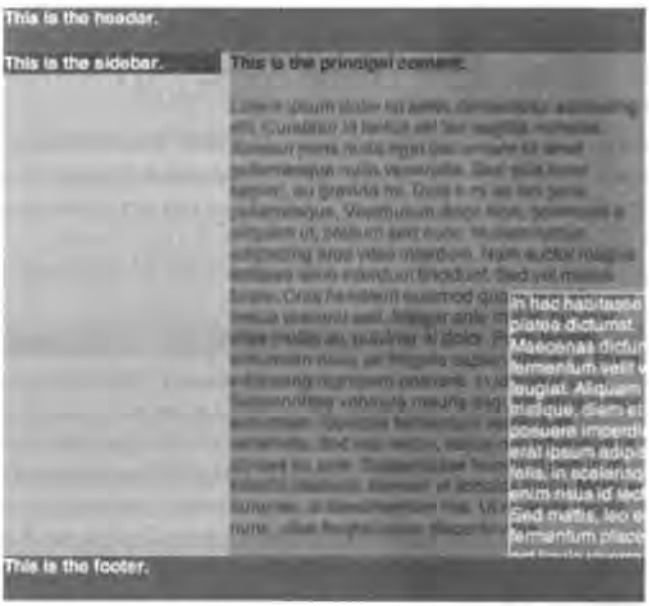
Все селекторы, используемые в этом разделе, преднамеренно написаны наиболее универсальным способом.

Следующий раздел написан в расчете на то, что первые четыре шага процесса привели к созданию списка, все элементы которого имеют тип `display: block`.

Создание макета страницы также будет подробно рассмотрено в главе 13.

Ориентирование списка

Независимо от того, где будет расположена первичная навигация в шаблоне, для упрощения процесса рекомендуется вначале разобраться с ее ориентацией на странице (рис. 6.12).




```
/* for the sake of illustration the positioning of
   #sidebar p has been removed */

#nav { position: absolute; left: .083em; top: 0;
margin: 0; padding: 0; list-style-type: none;
border-top-width: .083em; border-left-width: .083em; }

#nav li { display: block; float: left; width: 8.896em; }

#nav a { display: block; width: 6.813em; height: 2.333em;
line-height: 2.3333=em; padding: 0 lem 0 lem;
border-right-width: .083em; border-bottom-width: .083em;
background-color: #fff; }
```

Рис. 6.12. Горизонтально ориентированная навигация

На ориентацию навигационного списка могут повлиять три причины:

- общая вертикальная или горизонтальная ориентация;
- наличие или отсутствие подуровней;
- ориентация элементов подуровня.

Создание горизонтальной ориентации — чаще всего встречающейся в навигации на англоязычных коммерческих сайтах — проще всего достигается с помощью применения стилевых правил, аналогичных приведенному ниже:

```
    #nav { display: block; height: 1.5em; overflow: hidden; }
#nav li { width: 8.333em; float: left; overflow: hidden; }
```

Результат показан на рис. 6.12. Обратите внимание, что `#nav` получает свой контекст позиционирования от `#main`, и это позволяет ему находиться в исходном порядке ниже основного контента.

Пространство, на котором находится `#nav`, образовано отступами, добавленными к `#main`. Эти отступы обозначены серым пространством с каждой стороны. Данное пространство необходимо, чтобы учесть различные значения закругления у разных браузеров.

Если использование `float` невозможно по причине разницы закругления в различных браузерах или ошибок рендеринга в Internet Explorer, то рекомендуется присвоить списку параметр `position: relative` (за исключением случая, если `position: absolute` было присвоено для того, чтобы обойти исходный порядок), затем присвоить `position: absolute` вместе с желаемыми координатами каждому элементу списка.

Причина предпочтительного использования `float: left`, нежели `position: absolute`, при создании списка навигации имеет отношение к проблемам «стекового контекста», описанным ранее. Элементы с необычной организацией стека требуют дополнительного добавления в различные части документа свойств `position` и `z-index`, что в свою очередь приводит к проблемам во время тестовой фазы проекта. Эти проблемы многократно возрастают, особенно в контексте добавления коммерческих объявлений или контента, добавленного в Систему управления содержимым нерадивыми пользователями.

Вертикально ориентированная навигация гораздо удобнее для применения стилей. Самой большой проблемой при ее создании являются ссылки, состоящие из двух строк. Однако присвоение `id` для каждого элемента списка навигации помогает успешно справиться с этим. Подобно тому, как горизонтальная навигация ограничивается по высоте, вертикальная ограничена по ширине, и ей не требуется присваивать значения `float`, а достаточно довольно простых правил, например:

```
    #nav { display: block; width: 10em; overflow: hidden; }
#nav li { height: 1.5em; overflow: hidden; }
```

Размещение навигационного списка в заданной области

Как правило, список ссылок первичной навигации располагается в одной из двух локаций в исходном шаблоне и соответственно в одной из двух локаций в макете шаблона. Кроме того, при создании навигации не стоит забывать о положении навигационного списка в дереве документа.

Два первых сценария, представленных здесь, подразумевают следующую структуру страницы.

1. Корень документа
   A. Контейнер для контента
      I. Верхний колонтитул
      II. Основная навигация
         i. Индивидуальные ссылки
            a. Подуровни
      III. Колонки контента
      IV. Нижний колонтитул
         i. Дополнительная навигация
            a. Индивидуальные ссылки
         ii. «Все права защищены и т. п.»

Два других сценария подразумевают, что 1.A.II и 1.A.III (основная навигация и колонки контента) транспонированы относительно исходного порядка. Истинное (видимое на экране) расположение основной навигации за редким исключением всегда находится в непосредственной близости от верхнего колонтитула. Главный вопрос — каким образом она будет ориентирована: вертикально, то есть в виде колонки в левой части макета (возможно, над второстепенным контентом), или горизонтально, вдоль верхнего колонтитула.

Особенности создания навигационных списков обсуждались в предыдущем разделе, а также будут затронуты в главе 7; здесь же мы пытаемся ответить на вопрос: как поместить навигацию на необходимый вам видимый участок страницы?

Согласно примерам, приведенным в этой главе, рис. 6.13 показывает, что навигационный список помещен в конце `#main`. Данный способ не лишен недостатков, но ведет к более логичному размещению контента.

Все становится немного сложнее, когда ссылки вертикально ориентированы, как показано на рис. 6.13, и находятся рядом с левой колонкой макета. Используя рис. 6.13 в качестве руководства, рассмотрим расположение вертикально ориентированного навигационного списка по умолчанию на том же месте в исходном порядке и осмыслим следующие четыре шага.

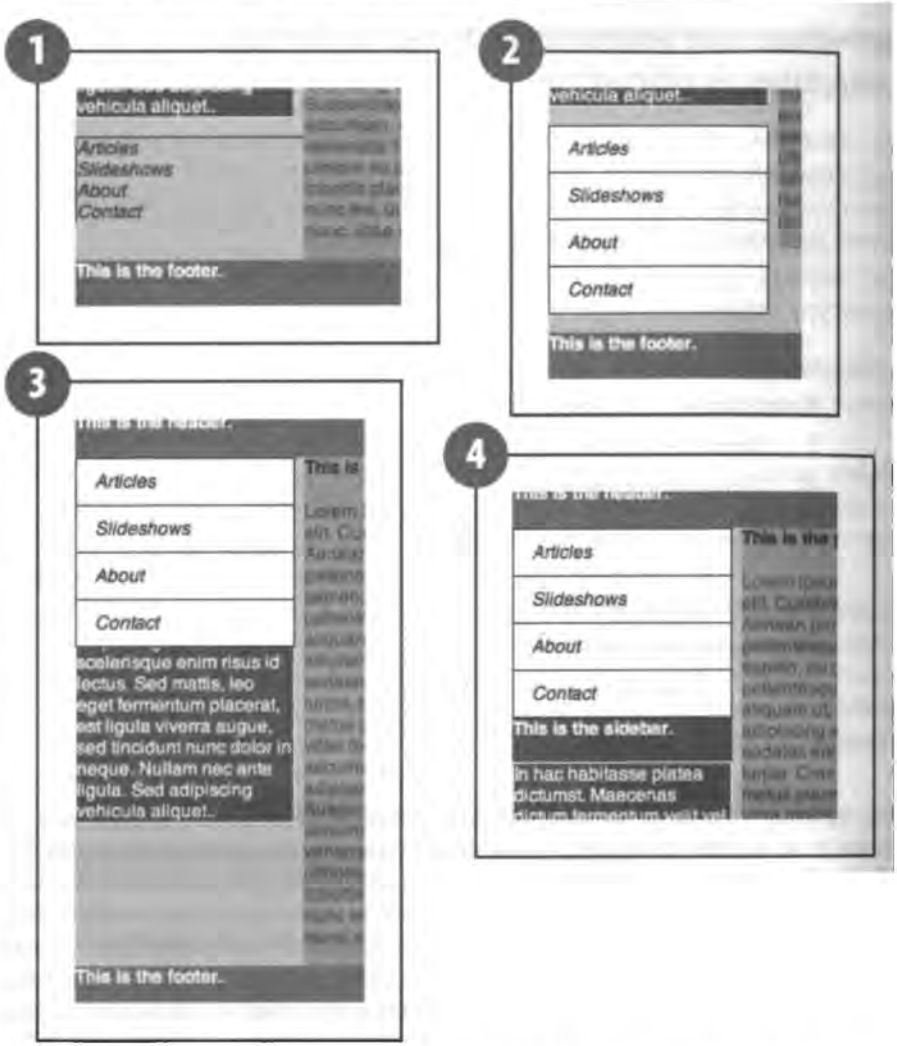


Рис. 6.13. Последовательность внедрения вертикально ориентированной навигации

1. Очистим список, применив к нему `list-style-type: none` и установим для его элементов `paddingleft: 0`.
2. Установим необходимые блочные свойства для списка и отдельно для каждого из его элементов. В этом случае отступы и две рамки будут применены со всех четырех сторон каждого элемента списка, а также две рамки к самому списку, наподобие того, как показано на рис. 10.3 в главе 10.
3. Благодаря позиционному контексту, установленному с помощью `@main`, навигационный список переместится в верхнюю часть левой колонки.
4. Левой колонке присваивается необходимое значение `padding-top`.

Типы верстки и сетка

Перед созданием макета страницы вам будет необходимо решить два важных вопроса — определить ширину макета в зависимости от границ браузера, а также определить сетку, вдоль которой будут размещены элементы страницы.

Существует три популярных подхода к тому, как связать ширину макета и ширину окна браузера, и два способа создания сетки.

Не рекомендуется смешивать фиксированные значения элементов с процентными, поскольку это может негативно повлиять на отображение рамок и закругленных углов. Для обоих этих элементов дизайна предпочтительнее, чтобы значения параметров задавались с помощью статических единиц, несмотря на целесообразность применения их к элементам с пропорциональной шириной.

Фиксированные, пропорциональные и «резиновые» макеты

Прежде чем начать работу над созданием страницы, дизайнер должен ответить на три вопроса:

- Как много места я могу выделить для сообщений сайта?
- Какое пространство будет отображать браузер типичного посетителя сайта?
- Необходимо ли пользователю, чтобы я задействовал все пространство?

Эти же вопросы с некоторыми отличиями стоит задать себе и при создании отдельных компонентов сайта: Сколько места я могу выделить для *x* и по каким причинам?

Абсолютно верного ответа на эти вопросы не существует, поскольку ответы меняются в зависимости от требований проектов, а также мастерства, опыта и взглядов дизайнеров. Тем не менее существует некоторое количество элементов дизайна, способных повлиять на горизонтальное построение макета.

- Размеры шрифтов.

Поскольку определено оптимальное количество слов в строке, а также известны самые распространенные слова данного языка, то из этого следует, что для более узких макетов предпочтительнее использовать более мелкие шрифты, а для более широких — более крупные.

- Размер изображений.

Если значительную часть контента занимают фотографии или иллюстрации, то их разрешение и размеры окажут существенное влияние на дизайн сайта, например потому, что их размеры нельзя изменить пропорционально, поскольку они имеют фиксированный размер.

- Негативное пространство и контраст.

Кроме свободного пространства, которое образуется в окне браузера, отображающего пропорционально узкую страницу, иногда бывает необходимо создавать свободное пространство (например, для канавок или полей параграфа) в различных областях макета. Области с сильной контрастностью также выигрывают от генерируемых вокруг контента полей свободного пространства.

- Ограничения браузера.

Неопытные дизайнеры могут при внедрении пропорциональных и резиновых макетов столкнуться с ошибками рендеринга.

- Правило третей, золотое сечение и ряд Фибоначчи.

Составляя дизайн страницы или ее отдельной части, вы можете опереться на эти повсеместно применяемые числа и задать с помощью них ширину колонок или другие необходимые вам размеры.

Характеристики, преимущества и недостатки трех основных типов макетов представлены в табл. 6.3.

Таблица 6.3. Типы макетов: их характеристики, преимущества и недостатки

Тип макета	Характеристики	Преимущества	Недостатки
Фиксированный	Все размеры элементов заданы в пикселах	• Наиболее терпим к канавкам, стилевым правилам, применению диакритических знаков • Менее подвержен ошибкам рендеринга • Хорошо подходит для сайтов, перегруженных изображениями	• Наименее доступен для людей с ограниченным зрением. • Уязвим для разрывов
Пропорциональный	Большинство размеров заданы в процентном соотношении	• Визуальное отображение сайта не зависит от типа браузера у посетителя. • Наиболее удобен для людей с ограниченным зрением	• Сложно управлять элементами со свойством `float`, если рамкам и канавкам присвоены фиксированные значения • Обработка браузером с широким окном очень длинных линий приводит к снижению читабельности контента
«Резиновый»	Большинство размеров задано в em	• Обеспечивает наилучший компромисс между доступностью и читабельностью. Поддерживает функцию Text Zoom • Позволяет управлять сеткой экрана • Нет необходимости использовать разные единицы измерения. Все параметры могут быть заданы в em без риска разрывов	• Дизайнеру необходимо проводить сложные вычисления, для определения точных размеров элемента. • Плохо смотрится в браузерах с маленьким экраном, особенно, если применен размер шрифта, превышающий 12 пикселов

колонок в макете, оптимальную высоту блоков контента, а также определяют, при каких обстоятельствах можно применять свойство `clear`.

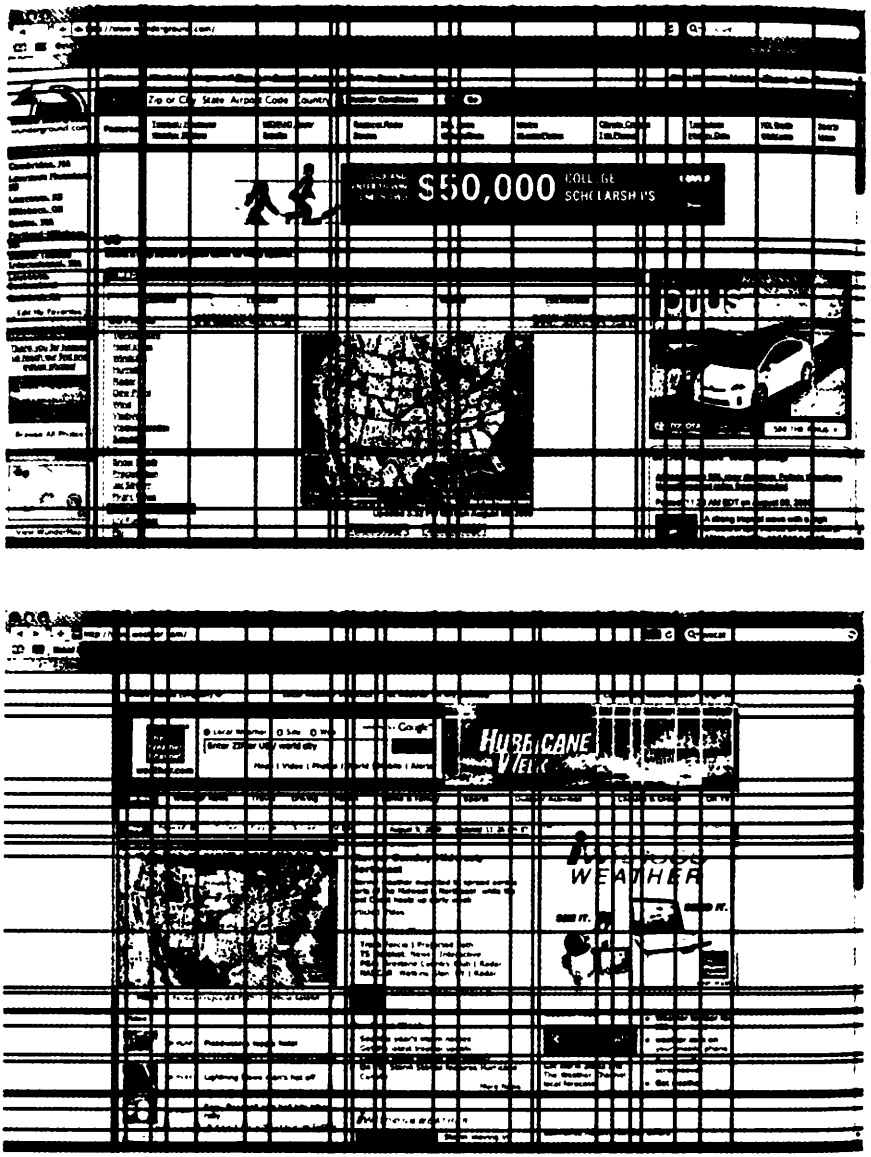


Рис. 6.14. Два однотипных сайта с линиями сетки

Альтернативой применению сетки будет способ, при котором контент собирается в одну или несколько колонок вне зависимости от занимаемой этими колонками высоты. Секции страницы строятся с различной шириной колонок,

На практике наибольшее влияние на выбор типа макета и его ширины оказывает ожидаемый тип браузера и расширение экрана у типичного посетителя страницы. На момент написания этой книги наименьшим общим знаменателем было разрешение 1024 × 768, при этом границы браузера составляли примерно 1000 × 600 (в Internet Explorer 7 с несколькими вкладками и подключенными панелями инструментов).

Большинство макетов страниц рассчитано на пользователей с большим окном браузера на мониторе с большим разрешением, например окно 1280 × 800 я запускаю на дисплее с расширением 1680 × 1050.

Если у вас и у вашего сайта большие амбиции или большая аудитория, то следует обратить внимание на разрешение дисплея 800 × 600. Для адаптации вашего сайта под это разрешение можно применить два способа:

- сократить ширину своего макета, чтобы соответствовать дисплею 800 × 600, при этом не забывая об идеальной длине линии;
- установить 1024 × 768 в качестве наименьшего общего знаменателя для окна браузера, при этом установите размер колонок для основного содержимого таким образом, чтобы его можно было без проблем просматривать при расширении 800 × 600.

Инструментальная панель веб-дизайнера для браузера Firefox позволяет проверить совместимость вашего макета с различными размерами окна. Она поддерживает диалог, который поможет вам выбрать несколько размеров окон и создать для каждого из этих типов пункты меню, которые будут изменяться в зависимости от размеров окна. Однако опытные пользователи и эксперты рекомендуют при тестировании разрешений пользоваться не специальными приложениями, а мониторами с требуемым разрешением экрана, поскольку разные дисплеи могут иметь разное зерно экрана (см. табл. 3.3).

Создание сетки

Последовательность называют главным достоинством эффективного графического дизайна. Действительно, сама природа каскадных таблиц способствует последовательности в дизайне.

В свою очередь, последовательности способствует создание *сетки*. Сетка применяется на всех уровнях дизайна сайта, но наиболее важна при создании макета/шаблона страницы, а также на «атомарном» уровне. Атомарная сетка подразумевает под собой разделение макета на множество мельчайших элементов, чаще всего (но не обязательно) представляющих собой квадраты. Многие специальные дизайнерские программы, например Adobe Photoshop, поддерживают подобную функцию. В Photoshop подобная сетка вызывается через меню View4Show Grid или при помощи горячих клавиш. Подобная сетка позволяет оценить расстояние между строками текста, возможное местоположение колонок и т. д.

Страничные сетки могут быть нескольких видов, но выполняют они практически такую же функцию, как и атомарные. Они помогают определить ширину

а последовательность приносится в жертву композиции заголовков или изображений. Результат такого лоскутного подхода можно наблюдать на рис. 6.14, где показаны домашние страницы Weather Channel и Weather Underground. Обратите внимание на беспорядок и перенасыщение на странице Weather Underground, которая расположена вверху. Большинство сайтов нельзя назвать идеальным с точки зрения полного соответствия сетке, но в случае с Weather Channel даже случайному посетителю будет понятно, что дизайнер хотя бы пытался соответствовать сетке страницы.

Совсем другая ситуация наблюдается у Weather Underground. Контент этого сайта разбит на три колонки, но с меньшим вниманием к расположению по оси Y. На обоих сайтах реклама помещена между основным контентом и нижним колонтитулом, но Weather Channel для размещения рекламных объявлений делит главную колонку, а Weather Underground разделяет всю область контента.

Обратите внимание, что основные проблемы Weather Underground возникли из-за слишком большого размера рекламных блоков. Таким образом:

Если вы собираетесь размещать на своем сайте рекламу, заранее выделите для нее место.

Поскольку предсказать высоту блока контента бывает очень сложно, проблемы вертикальной композиции следует решать с помощью редакторской дисциплины и планирования. В ситуациях, когда для создания макета страницы необходим контроль высоты, рекомендуется использовать `overflow: hidden`.

Правило третей, золотое сечение и ряд Фибоначчи

Для построения макетов страниц применимы правило третей, золотое сечение, а также ряд Фибоначчи. Правило третей самое простое для понимания и подсчетов. Согласно ему, в любой композиции взгляд фокусируется в пространстве, заключенном между четырьмя точками, находящимися на расстоянии одной трети от краев изображения.

Золотое сечение, или φ, — константа, приблизительно равная 1,618034, причем известно, что если единицу разделить на это число и прибавить к частному 1, то сумма будет равна φ.

Ряд Фибоначчи — это числовая последовательность, для которой справедливо следующее равенство: $xn + x(n+1) = x(n+2)$, наименьшее возможное значение $x(n+2)$ для этой последовательности равно $1(0+1)$. Таким образом, десять чисел последовательности Фибоначчи, начиная с единицы, это: 1, 2, 3, 5, 8, 13, 21, 34, 55 и 89.

Применение правила третей приводит к соотношению 3:2 или 5:3, что является грубыми приближениями к золотому сечению, а числа, взятые за основу для разбиения композиции в правиле третей, — числа Фибоначчи.

На рис. 6.15 показано применение правила третей и ряда Фибоначчи для построения сетки. Подобная сетка служит удобной основой для создания разметки макета, особенно при определении ширины колонок и размеров верхнего и нижнего колонтитулов.

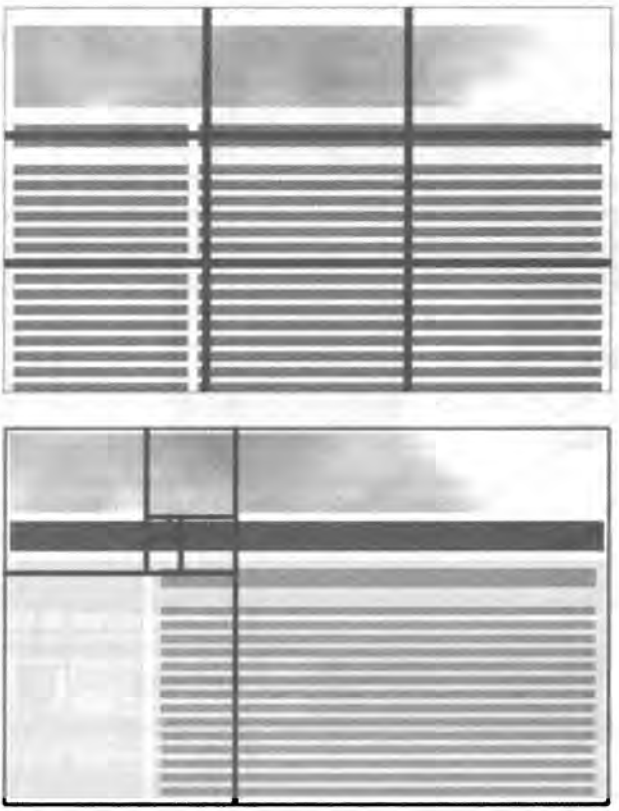


Рис. 6.15. Сетки, созданные с помощью правила третей и ряда Фибоначчи

Внедрение гибкой сетки

Часть стандартов, описанных в CSS3, посвящена методу определения «атомарной» сетки произвольного размера. Ссылку на спецификацию, освещающую этот вопрос, можно найти на сопутствующем сайте книги.

Большинство атомарных сеток, применяемых к макету страницы, следуют одному базовому правилу: высота ряда соответствует одной строке текста с междустрочием. Если я вношу в таблицу стилей правило `body { ... font-size:`

14px; line-height: 1.714em; ... }, то я предполагаю использовать сетку высотой 24 пиксела в случае, если не будет применено масштабирование. Но если даже параметр Text Zoom будет применен, междустрочия в моем тексте останутся пропорциональными.

Text Zoom упомянут здесь, поскольку он может оказать неоценимую помощь для доступности страницы, а также потому, что он имеет тенденцию наносить ущерб оформлению, не завязанному на сетку или использование em.

Краткая характеристика поддержки свойства Zoom современными браузерами приведена в табл. 6.4.

Таблица 6.4. Поддержка свойства Zoom браузерами

Тип свойства Zoom	IE 8	IE 7	IE 6a	Firefox 3	Firefox 2	Safari 4	Safari 3
Text Zoom	✓	✓	✓	✓	✓	✓	✓
Page Zoom	✓	✓	–	✓	–	✓	–

Ширина атомарной колонки может быть любой. Ширина, которая приводит к квадратам, кажется наиболее логичной, но не всегда приводит к удобным размерам канавок. Среди других вариантов стоит отметить:

- высоту ряда, разделенную (или умноженную) на φ;
- 1em или несколько em;
- среднюю ширину слова в вашем контенте.

При внедрении атомарной сетки прежде всего следует не забывать о последовательности. Ширина ваших колонок, а также других линий разметки на странице, должна соответствовать линиям вашей атомарной сетки, а если вы создаете «резиновый» макет, то вам следует выразить все размеры в em.

Например, предположим, что я использую последовательность Фибоначчи в качестве основы для создания двухколоночного макета с квадратной сеткой. При этом пропорции элементов до внедрения полей и канавок будут следующими:

- высота верхнего и нижнего колонтитулов — $8x$;
- ширина боковой панели — $13x$;
- ширина основного контента — $21x$;
- желательная ширина встроенных фото — $8x$;
- желательные размеры заголовков (с междустрочиями) — $3x$ и/или $5x$.

Теперь передо мной стоит задача найти x. Учитывая отношение ширины тела к ширине боковой панели, я хочу определить самую короткую из оптимальных длину строки: 12 слов, что примерно соответствует 36em, что в свою очередь дает нам значение x, равное 0,618em.

Поскольку мне неудобно использовать строки высотой 0,618em, я удвою коэффициент, следовательно, значение line-height будет равно 1,235em (что для

многих шрифтов близко к значению по умолчанию). Если бы я взял следующее число из ряда Фибоначчи, я бы получил для `line-height` значение 1,853em, которое кажется мне слишком большим.

С другой стороны, я бы мог подойти к моему макету с точки зрения пропорции (13:21 отношение боковая панель: тело при общей высоте в 34) и разделить его на ширину экрана. Результат подобных подсчетов даст нам размер элемента сетки, равный 28–30 пикселам, что предполагает больший размер шрифта при более короткой строке.

Независимо от того, какое значение я выберу, я могу внести результат в таблицу стилей, задав параметры сетки, присвоив размеры каждой важной части контента и по максимуму используя селекторы стилей.

7 Работа со списками

Ежедневно повсюду мы сталкиваемся со списками: в опросных листах, перечне покупок, ежедневниках, перечнях лучших и худших вещей, инструкциях и в обычных рейтингах. Также списки очень часто встречаются в разметке веб-страниц. Такие списки используются для всех вышеперечисленных целей и очень часто для поиска. Язык HTML поддерживает три типа списков: упорядоченные списки, неупорядоченные списки и списки определений.

Упорядоченные и неупорядоченные списки

Основное отличие упорядоченных списков от неупорядоченных кроется в самом определении: иногда бывает необходимо упорядочивание элементов по определенному критерию (например, по значимости элемента, порядку выполнения действий, времени, порядку добавления элементов, алфавиту), а иногда список представляет собой группу беспорядочных данных, объединенных по общему признаку.

Стили по умолчанию для упорядоченных и неупорядоченных списков в разных браузерах

На первый взгляд списки без стилей выглядят как блоковые элементы (которыми они, по сути, и являются), содержащие наборы других блоковых элементов. Каждый из этих элементов смещен относительно левого края списка. Однако, как показывает табл. 7.1, для списков в современных браузерах применяются стили, отличающиеся от тех, что применялись в браузерах предыдущих поколений.

Обратите внимание на значения `40px`. Это значит, что при использовании крупных шрифтов для отображения длинного упорядоченного списка маркеры элемента списка, скорее всего, скроются за левой границей. Этого можно избежать в случае необходимости, увеличивая показатели `margin-left` и `padding-left`, выставив их в `em`, а не `px`-единицах.

Таблица 7.1. Интерфейсные стили для упорядоченных, неупорядоченных списков, а также списков определений (ul, ol, dl)

Интерфейсы	Интерфейсный стиль
Firefox 2+ Internet Explorer 7+ Safari 3+	margin: 1em auto 1em 0; padding-left: 40px;
Firefox 1.0.x–1.4.x Internet Explorer 6 Quirks rendering modes	margin: 1em auto 1em 40px;

Создание эффективных упорядоченных и неупорядоченных списков

Делать эффективные списки не сложно, но надо соблюдать несколько основных правил:

- список должен содержать как минимум один элемент;
- в списке не должно быть ничего другого, кроме элементов списка;
- элементы списка должны создаваться непосредственно в самом списке;
- все производные `div`-элементы также являются производными элементов списка, включая простой текст и другие списки;
- HTML 4.01 позволяет создавать элементы списка без закрывающих тегов;
- атрибут `start` тега `ol`, атрибут `value` тега `li` в рамках атрибута `ol` и атрибут `type` допустимы для документов переходного, а не строгого типа;
- регистры элементов упорядоченного списка не фиксируют элементы с нулевым значением параметра `display`.

Свойство list-style-type и атрибут type

С элементами упорядоченного списка может использоваться целый набор маркеров. По умолчанию в качестве маркера используются десятичные числа, а латинские буквы и римские цифры могут быть представлены как в строчном, так и прописном вариантах.

Для элементов неупорядоченных списков номинально существует три типа маркеров: диски (буллиты), круги и квадраты.

За представленными далее поддерживаемыми значениями свойства `list-style-type` в скобках следуют аналогичные значения HTML `type`:

- `circle (type="circle")`
- `disc (type="disc") [по умолчанию для неупорядоченных списков]`
- `decimal (type="1") [по умолчанию для упорядоченных списков]`

- lower-alpha (type="a")
- lower-roman (type="i")
- none
- square (type="square")
- upper-alpha (type="A")
- upper-roman (type="I")

Часто используется значение none, поскольку навигация сайта редко оформляется с буллитами по умолчанию, особенно при использовании техник замещения текста графикой (см. раздел «Растровая копия и замещение текста графикой по методу Фарнера», с. 185).

Для атрибута list-style-image, заменяющего сгенерированный маркер специальным изображением, также существует ограниченная поддержка. Тем не менее результаты применения данного свойства, как правило, удручающие. Поэтому лучше включать специальные маркеры посредством свойств background-image и background-repeat.

В CSS 2.1 также представлены маркеры lower-latin, upper-latin, lower-greek и upper-greek, которые поддерживают далеко не во всех браузерах.

Nav-элемент (HTML5)

Среди новых элементов HTML5, нацеленных на усиление разнообразия структуры HTML-документов, nav-элемент, пожалуй, наиболее важный. В то время как другие предложенные структурные элементы, например элемент section, представляются разработчикам отличными функциями, конечные пользователи не видят их потенциала. Но вот с nav-элементом другая ситуация. Важнейшая особенность этого элемента заключается в том, что он предлагает стандартный способ оформления навигационных элементов, который в результате опознается браузерами и в случае необходимости обрабатывается соответствующими для браузеров способами. Это означает, что браузеры могут использовать nav-элемент для существенного улучшения своей работы.

Меню навигации, то есть список гиперссылок, который есть на любой веб-странице и который обеспечивает возможность перехода на другие страницы сайта или к другим функциям веб-приложения, может быть расположен в любой части страницы. Например, на странице с многоколонной верстkой вы можете расположить основное меню в левой крайней колонке страницы, а дополнительное меню разместить в правой крайней колонке. А на странице с другим дизайном можно разместить список ссылок в меню, расположенном в верхней части страницы. При выборе меню пользователем оно «разворачивается», и появляются элементы меню. А дополнительная навигация размещается в футере.

При отсутствии nav-элемента навигация, как правило, оформляется с помощью элемента div со своим атрибутом class. Значение «nav» в атрибуте

class фактически одно из примерно двадцати наиболее часто используемых во Всемирной паутине значений атрибута class. Для сайтов также используются такие значения, как primaryNav и secondaryNav, в целях более подробного разграничения секций навигации.

Проблема использования значений атрибута class, связанная с разметкой навигации, заключается в отсутствии согласованных стандартных значений для всех сайтов. Nav-элемент решает эту проблему путем предоставления стандартного элемента, который может использоваться всеми разработчиками в единых целях. Кроме того, стандартный nav-элемент помогает разрешить общую дилемму, характерную для многих разработчиков: необходимость изначально решить, где должна располагаться навигация в структуре документа, чтобы в итоге это не сказалось отрицательно на доступности документа или не снизило удобство пользования навигацией на портативных устройствах с минидисплеем.

Проблема доступности и удобства применения

Ряд вопросов, связанных с доступностью, сказывается на решении того, куда следует поместить навигацию в структуре документа. Например:

1. Существует общепринятое убеждение, что если вы не размещаете навигацию в верхней части документа, то этот документ становится менее доступным для пользователей со специальными устройствами, например для тех, кто пользуется приложением Screen reader. Проблема заключается в том, что эти пользователи привыкли видеть основное меню в верхней части документа, поэтому искать эту функцию в других областях они просто не станут.
2. Парадоксально, но, с другой стороны, существует мнение, что если вы размещаете систему поиска в верхней части документа, то доступность контента снизится. Это происходит потому, что при стандартном дизайне почти у всех страниц сайта будет меню с одинаковыми ссылками, через которые придется проходить пользователю, использующему специальные устройства, чтобы получить доступ к основному контенту. Подобный поиск удручает и отнимает время.
3. Вопрос удобства применения контента для браузеров портативных устройств с минидисплеями тесно связан со второй проблемой. Ряд браузеров для портативных устройств имеет встроенные функции автоматического переформатирования контента в одну колонку для удобного чтения информации с минидисплея. Портативные устройства проводят переформатирование, исходя из первоначальной структуры документа. Поэтому, если навигация находится в верхней части страницы, то пользователи этих браузеров в итоге сталкиваются с той же проблемой, что и пользователи специальных устройств. На каждой странице у них появляется длинное меню, которое приходится проматывать, чтобы перейти к основному контенту.

На практике вторая и третья проблемы решаются разработчиками путем стандартного размещения одной из следующих гиперссылок: «Перейти к основному контенту» или «Пропустить меню». Такая ссылка размещается на каждой странице перед меню. Это помогает, но обидно, что приходится искать обходные пути в отсутствие стандартного способа решения этой проблемы (за исключением эвристических). Появления HTML5 должно помочь в определении программой screen reader или браузером портативного устройства, какая часть документа представляет собой навигацию, а какая — фактический контент.

Альтернативные способы представления навигации

После добавления nav-элемента в HTML5 в браузерах появится стандартный элемент для реализации навигации. Поэтому браузеры смогут обеспечить альтернативные способы отображения навигации, как поясняет текущая версия документации по HTML5:

> Nav-элемент очень полезен не только для программ screen readers. Он также крайне полезен для браузеров портативных устройств с минидисплеем. Такие браузеры могут получать доступ к контенту с nav-элементом посредством меню функций, а не путем обработки основного текстового контента страницы.

Само собой, чудесное применение nav-элемента остается лишь несбыточной мечтой до тех пор, пока в современных геджетах, браузерах мобильных устройств и других пользовательских программах не будет реализован ряд функций, которыми все начнут пользоваться.

Изменение области упорядоченного списка

Возьмем, к примеру, такой упорядоченный список:

Длинноволновые

1. Красный.
2. Оранжевый.
3. Желтый.
4. Зеленый.

Коротковолновые

5. Синий.
6. Индиго.
7. Фиолетовый.

HTML-код этих списков выглядит следующим образом:

```
<h5>Длинноволновые</h5>
<ol>
    <li>Красный.</li>
    <li>Оранжевый.</li>
    <li>Желтый.</li>
    <li>Зеленый.</li>
</ol>
<h5>Коротковолновые</h5>
<ol start="5">
    <li>Синий.</li>
    <li>Индиго.</li>
    <li>Фиолетовый.</li>
</ol>
```

Взаимосвязь между этими списками должна быть такой же четкой, как и их компоновка. Тем не менее обратите внимание на использование атрибута `start`. Только в переходных версиях HTML приведенный код будет считаться валидным, поскольку единственно надежный способ прерывания и возобновления упорядоченного списка происходит посредством устаревших (нерекомендуемых) атрибутов `start` и `value` в HTML 4. С помощью атрибута `value` можно единовременно задать необходимое значение для конкретного элемента списка.

В CSS 2.1 описан механизм реализации счетчика, но этот механизм трудно понять, и его можно активировать только посредством псевдокласса `:before`. При использовании этого псевдокласса в списках контент отображается с ошибочным отступом.

Кроме того, в настоящий момент функция счетчика не поддерживается браузером Internet Explorer.

Для построения упорядоченного списка с произвольного значения примените атрибут `start` или атрибут `value` к первому элементу списка. Значения этих атрибутов должны быть заданы арабскими целыми числами, вне зависимости от значений `type` или `list-style-type` (которые будут сохранены).

Другие функции списков

У списков есть еще две довольно простые функции: контурные и внутристрочные серийные списки.

Контурные списки

В двух словах контурный HTML-список представляет собой серию структурно расположенных упорядоченных списков со следующими стилями:

```
    ol { list-style-type: upper-roman; }
ol ol { list-style-type: upper-alpha; }
```

```
    ol ol ol { list-style-type: decimal; }
  ol ol ol ol { list-style-type: lower-alpha; }
ol ol ol ol ol { list-style-type: lower-roman; }
```

6–10 уровни контурного списка в случае необходимости повторяют вышеприведенную последовательность. Однако следующий блок, возможно, будет включать другой шрифт (шрифт будет более мелким или светлым) при структурированном расположении текста по умолчанию.

Внутристрочные серийные списки

Не совсем обычный, но возможный вариант заключается в применении серии запятых для разграничения нескольких элементов в списке. Подобная разметка веб-документа объясняется рядом причин. Одна из наиболее распространенных — необходимость в сохранении места. Также вполне вероятно, что редактор будет настаивать на отображении списка в таком формате.

Чтобы задать внутристрочный список, поместите ваш параграф в элемент `div` и разбейте его на элементы списка. В результате получится два абзаца, разделенных списком. Далее добавьте следующие стили:

```
.foo p, .foo ul, .foo li { display: inline; }
                .foo ul { list-style-type: none; }
```

Если контент каждого элемента списка заканчивается запятой с пробелом (или в случае необходимости предпоследней командой `and`), то результат будет соответствовать строгим редакторским и семантическим требованиям. Дополнительный параметр `div` не является лишним, поскольку неопределенный контент изначально все равно был задан в виде одного абзаца.

Изменение внешнего вида ссылок в футере

О свойствах компоновки `inline-block` рассказывалось в главе 6. Эти свойства особенно важны при работе с ссылками в футере. Вместо того чтобы тратить время на решения с использованием `float`, можно воспользоваться следующим более простым подходом на основе исходных стилей:

```
#footer ul { list-style-type: none; text-align: center; }
#footer li { display: inline-block; }
```

Если требуются специальные буллиты ссылок на нижний колонтитул, то можно изменить значение `padding-left` элементов списка и при необходимости воспользоваться фоновым изображением.

Фоновые буллиты?

Да, вы не ошиблись. Свойство CSS `list-style-image` ввели специально для вставки специальных буллитов. Но в использовании этого свойства есть три серьезных недостатка.

Первым недостатком свойства `list-style-image` заключается в том, что вызываемое этим свойством изображение всегда отображается на уровне базовой строки, заставляя дизайнера создавать композицию предполагаемого буллита в графическом редакторе пиксел за пикселом. Это требование полностью идет вразрез с задачей разграничения содержания и формы.

Второй недостаток является прямым продолжением первого: если пользователь увеличивает масштаб страницы, пропорции буллитов перестают согласовываться с соответствующим текстом.

Третий недостаток заключается в том, что без дополнительных параметров (таких, какие дизайнер может задавать для фоновых свойств) нельзя применять спрайты (хотя в этом случае их применение может быть очень опасно) для создания специальных буллитов, задаваемых свойством `list-style-image`.

Оформление навигации

В этом разделе термин «nav» будет применяться и для обозначения списка, содержащего первичное меню навигации («nav-элементы»), и аналогичного дизайнерского элемента, который появится на сайте. Ссылки в рамках nav-элемента будут использоваться открыто.

Основное меню может быть ориентировано вертикально и горизонтально. Горизонтальное ориентирование предполагает размещение слева-направо. А вертикальное меню образует вертикальный стек. Другие шаги в процессе создания навигации примерно одни и для вертикального, и для горизонтального расположения.

Размещение навигации в коде документа

Первый возникающий вопрос: где будет располагаться мой nav-элемент в коде шаблона? При ответе на этот вопрос необходимо учитывать целый ряд факторов.

- *Пользователи специальных интерфейсов, как правило, ожидают увидеть (или услышать) nav-элемент в начале документа.*

  Это убеждение характерно для дизайнерских настроений и программ 1990-х годов, когда основной nav-элемент практически всегда содержал вторую часть данных страницы исходного документа (после заголовка сайта).

- *Однако в целях экономии времени пользователя специального интерфейса лучше размещать основной контент как можно ближе к началу страницы исходного документа.*

  Необходимость действовать в соответствии с данным принципом компенсируется удобной прокруткой списков и пользовательскими ожиданиями, однако

игнорировать эту необходимость не следует. Пониженное расположение основного nav-элемента может даже улучшить оптимизацию поисковых систем.

- *Размещение nav-элемента сразу в начале документа позволит вам применять упрощенные стили (до определенной степени).*
  Размещение основного nav-элемента в конце страницы документа практически на все сто гарантирует, что вы будете вынуждены полагаться на позиционный контекст для размещения навигации в желаемой области страницы. Это неизбежно усложнит таблицу стилей.
- *Поддержка навигации сильно упрощается, если ее разместить в отдельном включаемом файле.*
  Также в вашем шаблоне может сократиться количество подключаемых файлов на один. Для большинства сайтов это не большая проблема, но если сайт объемный, то даже один подключаемый файл может улучшить производительность.

Способ расположения основной навигации

Создав основной nav-элемент и определившись с областью его размещения в исходном документе, можно завершать его размещение в шаблоне сайта:

1. *Добавьте элемент* `id` *к элементу nav и его составляющим.* Вопрос, какое дать название элементу, рассматривается в разделе «Применение таксономии к таблицам стилей страницы», с. 93.
2. *Обнулите стили nav-элемента и его составляющих элементов.* Это делается просто путем добавления строки `{ margin: 0; padding: 0; list-style-type: none; }` в вашу таблицу стилей.
3. *Оставляйте список с nav-элементом там, где он появляется в верстке.* Применяемые для выполнения этого шага стили будут варьироваться в соответствии с системой координат, расположением документа, компоновкой nav-элемента, а также в соответствии со схемой компоновки всей страницы. Этот метод, в большинстве случаев помогающий избежать ошибок, требует нахождения nav-элемента в контентном контейнере страницы, которая затем проходит относительное размещение таким образом, что сам nav-элемент может быть размещен полностью на полях контейнера страницы.
4. *При необходимости назначьте новые значения параметров box и display.* На большинстве сайтов ссылки будут оформлены в идентичном стиле и будут иметь горизонтальную систему координат. Поэтому многие стили элементов будут содержаться внутри `#nav li`.
5. *Разместите nav-элементы в нужном порядке.* Для nav-элементов с горизонтальной координацией размещение можно провести командой `float: left` или `position: relative`. Последний подход сложнее адаптировать к дизайнерским изменениям, а вот первый подход создаст большую вероятность для возникновения ошибок в обозревателе Internet Explorer 6.

Вертикально ориентированным nav-элементам для выполнения этого шага, скорее всего, потребуется лишь команда `display: block`, но такие элементы, скорее всего, будут вложенными. Если вертикально ориентированные nav-элементы содержат более одного уровня, то у списков с подуровнями должны быть назначены свои собственные id-элементы (например, `#navSubAbout`). Более подробная информация об этом этапе представлена в разделе «Создание точного кода и макета для навигации» на с. 126.

6. *Разверните nav-линки в соответствии с размерами содержащихся в них элементов.* Перво-наперво это делается путем настроек ссылок командой `display: block`, которая помогает достичь преимуществ, подробно описанных в главе 8. Над поиском оптимальных значений параметров `width`, `height` и `padding` придется попотеть. Возможно, в содержащие элементы также будет целесообразно добавить параметр `overflow: hidden`.
7. *При желании можно выполнить замену изображений по методу Фарнера.* При выполнении этой функции фоновые изображения могут быть также присвоены nav-элементам и nav-ссылкам. Таким образом, ссылка в подвешенном состоянии будет отображать другой фон в отличие от содержащего и базового элементов списка.

Размещение навигации в футере

Задать стиль дополнительной системе навигации, как правило, легче, чем основной. Во-первых, редко возникают сомнения относительно ее размещения в структуре документа (практически в самом низу над заявлением о правах и условиях). Во-вторых, этим ссылкам редко задаются одинаковые размеры.

Скорее всего, верстка вспомогательной системы навигации будет осуществлена одним из следующих трех способов.

- *Верстка с размещением по центру страницы с большими отрицательными значениями полей со всех сторон.*

  Один из примеров такой верстки можно посмотреть на страницах сайта PayPal. При таком подходе элементы списка выстраиваются так: `display: inline`. И границы задаются по одной стороне (правой или левой).

- *Верстка с небольшим смещением к одному из нижних углов структурированной страницы.*

  Данный прием можно посмотреть на пользовательских страницах сайта Facebook. Основное отличие между этим приемом и центрированием заключается в выравнивании (`text-align: right` вместо `text-align: center`) и вертикальном колонтитуле (как правило, ограниченным одной строкой, а не разбитым на две или более строк).

- *Верстка с минимальной оформлением списка после третичного контента сайта.*

Этот подход становится популярным для блогов. Дополнительные nav-элементы спокойно могут находиться в разметке третичной колонки, а также в нижнем колонтитуле.

С последним типом верстки вообще не возникает никаких проблем. У первых двух есть небольшие проблемы с разделителями и свободным местом. Вот как осуществляется верстка:

1. В списке или элементах списка задайте свойство `list-style-type: none`
2. Если список должен располагаться по центру, задайте ему соответствующую ширину и отступы по оси *x*.
3. Задайте желаемое значение `text-align` в nav-списке и параметр `display: inline` по отдельным элементам списка.
4. Задайте соответствующие значения `padding` и `line-height` в списке и элементах списка, если только у ваших ссылок не проставлена команда `display: inline-block`. В последнем случае для ссылок нужно задать текстовые и блочные свойства. Установка одинаковых значений `padding` для правой и левой стороны каждого элемента со временем упростится, даже если в первое время будет казаться, что это не так.
5. Добавьте разделители для ваших элементов. Если разделители представлены в виде буллитов, то их, скорее всего, придется заменить на фоновые изображения.
6. Добавьте необходимые элементы `class` или `id` для удаления разделителей с открытых начальных и конечных элементов и разбивке списка на нужное количество строк. Во многих случаях свойство `white-space` может хорошо подойти для последнего этапа, хотя, если элементы обычных строк обладают очевидным семантическим показателем, их спокойно можно разбивать на множество списков.
7. Завершите работу над версткой в соответствии с требованиями композиции.

Списки определений

Если упорядоченные и неупорядоченные списки представляют собой обычные блоки данных с элементами неопределенной классификации, то списки определений представляют определенные связи между терминами (выраженными `dt`-элементами) и их определениями (выраженными `dd`-элементами). Каждый термин сопровождается одним или несколькими определениями, которые, как надо понимать, неразрывно связаны с этим термином.

Эффективный список определений должен содержать как минимум один `dt`- или `dd`-элемент. Семантически полезный список должен содержать минимум один `dt` и `dd`-элемент. `dt`-элементы могут лишь содержать текстовые и встроенные элементы, а `dd`-элементы могут содержать широкий спектр контента как `li`-элементы. По компоновке или количеству `dd`- и `dt`-элементов в пределах

списка определений ограничений не существует. Эффективное расположение элементов списка определений задается авторами контента.

Создание стилей для списков определений

Браузеры применяют минимальный набор стилей для списков определений. Данные стили имеют следующие отличительные признаки:

- `dt`-элементы похожи на абзацы без полей;
- `dd`-элементы смещены командой `margin-left`, но никогда не маркируются;
- у `dd`-элементов имеются (отсутствуют) те же ограничения по валидному контенту, как и у `li`-элементов;
- текстовый контент списка определений по умолчанию не имеет стилей.

Список определений чаще всего применяется для лексического материала (например, в глоссариях, словарях) и транскрибированного диалога. Первые из этих довольно простых программных стилей, как правило, будут соответствовать этой цели с учетом того, что придирчивый к шрифтам дизайнер, возможно, захочет выставить `dt`-элементы жирным шрифтом.

Оформление диалога требует других значений и поэтому проводятся следующие изменения.

- Параметр `width` у `dt`-элемента должен иметь постоянную величину и применяться параметр `clear: left`.
- Параметр `dd`-элементов `margin-left` должен быть немного завышен по сравнению с шириной `dt`-элементов.
- Одна из вариантных настроек шрифта (например, `font-weight: bold` или `text-transform: uppercase`) должна быть применена к `dt`-элементам.

Пример разметки текста словаря

Адаптированный вариант словаря английского языка в 4-м издании, издательство American Heritage:

> e·con·o·my n. Inflected forms: pl. e·con·o·mies **1. a.** Careful, thrifty management of resources, such as money, materials, or labor: learned to practice economy in making out the household budget. **b.** An example or result of such management; a saving. **2. a.** The system or range of economic activity in a country, region, or community: Effects of inflation were felt at every level of the economy. **b.** A specific type of economic system: an industrial economy; a planned economy. **3.** An orderly, functional arrangement of parts; an organized system: "the sense that there is a moral economy in the world, that good is rewarded and evil is punished" (George F. Will). **4.** Efficient, sparing, or conservative use: wrote with an economy of language. **5.** The least expensive class of accommodations, especially on an airplane. **6.** Theology The method of God's government of and activity within the world. adj. Economical or inexpensive to buy or use: an economy car; an economy motel.

Вот пример разметки для отрывка из словарной статьи:

```
<dl>
    <dt>e&middot;con&middot;o&middot;my</dt>
    <dd>
    <span class="partOfSpeech"><abbr title="noun">n.</abbr></span>
    <span class="variants">Inflected forms: <span class="partOfSpeech">
    <abbr title="plural">pl.</abbr>
    </span> e&middot;con&middot;o&middot;mies</span>
    <ol>
      <li>
        <ol>
          <li>Careful, thrifty management of resources, such as money,
materials, or labor:
        <span class="usageEx">learned to practice economy in making out
the household budget.</span></li>
    <li>An example or result of such management: a saving.</li>
    </ol>
  </li>
  <li>
    <ol>
    <li>The system or range of economic activity in a country, region, or
community: <span class="usageEx">Effects of inflation were felt at every
level of the economy.</span></li>
    <li>A specific type of economic system: <span class="usageEx">an industrial
economy; a planned economy.</span></li>
    </ol>
  </li>
<li>An orderly, functional arrangement of parts; an organized system:
<span class="usageEx"><q>the sense that there is a moral economy in the
world, that good is rewarded and evil is punished </q> <cite>(George F.
Will).
</cite></span></li>
<li>Efficient, sparing, or conservative use: <span class="usageEx">wrote
with an economy of language.</span></li>
<li>The least expensive class of accommodations, especially on an
airplane.</li>
<li><span class="subjectFocus">Theology</span> The method of God&rsquo;s
government of and activity within the world.</li>
</ol>
<span class="partOfSpeech"><abbr title="adjective">adj.</abbr></span>
Economical or inexpensive to buy or use: <span class="usageEx">an economy
car; an economy motel.</span>
 </dd>
</dl>
```

У данной разметки есть несколько интересных особенностей:

- *Различные определения в основном обрамлены упорядоченными списками, а не последовательными* dd*-элементами.*

Это было сделано благодаря дизайнеру, то есть мне. Я решил, что лучше сохранить нумерацию, а не пролистывать одно семантическое значение за другим. Если бы браузер Internet Explorer поддерживал генерируемую нумерацию, как другие браузеры, я бы не выбрал такой вариант разметки, но наличие недоработок в различных браузерах заставляет опытных дизайнеров ежедневно принимать подобные решения.

- *Пользователи других браузеров (в отличие от пользователей Internet Explorer) обнаружат различные определения внутри строки.*

  Тем не менее назначение элементам списка параметра `display: inline` приводит к исчезновению маркеров элементов. Таким образом, приходится назначать всем этим элементам новые маркеры с помощью функции `counter()`, которая может задавать значение свойству `content`. При добавлении дизайнером соответствующего базового правила в условную таблицу стилей пользователи Internet Explorer наблюдают обычные упорядоченные списки.

- *В источнике много элементов* `span` *и* `class`.

  Большое количество встроенной разметки порадует сторонников семантического подхода, считающих необходимым проставление семантических меток при каждом удобном случае. С дизайнерской точки зрения, дополнительная разметка делает возможным различные виды стилизации отличающихся друг от друга меток (в регистре).

Если опустить нумерацию и другие более тонкие детали, то можно использовать более типичную `dt/dd` структуру:

```
<dt>economy</dt>
  <dd>Careful, thrifty management of resources, such as money, materials,
or labor:
       <span class="usageEx">
learned to practice economy in making out the household budget.
       </span></dd>
<dd>An example or result of such management; a saving.
   The system or range of economic activity in a country, region, or
community:
       <span class="usageEx">
Effects of inflation were felt at every level of the economy.
       </span></dd>
 <dd>A specific type of economic system: an industrial economy; a planned
economy. An orderly, functional arrangement of parts; an organized system:
       <span class="usageEx">&ldquo;the sense that there is a moral
economy in the world, that good is rewarded and evil is punished&rdquo;
(George F. Will).
       </span></dd>
<dd>Efficient, sparing, or conservative use:
       <span class="usageEx">wrote with an economy of language.
       </span></dd>
 <dd>The least expensive class of accommodations, especially on an
airplane.</dd>
```

```
 <dd>The method of God&rsquo;s government of and activity within the
world.</dd>
 <dd>Economical or inexpensive to buy or use: an economy car; an economy
motel.</dd>
...
```

Пример разметки диалога

В отличие от списков определений диалоги строятся несколько по другому, но структурные элементы у них похожие. В качестве примера возьмем отрывок из 4-го акта пьесы Бернарда Шоу «Пигмалион»:

> HIGGINS: [In despairing wrath outside] What the devil have I done with my slippers? [He appears at the door]
>
> LIZA: [Snatching up the slippers, and hurling them at him one after the other with all her force] There are your slippers. And there. Take your slippers; and may you never have a day's luck with them!
>
> HIGGINS: [Astounded] What on earth—! [He comes to her] What's the matter? Get up. [He pulls her up] Anything wrong?
>
> LIZA: [Breathless] Nothing wrong—with you. I've won your bet for you, haven't I? That's enough for you. I don't matter, I suppose.
>
> HIGGINS: You won my bet! You! Presumptuous insect! I won it. What did you throw those slippers at me for?
>
> LIZA: Because I wanted to smash your face. I'd like to kill you, you selfish brute. Why didn't you leave me where you picked me out of—in the gutter? You thank God it's all over, and that now you can throw me back again there, do you? [She crisps her fingers frantically]

Далее представлена разметка источника:

```
<dl>
  <dt>Higgins</dt>
  <dd><span class="stageDir">In despairing wrath outside</span> What the
devil have I done with my slippers? <span class="stageDir">He appears at
the door</span></dd>
  <dt>Liza</dt>
  <dd><span class="stageDir">Snatching up the slippers, and hurling
them at him one after the other with all her force</span> There are
your slippers. And there. Take your slippers; and may you never have a
day&rsquo;s luck with them!</dd>
  <dt>Higgins</dt>
  <dd><span class="stageDir">Astounded</span> What on earth&mdash;!
  <span class="stageDir">
    He comes to her</span> What&rsquo;s the matter? Get up. <span
class="stageDir">
    He pulls her up</span> Anything wrong?</dd>
```

продолжение ⇩

```
  <dt>Liza</dt>
  <dd><span class="stageDir">Breathless</span> Nothing wrong &mdash; with
<em>you</em>.
    I&rsquo;ve won your bet for you, hav&rsquo;n&rsquo;t I? That&rsquo;s
enough for you. Idon&rsquo;t matter. I suppose.</dd>
  <dt>Higgins</dt>
  <dd><em>You</em> won my bet! You! Presumptuous insect! I won it. What
did you throw those slippers at me for?</dd>
  <dt>Liza</dt>
  <dd>Because I wanted to smash your face. I&rsquo;d like to kill you, you
selfish brute. Why didn&rsquo;t you leave me where you picked me out of
&mdash; in the gutter? You thank God it&rsquo;s all over, and that now you
can throw me back again there, do you? <span class="stageDir">She crisps
her fingers frantically</span>
</dl>
```

Как и в примере со словарным текстом, здесь применяются параметры span и class для передачи дополнительного информационного слоя, в данном случае — сценической ремарки.

Что касается оформления, то общий подход не отличается от описанного ранее помимо добавления параметра clear: left к dt-элементам. Последнее существенное отличие между обработанной пьесой и исходным кодом заключается в появлении имен персонажей в верхнем регистре. Эта ситуация регулируется параметром text-transform, описание которого представлено в главе 10.

Вначале в HTML5 был разработан dialog-элемент для этих случаев, но потом его убрали из спецификации.

Когда разработчики стали обращать внимание на использование списков в своей продукции, то пришло понимание уникальности этого элемента. Как выяснилось, единственный серьезный недостаток списков в противовес их функциональности и уникальности заключается в сложности их оформления.

8 Заголовки, гиперссылки, строковые элементы и цитаты

Эффективная работа над оформлением сайта начинается с компоновки, а заканчивается проработкой акцентов. В HTML и CSS представлен полный набор необходимых для этого средств, особенно что касается заголовков и гиперссылок.

В HTML также представлен целый ряд строковых элементов, позволяющий задать нужные оттенки значения для содержимого этих элементов.

Заголовки и их правильное использование

Как показано на рис. 8.1, в HTML существует 6 уровней заголовков: начиная с `h1` (наиболее важного) и заканчивая `h6` (наименее значимым). Представленные размеры и поля соответствуют стилям по умолчанию браузера Internet Explorer 8. Начинающему дизайнеру сложно применять заголовки: стили, используемые в браузерах по умолчанию для трех наиболее значимых уровней заголовков, задают такой громадный шрифт, что для обычного читателя работа с ними превратится в настоящую муку, равно как и управление верхними и нижними полями заголовков.

HTML и CSS оснащены хорошим дизайнерским инструментарием для решения всех проблем, возникающих у разработчика при работе с заголовками. Когда заголовки должным образом соотносятся с разделами документа, то эти заголовки акцентируют потенциально значимые детали структуры документа и при необходимости позволяют дизайнеру больше сосредоточиться на специальных заголовках, если они есть

Заголовки в печати

Если отбросить на некоторое время моменты, связанные со способами оформления заголовков, то остаются следующие вопросы. Как правильно применять

заголовки для идентификации контента и почему их вообще надо применять? Применение заголовков в печати грамотно разъясняет этот вопрос.

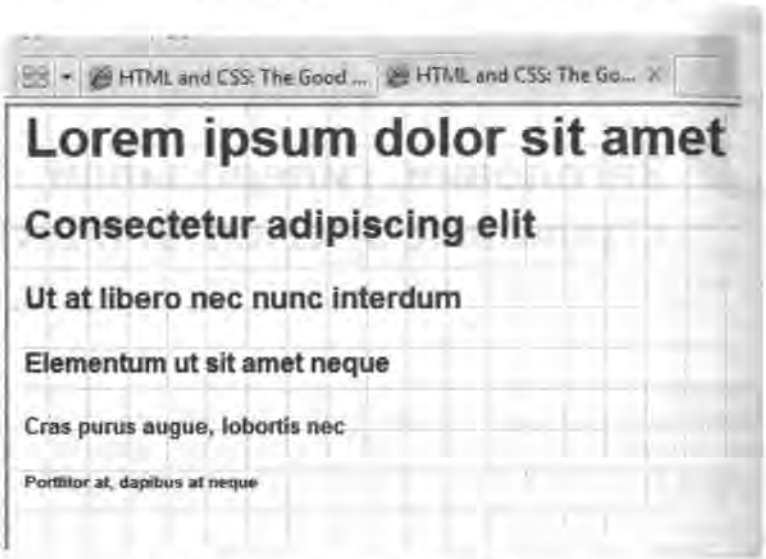


Рис. 8.1. Существует шесть уровней заголовков (так они отображаются в браузере Internet Explorer 7)

Любой печатный контент (если он структурно не минимален) можно разбить на блоки поменьше, которые все вместе и по отдельности можно отметить заголовками. Если книга не художественная, как, например, эта, то каждому крупному разделу книги (состоящему из одной или нескольких глав) назначается заголовок первого уровня, или «A-заголовок» (то есть заголовок `h1`), поскольку название книги в издательском оформлении должно быть представлено своим уникальным заголовком. B-заголовки (`h2`) предназначены для названий глав, а C-заголовки — для основных разделов главы, и так далее. В других книгах разделы оформляются как отдельная книга. Это значит, что у каждого раздела есть титульная страница, а отдельные главы представлены A-заголовками. В любом случае, помимо представления читателю идеи связности разделов документа и четком разграничении его разделов этот подход также разбивает текст на несколько частей для более легкого усвоения.

Распределение разделов и заголовков происходит в контексте книги как цельного документа. Такие элементы, как таблицы данных и иллюстрации, выделяются, но не заголовками. Они выделяются третьестепенными (но вместе с тем значимыми) элементами.

В HTML иерархия заголовков `h1-h6` соответствует заголовкам печатных книг и статей с двумя оговорками:

- *Единственная область, где отображается содержание элемента* `title`, *— это строка заголовка в браузере.*

Это заставляет сознательных разработчиков повторять информацию в тегах h1 (или, возможно, h2) в начале документа, даже если заголовок документа по смыслу «шире», чем текущий раздел.

- *Если данные веб-мастера сайта играют особую роль, когда, например, посетители часто производят поиск по данным веб-мастера, то размещение этих данных в заголовке* h1 *поможет немного повысить популярность сайта*

  Здесь нужно учитывать, что на каждой странице предполагается только один элемент h1, размещаемый в верхней части страницы исходного документа.

Оптимальное размещение заголовка

Как правило, чем выше уровень заголовка, тем больше информации для читателя содержит этот раздел.

Использование заголовков — это хорошее дополнение к применению элементов-«контейнеров», представленных в разделе «Работа с деревом документа» (с. 43) и «Правило 4: Придерживайся своего курса» (с. 80). Любой адекватно использованный элемент-«контейнер» указывает на контекст отрывка и улучшает модульность контента. Во Всемирной паутине главная цель заголовка — идентификация этих модулей на уровне читабельного текста.

Если дизайнер или программист использует заголовки должным образом, то в итоге каждому div-элементу модуля будет соответствовать заголовок определенного уровня.

Поскольку вся эта разметка направлена на структурирование документа, то обычно предполагается, что первый заголовок должен быть h1-элементом, за которым следуют элементы h2, h3 и т. д., детализирующие информацию. Заголовки верхних уровней должны применяться в правильных местах и для соответствующих разделов. Однако сильно не приветствуется пропуск уровней при использовании заголовков нижних уровней. Кроссмедийные инструменты и программы специальных возможностей могут воспроизводить документы в соответствии со структурой документа, а при пропуске уровней заголовков эта функциональность страдает.

Я использую следующий подход: обрамляю заголовок сайта и ссылку на домашнюю страницу элементом h1 (вопреки книгоиздательским стандартам), заголовок или название страницы — элементом h2, а различные заголовки раздела — остальными четырьмя уровнями заголовков.

Даже если вы не используете div-элементы в качестве контейнеров, программное обеспечение использует расположение заголовков для определения структуры документа, что достаточно, по крайней мере, для того, чтобы сгенерировать содержание документа из заголовков.

И последнее, содержимое заголовков (особенно заголовков h1 и h2) очень влияет на поисковые результаты, что сильно варьируется в зависимости от типа поисковой машины.

Оформление элементов заголовка

В отличие от других блочных элементов у заголовков отступы оформляются довольно нетипично. Поэтому работа с заголовками представляет одну из наиболее неприятных проблем для дизайнера.

Размеры и шрифты заголовков

Практически весь высокопрофессиональный дизайн строится на выделении размеров заголовков, отличающихся от установленных по умолчанию стилей браузеров. В немалой степени (давайте смотреть правде в лицо) это объясняется тем, что браузерные стили делают заголовки огромными и безобразными.

Первый шаг на пути создания привлекательных заголовков заключается в выборе шрифта. Нужно определить размер, высоту строки (`line-height`), цвет, поля и отступы для всех заголовков сайта.

По меньшей мере, для заголовков будет задана различная градация размеров, исходя из числа используемых уровней в контенте и целей дизайнера сайта.

Более подробно работа с заголовками рассмотрена в главе 10.

Нормализация размеров заголовка

Перво-наперво, перед началом работы с заголовками дизайнерам необходимо сбросить значения по умолчанию стилей браузера, выставив значения всех блочных элементов и параметра `font-size`. Для блочных элементов я рекомендую что-нибудь такое:

```
h2 { margin: 0 0 1.5em 0; padding: 0; border: 0; ... }
```

Однако это действие само по себе не решит потенциальные проблемы, связанные с разметкой. Скорее всего, придется также учитывать параметр `line-height`. А еще важнее будет проблема размещения заголовков по всей структуре страницы. Если каждый заголовок соединен с контейнером раздела посредством собственных значений, то работа по сбросу значений заголовка упростится. В ином случае, взаимосвязь между заголовками и соседними предшествующими блочными элементами также потребует сброса значений. При строгих требованиях к результату дизайнер может даже задать их сброс в каждом конкретном случае селектором + (следующий потомок) для корректной интерпретации браузерами.

Еще большая проблема возникает при попытке контролировать высоту заголовка на сайтах с жестко заданной сеткой или ограничениями по доступному пространству. Самым простым решением будет выставление точной высоты заголовков и принудительное ограничение длины параметром `overflow: hidden`, но отсутствие гибкости в этом подходе делает его непрактичным для сайтов, где заголовки используются эффективно.

Другая возможность контролировать высоту заголовка в сетке заключается в определении значений `line-height` основного текста, чтобы все размеры заголовков регулировались на одной линии путем варьировании значений `line-height`. Однако подобный подход плохо применим для сайтов, где прокрутка должна быть минимальной.

Еще одно возможное решение заключается в комбинировании размеров и сочетаемости заголовков с целью уменьшения обилия вариантов по размерам заголовков (и, таким образом, базовой линии шрифта).

Оптимальный подход по контролю высоты заголовков варьируется в зависимости от проекта и дизайнерских предпочтений. Вместе с тем управление высотой заголовков является отличным показателем необходимости эффективного сотрудничества между дизайнерами и разработчиками контента.

Выделение заголовков

В заголовках сайта могут также использоваться линии произвольной длины, цвета и стиля, размещаемые вместе с блоком копии, как правило, на одном из полей, а иногда и на базовой линии заголовка. Простые сплошные линии, размещаемые в поле заголовка, задаются достаточно простым путем — установкой свойства заголовка `border-bottom` (или `border-top`), например `h3 { border-bottom: 1px solid rgb(85,170,255); }`.

Большая часть других способов выделения, подходящих для заголовков, регулируются путем применения фоновых изображений.

Разметка ссылки

Все примеры ссылок, представленные в этом разделе, являются произвольными гиперссылками. Если вам нужна более подробная информация по атрибуту `link`, загляните в главу 3.

Группа строковых элементов в HTML добавляет нюансы значений словам, фразам или коротким отрывкам в более длинных блоках контента. Наиболее известные группы не только создают гиперссылки (ради которых эти группы, собственно, и затеваются) и элементы форм, но и применяются для расстановки выделений.

Элементы форм представлены в главе 11, а о семантических строковых элементах речь пойдет в конце этой главы.

Разработке ссылок уделяется пристальное внимание по ряду причин.

- Ссылки представляют собой абсолютно интерактивный элемент. При активации ссылки пользователь создает новое событие. Уже по одной этой причине ссылки должны постоянно с легкостью выделяться из окружающего контента.

- Оформленные в виде ссылок дизайнерские решения можно настроить под определенные цели посетителя сайта.
- Браузерные стили оформления ссылок основаны на устаревших идеях, которые практически не применяются в разработке современных сайтов

Атрибуты ссылок

У элементов ссылок есть следующие значимые атрибуты:

- `href`

  Основа основ веб-приложения, указывающая адрес ссылки и избавляющая пользователя от необходимости вводить или вставлять URI-ссылку в адресную строку браузера.

- `target`

  Значение этого атрибута представлено именем окна, таблицы или рамки, в которые должен загружаться адрес ссылки. Этот атрибут не поддерживается строгими шаблонами DTD, относящими сопутствующую функциональность к поведенческому уровню сайта. И еще этот атрибут представляет существенный компонент ссылок в контексте параметра `frame`, представленном в главе 12.

- `rel`

  Как и в случае с элементами `link`, этот атрибут вкратце описывает связь между текущим документом и адресом ссылки. Чаще всего в гиперссылках его используют для назначения спецпараметра `nofollow`, говорящего поисковым машинам не индексировать адрес URI-ссылки. Функция `nofollow` была разработана компанией Google как средство защиты владельцев сайтов от спамерских ссылок в генерируемом пользователем контенте. Функция `nofollow` присваивает нулевое значение спамерским ссылкам, выявленным в результате выполнения алгоритма ранжирования. Необходимо учитывать, что при возможном исключении параметра `nofollow` используемый в гиперссылках атрибут `rel` не обладает какими-либо универсальными согласованными значениями.

Существуют также универсальные атрибуты, среди которых особенно значимым для ссылок является атрибут `title`.

Эффективное применение атрибута href

Этот элемент используется повсеместно и его очень легко недооценить. Задашь значение этому атрибуту и работаешь дальше. Тем не менее при создании ссылок необходимо следовать четырем правилам.

- *Точность.*

  Ошибочные ссылки — очень неприятная вещь. Поэтому очень важно проверять и активировать ссылки на промежуточных этапах перед запуском сайта.

- *Пригодность.*
  Структурирование URI-адресов происходит в соответствии с четкими стандартами (IETF RFC 2396, http://www.ietf.org/rfc/rfc2396.txt), задающими компоновку адресного контента. При создании пригодного документа в URI-адресах необходимо избегать определенных символов. Этот процесс называется URL-кодировкой и представлен в разделе «Кодирование символов в URL: сущности ASCII», с. 282.

- *Актуальность.*
  Со временем веб-документы исчезают или перемещаются, что приводит к так называемым «ломаным ссылкам». Для эффективного закрепления гиперссылок необходимо, чтобы владельцы сайта задавали документам переадресацию и устраняли разные неисправности, вроде вызывающих перемещение документов, затирание или исчезновение полезных данных. Но, к сожалению, эффективное закрепление гиперссылок встречается крайне редко. Так или иначе, кто-то должен отвечать за актуальность ссылок. Я рекомендую проводить специальную ежеквартальную проверку ссылок на актуальность, если такое возможно. Проблемы актуальности ссылок и их решения, например выполнение автоматических проверок ссылок, представлены на сопутствующем сайте этой книги.

- *Краткость.*
  Если оптимизация поисковых систем не вызывает проблем, то значения `href` должны быть максимально краткими и при этом не скрывать дополнительную информацию об источнике адресного контента. Краткие значения `href` улучшают точность и снижают вероятность ошибок ввода.

Создание ссылок на специальные области документа

Если ваше знакомство с HTML начиналось в давние времена, то, скорее всего, вам знаком атрибут `name`, который более подробно будет представлен в главе 12. Современные методы упрощают значения строковых адресов ссылок посредством значений `id`. Поэтому кодовая строка

```
<a href="http://www.example.com/docs/swallow_airspeed.html#european" ... >
```

задает автоматическую прокрутку браузера до элемента со значением `id=european` при активации ссылки и завершении загрузки страницы `swallow_airspeed.html`.

Если посетители захотят узнать про наличие адреса данной ссылки (например, адреса постоянной ссылки), то можно воспользоваться одним популярным методом, работающим в большинстве современных браузеров. Нужно добавить ссылку на сам документ в конце первичного блочного элемента, применить эту ссылку к соответствующему символу, а также добавить примерно такие стили и разметку:

```
      p.containsInlineLink a { color: #fff; }
          /* same as background color,actually */
p.containsInlineLink:hover a { color: #999; }
          /* some contrast, but not as much given to the body copy */...
   <p class="containsInlineLink" id="european">
          The airspeed of an unladen European swallow is the subject
of extensive speculation, much of it ironic. <a href= "#european"
title="Permalink to the discussion of the airspeed of an unladen European
swallow.">&rarr;</a></p>
```

При такой структуре знакомые с сайтом пользователи смогут скопировать значение `href` строковой ссылки для вставки на свою веб-страницу.

Эффективные названия ссылок и значения заголовка

Назвать ссылку лучшим образом может быть сложно по ряду причин.

- Многие ссылки представляют собой часть интерфейса сайта, а не его контента, побуждающий дизайнеров насыщать ссылки идеограммами или максимально краткими ключевыми словами.
- На страницах, разработанных специально для упрощения процесса покупки или создания учетной записи, многие ссылки будут (в хорошем смысле слова) представлять собой призыв к действию (например, «нажмите тут»), что не очень грамотно с технической точки зрения.
- Иногда сайт (или часть сайта) будет разрабатываться, исходя из предположений о желании пользователя просматривать контент сайта в специальном контексте.
- В качестве названия ссылки дизайнер может выбрать логотип или какое-нибудь другое изображение.

При устранении этих факторов оптимальное название для ссылки приобретает три качественных признака из четырех, представленных при описании значений `href`: краткость, точность, актуальность (вот в таком порядке).

Обстоятельства, делающие невозможным создание полноценных описательных ссылок, приведут вас к атрибуту `title`, который превосходно подойдет пользователям, применяющим специальные вспомогательные интерфейсы.

При работе по вышеизложенному сценарию возможно появление следующих значений `title`, задающих эффективную ссылку:

- *Ключевые слова интерфейса.*

```
<a href="/contact/" title="Telephone numbers, e-mail addresses, and
driving directions.">Contact</a>
```

- *Призыв к действию.*

```
...To add your name to our mailing list, <a href="/promo/signup.php"
title="Mailing list signup form.">click here</a>.
```

- *Цельный контекст.*

```
<a href="/gallery.php?user=persist1&imgfile=DSC2112.JPG"
title="See "Playing Frisbee" in Ben"s "Day at the Dog Park"
photoset."><img src="/persist1/images/DSC2111.JPG" alt="A Shih Tzu makes
scary faces at the camera." class="lggallerypic" /></a>
```

- *Логотип в качестве ссылочного контента.*

```
<h4>This item is available at:</h4>
<ul>
...
  <li><a href="http://www.powells.com/" title="Buy this item at powells.
com."> <img src="/images/powells_logotype.png" alt="Powells.com" /></a></
li>
```

Особое внимание уделите примеру «призыв к действию», который, как правило, основан на контекстном значении элемента `title` для ссылки. Используемый здесь авторский подход часто приводится как отрицательный пример, но при всем при этом эффективные адресные страницы должны содержать краткий призыв к действию. По возможности дизайнеры должны придумывать более лаконичные фразы, чем «жмите сюда».

Дополнительное преимущество атрибута `title` заключается в грамотном применении, позволяющим суммировать программными средствами контент документа и исходящие ссылки. Как и заголовки, свидетельствующие о более объемной структуре документа, значения атрибутов `title` свидетельствуют о более объемном уровне контента.

Помимо представленных выше преимуществ у атрибута `title` существует ряд ограничений. Основное ограничение заключается в недоступности атрибута людям, использующим для поиска страниц клавиатуру. Этот недостаток может быть исправлен (после подгонки) только посредством специальных настроек JavaScript. По этой и другим причинам важный для освоения читателем контент всегда должен быть представлен в обычном виде.

Оформление ссылок

Ссылки, как наиболее интерактивные пользовательские элементы, даже без скриптовых преимуществ заслуживают самого пристального дизайнерского внимания.

Когда ссылки отображаются браузерами с использованием стилей по умолчанию, у ссылок появляется два значительных недостатка:

- Автоматически назначаемые ссылкам цвета выглядят просто безобразно и совершенно не подходят к большинству цветовых палитр.
- Ссылки как строковые элементы сами определяют занимаемое ими пространство. Если не будет приемлемого способа заполнения важных ссылок оптимальным количеством контента, то обнаружить и активировать эти

ссылки пользователям будет нелегко. Либо ссылки могут оказаться непомерно большими и не смогут уместиться в одной строке.

В отличие от большинства других элементов для оформления ссылок требуется больше усилий из-за их интерактивности. Большая часть этой работы связана с псевдоклассами.

Псевдоклассы ссылок

Поскольку ссылки могут быть представлены в нескольких видах, то вполне логично, но не совсем реально назначать стили ссылкам исключительно селектором элемента.

Для учета различных видов ссылок CSS предлагает псевдоклассы `:link`, `:visited`, `:hover` и `:active`. Первые два описывают активные ссылки и, соответственно, ссылки, указывающие на уже посещенные адреса. Два других псевдокласса описывают ссылку с наведенным на нее курсором и нажатой кнопкой мыши.

В таблице стилей селекторы со стандартной областью действия, включающей эти псевдоклассы, должны применять их в порядке `:link-:visited-:hover-:active`.

Такой порядок обусловлен приоритетностью селекторов. Поначалу ссылки не посещаются, но со временем ситуация меняется. И в любом случае не будет смены стиля ссылки при наведении курсора мыши.

Скорее всего, вы захотите отделить тип наведенных ссылок от других типов. И, само собой, ссылка не сможет быть активирована курсором, только если ей не назначен соответствующий стиль.

Если бы вы воспользовались неправильным вариантом и разместили бы псевдоклассы в порядке, обратном предполагаемому, то наведенный и активный стили всегда бы игнорировались движком обработки, поскольку в этом порядке удобному стилю отдается преимущество по сравнению с наведенным и активным стилями, даже если один из последних стилей — подлинный.

Не существует каких-либо особых ограничений на пары свойство/значение, назначаемых селекторами до псевдоклассов.

Существует одно важное исключение в присвоении приоритета селекторов. Если свойство установлено правилом, которое управляет элементом `a` без каких-либо псевдоклассов, то конфликтные правила на это свойство в правилах с псевдоклассами в других селекторах не будут применяться браузерами Firefox и Internet Explorer. Другими словами, если применяется правило `a { color: #00f; }`, то `a:visited { color: #808; }` будет игнорироваться. Если вместо этого появляется команда `a:link { color: #00f; }`, то ссылка `a:visited` не игнорируется.

Наконец, CSS задает применение псевдокласса `:focus`, когда элемент находится в активном состоянии, но при этом фактически не активирован. Этот элемент

не поддерживается во всех версиях Internet Explorer, кроме версии IE 8. Элемент `:focus` применяется чаще, когда пользователь полагается на клавиатуру или уводит курсор мыши со ссылки. Этот псевдокласс нужно размещать между селекторами `:hover` и `:active`.

Применение display: block для улучшения внешнего вида ссылки

Когда ссылка представляет собой часть интерфейса сайта (например, систему поиска) или обозначает призыв к действию, то оформление ссылки как кнопки может отлично улучшить интерфейс.

Первый шаг в достижении этого эффекта заключается в назначении ссылке элемента `display: block`. При изменении значения `display` внешний вид ссылки управляется элементами `width` и `height` для ширины и высоты, а также возможно использование свойства `padding` или различных фоновых свойств посредством функции Замены Изображений по методу Фарнера.

ОФОРМЛЕНИЕ ССЫЛОК ЭЛЕМЕНТАМИ :HOVER И :ACTIVE

Когда начинаешь работать с CSS, порой очень хочется поменять жирность, кегль текста или размеры ссылок, активно используемых пользователями.

Буду краток: *не надо*. Стили, меняющие область элемента без предупреждения, нарушают компоновку всей страницы. При этом взаимодействие с сайтом или даже просто прокрутка страницы может стать для пользователя настоящей пыткой.

У этого правила могут быть исключения, если обе ссылки и окружающий их контент намертво прикреплены к статичным областям макета, но отличные результаты в этом плане достигаются только при высоком мастерстве дизайнеров и программистов.

Перед реализацией подобных ссылок в интерфейсном стиле дизайнерам необходимо учитывать следующие факторы.

- *Значение свойства* `display` *у контейнера.*

  Для достижения желаемого результата эти ссылки нужно размещать в области элементов `block` или `inline-block` со значением `display`.

- *Статичная и свободная компоновка.*

  Фоновые изображения (либо исключительно для фонов, либо будучи частью функции замены фоновых изображений по методу Фарнера) требуют минимум усилий для размещения элементов при статичной компоновке. Однако в пропорциональной или сеточной компоновке элементы оптимально центрируются в рамках своей ссылки.

- *Расчет размеров ссылки.*

  При оптимальном варианте ссылкам задается 100%-я высота. Если требуется более пристальное внимание к деталям, то размеры, границы (если таковые есть) и параметр заполнения ссылок рассчитываются и задаются отдельно.

Методы форматирования ссылок также обсуждались при рассмотрении поисковой системы в главах 5 и 7, а поведенческие свойства курсора представлены на сопутствующем веб-сайте этой книги.

Свойство text-decoration

В ряде случаев полезно размещать строки под, над и внутри небольших кусков текста.

У свойства `text-decoration` есть следующие значения:

- `underline`

  Выставляемое по умолчанию значение для элементов `a` и `ins`

- `line-through`

  Выставляемое по умолчанию значение для атрибута `del`

- `overline`

  Редко применяемое аналоговое значение `underline`

В CSS 2.1 также упоминается значение `blink`, но его поддерживает только браузер Firefox при специальном включении этого значения через панель `about:config`.

Свойство cursor

CSS дает доступ к объектам курсора мыши из библиотеки своей операционной системы. Доступ к объектам происходит посредством свойства `cursor`, значение которого по умолчанию выставлено на `auto`. Специальные значения включают:

- `default` (стрелка-указатель);
- `pointer` (значение по умолчанию для ссылок);
- `crosshair`;
- `text` (значение по умолчанию для обычного текстового контента);
- `help`;
- `progress`;
- `wait`.

Из всех этих значений наиболее часто используется указатель `help` с просьбой подождать всплывающей подсказки для элемента. Указатель также можно использовать для обозначения ссылок, ведущих к разделу «Часто задаваемые вопросы» и другим ресурсам поддержки.

Использование других значений оптимально подходят для веб-приложений, а эффективность подобного использования должна тестироваться пользователями.

Добавление семантических значений строковыми элементами

Помимо ссылок существует ряд других строковых элементов, добавляющих контенту оттенки значения. Все эти элементы представлены в табл. 8.1.

Таблица 8.1. Обзор встроенных элементов HTML 4

Элемент	Полное название	Форма воспроизведения	Примечания
Em	Выделение	i, u	
Strong	Двойное выделение	b	
Cite	Цитата из источника	i	Применяется для названий книг, периодических изданий, телевизионных программ и объемных аудио/видео материалов. Для других имен собственных не применяется.
Code	Прогон кода	tt	См. kbd, passage
Kbd	Данные, вводимые пользователем с клавиатуры	tt	
Samp	Образец выходных данных	tt	По правилам HTML 4.01 образец отличается от кода тем, что кодовый контент подразумевает выполнение команд, а образец отсылает к выходным данным
abbr	Аббревиатура	[нет]	
acronym	Сокращение	[нет]	Сокращения и аббревиатуры распространяются уникальными способами. В каждой стране существует свой способ форматирования сокращений.
sup	Верхний индекс	[нет]	Оптимальное значение элемента line-height — минимальное
sub	Нижний индекс	[нет]	
ins	Вставка	[нет]	По умолчанию оформлено подчеркиванием
del	Удаление	strike	Лучше всего употреблять после контента с элементом ins там, где предполагается наличие «забоев и наложение знаков»
q	Строковые кавычки	[нет]	Программное поведение элемента зависит от браузера. Для длинных отрывков с кавычками предпочтителен элемент blockquote
var	Переменная; иррациональное число	i	Например, E.g., 2πr, (sin2θ + cos2θ) = 1
dfn	Определение	i и em (в контексте)	Разметка в рамках элемента definition играет роль дополнения к спискам определений («Списки определений», с. 149). Элемент definition особенно эффективен с элементом id для создания адреса ссылки на себя.

Большинство элементов в табл. 8.1 достаточно просты, особенно с учетом браузерных стилей по умолчанию. Но есть два элемента, которые не имеют стилей по умолчанию и заслуживают особого внимания. Это элементы `abbr` и `acronym`.

Как отмечалось ранее, иногда пользователи будут просматривать веб-контент вне контекста. При использовании технических и сленговых аббревиатур и сокращений такой просмотр становится проблематичным, поскольку содержание сайта становится непонятным посетителю, не знакомому с контекстом этих аббревиатур и сокращений.

С другой стороны, если подключить подсказки к сокращениям и аббревиатурам путем применения значений параметра `title`, то сайт станет более доступным для всех пользователей. В таблицу стилей последних версий браузера Firefox были включены следующие правила:

```
acronym, abbr { border-bottom: 1px dotted; }
```

К таким правилам принято добавлять элемент `cursor: help`, несмотря на то что в данной ситуации применяют его редко.

Цитаты

Для обозначения цитат в HTML используется строковый элемент `q` и блочный элемент `blockquote`. В спецификации HTML 4.01 говорится о поддержке этих элементов атрибутом `cite`, что означает наличие идентификатора URI, указывающего на первоисточник. На практике это требование игнорируется всеми браузерами, кроме Internet Explorer 8. Поэтому пользовательские цитаты приходится втискивать в элементы `cite` и `a`. Но в целях сохранения метаданных необходимо по-прежнему присваивать значения URI, допустимые для атрибута `cite`.

У элемента `blockquote` есть три характерных признака: в браузерных стилях к этому элементу применяют видимое значение `margin-left`. Кавычки должны специально добавляться к контенту с таким значением, а строгий DTD-шаблон требует наличия в контенте как минимум одного блочного элемента (обычно параграфа).

Элемент разметки «кавычки» дает ограниченную поддержку псевдоэлементам `:before` и `:after`, не поддерживаемым всеми версиями Internet Explorer, кроме IE 8. В браузерах, поддерживающих псевдоэлементы `:before` и `:after`, открытие и закрытие кавычек представлено в таблице браузерных стилей для контента элемента `q`. Ни в одной среде элемент `blockquote` не обладает таким преимуществом. Для добавления типографских кавычек к браузерным элементам `blockquote` в стилевой блок необходимо вставить следующие строки:

```
blockquote:before { content: open-quote; }
blockquote:after { content: close-quote; }
```

Псевдоэлементы :before и :after используются как обычно в той же форме, что и в q-элементе. Без свойства content эти псевдоэлементы бесполезны. Суть в том, что свойство content может применяться только с этими селекторами.

Значения свойства content может также использоваться для отмены браузерных установок по умолчанию как те, что заданы для q-элемента, что позволяет производителям контента заново насыщать контент буквенными символами. Этот продуктивный метод более затратный, но он улучшает совместимость с требованиями вспомогательных технологий.

Свойство content последовательно избирает значения open-quote, close-quote или произвольно заключаемой в кавычки строки символов ASCII. Необходимо избегать более объемных символов, универсальных буквенных кавычек и кодированных символов перевода строки. Подробнее об этом требовании рассказывается в главе 12.

9 Цвета и фоны

Большинство из нас воспринимают цвет как нечто само собой разумеющееся. Для веб-профессионала цвет таит в себе множество нюансов. Опытным разработчикам нужно знать, какая существует взаимосвязь между цветовыми данными и их кодовыми аналогами. В итоге комбинация знаний по веб-цветам становится более изощренной и субъективной. Веб-цвет — это совокупность чисел и искусства. И тут никогда не помешает знать оба аспекта.

С фонами существуют свои проблемы. Простейшая задача заключается в назначении однородного цвета для элемента. Но когда в игру вступают изображения, то вопросов в выборе фона становится еще больше.

В этой книге нет детального рассмотрения графики и цветов, но по данной теме существует много хороших материалов. На сопутствующем сайте этой книги даны ссылки по этой тематике. Кроме того, я рекомендую вам прочитать две книги: «Web Design Guide» Sara Norton, Patrick Lynch (3-е изд., изд-во Yale University Press) и «Painting the Web» Shelly Powers (изд-во O'Reilly).

Теория цвета и практика применения веб-цветов

В CSS 2.1 существует целый ряд свойств, задающих цветовые значения для фонов и границ. Существует также свойство `color`, задающее цвет текста и других элементов.

Удобство, доступность и цвет

При назначении цветов и фоновых цветов элементам следуйте таким правилам.

- Свойство `color` должно всегда следовать после свойств `background` или `background-color`. Это необходимо для обеспечения корректной работы «странных» браузеров.

- При использовании фоновых изображений необходимо специально назначить соответствующий фоновый цвет, что также повысит надежность.
- Доступность становится максимальной при сильном контрасте между цветом элемента и его фоном.

Нужно также учитывать остроту зрения посетителей старшего возраста. С возрастом у людей глаза становятся менее восприимчивыми. Им становится все труднее различать цвет и детали. Утрата этой функции особенно ощутима на близких расстояниях и является результатом состояния, называемого *пресбиопией*.

Дальтонизм — еще один вид нарушения зрения, который следует учитывать при создании дизайна с резким контрастом.

- Наиболее распространенная форма дальтонизма — неспособность отличить зеленый цвет от красного. Подобное отклонение встречается как минимум у 7 % американцев.
- Функционально эффект дальтонизма состоит в том, что определенные световые волны полностью исключаются из спектра. Инфракрасные и ультрафиолетовые световые волны не воспринимаются человеческим глазом напрямую. Пользователям-дальтоникам не хватает диапазона обычного оптического спектра.
- Посетители, плохо различающие красный цвет, видят его зеленым с сильными оттенками красного или синего. В случаях полной дисфункции человек вообще не видит красный цвет, что может также нарушить восприятие яркости других цветов.
- Остальные волны, которые видит дальтоник, могут давать сочетание, воспринимаемое как белый свет. Тем не менее тот же световой спектр у людей с обычным световым восприятием будет шире. Верно и обратное относительно черного цвета.

Ссылки на симуляторы дальтонизма и другие тесты контрастности представлены на сопутствующем сайте этой книги.

Аддитивная цветовая модель

Термин аддитивный цвет появился как описание процесса получения цвета за счет соединения нескольких материалов.

Этой модели обучают в школах искусств, так как ее применяют при рисовании. Синий, красный и желтый цвета с оттенками белого или без него смешивают и получают следующие цвета:

- синий + красный = пурпурный;
- синий + желтый = зеленый;
- красный + желтый = оранжевый.

К этим цветам можно добавить черный или белый пигмент (классический, с оттенком древесного угля или обработанного свинца соответственно) для

моментального создания нужного цвета. В цветной печати яркость цвета задается пигментным покрытием (например, чернила, тонер) и его насыщенностью.

Цветовая модель HSB

Для быстрой идентификации цветов при отсутствии навыков работы с красками наиболее доступной является модель HSB (тон-насыщенность-яркость). Эта модель предоставляет всю палитру *оттенков* (от красного до фиолетового с тончайшим цветовым спектром). Поскольку основных цветов — 3, согласно обозначениям HSB модели, 0 обозначает красный цвет, 120 — зеленый, а 240 — синий.

При идентификации цветов по значениям модели HSB важно помнить следующие правила.

- Значения тонов меняются *по принципу радуги* с красным тоном внизу и фиолетовым наверху.
- Значения насыщенности меняются от нейтральных до максимальных в определенной точке цветового спектра. Для белого цвета или оттенков серого цветовая насыщенность *должна* быть нулевой.
- Значения яркости меняются от 0 до 100%. Нулевое значение яркости всегда дает черный цвет, а яркость на 100% — белый цвет или любой другой спектр цветов.

Субтрактивная цветовая модель

В Соединенных Штатах аддитивной модели обучают в начальной школе, а субтрактивной цветовой модели — на уроках физики в старших классах. Один этот факт должен говорить вам о сложности модели субтрактивных цветов.

В физическом мире все цвета отражаются. Пигмент (специально накладываемый на основу, представленный в естественном виде на поверхности или производимый природными процессами) *абсорбирует* световые волны, *не* соответствующие его цвету. Чем темнее или насыщеннее цвет, тем меньше света он отражает. Отсюда и термин — «субтрактивный» цвет.

Однако современные дисплеи работают по принципу излучения, а не отражения. Электронно-лучевая трубка (ЭЛТ) или жидкокристаллический экран проецирует заданный пиксел в узком спектре красного, зеленого или синего с различной степенью интенсивности. Белый цвет — это результат работы всех трех цветовых каналов с максимальной интенсивностью, а черный — результат работы цветовых каналов с минимальной интенсивностью.

Если два канала работают с полной интенсивностью, отображаются вторичные цвета:

- красный + зеленый = желтый;

- зеленый + синий = сине-зеленый;
- синий + красный = пурпурный.

Третичные цвета получаются при работе всех трех цветовых каналов с разной степенью интенсивности.

24-битная цветовая палитра, в настоящий момент поддерживаемая веб-браузерами, предоставляет дизайнеру более 16 миллионов цветов на базе 256 оттенков по каждому каналу. Чем ближе значения комплексного цвета к нулю, то есть rgb(0,0,0), тем темнее цвет. Значения 256 оттенков серого легко отличить от других цветов, поскольку их канальные значения всегда одинаковые.

Более подробная информация о субтрактивных цветах представлена на сопутствующем сайте этой книги.

Дизайн, контраст и дополняющие цвета

Хорошо спроектированный по части дизайна сайт, как правило, строится на дизайнерских брифах и документации с четкими требованиями, суммирующими многочисленные известные факты и мнения по целевой аудитории и бизнес-задачам сайта. *Функциональная* природа дизайна становится значимым двигателем процесса: в отличие от изобразительного искусства графический дизайн пытается донести лаконичные идеи и помочь его владельцу в достижении конкретных целей или решении определенных задач. Эстетика играет важнейшую роль в дизайне, но она целиком и полностью состоит в том, чтобы транслировать идеи, стоящие за дизайном.

Идеальный вариант — задания получены, есть договоренность о принципах построения сайта, определены основные идеи и другие дизайнерские директивы. При этом учитываются ожидания клиентов и продавцов относительно целевой аудитории и бизнес-задач.

Важнейший визуальный компонент дизайна любого сайта (и его скрытых мотивов) — цветовая *палитра*: цвета используются дизайном для передачи идей. В палитре почти всегда представлено три цвета или три оттенка серого: для основного цвета, фонового цвета и выделения элементов. Встречаются также палитры из шести или более цветов.

При обсуждении визуальной дисфункции упоминалась идея о желательном применении контрастов. Эта идея также отлично подходит для пользователей с полноценным зрением. Чтобы достичь высокой контрастности, дизайнеры очень часто используют *дополняющие цвета* для основного и фонового цветов. Дополняющий цвет — это тоновое или цветовое дополнение. Дополняющий цвет тона можно обнаружить на противоположном краю цветовой палитры, представленной HSB моделью. Например, желтый цвет дополняет синий. Дополняющие цвета также придают насыщенность и яркость. Их можно обнаружить в субтрактивной модели путем перестановки значений цветового канала, как показано в табл. 9.1.

Таблица 9.1. Примеры дополняющих цветов в субтрактивной/RGB модели

Базовый цвет			Дополняющий цвет		
Красный	**Зеленый**	**Синий**	**Красный**	**Зеленый**	**Синий**
255	255	255	0	0	0
255	255	0	0	0	255
51	102	153	204	153	102
43	61	21	212	194	234
143	34	69	112	221	186
192	114	12	63	141	243

Инверсия обозначает полученное значение при вычитании первого значения из 255 (максимального десятичного значения для 24-битного цветного канала). Если вы посмотрите на два значения для каждого канала в представленном ряду, то увидите, что в сумме они дают 255.

Идентификация цветов, вкратце

Вы можете быстро идентифицировать цвета как по изображению, так и по фрагментам правил из таблицы стилей. К сожалению, единственный способ сделать это — воспользоваться старой доброй практикой. Далее вы можете почерпнуть изложенные ниже техники идентификации.

- Опытный дизайнер может взглянуть на шестнадцатеричное значение из шести цифр и без особого труда распределить его на три канала.
- Способность в уме конвертировать шестнадцатеричные числа в десятичные — это хорошо, но вряд ли необходимо. Если вы сможете привыкнуть к мысли о том, что A, B, C, D, E, F — это цифры, то справитесь. Это как у Тома Лерера: «Не паникуйте, база шестнадцать — это все равно, что база десять, если у вас есть еще шесть пальцев».
- Чем выше значение цвета по всем трем каналам, тем светлее цвет.
- Чем больше совокупность пропорциональных различий между канальными значениями, тем больше насыщенность цвета.
- Умозрительное разграничение цветового спектра на части между первичным и вторичным цветами часто упрощает идентификацию цвета. Все остальное — чистая арифметика.
- Если по-другому разграничить не удается, воспользуйтесь инструментом Пипетка (Eyedropper), копированием и вставкой.
- Может также помочь расширение ColorZilla для браузера Firefox, представленное на рис. 9.1. После установки ColorZilla размещает пиксел в левом углу строки состояния браузера Firefox, превращающий при активации курсор мыши в инструмент Пипетка (Eyedropper). Пользователи компьютеров Macintosh также могут запустить аналогичное (хоть и менее

распространенное) приложение, называемое DigitalColor Meter, которое можно найти в папке Утилиты.

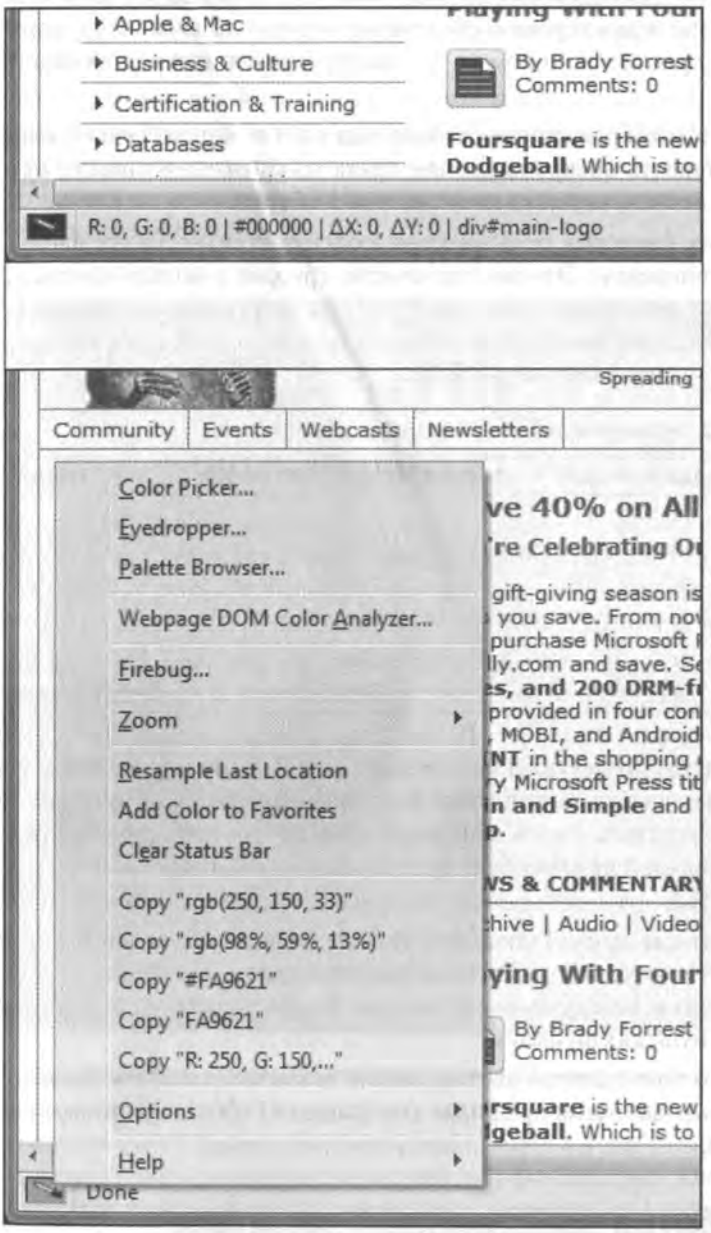


Рис. 9.1. Активация инструмента Пипетка (Eyedropper) приложением ColorZilla, позволяющим создавать образцы цветов и копировать их

Графические приложения и палитра, безопасная для Сети

В разделе «Единицы измерения CSS» на с. 57 описано все многообразие значений размера зерна экрана и объяснены методы включения цветовых значений в таблицу стилей, но в нем не говорится о результатах применения таких методов.

Технологии отображения воспроизводят цвет с учетом среды воспроизведения. Реалистичность воспроизведения цвета изображения зависит от года производства дисплея, качества компонентов и сборки.

Из-за этих факторов большинство людей частенько будут наблюдать неточную цветопередачу. Это обстоятельство говорит в пользу шестнадцатеричных цветовых значений, минимальные доли которых содержат спектр *веб-палитры*. Для создания так называемых веб-цветов используйте значения интенсивности каналов, представленные в табл. 9.2.

Таблица 9.2. Значения интенсивности каналов для веб-цветов

Шестнадцатеричный	Восьмибитный десятичный	Проценты
00	0	0
33	51	20
66	102	40
99	153	60
cc	204	80
ff	255	100

Веб-палитра, представленная спектром из 216 цветов, в 1990-е годы воспроизводилась слабыми графическими видеокартами. Сегодня эта цветовая палитра не утратила своей значимости, но по другой причине: ЖК-мониторы старых моделей не способны полноценно воспроизвести 24-битный градиент (см. рис. 9.2).

HD-мониторы и ЭЛТ-дисплеи, подключенные к хорошей видеокарте, как правило, не страдают дефицитом цветопередачи, в отличие от более дешевых ЖК-дисплеев, но в лучшем случае используют только половину разработанных пользовательских функций.

Проблемы полноценной цветопередачи сказываются на изображениях и встроенных в них цветовых профилях (см. раздел «Работа с цветовыми профилями» на с. 214).

Разработка собственной палитры

Если вы выполняете заказ или работаете на полной ставке в компании, то есть большая вероятность того, что, по крайней мере, пара цветов из палитры уже

для вас выбрана. Есть два популярных системных метода для балансировки вашего выбора: математические подсчеты и применение базовых цветов.

Математический подход создает пары, тройки и четверки: наборы тонов с базовым цветом. В случае с тройками можно предположить, что два цвета в палитре уже выбраны. Тогда давайте выберем вечно популярные синий и зеленый цвета, 205° и 105°, сине-зеленый и бледно-зеленый.

Биссектриса проходит в районе 155°, в области с аквамариновым или зеленоморским оттенком. Этот тон или дополняющий его цвет (335°, фиолетовый) используются в качестве третьего тона в тройке. Проблема заключается в выборе цвета к этому тону с совместимой насыщенностью и яркостью и применении этого цвета к вашему дизайну.

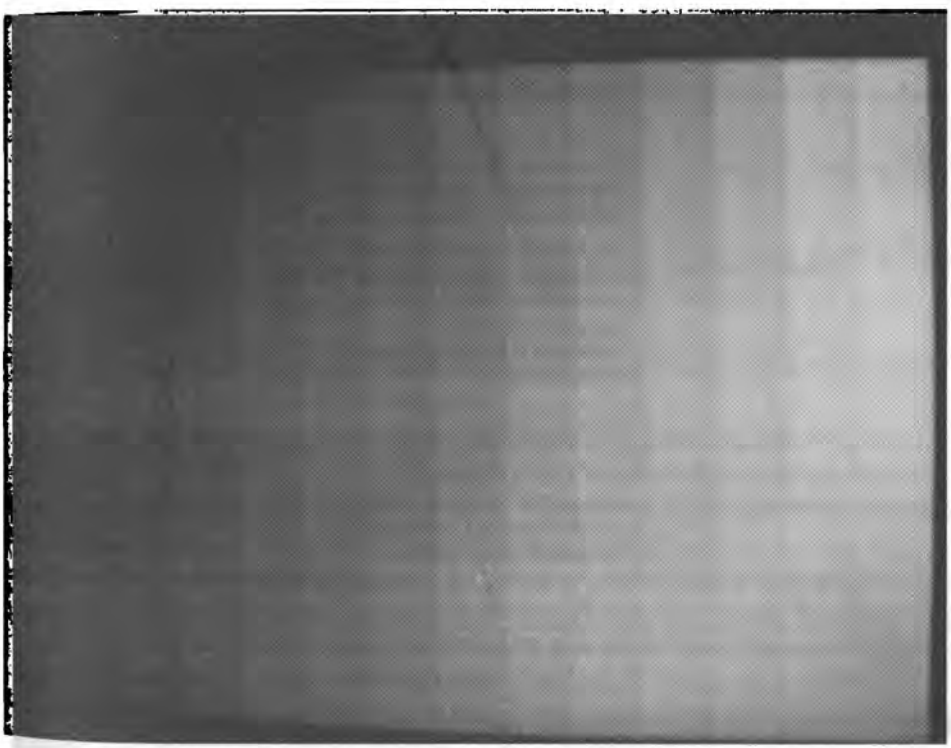


Рис. 9.2. Старые модели ЖК-дисплеев плохо отображают градиенты

Наиболее приемлемый вариант по сравнению с игрой в углы — использование базовых цветов. Для этого нужно взять красивую фотографию, выбрать из нее цвета с помощью инструмента **Пипетка** и определить для себя, какие из цветов составят оптимальную палитру.

Тема создания палитры более подробно отображена на сопутствующем сайте книги.

Фоны CSS

Вы можете использовать фоновые изображения во многих местах. Фоновое изображение можно применить к любому элементу с моделью вывода бокс.

Лучше всего начать с обзора CSS свойств, влияющих на внешний вид фона, как показано в табл. 9.3.

Таблица 9.3. Фоновые свойства CSS (Значения по умолчанию представлены полужирным начертанием)

Свойство	Назначение	Значения
`background`	Краткое свойство, свойство составных значений/пар значений	
`background-attachment`	Не дает прокручивать по контенту фоновые элементы с полосами прокрутки	• `fixed` • `scroll`
`background-color`	Определяет фоновый цвет	• `[color]` • `transparent`
`background-image`	Определяет фоновое изображение для элемента; накладывается поверх любого фонового цвета	• `none` • `[url]`
`background-position`	Содержит горизонтальные и вертикальные величины, разделенные пробелом. Определяет взаимосвязь между краями элемента и краями фонового изображения	• `[length]` • `[percentage]` • `bottom` • `center` • `left` • `right` • `top`
`background-repeat`	Определяет направляющие для потенциального копирования фонового изображения	• `no-repeat` • `repeat` • `repeat-x` • `repeat-y`

Свойство background-position

При установке значения свойству `background-position` в нем должны присутствовать два компонента: смещение влево и смещение вверх. При использовании таких единиц длины, как `px` или `em`, результаты будут однозначные: фоновое изображение смещается в обозначенные координаты элемента. При отрицательном значении левый верхний угол «первого» фонового изображения будет наложен вне зоны видимости.

Если вместо этого используются процентные показатели или значения ключевых слов, характер свойства `background-position` будет меняться. Вместо

размещения изображения на заданном расстоянии от верхнего левого угла его элемента механизм обработки выстраивает установленные координаты изображения с установленными координатами элемента. Значение 33% 33% выровняет координаты изображения на одну треть от его верхнего левого угла без изменений координат элемента. Это «взаимное расположение» представлено на рис. 9.3.

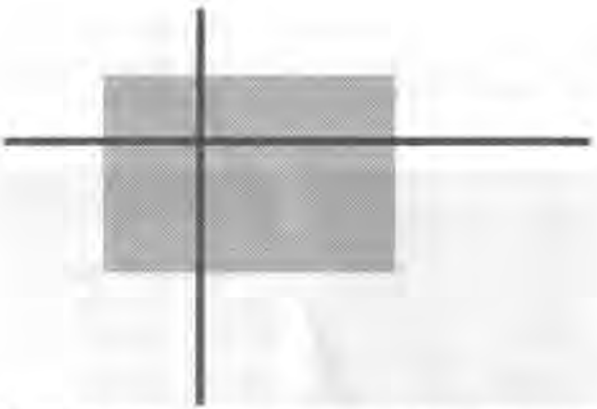


Рис. 9.3. Фоновое изображение со значением свойства background-position, выставленным на 33% 33%. Линии добавлены для наглядности

Значения ключевого слова свойства `background-position` в точности соответствуют 0 %, 50 % и 100 % по требованию.

Для дизайнеров, собравшихся применять в своем проекте Правило третей или Золотого сечения, данный подход может стать причиной серьезных проблем. Например, что если изображение на рис. 9.3 должно быть смещено к *центру* на 33 %?

Единственный эффективный метод решения этой проблемы — фиксация размеров вашего макета. Здесь тоже есть свои проблемы, которые, возможно, не стоит решать, если ваша единственная задача заключается в размещении фоновых изображений там, где вам это нужно.

Для CSS3 необходимым свойством является `background-size`, но в настоящий момент оно не поддерживается основными браузерами.

Свойство CSS background: краткая запись

Для свойства `background`, как и для свойств `margin`, `border`, `padding` и `font`, используется краткая запись. При этом его значения отделяются друг от друга пробелами и задаются в следующем порядке:

```
background-color background-image background-repeat background-position
background-attachment
```

Как и другие свойства, для которых используется краткая запись, background замечательно экономит место, но обладает одним существенным недостатком: если какие-то значения пропущены, результат может быть неоднозначным.

Составление фоновых изображений

Как оказалось, составление фоновых изображений может быть проблематичным. Прежде всего, существует масса различных *типов* фоновых изображений (рис. 9.4), что вызывает характерные технические проблемы.

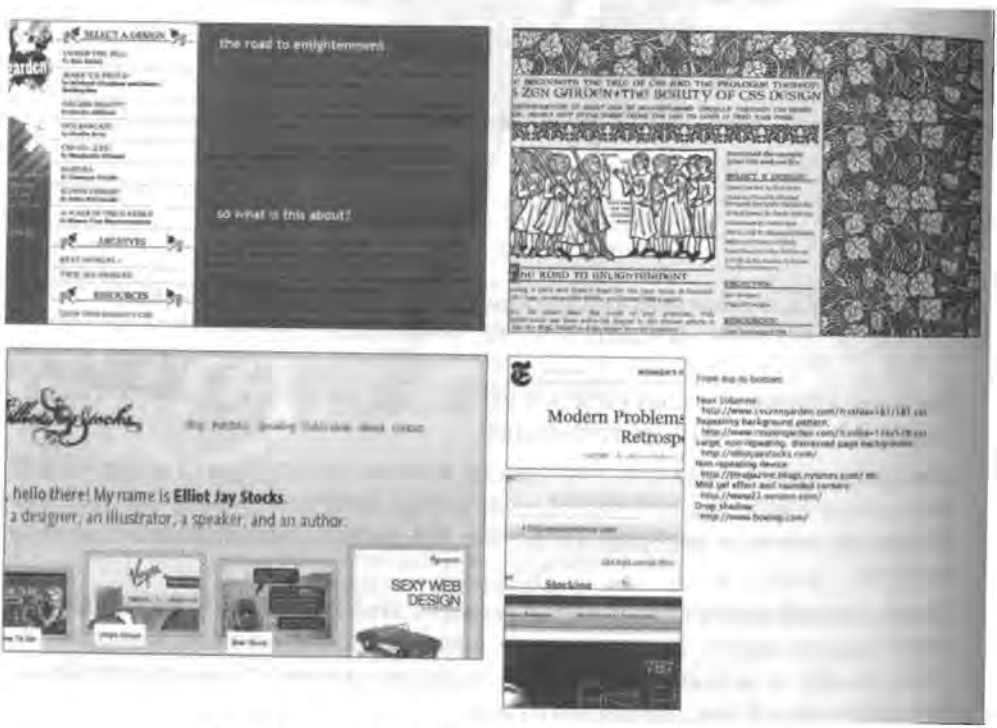


Рис. 9.4. Популярные стили для фоновых изображений

- *Ложные колонки.*

  Ложные колонки — это широкие и узкие цветовые полосы, одна из которых может быть прозрачной. В разделе про многоколонную верстку (см. «Создание макета с несколькими колонками» на с. 113) говорилось о том, что *принудительное* увеличение высоты содержимого элемента возможно, но в итоге возникают такие проблемы, что лучше этого не делать. Если содержимому колонки назначается фоновое изображение, воспроизводящее эти колонки, то в результате получится удобная иллюзия.

Там, где высота колонок не может быть обозначена точно, эту технику также можно использовать для размещения изображений по вертикали между колонками в качестве альтернативы свойствам `border-left` или `border-right`.

- *Фоновые текстуры и образцы.*
  Текстурные и искаженные фоны очень широко распространены, а когда-то вообще были на пике популярности, поскольку они придают дизайну стильный оттенок, сравнимый с ассоциированным контентом, претендующим на долговечность или оправдывающим свою плохую функциональность. Эти фоны могут быть сделаны в виде черепицы (повторяющихся элементов) или мозаики.

- *Тени и глянцевые эффекты.*
  Падающие тени располагают к себе. Они так же популярны, как и текстурные фоны. Глянцевые эффекты часто применяются к ссылкам и кнопкам, особенно на сайтах компаний, которые позиционируют себя как Web 2.0 и требуют схожих методов расположения изображений.

- *Закругленные углы.*
  Композиционные приемы для создания закругленных углов совершенно отличаются от приемов падающих теней, а вот проблем с этими приемами у дизайнеров столько же.

Для этих фоновых изображений мы можем добавить растровую копию, которая часто задается техникой замены изображения по методу Фарнера (см. раздел «Растровая копия и замена изображений по методу Фарнера» на с. 185).

«Ложные колонки»

Представьте себе дизайн в две колонки разной высоты. Несмотря на правила CSS, применяемые вами для верстки, одна проблема не исчезнет: в одной из колонок практически всегда будет меньше контента. Применение `{ overflow: auto; height: 1%; }` к элементу с колонками дает вам другой элемент, к которому вы можете эффективно применить фоны, но первоначальная проблема останется: как сделать его видимым для пользователя при двух (или более) колонках в полную высоту?

Если вам нужно сделать две колонки, окаймленные фоном, выполните следующие действия.

1. Начните с команды `{ overflow: auto; height: 1%; }`. Этот шаг *необязателен*, но он повышает гибкость управления разметкой.
2. Фоновое изображение должно быть всегда шире вашей верстки, чтобы учесть ширину закругленных углов и другие доработки макета. Фактическая ширина будет меняться в соответствии с характеристиками вашей разметки и способами применения фоновой картинки к макету. Если вы применяете один из гибких подходов к верстке, то, возможно, захотите увеличить

фоновое изображение так, чтобы его ширина *существенно* превышала запланированную ширину вашего макета.

3. Для сохранения скорости загрузки страницы фоновые изображения, затрагивающие шаблоны, не должны превышать высоту, необходимую для воспроизведения желаемого эффекта фрагментации изображения.
4. В большинстве случаев вам потребуется просчитать *пропорциональную* ширину колонок, если только в вашу задачу не входит создание одной колонки с постоянной шириной.
5. Для гибких макетов закрасьте полосы пропорционально. Для макетов по крайней мере с одной колонкой фиксированной ширины рассчитайте ширину полосы, соответствующей фиксированной колонке, сложив ширину колонки (в пикселах) и отступов. В случае использования специального правила максимальной высоты, а не колоночных полос, просто примените это правило к *X*-координате точки соприкосновения цветовых полос.
6. Взаимодополняющие пары свойство/значение должны быть заполнены следующим образом:

   - `background-color`

     В большинстве случаев фоновый цвет должен совпадать с цветом основной колонки.

   - `background-repeat`

     Если не хотите рисковать, рекомендуется применить элемент `repeat-y`.

   - `background-position`

     Примените сходное значение *X* (первое из двух представленных) к статичному или пропорциональному значению, определяющему ширину цветовых полос изображения. В случае фиксированных макетов это значение будет противоположно значению отступов. Поэтому отступ в 5 пикселов будет иметь следующий вид: `background-position: -5px 0`.

Обычно фон колонки не содержит прозрачных пикселов, а элемент `background-color` содержимого вашей колонки будет назначен в соответствии с цветами основной колонки. Единственное возможное исключение из этого метода — использование цветов колонки в качестве адресной команды. В ряде подобных случаев ресурсы сохраняются путем применения цельного фонового изображения, с прозрачными пикселами в «адаптивной» колонке, перенаправляющего элемент `background-color` к направляющим таблицы стилей, адресованных содержимому колонки (на основе чередования областей).

Этот метод отлично работает при макете в две колонки, но у макетов в три колонки с переменной шириной появляются новые проблемы. (Фоны фиксированной ширины можно, как правило, ограничить элементом `body`.) В примерах с макетом в три колонки, где необходимы три четкие фоновые полосы, вы будете применять *два* фоновых изображения: один к элементу `body`, самому широкому элементу-контейнеру, а второй — к межколоночному контейнеру (о котором было подробно рассказано в главе 6). Эти вышеизложенные методы можно выполнить повторно.

Текстуры и образцы черепичного фона

По умолчанию, фоновые изображения заполняют элемент по горизонтали и вертикали, начиная с левого верхнего угла этого элемента.

Черепичные фоны сложнее создавать из ассиметричных изображений, но у них есть преимущество в виде суженной пропускной полосы. Для создания и применения черепичных фонов пользователи Adobe Photoshop могут проделать следующий процесс (основанный на масштабировании и искривлении).

1. Фильтры→Другое→Смещение, чтобы *«отцентровать»* стыки. В результате стыки исходного изображения будут несимметричны.
2. Соедините исходное изображение с двумя его дубликатами (по одной на каждую ось). Использование решетки (Вид→Решетка→Прикрепить к→Решетка) существенно облегчит задачу.
3. Объедините три копии исходного изображения в один слой.
4. Для маскировки стыков используйте инструменты клонирования, восстановления, ретуширования и перемещения. Избегайте применения фильтров размывки, от которых минусов не меньше, чем плюсов. Применять их можно только на небольших участках изображения. Выбор инструмента будет зависеть от результатов экспериментирования и опыта.

   Этот шаг должен быть выполнен в тех областях, где стыки пересекаются с краями вашего изображения. Некоторые из этих доработок будут применяться только на копиях. Вставьте и объедините эти доработанные элементы в соответствующие области оригинала.
5. Обрежьте ваше рабочее изображение до первоначальных размеров.
6. Повторите шаги 2–4 на стыках центральной области изображения. Приготовьтесь часто применять команды Отменить, а лучше сразу выведите на экран палитру Журнал.
7. Сохраните изображение, загрузите его и примените на элементе. Остальные фоновые стили браузеров, применяемые по умолчанию, должны быть адекватны поставленной задаче.

Симметричные изображения также нужно смещать при создании или свойством `background-position`. В противном случае в верхнем левом краю элемента появится куча лишних дыр.

Огромные фоновые текстуры и специальные вставки

Крупные фоновые изображения можно адаптировать из имеющегося ассортимента минимальными усилиями, но эти изображения, как правило, требуют большей ширины полос. Суть работы с крупными фоновыми изображениями заключается в поиске оптимального сочетания файлового объема изображения,

размеров изображения и полезных деталей. Снижение первого параметра всегда происходит за счет двух других.

Когда стандартные фоны применяются к элементу `body`, проблема компромисса осложняется еще сильнее, поскольку размеры фонового изображения должны быть *как минимум* 1680×1050 пикселов. Размеры в 1920×1200 пикселов зачастую оказываются непригодны.

За исключением редких случаев, когда фоновое изображение в несколько раз выше применяемого к нему элемента или когда фон тщательно подобран к элементу с фиксированными размерами, фрагмент правил таблицы стилей должен быть примерно следующим:

```
#foo {
  background-image: url(/my/path/bar.ext);
  background-attachment: fixed;
  background-position: 50% 50%; }
```

Если вы создаете фоновое изображение, включающее специальную вставку (например, идеограмму, геометрическую форму, известную иллюстрацию общественного домена или уникальный элемент сайта), то эта вставка всегда будет меньше текстуры. Сочетание такого фонового изображения с программным CSS-кодом необходимо делать следующим образом.

- Делайте текстуру максимально контрастной, но не за счет четкости изображения контента, расположенного поверх фона.
- Если размер фонового изображения значительно меньше нижнего колонтитула его элемента, прикрепите фоновое изображение к краю или углу этого элемента. Результаты будут более очевидны.
- В большинстве случаев наиболее уместна команда `background-attachment: fixed`.
- Сохраните удобочитаемость элемента с фиксированным фоновым изображением, установив значение `padding-bottom` примерно равным исходной высоте фонового изображения.

Падающие тени, глянцевые эффекты и закругленные углы

Для создания падающих теней, глянцевых эффектов и закругленных углов в Photoshop можно воспользоваться диалоговым окном **Эффекты** или утомительным сочетанием разных инструментов и фильтров.

Эти три фоновых эффекта представлены отдельно, поскольку работающий с ними дизайнер полагается на макет с постоянной шириной, вставляет углы падающих теней в видимые грани макета (как правило, по верхней и нижней граням полотна браузера) или заставляет разработчика стилей делать лишнюю разметку.

Выбор этих приемов определяется фундаментальным принципом элементного потока: *невозможно* точно предсказать высоту элемента или изображения за

исключением встраиваемых. Поскольку фоновые изображения не могут быть произвольно отмасштабированы современными браузерами, эффекты, которые *необходимо* установить в нужных координатах макета (то есть в углах, как в случае с закругленными углами и угловыми областями падающих теней) в пределах элемента неопределенных размеров, должны размещаться *по одному*. Разметка и стили, используемые для закругленных углов, представленных в начале раздела «Правило № 1. Будь проще» (с. 72) представляют собой отрицательные примеры, демонстрирующие, к чему приводит простота.

Браузеры Firefox и Safari поддерживают собственные CSS-расширения для создания закругленных углов (`-moz-border-radius` и `-webkit-border-radius` соответственно), но Internet Explorer ничего подобного не предлагает. Стандартное свойство `border-radius` определено в CSS3.

Растровая копия и замена изображений по методу Фарнера

В главе 12 подробно пойдет речь о шрифтах, там будут разобраны все нюансы. У совместной темы шрифтов и фоновых изображений есть два сложных момента: узкий ассортимент универсальных шрифтов для приложений и преимущества от сильного сглаживания крупных шрифтов.

Встроенные и фоновые изображения гарантируют дизайнерам получение красивых типографских шрифтов для веб-приложений. Все было действительно так, пока в 1993 году браузеры не начали поддерживать встраиваемые изображения. В первой половине 2002 года несколько экспертов по CSS (включая меня) приступили к независимой разработке методов, позволивших перенести растровый шрифт из разметки в таблицы стилей. Результат применения одного из простейших методов представлен на рис. 9.5. Комплекс этих техник назывался заменой изображений по методу Фарнера в честь Тодда Фарнера (FIR), первопроходца в области применения CSS, который одним из первых разработал и опубликовал базовые принципы работы с таблицами стилей.

Основная задача FIR — не дать изображению находиться в контентном слое сайта, пока оно *фактически* не станет контентом. Как тогда сохранить свойства изображений при перемещении в уровень представления, где им и место?

Отчасти решение заключается, прежде всего, в применении фоновых свойств. Однако один этот шаг ничего не делает с текстом, содержащимся в элементе, к которому применяется метод FIR. Для решения этой проблемы разработчик стилей должен применять одну из модальностей для вывода контента из поля зрения без затрагивания композиции основного элемента.

Первый интересный подход заключался в размещении лишнего элемента `span` внутри главного элемента, после чего к нему применялась команда `display`:

none. Однако у такого подхода есть свой недостаток: платформы, предназначенные для людей с ограниченными возможностями, обрабатывают пару `display: none` (тем не менее тип носителя данных все равно конфликтует) и игнорируют ее, так же как браузер при отображении контента на экране.

Рис. 9.5. Три шага от обычного заголовка к FIR-заголовку: 1 — демонстрация обычного заголовка; 2 — демонстрация заголовка с новым фоновым изображением; и 3 — демонстрация конечного результата с применением функции структурированного текста

Разработчики проверили несколько других подходов (на базе «мусорной» разметки) к FIR-концепции, каждый из которых зависел от разных способов применения CSS (например, от доступности и позиционирования) для перемещения текста из зоны видимости. Вскоре один разработчик по имени Майк Рандл предложил использовать свойство `text-indent` для сдвига текста за край полотна, а другой разработчик по имени Расс Уикли предложил скрывать крошечные (действительно *крошечные*) отрывки текста за фоном идентичного цвета и одновременно задавать межстолбцовый промежуток на краю элемента для обработки этого текста до полной невидимости.

Правила таблицы стилей FIR

Исходный CSS-код для методов Рандла и Уикли, предполагающий наличие строки `<h2>Lorem ipsum dolor sit amet. consectetur adipiscing elit</h2>`, и фоновое изображение с 30 пикселами в высоту отображены ниже:

```
/* *** Rundle (Phark) Method *** */

h2 {
 height: 30px;
 margin: 0;
 text-indent: -10000px;
 background-image: url(/my/path/foo.ext);
 background-repeat: no-repeat;
}

/* *** Weakley Method *** */

h2 {
  height: 1px;
  margin-bottom: -1px;
  padding-top: 30px;
  color: #fff;
  background-color: #fff;
  background-image: url(/my/path/foo.ext);
  background-repeat: no-repeat;
  font-size: 1px;
  line-height: 1;
}
```

Метод Уикли гораздо подробнее, но при отклонении от него возникают серьезные проблемы с обработкой в Internet Explorer 6. И это обнаруживается, когда элемент `text-indent` прикрепляется к ссылке с элементом `float value:`. Текст реагирует на значение `text-indent`, а вот подчеркивание — нет.

Необходимо также учитывать, что мелкий полускрытый текст сайта, как в примере Уикли, может стать приманкой для поисковых провайдеров, занесенных в черный список, при использовании его исключительно в целях оптимизации для поисковых систем (SEO).

Минусы FIR

У замены изображений по методу Фарнера есть два существенных недостатка: разработчики стилей не контролируют появление фоновых изображений на главной странице (где они, возможно, вообще не обрабатываются) и малое число пользователей, отключающие опцию показа изображений в своих браузерах, будут в недоумении, прочитав улучшенную вами копию по методу FIR.

Оптимизация работы сервера с помощью спрайтов

Через год после оживления интереса к методу FIR веб-журнал A List Apart опубликовал статью Дейва Ши, в которой рассказывалось про артефакты эпохи 8-битных игр — спрайты. В ретро-играх этот термин обозначал группы изображений, которые все вместе создавали платформенный «ландшафт» со старой боковой прокруткой или видеоигру с видом сверху. Отображение спрайтов происходило в нужном виде благодаря аппаратным возможностям игровых приставок.

Сходный дизайнерский подход мог использоваться разработчиками стилей для комбинирования нескольких схожих фоновых изображений в единичный файл с целью сокращения серверного траффика и требований к техобслуживанию.

Спрайты очень хорошо используются для ссылок навигации. В ряде случаев система поиска переходит на комбинированные ссылки и навесные растровые отображения в точности так, как и задумывалось.

Рассмотрим настройки системы поиска, показанные на рис. 9.6: различные элементы представлены растровыми изображениями и обрабатываются с помощью FIR, поскольку желаемый шрифт не поддерживается приложениями различных дистрибьюторов операционных систем.

Рис. 9.6. Предполагаемый результат (представленный в данном разделе) применения спрайтов в списке поиска

Расположение растровых экспортов может быть произвольным: в этом случае я, наверное, помещу нависающее растровое изображение под растровое изображение, устанавливаемое по умолчанию, но вы можете скомпоновать элементы по-другому. Главное — записать изначальные координаты каждого растрового фрагмента, необходимого для дальнейшего применения значений `background-position`.

Для применения стандартных и парящих растровых изображений по методу FIR, получаемых из цельного изображения для системы навигации по сайту, выполните следующие действия.

1. Скомпонуйте систему навигации в соответствии с методом, описанным в разделе «Способ расположения основной навигации» на с. 147.

2. Задайте то же самое значение `background-image` для обоих элементов списка, содержащих ваши поисковые ссылки и парящие растровые изображения самих ссылок. На этом этапе ваша система поиска должна отображать идентичный растровый шрифт у всех элементов (см. рис. 9.7).

Рис. 9.7. Фоновое изображение, используемое для спрайтового поискового списка

3. Задайте уникальные значения `background-position` для различных направляющих стандартных и парящих растровых изображений (например, `#nav li { ... } and #nav li a:hover { ... }`). Основные демонстрационные стили, представленные в данном разделе, можно взять с сопутствующего сайта книги.

```
  #nav li { background-image: url(/images/bg_nav.gif); }
#nav li a { display: block; width: 100%; height: 100%; }
  #nav li,
    #nav li a { text-indent: -10000px; }
  #nav li a:link,
#nav li a:visited { background-image: none; }

  #navAbout { background-position: 0 0; }
#navContact { background-position: -144px 0; }
```

продолжение ↩

Продолжение листинга

```
  #navForum { background-position: -288px 0; }
#navGallery { background-position: -432px 0; }

  #navAbout a:hover { background-position: 0 -24px; }
#navContact a:hover { background-position: -144px -24px; }
  #navForum a:hover { background-position: -288px -24px; }
#navGallery a:hover { background-position: -432px -24px; }
```

10 Таблицы данных

Если вы занимаетесь веб-дизайном хотя бы несколько лет, то у вас наверняка есть книга, авторы которой рекомендуют использовать для верстки элементы `table`. Когда-то это был единственный надежный способ создания макета страницы, учитывая возможности браузеров на тот момент; кроме того, как следует из обсуждения многоколонных макетов, макетные таблицы сравнительно просто реализовать.

Но, если говорить кратко, *этого делать не стоит.*

Недостатки макетных таблиц

Нельзя сказать, что с макетными таблицами с самого начала было связано много неприятных сюрпризов, однако их использование лишает нас возможности сделать сайт практичным и удобным в обслуживании. Хотя на начальной стадии создания нового сайта использование макета на основе таблицы может оказаться более простым решением, обслуживание и добавление новых данных очень скоро становится основной причиной головной боли.

Исходный порядок: квадратный стержень, круглая дыра

Если для создания макета страницы вы используете таблицы, весь сайт должен быть реализован в соответствии с алгоритмом построения таблиц. А поскольку этот алгоритм изначально предназначался для представления данных, с точки зрения структуры контента он больше подходит для электронных таблиц, чем для документов, содержащих фрагменты текста. (Стали бы вы писать школьное сочинение или служебную записку в Microsoft Excel? Думаю, нет.)

Первая категория людей, страдающих от этого, — те, кто использует возможности вспомогательных технологий для работы в Сети. Так как создатели вспомогательного ПО не могут читать мысли каждого отдельного веб-дизайнера,

содержимое таблицы отображается в исходном порядке. Одним из следствий этого подхода является следующее: спустя 15 лет после того, как поддержка таблиц стала доступной для доверчивых пользователей Сети, те, кто использует вспомогательные технологии, *до сих пор* считают, что навигация сайта должна располагаться вверху страницы (независимо от того, насколько это *на самом деле* важно).

Если боковая панель навигации расположена в левой колонке, порядок следования контента нарушается еще сильнее. Представьте, что вы *обязаны* просмотреть три сотни несвязанных между собой и ненужных слов прежде, чем дойдете до заголовка, не говоря уже об основном тексте, о статье, которую вы собирались прочитать на этом сайте... едва ли вам удастся представить себе, с чем приходится иметь дело людям с ограниченными возможностями при работе в Сети.

CSS-дзен становится легендой

Правильное применение принципов CSS-дзен дает нам неограниченные возможности для быстрого изменения дизайна сайта. Структурная разметка и значения `class/id` делают такую гибкость возможной. Эта гибкость в свою очередь существенно уменьшает время, требующееся для реализации дизайна: так, на три из четырех последних попыток обновить дизайн моего сайта henick.net я потратил менее одного дня.

И даже в случае *крупного* проекта внимательное отношение к структуре сайта и страницы, а также разумный подход к перестройке дизайна позволяют группе разработчиков создавать на сайте абсолютно новую атмосферу за несколько дней.

Однако если для создания макета используются таблицы, *это становится невозможным* — дизайн должен украшать разметку, в то время как макеты таблиц *накладывают на нее ограничения*. На практике это означает, что при *любой* перестройке дизайна с сохранением существующей разметки вы будете вынуждены ограничиться изменением цветов и эффектов; добавление нового контента и удаление старого, а также существенные изменения в структуре макета потребуют создания для сайта абсолютно новых шаблонов.

Неизбежное рабство перед шаблонами

Если на вашем сайте не используется *огромное множество* шаблонов, гибкость макета, основанного на таблице, оказывается близкой к нулю. А если вы работаете над сайтом крупного магазина и заказчик требует реализовать сложную процедуру согласования, про дизайн можно заранее сказать, что он будет выгравирован на камне. И этот факт, безусловно, очень огорчает — а что если с сайтом возникнет какая-то глобальная проблема? Пользователи, которых она затронет, — это вторая категория людей, страдающих от недостатков макетных таблиц. При этом невозможность быстро уладить сложившуюся ситуацию может привести к дополнительным затратам.

Скорее всего, в течение первых нескольких недель группа специалистов, разработавших сайт на основе таблицы, ознакомится со списком проблем, указанных в отзывах пользователей, и начнет думать над созданием *следующего* дизайна, который потребует ничуть не меньше усилий. А когда дело дойдет до CSS, процесс создания сайта будет становиться *еще сложнее* по мере того, как разработчики будут приспосабливаться к лаконичности и основным понятиям CSS.

Когда как не сейчас было бы лучше всего покончить со всем *этим*?

Позиционирование оказывается бесполезным

Хотя при верстке с помощью CSS свойство `float` берет на себя большую часть трудностей, многими из его достоинств обладает также и свойство `position`.

Если для создания макета вы используете таблицы, то свойства, задающие позиционирование, не будут иметь практически никакой силы. Растягивание макета, расположение контента и множество других полезных приемов верстки оказываются недоступными. Табличные макеты не оставляют дизайнеру *абсолютно* никакого выбора: он вынужден работать с тем, что находится внутри блока, поскольку особенности отображения ячеек таблицы исключают из рассмотрения любые другие варианты.

А это уже очень похоже на конец света!

Части таблицы данных

Чтобы понять, как создается разметка таблицы данных, проще всего начать с наименьших частей таблицы (рис. 10.1).

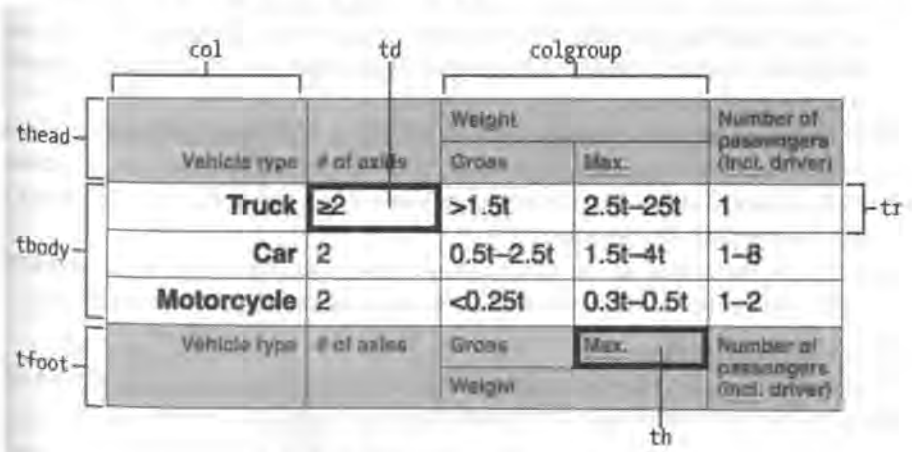


Рис. 10.1. Структура таблицы данных

- Ряды и ячейки: `tr` и `td`.

  Здесь, как правило, содержатся все значения, относящиеся к одному объекту. Ряды разделены на ячейки (`td`), а сходные ячейки в разных рядах всегда объединяются в соответствующие колонки. Как и элементы `tbody`, ряды и ячейки *обязаны* присутствовать в таблице.

  Если вы знакомы с реляционными базами данных, вы наверняка заметите, что тег `tr` обозначает границы одной записи.

- Колонки и группы колонок: `col` и `colgroup`.

  Подобно тому, как ряд объединяет все значения для одного объекта, колонка соответствует общему классу данных. Колонка включает в себя ячейки с шагом n, где n — количество ячеек в каждом ряду.

- Заголовки рядов и колонок: `th`.

  `th` служит альтернативой `td`, определяя тип данных в определенном ряду или колонке.

  Элементы `th` отличаются от элементов `td` не только оформлением, но и тем, что они предназначены *специально* для тех ячеек таблицы, которые не содержат фактических данных.

- Заголовки: `caption`.

  Заголовки, как правило, используются для обычного текста, в то время как элементы `caption` (несмотря на название) выполняют аналогичную роль для таблиц. Они всегда располагаются вверху таблицы — так же, как заголовок располагается относительно соответствующего ему текста.

- Верхние и нижние шапки таблицы: `thead` и `tfoot`.

  Они служат своеобразными «корзинами» для описания типов данных, содержащихся в различных колонках и (реже) рядах таблицы. От обычных рядов их отличает следующее.

  - Для выделения типов данных может понадобиться несколько рядов. Возможна, например, такая ситуация: в первом ряду «шапки» определены широкие *классы* данных, а во втором указаны типы данных для каждой отдельной колонки.
  - В соответствии со спецификацией HTML и CSS, если таблица занимает несколько страниц, элементы `thead` и `tfoot` должны отображаться на каждой из них. С помощью обычных рядов и колонок не удается добиться такого усовершенствования.

  Обратите внимание на то, что с точки зрения валидности, элементы `thead` и `tfoot` являются необязательными (в отличие от обязательного `tbody`).

- Тело таблицы: `tbody`.

  Тело таблицы включает все ячейки, в которых содержатся фактические *данные* (в противоположность метаданным).

Пример разметки таблицы: все вперемешку

Если использовать в таблице все элементы, о которых мы только что говорили, разметка будет выглядеть так (или, по крайней мере, очень похоже).

```
<table summary="Trucks, cars, and motorcycles, described according
to their number of axles, vehicle weight, and passenger capacity.">
<caption>Types of gasoline- and diesel-fueled road vehicles.</caption>
<colgroup id="type"></colgroup>
<colgroup id="axlect"></colgroup>
<colgroup id="wt">
    <col id="gvw" />
    <col id="mvw" />
</colgroup>
<colgroup id="pax"></colgroup>
<thead>
    <tr>
     <th rowspan="2" scope="colgroup" abbr="Type">Vehicle type</th>
        <th rowspan="2" scope="colgroup"># of axles</th>
        <th colspan="2" scope="colgroup">Weight</th>
        <th rowspan="2" scope="colgroup" abbr="Passengers">Number of
passengers (incl. driver)</th>
    </tr>
    <tr>
        <th scope="col">Gross</th>
        <th scope="col"><abbr title="maximum">Max.</abbr></th>
    </tr>
</thead>
<tfoot>
    <tr>
        <th rowspan="2" scope="col" abbr="Type">Vehicle type</th>
        <th rowspan="2" scope="col"># of axles</th>
        <th scope="col">Gross</th>
        <th scope="col"><abbr title="maximum">Max.</abbr></th>
        <th rowspan="2" scope="col" abbr="Passengers">Number of passengers
(incl. driver)</th>
    </tr>
    <tr>
        <th colspan="2" scope="colgroup">Weight</th>
    </tr>
</tfoot>
<tbody>
    <tr>
        <th scope="row">Truck</th>
        <td>&ge;2</td>
        <td>&gt;1.5t</td>
        <td>2.5t&ndash;25t</td>
        <td>1</td>
```

продолжение ⇩

```
    </tr>
    <tr>
        <th scope="row">Car</th>
        <td>2</td>
        <td>0.5t&ndash;2.5t</td>
        <td>1.5t&ndash;4t</td>
        <td>1&ndash;8</td>
    </tr>
    <tr>
        <th scope="row">Motorcycle</th>
        <td>2</td>
        <td>&lt;0.25t</td>
        <td>0.3t&ndash;0.5t</td>
        <td>1&ndash;2</td>
    </tr>
</tbody>
</table>
```

Если повнимательнее посмотреть на эту разметку, можно заметить следующие особенности.

- Элемент `caption` является прямым потомком `table`; он находится вверху таблицы. Элементы `caption` необязательны, однако если такой элемент все же используется, то по правилам он должен быть первым непосредственным потомком `table`.
- Атрибут `summary` (если он вообще используется) является *функциональным* заменителем атрибута `title`.
- С помощью `colgroup` можно задать одно название для целой «группы» колонок, в то время как элемент `col` используется для отдельных колонок с общей областью действия.
- По правилам XHTML для элементов `colgroup`, не содержащих контента, требуется закрывающий тег. В HTML-документах это является необязательным.
- Использование элементов `rowspan` и `colspan` — удобный способ указать, что информация повторяется в нескольких записях или полях (в случае таблицы данных); внутри тега `thead` эти элементы позволяют лучше отразить группировку полей.
- Для каждой колонки и группы колонок используется атрибут `id`. Благодаря этому можно задавать значения `width` для колонок, а не для ячеек первого ряда таблицы.
- В нижней шапке ряды и ячейки отображаются наоборот (по сравнению с верхней шапкой).
- Использование атрибута `scope` для элементов `th` в шапке и теле таблицы позволяет проще генерировать метаданные. Элемент `td` в свою очередь поддерживает атрибут `headers`, которому в качестве значения требуется разделенный пробелами список названий (`id`) элементов `th`, относящихся к данной ячейке.

- Иногда в элемент `th` добавляется дополнительный элемент `abbr`. Вместо обычного многословного контента агенты пользователей могут обращаться к этому значению; это удобно в тех случаях, когда предпочтение отдается занимаемому объему и времени работы.
- Элемент `tfoot` расположен после `thead`, а не после `tbody`. Это не очень понятное требование, необходимое для корректности разметки; о нем мы еще поговорим в главе 12.

В этом примере не используется атрибут `headers`. Он добавляется к элементам `td` и требует в качестве значения разделенный пробелами список названий (`id`) элементов `th`, относящихся к данной ячейке таблицы. Таким образом браузеры, полностью поддерживающие селекторы атрибутов, позволяют оформлять отдельные ячейки исходя из *контекста*, освобождая дизайнеров от необходимости использовать для ячеек селектор псевдоэлемента `:nth-child()` без учета тех данных, которые в них на самом деле содержатся.

Помимо `rowspan` и `colspan` существуют и другие атрибуты, с помощью которых можно управлять верстской таблицы, и они обладают неплохой поддержкой. Однако использование этих атрибутов вместо таблиц стилей уменьшает гибкость самой таблицы данных, особенно если она содержит контент, генерируемый пользователем.

Создание ячеек

Прежде чем задать ширину колонок и фон, лучше убедиться, что каждая отдельная ячейка выглядит так, как мы хотим. Мы уделяем столько внимания именно ячейкам потому, что из всех элементов таблицы они обладают наибольшим числом свойств.

Лучше всего начать с границ. Если внимательно посмотреть на произвольные фрагменты разметки современных html-документов, можно найти достаточно много тегов `table`, которые выглядят примерно так.

```
<table border="0" cellspacing="0" cellpadding="0">
```

Использование таких пар атрибут/значение означает, что различные ячейки таблицы будут располагаться вплотную к своим соседям; такого же эффекта можно добиться с помощью следующего CSS-кода.

```
table { border-collapse: collapse: }
td. th { padding: 0: }
```

Результат показан на рис. 10.2.

Оставим эту задачу и сделаем следующий шаг: попробует отделить ячейки друг от друга с помощью границы. Поскольку каждая ячейка расположена вплотную к своим соседям, нам понадобится только две границы. Если вы посмотрите

на рис. 10.3, то увидите, что все три ячейки имеют верхнюю и левую границы, а границы ячейки 1 создаются за счет границ ячеек 2 и 3. Для добавления такого эффекта используются следующие стили.

```
td. th { border-top: 1px solid #000: border-left: 1px solid #000: }
```

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight: Gross	Weight: Max.	Number of passengers (incl. driver)
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross Weight	Max.	Number of passengers (incl. driver)

Рис. 10.2. Таблица данных без границ и полей внутри ячеек

Рис. 10.3. Если добавлять к ячейкам таблицы только две границы (*1*), остальные две появятся благодаря тому, что соседние ячейки (*2*) и (*3*) расположены вплотную

В дальнейшем границы можно будет удалить; если только вы не задали значения `width` для таблицы и отдельных колонок (что и так не приветствуется), наличие или отсутствие границ ячеек не приведет к серьезным ошибкам или конфликтам верстки. А между тем, пока у нас еще есть граница, таблица будет выглядеть так, как показано на рис. 10.4.

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight: Gross	Weight: Max.	Number of passengers (incl. driver)
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross Weight	Max.	Number of passengers (incl. driver)

Рис. 10.4. Та же таблица, но с добавленными границами

Ни один из стилей, которые мы до сих пор использовали, не имел отношения к нижним и правым границам; кроме того, с помощью значения `border-collapse`

нам удалось скрыть крайнюю левую границу таблицы. И вот, что мы получили: в таком виде, без свободного пространства таблицу попросту неудобно читать (и таким образом теряется весь смысл таблицы данных). Поэтому, чтобы получить приемлемый результат (рис. 10.5), добавим следующие стили.

```
table { margin-left: 1px; border-right: 1px solid #000; border-bottom: 1px
solid #000; } td, th { padding: 5px; }
```

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight		Number of passengers (incl. driver)
		Gross	Max.	
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross	Max.	Number of passengers (incl. driver)
		Weight		

Рис. 10.5. Та же таблица, но с недостающими границами и отступами

Построение таблицы и размещение данных

Процесс более детального построения таблиц оказывается настолько сложным, что вызывает чувство безвыходности, правда обычно благодаря опыту разработчикам удается интуитивно предсказать, как будет выглядеть неоформленная таблица. Но дизайнеры могут быть уверены, что идеальная верстка таблицы будет сильно отличаться от того, что изначально предлагает браузер.

Первый шаг на пути к созданию таблицы данных — задание выравнивания содержимого ячеек с учетом отступов. Значения свойств выравнивания, использующиеся по умолчанию для ячеек и заголовков, указаны в табл. 10.1.

Таблица 10.1. Стили выравнивания по умолчанию для данных таблицы (направленных слева направо)

Элемент	text-align	vertical-align
`td`	`left`	`middle`
`th`	`center`	`middle`

В основном для размещения данных в таблице действуют следующие правила.

- Буквенные данные должны быть выровнены по верхнему и левому краю.
- Содержимое элемента `th` внутри тега `thead` должно быть выровнено по горизонтали так же, как данные, которые оно описывает; по вертикали оно должно быть выровнено по нижнему краю.

Однако стоит помнить, что дизайнеры особенно часто отступают от этих правил. Ниже приведены два ярких примера.

- Нередко для первых двух колонок таблицы используется одинаковый отступ: так дизайнеры пытаются выделить самую левую колонку данных.
- Когда в нескольких смежных ячейках информация дублируется, нередко в том месте, где эти данные встречаются впервые, используется соответствующее значение `rowspan` или `colspan`; данные выравниваются по центру полученной большой ячейки (хотя при этом снижается удобство чтения).

Проблема с применением стандартов построения таблиц заключается не в *значениях*, а скорее в *селекторах*. Internet Explorer отстает от остальных браузеров в поддержке селекторов, позволяющих создавать для таблицы *четкую* верстку, не злоупотребляя значениями атрибутов `class`.

Помимо *выравнивания данных* таблицы часто потребуется задавать значения ширины колонок. Для этого лучше всего использовать значения атрибута `id` для элементов `colgroup` и `col`. В результате получаются приблизительно такие правила.

```
   #type { width: 8em; }
#axlect { width: 8em; }
   #gvw { width: 4em; }
   #mvw { width: 4em; }
   #pax { width: 8em; }
```

Результат показан на рис. 10.6. Если принципы, которых вы придерживаетесь во время работы, не позволяют настраивать таблицы в индивидуальном порядке, то план стандартизации значений колонок ваших таблиц вполне может быть одобрен. Результат будет сопоставим с тем, что получается при использовании классов для задания размеров полей формы.

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight		Number of passengers (incl. driver)
		Gross	Max.	
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross	Max.	Number of passengers (incl. driver)
		Weight		

Рис. 10.6. Та же таблица, но с правильно оформленными колонками

Прежде чем проанализировать рис. 10.6, вы, вероятно, заметите, что для каждой колонки используется свое правило. Хотя оформление нескольких колонок

одним правилом не является *ошибочным*, такой синтаксис может в будущем привести к конфликту приоритетов; так что оставим отдельные правила.

Хотя свойства `width` и `background` единственные, применение которых к элементам `col` и `colspan` может привести к положительному результату, существует еще одна причина, по которой на эти элементы стоит полагаться при создании оформления. Если верстку или содержимое таблицы понадобится изменить, использование HTML-атрибутов для управления внешним видом колонки может оказаться неудачным решением из-за режима `overflow` или каких-либо других непредвиденных обстоятельств, что потребует внесения серьезных изменений в разметку. А если вместо этого вы зададите оформление таблицы с помощью нескольких стилей, изменения в разметке в большинстве случаев будут касаться только данных и исходного порядка.

Верхние/нижние шапки и заголовочные ячейки таблицы

Когда данные выровнены относительно известных отступов, результат становится более удобочитаемым. На рис. 10.7 показано, что получится, если применить к различным заголовочными ячейкам стили `text-align` и `vertical-align`.

```
        tbody th { text-align: right; }
        thead th { vertical-align: bottom; }
        tfoot th { vertical-align: top; }
thead th, tfoot th { text-align: left; }

thead th[rowspan]:first-child,
tfoot th[rowspan]:first-child {
    text-align: right;
}
```

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight		Number of passengers (incl. driver)
		Gross	Max.	
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross	Max.	Number of passengers (incl. driver)
		Weight		

Рис. 10.7. Та же таблица, но с выровненными заголовочными ячейками

Селекторы атрибутов и дочерние селекторы

Если вы посмотрите на правила из предыдущих таблиц стилей, вы заметите в последней из них странные селекторы. Такой селектор относится к любому элементу `th` — первому прямому потомку любого элемента внутри тегов `thead` или `tfoot`, а также имеет значение `rowspan`.

Вполне возможно, что вы смотрите на эти селекторы и думаете о том, что они были «высосаны из пальца» — но это не так. При правильном применении селекторов атрибутов и селекторов `:nth-child()` каскад может обращаться практически к любому элементу, который только можно придумать, и при этом вам не придется добавлять к произвольным элементам атрибуты `id` и `class`.

Конечно же, Internet Explorer не поддерживает никакие новые селекторы, о которых мы здесь говорим. Так что вам все равно придется добавлять в разметку атрибуты `class`, или же позволить отсутствующей поддержке, образно говоря, «ходить по улице в семейных трусах». А в случае IE 8 поддержка вообще граничит с безумием, так как *сам* селектор IE 8 *поддерживает*, но только не в сочетании с селекторами элементов.

Следует также отметить, что `:last-child` не поддерживается ни в одной версии Internet Explorer, а поддержка `:nth-child` недавно появилась в Firefox. Поэтому вам придется добавлять произвольные атрибуты `class` к элементам таблицы, если, например, захотите сделать полосатый фон (или добиться некоторых других эффектов).

```
tbody tr:nth-child(2n+1) td { background-color: #ccc: }
```

Чтобы понять, какой аргумент следует использовать для селектора `:nth-child()`, выполните следующее.

1. Определите интервал между элементами и подставьте это значение перед n.
2. Определите расположение (по порядку) первого элемента, для которого вы хотите задать такой стиль. Это значение будет вторым слагаемым.

В предыдущем примере аргумент селектора порождает множество всех нечетных элементов, начиная с первого (так же как 2n порождало бы множество четных элементов).

Уменьшение контраста верхней и нижней шапки

На данном этапе у верстки нашей таблицы есть слабое место: содержимое верхней и нижней шапки слишком явно претендует на внимание пользователя, отнимая его у реальных данных. Решить эту проблему можно двумя простыми способами: уменьшением контраста и уменьшением размера шрифта — и нет причин использовать только один из этих вариантов. Чтобы добавить такие эффекты к нашей таблице, применим следующие стили.

```
thead th, tfoot th { font-size: .75em; background-color: #ddd; color: #777; }
```

Еще один способ изменить контраст данных относительно остального содержимого таблицы — увеличить яркость границ. Результаты изменения заголовочных ячеек и границ показаны на рис. 10.8.

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight		Number of passengers (incl. driver)
		Gross	Max.	
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross	Max.	Number of passengers (incl. driver)
		Weight		

Рис. 10.8. Та же таблица, но с уменьшенным контрастом и размером шрифта в заголовочных ячейках, а также уменьшенным контрастом границ

Наконец нам осталось изменить заголовок и задать новые значения ширины колонок (вторая колонка стала слишком широкой, а третью и четвертую следует немного увеличить, чтобы контент был удобнее расположен).

Результат применения этих стилей показан на рис. 10.9; посмотреть, как наша таблица выглядела до и после задания оформления, вы можете на рис. 10.10.

Types of gasoline- and diesel-fueled road vehicles.

Vehicle type	# of axles	Weight		Number of passengers (incl. driver)
		Gross	Max.	
Truck	≥2	>1.5t	2.5t–25t	1
Car	2	0.5t–2.5t	1.5t–4t	1–8
Motorcycle	2	<0.25t	0.3t–0.5t	1–2
Vehicle type	# of axles	Gross	Max.	Number of passengers (incl. driver)
		Weight		

Рис. 10.9. Та же таблица, но с изменением ширины колонок и оформления заголовка

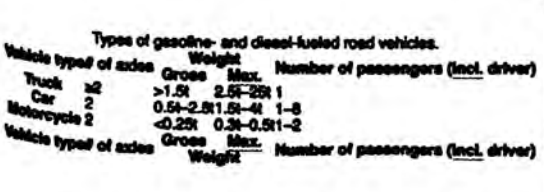


Рис. 10.10. Наш пример до и после задания оформления

Преимущества использования стилей для создания оформления теперь очевидны: какой из двух дизайнов таблицы, изображенных на рис. 10.10, вы бы хотели видеть в вашем веб-документе?

Добавление эффектов наведения

Таблицы, которые имеют очень большую ширину или используют элементы `rowspan/colspan`, чтобы данные отображались аккуратно, существенно выигрывают от эффектов наведения курсора мыши. Самый простой из таких эффектов — это выделение ряда таблицы; для его создания потребуются приблизительно такие правила (такой код будет работать во всех основных браузерах кроме Internet Explorer 6).

```
tr:hover { background-color: #fcc; }
```

Однако создание аналогичного эффекта для колонок потребует написания скриптов. При наведении курсора мыши на ячейку необходимо найти аналогичные ей ячейки из других рядов тела таблицы с помощью DOM API, а затем изменить их, используя свойства `style` или специально заданные значения атрибутов `class`.

Наконец, у ячеек могут быть атрибуты `title`, которые не используются в нашем примере, но могут служить для определения точки пересечения, в которой расположена ячейка (например, `[Range of truck gross vehicle weights]`). Такой подход является слишком детальным и может привести к ухудшению удобства работы с небольшими таблицами на экране. Для больших таблиц это зачастую помогает быстрее находить данные и понимать их смысл.

Вне зависимости от того, реагирует ли таблица на события, инициируемые пользователем, важно помнить, что создание таблицы далеко не ограничивается удалением лишнего свободного места внутри нее. Специалист, которому понадобится создать эффективный информационный дизайн, обнаружит в CSS множество инструментов, позволяющих выполнить эту задачу.

11 Изображения и мультимедиа

Сегодня трудно представить, что первые веб-страницы содержали только текст. Хотя изображения можно было просматривать в специальных окнах, они хранились отдельно от остального содержимого страницы. Именно поэтому появление браузера NCSA Mosaic, поддерживающего встраивание изображений в текст, стало своего рода революцией. А вскоре после этого появилась возможность встраивания других мультимедийных элементов (особенно после выхода платформы Flash в 1996 году).

Основные принципы веб-разработки, которые использовались еще на ранних этапах развития Сети, остаются актуальными и спустя 15 лет. Это происходит главным образом потому, что поддержка двух базовых элементов мультимедиа — `img` и `object` — с момента их появления изменилась *незначительно*. Эти принципы применяются наряду с более новыми методами, направленными на минимизацию требований, предъявляемых к браузерам и программному обеспечению серверов.

Замещенные элементы

Если просмотреть спецификации HTML 4.01 и CSS 2.1, то можно обнаружить, что некоторые элементы описываются как «замещенные». Но, как и следовало ожидать, спецификации не содержат понятного объяснения того, что такое замещенные элементы и как они работают. В спецификации CSS 2.1 (секция 3.1) написано следующее:

> [Замещенный элемент —] это элемент, содержимое которого находится *за пределами сферы действия* [выделение мое] модели форматирования CSS (сюда относятся изображения, встроенные документы, апплеты). Так, например, содержимое HTML-элемента `IMG` часто замещается изображением, добавленным с помощью атрибута `src`. Замещенные элементы часто имеют собственные размеры: ширину, длину, пропорции.

Проще говоря:

> Параметры расположения замещенных элементов если и могут быть изменены, то только прямым вмешательством разработчика, причем независимо от контекста.

Для замещенных элементов характерно то, что их содержимое отображается компонентом локального хоста, а не средствами визуализации браузера, который просто получает содержимое как входные данные и размещает его на странице.

Как и замещенные элементы, изображения имеют дискретную ширину и длину; среди прочего, это означает, что при применении свойства `layout` к изображению эти значения сохранятся. Но при отсутствии таких значений изображение будет размещено так же, как слово в тексте: его нижний край будет совпадать с базовой линией строки текста.

Если присвоить `display: inline-block` любому незамещенному строковому элементу, то он будет вести себя как замещенный элемент. Однако это не всегда верно для Internet Explorer 6, который применяет `display: inline-block` только к тем элементам, которые являются строковыми в спецификации HTML 4.

На практике можно увидеть, что в большинстве случаев создание макета начинается с присваивания `display: block` различным изображениям, или элементам-контейнерам, содержащим изображения и заголовки. Такой способ позволяет избежать выравнивания нижнего края изображения по базовой линии текста, которое используется по умолчанию, а также улучшает удобочитаемость источника: можно добавлять пробелы между изображениями в разметке, сохраняя нулевые отступы между непрерывно следующими изображениями.

Что касается последнего, рассмотрим пример, в котором три изображения расположены подряд: `<img /> <img /> <img />`. При использовании стилей по умолчанию агента пользователя эти промежуточные пробелы будут визуализированы. Но если свойству `display` этих изображений присвоить значение `block`, то их можно будет расположить в ряд в том порядке, в котором они указаны в разметке, используя дополнительное значение `float`. Тогда их края будут расположены вплотную друг к другу (однако если ширины контейнера будет недостаточно, изображение переместится ниже своего предшественника в потоке документа). В тех случаях, когда и форматирование источника, и качество отображения имеют высокий приоритет, почти невозможно обойтись без применения `display: block` к изображениям.

Но если вам по каким-либо причинам требуется нарезка изображений — как в случае с сайтами, к которым применяются устаревшие технические требования — обычно лучше отказаться от задачи безупречного форматирования разметки и убрать все пробелы (включая переносы строк) между изображениями. Тем не менее такие блоки все же можно отформатировать, если начинать каждую строку разметки с пары атрибут/значение, а не с тега `img`.

Подготовка изображения к обработке

Если ваша работа начинается с документа Photoshop, в котором используется множество дополнительных цифровых ресурсов и схем расположения, то самое главное определить цель, а именно: является ли данный рисунок деталью дизайна или фактическим содержимым?

Детали дизайна в большинстве случаев следует относить к фоновым изображениям, с возможным (и иногда весьма вероятным) исключением типа растровых заголовков. Для получения дополнительной информации о создании и оформлении таких изображений см. главу 9.

Если, напротив, изображение определяется как содержимое (например, фотография или иллюстрация утверждений, сделанных в документе), для него следует использовать элемент `img`, задав по крайней мере URI элемента и альтернативное текстовое содержимое.

Что такое атрибут alt

Атрибут `alt` необходим в тех случаях, когда с сайтом работают люди с ограниченными техническими возможностями; его наличие становится менее важным только тогда, когда изображения загружаются на страницу *и* просматриваются *точно* так, как это было задумано. Во всех остальных случаях на странице отображается значение этого атрибута. Это нужно для того, чтобы изображение было понятным с точки зрения содержания; поэтому значение атрибута `alt` должно сообщать пользователю либо существенную информацию, либо ничего вообще. Довольно удобно рассматривать атрибут `alt` как заголовок; второй вариант — задавать с его помощью тему изображения (в случае, если заголовок уже существует).

Если встроенные изображения используются в качестве деталей дизайна, не имеющих явного текстового эквивалента (то есть все, кроме растровых заголовков), значению атрибута `alt` должна быть присвоена пустая строка (`alt=""`). Текстовые браузеры и программы экранного доступа уберут все упоминания об этих изображениях, как если бы для них свойству `display` было присвоено значение `none`.

Размеры и границы изображения

В самых первых браузерах рисунков было принято явно указывать значения `width`, `height` и `border` для правильного отображения рисунков. Причина заключалась в том, что эти браузеры не могли отображать какую-либо часть страницы до тех пор, пока не рассчитана вся схема расположения — процесс, который не может быть завершен без точной информации о размерах изображения. При отсутствии явных значений `width`, `height` и `border` для отображения страницы необходимо было, чтобы все изображения были загружены, а с соединением 56 Кбит/с, которое было распространенным когда-то, это могло занять несколько минут.

Усовершенствованные технологии освобождают посетителей сайтов от необходимости ждать отображения рабочей области браузера. Тем не менее не будет лишним указать данные о размере изображения в разметке или CSS для того, чтобы расположение элементов на странице не менялось в ходе ее загрузки — результат, который может быть раздражающим независимо от скорости соединения.

Вопрос к разработчику: где *задать* эти значения: в таблице стилей или в разметке? Достоинства и недостатки каждого подхода приведены в табл. 11.1.

Таблица 11.1. Достоинства и недостатки задания размеров изображения в разметке и таблице стилей

Метод	Достоинства	Недостатки
Разметка	Поведение макета остается неизменным независимо от проблем с надежностью сети или сервера	Разметку следует изменять сразу после изменения изображений
	Пары атрибут/значение работают как обычные метаданные	Значения, заданные в разметке, имеют приоритет перед стилями, противоречащими этим значениям
Таблица стилей	Детализированное художественное оформление компенсируется уменьшением времени работы	Отделение свойств изображения от отображаемой информации может привести к неудобству использования изображений в некоторых ситуациях
	Изображения проще располагать каскадом	Плохое художественное оформление требует использования большого количества идентификаторов `class` и/или `id`

Обращаем внимание читателя на важность качественного художественного оформления. Из всех веб-технологий в CSS, пожалуй, больше всего приветствуется логичность дизайна. Рассмотрим дизайн, в котором все изображения одной тематической группы или одного раздела страницы имеют одинаковые параметры отображения. Вместо того чтобы использовать всевозможные значениz `width`, `height`, `class` и `id`, можно пойти коротким путем и написать следующие правила.

```
#bodyCopy img {
    float: right;
    clear: right;
    width: 288px;
    height: 144px;
    border: 0;
}
```

Или же можно использовать такой код.

```
body.annualReports .graph {
    display: block;
    width: 478px;
    height: 200px;
```

```
    margin: 1em auto 1em auto;
    border: 0;
}
```

Далее можно будет начать загрузку изображений, не задумываясь о расположении изображений.

Упоминание границ также заслуживает внимания, особенно учитывая тот факт, что границы могут быть добавлены непосредственно к изображению. Однако нельзя игнорировать привычку некоторых пользовательских агентов добавлять границу шириной в 5 пикселов вокруг связанных изображений, поэтому в любой таблице стилей приветствуется использование такого правила:

```
a img { border: 0; }
```

Обработка изображений

К рисункам, которые являются самостоятельными элементами содержимого страницы, обычно применяются те же основные правила построения, что и к любым другим графическим объектам того же типа (имеющим отношение к изобразительному или прикладному искусству, инфографике и т. п.). Несмотря на то, что исходные изображения могут быть великолепными, вам, возможно, захочется применить некоторое количество простых способов обработки перед загрузкой изображений в репозиторий сайта, в тестовую среду или среду обработки. Следующие пассажи, посвященные созданию и обработке изображений, предназначены главным образом читателям с большим техническим опытом.

Обрезка

Вы часто будете иметь дело с каталогом изображений, каждое из которых отличается форматом, качеством композиции или и тем, и другим одновременно. Чтобы удобно расположить их в макете, вам неоднократно придется использовать обрезку.

Дальнейшие инструкции описывают детальную последовательность действий, используемых в Adobe Photoshop. Для краткости мы не будем останавливаться на функциях преобразования Crop Tool, Edit → Transform и Image → Rotate Image.

В такой ситуации лучше всего использовать Rectangular Marquee Tool (верхний пункт в левой колонке панели инструментов, состоящей из двух столбцов, и второй пункт в панели инструментов из одного столбца), чтобы выбрать область, ограниченную Fixed Aspect Ratio, которая будет сохранена в памяти. Если

возможно, при выделении курсор следует перетаскивать так, чтобы сохраненное изображение наиболее соответствовало правилу третей, которое обсуждается на сайте, сопровождающем книгу.

Как только выделение сохраняемой области закончено, обрезку можно завершить, выбрав Image4Crop в меню приложения.

В данном случае лучше всего использовать панель Rectangular Marquee Tool Options в том виде, который она имеет, когда для инструмента выбран режим Fixed Aspect Ratio (рис. 11.1).

Рис. 11.1. Панель Rectangular Marquee Tool Options с выбранным режимом Fixed Aspect Ratio

Rectangular Marquee Tool предпочтительнее, чем Crop Tool, так как поведение Snap To последнего не позволяет корректировать мелкие ошибки ввода.

Матирование: создание воображаемой «рамки»

Если изображение не может быть обрезано без удаления важных деталей, его можно матировать до желаемых пропорций с использованием диалогового окна Canvas Size (рис. 11.2).

Рис. 11.2. Фрагмент диалогового окна Canvas Size, используемого для матирования

Чтобы матировать изображение с использованием диалогового окна Canvas Size, выполните следующие действия.

1. Убедитесь в том, что на панели инструментов отображается нужный цвет фона.
2. Принимая во внимание желаемые пропорции изображения, рассчитайте его новые размеры.

3. Используйте Image→Canvas Size..., чтобы открыть диалоговое окно Canvas Size, и замените отображенные (текущие) значения новыми.
4. Привяжите существующую область изображения к соответствующему углу или краю, если это необходимо.
5. Повторите шаги 2–4 по необходимости, пока к изображению не будет добавлено нужное матирование.

Диалоговое окно Canvas Size также может быть использовано для добавления границы к изображению. Чтобы это сделать, задайте нужный цвет фона, установите флажок Relative и введите значения, соответствующие насыщенности границы, которую вы бы хотели получить.

Ресемплинг: изменение абсолютного размера изображения

Диалоговое окно Image Size похоже на Canvas Size, за исключением того, что с его помощью можно эффективно менять разрешение изображения. При его использовании я советую следовать рекомендациям, приведенным ниже.

- Если вы не намереваетесь «расплющить» изображение, оставьте выбранным параметр Constrain Proportions.
- В диалоговом окне лучше изменять значения Width и Height, а не значение Resolution.
- По возможности избегайте повышения разрешения (увеличения) изображений. Если вам *необходимо* увеличить разрешение изображения, минимизируйте коэффициент увеличения. При значении этого коэффициента более 10 % в областях высокой контрастности изображение будет размытым.
- Если вы уменьшаете разрешение изображения с коэффициентом более –20%, используйте Unsharp Mask Filter (Filters→Sharpen→Unsharp Mask). Это позволит уменьшить размытие, которое появится в местах высокой контрастности (рис. 11.3). Экспериментальным путем вы найдете подходящую комбинацию значений Amount, Radius и Level.
- Если вы работаете с изображением, не имеющим очевидных дефектов цифрового сжатия, используйте бикубический алгоритм изменения разрешения. Если изображение *имеет* очевидные дефекты сжатия, используйте билинейный алгоритм. Алгоритм «ближайшего соседа» предусмотрен специально для создания эффекта «приближения» (в таком случае будут видны пикселы) при недробных коэффициентах увеличения.

На рис. 11.2 показаны данные рекомендации в действии. В любом случае лучше создавать копию изображения, измененного до нужного разрешения, вместо того, чтобы разрешать браузеру или серверу делать эту работу за вас. Инструменты для обработки изображений не в Сети позволяют задавать выходное качество в виде основного параметра изменения изображения, в то время как и браузер, и сервер будут пытаться изменить изображение с наименьшим использованием ресурсов, причем за счет ухудшения качества.

Рис. 11.3. Три вида одной вкладки: исходный размер, одна треть исходного размера и одна треть исходного размера после применения фильтра Unsharp Mask

Изменения уровня: оптимизация контраста фотографий

Безупречная фотография напрямую из камеры — редкость, так как в тот момент, когда открывается затвор объектива, на результат влияет множество факторов окружающей обстановки, которые фотограф не в состоянии контролировать.

Одним из самых важных инструментов, используемых для устранения недостатков в Photoshop, является диалоговое окно Levels (Image→Levels). В нем есть пять ползунков, каждый из которых изменяет гамму (то есть диапазон) выбранного канала: верхние три увеличивают диапазон, нижние три — уменьшают. Хотя это и эффективное средство управления контрастностью и яркостью, следует все же обращать внимание на гистограмму, которая позволяет точно узнать, где *должны* находиться границы цветовой гаммы фотографии, вместо того чтобы добавлять изменения статически (и, по сути, произвольно).

На рис. 11.4 показано, как может выглядеть диалоговое окно Levels. В первом случае вы видите гистограмму неизмененной фотографии, во втором — регули-

ровку гаммы с учетом первой гистограммы, в третьем — вид гистограммы после применения изменений гаммы. Появление полос в последнем случае является следствием округления, которое было использовано для применения увеличения цветовой гаммы. Лучший способ исправить это — ресемплинг фотографии. {3 таком случае происходит искажение областей высокой контрастности, в результате чего пустые полосы, как правило, заполняются.

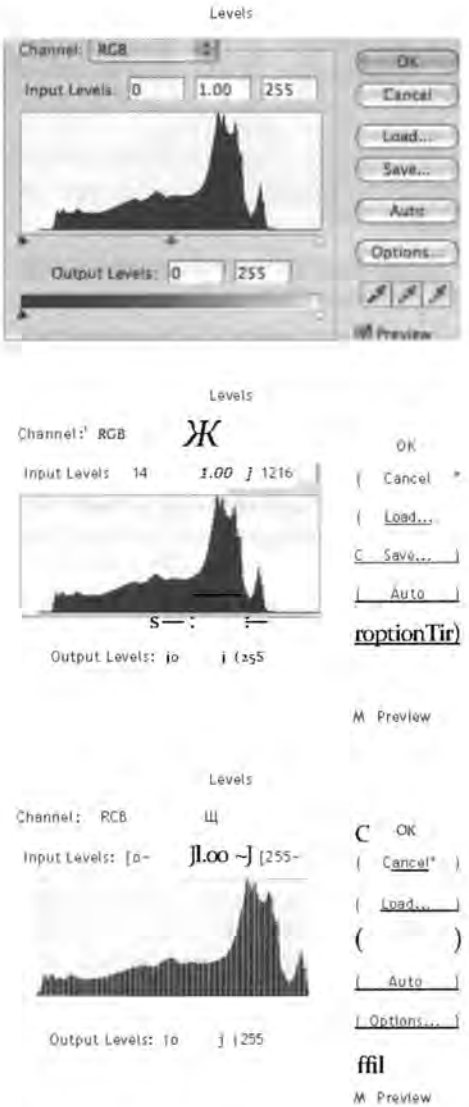


Рис. 11.4. Три примера увеличения контрастности изображения в диалоговом окне Levels в Photoshop

Полезный совет по оптимизации контрастности фотографии выглядит так.

> Когда гистограмма уровней достигает максимума где-то посередине, самые эффективные настройки контрастности достигаются, если сжать график за «плечи».

Применение нескольких настроек

Когда к изображению необходимо применить более чем одну из описанных выше настроек, их следует применять в следующем порядке.

1. *Обрезка*: если уровни изображения задаются до обрезки, эти настройки будут соответствовать гистограмме, отличной от той, которая описывает обрезанное изображение.
2. *Уровни*: без последующего ресемплинга промежутки в гамме фотографии будут перемешаны, а не удалены.
3. *Ресемплинг*: изменение разрешения изображения после настройки уровней позволяет программе обработки интерполировать высококонтрастные области (эта возможность теряется, если изменение разрешения производится сначала).
4. *Матирование*: интерполяция, которая заполняет промежутки в гистограмме изображения с измененным разрешением, размоет внутренние края матированного изображения, если матирование не будет произведено раньше.

Работа с цветовыми профилями

Как уже было сказано в главе 9, то, что вы видите при создании или реализации дизайна сайта, совсем *не обязательно* совпадает с тем, что будут видеть посетители сайта. Причиной таких отклонений чаще всего служит качество аппаратных средств, однако немаловажную роль играют также и *предположения*, исходя из которых аппаратные средства настраиваются.

Каждое из средств, обычно используемых для отображения веб-документов, — сюда относятся жидкокристаллические дисплеи (LCD), экраны на основе катодно-лучевых трубок (ЭЛТ, тип ТВ), проекторы, листы универсальной бумаги, бумага с покрытием — имеют различные физические характеристики, которые определяют, как отображаются цвета. Все эти носители данных изготовлены исходя из предположений о цветовой гамме, которую они будут отображать, и уровня окружающего освещения, доступного зрителю. *Все эти* факторы (и некоторые другие) влияют на цвета, которые видят потребители контента вашего сайта, поэтому графические файлы часто содержат *цветовые профили*, задающие особенности цветопередачи инструментов, используемых

для создания этих файлов. На рис. 11.5 описаны отношения между цветовыми профилями устройств и носителей.

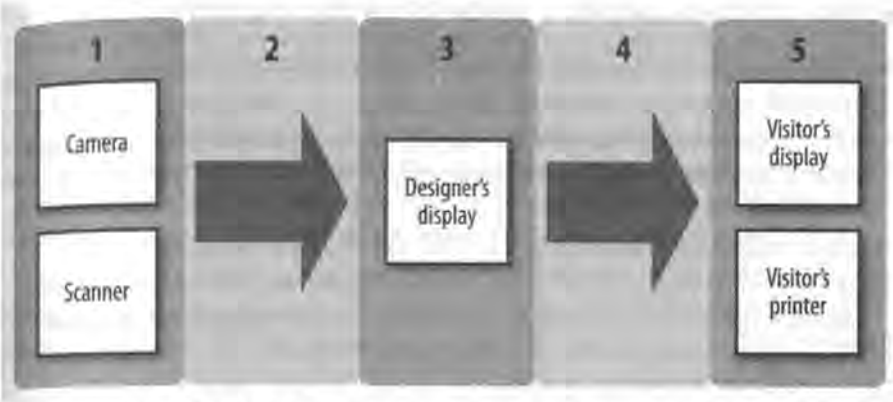


Рис. 11.5. Пять этапов управления цветом: создание, загрузка, изменение, передача, потребление

В середине 1990-х компании Microsoft и Hewlett-Packard попытались решить проблему различающихся носителей информации, создав *цветовое пространство sRGB* — физическое описание цвета, которое может быть использовано в дизайне и настройке электронных аппаратных средств отображения информации. Цветовое пространство sRGB рекомендовано W3C и включено в спецификацию Scable Vector Electronics (SVG) — один из двух графических форматов, созданных в ходе деятельности W3C.

Хотя sRGB — не единственное используемое цветовое пространство, но самое распространенное в Сети. Браузер Safari компании Apple, как и следовало ожидать, использует профили International Color Consortium (ICC), а Firefox 3 поддерживает профиль ICC, однако по умолчанию он отключен.

При работе с цветовыми профилями следует обратить внимание на следующие особенности.

- Если у вашего монитора есть цветовая настройка sRGB в дополнение к списку белых точек, выберите параметр sRGB. В противном случае используйте заводские настройки белой точки.
- Если исходное изображение содержит встроенный профиль, преобразуйте его цветовое пространство в sRGB перед началом обработки.
- В Photoshop диалоговое окно **Save for Web** отменяет цветовой профиль перед записью изображения на диск. Поэтому его не следует использовать в том случае, если качество изображения имеет больший приоритет, чем размер файла.

Можно с уверенностью предположить, что изображение, которое не содержит данных о цветовом профиле, будет отображаться в цветовом пространстве sRGB. Поэтому перед загрузкой на веб-сервер изображения всегда следует преобразовывать в цветовое пространство sRGB.

Оптимизация изображений

Выбор правильного формата изображения

Надежной поддержкой в Сети обладают три формата изображений.

- Graphics Interchange Format (GIF).

  GIF — самый старый из трех форматов, распространенных в Сети. Как правило, он используется для изображений, в которых много областей с однородной заливкой.

- Joint Photographic Experts Group (JPEG) File Interchange Format.

  Как следует из названия, формат JPEG особенно хорошо подходит для фотографий и поддерживается всеми цифровыми фотоаппаратами. Кроме того, JPEG — единственный распространенный формат с надежной поддержкой встроенных цветовых профилей.

- Portable Network Graphics (PNG).

  Спецификация PNG была разработана и утверждена W3C и являлась реакцией на ограниченность формата GIF с точки зрения технических характеристик и доступности. В действительности PNG превосходит GIF практически по всем параметрам, кроме одного: как и следовало ожидать, Internet Explorer 6 не поддерживает альфа-каналы PNG. Эта проблема обсуждается в главе 14.

  Четвертый формат, который называется Scalable Vector Graphics (SVG), на самом деле является диалектом XML. И хотя SVG обладает рядом впечатляющих свойств — а так же, как и PNG, является полностью открытым форматом — он не до конца поддерживается основными веб-браузерами (а Internet Explorer вообще его не поддерживает).

  Если изображение состоит из штриховых линий или содержит большие области однородной заливки, для него лучше использовать формат GIF или PNG. PNG лучше подходит в тех случаях, когда требуется использовать более одного уровня прозрачности (то есть вкл./выкл.).

  Фотографии следует всегда сохранять в формате JPEG. Использовать формат JFIF для изображений, не обладающих фотореалистичностью, — не очень хорошая идея, поскольку JFIF является форматом сжатия «с потерями» и предполагает постепенное, но сильное различие в контрасте между областями изображения. Изображение, для которого это окажется неверным, будет содержать множество неприятных искажений сжатия.

Как найти золотую середину между размером и качеством

Перед сжатием размер изображения можно уменьшить одним из двух простых способов: уменьшить объем (понизить разрешение) или уменьшить контрас-

тность. Оба эти метода напрямую и почти всегда отрицательно влияют на качество изображения, поэтому их следует применять с осторожностью.

Эти два метода уменьшают размер файла изображения по-разному. При понижении разрешения количество пикселов, для которых будет выполнено сжатие, уменьшается логарифмически — иными словами, быстро относительно постепенного уменьшения размеров изображения. *Слегка или умеренно* уменьшить контрастность — тоже эффективный способ, так как он увеличивает диапазон яркости изображения, что подходит как для форматов сжатия с потерями (JPEG), так и для форматов сжатия без потерь (GIF, PNG). Более подробно эта идея обсуждается на сайте, сопровождающем книгу.

Если перед нами запущенная программа Photoshop и набор изображений, которые требуется обработать для последующего использования при создании сайта, нам остается решить вопрос о показателе качества выходных данных. В случае JPEG-изображений показатель качества контролируется явно кодеком JFIF. В случае PNG- и GIF-изображений качество является более субъективным показателем; специалист уменьшает качество таких изображений понижая глубину цвета и индексируя цвета, которые должны сохраниться (при этом могут появиться искажения). *В общем случае*, самые лучшие настройки для GIF-изображений и низкокачественных PNG-изображений вы найдете в режиме Indexed Color при глубине цвета 2, 16, 64 или 256.

После того как вы выберете глубину цвета для изображения, которому требуется сжатие без потерь, вам необходимо будет выбрать метод дизеринга. При индексировании цветов изображения Photoshop предлагает четыре варианта, которые приведены ниже в соответствии с качеством выходных данных при низкой глубине цвета.

1. Pattern.
2. Diffusion.
3. Noise.
4. None.

Такая иерархия в некотором смысле субъективна; чем ниже контрастность изображения или число значимых цветов, тем меньше вероятность того, что случайный пользователь заметит очевидно неправильный выбор палитры или настроек дизеринга.

Публикация изображений

Публикация изображений может показаться простой и понятной задачей, однако это не всегда так. Папки назначения, правила именования файлов и поведение Системы управления содержимым (Content Management System, CMS) также имеют отношение к публикации изображений (в различной степени в зависимости от конкретного проекта).

Сохранение изображений в порядке

Если вы работаете над сайтом достаточно высокого качества, он, скорее всего, будет содержать десятки или сотни изображений. На более крупных сайтах (особенно тех, которыми управляют сервисы социальных сетей) часто хранятся сотни тысяч или даже *миллионы* отдельных файлов изображений.

Итак, если вы хотите сохранять все это в порядке, что вы будете делать?

Иногда организация графических ресурсов сайта отражает устройство самого сайта; единственная серьезная проблема возникает в случае, если изображений так много, что их требуется распределить по нескольким каталогам. Для маленьких сайтов я рекомендую загружать файлы в папку /images, а для названий использовать набор ключевых слов (например, /images/bg_sidebar_contact.jpg).

В общем виде название может быть представлено таким образом: type_pagescope_sitescope.ext.

Можно использовать и другие ключевые слова: описания тем, фамилии художников и даты создания адекватно отражают содержимое изображения (по крайней мере, для специалиста, знакомого с сайтом).

Ключевые слова в имени файла могут быть расположены в том порядке, который вам кажется наиболее подходящим. Я предпочитаю сначала указывать тип, зная по опыту, что в таком случае удобнее просматривать списки.

Для более крупных сайтов часто бывает удобнее создавать папки images для каждого большого раздела сайта. Тогда вместо /images/article_photo_reports_2009q2_factoryfloor.jpg вы сможете использовать в качестве значения атрибута `src`, например /reports/images/article_photo_2009q2_factoryfloor.jpg, оставляя корневую папку /images для графических ресурсов, общими для нескольких разделов сайта.

Еще одна проблема заключается в том, как и где следует размещать содержимое, которое было изменено. В случае крупных сайтов для хранения всего содержимого удобнее использовать систему управления версиями (Revision Control System, RCS) наподобие Subversion или Git. Если вы работаете над новой версией небольшого сайта, удобнее всего создать папку, содержащую данные о времени ее создания (например, метку даты или порядковый номер), и переписать в нее «старые» файлы, загрузив «новые» файлы в основные папки.

Публикация изображений и управление сайтом с помощью CMS

Безопасный доступ к расположению многих изображений, использующихся на пользовательских сайтах, осуществляется только с помощью панели управления Системы управления содержимым (CMS). На первый взгляд, этот метод не кажется плохим; если установлены расширения для работы с изображениями, CMS может автоматически задавать значения `src` и при необходимости

обеспечивать предварительный просмотр изображений. Однако здесь все же есть ряд недостатков, которые следует учитывать операторам сайтов.

- Интерфейсы для предварительного просмотра изображений неудобны в использовании.
- Правила именования файлов либо слишком строгие, либо слишком общие, благодаря чему работать с изображениями через галерею предпросмотра еще более неудобно.
- Реализация и поведение различаются на разных сайтах (в зависимости от используемой CMS и установленных расширений).
- В силу необходимости требования к оформлению изображений крайне строгие — факт, который не могут или не хотят принимать многие непрофессиональные пользователи.

Чтобы успешно обойти эти недостатки, лучше придерживаться следующих принципов.

- По возможности используйте инструменты для публикации, которые сохраняют сами изображения в доступной папке файловой системы хоста.

  У этого подхода есть свои недостатки; на самом деле вопрос, касающийся места хранения изображений в CMS, вызывает больше всего споров среди разработчиков CMS. Однако если изображения хранятся в файловой системе, профессионалу значительно проще обновлять их быстро и сразу.
- Лучше использовать инструменты для публикации с панелями управления, поддерживающими полнотекстовый поиск в именах изображений и описаниях.

  На практике оказывается, что такие инструменты поиска работают медленно и часто возвращают плохо отсортированные и недостоверные данные, но если единственной альтернативой является просмотр сотен страниц галереи изображений, приходится использовать то, что есть.
- Постарайтесь выбрать одну платформу CMS, которой вы будете отдавать предпочтение, и развивайте свои навыки работы с ней.

  Такой шаг имеет отношение к развитию хороших привычек; у любого инструмента есть свои дефекты и особенности, так что ваше лучшее оружие — опыт.
- Опишите основные принципы, касающиеся создания вашего фирменного стиля, и следите за их исполнением.

  Некоторые читатели не станут этого делать, но остальные вскоре обнаружат, что от обучения неопытных пользователей принципу «что посеешь, то и пожнешь» они выиграют во много раз больше, чем потратят на объяснение этого принципа. Понятие руководства по стилю оформления было введено в главе 5 и подробно рассматривается на сайте, сопровождающем книгу.

Поддержка высокого качества содержимого управляемого сайта одна — из самых трудных задач, с которыми сталкивается добросовестный разработчик, и решать ее проще всего вместе с опытными пользователями и достаточно

эффективной Системой управления содержимым. В ситуациях, когда контроль качества ведется не так тщательно, разработчикам, возможно, придется оставить все как есть — а это обычно означает позволить пользователям по возможности самим исправлять свои ошибки.

Правила публикации изображений

Поскольку порядок доступа к изображениям ничем не отличается от порядка доступа к страницам, значения `src` задаются по такому же принципу, что и значения `href`. Поскольку сделать это очень просто, неопытные и неразборчивые пользователи часто публикуют на своих сайтах изображения, которые на самом деле хранятся на других сайтах.

Когда речь идет об изображениях, старайтесь не быть неразборчивыми или наивными.

Самый лучший метод публикации изображений *требует* выполнения следующих условий.

1. Разрешение, явно выраженное или подразумеваемое, на публикацию изображения в контексте вашего URI было предоставлено правообладателем.
2. Должным образом указаны создатели изображений или, по крайней мере, их правообладатели.
3. Файлы изображений, опубликованные в контексте ваших URI, должны быть размещены также на ваших URI.

Невыполнение первого условия является нарушением закона об авторских правах, второго — крайней грубостью, а нарушение третьего приводит к тому, что вы пользуетесь сетевыми ресурсами, за которые платит кто-то другой, что эквивалентно краже.

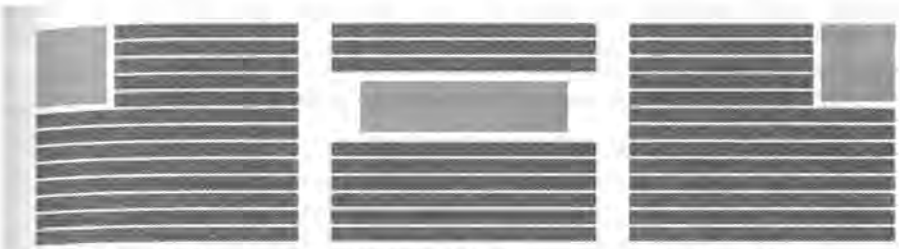
Заголовок запроса любого изображения, запрашиваемого через элемент `img`, содержит соответствующий URI в поле `referrer`. Если вы обнаружили, что какой-то недобросовестный оператор сайта нарушает третье условие в отношении ваших ресурсов, вы можете изменить настройки сервера так, чтобы он проверял источники запросов на изображения и в случае, если запрос пришел не с вашего URI, выдавал ответ, содержащий коды состояния 3xx–5xx.

Оформление изображений и встраиваемого контента

Если обратить внимание на поведение элементов `inline-block` (таких, как изображения) в макете, становится ясно, что для успешной публикации изображений все-таки стоит добавить хотя бы *немного* оформления.

Расположение изображения в колонке

На практике вам очень часто придется выравнивать изображения вдоль полей колонки или располагать их по центру относительно родительской колонки; стили, которые при этом обычно используются, показаны на рис. 11.6.



```
display: block;
float: left;
clear: left;
margin: 0 [x] [y] 0;
```

```
display: block;
width: [original];
clear: both:
margin: 0 auto;
```

```
display: block;
float: right;
clear: right;
margin: 0 0 [x] [y];
```

Рис. 11.6. Стандартные пары свойство/значение, использующиеся для форматирования строковых изображений с помощью CSS

Среди приведенных на рисунке пар свойство/значение есть пара `display: block`, которая нормализует поведения элементов в макете. Использование ее вместе с селектором `class` облегчает создание оформления для заголовков.

Создание заголовков для изображений

Если изображение отформатировано так, как показано на рис. 11.6, проще всего добавить к нему заголовок одним из следующих двух способов.

- Матируйте исходное изображение, создайте растровый тип в свободной области и задайте текст заголовка с помощью значений атрибутов `alt` и `title`.
- Добавьте текст заголовка сразу после изображения, затем заключите и то и другое в элемент `div` (или, что используется реже, в параграф). Для этого элемента задайте стили, аналогичные приведенным на рисунке.

Первый вариант, в отличие от второго, требует меньше разметки; кроме того, он гарантирует, что картинка всегда будет отображаться в соответствующем контексте. Принципиальный недостаток первого метода состоит в том, что качество изображения увеличивается несоразмерно добавленному содержимому. Хуже всего это сказывается на фотографиях, поскольку фотореализм в сочетании с однородными областями цвета требует гораздо более высокого качества, чем обычная фотография.

Если заголовок, встроенный в изображение, требуется скрыть, то для элемента, содержащего в себе все изображение, можно задать значения `height` и `overflow` так, чтобы заголовок оказался обрезанным.

```
div#altProductPhoto {
    float: left;
    width: auto;
    height: 176px;
    overflow: hidden;
}
```

Если вы хотите добавить заголовок в виде обычного текста, можно использовать такие стили.

```
div#productPhotoWithCaption {
    float: left;
    width: 176px;
    margin: 0 9px 9px 0;
    font-size: small;
    font-style: italic;
}
```

Во всех случаях, когда заголовок добавляется в виде простого текста, родительскому элементу `div` должно быть приписано значение `class` или `id`, сообщающее о том, что у рисунка есть заголовок.

Работа с эскизами в режиме галереи и показа слайдов

Существует два распространенных типа ресурсов, с которыми связаны одни и те же проблемы: эскизы галерей/слайд-шоу и интерфейсы приложений. Их реализация вызывает проблемы как в теории, так и на практике, и причина этого состоит главным образом в особенностях поведения замещенных элементов. Такие проблемы имеют множество решений: вы окажетесь (более-менее) правы, если будете использовать элементы `div` с атрибутами `class`, списки или даже таблицы.

Судя по моему опыту, наилучшим компромиссом между анонимностью и краткостью является использование упорядоченных или неупорядоченных списков, особенно в тех случаях, когда дизайн страницы предполагает произвольное число изображений в наборе. Списки определений — также возможный вариант для изображений с заголовком, если точные размеры макета известны заранее. Используя относительное позиционирование для контейнера `dl` и абсолютное позиционирование для различных элементов `dt` и `dd`, можно без труда закрепить за изображением и заголовком точные координаты.

Однако прежде, чем приступить к работе, следует обратить внимание на следующие вопросы.

- Все ли изображения имеют одинаковые размеры?
- Если изображения имеют различные размеры, можно с помощью матирования или обрезки сделать размеры одинаковыми?
- Если нет, то как следует выравнивать изображения и их метки?

Первые два вопроса относятся к ситуации, когда эскизы имеют (или должны иметь) одинаковые размеры. Изображения одинакового размера используют общие поля, в результате чего верстка становится более четкой. Более того, ImageMagic, G2 и другие библиотеки для работы с изображениями позволяют матировать и обрезать изображения на сервере. О том, как делать это с помощью ImageMagic, рассказывается в книге Shelly Powers «Painting the Web» издательства O'Reilly.

В общем, для правильного расположения эскизов изображений и другого контента с большим количеством плавающих элементов лучше использовать такой простой принцип.

При работе со смежными плавающими элементами вам придется выбирать между использованием изображений с заранее заданными размерами и использованием пустых элементов и свойства `clear`, если вы хотите добиться гармоничного расположения эскизов по вертикали.

Об использовании свойства `clear` подробно рассказывается в разделе «Свойства `float` и `clear`» на с. 110.

Для примера возьмем неупорядоченный список и эскизы изображений одинакового размера (240×180). Тогда стили будут выглядеть приблизительно так.

```
ul#thumbs { margin: 0; padding: 0; list-style-type: none; }
   ... li { display: block; float: left; width: auto; padding: 15px; }
... li img { display: block; width: 240px; height: 180px; }
```

Как вы видите, для эскизов одинакового размера, безусловно, стоит использовать стилевое оформление. Независимо от размера самих изображений важно, чтобы соседние *контейнеры* имели общую ширину или высоту. Это необходимо из-за особенностей поведения элементов, к которым применено `float: left` (рис. 11.7).

Рис. 11.7. Пример поведения плавающих элементов

На рисунке элемент 6 перемещается под элемент 5 потому, что иначе он бы вышел за границы своего контейнера. Если бы он был шире, чем расстояние между рамкой элемента 4 и правым полем общего контейнера, он бы переместился под элемент 2.

Элемент 7 вышел бы за границы контейнера, если бы располагался рядом с элементом 6 (где он должен быть по умолчанию), поэтому он помещается сразу под элементом 6 и сдвигается влево насколько это возможно, то есть к левому полю контейнера. В отсутствие выхода за пределы контейнера и наложений блок элемента должен иметь с предыдущим плавающим элементом общее верхнее поле — именно так расположены элементы 2–5 и 8–12.

Если несколько смежных изображений имеют одинаковые размеры, можно достоверно предсказать их поведение в макете. Благодаря этому гораздо проще избежать ситуаций, подобных той, которая изображена на рис. 11.7.

Lightbox: эскизы, галереи и показ слайдов

Специалистам по контенту и дизайнерам, которые не знают JavaScript или вынуждены использовать дополнительные инструменты для работы с контентом, перегруженным изображениями, стоит познакомиться с Lightbox. Это еще незавершенный проект независимого дизайнера и разработчика Локеша Дакара.

Вполне вероятно, что вам уже доводилось видеть результат применения Lightbox (или одного из его клонов) на других сайтах. Если после щелчка мыши на эскизе изображения открывается дополнительное окно с этим изображением, а фон затеняется, то, скорее всего, такой эффект был создан с помощью Lightbox.

Lightbox запускается непосредственно в браузере; его текущая версия использует общедоступные библиотеки JavaScript. Работать с Lightbox довольно просто: для этого необходимо сделать следующее.

1. Загрузите исходные изображения в нужную папку.
2. Загрузите и подключите необходимые файлы CSS и JavaScript.
3. С помощью элементов `a` добавьте в разметку ссылки на изображения, загруженные ранее (пункт 1); для каждой ссылки задайте значение `rel="lightbox"`.

Такой способ не всегда является универсальным и оптимальным решением, но если, не зная JavaScript, вы хотите добавить эффектное оформление, причем в сжатые сроки, Lighbox может оказаться превосходным инструментом для создания эскизов, галерей и слайд-шоу.

SlideShowPro

Если Lightbox и специальные библиотеки для показа слайдов не соответствуют вашим требованиям (например, если необходимо, чтобы работать со слайдами

могли даже неспециалисты), можно сделать еще один шаг вперед и установить инструмент под названием SlideShowPro — популярную платформу показа слайдов для сайтов предприятий.

Созданная Тоддом Домини и распространяемая его компанией Dominey Design, SlideShowPro основана на возможностях Flash и является упрощенной платформой для создания эффектов, используемых для показа слайдов. Ни для кого не секрет, что JavaScript-расширения способны выполнять сходный набор функций, однако для этого обычно требуется гораздо больше ресурсов, поскольку использование JavaScript требует тщательного тестирования продукта для каждого отдельного браузера. С помощью платформы SlideShowPro, обладающей исчерпывающей документацией, создавать галереи и слайд-шоу в среднем проще. Поэтому единственная причина, по которой стоит отдавать предпочтение JavaScript — снижение зависимости от других платформ (то есть от Flash и от браузеров, для которых вы разрабатываете сайт).

Для успешной реализации дизайна с помощью SlideShowPro вам необходимо создать или приобрести Flash-файл и загрузить специальные ресурсы SlideShowPro. Если вы это сделали, то для создания и подключения слайдов необходимо выполнить следующие действия.

- Загрузите файлы на ваш сайт с помощью FTP и подключите их с помощью XML.
- Для подключения сетевых ресурсов используйте RSS.
- Установите дополнительное приложение SlideShowPro Director, которое выполнит загрузку, добавит изображения в базу данных и сгенерирует необходимые XML-файлы.

Движение и звук: добавление Flash-видео и Flash-презентаций с помощью SWFObject

Платформа Shockwave Flash от компании Adobe на сегодняшний день самый популярный инструмент для добавления аудио и видео в традиционный веб-контент.

Если вы хотите сделать процесс публикации готовых Flash-презентаций более простым, попробуйте использовать открытый JavaScript-ресурс SWFObject, созданный Джеффом Стернсом. Чтобы успешно решать проблемы, связанные с Flash-контентом (в частности, проблему определения версий), SWFObject использует DOM API и другие интерфейсы.

Если на сервере уже есть готовый Flash-контент и файл **swfobject.js**, разработчику остается всего лишь вызвать этот файл с помощью элемента `script` в теге `head` документа, написать одну строку разметки в основном документе и добавить к этой строке еще один фрагмент JavaScript-кода, отвечающий за создание

специфического для данного приложения объекта SWFObject. Этот объект, в свою очередь, изменяет разметку, создавая и заполняя элемент, в котором будет содержаться нужная Flash-презентация.

Вы также можете написать свою разметку для элемента `object` в соответствии со стандартами и затем выполнить скрипт SWFObject *только* для того, чтобы получить доступ к определению версий и другим возможностям.

Далее, ближе к концу главы, вы узнаете, что в HTML5 появятся новые возможности добавления аудио и видео без использования Flash.

Если у вас есть обычный видеофайл, который необходимо разместить в Сети, и вы выбрали Flash в качестве среды воспроизведения, простого использования SWFObject для публикации презентации будет недостаточно. Чтобы подготовить видеоматериал, вам необходимо выполнить несколько дополнительных действий:

1) *сохранить на диск;*
2) изменить разрешение;
3) перекодировать;
4) создать или приобрести SWF, необходимый для воспроизведения;
5) загрузить оба файла (готовый FLV и его контейнер SWF).

На практике особенности выполнения пунктов 1–3 зависят от выбора программы для редактирования. Вопрос редактирования видео в Сети заслуживает написания отдельной книги (и я с нетерпением жду того дня, когда она появится на прилавках).

Возможно, вам потребуется использовать один SWF-файл в качестве контейнера для нескольких FLV-файлов; тонкости размещения таких видео зависят от их взаимного расположения.

Добавление мультимедиа без контейнера

Вы наверняка замечали, что у каждого сайта есть свои инструкции по добавлению его мультимедиа на другие сайты. YouTube предлагает использовать следующую разметку.

```
<object width="425" height="344">
    <param name="movie"
    value="http://www.youtube.com/v/iG9CE55wbtY&amp;hl=en&amp;
        fs=1&amp;rel=0"></param>
    <param name="allowFullScreen" value="true"></param>
    <param name="allowscriptaccess" value="always"></param>
```

```
    <embed src="http://www.youtube.com/v/iG9CE55wbtY&amp;
          hl=en&amp;
          fs=1&amp;rel=0"
    type="application/x-shockwave-flash" allowscriptaccess="always"
    allowfullscreen = "true"
    width="425" height="344">
  </embed>
</object>
```

Vimeo предлагает такую разметку.

```
<object width="400" height="270">
    <param name="allowfullscreen" value="true" />
    <param name="allowscriptaccess" value="always" />
    <param name="movie"
    value="http://vimeo.com/moogaloop.swf?clip_id=4841397&amp;
    server=vimeo.com&amp;show_title=1&amp;show_byline=1&amp;
          show_portrait=0&amp;color=&amp;fullscreen=1" />
    <embed src="http://vimeo.com/moogaloop.swf?clip_id=4841397&amp;
    server=vimeo.com&amp;
    show_title=1&amp;show_byline=1&amp;show_portrait=0&amp;
    color=&amp;fullscreen=1"
    type="application/x-shockwave-flash"
    allowfullscreen="true"
    allowscriptaccess="always"
    width="400" height="270">
  </embed>
</object>
```

А сайт Odeo рекомендует добавлять такой код.

```
<object type="application/x-shockwave-flash"
    data="http://static.odeo.com/flash/player_audio_embed_v2.swf"
          width="325"
    height="60"
    id="odeo_audio">
    <param name="movie" value=
    "http://static.odeo.com/flash/player_audio_embed_v2.swf" />
    <param name="FlashVars" value="jStr=[{"id": 24363363}]" />
</object>
```

Во всех этих примерах используется элемент embed (который не соответствует спецификации HTML 4) внутри элемента object (который соответствует). Причина этого состоит в том, что элементы object могут содержать помимо элементов param еще и резервный контент, который будет использован в случае, если браузер не сможет загрузить контент, указанный в элементе object. В идеале таким резервным контентом будет не более чем изображение и кое-какой текст, но, в соответствии с законом Постела (будьте либеральны к тому, что принимаете, и консервативны к тому, что отправляете), нет реальных причин, почему элементы embed не могут использоваться для тех же целей.

Рассказ о трех компаниях

На заре развития Сети одна из фирм-производителей браузеров — Netscape — обладала очень сильными позициями, а Microsoft и другие мелкие игроки тогда еще не спешили набирать скорость. О роли CSS в этой битве за долю рынка рассказывается в разделе «Приоритеты поставщиков» на с. 67. Похожий раскол в отношении встраивания мультимедиа привел к такому же результату: идея Microsoft одержала верх на бумаге, тогда как предложение Netscape (или, скорее, его влияние) сохраняло свои позиции в течение еще нескольких лет. Лишь с недавних пор элемент для публикации изображений `object`, введенный W3C, обладает чем-то, отдаленно напоминающим кроссбраузерную поддержку. Интересно, что элемент `embed` использовался с самого начала и был единственным элементом для встраивания мультимедиа, обладающим поддержкой оригинальной платформы Netscape.

Споры по поводу кроссбраузерной поддержки достигли пика, когда корпорация Eolas (которая существует исключительно за счет того, что приобретает интеллектуальную собственность и продает лицензии на ее использование) в 1999 году выиграла иск против компании Microsoft: было установлено, что использование элемента `object` компанией Microsoft нарушает корпоративные права Eolas. Этот судебный процесс еще больше омрачил ситуацию, и в результате в Internet Explorer стал использоваться новый метод загрузки мультимедийного контента.

Наконец, поддержка элементов `object` и `param` продуктами Microsoft тесно связана с ее собственными API, поэтому на практике разработчики, вынужденные использовать Windows Media Player для воспроизведения мультимедиа, только выиграют от этого, так как получат доступ к полной документации. Крупные компании могут себе это позволить, в то время свободным разработчикам, которые хотят добиться поддержки своих продуктов в различных операционных системах, такие затраты — несколько сотен долларов в год — могут оказаться не под силу.

Использование Flash

Большая часть проблем, которые конечный пользователь видит в применении аудио- и видеоконтента, заключается в кодировании, что почти (но все же не полностью) синонимично сжатию. Представьте себе, сколько новых проблем появилось бы, если бы каждый производитель операционных систем использовал свой собственный метод сжатия изображений; именно поэтому с видеофайлами так трудно работать. Существует лишь один формат видео (MPEG1), который изначально поддерживается во всех системах и является самым старым и самым неэффективным из распространенных форматов. Все остальные форматы кодирования требуют специального программного обеспечения, такого, как Adobe Shockwave Flash.

Достоинство Flash состоит в том, что он необычайно широко распространен — по данным Adobe, 99 % на ПК. Ни одно из наименований программного

обеспечения не является таким распространенным; даже для Microsoft Windows (безотносительно версий и использующихся браузеров) доля рынка на момент публикации этой книги составляет 90–95 %.

YouTube и другие сайты, которые существуют за счет публикации и просмотра видеоконтента, генерируемого пользователем, показали, что Flash не только практичный, но и простой формат видео, благодаря чему платформа Flash стала занимать лидирующее положение среди платформ для воспроизведения аудио и видео.

SWFObject (созданный одним из инженеров YouTube) позволяет без труда добавлять Flash в веб-документы. Но как насчет других программ, например QuickTime и Windows Media Player? Что делать, если у вас нет доступа к нужному SWF-контейнеру для Flash-видео? Как добавить встраиваемое содержимое без использования недопустимого кода?

Использование простой разметки для публикации мультимедийного контента

При добавлении аудио- или видеоконтента обойтись без элемента `embed` трудно, если вы хотите сделать это «с нуля». Но оказывается, что кое-кто уже сделал за вас всю работу — протестировал и исправил нужные фрагменты кода.

Нам очень помог человек по имени Саймон Джесси, который с 2006 года занимается поддержкой протестированной и *валидной* разметки с использованием элементов `object` для Dreamhost. В примерах, доступных на момент публикации этой книги, содержится разметка для:

- Flash;
- QuickTime;
- Windows Media Player.

На первый взгляд, этот список может показаться слишком коротким, однако вместе эти три среды для воспроизведения применимы практически ко всему содержимому Сети.

Результаты работы Джесси можно посмотреть на сайте http://wiki.dreamhost.com/Object_Embedding.

Стили для встраиваемого контента

Проблема использования стилей для встраиваемого контента состоит в том, что такие элементы больше напоминают модальные диалоговые окна; встраиваемый элемент похож на окно в окне, а не на обычный веб-контент. Если вы захотите применить усовершенствованные приемы верстки к страницам со встраиваемым контентом, лучше всего добиться того, чтобы встраиваемый контент ни в коем случае не накладывался на другие элементы или параметры `select`. Иначе остальное содержимое страницы может оказаться безвозвратно скрытым.

Как решить проблемы встраиваемого контента с помощью поля заголовка HTTP Content-Disposition

Как вы наверняка знаете, операторы сайтов часто рекомендуют посетителям использовать параметр контекстного меню для сохранения файлов. Такой параметр называется **Save Target As...** в Internet Explorer, **Save Link As...** в Mozilla и **Save Linked File As...** в Safari. Однако такого рода ситуации не требуют дополнительных инструкций для пользователей.

Если вы не хотите надоедать пользователям или создавать для них неприятные ситуации, связанные с поддержкой, вы можете сделать так, чтобы вместо воспроизведения файла в браузере появлялось диалоговое окно загрузки. Для этого в заголовке ответа должна содержаться следующая строка:

```
Content-Disposition: attachment; file=/video/developers.avi
```

Как и с другими заголовками, вроде Content-Type, с этим заголовком удобнее всего работать с помощью HTTP API того серверного скриптового языка, который вы обычно используете. В результате вместе с отображением обычной страницы браузер будет сохранять дополнительный файл (в данном случае **developers.avi**) в папке, указанной в настройках браузера, или откроет диалоговое окно **Save As**. Так ведет себя, к примеру, сайт **download.com**.

Добиться такого поведения без полного обновления страницы можно с помощью функции `XMLHttpRequest`, однако использовать ее нужно с осторожностью; ключевым действием при этом является добавление соответствующего поля `Content-Disposition` в ответ HTTP.

Важно сохранять объективность

Несмотря на свою очевидную значимость, публикация мультимедиа была объектом всеобщей ненависти на протяжении всего существования Сети. Причина этого заключается главным образом в широкой распространенности низкокачественных дизайнерских решений. Но даже несмотря на это тем, кто диктует условия и подписывает чеки, обычно хочется видеть на своих сайтах разного рода украшения. При этом разработчик обязан следить за соблюдением стандартов юзабилити, использованием ресурсов, показателями качества и будущей совместимостью.

Элементы video и audio (HTML5)

Благодаря элементам `video` и `audio`, видео- и аудиоконтент наконец-то становится пассажиром первого класса в Сети. А это означает следующее.

- Добавлять видео- и аудиоконтент в веб-документы так же просто, как добавлять текст, гиперссылки и изображения.

- Видео и аудио можно использовать вместе с другими базовыми веб-технологиями: например, можно применять SVG-фильтры и создавать CSS для видеоконтента.

Этих целей можно достичь с помощью элементов `video` и `audio`, так как они позволяют встраивать видео- и аудиоконтент прямо в веб-документы без использования плагинов. А благодаря общему интерфейсу этих элементов HTMLMediaElement видео- и аудиоконтент можно программно обрабатывать с помощью скриптов DOM точно так же, как и другой веб-контент первого класса.

Встраивание видео

Следующий пример показывает, как добавить в HTML-документ видео и элементы управления воспроизведением.

```
<video src="video.foo" controls="controls"></video>
```

Атрибут `src` в этом примере выполняет ту же роль, что и одноименный атрибут элемента `img`: он содержит URL встраиваемого видеофайла. Назначение атрибута controls очевидно: он заставляет браузер отобразить элементы управления, с помощью которых пользователи могут запустить или приостановить воспроизведение, изменить громкость, перейти к произвольному месту видеофайла, а также открыть видео в новом окне или использовать полноэкранный режим. Одним словом, атрибут controls используется для того, чтобы добавить точно такие же элементы управления воспроизведением, как у обычного встраиваемого проигрывателя. Разница в том, что в случае элемента `video` браузер сам генерирует элементы управления и напрямую взаимодействует с видеоконтентом, не перепоручая этого плагину.

Поддержка альтернативных видеоформатов

Если вы достаточно наблюдательны, то могли заметить, что в предыдущем примере используется видеофайл parrot.foo с незнакомым расширением. И действительно, такого формата на самом деле не существует. Предполагается, что вместо .foo имя файла должно заканчиваться расширением, соответствующим такому видеоформату, который поддерживается всеми современными браузерами.

Проблема в том, что в настоящее время не существует ни одного такого формата: Firefox поддерживает файлы Ogg Vorbis, которые кодируются с помощью Theora, а Safari, похоже, в скором времени остановится на формате MPEG4, который кодируется с помощью H.264. Попытки производителей прийти к соглашению и выбрать стандартный видеоформат продолжаются, однако до окончательного разрешения этой проблемы еще довольно далеко.

Чтобы сделать видео доступным для большинства пользователей, потребуется кодировать файл в нескольких форматах; при этом вместо того, чтобы использовать атрибут `src` для элемента `video`, можно добавлять элементы `source` в качестве его содержимого (см. пример).

```
<video controls="controls">
<source src="parrot.ogg"/>
<source src="parrot.mov"/>
<source src="parrot.wmv"/>
<source src="parrot.3gp"/>
</video>
```

Браузер будет просматривать элементы source до тех пор, пока не найдет видеофайл в том формате, который он может воспроизвести.

Добавление видеоконтента для браузеров, не поддерживающих элемент video

Теперь рассмотрим еще один вопрос: как добавить видеоконтент для устаревших браузеров, в которых нет поддержки элемента video. Решить эту проблему можно, например, добавив элемент object, встраивающий видео с помощью плагина (см. пример).

```
<video controls="controls">
<source src="parrot.ogg"/>
<source src="parrot.mov"/>
<source src="parrot.wmv"/>
<source src="parrot.3gp"/>
<object data="parrot.swf" type="application/x-shockwave-flash">
    <param name="movie" value="parrot.swf"/>
</object>
</video>
```

Пользователи, в браузерах которых не установлен (или отключен) соответствующий плагин, не смогут просмотреть содержимое, добавленное с помощью элемента object.

Элемент canvas (HTML5)

Говоря об HTML5, нельзя оставить без внимания элемент canvas.

Элемент canvas — практически то же самое, что и элемент img; единственное отличие состоит в том, что он не статический, а динамический. Он задает особое место на странице с определенными размерами, где можно динамически (то есть программно) отображать картинки, анимацию и т. д. С его помощью можно, например, динамически генерировать схемы и графики, создавать браузерные приложения для рисования (или даже для редактирования текста) или браузерные игры — в общем, все то же, что можно сделать с помощью плагинов (таких, как Flash).

Интерфейс CanvasRenderingContext2D

Также элемент canvas можно рассматривать как часть двухкомпонентного «свойства Canvas», которое включает еще и интерфейс программирования

приложений CanvasRenderingContext2D, без которого невозможно использовать это свойство в полную силу. Сравните этот элемент с другими интерактивными элементами, появившимися в HTML5 (например, `video`), которые можно прекрасно использовать и без соответствующих API. Но для `canvas` это не так.

Детали, касающиеся использования элемента `canvas`, касаются главным образом программирования на JavaScript с помощью CanvasRenderingContext2D API, что выходит за рамки нашей книги. Этот элемент — скорее не свойство разметки, а хитрый прием, позволяющий решать задачи программирования. Если вы захотите научиться его использовать, то найдете множество примеров и описаний, в том числе главу в книге Painting the Web.

SVG как альтернатива canvas

Если вы не очень хорошо знакомы с JavaScript, вам вряд ли стоит сразу использовать `canvas` для добавления анимации и интерактивных изображений. Попробуйте сначала проверить, может ли SVG справиться с поставленной задачей. Если вы занимаетесь в основном написанием разметки или дизайном, вам, возможно, больше понравятся принципы декларативного программирования, использующиеся в SVG (и напоминающие принципы CSS), чем принципы императивного программирования, которые лежат в основе `canvas`.

12 Веб-типографика

Хотя некоторые аспекты CSS — особенно свойство `float` — могут быть снисходительно описаны как «трудные», существуют и другие стороны технологии, эффективность которых с лихвой компенсирует время, затраченное на их изучение. Свойства шрифта и текста принадлежат именно к этой группе.

В начале главы дается общее представление о принципах печатного дела на Западе, который имеет большое значение для понимания того, почему у неопытных заказчиков часто возникают далекие от реальности ожидания, связанные с возможностями Сети по управлению представлением.

Краткая история буквенных форм

В наши дни, когда грамотность доступна всем слоям общества, кроме, возможно, самых нуждающихся и незаинтересованных, возможность писать воспринимается как должное. На самом деле, за плечами у систем письменности — 5000 лет постоянной эволюции, и большая часть этих перемен осталась в памяти людей.

История письменности и печатного дела учит *выдержке* — художники и дизайнеры в течение веков могли оттачивать свое навыки по созданию напечатанной страницы. В этом и состоит самое существенное отличие от Интернета, в котором накладываются многочисленные ограничения на дизайн. Применение Adobe Flash для создания сетевого содержимого хоть и сокращает количество ограничений, взамен накладывает свои собственные, и ни одна технология не может компенсировать тот факт, что пользовательские среды сильно различаются между собой.

Учитывая важность влияния профессионально обученных графических дизайнеров на Сеть, полезно знать, как развивалась их продукция — и шрифтов это касается в наибольшей степени.

Происхождение современных западных буквенных форм

Большинство древнейших цивилизаций Земли развивали свою письменность независимо друг от друга, стремясь к ведению непрерывной записи событий — стремясь, в сущности, к расширению памяти.

В этом разделе опускается история азиатского вклада в развитие письменности — в связи с нехваткой высококачественных англоязычных источников информации по теме, слабой ориентированностью в теме автора книги и тем фактом, что книгопечатание в промышленных масштабах было по большей части европейским изобретением.

История западных алфавитов, используемых в наши дни, может быть прослежена до момента распространения египетских иероглифов на территорию Леванта около 3500 лет назад. Это привело к возникновению финикийского алфавита, на основе которого позже появился греческий. Греческий алфавит, в свою очередь, стал основой для латинского и кириллического алфавитов.

Современные печатные прописные буквы (верхний регистр или заглавные буквы) заимствуют свою форму у классических латинских букв, высеченных на пьедесталах римских статуй и фасадах зданий, а строчные буквы (нижний регистр) появились позже, развившись в Средние века из римского курсива.

Письменная литература, какой мы знаем ее сегодня, на Западе начала оформляться лишь в конце эпохи Средневековья, незадолго до изобретения Иоганном Гутенбергом печатного пресса. До этого момента практически всегда процесс письма производился от руки и чаще всего служил для составления и копирования документов.

Пресс Гутенберга и искусство типографики

Создание Гутенбергом первого механического пресса, использовавшего отлитые из металла сменные буквы, изменило экономику книгоиздательства и послужило рождением *типографики*, то есть разработки буквенных форм для определенных целей и конкретных методов копирования, как правило, в соответствии с тенденциями изменений художественного вкуса среди начитанной публики. Помимо многих других преимуществ, такой подход обеспечивал неслыханное доселе постоянство внешнего вида печатного материала — то постоянство, которое сейчас принимается как нечто само собой разумеющееся.

Разработка прессов, производство бумаги и переплетное дело развивались и улучшались на протяжении веков, однако в типографику инновации проникали медленнее: в последней четверти XIX века печатники набирали текст тем же способом, что и Гутенберг.

Новые идеи стали возникать чаще с появлением процесса набора на отливных наборных машинах: строчки набирались из свинцовых литер в специальной машине, а затем располагались на рабочей поверхности пресса. Термин «интерлиньяж» (leading), который до сих пор широко используется графическими дизайнерами и специалистами по допечатной подготовке, имеет прямое отношение к атрибуту CSS `line-height`. Этимология этого термина восходит к свинцовым полоскам (lead), которые вставлялись между свежими печатными отливками для создания расстояния между строчками — эту технологию использовали еще в начале эры книгопечатания.

Позже, в середине XX века зародился фотонабор, при котором для создания печатных пластин используются светочувствительные соединения — это было новаторство в процессе литографии, которой к тому времени уже несколько сотен лет пользовались печатники.

Появление цифрового типографского набора

Развитие реального компьютеризированного набора произошло в течение нескольких лет после изобретения лазерного принтера, и за этими двумя новшествами вскоре последовало появление доступных по цене компьютеров, совместимых с программами для разработки шрифтов.

Последние кусочки головоломки под названием «цифровой шрифт» встали на место с появлением доступных лазерных принтеров и пользовательских программ для обработки текста с графическими интерфейсами, за которыми через несколько лет последовали полнофункциональные комплексы издательских программ. За промежуток в 20 лет компьютеризированный набор развился от систем, обрабатывавших отдельные колонки для фотонабора — что более чем удваивало опыт и умения (редкие, впрочем) бывалых операторов отливных наборных машин — до готовых наборов программного обеспечения, работающих на вполне доступных компьютерах и способных создавать целые макеты страниц с графикой. К началу 1990-х годов как рабочие станции, так и принтеры были способны не только использовать шрифт на пленке, но и работать с графикой на пленке.

Сейчас при офсетной печати эти набранные цифровым образом макеты экспортируются прямо в машину, называемую *плейтсеттером*, которая создает смолистые печатные пластины из растризованных данных на входе. Реже макеты печатаются на лазерном принтере, фотографируются — отсюда и происходит термин «на пленке» — а затем вручную подаются в плейтсеттер в виде негативов.

Важное событие для нынешней индустрии цифрового набора произошло в 1991 году, когда формат шрифтов TrueType был лицензирован для использования в операционной системе Microsoft Windows. Позже помимо формата TrueType стал широко использоваться формат OpenType, однако различия между этими двумя форматами с точки зрения веб-дизайна минимальны.

Ограничений много, но ожидания не меняются

Веками набор текста и печатание были *осязаемыми* процессами, и в течение большей части этого периода ограничения по части разработок накладывались скорее *инструментальной частью*, а не *средой*. Перспективы развития браузеров перевернули эту парадигму с ног на голову: теперь почти все ограничения накладываются именно средой, а инструментальные средства в руках опытных людей способны сотворить вещи, ранее считавшиеся воистину невозможными.

Типографский глоссарий на практике

Рисунок 12.1 иллюстрирует несколько терминов, применимых к различным частям и особенностям шрифта, и дает представление о характерных чертах отдельных букв.

Рис. 12.1. Некоторые термины, касающиеся буквенных форм

Существует много других терминов, о которых необходимо знать дизайнерам сайтов.

- *Blackletter.*
  Стиль шрифта, вдохновленный немецкой каллиграфией конца Средневековья и по сей день четко ассоциируемый с Германией. Иногда его называют «готическим», но это название устарело (см. *Gothic (Готический)*).
- *Condensed (Уплотненный).*
  Стиль шрифта без засечек, отличающийся от других более плотным расположением букв. Противоположный термин — *extended (разреженный)*.
- *Copy (Копия).*
  Обобщенное название продукта работы писателя. Отличается от термина «текст» тем, что к «текстам» относятся только нечисловые данные в самом общем смысле; любая копия является текстом, но не любой текст — копией.
- *Diacritic (Диакритика).*
  Глиф, добавляемый к букве для обозначения интонации или произношения. Самые распространенные примеры — ударение (´), умляут (¨) и седиль (¸).

- *Dingbats (Символьные шрифты).*
  Набор (обычно встречающийся и в распространенных шрифтах) знаков, которые представляют собой простые рисунки (к примеру, ноты или символы схем) и не имеют ничего общего со стандартизированной орфографией.
- *Extended (Разреженный).*
  Дополнение к уплотненным шрифтам. Буквы шире, чем в обычных шрифтах, межбуквенные расстояния, как правило, тоже более широкие. Также называется «широким» и/или «экстраширокими».
- *Glyph (Глиф).*
  Самая маленькая единица шрифта, отдельный элемент, вносящий некую степень значимости. (Некоторые символы состоят из одного глифа, а другие сочетают в себе несколько.)
- *Gothic (Готический).*
  На протяжении прошлого столетия с большой вероятностью был синонимичен *шрифту без засечек (sans-serif)*; был назван так потому, что самые ранние шрифты без засечек зародились в Германии и немецко-говорящих регионах Швейцарии.
- *Gutter (Крупный пробельный материал).*
  Пустое пространство между границей текста и линейкой, между двумя колонками либо двумя отрывками текста. Во многих случаях управляется свойствами заполнения CSS.
- *Italic (Курсив).*
  Шрифт, развившийся из эквивалентного прямого шрифта с засечками при помощи каллиграфических деталей. Обычно отклонен от вертикали на 5–10° по часовой стрелке. (См. *Oblique (Наклонный).*) Получил свое название потому, что самые ранние варианты дизайна разрабатывались в Италии. Курсивные шрифты имеют более узкое начертание, чем прямые, — тем самым увеличивается количество слов, умещающихся в одной строчке.
- *Justification (Выравнивание).*
  Относится к границе (границам), которой следуют линии шрифта. У выровненной по левому краю колонки все строки начинаются от общей левой границы, у выровненной по правому — все строки заканчиваются у общей правой границы, а у полностью выровненного абзаца все строки, кроме последней, начинаются и заканчиваются у одних и тех же границ. Этот аспект верстки в CSS управляется свойством `text-align`. Противоположный термин — *ragging (дробление).*
- *Kerning (Кернинг).*
  Нетипичные пробелы в середине специфических пар глифов, особенно тех, что включают в себя A, f, j, J, L, T, V, W, w и y. Некоторые сочетания операционных систем, программного обеспечения и шрифтов требуют частого проведения вручную кернинга высококачественного растризованного шрифта.

- *Leading (Интерлиньяж).*
  Как было сказано выше — пространство между строками, названное так из-за свинцовых плашек (lead), которые разделяли свежие отливки и тем самым создавали это пространство. Близко (но не совсем) свойству CSS `line-height`.

- *Letterspacing (Межбуквенный интервал).*
  Область постоянной ширины, присутствующая между индивидуальными глифами в отдельных отрывках текста. Управляется свойством CSS `letter-spacing` (и в некоторой степени — свойством `word-spacing`).

- *Lower-/uppercase (Нижний/верхний регистр).*
  Соответствует строчным и прописным буквам. Такое название связано с тем, что печатники раньше хранили отдельные литеры конкретного шрифта в верхних или нижних ящиках.

- *Mono (Моно).*
  Термин, используемый для различения гарнитур с фиксированной шириной от гарнитур с переменной шириной.

- *Negative space (whitespace) (Свободное пространство).*
  Любая область пространства на странице или холсте, не занятая шрифтом, иллюстрациями или линейками. Использование словосочетания «свободное пространство» (whitespace) в качестве дизайнерского термина разъясняет его использование в качестве термина компьютерного, хотя он и является достаточно общим.

- *Oblique (Наклонный).*
  Шрифт без засечек, эквивалентный *курсиву (italic)*, но без каллиграфических особенностей.

- *Orthography (Орфография).*
  Теория и практика письменности; отдельная система письма.

- *Ragging (Дробление).*
  Удаление всех дополнительных пробелов между буквами и между словами из отрывка таким образом, чтобы текст не был выровнен по границе. По умолчанию, текст в Интернете, выровненный по левой границе, имеет дробление справа — и наоборот.

- *Roman (Прямой).*
  При обозначении гарнитуры этот термин, как правило, синонимичен *шрифту с засечками (serif).*

- *Rule (Линейка).*
  Линия, приложенная к одной стороне текстового блока. Обычно управляется при помощи свойств рамки CSS.

- *Sans-serif (Шрифт без засечек).*
  Класс шрифтов, различаемых благодаря их склонности к простым геометрическим формам, имеющим постоянную толщину линий и без дополнительных концевых элементов (засечек и пр.).

- *Script (Рукописный шрифт).*
  Класс гарнитур, разработанный ради имитации непрерывного почерка; как правило, используется в декоративных целях.
- *Serif (Шрифт с засечками).*
  Такие шрифты происходят от классических гравированных букв. Характеризуются различной толщиной линий и наличием *засечек* — легких кромок или выступов — на окончаниях.
- *Weight (Жирность).*
  Ширина линейки, штриха или шрифта. Применительно к тексту — управляется свойством CSS `font-weight`. Жирность обычных шрифтов для печатных приложений варьируются от толщины волоса (самый тонкий) до экстрачерного (самый жирный); типичный вес основного текста иногда называют «средним» или «книжным», но обычно у него нет специального названия. На рис. 12.2 и 12.3 показаны примеры использования этих компонентов шрифтов. На рис. 12.2 фигурируют стандартные шрифты Microsoft для Интернета, а на рис. 12.3 показаны установки по умолчанию Safari/Macintosh и IE7/Windows для различных символьных значений `font-family`.

Номенклатура кодирования символов рассматривается ниже в этой главе.

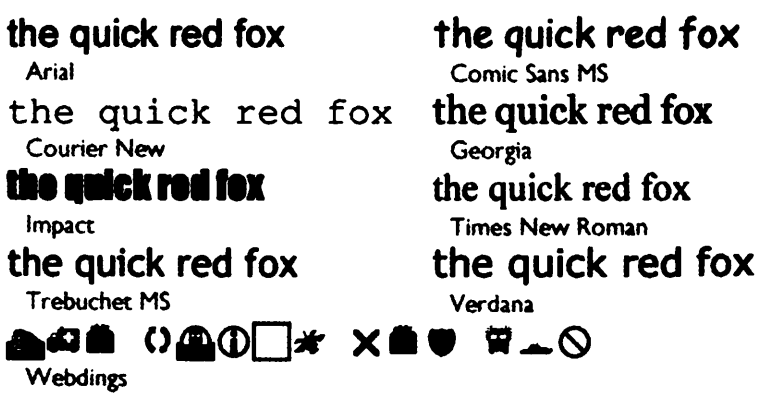


Рис. 12.2. Распространенные семейства стандартных шрифтов Microsoft

Алиасинг и сглаживание

Сначала хорошая новость: веками типографика являлась краеугольным камнем высоких эстетических стандартов.

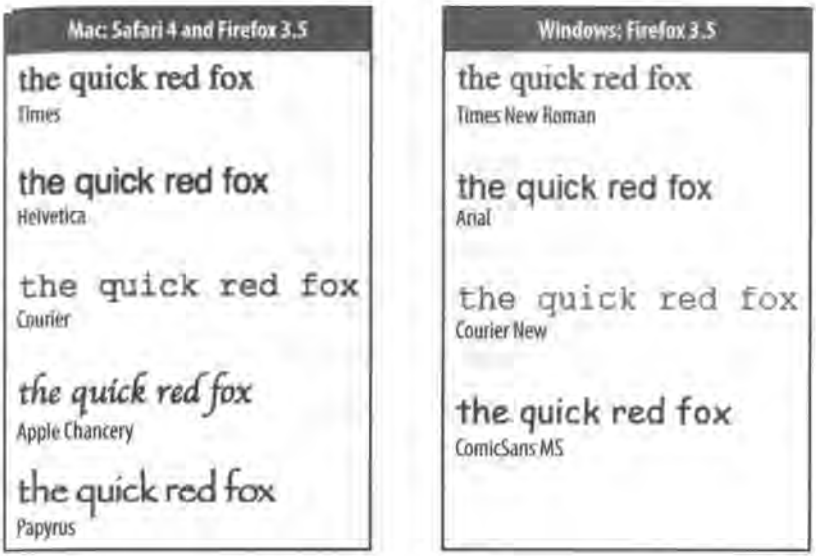


Рис. 12.3. Установки по умолчанию для шрифтов с засечками и без засечек, а также семейств моноширинных, курсивных и акцидентных шрифтов

Плохая же новость состоит в том, что в течение этих веков никому не приходило в голову, что когда-нибудь их работе *может* потребоваться пройти через пикселизацию, как это происходит на электронном дисплее.

Говоря проще, это означает, что большинство традиционных шрифтов на экране выглядят ужасно. Это вызвано *алиасингом* (эффектом «ступенчатости» изображения), который иллюстрирует рис. 12.4.

Форма буквы близка к верной, но близка не совсем, что приводит к ее внешнему виду «с большими пикселами». По этой причине шрифты, разработка которых велась без учета возможного появления искажений, обычно имеют внешний вид, отличный от печатных аналогов (и куда более невзрачный). Этот эффект усиливается тем, что человеческое зрение особенно остро реагирует на контрастные изображения.

Сглаживание направлено на уменьшение ущерба при отображении углов символов, как показано на рис. 12.4. Сглаживающие алгоритмы создают по краям букв области, в которых разница между фоном и передним планом меняется *постепенно*, что уменьшает контраст и позволяет применять прием, называемый хинтингом.

Хинтинг предполагает небольшое изменение контура шрифта. В печатной типографике наиболее явный хинтинг применяется к развилкам букв вроде «k» и «v», которые в обычном случае размываются из-за утечек чернил при печати. Если проводить сравнение, то к шрифту, разработанному для отображения на экране, хинтинг применяется для вычисления эффектов неровностей и сглаживания.

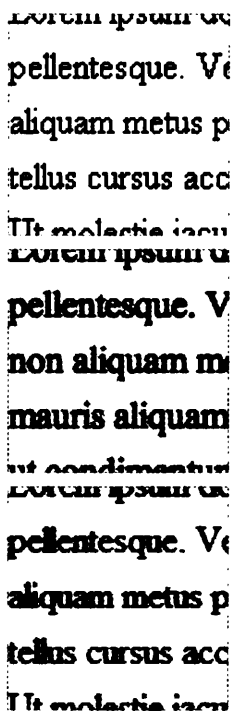


Рис. 12.4. Иллюстрация искажений и сглаживания на примере шрифта Helvetica размером 16/12 пикселов

Самый большой недостаток сглаживания состоит в том, что оно делает мелкий шрифт практически нечитаемым. Как и небольшая копия изображения при существенном размытии, мелкий шрифт с невысоким разрешением будет выглядеть просто набором контрастных пятен, что показано на рис. 12.5.

Один из недостатков шрифтов, разработанных для отображения на экранах, — в том, что они, как правило, более привлекательно выглядят при маленьких размерах, чем при больших. Причины этого вполне логичны: так как большинство документов содержат текст размером около 12 пунктов (16 пикселов), то имеет смысл, чтобы шрифт была читаемым при таких размерах. По этим причинам хинтинг, внутренние закругления и различные начертания в случае оптимизированных под экран шрифтов осуществлять проще.

С другой стороны, оптимизация для дисплея при небольших размерах может привести к тому, что оптимизированные шрифты будут казаться чрезмерно простыми при больших размерах, если сравнивать их с традиционными шрифтами. В этом и кроются причины того, почему дизайнеры часто предпочитают использовать растровые заголовки, набранные при помощи традиционных шрифтов: при больших размерах сохраняется (и даже расширяется) представление деталей гарнитур, предназначенных для печати, что делает их более привлекательными, чем шрифты, оптимизированные под дисплей.

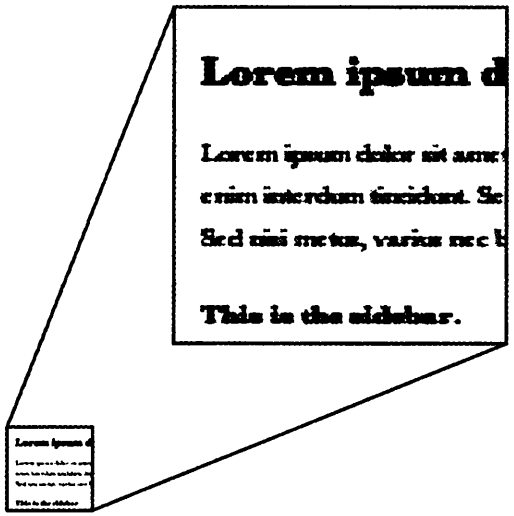


Рис. 12.5. Результат уменьшения размера шрифта на электронных дисплеях; если к шрифту такого маленького размера применить сглаживание, то в результате текст невозможно будет прочесть

Стили шрифтов, читаемость и разборчивость

В издательском дизайне существует два понятия, которые влияют на выбор шрифта: читаемость и разборчивость. *Читаемость* — это качество копии, которое обеспечивает легкость ее чтения в больших объемах и на протяжении длительного времени; *разборчивость* относится к простоте обнаружения данных, слов и коротких фраз при чтении отрывка.

Обеспечение читаемости

Охарактеризовать читаемость могут наши ожидания от книжного дизайна:

- шрифты с засечками;
- 12–15 слов на строку[1];
- выровненные по ширине строки;
- небольшие и постоянные значения межбуквенных интервалов.

При высококачественной печати часто применяются увеличенный габарит (20% или больше от размера основного текста) и детализированные шрифты для более легкой работы с полями, строками и буквами. Причиной этого

[1] Для англоязычного текста.

является тот факт, что прибыль от продажи низкокачественных изданий варьируются от плохой до откровенно отвратительной, отчего возникает потребность в минимизации затрат на производство и распространение.

В свою очередь, затраты на бумагу уменьшаются за счет использования низкосортной бумаги и уменьшения ее количества. Это и является причиной того, что на странице умещается больше строк, уменьшаются поля и — из-за качества бумаги — используются менее детализированные шрифты, которые меньше страдают от утечки краски при печати на низкосортной бумаге.

Затраты на бумагу вряд ли беспокоят веб-дизайнеров, так что мы можем использовать целый арсенал различных инструментов для повышения читаемости. Для экранного отображения результат может быть, например, таким:

```
#bodycopy
    p {
    width: 50em;
    font-family: Georgia,"Times New Roman",serif;
    font-size: 14px;
    line-height: 17px;
    text-align: justify;
}
```

Однако экранная среда порождает препятствие, отсутствующее в книгах: потребность в прокрутке содержимого означает, что строки текста на самом деле двигаются вверх по странице. По этой причине на веб-сайтах обычно используется выравнивание по левой стороне; при раздробленной правой границе проще находить конкретные строчки, что важно для навигации по тексту.

Обеспечение разборчивости

В то время как читаемость актуальна для длинных текстов, заголовки, краткие отрывки и данные требуют разборчивости. Газетный дизайн предоставляет отличную возможность учиться разборчивости на примерах и подчиняется некоторым (или всем) следующим принципам:

- шрифты без засечек (рубленые);
- для заголовков шрифт крупнее, для текста статей — меньше;
- четкое следование сетке;
- расширенное применение правил для полей текста/данных по одной или двум осям;
- фоновое выделение по строкам (или по колонкам, что реже);
- выравнивание по краям — соответствующие колонки содержимого выравнивается по общей границе.

В главах 10 и 13 содержатся несколько подробных рекомендаций по оформлению «разборчивого» контента для таблиц и форм.

При работе с системами письма «слева-направо» (в европейских языках) наиболее разборчивый текст обычно выровнен по левому краю и имеет дробление справа, кроме тех случаев, когда есть большое различие между правыми границами строк в конкретном блоке текста. В подобных случаях читаемость можно повысить при помощи разбития строк атрибутом `white-space` (более подробное описание дается ниже), выравнивания по правому краю или применения обоих методов одновременно.

Шапка и мелкий шрифт

Представьте себе сложенную газету на торговой витрине или на стойке для газет — в таком положении, при котором покупатель видит верхнюю часть первой полосы. Гарантированная видимость заголовков и содержимого этой части газеты ведет к особому вниманию к верстке макета, и заметки для области «над сгибом» выбирают с особой тщательностью.

Аналогичное пространство веб-сайта — это видимая часть домашней страницы или страницы назначения (рис. 12.6), которую многие профессионалы называют также «областью над сгибом» («шапкой»).

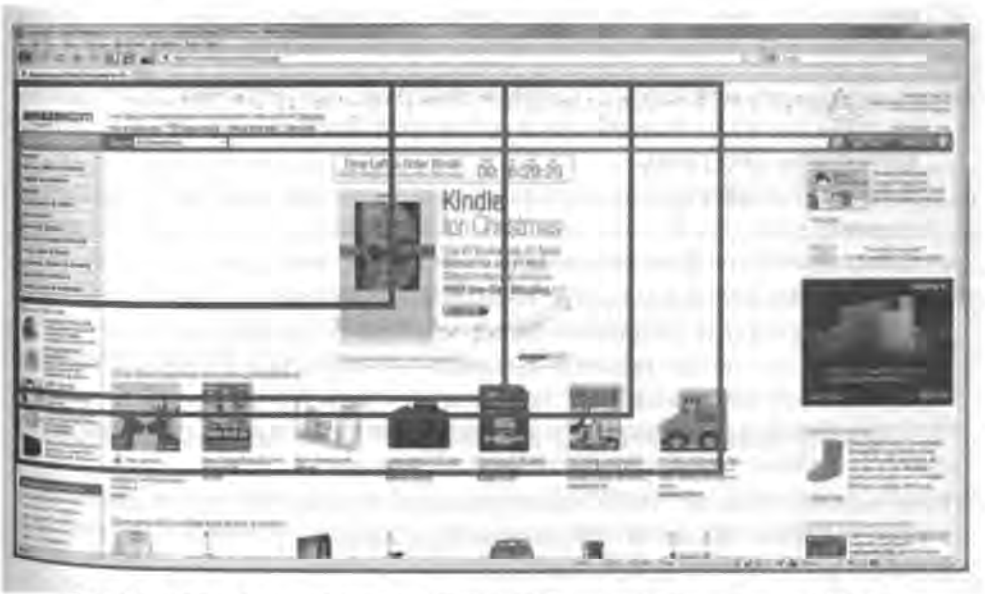


Рис. 12.6. Окно браузера размером 1920×1080 с рамками, обозначающими область «над сгибом» при различных обстоятельствах геометрии браузера

Многие считают пространство в «шапке» самым ценным на веб-сайте, и очень легко поддаться искушению и заполнить его информацией — ведь чем больше содержимого там находится, тем больше соблазнов можно предложить посетителю. Эта мысль ошибочна по четырем причинам.

- *Беспорядок всегда отпугивает посетителей, ищущих конкретную информацию.*

  Парадигма области «над сгибом» предполагает, что обычный посетитель скорее покинет страницу, чем начнет прокручивать ее до нужного места. Этот аспект часто доводят до абсурда — к примеру, стараясь обеспечить полное отсутствие необходимости использования вертикальных полос прокрутки. Как бы там ни было, для прокрутки нужно усилие, но попытки найти метафорическую иглу в стоге сена, который представляет собой перегруженная информацией домашняя страница, могут потребовать больших усилий, чем прокрутка.

- *Беспорядок затрудняет различение содержимого.*

  Наличие в макете многочисленных элементов делает цвет единственным средством для различения содержимого. Основополагающий момент парадигмы области «над сгибом» точно так же может привести к использованию нескольких одинаково насыщенных цветов, что в итоге приведет к невероятно уродливому внешнему виду сайта.

- *Использование мелкого шрифта для того, чтобы впихнуть содержимое в область «над сгибом», приводит к появлению страниц, непригодных для людей с неидеальным зрением.*

  Вам даже не придется выходить в Интернет для проверки этого утверждения: просто прочтите страницу контракта, набранную мелким шрифтом, и задайте себе вопрос, будет ли такое решение уместным на веб-сайте.

- *Разнообразие разрешений дисплея означает, что обширное содержимое «шапки» для одних пользователей будет представлять собой всего лишь тонкую полоску для других.*

  В случае с посетителями, использующими дисплеи высокого разрешения, попытки выделить область «шапки» при более низких разрешениях полностью теряются на фоне окружающего пространства страницы.

Анализ противоречий парадигмы области «шапки» приводит к выводу о том, что меньший шрифт основного текста никогда не должен быть частью дизайна, кроме как по тщательно обдуманным эстетическим причинам, которые придутся по душе большинству посетителей сайта. Говоря прямо, крохотные буковки не выглядят выигрышно и не приносят пользы, если не принимать в расчет некоторые очень немногочисленные ситуации.

Конечно, это не значит, что не надо определять приоритеты содержимого, имея в виду, что самое важное всегда должно быть наверху. Вместо этого стоит попробовать расположить содержимое «слоями» на сетке.

Размер шрифта

Если и существует какая-то причина, способная взволновать опытных дизайнеров, то это управление шрифтом. CSS предоставляет им такую возможность: если шрифт может быть обработан браузером пользователя, то дизайнер может

управлять его характеристиками. Это особенно важно в отношении размера шрифта, который должен подчиняться некоторым основным правилам.

- У опытных дизайнеров, застрявших в рамках печатного мышления, иногда не получается осознать степень контроля, который большинство пользователей имеют над размерами шрифтов. По этой причине лучше указывать размер базового шрифта в пикселах, например:

```
body { font-size: 14px; ... }
```

  а затем управлять размером шрифта в каскаде при помощи `em` или процентного выражения (что идентично с функциональной точки зрения). Дизайнерам, работающим над сайтами, нацеленными на большую аудиторию пользователей IE 6, возможно, придется проигнорировать эту рекомендацию из-за соображений удобства пользования, так как IE 6 совершенно не умеет увеличивать масштаб текста, выраженный в значениях `px`.

- Размеры заголовков устанавливаются браузером пропорционально размеру базового шрифта. Как правило, при установке исходных данных заголовка лучше оставить эту зависимость без изменений, поскольку это позволит заголовкам лучше отражать изменения макета. Если вы используете растровые заголовки, то будет полезно добавить в правила отображения изображений значение `background-position`.
- Если вам необходим *полный* контроль над размерами элемента, сделайте его содержимое изображением и введите нужный текст в значении `alt` этого изображения. Таких ситуаций лучше избегать, однако в реальном мире с ними иногда приходится сталкиваться.
- Избегайте выражения значений `line-height` в пикселах, если вы не собираетесь назначить в пару каждому `font-size` свойство `line-height`. Значения размера заголовка присваиваются в каскаде автоматически, как и значения `line-height`. Это значит, что если вы задаете значение `line-height` в пикселах ради лучшего отображения основного текста, то этот же статичный габарит может быть применен к более крупному шрифту и в результате может получиться наплыв строк друг на друга.

Выбор правильных единиц размера шрифта

При оформлении шрифта для отображения в браузере вы можете выбрать одну из четырех единиц размера: пикселы, `em`, процентные соотношения или ключевые слова.

- Пикселы (`px`) — *абсолютные* единицы (до некоторых пор); в обычных обстоятельствах текст, размер которого задан в пикселах, будет выглядеть одинаково во всех средах, независимо от пользовательского размера текста по умолчанию и значений, установленных выше в каскаде. В использовании пикселов есть две ловушки: первая заключается в ограничениях Internet Explorer 6, а вторая — это риск разрывов в тех случаях, когда размер текста увеличивается пользователем за границами лежащих ниже фоновых изображений или боксов элементов.

- Единицы em и проценты (%) — *относительные единицы*; установленное значение будет умножено на то значение, которое имеется выше в каскаде, даже если оно является значением браузера по умолчанию. Принцип действия относительных величин разъясняется в следующем разделе специально для тех, у кого проблемы с математикой.
- Ключевые слова могут принимать одно из семи значений и относятся к размерам заголовков по умолчанию. Важно запомнить, что ключевые слова всегда задают шрифт относительно установок браузера по умолчанию (находящихся в панели настроек), причем значение `medium` равно явному значению браузера по умолчанию (обычно 16 пикселов).

Пересчет размеров для em и процентов

Если значение `font-size`, выраженное в `em` или процентах, находится в пределах каскада, то реальное значение *кратно* наследованному размеру. Происходит это следующим образом:

```
body { font-size: 15px; }
.lede { font-size: 1.4em; }
.note { font-size: .667em; }
```

В отсутствие других значений `font-size`, действующий размер шрифта в `.lede` будет 21 пиксел, а в `.note` — 10 пикселов.

Ради наглядности допустим, что существует другое правило:

```
.lede em { font-size: 1.429em; font-style: normal; }
```

Предполагаемый эффект — выделить отрывки *увеличением*, но не *курсивом*; такое решение может быть применено, если цель дизайна — соответствовать какому-либо капризу или передать дерзость. Оставив в стороне эстетику и психологию, нужно отметить, что два значения `font-size` *умножаются*, а не *прибавляются*:

$(15 \times 1{,}4 \times 1{,}429) \approx (15 \times 2) \approx 30$

Подобный эффект суперпозиции работает и в отношении уменьшающихся величин. По этой причине, когда я участвую в дискуссиях на форуме с новичками в области CSS, я намеренно (и иногда спорно) отговариваю людей от использования единиц `em` и `%` для значений `font-size` элемента `body`, поскольку необдуманное добавление уменьшающихся значений `font-size` в нисходящую таблицу стилей может быстро привести к неразборчивости текста.

Ключевые слова для размеров шрифтов

Согласно спецификации CSS 2.1, к размерам заголовков относятся семь значений `font-size`, описанных в табл. 12.1.

Поддержка значений ключевых слов `font-size` вызывает атрибут `size` унаследованного элемента `font`, и на практике они оба имеют схожие значения.

Таблица 12.1. Зависимость между ключевыми словами и размерами заголовков

Ключевое слово	xx-small	x-small	small	medium (по умолчанию)	large	x-large	xx-large
Заголовок	h6	—	h5	h4	h3	h2	h1

Учитывая настройки размера текста по умолчанию в Firefox 3, Internet Explorer 8 и Safari 3/4, можно сделать вывод о размерах текста, установленных при помощи ключевых слов, и заключить их в следующую таблицу (табл. 12.2).

Таблица 12.2. Размеры текста по умолчанию (в пикселах), заданные при помощи ключевых слов

xx-small	x-small	Small	medium	large	x-large	xx-large
9	10	13	16	18	24	32

Работа со шрифтами

Возможность устанавливать шрифты за пределами разметки — одна из самых сильных сторон CSS. По этой причине представление типографики в Интернете теперь сильно отличается от того состояния, в котором оно пребывало в период зарождения, когда все страницы были набраны единой гарнитурой, выбираемой посетителем.

Проблема небольшого выбора

Все 16 гарнитур, доступных в Windows XP, подходят и для использования в Интернете — однако три из них функционально бесполезны (каждая по своим собственным причинам). OS X предлагает более широкий выбор, но, учитывая ее долю на рынке, лучший вариант для дизайнера — это выбрать одну из гарнитур XP в качестве запасного варианта для шрифта OS X, как показано в табл. 12.3.

Вездесущность Microsoft Office (и поставляемых с ним шрифтов) — это яркое пятно на невзрачном ландшафте типографики, однако большинство из этих шрифтов предназначены для печатного, а не для сетевого использования.

Таблица 12.3. Латинские гарнитуры[1], доступные пользователям Сети в зависимости от операционной системы. Широко распространенные шрифты выделены; запасные варианты[2] названы по именам

Гарнитура	Операционная система			
	Windows XP	Windows Vista и 7	Mac OS X 10.3	Mac OS X 10.5
American Typewriter	Courier New	Courier New	✓	✓
Andale Mono[3]	✓	✓	Monaco	✓
Apple Gothic	Microsoft Sans Serif	Microsoft Sans Serif	✓	✓

продолжение

Таблица 12.3 *(продолжение)*

Гарнитура	Операционная система			
	Windows XP	Windows Vista и 7	Mac OS X 10.3	Mac OS X 10.5
Arial[3]	✓	✓	✓	✓
Arial Black[4]	✓	✓	✓	✓
Arial Narrow[4]	✓	✓	✓	✓
Arial Rounded	Arial	Arial	✓	✓
Arial Unicode	Arial Unicode MS	Arial Unicode MS	Arial	×
Arial Unicode MS	✓	✓	Arial Unicode	✓
Baskerville	Palatino Linotype	Palatino Linotype	✓	✓
Big Caslon	Bookman Old Style	×	✓	✓
Book Antiqua4	✓	✓	✓	✓
Bookman Old Style[4]	✓	✓	✓	✓
Brush Script	cursive	cursive	✓	✓
Calibri	Trebuchet MS	✓	Trebuchet MS	Trebuchet MS
Century Gothic[4]	✓	✓	✓	✓
Cambria	serif	✓	serif	Plantagenet Cherokee
Cambria Math	×	✓	×	×
Candara	Tahoma	✓	Tahoma	Tahoma
Chalkboard			✓	✓
Cochin	serif	serif	✓	✓
Comic Sans MS [3]	✓	✓	✓	✓
Consolas	Comic Sans MS	✓	Comic Sans MS	×
Constantia	Book Antiqua	✓	Book Antiqua	×
Corbel	Tahoma	✓	Tahoma	×
Courier New [3]	✓	✓	✓	✓
Didot	serif	serif	✓	✓
Franklin Gothic	sans-serif	✓	Gill Sans	Gill Sans
Futura	Century Gothic	Century Gothic	✓	✓
Garamond[4]	✓	✓	✓	✓
Georgia[3]	✓	✓	✓	✓
Gill Sans[5]	sans-serif	Franklin Gothic	✓	✓
Helvetica	Arial	Arial	✓	✓
Helvetica Neue	Arial	Arial	✓	✓
Herculanum	fantasy	fantasy	✓	✓
Hoefler Text	Georgia	Georgia	✓	✓
Impact[3]	✓	×	✓	✓
Lucida Console	✓	✓	Monaco	Monaco
Lucida Grande	Lucida Sans Unicode	Lucida Sans Unicode	✓	✓
Lucida Handwriting[4]	✓	✓	✓	✓

Гарнитура	Операционная система			
	Windows XP	Windows Vista и 7	Mac OS X 10.3	Mac OS X 10.5
Lucida Sans Unicode	✓	✓	Lucida Grande	Lucida Grande
Marker Felt	cursive	cursive	✓	✓
Microsoft Sans Serif	✓	✓	Apple Gothic	✓
Mistral	cursive	✓	Brush Script	Brush Script
Monaco	Lucida Console	Lucida Console	✓	✓
Monotype Corsiva[4]	✓	✓	✓	✓
Nyala	fantasy	✓	Papyrus	Papyrus
Optima	sans-serif	sans-seris	✓	✓
Palatino Linotype	✓	✓	Baskerville	Baskerville
Papyrus[4]	✓	✓	✓	✓
Plantagenet Cherokee[6]	serif	✓	serif	✓
Segoe Print	Lucida Handwriting	✓	Lucida Handwriting	Lucida Handwriting
Segoe Script	cursive	✓	cursive	cursive
Segoe UI	sans-serif	✓	Gill Sans	Gill Sans
Skia	sans-serif	sans-serif	✓	✓
Sylfraen	✓	✓	Baskerville	Baskerville
Tahoma	✓	✓	✓	✓
Times	Times New Roman	Times New Roman	✓	✓
Times New Roman[3]	✓	✓	✓	✓
Trebuchet MS[3]	✓	✓	✓	✓
Verdana[3]	✓	✓	✓	✓
Zapfino	cursive	cursive	×	✓
Dingbats				
Apple Symbols			✓	✓
Marlett[7]	✓	✓	×	×
Symbol	✓	✓	✓	✓
Webdings[3]	✓	✓	✓	✓
Wingdings	✓	✓	×	×
Zapf Dingbats	×	×	✓	✓

[1] В составе у всех упомянутых операционных систем есть несколько кириллических и азиатских шрифтов с ограниченной поддержкой латиницы. К тому же не все перечисленные гарнитуры поддерживаются полной коллекцией шрифтов.

[2] Предлагаемые на замену гарнитуры выбраны по чисто субъективным критериям. Некоторые, хоть и не все, замены работают во всех операционных системах.

[3] В состав сборника стандартных шрифтов Microsoft для использования в Интернете входят: Arial, Arial Black, Comic Sans MS, Courier New, Georgia, Impact, Tahoma, Times, New Roman, Trebuchet MS и Verdana.

[4] Некоторые из перечисленных в таблице гарнитур предоставляются не Windows, а Microsoft Office, который установлен в качестве пробной версии на большинстве систем от изготовителей комплектного оборудования, включая Mac. Даже если пробный период подходит к концу или если вы стерли Office, связанные шрифты все равно остаются в системе — в случае Mac они остаются, если пробная версия программы была запущена хотя бы однажды. Использовать эти шрифты нужно аккуратно, так как большинство из них лучше подходит для печатного применения.

[5] Gill Sans — это гарнитура, используемая для текстовых меток пользовательского интерфейса OS X.

[6] Включает в себя латинские и символьные (к примеру, знаки валют) глифы, а также элементы азбуки Чероки.

[7] Включает в себя множество глифов, используемых для отметок элементов управления интерфейса Windows.

Представительные образцы гарнитур, перечисленных в табл. 12.3, можно найти в ряде источников, включая веб-сайт, дополняющий данную книгу (http://www.htmlcssgoodparts.net).

Использование шрифтов: свойство font-family

Если вы пристально всмотритесь в документ CSS, то свойство `font-family` может показаться вам довольно-таки примитивным: свойство, двоеточие, гарнитура, запятая, гарнитура, запятая, гарнитура, запятая. Похоже, что с таким справится даже одаренный шимпанзе с навыками печатания.

Но на самом деле у `font-family` есть свои правила:

1. Все установленные гарнитуры должны ссылаться на точные названия семейств, найденные в библиотеках шрифтов клиентских узлов с учетом заглавных букв, версий начертания обозначений кодирования, если они присутствуют. К примеру, Arial и Arial Unicode MS — это два совершенно разных семейства, что относится и к лицу, выдававшему лицензию, и к схеме кодирования. Имейте в виду, что из-за этого вам может понадобиться указывать различные варианты одной и той же гарнитуры.
2. Если название семейства содержит пробелы, то оно должно быть заключено в одинарные или двойные кавычки.
3. Полный список названий семейств, разделенных запятыми, должен быть отсортирован в порядке убывания предпочтительности, даже если фигурирующий в списке шрифт может быть недоступен. Следовательно, если вы хотите, чтобы посетители видели текст, напечатанный шрифтом Futura, в соответствующем правиле таблицы стилей вы должны прописать: `font-family: Futura, "Century Gothic"`.
4. Все значения `font-family` должны заканчиваться желаемым обобщенным именем. В список действующих обобщенных имен семейств (см. табл. 12.3) входят следующие имена.

- *serif*

  Римские гарнитуры — обозначаются как «Latin», «Old Style», «Antiqua» и «Copperplate».

- *sans-serif*

  Готические гарнитуры — обозначаются как «Geometric» и «Grotesk».

- *monospace*

  Гарнитуры с фиксированной шириной, такие, как Courier New; иногда обозначаются как «Mono» или «Typewriter». Шрифты, чьи названия оканчиваются на «10», «12» или «15», обычно являются шрифтами с фиксированной шириной; эти числа относятся к расстоянию между символами на дюйм при размере 12 пунктов.

- *cursive*

  Каллиграфические, или непрерывные, гарнитуры, в названиях которых часто появляются «Calligraphic» или «Cursive». Не путайте их с курсивными шрифтами (italic), которые могут быть и непрерывными, но обычно таковыми не являются.

- *fantasy*

  Декоративные гарнитуры, не относящиеся к каллиграфическим/рукописным.

5. Более новые версии браузеров в случае столкновения с действительными значениями заголовков HTTP `Content-Type`, `Content-Language` и `charset` смогут отобразить верно реализованные шрифты, как и положено. А с более старыми версиями не все так просто — разработчики, вынужденные поддерживать такие версии, должны указывать шрифты, кодированные исключительно для объявленного `charset`, если таковой имеется (см. раздел «Что такое кодировка?» на с. 256). Для страниц, написанных на латинице, самый безопасный подход в таком случае — намеренно распознать и выдать содержимое, которое закодировано в соответствии с подходящей кодовой страницей ISO 8859-x.

Хотя Internet Explorer 8 во многих аспектах демонстрирует выдающуюся работу, иногда он выдает непредсказуемые — и зачастую неприемлемые — результаты при необходимости опираться на обобщенное значение `font-family`. По этой причине стоит удостовериться, что в каждом списке названий шрифтов, использованных в значениях `font-family` или `font`, обобщенному названию предшествует хотя бы один шрифт, нормально обрабатываемый системами Windows. За более подробной информацией обратитесь на веб-сайт, дополняющий данную книгу (http://www.htmlcssgoodparts.net).

Поиск канонических названий гарнитур

Необходимое условие правила 1: значения `font-family` должны ссылаться на точное название гарнитуры, используемой клиентским узлом. Для того чтобы найти это название в системе Windows, проделайте следующие шаги:

1. Откройте Панель управления (Control Panel), которая расположена в меню Пуск (Start) под заголовком Настройка (Settings) или Панель управления (Control Panel). Либо введите в адресной строке Проводника (Windows Explorer) `C:\WINDOWS`.
2. И в Панели управления (Control Panel), и в папке `\\%WINDOWS` есть объект под названием Fonts (только первым способом найти его легче). Откройте его и выберите один из файлов шрифтов, который вы хотите использовать.
3. Откройте нужный шрифт. Первая строчка в свойствах файла — это имя родительского семейства.
4. Если вы используете Mac, то все еще проще. Откройте Font Book в папке Applications, в левой панели выберите All Fonts и найдите нужную гарнитуру в средней панели. Найденное там имя и должно упоминаться в правилах таблицы стилей.

Рисунок 12.7 показывает состояние пользовательского интерфейса в финале процедуры.

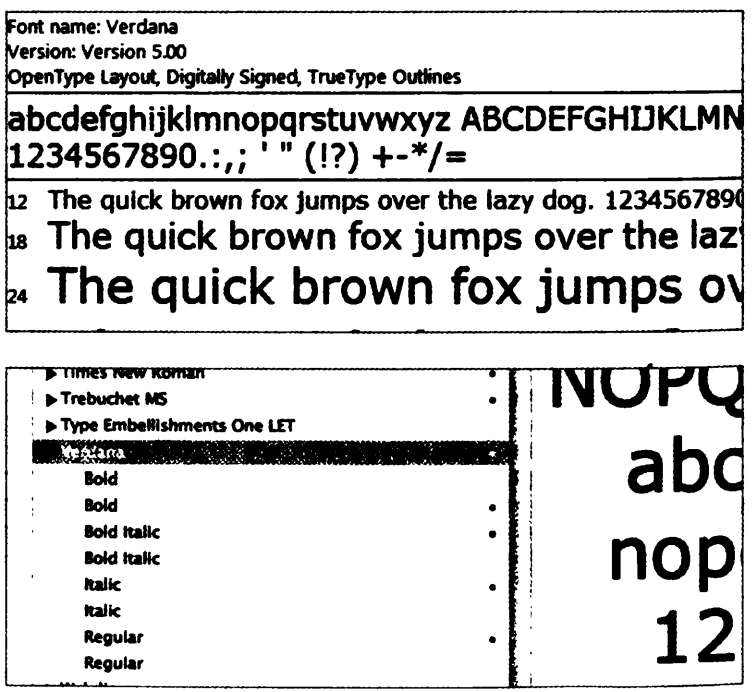


Рис. 12.7. Каноническое название шрифта

Доступ к шрифту системы по умолчанию при помощи свойства font

Я, как правило, не рекомендую применять свойство font: оно заставляет дизайнера использовать значения гарнитур (в других случаях ненужные), которые придется отменять другими правилами, что существенно повышает сложность таблицы стилей. Наиболее важный случай исключения — когда вам необходимо сослаться на шрифты системы по умолчанию, что можно сделать только при помощи свойства font.

Структура значения font выглядит следующим образом:

```
font-style font-variant font-weight font-size/line-height
[font-family|Имя шрифта для пользовательского интерфейса]
```

Фрагменты значений font должны быть расположены в вышеуказанном порядке. В тех случаях, когда явное значение line-height не нужно, косая черта, отделяющая его от значения font-size, может быть опущена.

Фрагмент font-family должен быть описан так же, как если бы это был отдельный случай применения font-family или заменен ключевым словом, относящимся к шрифту объекта в пределах пользовательского интерфейса клиентского узла. Эти ключевые слова описаны в табл. 12.4 согласно соответствующим элементам управления форм и конфигурации окна браузера.

Таблица 12.4. Значения фрагмента ключевых слов шрифта для пользовательского интерфейса и соответствующие элементы управления браузером, использующие их по умолчанию

Ключевое слово	Соответствующий элемент управления формы или шрифт метки окна браузера
caption	input type="submit"
icon	Метки кнопок панели инструментов; текст в боковой панели
menu	option
message-box	Аргументы window.alert()
small-caption	Содержание конфигурации окна; обычно — символьный шрифт
status-bar	Строка текущего состояния

До сих пор до конца не ясно, будут ли в CSS3 поддерживаться ключевые слова.

Кратко о кодировке

Последнее правило касательно значений font-family может создать некие затруднения, поскольку дизайнеры не всегда могут контролировать заголовки ответа HTTP.

Каждый верно сконфигурированный сервер, использующий адекватную реализацию HTTP (то есть практически любую), определяет язык и набор символов

для каждого документа, который он посылает клиентским узлам. Дополнительные интерфейсы, вроде элемента `meta` либо функции PHP `Header()`, позволяют разработчикам при необходимости менять или обходить эти установки.

Что такое кодировка?

Надеюсь, вам знакомы понятия битов и байтов; биты представляют состояние отдельной ячейки в ОЗУ системы, а байт равен восьми битам в логической последовательности, которая может быть составлена 256 разными способами.

Технические специалисты англоговорящего мира привыкли к тому, что один латинский символ — или, на типографском жаргоне, глиф — можно представить при помощи одного байта. Подобное правило распространялось и для других алфавитов.

Рассмотрим пример азбуки Морзе: его представление символов состоит из последовательностей точек (аналог нулевых битов) и тире (аналог единичных битов) разной длины. В этом случае назначение символу уникальной последовательности точек и тире зависит от средней частоты его появления в телеграфных сообщениях.

Что касается компьютерной области, то здесь применяются систематические правила. Термин «систематический» вызывает множество вопросов.

- Должны ли буквы верхнего регистра быть помещены перед буквами нижнего регистра или после них?
- Сигналы управления должны идти в начале или в конце последовательности символов?
- Определяет ли кодирующая схема следующее: первый бит кода является нулем/единицей или значением из диапазона 0–255?
- Какие ограничения системы могут повлиять на финальную схему кодировки?
- Каким кодовым позициям лучше не присваивать значения ради будущих изменений, например ради введения новых символов валюты?
- Какая логика присваивания должна применяться в случае с иероглифическими системами письменности вроде ханзи и кандзи, а также в случае сложных слоговых азбук вроде Хангул?
- Будет ли конкретная кодировка удобна для применения всеми пользователями данного правописания?

Избыток различных кодировок, применяемых в наши дни, объясняется тем, что разные инженеры находили на эти вопросы разные ответы. В Сети многие из этих кодировок должны поддерживаться обязательно.

ASCII, ISO 8859-1, Unicode и UTF-8

В середине 1960-х годов несколько команд разработчиков объединили усилия для создания базовой 7-битной (128-позиционной) кодирующей схемы

постоянной ширины для латинских символов, использующихся в американском английском языке, и назвали ее ASCII (American Standard Code for Information Interchange — Американский стандартный код для обмена информацией). Основой для ASCII стали ранние схемы кодирования для телетайпов. Несколько лет спустя вышло постановление о том, что все компьютеры, хранилища и конфигурации оборудования для передачи информации, заказываемые правительством США, должны поддерживать ASCII — и очень скоро ASCII стала использоваться во всем англоговорящем мире.

В 1980-х годах Международная организация по стандартизации (International Standards Organization, ISO) опубликовала стандарт для кодирования нескольких европейских и ближневосточных алфавитов, многие из которых базировались на латинском алфавите. Все эти схемы кодирования — которые повсеместно используются и сегодня в виде кодовых страниц ISO 8859 — были наполовину составлены из ASCII.

Постепенно этот процесс дошел до Востока, и была проделана работа по приспособлению стандартных схем кодирования для систем письменности и в других странах, особенно в Японии. В начале 1990-х результаты работы вместе с поправками были отражены в стандарте Юникод, который с тех пор непрерывно расширялся и стремился к представлению всех известных систем письменности, включая и мертвые алфавиты, используемые в исторических записях.

Таблицы кода Юникод на данный момент содержат более 100 000 символов. В Сети символы каждого документа обычно кодируются при помощи схемы UTF-8 (8-bit Unicode Transformation Format — 8-битный формат преобразования Юникод). Это схема с меняющейся шириной, которая кодирует все символы ASCII в один байт (из максимум четырех), тем самым обеспечивая обратную совместимость со всеми системами, опирающимися на ASCII, кроме самых ранних.

Выбор кодировки

По умолчанию программное обеспечение сервера работает с документами, закодированными в UTF-8. Если содержимое написано на английском языке и для объявления символов не из набора ASCII вы используете сущности HTML (которые рассматриваются ниже в этой главе), то нет необходимости настраивать сервера или добавлять свой заголовок HTTP к серверным сценариям только для того, чтобы объявить правильный набор символов. Это верно по двум причинам.

Первая причина связана с эффективностью кодирующей схемы: содержимое в формате ASCII использует один байт на символ, и добавление к нему ссылок на сущности — ничто в сравнении с трудностями, связанными с включением символов старших битов неопределенной кодировки.

Вторая причина того, почему UTF-8 вполне пригодна в случае первоначально англоязычного содержимого, заключается в способах обработки шрифтов

в современных браузерах. Настройка кодировки по умолчанию для них — Auto, что в случае символов ASCII мало что значит. Однако эта же настройка позволяет браузеру обрабатывать сущности вне зависимости от изначальной кодировки заданных шрифтов, так как связи между позициями кода Юникод и позициями кода в других схемах хорошо задокументированы. Браузер явным образом преобразовывает позиции кода, и в итоге выводится нужный символ.

Конечно, вывод шрифтов браузером не заканчивается на UTF-8 и настройке Auto. Некоторые последовательности действий могут привести к тому, что в документе появляются странные символы. Самые распространенные случаи такого рода — это подтверждение форм браузерами с пользовательскими настройками кодирования и публикация содержимого, вставленного прямо из программ обработки текста (которое в случае латинского алфавита, как правило, закодировано 8-битными символами одинаковой длины).

Когда эти «странные» символы попадают в продукт, они представляют собой нечто вроде «изъянов» в потоке символов, которые браузер посетителя может обработать неправильно. Если такие случаи происходят часто, то вам стоит сменить кодировку документов на ISO 8859-x и Windows-1252 (которые представляют собой практически одно и то же). Для разработчиков, имеющих дело с содержимым на азиатских языках, существуют похожие методы решения проблемы.

Представления символов, не принадлежащих ASCII, в виде сущностей

В европейских языках встречается несколько диакритических знаков. Единственный из них, который используется в английских словах, это умляут (¨), чей краткий пик популярности пришелся 1960-е и 1970-е годы — тогда он использовался в качестве сигнала читателю о том, что вторая гласная в паре должна быть произнесена отчетливо и кратко (к примеру, «coördination»). С тех пор умляут был в таких случаях вытеснен дефисом, вставляемым в середину подобных пар. Кроме того, умляут может быть частью фирменного стиля.

Многие фирменные стили используют заимствованные слова, имеющие омографы в английском языке (распространенный пример — «résumé»). В них содержатся диакритические знаки, используемые в языке происхождения. Большинство распространенных европейских диакритических знаков имеется в списке сущностей символов HTML 4 и перечислено в табл. 12.5 по алфавиту.

Не все диакритические знаки сочетаются со всеми буквами; те, что сочетаются с конкретной буквой, могут быть применены и к заглавной, и к строчной букве. На сайте, дополняющем данную книгу (http://www.htmlcssgoodparts.net), вы сможете найти полную таблицу сущностей HTML. Это список, составленный

Эдриеном Роселли, также включает десятичные коды для символов старших битов схемы ISO 8859-x, которые не поддерживаются XHTML. В дополнение к буквам с диакритическими знаками существуют несколько специализированных символов, которые пригодятся дизайнерам, стоящим перед необходимостью создания содержимого, соответствующего высоким стандартам типографики. Многие из них перечислены в табл. 12.6.

Таблица 12.5. Распространенные западноевропейские диакритические знаки, используемые в HTML 4

Название	Диакритический глиф	Шаблон сущности	Пример
Акут	´	&?acute;	é, ĺ
Седиль	¸	&?cedil;	ç
Циркумфлекс	ˆ	&?circ;	ô
Тупое ударение	`	&?grave;	à
Тильда	˜	&?tilde;	ñ, ã
Умляут, трема	¨	&?uml;	ö

Таблица 12.6. Полезные сущности HTML, перечисленные в алфавитном порядке

Символ(-ы)	Литерал (-ы)	Буквенно-цифровое значение/значения	Значение/значения Юникод [1]	Лучше использовать вместо
Абзац		¶	¶	
Авторское право	©	©	©	(c)
Деление	÷	÷	÷	/ не в вычислительном контексте
Длинное тире	—	—	—	Окруженного пробелами - или --
Дробная половина 2	½	½	¼	½
Евро	€	€	€	
Жирная точка	•	•	• (U+2022)	*
Зарегистрированный торговый знак	®	®	®	(R)
Йена; юань; жэньминьби	¥	¥	¥ (U+00A5)	RMB для валюты KHP
Кавычки двойные [универсальные] [3]	«	"	"	
Кавычки двойные	" "	“ ”	“ ” (U+201C, U+201D)	", ", или ``
Кавычки двойные нижние	„	„	„	" или .. в некоторых языках
Кавычки одинарные	'	‘ ’	‘ ’ (U+2018, U+2019)	' или '

продолжение ⇨

Таблица 12.6 *(продолжение)*

Символ(-ы)	Литерал (-ы)	Буквенно-цифровое значение/значения	Значение/значения Юникод [1]	Лучше использовать вместо
Кавычки одинарные нижние	‚	&sbquo;	&#8218;	' или запятой в некоторых языках
Кавычки парные (елочки)	«»	« »	« » (U+00AB, U+00BB)	" в некоторых языках
Короткое тире	–	&ndash;	&#8211;	- в диапазонах
Крест (одинарный и двойной)	† ‡	&dagger;	† ‡	* и ** в некоторых аннотациях
Меньше или равно/больше или равно	≤ ≥	≤ ≥	≤ ≥	<= и >= не в вычислительном контексте
Не равно	≠	&ne;	&#8800;	!=
Обратные восклицательный и вопросительный знаки	¡ ¿	¡ ¿	¡ ¿	
Опущение	…	&hellip;	&#8230;	...
Параграф	§	&sect;	&#167;	
Плюс—минус	±	&plusmn;	&#177;	+/-
Примерно равно	≈	&asymp;	&#8776;	~
Пробел (неразрывный)				
Промилле [4]	‰	&permil;	&#8240;	
Степень	°	&deg;	&#176;	
Торговая марка	™	&trade;	&#8482;	(tm) и т. п.
Точка (умножение)	·	&middot;	&#183;	
Умножение	×	&times;	&#215;	x, X; * не в вычислительном контексте
Фунт (валюта)	£	&pound;	&#163;	
Цент (валюта)	¢	&cent;	&#162;	
Штрих (одинарный и двойной)	′ ″	′ ″	′ ″	', ", ', ", окруженного m:s
Эсцет	ß	&szlig;	&#223;	«ss» (в немецком языке)

[1] Некоторые символы могут использоваться в качестве значений `content`, связанных с англоязычным содержимым; в таких случаях в таблице присутствуют и десятичная сущность, и шестнадцатеричные значения позиций кода Юникод. За более подробной информацией обратитесь к рассмотрению свойства `content` в главе 14.

[2] Четверть и три четверти также могут быть представлены похожими структурами.

[3] Единственными именованными сущностями, поддерживаемыми XML, являются `&amp;`, `&lt;` и `&gt;` (&, < и > соответственно). При работе в XML на все сущности, описанные в таблице, надо ссылаться, используя их десятичное значение.

- Символ, обозначающий сотую часть процента (‰), также имеется в коде, но не упоминается в спецификации HTML 4.

Наконец, имейте в виду, что на значения кода Юникод, представленные в четвертой колонке таблицы, нельзя опираться в том случае, если документ закодирован не при помощи UTF-8.

Руководство по использованию шрифтов

Если вы какое-то время занимались разработкой или выпуском документов, то вы почти наверняка знаете, как избегать того, что называется «эффект записок с требованиями выкупа» («ransom note effect») — размещение в одном документе слишком большого количества гарнитур.

Однако в Сети этот эффект затрагивает не только шрифты, но и цвета, размеры и стили, которые вы к ним применяете.

Предсказуемость, предпочтение, паника

Сайт разбит на разделы, а каждый раздел содержит одну или более страниц. Легкость, с которой сетевые документы могут быть организованы с точки зрения иерархии, означает, что увеличение степени постоянства в представлении сайта (а это пригодится в дальнейшем) не только возможно, но и *легко реализуемо*. Нагрузка на разработчика и дизайнера может быть сокращена, а поиск и выделение частей макетов страниц сайта, которые служат указателями для посетителей, становятся простыми.

Проблема состоит в том, что у постоянства, как и у большинства аспектов Сети, есть свои пределы: во многих случаях, особенно в тех, где задействовано создаваемое пользователями содержимое, невозможно предсказать количество содержимого, которое может появиться на определенной странице. Эта непредсказуемость приводит к определенной потере контроля — а затем к попыткам в панике подчинить себе внешний вид, поведение и границы содержимого макетов сайта. Общепринятая реакция на эту панику — внесение легких перемен в настройки размеров промежутков, линеек и шрифтов по всему сайту, что, на первый взгляд, делает борьбу с возникающими проблемами макетов несложной.

Оценка масштабов содержимого

Первый этап процесса ужесточения контроля над шрифтами — это организовать содержимое иерархически. На вашем сайте могут присутствовать некоторые (или все) из перечисленных ниже элементов:

- имя сайта;
- основной текст;

- названия/заголовки;
- передовые абзацы;
- содержимое боковых панелей;
- дополнительное содержимое;
- навигация:
  - основная,
  - вторичная,
  - исходящая/третичная,
  - «хлебные крошки»;
- точки привлечения внимания и/или реклама (мультимедийное содержимое, призывающее пользователей проследовать по ссылке, ведущей на другую страницу);
- формы;
- средства приложений;
- гипертекстовые ссылки.

При таком широком разнообразии функциональных элементов дизайнеров подстерегают две серьезнейшие ошибки. Первая из них уже была упомянута: поддаться желанию «подкрутить настройки», если того требует случай, что превращает работу дизайнера в пародию на работу с CSS. Большое количество накапливающихся элементов `id`, `class` и откровенного мусора приводит к тому, что в результате получается полная каша.

Вторая ошибка — разработка под влиянием убеждения, что каждый важный элемент требует своего собственного шрифта, что в общем случае неверно.

На практике вам понадобятся шесть значимых классов шрифта, которые должны различаться:

- имя;
- названия/заголовки;
- второстепенные заголовки;
- основной текст;
- навигация;
- гипертекстовые ссылки.

В некоторых случаях следует оформлять выделяющимися шрифтами и вторичное содержимое, такое как длинные цитаты или боковые панели.

При обдумывании этого списка (и реже — при добавлении к нему новых пунктов) главный вопрос таков:

Почему этот элемент должен отличаться от остальных?

Каждый элемент списка представляет собой фундаментальную часть сайта (может стать необходимым указателем внутри страницы и внутри узла либо

быть первичным содержимым), и это оправдывает необходимость выделения.

Как разнообразить шрифт: гарнитура, размер, вес, стиль, цвет

Хотя информация, представленная в данном разделе, может показаться очевидной, я включил ее в эту книгу для того, чтобы она могла предоставить некую основу в процессе принятия дизайнерских решений.

Когда вы определили классификацию содержимого и ее взаимосвязь со шрифтами, вам необходимо решить, как выполнить и реализовать дизайнерские решения.

Гарнитура, размер, вес, стиль и цвет: когда вам нужно, чтобы текст выделялся, выбирать приходится именно из этих свойств. Возможность выделять определенные области документа при помощи цвета (или тени) фона может помочь вам принять решение, однако сама по себе решением не является.

В свете традиций и прочих факторов, каждое из этих свойств предлагает обычному читателю разные возможности.

- *Гарнитура.*

  Различные гарнитуры помогают классифицировать содержимое; к примеру, в некоторых газетах заголовки набраны шрифтом без засечек, а основной текст — шрифтом с засечками. Однако такой подход не настолько популярен, насколько может показаться — либо потому, что заголовки не являются второстепенным содержимым, либо по причинам случайного (традиции) и практического (различия между метриками шрифтов) свойства.

- *Размер.*

  Увеличенный или уменьшенный шрифт дает понять, насколько важен в относительном плане конкретный отрывок. Также он приводит к наиболее тяжелым проблемам с таблицей стилей. Каждое применение различных размеров шрифтов ведет к изменениям в композиции, которые, в свою очередь, создают множество проблем, из-за чего дизайнерам приходится ложиться спать ближе к утру.

- *Жирность.*

  При помощи жирного шрифта можно выделить нужное содержимое, но только в том случае, если такой текст будет выделяться на фоне расположенных рядом объектов. Увеличение жирности текста лучше всего применять в паре с увеличением размера.

- *Стиль.*

  Применение курсивных/наклонных шрифтов дает представление о смене тона автора и обычно отвечает за акцент на определенных словах. Эти

шрифты также применяются для выделения некоторых существительных, например, названий периодических изданий, названий творческих изделий, кораблей, именованных самолетов и (редко) известных мест. Для обозначения ударения также используются мелкие заглавные буквы и широкие пробелы между буквами, но это происходит намного реже.

- *Цвет.*

  Применение цвета в Сети происходит гораздо проще, чем в других медиа, однако оно ставит в невыгодное положение пользователей с нарушенным цветовым восприятием. Поэтому при применении цвета для обеспечения акцента на ссылках и другом тексте нужно полагаться и на яркость, и на оттенок. В идеале использование цвета при оформлении должно сочетаться с другими видами дифференциации. Об этом подробно сказано в главе 9.

Как правило, лучше уменьшить количество способов выделения шрифта, так же как и количество выделяемых элементов. Когда к одному и тому же отрывку применено много выделений, посетителю в результате трудно разобраться в нем.

После того, как вы сузили круг используемых инструментов, вам следует подумать над подробностями композиции. Самое сложное в этом процессе — определить, что случится в нестандартном случае.

- Что произойдет с содержимым, которое занимает больше места, чем было задумано, особенно в отношении используемой сетки?
- Как будут различаться отрывки с содержанием одинаковой важности, но с разными функциями, такие как заголовок основного текста и заголовок элемента боковой панели?
- Если разработчик воспользовался печатным подходом к композиции и внес небольшие различия между композициями, то каким образом дизайнер сможет собрать воедино все правила?

Настройка шрифтов вокруг разрывов

На последние два из заданных выше вопросов можно ответить при помощи хорошо составленной таблицы стилей.

Самое *легкое* решение проблемы потенциальных разрывов такое: установить свойства текста и бокса нужных элементов при помощи наименьшего возможного числа правил, а затем добавить в свойства бокса `overflow: hidden`, держа в уме, что теперь ответственными за разрывы становятся авторы содержимого. В условиях, когда отсутствие ограничений для содержимого — частый, но не необходимый факт, я обычно являюсь приверженцем такого подхода. (Ограничение количества места под содержимое, как правило, ведет к простоте и ясности, что хорошо знакомо пользователям сети Twitter.)

Если такое решение проблемы либо непрактично, либо просто невозможно — а так получается достаточно часто, — то ответственность за различия с желаемым результатом ложится на плечи дизайнера. Возьмем, к примеру, заголовок,

который должен уместиться в одну строку; как быть с заголовками, которые занимают две строки? Уменьшение размера шрифта, возможно, не будет решением, а вот изменение `line-height` для того, чтобы длинный заголовок поместился в сетку, может послужить таковым.

Другой случай — слово, написанное через дефис и слишком длинное, чтобы уместиться в одну колонку; Internet Explorer прервет строку на дефисе, но другие браузеры так не сделают. Вставка элемента `&zwnj;` нулевой ширины (как правило, он используется для обозначения логического, но невидимого пространства между двумя глифами) также не помогает.

Во всех случаях во избежание разрывов работа со шрифтами должна следовать трем правилам.

- Следовать сетке, размеченной для всего макета.
- Логически подстраиваться под размеры, установленные существующими стилями и/или обработкой шрифтов.
- Беречь свободное место при минимальном изменении процесса.

Оформление отрывков одинакового приоритета

Работа дизайнера упрощается, если он начинает с одного базового стиля шрифта и «отталкивается» от этой базы; CSS предназначен именно для такого подхода. Для простого сайта с двумя колонками текста результат может выглядеть так.

- *Основной текст.*
  Georgia, 12 пикселов, черный цвет.
- *Заголовки.*
  Увеличиваются от `h4` (как правило, это размер текста по умолчанию) с шагом в 4 пиксела.
- *Боковая панель.*
  Весь текст высветлен до 75% черного.
- *Навигация.*
  Увеличение размера на 4 пиксела для основной навигации, обычный размер для вторичной; управление цветами через псевдоэлементы ссылок.

Таблица стилей, отражающая такой подход, содержит следующие правила и пары свойств–значений:

```
body { color: rgb(0.0.0); font-size: 12px; font-family: Georgia. serif; }
h1 { font-size: 2em; }
h2 { font-size: 1.667em; }
h3 { font-size: 1.333em; }
#sidebar { color: rgb(64.64.64); }
#navPrimary a:link { color: rgb(0.0.192); }
#navSecondary a:link { color: rgb(0.192.0); }
#navPrimary a:active.
#navSecondary a:active { color: rgb(192.0.0); }
```

Для дифференциации различных разделов сайта обычно требуется дополнительный труд и дополнительные селекторы, однако базовый принцип должен быть ясен.

Проблемы возникают тогда, когда корректировкой начинает заниматься дизайнер, обученный печатному подходу. Предположим, что заголовок колонки не может быть ужат по горизонтали, и дизайнер уменьшает размер шрифта заголовков в этой колонке для поддержания постоянства.

Повторите это действие три или четыре раза на одном и том же сайте. Спустя некоторое время вам придется писать правила с селекторами вроде:

```
body.about#contact #sidebar h3.telephoneContact
```

Также вполне вероятно что, если вы употребляете такие селекторы, то вам приходится писать еще и селекторы для многих элементов между `body` и `body.about#contact #sidebar h3.telephoneContact`.

Работа со шрифтами

Только что описанного сценария можно было бы избежать, если бы было заранее условлено, что заголовки боковых панелей могут занимать две строки. Такие вещи оговариваются в руководстве по стилю. Руководство по работе со шрифтом — это дополнение к руководству по стилю, которое содержит все шрифты, используемые в продукте.

Logotype	ACME WIDGETS, INC.	Copperplate Gothic Bold, 48px, #000000
A-head	This is a headline.	Helvetica Neue, 36px, #404040
B-head	Quick red fox	Helvetica Neue, 36px, #606060
C-head	Lazy brown dog	Helvetica Neue Bold, 16px, #404040, 24px line-height
Body copy	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur id lectus	Helvetica Neue, 16px, #404040, 24px line-height
Navigation	PRODUCTS	Copperplate Gothic Bold, 24px, #404040
Inline links	visit one of our locations	Helvetica Neue, 16px, #000000, 24px line-height, underlined

Рис. 12.8. Пример руководства по работе со шрифтами в три столбца (создан автором)

Когда я создаю руководство по обработке шрифтов, результат обычно выглядит примерно, как на рис. 12.8 — это таблица, поделенная на три столбца:

1. Назначение и, возможно, селектор/селекторы для данного шрифта (например, имя сайта, основной текст, заголовки).
2. Образец шрифта.
3. Метрики шрифта, написанные либо на нужном языке, либо в форме пар свойств-значений CSS.

Если разработчик сайта оснастил продукт исключениями и сторонними значениями, то такое руководство может вырасти до нескольких страниц, и это будет значить, что некоторые аспекты процесса вышли из-под контроля.

Что еще более важно, эффективное руководство документирует многие дизайнерские решения в более удобочитаемом формате, чем в таблице стилей, и это может спасти огромное количество времени, если вы столкнетесь с проблемами вроде текучки кадров и пренебрежения поддержкой сайта.

Разное о типографике в CSS

У CSS есть несколько странных свойств, которые демонстрируют, что при помощи CSS дизайнеры не могут управлять всем, однако имеется ряд важных деталей, располагающихся где-то посередине между простотой и элегантностью.

Свойство line-height

Как было сказано ранее, свойства `line-height` вставляют пустое пространство между строками. Однако этот атрибут действует по обе стороны от строки, как показано на рис. 12.9. К счастью, в современных браузерах так происходит постоянно; прежние версии помещали все дополнительное пространство либо над строкой (как в Internet Explorer 6), либо под ней, что также отображено на рис. 12.9.

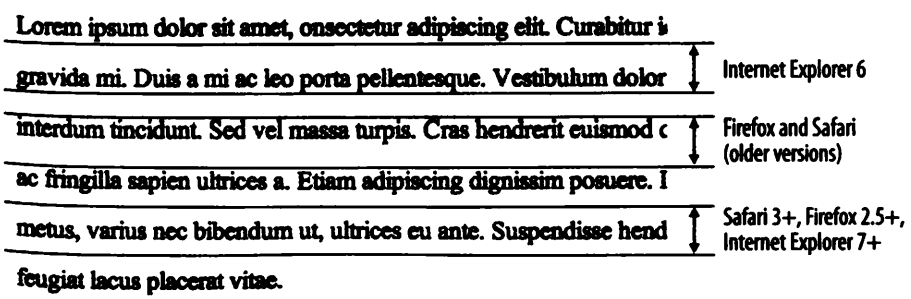


Рис. 12.9. Поведение свойства line-height в различных браузерах (в данном примере использован шрифт Times New Roman размером 16 пикселов при значении line-height 32 пиксела)

Другая важная особенность свойства `line-height` состоит в том, что ее область значений включает в себя числа без единиц, например:

```
line-height: 1.5;
```

что с точки зрения функциональности равносильно значению 150% или 1,5 `em`.

Я считаю, что ради постоянства и порядка лучше использовать одни и те же единицы размера и для свойства `line-height`, и для других свойств шрифта и текста.

Значения по умолчанию свойства `line-height` задаются значением `normal`, меняются от одной комбинации шрифта и платформы до другой и, как правило, очень малы, порядка 120–125%.

Свойства font-variant и text-transform

Самое привычное назначение свойств `font-variant` и `text-transform` — это обеспечение необычных форм выделения, к примеру выделения торговой марки или отрывка, произносимого повышенным тоном.

Свойство `text-transform` используется редко, и его значения обыкновенны (самое распространенное среди них — `uppercase`). Свойство же `font-variant` и его единственное значение не по умолчанию — `small-caps` — это совсем другой случай.

Шрифты, которые используются для вывода заглавных буквенных форм при изменяющихся размерах для обозначения капитализации, часто перерисовываются таким образом, чтобы линии заглавных букв были сопоставимы с другими буквами этого шрифта. Когда вместо этого используется свойства `font-variant`, обычные строчные буквы заменяются заглавными буквами похожего размера, и в результате прописные буквы выглядят чуть больше строчных. Тот же результат наблюдается и в случае некоторых обычных шрифтов, особенно разработанных исключительно для печати, но в случае с текстом, к которому применен `font-variant: small-caps`, он в особенности очевиден.

Свойства letter-spacing и word-spacing

Изменений ширины пробелов между буквами и словами, как правило, лучше избегать, однако иногда приходится прибегать к ним для того, чтобы какая-то часть композиции выглядела именно так, как надо. Другое, нетипичное применение этих свойств — это обозначение ударения, как будто рассказчик *вы-ы-ытя-я-я-ягивает* слоги в слове. Пробелы между буквами помогают передать такой голос вне фраз диалога.

Значения `letter-spacing` и `word-spacing` обычно равны небольшому числу в `em`, но здесь есть одна заминка: в случае компьютеров Mac это значение перед применением к странице округляется до целого числа пикселов. Windows при разрешенном ClearType, более гибок в этом плане.

Свойство white-space

Элемент pre необычен тем, что он может обозначать содержимое с семантически важными внешними характеристиками — например, цитаты из сообщений электронной почты. С другой стороны, очень легко злоупотребить его возможностью управлять поведением обрыва строки без необходимости вставки элементов br или дочерних элементов div.

Если свойство white-space и его значение pre назначены какому-либо элементу, то содержимое выводится таким образом, как если бы оно находилось в пределах элемента pre. Использование этой пары свойства/значения также обеспечивает дополнительную гибкость для разработчиков, которые различают настоящее предварительно форматированное содержимое от содержимого, которое должно отличаться высокой степенью контроля презентации.

Кроме normal (значение по умолчанию), у свойства white-space есть три других значения, которые отличаются от pre тем способом, которым они обеспечивают мягкий и жесткий обрыв строки.

Веб-типографика на практике

Важнейшее впечатление от сайта формируется его внешним видом. Отвечая на высокие ожидания современных посетителей, интернет-платформы предлагают множество инструментов для работы с характеристиками шрифтов и приближают веб-типографику к эстетическим ожиданиям посетителей сильнее, чем раньше.

Сложность относительно этих повышенных ожиданий состоит в том, что отличное качество веб-типографики требует отличного же знания теории... и если вы усвоите теорию, то дела с практическим применением будут обстоять намного проще!

13 Понятные и доступные формы

Разработка веб-приложений требует большего, чем просто создание макета CSS и подбор шрифтов. Если веб-сайты просто предоставляют информацию посетителям, то веб-приложениям необходимо получать информацию от посетителей. Сетевые приложения процветают либо увядают в зависимости от своего дизайна и реализации.

Там, где нужна информация от пользователей, в игру вступает форма, а там, где есть форма, всегда есть множество вариантов испортить взаимодействие пользователя с сайтом. В этой главе содержится обзор методов разработки и реализации форм, которые помогут вам минимизировать риск возникновения фатальных ошибок.

Разработка эффективных форм

Для создания хороших форм недостаточно знаний в области разметки. Важнейшим фактором для разработчика является понимание того, каким образом пользователь взаимодействует с формой.

Сетевые приложения, пользовательская перспектива и выбор дизайна

Представьте себе большой набор данных — скажем, сборник стихов.

Веб-сайт, содержащий стихотворения, скорее всего, хранит их в базе данных SQL, которая располагает бесконечным числом способов упорядочивать и размещать свое содержимое по требованию разработчика. Основа такого воображаемого сайта описана в следующей команде создания таблицы MySQL, которую стоит привести здесь, поскольку она более или менее понятна:

```
CREATE TABLE poems(
            id SERIAL,
     author_id MEDIUMINT,
    date_added DATETIME,
      date_pub DATETIME,
    discussion TEXT,
     editor_id SMALLINT,
      folio_id MEDIUMINT,
       lang_id SMALLINT,
lang_source_id SMALLINT,
    marginalia TEXT,
         title VARCHAR(1024),
 translator_id MEDIUMINT,
         verse MEDIUMTEXT
);
```

Дополнительные ограничения были опущены ради сохранения читабельности.

Изучив структуру таблицы, опытный разработчик приложений может выявить некоторые принципы, лежащие в основе административной части сайта — к примеру, то, что данные об авторах, переводчиках и поддержке сайта хранятся в разных таблицах одной и той же базы данных.

Если рассматривать структуру с точки зрения пользователя, то можно говорить о различных *представлениях*, которые можно применить к содержимому гипотетического сайта без серьезных требований к ресурсам. Затем эти представления данных повлияют на все, что касается сайта, особенно на информационную архитектуру и процесс разработки.

Представления, которые можно сформировать при помощи простых запросов `SELECT`, могут использовать следующие поля:

- автор;
- дата первой публикации;
- том/сборник;
- язык вывода;
- язык источника;
- название;
- переводчик.

В дополнение к этим представлениям, поле `date_pub` позволяет легко экспортировать новое содержимое в RSS, а другие поля — особенно `verse`, содержащее собственно текст — обеспечивают возможность полнотекстового поиска.

Для каждого из этих представлений существует несколько методов построения форм и таблиц содержимого, которые могут быть использованы для поиска. Если это содержимое создается пользователями, то имеется большая свобода для создания дизайна форм для публикации содержимого.

Самый простой подход к разработке — это использование CRUD.

Организация пользовательского интерфейса при помощи функций

Если рассматривать содержимое в терминах *записей* — наименьших групп содержимого, которые могут быть представлены вне контекста, то для них традиционно существуют четыре вида действий. CRUD расшифровывается как «create, read, update and delete» («создание, чтение, обновление и удаление»); соответствующие SQL-запросы для каждого действия указаны в скобках.

- *Создание* (INSERT, CREATE).

  Создает записи или наборы записей, которые прежде не существовали. Формы, созданные посредством этой функции, будут иметь поля пустые или заполненные различными значениями.

- *Чтение* (SELECT).

  Находит и отображает существующие записи в режиме «только для чтения». Самые простые формы, созданные с использованием этой функции, нужны для полнотекстового поиска; на сайтах, которые разработаны для просмотра, а не для поиска, пользовательский интерфейс этой функции состоит исключительно из гипертекстовых ссылок.

- *Обновление* (UPDATE).

  Изменяет либо часть содержимого записи, либо всю запись целиком. Верно разработанные приложения, как системно-резидентные, так и основанные на архитектуре клиент-сервер, будут обладать формами, схожими с использующимися для функции «создание», но сперва будут считывать существующую запись и вставлять ее значения в формы.

- *Удаление* (DELETE, DROP).

  Полностью удаляет запись или набор записей. Сетевые приложения редко выносят *настоящие* функции удаления на общее обозрение, а вместо этого по практическим причинам помечают «удаленные» записи как невидимые, оставляя реальное удаление на усмотрение владельца сайта. Обычный (но не универсальный) тип дизайна пользовательского интерфейса для этой функции — это список записей, каждая из которых снабжена управляющим элементом `input type="checkbox"`; эти записи занесены в таблицу, с которой связана ссылка или кнопка, названная Удалить или Стереть.

Здравый смысл подсказывает, что при разработке приложений и форм на основе данных функций следует быть крайне аккуратным. Некоторые из следующих советов относятся непосредственно к разработке и реализации интерфейса. При разработке любой таблицы стилей, касающейся форм, должны учитываться хотя бы несколько из перечисленных ниже правил.

Десять правил разработки эффективных веб-форм и приложений

При создании эффективных веб-приложений следует обращать особое внимание на безопасность, простоту и прозрачность. Для этого необходимо соблюдать следующие правила.

1. *Не запрашивайте (и тем более — не требуйте) больше информации, чем необходимо.* Даже при хорошей пропускной способности и достаточном количестве памяти пользователь больше всего ценит *время*. Уважайте время посетителей (и не забывайте про безопасность) при разработке форм, делая упор на краткость. Дополнительные формы могут подождать до более подходящего момента, а записи пользователей должны обновляться своевременно.
2. *Отличайте обязательные для заполнения поля от необязательных.* Для указания того, что для успешного подтверждения формы *необходимо* правильным образом заполнить определенное поле, используйте две подсказки: визуальную и текстовую. Если необходимо заполнить *все* поля, то четко напишите об этом в инструкции.
3. *Составляйте четкие инструкции и сообщения об ошибках и неисправностях.* Если при работе вашего сетевого приложения — пусть даже это простейшая форма электронной почты — происходит ошибка из-за несоблюдения пользователем ограничений отправки, опишите эти ограничения ясным и доступным языком.
4. *Заранее объясните, какой будет результат в случае успешного подтверждения формы.* Во многих случаях, особенно при поиске, результаты успешного заполнения формы либо подразумеваются, либо соответствуют общепринятой практике. В том случае, если вы не разрабатываете одну из привычных для посетителя страниц — и *особенно* в том случае, если вы запрашиваете ценную информацию — посетителю захочется знать: а что я с этого буду иметь? Четко ответьте на этот вопрос. По той же причине вам необходимо предусмотреть предупреждение в случае, если результаты подтверждения нетипичны.
5. *Не используйте в работе своего приложения сомнительные предположения относительно состояния браузера и сессии посетителя.* Протокол HTTP не использует информацию о состоянии; REST (Representational State Transfer) — это практический прием, позволяющий разрабатывать сервисы на основе HTTP с учетом этого факта. В частности, не стоит предполагать, что посетитель пользовался другими частями приложения в течение текущей сессии, если только вы не настроили механизм (например, хэш сессии), который может это точно подтвердить.
6. *Выбирайте типы полей, уменьшающие требования к количеству и точности движений, совершаемых пользователем.* Для ввода непроизвольных значений, таких как булевы значения и списки стран, используйте

select, checkbox и radio. Если вы хотите добавить checkbox для ввода булева значения, используйте только один флажок. Располагайте элементы управления таким способом, чтобы варианты выбора были видны; использование колеса прокрутки или клавиши Page Down для перемещения к конкретному полю option может быть оправдано разве что очень длинным (и предсказуемым) списком вариантов. Всегда избегайте применения select multiple.

7. *Вместе с элементами управления формами всегда используйте ярлыки.* Элемент label выходит за рамки всей семантической ерунды: он является активной частью интерфейса, связанного с определенным элементом управления формой при помощи атрибута for. Когда пользователь щелкает на label, вступает в действие элемент управления формой с id, относящимся к значению for этого label.
8. *Сделайте ввод информации настолько понятным, насколько это возможно.* Формы делайте короткими, оформляйте элементы управления текстом так, чтобы сфера их действия распространялась на наибольшее возможное количество вводимых данных, а текст в форме располагайте так, чтобы его размеры были больше или равны размерам основного текста. С другой стороны, старайтесь уместить все обязательные поля на одной странице — во избежание ненужных зависимостей, приводящих к сбоям сессий.
9. *Придерживайтесь постоянных размеров полей, выравнивания и ограничения колонок.* Когда средства управления имеют постоянный размер и выравнивание, то потребность обычного пользователя в зрительном поиске сводится к минимуму. Для повышения понятности ввода рекомендуется использовать небольшие наборы длин полей и стилей.
10. *Сфокусируйте активность пользователя на одном из четырех основных действий: создании, чтении, обновлении или удалении.* Ограничивая интерфейс каждого приложения определенным набором действий, вы существенно снижаете риск ошибки пользователя в случае, например, диалогов подтверждения.

Эти десять правил — не единственные, которым стоит следовать, однако они применимы практически во всех случаях.

Оценка и структура

Наиболее удачные реализации форм представляют собой пример простоты и правильной расстановки приоритетов.

Первый шаг на пути к этой цели — определить, *что именно* форма запрашивает у посетителя. В некоторых случаях эти требования и используемые дизайнерские решения подсказаны традицией и здравым смыслом, но в других случаях на реализацию влияет множество факторов: традиции отдельных фирм

соблюдение законов, ожидания посетителей и использование/отказ от использования Ajax и т.п.

Говоря обо всем этом, я пытаюсь показать, насколько важны *внимательность* и *осторожность*; высочайший уровень значимости форм не всегда уживается с необдуманным выбором дизайнера.

Безопасность приложения остается за границами данной книги, однако эта область не менее важна и требует внимательного изучения. Пожалуйста, всегда пользуйтесь стандартными методами обеспечения безопасности, особенно защитой вводимых данных формы от атак, основанных на внедрении кода SQL.

Определение требований

Перед началом разработки, скажем, каркаса или составной структуры, необходимо определить форму и функцию (если хотите) интерфейса формы. При этом зам необходимо будет решать следующие задачи.

- *Оценка.*

  Определите требования к форме как с точки зрения посетителя, так и с точки зрения владельца сайта. Это замечание может показаться слишком очевидным, разработчикам гораздо проще создавать формы, пользуясь только собственной точкой зрения. Если же учитываются требования посетителей, то полезными могут оказаться совершенно другие решения.

- *Определение границ.*

  В случае с веб-приложениями и другими средствами, запрашивающими у пользователя большие объемы информации, становится необходимым разделение задач на легко контролируемые части или шаги. Определяя границы для каждой формы на сайте, вы приближаетесь к «зоне умеренности» для длины каждой формы: не слишком длинная и не слишком короткая, но *именно такая, какая нужна.*

- *Выгода.*

  Задайте себе вопрос: кому и каким образом выгодно получать запрашиваемую информацию? В ходе размышлений возникают три варианта: посетитель, владелец сайта или они оба. Те поля ввода, которые приносят выгоду только владельцу сайта, должны быть удалены или оснащены неким стимулом.

- *Определение приоритетов.*

  В большинстве случаев цель посетителя сайта может быть выражена в виде простой команды и при помощи одного объекта. «Отправить сообщение», «создать учетную запись», «получить промокод» — вот типичные цели посетителей. Если форма включает поля, не относящиеся явным образом

к главной цели посетителя, то объясните посетителю, зачем они нужны, и поместите их ближе к низу формы.

- *Типы данных.*

  Установите тип данных, которые необходимо вводить, и назначьте соответствующий элемент для ввода. В этом вам поможет таб. 13.1.

Таблица 13.1. Элементы форм, описываемые типом данных

Тип	Элемент	Дополнительная информация
Произвольный текст или числа (небольшая длина)	`input type="text"`	Лучше использовать для слов, фраз и фрагментов строк (к примеру, URI).
Произвольный текст (большая длина)	`textarea`	Лучше использовать для предложений, абзацев и списков произвольной длины, которые содержат данные, отделенные друг от друга разделителями строк — к примеру, URI или номера отслеживания; амперсанды перед использованием данных `textarea` в обновленном контексте должны быть переведены в формат `&amp;`.
Пароли	`input type="password"`	Оформляются подобно `input type="text"`; данные должны быть представлены в чистом виде, а шифрование должно происходить при помощи отдельного механизма.
Личные данные	`input type="checkbox"`	Для выбранного элемента в XHTML требуется `checked="checked"`; невозможные для выбора варианты должны быть отображены в качестве неактивных элементов.
Двоичный/троичный выбор	`input type="radio"`	Содержится в пределах единого `fieldset`; для активированного элемента в XHTML требуется `checked="checked"`; исключающие друг друга варианты выбора должны иметь одно и то же значение `name`.
Большая неизменяемая область (одно значение)	`select`	Для выбранного варианта в XHTML требуется `selected="selected"`; зоны значений, незнакомые посетителю, должны быть немедленно понятны без необходимости прокрутки.
Большая неизменяемая область (несколько значений)	`select multiple` `input type="checkbox"`	Должно содержаться в пределах одного `fieldset`, преимуществом станет аккуратный выбор значений `width` и `overflow`; также см. пункты «Личные данные» и «Большая неизменяемая область (одно значение)» выше в таблице.
Загрузка файлов	`input type="file"`	Смотрите раздел «Метод post и загрузки файлов» на с. 283.
Неизменяемые, закодированные в ASCII, зависящие от сессии и от пользователя данные	`input type="hidden"`	Альтернатива «cookies»-сессии, при использовании которой данные не удаляются окончательно до того момента, когда закрывается окно браузера и удаляется кэшированная страница.

Тип	Элемент	Дополнительная информация
Кнопки управления интерфейсом	`input type="button", button`	Лучше всего вставлять их при помощи пользовательского скрипта, так как он может лишь запускать события; удобно присваивать такому элементу `class`, который зарезервирован под элементы управления в виде кнопок; для вывода изображения вместо кнопки могут использоваться свойства фона CSS; `button` обладает более подробной поддержкой при отображении, чем его аналоги, однако его поддержка не настолько надежна.
Текстовое управление подтверждениями	`input type="submit"`	Лучше присваивать класс, зарезервированный для кнопок.
Растеризованное управление подтверждениями	`input type="image"`	С такими элементами используйте такие же значения, как и с обычными строковыми изображениями; когда такой элемент активируется в графическом браузере, он кодирует значения *x* и *y* в соответствии с координатами, которые получало событие `onclick` кнопки; скрипты, использующие эти значения, должны предусматривать вариант по умолчанию для пользователей вспомогательных технологий и текстовых платформ (если вы не стремитесь к большей доступности и не избегаете этого свойства вообще).

Для указаний, перечисленных в этой таблице, существуют и исключения. Например, механизм создания рейтинга на основе 7- или 10-балльной шкалы может быть реализован при помощи элементов `input type="radio"`, а не при помощи `select`.

Разметка и структура

Если вы определились с содержанием и порядком расположения полей в форме, можно переходить к разметке.

В дополнение к элементу `form` и элементам полей, описанным в таб. 13.1, вам понадобятся четыре других элемента.

- `fieldset`

  `fieldset` употребляется только в контексте формы. Его задача соответствует названию: создание особого контекста для идущих подряд похожих элементов, подробности к которым добавляются при помощи содержимого элементов `legend`. Только один из элементов `legend` должен присутствовать в `fieldset`, что соответствует требованиям стандартного типа документов HTML. Вполне понятно, что кандидатами на заполнение `fieldset` являются элементы `input type="radio"`; также возможно применение полей checkbox, полей даты/времени и нескольких полей `input`

type="text", предлагающих пользователю ввести информацию произвольного содержания.

- `legend`

  `legend` употребляется только в пределах элемента `fieldset` и применяется вместе с парой `label`/средство управления так же, как `label` используется вместе со средством управления.

- `ul` и `li`

  Большая часть разметки существует в смысловых парах: `label` — в паре с отдельным средством управления, `legend` — в паре с несколькими элементами `label` и элементами управления. Как правило, для пары необходим общий родительский элемент. Так как в пару можно вставить любой блочный элемент, то лучший выбор для связующего элемента — это `li`, что, в свою очередь, ведет к использованию `ul`.

Последний из этих вариантов противоречив. Некоторые разработчики предпочитают помещать объекты форм в элемент, который выражает некую степень цельности содержимого, а другие предпочитают выражать данное качество неявным образом. Отмечу, что средства чтения с экрана и другие вспомогательные технологии добавляют функциональные возможности, которые могут привести к увеличению времени, необходимое посетителю с ограниченными возможностями на изучение и использование формы.

В результате разметка для простой формы входа в систему может выглядеть так:

```
<form id="loginForm" ... >
    <fieldset><legend><span>Sign In</span></legend></fieldset>
    <ul>
        <li>
            <label for="username">Username:</label>
            <input type="text" name="user" id="username" value="" />
        </li>
        <li>
            <label for="password">Password:</label>
            <input type="password" name="pass" id="password" value="" />
        </li>
    </ul>
    <input type="submit" class="button terminalButton" value="Log in" />
</form>
```

Более сложная форма — например, в которой есть несколько полей `input type="radio"` — размещает такие поля и соответствующие им элементы `label` в пределах `fieldset`, который затем в исходном порядке располагается там, где обычно идет простой элемент `input type="text"` или `select` (рис. 13.1). Элемент `legend` также вставляется в подобные элементы `fieldset`, о чем будет сказано чуть дальше.

Примеры последнего подхода можно найти на сопутствующем сайте книги, а также в расширенном учебном пособии, посвященном формам и входящем

в учебный курс веб-стандартов компании Opera (http://dev.opera.com/articles/view/34-styling-forms/).

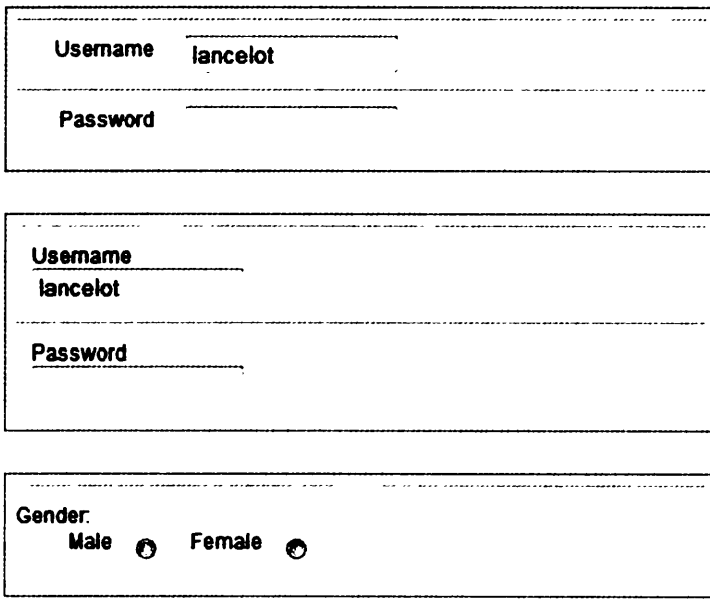


Рис. 13.1. Расположение типичных пар ярлык/поле

В приведенном выше примере разметки есть несколько ключевых моментов, которые могут не быть очевидными для невнимательного читателя.

- *Были опущены атрибуты* `length` *и* `maxlength`.

  Использование свойства `width` в CSS вытесняет на многих платформах атрибут `length`, а использовать `maxlength` в качестве меры предосторожности — не всегда хорошая идея. С другой стороны, эти атрибуты могут быть вставлены в разметку без вреда для пользователя и в некоторых случаях даже улучшить могут взаимодействие с ним.

- *В пределах* `legend` *формы присутствует анонимный тег* `span`.

  К сожалению, элемент `legend` плохо подчиняется применению стилей, а вот `span`, к счастью, таким свойством не обладает.

- *Источник формы тщательно отформатирован, что, на первый взгляд, не согласуется с поведением многих используемых строковых элементов и элементов inline-block.*

  В случае с формами доступность и надежность особенно важны, поэтому в процессе разработки таблицы стилей различным элементам все равно будет присвоено значение `display: block`.

- *Кнопка подтверждения отсутствует в пределах разметки списка и имеет два значения* `class`, *которые напрямую указывают на ее функцию.*

Назначение этого элемента и его `value` делают структуру `li/label/`элемент управления избыточной. Однако для того, чтобы выровнять данные элементы управления в соответствии с ограничениями колонок, необходимы специальные стили, а плохая поддержка селекторов атрибутов в Internet Explorer приводит к тому, что для идеального отображения на экране приходится применять большое количество «топорных» методов.

- *Значение* `id` *было назначается самой форме.*

  Значение `id`, присвоенное форме, может и не понадобиться, но если вы используете Ajax или размещаете на странице более чем одну форму, добавление `id` сократит время разработки и длину кода.

В некоторых рабочих средах может быть полезно присвоение значения `id` элементам `li` и кнопке подтверждения, особенно если дизайн средства обработки ошибок формы в браузере требует сложных стилей.

В формах, различающих обязательные и необязательные данные, значения `class` должны быть присвоены всем элементам `li`, окружающим поля, отвечающим за обязательные данные.

Итак, когда вы структурируете и размечаете форму, вы пытаетесь достичь трех целей.

- Обеспечить каждую пару `label`/поле достаточным количеством точек связи со стилями, DOM API и требованиями к совместимости со средами вспомогательных технологий.
- «Переусложнить» форму для борьбы с ошибками обработки в Internet Explorer, особенно версии 6.
- Сделать создание нормированных модульных выходных данных настолько простым для разработчиков приложений, насколько это возможно.

Последняя цель особенно труднодостижима, но если вы разрабатываете полное веб-приложение, то попытка стоит того. Хорошо нормализованную модульную разметку намного легче обрабатывать при помощи объектно-ориентированного скрипта, чем ее аналоги.

Перед тем как перейти к аспектам отображения форм, стоит рассмотреть несколько деталей — иногда неочевидных, иногда довольно-таки важных, — которые относятся непосредственно к разметке и поведению форм.

Структура, отображение и поведение простой формы

Тем из вас, кто работает в области разработки или редактирования, наверное, не терпится узнать: каковы же принципы двусторонней работы форм? (Это, кстати, был мой первый вопрос в процессе работы над моим первым большим веб-приложением в 1999 году.) С разметкой и поведением форм связано

несколько особенностей, хорошо знакомых опытным разработчикам, но неизвестных другим читателям.

Созданные формой запросы get

Если вы часто работаете с разметкой форм, вы наверняка заметили, что каждый элемент `form` имеет атрибут `action`, а у каждого элемента-поля есть атрибут `name`. Последние связаны с соответствующими значениями `value` и кодируются браузером следующим образом:

```
content=Hello+World%21
```

Это текстовое сообщение для веб-сервера, которое на привычном языке выглядит как «Hello World!»

Существуют два надежных метода отсылки этих данных серверу: `get` и `post`. Метод `get` прикрепляет закодированные данные к URI, обозначенном в атрибуте `action` формы, в результате чего получается следующее:

```
http://example.com/printmystuff.php?content=Hello+World%21
```

Отметьте, что константа? отделяет подтверждение данных от имени запрашиваемого ресурса — в данном случае, скрипта под названием **printmystuff.php** в корневом каталоге публичной файловой системы хоста.

Дополнительные пары `name/value` разделяются при помощи символьной константы & (амперсанда), как в данной строчке:

```
http://example.com/printmystuff.php?content=Hello+World%21&color=red&size=
xx-large
```

Хотя итоговые URI по существу не отличаются от URI без данных формы, преимущество подтверждения даты через формы состоит в том, что браузер кодирует данные без вмешательства человека (которое может включать в себя, скажем, необходимость наизусть знать половину таблицы кодов ASCII).

Поскольку амперсанды играют свою, особую роль, их необходимо изменить, чтобы они не могли быть использованы в качестве действительных значений `href` и `src`:

```
http://example.com/printmystuff.php?content=Hello+World%21&amp;color=red&a
mp;size=xx-large
```

Это странное требование привело к появлению профессионального термина «damnpersands» (в дословном переводе — «чертовы амперсанды»). Данный термин обязан своим происхождением амперсандам, не убранным из подтверждений содержимого несведущими (что вполне понятно) обычными пользователями, и сравнительно неважным выходным данным устаревших плагинов. Зачастую эти амперсанды являются единственным, что стоит между аккуратно составленной разметкой и успешной валидацией документа.

Если скрипт, на вход которого подаются кодированные данные, также имеет статус ресурса директории по умолчанию (в Apache обозначается как

DirectoryIndex), а соответствующий интерпретатор скриптов на сервере также сконфигурирован верно, то имя скрипта может быть вообще опущено:

```
http://example.com/?content=Hello+World%21&amp:color=red&amp:size=xx-large
```

Наконец, Internet Explorer ограничивает длину URI 2083 символами, из которых не более чем 2048 могут относиться к любой комбинации расположений файловой системы и закодированных данных формы. Если длина URI превышает это значение, Internet Explorer сообщает пользователю об ошибке.

Кодирование символов в URL: сущности ASCII

Вы, вероятно, заметили, что в предыдущих примерах вместо ! используется %21, поскольку ! в RFC 3986 является зарезервированным символом. Полный список зарезервированных символов приведен в табл. 13.2.

Таблица 13.2. Зарезервированные символы в URI и соответствующие кодировки ASCII

Символьная константа	Полное название	Кодировка	Десятичное значение
	Пробел	%20	32
!	Восклицательный знак	%21	33
#	Знак решетки	%23	35
%	Знак процента	%25	37
&	Амперсанд	%26	38
$	Знак доллара	%24	36
'	Апостроф	%27	39
(	Открывающая скобка	%28	40
)	Закрывающая скобка	%29	41
*	Астериск	%2A	42
+	Знак сложения	%2B	43
,	Запятая	%2C	44
/	Слэш (косая черта)	%2F	47
:	Двоеточие	%3A	58
;	Точка с запятой	%3B	59
=	Знак равенства	%3D	61
?	Знак вопроса	%3F	63
@	Коммерческий знак «собака»	%40	64
[	Открывающая квадратная скобка	%5B	91
]	Закрывающая квадратная скобка	%5D	93

Буквенные пробелы *всегда* кодируются — в рабочих путях URI при помощи %20, а в значениях, кодированных URL (вроде тех, что показаны выше), — при помощи константы +. Это приводит к некоторым неприятным, но необходимым изменениям: к примеру, если нужно отправить из формы 2 + 2 = 4, то данные перед подтверждением надо будет закодировать в вид 2+%2B+2+%3D+4.

Некоторые символы, описанные в табл. 13.2, присутствуют в неизменном виде в путях, однако кодируются при подтверждении в качестве данных формы.

Браузеры кодируют входные данные пользователя в URL, не нуждаясь во вмешательстве разработчика. Однако в состав всех распространенных в Сети скриптовых языков входят функции, преобразовывающие входные данные пользователя в формат URL и наоборот.

Метод post и загрузки файлов

Когда метод запроса, связанный с формой, меняется на post, закодированные данные формы прикрепляются к телу запроса, а не к URI. Помимо того, что подтверждения с помощью post помогают обойти ограничение в 2083 символа, с ними пользователям тяжелее работать методами, не предусмотренными дизайном формы. По этим причинам post является методом запроса для загрузки файлов.

Допустим, значение атрибута `enctype` элемента `form` равняется `application/x-www-form-urlen coded`; тогда формы, поддерживающие загрузку файлов, должны иметь значение `enctype`, равное `multipart/form-data`.

К несчастью для дизайнеров, элемент `input type="file"` достаточно противоречиво взаимодействует с CSS. Его наиболее важная характеристика в отношении макета заключается в том, что он рисует *два* объекта пользовательского интерфейса: аналог ввода текста и кнопку. По этой причине единственными свойствами CSS, которые вы можете применить к элементу управления загрузкой файлов без опасения за результат, являются `color`, `background-color` и свойства, задающие размещение потока текста на странице (к примеру, поля и свойство `position`). Эти ограничения особенно сильно влияют на кнопки, для которых в большинстве случаев невозможно создать оформление, отличное от определенного операционной системой.

Если вы применяете произвольные визуальные стили к элементам форм, не забывайте о консервативном подходе к элементам управления загрузкой файлов и о том, что их внешний вид по умолчанию изменить невозможно.

Изменение размера и внешнего вида отдельных элементов управления

Элементы управления формы ведут себя немного не так, как другие компоненты HTML.

- Значения свойств шрифта и текста в элементах управления формы не наследуются; эти значения должны быть заданы при помощи селекторов, в которых содержится прямая ссылка на элементы управления формы.
- В отличие от обычных элементов, элементы управления формы обрабатываются в «режиме совместимости» — все блочные свойства (кроме полей) находятся в пределах заданных для элемента управления значений `width` и `height` (если они заданы).

- Элементы управления формы (особенно `textarea`), как правило, игнорируют любое присвоенное им значение `line-height`.
- Элементы управления формы, за исключением элементов `input type="file"` и содержимого элементов `select`, могут быть оформлены при помощи прозрачного фона и специальных фоновых цветов.
- Вычисленная ширина элемента `input type="text"` рассчитывается по любому значению `length`, которое его определяет. Для удобства управления представлением следует избегать использования `length`; вместо этого можно задать в таблице стилей значения `width`, как и было сделано в приведенных выше примерах.
- Видимые размеры элементов управления на основе флажков/переключателей контролируются не значениями `width` или `height`, а значениями `font-size`.

Решить проблему отклонения от блочной модели в современных браузерах легко, поскольку все они поддерживают свойство `box-sizing` версии CSS3 (хотя в Firefox с Safari пока только в качестве расширений). Оно может принимать одно из двух значений:

- `content-box`

  Принудительно определяет элемент в соответствии со «строгой» блочной моделью: значение ширины элемента не включает значения границ и отступов. Если вы хотите, чтобы элементы управления формы обрабатывались в соответствии с «нормальной» блочной моделью, им должно быть присвоено это значение.

- `border-box`

  Элемент обрабатывается в соответствии с «совместимой» блочной моделью: вычисленная ширина включает в себя все установленные границы и отступы.

Наконец, у элемента `select` есть свои особенности:

- В обычных условиях элемент управления `select` со значением `width`, равным `auto`, примет ширину его самого длинного `option`.
- Если ширина `select` меньше самого длинного `option`, то «раскрывающийся» блок при активации все равно примет ширину самого длинного `option` (если он сможет «раскрыться», не перекрывая родительское окно просмотра).
- В Safari для элемента управления `select` могут быть заданы значения `width` и `font-size` , однако на все остальные аспекты оформления влияют особенности базовой библиотеки интерфейса, и изменить их нельзя.
- Элементы `option` могут быть сгруппированы в пределах элементов `optgroup` (рис. 13.2). В особенно длинных списках элементов `option` разделители должны добавляться с помощью `optgroup`; добавление пустых элементов `option` для этой цели — не очень хорошая идея.
- Если пользователь выбирает `option` без ненулевого значения параметра `value`, то видимое пользователю содержимое этого `option` будет закодировано и отправлено на сервер.

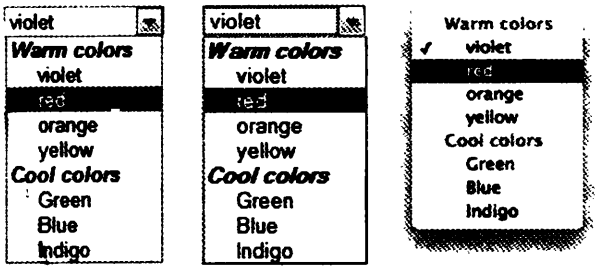


Рис. 13.2. Элемент управления select с членами optgroup, пказанный в IE, Firefox и Safari

На сопутствующем сайте этой книги вы можете найти набор тестов, более подробно демонстрирующий поведение элемента `select`.

Создание прототипа и макета

Если форма разработана на функциональном уровне и реализована, то следующим шагом должно стать создание макета формы, чтобы можно было провести ее тестирование в качестве прототипа.

Основы прототипирования

Прототип веб-приложения выполняет ту же функцию, что и любой другой результат работы инженера: помогает удостовериться в том, что продукт работает так, как и было задумано его разработчиками. Смысл создания прототипа приложения заключается в очистке исполняемого кода от ошибок, однако создание прототипов — это еще и возможность убедиться в удобстве пользовательского интерфейса.

Тестирование прототипа должно дать ответы на следующие вопросы, касающиеся пользователей.

- Достаточно ли легко целевой аудитории пользоваться формами приложения?
- Не приводит ли черновой дизайн какой-либо формы к типичным ошибкам пользователей?
- Содержит ли форма поля, которые пользователи регулярно игнорируют, или поля, которым пользователи уделяют слишком много внимания?
- Можно ли найти такие аспекты визуального оформления и верстки, которые требуют слишком больших усилий, несравнимых с пользой, которую в итоге можно получить?

Вся информация, собранная в ходе тестирования прототипа, может повлиять на взаимодействие пользователя с приложением, однако в этой главе рассматриваются только те аспекты, которые относятся к разметке и CSS.

Дизайнерам следует избегать следующих ситуаций.

- Взаимосвязь между полями формы и соответствующими элементами `label` неясна или неадекватна.
- Поля расположены в порядке, который плохо отражает приоритеты пользователя.
- Инструкции и предупреждения либо плохо продуманы, либо неудачно написаны, либо недостаточно подробны.
- Специфические требования дизайна приводят к грубым ошибкам отображения.

Для того чтобы форма могла служить эффективным прототипом, она должна отвечать следующим презентационным требованиям.

- Макет формы должен соответствовать условиям, определяемым каркасом и составной структурой.
- Контраст цветов фона и переднего плана должен быть адекватным, однако излишняя проработка не нужна.
- Внешний вид инструкций и подсказок, касающихся вводимых данных, должен быть максимально близким к тому виду, какой они должны иметь на разрабатываемом сайте.
- Если в приложении используется Ajax, он также должен был внедрен в прототипах как минимум в отношении самых важных аспектов. Кроме того, дизайнер обязан обеспечить соответствие выходных данных Ajax требованиям, описанным в данной главе.

Некоторые базовые дизайнерские шаблоны для макетов форм описаны в следующем разделе.

Дизайнерские шаблоны, исходные стили и макеты форм

По умолчанию, элементы управления формы подчиняются модели визуального форматирования inline-block, а ярлыки подчиняются модели inline. Поскольку этим элементам во многих случаях присваиваются значения `float`, а наличие в коде пробелов меняет вывод элементов inline-block, с этими элементами, как правило, намного легче работать, когда им присвоено значение `display`, равное `block`. Если принимать во внимание прочие элементы, то консервативные правила «обнуления» стилей для форм будут выглядеть примерно так:

```
form, form ul, form li,
    fieldset, textarea { margin: 0; padding: 0; }
label, input, select,
textarea, form li span { display: block; }
      form ul, form li { list-style-type: none; }
               form li { clear: both; height: 1%; overflow: auto; }
              fieldset { border: 0; }
```

Шаблоны макетов, рекомендуемые для веб-форм, перечислены ниже в порядке приоритетности. Каждый шаблон снабжен образцами стилей из тех шаблонов разметки, которые обсуждались в разделе «Оценка и структура» на с. 275.

В приведенных ниже примерах стилей значения `width` и значения блоков были выбраны исключительно для демонстрационных целей. Обратите внимание, что значения полей, как правило, дополняют значения `width` соответствующих элементов `label`.

- *Ярлыки слева от полей.*

```
          label, form li span { float: left; width: 9em;
                                padding-right: 1em; }
      input, select, textarea { float: left; margin-left: 10em; }
fieldset label, fieldset input { float: left; }
                fieldset label { clear: left; width: 4.25em;
                                 padding-right:.75em; }
                fieldset input { margin-left: 5em; margin-right: 1.5em; }
```

- *То же, что и в предыдущем пункте; в дополнение ярлыки и поля выровнены относительно общей границы.*

```
          label, form li span { text-align: right; }
```

- *Поля под ярлыками.*

```
                      form li { clear: both; height: auto !important;
                                overflow: visible !important; }
fieldset label, fieldset input { float: left; }
                fieldset label { clear: left; width: 4.25em; padding-
                                 right: .75em; }
                fieldset input { margin-left: 5em; margin-right: 1.5em; }
```

- *Ярлыки справа от полей.*

  Используйте те же стили, что применялись в первом примере, только измените значения полей и `float` на противоположные.

- *Несколько колонок пар ярлыков и элементов управления.*

  Поделите ваши пары ярлыков и элементов управления на нужное количество отдельных списков и используйте описанные ранее свойства для получения нужного макета для каждой колонки. Для более приемлемых результатов добавьте к исходному стилю еще несколько правил:

```
  form { height: 1%; overflow: auto; }
form ul { float: left; width: 45%; margin-right: 5%; }
```

Если вы попробуете применить эти примеры в рабочем прототипе, то обнаружите, что относительно сетки все далеко не так гладко, как хотелось бы — особенно это касается элементов управления `input type="radio"` и `input`

type="checkbox". Частично это связано с тенденцией большинства браузеров возлагать задачу отображения элементов управления формы на библиотеку пользовательского интерфейса, принадлежащую базовой операционной системе. Последствия этого рассматриваются в главе 14.

Вторая, более понятная причина кроется в иерархическом расположении стилей форм. Для того чтобы повлиять на размеры элементов управления формы, вам придется установить отдельные значения width и height для каждой формы. Кроме того, вам понадобится установить для каждой формы свое значение font-size, если вы хотите воспользоваться основанным на em подходом к элементарным сеткам, описанным в разделе «Типы верстки и сетка» на с. 131.

На практике разработка, в которой необходимо реализовать готовый к работе макет формы, потребует множества улучшений и тестов. Вам придется создавать правила, использующие значения class, настраивать множество значений блоков и постоянно добавлять мелкие исправления. Особенно сложные макеты форм могут потребовать добавления мелких поправок, относящихся к отдельным полям, например:

```
form#supportTicket li#severity input { padding-top: .333em; }
```

В данном примере определить функцию полей можно по соответствующим селекторам, — это форма запроса на подтверждение поддержки, содержащая ряд элементов управления, которые дают возможность пользователю определять важность проблемы, скорее всего, ряд элементов управления типа input type="radio" (при условии, что в селекторе используется input, а не select).

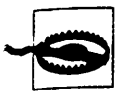
Если обсуждать каждую потенциальную проблему, связанную с макетом формы, в результате книга получится слишком длинной. Дополнительные наборы тестов, правила библиотек и советы по этой теме вы можете найти на сопутствующем сайте этой книги.

Группировка элементов управления по внешнему виду

С макетами форм связан ряд трудностей, и из них чаще других встречаются следующие.

- Элементы input type="text", textarea и select, ответственные за вводимую информацию разного размера, к примеру почтовые адреса или индексы.
- Элементы input type="submit" в конце формы.
- Правила типа input { border: 1px solid rgb(0,0,0); ... }, которые должны быть «обнулены» для средств управления input type="checkbox" и input type="radio".
- Не имеющие ярлыков (или имеющие странные ярлыки) элементы управления, которые располагаются группами: даты, шкалы оценок и телефонные номера, поделенные на составные части.

- Исходные значения полей для элементов управления `input type="checkbox"` и `input type="radio"`.



Постоянно отстающая поддержка селекторов в Internet Explorer, в особенности селекторов атрибутов и родственных элементов, а также селекторов непосредственных дочерних элементов, является одной из отрицательных сторон. Особенно сильно это сказывается на результатах авторской разработки таблицы стилей для представления формы.

В приведенных выше примерах лучше всего выбирать имена селекторов согласно цели или длине данных, которые должны содержаться в конкретном поле.

- *Для размеров текстовых полей или полей* `select` *используйте*:
  - `.short`
  - `.medium`
  - `.long`
  - `.small`
  - `.large`
- *Правильно выбирайте размеры содержимого; слишком узкие поля могут сделать содержимое неясным, в некоторых случаях — безвозвратно. Для кнопки отправки формы и элементов управления изображениями используйте*:
  - `.terminalButton`
  - `.submitButton`
- *Чтобы отличать друг от друга текст и переключаемые курсором мыши элементы* `input`, *присвойте соответствующий элемент* `class` *наиболее реже повторяющемуся из типов*:
  - `.textControl`
  - `.buttonControl`
  - `.boolControl`
  - `.enumControl`
- *Что касается «hidden»*:

  Скрывать или опускать ярлыки — это плохая идея, но если они не должны быть видны обычным пользователям, задействуйте отрицательные значения `left` вместе с `position: absolute;`, чтобы убрать их из зоны видимости пользователей. Для того чтобы пометить поля, которые хранят несколько произвольных вариантов выбора подряд (к примеру, после элемента `legend`, который просит пользователя перечислить три любимые вещи), задействуйте значения `label`, расположенные по порядку (1-й, 2-й, 3-й...), не учитывая видимость элементов `label` в пределах интерфейса формы.

Во многих описанных случаях встречается элемент `fieldset`; используйте его с умом, как в разметке, так и в таблицах стилей.

Обязательные для заполнения поля и другие ограничения при подтверждении

Некоторые поля формы обязательно *должны быть* заполнены, что вполне понятно: в качестве простейших примеров можно привести поля поиска и имени пользователя.

Однако в большинстве ситуаций необходимость заполнения пользователем какого-либо поля перед подтверждением очевидна только для разработчика формы, причем даже в простых случаях: например, в случае формы для сбора адресов электронной почты (и названных так же). Если необходимость в заполнении поля действительно существует, то элементы формы должны быть помечены как «обязательные», а в логике сайта присутствовать соответствующие пометки.

В других случаях данные, вводимые пользователем (например, телефонные номера или адреса электронной почты), подпадают под другие ограничения. Эти ограничения реализуются при помощи проверок значений строк и регулярных выражений.

В задачу дизайнеров в таких случаях входит следующее: обязательные поля должны быть надлежащим образом отмечены, ограничения — прописаны ясно и четко, а ошибки — адекватно оформлены (желательно с подробными описаниями ошибок).

Определение обязательных для заполнения полей

Рассмотрим одну из описанных ранее на рис. 13.1 пар ярлык/элемент управления, которая задает общую схему элементов контейнера `li`, элементов `fieldset`, `label` и элементов управления формой. Если при разработке макета формы вы используете стили, предложенные в этой главе, то вам, скорее всего, необходимо будет добавить приблизительно такие правила:

```
.required { position: relative; padding-right: 3.3em; }
.required label span.warning { position: absolute; right: 0;
                               width: auto; color: rgb(192.0.0); }
```

Такие стили соответствуют следующей разметке:

```
...
<li id="ZIPControl" class="required">
    <label for="zipcode">ZIP Code<span class="warning"> [required]
    </span>:</label>
    <input type="text" name="zip" id="zipcode" length="5" value="" />
</li>
```

При их совместном использовании результат будет таким: ярлык «[обязательное заполнение]» будет находиться у правой границы пары `label`/элемент ввода, а его цвет будет задаваться в значениях H/S/B как 0°/100%/63% (близко к темно-малиновому). Ярлык оформляется таким образом, что то же самое правило может быть применено ко *всем* обязательным полям документа без риска появления разрывов, при условии, что элементы `li` и `fieldset` имеют значение `width`, равное `auto`.

Недостаток этого метода заключается в том, что он может быть не замечен некоторыми экранными лупами, особенно той, что включена во все версии Windows. Чтобы при разработке учесть эту особенность, метки «обязательного заполнения» должны находиться вплотную к левой границе соответствующего поля, либо над ним, либо под ним. Последствия таких изменений легко оценить, хотя, скорее всего, в результате сетка макета будет иметь гораздо больше пустого вертикального пространства, чем представленная здесь.

При разработке многих типов форм для добавления меток обязательного заполнения может быть достаточно применения обычной модели визуального форматирования, но в некоторых случаях (например, описанных в этой главе) полезно помнить, что вы всегда можете воспользоваться контекстом расположения элементов (см. раздел «Свойства позиционирования в CSS» на с. 121).

Поиск и распознавание пользовательских ошибок ввода

Следующий шаг на пути подтверждения правильности введенной информации — это проверка ее на наличие ошибок, задача, которую лучше всего решать одновременно со стороны сервера и со стороны клиента. Последовательность действий приблизительно такова.

1. Информация, введенная пользователем, проходит проверку относительно объектов `.length`, `indexOf()`, `parseInt()` и `RegExp` со стороны клиента.
2. Если JavaScript разрешен, то введенная информация с ошибками отмечается скриптом валидации; существующему значению соответствующего контейнера `li` или `fieldset` присваивается нужное значение `class`, например `error`; также туда добавляется содержимое для распознавания причин ошибки. Если введенная информация соответствует требованиям, то отпадает необходимость в дополнительном обмене сообщениями между клиентом и сервером.
3. Сервер получает подтверждение, и оно проверяется на наличие внедрений вредоносных выражений SQL, исполняемого кода и разметки.
4. Обработанная введенная информация проверяется логикой сервера таким же образом, как и в браузере; если обнаруживаются ошибки, то отклик подтверждения включает в себя ту же переработанную разметку, которая должна была быть выполнена браузером.

5. Шаги 1, 3 и 4 повторяются до тех пор, пока введенная информация не очистится от ошибок.

Описание ошибки должно находиться либо вверху формы, либо в начале элемента `li`, связанного с элементом управления, содержащим ошибочные введенные данные. Лучше всего использовать *обе* позиции или только вторую; из-за сообщений об ошибках, расположенных только вверху формы, пользователям придется прокручивать форму вверх-вниз от ошибок к сообщениям и обратно — если они в таких условиях вообще смогут распознать ошибку.

Для нормальной работы сообщений об ошибках вы можете попробовать применить следующие правила, чтобы дополнить стили `.required span.warning`, описанные ранее:

```
.error .warning { display: block; background-color: rgb(160,0,0);
                  color: rgb(255,255,255); font-weight: bold; }
   .error input { border: 1px solid rgb(160,0,0); background-color:
rgb(255,160,160);
}
```

В дополнение к этим стилям (приведенным здесь исключительно для примера) могут быть добавлены значения `clear`, `width` и `margin`, чтобы уведомление об ошибке было выровнено по горизонтали с нужным элементом управления вводом или `fieldset`.

Атрибуты disabled и readonly

Если вы когда-нибудь устанавливали операционную систему, особенно такую, которой требуется лицензионный ключ, то вам знакомо понятие отключенных элементов управления. При установке программ конфигурация осуществляется в несколько этапов, каждый из которых должен быть завершен перед началом следующего, поэтому кнопка **Далее** в конце каждого этапа выкрашена в серый цвет и не может быть нажата до тех пор, пока текущий этап не завершится.

Вы можете добиться того же эффекта и в случае с формами, применяя либо атрибут `disabled`, либо (реже) атрибут `readonly`. Они поддерживаются разными элементами форм, что иллюстрирует табл. 13.3.

Таблица 13.3. Обзор поддержки атрибутов disabled и readonly

Атрибут	button	input	option	Select	textarea
`disabled`	✓	✓	✓	✓	✓
`readonly`		✓			✓

Хотя использование этих атрибутов может привести к достаточно элегантным дизайнерским решениям — по большей части благодаря возможности запрета на ввод значений до разрешения зависимостей — эти значения могут быть изменены только с помощью JavaScript. По этой причине избегайте применения

таких атрибутов в отсутствие альтернативных решений проблемы соблюдения зависимостей в информации, вводимой пользователем.

Если вы захотите затемнить поля, которые должны быть заполнены только после завершения других операций внутри формы, отведите под них место, которое они будут занимать, когда будут полностью видны, и либо загружайте их при помощи Ajax, либо используйте вместо `display` другие свойства CSS. Надежда исключительно на `display` в плане обеспечения видимости полей не оправдывает себя и может ухудшить качество работы формы для пользователей с ограниченными возможностями.

Создание форм, доступных для всех

Поскольку формы — это «альфа и омега» достижения коммерческих целей многих сайтов, важно иметь в виду вероятность того, что существенное число посетителей может иметь физические или умственные отклонения, что затрудняет им работу в Сети. Проблемы, связанные с дизайном веб-сайтов для таких людей, могут быть сгруппированы по нескольким основным категориям, каждую из которых можно проиллюстрировать примерами.

- *Нарушения опорно-двигательного аппарата*

  Если пользователь ограничен в движениях, то, соответственно, ограничены и возможности использования им мыши, клавиатуры или и того, и другого.

  - Сломанные руки и/или пальцы.
  - Хронический тендинит (дисфункция ткани сухожилия) или хронические растяжения.
  - Периферийные нервные заболевания.
  - Параплегия или паралич четырех конечностей.

- *Нарушения зрения*

  Пользовательский интерфейс персональных компьютеров и мобильных устройств дает возможность пользователю совершать ответные действия при помощи зрения, слуха и осязания. Из этих трех чувств самым важным для разработки веб-сайтов является зрение, и особенно это касается тех веб-сайтов, которые для достижения своих целей используют формы. Если пользователи не могут видеть форму или данные, которые они в нее вводят, то их возможности использования таких сайтов резко уменьшаются. К этому приводят:

  - близорукость;
  - астигматизм;
  - возрастное снижение остроты зрения;
  - дегенерация желтого пятна;
  - глаукома;

- ди- и монохроматичность (цветовая слепота);
- полная слепота.

- *Нарушения функций мозга*

  Осмысление значения слов и изображений, включая контент Сети, полностью является функцией мозга и сознания. Пользователь, который не может сконцентрироваться на сайте или быстро воспринять смысл, вряд ли задержится на нем надолго — отсюда вытекает требование краткости.

  - Синдром дефицита внимания.
  - Дизлексия.
  - Нарушение обработки слуховой информации.
  - Симптоматическая травма мозга.
  - Острый дефицит витамина B.

Существует множество других условий, влияющих на возможность человека пользоваться Сетью. Нехватка времени, к примеру, не относится к нарушениям или болезням, но является привычной проблемой.

Почти наверняка вы и ваши знакомые имеют представление хотя бы о нескольких перечисленных здесь нарушениях. У меня самого есть большой личный опыт, относящийся трем из них, причем с двумя я вынужден справляться каждый день — и при этом все равно пользуюсь Интернетом.

Так-то.

Реализация форм с учетом доступности

Вы можете сделать свой сайт удобным для использования практически любыми посетителями, следуя достаточно простым правилам.

- *В процессе разработки и тестирования проекта сверяйтесь с Руководством по созданию доступного веб-контента (Web Content Accessibility Guidelines, WCAG).*

  Это руководство скорее направлено на приобретение привычки: со временем вы будете все лучше и лучше знать методы, описанные в рекомендациях, и возьмете за правило реализовывать их, прикладывая для этого гораздо меньше усилий.

- *Никогда не используйте только один способ указать на контекст или подтвердить информацию, введенную пользователем.*

  Когда вы используете текст, не забывайте и о визуальной подсказке (и наоборот). Обеспечьте для пользователя возможность взаимодействия с сайтом посредством только мыши или только клавиатуры, в то же время оставляя возможность предпочтения одного устройства другому.

- *Всегда располагайте разметку и содержимое в правильном порядке, чтобы они читались связно и не полагались на поддержку CSS.*

  Для этого самое важное всегда следует помещать в начале.

- *Если вы хотите, чтобы пользователь сделал нечто более сложное, чем «иди куда хочешь и введи ту информацию, которая придет тебе на ум», убедитесь, что ваши требования ясны и просты для восприятия.*

  Инструкции должны быть лаконичными. Уважайте умственные способности посетителей и не разговаривайте с ними «сверху вниз». Однако зачастую взаимодействие пользователя с сайтом может ухудшиться, если вы будете считать, что посетители разделяют и глубину вашего знания, и вашу точку зрения.

- *Избегайте применения* `display: none` *и* `visibility: hidden` *в ходе работы, если только вы не хотите, чтобы пользователи с ограниченными возможностями никогда не увидели связанное с этими свойствами содержимое.*

  Так как платформы вспомогательных технологий при работе опираются на тот факт, что все доступные стили были написаны для экранных средств, то такие пары свойств/значений предполагают, что содержимое должно быть проигнорировано.

- *Постарайтесь, чтобы пользователи могли прочесть то, что они вводят, не прилагая к этому усилий.*

  Если не учесть это требование при оформлении, то пользователю придется совершать много необязательных действий вроде визуального поиска, прокрутки и передвижения курсора — действий, которые пользователям с двигательными или визуальными нарушениями даются непросто.

- *Порядок элементов ввода делает ваш сайт более понятным и предсказуемым; оформляйте и реализуйте макеты соответствующим образом.*

  Используйте одни и те же значения границ и размеров для элементов управления. Содержимое `label` должно быть коротким, но в то же время не должно утрачивать смысл. Пары `label`/элемент ввода располагайте в столбец таким образом, чтобы каждый элемент ввода следовал в том же порядке, в котором эти данные идут в документе.

- *Если вы используете Ajax для переключения значений* `display`*, реализуйте поддержку Стандарта доступности активных Интернет-приложений W3C (W3C Web Accessibility Initiative Accessible Rich Internet Applications, WAI-ARIA) при помощи фреймворка JavaScript.*

  Более подробную информацию по поводу WAI-ARIA вы можете получить на сопутствующем сайте этой книги.

- *При написании разметки следуйте скорее не букве, но духу спецификаций HTML и CSS.*

  Здесь имеется в виду, что для размещения содержимого страницы никогда не используйте элементы `table`, а изображения всегда снабжайте атрибутом `alt`. Используйте элементы, предназначенные для вашей задачи, если они поддерживаются, а если нет, то используйте `id` и `class` для передачи содержимого. Что более важно, структурируйте *разметку в целом* именно ради содержимого, а не ради представления.

Поддержка перемещения по форме при помощи клавиатуры

Выбрать элемент формы и ссылку можно при помощи одной только клавиши Tab; клавишу Enter можно использовать для работы со ссылками и для подтверждения форм. Эти возможности реализуются на структурном уровне при помощи атрибута `tabindex`, который можно использовать для изменения порядка выбора элементов при нажатии Tab.

В обычных обстоятельствах нажатие клавиши Tab перемещает фокус с одного элемента на другой в исходном порядке. Когда атрибутам `tabindex` соответствующих элементов присваиваются значения, порядок выбора меняется таким образом, что элементы с `tabindex` располагаются в порядке возрастания, а за ними следуют в исходном порядке элементы, которым значения атрибута не присвоены. Если двум или более элементам присвоено одно значение `tabindex`, то они выбираются в соответствии с исходным порядком.

Однако у атрибута `tabindex` есть два недостатка: во-первых, когда страница длинная, а `tabindex` внедрен неудачно, ползунок прокрутки может перемещаться на очень большие расстояния, что приводит к дезориентации. Во-вторых, в ходе обновления и доработки сайта значения `tabindex` на практике обновляются очень редко.

С другой стороны, странице с правильным порядком элементов ввода может пойти на пользу добавление `tabindex="0"` важным элементам, которые не активизируются в начале посещения страницы (а бывает, что и никогда). Кроме того, вы можете присваивать отрицательные значения `tabindex` тем элементам, которые вы хотите исключить из порядка табуляции.

Атрибут `accesskey` позволяет пользователям обойти порядок табуляции. Атрибут `accesskey` может принимать значения в виде отдельных символов (к примеру, любого буквенного печатного символа). Эти значения активизируются, когда пользователь нажимает соответствующий символ вместе с одной из вспомогательных клавиш, перечисленных в табл. 13.4.

Таблица 13.4. Активизация значений accesskey в распространенных браузерах

Браузер	Комбинация клавиш
Firefox/Windows	Alt+Shift+?
Internet Explorer	Alt+?
Safari and Firefox/Mac	Ctrl+?

Две платформы, описанные в таблице, обладают и специальными возможностями. Первая из них — это среда OS X, где клавишей, используемой для переключения на клавиши доступа, является скорее Ctrl, а не Cmd. Такое назначение имеет смысл и уменьшает вероятность конфликтов, так как Cmd, как правило, используется для выполнения команд приложений с клавиатуры

Macintosh (а клавиша Ctrl за пределами сред командной строки игнорируется).

Ситуация с реализацией `accesskey` в Internet Explorer далеко не так хороша. Приложения Windows активизируют главные меню приложений при нажатии клавиши Alt, той же клавиши, которая используется для переключения `accesskey` — и поэтому Alt+F активизирует меню Файл (File), Alt+E активизирует меню Правка (Edit), и так далее. Важно помнить о том, что, если конфликтующее значение используется атрибутом `accesskey`, то соответствующий переключатель интерфейса *приложения*, как правило, уступает приоритет и не функционирует.

Учитывая перевес Internet Explorer на рынке, символы F, E, V, A, T и H в качестве значений атрибута `accesskey` лучше не использовать. Подобные исключения могут иметь место для браузеров под Windows в целом по причине применения некоторых панелей инструментов и расширений.

Свойства форм в HTML5

HTML5 был нацелен на внедрение новых свойств HTML-форм. Возможно, это та область, в которой HTML5 несет наибольшие изменения, однако они все еще находятся в процессе разработки. В этом разделе вы найдете обзор свойств форм и подробности, касающиеся свойств, которые имеют неплохую возможность существенно повлиять на конечных потребителей, несмотря на свою сравнительно малую известность.

Новые типы вводимых данных

Формы в HTML5 направлены на улучшение форм HTML 4 посредством добавления следующих 13 новых типов элементов управления `input`:

- `datetime` (ввод глобальной даты и времени);
- `datetime-local` (ввод локальной даты и времени);
- `date` (дата);
- `month` (ввод года и месяца);
- `time` (ввод времени);
- `week` (ввод года и недели);
- `number` (ввод числа);
- `range` (ввод неопределенного числа);
- `email` (ввод адреса электронной почты);
- `url` (ввод URL);
- `search` (поле поиска);
- `tel` (ввод телефонного номера);
- `color` (средство выбора цвета).

Эти новые типы облегчают жизнь разработчикам: в случае, если с новым типом связан специальный пользовательский интерфейс, этот интерфейс обрабатывается непосредственно браузером, и разработчику не приходится писать свой собственный JavaScript-код или использовать библиотеки JavaScript. Возьмем для примера ввод цвета: браузер создает палитру цветов, которая дает возможность пользователям выбирать цвет, просто подведя курсор к нужному цвету и щелкнув мышью. В случае с вводом даты браузер создает возможность выбора по календарю. Предоставление пользователю графического интерфейса управления для таких типов вводимых данных также подразумевает наличие встроенного (в отличие от дополнительного) механизма, который следит за тем, чтобы пользователи вводили значения в правильном диапазоне.

Для конечных пользователей имеет большое значение также наличие автоматической системы валидации в браузере — и для того, чтобы пользователь не мог ввести неверное значение, и для того, чтобы быстро оповестить его, если он все-таки его введет. (В реализациях форм с помощью HTML5 валидация в браузере включена по умолчанию и может быть отключена либо через меню браузера, либо разработчиком страницы, который может явно присвоить целой форме или конкретному элементу управления атрибут `formnovalidate`.)

Атрибут required

Атрибут `required` — это одна из новых возможностей HTML5, направленная на выполнение простой, но очень важной цели. Если в определенной форме настроить `required` для элемента `input` или `textarea`, то это будет значить, что пользователям будет необходимо заполнить эти поля для того, чтобы отправить форму.

Если для правильного функционирования формы нужно, чтобы определенные поля не содержали пустых значений (к примеру, поле имени в форме создания учетной записи), то необходимо проверить содержимое полей формы и оповестить пользователя о том, что он пытается отправить форму с пустыми полями. Такая проверка сейчас может быть реализована двумя способами. Первый — совершить проверку со стороны сервера после подтверждения формы пользователем; этот вариант условно оптимален, поскольку требует дополнительного сетевого обмена данными и времени ожидания пользователя. Второй, более удачный способ — совершить проверку со стороны браузера перед тем, как он отошлет форму на сервер. Однако недостаток всего этого состоит в том, что разработчикам приходится совершать проверку при помощи JavaScript-кода или библиотеки JavaScript. И здесь в игру вступает атрибут `required`. Преимущество атрибута `required` для вас как для разработчика состоит в том, что он освобождает вас от необходимости использования JavaScript-кода для проверки заполнения обязательных полей формы со стороны браузера. Вместо этого вы просто присваиваете нужным полям атрибут `required`, и браузер автоматически делает проверку.

Использование атрибута `required` помогает пользователям сэкономить время и нервы; кроме того, этот атрибут обеспечивает стабильное взаимодействие пользователя с формами, у которых отсутствуют значения в обязательных полях. Он также позволяет пользователям получать локализованные сообщения «не заполнено обязательное поле формы» от их браузеров на предпочитаемом ими языке (вместо того, чтобы читать сообщения на языке, который конкретное сетевое приложение использует для своих сообщений об ошибках).

14 Отрицательные стороны

Интернет — чудо современной цивилизации. Наша эпоха отличается от любой другой тем, что в наши дни стало возможным мгновенно публиковать информацию без чьего-либо разрешения, и Интернет является ключевым инструментом, который помог нам добиться этого.

К сожалению, Сеть, как и любая другая система, имеет свои недостатки, технические ограничения и слабые места. Данная глава содержит описания некоторых отрицательных сторон Сети, а также краткие пояснения о том, как их следует использовать (если это вообще стоит делать).

Главу завершает раздел «Ужасные стороны» на с. 324: в нем рассказывается о том, чего стоит избегать во всех ситуациях (за редким исключением).

Поразительные свойства Internet Explorer (особенно IE 6)

Проблемы с Internet Explorer — результат скорее не недостатков HTML и CSS, а их реализации, однако эти недостатки в реализации имеют далеко идущие последствия. В 2001 году, когда был выпущен Internet Explorer 6, это был *решительно* лучший браузер из всех существующих на тот момент.

Почему?

- Он обеспечивал более широкую поддержку веб-стандартов, чем любой из предшественников, а также лучшую на тот момент поддержку DOM API.
- Графической технологии под кодовым названием Gecko, лежащей в основе Firefox, предстояло пребывать в стадии разработки еще более двух лет.
- Реализация CSS в Internet Explorer 5 для Macintosh соответствовала всем стандартам, однако этот браузер использовал свой собственный графический движок, а его распространению препятствовали небольшая доля рынка и отсутствие внутренней поддержки.
- Netscape тихо умирал... потому что все самое лучшее еще ожидало своего воплощения в Netscape 4.

- О Safari вообще не было слышно ни слова, и своего расцвета он ждал еще четыре года. Кроме того, Safari потребовался еще один год на то, чтобы приобрести функциональность под стать своим современникам, не забывая о режимах отображения и производительности.

Другими словами, когда Internet Explorer 6 был выпущен, он оказался сам по себе весьма убедителен и рынок был открыт для него.

Компания Microsoft, изначальным стимулом для которой являлось господство Netscape в сетевом пространстве, проделала невероятный объем работы, выпуская потрясающие браузеры четыре года подряд, но в итоге она превратилась в дремлющего зайца из басни Эзопа.

Компании Apple, Inc. и Mozilla Foundation слились воедино в роли медлительной, но неутомимой черепахи: они продолжали кропотливо разрабатывать и выпускать свои браузеры, пока в 2005 году не стало ясно, что Internet Explorer устарел.

Войны браузеров: версия 2.0

В середине 2005 года правительство США порекомендовало пользователям избегать работы в Internet Explorer по соображениям безопасности. Приверженцы открытого программного обеспечения и критики монокультуры принялись с энтузиазмом восхвалять достоинства альтернативных платформ, и в результате этого доля Internet Explorer на рынке уменьшалась месяц за месяцем.

Наконец, стряхнув оцепенение, компания Microsoft выпустила две новые версии Internet Explorer, существенно превосходящие продукт 2001 года, но тот динозавр — которому на момент написания этих строк уже исполнилось восемь лет — все еще держится на ногах. Помимо простых сетевых тенденций есть несколько причин, объясняющих подобную стойкость IE 6.

- В момент первоначального выпуска строка меню Internet Explorer 7 была по умолчанию спрятана, что смутило многих обычных пользователей и заставило их отказаться от этого браузера.
- Так как студии информационных технологий для предприятий (важнейшая статья доходов Microsoft) рассматриваются как учетные отделы, несмотря на всю эффективность их работы, всем их благим начинаниям мешает принцип «не сломано — не чини». По этой причине обновление Internet Explorer многими считается ненужным и излишне трудным делом.
- С момента своего выпуска Windows XP, в поставочный комплект которой входит Internet Explorer 6, приобрела большее доверие пользователей по сравнению с более новыми операционными системами от Microsoft. Windows 7 присутствует на рынке с октября 2009 года и, возможно, со временем превзойдет и заменит Windows XP, однако еще некоторое время она будет занимать меньшую долю рынка, чем ее предшественник.
- В Internet Explorer 6 достаточно просто устанавливать фильтры для CSS, тем самым скрывая многие недостатки продукта от простых пользователей — во всяком случае, до определенного момента.

- В период между 2001 и 2005 годами поставщики услуг, в частности, несколько банков, запустили ориентированные на клиентов сайты, нацеленные на использование в Internet Explorer под Windows, либо воспринятые именно так, поскольку для их разработчиков это был наилучший из известных им вариантов. Такое обилие сервисов, ориентированных на конкретную платформу, породило волну страха, неуверенности и сомнений по поводу надежности альтернативных браузеров.
- Если оставить в стороне вопросы безопасности, фильтров и ориентированности на конкретную платформу, то большинству пользователей просто-напросто все равно, какой браузер использовать, если он достаточно удобен и работает.

Из всего вышесказанного можно сделать вывод, что Internet Explorer 6 рассматривается в качестве отрицательного явления не потому, что это бремя, мешающее пользователю, а потому что он *вездесущ*. Если он не поддерживает какую-либо новую технологию, то все пользователи оказываются отрезанными от доступа к этой технологии, поскольку большинство коммерческих заказчиков не будут тратить средства на продукты, которыми сможет воспользоваться лишь небольшая часть предполагаемой аудитории.

Другими словами, если Internet Explorer и поддерживает некоторые специфические возможности, то посредством методов настолько запутанных, что разработчики считают их реализацию слишком обременительной.

Представленные ниже разделы включают в себя обзор недостатков Internet Explorer. Также описаны пути их обхода в тех случаях, когда они существуют.

Слабая поддержка селекторов (или ее отсутствие)

В таблице 14.1 описаны недостатки поддержки селекторов в Internet Explorer.

Если приоритетом для IE 6 является поддержка высшего уровня, то в общем случае лучше не использовать эти селекторы. В качестве альтернативы их функции могут быть дублированы при помощи JavaScript, но это решение требует написания кода, который позволяет достичь нужного результата с помощью «грубой силы».

Вывод

Используйте данные селекторы, если потребуется, но не забудьте сообщить заинтересованным сторонам и дизайнерам о том, чего следует в таком случае ожидать. Замените селекторы `:first-child`, `:last-child` и > на селекторы `class` там, где это необходимо.

Чтобы отследить и разрешить проблемы с отображением страниц, вызванные дефектами Internet Explorer, обратитесь к следующим источникам: «Расположение важнее всего» («Position Is Everything», http://www.positioniseverything.net/); база знаний и архив сайта http://css-discuss.incutio.com/.

Таблица 14.1. Важные недостатки поддержки селекторов в Internet Explorer

Селектор	Наиболее ранняя версия, обладающая поддержкой	Примечания
> (прямое наследование)	7	
[foo] (атрибут)	8	Простые селекторы атрибутов работают в IE 8, однако попытки дальнейшего сужения области рассмотрения результатов не принесли
:first-child, :last-child	8	
:nth-child()	Нет данных	
:before	8	Счетчики не учитываются
:after	Нет данных	
.foo.bar	7	IE 6 воспринимает этот переключатель как .bar
:hover	3	Псевдокласс :hover применялся только к элементам, отличающимся от a, начиная с версии 7; в случае :active — начиная с версии 8

Свойство hasLayout

hasLayout — это странное свойство, присущее только Internet Explorer, которое принимает значение true или false при поступлении запроса посредством DOM API. Если значение равно true, то характеристики представления соответствующего элемента определяются где-то в пределах логики графического движка, а не исходя из результата запроса. Последний вариант приводит к многочисленным ошибкам, с которыми сталкиваются дизайнеры при тестировании своих работ в IE 6.

Если вы хотите исправить ошибку, присвоив hasLayout значение true, то вам необходимо удостовериться, что этот элемент принимает *определенные* размеры по крайней мере по одной из осей. Этого результата можно достигнуть, задав для свойств position и height значения, отличные от static и auto, соответственно. Этот метод называется «приемом Холли» («Holly Hack») по имени Холли Бергевин — разработчицы, впервые популяризовавшей его.

Также присвоения свойству hasLayout значения true можно добиться, применив zoom:1. Этому способу отдают предпочтение многие дизайнеры, так как он не приводит к проблемам наложения, характерным для позиционированных элементов (см. раздел «Стек» на с. 125).

Этих проблем в большинстве случаев можно избежать при помощи гибких макетов и сеток (см. раздел «Типы верстки и сетка» на с. 131), используемых в паре с заданным выше в каскаде значением font-size, установленным в px. С другой стороны, такое решение может поставить в крайне невыгодное положение пользователей Internet Explorer 6 с нарушениями зрения, так как

размеры шрифта в меню **Вид (View)** IE 6 неприменимы к тексту, размер которого выражен в px.

Вывод

Подобные проблемы, увы, неизбежны. Разбирайтесь с ними теми способами, какие посчитаете нужными, лучше — при помощи условной таблицы стилей, нацеленной на работу в Internet Explorer, в комбинации со специфичным фильтром * html.

Удвоение отступа

Рассмотрим следующий пример.

```
#someDiv { display: block: float: left: width: 20em: margin-left: 10em: }
```

В данном примере значения отступа и `float` соответствуют одному и тому же краю исходного элемента, и Internet Explorer необъяснимым образом *удваивает* значение `margin-left` до `20em`, именно потому, что свойства границы и `float` относятся к одному и тому же краю.

Эта проблема наполовину решена W3C в разделе 9.7 спецификации CSS 2.1. Вторая часть решения не особо поддается логике: установите для свойства `display` значение `inline`. Это сработает, так как при имеющихся значениях `width` и `float` значение `display` соответствующего элемента будет равно `block`.

На сайте **positioniseverything.net** указано, что решение данной проблемы принадлежит Стиву Клэсону.

Вывод

Решение проблемы с удвоением границы, хотя и является странным и нелогичным, представляет собой просто малозаметное изменение свойств элемента, чего мы и жаждем от всех обходных путей. Расходитесь. Здесь абсолютно не на что смотреть.

Значения expression()

Вы знали, что JavaScript может быть добавлен в таблицу стилей, и в таком случае Internet Explorer проанализирует этот код?

Если вы зададите объект `expression()` в качестве значения свойства CSS и введете валидное выражение JavaScript в качестве его единственного аргумента, Internet Explorer вычислит это выражение и примет результат вычисления за значение, присвоенное соответствующему свойству. Данная особенность

в высшей степени удобна, но настолько не соответствует духу прогрессивного улучшения, что я со всей доступной мне убедительностью рекомендую вам не использовать ее.

Дополнительное неудобство вызывает тот факт, что в выражениях `expression()` невозможно эффективно ссылаться на узлы документа без использования атрибута `defer` в скрипте, который, в свою очередь, загружает таблицу стилей со значением `expression()`. Это равносильно созданию «вспышки нестилизованного контента».

Вывод

Избегайте подобных вещей насколько это возможно, ради сохранения структурной целостности и безопасности. Или, по крайней мере, старайтесь применять такие решения в минимально возможном объеме. (Этот раздел не относится к «ужасным сторонам» только потому, что в один прекрасный день такое решение может и спасти вас.)

Фильтры и переходы ActiveX

Платформа ActiveX обеспечивает для Internet Explorer доступ к некоторым фильтрам и переходам, в частности к фильтру `Alpha()`, который является аналогом свойств `opacity`/`-moz-opacity` в IE. Этот фильтр также крайне важен для отображения альфа-канала файлов формата PNG в Internet Explorer 6, о чем пойдет речь в следующем разделе.

Чтобы воспользоваться прозрачностью в Internet Explorer, надо сделать следующее:

```
#foo { filter: progid:DXImageTransform.Microsoft.
Alpha(opacity=xxx); }
```

В данном случае `opacity` может принимать целочисленное значение от 0 до 100, но не вещественное значение с плавающей точкой от 0 до 1, как для `opacity` и `-moz-opacity`.

Вывод

Использование данной возможности — и ее производных в CSS3 — лучше ограничить двумя случаями: переходы альфа-канала в скриптах посредством DOM API и ситуации, в которых не получается визуально объединить два пересекающихся фоновых изображения в фон родительского элемента. Во всех других случаях этого приема следует избегать, пока свойство `opacity` не будет поддерживаться *действительно широко*, как это и должно быть согласно документации CSS3.

Поддержка PNG (или ее отсутствие)

В реальности все распространенные версии Internet Explorer *будут* поддерживать файлы формата PNG. Но с IE 6 не все так просто.

При полной глубине цвета PNG-изображения поддерживают 24 бита на пиксел цветовой информации (8 бит на канал, как и для JPEG, Photoshop, веб-цветов и sRGB) и дополнительные 8 бит на пиксел (256 уровней) для прозрачности.

Однако когда Internet Explorer 6 сталкивается с пикселом со значением альфа-канала, отличающимся от `FF` (непрозрачность), вместо ожидаемого результата он выдает серый пиксел.

Для решения данной проблемы была введена «процедурная плоскость» `AlphaImageLoader`. В этом случае она меняет связь строковых изображений с остальным содержимым и требует использования всех хитрых приемов DOM API, эффективных при работе с фоновыми изображениями.

Наилучшее решение проблемы (с точки зрения дизайнера) включает в себя использование файла HTML Component, написанного Ангусом Тернбуллом (http://www.twinhelix.com/). Ссылки на документацию (а также на несколько альтернативных вариантов, включая решения на основе фреймворка JavaScript) можно найти на сопутствующем сайте этой книги.

Вывод

В большинстве случаев обходные пути можно найти при помощи других средств и других форматов сжатия изображения — к счастью, споры насчет интеллектуальной собственности в этой области уже давно не вызывают такого же фурора, как в эпоху выхода Internet Explorer 6. Подождите, пока доля IE 6 на рынке упадет до однозначных чисел в диапазоне от средних до малых, затем используйте формат PNG, и вы увидите, что произойдет, когда вы соберетесь оставить четырехканальное формирование PNG-изображений как есть. (Если так поступят достаточное количество производителей, то заинтересованные лица начнут, наконец, понимать, насколько древним стал Internet Explorer 6.)

Слабая поддержка свойств

Существует несколько комбинаций свойство/значение, которые уже долгое время представляют определенный интерес для опытных дизайнеров.

Говоря проще, очень плохо, что IE 6 не поддерживает эти комбинации так, как должен, ведь сделать для этого надо не так уж и много.

Вывод

Создавайте альтернативные стили — если возможно, без создания дополнительных элементов — это позволит вам добиться нужных результатов на всех платформах без необходимости в избыточном применении фильтров и прочих

интеллектуальных упражнениях. Если в вашем конкретном случае необходима поддержка `position: fixed`, то это значит, что вам просто не повезло.

Проблемы с XHTML и XML

Слабая степень поддержки правильно примененного XHTML вызывает некоторую досаду, так как это сводит на нет удобство практического применения XHTML. Чтобы прикрыть эту досаду иронией, скажу, что компания Microsoft стала вообще первым производителем популярных браузеров, попытавшимся внедрить поддержку XML.

Польза, которую можно извлечь из XHTML, даже в отсутствии должной поддержки, важна для привычного стиля разработчиков. Однако при разработке несложных веб-проектов стоит учитывать тот факт, что пользователи могут не обладать поддержкой всех возможностей XHTML. Если вы хотите пойти на компромисс, можно проанализировать поле `User-Agent` в заголовке запроса и послать в ответ наиболее подходящий `Content-Type`, но такой подход ведет к непредвиденным последствиям (к примеру, отсылке неверного значения `Content-Type` пользователям, работающим в Интернете через прокси-серверы).

Другой недостаток представления содержимого в виде XML состоит в том, что спецификация XML неумолима при обработке неправильно оформленной разметки со стороны пользовательских агентов; если вы поддерживаете работу сайта или хранилища данных, которое в какой-либо степени опирается на стороннее содержимое, то, представляя ваши документы в виде `text/xhtml+xml`, вы явно столкнетесь с проблемами.

Вывод

Идеальная среда для XHTML может привести к самым блестящим результатам — но, увы, Интернет при повседневном использовании такой средой *не* является. Эх. Поэтому используйте XHTML 1.0 и применяйте его с типом `Content-Type`, равным `text/html`, либо просто вернитесь к HTML.

Системное безобразие

Люди приспосабливались к устройству Интернета в целом... до определенного момента. Мир Сети намного шире индивидуальных миров, в которых обитает большинство пользователей — Сеть же и вправду всемирная! — что приводит к некоторым интересным особенностям этой системы.

Слабость шаблонов и сторонний контент

При существовании посреднической рекламы, комментариев в блогах, сообщений в социальных сетях и обилии стороннего контента, который легко может

оказаться и на вашем сайте, вряд ли стоит надеяться, что документы будут выглядеть так же, как в тот момент, когда вы вводили их в разработку.

Говоря проще, не все разделяют ваши намерения.

К сожалению, это проблема, которой нельзя избежать и которую нельзя решить, хотя она и может быть сведена к минимуму при помощи переходных типов документов (Transitional). Иногда подобный мусор все равно будет пробираться на ваши сайты. Смиритесь.

Вывод

Объясните ваши правила максимально возможному числу посетителей и партнеров вашего сайта так, чтобы это не сказалось на вашем рабочем процессе и образе жизни, а также пользуйтесь самыми надежными инструментами, какие сможете найти, для работы с таким контентом, как комментарии пользователей.

Валидация разметки как предпосылка к правильной реализации стилей

Это одна из важнейших отрицательных сторон — не потому, что она иллюстрирует привычки, с которыми следует бороться, а потому, она проливает свет на один из самых трудноизлечимых недостатков системы HTML+CSS.

Проще говоря, когда вы не добавляете закрывающий тег или присваиваете неверное значение `id/class`, то тем самым неумышленно вызываете отклонение в работе браузера, на который рассчитываете. Хотя покорность производителей закону Постела (см. раздел «Синтаксис HTML» на с. 32) зачастую скрывает недостатки в работе браузера и таблиц стилей, вряд ли стоит ожидать, что это будет происходить во всех случаях... а если ваша разметка особенно сложна, то лучший способ найти недостатки в рабочем каскаде — это проверить правильность разметки.

Вывод

Это необходимое зло, которое особенно строгие адепты указывают в качестве одного из немногих барьеров для начинающих веб-дизайнеров.

Рекомендуется просматривать в...

Своеобразие платформ, наследие раннего этапа развития Сети — когда Netscape был куда лучше, чем Mosaic, однако его место было вскоре занято Internet Explorer версии 3.0x, обладавшей намного лучшей поддержкой отображения, чем любая из версий Netscape, — больше не является повсеместным. Значение имеет ваша платформа разработки, и это должен быть Firefox (из-за бесподобной поддержки стандартов), если только вы не занимаетесь

разработкой для аудитории, которая пользуется исключительно Internet Explorer.

Хорошая новость: никому не важен ваш выбор платформы, да он и не должен быть никому важен. Но, как бы то ни было, вы будете сталкиваться с пользователями, которым надо, чтобы продукты выглядели именно так, а не иначе, на их рабочих станциях, даже если они используют древнее оборудование и после полудня работают, сидя спиной к солнцу.

Вывод

Из возможных сражений, в которых вам придется принимать участие, это, возможно, стоит того. Приобретайте опыт. Старайтесь вытаскивать заказчиков из-за столов и предлагайте им взглянуть на ваш продукт в какой-либо другой среде. Играйте в политику, если необходимо. Настаивайте на ступенчатой поддержке (см. следующий раздел). Не позволяйте вытирать об себя ноги.

Ступенчатая поддержка

Если вы следили за этой темой или вам действительно небезразличны дизайнерские решения Apple, то вы, скорее всего, заметили, что использование 64-битных персональных компьютеров медленно, но верно становится повсеместными. *Это дает толчок к первому большому прорыву в разработке программного обеспечения за несколько лет.* С другой стороны, даже сейчас веб-разработчики не сталкиваются с широким перечнем требований к поддержке, с каким они могли столкнуться всего каких-то десять лет назад, когда двухлетняя рабочая станция имела неплохие шансы быть названной «устаревшей».

Самый большой разброс, с которым могут столкнуться дизайнеры, относится к разрешениям дисплея. Некоторые пользователи просматривают страницы через коммутируемое соединение или мобильные телефоны 2.5G, но, по крайней мере, в отношении работы с Сетью, большинство посетителей сайта имеют возможность применения в достаточной степени современных программ.

Даже находясь между апатией большинства обычных пользователей Сети и сравнительно медленным прогрессом поддержки возможностей, вам придется обеспечивать поддержку широкого спектра платформ для просмотра. Некоторым дизайнерам приходится тяжелее, чем остальным; разработчики внутренних сайтов, работающие в студиях Windows, вообще не беспокоятся на этот счет, если только они не используют имеющееся в наличии программное обеспечение от независимых производителей.

Когда речь заходит о разнообразии, компаниями, которым приходится иметь дело с длинным списком возможностей просмотра, являются известные имена вроде Yahoo! и Amazon; крупные новостные сайты; крупные социальные сети

вроде Facebook; а также поставщики услуг, такие как коммунальные службы, банки и платежные операторы.

Yahoo! была упомянута первой не просто так: она была первой организацией, которая разработала, реализовала и популяризовала систему тестирования, которая подчинила все браузеры определенным стандартам производительности.

Каждая комбинация браузера и платформы операционной системы тестируется департаментом оценки качества Yahoo! и получает одну из трех перечисленных ниже степеней производительности.

- *Степень «A».*

  Эти платформы современны, распространены, способны поддерживать новые популярные возможности и предлагают отличный уровень поддержки для соответствующих стандартам методов реализации. Они позволяют получать более или менее те же впечатления от просмотра сайта, что и задумывались изначально.

- *Степень «C».*

  Платформы получают наибольшую выгоду от прогрессивного улучшения; хотя впечатления пользователей не достигают полной глубины, сайт все равно может использоваться достаточно эффективно.

- *Степень «X».*

  Эти платформы намеренно не были удостоены других оценок. Многие из них справляются с поставленными задачами, по крайней мере, до определенного момента. Наиболее заметные представители, как правило, выпущены недавно, а те, что не попадают под это правило, имеют небольшие рыночные доли. Запросы поддержки для таких платформ игнорируются, пока их рыночная доля не станет достаточно заметной, чтобы оправдать формальное тестирование.

В дополнение к этим трем степеням я представляю моим будущим клиентам степень «B», в которой нет поддержки поведения, однако есть неплохая поддержка представления. Другие студии могут при желании поменять эти приоритеты местами.

Так как над большинством своих проектов я работаю в качестве единственного правообладателя и, как правило, не одобряю стороннюю поддержку моих попыток тестирования, степенью «X» обладает большая часть моих платформ, чем у любой другой крупной студии.

Одна особенность спецификаций всех моих проектов, которая, похоже, отсутствует в описании метода ступенчатой поддержки Yahoo!, состоит в том, что все пользовательские впечатления от работы при определенной степени поддержки должны быть постоянны для каждого индивидуального посетителя, а в случае платформ ступени «A» — явно постоянны, без учета используемой платформы.

Сейчас вы, должно быть, задаетесь вопросом: почему же это — *отрицательная* сторона?

На это есть несколько причин. Важнейшая из них заключается в том, что ступенчатая поддержка подтверждает рыночную роль иждивенцев вроде Internet

Explorer 6 (а еще раньше Netscape 4) — браузеров, популярных исключительно благодаря силе инерции и сетевым тенденциям, а не другим достоинствам, которыми они могли бы обладать. Другая важная причина состоит в том, что к конечному продукту сетевые команды ближе всего подбираются лишь при тестировании для обеспечения качества: клиентам нравятся результаты, но они вряд ли захотят увидеть сам процесс изготовления.

Вывод

Применяйте степени поддержки правильно и часто, а после получения результатов тестирования действуйте исходя из интересов посетителей вашего сайта. Вам может это не понравиться, но зато это понравится им.

embed и object

То, что элемент `embed` все еще жив, — вероятно, самый вопиющий пример прикладных сетевых тенденций. В те времена, когда Windows Media Player был всего лишь дополнением, а платформа Flash была известна разве что благодаря сайту под названием Gabocorp, Netscape стал первым браузером, поддерживающим дополнительный модуль аудио-видео. Тем самым он весело обскакал всю работу W3C над HTML 4, не обратив на нее ни малейшего внимания.

В то же время компания Microsoft упорно старалась догнать конкурента и принялась за реализацию поддержки того элемента, который W3C предлагала использовать для добавления встраиваемого содержимого.

Те, кто проникся идеей позже, оказались где-то посередине, а поскольку браузеры Netscape были повсюду, Mozilla и Apple сфокусировались на обеспечении стабильной поддержки запуска мультимедиа с помощью `embed` — на этот выбор не повлияла даже доминирующая доля рынка, занятая Microsoft.

Вывод

С точки зрения перспективы совместимости снизу вверх `embed` совершенно несравним с `object`. Избегайте `embed`, если у вас, конечно, есть выбор. Повсеместная поддержка элемента `object` наконец привела к тому, что ее можно считать надежной во всех возможных случаях.

Управление формами, встраиваемые модули и наложение элементов

Проблема встраиваемых модулей и элементов управления формами, оказывающихся поверх всех остальных элементов и совершенно пренебрегающих абсолютным позиционированием и значениями `z-index`, хоть и менее распространена, чем в прошлом, все равно периодически всплывает.

Причина врожденной неуклюжести этих элементов состоит в том, каким образом они обрабатываются браузером. Во многих случаях подобные элементы

создаются и подключаются не движком браузера, а напрямую компонентами операционной системы. В случае Internet Explorer эти компоненты включают в себя платформу .NET Framework и технологию ActiveX, причем последняя вставляет на страницу элементы `object` так, как если бы они были модальными диалоговыми окнами.

Эта проблема смахивает и на «ужасную сторону», поскольку, если такие инциденты происходят, то исправить ситуацию очень сложно, и `z-index` в таких случаях не помогает.

Решения методом «в лоб» могут быть следующими.

- Переделайте макет так, чтобы наложения поверх нарушающего правила элемента не происходило.
- Поместите самое важное содержимое в пределах элемента `iframe`, который находился бы в исходном порядке ниже, чем нарушающий правила элемент.

Вывод

Эта проблема проявляется в Internet Explorer намного сильнее, чем в прочих браузерах. К сожалению, как и в случае с другими отрицательными сторонами, берущими начало именно от IE, простого решения этой проблемы найти не удастся, если производители браузеров не поменяют программное обеспечение.

Глупые причины неправильной разметки

Самая распространенная глупая причина невозможности подтвердить правильность разметки — это появление в коде знака «&», как и было сказано в главе 13.

Однако существуют и другие причины, куда более неприятные. Мой любимый пример странных требований HTML датирован 2002 годом, когда я только что начал работать над сайтом Webstandards.org. Я завершил входящее электронное письмо с соответствующей записью для блога на сайте, и моим следующим шагом — учитывая цель работы и аудиторию сайта — должна была стать валидация.

Но сделать это мне не удалось, и именно поэтому я узнал об этой глупой причине.

Webstandards.org использует строгий тип документа (Strict), а в моей записи содержался элемент `blockquote`. А я и понятия не имел (поскольку имел дело в основном с переходными типами документов), что при строгом типе `blockquote` должен содержать как минимум один элемент — возможно, абзац или `div`.

Другое непонятное требование при валидации было обнаружено во время технической редактуры черновой рукописи данной книги: `tbody` должен обязательно находиться *перед* `thead` и `tfoot`.

Эти требования, а также некоторые другие, описаны в сводной таблице на сопутствующем сайте этой книги.

Вывод

Если уж вы взяли на себя смелость добиться валидной разметки, то столкнетесь с некоторыми необычными требованиями, накладывающими ограничения на дочерние и родительские элементы, элементы одного уровня, атрибуты, значения и содержимое. Старайтесь не выйти из себя и *просто исправьте* все, что потребуется.

Плохие соседи HTML и «тупиковые» элементы

Тому HTML, который мы знаем, уже 20 лет, и разработан он был изначально для осуществления обмена научными работами и другим письменным материалом. Тогда никто и не задумывался о Flash, Ajax, подкастах и социальных сетях.

В результате новые возможности присоединялись к языку по одной за раз — и до конца 1997 года таблицу дополнений можно было назвать разве что промежуточной.

Учитывая эволюцию HTML, эволюцию инфраструктуры, поставляющей содержимое, и смену ожиданий сетевых пользователей, в HTML присутствуют и рудиментарные органы, и «идеи, которые в свое время казались отличными». «Ужасные стороны» — это все рудиментарные аспекты HTML (см. одноименный раздел на с. 324). Те из них, которые в редких случаях могут оказаться полезными, перечислены в нижеследующих разделах.

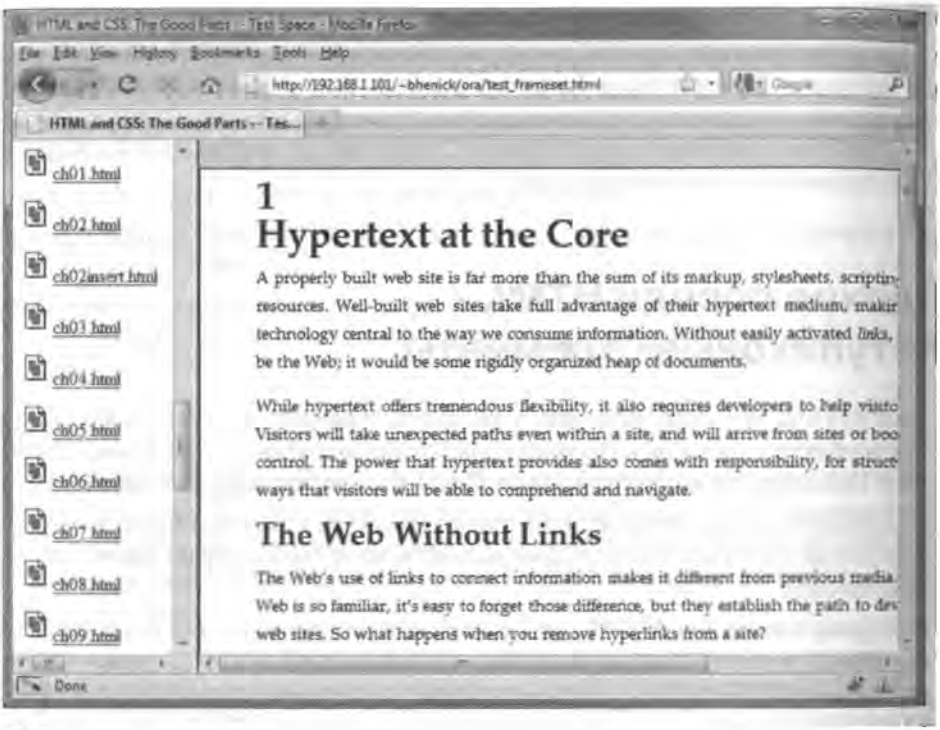
Фреймы

Фреймовые определения типа документа (Frameset) относятся к двум элементам, которые пребывали в расцвете сил в 1996 году, но сейчас находятся далеко не в лучшей форме: `frameset` и `frame`.

`frameset` является заменой `body` и поддерживает атрибуты `rows` и `cols`. Каждая строка и каждый столбец — в виде списка значений в пикселах или процентах, разделенных запятыми, причем в каждом списке должно быть как минимум одно значение (а лучше два или больше) — в свою очередь, соответствует элементу `frame` в пределах того же документа. Каждый из элементов `frame` имеет атрибут `src`, который ссылается на другой документ. Этот дочерний документ является либо обычной страницей, либо другим элементом `frameset`.

Наконец, каждому `frame` можно присвоить `name`, который может иметь `target` на любую из ссылок в пределах `frameset`. Это позволяет содержимому одного демонстрационного окна отличаться от содержимого других.

Рисунок 14.1 содержит визуальное описание вложенных элементов `frameset` и составляющих их документов.




```
_
</head>
<frameset cols="20%",*">
   <frame name="left_frame" src="ms/" />
   <frame name="left_frame" src="ms/ch01.html" />
</frameset>
</html>
```

Рис. 14.1. Рабочий набор фреймов и соответствующая разметка

Фреймы на самом деле достаточно эффектны и вызывают ощущение уникальности и превосходства в духе XX века, но это не средство — это, скорее, дефект. Вспомогательные технологии вряд ли могут обладать тем же взглядом на содержимое фреймов, который пользователи воспринимают как должное; но более важно то, что многомерность сетевых документов растет экспоненциально с каждым новым добавленным фреймом. Редкий дизайнер сможет сделать нечто полезное с помощью фреймов; более вероятно, что у дизайнера, который попытается согласовать фреймы, ничего не получится.

Единственное подходящее применение фреймов — это поместить навигацию по сайту в один документ и один фрейм, а содержимое сайта при этом поместить

в другой фрейм. Однако, помимо трудностей с доступностью, такой подход повышает сложность механизмов, требующихся для организации пометок типа «Вы находитесь здесь» для всех пользователей. Так происходит потому, что вам понадобится JavaScript для передачи состояния фрейма «содержимого» фрейму «перемещения», который, в свою очередь, должен будет выполнять свой JavaScript для выработки соответствующих сигналов.

Также существуют элементы `iframe` (плавающие фреймы), которые описаны в переходных определениях типа документа и на уровне разметки отличаются от изображений только тем, что могут загружать документы любого MIME-типа, который распознается браузером. На уровне пользовательского интерфейса они делят с фреймами способность отображать либо скрывать полосы прокрутки при помощи атрибута `scrolling`.

Вам с куда большей вероятностью могут понадобиться именно плавающие, а не внешние фреймы, особенно если на своем сайте вы размещаете посредническую рекламу. Плавающие фреймы также предоставляют пространство для асинхронных транзакций — «Ajax для бедных», если позволите. Вместо того чтобы осуществлять такие запросы при помощи объектов `XMLHttpRequest` и копировать узлы нового документа с их выходов, разработчик может загрузить несколько последовательных документов в закадровый плавающий фрейм, а затем скопировать узлы документа из этого фрейма для присоединения к документу, предназначенному пользователю.

Вывод

Просто *не используйте их* — ни сегодня, ни завтра, вообще никогда. Оправдано может быть разве что использование `iframe`, если поставщики рекламы рассчитывают именно на них (и тем самым настаивают на том, чтобы вы предложили хакерам еще одно направление для атаки вашего сайта); в противном случае вы можете воспользоваться возможностями сервера и `XMLHttpRequest` для поддержки функциональных средств фреймов и плавающих фреймов.

Элемент strike

Если говорить вкратце, то элемент `del` делает то, для чего предназначен элемент `strike`, и его поддержка находится на достойном уровне.

Вывод

Используйте элемент `del`. Особенно на это стоит обратить внимание разработчикам платформ для блогов.

Атрибут name

В табл. 14.2 приведены элементы, для которых может использоваться атрибут `name`.

Таблица 14.2. Элементы с атрибутом name и связанные функции

Элемент	Функция
`a`	Задает местоположение ссылки в документе.
`form controls`	Определяет имя данной пары имя/значение; необходим для эффективного шифрования данных формы.
`frame` и `iframe`	Однозначно отождествляет единичный фреймовый объект так, чтобы к нему мог осуществляться доступ при помощи атрибута `target` данной ссылки в пределах одного и того же представления.
`link`	Определяет имя объекта `linked` (к примеру, альтернативной таблицы стилей), на который можно ссылаться из постоянного пользовательского интерфейса браузера.
`map`	Ссылается на элемент `map`, связанный с конкретным изображением
`meta`	Альтернатива `http-equiv`; значение этого атрибута устанавливает имя произвольной пары имя/значение, определяющей метаданные документа, к примеру, `keywords`.

Вывод

Использование атрибута `name` нежелательно по сравнению с методами, описанными в таб. 14.3.Употребление атрибута `name` признано целесообразным только в случае элемента `link` (по желанию) и элементов управления формами (обязательно).

Таблица 14.3. Альтернативы атрибуту name

Элемент	Возможная альтернатива
`a`	Значения `id` поддерживаются всеми современными браузерами как значения хэш-функции ссылки.
`iframe`	Реализуйте необходимые функции при помощи JavaScript (в тех случаях, когда политика одного домена разрешает доступ к объекту) и создайте резервную реализацию, если возможно.
`map`	Учитывая хорошую поддержку значений `x` и `y` для `input type="submit"` и другие современные возможности браузеров, использование данного элемента нежелательно.
`meta`	Избегайте его в любых ситуациях, поскольку самое популярное значение элемента `meta` (`"keywords"`) полностью игнорируется главной его «аудиторией» — поисковыми машинами.

Элементы noscript и noframes

Содержимое `noscript` нужно только в том случае, если ваш стиль игнорирует все достоинства прогрессивного улучшения. Вместо этого вам следовало бы разрабатывать сайты и приложения, приняв в качестве условия тот факт, что скрипты недоступны, и полагаться на скрипты для того, чтобы возможности пользователя могли выйти за пределы так называемого «базового» уровня функциональности. На самом деле, любая другая философия дизайна призывает вас преступить законы и принципы сетевой доступности.

В свою очередь, элемент `noframes` мало чем отличается от тех элементов, которые призван заменять.

Вывод

Эти элементы существуют для обеспечения отказоустойчивости, что было хорошим дизайнерским приемом со стороны разработчиков из Netscape, которые изначально решили ввести элементы `script` и `frame`. Однако если вам действительно нужен какой-либо из этих элементов, это свидетельствует о плохих дизайнерских привычках. Если вам все же необходимо использовать их, используйте, но только ради расширения собственного опыта работы «с низкой точностью». В случае же `noframes`, единственное нормальное применение, приходящее мне на ум, — это обеспечение ссылки на рабочую карту сайта.

Семантические искажения и ограниченный словарный запас HTML

Было отмечено, что HTML распространился далеко за пределы своего изначального предназначения, и это зачастую заставляет семантически мыслящих дизайнеров засовывать квадратные стержни в круглые дыры.

К счастью, люди, ответственные за HTML5, и приверженцы микроформатов прикладывают невероятное количество усилий для того, чтобы сделать жизнь семантически (и буквально) мыслящих людей проще.

Пока же лучшее, что мы можем сделать, — это использовать уже существующие возможности; выжимать каждую возможную каплю смысла из таблиц стилей, систематики и универсальных атрибутов; отклоняться в сторону «переусложнения» нашего продукта.

Вывод

Избегайте неподходящих элементов, старайтесь пользоваться универсальными атрибутами и, затаив дыхание, ждите, пока HTML5 станет популярным.

Строковые элементы

Хотя HTML 4 и ограничен с точки зрения семантики, его элементами представления официально пренебрегают из благих побуждений.

Элементы `strong`, `em` и `code` существуют именно для того, чтобы выполнять типичные функции элементов строкового представления (`b`, `i`, `kbd`, `font` и т. д.). Эти элементы применяются лишь для того, чтобы обеспечивать выполнение типографских соглашений длиной в несколько поколений, к примеру для выделения иностранных слов курсивом.

Вывод

Используйте эти элементы только в следующих случаях.

- Цель разметки явным образом не согласуется с семантически ориентированными элементами.
- Существуют типографские соглашения по набору, которые полностью поддерживают использование подобных элементов.
- Вам очень хочется присоединить к элементу соответствующие универсальные атрибуты, которые проливали бы свет на взаимосвязи между элементом, применимым типографским соглашением по набору и содержимым элемента.
- Вы получите больше пользы от стилей, если будете использовать элементы представления вместо `span`.

В данном случае при строгом типе документов `font` считается неверным и не должен использоваться вообще. Если вы сталкиваетесь с ним в устаревшем содержимом, следует удалить его атрибуты, добавить нужное значение `class` и удалить соответствующий ему фрагмент в таблице стилей.

Управление вертикальным пространством: hr и br

Элементы `hr` (горизонтальная линия) и `br` (перевод строки) — одни из самых старых в пространстве имен HTML, однако они — пережитки эры, предшествующей появлению таблиц стилей.

При этом они все же иногда появляются в некоторых странных случаях. Элемент `hr` может располагаться вне поля зрения и использоваться для присвоения значения `clear` вместо `none`. Элемент `br` может быть полезен в ситуациях, когда приложение свойства `white-space` вызывает больше проблем, чем решает, а содержимое не может быть отформатировано при помощи `div` или `span`.

Вывод

Возможно, вам удастся найти лучший способ, но всегда существует вероятность того, что когда-нибудь где-нибудь вам придется обратиться к этим элементам за помощью. Идите другим путем, если сможете найти его, но не думайте над этим слишком долго, особенно если у вас мало времени.

Элемент pre и свойство white-space

Самое большое преимущество элемента `pre` состоит в его стиле по умолчанию агента пользователя, который соответствует добавлению следующей строчки в таблицу стилей:

```
p.preformatted { white-space: pre; }
```

Другими словами, на вставку элементов `pre` затрачивается меньше усилий, чем на подчеркивание их семантической бессмысленности с помощью описанного выше стиля.

В то же время `pre` часто и при этом неуместно используется для форматирования структуры строк и строф традиционной поэзии, для чего этот элемент совершенно непригоден. Для этой цели существуют другие решения, с которыми вы можете ознакомиться на сопутствующем сайте этой книги.

Вывод

> `pre`, на самом деле, лучше не использовать, только если вы не феноменально ленивы или у вас настолько мало свободного времени, что вы готовы удовлетвориться решением «на скорую руку» и исправить его позже.

Пародии на CSS

Те из нас, кто был причастен к индустрии в середине 1990-х годов, знали про CSS и истово молились, чтобы его внедрили побыстрее и сделали более эффективным, чем он был. Короче говоря, сетевое программирование в конце 1990-х было ужасным, потому что этого не случилось.

Через десять лет после периода «абсурдного изобилия» у нас есть лучшая версия CSS. Однако она все равно не всегда справляется с требованиями современной Сети. В дальнейших разделах рассматриваются достаточно грубые способы исправить недостатки CSS.

Директивы

Для управления элементами дизайна директивы очень удобны. Они предоставляют дополнительный механизм для управления таблицами стилей, что само по себе невероятно элегантно.

К сожалению, поддерживаются они слабо. Имейте в виду следующее.

- Директивы `@import` могут применяться или не применяться конкретным браузером, и это зависит от того, как они написаны и где они расположены в исходном порядке документа или таблицы стилей.

- Директивы `@import` и `@media` могут располагать дополнительными атрибутами произвольного выбора (определяются производителем), для которых обычно требуется четкая документация.
- Поддержка свойств альтернативных средств оставляет желать много лучшего, о чем было сказано в главе 3.

Вывод

Когда директивы работают, они прекрасны, но когда они не работают, то... не работают. Некоторые из рассмотренных здесь возможностей могут неожиданно прийти вам на ум и сэкономить вам несколько часов работы, если время поджимает, но крайне маловероятно, что директивы когда-либо окажутся среди них.

Вычисленные значения и их округление

Каждое средство визуального отображения измеряется в определенных единицах длины, что подробно обсуждалось в главе 3. В процессе построения изображения браузеру необходимо перевести значения, выраженные в одних единицах (вроде `em`) в основные единицы (к примеру, в `px` в случае средств `screen`). Итоговые значения называются «вычисленными», что особо касается измерения длины. Определенным значениям могут соответствовать совершенно другие вычисленные значения, как в случае ошибки с удвоением границы, рассмотренной ранее в данной главе.

Различия между визуальными характеристиками и алгоритмами отображения шрифтов различных популярных браузеров периодически выливаются в ошибках округления, которые слишком очевидны, чтобы их игнорировать, например дробное значение `letter-spacing` в пикселах неприменимо в случае Mac, и это влияет на размеры элемента.

Если эти различия ведут к существенной разнице между макетами для разных платформ, что может произойти с дизайнерскими разработками, страдающими от низкой устойчивости к ошибкам, то итог может сильно разочаровать.

Вывод

Когда разработка требует такой степени точности, которая превращает ошибки округления в проблему, возможность ущерба можно минимизировать следующим образом. Во-первых, указывайте значения длины в форме с плавающей точкой с высокой степенью точности, например 10-3 или даже 10-4. Во-вторых, приведите таблицы стилей к единому средству и указывайте все длины в статических единицах, подходящих данному средству, к примеру в `px` для представления `screen` или `pt/in/cm` для представления `print`.

Префиксы свойств -moz и -webkit, специфичные для производителей

Поскольку пользователи Firefox и Safari, как правило, сознательно относятся к собственной работе в Интернете и обычно рано усваивают новые технологии, у разработчиков этих браузеров есть причины внедрять в свои продукты самые передовые средства.

В то же время стабильная поддержка CSS 2.1 и других «официальных» стандартов находится в самом верху маркетингового списка для этих браузеров, а свойства CSS3, поддерживаемые ими, еще будут меняться. На самом деле большинство модулей CSS3 пока даже не получили статус Last Call Working Draft — последнего этапа процесса рекомендации (де-факто — стандартизации) W3C, в течение которого широкая общественность может повлиять на развитие веб-технологии.

Поскольку CSS3 все еще не достигла зрелости, должный уровень поддержки успешнее всего моделируется в ее же собственной «песочнице» инженерами браузера. Поэтому и существуют префиксы `-moz` и `-webkit`.

Желательность применения этих свойств — уже другой вопрос. И каждая команда разработчиков должна ответить на него самостоятельно в соответствии со своей философией и целями проектов.

Вывод

Используйте их, если хотите, но будьте готовы к непредвиденным последствиям — особенно когда новые версии браузеров выпускаются после того, как модули CSS3 начинают проходить процесс рекомендации W3C.

Значение inherit

Наследование — дело тонкое. В большинстве случаев цвета переднего плана и размеры типов наследуются дочерними элементами от родительских элементов, а свойства блоков и им подобное — нет. *Существуют* сценарии, при которых вам может понадобиться принудительное наследование значений блоков и других свойств макета, однако они достаточно редки.

С другой стороны, имеет место следующая практически карикатурная ситуация: браузеры не полностью поддерживают значение `inherit`.

Вывод

Было бы неплохо, если бы когда-нибудь эта функция была реализована, хотя это и потребует четкого понимания относительно правильного ее применения. А пока придется повторять все по старинке.

Как прятать вещи: z-index и clip

На первый взгляд, и свойство `z-index`, и свойство `clip` кажутся в высшей степени полезными. Первое меняет позицию элемента в вертикальной схеме расположения, а второе частично влияет на видимость.

На практике применение этих свойств является куда более сложным, чем вы могли подумать.

Для правильного использования `z-index` надо запомнить, что контекст наложения документа организован в виде слоев, со следующими слоями снизу вверх.

1. Холст страницы.
2. Блоки без значений `float` или `position`.
3. Элементы со значениями `float`.
4. Элементы со значениями `position`, отличными от `static`.

Каждый элемент содержит его свойства фона, затем содержимое и, наконец, все перекрывающие элементы того же слоя, которые лежат ниже в исходном порядке документа. Порядок наложения повторяется для каждого слоя.

В то же самое время происходит ошибка со значением `clip`. Его цель похвальна: заполнить элемент, не сжимая при этом блок содержимого и оставив его значения для последующего управления посредством DOM API. Однако, в отличие от свойства `padding`, все его значения измеряются от верхнего и левого краев элемента, а это означает, что с функциональной точки зрения все это бесполезно, если вы не знаете всех размеров соответствующего элемента во время работы.

Вывод

Эти свойства прекрасны в теории, но весьма неприятны на практике; они принадлежат к тем дефектам, из-за которых дизайнерам приходится работать по ночам.

Счетчики

Перспектива использования автоматического подсчета в CSS хотя и не ясна, но все равно стоит того, чтобы о ней поразмышлять. Плохая новость заключается в том, что вся система целиком не работает, потому что правила эксплуатации совершенно непостижимы, а функциональность не поддерживается Internet Explorer.

Вывод

Чем возиться со смысловыми переменными (чего требуют существующие правила), гораздо удобнее было бы, если бы эта возможность была заменена одним значением `counter`, которое зависело бы от предыдущего состояния элемента и имело бы два аргумента: стартовое значение и уровень

значимости. Большинство дизайнеров, в том числе и я, скорее согласились бы на ограниченную автоматическую нумерацию, чем на мощную автоматическую нумерацию, которую невозможно реализовать.

Модели визуального представления элементов

Сидели ли вы когда-нибудь с отвисшей челюстью перед монитором, пытаясь понять, каким образом браузер взял ваш макет и выдал *вот это*? Это, наверное, происходит из-за большого объема спецификации CSS. Прочитайте, например, вот это:

> Строчный блок всегда имеет достаточную высоту, чтобы вместить в себя все нужные блоки. Однако он может быть выше, чем самый высокий из блоков, которые он содержит (если, к примеру, блоки выровнены так, что базовые линии выстроены в ряд). Если высота блока B меньше, чем высота содержащего его блока, то вертикальное выравнивание B в пределах блока определяется свойством 'vertical-align'. Если несколько внутренних боксов не помещаются по горизонтали в один строчный блок, они распределяются на два или больше уложенных вертикально строчных блока. Таким образом, абзац — это вертикальный набор строчных блоков. Строчные блоки укладываются без разделения по вертикали и никогда не перекрываются.
>
> В общем случае левый край строчного блока касается левого края блока, который он содержит, а правый край, соответственно, касается правого. Однако между краями могут оказаться еще и плавающие блоки. Таким образом, хотя строчные блоки в одном и том же контексте внутреннего форматирования имеют одинаковую ширину (равную ширине содержащегося внутри блока), они могут различаться по ширине, если имеющееся пространство по горизонтали уменьшено из-за плавающих блоков. Строчные блоки в одном и том же контексте внутреннего форматирования обычно различаются по высоте (к примеру, одна строка может содержать в себе высокое изображение, а остальные строки — только текст).
>
> *Спецификация CSS: раздел 9.4.2 «Контекст внутреннего форматирования»*

Усвоили?

А тем временем некий невезучий инженер-разработчик ПО был вынужден постичь смысл этого отрывка и разработать программу, которая делает именно так, как там написано. Вы пользуетесь ей каждый день.

Это один из недостатков CSS, который иллюстрирует преимущество сообществ: если вы можете присоединиться к активному сообществу людей, занимающихся разработкой, то вы, скорее всего, столкнетесь там хотя бы с одним человеком, который постиг дзен, вчитываясь в абзацы вроде приведенных выше.

Вывод

Просветление — это настолько же *процесс*, насколько состояние. Не забывайте о профессиональном обучении, даже если вам иногда приходится сжимать зубы и пробовать снова и снова.

Значения кодовых позиций Unicode и свойство content

Свойство `content`, которое было представлено в главе 7 и затем упомянуто в главе 8, имеет странный недостаток: оно не поддерживает объекты. Если в вашей таблице стилей присутствуют `:before`/`:after` или `content`, у вас есть три «безопасных» пути, чтобы сослаться на символы не из 7-битного диапазона ASCII.

- Перед тем как вставлять символ, убедиться, что и ваше средство разработки, и сетевой сервер используют одну и ту же кодировку символов.
- Дать необходимое объявление `@charset` в таблице стилей и вставить символ.
- Использовать кодовые позиции Unicode без обратного слэша для указания нужного содержимого.

Ни один из этих способов на практике не обходится без подводных камней. Первый из них опасен, если у вас нет абсолютного контроля над средой редактирования и сервером, чего достоверно нет у большинства людей. Второй непрактичен, поскольку некоторые дошедшие до наших дней версии Internet Explorer и Safari не понимают объявления `@charset`, причем каждая по-своему. Третьему недостает надежной поддержки, но у него есть свои достоинства — совместимость снизу вверх и устойчивость к переменам в конфигурации сервера и условиям сети.

Если вы попытаетесь сослаться на значения кодовых позиций Unicode в свойстве `property`, то значение, которое вы хотите передать, — это кодовая позиция Unicode, выраженная в шестнадцатеричной системе и предваренная обратной косой чертой; к примеру:

```
blockquote>p:before { content: "\201C"; }
```

В результате перед каждым абзацем будет поставлена двойная открывающая «английская» кавычка, которая будет иметь элемент `blockquote` в качестве прямого родителя, так как 0×201C = 8220.

Вывод

Вставьте символы UTF-8 там, где необходимо, и убедитесь, что сервер не будет возражать.

Ужасные стороны

Дальше — хуже.

В HTML и CSS полно разных ненужных, непродуманных и безжалостно неизбежных вещей... которые, однако, делают вид, что имеют важное значение для общества.

Они подобны некоему веществу, которое необъяснимым образом зародилось у кого-то на задворках мозга, выбралось оттуда и, преодолев совещания

и главного менеджера, попало в код и в итоге ожило в электрических внутренностях работающего браузера. *Никогда не пользуйтесь средствами, описанными в финальном разделе, если только за отказ пользоваться ими вас не уволят.*

Элементы marquee и blink

Эти элементы и в самом деле являются пережитками подросткового возраста Интернета: в Firefox элемент `blink` по умолчанию отключен, если еще поддерживается. Эти элементы *насильно* отвлекают посетителей. К тому же у эпилептиков они могут и припадок вызвать.

Вывод

Анимация принципиально отличается от остального содержимого, она является поведением, а не структурой или представлением. Если вам необходимо использовать анимацию, реализуйте ее при помощи JavaScript или Flash, будьте добропорядочным гражданином.

Свойства пользовательского интерфейса MSIE

Internet Explorer дает дизайнерам доступ к полосам прокрутки и хромированным окнам, которые, к сожалению, активно используются. Это дает возможность расширить фирменное оформление на другие элементы пользовательского интерфейса, что в глазах некоторых выглядит как победа.

Но на самом деле это нисколько не победа. Когда вы балуетесь с пользовательским интерфейсом, который остается неизменным в любом другом приложении, то обманываете ожидания пользователя и, возможно, заставляете наиболее легкомысленных ваших посетителей подумать, что их система заражена вирусом.

Вместо этого

Разработайте и выпустите приложение для Adobe AIR. Для этого вам, возможно, придется заново изобрести велосипед, но, во всяком случае, этим вы сможете заработать себе на жизнь.

Атрибут align

В старые недобрые времена, когда CSS еще не существовал, а там, где существовал, на него нельзя было положиться, веб-разработчики пользовались табличной разметкой и атрибутом `align`.

Если вы внимательно прочли эту книгу, то могли заметить, что технология за прошедшие годы слегка ушла вперед.

Вместо этого

Избавьтесь от него везде, где найдете, и вместо него используйте соответствующие значения `float`, `margin` или `text-align`.

Атрибут style

Так... у нас имеется этот универсальный атрибут, позволяющий применять свойства CSS в самых узких возможных рамках, но с полной уверенностью в том, что указанные значения будут применены.

А некоторые люди, к примеру, больны гриппом. Но они же не лезут из кожи вон, чтобы заразить им всех остальных.

Когда вы используете атрибут `style`, вы разрушаете саму идею таблицы стилей, объявления типов средств и всякую надежду на возможность поддержки ваших комплектующих.

Просто не делайте так.

Вывод

Сядьте, пролистайте книгу к началу раздела «Конфликты правил, приоритеты и очередность» на с. 55, прочтите ее и, наконец, используйте на практике то, о чем прочли. Возможно, атрибут `style` кажется более простым решением, если вы не можете овладеть правильной верстkой, но в веб-разработке это аналог термоядерной бомбы для разрушения бункеров.

div-itis

Разработчики, незнакомые с понятием «CSS-дзен», слишком часто и слишком легко попадают в ловушку стремления подчинить верстку внешнему виду — так же, как они сделали бы, если бы использовали для макета разметку `table`. Эта привычка — названная «div-itis» по понятным причинам — приводит к разметке, где много элементов, получивших свое расположение не ради правильной разметки, а чтобы обеспечить конкретный результат в представлении.

Эти элементы легко обнаружить, так как значения их атрибутов `class` или (реже) `id` относятся не к цели содержимого, но к характеристикам представления содержимого. Фрагменты `class` и `id`, которые ссылаются на относительные направления, к примеру `right` и `left`, регулярно появляются в таких разметках.

Вывод

Существует несколько методов, позволяющих избежать потребности в «div-itis». Фундаментальный прием — это присвоить `class` и `id` значения, которые отражают *цель* элемента, а не его взаимосвязь с представлением документа. Список также включает в себя следующие методы.

- Небольшая структурная надстройка (см. раздел «Четыре правила для эффективного создания стиля» на с. 71).
- Изменение контекста позиционирования (см. раздел «Свойства позиционирования в CSS» на с. 121).
- Осторожное применение фоновых изображений (см. раздел «Составление фоновых изображений» на с. 180).

- Эффективное использование всех свойств блоков (см. раздел «Отступы, поля и рамки» на с. 103).

Атрибуты обработчика событий

Обработчики событий в разметке не сильно отличаются от атрибута `style`, однако представляют собой *еще более ненужную* вещь (если такое вообще возможно). Присоединяя их напрямую к тегам, вы помещаете поведение в структурный слой вашего продукта, что непременно приводит к проблемам (и делает вашу разметку еще более трудной для прочтения). В конце концов, подход «ненавязчивого JavaScript» куда более прост в обращении.

Вывод

1. Оцените, к каким элементам вы хотите присоединить события, и добавьте нужные универсальные атрибуты, которые понадобились бы вам для доступа к ним посредством DOM API.
2. Создайте и сошлитесь на функцию, запускающую `onload`, например:
   ```
   window.onload = assignEvents;
   ```
3. В пределах функции `onload` пройдитесь по списку различных элементов, оцененных во время первого шага, и там, где надо, присвойте обработчики событий и связанный с ними код.
4. Если ваши потребности растянуты по временной оси или зависят от нескольких факторов, вам может потребоваться реализовать замыкания для обработки событий. На сопутствующем сайте этой книги вы сможете найти ссылки на обсуждения, касающиеся замыканий JavaScript.

Беспричинное подчеркивание

Если вы любите подчеркивания, то проверьте, пожалуйста, как обстоят дела в реальном мире: подчеркивания нужны для ссылок, только для ссылок и ни для чего, кроме ссылок. Это отвечает ожиданиям посетителей, и эти ожидания — одни из важнейших, если только вы не хотите, чтобы люди щелкали курсором где попало, пытаясь попасть на другую страницу сайта.

Вывод

Есть два применения подчеркиваний и за пределами ссылок: содержимое `ins` (для которого это стиль пользовательского агента по умолчанию) и выделение в макете, дизайн которого разработан для печатного представления. В первом случае убедитесь как минимум в том, что цвет и яркость подчеркнутых вставок *полностью* отличаются от цвета и яркости всех ссылок; во втором случае применяйте подходящий стиль к элементу `em` и размещайте «печатное» содержимое в зрительной плоскости, полностью отделенной от остальной страницы.

Атрибут http-equiv

Некоторые читатели работают с системными администраторами, обладающими чрезмерно жестким — и зачастую непродуктивным — подходом, заключающимся в блокировке каждого возможного варианта конфигурации сетевого сервера. Если вы принадлежите именно к этой группе читателей, то следующий раздел к вам не относится.

Когда простые люди начали выпускать свои собственные веб-сайты, над чьей-то головой зажглась лампочка идеи виртуальных доменов. Поддержка скриптов на стороне сервера была и дорогостоящей, и сложной в применении. В то время (около 15 лет назад) программный интерфейс приложений (API) веб-сервера был территорией, закрытой для непосвященных.

Атрибут `http-equiv`, как и фреймы, был внедрен для того, чтобы владельцы сайтов могли воспроизводить поведение сайта, которое во всех остальных случаях могло быть реализовано только после значительных затрат денег, времени и сил.

Вывод

К счастью, Сеть уже давно оставила позади период метафорических каменных ножей и медвежьих шкур. Если вы хотите повлиять на работу сервера, поддерживающего ваш сайт, то можете с тем же успехом использовать модули сценарных языков HTTP API, поддерживаемые вашим тарифом хостинга, или применить Apache и его платформу `.htaccess`. Это на самом деле проще, чем кажется, и в Сети вы можете найти немало кода в качестве примера; за примерами можете зайти на сопутствующий сайт этой книги.

Подводя итог

У HTML и CSS есть немало положительных сторон, которые помогают добиться отличных результатов. У них есть и отрицательные стороны, и вполне понятно ваше желание обойтись без необходимости разбираться с ними. Наконец, как и у всех других технологий, у HTML и CSS есть и ужасные стороны, которые вообще никогда не должны были появляться на свет.

Отличая одну из этих сторон от другой и зная, где и когда применить нужные средства, вы сможете избавиться от большинства недостатков вашего продукта и добиться от своих сайтов наилучшего удобства эксплуатации.

Приложение
URI, архитектура клиент-сервер и HTTP

При том что гипертекст является основным средством создания уникальной среды (виртуальной), тот факт, что гипертекст зависит от Интернета, часто остается незамеченным. Веб-разработчикам не обязательно знать все о глубинном устройстве Интернета – например, о TCP, IP и DNS. Однако знание HTTP (протокола передачи гипертекста) принципиально важно при создании сайта, поскольку он является основой URI вида http://, которые постоянно встречаются в ссылках.

Базовая архитектура клиент-сервер

Процедура обработки данных протоколов клиент-сервер (включая HTTP) обычно выглядит так, как показано на рис. А.1.

Рис. А.1. Как работает архитектура клиент-сервер: 6 этапов

1. Клиент ищет IP-адрес нужного сервера, при необходимости используя сервер имен.
2. Клиент ищет и устанавливает с сервером соединение транспортного уровня; в случае HTTP для этого используется TCP/IP.
3. Далее клиент отправляет данные на сервер через соединение, позволяющее получить ответ с этого сервера. Такая передача данных обычно называется *запросом*.
4. Сервер получает запрос и обрабатывает его, запуская весь необходимый исполняемый код, а затем упаковывает выходные данные.
5. Сервер отправляет упакованные данные обратно через соединение, открытое клиентом на этапе 1. Этот процесс обычно называется *ответом*.
6. Клиент получает данные и — если они имеют подходящий формат — хранит и обрабатывает их локально. В случае веб-транзакций «обработкой» часто является отображение страницы.

Сбои могут происходить на различных этапах. Наиболее известной является ошибка HTTP «404 Not Found», вызванная проблемами при выполнении шага 3.

Что каждый веб-разработчик должен знать о HTTP

Более мелкие детали, касающиеся использования HTTP и работы с веб-серверами, не рассматриваются в этой книге, однако есть несколько вещей, которые необходимо знать при создании любого сайта.

- Существуют ограничения на время задержки в сети, после чего происходит завершение сессии (более известное как тайм-аут).

  Поскольку невозможно предсказать состояние сети и объем передаваемых данных между сервером и браузером, разработчику не удастся угадать порядок загрузки отдельных фрагментов содержимого страницы и время, требующееся на загрузку всей страницы.

- Запрос на одну страницу, как правило, приводит к нескольким запросам на разные URI.

  Для отображения страниц требуются изображения, мультимедиа, библиотеки кода и другие внешние ресурсы (рис. А.2). При большом времени задержки может значительно ухудшиться взаимодействие с пользователем; при передаче больших объемов данных неправильное использование ресурсов может привести к чрезмерной нагрузке на сервер. Поэтому лучше всего найти золотую середину между использованием ресурсов сервера и сети, и ответственность за это (по крайней мере, отчасти) лежит на специалисте по HTML/CSS.

- HTTP — протокол без запоминания состояния.

Это означает, что каждая транзакция между веб-сервером и клиентом происходит отдельно, без окружающего контекста. С помощью специальных приемов (например, cookie-файлов) можно добиться того, чтобы сервер отслеживал контекст. К URI могут быть добавлены закодированные данные, относящиеся к конкретной сессии, однако при этом существенно увеличивается длина URI. Основным следствием такого «незапоминания» состояния для специалиста по HTML/CSS является невозможность точно определить состояние среды клиента пользователя. В результате практически ничего нельзя сказать о взаимодействии системы с пользователем в текущей сессии, что значительно усложняет процесс оптимизации сайта.

- Чаще всего используются HTTP-запросы GET и POST.

  Можно утверждать, что большинство серверных запросов являются запросами GET, но существует достаточно много ситуаций (особенно если требуется подтверждение формы с данными о посетителе), в которых предпочтительнее использовать метод POST по техническим соображениям или в целях безопасности. Более подробно о различиях между запросами GET и POST рассказывается в главе 13. (HTTP поддерживает и некоторые другие методы, однако GET и POST наиболее универсальны.)

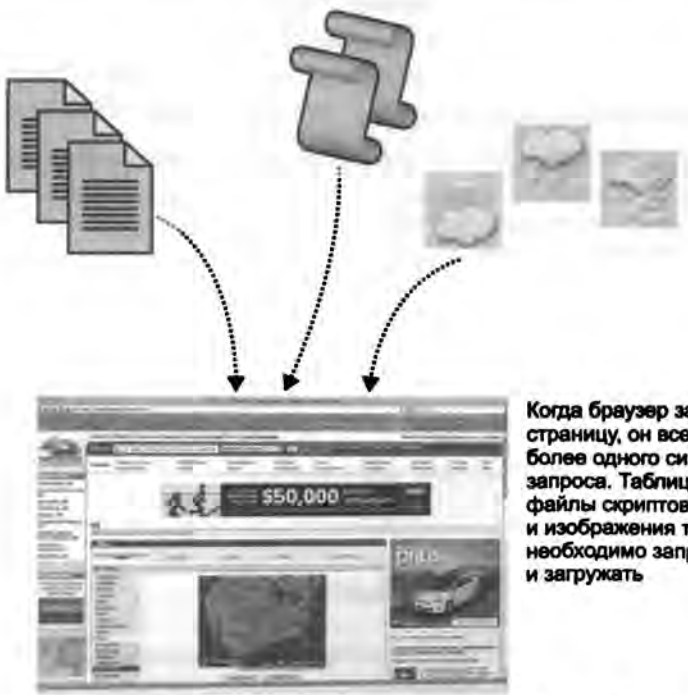


Рис. А.2. Запрос на одну страницу, как правило, приводит к нескольким запросам на разные URI

Коротко о MIME-типах

В дополнение к объявлению типа документа, которое задается автором вверху исходного файла страницы, сервер отправляет свои собственные данные об используемых форматах. Эти сведения передаются в формате MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения почты интернета); такой способ используется по соображениям безопасности. Наиболее часто используемые MIME-типы приведены в таблице.

Тип ресурса	Символьный MIME-тип
HTML	`text/html`
Правильно построенный XHTML	`application/xhtml+xml`
Изображения	`image/gif`, `image/jpeg`, `image/png`, `image/svg+xml`
Adobe Portable Document Format	`application/pdf`
Adobe Flash	`application/x-shockwave-flash`
Аудиозапись в формате MP3	`audio/mpeg`
Windows Media	`video/x-ms-wmv`, `audio/x-ms-wma`

Существует три способа задания MIME-типа файла или ответа сервера.

- Заголовок ответа сервера по умолчанию.

  Средства настройки веб-сервера позволяют устанавливать соответствие между MIME-типами и расширениями файлов.

- Специально созданный заголовок ответа.

  HTTP-функции основных скриптовых языков веб-серверов позволяют задавать MIME-тип отдельно для каждого случая (например, `Header("Content-Type: text/xhtml+xml\r\n");` в PHP).

- Расширение файла.

  Браузеры просматривают полученное содержимое в таком порядке: сначала происходит проверка MIME-типа, заданного в заголовке ответа, затем — определение расширения файла, а после этого — чтение заголовков файла.

Следует, однако, помнить о том, что полагаться на поведение браузера не рекомендуется, поскольку в таком случае непонятно, как обрабатывать поврежденные данные (например, видеофайлы и видеопотоки).

Управление объемом запроса

Слишком сложные таблицы стилей могут вызвать лишь некоторые проблемы из тех, что возникают, когда дизайнерам и разработчикам не удается оптимизировать производительность хоста клиента и сервера. Если вы занимаетесь оформлением, наиболее важными для вас будут следующие рекомендации.

- Старайтесь, чтобы все файлы, относящиеся к одному запросу, передавались с одного сервера или локальной группы серверов.

  Этой рекомендации трудно следовать, если в вашем проекте используется реклама или сторонний контент (то есть мультимедиа и инструменты социальных сетей). Однако размещение ресурсов сайта на одном сервере или в одной сети уменьшает риск некорректного отображения данных, которое оператор не в состоянии исправить.

- Старайтесь уменьшить число серверных запросов.

  Правильно настроенный хост сервера с достаточными возможностями соединения может легко обрабатывать несколько сотен тысяч HTTP-запросов в час, однако это не то же самое, что несколько сотен тысяч *пользователей* в час. Даже для самой простой статичной страницы может потребоваться несколько десятков запросов на таблицы стилей, JavaScript-файлы и изображения. Существует ряд методов уменьшения загрузки запросов: использование спрайтов, создание уникальных таблиц стилей с помощью элемента `style`, а также *кэшированное* соединение серверной разметки и CSS-ресурсов с помощью директивы `include` и функций для буфера вывода.

- Помещайте JavaScript-файлы в конце исходного файла страницы всегда, когда это возможно.

  Это позволяет приостановить все сетевые запросы и выполнение до тех пор, пока не загрузится *контент* страницы — это может значительно улучшить время отображения страницы.

Несколько секунд тут, несколько секунд там — и уже скоро мы сможем говорить о *реальном* времени работы

Бен Хеник

HTML и CSS: путь к совершенству

Перевели с английского Т. Качковская, С. Коновалов, Е. Шикарева

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Научный редактор	*А. Гребеньков*
Литературный редактор	*В. Шрага*
Художественный редактор	*Л. Адуевская*
Корректор	*М. Рошаль*
Верстка	*А. Ганчурин*

Подписано в печать 16.11.10. Формат 70x100/16. Усл. п. л. 27,09. Тираж 2500. Заказ 24495.
ООО «Лидер», 194044, Санкт-Петербург, Б. Сампсониевский пр., 29а.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная
Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., 15.

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают эксклюзивный ассортимент компьютерной, медицинской, психологической, экономической и популярной литературы

РОССИЯ

Санкт-Петербург м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва м. «Электрозаводская», Семеновская наб., д. 2/1, корп. 1, 6-й этаж
тел./факс: (495) 234-38-15, 974-34-50; e-mail: sales@msk.piter.com

Воронеж Ленинский пр., д. 169; тел./факс: (4732) 39-61-70
e-mail: piterctr@comch.ru

Екатеринбург ул. Бебеля, д. 11а; тел./факс: (343) 378-98-41, 378-98-42
e-mail: office@ekat.piter.com

Нижний Новгород ул. Совхозная, д. 13; тел.: (8312) 41-27-31
e-mail: office@nnov.piter.com

Новосибирск ул. Станционная, д. 36; тел.: (383) 363-01-14
факс: (383) 350-19-79; e-mail: sib@nsk.piter.com

Ростов-на-Дону ул. Ульяновская, д. 26; тел.: (863) 269-91-22, 269-91-30
e-mail: piter-ug@rostov.piter.com

Самара ул. Молодогвардейская, д. 33а; офис 223; тел.: (846) 277-89-79
e-mail: pitvolga@samtel.ru

УКРАИНА

Харьков ул. Суздальские ряды, д. 12, офис 10; тел.: (1038057) 751-10-02
758-41-45; факс: (1038057) 712-27-05; e-mail: piter@kharkov.piter.com

Киев Московский пр., д. 6, корп. 1, офис 33; тел.: (1038044) 490-35-69
факс: (1038044) 490-35-68; e-mail: office@kiev.piter.com

БЕЛАРУСЬ

Минск ул. Притыцкого, д. 34, офис 2; тел./факс: (1037517) 201-48-79, 201-48-81
e-mail: gv@minsk.piter.com