Вадим Дунаев

Самоучитель основы WEB дизайна

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.06
ББК 32.973.26-018.2
Д83

Дунаев В. В.

Д83 Основы Web-дизайна. Самоучитель. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 480 с.: ил.

ISBN 978-5-9775-0110-1

Рассмотрены основные сведения по разработке Web-приложений, необходимые начинающему Web-разработчику. Приведены общие понятия, а также вопросы, связанные с разработкой, публикацией и дальнейшим сопровождением сайта в целом. Подробно рассмотрены разметка гипертекстовых документов (XHTML, HTML 4 и HTML 5) и применение каскадных таблиц стилей (CSS 2/3). Описаны основные средства разметки страниц сайта и форматирования текстов. Показано, как помещать в HTML-документ содержимое из внешних источников (графические изображения, Flash-ролики, звук, видео и т. п.), а также как создавать специальные визуальные эффекты. Уделено большое внимание элементам пользовательского интерфейса, с помощью которых обеспечивается интерактивность Web-страниц. Рассказано о решении различных задач с помощью сценариев, написанных на языке JavaScript. Второе издание переработано и дополнено с учетом современных Web-технологий.

Для начинающих Web-дизайнеров

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Игорь Цырульников*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Подписано в печать 30.09.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0110-1

© Дунаев В. В., 2011
© Оформление, издательство "БХВ-Петербург", 2011

Оглавление

Предисловие ко второму изданию **9**

Введение **11**

Глава 1. Что такое Web-дизайн **15**
1.1. С чего начать? 15
1.2. Графический дизайн 17
1.3. Шаблоны страниц 19
1.4. Реализация проекта 20
1.5. Наполнение информационным содержимым 24
1.6. Тестирование и опытная эксплуатация 26
1.7. Публикация и оптимизация сайта для поисковых систем 28
1.7.1. Доменное имя сайта 28
1.7.2. Регистрация и индексирование в поисковых системах 29

Глава 2. Как устроен HTML-документ **33**
2.1. Что такое HTML 33
2.2. Определение типа документа 41
2.3. Структура (X)HTML-кода 44
2.3.1. Раздел заголовка документа <*head*> 46
2.3.2. Раздел тела документа <*body*> 52
2.4. Основные атрибуты тегов 53

Глава 3. Основы каскадных таблиц стилей **56**
3.1. Что такое CSS 56
3.2. Присоединение таблиц стилей к (X)HTML-документу 59
3.3. Правила форматирования 60
3.3.1. Селекторы 61
3.3.2. Контекстные селекторы 62
3.3.3. Псевдоселекторы и псевдоэлементы 63
3.4. Приоритеты определений параметров стилей 65
3.5. Размерность 68
3.6. Цвет 69
3.7. Поля, отступы, границы и размеры 70
3.8. Наследование параметров 81

Глава 4. Позиционирование элементов ... 83
4.1. Расположение элементов в нормальном потоке ... 83
4.2. Позиционирование с помощью CSS ... 86
4.2.1. *position:static* ... 87
4.2.2. *position:relative* ... 87
4.2.3. *position:absolute* ... 87
4.2.4. *position:fixed* ... 91
4.3. Отсчет координат ... 92
4.4. Слои, или третья пространственная координата ... 93
4.5. Обтекание (*float*) ... 94
4.6. Видимость ... 97
4.6.1. *overflow* ... 97
4.6.2. *clip* ... 98
4.6.3. *visibility* ... 98
4.6.4. *display* ... 99
4.7. Размеры элементов ... 100
4.8. Практические примеры ... 102
4.8.1. Центрирование элемента ... 102
4.8.2. Управление положением элемента с помощью мыши ... 104
4.8.3. Раскрывающаяся панель ... 105

Глава 5. Фон элементов и границ ... 107
5.1. *background* ... 107
5.2. Прозрачность ... 111
5.3. Тень ... 115

Глава 6. Компоновка страницы ... 117
6.1. Базовые схемы компоновки страницы ... 117
6.2. Жесткая схема ... 123
6.3. Резиновая схема ... 125
6.4. Центрирование страницы ... 126
6.5. Декорация схемы ... 126
6.6. Вставка плавающего фрейма *(<iframe>)* ... 129

Глава 7. Ссылки ... 133
7.1. Текстовые ссылки ... 134
7.1.1. Простое меню ссылок ... 134
7.1.2. Двухуровневое меню ссылок ... 139
7.2. Графические и комбинированные ссылки ... 143
7.3. Графические карты ссылок ... 144
7.4. Внутренние ссылки ... 146
7.5. URL-адреса ... 148
7.5.1. Структура URL ... 148
7.5.2. Абсолютные и относительные пути ... 150
7.5.3. Псевдо-URL JavaScript ... 151

Глава 8. Форматирование текстов ... 152
8.1. Шрифты ... 152
8.2. Основные теги разметки текстов ... 156

8.3. Специальные символы ... 158
8.4. Работа с текстом ... 158
8.4.1. Красная строка ... 159
8.4.2. Выравнивание ... 159
8.4.3. Межстрочное расстояние ... 159
8.4.4. Межсловное расстояние ... 160
8.4.5. Межбуквенное расстояние ... 161
8.4.6. Декорация ... 161
8.4.7. Индексы ... 163
8.4.8. Выделение первой буквы строки и первой строки в блоке текста ... 163
8.4.9. Текст с тенью ... 163
8.4.10. Преобразование регистра ... 166
8.4.11. Мультиколоночная верстка ... 166
8.5. Предварительно отформатированный текст ... 166
8.6. Генерируемое содержимое ... 167

Глава 9. Списки ... 172
9.1. Маркированный список ... 172
9.2. Нумерованный список ... 174
9.3. Автоматическая нумерация элементов списка ... 175
9.4. Иерархический раскрывающийся список ... 179
9.5. Меню на основе списка ... 184
9.6. Выравнивание элементов списка ... 188
9.7. Список определений ... 191

Глава 10. Таблицы ... 193
10.1. Табличные теги ... 193
10.2. Рамки таблицы ... 195
10.3. Размеры таблицы ... 199
10.4. Выравнивание содержимого ячеек таблицы ... 203
10.5. Задание параметров столбцов ... 206
10.6. Сложные таблицы ... 208
10.6.1. Расширение ячеек ... 208
10.6.2. Прокручиваемая таблица ... 212
10.7. Декорирование таблицы ... 214

Глава 11. Формы и элементы пользовательского интерфейса ... 216
11.1. Поля ввода, кнопки и переключатели: тег <*input*> ... 216
11.2. Кнопка: тег <*button*> ... 219
11.3. Комбинированный раскрывающийся список: тег <*select*> ... 222
11.4. Текстовая область: тег <*textarea*> ... 225
11.5. Декорации элементов интерфейса ... 227
11.6. Форма: тег <*form*> ... 229

Глава 12. Вставка внешнего содержимого ... 232
12.1. Графические изображения ... 232
12.1.1. Растровая графика ... 232
12.1.2. Основные форматы растровой графики ... 240
12.1.3. Векторная графика ... 241
12.1.4. Вставка графики в (X)HTML-документ ... 243

12.2. Звук и видео ... 251
12.2.1. Основные форматы звуковых и видеофайлов ... 251
12.2.2. Вставка звука и видео в (X)HTML-документ ... 252
12.2.3. Вставка FLV-видео ... 255
12.2.4. Вставка Flash-фильмов ... 257
12.3. Вставка (X)HTML-документов ... 260
12.4. Вставка элементов управления ActiveX ... 261
12.4.1. Что такое ActiveX ... 261
12.4.2. Примеры элементов ActiveX ... 262
12.5. Вставка апплетов Java ... 269
12.5.1. Что такое апплет ... 269
12.5.2. Вставка апплета посредством тега <*applet*> ... 270
12.5.3. Вставка апплета посредством тега <*object*> ... 271

Глава 13. Что такое JavaScript ... 274
13.1. Из истории ... 274
13.2. Общая характеристика языка ... 277
13.3. Вставка сценариев в (X)HTML-документ ... 278
13.4. Специальные термины и понятия ... 283

Глава 14. Объектная модель браузера и документа ... 286
14.1. Общие сведения ... 286
14.2. Доступ к объектам ... 290
14.3. Доступ к свойствам элементов документа ... 294
14.3.1. Доступ к атрибутам ... 294
14.3.2. Доступ к свойствам CSS ... 295
14.3.3. Доступ к содержимому элемента ... 300
14.4. Обработка событий ... 302
14.4.1. Привязка обработчиков событий ... 303
14.4.2. Область видимости обработчиков событий ... 309
14.4.3. Изменение поведения элементов по умолчанию ... 310
14.4.4. Программный вызов обработчика события ... 311
14.4.5. Прохождение событий ... 314
14.4.6. Информация о событии: объект *Event* ... 317
14.4.7. Основные события ... 323
14.5. Основные объекты браузера и документа ... 326
14.5.1. Объект *window* ... 326
14.5.2. Объект *screen* ... 329
14.5.3. Объект *location* ... 329
14.5.4. Объект *history* ... 331
14.5.5. Объект *navigator* ... 331
14.5.6. Объект *document* ... 333

Глава 15. Работа с основными объектами посредством JavaScript ... 336
15.1. Управление окнами и фреймами ... 336
15.1.1. Создание окон ... 336
15.1.2. Взаимодействие окон ... 338
15.1.3. Работа с фреймами ... 340
15.1.4. Окно *PopUp* в Internet Explorer ... 345

15.2. Работа с таблицами ... 346
15.3. Работа с табличными данными в текстовых файлах ... 350
15.3.1. Применение ActiveX Tabular Data Control ... 350
15.3.2. Применение объекта *XMLHttpRequest* ... 356
15.4. Работа с формами ... 358
15.4.1. Проверка данных перед отправкой ... 358
15.4.2. Баннер как форма ... 360
15.4.3. Переходы между полями по клавише <Enter> ... 361
15.5. Работа с локальным хранилищем данных ... 363
15.5.1. Cookie ... 363
15.5.2. Объект *localStorage* ... 368
15.6. Работа с графическими изображениями ... 369
15.6.1. Объект элемента <*img*> ... 369
15.6.2. Объект *Image* ... 370
15.6.3. Управление свойствами изображения ... 370
15.6.4. Предварительная загрузка изображений ... 372
15.6.5. Нетипичные применения объекта *Image* ... 375
15.7. Взаимодействие с сервером: объект *XMLHttpRequest* и AJAX ... 380
15.7.1. Объект *XMLHttpRequest* ... 381
15.7.2. AJAX ... 391
15.8. Управление во времени ... 391

Глава 16. Примеры сценариев на JavaScript ... 396
16.1. Подсветка кнопки ... 396
16.2. Меню ... 399
16.2.1. Моментально раскрывающееся вертикальное меню ... 399
16.2.2. Плавно раскрывающееся меню ... 401
16.3. Раскрывающийся комбинированный список ... 403
16.4. Иерархический раскрывающийся список ... 404
16.5. Эффект пишущей машинки ... 406
16.6. Отображение кода на странице ... 407
16.7. Перемещение элементов мышью ... 411
16.8. Движение по траектории ... 414
16.8.1. Движение по произвольной кривой ... 415
16.8.2. Движение по эллипсу ... 417
16.9. Рисование линий посредством <*div*> ... 418
16.9.1. Прямая линия ... 419
16.9.2. Произвольная линия ... 422
16.9.3. Графики зависимостей ... 426
16.9.4. Перерисовка линий ... 428
16.10. Рисование посредством <*canvas*> ... 429
16.10.1. Как вставить <*canvas*> в (X)HTML-документ ... 429
16.10.2. Фигуры и линии ... 431
16.10.3. Градиенты ... 437
16.10.4. Трансформации ... 438
16.10.5. Импорт растровых графических изображений ... 440
16.10.6. Анимация ... 443
16.10.7. Композиция графики ... 446
16.10.8. Текст ... 448

16.11. Дата и время ... 450
16.11.1. Отображение даты и времени в виде текста ... 450
16.11.2. Часы ... 451
16.11.3. Вечный календарь ... 453
16.12. Виджет ... 458

Приложение 1. Перечень тегов HTML 5 ... 465

Приложение 2. Перечень параметров CSS ... 470
Позиционирование ... 470
Размеры ... 470
Цвет и фон ... 471
Текст ... 471
Шрифты ... 471
Блоки (поля, отступы и границы) ... 471
Таблицы ... 472
Печать ... 472
Интерфейс ... 472
Звук ... 472
Прочее ... 472

Предметный указатель ... 473

Предисловие ко второму изданию

Второе издание настоящей книги кардинально отличается от предыдущего как по структуре, так и по содержанию. Существенно расширены главы, посвященные языку разметки гипертекста HTML и CSS. При этом рассматриваются некоторые новые элементы HTML 5 и CSS 3, которые поддерживаются всеми ведущими браузерами.

Я старался раскрыть возможности (X)HTML, CSS и JavaScript для разработки Web-сайтов, не претендуя на полноту изложения. Это не справочник, а расширенное практическое руководство по самостоятельному изучению разнообразных средств и приемов Web-программирования, рассчитанное в первую очередь на начинающих.

Введение

Настоящая книга предназначена для начинающих Web-дизайнеров, а также для тех, кто хотел бы создать свой собственный сайт и опубликовать его в Интернете. Для понимания материала книги каких-либо специальных знаний, кроме обычных навыков владения персональным компьютером, не требуется. Из программного обеспечения понадобятся обычный текстовый редактор (например, Блокнот Windows) и Web-браузер. Желательно иметь выход в Интернет.

Дизайн (англ. design — проектирование, конструирование, лежащая в основе схема, модель) есть деятельность, направленная на определение основных качеств изделий, важных как с точки зрения потребителей, так и производителей. Внешний облик с точки зрения эстетики, а также структурные и функциональные межкомпонентные связи и отношения с окружающей средой — вот вопросы, которыми занимаются дизайнеры при проектировании дизайна изделия.

Термин "дизайн" появился еще в XVI веке и первоначально использовался для обозначения идеи, концепции, эскиза будущего художественного произведения. В середине XX века под дизайном понимали творческую деятельность, цель которой — определение формы предметов, производимых промышленностью. При этом формообразование рассматривалось не только в смысле внешнего вида изделия, но и с точки зрения технологии его производства и последующего использования. Недаром профессионалы обычно применяли слово "дизайн" лишь как сокращение для понятия "индустриальный дизайн". Сейчас, в постиндустриальную эпоху развития цивилизации, термин "дизайн" стал более многозначным и широко употребительным. Вместе с тем, общим местом его различных толкований остается синтез художественного и технологического аспектов проектной деятельности. Дизайн должен обеспечить гармонию внешнего облика и внутреннего содержания проектируемого объекта как единого целого.

Объектом дизайна может быть практически все, что угодно — интерьер помещения, мебель, автомобиль, рекламный постер, Web-сайт и др. В дизайне выделяют отрасли: промышленный, ландшафтный, графический, архитектурный, информационный, дизайн одежды и др. Дизайн Web-сайтов и Web-приложений или, иначе говоря, Web-дизайн иногда рассматривают как разновидность графического дизайна — отрасли, к которой относят также полиграфию, разработку шрифтов и товарных знаков. При такой классификации делается акцент на художественном оформлении страниц сайта. Однако правильнее, на мой взгляд, рассматривать Web-дизайн как отрасль Web-разработки и разновидность дизайна, цель которого заключается в проектировании и создании поль-

зовательских интерфейсов для сайтов и Web-приложений. Именно с такой точки зрения и предлагается рассмотреть Web-дизайн в данной книге.

К настоящему времени о Web-дизайне написано немало книг и создано огромное количество публикаций в Интернете. Тем не мене это — по-прежнему актуальная тема, особенно для новичков. Приступая к разработке сайта, все хотят создать, как часто говорят, стильный дизайн. Что это такое, трудно определить словами, как и все относящееся к искусству. Так, например, в архитектуре различают романский и готический стили, барокко, классицизм, конструктивизм и др. При этом вряд ли можно давать им оценку по шкале хорошо-плохо. Однако такую оценку можно применять к конкретным сооружениям любого стиля. То же самое относится и к сайтам.

Разработка сайта больше напоминает создание архитектурного проекта, нежели верстку газеты, журнала или книги. Лишь очень простые сайты (например, так называемые сайты-визитки) можно уподобить рекламным листкам и афишам — флайерам и постерам. Так, архитектор не просто рисует внешний облик будущего здания, но и учитывает при этом его функциональное назначение, окружающую среду, материалы, технологию строительства и т. п. Он решает задачи эстетического плана в технологическом контексте. Проектируемое им здание должно иметь внутри технические системы электро-, тепло- и водоснабжения, канализации и телекоммуникации, гармонировать с ландшафтом, вписываться в уже имеющуюся инфраструктуру, быть удобным в использовании и т. д. Все это весьма разнородное хозяйство необходимо как-то скомпоновать, чтобы оно не мешало использовать здание по своему прямому назначению и в то же время не создавало излишних трудностей при технической эксплуатации (в частности, при ремонте).

Создавая сайт, также приходится учитывать его назначение, имеющиеся средства обеспечения функциональности, удобства последующего сопровождения и развития, сложившиеся у пользователей стереотипы и многое другое. Сайт создается для посетителей и потому должен быть удобен для них и в художественном отношении не безобразен. Например, вы же не будете применять интерфейс, который может ввести посетителя сайта в замешательство. Интерфейс может иметь необычное внешнее оформление, но при этом должен оставаться интуитивно понятным.

Сайт — это средство визуального представления информации. Как и собственно представляемая информация (тексты, графические изображения, аудио и видео), он имеет внешний видимый облик, образуемый композицией и оформлением информационного содержимого вместе с элементами пользовательского интерфейса. Композиция и оформление элементов сайта могут способствовать легкому, быстрому и удобному получению представляемой информации, а могут, наоборот, затруднять ее поиск и усвоение. Красивый внешний вид сайта привлекает и удерживает внимание посетителей. Плохой дизайн может даже "похоронить" саму по себе очень важную и интересную информацию.

Итак, композиция и оформление элементов — то, что в основном определяет внешний облик сайта. А раз так, то его можно нарисовать еще до начала разработки технической части. Обычно это делается с помощью какого-нибудь графического редактора, например Adobe Photoshop, CorelDRAW или GIMP. Создание графического эскиза главной или нескольких страниц сайта составляет лишь одну часть Web-дизайна, называемую графическим или художественным дизайном сайта. Графический дизайн обычно пред-

ставляют заказчику перед заключением договора на разработку сайта в целом. Он нередко служит отправной точкой для технического задания на разработку и ориентиром для разработчиков структуры и функциональности сайта, т. е. технической части проекта. Графический дизайн — дело художественного вкуса, эстетики и психологии, с трудом поддающееся формальному описанию, и в предлагаемой книге по существу не рассматривается. А вот композиция структурных элементов и обеспечение основных функциональных возможностей сайта — главная тема данной книги.

Несмотря на огромное внешнее разнообразие сайтов в сети, подавляющее большинство из них имеют сходные структуру и функциональность. Именно поэтому в Интернете сейчас можно легко найти готовые к применению шаблоны для страниц сайта, которые нетрудно адаптировать к своим задачам и сообразно со своими художественными замыслами и вкусом. Но чтобы сделать это, необходимо понимать, как и из чего устроены страницы и шаблоны. Если вы намерены в дальнейшем заниматься созданием сайтов профессионально, то вам следует научиться самому создавать шаблоны страниц. Страницы сайта создаются посредством HTML (HyperText Murkup Language — язык разметки гипертекста), CSS (Cascading Style Sheets — каскадные таблицы стилей) и сценариев (скриптов), написанных на специальных языках, таких как JavaScript, PHP и др. Применению этих средств к решению типовых задач сайтостроительства посвящена настоящая книга.

Предлагаемую вашему вниманию книгу желательно читать последовательно, выполняя на компьютере рассматриваемые примеры. Общие понятия, а также вопросы, связанные с разработкой, публикацией и дальнейшим сопровождением сайта в целом, изложены в *главе 1*. В *главах 2—12* рассматривается применение HTML и CSS. *Главы 13—16* посвящены сценариям (скриптам) на языке JavaScript. Справочные сведения о HTML и CSS приведены в приложениях, которые следует рассматривать лишь как ориентиры: точные сведения следует искать на сайтах, посвященных соответствующим спецификациям.

Все затронутые в книге темы иллюстрируются большим количеством примеров. Многие из них воспроизведены на моем сайте "Сам себе Web-дизайнер" по адресу: **http://dunaevv1.narod.ru**.

ГЛАВА 1

Что такое Web-дизайн

Суть дизайна — формообразование. Форма — внешняя оболочка, представление и способ существования содержания любой вещи или явления, в том числе и сайтов, созданием которых занимаются Web-дизайнеры.

Вы можете заняться разработкой сайта для себя или стороннего заказчика. В любом случае работа состоит из нескольких этапов. Некоторые из них особенно важны, если заказчик и подрядчик (исполнитель) — разные лица. С заказчиком бывает трудно договориться, по крайней мере на первых этапах, не столько из-за его привередливости или скупости, сколько из-за того, что зачастую он сам не вполне осознает, что конкретно ему нужно. Идеальная ситуация — заказчик выдает вам техническое задание, достаточно ясное и подробное, чтобы немедленно приступить к проектированию. В реальности же техническое задание на проект чаще всего вырабатывается совместными усилиями исполнителя и заказчика итеративно.

В данной главе мы рассмотрим в общих чертах основные этапы создания Web-сайта. Попробуйте представить себя как в роли исполнителя, так и в роли заказчика сайта.

1.1. С чего начать?

Как известно, хорошее начало — более половины всего дела. А начать следует с выяснения цели разработки сайта, т. е. того, для чего он нужен. Например, сайт может предназначаться для рекламы и/или продажи товаров и услуг (интернет-магазин), представления сведений о фирме или персоне (сайт-визитка), размещения информационных статей или медиаресурсов и т. д. Для информационных сайтов также важна тема: культура, наука, искусство, образование, технология, развлечения и т. д. При этом важно знать, какого типа информацию (тексты, графика, видео, аудио) и какие сервисы должен предлагать сайт. Желательно хотя бы приблизительно оценить объем публикуемой информации и частоту ее обновления. Сайты бывают статическими, содержимое которых обновляется время от времени, и динамическими, информация в которых может изменяться чуть ли не ежеминутно (блоги, форумы, гостевые книги). Неплохо также иметь предположения о потенциальной аудитории сайта (социальная и профессиональная группы, образование, возраст). Далее заказчик обычно высказывает свои предположения и пожелания о том, как должна выглядеть главная (домашняя, начальная) страница сайта. Здесь речь может зайти о стиле, цветовой гамме, композиции и т. п. Постарайтесь выяснить, каким должен быть дизайн с эмоциональной точки зрения за-

казчика: спокойным или вызывающим, строгим или легкомысленным, жизнерадостным или мрачным, пестрым или аскетичным, насыщенным деталями или лаконичным. Ваша задача — выяснить вкусы заказчика. Разумеется, они могут не совпадать с вашими. Но о вкусах, как говорят, не спорят. В данной книге я всячески пытался избежать обсуждения тем, связанных с вкусовщиной. Не критикуйте вкус заказчика, но учитывайте его, предлагая свое решение.

Заказчик может показать вам несколько уже существующих сайтов, которые ему нравятся, но при этом пожелать, чтобы его будущий собственный сайт обладал явно выраженной индивидуальностью. Казалось бы, здесь возникает противоречие: с одной стороны, требуется выдержать некий стиль, схожесть с уже существующими дизайнами, а с другой — обеспечить заметное отличие от последних. Это противоречие и должен разрешить дизайнер. В архитектуре, например, оно ведь разрешается. Так, среди сохранившихся соборов готического стиля много очень похожих, как близнецы, друг на друга, но есть и неповторимые, например Нотр-Дам в Реймсе, Нотр-Дам в Париже и собор Святого Павла в Страсбурге.

Не следует забывать, что при всех изысках внешнего вида сайта его потенциальный посетитель не должен испытывать ни малейших затруднений относительно элементов управления (пользовательского интерфейса). Ссылки, кнопки, поля ввода данных и т. п. должны легко узнаваться, не требуя от посетителя экспериментальных исследований с помощью мыши, чтобы выяснить, с чем именно он имеет дело. Главная цель дизайна — сделать сайт как можно более удобным и облегчить доступ к находящейся на нем информации. Удивить и заворожить посетителя — дело третьестепенное, если не десятое.

Хорошо, если заказчик ограничивается лишь общими замечаниями относительно облика главной страницы. Тем не менее часто бывает, что он пытается излишне детализировать задание на дизайн. Например, заказчик может явно указать, что заголовки должны мигать, там-то следует разместить Flash-ролики, а фон страницы должен иметь вид кирпичной стены. Казалось бы, это должно упростить задачу дизайнера. Но, как правило, подобного рода указания существенно ограничивают необходимые для творчества степени свободы дизайнера и, наоборот, усложняют его задачу. Может статься, понравившийся заказчику дизайн вызовет отвращение у самого дизайнера. Тогда, разумеется, можно утешиться тем, что заказчик всегда прав или же просто отказаться от заказа. Однако на практике дизайн, непосредственно воплощающий прямые и слишком детализированные указания заказчика, часто потом не нравится и ему самому. В лучшем случае заказчик получит урок, что "пироги должен печь пирожник, а сапоги тачать — сапожник", а в худшем — он обвинит вас в непрофессионализме. И тогда дизайнер приступает к разработке макета или эскиза сайта сообразно с собственными знаниями, опытом, навыками и вкусом, но, конечно, учитывая общие пожелания и пристрастия заказчика. Впрочем, опытный дизайнер должен прийти к этому, не тратя излишне свои силы и время на бесплатные уроки. Он просто заранее имеет достаточно весомые аргументы, чтобы убедить заказчика довериться профессионалу. Другое дело, если заказчиком и исполнителем является одно и то же лицо, причем новичок. В этом случае редко когда удается сразу, без переделок, создать хороший проект, потому что обычно "первый блин — комом".

Если вы создаете сайт для себя лично и в первый раз в жизни, то при выборе дизайнерского решения критически посмотрите на чужие разработки и мысленно представьте

себе, что вы делаете не первый и не последний сайт в своей жизни. Даже если в дальнейшем вы не будете этим заниматься, данный психологический прием убережет вас от неуместного загромождения своих страниц всякими экзотическими штучками. Как говорил Л. Толстой, "главное свойство во всяком искусстве — чувство меры". А меру мудрецы считали синонимом мудрости. Дело в том, что новички в своем первом проекте стараются продемонстрировать, что они очень много знают и умеют. Аналогично, многие заказчики сайтов для своих фирм нередко хотят, чтобы на их сайте присутствовало все самое модное и "передовое", чуть ли не все такое, что они уже где-то видели, и даже более того. Так что, неопытные дизайнер и заказчик могут совместными усилиями породить чудовище не только по внешнему виду, но и в структурно-функциональном смысле, отталкивающее посетителей и внешним видом, и плохой функциональностью. На мой взгляд, красивое и гармоничное не может быть чрезмерно сложным. По крайней мере, сложность создаваемого объекта не должна быть видна.

1.2. Графический дизайн

Рассмотренный в предыдущем разделе этап Web-дизайна считается предварительным, поскольку он обычно не завершается заключением письменного договора или хотя бы устного соглашения между заказчиком и исполнителем. Перед тем как заключить такое соглашение заказчик, как правило, хочет получить от исполнителя эскиз (макет) главной и, возможно, еще нескольких страниц будущего сайта, поскольку "лучше один раз увидеть, чем сто раз услышать". Так возникает задача графического дизайна сайта.

Заказчик может потребовать графический дизайн не только для того, чтобы после возможной его коррекции принять наконец-то решение о дальнейшей разработке сайта. Графический дизайн может представлять собой и самостоятельный объект договора, т. е. рассматриваться как законченный продукт. Его разработка может быть поручена одному исполнителю, а все остальное (воплощение в кодах, заполнение содержимым и т. д.) — другому. Так что, приступая к графическому дизайну, вы должны отдавать себе отчет о статусе этой разработки: является ли она только промежуточным или же окончательным этапом вашей работы. Нередко разработку и оплату графического дизайна оформляют специальным соглашением.

Если графический дизайн рассматривается как промежуточный этап разработки сайта в целом, то вам следует подготовить несколько (два-три) его вариантов. В результате обсуждения этих вариантов дизайна с заказчиком в идеале должен быть выбран один из них. Однако бывает и так, что заказчик желает сделать некий гибрид из представленных кандидатов. В этом случае возникает опасность разрушения концептуального и художественного единства вашего замысла проекта, о которой уже говорилось в предыдущем разделе. Тем не менее вы должны быть готовы к итеративному процессу рассмотрения вариантов графического дизайна. Окончательный выбор все равно следует пока рассматривать лишь в качестве рабочего, поскольку дальнейшая разработка сайта может потребовать внесения коррективов и в его графическое оформление. Однако если графический дизайн хорош не только с эстетической, но и с технологической (структурно-функциональной) точки зрения, то необходимая впоследствии его коррекция будет и минимальна, и относительно безболезненна. Иначе говоря, создавая эскизы страниц сайта, нужно все время помнить о технологических аспектах.

Теперь вкратце рассмотрим, как делают графический дизайн страниц сайта. Напомню, что заказчику обычно требуется дизайн одной-двух страниц — главной, которая загру-

жается в браузер первой, и той, которая отображается при переходах по ссылкам. Различные страницы сайта могут иметь разную компоновку и средства навигации, но должны быть выполнены в едином стиле. Это хорошо не только в эстетическом смысле, но и с точки зрения поддержки ориентации потенциального посетителя в информационном пространстве. Единство стиля различных страниц создает ненавязчивое ощущение, что мы находимся в пределах одного и того же сайта. Самый простой способ обеспечения единства стиля различных страниц — применять единые цветовые решения, параметры шрифтов и средств навигации.

Конечный результат графического дизайна можно представить в виде:

- графических изображений страниц в целом;
- шаблонов страниц, т. е. набора файлов с HTML- и CSS-кодами и графическими изображениями.

Графическое изображение страницы сайта можно нарисовать на бумаге с различной степенью подробности и тщательности отделки. Этот способ многие используют при создании личного сайта. Однако стороннему заказчику нужна не бумага, а файл графического формата, и тогда дизайн выполняют с помощью графических редакторов, таких как коммерческие Adobe Photoshop, CorelDRAW и свободно распространяемые GIMP, Inkscape.

В графическом редакторе создают изображение страницы целиком, включая элементы пользовательского интерфейса (гиперссылки, кнопки, переключатели, поля ввода данных и т. п.). Если графический дизайн не является предметом отдельного заказа, а служит лишь в качестве эскиза или прототипа страницы для согласования с заказчиком всего проекта создания сайта, то его сохраняют в файле растрового формата, например, PNG или JPEG. В противном случае (специальный заказ) графический дизайн страницы выполняют как многослойное изображение, в котором каждый элемент страницы рисуется в отдельном слое, наложенном на предыдущие. Это делается для того, чтобы элементы графического дизайна можно было в дальнейшем скорректировать или заменить другими, а также использовать при реализации страниц сайта. Элементы графического дизайна имеют самостоятельную ценность, и желательно иметь возможность извлечения их из первоначальной целостной конструкции. В данном случае изображение страницы сохраняют либо в растровом формате с поддержкой слоев (например, PSD, TIFF), либо в векторном формате (например, EPS). Файлы таких форматов можно открыть и изменить во многих графических редакторах, например Adobe Photoshop, Adobe Illustrator, CorelDRAW, GIMP и др.

Графический дизайн в виде картинки в дальнейшем используется как ориентир для разработки структуры и функциональности страницы. Разработчик, взглянув на такое графическое изображение и сопровождающее его словесное описание, должен придумать, какими программными средствами воплотить замысел художника. Например, он решает, сделать ли название сайта как графическое изображение, или вставить его как соответствующим образом отформатированный текст, т. е. применяя HTML и CSS; создать ли обычные гиперссылки или же графическую карту ссылок.

Вместо того чтобы рисовать картинку страницы, можно сразу создать ее шаблон, используя язык разметки HTML, каскадные таблицы стилей (CSS) и, если потребуется, графические изображения. Шаблон — это прототип страницы без информационного

содержимого, с ограниченной функциональностью или вообще без таковой (например, не работают ссылки, меню, формы и т. п.). Преимущество данного способа заключается в том, что не требуется в совершенстве владеть графическим редактором и уметь создавать с его помощью оригинальные изображения, навыки рисования в данном случае не являются необходимыми. Важно также, что шаблон легко модифицировать практически на любом этапе разработки сайта. Кроме того, его в дальнейшем можно использовать (немного скорректировав) и для других сайтов. Иначе говоря, использование шаблона позволяет путем относительно небольших модификаций его параметров получить существенные изменения внешнего вида Web-страниц. Если ваша задача состоит в разработке только лишь графического дизайна, то конечный результат вашей работы можно получить просто как скриншот (снимок экрана) загруженного в браузер шаблона.

Даже если графический дизайн выполнен как картинка, последующая разработка сайта все равно предполагает создание прототипов (шаблонов) его страниц, которые можно загрузить и просмотреть в браузере. Так что создание шаблона — не просто один из методов графического дизайна, но одновременно и этап разработки дизайна сайта вообще.

1.3. Шаблоны страниц

Готовые шаблоны нетрудно найти в Интернете, а можно сделать их и самому. Корректировать имеющиеся и создавать новые шаблоны, наполнять их содержимым (т. е. делать собственно Web-страницы) можно с помощью специальных программ визуальной разработки, таких как Adobe (Macromedia) Dreamweaver, Microsoft FrontPage или Microsoft SharePoint Designer. Кстати, Microsoft SharePoint Designer 2007 предоставляется бесплатно.

Системы визуальной разработки популярны в среде новичков, не желающих возиться с HTML- и CSS-кодами, не говоря уж о скриптах. Тем не менее их используют и профессионалы, особенно в условиях разработки одновременно нескольких сайтов, когда необходимо существенно повысить свою производительность. Однако следует иметь в виду, что системы визуальной разработки автоматически создают соответствующие программные коды, причем далеко не оптимальные ни по объему, ни по структуре, ни по выбору привлеченных HTML-элементов. В автоматически созданных кодах нелегко разобраться, возможно придется потратить на это неделю. А ведь сайт, скорее всего, придется сопровождать (дополнять, модифицировать) после сдачи заказчику. В данной книге мы будем рассматривать вопросы дизайна в плане создания шаблонов вручную, т. е. методом прямого программирования. Такой подход, разумеется, не исключает практическое применение систем визуальной разработки, а наоборот, позволяет увеличить эффективность их использования. Действительно, зная HTML и CSS, вы можете вносить коррективы непосредственно в автоматически сгенерированный код и в то же время пользоваться удобными инструментальными средствами визуального проектирования.

Суть создания шаблона заключается в том, что:

- разрабатывается компоновочная схема или, другими словами, структурная разметка страницы сайта. Такая схема может быть представлена в виде таблицы, ячейки которой определяют места размещения оформительской графики (логотип, элементы

украшения), средств пользовательского интерфейса (меню, кнопки, ссылки и другие элементы навигации и взаимодействия с пользователем) и собственно содержимого (контента, как еще говорят) страницы. Для сайта может потребоваться более одной компоновочной схемы: одна для главной страницы, а другие — для страниц перехода по ссылкам с главной. Очевидно, тривиальная схема представляется одноячеечной таблицей. Простая схема содержит две ячейки. Наиболее часто встречаются схемы из трех-пяти ячеек. Примеры типичных компоновочных схем показаны на рис. 1.1. Выбор конкретной схемы зависит от многих факторов, основные из которых — характер размещаемых в ее ячейках элементов (графические изображения, средства навигации, тексты и т. д.), степень важности, а также тематической и структурной разнородности информации;

- для ячеек компоновочной схемы определяются цвет фона и/или фоновые графические изображения, а также параметры границ;
- для текстовой информации (прежде всего, для шрифтов) определяются стилевые параметры. Причем такие параметры могут задаваться и для всей страницы, и отдельно для каждой ее ячейки с текстом.

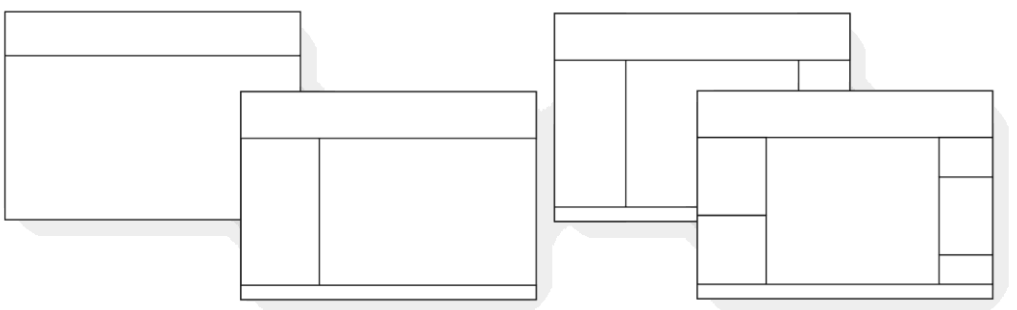


Рис. 1.1. Типовые варианты компоновочных схем страниц сайта

Наконец, определяется, каким образом компоновочная схема будет позиционироваться в окне браузера: должны ли изменяться ее размеры в зависимости от размеров клиентской области браузера или нет, должна ли она центрироваться относительно краев этой области.

1.4. Реализация проекта

После выбора компоновочной схемы страницы сайта приступают к реализации проекта, т. е. программированию с использованием HTML, CSS, а если понадобится, то и языков сценариев, таких как JavaScript, PHP и др. На данном этапе разработки сайта следует учесть несколько важных моментов:

- не исключено, что на какой-то стадии реализации придется вернуться к пересмотру компоновочной схемы, а может быть, скорректировать и графический дизайн. Вероятность такого развития событий тем меньше, чем лучше выполнена работа на предыдущих этапах. Последнее означает, что графический дизайн и структура страницы выполнены с учетом технологических особенностей предстоящей реализации проекта в целом;

- разработанный сайт после публикации в Интернете, скорее всего, придется сопровождать: изменять частично или полностью информационное содержимое страниц, добавлять новые страницы и т. п. Сложные технические устройства, например, должны обладать целым рядом эксплуатационных свойств, в числе которых — ремонтопригодность. Аналогично, хорошо разработанный сайт также должен обладать хорошими эксплуатационными качествами. Нередко заказчик особо требует, чтобы разработчик сайта обеспечил ему простоту и удобство обновления информационного содержимого без помощи разработчика и даже без обращения к сведущему в Web-дизайне специалисту. Поэтому уже на этапе разработки следует предусмотреть такую возможность;
- современный сайт должен быть инвариантен (одинаково выглядеть и функционировать) относительно всех современных ведущих браузеров, таких как Microsoft Internet Explorer, Mozilla Firefox, Opera, Google Chrome, Apple Safari. При этом инвариантность должна достигаться не только для самых свежих, но и для хотя бы самых распространенных старых версий. В настоящее время, пожалуй, только Internet Explorer широко представлен в сети сразу несколькими своими версиями (6—9). Дело в том, что многие используют тот браузер (а это, как правило, интегрированный в Windows Internet Explorer), который они получили с приобретением компьютера. Поэтому следует проверять разрабатываемый сайт по крайней мере для версии 8 — последней для все еще широко распространенной Windows XP и последующих версий браузера, работающих под Windows 7 и более поздними выпусками данной операционной системы. Пользователи других браузеров в подавляющем числе случаев имеют самые последние их версии. Так что при разработке сайта следует обзавестись несколькими браузерами, хотя бы четырьмя: Internet Explorer, Firefox, Opera и Chrome. Ко времени выхода в свет данной книги доля пользователей указанных браузеров составляла более 97%. В настоящее время браузеры распространяются через Интернет свободно.

Теперь рассмотрим, каким минимальным оснащением должен обладать разработчик сайта. Прежде всего, нужен обычный текстовый редактор вроде блокнота Windows. Лучше использовать более специализированный редактор, обеспечивающий подсветку синтаксиса программных кодов на различных языках, например, свободно распространяемый Notepad++ (**http://notepad-plus-plus.org**). В текстовом редакторе создаются коды HTML и CSS, а также сценарии на скриптовых языках, таких как JavaScript и PHP. Само собой разумеется, нужен браузер, чтобы просматривать результаты своей работы. Точнее, нужны несколько ведущих браузеров для проверки, что создаваемый сайт достаточно инвариантен относительно них. Для подготовки графических элементов потребуется графический редактор. Сейчас для этой цели широко применяется коммерческий Adobe Photoshop, но вполне можно использовать достаточно мощный, удобный и бесплатный GIMP (**http://www.gimp.org**). Современные ведущие браузеры оснащены средствами разработчика, которые являются хорошим подспорьем при создании и отладке страниц сайта. Так, вы можете просмотреть HTML-код, параметры CSS, скрипты, сообщения об ошибках, относящиеся к загруженной в браузер странице. Наконец, необходим FTP-клиент — программное обеспечение закачки файлов на сервер по протоколу FTP (File Transfer Protocol). В настоящее время существует множество бесплатных и хороших FTP-клиентов. Кроме того, на сайтах компаний, предоставляющих хостинг, обычно есть и сервис для FTP-обмена файлами.

Если вы создаете сайт без серверной части (т. е. без серверных сценариев), то минимальный набор перечисленных средств достаточен. Чтобы протестировать страницу сайта, вы можете на локальном компьютере открыть в браузере соответствующий HTML-файл и посмотреть, работает ли страница согласно вашим требованиям и ожиданиям и как она выглядит.

Серверная часть сайта не является необходимой, если вам не нужно принимать, обрабатывать и сохранять данные, введенные посетителем, а также вести базы данных. В противном случае, например, при создании блога, форума, гостевой книги, интернет-магазина и т. п. без серверной части не обойтись. Тогда для тестирования и отладки сайта потребуется Web-сервер.

Где взять Web-сервер? Во-первых, вы можете воспользоваться Web-сервером хоста в Интернете, на котором в дальнейшем собираетесь опубликовать свой сайт или же на другом, который доступен вам и подходит для отладочных целей. В данном случае необходимо перенести уже разработанные файлы на удаленный сервер и далее производить отладку в режиме онлайн, заменяя при необходимости устаревшие версии файлов новыми. В настоящее время имеется довольно много как бесплатных, так и платных хостингов. Однако при выборе хостинга следует учитывать состав предоставляемых возможностей. Например, популярный бесплатный хостинг Яндекс.Народ (**http://narod.yandex.ru**) не дает возможности использовать серверные сценарии и ограничивает объем файлов. Хостинг "Веб Сервис" (**http://www.webservis.ru**) предоставляет возможность применять серверные сценарии на PHP бесплатно, но не позволяет публиковать мультимедийные файлы некоторых форматов. Хостинг uCoz (**http://www.ucoz.ru**) предоставляет удобный сервис конструктора в режиме онлайн, обеспечивающий создание динамических сайтов (блогов, форумов, гостевых книг). Однако статические сайты, состоящие только из HTML-документов и не применяющие средства взаимодействия с посетителями, не приветствуются данным хостингом.

Общее ограничение, накладываемое всеми бесплатными хостингами:

- ❒ публикуемый сайт не должен иметь коммерческой направленности (так что интернет-магазины на бесплатных хостингах устраивать не стоит);
- ❒ на страницах вашего сайта хостинг разместит свой баннер. Он может появляться где попало и портить внешний вид страниц, даже если его можно закрыть щелчком мыши. Такова плата за бесплатный хостинг;
- ❒ объем отдельных файлов и общий объем дискового пространства для сайта в целом ограничиваются. Например, под сайт может быть отпущено несколько сот мегабайтов, но отдельный файл не должен превышать 2 Мбайт.

Разумеется, платный хостинг предлагает более широкий набор услуг при меньших ограничениях. Выбираемый для отладки сайта хостинг должен иметь параметры, максимально близкие к параметрам того хостинга, на котором вы собираетесь опубликовать окончательную версию сайта. В идеале это должен быть один и тот же хостинг. Отладка сайта в режиме онлайн хороша тем, что вам не нужно самому устанавливать и, в большинстве случаев, настраивать Web-сервер, а также модули, выполняющие серверные сценарии, системы управления базами данных и т. п. Все это сделает администратор платного хостинга. Если же хостинг бесплатный, то вам нужно лишь адаптироваться к предлагаемому набору уже настроенных средств, получив необходимую информацию на сайте хостинговой компании либо у ее администратора. Однако вам

придется заплатить за трафик. Тем не менее, данный режим широко применяется при создании простых и средних по сложности сайтов.

Во-вторых, вы можете отлаживать разрабатываемый сайт в режиме оффлайн, установив на локальном компьютере Web-сервер самостоятельно, например, Apache или Internet Information Services (IIS). Кроссплатформенный Apache распространяется свободно (**http://www.apache.org**), а IIS входит в дистрибутивы Windows XP Professional, Windows 7 Professional/Enterprise/Ultimate, но не устанавливается как служба автоматически. При необходимости IIS следует найти среди компонентов Windows и установить самостоятельно.

Серверные сценарии пишут на различных языках. Одним из наиболее популярных является PHP, и его поддерживает большинство Web-серверов в Интернете. Если вы установили на локальном компьютере Web-сервер, то вам придется также установить и модуль PHP, распространяемый свободно (**http://www.php.net** или **http://ru.php.net**).

Если ваш сайт должен взаимодействовать посредством серверных сценариев с базами данных, то потребуется еще установить и систему управления базами данных (СУБД), причем такую, которая поддерживается Web-сервером того хостинга, на котором вы собираетесь опубликовать свой сайт. Это может быть, например, Microsoft SQL Server (для платформы Windows) или кроссплатформенная СУБД MySQL, распространяемая бесплатно (**http://www.mysql.ru**). Практически все серверы, поддерживающие PHP, поддерживают также и MySQL.

Самостоятельная установка на локальном компьютере Web-сервера, модулей PHP и СУБД — несложная задача, и начинающему Web-дизайнеру было бы неплохо попытаться ее решить хотя бы в ознакомительных целях, даже если принято решение работать в режиме онлайн. Однако она может быть существенно упрощена, если воспользоваться какой-нибудь системой автоматизации разработки сайтов, например бесплатной системой WebMatrix от Microsoft (**http://www.microsoft.com/web/webmatrix**). WebMatrix — среда разработки на платформе Windows с хорошим пользовательским интерфейсом, содержащая все необходимое для создания сайтов: Web-сервер IIS Express, СУБД SQL Server Compact, среды программирования PHP и ASP.NET, удобные редакторы кодов, средства публикации сайта на хостинге по протоколам FTP, FTPS и WebDeploy, функцию анализа сайта с целью выработки рекомендаций по его оптимизации для поисковых систем (т. е. функцию SEO — Search Engine Optimization) и др.

В данной книге мы не будем рассматривать, как пользоваться системами автоматизации разработки и сопровождения сайтов, а остановимся на выполнении необходимых операций вручную. Это позволит лучше понять, как устроен сайт и как его модифицировать.

При любом варианте (онлайн или оффлайн) отладки разрабатываемого сайта вам следует продумать структуру папок, в которых будут храниться файлы. Имена папок и файлов должны быть в латинском нижнем регистре. Создайте на локальном компьютере корневую папку вашего сайта, например mysite. HTML-файл главной (домашней, начальной) страницы должен иметь имя index.html или index.htm и находиться в корневой папке сайта. Все остальные файлы сайта могут располагаться в той же папке, но это неудобно, особенно если их несколько десятков или сотен. Так, желательно хранить файлы каскадных таблиц стилей, графические изображения, скрипты, документы и т. п.

в отдельных папках. Пример файловой структуры сайта показан на рис. 1.2. Разумеется, иерархия и имена папок могут быть иными. Так, файлы дополнительных страниц сайта могут располагаться в отдельных папках, например page1, page 2 и т. д. Главное — чтобы структура папок была удобна вам как разработчику, а также тем, кто будет в дальнейшем сопровождать сайт. Следует иметь в виду, что имя папки для серверных сценариев нередко предопределяется хостингом. Например, это может быть папка cgi-bin.

Рис. 1.2. Пример файловой структуры сайта

При публикации сайта в сети или при использовании Web-сервера на локальном компьютере *содержимое* корневой папки сайта (а не сама корневая папка) должно быть перенесено в корневую папку (домашний каталог) Web-сервера. Например, если ваш сайт на локальном компьютере имеет файловую структуру как на рис. 1.2, то в корневой каталог Web-сервера следует перенести не папку mysite, а вложенные в нее другие папки и файл index.html. Иногда хостингом предопределяется подпапка корневой папки Web-сервера, в которой вы должны разместить содержимое локальной корневой папки сайта. Замечу, что при работе с локальным Web-сервером можно указать, какую папку следует считать домашним каталогом последнего, чтобы не переносить файлы в каталог, принятый по умолчанию.

Чтобы открыть HTML-файл в браузере при работе с локальным Web-сервером, следует в адресной строке указать URL-адрес, а не просто путь к файлу на диске: **http://localhost/*имя_файла*** или **http://*имя_сервера*/*имя_файла***. Здесь ***имя_сервера*** — сетевое имя или цифровой IP-адрес компьютера, на котором установлен Web-сервер; ***имя_файла*** — имя требуемого файла, содержащее при необходимости путь к нему относительно домашнего каталога Web-сервера. Если сервер установлен на вашем локальном компьютере, то вместо конкретного имени сервера можно указать **localhost**, например, **http://localhost/docs/page1.html**. Очевидно, **http://localhost** — URL-адрес главной страницы сайта, т. е. файла index.html или index.htm. Подробно он записывается либо как **http://localhost/index.html**, либо как **http://localhost/index.htm** в зависимости от того, какое именно расширение имеет файл главной страницы.

1.5. Наполнение информационным содержимым

В идеале, наполнение страниц сайта информационным содержимым (контентом) — отдельная задача, а на практике она оказывается тесно связанной с дизайном, в том числе и графическим. Какая-то часть контента, полученная от заказчика, анализируется и форматируется разработчиком в соответствии с общей концепцией дизайна сайта. Например, оформление названия сайта, заголовков рубрик, гиперссылок, данных о кон-

тактах и т. п. выполняется дизайнером с самого начала так, как это нужно для публикации в Интернете. Однако обычно сайт справедливо рассматривается заказчиком как система представления информации, первоначально созданной и оформленной посредством других весьма разнообразных средств, таких как текстовый редактор (например, Microsoft Office Word), фото и видеокамера, графический редактор (например, CorelDRAW) и т. д. В этом случае требуются мероприятия по преобразованию исходного материала к надлежащему виду.

Вот типичная ситуация. Заказчик сайта имеет файл, созданный в Word и содержащий иллюстрированную хорошо отформатированную статью объемом более десятка страниц. Эта статья вполне хорошо оформлена с точки зрения вывода на печать и неплохо, если ее просматривать на экране компьютера во вьювере, снабженном средствами навигации. Но заказчик хочет разместить эту статью в одном из разделов своего сайта. Для этого проще всего преобразовать (конвертировать) Word-документ в Web-страницу средствами самого редактора Word. Так обычно и поступают на скорую руку, хотя это далеко не лучшее решение.

Так, дизайн статьи в формате Word может не гармонировать с общим дизайном сайта, например, по цвету, шрифту и композиции разделов. Разумеется, можно перед конвертированием изменить стиль документа надлежащим образом средствами самого Word. Но это не снимает всех проблем.

HTML-документ, полученный автоматическим конвертированием Word-документа, обычно оказывается очень избыточным за счет слишком большого объема форматирующей стилевой информации. Содержательная информация в нем перемешана со служебной чуть ли не в одинаковой пропорции; структуру статьи в таком представлении трудно разглядеть, не говоря уже о редактировании ее содержания. В данном случае следует в текстовом редакторе создать содержательно пустой HTML-документ и перенести в него посредством буфера обмена статью из редактора Word, а затем разметить ее HTML-дескрипторами (тегами), указав заголовки, абзацы и места, требующие особого форматирования (например, выделения). Окончательный вид текста статьи следует задать с помощью параметров CSS (шрифты, цвета и др.) для заголовков различных уровней, основного текста и особо выделенных мест.

Статью объемом в несколько страниц не назовешь большой, если она на бумажном носителе. При публикации в Интернете ее объем лучше измерять в экранных страницах. Экранная страница — это область Web-страницы в полностью развернутом окне браузера, отведенная для показа статьи. Если статья занимает более двух-трех экранных страниц, то ее желательно снабдить средствами навигации. Очень большие статьи размещают не в одном, а в нескольких HTML-файлах. В этом случае система навигации по статье становится необходимой, а не только желательной.

Кроме того, следует проанализировать графические изображения (если они есть) с точки зрения их разрешения, а также соответствующие файлы — с точки зрения графических форматов и объема. Как правило, разрешение графических изображений (а значит, и объем файлов) можно уменьшить в несколько раз, сохранив достаточный уровень качества. Формат файлов изображений должен входить в число широко употребимых в Web-публикациях: JPEG, PNG, GIF, SWF и некоторые другие.

Аналогичным образом дело обстоит с видео и аудио, которые заказчик собирается опубликовать на своем сайте. Часто оказывается, что исходные материалы представле-

ны не в том формате, который нужен. Например, файлы видеороликов, полученные с помощью видеокамеры, обычно требуется конвертировать в формат потокового видео (например, FLV), если требуется демонстрировать их на странице сайта, не дожидаясь окончания загрузки, и не во внешнем проигрывателе, которого может и не быть у пользователя.

1.6. Тестирование и опытная эксплуатация

Отдельные страницы и сайт в целом неоднократно тестируются практически на всех этапах процесса их разработки. Тем не менее требуется выполнить тщательное генеральное тестирование сайта в режиме онлайн прежде, чем он будет передан заказчику. Помимо прочего, следует особо проверить следующее:

- внешний вид страниц сайта на дисплеях с различными разрешением и глубиной цвета. Типичные разрешения по горизонтали в настоящее время — 1024 и 1280 пикселов, но неплохо проверить как при меньших (800), так и при больших (1366, 1680) разрешениях. Следует обратить особое внимание на возможные изменения относительного расположения элементов внутри страниц, а также на положение страниц относительно краев окна браузера.

  Типичная глубина цвета — 24 и 32 бита, но неплохо проверить и при 8 битах (256 цветов). Если вы использовали цвета не из Web-палитры, то на дисплеях с малой цветовой глубиной возможны существенные искажения графики и цветов текстовых элементов;

- правильность функционирования всех ссылок, как внутренних (на страницы своего сайта), так и внешних (на страницы других сайтов). Если обнаруживается, что какая-то страница перехода отсутствует, то следует создать ее содержательно пустой аналог с сообщением вроде "Данная страница находится в разработке". Это не самое лучшее решение. Лучше не помещать ссылку на неготовую страницу совсем или отложить публикацию сайта;

- правильность функционирования всех скриптов и поведение сайта в случае, когда выполнение скриптов отключено в настройках браузера;

- скорость загрузки страниц сайта при небольшой и средней пропускной способности канала связи (1—5 Мбит/с). Хорошо, если время загрузки страниц не превышает нескольких секунд. Для оценки времени загрузки и других параметров можно воспользоваться сайтами, предоставляющими сервис по анализу указанных посетителем опубликованных сайтов, например, **http://www.cy-pr.com/analysis**;

- межбраузерную совместимость (инвариантность) внешнего вида и функционирования сайта. Данную проверку нужно выполнить хотя бы для трех-четырех ведущих браузеров — Mozilla Firefox, Opera, Google Chrome и Microsoft Internet Explorer (для версии 8 и самой свежей). Напомню, что версия 8 Internet Explorer — последняя, поддерживаемая Windows XP/Vista, а версия 9 поддерживается Windows 7/Vista. Возможно, что в различных браузерах некоторые части вашего сайта будут выглядеть и/или функционировать не одинаково. Если вы не можете это устранить, то предусмотрите альтернативные замены элементов для "неадекватных" браузеров или хотя бы предупредите об этом посетителей. Подумайте, а не исключить ли неинвариантные элементы совсем или хотя бы до лучших времен: новые версии брау-

зеров лучше работают, а также точнее и полнее соблюдают стандарты (спецификации языков программирования);

- правильность (валидность) кода Web-страниц. Страницы сайта могут отображаться в браузере согласно вашему замыслу даже при некоторых нарушениях стандартных правил написания их кодов. При других же нарушениях возможны видимые неполадки и проявление межбраузерной несовместимости. Поэтому рекомендуется проверять правильность кода страниц. Это можно сделать с помощью онлайновой службы консорциума W3C. На сайте **http://validator.w3.org** вы можете ввести URL-адрес своей страницы, расположенной на Web-сервере, и получить сведения о ее валидности. Даже если все правильно, еще нет полной гарантии межбраузерной и межплатформенной совместимости вашей страницы. Тем не менее это шаг в направлении к идеалу, даже если он пока достижим не в полной мере.

После успешного, как считает разработчик, тестирования сайт сдается заказчику. Продолжительность приема-сдачи зависит от сложности и объема сайта и может составить от часа до нескольких недель. Как бы тщательно ни проводилось тестирование разработчиком, некоторые ошибки все равно могут остаться незамеченными. Если сайт достаточно сложен и процесс его приема заказчиком может затянуться, то следует договориться о проведении определенного во времени этапа опытной эксплуатации (гарантийного обслуживания), в течение которого разработчиком устраняются обнаруженные ошибки и недочеты. В данном случае этапы окончания разработки и опытной эксплуатации желательно оформить отдельными актами. При этом акт об успешном окончании опытной эксплуатации является одновременно и актом об окончании всех работ по созданию сайта.

Поскольку в период опытной эксплуатации сайт может корректироваться не только вами (автором), но и заказчиком, следует сохранять копии исходной и последующих версий сайта.

Акт о передаче сайта в опытную эксплуатацию (если она предусмотрена договором) очень важен, поскольку оформляет этап завершения его разработки (дизайна). Дело в том, что в период опытной эксплуатации заказчик обычно "созревает": он наконец-то начинает хорошо понимать, что же ему в действительности было нужно, когда он заказывал сайт. Поэтому будьте осторожны: в данный период он может попытаться под видом исправления ваших недочетов и ошибок "дозаказать" вам то, что упустил при заключении договора или обсуждении на ранних этапах проектирования.

Одно дело выполнить некоторые доработки сайта, хорошо вписывающиеся в уже реализованные структурно-функциональную и графическую схемы, а другое — существенно их перестраивать. Остерегайтесь под нажимом заказчика надстроить пару-тройку "железобетонных этажей" над уже готовым "щитовым дачным домиком", чтобы ваша модернизированная конструкция не рухнула под собственной тяжестью. Помните также, что, с одной стороны, у совершенства нет предела, а с другой, что лучшее — враг хорошего. Процесс доработок может приобрести циклический характер и затянуться во времени. Быть может, уже необходима не доработка, а редизайн сайта?

Редизайн сайта — это модификация графической и/или структурно-функциональной составляющих уже существующего сайта. На первый взгляд может показаться, что редизайн проще создания сайта "с нуля", поскольку прототип имеется, а требуется изменить лишь некоторую его часть, не поломав при этом всего остального. К тому же и

заказчик обычно считает, что редизайн обойдется ему существенно дешевле разработки нового сайта.

Тем не менее если не вы автор сайта, то при редизайне вам придется разбираться с чужим кодом, а это, поверьте, очень хлопотно. Многие программисты не без основания считают, что легче написать программу заново, чем скорректировать чужую. Редизайн можно уподобить ремонту квартиры с перестилкой полов без выноса мебели. Таким образом, следует иметь в виду, что по трудоемкости редизайн может превосходить разработку сайта с самого начала. Вам придется тщательно исследовать имеющийся сайт и проанализировать требования к его модернизации прежде, чем решиться на последнюю.

Если же автором сайта являетесь вы, то задача редизайна несколько упрощается. Она будет тем проще, чем лучше в структурном отношении был выполнен исходный проект. Таким образом, еще на этапе разработки сайта стоит предпринять специальные меры, облегчающие возможный его редизайн в будущем.

1.7. Публикация и оптимизация сайта для поисковых систем

1.7.1. Доменное имя сайта

Как уже говорилось, создание сайта предполагает размещение его файлов на Web-сервере еще на ранних этапах его разработки для тестирования в режиме онлайн. Прежде чем это сделать, необходимо сайт, пусть еще и не готовый, зарегистрировать на сайте провайдера, предоставляющего хостинг. На этом сайте можно найти достаточно подробные сведения о том, как это сделать практически.

При регистрации своего сайта вы получаете его доменное имя, а также логин и пароль, чтобы иметь доступ к его файлам по протоколу FTP. Обычно на сайте провайдера хостинга предлагается и сервис: онлайн-редактор кода, FTP-клиент, шаблоны Web-страниц и др.

Доменное имя (домен) определяет уникальное в Интернете имя вашего сайта, например, **www.mysite.ru**. Доменное имя состоит из нескольких слов, разделенных точками, каждое из которых соответствует некоторому уровню иерархии в системе доменных имен (DNS — Domain Name System). Доменное имя читается слева направо, от нижних в иерархии имен к верхним. Самое правое слово представляет домен первого уровня, называемый еще зоной. Примеры доменов первого уровня: ru, uk, de, fr и т. п. — национальные; com — коммерческие; org — некоммерческие; edu — образовательные; net — ресурсы и организации, связанные с сетью; mobi — сайты для мобильных телефонов. Ресурсы, размещенные в зоне ru, образуют сегмент Интернета, называемый еще Рунетом.

Доменное имя второго уровня может быть именем вашего сайта, но его регистрация и получение прав владения — платная услуга. Существует множество фирм, обеспечивающих регистрацию таких доменов. Чтобы их найти, достаточно в поисковой системе подать запрос "регистрация домена".

Хостинговые компании, предоставляющие возможность размещения сайтов на своих серверах, обычно имеют домен второго уровня и предлагают регистрацию, платную

или бесплатную, доменов третьего уровня, которые могут стать именами вашего сайта. В данном случае полное имя вашего сайта будет состоять из трех слов, например, **mysite.narod.ru**, **mysite.far.ru** или **mysite.spb.ru**. При этом владеет доменом третьего уровня хостинговая компания, а вам он сдается в аренду.

Солидные фирмы и организации, особенно коммерческие, создают сайты с доменными именами второго уровня, чтобы поддержать свою репутацию и снять ограничения, свойственные сайтам с именами третьего уровня. Как известно, имена таких сайтов лучше запоминаются и занимают верхние позиции в индексах поисковых систем и каталогов, а потому и более популярны.

В русскоязычной зоне Интернета наиболее популярным провайдером бесплатного хостинга является Яндекс.Народ (**http://narod.yandex.ru**). Перед тем как воспользоваться его услугами, требуется в Яндексе завести почтовый ящик. Выбор имени учетной записи почты станет именем вашего сайта. Так, если **mysite@yandex.ru** — адрес вашего почтового ящика, то доменное имя вашего сайта будет **mysite.narod.ru**. Таким образом, следует хорошо подумать, как назвать свой почтовый ящик.

Имя сайта должно быть уникальным, как можно более коротким и легко запоминаемым. Неплохо также, если оно благозвучно, чтобы услышав, можно было правильно написать его. Желательно, чтобы имя чему-то соответствовало: тематике сайта, названию фирмы, роду деятельности, предлагаемым товарам и услугам, имени владельца персонального сайта и т. п. Например, сайт издательства "БХВ-Петербург" имеет имя **www.bhv.ru**, сайт **htmlbook.ru** посвящен HTML, CSS и верстке Web-страниц, **www.1tv.ru** — официальный сайт первого канала телевидения, а **dunaevv1.narod.ru** — персональный сайт автора данной книги, посвященный Web-разработкам.

Выбор подходящего доменного имени сайта ограничен тем, что многие "хорошие" имена уже заняты другими сайтами. Поэтому приходится как-то изощряться, например, использовать аббревиатуры, дефисы (они не могут быть первым и последним символом в имени) и цифры. Регистр символов в доменном имени не имеет значения.

1.7.2. Регистрация и индексирование в поисковых системах

Итак, ваш сайт имеет адрес (доменное имя) в Интернете, а его файлы находятся на сервере. Теперь он доступен для посетителей. Но как они узнают о его существовании? Для этого необходимо, прежде всего, зарегистрировать сайт в поисковых системах. Вот список наиболее популярных из них:

- Яндекс (**http://webmaster.yandex.ru**);
- Google (**http://www.google.com/addurl/**);
- Rambler (**http://www.rambler.ru/doc/add_site_form.shtml**);
- Апорт (**http://catalog.aport.ru/rus/add/AddUrl.aspx**);
- Yahoo! (**http://siteexplorer.search.yahoo.com/submit**);
- Bing (**http://www.bing.com/webmaster/SubmitSitePage.aspx**);
- Поиск@mail.ru (**http://go.mail.ru/addurl**);
- Нигма.рф (**http://www.nigma.ru**).

На Web-страницах поисковых систем, предназначенных для регистрации вашего сайта, будет предложено ввести его URL и, возможно, другие данные: имя владельца, адрес электронной почты, регион, рубрику, к которой относится сайт, его краткое описание, ключевые слова и т. п. Поисковая система проанализирует ваш сайт и через некоторое время (от нескольких дней до двух-трех недель) он может появиться в так называемом индексе. Индексирование можно ускорить, если разместить ссылку на ваш сайт на каком-нибудь другом сайте или популярном форуме, уже известных поисковой системе.

Индекс — это упорядоченный список ресурсов (в том числе и Web-страниц) сети, которые известны в текущий момент времени поисковой системе. Обрабатывая запрос, поисковая система ищет адреса соответствующих (релевантных) ресурсов именно в этом списке. Ответ на запрос представляется списком релевантных ресурсов, который может занимать не одну экранную страницу. Пользователи при поиске обычно просматривают лишь одну-две таких страницы. Поэтому важно, чтобы ваш сайт оказывался возможно ближе к началу индекса.

Положение сайта в индексе (списке), а следовательно, и количество его посетителей (популярность) зависит от многих факторов, таких как уровень доменного имени, популярность рубрики, степень уникальности и информативности содержимого, количество ссылок на ваш сайт из других и т. д. Поисковые системы автоматически вычисляют по некоторому алгоритму числовой показатель авторитетности (важности, значимости, информативности) сайта и по нему ранжируют сайты в индексе. Так, поисковая система Яндекс вычисляет тематический индекс цитирования (тИЦ, CY), а Google использует множество показателей, из которых существенную роль играет PageRank (PR). Интересно, что слово PageRank можно перевести с английского как Ранг Страницы, хотя слово Page (страница) Google связывает с именем автора алгоритма ранжирования — Larry Page. Значение PR, также как и тИЦ, определяется количеством и качеством ссылок на сайт.

Алгоритмы ранжирования у различных поисковых систем могут быть разными, но суть у них одна — ранжирование с учетом индекса цитируемости, вычисляемого по количеству и *качеству* ссылок на ваш сайт с других сайтов. Качество ссылки прямо зависит от авторитетности ссылающегося сайта, которая сама определяется его индексом цитирования. Больший вес имеют односторонние, а не взаимные ссылки. Иначе говоря, лучше, чтобы кто-то ссылался на вас, а вы на него — нет. Предпочтительнее также, чтобы ссылающийся сайт принадлежал той же рубрике (теме), что и ваш. Внутренние ссылки (т. е. ссылки с одних страниц на другую в рамках вашего сайта) не увеличат индекс цитирования последней.

Поисковые системы анализируют также и содержимое Web-страниц: краткое описание, ключевые слова и основной текстовый контент. Ключевые слова и слова краткого описания должны встречаться в основном тексте страницы и лучше — неоднократно, а еще предпочтительнее — в заголовках разделов, причем определяемых специальными дескрипторами HTML, а не представленных графическими изображениями. С другой стороны, ключевые слова вашей страницы должны как можно реже встречаться в языке вообще. Иначе говоря, ключевые слова должны быть редкими вообще и часто встречаться в тексте Web-страницы. Такое соотношение создает у поисковика впечатление, что ваш сайт в чем-то уникален, а значит, информативен в некоторой области, поэтому следует увеличить его вес при определении ранга. Вместе с тем, ключевые слова, пусть не все, должны быть из тех, которые часто встречаются в поисковых запросах. Выдер-

жать все эти требования довольно трудно. Следует найти некий компромисс. Если сайт посвящен некоторой очень популярной теме, то список всех таких сайтов огромен и ваш сайт-новичок займет в нем одно из последних мест. Если же в нем представлены специфические, особенные разделы на популярную тему, то это способствует продвижению сайта вверх по списку. Серьезных пользователей Интернета больше интересуют специфические сведения, а не банальные общие места.

Таким образом, при формировании текста страниц следует учитывать и некоторые формальные правила, помогающие продвинуть ваш сайт к началу индекса. Вместе с тем, главное, что существенно способствует продвижению сайта, это его содержание — интересное, полезное и информативное для некоторой целевой аудитории. Хорошо также, если оно грамотно, красиво и удобно представлено.

Время от времени специальные программы поисковых систем, называемые роботами, посещают проиндексированный сайт, чтобы в очередной раз проанализировать его содержимое и скорректировать индекс. В корневой папке сайта при желании можно разместить текстовые файлы sitemaps.xml и/или robots.txt с информацией для поисковых роботов. Так, файл sitemaps.xml содержит в XML-формате сведения об URL-адресах страниц сайта, которые следует проиндексировать, о времени и частоте их обновления. А файл robots.txt, наоборот, содержит инструкции по ограничению доступа поисковыми роботами к ресурсам сайта. Эту же информацию можно сохранить не в файле, а в тегах `<meta name="robots" ...>` в HTML-кодах Web-страниц. Синтаксис инструкций довольно прост, а его описание легко найти в Интернете. Применение данных файлов не обязательно. С другой стороны, указание URL в файле sitemaps еще не гарантирует, что соответствующий ресурс будет проиндексирован поисковой системой.

После опубликования и регистрации своего сайта в поисковых системах, сразу или немного погодя, желательно поместить информацию о нем в каталогах. Каталог — информационно справочный сайт, содержащий описания сайтов (ресурсов сети) и ссылки на них. Каталоги могут содержать несколько тематических разделов (подкаталогов) или быть целиком посвящены какой-либо узкой теме. Ссылки на ваш сайт из каталога имеют повышенный вес при вычислении индекса цитирования. Поэтому и стремятся занести сайт в каталог.

Свои каталоги имеют и поисковые системы. Вы можете попытаться внести свой сайт в авторитетные каталоги Яндекс (**http://yaca.yandex.ru**), Rambler Top 100 (**http://top100.rambler.ru**), Апорт (**http://catalog.aport.ru/rus**), Google (**http://www.google.com/dirhp**) и др. Для этого следует подать заявление — ввести на странице каталога и отправить сведения о себе и своем сайте. Заявка может рассматриваться редакторами каталога в течение нескольких месяцев. За некоторую плату этот процесс можно несколько ускорить. О результате рассмотрения заявки, как положительном, так и отрицательном, вы не будете извещены каким-либо специальным образом. Об успехе можно судить только по тому, что ваш сайт появился в каталоге. В противном случае объяснений редакторы каталога не дают, а претензии не принимают.

В настоящее время известны методы недобросовестного увеличения индекса цитирования, раскрутки сайтов. Есть в Интернете и предложения такого рода услуг. Однако алгоритмы индексирования постоянно совершенствуются и предусматривают, так или иначе, возможность плутовства. В результате ранг сайта может быть понижен вплоть до исключения из индекса.

Для контроля посещаемости сайта на его страницах устанавливают счетчики (информеры). Они могут быть как видимыми всеми посетителями, так и невидимыми. Доступ к более подробной информации о посетителях обычно защищается паролем. Для получения более объективной информации можно установить сразу несколько счетчиков от различных систем. Например, вы можете поставить счетчик от Яндекс.Метрика (**http://metrika.yandex.ru**), Rambler Top100 или создать его самостоятельно с помощью серверного сценария. Счетчики-информеры предоставляют доступ к статистике, развернутой во времени. Так, вы можете узнать количество визитов, уникальных посетителей, их географическое положение, какие поисковые запросы, браузеры, операционные системы и мониторы они используют (рис. 1.3).

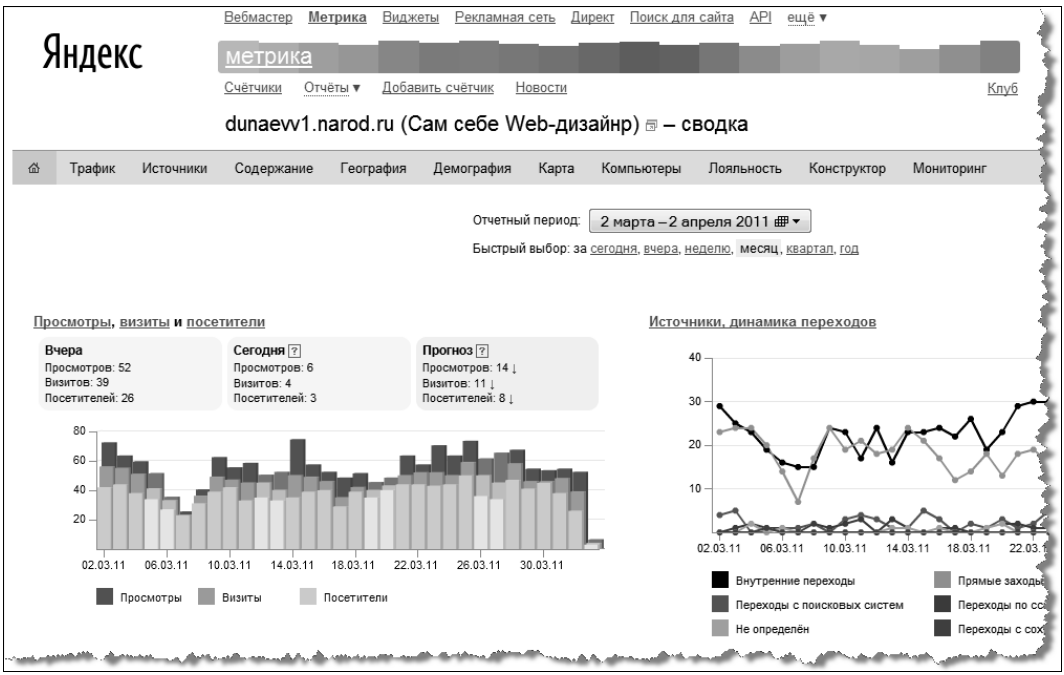


Рис. 1.3. Одна из страниц сервиса Яндекс.Метрика

Для получения сведений о сайте с точки зрения поисковых систем (тИЦ, PR, количество внешних ссылок и др.) можно воспользоваться соответствующими сервисами, предоставляемыми на специальных сайтах, например, Яндекс.Вебмастер (**http://webmaster.yandex.ru**), CY-PR.com (**http://www.cy-pr.com/analyisis**) и др.

ГЛАВА 2

Как устроен HTML-документ

Как известно, сайт (Web-узел) состоит из одной или нескольких страниц, каждая из которых представлена своим кодом, написанным на языке HTML или XHTML (расширяемый HTML). Такой код мы будем еще называть HTML-документом. Он имеет определенную структуру и сохраняется в обычном текстовом файле с расширением html или htm.

Посредством HTML производится разметка страницы, вставляются ссылки на другие страницы и ресурсы, встраиваются графические изображения и мультимедиа и т. д. HTML-документ является несущей конструкцией Web-страницы.

В данной главе рассматриваются базовые сведения о языке HTML и структуре HTML-документа. Если что-то окажется непонятным, не отчаивайтесь: туман мало-помалу рассеется в следующих главах.

2.1. Что такое HTML

При подготовке документа к публикации в Интернете его требуется отформатировать или, иначе говоря, разметить — с помощью специальных символов явно указать, что и каким образом следует представить пользователю (читателю) этого документа. Набор специальных символов (меток, дескрипторов, команд или тегов), а также правила их употребления составляют то, что называют языком разметки (Markup Language). В настоящее время существуют несколько таких языков, среди которых наиболее популярны HTML (HyperText Markup Language — язык разметки гипертекста), XHTML (eXtensible HyperText Markup Language — расширяемый язык разметки гипертекста) и XML (eXtensible Markup Language — расширяемый язык разметки). HTML и XHTML применяются для разметки Web-документов, а XML — как средство структуризации информации для обмена между компьютерными программами. Между этими языками очень много общего, хотя имеются и различия, о которых будет рассказано позже.

Разметка документа, как уже упоминалось, осуществляется посредством специальных дескрипторов, называемых тегами (tag — метка, признак, ярлык). В языках разметки в большинстве случаев они имеют вид: <*имя_тега*> и </*имя_тега*>. Например, `<p>` и `</p>`, где `p` — имя тега. Эти две формы одного и того же тега играют роль подобно открывающей и закрывающей скобкам при написании математических формул. Между ними можно разместить некоторое содержимое — обычный текст и/или тег, содержа-

щий, возможно, другие теги. Тег, имеющий открывающую и замыкающую части, называют еще контейнерным. Вот его синтаксис:

<имя_тега> содержимое контейнерного тега </имя_тега>

Существуют и неконтейнерные теги (без замыкающей части), например, <img> — для вставки графического изображения и
 — для указания перехода на новую строку. Неконтейнерные теги называют еще самозакрывающимися или пустыми.

> ***Внимание!***
>
> Имя тега должно непосредственно (без пробелов) следовать за открывающей уголковой скобкой. С другой стороны, закрывающая уголковая скобка может находиться сколь угодно далеко.

Теги не отображаются в окне браузера, а лишь указывают ему, какие элементы входят в документ и как они между собой соотносятся. Так, текстовый документ состоит из абзацев, которые могут объединяться в разделы. Разделы, в свою очередь, могут входить в более крупные структурные единицы, например главы или части и т. д. Язык HTML обеспечивает поддержку иерархической декомпозиции документа довольно простыми средствами. Например, чтобы указать, что данный фрагмент текста относится к одному и тому же абзацу, достаточно просто перед этим фрагментом написать тег , а после него — . Другими словами, текст абзаца следует заключить в контейнер , например,

```
<p>Здравствуйте, дорогой Иван Иванович.</p>.
```

Чтобы несколько элементов документа (например, абзацев) объединить в один раздел, их следует заключить в какой-нибудь подходящий для этой цели контейнер, например (листинг 2.1).

**Листинг 2.1. Пример использования контейнера **

```
<div>
       <p>
           Здравствуйте, дорогой Иван Иванович.
       </p>
       <p>
           В первых строках своего письма передаю поклон Марье Ивановне.
       </p>
</div>
```

Здесь контейнер содержит два контейнера , каждый из которых заключает в себя некий текст.

Самый крупный контейнер для содержательной части документа — тег <body> (тело документа), он содержит другие контейнеры. Например, формируя письмо, мы могли бы написать фрагмент кода на языке разметки, приведенный в листинге 2.2.

Листинг 2.2. Фрагмент кода на языке разметки

```
<body>
   <div>
```

```
        <p>
           Здравствуйте, дорогой Иван Иванович.
        </p>
        <p>
           В первых строках своего письма передаю поклон Марье Ивановне.
        </p>
    </div>
</body>
```

Итак, мы уже упомянули несколько тегов: <p>, <br>, <body>. В HTML имеется много других тегов для группировки элементов документа в логические блоки, образующие в совокупности структуру документа.

На рис. 2.1 в окне обычного текстового редактора показан пример текста, размеченного тегами абзаца <p> и перевода строки <br>, а также его вид в окне браузера. Хотя данный документ в целом оформлен не по всем правилам и даже совсем не по правилам HTML, основные Web-браузеры все-таки отображают его, причем одинаково. Все современные браузеры при открытии в них файлов с расширениями htm или html пытаются интерпретировать последние как HTML-коды, т. е. как поток тегов языка разметки. Однако это еще не означает, что формировать HTML-документ можно "как попало". Правила создания HTML-документов существуют, и соблюдать их настоятельно рекомендуется во избежание несовместимости с различными браузерами и других проблем.

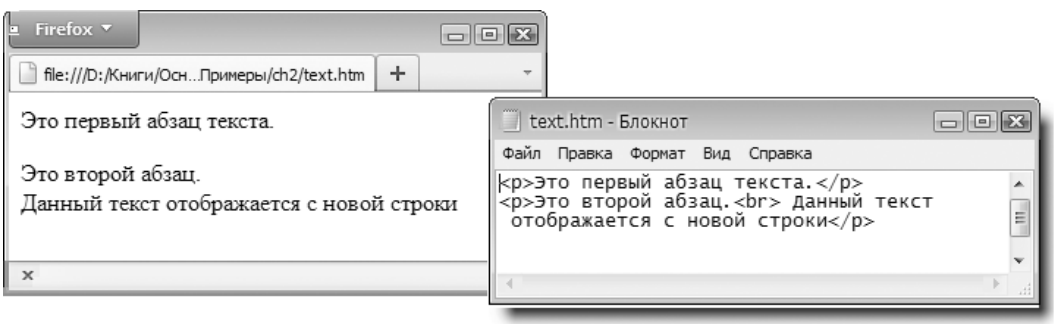


Рис. 2.1. Текст, размеченный тегами абзаца <p> и перевода строки <br>, в текстовом редакторе и в браузере

Обратите внимание, что в рассматриваемом примере расположение текста в окнах редактора и браузера различается. В текстовом редакторе символы могут располагаться в одной или нескольких строках, что еще не определяет однозначно их расстановку в окне браузера. При наборе символов в текстовом редакторе нажатие на клавиши <Enter>, <Tab> и т. п. приводит к форматированию текста, визуально контролируемому с помощью дисплея. Текстовый редактор отвечает на нажатие указанных клавиш расстановкой во вводимом тексте специальных управляющих символов, которые не видны в общем потоке печатаемого текста. Для управления расположением текста в окне браузера требуется явное задание управляющих символов в виде специальных тегов языка разметки. В рассматриваемом примере тег абзаца <p> указывает браузеру отобразить заключенный в него текст с новой строки и с отступом в одну строку. В обычном текстовом редакторе этого же эффекта можно добиться, дважды нажав на клавишу

<Enter> в конце введенного текста. Далее, переход на новую строку внутри второго абзаца задан с помощью тега <br>. Без него этот переход либо не произошел бы, либо мог произойти лишь при достаточно малой ширине окна браузера. Результат зависит от используемого браузера. Дело в том, что браузеры пытаются так или иначе перенести поток символов на новую строку, если ширина окна не позволяет расположить текст вдоль одной горизонтальной линии.

Чтобы отображение текста и других элементов документа было инвариантно относительно различных браузеров, необходимо принудительно указывать переносы, отступы и другие управляющие символы с помощью тегов языка разметки и специальных средств (атрибутов тегов и параметров CSS, о чем будет рассказано далее).

О правилах оформления HTML-документов мы поговорим позже, а сейчас остановимся на одной важной особенности HTML. Не все, но многие теги определяют не только структуру документа, но и внешний вид его элементов в окне браузера, т. е. задают еще и способ отображения соответствующей информации в браузере. Например, тег абзаца <p> определяет, что все следующее за ним будет отображено с новой строки и с пропуском одной строки; текст, заключенный между тегами <h1> и </h1>, будет выведен с новой строки максимальным по размеру шрифтом, а последующая информация будет размещена с новой строки (иначе говоря, контейнер <h1> задает заголовок первого уровня); текст в контейнере <i> выводится курсивом.

Таким образом, разметка тегами HTML в большинстве случаев предопределяет не только структурную декомпозицию документа, но и его внешний вид в окне браузера. Разумеется, структурная декомпозиция в конечном счете должна быть как-то выражена внешним образом. Например, листая книгу, мы видим ее структуру в виде иерархии глав, разделов, подразделов и абзацев. Вместе с тем, нетрудно понять различие между собственно структурной декомпозицией и внешним представлением ее элементов. В самом деле, главы, разделы и подразделы можно отобразить как линейно следующие друг за другом, так и расположенные на страницах в двух и более колонках; заголовкам можно придать тот или иной вид, манипулируя такими параметрами шрифта, как цвет, размер и начертание. При этом логическая структура документа остается постоянной, а изменяется только ее представление на бумаге или в окне браузера. Однако идея четко разделить структурный и представительский аспекты языка разметки не реализована в полной мере в HTML. В HTML средства структурной декомпозиции документа и его внешнего представления оказались изначально сильно связанными между собой. Именно это обстоятельство и сыграло, на мой взгляд, решающую роль в чрезвычайно широкой популярности данного языка разметки: разработчики Web-страниц хотели сразу видеть результат разметки документа в браузере и получили это (см. рис. 1.1).

Для определения элементов, вставляемых в документ, одних только тегов бывает недостаточно. Так, для вставки графического изображения служит тег <img>. Однако требуется еще указать источник изображения, т. е. адрес или, точнее, URL (Uniform Resource Locator — унифицированный указатель ресурса) файла. В подобных случаях используются параметры тегов, называемые еще атрибутами. Один или несколько атрибутов записывают в открывающей части тега в произвольном порядке в следующем виде: *имя_атрибута="значение"*. Атрибуты отделяют друг от друга пробелами (их количество может быть любым). Вот пример тега для вставки графического изображения:

```
<img src="images/mypicture.jpg">
```

Тег <img> не имеет заключительной части вида </img>.

Чтобы указать размеры графического изображения на странице (в пикселах), можно записать следующее выражение:

```
<img src="images/mypicture.jpg" width="200" height="100">
```

Здесь `src`, `width` и `height` — имена атрибутов тега `<img>`, за которыми следует знак равенства и, далее, значения, указанные в кавычках, двойных или одинарных.

Атрибуты позволяют указать, например, цвет фона, размеры, начертание и цвет шрифта, характеристики выравнивания элементов документа и т. п. Так, в контейнерном теге `<body>`, задающем основную часть документа, с помощью атрибутов `text` и `bgcolor` можно определить цвет текста и фона:

```
<body text="#ff0000" bgcolor="#00ffee">.
```

Таким образом, развитие HTML пошло в сторону добавления средств, определяющих внешнее представление документов. Стали появляться как новые теги, так и новые их атрибуты. На первых порах производители браузеров (Microsoft, Netscape, Sun Microsystems и др.) создавали свои варианты HTML, отличающиеся не только наборами тегов, но и способами их интерпретации. В сложившихся условиях один и тот же документ мог выглядеть по-разному в зависимости от браузера, что явно противоречило цели всемирного распространения информации. Очевидно, стандарты стали насущной необходимостью.

Разработкой стандарта для HTML занялась международная организация World Wide Web Consortium (W3C, Консорциум Всемирной паутины), в состав которой вошли крупнейшие производители программного обеспечения для работы в Интернете (в том числе Microsoft, Sun Microsystems, Netscape и др.). Сведения о стандартах (спецификациях) можно найти на официальном сайте W3C по адресу **www.w3.org**. Браузеры должны интерпретировать язык разметки соответственно стандарту, хотя на практике это происходит не в полной мере. Одно дело идея, другое — ее реализация.

В декабре 1997 г. консорциум W3C выпустил официальную спецификацию HTML 4.0 — рекомендацию для производителей Web-браузеров. С декабря 1999 г. по настоящее время существует версия 4.01. Тем не менее различия в интерпретации основными браузерами некоторых тегов и атрибутов официальной спецификации HTML все же остались. Для обеспечения дополнительных возможностей визуального представления, программирования поведения, динамичности и интерактивности документов появились каскадные таблицы стилей (CSS — Cascading Style Sheets) и скрипты (сценарии), написанные на специальных языках, наиболее популярный из которых — JavaScript.

Параллельно с HTML развивался другой язык разметки — XML (eXtensible Markup Language — расширяемый язык разметки). Он был создан как средство структуризации информации для обмена между программами. Web-браузер — одна из них, но далеко не единственная. При этом на способ внешнего представления структурированных данных никаких ограничений не накладывалось. Главная задача, которая ставилась при разработке этого языка, состояла в обеспечении совместимости между различными системами обработки структурированных данных. Передающей стороне не важно, как будет отображен XML-документ получателем, требуется лишь, чтобы он однозначно разобрался в структуре принятого документа.

В XML, так же как и в HTML, есть теги и атрибуты. Однако, в отличие от HTML, в языке XML они не предопределены изначально, а могут создаваться автором доку-

мента по своему усмотрению. Теги XML задают лишь структуру документа, но не его представление. Внешний вид элементов XML-документа определяется отнюдь не тегами, а специальными программами-анализаторами, часто называемыми парсерами (parser). Основные Web-браузеры имеют такие встроенные парсеры, но чтобы подключить их к работе и заставить отобразить собственно информацию (без тегов), необходимо указать, что содержимое документа является XML-кодом и, кроме того, достаточно сослаться на каскадную таблицу стилей (CSS).

В XML воплотилась хорошая идея разделения между структурным и внешним представлением содержания документа. В HTML 4 это не так, но его можно немного модернизировать, чтобы существенно приблизить к XML. На пути к достижению этой цели появился XHTML (eXtensible HyperText Markup Language — расширяемый язык разметки гипертекста).

> ***Примечание***
>
> XML — подмножество более сложного языка SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки). На основе XML созданы более специализированные языки разметки, например XHTML, SVG, RSS, MathML и др. Такие языки еще называют словарями данных языка XML. HTML не является ни словарем XML, ни даже подмножеством SGML.

Еще до появления XHTML многие разработчики Web-сайтов практиковали следующий технологический прием. С помощью HTML-тегов создавалась лишь структура документа, при этом теги и атрибуты, предназначенные только для задания внешнего вида элементов, практически не использовались (насколько это было возможно), а отображение элементов документа в окне браузера или при печати определялось только в каскадных таблицах стилей. Данный прием можно назвать "применением HTML в стиле XML".

Разметку Web-страницы можно реализовать на основе языка как HTML, так и XHTML. Однако XHTML более строг, ближе к универсальному языку XML и должен поддерживаться одинаковым образом всеми современными браузерами. В данной книге все примеры написаны на языке XHTML. Однако во многих случаях я применял следующий шаблон: в приводимых примерах задан тип документа, указывающий на стандарт HTML 4.01, а собственно код написан в соответствии со стандартом XHTML. Хотя дескриптор `<!DOCTYPE ...>` типа документа указывает на его соответствие стандарту HTML 4.01, а не XHTML, ничего крамольного в этом нет. Объявив в дескрипторе `<!DOCTYPE ...>` свою приверженность HTML 4.01, вы можете писать свои коды разметки в соответствии с официальной спецификацией как HTML 4.01, так и XHTML.

Читатели, знакомые только с HTML 4, увидят не слишком много отличий этого языка от XHTML. Перейти от HTML-кодов к XHTML-кодам довольно легко. Языки HTML и XHTML очень схожи, но различия все же есть. Вот основные (но далеко не все) из них:

- Язык HTML регистронезависимый: не имеет значения, строчными или прописными буквами записаны имена тегов и атрибутов. В XHTML для записи имен тегов и атрибутов рекомендуется указывать только строчные буквы. Например, вместо

  ```
  <IMG SRC="mypicture.jpg">
  ```

  следует писать

  ```
  <img src="mypicture.jpg"/>.
  ```

- В HTML большинство тегов контейнерные (например, `<h1>...</h1>`), т. е. имеют открывающий и закрывающий дескрипторы. Вместе с тем, есть и неконтейнерные теги: для дескриптора `<тег>` нет закрывающего дескриптора `</тег>`. Например, для вставки графического изображения в HTML применяется тег `<img src="mypicture.jpg">`, а в XHTML — `<img src="mypicture.jpg"/>` Неконтейнерные теги еще называют пустыми. В XHTML для таких тегов необходимо указывать слэш (косую черту) перед закрывающей уголковой скобкой: `<тег ... />`, например,

  `<img src="mypicture.jpg"/>` или `<br/>`.

- В некоторых тегах HTML имеются атрибуты без значений (так называемые булевы или логические атрибуты), например, `<option selected>`. В XHTML для подобных атрибутов следует явно указывать строковое значение, совпадающее с именем соответствующего атрибута, например,

  `<option selected="selected" />`.

- В XHTML значения атрибутов тегов нужно заключать в кавычки. В HTML это делать не обязательно. Например, в XHTML следует писать `<img src="mypicture.jpg"/>`, а не `<img src=mypicture.jpg>`, как допустимо (хотя тоже не рекомендуется) в HTML. Некоторые атрибуты принимают числовые значения, которые в HTML допустимо (хотя и не рекомендуется) указывать без кавычек, но в XHTML их следует заключать в кавычки, двойные или одинарные, например,

  `<img src="mypicture.jpg" width="100" height="150"/>`.

- В содержательной текстовой части XHTML-документа нельзя непосредственно использовать символы, которые применяются в коде, такие как `<`, `>` и `&`, причем их нельзя записывать даже в значении URL-адреса. Эти специальные символы следует заменять соответствующими буквенными или числовыми эквивалентами `<`, `>` и `&amp` (или `<`, `&#62` и `&`) соответственно.

- Кодировка файла по умолчанию для HTML-документа — ISO 8859-1, а XHTML-документа — UTF-8.

Ошибки в HTML-коде браузеры пытаются преодолеть тем или иным образом, так что даже весьма небрежно написанный код все же как-то отображается (во многих случаях вполне удовлетворительно). Более других терпим к ошибкам в HTML-коде и лоялен к разработчику браузер Microsoft Internet Explorer. По рекомендации W3C в случае ошибки в XHTML-коде браузеры должны сообщать об этом и прекращать дальнейшую обработку. Таким образом, XHTML-код подвергается более тщательному анализу прежде, чем начнется его интерпретация — отображение в окне браузера. HTML-код, напротив, не анализируется предварительно на правильность, а отображается в порядке следования тегов. При этом сообщения об ошибках не выводятся, а отображение продолжается, даже если ошибки все же возникли. Очевидно, HTML более дружествен разработчику, чем XHTML. Но более других эту "дружественность" ценят малоопытные и небрежные разработчики своих домашних страниц, а не авторы серьезных Web-проектов. Для последних однозначность интерпретации и дисциплина важнее "угодливости" браузеров, вредность которой проявляется при поиске трудноуловимых смысловых ошибок и при попытках обеспечить межбраузерную инвариантность представления документов.

В XHTML сохранилось большинство тегов HTML 4. Однако некоторые старые HTML-теги, введенные в свое время для обеспечения дополнительных возможностей отображения, теперь применять не рекомендуется.

В настоящее время существуют официальные версии XHTML 1.0 и 1.1. и черновая спецификация XHTML 2.0. Однако работа над последней остановлена в пользу создания нового стандарта — HTML 5.

Начиная с 2004 г. совместными усилиями рабочих групп W3C HTML WG и WHATWG с привлечением таких компаний, как Apple, Mozilla, Opera, Google, Microsoft и др., ведется работа по созданию спецификации HTML 5. К середине 2011 г. существовал лишь черновой ее вариант (**www.w3.org/TR/html5**, **www.whatwg.org/specs/web-apps**). Когда появится официальная спецификация от W3C, точно не известно. Полная поддержка HTML 5 браузерами осуществится, видимо, еще позднее, но некоторые новые элементы Firefox, Opera, Safari и Chrome, Internet Explorer 9 воспринимают уже сейчас. Однако браузер Internet Explorer версии 9 уже не совместим с Windows XP и более ранними версиями.

HTML 5 призван стать преемником как HTML 4, так и XHTML 1.0, обеспечивая совместимость с ними, по крайней мере, в самом важном. В HTML 5 можно использовать синтаксис как обычного HTML, так и XHTML. Вместе с тем, ряд тегов и атрибутов, не рекомендованных к использованию в прежних версиях, в новом стандарте отсутствуют. Например, теги `<applet>` для вставки Java-апплетов, `<center>` для центрирования содержимого по горизонтали и `<frameset>`, `<frame>` для разбиения окна браузера на фреймы в HTML 5 не поддерживаются. С целью обеспечения обратной совместимости в спецификации HTML 5 описывается, как должен поступить браузер в случае использования непредусмотренных тегов. Уже существующие сайты не должны пострадать из-за появления нового языка разметки.

В HTML 5 вводится в оборот ряд новых тегов с отчетливо выраженной семантикой, предназначенных для повышения эффективности разработки Web-приложений, а также для помощи поисковым системам. Так, для создания логической структуры и упрощения типовой верстки документа предлагаются теги `<header>` (заголовок), `<nav>` (навигационная панель), `<article>` (статья), `<section>` (раздел, секция статьи), `<aside>` (боковая колонка), `<footer>` ("подвал"), `<menu>` (меню) и др. В прежних версиях данные элементы обычно создаются тегами `<div>` и сразу понять смысл их применения довольно трудно. Иначе говоря, семантика для `<div>` в спецификации неоднозначна, но зато он универсален.

Элементы `<input>` полей ввода данных в HTML 5 приобрели новые атрибуты, заметно облегчающие создание пользовательского интерфейса: `time`, `email`, `url`, `required` и др. Эти атрибуты во многих случаях позволят обойтись без скриптов, проверяющих правильность введенных данных.

Особого внимания заслуживает элемент `<canvas>`, предоставляющий область на странице, в которой с помощью скриптов на языке JavaScript можно рисовать изображения, подобные тем, которые создаются посредством SVG или Flash. Для вставки мультимедиа в HTML 5 предусмотрены теги `<audio>` и `<video>`, обеспечивающие внедрение на страницу соответствующего ресурса и возможность управления его воспроизведением. Браузеры, поддерживающие эти теги, воспроизводят видео и аудио без соответствующих плагинов.

HTML 5 предусматривает и другие полезные и удобные возможности. В данной книге будут описаны лишь некоторые из них, которые уже поддерживаются ведущими браузерами.

> ***ВНИМАНИЕ!***
>
> В данной книге описываются возможности HTML 4 и XHTML, а также применение некоторых элементов HTML 5. Коды примеров написаны в соответствии с синтаксисом XHTML, хотя дескриптор типа документа `<!DOCTYPE...>` в некоторых из них указывает на стандарт HTML 4.01. Такая эклектика вполне допустима. Разумеется, вы можете явно указать в этом дескрипторе на стандарт XHTML, что я и рекомендую. В любом варианте коды примеров должны отображаться одинаково в ведущих браузерах, таких как Internet Explorer, Firefox, Opera, Safari и Chrome. Напомню, что синтаксисы XHTML и HTML 5 совместимы.
>
> В дальнейшем, когда какие-то сведения справедливы как для HTML, так и для XHTML, мы будем писать (X)HTML.

2.2. Определение типа документа

Для разметки документа могут использоваться различные языки, такие как HTML, XHTML, XML и др. Для правильной интерпретации языка, на котором написан документ, браузеру следует сообщить, какой именно язык выбран. Это делается посредством специального дескриптора (тега).

Собственно (X)HTML-код Web-страницы заключается в контейнерном теге `<html>` (т. е. между дескрипторами `<html>` и `</html>`). Однако при создании настоящего (X)HTML-документа рекомендуется с помощью `<!DOCTYPE ...>` указать его тип (DTD — Document Type Definition). Данный дескриптор располагается перед тегом `<html>`.

Тип документа HTML 5 задается специальным дескриптором:

```
<!DOCTYPE html>
```

Тип документа HTML 4 и XHTML задается дескриптором `<!DOCTYPE...>`, содержащим атрибуты в соответствии со следующей схемой:

```
<!DOCTYPE элемент_верхнего_уровня публичность
"регистрация//организация//тип имя//язык" "url">.
```

Рассмотрим атрибуты дескриптора `<!DOCTYPE...>`:

- *элемент верхнего уровня* — для HTML и XHTML элемент верхнего уровня — `html`;
- *публичность* — возможны значения `PUBLIC` (для публичных документов) и `SYSTEM` (для системных ресурсов);
- *регистрация* — возможны значения + (разработчик DTD зарегистрирован в международной организации по стандартизации ISO) и – (разработчик не зарегистрирован в ISO); для консорциума W3C указывается –;
- *организация* — уникальное название организации, разработавшей данный DTD; для (X)HTML-документов такой организацией является W3C;
- *тип* — тип описываемого документа; для (X)HTML-документов указывается DTD;
- *имя* — уникальное имя документа с описанием DTD, например, `XHTML 1.0 strict` (для XHTML версии 1.0 строгой схемы) или `HTML 4.01 transitional` (для HTML версии 4.01 переходной схемы);

- *язык* — язык, на котором описан объект; значение состоит из двух букв в верхнем регистре; для (X)HTML-документов указывается английский язык (EN);
- *url* — URL-адрес файла с описанием DTD, например, `http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd`.

Для XHTML наиболее употребительны следующие варианты (схемы) задания содержимого данного дескриптора:

- **Строгая (Strict) схема** — исключает все нерекомендованные консорциумом W3C теги и атрибуты, предназначавшиеся ранее для внешнего оформления, такие как `<font>`, `<center>`, `align`, `bgcolor` и др., а также применение фреймов. Вместо них следует применять CSS.

  Пример:

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  ```

  Многие разработчики сайтов, приверженцы стандартов, рекомендуют придерживаться строгой схемы. Если вам более всего важна межбраузерная и межплатформенная совместимость (сейчас или в будущем), следуйте этой схеме.

- **Переходная (Transitional) схема** — включает то же, что и схема Strict, а также все нерекомендованные (устаревшие) теги и атрибуты для обеспечения обратной совместимости. Однако данная схема исключает применение фреймов, т. е. тегов `<frameset>` и `<frame>`.

  Пример:

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  ```

  Данная схема целесообразна при переводе HTML-кодов старых документов на язык XHTML, а также для сохранения совместимости с более старыми версиями как браузеров, так и HTML. Эта схема подойдет также тем, кто создает свои Web-документы, не разбираясь в стандартах и версиях браузеров и (X)HTML. Однако данная схема не предполагает наличие фреймов. Если они вам нужны, то выбирайте фреймовую схему.

- **Фреймовая (Frameset) схема** — включает то же, что и схема Transitional, разрешая при этом работу с фреймами.

  Пример:

  ```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 frameset//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
  ```

  Если вы применяете фреймы для компоновки Web-страниц и желаете обеспечить межбраузерную совместимость, то данная схема вам подойдет. Однако следует иметь в виду, что HTML 5 не поддерживает фреймы.

> ***Примечание***
>
> Кроме рассмотренных вариантов XHTML существуют и другие: XHTML Basic (Основной) для миниатюрных устройств, XHTML Mobile Profile (Мобильный профиль) — добавляет к Basic элементы, специфические для мобильных телефонов, XHTML 1.1 Module based (Модульный) — с возможностями расширения.

Итак, мы рассмотрели три основные схемы документов на языке XHTML — три определения типа документа (DTD). В документе HTML 4 также рекомендуется указывать DTD одной из схем, таких же, что и для XHTML.

Вот примеры для HTML 4.01:

- **Строгая (Strict) схема**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 strict//EN"
"http://www.w3.org/TR/HTML4.01/strict.dtd">
```

- **Переходная (Transitional) схема**:

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
```

- **Фреймовая (Frameset) схема**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 frameset//EN"
"http://www.w3.org/TR/HTML4.01/frameset.dtd">
```

В настоящее время браузеры воспринимают HTML-код и без дескриптора `<!DOCTYPE...>`, но в этом случае не исключены непредвиденные ситуации (особенно в Internet Explorer), которых серьезный разработчик обязан избегать.

Объявив с помощью дескриптора `<!DOCTYPE...>`, что документ написан на языке XHTML или HTML с той или иной схемой, следует далее придерживаться синтаксиса и соответствующих ограничений.

> ***Внимание!***
>
> Необходимо иметь в виду, что ошибка в дескрипторе `<!DOCTYPE...>`, и в частности пропуск URL-адреса файла с DTD, может привести к тому, что браузер (прежде всего это касается Internet Explorer) просто проигнорирует его и будет интерпретировать последующий код как HTML 4.0 в режиме обратной совместимости с ранними версиями. В Internet Explorer 8.0 имеется режим предоставления совместимости, в котором можно посмотреть, как будет выглядеть документ в более ранних версиях данного браузера.
>
> В этой книге во многих листингах используется определение типа документа HTML 4.01 переходной схемы. Вместе с тем, при написании кодов почти везде применяется синтаксис XHTML, который правильно интерпретируется всеми основными браузерами (включая Internet Explorer версий 7 и 8 в стандартном режиме) при объявлении в `<!DOCTYPE...>` как HTML 4.01, так и XHTML 1.0. На всякий случай задана переходная схема (transitional).

> ***Примечание***
>
> XHTML-код с соответствующим описанием DTD браузеры Mozilla Firefox, Opera, Apple Safari, Google Chrome интерпретируют так же, как и аналогичный XML-код, чего нельзя сказать, к сожалению, о Microsoft Internet Explorer 8.0 и более ранних его версиях. Иначе говоря, для перечисленных браузеров за исключением Internet Explorer в XHTML-коде допустимы авторские (придуманные, как это делается в XML) теги при условии, что их представление определено в таблице CSS. Таким образом, для этих браузеров XHTML — действительно расширяемый, а не просто синтаксически похожий на XML язык.

> ***Совет***
>
> Если вы начинающий разработчик, то рекомендую вам сразу выбрать строгую схему и время от времени проверять валидность написанного вами кода с помощью онлайновой службы консорциума W3C. На сайте **http://validator.w3.org** вы можете ввести URL-адрес своего документа, расположенного на Web-сервере, и получить сведения о его валидности.

Возможны и другие варианты дескриптора <!DOCTYPE...> в зависимости от содержимого документа. Например, если (X)HTML-документ содержит вставку SVG-файла (описание масштабируемой векторной графики на языке SVG), то определение типа может иметь следующий вид:

```
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
```

Собственно SVG-документ может иметь такое определение типа:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

2.3. Структура (X)HTML-кода

В предыдущем разделе мы рассматривали варианты типа (X)HTML-документа, описываемые в дескрипторе <!DOCTYPE ...>, который располагается в самом начале документа. Далее следует собственно (X)HTML-код, заключенный между дескрипторами <html> и </html> (рис. 2.2). Теперь рассмотрим его структуру.

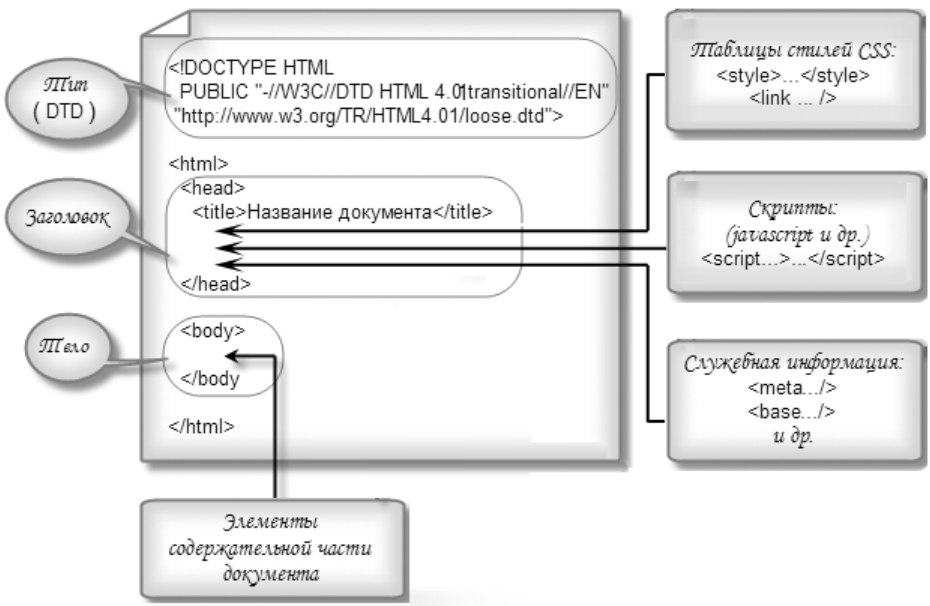


Рис. 2.2. Основные теги (X)HTML-документа

Контейнерный тег <html> может содержать атрибут lang, значением которого является идентификатор языка текстов вашего документа. Например, <html lang="ru"> для русскоязычных документов или <html lang="en"> — для англоязычных.

Примечание

Как уже отмечалось, некоторые браузеры обрабатывают XHTML-код по правилам XML и допускают авторские теги. Если у вас есть свои собственные теги в различных документах,

то может возникнуть коллизия имен тегов и атрибутов. Чтобы ее избежать, следует указывать пространство имен, т. е. URL-адрес файла, содержащего описание тегов и их атрибутов. Например, вы можете указать, что используете пространство имен HTML или же свое собственное пространство, заданное по правилам описания DTD. В (X)HTML-документах это делается с помощью атрибута `xmlns` тега `<html>`, например, так:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Подробнее о пространствах имен можно узнать на сайтах **http://www.w3.org/TR/REC-xml-names** (англ.) или **http://www.rol.ru/news/it/helpdesk/xnamsps.htm** (русский перевод).

Внутри тега `<html>` могут находиться еще несколько контейнеров, как показано на рис. 2.2.

В контейнере `<head>` должен быть контейнер `<title>`, в котором указывается название документа, отображаемое в заголовке окна или на вкладке браузера. Если он отсутствует, то браузер Internet Explorer отображает в своем заголовке окна путь к файлу, но другие браузеры могут этого не делать. Кроме того, в `<head>` помещают служебную информацию, а также те элементы, которые должны быть загружены в браузер прежде содержательной части документа, например, таблицы стилей и сценарии. Видимые элементы документа, относящиеся к его содержанию, в контейнер `<head>` не помещают. В (X)HTML-документе может находиться только один контейнер `<head>`.

Основная или содержательная часть (тело) документа размещается в контейнере `<body>`. В (X)HTML-документе может находиться только один контейнер `<body>`. В документе, приведенном на рис. 2.2, ничего содержательного, кроме названия, нет, поэтому клиентская область браузера пуста, а в заголовке его окна отображается название документа — содержимое тега `<title>`.

Совет

Создайте HTML-файл, содержащий "пустую" заготовку для будущей Web-страницы наподобие той, что показана на рис. 2.2. В дальнейшем вы будете добавлять в нее необходимое и удалять лишнее.

В программы на любых языках часто вставляют комментарии — тексты, которые не интерпретируются, а просто игнорируются исполнительной системой (например, браузером). В (X)HTML для этой цели предусмотрен специальный дескриптор

`<!--`*комментарий* `-->`.

Разумеется, комментарий не отображается, но загружается в браузер вместе с основным кодом и, следовательно, его наличие не способствует ускорению загрузки. Тем не менее ценность комментариев обнаруживается, когда вам требуется подкорректировать ваш документ, особенно спустя хотя бы несколько недель.

Совет

При разработке Web-страниц не скупитесь на комментарии к кодам. Перед окончательной публикацией сайта их можно удалить, чтобы сократить общий объем файлов.

На рис. 2.3 показан более содержательный пример HTML-документа. Это по существу конкретное воплощение структуры, рассмотренной ранее. В данном документе заголовок содержит название (`<title>`) документа "Тест", служебную информацию (`<meta>`) с указанием кодовой страницы windows-1251, а также таблицу стилей (`<style>`). В теле документа (`<body>`) находятся комментарий, заглавие (`<h1>`) и абзац (`<p>`).

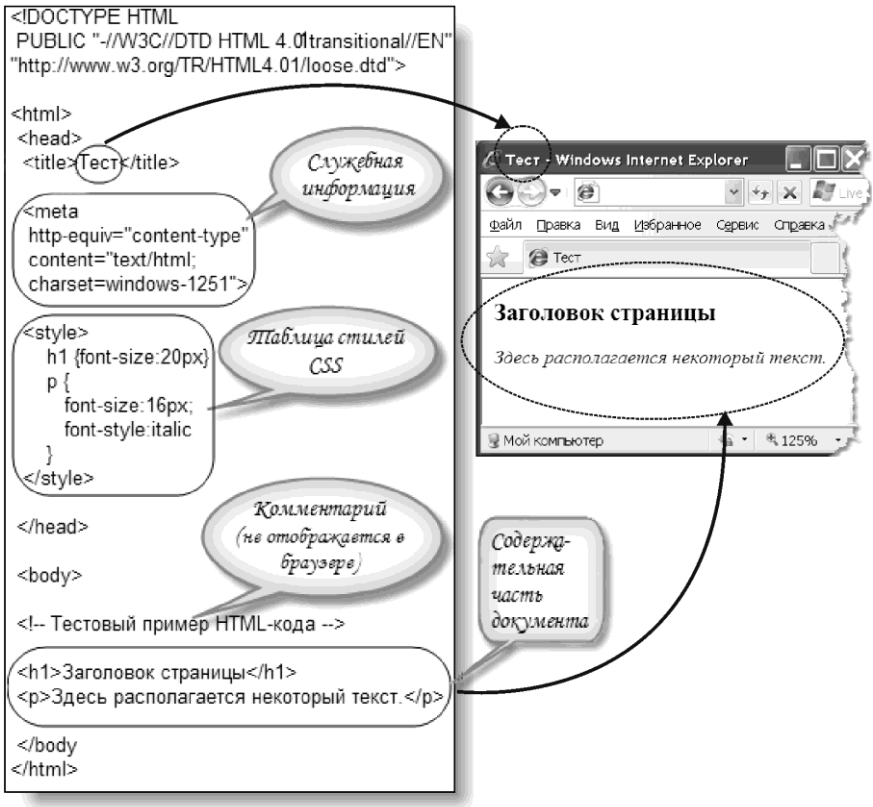


Рис. 2.3. Пример HTML-документа

Браузер Internet Explorer имеет фирменные расширения CSS, объектной модели документа и JavaScript, которые не соответствуют стандартам W3C и могут не восприниматься другими браузерами. Если же вы решили использовать специфические для Internet Explorer средства, то можно написать соответствующие фрагменты кода между дескрипторами `<!--[if IE]>` и `<![endif]-->`. Содержимое данного дескриптора будет воспринято Internet Explorer, а другие браузеры проигнорируют его подобно комментарию.

Итак, в контейнере `<html>` располагаются контейнеры `<head>` и `<body>`. В `<head>` вставляют контейнер `<title>`, а также, при необходимости, другие теги, содержащие информацию, которая не должна отображаться в клиентской области браузера.

2.3.1. Раздел заголовка документа *<head>*

Теперь рассмотрим типичное содержимое контейнерного тега `<head>`, определяющего заголовок (X)HTML-документа. Это не название, а раздел (X)HTML-кода, в котором размещают в основном служебную информацию. Как уже отмечалось, в нем должен находиться контейнер `<title>`, содержащий название документа, которое отображается лишь в заголовке окна или вкладки браузера. Если тега `<title>` нет, то Internet Explorer выводит в заголовке окна относительный путь к файлу, а другие браузеры ничего не

отображают. Другие элементы в <head> не обязательны и добавляются при необходимости. Рассмотрим основные из них.

Тег <*meta*>

Очень часто в <head> располагается служебная информация, заключенная в тегах <meta...>. В них указывают сведения о документе для браузеров и поисковых систем.

Листинг 2.3 иллюстрирует типичный пример тегов <meta>.

Листинг 2.3. Пример тегов <meta>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
  <title>Пример</title>
  <meta http-equiv="content-type" content="text/html;
        charset=windows-1251"/>
  <meta name="description"
       content="Статьи по веб-программированию"
  />
  <meta name="keywords" content="примеры html css javascript"/>
 </head>
 ... здесь располагаются другие теги документа
</html>
```

Здесь указано, что документ содержит текст и (X)HTML-код в файле, сохраненном в кодировке windows-1251 (кириллица для Windows), а также приведены описание и ключевые слова, которые должны учитываться поисковыми системами при индексировании данного документа. Обычно этих тегов вполне достаточно. Иногда их вообще не применяют. Однако в них можно заключить полезную служебную информацию для браузеров и поисковых систем.

> ***Совет***
>
> На первых этапах разработки кода страниц вы можете вставить только один тег <meta> с указанием кодировки. Описание, ключевые слова и др. обычно задаются на завершающих этапах, когда уже имеется информационное содержимое страницы.

Тегов <meta> в (X)HTML-документе может быть несколько в зависимости от того, какую информацию вы собираетесь сообщить браузеру. Эта информация задается с помощью атрибутов, которые удобно рассмотреть по группам.

Группа *HTTP-EQUIV* (HTTP-эквиваленты)

Данная группа включает следующие атрибуты.

- `expires` — дата устаревания документа.

  По истечении указанного срока документ будет каждый раз загружаться заново, а не браться из кэша на вашем локальном диске.

  Формат даты: RFC850.

Пример:

```
<meta http-equiv="expires" content="Tue, 26 Aug 2010 09:32:54 GMT"/>
```

- `pragma` — управление кэшированием.

  Возможно одно значение `NO-CACHE`, чтобы указать, что данный документ не должен кэшироваться браузером. В этом случае запрашиваемая страница будет браться с сервера, а не из кэша (буфера) пользовательского компьютера.

  Пример:

```
<meta http-equiv="pragma" content="no-cache"/>
```

- `content-type` — тип документа и его кодировка.

  Выбор кодовой страницы для правильного отображения символов браузером. Файлы (X)HTML-документов (файлы с расширением имени htm или html) могут быть сохранены в различных кодировках, например windows-1251 (кириллица для Windows), koi-8r (кириллица в Linux/UNIX), unicode-8 и др. В какой кодировке следует хранить (X)HTML-документы вашего сайта, зависит от Web-сервера, на котором вы их размещаете.

  Пример:

```
<meta http-equiv="content-type"
      content="text/html;charset=windows-1251"/>
```

- `content-language` — указание языка документа.

  Значение этого параметра может использоваться как поисковыми роботами, так и Web-серверами.

  Формат: *<Язык>-<Диалект>*.

  Примеры:

```
<meta http-equiv="content-language" content="en-GB"/>
<meta http-equiv="content-language" content="ru"/>
```

- `refresh` — время (в секундах), через которое произойдет автоматическая перезагрузка документа или переход на другой документ с заданным URL.

  Формат: *"время"* или *"время; URL"*

  Пример:

```
<meta http-equiv="refresh" content="30"/>
```

  Здесь указано, что документ следует перезагружать через 30 с.

```
<meta http-equiv="refresh" content="5; http://www.microsoft.com"/>
```

  Здесь указано, что спустя 5 с следует перейти к документу по заданному адресу.

  Автоматическую периодическую перезагрузку документа обычно назначают в случаях часто обновляемых данных (котировки акций, результаты спортивных соревнований и т. п.)

- `cache-control` — управление кэшированием.

  Возможные варианты: кэширование в общем (`public`) или частном (`private`) кэше. Документ вообще не кэшируется (`no-cache`) или кэшируется, но не сохраняется (`no-store`).

Пример:

```
<meta http-equiv="CACHE-control" content="no-store"/>
```

Группа *NAME* (имя)

К этой группе относятся следующие атрибуты.

- `description` — описание документа. Один из наиболее важных параметров. Информация, содержащаяся в нем, влияет на результаты поиска, осуществляемого поисковыми системами. В общем случае вид результатов поиска, как правило, выглядит так:
  - URL документа;
  - название документа (содержимое тега `<title>`);
  - описание документа, т. е. `description` или начальный фрагмент HTML-документа, если `description` отсутствует (нередко — и то и другое). В первом случае пользователь получает достаточно краткое и информативное описание документа, а во втором случае это может быть бессмысленный набор слов или несколько первых фраз из вашего документа;
  - рейтинг (коэффициент соответствия документа запросу пользователя).

  Пример:

  ```
  <meta name="description" content="Краткое описание данного документа />
  ```

- `keywords` — ключевые слова. Набор слов и фраз, наиболее полно характеризующих данный документ, которые, как предполагалось, являются основным критерием поиска вашего документа поисковыми системами. В конечном счете эти слова учитываются при выдаче результатов поиска и способствуют повышению рейтинга вашего сайта. Однако поисковые системы, создающие индексацию документов, могут игнорировать декларированные вами ключевые слова и учитывать некоторую часть текста самого документа, заключенного в контейнере `<body>`.

  Пример:

  ```
  <meta name="keywords" content="бизнес девушки алкоголь лошади яхты дельтапланы"/>
  ```

- `document-state` — статус документа. Данный параметр управляет частотой индексации вашего документа поисковыми серверами и может принимать два значения:
  - `static` (документ статичен, т. е. не меняется и, следовательно, индексировать его нужно только один раз).

    Пример:

    ```
    <meta name="document-state" content="static"/>
    ```

  - `dynamic` (для часто изменяющихся документов, которые нужно реиндексировать).

    Пример:

    ```
    <meta name="document-state" content="dynamic"/>
    ```

- `robots` — управление процессом индексации. Возможные варианты:
  - `index` — возможность индексирования данного документа (иначе — `noindex`);
  - `follow` — возможность индексирования всех документов, на которые есть ссылки в данном HTML-файле (иначе — `nofollow`);

- `all` — одновременное выполнение условий `index` и `follow`;
- `none` — одновременное выполнение условий `noindex` и `nofollow`.

Пример:

```
<meta name="robots" content="index, nofollow"/>
```

- `resource-type` — тип ресурса. Для обычных HTML-документов значение этого параметра устанавливается равным `"DOCUMENT"`.

  Пример:

  ```
  <meta name="resource-type" content="document"/>
  ```

- `updated` — дата обновления страницы.

  Пример:

  ```
  <meta name="updated" content="12.09.10"/>
  ```

- `url` — базовый URL-адрес; определяет, какой документ следует индексировать (чтобы не обрабатывать "зеркала").

  Пример:

  ```
  <meta name="URL" lang="ru" content=http://www.dunaevv1.narod.ru/>
  ```

- `author` — информация об авторе данного документа.

  Пример:

  ```
  <meta name="author" content="Вадим Дунаев"/>
  ```

- `copyright` — информация об авторских правах.
- `generator` — название программы, создавшей HTML-код.

Допустимо, но не обязательно, добавлять в метатеги атрибут `lang`, указывающий язык данных.

Пример:

```
<meta name="keywords" lang="ru" content="музыка видео графика"/>
```

Тег *<base>*

В (X)HTML-документах обычно используются относительные ссылки на ресурсы (другие (X)HTML-документы, графику и т. п.). Например, в ссылке нередко приводится только имя файла. В этом случае предполагается, что его местоположение (папка) совпадает с расположением документа, из которого он вызывается. Тем не менее можно явно указать базовый URL-адрес, который будет действовать во всех случаях применения относительных ссылок. Это делается с помощью тега `<base>` с атрибутом `href`, значением которого является базовый URL-адрес:

```
<base href="URL-адрес"/>
```

В листинге 2.4 графическое изображение имеет относительную ссылку (просто имя файла). При отсутствии базового URL-адреса браузер искал бы файл picture.jpg в той же папке, где расположен (X)HTML-документ со ссылкой на него. Поскольку базовый URL-адрес задан, то браузер будет искать графический файл в папке / images на Web-узле **www.dunaevv1.narod.ru**.

Листинг 2.4. Пример относительной ссылки

```
<html>
 <head>
  <title>Пример</title>
  <base href="http://www.dunaevv1.narod.ru/images"/>
 </head>
 <img src="picture.jpg">
</html>
```

При указании в теге <base> базового URL-адреса относительные ссылки продолжают работать, даже если вы переместите (X)HTML-документ в другую папку. Это обеспечивает определенные удобства при работе с локальной версией (X)HTML-документа в условиях, когда компьютер подключен к сети.

Другие теги внутри *<head>*

Нередко внутри контейнера <head> размещают еще теги <style> и <script>, хотя они могут находиться и в теле документа (т. е. в <body>, см. *разд. 2.4*). Эти теги служат, как уже говорилось, для размещения таблиц стилей (CSS) и сценариев. Располагать их именно в <head> целесообразно, если мы хотим, чтобы они загрузились в браузер прежде элементов документа, которые должны быть отображены в окне браузера.

Если таблица стилей находится во внешнем файле, то ссылку на нее (<link...>) также следует поместить в контейнер <head>. То же правило относится и к сценариям во внешних файлах, ссылки на которые задаются как значения атрибута src контейнера <script>. Листинг 2.5 содержит пример XHTML-кода пустой страницы, который загружает в браузер каскадную таблицу стилей и сценарий из внешних файлов mystyle.css и myscript.js соответственно.

Листинг 2.5. Пример XHTML-кода пустой страницы

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head>
  <title>Пример</title>
  <meta http-equiv="content-type" content="text/html;
        charset=windows-1251"/>
  <meta name="keywords" content="примеры xhtml html javascript Flash"/>

  <link rel="stylesheet" type="text/css"
    href="http://www.anyserver.ru/mystyle.css"/>
    <script type="text/javascript"
            src="http://www.anyserver.ru/myscript.js">
    </script>
 </head>
 </html>
```

Тег <link> позволяет установить связь данного документа с другим (например, с файлом, содержащим таблицу стилей или описания шрифтов).

Он имеет следующие атрибуты:

- href — URL-адрес файла, с которым устанавливается отношение, например, href="http://www.anyserver.ru/mystyle.css"; это не обязательный атрибут;
- rel — определяет тип отношения с внешним файлом; значения указывают, что связываемый файл:
  - stylesheet — содержит таблицу стилей;
  - application/rss+xml — содержит описание ленты новостей в формате XML;
  - home — является главной страницей сайта;
  - contents — содержит оглавление данного документа;
  - index — содержит информацию для индексного поиска в текущем документе;
  - glossary — содержит перечень терминов, относящихся к текущему документу;
  - copyright — содержит информацию об авторских правах;
  - help — содержит справочную информацию (подсказку);
  - first, start — указывает на первую страницу;
  - last, end — указывает на последнюю страницу;
  - next — указывает на следующую страницу;
  - previous — указывает на предыдущую страницу.
- type — указывает MIME-тип внешнего файла; например, для таблицы стилей задается "text/css", для HTML-файла — "text/html", для файла в формате PDF — "application/pdf";
- media — указывает тип устройства, для которого следует применять таблицу стилей; этот необязательный атрибут принимает значения:
  - all — все устройства;
  - screen — экран монитора (значение по умолчанию);
  - print — принтер;
  - projection — проектор;
  - braille — устройство, основанное на системе Брайля для слепых;
  - speech — синтезатор речи.

2.3.2. Раздел тела документа <*body*>

Основное содержимое документа задается тегами, размещаемыми в контейнере <body>, который следует непосредственно за <head>. В теле документа можно разместить обычный текст, графику и другие объекты из внешних файлов. Для этого служат специальные теги:

- <h1>, <h2>,..., <h6> — текстовые заголовки соответственно первого, второго, ..., шестого уровней (выделяются шрифтом и расположением);

- `<p>` — абзац (выделяется расположением);
- `<div>`, `<span>` — структурные элементы широкого назначения; часто используются при компоновке Web-страницы;
- `<a>` — ссылки на внешние документы и/или фрагменты того же самого документа;
- `<img>` — вставка графического изображения из внешнего файла;
- `<table>` — вставка таблицы (внутри данного тега обычно присутствуют дополнительные теги для задания строк и столбцов таблицы, такие как `<tr>`, `<td>` и др.); применяется при табличном представлении данных, а также при компоновке Web-страницы;
- `<ul>`, `<ol>` — списки; элементы списков задаются тегами `<li>`;
- `<object>`, `<embed>`, `<iframe>` — вставка видео, звука, графических изображений, Flash-роликов, (X)HTML-документов и других объектов;
- `<frameset>` — разбиение клиентской области браузера на фреймы, ведущие себя подобно главному окну браузера (внутри данного тега вставлены дополнительные теги `<frame>`); служит для компоновки Web-страницы (не рекомендуется применять);
- `<form>` — форма, содержащая обычно элементы пользовательского интерфейса (поля ввода данных, кнопки и переключатели), задаваемые посредством тегов `<input>`; данные, введенные в элементы формы, можно передать на сервер.

Здесь перечислены далеко не все, а только основные, наиболее употребительные теги. В контейнер `<body>` можно также вставлять таблицы стилей (`<style>`) и скрипты (`<script>`), хотя часто (но не всегда) лучшее для них место — контейнер `<head>`. Перечисленные, а также другие теги более подробно будут рассмотрены в следующих главах.

> ***Примечание***
>
> Если вы не упомянули в своем (X)HTML-коде тег `<body>`, то соответствующий ему контейнер все равно автоматически появится в структуре документа.

2.4. Основные атрибуты тегов

Как уже отмечалось, теги могут иметь атрибуты (параметры), записываемые в открывающем дескрипторе (X)HTML после имени тега в виде *имя_атрибута=значение*, например,

```
<img src="mypicture.jpg"/>.
```

Если атрибутов несколько, то их можно записывать в произвольном порядке, отделяя друг от друга пробелами. Например, записи

```
<img src="mypicture.jpg" alt="Мой портрет"/>
```

и

```
<img alt="Мой портрет" src="mypicture.jpg" />
```

эквивалентны.

Значения атрибутов в XHTML требуется указывать в кавычках, двойных или одинарных. В HTML кавычки обязательны только для значений, содержащих пробелы, т. е. для текстовых строк.

Для большинства тегов атрибуты не обязательны. Однако для некоторых тегов есть необходимые атрибуты, без указания которых тег теряет смысл (интерпретация его неоднозначна). Так, в теге

```
<img src="mypicture.jpg"/>
```

для вставки картинки из файла атрибут `src` обязателен, поскольку он указывает URL-адрес файла графического изображения. В теге `<a ...>` для ссылки необходимо указать либо атрибут `href="`*`URL-адрес`*`"`, либо `name="`*`имя_анкера`*`"` (если ссылка внутренняя, т. е. на какой-либо раздел текущего документа). В противном случае не ясно, на что собственно делается ссылка.

В HTML много атрибутов, предназначенных исключительно для определения внешнего вида элементов, таких как `align` (выравнивание), `color` (цвет), `width` (ширина), `height` (высота) и др. Однако вместо них можно (и рекомендуется) применять стилевые параметры CSS, а потому в данной книге мы будем стараться избегать атрибутов, предназначенных главным образом для визуализации элементов документа.

Некоторые теги имеют специфические атрибуты, употребляемые только в них, например, у тега `<link>` есть особенный атрибут `rel`. Вместе с тем, существуют атрибуты, которые пригодны для большинства тегов, и их удобно разделить на следующие группы:

- базовые, применяемые к большинству тегов:
  - `id` — идентификатор элемента для обеспечения доступа к элементу из сценария, а также для ссылки на правила CSS;
  - `class` — для ссылки на класс CSS (поименованный набор стилевых параметров);
  - `style` — для встраивания набора стилевых параметров, относящихся к данному тегу;
  - `title` — для указания текста всплывающей подсказки.

  Пример:

  ```
  <a href="mypicture.jpg"
    id="mybestref" class="myref" title="Полная версия"
    style="font-size:14pt">Подробности
  </a>
  ```

  Здесь задается ссылка на графическое изображение из файла mypicture.jpg; отображение ссылки диктуется классом (`class="myref"`) стилевых параметров CSS, который дополнен (или модифицирован) параметрами, указанными в значении атрибута `style`; при наведении указателя мыши на ссылку появится подсказка с текстом "Полная версия" (о стилевых параметрах см. *главу 3*); чтобы иметь доступ к данному элементу (ссылке) из CSS или сценария, ему был присвоен уникальный идентификатор (`id="mybestref"`);
- атрибуты-события, используемые в сценариях для обработки соответствующих событий (щелчок мышью, наведение указателя мыши и т. п.): `onclick`, `ondblclick`, `onkeydown`, `onkeypress`, `onkeyup`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`.

  Эти атрибуты позволяют привязать к элементам сценарии, которые должны выполниться, если произойдет соответствующее событие. Они применимы для всех тегов,

задающих видимые элементы документа, такие как заголовки, абзацы, картинки, поля ввода, кнопки и др.

Для некоторых элементов возможны и другие события, например `onload` (окончание загрузки документа), `onselect` (выбор элемента) и т. д.

Пример:

```
<img src="myphoto.jpg" onclick="myfunction()"
     title="Щелкните для увеличения"
/>
```

Здесь щелчок на картинке вызывает некоторую функцию `myfunction()`, код которой написан на скриптовом языке, например JavaScript;

- языковые, указывающие на особенности языка: `lang` (язык текста), `dir` (направленность текста).

  Пример:

  ```
  <html lang="ru">
  ```

- фокус (ссылки на состояние элемента, когда он выделен и готов принять данные): `accesskey`, `onblur`, `onfocus`, `tabindex`.

В HTML 5 существенно расширен список атрибутов-событий и добавлены следующие общие атрибуты (**www.w3.org/TR/html-markup/common-attributes.html**):

- `conteneditable` — определяет, можно ли редактировать содержимое элемента;
- `contextmenu` — идентифицирует ассоциированное с элементом меню как контекстное;
- `draggable` — определяет, можно ли элемент перемещать мышью;
- `spellcheck` — определяет, подлежит ли содержимое элемента проверке правописания.

ГЛАВА 3

Основы каскадных таблиц стилей

Если (X)HTML-код задает логическую структуру (каркас) Web-страницы, то каскадные таблицы стилей (CSS) определяют ее внешний вид. В данной главе рассматривается устройство таблиц стилей и их интеграция с (X)HTML-кодом.

3.1. Что такое CSS

Параметры внешнего вида документа можно задать с помощью *каскадных таблиц (листов) стилей* (Cascading Style Sheets, CSS). Таблицу стилей, подобно шаблону форматирования текстов, можно разработать отдельно от конкретного (X)HTML-документа, а затем применить к нему. Модификация содержимого таблицы стилей меняет внешний вид (X)HTML-документов, не затрагивая их структуры и информационного содержания. Одна и та же таблица стилей может применяться к нескольким документам и, наоборот, к одному и тому же документу может быть применено несколько таблиц стилей. В последнем случае браузер учитывает приоритеты таблиц и по определенным правилам разрешает возникающие конфликты, в результате чего таблицы выстраиваются неким каскадом (отсюда и название — каскадные таблицы стилей).

Кроме технологичности стилизации (X)HTML-документов, CSS обеспечивает еще и произвольное позиционирование элементов. Для любого элемента можно задать размеры и координаты расположения, а также другие параметры визуализации. Так что, применяя CSS, можно обойтись без тегов таблиц и фреймов, которые все еще широко используются как средство компоновки Web-страниц.

CSS содержит наборы стилевых параметров или, другими словами, правила форматирования. Например, если требуется определить для всех заголовков первого уровня шрифт Courier 20 пунктов красного цвета, то соответствующее правило можно записать так:

```
h1 {font-family: Courier; font-size: 20pt; color: red}
```

Здесь в фигурных скобках указан список стилевых параметров — пар вида *имя:значение*, разделяемых точкой с запятой; левее этого списка указано имя `h1` тега, к которому данные параметры требуется применить. В результате заголовок первого уровня (содержимое тега `<h1>`) приобретает специальные параметры, установленные автором стиля, а не принятые по умолчанию, когда авторский стиль не задан.

Одно и то же правило можно применить к различным элементам и, наоборот, для одного и того же элемента можно задать несколько правил. Следующее правило определяет одинаковые стилевые параметры одновременно для заголовков первого и второго уровней:

```
h1, h2 {font-family: Courier; font-size: 24pt; color: red}
```

А вот пример каскадного задания стилевых параметров:

```
h1, h2 {font-size: 24pt; color: red}
h1  {font-family: Arial}
h2  {font-family: Courier}
```

Здесь сначала задаются размер и цвет шрифта для обоих заголовков, а затем для каждого из них определяется своя гарнитура шрифта (имя шрифта).

Стилевые параметры, заключенные в фигурные скобки, можно назначить не только именам тегов, но и тегам по значениям их атрибута `id` (идентификатора элемента). Так можно изменить представление не всех, например, заголовков первого уровня (тегов `<h1>`), а только тех, которые имеют указанное значение атрибута `id`. Существует также возможность определить правила, изначально не связанные ни с какими элементами документа, а затем сослаться на них из любого элемента с помощью атрибута `class`. Указатели на элементы, к которым применяются стилевые параметры в фигурных скобках, называются селекторами. Селекторы, как мы только что видели, могут быть различных видов. Подробнее о них будет рассказано в *разд. 3.3*.

Правила CSS можно применить к любому видимому элементу (X)HTML-документа. Почти каждому элементу, заданному тем или иным тегом, можно придать произвольное представление. Поэтому оказывается вполне достаточно одного или нескольких тегов, например, `<div>` и/или `<span>`, а визуальное разнообразие их вхождений в документ можно обеспечить посредством только CSS. Однако при этом, возможно, будет нанесен ущерб свойству (X)HTML представлять логическую структуру документа. Действительно, глядя на (X)HTML-документ, содержащий одни только теги `<div>`, будет трудно понять, где заголовок текстового документа, а где абзац основного текста. Поэтому многие опытные разработчики рекомендуют не придерживаться чрезмерного "тегового минимализма", а использовать основные структурообразующие теги по прямому назначению. Например, теги `<h1>`, ..., `<h6>` следует применять только для создания заголовков, а не для оформления, скажем, гиперссылок. Кстати сказать, HTML 5 явно поддерживает идею семантической определенности тегов, вводя в оборот элементы `<header>`, `<nav>`, `<article>`, `<section>` и т. п.

Правила CSS можно записать непосредственно в (X)HTML-документе или сохранить в отдельном файле, чтобы применить их одновременно к нескольким документам. В последнем случае браузер кэширует таблицу стилей и, следовательно, последующие страницы, задействующие ту же таблицу стилей, загружаются быстрее. Правила, записываемые непосредственно в самом (X)HTML-документе, заключают в контейнерный тег `<style>`, а для ссылки на таблицу стилей, расположенную во внешнем файле, в контейнер `<head>` вставляют тег

```
<link href="URL css-файла"...>
```

или в контейнер `<style>` добавляют директиву

```
@import url("URL css-файла");
```

На рис. 3.1 показан пример таблицы стилей, непосредственно вставленной в (X)HTML-документ. Если правила этой таблицы разместить в текстовом файле mystyle.css, то вместо тега `<style>` следует указать ссылку на этот файл, например, так:

```
<link rel="stylesheet" type="text/css"
href="http://www.anyserver.ru/mystyle.css" />
```

или (в рамках тега `<style>`) так:

```
@import url("http://www.anyserver.ru/mystyle.css");
```

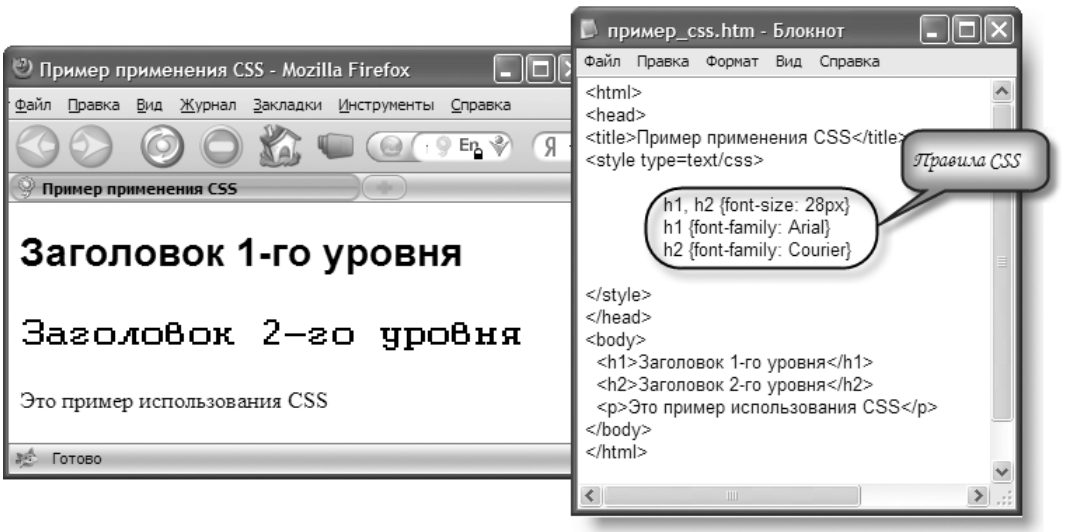


Рис. 3.1. Пример использования таблицы стилей в (X)HTML-документе

Спецификация каскадных таблиц стилей уровня 1 (CSS 1) была принята в качестве рекомендации W3C в декабре 1996 г., а в январе 1999 г. вышел ее откорректированный вариант. Рекомендация спецификации уровня 2 (CSS 2) была выпущена в мае 1998 г. Она сохраняла обратную совместимость с CSS 1. Пересмотренный ее вариант (CSS 2.1) как кандидат в рекомендации увидел свет в апреле 2008 г. (**www.w3.org/TR/CSS2**). Работа над CSS 2.1 продолжается, в настоящее время это наиболее актуальная версия, поддерживаемая всеми ведущими браузерами почти полностью. Предполагается, что CSS 2.1 войдет в будущую спецификацию CSS 3, которая предоставляет существенно новые возможности, начиная от простых визуальных эффектов (тени, скругленные углы) и кончая трансформацией и анимацией. С текущим состоянием работ по CSS 3 можно познакомиться, например, по адресу **www.w3.org/Style/CSS/current-work**.

Хотя спецификация CSS 3 пока находится в разработке, многие браузеры уже поддерживают, в той или иной степени, некоторые из ее элементов. При этом имена параметров, возможно, придется модифицировать путем добавления префикса в зависимости от движка браузера. Так, для Firefox (движок Gecko) добавляется префикс `-moz-`, для Safari и Chrome (движок WebKit) — `-webkit-`. Для Opera префикс не требуется. Например, параметр `box-shadow`, задающий тень от элемента, для Firefox записывается как `-moz-box-shadow`, а для Safari и Chrome — как `-webkit-box-shadow`. В то же время параметр `opacity`, задающий степень прозрачности элемента, записывается без префикса для всех

перечисленных браузеров. С выпуском новых версий браузеров необходимость добавления к именам параметров специфических префиксов постепенно отпадет. Наиболее заметно это стало с выходом в свет Firefox 4 и Chrom 10. Internet Explorer 8.0 не поддерживает CSS 3.

Следует также иметь в виду, что Microsoft Internet Explorer поддерживает ряд своих фирменных (нестандартных) расширений CSS, которые не воспринимаются другими браузерами. К таким расширениям относятся так называемые графические фильтры, которые позволяют в Internet Explorer модифицировать загруженное графическое изображение (например, повернуть его, сделать полупрозрачным и др.). Задать цвет полосы прокрутки окна браузера также можно с помощью специальных параметров CSS, но только для Internet Explorer. Для обеспечения межбраузерной совместимости рекомендуется не применять специфические фирменные расширения CSS и осторожно использовать новые возможности CSS 3. По крайней мере, следует проверить их работу в нескольких ведущих браузерах. В данной книге применение параметров CSS 3 будет оговариваться особо.

Теперь мы рассмотрим более подробно, как каскадные таблицы вставляются в (X)HTML-код и как в них создаются правила форматирования документа.

3.2. Присоединение таблиц стилей к (X)HTML-документу

CSS содержат правила форматирования (стилевые параметры или свойства элементов), которые можно присоединить к (X)HTML-документу несколькими способами:

- Внедрение — запись правил непосредственно в (X)HTML-коде внутри контейнера `<style>`, например,

```
<style type="text/css">
  p {color:blue; font-size:12pt}
</style>
```

  Внедренная указанным образом таблица стилей будет действовать только на тот документ, в котором находится. Контейнер `<style>` рекомендуется размещать внутри `<head>`, хотя возможно и иное его расположение.

- Связывание — запись правил в отдельном текстовом файле (обычно с расширением css) и помещение в (X)HTML-коде ссылки на него. Это можно сделать с помощью тега `<link>`, который обычно размещают в контейнере `<head>`:

```
<head>
<link rel="stylesheet" type="text/css"
      href="URL-адрес файла с правилами CSS" />
</head>
```

  или с помощью директивы `@import` в контейнере `<style>`:

```
<style type="text/css">
  @import url("URL-адрес файла с правилами CSS");
</style>
```

Внимание!

Обратите внимание на атрибуты тегов <link> и <style>, а также на точку с запятой в конце записи директивы @import. Теги <link> и <style> имеют еще необязательный атрибут media для указания типа устройства вывода, принимающий значения "all" (все устройства), "screen" (экран монитора, принимается по умолчанию), "print" (печатающие устройства), "projection" (проектор), "braille" (устройства, основанные на системе Брайля для слепых), "speech" (речевые синтезаторы и программы звукового воспроизведения текста). В конце директивы @import можно указать через запятую и без кавычек список типов устройств: all, screen, print, projection, braille, aural (речевые синтезаторы и программы звукового воспроизведения текста), handheld (карманные компьютеры и т. п.), tv (телевизор). Вместо данных способов указания устройств вывода можно применить директиву @media (см. *разд. 3.3*).

Директиву @import можно использовать также и в таблице стилей, сохраненной во внешнем файле.

Таблица стилей из внешнего файла действует на те документы, в которых содержится ссылка на нее. Разумеется, она передается браузеру вместе с документом, который на нее ссылается. Однако браузеры могут кэшировать загруженные данные и при повторном запросе таблицы стилей, последняя не будет передаваться с сервера снова, если она еще находится в кэше. Хранение таблицы стилей в отдельном файле выгодно, если она применяется к нескольким документам одного сайта. В этом случае вы корректируете стилевые параметры в одном файле, а их действие распространяется сразу на несколько (X)HTML-документов.

- Встраивание — указание стиля непосредственно в теге с помощью атрибута style: <тег style="*список параметров*">, например,

```
<h1 style="color:red; font-size:36pt">Большой красный заголовок</h1>
```

Данный способ удобен, когда хотят назначить индивидуальные стилевые параметры для некоторых элементов только в одном документе, не корректируя основную таблицу стилей в контейнере <style> или во внешнем файле. Правила форматирования, заданные таким образом, обладают наивысшим приоритетом (подробнее см. *разд. 3.4*).

К одному и тому же документу можно присоединить несколько таблиц стилей, причем различными способами. Их взаимодействие будет определяться приоритетами, которые мы рассмотрим в *разд. 3.4*.

3.3. Правила форматирования

Таблица стилей, внедренная или связанная (расположенная во внешнем файле), содержит одно или несколько правил форматирования, каждое из которых записывают в соответствии с синтаксисом:

список_селекторов {*имя_параметра*:*значение*; *имя_параметра*:*значение*;... }

Здесь *список_селекторов* — один или несколько идентификаторов (селекторов), разделенных запятыми. Селекторы предназначены для указания элементов документа, к которым данное правило применяется. Стилевые параметры (свойства) заключают в фигурные скобки и разделяют точкой с запятой; имя параметра и его значение разделяют двоеточием.

3.3.1. Селекторы

Элементарные селекторы:

- ***** — параметры с данным селектором (звездочка) применяются ко всем элементам, например,

```
* {font-size:20px};
```

  Данное правило предписывает отображать шрифт всех элементов документа размером 20 px;

- *тег* — имя HTML-тега без уголковых скобок, например, div; параметры применяются ко всем элементам, заданным тегом <*тег*>; например,

```
h1, h2 {font-family: Arial; font-size:14pt}
```

  Здесь всем заголовкам первого и второго уровней (т. е. элементам, заданным посредством тегов <h1> и <h2>) назначается шрифт Arial размером 14 pt;

- *#идентификатор* — произвольное слово, подобное имени переменной, перед которым ставится #, например, #myelement; для применения параметров с данным селектором к элементу последний должен иметь атрибут id="*идентификатор*" (листинг 3.1). Поскольку значение атрибута id должно быть уникальным, то стилевые параметры применяются к единственному элементу.

Листинг 3.1. Пример использования значения атрибута id как селектора

```
<style type="text/css">
     #mytext {color:red}
</style>
<p id="mytext">Совершенно секретно</p>
<p>Подробности правил форматирования можно найти в справочнике по CSS</p>
```

  Здесь красный цвет текста назначается только первому абзацу, заданному тегом <p id="mytext">, а второй абзац будет отформатирован согласно правилам по умолчанию;

- *.класс* — произвольное слово, подобное имени переменной, перед которым ставится точка, например, .mystyle2; для применения параметров CSS с данным селектором к элементам последние должны иметь атрибут class="*класс*"; класс применяется к тем элементам, у которых значением атрибута class является имя этого класса (листинг 3.2).

Листинг 3.2. Пример использования атрибута class

```
<style type="text/css">
    .italic {font-style:italic}
    .underline {text-decoration:underline}
</style>
<h1>Обычный заголовок</h1>
<h1 class="italic">Заголовок курсивом</h1>
<h1 class="underline">Заголовок подчеркнутый</h1>
```

В листинге 3.2 определены два класса (`italic` и `underline`), применимые только к элементам, у которых атрибут `class` имеет соответствующее значение (`"italic"` или `"underline"`). Три заголовка первого уровня в данном (X)HTML-коде будут иметь различный вид. Первое вхождение в документ элемента `<h1>` будет отображено в окне браузера в соответствии с параметрами, установленными по умолчанию, поскольку оно не ссылается на CSS. Представления остальных двух вхождений `<h1>` будут подкорректированы в соответствии с наборами параметров, на которые они ссылаются посредством атрибута `class`: содержимое второго вхождения `<h1>` будет представлено курсивом (`font-style:italic`), а третьего — подчеркнутым (`text-decoration:underline`).

Из элементарных образуются составные селекторы, позволяющие уточнить область действия соответствующих стилевых параметров:

- ***селектор*.*класс*** — элементы, соответствующие *селектор*, у которых атрибут `class="`*класс*`"`. Например, область действия селектора `div.small` — все элементы вида `<div class="small">`;
- ***селектор*#*идентификатор*** — элементы, соответствующие *селектор*, у которых атрибут `id="`*идентификатор*`"`. Например, область действия селектора `div#mybox` — элемент вида `<div id="mybox">`.

В листинге 3.3 класс `h1.mystyle` определен только для элементов вида `<h1 class="mystyle">`, а класс `.mystyle` — для всех элементов, имеющих атрибут `class="mystyle"`. Поэтому заголовок `<h1>` будет подчеркнутым и красным, первый абзац `<p>` — красным, а второй абзац в соответствии с параметрами по умолчанию будет обычным (черным).

Листинг 3.3. Пример составного селектора

```
<style type="text/css">
     h1.mystyle {text-decoration:underline;}
     .mystyle {color:red}
</style>
<h1 class="mystyle">Заголовок подчеркнутый</h1>
<p class="mystyle">Красный текст</p>
<p>Обычный</p>
```

Кроме перечисленных, составными селекторами являются псевдоселекторы и псевдоэлементы (*см. далее*).

3.3.2. Контекстные селекторы

Иногда требуется установить стиль для элементов, которые находятся в определенном контексте с другими элементами. В данном случае селектор образуется из элементарных и составных селекторов, между которыми устанавливается пробел, `>` или `+` в зависимости от вида учитываемого контекста. Есть и другие правила образования контекстных селекторов. В табл. 3.1 указаны основные виды контекстных селекторов и область действия правил с их участием.

Таблица 3.1. *Контекстные селекторы*

Селектор	Область действия
селектор1 селектор2	Элементы, соответствующие *селектор2*, которые находятся внутри элементов, соответствующих *селектор1*. Например, область действия селектора `div span` — все элементы `<span>` внутри элементов `<div>`
селектор1>селектор2	Элементы, соответствующие *селектор2*, для которых родительский элемент соответствует *селектор1*. Например, область действия селектора `div>span` — все элементы `<span>`, для которых родительским является `<div>`
селектор1+селектор2	Элементы, соответствующие *селектор2*, которым непосредственно предшествует элемент, соответствующий *селектор1*. Например, область действия селектора `div+span` — все элементы `<span>`, непосредственно перед которыми находится `<div>`
селектор[*attr*]	Элементы, соответствующие *селектор*, у которых установлен атрибут *attr* (независимо от его значения). Например, область действия селектора `img[alt]` — все элементы графических изображений с установленным атрибутом `alt`
селектор[*attr*='*значение*']	Элементы, соответствующие *селектор*, у которых атрибут *attr* имеет значение *значение*. Например, область действия селектора `img[alt='Мой портрет']` — все элементы графических изображений с атрибутом `alt ='Мой портрет'`
селектор[*attr*~='*значение*']	Элементы, соответствующие *селектор*, у которых атрибут *attr* имеет в качестве значения список слов, разделенных пробелами, и одно из этих слов есть *значение*
селектор[*attr*|='*значение*']	Элементы, соответствующие *селектор*, у которых атрибут *attr* имеет в качестве значения список слов, разделенных дефисами, и первое слово есть *значение*

Вот несколько примеров контекстных селекторов:

```
div #mytext { параметры }
div .myclass { параметры }
.myclass1 .myclass2 { параметры }
#myelem >.myclass { параметры }
div * { параметры }
```

Сложный селектор может содержать более двух простых.

3.3.3. Псевдоселекторы и псевдоэлементы

В табл. 3.2 приведены псевдоселекторы и псевдоэлементы, которые применяются как составные части селекторов: перед псевдоклассами и псевдоэлементами указывается элементарный или составной селектор из рассмотренных ранее.

В следующем примере псевдоклассы служат для задания ссылкам с различным статусом (обычные, посещенные и под указателем мыши) различных оттенков красного цвета:

```
a:link {color:#aa0000}
a:visited {color:#800000}
a:hover {color:#ff0000}
```

На рис. 3.2 в качестве примера показан документ, содержащий три абзаца (<p>), которые отформатированы по-разному, поскольку они ссылаются на различные правила CSS.

Таблица 3.2. *Псевдоселекторы и псевдоэлементы*

Наименование	Обозначение
:first-child	Первый дочерний элемент
:first-line	Первая строка элемента
:first-letter	Первая буква в элементе
:hover	Элемент, на котором находится указатель мыши
:active	Активный элемент
:focus	Элемент, на котором установлен фокус
:link	Неиспользованные ссылки
:visited	Использованные ссылки
:lang(язык)	Элемент с указанным языком язык
:before	Предшествующий элемент
:after	Следующий элемент

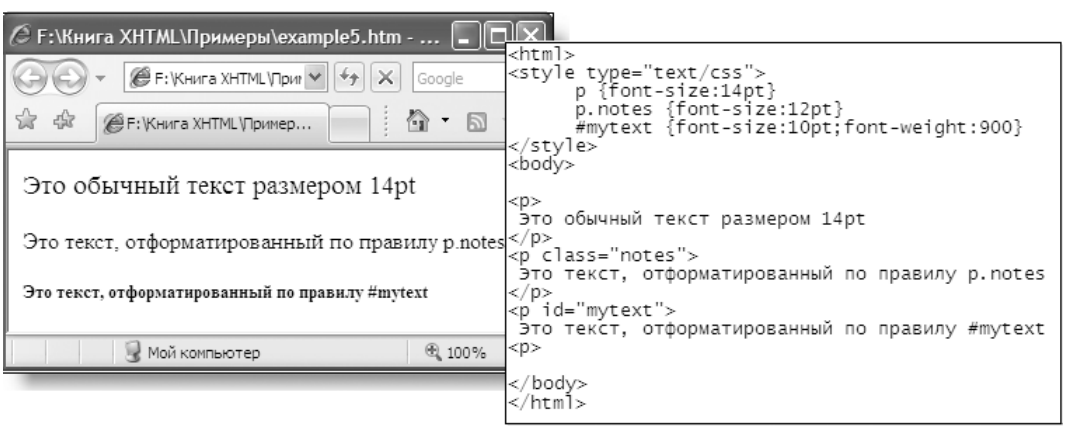


Рис. 3.2. Примеры ссылок на правила CSS

В коде CSS можно и полезно добавлять комментарии, которыми считается все, что заключено между символами /* и */.

Важно иметь в виду, что браузеры игнорируют правила CSS с ошибками. Наиболее типичные ошибки — отсутствие размерности для числовых значений и пропуск точки с запятой в качестве разделителя между определениями параметров.

Для различных элементов документа можно определить разные правила форматирования в зависимости от устройства вывода (типа носителя). Например, представление документа на листе бумаги нередко требуется сделать иным, чем на экране компьютера. С этой целью можно применить директиву

```
@media список_типов_носителей {
     правила_форматирования
}
```

Здесь после ключевого слова `@media` указывается без кавычек один или несколько типов носителей (через запятую): `all` (все типы), `screen` (экран монитора, принимается по умолчанию), `print` (печатающее устройство), `projection` (проектор), `braille` (устройство, основанное на системе Брайля для слепых), `aural` (речевые синтезаторы и программы звукового воспроизведения текста), `handheld` (карманные компьютеры и т. п.), `tv` (телевизор). Далее в фигурных скобках следуют правила форматирования.

Пример

```
@media screen {
  body {font:12px Arial; color:blue}
}
@media print, handheld {
  body {font: 8pt Times; color:black}
  img {display:none}
}
```

3.4. Приоритеты определений параметров стилей

При использовании CSS очень важно понимать, что такое приоритеты определений параметров стилей. Хотя это — весьма запутанная тема, можно выделить несколько относительно простых и наиболее часто применимых правил учета приоритетов.

Не исключено, что в таблице стилей для одного элемента (X)HTML-документа может оказаться несколько различных определений одного и того же параметра. Вероятность такого события возрастает, если к одному документу применено одновременно несколько различных таблиц. Возникающие при этом противоречия браузер разрешает по некоторым правилам, учитывая приоритеты (веса) определений параметров во всех правилах CSS, относящихся к данному элементу документа.

Может статься, что для какого-то элемента автор явно не задал в CSS правило для одного или нескольких параметров. В этом случае элемент все равно приобретет некоторые значения для таких параметров. Кроме таблиц стилей, создаваемых автором документа, браузер учитывает CSS пользователя, заданные в настройках браузера, а также значения параметров, установленные по умолчанию. По умолчанию приоритет таблиц стилей пользователя меньше, чем автора документа. Если между ними возникает конфликт, то применяются таблицы автора. Самым низким приоритетом обладают стилевые параметры, принимаемые браузером по умолчанию. Так что ни один элемент вашего документа не останется без параметров форматирования. Вместе с тем, если вы не контролируете форматирование, результат может оказаться неожиданным.

Итак, если для одного и того же элемента подходят одновременно несколько правил форматирования из одной или нескольких различных таблиц, то они применяются каскадно по следующему алгоритму:

1. Наивысшим приоритетом обладает определение параметра, заданное атрибутом `style` тега рассматриваемого элемента.

Пример

```
<style type="text/css">
p {font-size:16pt}
</style>
<p style="font-size:20pt">Привет!</p>
```

Здесь для абзаца (содержимого элемента `<p>`) задано два определения размера шрифта (`font-size`), но второе из них имеет больший приоритет, а потому текст абзаца будет иметь размер 20 pt.

Если определения, заданного атрибутом `style`, нет, то рассматриваются все определения в правилах с селекторами, соответствующие рассматриваемому элементу (см. п. 2). Если таковых нет, то используется значение, наследуемое от родительского элемента. Если оно не определено (не все параметры наследуются), то берется значение по умолчанию. О наследовании см. *разд. 3.8*.

2. Если элементу соответствует несколько определений параметра, то они упорядочиваются по специфичности селектора, вычисляемой следующим образом. Подсчитываются три числа:
   - k1 — количество в селекторе значений атрибута `id`;
   - k2 — количество в селекторе классов;
   - k3 — количество в селекторе имен тегов.

   Эти числа записывают как k1k2k3, и полученное таким образом число представляет уровень специфичности (вес) селектора. Выбираются определения параметров с наибольшей специфичностью. В табл. 3.3 приведены примеры.

3. Если элементу все еще соответствует несколько определений параметра, то они упорядочиваются по своему происхождению (от низшего к высшему): принятые браузером по умолчанию, пользовательские, авторские.

4. Если элементу все еще соответствует несколько определений параметра, то они сортируются по порядку их упоминания в таблице стилей. При этом определения в импортированной таблице (вызываемой директивой `@import`) считаются предшествующими определениям таблицы, в которую они были импортированы. Наибольший приоритет имеет последнее определение.

Чаще всего употребляют определения стилевых параметров, заданные не атрибутом `style`, а правилами с селекторами, причем простыми, такими как имя тега, класс и значение атрибута `id`. Приоритеты (специфичность или вес) определений параметров в правилах с такими селекторами нетрудно запомнить (табл. 3.4).

Таблица 3.3. *Примеры специфичности селекторов*

Селектор	Значения k1, k2, k3	Специфичность
`h1`	k1=0, k2=0, k3=1	1
`div p`	k1=0, k2=0, k3=2	2
`.myclass`	k1=0, k2=1, k3=0	10
`h1.myclass`	k1=0, k2=1, k3=1	11
`#myelement`	k1=1, k2=0, k3=0	100
`#myelement.myclass`	k1=1, k2=1, k3=0	110

Таблица 3.4. *Приоритеты простых селекторов*

Вид селектора	Вес
имя_тега	1
.имя_класса	10
имя_тега.имя_класса	11
#значение_id	100
#значение_id. имя_класса	110

На рис. 3.3 показано применение четырех правил к одному элементу, заданному тегом `<p>`. Результат определяется приоритетами правил CSS. Правило `p.notes` сильнее, чем правило `p`, но слабее правила `#mytext`. Самое сильное правило задано значением атрибута `style`. Поэтому размер шрифта элемента `<p>` определится именно этим правилом. Кроме того, шрифт приобретет вес (жирность), указанный параметром `font-weight` в третьем правиле, поскольку он не задан в других правилах.

Приоритетом правил можно управлять с помощью директивы `!important` (англ. *important* — важный, значительный). Эта директива записывается сразу после стилевого параметра, приоритет которого следует увеличить, например,

```
h2 {font-size:24pt !important; color:red}
```

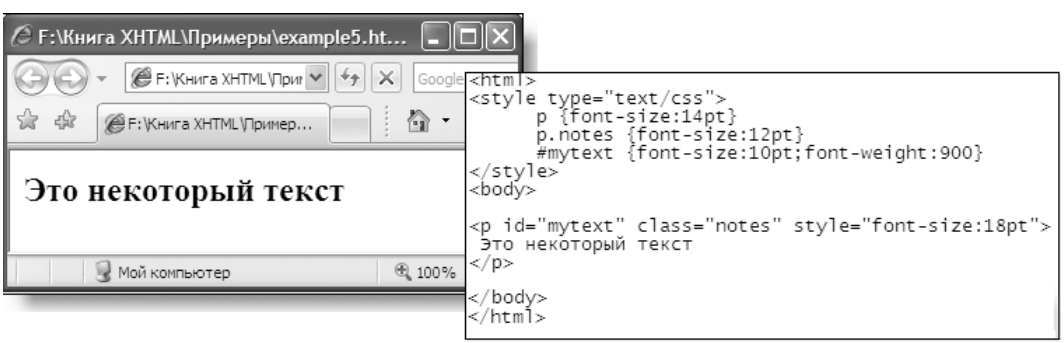


Рис. 3.3. Пример использования нескольких правил CSS к одному элементу

Параметр таблицы стилей пользователя, помеченный как `!important`, имеет более высокий приоритет, чем тот же параметр автора, заданный без этой директивы. Авторский параметр с пометкой `!important` обладает более высоким приоритетом, чем пользовательский, даже если он имеет такую же пометку.

3.5. Размерность

Многие стилевые параметры принимают в качестве значений числа. Так задают размеры шрифта, видимых элементов, отступов от границ экрана или объемлющих контейнеров. При этом необходимо для всех чисел, кроме нуля, указывать размерность, т. е. в каких единицах измеряется та или другая числовая величина.

В CSS имеются следующие единицы измерения:

- абсолютные — `cm` (сантиметры), `mm` (миллиметры), `in` (дюймы), `pt` (пункты) и `pc` (пики);
- относительные — `em`, `ex` и `%`;
- пикселы — `px`.

Величины абсолютных единиц не зависят от устройств отображения (дисплеев, принтеров и т. п.). Например, отрезок длиной 1 cm будет иметь эту же длину и на дисплее, и на листе бумаги. Абсолютные единицы связаны между собой следующими равенствами: 1 in = 2.54 cm, 1 pt = 1/72 in, 1 pc = 12 pt.

Все иначе для относительных единиц. 1 em равен размеру так называемого родительского шрифта, а 1 ex — высоте строчной буквы "x". Например, в следующем правиле устанавливается отступ заголовка (`<h1>`) от краев контейнера в четыре раза больший, чем размер его шрифта (значение `font-size`, установленное браузером по умолчанию или в других правилах CSS):

```
h1 {margin:4em}
```

Проценты (%) показывают отношение к величине некоторого другого элемента. Для каждого свойства, допускающего задание значений в процентах, подразумевается еще и величина, относительно которой определяется процентное соотношение. Например, такой величиной может выступать значение ширины или высоты контейнера, содержащего данный элемент, или величина текущего шрифта.

Пикселы (`px`) часто относят к абсолютным единицам измерения, хотя они имеют относительный характер. Именно поэтому здесь они выделены в отдельную категорию. Дело в том, что видимый размер отрезка прямой, заданный в пикселах, зависит от устройства отображения (в отличие от абсолютных единиц), но не зависит от размеров других элементов (в отличие от относительных единиц).

Устройства отображают графическую информацию дискретно, в виде набора цветных точек, размеры и общее количество которых ограничено. Эти точки (dots) нередко называют пикселами (pixels), хотя последние — элементы графического растрового изображения, а не физической структуры дискретного отображения. Элемент изображения (pixel) не имеет абсолютных размеров, а элемент системы отображения (dot) — имеет, что и ограничивает ее разрешающую способность. Если, например, монитор настроен на отображение по горизонтали 1280 точек (экранных пикселов), то горизонтальный

отрезок с установленной длиной `256px` займет на экране 20% ширины. А если монитор настроен на отображение 256 точек, то тот же отрезок займет всю ширину экрана. Поскольку существуют мониторы с одинаковым количеством точек, но имеющие различные абсолютные линейные размеры и наоборот, то становится очевидным — длина отрезка прямой в пикселах является относительной величиной. Тем не менее в данной книге, говоря об абсолютных единицах измерения, мы будем иметь в виду и пикселы.

3.6. Цвет

Цвет задают в системе RGB последовательностью из трех чисел, определяющих яркость красной (Red), зеленой (Green) и синей (Blue) составляющих, записываемой в шестнадцатеричном или десятичном форматах.

В шестнадцатеричном формате каждая составляющая цвета представляется двухразрядным шестнадцатеричным числом от `00` до `ff` (всего 256 различных значений), указывающим ее яркость. Самый темный цвет — `00`, а самый яркий — `ff`. Например, черному соответствует шестнадцатеричное число `000000`, а белому — `ffffff`. Вот пример задания чистого красного цвета для текста в CSS:

```
color:#ff0000.
```

Если в каждой паре шестнадцатеричных чисел цифры одинаковые, то допускается сокращенная трехразрядная запись. Например, вместо `#ffbb88` можно написать `#fb8`.

Чтобы задать цвет десятичными числами, применяется функция `rgb(r, g, b)`, где *r*, *g*, *b* — целые десятичные числа от 0 до 255, соответствующие яркости красной, зеленой и синей составляющих смеси цветов. Возможно также указание значений яркости в процентах. Например, красный цвет для текста в CSS задается так:

```
color: rgb(255,0,0)или color: rgb(100%,0,0).
```

Напомню, что различные оттенки серого цвета (от черного до белого) задаются равными значениями для трех составляющих, например, `color:#808080` или `color: rgb(100,100,100).`

В CSS 3 имеется функция `rgba(r, g, b, a)`, отличающаяся от `rgb(r, g, b)` четвертым параметром, задающим уровень непрозрачности `(от 0 до 1)`. Данный, очень удобный способ задания цвета поддерживается всеми современными браузерами, кроме Internet Explorer 8.

Кроме числовых значений цвета допустимо указывать их имена, являющиеся ключевыми словами, например, `color: red`. В табл. 3.5 приведены имена и шестнадцатеричные значения для некоторых наиболее популярных в Web цветов.

Таблица 3.5. *Имена и числовые значения основных цветов*

Имя в CSS	Числовое значение	Название
aqua	00ffff	Аква (бирюзовый)
aquamarine	7fffd4	Аквамарин
beige	f5f5dc	Бежевый

Таблица 3.5 *(окончание)*

Имя в CSS	Числовое значение	Название
black	000000	Черный
blue	0000ff	Голубой
brown	a52a2a	Коричневый
cyan	00ffff	Циан (бирюзовый)
gold	ffd700	Золотой
gray	808080	Серый
green	008000	Зеленый
magenta	ff00ff	Светло-пурпурный
navy	000080	Синий
olive	808000	Оливковый
pink	ffc0cb	Розовый
purple	800080	Пурпурный
red	ff0000	Красный
sienna	a0522d	Охра
silver	c0c0c0	Серебристый
violet	ee82ee	Фиолетовый
white	ffffff	Белый
yellow	ffff00	Желтый

Все графические изображения, создаваемые для Web, должны использовать либо цветовую модель RGB, либо подмножество индексированных цветов, либо представляться в градациях серого цвета. Это общее правило. Указание имен (некоторые из которых приведены в табл. 3.5) гарантирует принадлежность ваших цветов Web-палитре, содержащей 216 цветов, которые отображаются одинаково всеми цветными мониторами.

3.7. Поля, отступы, границы и размеры

Элементы (X)HTML-документа с точки зрения CSS представляются в виде прямоугольных блоков, каждый из которых имеет информативную область (content — содержимое), отступы (padding), границы (border) и поля (margin), как показано на рис. 3.4.

Периметр каждой из четырех областей называется ее краем или краевой линией. Край имеет четыре сегмента: верхний (top), правый (right), нижний (bottom) и левый (left).

Фон различных областей блока определяется следующим образом:

- **информативная область (content)** — свойство `background` элемента, порождающего блок (подробнее см. *разд. 5.1*);

- **отступы (padding)** — свойство `background` элемента, порождающего блок;
- **граница (border)** — специальные свойства границ элемента, порождающего блок;
- **поля (margin)** — всегда прозрачны (transparent).

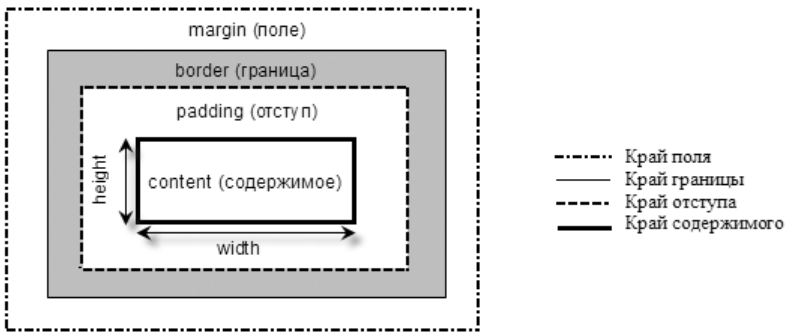


Рис. 3.4. Блоковая модель элемента документа

Поля, отступы и границы могут иметь различную ширину (в том числе и нулевую, а поля — **даже отрицательную**) по каждой из четырех сторон прямоугольника. Например, ширина верхнего поля задается свойством `margin-top`, а левого — `margin-left`; ширина правого и нижнего полей задается свойствами `margin-right` и `margin-bottom` соответственно. Чтобы задать одинаковую ширину в 10 пикселов для всех четырех полей, можно воспользоваться свойством `margin:10px`; выражение `margin:10px 20px 10px 20px` задает указанные значения соответственно для верхнего, правого, нижнего и левого полей. Запись такого вида еще называют сокращенной. Неплохо запомнить эту последовательность, начинающуюся от верха (top) и далее — по часовой стрелке. Числовые значения задаются обязательно с указанием единиц измерения, абсолютных и относительных. Аналогичные свойства имеются для отступов и границ: `padding` и `border`. Поля (margin) могут принимать еще значение `auto`, при котором браузер вычислит их ширину автоматически. В частности, данное значение может быть использовано для расположения элемента в центре окна браузера.

Для всех видимых элементов ширина полей, отступов и границ имеет по умолчанию нулевое значение. Исключение представляет элемент `<body>`, для которого свойство `margin` имеет ненулевые значения, причем различные в разных браузерах. Так, в Internet Explorer 7.0 и более ранних версиях значение `margin` по умолчанию равно `15px` для верха и низа и `10px` — для боковых полей; в остальных браузерах (в том числе и в Internet Explorer 8.0) все поля имеют ширину `8px`. Для обеспечения межбраузерной инвариантности внешнего вида документа следует явным образом задавать значение `margin` для `<body>` в вашем документе.

В листинге 3.4 приведен HTML-документ с правилами CSS для элементов `<body>` и `<img>`. На соответствующем ему рис. 3.5 видны поля, отступы и границы элемента `<img>`. Обратите внимание, что элемент `<body>` имеет ненулевое значение свойства `margin`, задаваемое по умолчанию. Для других элементов свойства `margin`, `padding` и `border-width` имеют нулевое значение и, следовательно, поля, отступы и границы не видны.

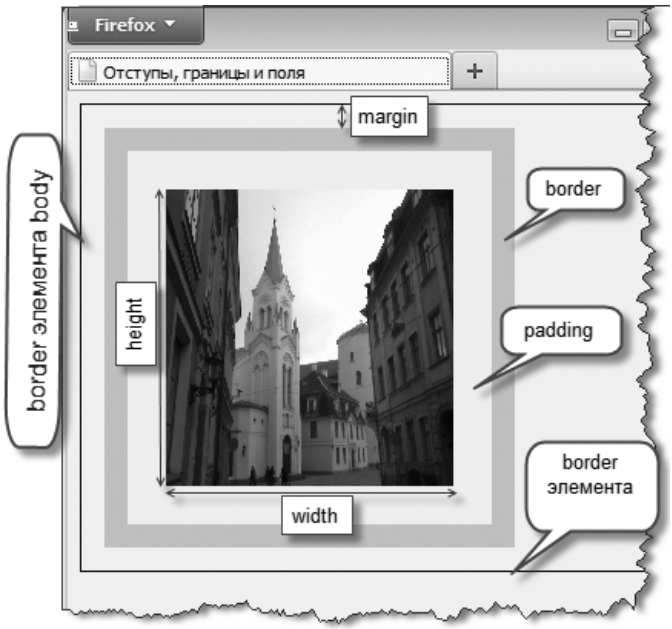


Рис. 3.5. Поля, отступы и границы картинки, заданной тегом <img>

Листинг 3.4. Поля, отступы и границы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
 <title>Отступы, границы и поля</title>
</head>
<style type="text/css">
body {
     background:#f0f0f0;   /* цвет фона */
     border-width:1px;     /* ширина границы */
     border-style:solid;   /* граница — сплошная линия */
     border-color:#000000; /* цвет границы — черный */
}
img {
     width:187px;          /* ширина картинки */
     height:193px;         /* высота картинки */
     padding:25px;         /* отступ */
     margin:15px;          /* поле */
     background:#eeeeee;   /* цвет фона */
     border-width:15px;    /* ширина границы */
     border-style:solid;   /* граница — сплошная линия */
     border-color:#c0c0c0  /* цвет границы — серый */
}
</style>
```

```
<body>
<img src="picture.jpg" />
</body>
</html>
```

Границам можно назначить еще цвет и стиль. Цвет задается значением, присваиваемым параметру `border-color`, например,

```
border-color : red или border-color : #ff0000
```

Можно также задавать цвет отдельных составляющих границы: `border-top-color`, `border-right-color` и т. д. Задание стиля (типа) границы в явном виде необходимо, чтобы она была отображена, т. к. по умолчанию граница не видна и не занимает место. Если стиль границы не задан в явном виде, другие ее параметры не будут действовать.

Стиль четырех сторон границы задается следующими параметрами: `border-top-style`, `border-right-style`, `border-left-style`, `border-bottom-style`.

Значения стиля границы задаются ключевыми словами:

- `none` — граница не отображается (по умолчанию), даже если указан размер границы (`border-width`);
- `solid` — сплошная линия;
- `double` — двойная линия (сумма толщины двух линий и промежутка между ними определяется параметром `border-width`);
- `dotted` — точечная линия;
- `dashed` — штриховая линия;
- `grove` — вдавленный желобок;
- `ridge` — выпуклый валик;
- `inset` — весь блок элемента вдавлен;
- `outset` — весь блок элемента выпуклый.

Кроме того, для задания стиля сразу всех или только некоторых сторон границы можно использовать параметр `border-style` (без уточнений типа `top`, `right` и т. п.), подобно тому, как применяются описанные параметры `margin` и `padding`. Сокращенная запись для задания параметров всех четырех фрагментов границы имеет вид

```
border: стиль ширина цвет
```

Например,

```
border: solid 1px #ffbb00
```

В листинге 3.5 приведен код, иллюстрирующий применение нескольких различных стилей к элементу `<p>` (абзац). Для всех абзацев задана ширина границы `5px`. Первый абзац не стилизован (`border-style:none`) и поэтому он не имеет видимых границ. Для остальных абзацев указаны конкретные значения стилей. Различия отображения границ разными браузерами обусловлены различиями в значениях цвета границ по умолчанию (рис. 3.6). С целью обеспечения межбраузерной инвариантности вы можете принудительно задать цвет для различных сторон границы с помощью свойства `border-color` или ряда свойств типа `border-top-color`, `border-right-color` и т. д.

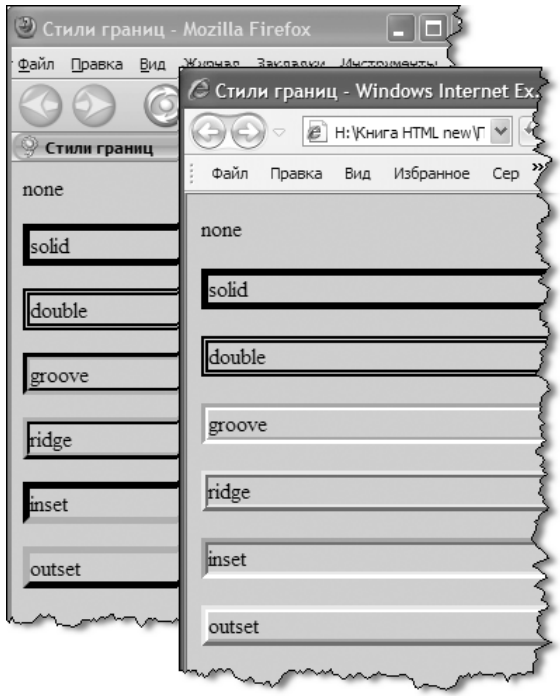


Рис. 3.6. Стили границ

Листинг 3.5. Стили границ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
  <head><title>Стили границ</title>
    <style>
      body {background-color:#d0d0d0}
      p    {
        border-width:5px;
        border-style:none
      }
      #p1  {border-style:solid}
      #p2  {border-style:double}
      #p3  {border-style:groove;}
      #p4  {border-style:ridge;}
      #p5  {border-style:inset;}
      #p6  {border-style:outset}
    </style>
  </head>
  <body>
    <p>none</p>
    <p id=p1>solid</p>
    <p id=p2>double</p>
```

```
    <p id=p3>groove</p>
    <p id=p4>ridge</p>
    <p id=p5>inset</p>
    <p id=p6>outset</p>
  </body>
</html>
```

Границу элемента можно заполнить графическим изображением из файла. Для этого в CSS 3 предусмотрен параметр `border-image`. Здесь мы рассмотрим лишь основные значения данного параметра в следующем формате:

```
border-image: url("URL файла изображения") ломтики повторение;
```

Первая часть значения параметра `border-image` задает графическое изображение, которое будет использоваться для заполнения границы элемента.

Вторая часть определяет девять фрагментов (ломтиков, slice) исходного изображения, применяемых для заполнения четырех углов и четырех сторон границы элемента, а также окаймляемой ею центральной области. Таким образом, заполняется не только граница (ее размеры задаются параметром `border-width`), но и создается графический фон самого элемента.

Третья часть определяет, как графические изображения ломтиков будут заполнять стороны границы и центральную область. Для отмены заполнения границы изображением служит значение `none`.

Ломтики задаются посредством от одного до четырех чисел, определяющих отступы линий разреза от верхнего, правого, нижнего и левого краев исходного графического изображения. Для задания относительных величин отступов указывается символ %. В противном случае (без единиц измерения) подразумеваются пикселы. Например,

```
border-image: url("myimage.png") 10% 20 15 25;
```

На рис. 3.7 показана схема деления исходного изображения на девять ломтиков с помощью четырех чисел n1, n2, n3 и n4.

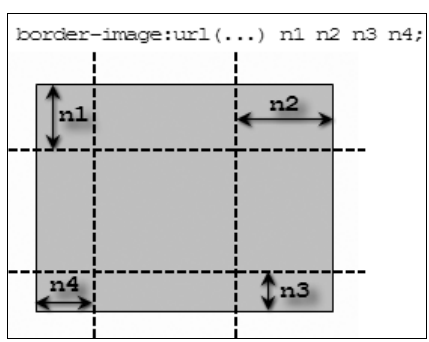


Рис. 3.7. Разрезка графического изображения на ломтики

Если четвертое число не указано, то его значение устанавливается равным второму; если третье число также отсутствует, то его значение равно первому; если и второе число отсутствует, то его значение равно первому. Отрицательные значения недопус-

тимы, а значения, превышающие соответствующий размер изображения, интерпретируются как 100%.

Ломтики могут перекрываться. Так, если n2 + n4 больше или равно ширине изображения, то ломтики для верхней и нижней сторон границы элемента, а также центральная область окажутся пустыми. Аналогично, пустыми будут ломтики для левой, правой сторон и центральной области, если n1 + n3 окажется больше либо равным высоте изображения.

Ломтики могут по-разному заполнять стороны границы и центральной области в зависимости от третьей части значения параметра `border-image` (*повторение*), которое может принимать в качестве значения одно или два из ключевых слов: `stretch` (по умолчанию), `round` и `repeat`. Если второе слово отсутствует, то подразумевается, что оно совпадает с первым. Если оба слова опущены, то предполагается `stretch`. На рис. 5.6 показано, как действуют данные значения на заполнение сторон границы и центральной области. Исходное изображение разрезано на девять примерно одинаковых по размерам ломтиков. Если значение *повторение* опущено или задано `stretch`, то ломтики будут масштабированы так, чтобы полностью заполнить соответствующий участок границы и центральной области. При значении `round` ломтики масштабируются так, чтобы в заполняемой области поместилось целое их количество. При значении `repeat` ломтики тоже масштабируются и повторяются, но без учета требования их целочисленности. Более сложные преобразования происходят в случае применения двух различных ключевых слов. Однако в настоящее время различные браузеры выполняют их по-разному. Как это должно быть, можно узнать из спецификации (**www.w3.org/TR/css3-background/#the-border-image**).

На рис. 3.8 показаны элемент (например, `<div>`), исходное графическое изображение, разделенное на девять одинаковых по размеру ломтиков, и четыре варианта заполнения ими данного элемента.

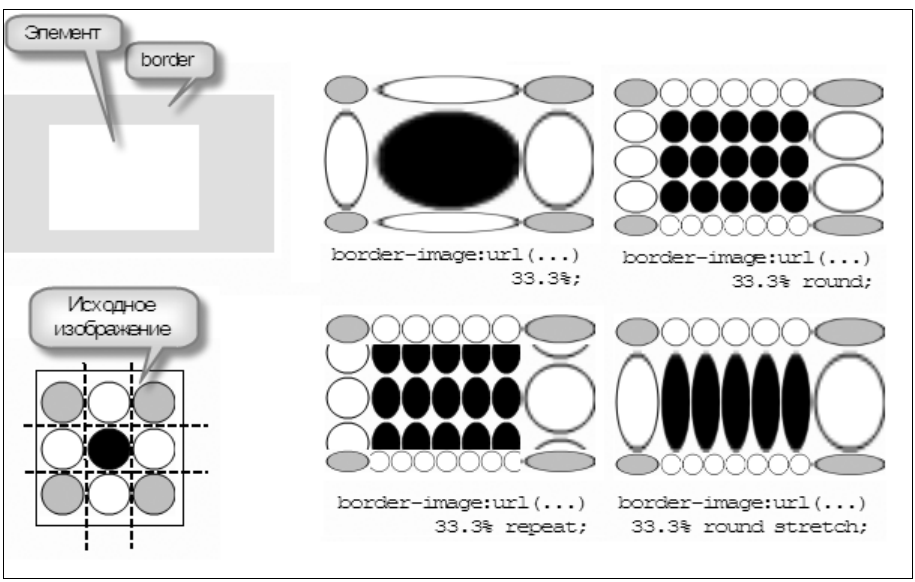


Рис. 3.8. Варианты заполнения границ и центральной области ломтиками изображения

Пример использования параметра `border-image` для создания панели с закругленными верхними углами и градиентным фоном приведен на рис. 3.9. Исходное изображение в данном случае разделяется на шесть ломтиков, поскольку отступ линии разреза от нижнего края равен нулю. Размеры границ элемента `<div>` и угловых ломтиков примерно одинаковы. В настоящее время для применения `border-image` в Firefox, Safari и Chrome рекомендуется к имени параметра добавлять специфический префикс (для Opera этого не требуется). В приведенном на рис. 3.9 коде определение `border-image` подойдет для всех перечисленных браузеров.

Для создания границ элементов с закругленными углами в CSS 3 имеется параметр `border-radius`, задающий радиус закругления в единицах длины или процентах. Например, `border-radius: 5px` определяет радиус закругления всех углов равным 5 пикселам. В качестве значения можно задать радиусы верхнего левого, верхнего правого, нижнего правого и нижнего левого углов. Например, `border-radius: 5px 10px 10px 30px`. При этом закругления будут представляться дугой окружности. Возможно также задание закруглений по эллиптической дуге, задаваемой двумя радиусами — горизонтальным и вертикальным. Наборы горизонтальных и вертикальных радиусов разделяются прямым слэшем, например,

```
border-radius: 5px 10px 10px 30px/10px 20px 15px 5px
```

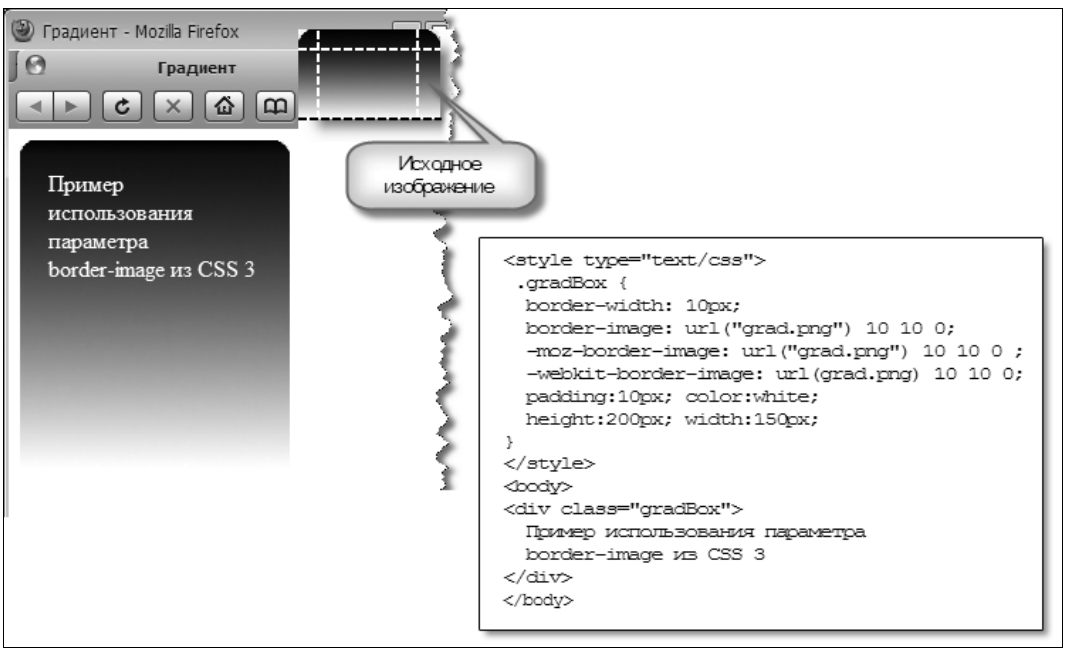


Рис. 3.9. Пример использования параметра `border-image`

В частности, параметр `border-radius: 5px/10px` задает для всех углов границы элемента горизонтальный и вертикальный радиусы, равные соответственно 5 и 10 пикселам. Значение в процентах для горизонтального радиуса определяется относительно ширины элемента, а для вертикального — относительно высоты.

На рис. 3.10 показано несколько вариантов применения параметра `border-radius`.

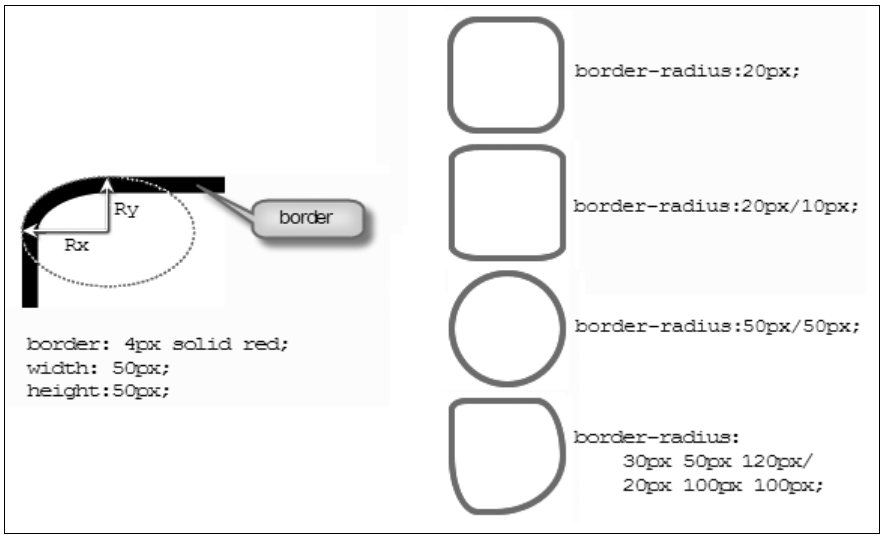


Рис. 3.10. Примеры использования параметра `border-radius`

Значения радиусов углов можно также определить и отдельно с помощью стилевых параметров `border-top-right-radius`, `border-bottom-right-radius`, `border-bottom-left-radius` и `border-top-right-radius`. Подробнее о параметре `border-radius` можно узнать по адресу **www.w3.org/TR/css3-background/#border-radius**.

Итак, любой элемент занимает на экране некоторую прямоугольную область. Размеры области содержимого прямоугольного блока определяются свойствами `width` (ширина) и `height` (высота). По умолчанию эти свойства имеют значение `auto`, при котором размеры элемента выбираются автоматически так, чтобы по возможности отобразить его целиком. Так, по умолчанию элемент `<div>` вместе с полями, границами и отступами занимает всю ширину клиентской области браузера, а по высоте — столько, сколько необходимо для отображения целиком содержащихся в нем элементов плюс поля, границы и отступы. Вместе с тем, ширину и высоту можно задать явным образом числами с указанием единиц измерения, например, `width:120px`; `height:50mm`, `width:80%`. Однако в этом случае возможны различные варианты отображения элементов. Более подробно параметры `width` и `height` будут рассмотрены в *разд. 4.10*.

Если указать свойства `width` и `height` для графического элемента (`<img>`), то картинка будет сжата или растянута в зависимости от ее исходных размеров. В случае контейнерных тегов (например, `<div>`) видимое взаимное расположение собственно контейнера и его содержимого может быть различным. Например, картинка (`<img>`), расположенная в контейнере `<div>`, в окне браузера может оказаться вне прямоугольника своего контейнера. Иначе говоря, видимое расположение элементов может не соответствовать логической структуре документа. Рассмотрим несколько примеров.

В листинге 3.6 приведен HTML-документ, в котором три контейнера `<div>` содержат тексты, а на рис. 3.11 показан его внешний вид. Браузер создает блоки `<div>` с указанными в CSS размерами, но при этом содержимое может выходить за их границы. В данном случае строка `Текст2Текст2Текст2` без внутренних пробелов не поместилась

во второй (вложенный) контейнер . Текстовая строка в третьем состоит из слов, разделенных пробелами, и поэтому браузер с помощью переносов разместил ее в блоке.

Листинг 3.6. Контейнеры с текстовым содержимым

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>div с текстом</title>
<style type="text/css">
div {
     width:100px;
     height:120px;
     background:#cccccc;
     padding:10px;
     margin:20px;
     border-width:1px;
     border-style:solid; /* сплошная линия */
     border-color:#000000
}
#indiv { /* вложенный div */
     width:50px;
     height:60px;
     background:#eeeeee;
     padding:10px;
     margin:20px;
     border-width:1px;
     border-style:solid;
     border-color:#000000
}
</style>
</head>
<body>
<div>
  Текст1
  <div id="indiv">
    Текст2Текст2Текст2
  </div>
</div>
<div>
  Текст3 Текст3 Текст3 Текст3
</div>
</body>
</html>
```

В листинге 3.7 приведен HTML-документ, в котором контейнер содержит графическое изображение, а его внешний вид в браузере показан на рис. 3.12.

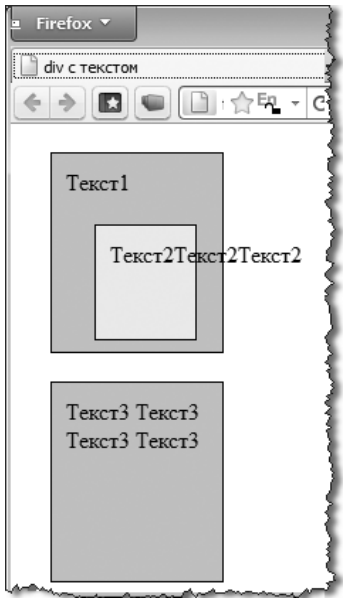


Рис. 3.11. Вид HTML-документа листинга 3.6 в браузере

Рис. 3.12. Вид HTML-документа листинга 3.7 в браузере

Листинг 3.7. Контейнер с графическим изображением

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>div с графикой</title>
<style type="text/css">
div {
     width:100px;
     height:120px;
     background:#cccccc;
     padding:10px;
     margin:20px;
     border-width:1px;
     border-style:solid;
     border-color:#000000
}
img {
     background:white;
     padding:10px;
     margin:20px;
     border-width:5px;
     border-style:solid;
     border-color:#000000
}
</style>
</head>
```

```
<body>
<div>
  <img src="picture.jpg"/>
</div>
</body>
</html>
```

Мы рассмотрели несколько случаев, когда содержимое контейнера полностью в нем не помещается. Однако отображением содержимого контейнера можно управлять. Как это делать, будет рассказано в *главе 4*.

3.8. Наследование параметров

Некоторые стилевые параметры, установленные для одних элементов, наследуются их дочерними элементами. Например, цвет текста (`color`), установленный для элемента `<body>`, будет иметь место и для всех вложенных в него других элементов (например, `<h1>`, `<p>`, `<div>` и т. д.), если только для них данный параметр не задан явным образом. Наследование параметров позволяет сэкономить место и время при создании таблиц стилей.

В листинге 3.8 представлен код документа с тремя элементами `<div>`, причем второй вложен в первый. Весь текст внутри первого и второго `<div>`, включая заголовок `<h1>`, будет красным, т. к. параметр `color:red` здесь явно задан для первого `<div>` и наследуется вложенным в него вторым `<div>`. Текстовое содержимое третьего `<div>` будет зеленым, поскольку параметр `color:green` задан для родительского элемента `<body>`.

Листинг 3.8. Наследование параметров

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Наследование</title>
<style type="text/css">
body {
  color:green
}
#div1 {
  border: solid 1px; /* граница */
  color:red          /* цвет текста */
}
</style>
</head>
<body>
<div id="div1">Блок 1
  <div id="div2">Блок 2
    <h1>Это некоторый заголовок</h1>
  </div>
</div>
```

```
<div id="div3">Блок 3</div>
</body>
</html>
```

По умолчанию наследуются все параметры шрифта (`font`) и списков (`list`), цвет (`color`), многие параметры текста (`text`) и некоторые другие. Однако большинство параметров CSS по умолчанию не наследуется. Если они не заданы явно ни в авторской, ни в пользовательской таблицах стилей, то получают некоторые значения по умолчанию, принятые браузером. Например, параметры границы (`border`) не наследуются по умолчанию. В примере листинга 3.8 для первого элемента `<div>` задана граница (`border: solid 1px`), но ее не будет у вложенного (дочернего) второго элемента `<div>`.

Вместе с тем можно принудительно задать наследование параметра, т. е. установить такое же значение, что и у родительского элемента. Для этого достаточно в определении параметра присвоить ему значение `inherit`. Например, если в таблицу стилей листинга 3.8 добавить правило `#div2 {border:inherit}`, то второй элемент `<div>` отобразится с видимой границей.

ГЛАВА 4

Позиционирование элементов

Для верстки Web-страниц важно уметь позиционировать ее элементы относительно окна браузера и друг друга. В данной главе рассматривается естественное расположение элементов и принудительное, задаваемое параметрами CSS.

4.1. Расположение элементов в нормальном потоке

Прежде чем научиться верстать Web-страницы, желательно понять, как происходит отображение задаваемых тегами элементов самих по себе, без применения каскадных таблиц стилей (CSS) и специальных атрибутов форматирования. Другими словами, речь идет о так называемом отображении элементов в нормальном или, как еще говорят, прямом потоке, т. е. в порядке их упоминания в (X)HTML-коде без специальных атрибутов, влияющих на отображение, а также без применения правил CSS, задающих внешний вид и геометрические параметры.

Теги (X)HTML, задающие видимые элементы Web-страницы, разделяются на два вида: блочные (block) и внутристрочные (in-line). Если не предпринимать специальных мер, то блочные элементы отображаются друг под другом и занимают всю ширину клиентской области браузера. Внутристрочные элементы, напротив, следуют друг за другом по горизонтали (в одной строке) насколько позволяет ширина окна; только в противном случае (ширина окна уже не позволяет) они отображаются ниже (переходят на новую строку). Таким образом, набор внутристрочных элементов будет выглядеть в браузере так или иначе в зависимости от ширины его окна, установленной пользователем. Не забывайте, что пользователь может ее менять несколько раз в течение одного сеанса посещения вашей страницы, и элементы могут "прыгать" по странице.

Типичные блочные элементы — контейнеры `<div>`, `<h1>`, `<h2>`,...,`<h6>`, `<p>`, а внутристрочные — `<img>`, `<a>`, `<span>`, `<iframe>`. Тег `<div>` задает универсальный блочный, а `<span>` — универсальный внутристрочный контейнер. По умолчанию эти универсальные контейнеры не видны в окне браузера, а служат главным образом для формирования структуры документа. В них можно вставить что угодно. Именно поэтому их называют универсальными. Хотя универсальные контейнеры сами по себе не видны, они влияют на расположение своего содержимого (например, текста и картинок). Однако с помощью CSS этим элементам можно придать видимое оформление, например, цвет

внутренней области и границы, а также размеры и расположение. В результате они предстанут в браузере как цветные прямоугольники, расположенные друг относительно друга некоторым заданным образом. Если же стилевые параметры позиционирования не применять, то невидимые прямоугольники вместе со своим содержимым будут размещаться, как говорят, в прямом (нормальном) потоке их упоминания в (X)HTML-коде. Так, контейнеру `<div>` соответствует прямоугольник, занимающий по ширине всю клиентскую область окна браузера, а по высоте — столько, сколько необходимо для размещения его содержимого. Прямоугольник для `<span>` принимает размеры, объемлющие содержимое этого контейнера, но не более.

На рис. 4.1 в качестве примера показано, как располагаются контейнеры `<div>` и `<span>`. Первый контейнер `<div>` содержит обычный текст, текст в контейнере `<span>`, а также контейнер `<div>`; третий `<div>` содержит только текст. Чтобы они стали видимыми в окне браузера, пришлось с помощью CSS задать фоновый цвет и границы.

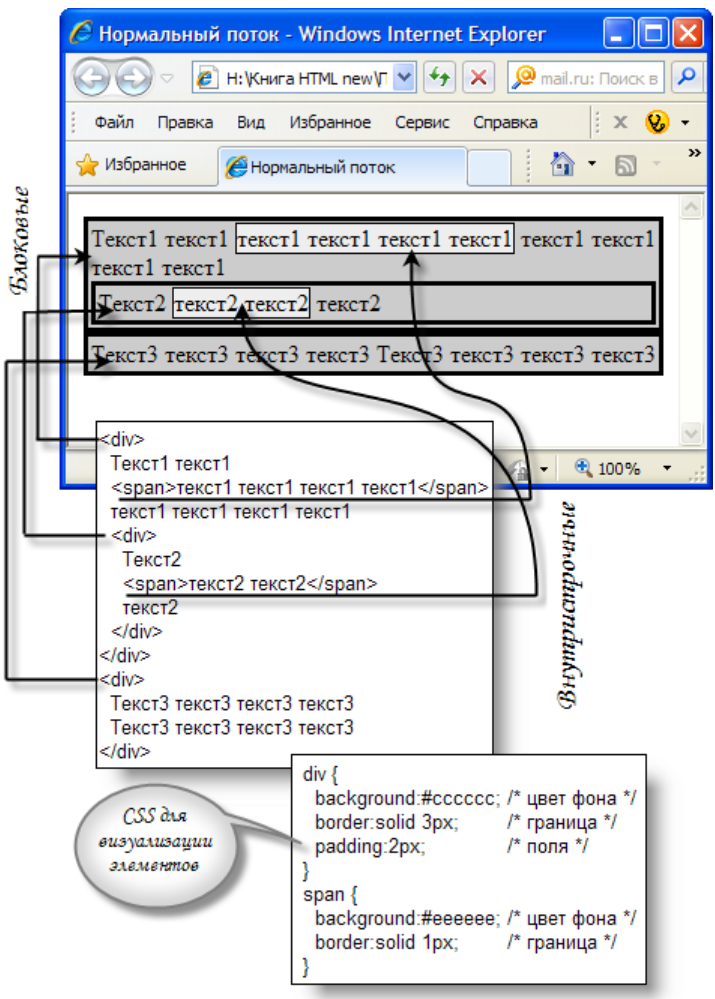


Рис. 4.1. Расположение блочных и внутристрочных контейнеров в нормальном потоке

Подчеркну еще раз, приведенная мной таблица стилей необходима только для создания наглядного изображения контейнеров в окне браузера (я просто не хотел вычерчивать прямоугольники в каком-либо графическом редакторе). Если же не применять данную таблицу стилей, то собственно прямоугольники (границы и заливка цветом) не будут отображаться, а видимым останется только конечное содержимое контейнеров (в данном случае — текст).

На рис. 4.2 показано расположение в нормальном потоке блочных элементов `<h1>`, `<p>` и внутристрочных `<img>`. Здесь, как и в предыдущем примере, для наглядности с помощью CSS были заданы цвет фона и границ элементов. Нетрудно заметить, что это имитация обычного иллюстрированного текста (статьи). Для публикации подобного документа ему прежде следует придать надлежащий вид: задать относительное расположение собственно текста и картинок, отступы и другие характеристики. Все это можно сделать с помощью CSS.

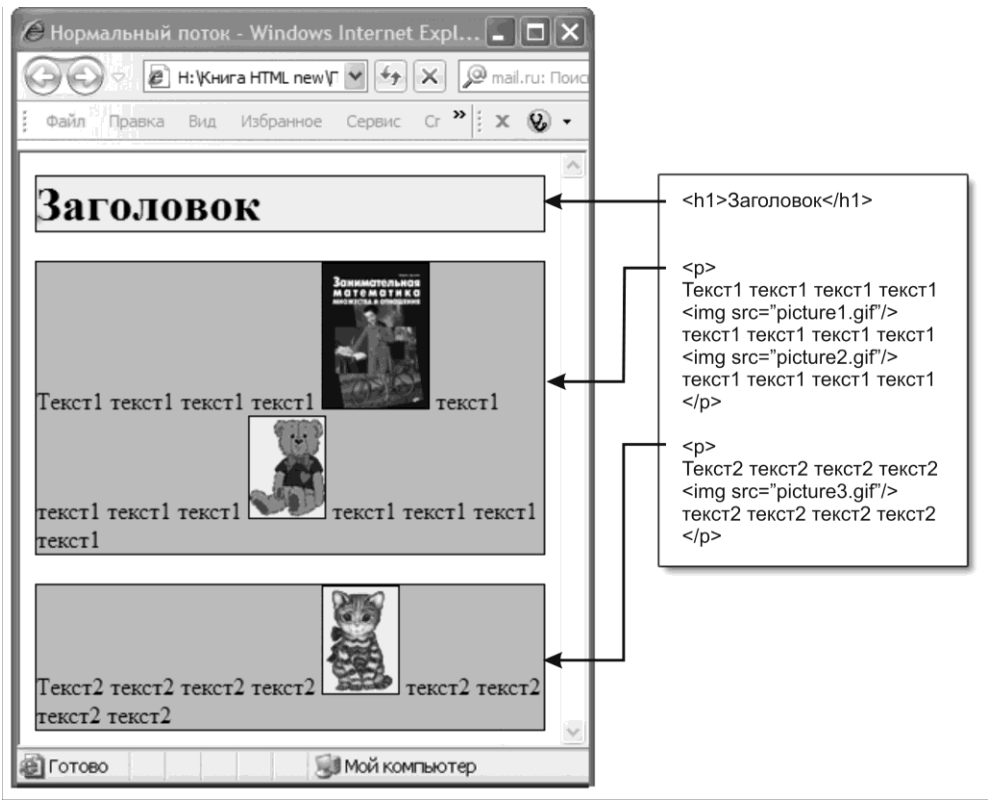


Рис. 4.2. Расположение элементов `<h1>`, `<p>` и `<img>` в нормальном потоке

Внимание!

Контейнерные теги могут содержать в себе другие теги, в том числе и контейнерные. Однако они не должны пересекаться. Например, следующая запись ошибочна:

```
<div><h1>Добро пожаловать</div></h1>
```

Правильно записать так:

```
<div><h1>Добро пожаловать</h1></div1>
```

Ошибки подобного рода в HTML-коде браузер попытается преодолеть, хотя и не гарантирует правильного отображения. Интерпретация же XHTML-кода с такими ошибками может быть приостановлена.

4.2. Позиционирование с помощью CSS

Понимание правил расположения элементов в нормальном (прямом) потоке, рассмотренных в предыдущем разделе, очень важно для создания требуемой компоновки Web-страницы. С помощью CSS задают параметры визуализации контейнеров и их содержимого (расположение, размеры и другие элементы оформления) вопреки той их визуализации, которая принята по умолчанию или, другими словами, которая реализуется в соответствии с моделью нормального потока.

Любой видимый элемент (X)HTML-документа с помощью CSS можно расположить (позиционировать) на Web-странице практически как угодно, вопреки его естественному положению, диктуемому нормальным потоком. Так что, CSS позволяет сверстать Web-страницу подобно странице какого-нибудь глянцевого журнала или газеты.

На рис. 4.3 показаны код и внешний вид HTML-документа, отображаемого в нормальном потоке. Здесь CSS использовалась только для того, чтобы окрасить фон и обозначить границы контейнеров `<div>` в нормальном потоке, не изменяя их естественного позиционирования. Как и положено, три контейнера `<div>` блокового типа следуют один ниже другого в естественном порядке, т. е. в порядке их упоминания в (X)HTML-коде.

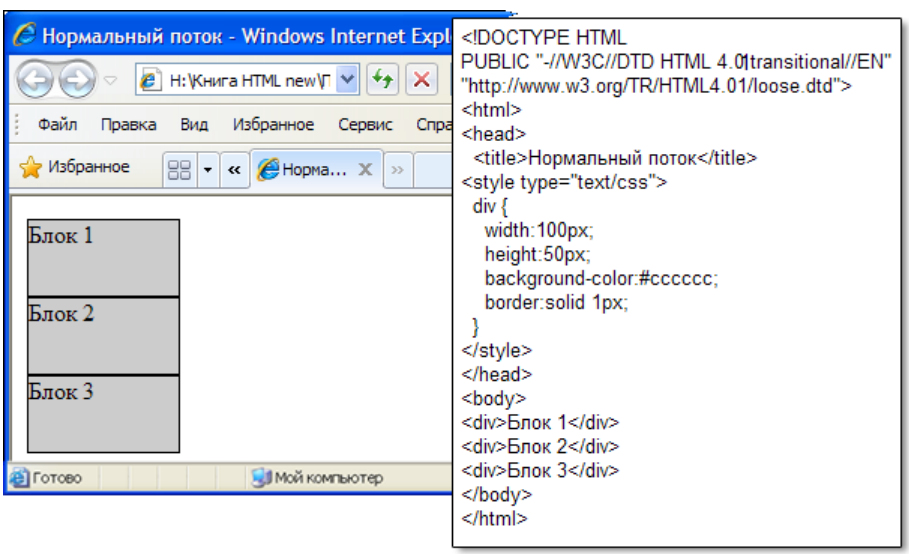


Рис. 4.3. Позиционирование трех контейнеров `<div>` в нормальном потоке

Посредством CSS можно задать параметры `top` и `left` — вертикальную и горизонтальную координаты верхнего левого угла элемента соответственно. Можно также задать параметры `bottom` и `right` — соответственно вертикальная и горизонтальная координаты для правого нижнего угла элемента. Но этого еще недостаточно. Необходимо преж-

де указать с помощью параметра `position` способ позиционирования. Этот параметр определяет, как следует понимать принудительное позиционирование элемента и располагать остальные элементы в нормальном потоке. Кроме того, от значения параметра `position` зависит начало отсчета координат элементов `top`, `left`, `bottom` и `right`.

Итак, чтобы позиционировать элемент в соответствии с заданными координатами, необходимо указать способ позиционирования и собственно требуемые координаты верхнего левого угла элемента.

Возможные значения параметра `position`:

- `static` (статический);
- `relative` (относительный);
- `absolute` (абсолютный);
- `fixed` (фиксированный).

Далее мы рассмотрим способы позиционирования более подробно.

4.2.1. *position:static*

Статическое позиционирование (`position:static`) аналогично расположению элемента в нормальном потоке, при этом координаты `top`, `left`, `bottom` и `right` не играют никакой роли. Поэтому в CSS данный способ практически никогда не задают с помощью параметра `position:static`.

4.2.2. *position:relative*

При относительном позиционировании (`position:relative`) элемент размещается по координатам, задаваемым параметрами `top` и `left` относительно своего положения в нормальном потоке. Иначе говоря, `top` и `left` — это смещения элемента относительно его естественного положения. Последующие элементы в нормальном потоке позиционируются так, будто данный элемент продолжает оставаться в нормальном потоке, т. е. занимать свое естественное положение. На рис. 4.4 приведен пример относительного позиционирования второго элемента `<div>`. Обратите внимание на пустое место между первым и третьим элементами `<div>`, освобожденное вторым элементом `<div>`, который смещен посредством параметров `top` и `left`. Третий `<div>`, будучи в нормальном потоке, позиционируется так, будто второй `<div>` остается в том же нормальном потоке.

Предположим, ваш документ отображается в нормальном потоке и требуется немного скорректировать положение лишь некоторых элементов, оставив другие на своих местах. Тогда лучше всего это сделать путем относительного позиционирования.

4.2.3. *position:absolute*

При абсолютном позиционировании (`position:absolute`) элемент располагается по координатам, задаваемым параметрами `top`, `left`, `bottom` и `right` относительно содержащего его контейнера. Достаточно указать какую-нибудь одну пару координат. Координаты `top` и `left` задают положение верхнего левого угла элемента относительно верхнего левого угла содержащего контейнера. Координаты `bottom` и `right` определяют положение нижнего правого угла элемента и отсчитываются от нижнего правого угла контейнера. Если задать все четыре координаты для контейнерного тега, то у прямоугольной области соот-

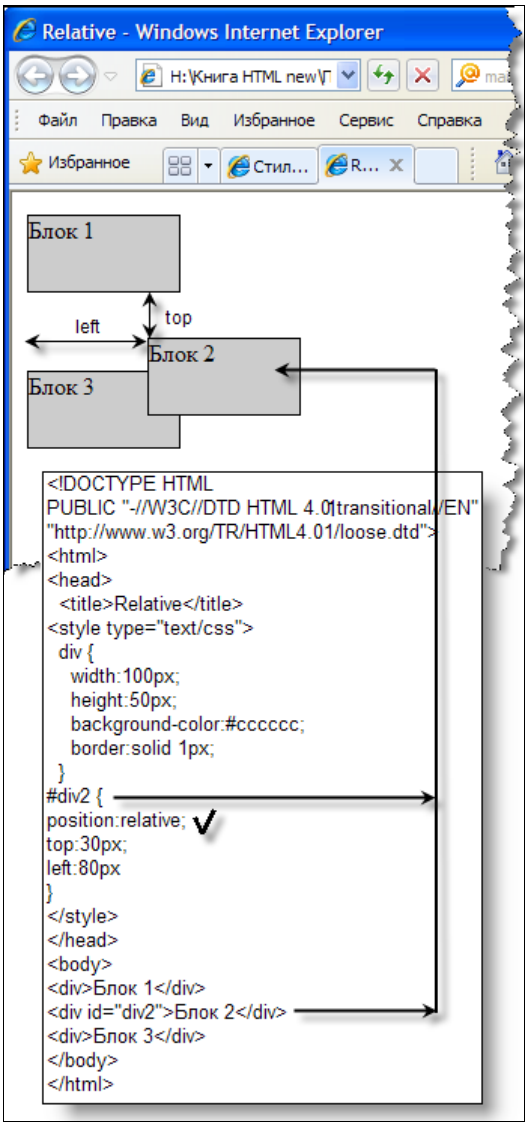


Рис. 4.4. Относительное позиционирование второго контейнера `<div>`

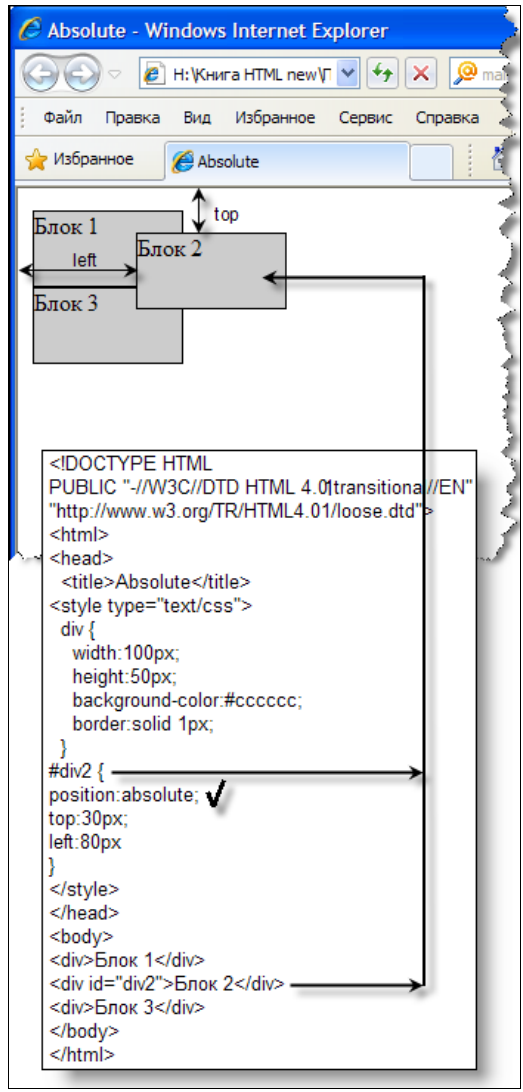


Рис. 4.5. Абсолютное позиционирование второго контейнера `<div>`

ветствующего элемента будут позиционированы одновременно два угла — верхний левый и нижний правый. Некоторые элементы (например, `<img>`) позиционируются посредством какой-нибудь одной пары координат. Если указаны две пары, то предпочтение отдается первой, т. е. `top` и `left`.

При абсолютном позиционировании последующие элементы в нормальном потоке располагаются так, будто данный элемент изъят из нормального потока. На рис. 4.5 приведен пример абсолютного позиционирования второго элемента `<div>`. В данном случае свойства `top` и `left` задают координаты элемента относительно верхнего левого угла документа (окна браузера). Обратите внимание на то, что абсолютно позиционируемый

элемент освобождает место, которое он занимал бы в нормальном потоке. Освободившееся место занимает последующий элемент из нормального потока.

Теперь рассмотрим подробнее, относительно чего абсолютно позиционируется элемент. В предыдущем примере (рис. 4.5) второй `<div>` абсолютно позиционировался относительно окна браузера. Если элемент находится непосредственно в контейнере `<body>`, то он позиционируется относительно верхнего левого угла документа (клиентской области окна браузера). Напомню, что если вы не упомянули в своем (X)HTML-коде тег `<body>`, то соответствующий ему контейнер все равно автоматически появится в структуре документа и, более того, он будет иметь некоторые параметры CSS, принятые по умолчанию данным браузером, в том числе и параметры позиционирования. В рассмотренном примере все теги `<div>` находятся в одном и том же контейнере `<body>`, а потому любой из них абсолютно позиционируется относительно левого верхнего угла документа.

А что будет, если элемент находится в другом контейнере, отличном от `<body>`? Результат зависит от того, как позиционирован контейнер, содержащий данный элемент.

На рис. 4.6 показаны четыре элемента `<div>`, расположенные в нормальном потоке, поскольку в CSS не задано их позиционирование. При этом четвертый `<div>` вложен во

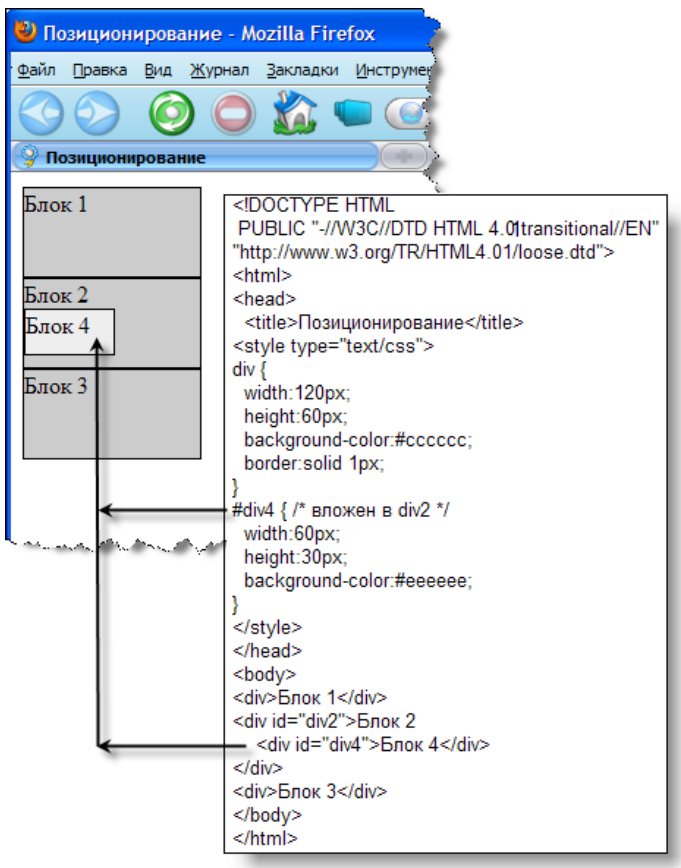


Рис. 4.6. Элементы `<div>` в нормальном потоке, при этом четвертый `<div>` вложен во второй

второй. Размеры данных элементов таковы, что вложенность соответствующих им блоков хорошо видна на рисунке.

На рис. 4.7 показаны два варианта того же HTML-документа, но с позиционированием второго и четвертого элементов . В обоих примерах четвертый позиционирован абсолютно (position:absolute), а содержащий его второй — по-разному.

Так, в первом варианте (сверху на рис. 4.7) второй имеет относительное позиционирование (position:relative). Поскольку при этом параметры top и left не заданы, то данный элемент остается на месте (смещения относительно его положения в нормальном потоке равны нулю). Однако четвертый позиционируется относительно верхнего левого угла содержащего его второго , т. е. своего контейнера, а не окна браузера.

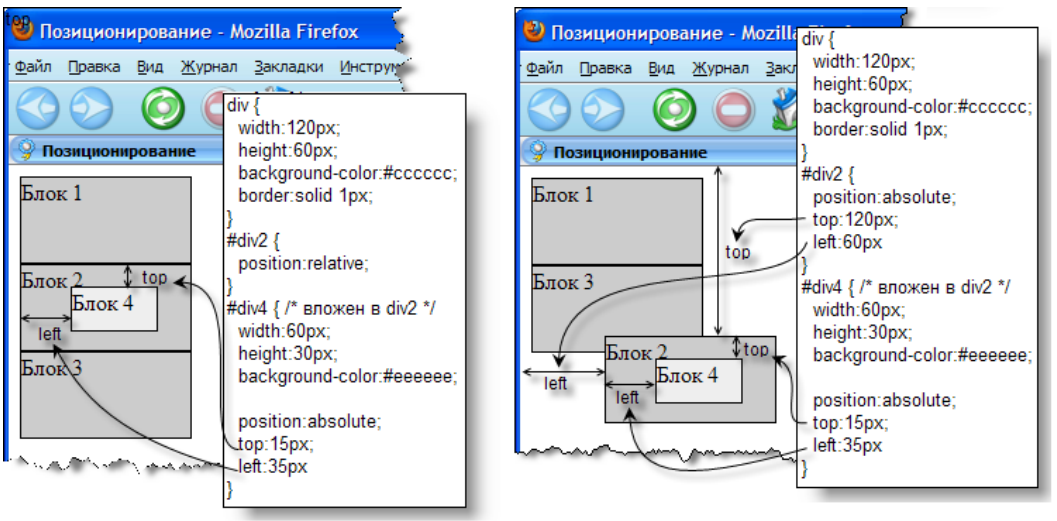


Рис. 4.7. Позиционирование элементов , при этом четвертый вложен во второй

Во втором варианте (снизу на рис. 4.7) второй имеет абсолютное позиционирование (position:absolute). Поскольку его ближайшим контейнером служит <body>, то он расположится относительно окна браузера, выйдя из нормального потока. Третий займет освободившееся место. При этом четвертый позиционируется относительно второго так же, как и в первом варианте.

Пусть родительский элемент не позиционирован относительно или абсолютно, а его дочерний элемент позиционирован. Тогда он будет размещен относительно ближайшего прародительского элемента. Если таковым является <body>, то элемент позиционируется относительно окна браузера.

А что произойдет, если второй позиционировать абсолютно, не задавая координаты явно? Он отобразится на том же месте, что и в нормальном потоке или при относительном позиционировании (position:relative), но при этом будет изъят из нормального потока, освободив свое место другому элементу нормального потока. В результате

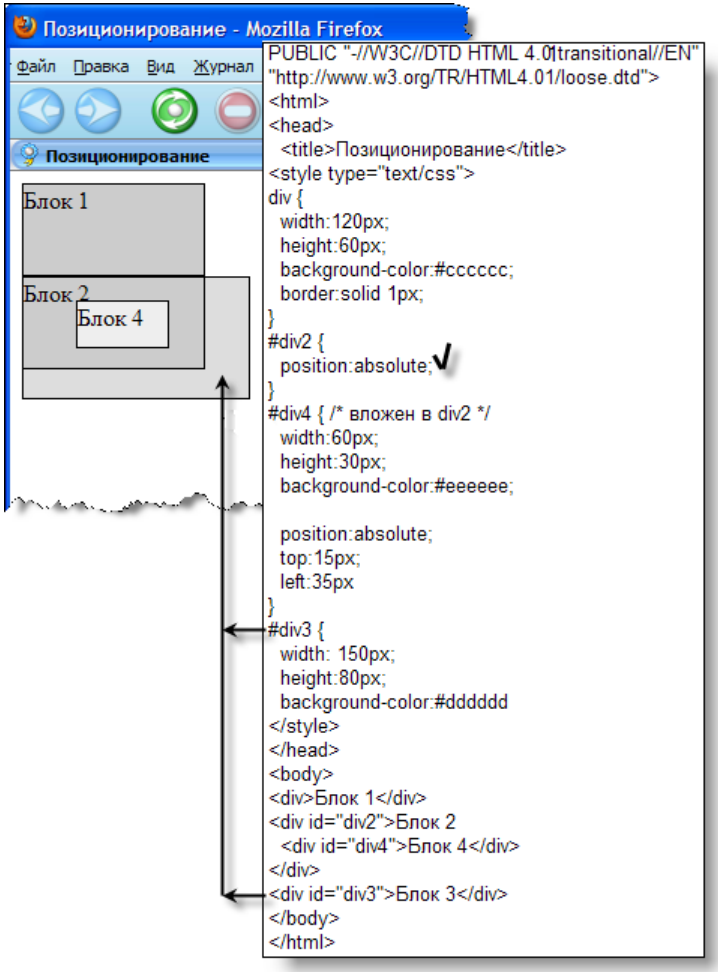


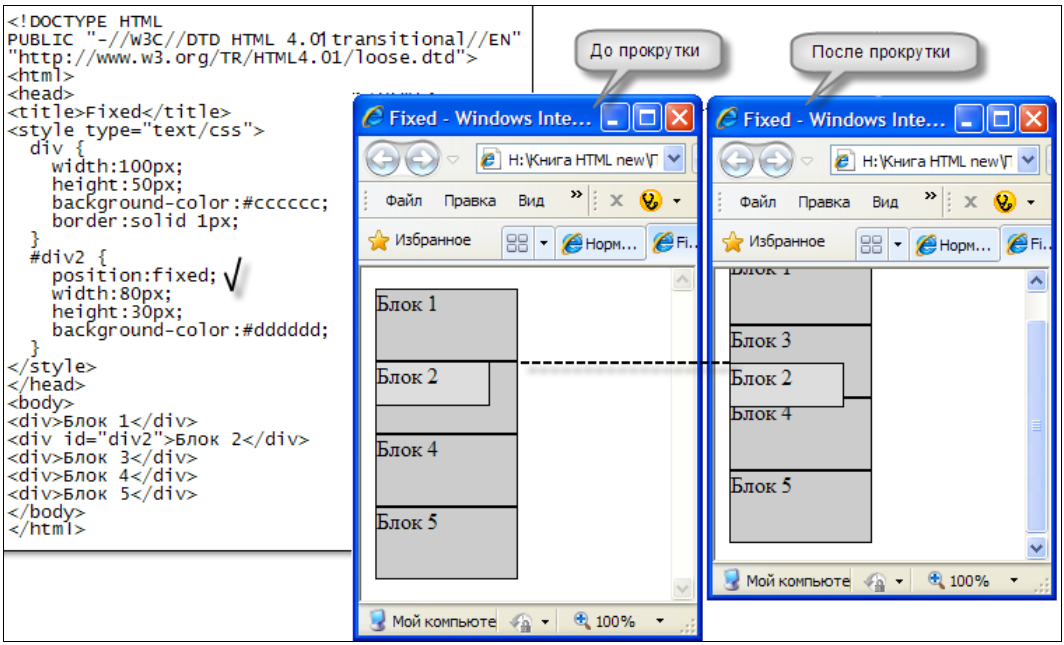
Рис. 4.8. Позиционирование элементов `<div>`, при этом третий `<div>` находится под вторым

может возникнуть перекрытие первого вторым. В примере, показанном на рис. 4.8, второй `<div>` оказался отображенным над третьим и частично закрыл его.

4.2.4. *position:fixed*

Фиксированное позиционирование (`position:fixed`) позволяет зафиксировать элемент так, чтобы он оставался видимым при прокрутке содержимого Web-страницы и занимал неизменное положение относительно окна браузера. При этом собственно позиционирование аналогично абсолютному, просто появляется дополнительный эффект. На рис. 4.9 второй элемент `<div>` имеет параметр `position:fixed`, слева внизу показана страница до ее вертикальной прокрутки, а справа — после. В обоих случаях положение второго `<div>` относительно окна браузера одинаково.

В исходном состоянии второй `<div>` имеет координаты, как если бы он находился в нормальном потоке, поскольку других ему в данном примере не назначено. Тем не




```
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Fixed</title>
<style type="text/css">
  div {
    width:100px;
    height:50px;
    background-color:#cccccc;
    border:solid 1px;
  }
  #div2 {
    position:fixed;
    width:80px;
    height:30px;
    background-color:#dddddd;
  }
</style>
</head>
<body>
<div>Блок 1</div>
<div id="div2">Блок 2</div>
<div>Блок 3</div>
<div>Блок 4</div>
<div>Блок 5</div>
</body>
</html>
```

Рис. 4.9. Фиксированное позиционирование

менее он изъят из нормального потока (как и в случае абсолютного позиционирования), а потому перекрывает следующий за ним третий `<div>`.

4.3. Отсчет координат

В предыдущих разделах уже упоминалось, относительно чего отсчитываются координаты `top` и `left` позиционируемого элемента. Здесь мы сделаем несколько уточняющих замечаний для случая абсолютного позиционирования (`position:absolute`) элементов.

Рассмотрим два элемента, из которых один непосредственно вложен в другой или, иначе говоря, один элемент дочерний по отношению к родительскому. Родительский элемент может быть в свою очередь дочерним по отношению к другому элементу и т. д. Родительский элемент наивысшего уровня — `<body>`. Так что любой элемент имеет, по крайней мере, одного родителя — элемент `<body>`. Итак, непосредственным родителем для элемента может быть `<body>` или какой-нибудь другой элемент. Однако эти две ситуации с точки зрения отсчета координат существенно различаются.

Если непосредственным родительским элементом является `<body>`, то его дочерний элемент абсолютно позиционируется относительно клиентской области окна браузера, т. е. значения параметров `top` и `left` отсчитываются соответственно от верхней и левой границ окна до верхнего и левого краев поля (`margin`) элемента.

Если непосредственный родительский элемент отличен от `<body>`, то его дочерний элемент абсолютно позиционируется относительно родительского: значения параметров `top` и `left` отсчитываются соответственно от внутренних краев границы (`border`) родительского элемента до верхнего и левого краев поля (`margin`) дочернего элемента (рис. 4.10).

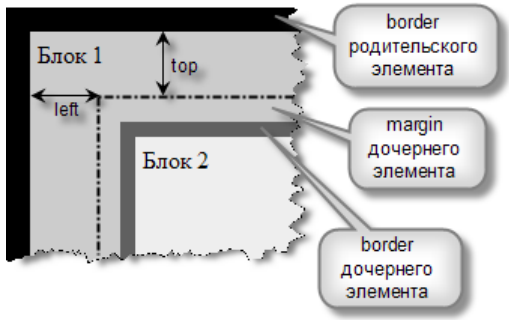


Рис. 4.10. Отсчет координат при абсолютном позиционировании, когда непосредственный родительский элемент отличен от `<body>`

4.4. Слои, или третья пространственная координата

Порядком отображения (над/под) элементов можно управлять с помощью параметра `z-index`, который принимает целочисленные значения. Он определяет третье измерение для Web-страницы, т. е. уровень слоя, в котором находится элемент. Чем больше значение этого параметра, тем выше (ближе к переднему плану) будет находиться элемент, и наоборот. По умолчанию все элементы находятся в одном слое: значение параметра `z-idex` для всех элементов равно нулю.

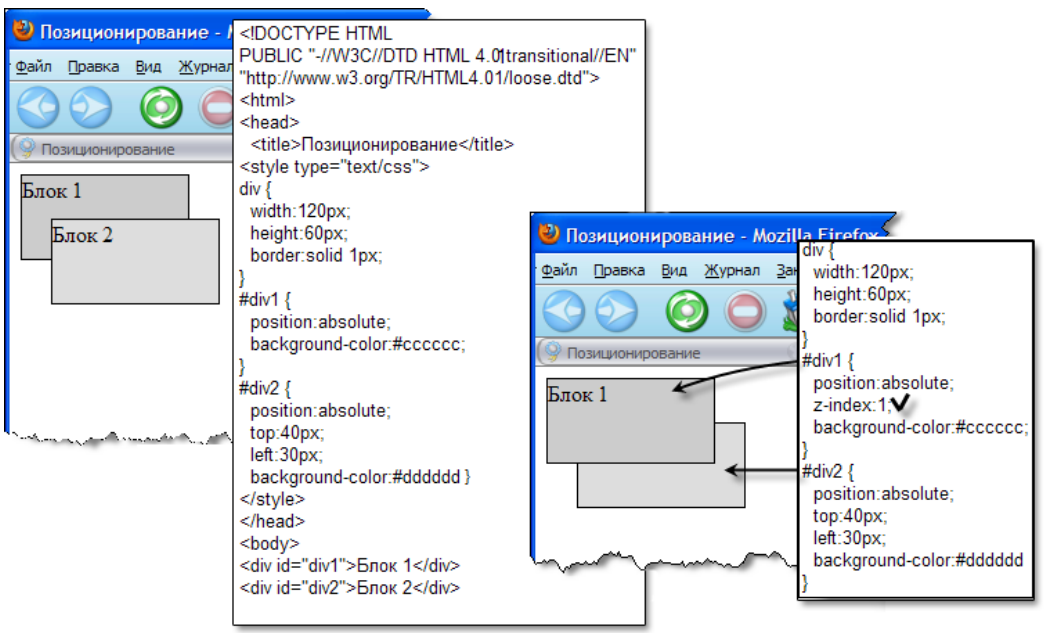


Рис. 4.11. Управление расположением элементов с помощью параметра `z-index`

Как и другие параметры, задающие координаты, z-index действует, только если указан способ позиционирования (position) со значениями relative, absolute или fixed.

На рис. 4.11 в качестве примера показано абсолютное позиционирование двух элементов . Координаты top и left выбраны так, что элементы перекрываются. Поскольку второй упомянут в HTML-коде последним, то он располагается над первым , как показано в верхней части рисунка. Однако этот порядок можно изменить, если установить достаточно большое значение параметра z-index для первого (нижняя часть рисунка). В рассматриваемом случае достаточно было задать z-index:1. Другой пример использования параметра z-index рассмотрен в *разд. 4.8.2*.

4.5. Обтекание (*float*)

Способы позиционирования, рассмотренные в предыдущих разделах, позволяют решить все задачи расположения элементов на Web-странице. Вместе с тем, имеются и другие возможности.

Положение элементов в нормальном потоке можно изменить и с помощью параметра float, который принимает значения left и right. Данный параметр сдвигает элемент по горизонтали, прижимая его к левому или правому краю контейнера. При этом последующие элементы в нормальном потоке располагаются справа или слева от сдвинутого элемента, если, конечно, позволяет освободившееся место. Таким образом, они как бы обтекают элемент с параметром float. На рис. 4.12 второй из пяти элементов сдвинут влево (float:left), а остальные его обтекают справа. Третий и четвертый элементы сдвинулись вверх и заняли освободившееся место справа от второго, а для пятого места справа не нашлось, и поэтому он остался ниже четвертого. Если бы не параметр float:left для второго , то все элементы в данном примере расположились бы друг за другом по вертикали.

Если в рассматриваемом примере для второго не указывать размеры (width и height), они будут заданы автоматически минимальными так, чтобы поместилось их содержимое. На рис. 4.13 показан соответствующий пример, в котором применяется обтекание как слева, так и справа.

Отменить ранее установленный режим обтекания элемента позволяет параметр clear, принимающий значения left, right и both (для отмены обтекания как слева, так и справа). На рис. 4.14 показан пример, в котором для третьего отменяется режим обтекания, установленный для предыдущего элемента (второго), и назначается режим обтекания справа. Сравните данный рисунок с рис. 4.13.

С помощью параметра float удобно управлять относительным расположением картинок и текста. На рис. 4.15 приведен пример расположения текста и графических изображений. В контейнере <p> находится некоторый текст и две картинки. Первое графическое изображение прижимается к левому краю окна (float:left) и обтекается текстом справа, а второе прижимается к правому краю окна (float:right) и обтекается текстом слева. Если бы не параметр float, то картинки располагались бы в нормальном потоке в зависимости от текущих размеров окна браузера, как показано на рис. 4.16. Очевидно, что такое расположение текста с картинками не годится для публикации иллюстрированной статьи.

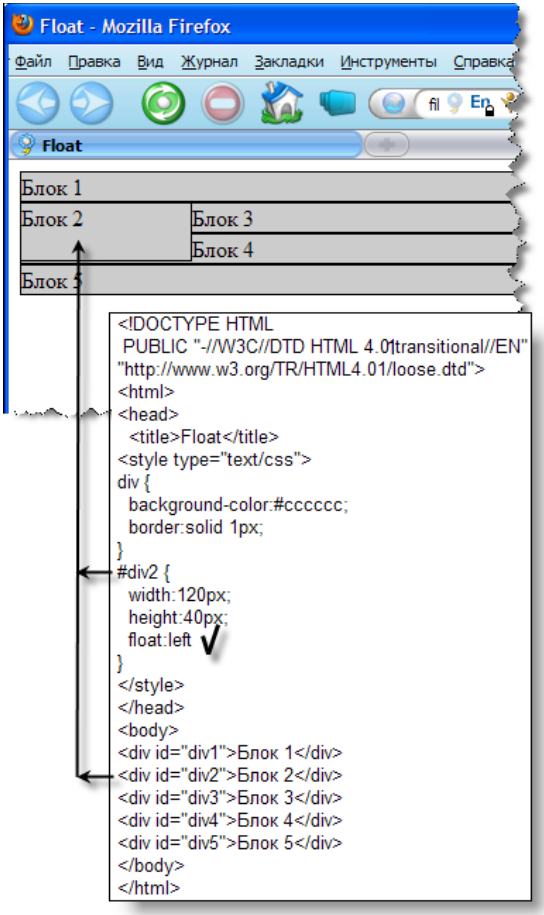


Рис. 4.12. Позиционирование с помощью параметра `float`

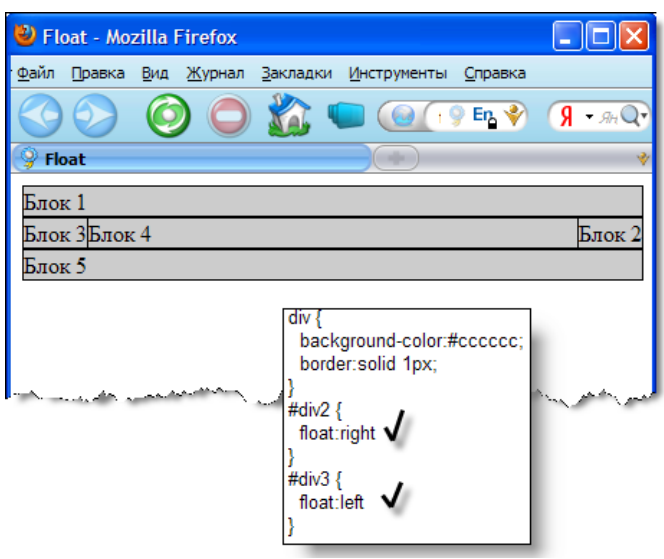


Рис. 4.13. Позиционирование с помощью параметров `float:left` и `float:right`

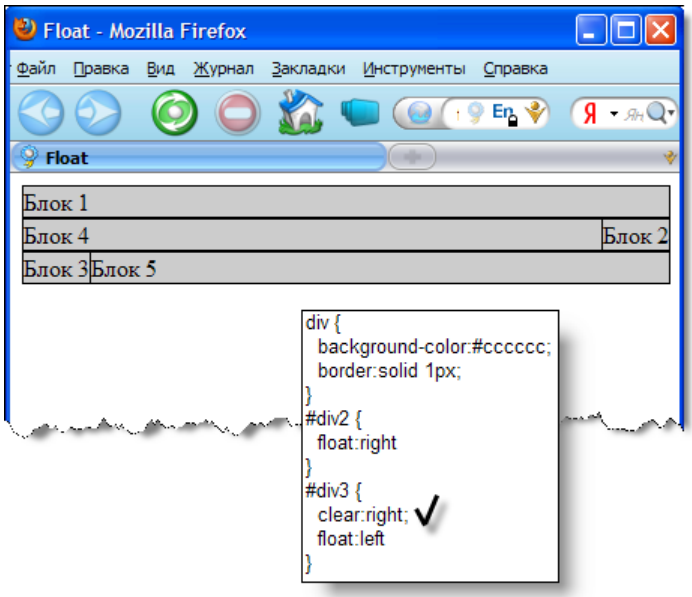
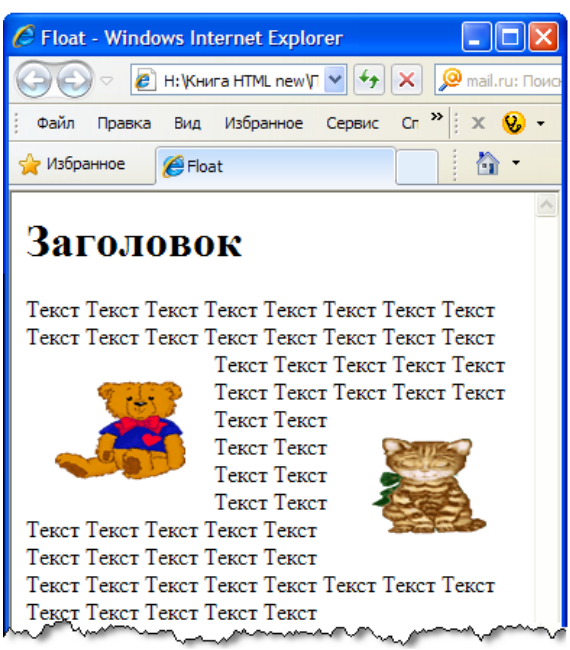


Рис. 4.14. Применение параметра `clear`




```
<html>
<head>
  <title>Float</title>
<style type="text/css">
img {
  width:90px;
  height:70px;
  margin:20px
}
</style>
</head>
<body>
<h1>Заголовок</h1>
<p >
Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст
<img src="мишка.gif" style="float:left"/>
Текст Текст Текст Текст Текст Текст Текст Текст
<img src="кошка.gif" style="float:right"/>
Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст
Текст Текст Текст Текст Текст Текст Текст
</p>
</body>
</html>
```

Рис. 4.15. Позиционирование текста и картинок с помощью параметра `float`

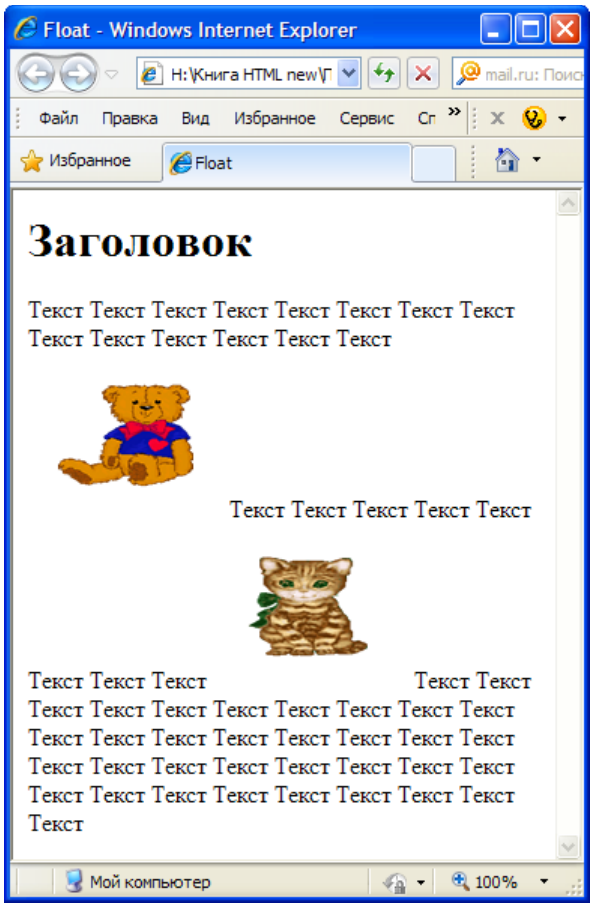


Рис. 4.16. Позиционирование текста и картинок в нормальном потоке, т. е. при отсутствии параметра `float`

4.6. Видимость

Видимостью элементов на странице можно управлять с помощью ряда параметров CSS, которые мы рассмотрим в данном разделе.

4.6.1. *overflow*

Может случиться, что содержимое контейнера полностью в нем не помещается, т. е. размеры контейнера меньше размеров его содержимого. Например, вы создали контейнер `<div>` с заданными размерами `width=500px` и `height=100px` и поместили в него фотографию размером 1280×1024 px, полученную посредством цифровой фотокамеры, даже не задав нужных размеров для ее публикации в Web. Эту ситуацию еще называют переполнением контейнера. Как отобразится содержимое переполненного контейнера? По умолчанию оно будет показано в полном своем объеме, как будто границ контейнера не существует (см. рис. 3.12). Тем не менее, в таких случаях отображением содержимого контейнера можно управлять с помощью параметра `overflow`, который принимает следующие значения:

- `visible` — если содержимое и выйдет за пределы контейнера, оно все равно будет показано полностью; это значение принято по умолчанию;
- `hidden` — выступающие за границы контейнера части содержимого будут обрезаны, т. е. окажутся невидимыми; это значение принято по умолчанию в Internet Explorer, но не в других браузерах;
- `auto` — добавляет полосы прокрутки, если содержимое выходит за границы элемента-контейнера;
- `scroll` — добавляет полосы прокрутки в любом случае.

4.6.2. *clip*

Параметр `clip` позволяет показать некоторую прямоугольную часть содержимого элемента (например, графического изображения), который абсолютно позиционирован (`position:absolute`). Данный параметр принимает значения:

- `auto` — элемент полностью виден (значение по умолчанию);
- `rect(top right bottom left)` — в скобках, разделяя пробелом, указывают верхнюю, правую, нижнюю и левую координаты, задающие прямоугольную область содержимого элемента, которую требуется показать.

На рис. 4.17 показано применение параметра `clip:rect(top right bottom left)` к графическому изображению. Обратите внимание, от чего отсчитываются координаты.

Рис. 4.17. Применение параметра `clip`

4.6.3. *visibility*

Иногда в документе требуется использовать элемент, но не показывать его. Например, можно загрузить в браузер таблицу с данными, которые будут как-то задействованы сценарием, но пользователь не должен их видеть. Элементы, по умолчанию видимые в окне

браузера, можно сделать невидимыми с помощью параметра `visibility`, который принимает следующие значения:

- `visible` — элемент виден (задано по умолчанию);
- `hidden` — элемент не виден (скрыт), однако занимает на странице место, т. е. он прозрачен, а остальные элементы нормального потока отображаются так, будто бы он виден;
- `inherit` — элемент отображается, если виден его родительский элемент;
- `auto` — добавляет полосы прокрутки, если содержимое выходит за пределы контейнера;
- `scroll` — добавляет полосы прокрутки в любом случае.

4.6.4. *display*

Управлять видимостью, а также изменять расположение элемента можно с помощью параметра `display`, принимающего следующие значения:

- `none` — элемент не виден, при этом он изымается из нормального потока (в отличие от параметра `visibility:hidden`), т. е. не занимает место на странице. Следовательно, последующие элементы документа в нормальном потоке отобразятся так, будто невидимого элемента нет в документе;
- `block` — элемент создает структурный блок, который визуализируется в нормальном потоке как блоковый элемент;
- `inline` — элемент создает внутристрочный блок.

> ***Примечание***
>
> Здесь перечислены основные, но далеко не все возможные значения параметра `display`.

В верхней части рис. 4.18 показаны пять блоковых элементов `<div>` и один внутристрочный элемент `<span>`, расположенные в нормальном потоке при отсутствии пара-

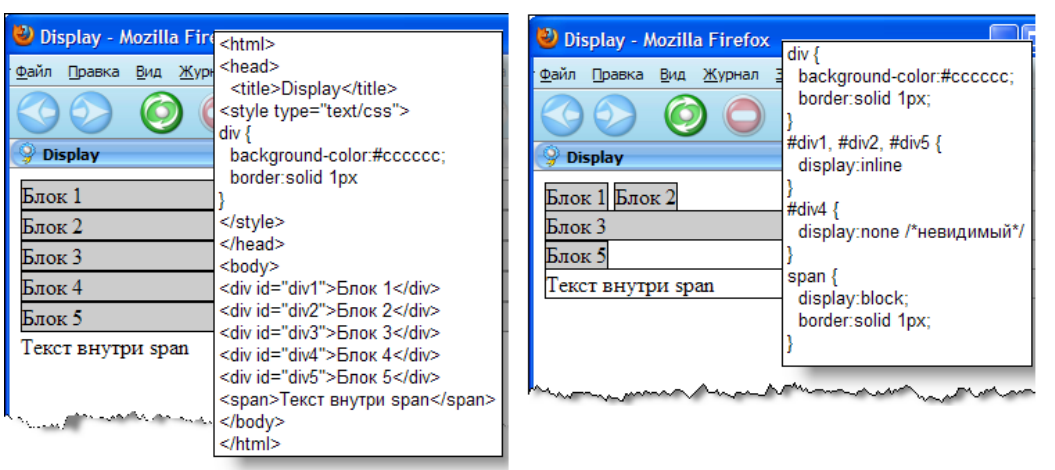


Рис. 4.18. Влияние параметра `display` на отображение блоковых и внутристрочных элементов

метра display, а в нижней части эти элементы расположены иначе, благодаря использованию данного параметра. Так, три блоковых элемента (div1, div2 и div5) превращаются во внутристрочные блоки, div4 становится невидимым, выпадая из потока, а внутристрочный элемент превращается в блоковый.

Таким образом, с помощью параметра display можно удалить из нормального потока элемент, а также превратить блоковый элемент во внутристрочный и наоборот. Данный параметр широко применяется при создании раскрывающихся меню. Его значением управляют посредством сценариев.

4.7. Размеры элементов

Размеры элементов задаются параметрами width (ширина) и height (высота) в линейных единицах или процентах (см. *разд. 3.5*). Данные параметры не включают в себя соответствующие величины полей, границ и отступов (см. рис. 3.4). Размеры в процентах задают относительно аналогичных величин родительского элемента.

Примеры:

```
div {width:300px; height:50%}
p {width:15cm; height:20mm}
```

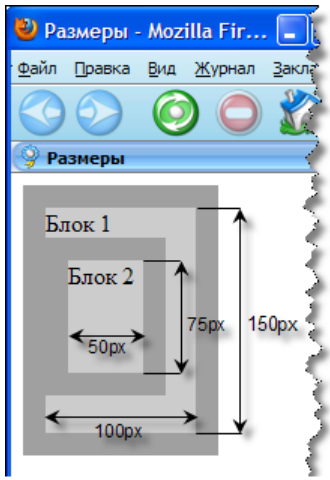
> ***Примечание***
>
> Указанное определение размеров элементов соответствует блочной модели от W3C, которую поддерживают все современные браузеры. Internet Explorer ранних версий (4, 5 и 5.5) и Opera 7 реализовали модель, в которой размеры включали в себя отступы и границы. В спецификации CSS 3 введен параметр box-sizing со значениями content-box для указания на использование стандартной модели W3C и border-box — модели Internet Explorer 5.

Если параметры width и height не указывать, то размеры элемента на Web-странице будут зависеть от его типа. Например, ширина блоковых элементов и <p> определяется так, чтобы они вместе с полями, границами и отступами заняли всю ширину клиентской области браузера, а высота определяется так, чтобы содержимое данных элементов поместилось целиком. Графическое изображение из файла, вставляемое в документ посредством внутристрочного тега <img>, будет иметь оригинальные размеры в пикселах. Элемент <object>, представляющий содержимое внешнего файла (например, видео- или аудиоклипа), имеет некоторые размеры в зависимости от используемого плагина.

Если размеры заданы в абсолютных единицах, то блоковые элементы (например, и <p>) теряют свое свойство изменять их в зависимости от текущих ширины окна браузера и объема содержимого. При указании относительных единиц (например, процентов) данное свойство сохраняется, если только родительский элемент имеет относительные размеры.

На рис. 4.19 в качестве примера показан HTML-документ, в котором второй элемент вложен в первый. При этом размеры первого заданы в пикселах, а второго — в процентах относительно размеров родительского элемента, т. е. первого .

Кроме параметров width и height, можно задавать экстремальные (т. е. минимальные и максимальные) размеры: min-width, min-height и max-width, max-height. Их действие легче понять в случае, когда соответствующие обычные размеры (width и height) не заданы.




```
<html>
<head>
  <title>Размеры</title>
<style type="text/css">
div { /* для всех div */
  width:100px;
  height:150px;
  background-color:#cccccc;
  border:solid 15px #a0a0a0;
  }
#div2 { /*для вложенного div */
  width:50%;
  height:50%;
}
</style>
</head>
<body>
<div>Блок 1
  <div id="div2">Блок 2</div>
</div>
</body>
</html>
```

Рис. 4.19. Установка размеров для блоковых элементов

Примечание

Браузер Internet Explorer 6.0 и 5.5 не воспринимает параметры `min-width`, `min-height`, `max-width` и `max-height`.

Рассмотрим сначала случай, когда для элемента `<div>` не указан параметр `width`, но задан `min-width`. Тогда ширина данного элемента должна быть не меньше значения параметра `min-width`. Какой именно она будет, зависит от текущей ширины клиентской области браузера. Больше она может стать, только если клиентская область браузера достаточно широка, чтобы вместить элемент. Если задан параметр `max-width`, то ширина элемента не может быть больше значения данного параметра. Меньше она становится, только если ширина клиентской области слишком мала, чтобы вместить элемент. Параметры `min-height` и `max-height` действуют аналогично.

Примечание

Различные браузеры позволяют изменять размеры окна браузера в разной степени, и может оказаться, что при достаточно малых установленных значениях `min-width` действительная минимальная ширина элемента будет несколько больше. Моменты появления полос прокрутки также могут различаться. Иначе говоря, в различных браузерах эффекты изменения размеров элемента при вариации размеров окна могут несколько отличаться.

Совместное применение `min-width` и `max-width` (`min-height` и `max-height`) позволяет регулировать ширину (высоту) элемента в заданных пределах при изменении размеров окна браузера. Это полезно при определении, например, области, занимаемой текстом. Такой текст можно заключить в тег `<div>` или/и `<p>` и сделать так, чтобы занимаемая им область могла расширяться или сужаться в зависимости от размеров окна браузера, но в заданных пределах.

Более сложный случай возникает тогда, когда заданы как обычные, так и экстремальные размеры, причем одного и того же типа (ширина или длина). Например, параметры `width` и `min-width` относятся к одному типу — ширине, а `width` и `min-height` — к разным. Эффект совместного применения данных параметров при изменении размеров окна браузера зависит от соотношения значений параметров и типа единиц их измерения

(абсолютные и относительные). Не останавливаясь на изучении данного случая, отмечу лишь, что его нетрудно исследовать экспериментальным путем.

4.8. Практические примеры

Примеры, рассмотренные в предыдущих разделах, служили в основном для изучения параметров CSS. Теперь перейдем к решению некоторых простых, но практически важных задач, таких как центрирование элемента (важно, в частности, при компоновке страниц), управление его положением и размерами (важно, в частности, при создании раскрывающихся меню).

4.8.1. Центрирование элемента

При разработке Web-страниц нередко требуется расположить элемент фиксированных размеров в центре клиентской области окна браузера или же в центре какого-нибудь элемента-контейнера. При изменении размеров окна браузера элемент должен сохранять свои заданные размеры, но центрироваться по горизонтали и/или вертикали. Для этой цели в HTML предусмотрен контейнерный тег `<center>`. Все элементы в нем центрируются по ширине окна браузера или содержащего контейнера. Однако сейчас его не рекомендуется использовать, и для центрирования предлагается применять надлежащие параметры CSS. Рассмотрим два варианта решения этой задачи на примере центрирования элемента `<div>`.

Первый вариант основан на применении параметра `margin`, определяющего ширину прозрачных полей, со значением `auto`. На рис. 4.20 показан пример, когда элемент `<div>` с фиксированными размерами центрируется по горизонтали относительно окна браузера, а на рис. 4.21 — относительно содержащего его другого элемента `<div>`. Обратите внимание, эффект центрирования происходит благодаря параметрам `margin-left:auto` и

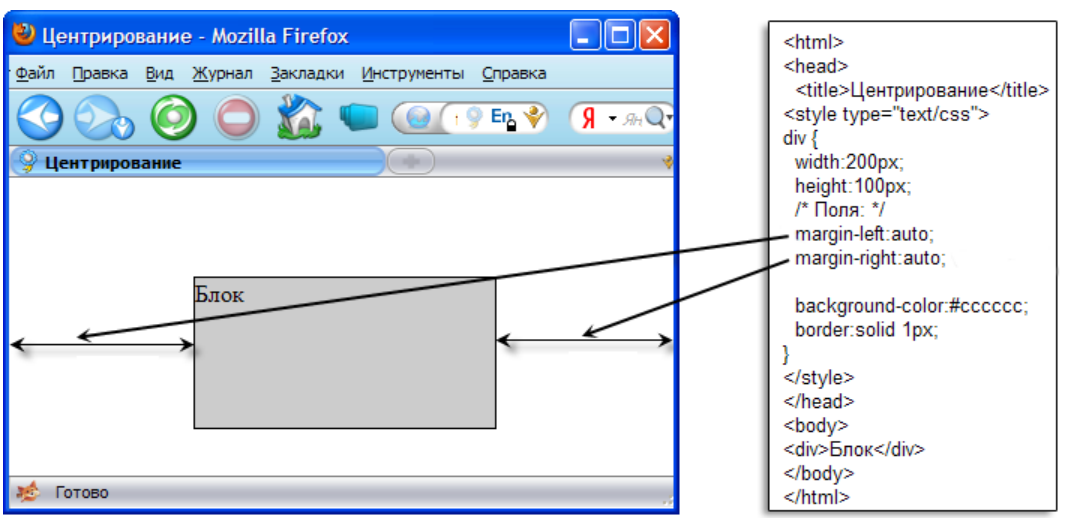
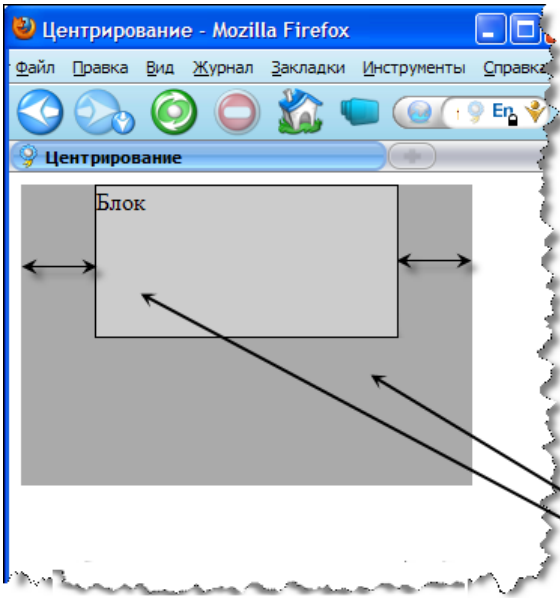
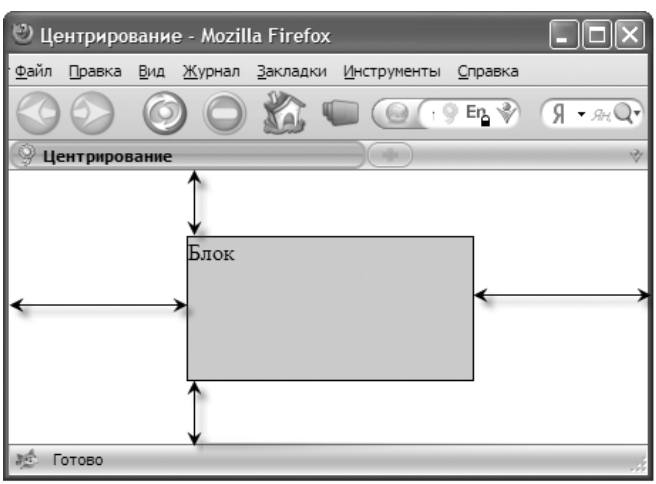


Рис. 4.20. Центрирование элемента `<div>` по горизонтали относительно окна браузера




```
<html>
<head>
  <title>Центрирование</title>
<style type="text/css">
div { /* внешний */
  width:300px;
  height:200px;
  background-color:#aaaaaa;
}
#inbox{ /* внутренний */
  width:200px;
  height:100px;
  /* Поля: */
  margin-left:auto;
  margin-right:auto;
  background-color:#cccccc;
  border:solid 1px;
}
</style>
</head>
<body>
<div>
<div id="inbox">Блок</div>
</div>
</body>
</html>
```

Рис. 4.21. Центрирование элемента `<div>` по горизонтали относительно содержащего его другого элемента `<div>`




```
<html>
<head>
  <title>Центрирование</title>
<style type="text/css">
div {
  width:200px;
  height:100px;
  position:absolute;
  left:50%;
  top:50%;
  margin-top:-50px;  /*половина height*/
  margin-left:-100px;/*половина width*/
  background-color:#cccccc;
  border:solid 1px;
}
</style>
</head>
<body>
<div>Блок</div>
</body>
</html>
```

Рис. 4.22. Центрирование элемента `<div>` относительно окна браузера одновременно по горизонтали и по вертикали

`margin-right:auto` при условии, что параметры `width` и `height` для центрируемого элемента заданы. Однако применение параметров `margin-top:auto` и `margin-bottom:auto` не приведет к центрированию по вертикали.

Второй вариант позволяет центрировать элемент как по горизонтали, так и по вертикали. Как известно, положение элемента при абсолютном позиционировании (`position:`

absolute) определяется координатами left и top, отсчитываемыми от края окна браузера или внутреннего края границы контейнера до внешнего края прозрачного поля (margin) элемента (см. *разд. 4.3*). Если ширина поля отлична от нуля, то видимое содержимое элемента располагается по координатам left + margin-left и top + margin-top. Однако ширина поля может быть задана и отрицательным числом. Если задать left:50%, а margin-left указать равной половине ширины элемента со знаком минус, то последний будет центрироваться по горизонтали. Аналогично, если задать top:50%, а margin-top указать равной минус половине высоты, элемент будет центрироваться по вертикали. Однако при определенном соотношении размеров окна браузера и элемента края последнего могут оказаться за границами окна. На рис. 4.22 показан пример центрирования элемента относительно окна одновременно и по горизонтали, и по вертикали.

4.8.2. Управление положением элемента с помощью мыши

Динамически изменять положение элемента и другие его параметры CSS при наведении указателя мыши можно посредством псевдокласса :hover (см. *разд. 3.3.3*).

На рис. 4.23 показан пример, в котором изображения игральных карт абсолютно позиционированы так, чтобы они частично перекрывали друг друга, располагаясь в "стопке". Это исходное положение карт. При наведении указателя мыши на какую-либо карту последняя принимает параметры CSS, указанные в правиле с селектором img:hover. В данном случае это правило задает достаточно большое значение параметра z-index, определяющее номер слоя (см. *разд. 4.4*). В результате изображение будет выведено на

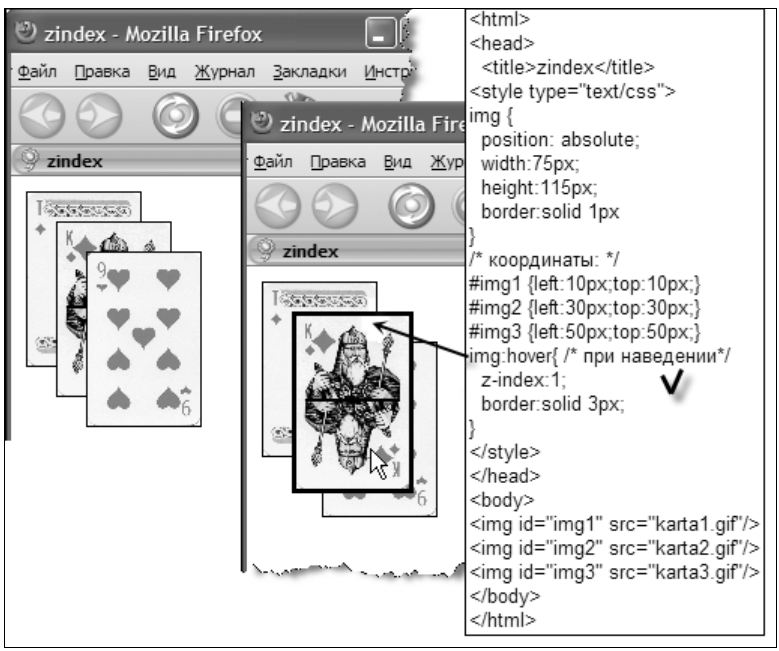


Рис. 4.23. Выведение картинки на передний план при наведении на нее указателя мыши

передний план, поверх остальных. Кроме изменения слоя в данном примере меняется ширина границы карты, на которой находится указатель мыши. При желании можно в правиле `img:hover` установить новые значения для ширины и высоты самой карты. С уходом указателя мыши данная карта принимает свои исходные параметры.

Теперь рассмотрим случай, когда требуется переместить картинку, на которую наведен указатель мыши. Казалось бы, для этого достаточно использовать правило `img:hover {top:значение; left:значение}`. Однако такое решение, как ни странно, не приведет к цели. Чтобы перемещение все-таки было возможно, необходимо назначить псевдокласс `:hover` индивидуально для каждой картинки, т. е. `#img1:hover{...}`, `#img1:hover{...}` и т. д. Однако при этом возникает нестабильная ситуация: при наведении указателя мыши картинка перемещается на новое место, а указатель остается на прежнем и при малейшем движении указателя картинка возвращается в исходное состояние. Поэтому для перемещения элементов лучше использовать соответствующий скрипт, а не псевдокласс `:hover`.

4.8.3. Раскрывающаяся панель

Раскрывающаяся панель — это прямоугольник, наведение указателя мыши на который приводит к увеличению его размеров и, как следствие, к видимости некоторого содержимого, например текста, ссылок и т. п. Уход указателя за границы данного прямоугольника приводит к сворачиванию последнего до исходных размеров. Очевидно, раскрывающееся меню является вариантом раскрывающейся панели или совокупности таких панелей.

Создать такую панель можно на основе элемента `<div>`, используя правило CSS `div:hover`, устанавливающее ее параметры при наведении указателя мыши. При этом необходимо решить вопрос о том, как раскрытие панели повлияет на расположение других элементов документа. Возможно, что они должны потесниться, уступая место увеличившемуся элементу, а возможно, все они останутся в исходном положении, но тогда раскрытая панель должна расположиться на переднем плане (в верхнем слое). Первый вариант характерен для раскрывающихся списков, а второй — для всплывающих подсказок и раскрывающихся меню. Остановимся на втором варианте, т. е. на панели, которая при раскрытии не сдвигает остальные элементы, а занимает верхний слой.

На рис. 4.24 приведен пример раскрывающейся панели с некоторым текстом. Для расположения панели на переднем плане применено абсолютное позиционирование с достаточно большим значением параметра `z-index`. В исходном состоянии размеры панели невелики (`width:150px`; `height:20px`) и выходящее за ее пределы содержимое не видно (`overflow:hidden`). При наведении на панель указателя мыши действует правило, устанавливающее новое значение ее высоты (`height:100px`). Вообще говоря, значение `height` подбирают экспериментально так, чтобы все содержимое панели в ее раскрытом состоянии было видно. Впрочем, при необходимости можно добавить полосы прокрутки, для чего в правиле с селектором `div.pane:hover` достаточно добавить параметр `overflow:scroll` или `overflow:auto`.

Развитие данной темы применительно к созданию раскрывающихся меню можно найти в *разд. 7.1*.

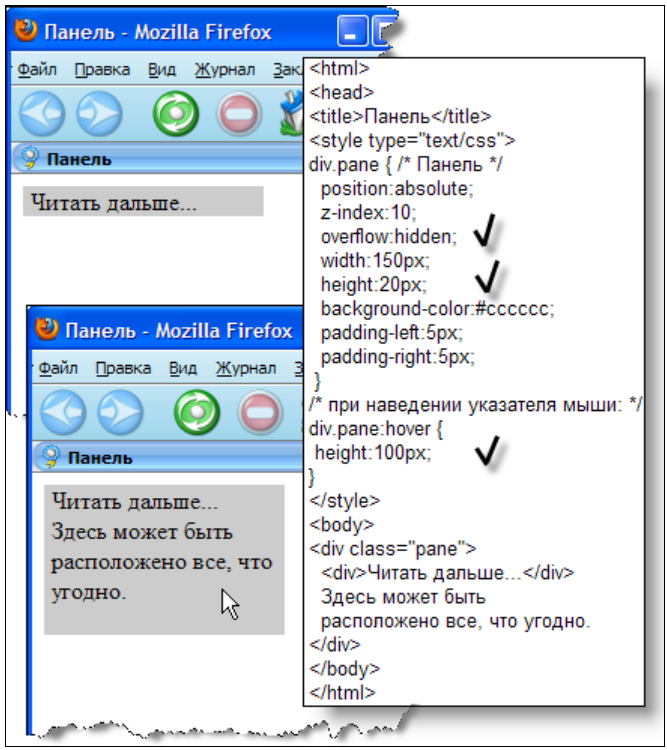


Рис. 4.24. Раскрывающаяся панель

ГЛАВА 5

Фон элементов и границ

В предыдущих главах уже применялся параметр "раскрашивания" элементов страницы — background. Здесь мы рассмотрим его, а также другие параметры CSS более подробно.

Фон страницы в целом, а также ее фрагментов задают, указывая цвет или графическое изображение из внешнего файла. Для этого существует целый ряд параметров CSS. Группа параметров `background` позволяет задать фон элементов, а `border` — границы и фон (см. *разд. 3.7*). Параметр `opacity` задает уровень прозрачности элемента, а `box-shadow` — эффект тени от него.

5.1. *background*

Параметры для задания фона элементов:

- `background-color` — цвет фона; принимает значения:
  - *`значение цвета`* (см. *разд. 3.6*);
  - `transparent` (полностью прозрачный).

  Примеры:
  - `body {background-color: #808080}`
  - `p {background-color: yellow}`
  - `div {background-color: rgb(255,128,0)}`
- `background-image` — графическое изображение в качестве фона; принимает значения:
  - `url("`*`URL файла изображения`*`");`
  - `none` (изображение отсутствует).

  Если заданы цвет фона и изображение, то последнее отображается сверху цветового фона, если оно доступно. Фоновое графическое изображение применяется в своих исходных (оригинальных) размерах и не масштабируется. Если оно не помещается в контейнер, для которого служит фоном, то соответствующим образом усекается. В этом случае по умолчанию отображается верхняя левая часть картинки, но можно указать и иную ее область с помощью параметра `background-position` (*см. далее*).

Пример:

```
body {background-image: url("/images/mypicture.jpg")}
```

- `background-repeat` — определяет, будет ли фоновое изображение дублироваться, чтобы покрыть всю страницу или элемент; принимает значения:
- `repeat` — дублируется по горизонтали и вертикали (принято по умолчанию);
- `repeat-x` — дублируется по горизонтали;
- `repeat-y` — дублируется по вертикали;
- `no-repeat` — не дублируется (используется единственная копия).

Пример:

```
body {
  background-image: url("/images/mypicture.jpg");
  background-repeat:repeat-y
}
```

- `background-attachment` — определяет способ фиксации фонового изображения; принимает значения:
- `fixed` — фоновое изображение фиксируется относительно окна просмотра;
- `scroll` — фоновое изображение фиксируется относительно документа и перемещается вместе с ним при прокрутке;
- `background-position` — определяет начальное положение фонового изображения с помощью двух значений (горизонтальной и вертикальной координат), разделенных пробелом. Значения задаются в абсолютных величинах длины (например, в пикселах), процентах или посредством ключевых слов `left`, `top`, `right`, `bottom`, `center`. Значения по умолчанию: `left`, `top`. Если указаны координаты *x*, *y*, то точка изображения, смещенная относительно его левого верхнего угла на *x* вправо по горизонтали и на *y* вниз по вертикали, помещается в точку, смещенную вдоль области отступов, на *x* вправо по горизонтали и на *y* вниз по вертикали. В частности, пара координат `0% 0%` означает выравнивание левого верхнего угла изображения относительно левого верхнего угла краевой линии отступов, а `100% 100%` — выравнивание правого нижнего угла изображения относительно правого нижнего угла краевой линии отступов. Если указана лишь одна координата, то считается, что обе координаты равны.

Пример:

```
body { background-image: url("/images/mypicture.jpg");
       background-position: 50% 25%
}
```

- `background` — позволяет задать значения сразу нескольких свойств, перечисленных ранее (так называемая сокращенная запись). Значения всех или только некоторых частных свойств (`background-color`, `background-image` и др.) указывают через пробел в произвольном порядке.

Примеры:

```
body {background: blue url("/images/mypicture.jpg") repeat-y}
body {background: url("/images/mypicture.jpg") no-repeat}
```

В листинге 5.1 приведен код для демонстрации применения параметров фона для <body> и двух , а на рис. 5.1 — вид соответствующей страницы в окне браузера. Здесь фон страницы (<body>) полностью "замощен" копиями графического изображения некоторой затейливой линии. Два блока внутри <body> окрашены синим цветом и, кроме того, заполнены изображением автомобиля, причем по-разному. В первом блоке фоновая картинка дублируется по вертикали и, поскольку параметр background-position для нее явно не задан, ее первая копия выравнивается по верхнему левому углу контейнера. Во втором блоке фоновая картинка позиционируется не в верхнем левом углу и, кроме того, ее первая копия видна лишь частично. Это объясняется выбором значений параметра background-position:25% 70%.

Листинг 5.1. Применение параметров фона

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Фон</title>
<style type="text/css">
     body {
          margin:0;
          padding-left:10px;
          padding-right:10px;
           color:white;
        /* картинка в качестве фона всего документа: */
          background: url("line.jpg") repeat
     }
     div { /* общие параметры для всех div */
           position:absolute;
          height:150px;
          width:250px;
          padding:10px;
          text-align:right; /* выравнивание по правому краю */
      /* синий фон с картинкой: */
           background: blue url("auto.gif")
     }
     #div1 {
          top:30px;
          left:30px;
          /* дублирование фона по вертикали: */
          background-repeat:repeat-y
     }
     #div2 {
          position:absolute;
          top:220px;
          left:30px;
          /* дублирование фона по горизонтали и задание начальной позиции: */
          background-repeat:repeat-x;
          background-position:25% 70%
     }
```

```
</style>
</head>
<body>
<div id="div1">
     Это элемент div1
</div>
<div id="div2">
     Это элемент div2
</div>
</body>
</html>
```

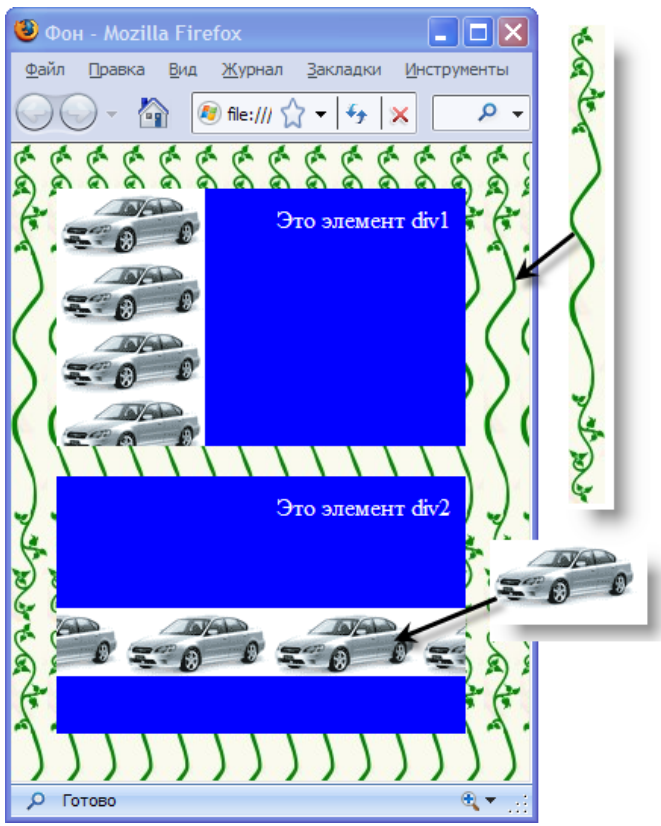


Рис. 5.1. Вид в окне браузера страницы, где применен код листинга 5.1

Нередко в качестве фона страницы выбирают так называемую градиентную заливку: цвет постепенно переходит от более темных к светлым оттенкам, или наоборот. Для достижения данного эффекта в графическом редакторе (например, Adobe Photoshop или GIMP) создают длинную и узкую (шириной от одного до нескольких пикселов) полоску с градиентной заливкой и сохраняют ее в файле. Полосу необходимо сделать как можно у́же, чтобы объем файла с ее изображением был минимальным. Затем эту полоску дублируют по вертикали или горизонтали. Бордюры создают аналогично дублированием небольшого изображения в одном из направлений.

Другой способ создания градиентной заливки — применить параметр `border-image`, который позволяет на основе небольшого графического изображения определить не только заполнение границ элемента, но и его центральной части (см. *разд. 3.7*).

Графическое изображение в качестве фонового можно поместить в единственном экземпляре (`background-repeat:no-repeat`) в любом месте страницы (`background-position`). Нередко графический дизайн всей страницы выполняют в каком-либо графическом редакторе, а затем полученное изображение используют в качестве фонового. Однако при этом не следует забывать, что большому графическому изображению, покрывающему в качестве фона всю страницу, соответствует немалый объем файла. Как известно, объем файла и время его загрузки в браузер находятся в прямой зависимости.

5.2. Прозрачность

В CSS 3 для задания прозрачности предлагается применять параметр `opacity` со значениями от нуля (прозрачный) до единицы (непрозрачный) и значение `rgba(r,g,b,a)` для параметра `background` или `background-color`. В отличие от обычного значения вида `rgb(r, g, b)` значение вида `rgba(r,g,b,a)` позволяет кроме уровней *r*, *g*, *b* яркости красной, зеленой и синей составляющих цвета задать еще и уровень прозрачности *a* (так называемый альфа-канал) со значениями от нуля (прозрачный) до единицы (непрозрачный).

Параметр `opacity` устанавливает уровень прозрачности всего элемента, к которому применяется (например, графического изображения, элемента `<div>` и др.), а значение вида `rgba(r, g, b, a)` устанавливает лишь прозрачность цвета фона.

Браузер Internet Explorer версии 8 и более ранних не поддерживает задание прозрачности в соответствии с CSS 3 и просто игнорирует параметр `opacity` и значение вида `rgba(r, g, b, a)`. Чтобы задать уровень прозрачности для Internet Explorer указанных версий, необходим графический фильтр — фирменное расширение css, которое не поддерживают другие браузеры. Задать фильтр можно двумя способами:

- `filter:alpha(opacity=`*число*`);`
- `filter:progid:DXImageTransform.Microsoft.Alpha(opacity=`*число*`).`

Здесь *число* от 0 до 100 (а не до 1) — уровень прозрачности. Фильтр `alpha` имеет и другие параметры, которые в данной книге не рассматриваются.

В листинге 5.2 в качестве примера приведен код документа с тремя графическими изображениями, которые частично перекрываются. Для всех картинок установлен уровень прозрачности 0,5. Данный код будет работать во всех современных браузерах. В Firefox, Opera, Safari и Chrome сработает параметр `opacity:05`, а в Internet Explorer — `filter:alpha(opacity=50)`. Обратите внимание, что часть HTML-кода с параметрами CSS для Internet Explorer выделена специальным образом. Другие браузеры проигнорируют ее. На рис. 5.2 показан вид в браузере документа с тремя изображениями игральных карт, когда прозрачность равна 1 и когда она равна 0,5.

Листинг 5.2. Установка уровня прозрачности

```
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
```

```
<html>
<head>
<title>Прозрачность</title>
<style type="text/css">
  img {
   position: absolute;
   width:150px;
   height:225px;
   border:solid 1px;
   opacity:0.5 /* прозрачность */
  }
/* координаты */
  #img1 {left:10px;top:10px;}
  #img2 {left:30px;top:30px;}
  #img3 {left:50px;top:50px;}
</style>
<!--[if IE]>
 <style type="text/css">
    img {filter:alpha(opacity=50)} /* для IE */
 </style>
<![endif]-->
</head>
<body>
<img id="img1" src="karta1.gif"/>
<img id="img2" src="karta2.gif"/>
<img id="img3" src="karta3.gif"/>
</body>
</html>
```

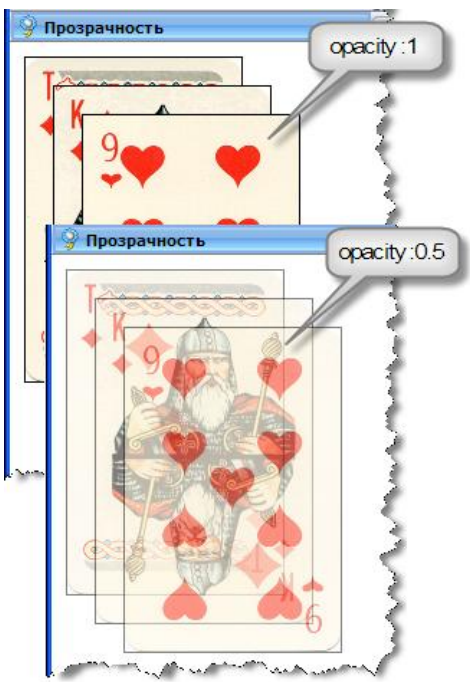


Рис. 5.2. Отображение в браузере документа листинга 5.2

На рис. 5.3 показан пример, в котором элемент `<div>` с параметром прозрачности 0,6 черного цвета расположен поверх графического изображения. Если цвет фона для `<div>` не указать, то эффект полупрозрачности не получится.

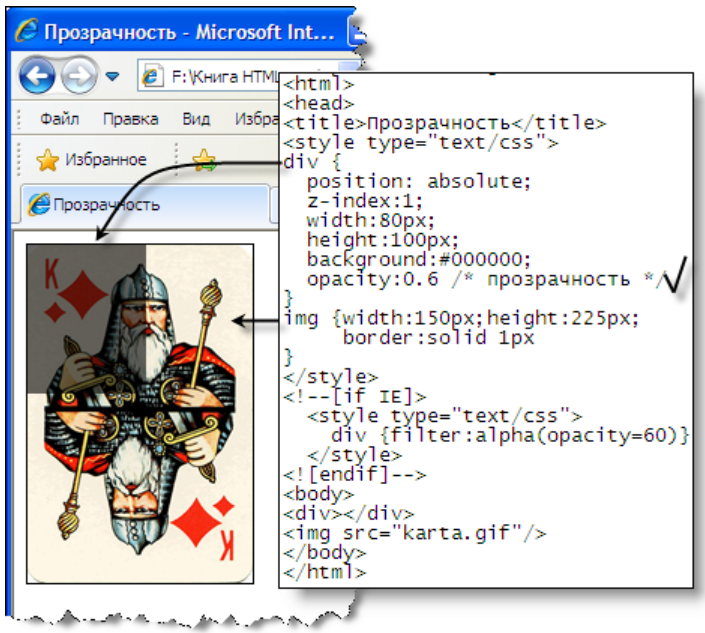


Рис. 5.3. Полупрозрачный элемент `<div>` расположен поверх картинки

Если контейнер `<div>` содержит другие элементы (например, картинки, текст и т. д.) и для него задан уровень прозрачности, то последний будет действовать на все содержимое данного контейнера. С другой стороны, полупрозрачные элементы внутри контейнера останутся полупрозрачными, если контейнер сделать непрозрачным.

Одна из типичных задач применения свойства прозрачности — создание полупрозрачной тени, отбрасываемой элементом, например, `<div>` или `<img>`. В листинге 5.3 приведен пример, в котором панель отображается с тенью на фоне картинки (рис. 5.4). Такую панель создают посредством трех элементов `<div>`. Первый из них — контейнер, включающий в себя остальные два, из которых один представляет лицевую сторону (face) панели, а другой — полупрозрачный прямоугольник, изображающий тень (shadow) от панели. Лицевая сторона и ее тень — равновеликие прямоугольники, но "теневой" прямоугольник сдвинут относительно лицевого вправо и вниз (будто бы источник света находится в верхнем левом углу) благодаря положительному значению координаты `left`, отрицательному значению координаты `top` и параметру `position:relative`. В данном примере ширина тени равна 10 px, хотя в типичных дизайнерских решениях ее задают в пределах 2—5 px. Общий контейнер (`<div class="pane">`) позиционирован абсолютно (`position:absolute`), и его расположение на экране задают параметры `left` и `top`. При этом элементы его содержимого (`<div class="face">` и `<div class="shadow">`) позиционируются на экране, сохраняя заданное положение друг относительно друга. Обратите внимание, что лицевая сторона панели имеет большее значение параметра

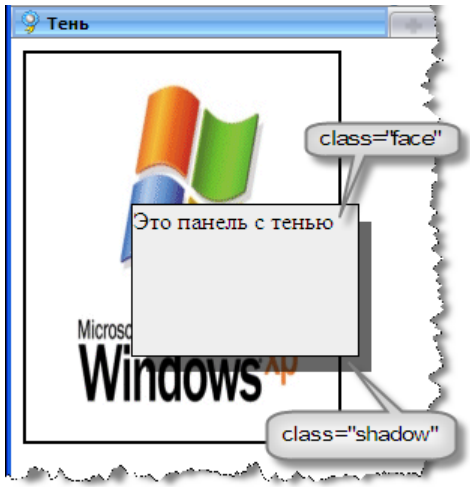


Рис. 5.4. Отображение в браузере документа листинга 5.3

z-index, чем теневая. Общему контейнеру ( ) назначены нулевые размеры.

Листинг 5.3. Панель с тенью

```
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Тень</title>
<style type="text/css">
pane{ /* панель */
   position:absolute;
   top:110px;
   left:80px;
   width:0;
   height:0
}
face { /* лицевая сторона */
  position: relative;
  z-index:1;
  width:150px;
  height:100px;
  background:#eeeeee;
  border:solid 1px;
}
.shadow { /* тень */
  position: relative;
  z-index:0;
  left:10px;/* смещение вправо */
  top:-90px; /* смещение вверх */
```

```
  width:150px;
  height:100px;
  background:#000000;
  opacity:0.5 /* прозрачность */
}
</style>
<!--[if IE]>
  <style type="text/css">
    .shadow {filter:alpha(opacity=50)}
  </style>
<![endif]-->
</head>
<body>
<img src="winlogo.jpg" style="border:solid 2px"/>
<div class="pane">
  <div class="face">Это панель с тенью</div>
  <div class="shadow"></div>
</div>
</body>
</html>
```

Для создания эффекта тени в CSS 3 имеется специальный параметр `box-shadow` , который мы рассмотрим в следующем разделе.

5.3. Тень

В *разд. 5.2* рассматривалось применение параметра `opacity` для создания эффекта тени. Такого же эффекта можно добиться с помощью параметра CSS 3 `box-shadow`, поддерживаемого браузерами Firefox, Opera, Safari, Chrome и Internet Explorer 9.

Значением `box-shadow` является строка параметров тени:

- `inset` — если указано, то тень будет направлена внутрь элемента;
- *x* — смещение тени по горизонтали;
- *y* — смещение тени по вертикали;
- *r1* — радиус размытия тени (не обязательный);
- *r2* — растяжение тени (не обязательный);
- *цвет* — значение цвета тени (не обязательный).

Смещения и радиусы задаются числами с указанием размерности. Например,

```
box-shadow: 5px 5px 10px #808080;
```

Параметр `inset` пока воспринимается различными браузерами по-разному. В настоящее время для применения `box-shadow` в Safari 5.0 и Chrome 10.0 рекомендуется к имени параметра добавлять специфический префикс. В следующем примере определение тени подойдет для всех перечисленных браузеров:

```
shadow { box-shadow: 5px 5px 10px black; /* стандартное и для Firefox, Opera */
  -webkit-box-shadow: 5px 5px 10px black; /* для Safari и Chrome */
}
```

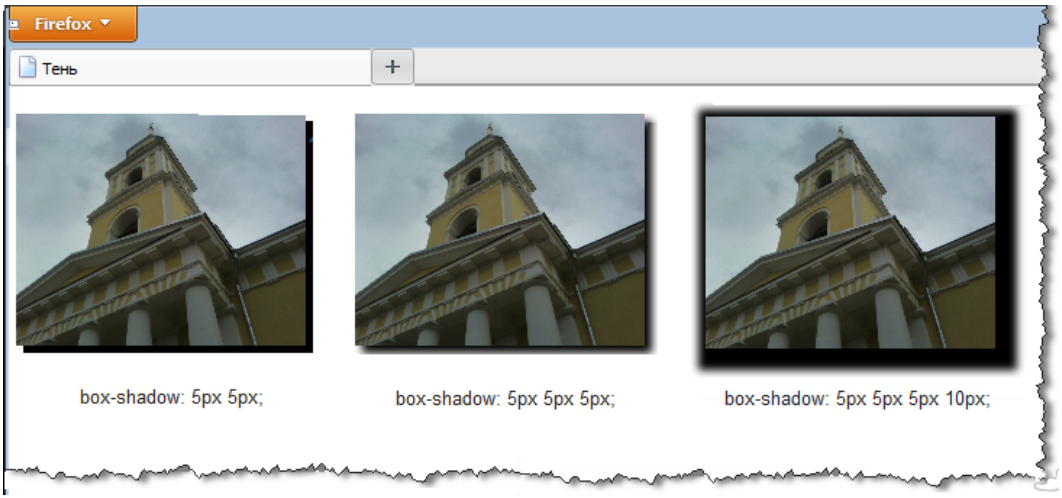


Рис. 5.5. Примеры использования параметра `box-shadow`

На рис. 5.5 показаны несколько примеров применения параметра `box-shadow` с различными значениями к графическим изображениям (элементам `<img>`).

Для Internet Explorer 8 эффект тени можно создать с помощью фирменного расширения css — графических фильтров `dropShadow` и `shadow`. Фильтр `dropShadow` имеет параметры `offX` и `offY`, определяющие смещение тени в пикселах в горизонтальном и вертикальном направлениях соответственно. Параметр `positive` определяет, от каких пикселов следует отбрасывать тень: от непрозрачных (`true`) или прозрачных (`false`). Параметр `color` определяет цвет тени. Этот фильтр действует на графические изображения и дает видимые результаты, если они содержат прозрачные участки. Графические изображения с прозрачными областями можно сохранять, например, в gif-файлах. Вот пример:

```
filter: dropShadow(offX=4, offY=4, color=#808080)
```

Фильтр `dropShadow` можно применять и к текстам.

Фильтр `shadow` имеет параметры `direction` (направление) и `color` (цвет). Параметр `direction` принимает значения 0 и кратные 45 — величина в градусах, отсчитанная от вертикали по часовой стрелке. Вот пример:

```
filter: shadow(direction=225, color:black)
```

ГЛАВА 6

Компоновка страницы

Сведений, изложенных в предыдущих главах, достаточно, чтобы реализовать компоновку Web-страницы. Компоновка материала на странице может быть различной, но типовых схем все же довольно мало (см. рис. 1.1). Обычно страницу разбивают на несколько прямоугольных областей. Например, в одной области располагают название страницы или сайта в целом, в другой — панель навигации (меню ссылок), в третьей — информационное содержимое (content) публикуемого документа. Впрочем, для информационной части, как и для ссылок, может быть отведено и несколько полос (прямоугольных областей). Как бы то ни было, сначала необходимо создать каркас страницы — систему из нескольких прямоугольных ячеек. Затем, при необходимости, в имеющиеся ячейки можно вставить другие.

6.1. Базовые схемы компоновки страницы

Как уже отмечалось, схем компоновки страниц может быть несколько. Любую из них можно реализовать посредством контейнерных блочных тегов `<div>`.

> ***Примечание***
>
> Несколько лет тому назад компоновку страниц часто выполняли, как правило, с помощью табличных тегов (`<table>`, `<tr>`, `<td>` и др.). Это делали не только вручную, но и посредством систем визуальной разработки (FrontPage, Dreamweaver и др.). Иногда применяли фреймы (`<frameset>`, `<frame>`). Теперь же считается приличным вместо табличных тегов применять для компоновки теги `<div>`. А в HTML 5 для этой же цели предусмотрены еще и специальные теги, такие как `<article>` (статья), `<header>` (заголовок), `<nav>` (навигационная панель), `<aside>` (боковая панель) и др. Фреймы не будут включаться в спецификацию HTML 5. В данной книге мы рассмотрим применение для компоновки лишь `<div>` и `<iframe>`.

Сначала рассмотрим самую простую схему, состоящую из двух ячеек, расположенных одна ниже другой. В верхней ячейке обычно располагают название сайта, оформительскую графику, логотип, а также горизонтальное меню ссылок на другие важнейшие страницы (разделы) сайта. Разумеется, содержимое верхней ячейки может быть и иным. Во второй ячейке отображается основное содержимое страницы.

Прежде всего, создадим посредством тегов `<div>` логическую структуру (каркас) страницы. Чтобы увидеть ее в браузере, зададим посредством CSS видимые границы эле-

ментов . Данный прием аналогичен разметке ткани мелом или мылом, которую используют портные. По умолчанию границы не видны и имеют нулевую ширину, т. е. не занимают места. По окончании разработки границы можно убрать.

В листинге 6.1 приведен HTML-код двухъячеечной структуры страницы. В секции <style> находятся несколько параметров CSS. Вид данного кода в браузере показан на рис. 6.1. Разумеется, значения атрибутов id (идентификаторов) тегов могут быть и иными, но уникальными. Значения атрибутов id используются в качестве селекторов в наборах стилевых параметров. В тегах <meta> для ключевых слов и краткого описания значения атрибутов content пока пусты, поскольку сейчас мы создаем схему страницы без содержимого.

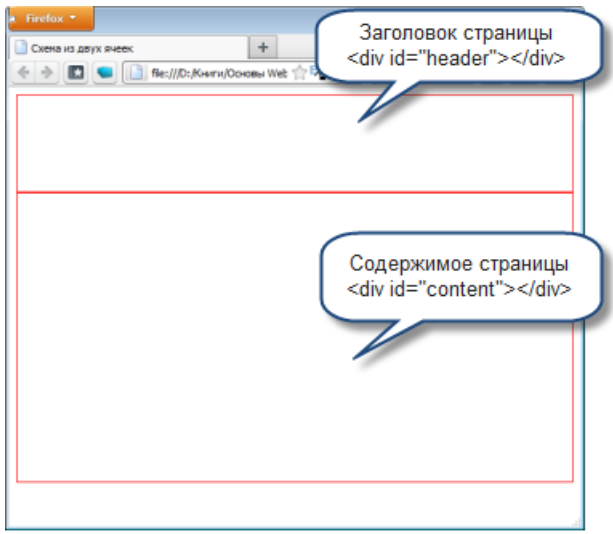


Рис. 6.1. Схема страницы из двух ячеек (листинг 6.1)

Листинг 6.1. Схема из двух ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Схема из двух ячеек</title>
<meta http-equiv="Content-Type"
      content="text/html;
      charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
 div {border:solid 1px red} /* для видимости в браузере*/
 #header {  /* заголовок */
           height:100px;
           overflow:hidden;
 }
```

```
 #content { /* основное содержимое */
           height:300px;
           overflow:auto;
 }
</style>
</head>
<body>
  <div id="header">
  </div>
  <div id="content">
  </div>
</body>
</html>
```

Так, параметр `border` для всех элементов (блоков) `<div>` задает границы, чтобы визуализировать структуру страницы. Иногда удобно для каждого блока назначать свой цвет границы. Это — технологический параметр, актуальный на первых этапах разработки, потом его следует удалить, если, конечно, видимые границы ячеек не предусмотрены в вашей дизайнерской концепции.

Для блоков `<div id="header">` и `<div id="content">` заданы высоты (`height`), значения которых впоследствии могут быть уточнены. В рассматриваемом примере высота ячейки для содержимого страницы выбрана чересчур малой (300 px), чтобы снимок экрана не был слишком велик. В действительности высоту следует выбирать, исходя из объема содержимого, но не более удвоенной высоты экрана. Сейчас типовая высота экрана составляет 800—1024 px. Если параметр `height` вообще не указывать, то высота блока `<div>` автоматически станет такой, чтобы в нем полностью поместилось все содержимое. В нашем примере содержимое блока `<div id="content">` не задано, поэтому отсутствие для него параметра `height` означает нулевую высоту.

У блока `<div id="content">` должны автоматически появляться полосы прокрутки при переполнении содержимым, а блок `<div id="header">` должен обрезать непомещающееся в нем содержимое. С этой целью задаются соответствующие значения параметра `overflow` (см. *разд. 4.6.1*).

> ***Совет***
>
> Попробуйте поэкспериментировать с рассматриваемыми в данной главе схемами компоновки, изменяя параметры CSS и вставляя дополнительные ячейки. В результате вы получите хороший опыт верстки страниц.

Более сложные схемы компоновки страницы получаются из простой путем добавления ячеек. Добавляемые ячейки можно вставлять как непосредственно в `<body>`, так и в уже существующие блоки `<div>`. Далее мы рассмотрим, каким образом можно модифицировать одну схему, чтобы получить другую. Обратите внимание на соответствующие изменения HTML-кода и CSS.

На рис. 6.2 показана четырехъячеечная схема компоновки страницы, а ее реализация — в листинге 6.2. Для боковой панели `<div id="pane">` заданы ширина (`width`) и обтекание (`float:left`), чтобы блок `<div id="content">` расположился справа от нее. Здесь можно разместить ссылки на разделы сайта. Подвал (нижняя панель) `<div id="footer">` реали-

зуется подобно заголовку страницы, в нем обычно размещают данные о владельце и разработчике сайта, счетчики посещений страницы и т. п. Впрочем, подвала у страницы может и не быть.

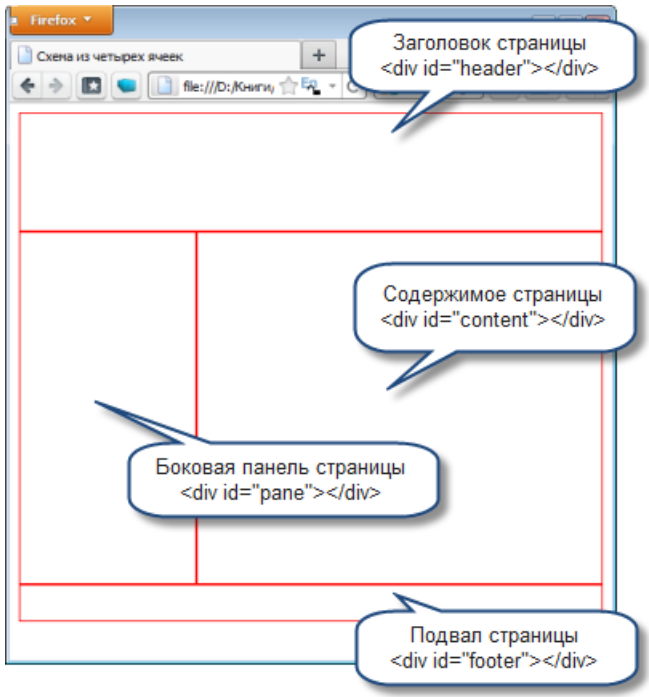


Рис. 6.2. Схема страницы из четырех ячеек (листинг 6.2)

Листинг 6.2. Схема из четырех ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Схема из четырех ячеек</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
 div {border:solid 1px red} /* для видимости в браузере*/
 #header {  /* заголовок */
           height:100px;
           overflow:hidden;
 }
 #content { /* основное содержимое */
           height:300px;
```

```
            overflow:auto;
  }
  #pane { /* боковая (левая) панель */
     height:300px;
     width:150px;
     float:left; /* обтекание */
     overflow:auto;
  }
  #footer { /* подвал */
     height:30px;
     overflow:hidden;
  }
</style>
</head>
<body>
   <div id="header">
   </div>
   <div id="pane">
   </div>
   <div id="content">
   </div>
   <div id="footer">
   </div>
</body>
</html>
```

Главные страницы многих сайтов содержат не одну, а две боковых панели, расположенные слева и справа от блока основного содержимого. Казалось бы, такая компоновка легко получается добавлением в четырехъячеечную схему после элемента `<div id="content">` еще одного блока `<div>` для правой боковой панели. Однако при этом возникает проблема обеспечения стабильности относительного положения всех ячеек при изменении размеров окна браузера. Я не буду показывать различные варианты разрушения схемы компоновки при тех или иных размерах окна того или иного браузера, а приведу лишь один из возможных стабильных вариантов пятиячеечной схемы (листинг 6.3, рис. 6.3).

Листинг 6.3. Схема из пяти ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Схема из пяти ячеек</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
```

```
 div {border:solid 1px red} /* для видимости в браузере*/
 #page {  /* страница (контейнер)*/
     min-width:500px; /* для резиновой схемы */
 }
 #header {  /* заголовок */
     height:100px;
     overflow:hidden;
 }
 #pane { /* панель левая*/
    height:300px;
    width:150px;
    float:left;
    overflow:auto;
 }
 #content { /* основное содержимое */
    height:300px;
    float:left;
    overflow:auto;
 }
 #pane2 { /* панель правая*/
    height:300px;
    width:150px;
    float:right;
    overflow:auto;
 }
 #footer { /* подвал */
    height:30px;
    width:100%;
    overflow:hidden;
 }
</style>
</head>
<body>
<div id="page">
  <div id="header">
  </div>

  <div id="pane">
  </div>
  <div id="content">
  </div>
  <div id="pane2">
  </div>
  <div id="footer">
  </div>
</div>
</body>
</html>
```

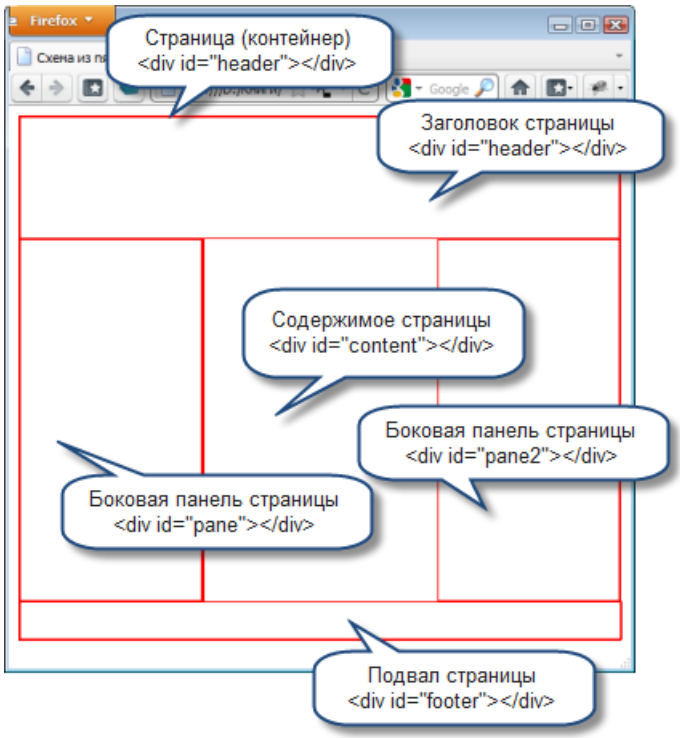


Рис. 6.3. Схема страницы из пяти ячеек (листинг 6.3)

Пятиячеечная (с двумя боковыми панелями) схема получается из рассмотренной ранее четырехъячеечной следующим образом.

Во-первых, добавляем блок `<div id="pane2">` для правой боковой панели с параметром `float:right`.

Во-вторых, все блоки схемы заключаем в контейнер `<div id="page">` с параметром `min-width` или `width`, значение которого должно быть больше суммарной ширины левой и правой панелей; разница — ширина блока для основного содержимого страницы (он не должен быть чрезмерно узким). Параметр `min-width` выбирается, если требуется так называемая "резиновая" схема, при которой блоки изменяют свою ширину в зависимости от ширины окна браузера. А параметр `width` выбирают для фиксированных по ширине страниц (жестких схем). Окончательные значения указанных параметров необходимо проверить в различных браузерах, по крайней мере, в Internet Explorer и еще каком-нибудь другом. Жесткая и резиновая схемы более подробно рассматриваются в следующих разделах.

6.2. Жесткая схема

Жесткая схема компоновки — это схема, в которой размеры всех ячеек (блоков) фиксированы посредством явного задания в абсолютных единицах стилевых параметров `width` и `height`. При изменении размеров окна браузера относительное расположение

блоков и их размеры остаются постоянными. Иногда жесткой называют схему, в которой фиксированы относительное положение и горизонтальные размеры (ширина) блоков, а высота может изменяться в зависимости от содержимого. Все схемы, рассмотренные в предыдущем разделе, не являются жесткими, поскольку ширина блоков заголовка, основного содержимого и подвала изменяется в зависимости от ширины окна браузера.

Чтобы придать жесткость, например, четырехъячеечной схеме (рис. 6.2, листинг 6.2), можно заключить все имеющиеся блоки в контейнер `<div id="page">`, которому назначить стилевой параметр `width` со значением в абсолютных единицах измерения. Значение должно быть достаточно большим, чтобы левая панель и блок для основного содержимого страницы поместились в контейнере полностью.

Ширину контейнера (а значит, и страницы в целом) выбирают исходя из типового горизонтального размера дисплея, а также с учетом характера основного содержимого. Например, текст не должен быть слишком широким, чтобы его было удобно читать. Разумеется, при большой ширине блока основного содержимого текст можно сверстать в две и более колонок или задать подходящие боковые отступы (`paddig-left`, `padding-right`).

В листинге 6.4 представлен код жесткой схемы четырехъячеечной страницы, а на рис. 6.4 — ее вид в окне браузера. Обратите внимание, что схема при данных параметрах занимает лишь некоторую часть окна браузера, а ее верхний левый угол совпадает с верхним левым углом элемента `<body>`. Видимый отступ от верхнего и левого краев окна браузера обусловлен полем (`margin`) в 8 px, принятым для `<body>` по умолчанию.

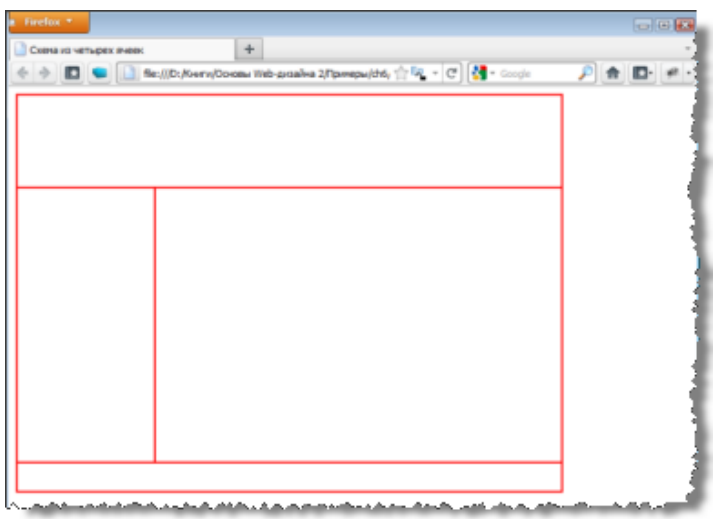


Рис. 6.4. Жесткая схема страницы из четырех ячеек (листинг 6.4)

Листинг 6.4. Жесткая схема из четырех ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
```

```
<title>Схема из четырех ячеек</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
 div {border:solid 1px red} /* для видимости в браузере*/
 #page { /* страница (контейнер)*/
    width:600px;
 }
 #header {  /* заголовок */
    height:100px;
    overflow:hidden;
 }
 #content { /* содержимое */
    height:300px;
    overflow:auto;
 }
 #pane { /* панель */
    height:300px;
    width:150px;
    float:left;
    overflow:auto;
 }
 #footer { /* подвал */
    height:30px;
    overflow:hidden;
 }
</style>
</head>
<body>
<div id="page">
  <div id="header">
  </div>
  <div id="pane">
  </div>
  <div id="content">
  </div>
  <div id="footer">
  </div>
</div>
</body>
</html>
```

6.3. Резиновая схема

Резиновая (гибкая) схема компоновки — это схема, в которой размеры некоторых ячеек (блоков) не фиксированы, а зависят (возможно, в некоторых пределах) от размеров окна браузера. При этом относительное положение блоков остается постоянным.

Резиновые схемы создаются с помощью параметров `width` со значениями в процентах, а также `min-width` и `max-width` — для задания ограничений на ширину и обеспечения стабильности относительного положения блоков. Заметим, что все схемы, рассмотренные в *разд. 6.1*, — резиновые.

Чтобы жесткую схему листинга 6.4 превратить в резиновую, достаточно для контейнера `<div id="page">` установить следующие стилевые параметры:

```
width:число%;
min-width:числоpx;
```

Например,

```
width:100%;
min-width:800px;
```

В данном случае схема растягивается по ширине окна браузера, если последняя превышает 800 px. В противном случае ширина схемы останется равной 800 px даже при более узком окне браузера. Если значение параметра `width` установить, например, 70%, то ширина схемы будет составлять 70% от ширины окна браузера, только если последняя превышает 800 px.

6.4. Центрирование страницы

По некоторым причинам схема страницы может быть выбрана так, что ее ширина может оказаться меньше ширины окна браузера. В этом случае неплохо центрировать страницу по горизонтали, т. е. сделать одинаковыми по ширине левое и правое поля (см. *разд. 4.8.1*).

Для центрирования элемента по горизонтали необходимо установить для него параметр ширины `width` (в абсолютных или относительных единицах измерения), а также поля `margin-left:auto` и `margin-right:auto`. При этом элемент будет центрироваться относительно содержащего его контейнера. Например, в листинге 6.4 набор стилевых параметров для контейнера `<div id="page">` достаточно определить так:

```
#page { /* страница (контейнер)*/
  margin-left:auto;
  margin-right:auto;
  width:1000px;
}
```

Поскольку элемент `<div id="page">` находится непосредственно в контейнере `<body>`, то он будет центрироваться относительно документа, т. е. окна браузера (точнее, клиентской его части). Следует иметь в виду, что `<body>` имеет свои поля с установленными по умолчанию значениями 8 px. Эти значения, разумеется, можно установить по своему выбору, например обнулить.

6.5. Декорация схемы

Выбрав компоновочную схему страницы, можно заняться цветовыми решениями — установкой фоновых цветов или/и графических изображений, а также других стилевых параметров, влияющих на внешний вид (см. *разд. 3.7* и *главу 5*). В данном разделе мы

рассмотрим лишь пример оформления одной из схем, а именно жесткой четырехъячеечной с центрированием по горизонтали. Выбор конкретных значений параметров — дело вашего вкуса как дизайнера. Предлагаемые здесь значения параметров не следует рассматривать как хорошие или рекомендуемые.

Поскольку схема центрирована, то следует задать фон документа или, что то же самое, клиентской области окна браузера. Для этого установим цвет фона для элемента `<body>`, например, серый:

```
body { background:#b0b0b0}
```

Элементы `<div>` имеют по умолчанию прозрачный фон и, следовательно, на данном этапе они будут того же цвета, что и `<body>`.

Заголовок, левую панель и подвал зальем одинаковым, например, бирюзовым цветом, а блок основного содержимого — белым цветом:

```
#page {background: cyan}
#content {background:white}
```

Обратите внимание на то, что для окраски всех блоков схемы одинаковым цветом (в нашем примере — cyan) достаточно установить цвет фона лишь для одного контейнера `<div id="page">`. Для блока основного содержимого страницы цвет фона устанавливаем отдельно.

Если требуется, чтобы правый край блока основного содержимого был такого же цвета, что и у других блоков, то достаточно установить поле:

```
#content {margin-right:10px}
```

Чтобы закруглить углы страницы и создать эффект тени, установим такие параметры:

```
#page {
   border-radius: 25px/20px;
   box-shadow: 5px 5px 10px black;
   -webkit-box-shadow: 5px 5px 10px; /* для Chrome и Safari */
}
```

> ***Примечание***
>
> Закругление углов и тень, заданные параметрами CSS 3, не будут воспроизводиться в Internet Explorer 8 и более ранних версиях. Для них аналогичные эффекты можно создать применением соответствующих фоновых графических изображений, а также графических фильтров — фирменного (нестандартного) расширения CSS от Microsoft.

В листинге 6.5 приведен код, а на рис. 6.5 — его вид в браузере Firefox. При этом для наглядности мы пока оставили технологические видимые границы блоков. Размеры блоков следует скорректировать с учетом типовых размеров экрана, а также того, что тень занимает дополнительное пространство, сверх установленного параметрами `width` и `height`. Не помешает еще зарезервировать место (примерно 20—30 px) для вертикальных полосы прокрутки и границ окна браузера.

Листинг 6.5. Компоновочная схема с декорацией

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Схема из четырех ячеек</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
 div {border:solid 1px red} /* для видимости границ в браузере*/
 body {
    background:gray;
 }
 #page { /* страница (контейнер)*/
  margin-left:auto;
  margin-right:auto;
  width:500px;
  background:cyan;
  border-radius:25px/20px;
  box-shadow:5px 5px 10px black;
  -webkit-box-shadow:5px 5px 10px black; /* для Chrome и Safari */
 }
 #header {  /* заголовок */
    height:100px;
    overflow:hidden;
 }
 #content { /* содержимое */
    height:300px;
    overflow:auto;
    background:white;
    margin-right:10px;
 }
 #pane { /* панель */
    height:300px;
    width:150px;
    float:left;
    overflow:auto;
 }
 #footer { /* подвал */
    height:30px;
    overflow:hidden;
 }
</style>
</head>
<body>
<div id="page">
  <div id="header">
  </div>
  <div id="pane">
  </div>
  <div id="content">
  </div>
```

```
  <div id="footer">
  </div>
</div>
</body>
</html>
```

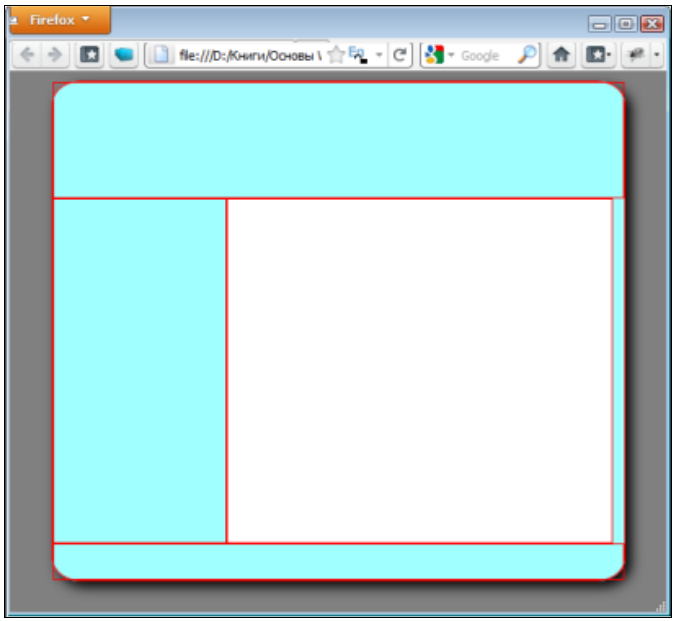


Рис. 6.5. Вид декорированной компоновочной схемы с видимыми границами блоков

Совет

Попробуйте установить в качестве фона блоков графические изображения, используя для этого параметры `background-image` или `border-image`.

6.6. Вставка плавающего фрейма *(<iframe>)*

Основное содержимое страницы в виде обычного текста или/и HTML-кода можно вставить непосредственно в соответствующий тег `<div>` (в наших примерах — `<div id="page">`). Но тогда другие страницы сайта придется показывать вместо данной либо в другой вкладке или окне браузера. Если при этом вы хотите, чтобы на страницах перехода сохранились некоторые элементы дизайна главной страницы (например, панель навигации, заголовок и подвал), то последние придется повторить в кодах вызываемых страниц. В данном случае при переходе к другой странице сайта будет происходить и перезагрузка общих элементов страниц. Альтернативой является использование так называемого плавающего или внутреннего фрейма, который задается контейнерным внутристрочным тегом `<iframe>`.

В элемент `<iframe src="`*URL-адрес*`" name="`*имя_фрейма*`"></iframe>` можно загрузить файл, указанный в атрибуте `src`. Атрибут `name` содержит имя фрейма, используемое в ссылках для указания, что требуемый ресурс следует загрузить в данный фрейм. Для этого в

теге <a> ссылки предусмотрен атрибут target, в качестве значения которого можно указать имя фрейма. Например, щелчок на ссылке Моя фотогалерея приведет к попытке загрузить файл myphoto.htm в плавающий фрейм с атрибутом name="myframe" (если, конечно, он задан).

Загрузка файла в плавающий фрейм происходит без перезагрузки страницы, в которой он находится. Используя его для загрузки и отображения дополнительных страниц сайта (со своим дизайном), вы не перезагружаете в браузер главную страницу с ее заголовком, элементами навигации и другими элементами общего назначения. Это, пожалуй, главное достоинство применения <iframe>. Кроме того, данная конструкция требует меньших трудозатрат при модификации общих элементов: требуемые изменения достаточно произвести лишь в одном файле главной страницы, а не во всех.

К недостаткам следует отнести, прежде всего, то обстоятельство, что важная для индексирования поисковыми системами информация находится не в файле главной страницы, а в файле, загружаемом в плавающий фрейм. Однако данный недостаток преодолим специальными мерами. Так, достаточно разместить текст для поисковых систем под плавающий фрейм. При этом желательно разметить его тегами заголовков <h1>, <h2>, ... <h6>. С помощью CSS вы можете задать небольшой размер и неяркий цвет шрифта, но не следует выбирать цвет текста, совпадающий с фоном или делать невидимым с помощью параметра visibility:hidden (системы индексации могут распознать данную уловку).

Другой недостаток применения плавающих фреймов заключается в том, что посетитель может загрузить документ из фрейма в отдельной вкладке или окне браузера. В этом случае он отобразится без общих для всех страниц сайта элементов. Иногда это вполне допустимо, а иногда очень нежелательно. Но и данный недостаток легко преодолим. Для этого достаточно в каждый документ, загружаемый во фрейм, вставить простой скрипт (см. *разд. 15.1.3*), который при попытке загрузки в отдельную вкладку или окно выполнит переадресацию к главной странице сайта. Так, например, главную страницу Яндекса нельзя загрузить во фрейм.

Рассмотрим пример страницы с плавающим фреймом. Для этого возьмем за основу код, представленный в листинге 6.5, и модифицируем его. Результат приведен в листинге 6.6 (рис. 6.6). В данном примере во фрейм загружается главная страница **www.rambler.ru.**

Листинг 6.6. Страница с плавающим фреймом

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Схема с фреймом</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
<style type="text/css">
  body {background:gray;}
 #page { /* страница (контейнер)*/
  margin-left:auto;
```

```
  margin-right:auto;
  width:500px;
  background:#a0ffff;
  border-radius:25px/20px;
  box-shadow:5px 5px 10px black;
 }
 #header {  /* заголовок */
    height:100px;
    overflow:hidden;
 }
 #content { /* содержимое */
    height:300px;
    overflow:auto;
    background:white;
    margin-right:10px;
    overflow:hidden;
 }
 #pane { /* панель */
    height:300px;
    width:150px;
    float:left;
    overflow:auto;
 }
 #footer { /* подвал */
    height:30px;
    overflow:hidden;
 }
 #myframe { /* плавающий фрейм*/
  width:100%;
  height:100%;
  border-width:0; /* без границы */
 }
 #xcontent { /* текст, маскируемый фреймом */
  width:100%;
  position:relative;
  top:-300px;
  z-index:-1;
 }
</style>
</head>
<body>
<div id="page">
  <div id="header">
  </div>
  <div id="pane">
  </div>
  <div id="content">
    <iframe id="myframe" name="myframe"
            src="http://www.rambler.ru">
    </iframe>
```

```
    <div id="xcontent">
      <h1>Вместо данных слов напишите текст,
          содержащий ключевые слова
      </h1>
    </div>
  </div>
  <div id="footer">
  </div>
</div>
</body>
</html>
```

Рис. 6.6. Страница с плавающим фреймом (листинг 6.6)

Плавающий фрейм и маскируемый им текст вставлены в контейнер `<div id="content">`, для которого запрещено появление полос прокрутки (`overflow:hidden`), поскольку при необходимости они появятся у фрейма. По умолчанию фрейм имеет видимые границы. Чтобы их убрать, следует просто задать нулевое значение параметра `border`. Это сработает во всех браузерах, кроме Internet Explorer 8 и более ранних версий.

Маскируемый фреймом текст размещен в контейнере `<div id="xcontent">`, который позиционируется относительно контейнера `<div id="content">` со смещением вверх на высоту фрейма (`position:relative; top:-300px`). При этом он размещается под фрейм (`z-index:-1`).

> ***Совет***
>
> Уже на данном этапе изучения вы можете в порядке эксперимента попробовать вставить в код выбранной компоновочной схемы некоторые элементы, такие как название сайта, ссылки и графические изображения. Как это делать, будет более подробно рассмотрено в следующих главах.

ГЛАВА 7

Ссылки

Главным инструментом большинства Web-страниц являются ссылки (или *гиперссылки*). Ссылки могут быть текстовыми и графическими в зависимости от вида указателя ссылки — отображаемого элемента, щелчок на котором левой кнопкой мыши инициализирует переход к другому документу. Текстовые ссылки представляют собой слово или целую фразу, выделенную цветом и/или подчеркиванием, в зависимости от настройки браузера. В графических ссылках роль чувствительного к щелчку элемента играет изображение. Можно также создать комбинированные ссылки, указателем в которых будет и изображение, и текст. Ссылки на фрагменты текущего документа (внутренние ссылки) организуются особым образом. В любом случае ссылку задает контейнерный тег `<a>`.

Для создания ссылки на другой документ используется следующий синтаксис:

```
<a href="адрес_ссылки">указатель_ссылки</a>
```

Здесь атрибут `href` (от англ. *hyper reference* — гиперссылка) принимает в качестве значения URL-адрес или имя файла (возможно, с указанием пути), к которому следует перейти при инициализации ссылки; *указатель_ссылки* — текст или/и тег `<img>`, вставляющий графическое изображение, на которых можно щелкнуть, чтобы активизировать ссылку. Об URL будет рассказано в *разд. 7.5*.

Документ, вызываемый при активизации ссылки, по умолчанию открывается в текущем окне браузера, заменяя документ, содержащий эту ссылку. Это происходит, если браузер способен открыть документ в своем окне (некоторые документы открываются в окне соответствующих приложений). Однако вы можете принудительно назначить, в каком окне браузера или фрейме следует открыть вызываемый документ. Это делается с помощью атрибута `target`, который принимает следующие значения:

- *имя_окна* — имя (значение атрибута `name`) окна или фрейма;
- `_blank` — указатель на новое окно;
- `_parent` — указатель на родительский фрейм, а если фреймов нет, то данный параметр работает как `_self`;
- `_top` — отменяет все фреймы и загружает документ в главное окно браузера, а если фреймов нет, то данный параметр работает как `_self`;
- `_self` — указатель на тот же самый (текущий) фрейм или окно.

Если требуется, чтобы при наведении указателя мыши на указатель ссылки появлялась всплывающая подсказка (поясняющий текст), то в теге <a> следует добавить атрибут title, принимающий в качестве значения текст всплывающей подсказки.

7.1. Текстовые ссылки

Структура текстовой ссылки имеет следующий вид:

```
<a href="адрес_ссылки">текст_ссылки</a>
```

Например, следующий тег описывает ссылку на HTML-файл мой_документ.htm, при этом ссылка на экране будет выглядеть в виде слова "Портфолио":

```
<a href="мой_документ.htm">Портфолио</a>
```

Браузер не выводит в клиентскую область окна имя файла, к которому требуется перейти по ссылке, а лишь показывает текст, заключенный между открывающей и закрывающей частями тега <a>.

Можно ссылаться не только на другие файлы, но и на собственный файл. Поскольку настройки цвета в браузере у различных пользователей могут отличаться, возникает задача принудительного задания цвета, чтобы ссылки были хорошо видны и соответствовали дизайну вашего сайта. Это делается с помощью псевдоклассов CSS:

- a:link {color:white} — неиспользованные ссылки;
- a:visited {color:gray} — использованные ссылки;
- a:hover {color:#00ffff} — ссылка, на которой находится указатель мыши;
- a:active {color:yellow} — активная ссылка.

> ***Примечание***
>
> В приведенных примерах ссылка задавалась как имя файла. В этом случае предполагается, что файл находится в той же папке, что и документ, содержащий эту ссылку. Если этого не достаточно, то вы можете указать базовый адрес с помощью тега <base>, задать относительные и абсолютные пути, а также URL-адрес файла (см. *разд. 7.5*).

7.1.1. Простое меню ссылок

Нередко набор ссылок оформляют в виде так называемого кнопочного меню. Опции такого меню в окне браузера выглядят как прямоугольники с текстом. Прямоугольники можно создать с помощью тегов , а в них поместить ссылки. Как для контейнеров , так и для содержащихся в них ссылок (<a>) с помощью параметров CSS можно задать оформление. Для меню обычно требуется, чтобы его опция под указателем мыши как-то выделялась (подсвечивалась). Данного эффекта легко достичь применением псевдокласса :hover.

Простое или, другими словами, одноуровневое меню может быть горизонтальным и вертикальным. На рис. 7.1 приведен пример горизонтального меню. Опции (кнопки) меню созданы посредством элементов , заключенных в контейнер . Элементы внутристрочные и, следовательно, располагаются в одном ряду, что и требуется для горизонтального меню. Однако чтобы соседние прямоугольники вплотную

примыкали друг к другу, для элементов `<span>` назначен параметр `float:left`. Для ссылок задан параметр `text-decoration:none`, чтобы они не подчеркивались, поскольку и так ясно, что кнопки это элементы, на которых можно щелкнуть.

Обратите внимание на селекторы в правилах CSS. Например, двухэлементный селектор `.menu a` задает стилевые параметры только для тех ссылок, которые находятся внутри элемента с атрибутом `class="menu"` (а в нашем примере это контейнер `<div class="menu">`).

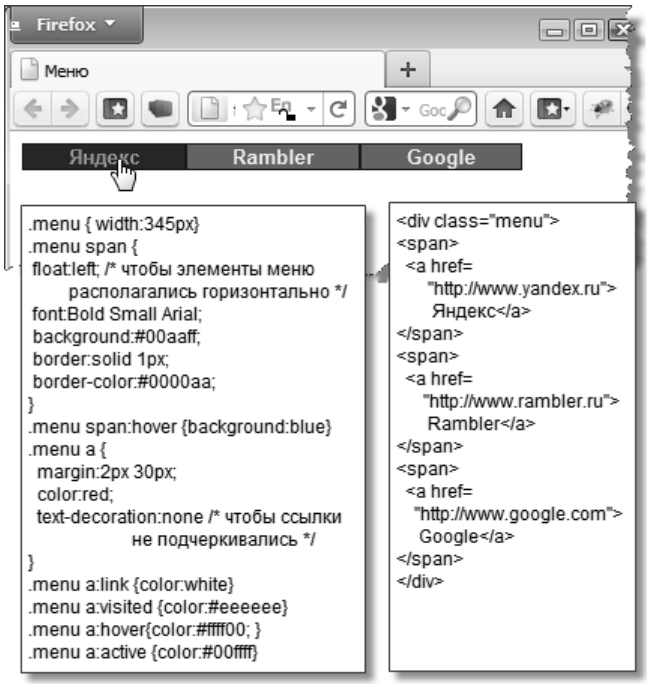


Рис. 7.1. Пример горизонтального меню

В листинге 7.1 показано, как вставить данное горизонтальное меню в раздел для заголовка страницы, которая рассматривалась в *разд. 6.6*. Здесь мы ее немного модифицировали, вставив название сайта и меню. Вид соответствующей страницы в браузере представлен на рис. 7.2.

Листинг 7.1. Горизонтальное меню

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Схема с фреймом</title>
<meta http-equiv="Content-Type" content="text/html;
      charset=windows-1251"/>
<meta name="Keywords" content=""/>
<meta name="Description" content=""/>
```

```
<style type="text/css">
  body {
    background:gray;
 }
 #page { /* страница (контейнер)*/
  margin-left:auto;
  margin-right:auto;
  width:800px;
  background:#a0ffff;
  border-radius:25px/20px;
  box-shadow:5px 5px 10px black;
 }
 #header {  /* заголовок */
    height:100px;
    overflow:hidden;
 }
 #content { /* содержимое */
    height:300px;
    overflow:auto;
    background:white;
    margin-right:10px;
    overflow:hidden;
 }
 #pane { /* панель */
    height:300px;
    width:150px;
    float:left;
    overflow:auto;
 }
 #footer {
    height:30px;
    overflow:hidden;
 }
 #myframe { /* плавающий фрейм*/
    width:100%;
    height:100%;
    border-width:0;
 }
 #xcontent { /* текст, маскируемый фреймом */
    width:100%;
    position:relative;
    top:-300px;
    z-index:-1;
 }
#header h1 {margin-left:50px}
/*** Меню: ******************************************************/
.menu {
   width:345px;
   position:relative;
```

```
   top:2px;
   float:right; /* прижимаем к правому краю */
}
.menu span {
   float:left; /* чтобы элементы меню располагались горизонтально */
   font:Bold Small Arial;
   background:#00aaff;
   border:solid 1px;
   border-color:#0000aa;
}
.menu span:hover {background:blue}
.menu a {
   margin:2px 30px;
   color:red;
   text-decoration:none /* чтобы ссылки не подчеркивались */
}
.menu a:link {color:white}
.menu a:visited {color:#eeeeee}
.menu a:hover{color:#ffff00; }
.menu a:active {color:#00ffff}
</style></head>
<body>
<div id="page">
  <div id="header">
    <h1>Мой сайт</h1>
    <div class="menu">
       <span><a href="http://www.yandex.ru" target="myframe">Яндекс</a>
       </span>
       <span><a href="http://www.rambler.ru" target="myframe">Rambler</a>
       </span>
       <span><a href="http://www.google.com" target="myframe">Google</a>
       </span>
    </div>
  </div>
  <div id="pane">
  </div>
  <div id="content">
    <iframe id="myframe" name="myframe" src=""></iframe>
    <div id="xcontent">
      <h1>Вместо данных слов напишите текст,
          содержащий ключевые слова
      </h1>
    </div>
  </div>
  <div id="footer">
  </div>
</div>
</body>
</html>
```

Рис. 7.2. Горизонтальное меню, встроенное в заголовок страницы (листинг 7.1)

Подсветку опции меню под указателем мыши можно сделать более интересной, если указать в качестве фона элемента `<span>` в обычном состоянии одно графическое изображение, а под указателем мыши — другое:

```
.menu span: {background:url("button1.jpg")};
.menu span:hover {background:url("button2.jpg")};
```

Рассмотренное горизонтальное меню легко превратить в вертикальное. Для этого достаточно в CSS для элемента `<span>` удалить параметр `float:left` и добавить параметр `display:block` (превратить элемент из внутристрочного в блочный), а для `<div class="menu">` подобрать подходящее значение параметра `width`.

Вертикальное меню можно сделать раскрывающимся. В исходном (свернутом) состоянии видна только первая опция, ссылка к которой может быть и не привязана (первая опция обычно служит только для раскрытия всего меню). Наведение на нее указателя мыши приводит к появлению остальных опций с привязанными ссылками, а уход мыши свертывает меню в исходное состояние. Для реализации данного меню достаточно немного модифицировать класс `.menu` и добавить набор параметров `.menu:hover`. При этом необходимо эмпирически подобрать исходную высоту элемента `<div class="menu">` и его высоту под указателем мыши, а также ширину. В нашем примере можно внести следующую коррекцию в CSS для меню:

```
.menu {
   width:120px;
   height:19px; /* чтобы была видна только одна опция */
   overflow:hidden; /* остальные опции не видны */
```

```
   border:solid 1px red;
 }
.menu:hover {height:54px /* чтобы были видны все опции*/}
```

Чтобы при раскрытии меню остальные элементы страницы оставались на своих исходных местах, следует либо предусмотреть свободное место на странице, которое займет раскрытое меню, либо позиционировать его абсолютно. При абсолютном позиционировании (`position:absolute`) место можно не резервировать, поскольку меню раскроется над остальными элементами, если задано достаточно большое значение параметра `z-index`. На практике раскрывающееся меню обычно позиционируют абсолютно. Тогда набор параметров для элемента `<div class="menu">` будет выглядеть следующим образом:

```
.menu {
   position:absolute;
   z-index:100;
   width:120px;
   height:19px; /* чтобы была видна только одна опция */
   overflow:hidden; /* остальные опции не видны */
   border:solid 1px red;
 }
```

7.1.2. Двухуровневое меню ссылок

Простейшее двухуровневое меню представляет собой несколько рядом расположенных вертикальных меню, рассмотренных в предыдущем разделе. В листинге 7.2 приведен код, а на рис. 7.3 — внешний вид двухуровневого меню, состоящего из трех вертикальных меню. В исходном состоянии видна только горизонтальная часть меню. Это обеспечивается параметрами `overflow:hidden` и `height` с достаточно малым значением. При наведении указателя мыши на первую или вторую опции горизонтальной части меню раскрывается соответствующее вертикальное подменю, содержащее ссылки (увеличивается значение `height`). Для третьей опции подменю не предусмотрено, она сама содержит единственную ссылку. Чтобы при наведении мыши на опцию меню (`<div>`) ее указатель сохранял форму в виде стрелки, установлен параметр `cursor:default`. Однако при наведении мыши на собственно ссылку указатель примет обычный для такого случая вид руки.

Набор вертикальных подменю заключен в общий контейнер `<div class="supermenu">`, который позиционируется абсолютно. Вы можете расположить его на странице где угодно, раскрытые подменю будут перекрывать находящиеся под ними элементы.

Листинг 7.2. Двухуровневое меню

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Двухуровневое меню</title>
<style type="text/css">
```

```
.supermenu { /* двухуровневое меню */
     position:absolute;
     z-index:100;
     top:20px;left:90px
}
.menu { /* вертикальное подменю */
     width:100px;
     height:19px;
     background:#00ddff;
     border:solid 1px #000000;
     text-align:center;
     font-weight:bold;
     color:white;
     float:left; /* для расположения рядом друг с другом */
     overflow:hidden /* подменю не видны */
}
/* высота при наведении указателя мыши: */
#menu1:hover{height:80px;cursor:default;
}
#menu2:hover{height:60px;cursor:default;}
/* ссылки: */
.menu a {
     padding-left:10px;
     display:block;
     color:#000000;
     text-decoration:none
}
.menu a:visited {color:#808080}
.menu a:hover{color:#ffff00; background:blue}
.menu a:active {color:#ffffff}
</style>
</head>
<body>
<div class="supermenu">
<div id="menu1" class="menu">
  Поисковики
  <a href="http://www.rambler.ru/">Rambler</a>
  <a href="http://www.yandex.ru/">Яндекс</a>
  <a href="http://www.google.com/">Google</a>
</div>
<div id="menu2" class="menu">
  Почта
  <a href="http://www.mail.ru/">Mail.ru</a>
  <a href="http://mail.google.com/mail/">Gmail</a>
</div>
<div  class="menu">
  <a href="http://ru.wikipedia.org/" style="color:white">Википедия</a>
</div>
</div>
```

```
<!-- Просто текст -->
<p style="position:relative;top:30px">
   На данной странице представлен пример<br/>
   двухуровневого меню, составленного из трех вертикальных меню
</p>
</body>
</html>
```

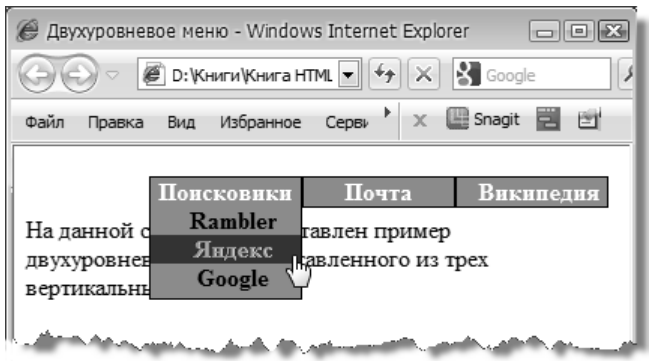


Рис. 7.3. Двухуровневое меню (листинг 7.2)

ВНИМАНИЕ!

В данном примере, в отличие от рассмотренного в предыдущем подразделе, ссылка сработает не только при щелчке на ее тексте, но и на всей соответствующей строке подменю.

Небольшая модификация кода листинга 7.2 позволяет создать вертикальное двухуровневое меню, в котором подменю появляются правее и немного ниже соответствующих опций главного меню. На рис. 7.4 показан внешний вид такого меню в окне браузера, а в листинге 7.3 — его код. В данном примере вертикальное расположение опций главного меню достигается тем, что в CSS в классе `.menu` не применяется параметр `float:left`. Появление подменю задается параметром `overflow:visible`, а их положение — координатами `top` и `left`, определяющими смещение при относительном пози-

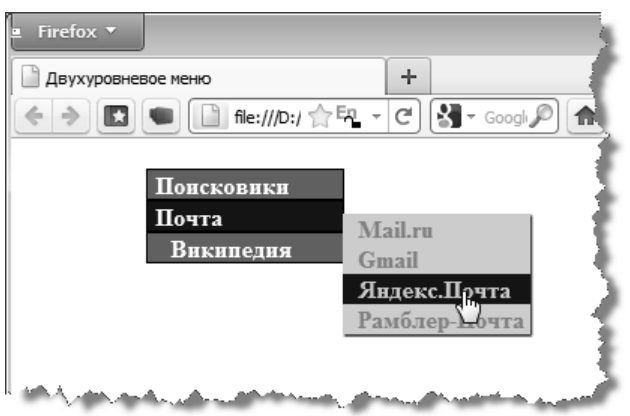


Рис. 7.4. Вертикальное двухуровневое меню (листинг 7.3)

ционировании (position:relative). Так, в исходном состоянии подменю не видно потому, что для соответствующего контейнера задан параметр overflow:hidden и, кроме того, содержащийся в нем имеет горизонтальную координату (left:125px), которая равна ширине контейнера (width:100px). Напомню, что в предыдущем примере появление подменю управлялось назначением достаточно большой высоты (height) для элемента , содержащего подменю вместе с соответствующей опцией главного меню (селекторы #menu1:hover и #menu2:hover).

Листинг 7.3. Вертикальное двухуровневое меню

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Двухуровневое меню</title>
<style type="text/css">
.supermenu { /* Двухуровневое меню */
     position:absolute;
     z-index:100;
     top:20px;left:90px
}
.menu { /* вертикальное меню */
     width:125px;
     height:19px;
     background:#00ddff;
     border:solid 1px black;
     text-align:left;
     padding-left:5px;
     font-weight:bold;
     color:white;
     overflow:hidden /* подменю не видны */
}

#menu1:hover, #menu2:hover {
    background:blue;
    cursor:default;
    overflow:visible; /* подменю видны */
}
/* ссылки: */
.menu a {
     padding-left:10px;
     display:block;
     color:black;
     text-decoration:none
}
.menu a:visited {color:#808080}
.menu a:hover{color:#ffff00; background:blue}
.menu a:active {color:#ffffff}
.submenu {
     position:relative;
```

```
    left:125px;
    top:-12px;
    background:#cccccc;
    box-shadow:1px 1px 1px black;
}
</style>
</head>
<body>
<div class="supermenu">
<div id="menu1" class="menu">
  Поисковики
  <div class="submenu">
  <a href="http://www.rambler.ru/">Rambler</a>
  <a href="http://www.yandex.ru/">Яндекс</a>
  <a href="http://www.google.com/">Google</a>
  </div>
</div>
<div id="menu2" class="menu">
  Почта
  <div class="submenu">
  <a href="http://www.mail.ru/">Mail.ru</a>
  <a href="http://mail.google.com/mail/">Gmail</a>
  <a href="http://mail.yandex.ru/">Яндекс.Почта</a>
  <a href="http://mail.rambler.ru/">Рамблер-Почта</a>
  </div>
</div>
<div  class="menu">
  <a href="http://ru.wikipedia.org/" style="color:white">Википедия</a>
</div>
</div>
</body>
</html>
```

Меню можно реализовать и на основе списков (см. *разд. 9.5*).

7.2. Графические и комбинированные ссылки

Ссылку, указатель которой является картинкой, называют графической. Ее структура имеет вид:

```
<a href="адрес_ссылки"><img src="адрес_графического_файла"/></a>
```

Например, следующий тег описывает ссылку на (X)HTML-файл мой_документ.htm, при этом ссылка на экране будет представлена изображением из файла picture.gif:

```
<a href="мой_документ.htm"><img src="picture.gif"/></a>
```

К графической ссылке можно добавить поясняющий текст:

```
<a href="мой_документ.htm"><img src="picture.gif"/>Щелкните здесь</a>
```

В этом примере переход к документу мой_документ.htm произойдет при щелчке левой кнопкой мыши как на графическом изображении, так и на поясняющем тексте, поскольку и то и другое находятся в контейнерном теге <a>.

Графическое изображение внутри ссылки по умолчанию выводится на экран в рамке такого же цвета, каким выделяются текстовые ссылки. Цвет рамки около изображения графических ссылок в различных состояниях (активная, посещенная и т. п.) можно установить в настройках браузера или в таблице CSS посредством параметра color так же, как и для текстовых ссылок (см. *разд. 7.1*). Чтобы исключить окаймление изображения в графической ссылке, достаточно применить параметр text-decoration:none.

> ***Примечание***
>
> В приведенных примерах ссылка задавалась как имя файла. В этом случае предполагается, что файл находится в той же папке, что и документ, содержащий эту ссылку. Если этого недостаточно, то вы можете указать базовый адрес с помощью тега <base>, задать относительные и абсолютные пути, а также URL-адрес файла (см. *разд. 7.5*).

7.3. Графические карты ссылок

В рассмотренных ранее графических ссылках одному изображению соответствовал один адрес ссылки. Однако существует возможность сопоставить одному изображению несколько ссылок, привязав каждую из них к некоторой области изображения. Такие области называют *"горячими"* или *активными*, а сам технологический прием — *графической картой ссылок* или *сегментированной графикой*. "Горячие" области графической карты могут быть различной формы: прямоугольной, многоугольной или в виде круга. Это очень удобный способ оформления группы ссылок, однако при выборе рисунка, служащего основой карты ссылок, следует стремиться к тому, чтобы границы "горячих" областей были хорошо очерченными и не пересекались. Кроме того, необходимо позаботиться о том, чтобы пользователь понял, что имеет дело с картой ссылок, а не просто с графическим фоном. Для этого можно предусмотреть поясняющие тексты. При наведении на "горячую" область указатель мыши изменяет свою форму, как и при обычных ссылках. При щелчке на "горячей" области ее границы становятся видимыми.

Возможны два варианта реализации графических карт ссылок: клиентский и серверный. В первом варианте вся информация о конфигурации графической карты ссылок размещается в HTML-документе, в котором она применяется. Во втором варианте предполагается наличие так называемого CGI-сценария, который располагается на сервере и выполняет обработку запросов браузера клиента при работе с графической картой ссылок. Кроме того, на сервере должен находиться конфигурационный файл, в котором описываются "горячие" области карты. Хотя серверный вариант появился раньше, теперь чаще встречается клиентский вариант. В данной книге рассматривается только клиентская графическая карта ссылок.

Графическую карту ссылок в клиентском варианте задают несколько тегов. Первый — контейнерный тег <map> (карта) с атрибутом name для указания имени карты. Имя карты выбирается как имя переменной. Далее, между дескрипторами <map> и </map> следуют теги <area> (область) для задания "горячих" областей.

Тег <area> имеет ряд атрибутов, описывающих собственно ссылку, а также форму и положение "горячей" области:

- `href` — строка, определяющая адрес ссылки;
- `shape` — определяет форму области; принимает аргументы:
  - `"rect"` или `"rectangle"` (прямоугольник);
  - `"poly"` или `"polygone"` (многоугольник);
  - `"circle"` (круг);
- `coords` — координаты области, которые задаются в пикселах перечнем чисел, разделенных запятыми; весь перечень заключают в кавычки (для прямоугольника задают четыре числа — координаты верхнего левого и правого нижнего угла; для многоугольника указывают координаты каждого угла; для круга задают три числа — координаты центра и радиус); координаты углов перечисляют в следующем порядке: сначала горизонтальная, а затем вертикальная координата; начало отсчета координат — верхний левый угол изображения в своих исходных размерах (масштабирование изображения назначением параметров `width` и `height` может привести к смещению "горячих" областей);
- `title` — текстовая строка всплывающей подсказки, появляющейся при наведении указателя мыши на "горячую" область;
- `target` — указывает окно или фрейм, в котором следует открыть документ, заданный в адресной части ссылки; имеет такие же значения, как и для тега `<a>`;
- `nohref="nohref"` — атрибут для указания того, что данная область не имеет адреса перехода; атрибуты `href` и `nohref` взаимно исключают друг друга (следует использовать только какой-нибудь один из них); атрибут `nohref` обычно применяется для того, чтобы исключить переход по ссылке для части уже определенной "горячей" области.

После закрывающего дескриптора `</map>` следует указать тег, вставляющий изображение и реализующий привязку карты к нему — это уже известный тег `<img>`, в котором помимо прочих возможных атрибутов задается атрибут связи с картой:

`usemap="#`*`имя_карты`*`"`

В качестве имени карты приводится значение атрибута `name` тега `<map>`.

В листинге 7.4 приведен пример графической карты ссылок, в основе которой находится изображение меню (файл map.jpg), показанное на рис. 7.5. На ней определены три прямоугольные активные области, соответствующие опциям меню. При щелчке на активной области в окне браузера будет выведен соответствующий документ.

Листинг 7.4. Пример графической карты ссылок

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Карта ссылок</title>
<style type="text/css">
#mymap { /* изображение карты */
  position:absolute;
  left:0;
  top:0;
```

```
  width:800px;
  heigh:800px;
  border-style:none;
}
</style>
</head>
<body>
<map name="mymap">
     <area href="http://www.yandeks.ru"
        shape="rect" coords="25,40,150,70"
        target="_blank" title="Яндекс"/>
     <area href="http://dunaevv1.narod.ru"
        shape="rect" coords="25,100,370, 130"
        target="_blank" title="Помощь начинающим разработчикам"/>
     <area href="http://www.bhv.ru"
        shape="rect" coords="25,150,270,190"
        target="_blank" title="Издательство БХВ-Петербург"/>
</map>
<img usemap="#mymap" id="mymap" src="map.jpg" />
</body>
</html>
```

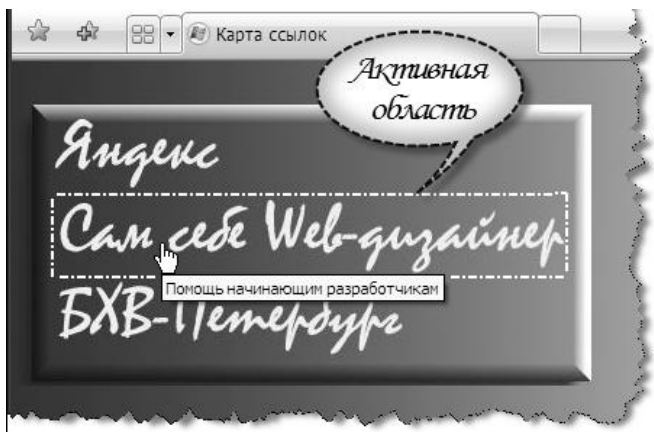


Рис. 7.5. Внешний вид графической карты ссылок

В теге `<area>` атрибут `target="_blank"` указывает браузеру открыть документ в новом окне или в новой вкладке (в зависимости от настроек).

7.4. Внутренние ссылки

Ссылки на разделы одного и того же документа называют *внутренними* или *закладками*. Например, на своей странице вы размещаете статью объемом в несколько десятков страниц как единый (X)HTML-документ. Скорее всего, вам захочется сделать ссылки на предыдущие или последующие разделы этого документа. Предметный указатель к книге также можно оформить с помощью внутренних ссылок.

Для организации внутренней ссылки необходимы якорь (anchor, анкер) и собственно ссылка. Якорь определяет место в документе, к которому происходит переход по ссылке. Ссылка использует имя якоря вместо имени (адреса) файла.

Формат якоря:

```
<a name="имя_якоря">текст</a>
```

Имя якоря (значение атрибута `name`) может быть произвольным и задается просто как имя переменной. Текст внутри тега `<a>` для создания якоря не обязателен и чаще всего отсутствует.

Формат ссылки:

```
<a href="#имя_якоря">указатель_ссылки</a>
```

Указатель ссылки может быть как текстовым, так и графическим.

В листинге 7.5 приведен пример, демонстрирующий внутренние ссылки (рис. 7.6).

Листинг 7.5. Использование внутренних ссылок

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Внутренние ссылки</title>
<style type="text/css">
  a {font-weight:bold}
</style>
</head>
<body>
<h2>Содержание</h2>
<a href="#Глава1">Глава 1. Основы HTML</a>
<br/>
<a href="#Глава2">Глава 2. Примеры</A>
<br/>
<a name="Глава1"></a>
<h2>Глава 1.Основы HTML</h2>
В этой главе мы собираемся рассмотреть основные элементы
языка HTML. На языке HTML создаются документы, которые могут
быть опубликованы в Интернете. После изучения основных
конструкций языка мы подробно
рассмотрим примеры программ (<a href="#Глава2">см. главу 2</a>).
<br/>
<a name="Глава2"></a>
<h2>Глава 2. Примеры</h2>
Здесь мы рассмотрим примеры программ с использованием элементов языка,
которым посвящена
<a href="#Глава1">Глава 1.</a>
Прежде всего, рассмотрим применение текстовых ссылок.
</body>
</html>
```

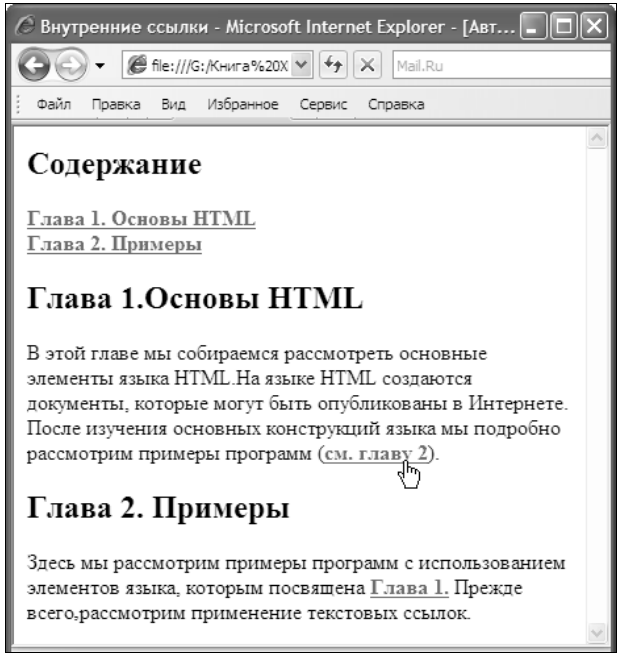


Рис. 7.6. Внутренние ссылки

7.5. URL-адреса

7.5.1. Структура URL

Адрес ресурсов Интернета, как известно, вводится в соответствующее поле браузера, а в (X)HTML-документах он задается обычно как значение атрибутов:

- `href` — в тегах `<a>`, `<area>`, `<link>`;
- `src` — в тегах `<img>`, `<iframe>`, `<script>`, `<embed>` и др.;
- `action` — в теге `<form>`;
- `data` — в теге `<object>`.

Адрес задается в виде URL (Uniform Resource Locator — унифицированный указатель местоположения ресурса), который имеет довольно простую структуру. Существуют различные схемы URL, но в большинстве случаев применяется такая:

префикс`://`*логин*`:`*пароль*`@`*адрес_сервера*`:`*порт*`/`*путь*`/`*файл*`?`*данные*`#`*якорь*

Здесь двоеточие, слэши (косые черты), `@`, `?` и `#` — специальные символы, разделяющие компоненты URL. Не все указанные компоненты URL обязательны для обращения к тому или иному ресурсу. Рассмотрим их назначение.

Префикс — это обычно краткое название сетевого протокола. Вот типичные примеры префиксов:

- `http` — протокол передачи гипертекста HTTP (HyperText Transfer Protocol), основной протокол службы WWW (World Wide Web — Всемирная паутина) для передачи (X)HTML-документов;

- `https` — протокол HTTPS для безопасного (с шифрованием) доступа к Web-ресурсам;
- `ftp` — протокол передачи файлов FTP (File Transfer Protocol); может применяться при перемещении любых файлов с одного компьютера на другой. Протокол FTP обеспечивает высокую надежность передачи файлов. К примеру, если потерю до 10% обычной текстовой информации еще можно пережить, то при передаче программ или сжатых архиватором файлов потери вообще недопустимы — неточно переданная программа просто не будет работать, а архивный файл не распакуется;
- `mailto` — указание на протокол передачи сообщений по электронной почте (служба Mail);
- `news` — протокол службы UseNet обеспечения телеконференций, это система типа доски объявлений, на которую вы можете поместить свое сообщение, и где можно прочитать объявления других участников телеконференции;
- `gopher` — протокол, предназначенный в первую очередь для работы неграфических браузеров, предоставляет систему доступа к информации, основанную на меню;
- `file` — означает, что указанный далее файл или папка находится на локальном компьютере.

За префиксом URL обязательно следует двоеточие. Если префикс отличен от `mailto`, то за двоеточием указывают два прямых слэша (две косые черты `//` без пробелов между ними). Например, `http://`, `mailto:`. Если в поле адреса браузера протокол не указан, то подразумевается HTTP, а если он опущен в ссылке, то имеется в виду протокол, по которому был загружен документ, содержащий данную ссылку.

Логин (регистрационное имя пользователя) и пароль, разделенные двоеточием, за которыми следует символ `@`, указывают, если они необходимы для доступа к ресурсу.

Далее идет адрес (доменное имя) сервера (хоста), записанный в системе DNS (Domain Name System — система доменных имен) или в виде IP-адреса (четыре целых числа, разделенные точками и находящиеся в диапазоне от 0 до 255).

Примеры

```
http://www.yandex.ru
http://192.168.2.103
http://localhost
ftp://ftp.narod.ru
mailto:myname@mail.ru
mailto:myname:12345678@yandex.ru
```

Если в поле адреса браузера адрес сервера не указан, то подразумевается локальный компьютер. Для файлов на локальном компьютере в URL можно указать префикс `file`. Поскольку в данном случае адрес сервера не требуется, то сразу же за `file://` через слэш записывается путь, например,

```
file:///C:/MyDoc/book1.html.
```

Таким образом, между `file:` и путем записывают три слэша.

К Web-серверу, установленному на локальном компьютере, можно обращаться по адресу `http://localhost`, или `http://127.0.0.0`, или еще короче: `localhost`.

За адресом сервера через двоеточие иногда следует порт — число от 1 до 65 535, указывающее серверу, какой программе передать на обработку данный запрос.

Пример:

```
http://anyserver.ru:80.
```

Если порт не указан, то для протокола HTTP предполагается 80, для HTTPS — 443, а в случае FTP — 21.

За номером порта, а если он не указан, то сразу же за адресом сервера, через прямой слэш указывается путь к файлу. Элементы пути (папки) разделяют прямыми слэшами, например,

```
http://anyserver.ru/folder1/folder2/folder3.
```

После пути к файлу через прямой слэш может быть указано имя файла, например,

```
http://anyserver.ru/folder1/folder2/folder3/mypage.html.
```

Если путь к файлу не указан, то при наличии адреса сервера подразумевается корневая папка сервера, а при отсутствии адреса сервера — папка последнего загруженного документа. Если не указано имя файла, то предполагается index.htm или index.html. Обычно главный документ сайта (т. е. документ, загружаемый первым) находится в файле с таким именем, чтобы можно было в URL-адресе сайта не указывать его, полагаясь на значение по умолчанию. Однако следует иметь в виду, что имя файла по умолчанию может быть изменено администратором сервера.

Если указанный файл является программой (серверным сценарием), то ему могут быть переданы данные (параметры). В этом случае данные отделяют от предшествующей части URL символом вопроса (`?`). Данные это просто текст. Если требуется передать набор из нескольких данных, то последние разделяют символом `&` (амперсанд). Часто элементы данных представляют в виде *`имя_параметра=значение`*. Например, в следующем URL файлу со скриптом на языке PHP передаются два параметра:

```
http://anyserver.ru/prg/myprogram.php?username=alex&pw=12345
```

Если данные в URL не указаны, то предполагается, что их нет.

Наконец, в URL можно указать якорь (метку внутренней ссылки), чтобы при загрузке документа браузер сразу же прокрутил его в своем окне, сделав помеченный фрагмент видимым. Для этого к концу URL через символ `#` добавляют текст метки, например,

```
http://anyserver.ru/docs/book1.html#chapter5
```

Если якорь не указан, то считается, что он не нужен.

7.5.2. Абсолютные и относительные пути

Пути к файлу могут быть абсолютными и относительными. Абсолютный путь описывает местоположение файла (папки), начиная с самого высокого уровня, и включает имена всех папок в иерархии их вложенности, ведущих к файлу (папке). На самом высоком уровне иерархии находится корневая папка сайта на Web-сервере. Ее имя обычно не указывается в URL.

Например, в URL `http://anyserver.ru/mypage.htm` указано, что файл mypage.htm находится в корневой папке сервера с доменным именем anyserver.ru, а в URL `http://anyserver.ru/docs/mypage.htm` указано, что файл mypage.htm находится в папке docs, которая расположена в корневой папке сервера. Ошибка в записи абсолютного пути (адреса) приводит к тому, что нужный файл не будет найден.

Относительный путь (адрес) описывает местоположение файла относительно места расположения текущего документа, в котором инициализируется ссылка с данным адресом. Так, если вы указываете просто имя файла myfile.htm без пути к нему, это означает, что задается относительный адрес. В данном случае браузер будет искать его в той же папке, где находится текущий документ. Если перед именем файла поставить `../` (например, `../myfile.htm`), то браузер будет искать файл в папке, находящейся на один уровень выше, чем тот, в котором находится текущий документ. Аналогично, если перед именем файла поставить `../../` (например, `../../myfile.htm`), то браузер будет искать файл в папке на два уровня выше, чем текущий.

Допустим, в корневой папке вашего сайта на сервере находятся папки images и docs для хранения файлов соответственно графических изображений и (X)HTML-документов. Тогда относительный адрес ссылки из любого файла в папке docs на изображение в файле, например, img1.jpg записывается так: `..//images/img1.jpg`. В то же время адрес ссылки на это же изображение, но из файла в корневой папке имеет такой вид: `images/img1.jpg`.

7.5.3. Псевдо-URL JavaScript

Иногда требуется, чтобы при активизации ссылки происходил вызов сценария на языке JavaScript, а не просто загрузка документа. В этом случае в значении атрибута `href` тега `<a>` следует указать префикс `"javascript:"`, за которым следует код сценария. Такую конструкцию еще называют псевдо-URL JavaScript.

Вот пример:

```
<a href="javascript:myfunction()/">Новости </a>.
```

Это нестандартная, но широко поддерживаемая конструкция.

ГЛАВА 8

Форматирование текстов

Тексты являются основной формой представления информации на Web-страницах. Поэтому их форматированию уделяют большое внимание. Тексты могут быть внесены в (X)HTML-документ непосредственно, т. е. без использования каких бы то ни было тегов вставки или форматирования. При отображении такого текста браузер проигнорирует невидимые служебные символы перехода на другую строку, табуляции и длинные последовательности пробелов. Например, вы можете ввести несколько строк текста, а браузер отобразит их как одну строку; несколько пробелов, введенных вами для обозначения отступа (красной строки), браузер заменит одним пробелом. Иначе говоря, форматирование текста средствами текстового редактора не соответствует форматированию этого текста браузером. Как уже упоминалось, управлять разметкой текста в окне браузера можно специальными тегами и атрибутами, которых достаточно много в HTML. Так, для перевода строки предназначен тег `<br/>`, для обозначения абзаца — `<p>`, для задания заголовков шести различных уровней служат контейнерные теги `<h1>`, `<h2>`,...,`<h6>`.

Что нужно сделать, чтобы оформить текст? Очевидно, необходимо выбрать подходящий шрифт (гарнитуру, размер, цвет и др.), выделить в нем абзацы, заголовки, задать отступы и т. п. Как это реализовать с помощью тегов и параметров CSS, мы и рассмотрим в данной главе.

8.1. Шрифты

Если в (X)HTML-документе не задавать параметры шрифтов, то браузер отобразит текст, установив общедоступные шрифты и их параметры, принятые по умолчанию или заданные в настройках. Поскольку пользователи могут по-разному настроить свои браузеры, ваша Web-страница может выглядеть совсем не так, как вы рассчитывали. Чтобы достичь инвариантности оформления текста относительно браузеров, необходимо самому задать шрифты с помощью CSS. Однако при этом не следует выбирать слишком редкие (экзотические) шрифты, которых может не оказаться на компьютере пользователя.

Каждый шрифт имеет свое название и отличается характером рисунка символов, их размером, насыщенностью и другими параметрами. Кажется естественным идентифицировать шрифт по имени, которое представляет собой одно или несколько слов,

разделенных пробелами, например, BT Swiss 721 Heavy Italic. Однако в настоящее время не существует хорошо разработанной и общедоступной системы классификации шрифтов по именам и, более того, термины, употребляемые для одной гарнитуры шрифтов, могут не подойти для другой. Например, термин "курсив" (italic) обычно обозначает наклонный текст, которому также соответствуют англоязычные термины *Oblique*, *Slanted*, *Incline*, *Cursive* или *Kursiv*. Аналогично, имя шрифта может содержать термины, описывающие его вес — яркость или насыщенность (жирность) отображения символов. Однако до сих пор термины, обозначающие вес шрифтов, не имеют общепринятого значения, а их использование варьируется в очень широком диапазоне. Например, шрифт, который, по вашему предположению, должен быть жирным, может оказаться нормальным (Regular, Roman, Book или Medium), полужирным (Semi-Bold или Demi-Bold), жирным (Bold) или сверхнасыщенным (Black) в зависимости от того, насколько темным был задан "нормальный" тип начертания шрифта при его разработке. Отсутствие систематически построенных наименований в общем случае делает невозможным создание модифицируемого названия типа начертания шрифта, имеющего специфические отличительные черты, такие как, например, насыщенность.

По этой причине в CSS принята следующая модель. Шрифты запрашиваются не только по своему имени, но и по совокупности заданных для них свойств. Значения этих свойств формируют основу для выбора шрифта агентом пользователя (в частности, браузером). Изменение значения каждого свойства шрифта может осуществляться индивидуально, например, можно отдельно увеличить вес. После этого для выбора шрифта из базы данных будет использоваться новая совокупность значений его свойств.

Теперь рассмотрим параметры CSS, с помощью которых можно задать характеристики шрифтов.

- *Гарнитура шрифта* — семейство шрифтов, обладающих некоторым сходством. Другими словами, гарнитура — нечто общее между конкретными шрифтами, имеющими собственные названия. Один шрифт гарнитуры может быть курсивным, второй полужирным, третий может быть сжатым или состоять из малых прописных букв. Примеры названий гарнитур: Arial, Times, Helvetica, New Century Schoolbook и Kyokasho ICA L. Гарнитуры шрифтов могут быть сгруппированы в различные категории: с засечками (например, Times New Roman, Serif) или без засечек, называемые еще рублеными (например, Arial, Sans-serif); гарнитуры, символы которых располагаются друг относительно друга на одинаковом (моноширинные) или различном расстоянии; гарнитуры, тип начертания символов в которых напоминает рукописный и др.

  Гарнитуры задаются параметром `font-family`. Данный параметр (свойство) принимает в качестве значения список имен гарнитур или шрифтов в порядке их приоритета. Обычно шрифты в списке имеют одинаковый или близкий стиль начертания; последним указывают имя похожей общедоступной гарнитуры. Для Web-документов полезно указывать несколько гарнитур или имен шрифтов, тогда браузер будет пытаться отобразить текст в соответствии с первой гарнитурой или шрифтом в списке, а если не получится, то в соответствии со второй и т. д.

  Пример: `font-family: Arial, Helvetica, sans-serif`.

  Если имя гарнитуры состоит из нескольких слов, то его обязательно следует заключать в кавычки:

```
font-family: "Times New Roman", Helvetica, serif.
```

Однословные имена гарнитур (кроме общедоступных serif, sans-serif, cursive, fantasy, monospace) также рекомендуется записывать в кавычках.

Типичные названия гарнитур для Web-документов: Times и Serif (шрифты с засечками), Arial и Sans-serif (без засечек или рубленые), Courier или Monospase (моноширинный), Cursive (курсив, наклонный). Гарнитуры типа Courier или Monospase применяют для выделения внутристрочных фрагментов текста и кодов программ, а остальные — для отображения обычного текста и заголовков в зависимости от вкуса дизайнера. Кроме того, часто встречаются шрифты Helvetica и Verdana — рубленые или без засечек. Шрифты без засечек хороши, как мне кажется, если их размер мал (менее 12 pt); шрифты с засечками предпочтительнее для заголовков и обычного текста с размером шрифта 12—16 pt. Однако окончательный выбор гарнитуры остается за автором-дизайнером.

На рис. 8.1 показаны некоторые гарнитуры.

Рис. 8.1. Примеры некоторых гарнитур

- *Стиль шрифта* — параметры `font-style`, `font-variant`, `font-weight`, `font-stretch`.
  - `font-style` — определяет нормальный (`normal`), курсивный (`italic`) или наклонный (`oblique`) тип начертания символов шрифта одной и той же гарнитуры. Практически для всех основных шрифтов в Web-дизайне значения `italic` и `oblique` дают одинаковый результат — наклонное начертание;
  - `font-variant` — вариант стиля шрифта. Принимает значения: `normal` (по умолчанию) и `small-caps` (капитель). Значение `normal` не влияет на отображение шрифта, а `small-caps` заменяет строчные буквы прописными, но несколько меньшими по размеру, чем прописные буквы текущего шрифта (рис. 8.2);

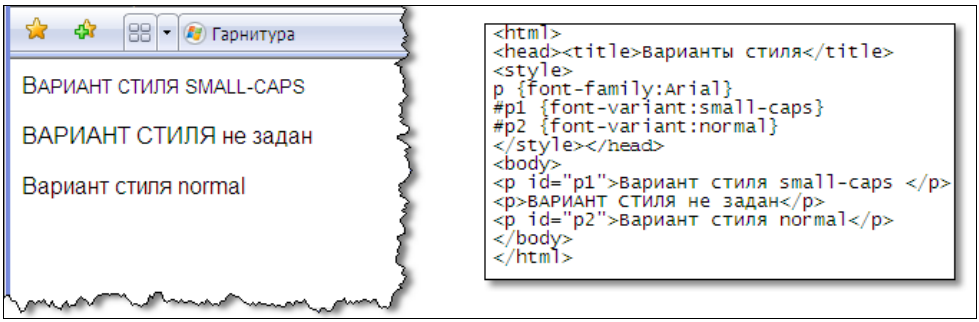


Рис. 8.2. Варианты стиля шрифта

- `font-weight` — вес (насыщенность) шрифта. Значения задаются ключевыми словами `normal` (обычный), `bold` (полужирный), `bolder` (жирнее), `lighter` (светлее, т. е. менее насыщенный) или одним из девяти целых чисел от 100 до 900. Значение `normal` эквивалентно 400, а `bold` — 700. Значение `bolder` (`lighter`) задает вес шрифта, больший (меньший), чем унаследованный им вес. Следует иметь в виду, что в текущем семействе может и не быть шрифта с заданной степенью насыщенности. В этом случае можно лишь гарантировать, что шрифт с бóльшим значением `font-weight` будет не светлее шрифта с меньшим значением этого параметра;
- `font-stretch` — позволяет выбрать из гарнитуры шрифтов нормальное (`normal`), сжатое (`condensed`) или растянутое (`extended`) начертание. Принимает значения: `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `normal`, `semi-expanded`, `expanded`, `extra-expanded`, `ultra-expanded`. Кроме того, возможны значения `wider` (более широкий) и `narrower` (более узкий).

❒ *Размер шрифта* — параметр `font-size`. Данный параметр принимает значения в абсолютных единицах измерения (например, pt, px), в процентах от размера родительского шрифта, а также в виде ключевых слов: `xx-small`, `x-small`, `small` (малый), `medium` (средний), `large` (большой), `x-large`, `xx-large`, `larger` (больше), `smaller` (меньше). Последние два значения задают размер, бóльший или меньший размера родительского шрифта соответственно. На рис. 8.3 показаны примеры задания размера шрифта.

❒ *Сокращенная запись свойств шрифта* — параметр `font`. Значение данного параметра представляет собой строку частных параметров шрифта: `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height` и `font-family`. Частные значения записываются в указанном порядке через пробел. Первые три значения, устанавливаемые по умолчанию в `normal`, можно опустить. Если задается высота строки (`line-height`), то ее значение отделяется от размера шрифта (`font-size`) прямым слэшем (/). Если в списке гарнитуры (`font-family`) содержится более одного элемента, то последние разделяются запятыми.

Примеры:

```
p {font: italic small-caps bold 12pt/105% "Times", serif}
p {font: bold 12pt "Arial", serif}
p {font: 10pt/12pt sans-serif}
```

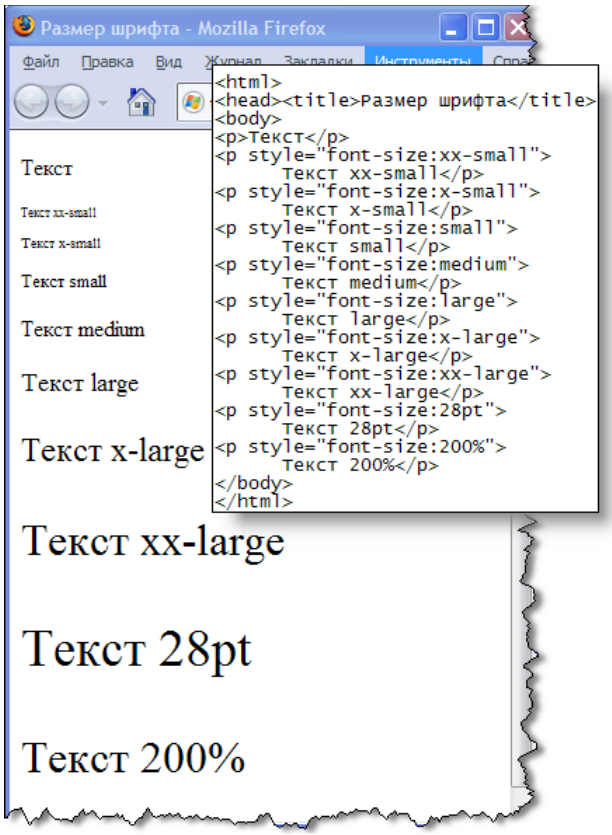


Рис. 8.3. Размеры шрифта

Параметры шрифта наследуются. Поэтому основные из них можно привязать к телу документа (элементу `<body>`).

8.2. Основные теги разметки текстов

Как уже отмечалось, для задания заголовков шести различных уровней служат контейнерные теги `<h1>`, `<h2>`, ..., `<h6>`, для обозначения абзаца — контейнерный тег `<p>`. Для управления переходом на другую строку предназначен тег `<br/>`, который указывает место перехода на другую строку; данный тег может использоваться не только для переноса текстовых строк, но и для других элементов, например `<img>`, `<div>`, `<span>`, чтобы обеспечить позиционирование этих элементов в нормальном потоке с новой строки; в отличие от тега `<p>` абзаца тег `<br/>` не вставляет пустую строку перед следующим элементом.

В тексте нередко требуется выделить некоторые фрагменты. Для этого имеются специальные контейнерные теги:

- `<span>` — отображает текст обычным прямым шрифтом;
- `<em>` — выводит текст курсивом;
- `<address>` — служит для представления сведений об авторе, дате создания и обновления документа; текст внутри этого контейнера показывается обычным шрифтом;

- `<kbd>` — обычно отображает мелким моноширинным шрифтом текст, вводимый пользователем;
- `<code>` — обычно применяется для отображения мелким моноширинным шрифтом текста, содержащего программный код. Если внутри этого тега находится контейнер `<script>`, содержащий сценарий, то сценарий будет выполнен, но не отображен;
- `<samp>` — выводит мелким моноширинным шрифтом текст, содержащий пример программного кода. Если внутри этого тега находится контейнер `<script>`, включающий сценарий, то этот сценарий будет выполнен, но не отображен;
- `<var>` — служит для отображения имен переменных мелким моноширинным шрифтом;
- `<cite>` — обычно применяется для отображения текста (например, цитат) курсивом;
- `<blockquote>` — выводит текст с отступом (со сдвигом вправо).

Текст, заключенный в перечисленные теги, отображается так или иначе в соответствии с правилами, принятыми по умолчанию (рис. 8.4). Вместе с тем, ничто не мешает вам задать свой собственный стиль вывода содержимого для каждого из данных тегов.

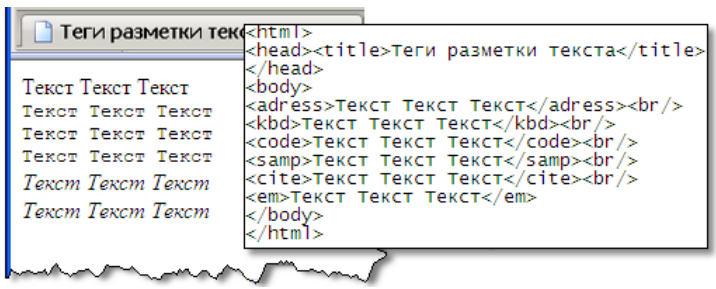


Рис. 8.4. Примеры использования специальных тегов разметки текста

Выделить текст полужирным шрифтом и курсивом можно контейнерными тегами `<b>` и `<i>` соответственно. В свое время данные теги были придуманы для поддержки форматирования текста. Теперь форматирование следует преимущественно осуществлять с помощью CSS, а от тегов и атрибутов, предназначенных в основном для определения визуализации, рекомендуется отказаться. Однако теги для разметки текста все-таки нужны. Главная их функция — выделить фрагмент текста как элемент структуры документа. Так, например, применяя тег `<cite>` для выделения цитат, вы можете затем из всего текста документа выделить и отобразить только цитаты. Подобные операции выполняют обычно скрипты, но для обеспечения их принципиальной возможности необходима структурная разметка документа специальными тегами. Разумеется, вы можете не полагаться на принятое по умолчанию отображение содержимого этих тегов, а просто задать для них нужные вам параметры CSS, например, указать в таблице стилей, что содержимое тега `<address>` должно отображаться полужирным курсивом размером 10 pt:

```
address { font: italic bold 10pt Times}
```

8.3. Специальные символы

При формировании (X)HTML-документа может потребоваться ввести символы, которые воспринимаются браузером как служебные. Например, символы < и > для обозначения знаков "меньше" и "больше" недопустимы, т. к. они интерпретируются как символы тегов. Кавычки (") и амперсанд (&) также применяются в языке HTML для служебных целей. Все символы, которые можно набрать при нажатых клавишах управления, отличных от <Shift>, могут не воспроизводиться при просмотре (X)HTML-документа, поскольку они зарезервированы. Кроме того, несколько пробелов, введенных подряд, браузер воспримет как один символ (проблема создания отступов с помощью символа пробела). В этих случаях можно прибегнуть к буквенным или числовым эквивалентам символов. Последние однозначно соответствуют их кодам ASCII.

Формат буквенного эквивалента символа:

```
&буквенный_эквивалент;
```

Например, `<` обозначает символ < (от англ. *less then* — меньше чем).

Формат числового эквивалента:

```
&#код_ASCII;
```

Например, символ < можно вставить в документ как `<`. Символ пробела имеет буквенный эквивалент ` ` и числовой — ` `.

Таким образом, эквиваленты заключаются между символами `&` и `;`. В случае цифрового эквивалента перед ASCII-кодом символа следует поставить символ `#`.

На рис. 8.5 показано соответствие между часто встречающимися символами, их числовыми и буквенными эквивалентами.

"	34	quote	±	177	plusmn
&	38	amp	µ	181	micro
<	60	lt	¶	182	para
>	62	gt	¼	188	frac14
	160	nbsp	½	189	frac12
¢	162	cent	¾	190	frac34
£	163	pound	Æ	198	AElig
§	167	sect	Ø	216	Oslash
©	169	copy	æ	230	aelig
®	174	reg	÷	247	divide

Рис. 8.5. Соответствие между некоторыми символами и их числовыми и буквенными эквивалентами

8.4. Работа с текстом

Для выделения абзацев, как уже неоднократно говорилось, предназначен контейнерный блоковый тег <p>. Текст внутри данного тега отображается с новой строки без отступа (т. е. без красной строки) во всю ширину клиентской части окна браузера. Если

окно браузера изменить, то содержимое абзаца снова распределится по всей ширине окна.

Вы можете задать параметры размеров (`width` и `height`), позиционирования (`top`, `left`, `position`) и отступов (`padding`) для элемента `<p>`, чтобы повлиять на расположение текста в абзаце. Разумеется, эти же параметры можно применить и к другим контейнерным тегам, содержащим текст, например `<div>`, `<span>`, `<em>` и т. д.

Цвет текста, заключенного в какой-нибудь контейнерный тег, задается параметрами CSS для данного контейнера. Так, цвет символов определяется параметром `color`, а фона — `background` или `background-color`.

Далее мы рассмотрим параметры CSS, влияющие на визуальное представление символов, слов и абзацев.

8.4.1. Красная строка

Для задания отступа первой строки текста в блоке (так называемой красной строки) служит параметр `text-indent`, принимающий в качестве значения величину отступа с указанием единиц измерения; проценты выражают величину отступа относительно ширины содержащего блока (например, `<div>`, `<body>`).

Примеры

```
p {text-indent:50px}
p {text-indent:10pt}
p {text-indent:10em}
p {text-indent:10%}
```

При отсутствии параметра `text-indent` отступ типа красной строки пришлось бы создавать последовательностью из нескольких специальных символов пробела ` ` или ` ` (см. *разд. 8.3*):

```
<p>         Текст текст текст текст</p>
```

8.4.2. Выравнивание

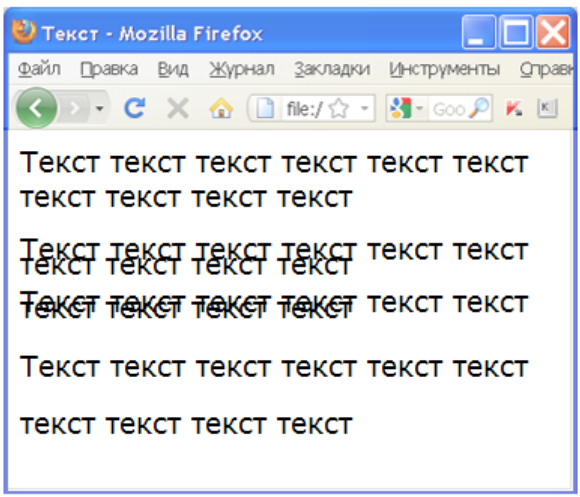
Для горизонтального выравнивания текста (любого последовательного содержимого блока) используется параметр `text-align`, принимающий значения: `left` (по левому краю), `right` (по правому краю), `center` (по центру) и `justify` (одновременно по обоим краям).

Для вертикального выравнивания текста (и не только) служит параметр `vertical-align`, принимающий значения `top` (по верхнему краю), `bottom` (по нижнему краю), `middle` (посередине) и `baseline` (по базовой линии) и др. (см. *разд. 8.4.7*).

8.4.3. Межстрочное расстояние

Расстояние между строками (интерлиньяж) можно назначить с помощью параметра `line-height`. Точнее говоря, он определяет высоту строки, задаваемую ключевым словом `normal` (значение по умолчанию) или числом с указанием единиц измерения, включая проценты.

При задании процентов высота строки вычисляется относительно размера шрифта. Вообще говоря, значения данного параметра интерпретируются различными браузерами не совсем одинаково. Точную картину можно выяснить экспериментально. На рис. 8.6 показаны примеры, иллюстрирующие в некоторой мере влияние данного параметра на межстрочное расстояние. На практике параметр `line-height` со значением, отличным от `normal`, применяют в исключительных случаях, например, когда требуется разместить текст в контейнере с очень ограниченными размерами (на кнопке).




```
<style>
p {font-family:Verdana;
   font-size:14pt;
}
#p2 {line-height:10px}
#p3 {line-height:20%}
#p4 {line-height:200%}
</style>
<body>
<p id="p1">Текст текст текст текст
текст текст текст текст текст текст
</p>
<p id="p2">Текст текст текст текст
текст текст текст текст текст текст
</p>
<p id="p3">Текст текст текст текст
текст текст текст текст текст текст
</p>
<p id="p4">Текст текст текст текст
текст текст текст текст текст текст
</p>
</body>
```

Рис. 8.6. Влияние параметра `line-height` на межстрочный интервал

8.4.4. Межсловное расстояние

Расстояние между словами можно установить с помощью параметра `word-spacing`, который принимает следующие значения:

- `normal` — расстояние между словами обычное, как при отсутствии параметра;
- *значение* — число с указанием единиц измерения.

Если установлен параметр `text-align:justify` (выравнивание одновременно по левому и правому краям), то `word-spacing` не действует. В этом случае межсловное расстояние устанавливается автоматически исходя из цели выравнивания текста. Нулевое значение соответствует обычному расстоянию меду словами, а при отрицательных значениях результат зависит от браузера.

Так, Internet Explorer и Opera в последнем случае отображают слова слитно, а остальные браузеры — с перекрытием. По существу значение `word-spacing` означает приращение длины пробела, принятой по умолчанию.

На рис. 8.7 показан пример форматирования текстовой строки с помощью параметра `word-spacing`.

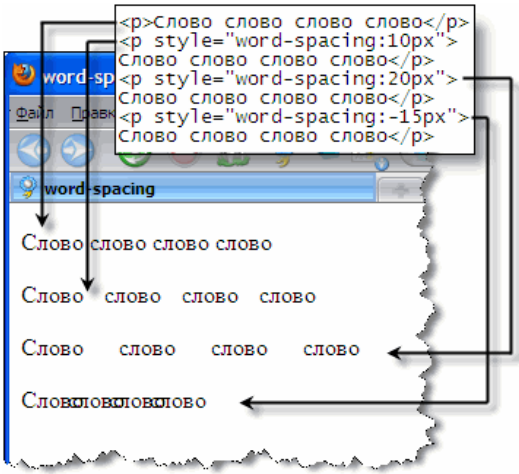


Рис. 8.7. Влияние параметра `word-spacing` на расстояние между словами

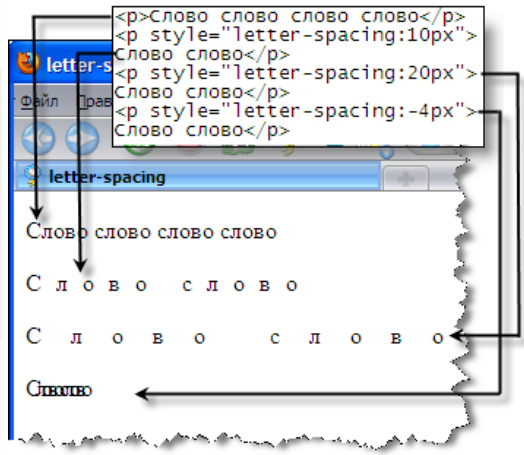


Рис. 8.8. Влияние параметра `letter-spacing` на расстояние между буквами

8.4.5. Межбуквенное расстояние

Расстояние между буквами регулируется параметром `letter-spacing`, который принимает следующие значения:

- `normal` — расстояние между словами обычное, как при отсутствии данного параметра;
- *значение* — число с указанием единиц измерения.

Без данного параметра межбуквенное расстояние устанавливается автоматически на основе характеристик шрифта. С помощью `letter-spacing` можно представить слово или блок текста вразрядку или, наоборот, в сжатом виде. При этом межсловное расстояние изменяется пропорционально межбуквенному. Нулевое значение соответствует обычному расстоянию меду буквами, а при отрицательных значениях параметра строка может стать нечитабельной.

На рис. 8.8 показан пример форматирования текстовой строки с помощью параметра `letter-spacing`.

8.4.6. Декорация

Текстовые строки можно оформить специальным образом или, как еще говорят, декорировать. Например, их можно или подчеркнуть снизу или сверху, перевести в верхний или нижний регистры, или зачеркнуть. Иногда требуется, чтобы каждое слово в строке начиналось с прописной буквы. Для такой декорации текста применяется параметр `text-decoration`, который принимает следующие значения:

- `none` — отменяет любое декорирование (значение по умолчанию для большинства тегов); в частности, применение данного значения к тегу ссылки `<a>` отменяет принятое по умолчанию подчеркивание текста или отображение рамок около графического изображения ссылок;
- `underline` — подчеркивает текст;

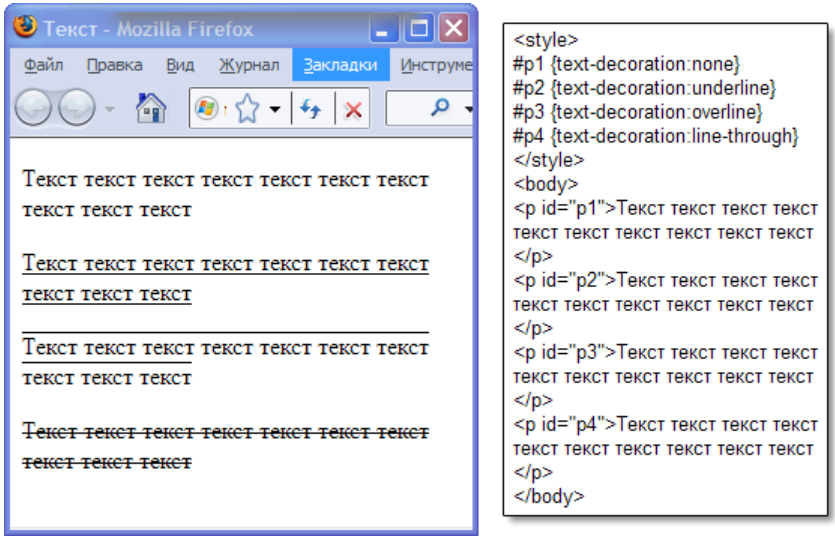


Рис. 8.9. Подчеркивание, надчеркивание и зачеркивание текста с помощью параметра `text-decoration`

- `overline` — добавляет черту над текстом;
- `line-through` — зачеркивает текст;
- `lowercase` — отображает текст в нижнем регистре (строчными символами);
- `uppercase` — выводит текст в верхнем регистре (прописными символами);
- `blink` — создает эффект мигания (в Firefox и Opera).

На рис. 8.9 приведен пример для некоторых значений параметра `text-decoration`.

На рис. 8.10 приведен пример создания простой математической формулы с помощью параметра `text-decoration`. Здесь квадратный корень отображается специальным символом `√`, а линии прорисованы посредством установки соответствующих значений параметра `text-decoration`.

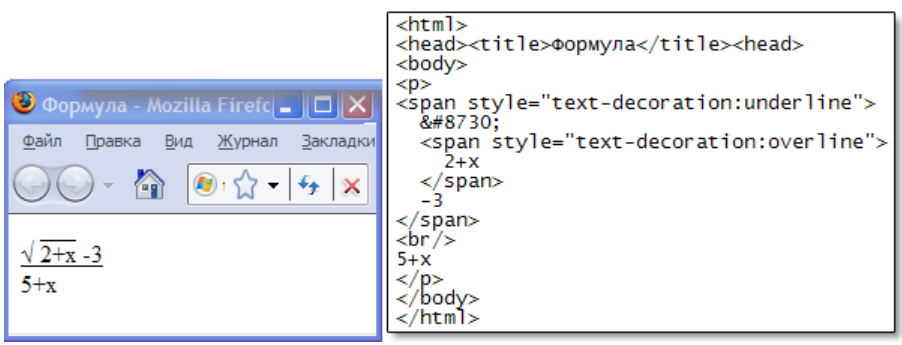


Рис. 8.10. Математическая формула, созданная посредством параметра `text-decoration`

8.4.7. Индексы

Иногда требуется, чтобы текст располагался в той же строке, что и основной, но немного выше или ниже, например, ссылки на подстрочные замечания (цифры или звездочки) или номера переменных в математических формулах. Такие тексты, смещенные вверх или вниз относительно основной строки, называют верхними или нижними индексами.

В HTML для отображения текста в виде индексов предусмотрены контейнерные теги `<sup>` и `<sub>` соответственно для верхних и нижних индексов. Очевидно, главное назначение данных тегов в том, чтобы обеспечить определенное отображение их содержимого. Но визуализацию содержимого лучше задавать посредством CSS, а не тегами или их атрибутами.

В CSS есть параметр `vertical-align`, который позволяет решить различные задачи вертикального выравнивания, но здесь мы рассмотрим лишь создание верхних и нижних индексов посредством значений `super` и `sub` соответственно (рис. 8.11). Возможно, размеры индексов следовало бы немного уменьшить. Для этого достаточно добавить к ним стилевой параметр, задающий размер шрифта.

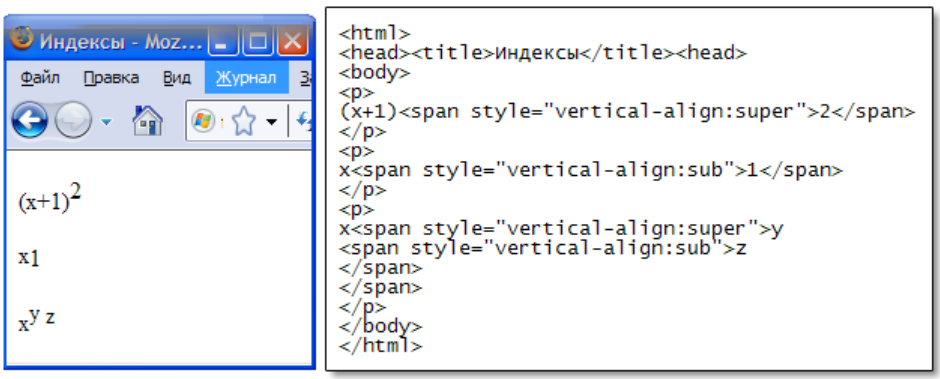


Рис. 8.11. Создание индексов

8.4.8. Выделение первой буквы строки и первой строки в блоке текста

Чтобы выделить первую букву строки в блоке текста (сделать так называемую буквицу), достаточно задействовать псевдокласс `:first-letter`. Аналогично, для выделения первой строки служит псевдокласс `:first-line`. Данные псевдоклассы применяются к блоковым (а не внутристрочным) элементам, таким как `<p>`, `<div>`, `<body>`.

На рис. 8.12 показан пример создания буквицы для абзаца и выделения подчеркиванием первой строки в блоке `<div>`.

8.4.9. Текст с тенью

Текстовые заголовки с эффектом тени, создающим впечатление объемности, чаще всего выполняют средствами графического редактора и вставляют в Web-документ как

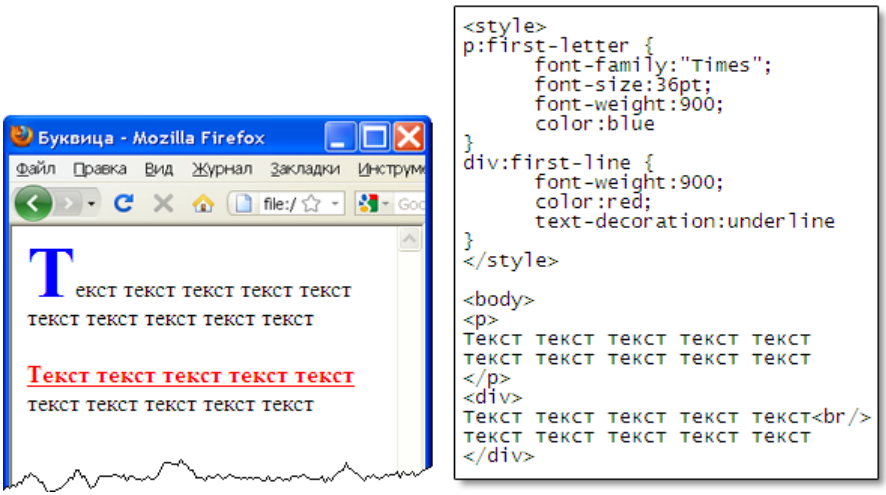


Рис. 8.12. Пример буквицы и выделения первой строки

картинку. Вместе с тем, аналогичный результат можно получить с помощью CSS. Для этого необходимо наложить текст одного цвета на такой же текст другого цвета с некоторым сдвигом по вертикали и горизонтали. Лучшего эффекта можно достичь наложением друг на друга трех идентичных по содержанию текстов, чтобы создать иллюзию не только тени, но и подсветки краев букв.

На рис. 8.13 показан пример создания объемного заголовка только с тенью (без подсветки). Здесь два экземпляра текста заголовка помещаются в контейнеры `<div>` с одинаковыми значениями атрибута `id`, чтобы они отличались от блоков `<div>`, предназначенных в документе для иных целей (в примере `id="head"`). Первый экземпляр — перед-

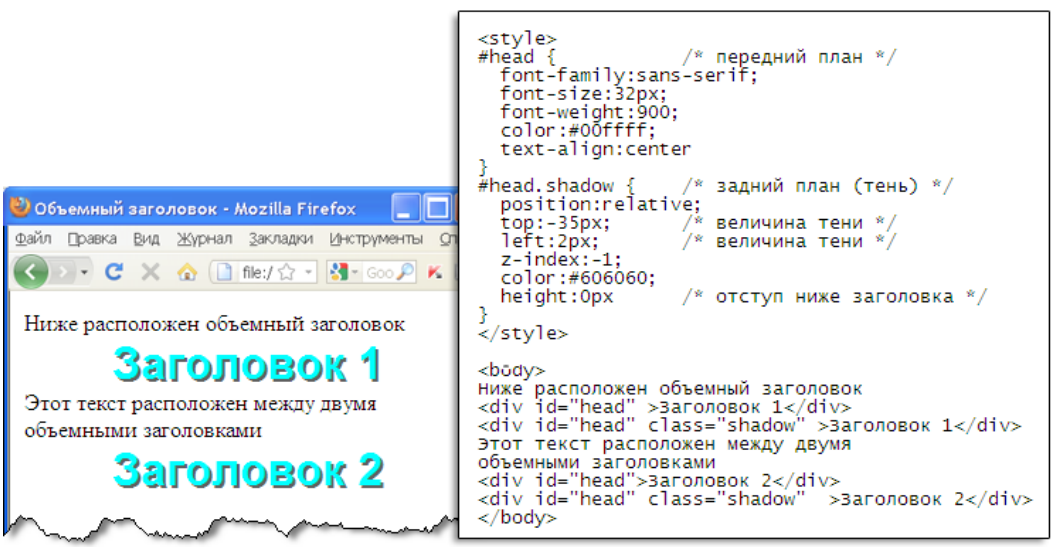


Рис. 8.13. Пример создания объемных заголовков

ний план заголовка, а второй — его тень. Второй экземпляр позиционируется с некоторым сдвигом относительно своего положения в нормальном потоке и перемещается на задний план с помощью параметра `z-index:-1`. Параметры сдвига (`top` и `left`) подбираются экспериментально в зависимости от установленного шрифта (`font-family`), его размера (`font-size`) и веса (`font-weight`). Поскольку при относительном позиционировании (`position:relative`) элемент не изымается из нормального потока (т. е. за ним резервируется прежнее место), то за созданным объемным заголовком снизу получается слишком большой отступ. Его можно отрегулировать с помощью параметра `height` (в рассматриваемом примере `height:0`).

В CSS 2 для создания эффекта тени для текста предусмотрен параметр `text-shadow`, который поддерживают все современные браузеры, кроме Internet Explorer 9 и более ранних версий. Последний просто не создает эффекта тени. Браузер Internet Explorer поддерживает расширение css от Microsoft — так называемые графические фильтры, среди которых имеются `dropShadow` и `shadow`, позволяющие создать эффект тени (см. *разд. 5.3*).

Параметр `text-shadow` принимает значения `none` (нет эффекта тени), `inherit` (эффект тени наследуется), а также список эффектов, разделенных запятыми. Эффект в списке содержит параметры, перечисляемые через пробел: цвет, смещение по горизонтали, смещение по вертикали и радиус рассеяния тени. Параметры эффекта можно опускать, полагаясь на значения по умолчанию, а также указывать цвет тени после числовых параметров с единицами измерения. По умолчанию цвет тени определяется цветом самого текста. Смещения могут быть как положительными, так и отрицательными. Если в списке указано несколько эффектов, то последние накладываются друг на друга по

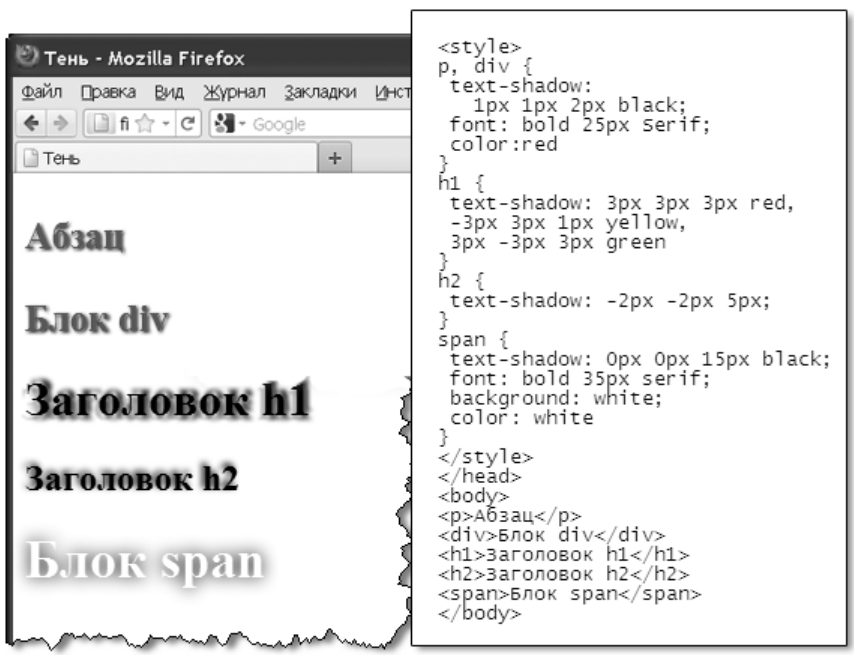


Рис. 8.14. Применение параметра `text-shadow`

некоторому алгоритму, который не специфицирован в CSS 2, и, следовательно, в различных браузерах результирующие эффекты могут отличаться. На рис. 8.14 показан пример использования параметра `text-shadow`.

8.4.10. Преобразование регистра

Для преобразования регистра отображения текста служит параметр `text-transform`, который принимает следующие значения:

- `none` — преобразование не производится;
- `capitalize` — каждое слово в строке начинается с прописной (заглавной) буквы;
- `lowercase` — все символы текста отображаются как строчные (в нижнем регистре);
- `uppercase` — все символы текста отображаются как прописные (в верхнем регистре).

8.4.11. Мультиколоночная верстка

Расположение текста в нескольких колонках можно обеспечить с помощью разметки страницы табличными тегами или тегами `<div>`. Однако при этом текст и другое содержимое придется распределять между колонками вручную. Нередко требуется, чтобы текст автоматически отображался в одной или нескольких колонках в зависимости от ширины окна браузера или содержащего его контейнера. Для решения данной задачи в CSS 3 предлагается ряд параметров:

- `columns` — определяет ширину и количество колонок (имеются и отдельные параметры `column-width` и `column-count`);
- `column-gap` — определяет ширину промежутков между колонками;
- `column-rule` — определяет ширину, стиль и цвет рамок между колонками (имеются и отдельные параметры `column-rule-width`, `coumn-rule-style` и `column-rule-color`).

Пример

```
body {columns:2 15em;
      column-gap:1em;
      column-rule:thin solid blue
}
```

Для настройки колонок есть и другие параметры (**www.w3.org/TR/css3-multicol**). В настоящее время они поддерживаются, хотя и по-разному, браузерами Opera, а также Safari и Chrome, если к именам параметров добавить префиксы `-webkit-`.

8.5. Предварительно отформатированный текст

Иногда в (X)HTML-документ необходимо включить фрагмент текста, предварительно отформатированный обычными средствами текстового редактора, т. е. с использованием пробелов, символов перевода строки, табуляции и т. п. Для этого предусмотрен контейнерный тег `<pre>`. В примере на рис. 8.15 текст, заключенный в контейнерный тег `<pre>`, представляет собой таблицу, построенную с помощью символов перевода строки

и табуляции. Если обычно браузер игнорирует эти символы, то в случае применения тега <pre> он их учитывает при разметке текста. Текст внутри тега <pre> выводится моноширинным шрифтом. Разумеется, с помощью таблицы стилей можно добавить дополнительные параметры форматирования, такие как размер, толщина, имя шрифта и др.

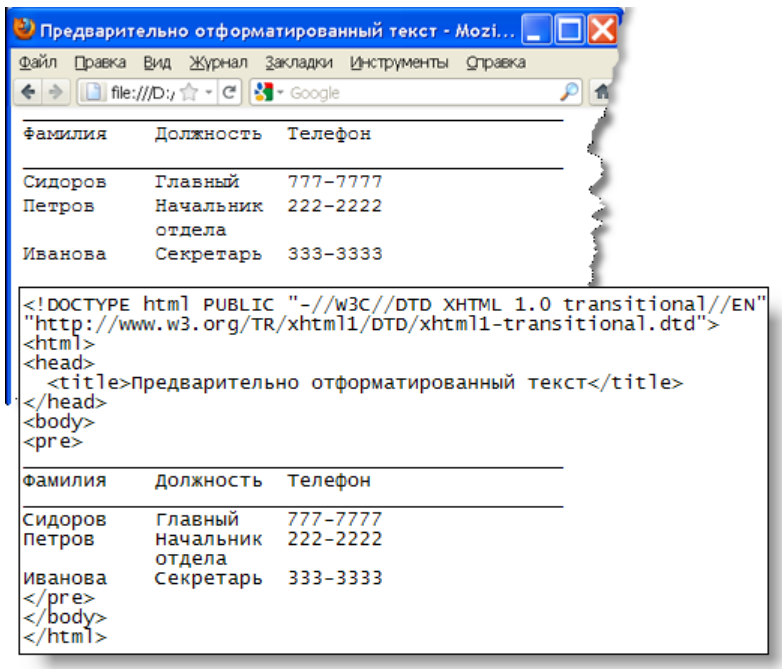


Рис. 8.15. Пример использования тега <pre>

Для отображения предварительно отформатированного текста можно также применять параметр CSS white-space, который принимает следующие значения:

- normal — текст выводится как обычно;
- nowrap — текст выводится одной строкой без переносов, если только не использовался тег <br/>;
- pre — текст выводится с учетом всех пробелов и других служебных символов, как он был набран в текстовом редакторе.

В отличие от тега <pre>, текст с параметром white-space выводится не обязательно моноширинным шрифтом.

8.6. Генерируемое содержимое

С помощью CSS можно добавлять к элементам некоторое содержимое, изначально отсутствующее в теле документа. Например, заголовки глав книги не обязательно начинать со слова "Глава". Это слово, как и номер главы, можно сгенерировать и добавить к содержимому элемента, например <h1>, при отображении документа.

Содержимое, располагающееся до и после элемента, добавляется с помощью псевдоклассов :before и :after соответственно, а его определение — с помощью параметра content. Например, если вы хотите добавить перед всеми заголовками первого уровня (элементами <h1>) слово "Раздел", то достаточно использовать такое правило:

```
h1:before {content:"Раздел"}
```

Параметр content, поддерживаемый всеми современными браузерами, в том числе и Internet Explorer 8.0, принимает следующие значения:

- *строка* — добавляемый текст, заключенный в кавычки; специальные символы представляются шестнадцатеричным кодом Unicode с обратным слэшем в качестве префикса; например, символ параграфа (§) представляется как \A7, символ доллара ($) — как \24;
- url(*адрес*) — URL-адрес вставляемого объекта; например,

  ```
  h1 {content:url("picture.jpg")}
  ```

  если браузер не может отобразить объект, то данный параметр просто игнорируется;
- attr(*атрибут*) — значение указанного атрибута элемента, к которому оно добавляется; например, для добавления к картинке ее адреса можно использовать правило img:after{attr(href)};
- open-quote — добавляет открывающую кавычку, тип которой устанавливается с помощью параметра quotes; например,

  ```
  p:before {content:open-quote; quote: "<"}
  ```
- close-quote — добавляет закрывающую кавычку, тип которой устанавливается с помощью параметра quotes;
- no-open-quotes — отменяет добавление открывающей кавычки (применяется для тех элементов, которые автоматически добавляют кавычки к содержащемуся внутри них тексту);
- no-close-quote — отменяет добавление закрывающей кавычки.

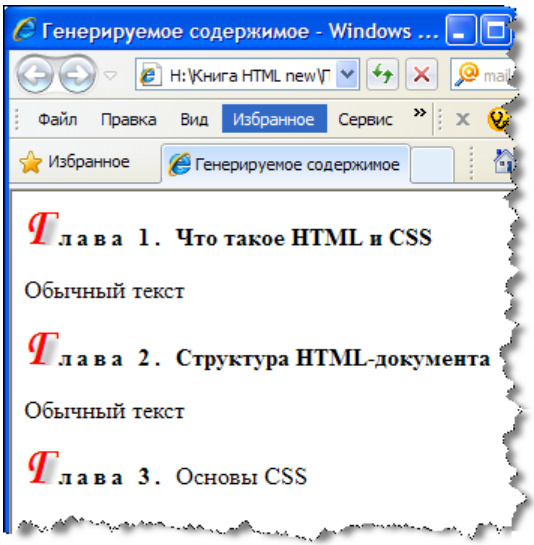
В качестве значения параметра content можно указать сразу несколько из перечисленных, разделяя их пробелами. Кроме того, параметр content может принимать в качестве значения функцию counters() вычисления значения счетчика для автоматической нумерации элементов, но об этом далее.

На рис. 8.16 показан пример, в котором перед элементами <h1> добавляется генерируемое содержимое, а именно картинка с изображением буквы "Г" из файла g.jpg, текст "лава" и вычисляемый номер элемента, за которым следует точка и пробел:

```
h1:before {
  content: url("g.jpg") "лава" counters(item, "") ". ";
  counter-increment:item
}
```

Номер главы (элемента <h1>) здесь вычисляется следующим образом:

- counters(item, "") — текущее значение счетчика с идентификатором item (назначается произвольно); второй параметр задает разделитель при нумерации иерархических списков и здесь не играет никакой роли;




```
<html>
<head>
<title>Генерируемое содержимое</title>
<style>
div {
 counter-reset:item
}
h1 {font-size:12pt}
h1:before {
  letter-spacing:3px;
  content:url("g.jpg")
          "Глава "
          counters(item,"") ". ";
  counter-increment:item
}
</style>
</head>
<body>
<div>
 <h1>Что такое HTML и CSS</h1>
  <p>Обычный текст</p>
 <h1>Структура HTML-документа</h1>
 <p>Обычный текст</p>
 <h1>Основы CSS</h1>
</div>
</body>
</html>
```

Рис. 8.16. Пример добавления генерируемого содержимого к элементам `<h1>`

- `counter-increment:item` — увеличение текущего значения счетчика `item`;
- `counter-reset:item` — обнуление счетчика задается в правиле для контейнера `<div>`.

> ***Примечание***
>
> Следует иметь в виду, что пользователь браузера не может выделить содержимое, генерируемое посредством CSS. Его нельзя найти средствами поиска и скопировать в буфер обмена.

Теперь рассмотрим в качестве примера создание двухуровневого оглавления книги (листинг 8.1, рис. 8.17). Структура оглавления размечена тегами `<div>` и `<span>`. Названия глав и параграфов заключены в контейнеры `<span>`, отображаемые в блочном виде (`display:block`). Обратите внимание на разметку оглавления, обеспечивающую его многоуровневость. Параграфы, относящиеся к одной главе, заключены в тег `<div>`, который включается в тег `<span>`, соответствующий данной главе. В свою очередь теги `<span>` для глав включены в общий `<div>` самого верхнего уровня.

Первая буква в слове "Глава" — графическое изображение, а символ параграфа представлен в Unicode как `\A7`. Нумерация элементов оглавления отражает его иерархическую структуру. Чтобы элементы номера разделялись точкой, применено значение `counters(item,".")` параметра `contents`.

Листинг 8.1. Оглавление книги

```
 <!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Оглавление книги</title>
<style type="text/css">
div {
 counter-reset:item
}
span {
  display:block;
  font-size:12pt
}
div span:before {              /* 1-й уровень */
  letter-spacing:3px;
  font-weight:bold;
  content:url("g.jpg")         /* буква Г */
          "лава "
          counters(item,".") ". ";
  counter-increment:item
}
div span div span:before { /* 2-й уровень */
  letter-spacing:0;
  padding-left:30px;
  content:
          "\A7"              /* параграф */
          counters(item,".") ". ";
  counter-increment:item
}
</style>
</head>
<body>
<div>
 <span>Что такое HTML и CSS
    <div>
      <span>Языки разметки</span>
      <span>Таблицы стилей</span>
    </div>
 </span>
 <span>Структура HTML-документа
   <div>
      <span>Определение типа документа</span>
      <span>Структура собственно документа</span>
      <span>Разделы документа</span>
    </div>
 </span>
 <span>Основы CSS</span>
 <span>Позиционирование с помощью CSS</span>
</div>
</body>
</html>
```

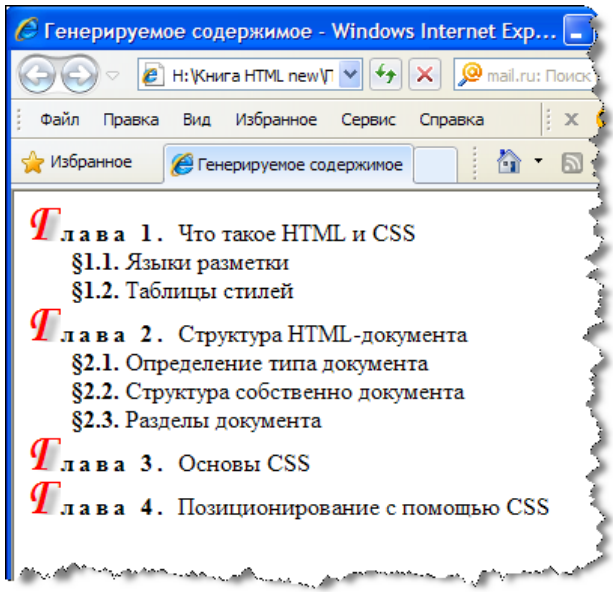


Рис. 8.17. Двухуровневое оглавление (листинг 8.1)

Аналогично можно построить оглавление и большей вложенности. По существу, мы реализовали посредством тегов `<div>` и `<span>` иерархический список, хотя для решения этой задачи в HTML имеются специальные теги `<ul>`, `<ol>` и `<li>`, которые мы рассмотрим в *главе 9*.

ГЛАВА 9

Списки

Оглавление статьи или книги, перечень товаров и услуг, словарь терминов и т. п. обычно представляют в виде простых или иерархических (многоуровневых, древовидных, вложенных) списков. Для задания структуры списка применяют специальные теги HTML, а для определения его внешнего вида — CSS.

Списки бывают маркированными (неупорядоченными) и нумерованными (упорядоченными), задаваемыми соответственно контейнерными тегами `<ul>` (от англ. *unordered list*) и `<ol>` (от англ. *ordered list*).

Элементы списков обоих типов задают тегами `<li>` (от англ. *list item*). Теги `<li>` можно записывать как неконтейнерные и как контейнерные. Если использовать неконтейнерную запись, то за `<li/>` непосредственно следует содержимое элемента: обычный текст, ссылка, графическое изображение и другие элементы. Контейнерная запись более предпочтительна, поскольку позволяет отчетливей представить структуру сложного списка. Однако возможны ситуации, когда данные варианты оказываются неэквивалентными (см. *разд. 9.3*).

Кроме тегов `<ul>` и `<ol>`, иногда применяют контейнерный тег `<dl>` (от англ. *definition list*) для создания списка типа словаря или определений (см. *разд. 9.7*).

Теги списков задают элементы блочного типа. Внешне по умолчанию они выделяются относительным положением (отступами). Так, элементы вложенного списка позиционируются ниже и правее содержащего их элемента (или элемента более высокого иерархического уровня). Впрочем, с помощью параметров CSS можно управлять расположением элементов списка.

9.1. Маркированный список

Маркированный список задается контейнерным тегом `<ul>`, в котором располагают теги `<li>`. Тег `<ul>` может содержать и другие `<ul>`, и в таком случае получаются вложенные списки.

На рис. 9.1 показан пример иерархического маркированного списка. Список отображается в окне браузера с отступами и маркерами в зависимости от уровня вложенности одного списка в другой. Маркеры в виде кружков и квадратиков появляются по умолчанию. Все браузеры, кроме Opera, по умолчанию создают различные маркеры для первых трех уровней вложения списков.

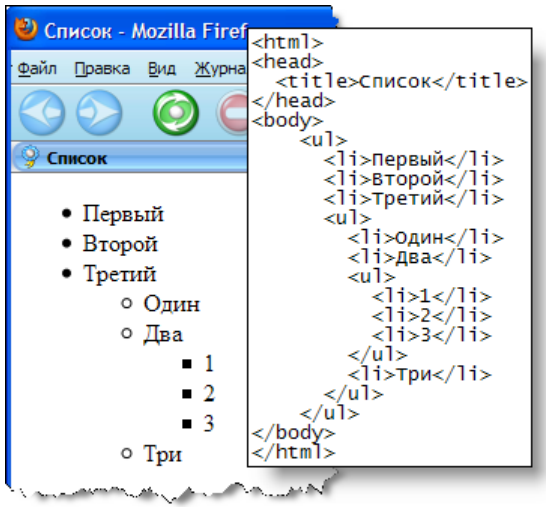


Рис. 9.1. Пример маркированного списка

С помощью параметра CSS `list-style-image:url("`*URL-адрес файла*`")` можно задать маркеры в виде картинок из графических файлов. Если по какой-либо причине файл с изображением маркера окажется недоступным, то будет отображен маркер по умолчанию. На рис. 9.2 приведен пример, в котором задаются различные графические маркеры для разных списков в зависимости от их уровня вложенности. Здесь в таблице стилей сначала назначается маркер для всех списков (`ul`), затем — для списков второго уровня (`ul ul`), и, наконец, определяется изображение маркера для всех спи-

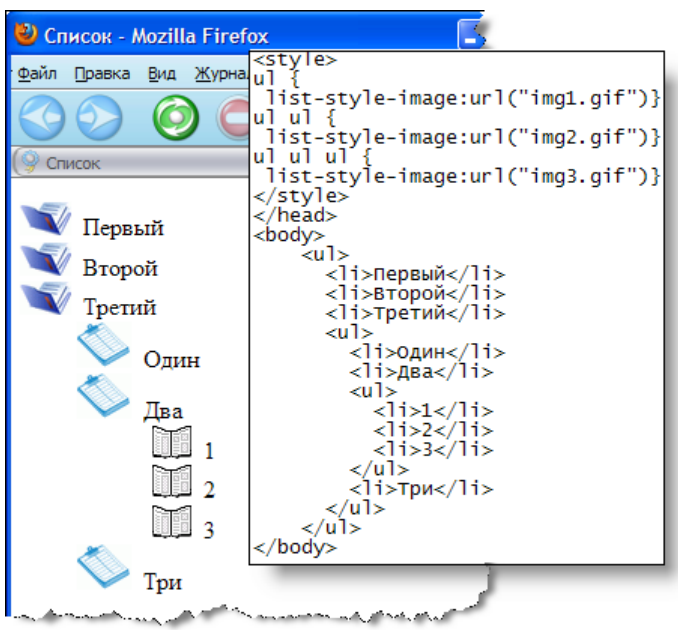


Рис. 9.2. Пример задания графических маркеров

сков внутри списка второго уровня (`ul ul ul`). Иначе говоря, мы используем контекстные классы стилей. Разумеется, данную задачу можно решить и иначе. Например, просто задать три класса стилевых параметров, а затем в каждом из трех элементов `<ul>` с помощью атрибута `class` сослаться на соответствующий класс.

Обычные маркеры в виде кружков и квадратиков для списков различных уровней вложенности можно установить по своему усмотрению, а также отменить их отображение с помощью параметра `list-style-type`. Данный параметр принимает множество значений, из которых отметим здесь лишь некоторые:

- `none` — отмена отображения маркеров;
- `disc` — закрашенный кружок;
- `circle` — незакрашенный кружок;
- `square` — закрашенный квадратик.

Кроме перечисленных значений возможны и такие, которые задают маркеры в виде чисел (десятичных целых и римских) и букв. Мы рассмотрим их в следующем разделе.

9.2. Нумерованный список

Нумерованный список строится аналогично маркированному. Его особенность состоит в том, что он определяется тегом `<ol>`, а маркеры его элементов по умолчанию имеют вид десятичных чисел (рис. 9.3).

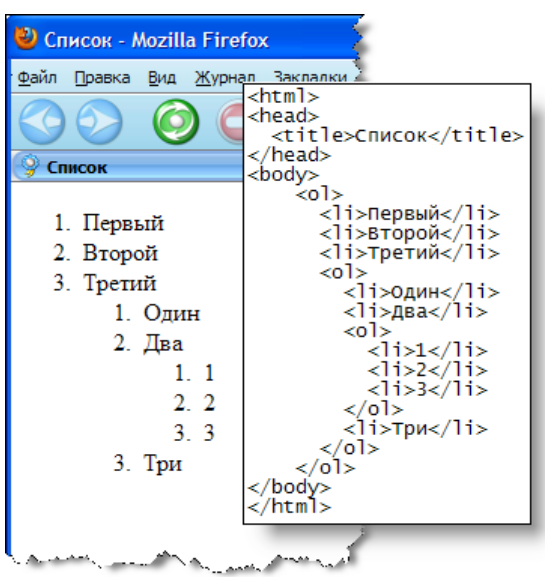


Рис. 9.3. Пример нумерованного списка

С помощью параметра `list-style-type` можно выбрать и другие маркеры, в том числе и такие, которые применяются для маркированных списков, задаваемых тегом `<ul>` (например, `disc`, `circle` и т. п.).

Вот несколько возможных значений, специфических для нумерованных списков:

- `decimal` — десятичные числа (1, 2, 3, ...);
- `decimal-leading-zero` — десятичные числа с ведущими нулями (01, 02, 03, ...);
- `lower-roman` — маленькие римские числа (i, ii, iii, iv, ...);
- `upper-roman` — большие римские числа (I, II, III, IV, ...);
- `lower-alpha` — строчные латинские буквы (a, b, c, ...);
- `upper-alpha` — прописные латинские буквы (A, B, C, ...).

Перечисленные значения можно применять и для списков, заданных посредством тега `<ul>`.

На рис. 9.4 приведен пример различных маркеров для трехуровневого нумерованного списка.

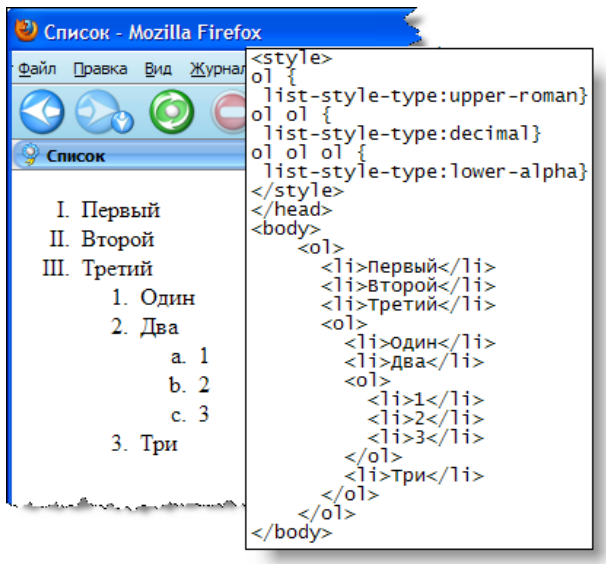


Рис. 9.4. Пример использования различных маркеров для трехуровневого нумерованного списка

Итак, для создания списка предназначены два тега: `<ul>` и `<ol>`. Различие между ними проявляется только тогда, когда вы применяете их без указания вида маркеров с помощью CSS. В этом случае маркеры в виде кружков и квадратов или десятичных номеров отображаются автоматически (по умолчанию) в зависимости от тега — `<ul>` или `<ol>`. Если вы решили отформатировать списки с помощью CSS, то уже не важно, какой именно тег будет применен.

9.3. Автоматическая нумерация элементов списка

При создании сложного и длинного списка нередко возникает желание автоматически пронумеровать элементы так, чтобы отразить их иерархию. Это делается с помощью

стилевых параметров `content`, `counter-increment` и `counter-reset` для организации счетчика элементов списка. Мы уже встречались с этими параметрами в *разд. 8.6*.

Оказывается, что правильность результата автоматической нумерации существенно зависит от разметки списка. На рис. 9.5 показан пример трехуровневого списка с автоматической нумерацией элементов в двух вариантах разметки. В одном варианте теги `<li>` применялись в неконтейнерной нотации, а в другом — в контейнерной. Во втором варианте пришлось включить подсписок третьего уровня в контейнер `<li>`. Если этого не сделать, то нумерация будет неправильной, как показано на рис. 9.6. В Opera, в отличие от других браузеров, неправильный номер будет 3.4.

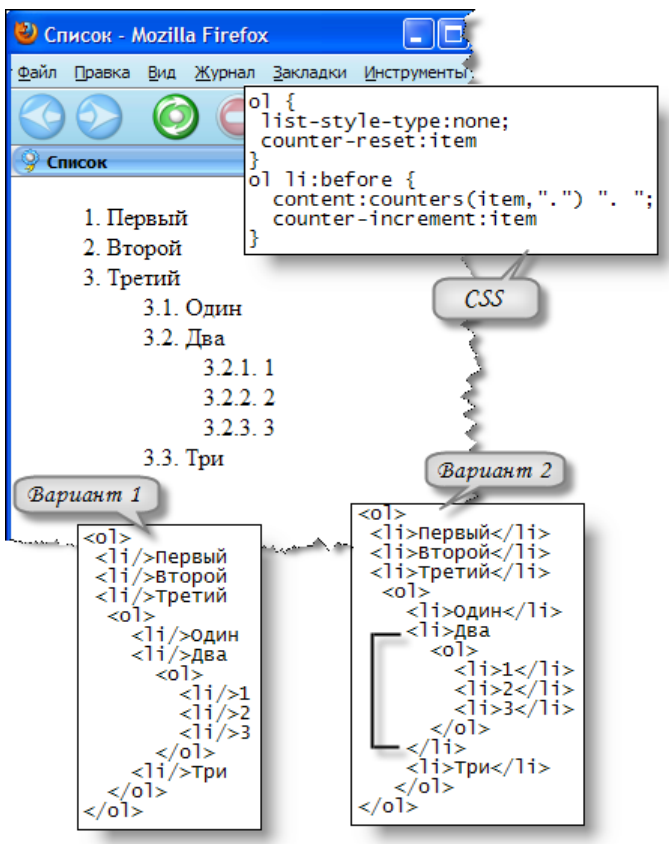


Рис. 9.5. Пример трехуровневого списка с автоматической нумерацией элементов в двух вариантах разметки списка

Рассмотрим теперь правила CSS, обеспечивающие автоматическую нумерацию. Для элементов списка использован псевдокласс `li:before`, задающий предшествующие им элементы (в нашем случае — номера). Конкретно, в `li:before` определены два параметра:

- `content` — вычислить содержимое элемента функцией `counters(item, ".")`, где `item` — идентификатор счетчика (задается произвольно), а `"."` — разделитель компонентов номера; кроме того, в конце номера следует поставить точку и пробел (`". "`);

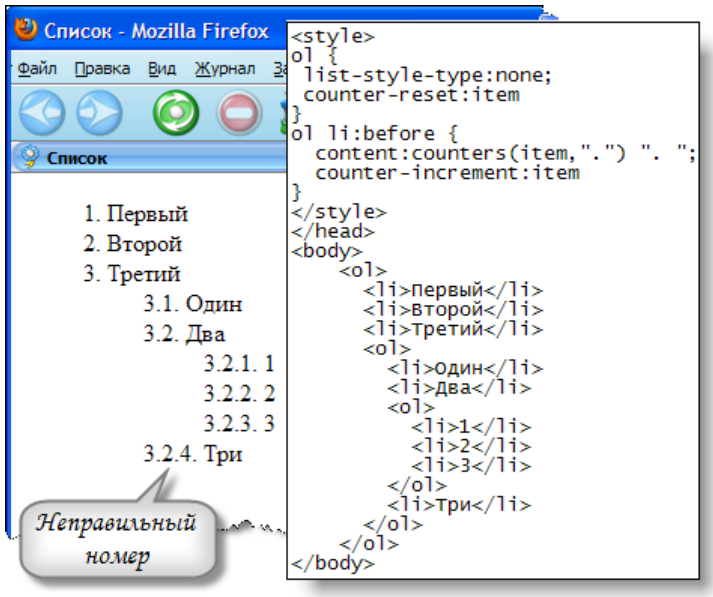


Рис. 9.6. Пример трехуровневого списка
с неправильной автоматической нумерацией элементов

- `counter-increment:item` — увеличить содержимое счетчика `item` на единицу.

Для списка в целом (элемент `<ol>`) определены такие параметры:

- `list-style-type:none` — не выводить маркеры элементов списка;
- `counter-reset:item` — обнулить содержимое счетчика `item`.

Параметр `content` позволяет не только вычислить номер элемента в списке, но и добавить к нему строки (декорировать номера генерируемым содержимым). В листинге 9.1 приведен пример двухуровневого списка, в котором маркеры элементов первого уровня отображаются в виде `"Глава 1"`, `"Глава 2"` и т. д., а маркеры элементов второго уровня — номером, перед которым стоит символ параграфа §. Специальные символы в CSS представляются шестнадцатеричным кодом Unicode с обратным слэшем в качестве префикса. Так, символ § следует представить как `\A7`. Внешний вид данного документа показан на рис. 9.7.

Листинг 9.1. Список с автоматической нумерацией и декорацией номеров

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
  <title>Список</title>
<style type="text/css">
ol {
 list-style-type:none;
 counter-reset:item
}
```

```
ol li:before { /* перед элементами 1-го уровня */
  content:"Глава " counters(item,".") ". ";
  counter-increment:item
}
ol ol li:before { /* перед элементами 2-го уровня */
  content: "\A7" counters(item,".") ". ";
  counter-increment:item
}
</style>
</head>
<body>
<ol>
 <li>Что такое HTML и CSS
   <ol>
     <li>Языки разметки документа</li>
     <li>Таблицы стилей</li>
   </ol>
 </li>
 <li>Структура HTML-документа
 <li>Основы CSS
   <ol>
     <li>Присоединение таблиц стилей</li>
     <li>Правила форматирования</li>
     <li>Приоритеты определений</li>
     <li>Размерность и цвета</li>
     <li>Блоки</li>
   </ol>
 </li>
</ol>
</body>
</html>
```

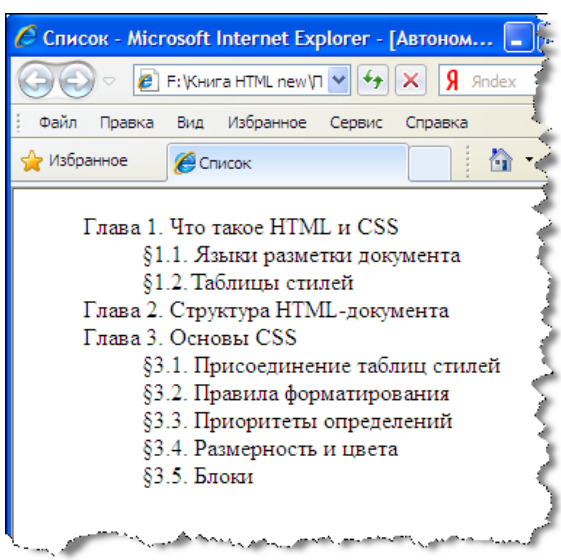


Рис. 9.7. Список с автоматической нумерацией и декорацией номеров (листинг 9.1)

Средства автоматической нумерации можно применить не только к спискам, но и к другой системе тегов. В *разд. 8.6* рассматривался пример нумерации элементов `<h1>`, а также создание нумерованного иерархического списка на основе тегов `<div>` и `<span>`.

9.4. Иерархический раскрывающийся список

Иерархические списки, рассмотренные в предыдущих разделах, являются статическими и полностью развернутыми. Для экономии места на странице можно скрыть элементы нижних уровней и раскрывать их при необходимости наведением указателя мыши на ближайший элемент верхнего уровня. Такие списки будем называть раскрывающимися.

Рассмотрим сначала создание двухуровневого списка. Прежде всего, следует представить себе его структуру в развернутом виде и написать соответствующий HTML-код (т. е. сделать разметку списка). На рис. 9.8 показан пример двухуровневого списка: внешний вид и HTML-код. Это всего лишь начальный эскиз нашего проекта. Пункты "Раздел 1" и "Раздел 4" содержат списки второго уровня вложенности.

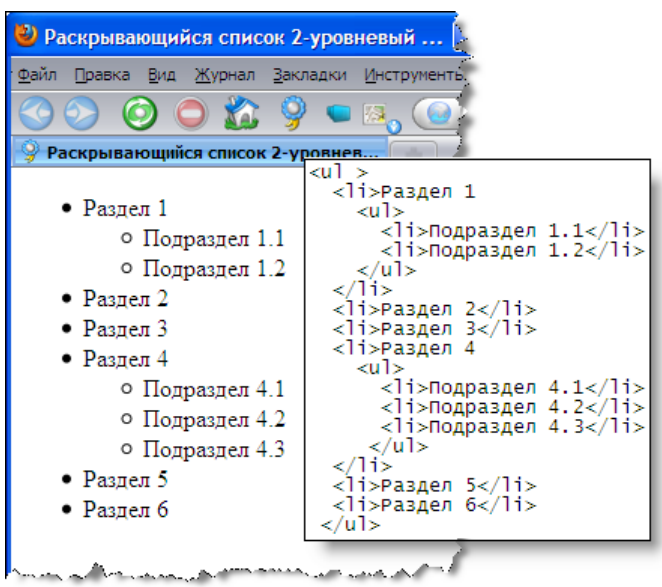


Рис. 9.8. Эскиз двухуровневого списка

В начальном состоянии все списки второго уровня (подсписки) должны быть невидимыми и появляться только при наведении указателя мыши на элемент `<li>`, содержащий подсписок. Видимость подсписка можно регулировать параметром `display`. Чтобы пользователь видел, что элемент списка можно раскрыть, последний можно снабдить соответствующим графическим маркером или/и еще как-то выделить (например, цветом, шрифтом, формой указателя мыши). В данном примере раскрывающиеся элементы ("Раздел 1" и "Раздел 2") выделены графическим маркером с изображением символа "+", а также полужирным шрифтом. Когда элемент раскроется, маркер должен стать

обычным, а шрифт — нормальным. Вид маркера задается параметром `list-style-image`. Итак, нам остается создать таблицу стилей.

В листинге 9.2 приведен код двухуровневого раскрывающегося списка с указанными характеристиками, а на рис. 9.9 — его внешний вид до и после раскрытия одного из подсписков. В данном примере список заключен в элемент `<div id="mylist">`, чтобы при желании ему можно было назначить общие параметры, например, размеры, позиционирование, шрифт и др.

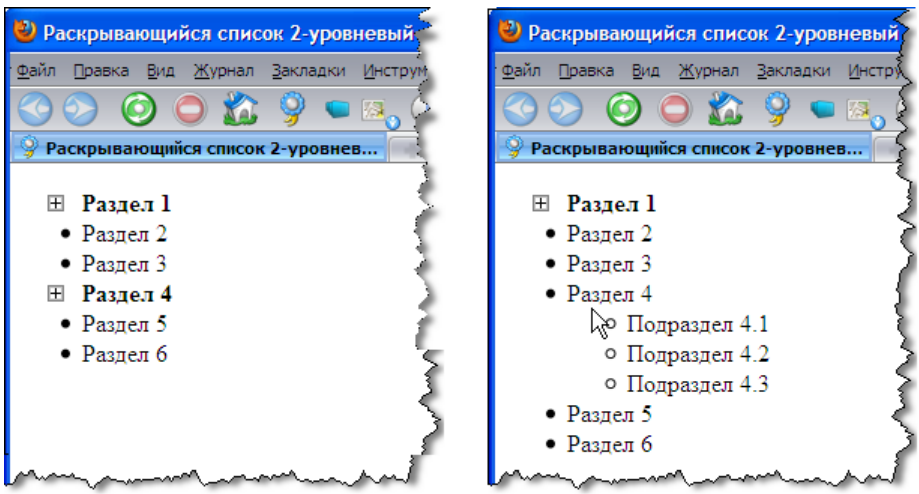


Рис. 9.9. Двухуровневый раскрывающийся список (листинг 9.2)

В обычном состоянии все подсписки не видны, что задается правилом

```
#mylist ul li ul {display:none}
```

Сложный селектор данного правила лучше читать справа налево: список, расположенный в элементе списка, который находится в `#mylist`. Если на элементе списка, содержащем подсписок, находится указатель мыши, то этот подсписок виден, о чем говорит правило

```
#mylist ul li:hover ul{display:block}.
```

Для элементов, содержащих раскрывающиеся подсписки, предусмотрен класс `.opening`, в котором указаны графический маркер, полужирный шрифт, а также указатель мыши в виде стрелки (`cursor:default`), т. к. по умолчанию для списков устанавливается курсор ввода (`cursor:text`). Разумеется, оформление списка может быть и иным. Элементы подсписков, а также элементы, не содержащие подсписков, вместо обычных текстов могут включать ссылки.

Листинг 9.2. Раскрывающийся двухуровневый список

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
```

```
<title>Раскрывающийся список 2-уровневый</title>
<style type="text/css">
  /* Появление/скрытие: */
  #mylist ul li ul      {display:none;}
  #mylist ul li:hover ul{display:block;}
  #mylist ul li:hover   {
     list-style-image:none;
     font-weight:normal
  }

  .opening { /* раскрывающийся пункт */
    list-style-image:url("plus.gif");
    font-weight:bold;
    cursor:default
  }
  </style>
</head>
<body>
<div id="mylist">
  <ul >
    <li class="opening">Раздел 1
      <ul>
        <li>Подраздел 1.1</li>
        <li>Подраздел 1.2</li>
       </ul>
    </li>
    <li>Раздел 2</li>
    <li>Раздел 3</li>
    <li class="opening">Раздел 4
      <ul>
        <li>Подраздел 4.1</li>
        <li>Подраздел 4.2</li>
        <li>Подраздел 4.3</li>
      </ul>
    </li>
    <li>Раздел 5</li>
    <li>Раздел 6</li>
   </ul>
</div>
</body>
</html>
```

В основу раскрывающегося трехуровневого списка можно положить уже рассмотренные принципы создания двухуровневого списка. Однако при этом мы не можем ограничиться модификацией одной лишь HTML-разметки. Придется доработать и таблицу стилей. На рис. 9.10 показан эскиз проекта трехуровневого списка: внешний вид и HTML-разметка. В данном примере пункт "Раздел 1" содержит список второго уровня вложенности, в котором пункт "Подраздел 1.2" содержит список третьего уровня; пункт "Раздел 4" содержит только список второго и единственного уровня.

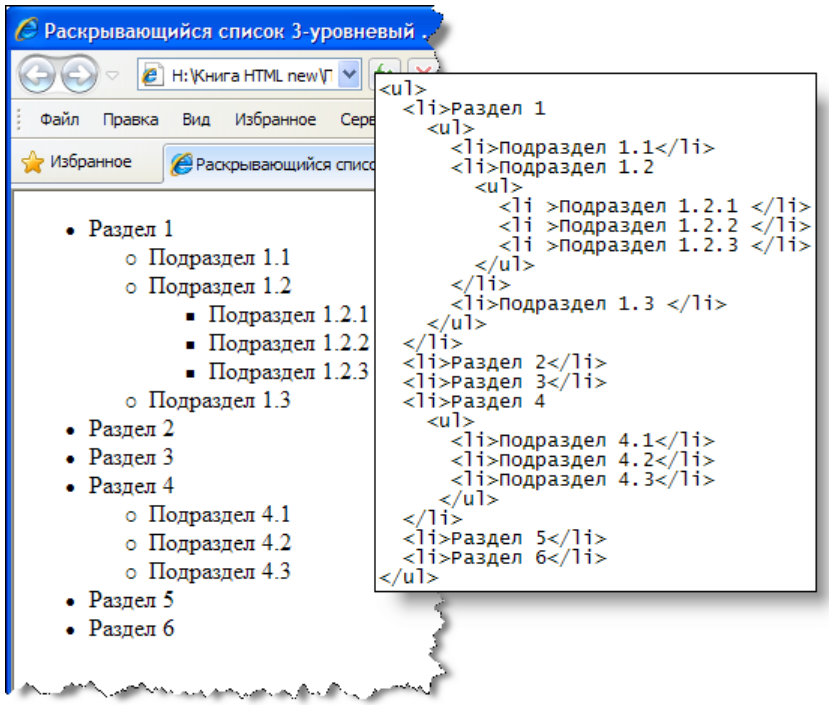


Рис. 9.10. Эскиз трехуровневого списка

В листинге 9.3 приведен код трехуровневого раскрывающегося списка с указанными характеристиками, а на рис. 9.11 — его внешний вид до и после раскрытия одного из подсписков.

Листинг 9.3. Раскрывающийся трехуровневый список

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">

<html>
<head>
<title>Раскрывающийся список 3-уровневый</title>
<style type="text/css">
  /* Появление/скрытие: */
  #mylist ul li ul       {display:none;}
  #mylist ul li:hover ul {display:block}
  #mylist ul li:hover    {
                          list-style-image:none;
                          font-weight:normal
  }

  #mylist ul li:hover ul ul           {display:none;}
  #mylist ul li:hover ul li:hover ul {display:block;}
  #mylist ul li:hover ul li:hover     {list-style-image:none}
```

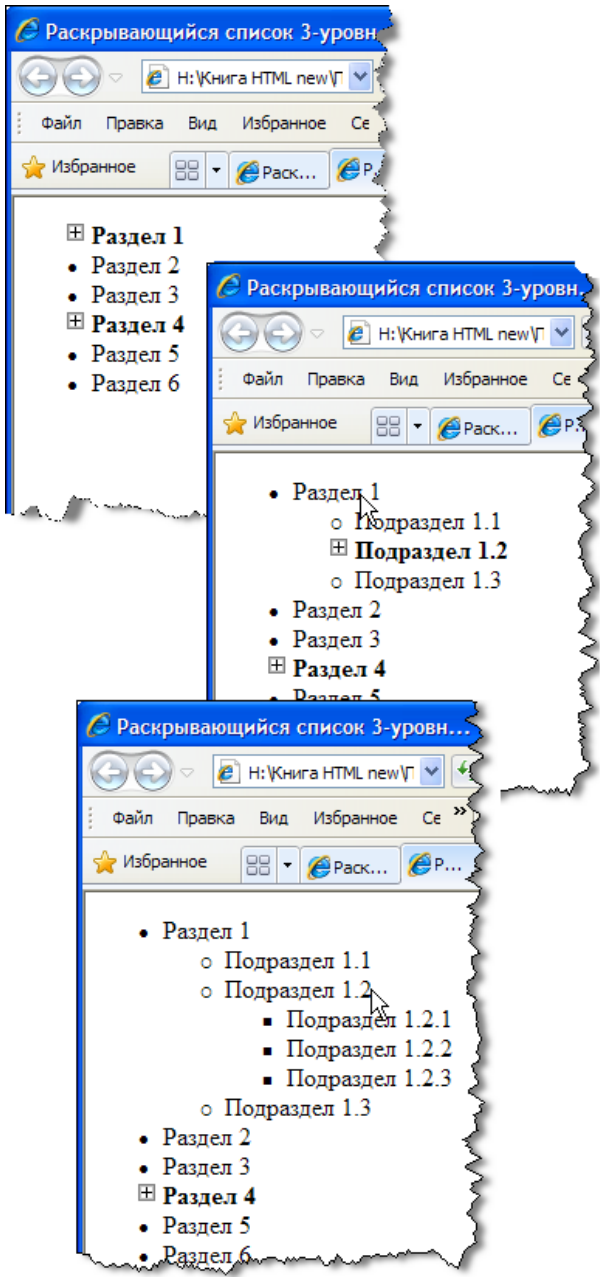


Рис. 9.11. Трехуровневый раскрывающийся список (листинг 9.3)

```
  .opening { /* раскрывающийся пункт */
    list-style-image:url("plus.gif");
    font-weight:bold;
    cursor:default
  }
  .opening2 { /* раскрывающийся подпункт */
    list-style-image:none;
    font-weight:normal
  }

</style>
</head>
<body>
<div id="mylist">
  <ul >
    <li class="opening">Раздел 1
      <ul class="opening2">
        <li>Подраздел 1.1</li>
        <li class="opening">Подраздел 1.2
          <ul class="opening2">
            <li >Подраздел 1.2.1 </li>
            <li >Подраздел 1.2.2 </li>
            <li >Подраздел 1.2.3 </li>
          </ul>
        </li>
        <li>Подраздел 1.3 </li>
       </ul>
    </li>
    <li>Раздел 2</li>
    <li>Раздел 3</li>
    <li class="opening">Раздел 4
      <ul>
        <li>Подраздел 4.1</li>
        <li>Подраздел 4.2</li>
        <li>Подраздел 4.3</li>
      </ul>
    </li>
    <li>Раздел 5</li>
    <li>Раздел 6</li>
   </ul>
</div>
</body>
</html>
```

9.5. Меню на основе списка

Создание меню на основе элементов рассматривалось в *разд. 7.1*. Аналогичную задачу можно решить и на основе тегов, задающих списки. Здесь мы рассмотрим построение двухуровневого раскрывающегося меню, состоящего из горизонтального

главного меню и вертикальных подменю, которые появляются при наведении указателя мыши на соответствующий пункт главного меню. Необходимый для этого код можно получить путем некоторой модификации кода для двухуровневого раскрывающегося списка (листинг 9.2).

В нашем примере главное горизонтальное меню содержит три пункта, первые два из которых раскрываются. В листинге 9.4 приведен первый вариант такого меню, а на рис. 9.12 — его внешний вид. Обратите внимание, что горизонтальное расположение пунктов меню обеспечивается параметром `float:left`. Маркеры элементов списка удалены параметром `list-style-type:none`.

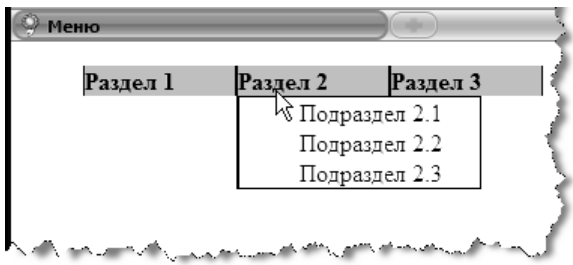


Рис. 9.12. Внешний вид меню листинга 9.4

Листинг 9.4. Раскрывающееся двухуровневое меню (вариант 1)

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Меню</title>
<style type="text/css">
#mylist ul {
  list-style-type:none; /* маркеры не нужны */
  cursor:default
}
#mylist ul li.opening { /* горизонтальное меню*/
  float:left;
  width:100px;
  border-left:solid 1px;
  border-right:solid 1px;
}
  /* Появление/скрытие: */
#mylist ul li ul    {
  display:none;
  border:solid 1px;
  width:120px;
  background:yellow;
  font-weight:normal;
}
```

```
#mylist ul li:hover ul{
  display:block
}

.opening { /* пункт главного меню*/
  background:#00ffff;
  font-weight:bold;
}
</style>
</head>
<body>
<div id="mylist">
  <ul >
    <li class="opening">Раздел 1
      <ul>
        <li>Подраздел 1.1</li>
        <li>Подраздел 1.2 </li>
       </ul>
    </li>

    <li class="opening">Раздел 2
      <ul>
        <li>Подраздел 2.1</li>
        <li>Подраздел 2.2</li>
        <li>Подраздел 2.3</li>
      </ul>
    </li>
    <li class="opening">Раздел 3</li>
   </ul>
</div>
</body>
</html>
```

Данное меню обладает рядом очевидных недостатков, связанных с выравниванием элементов: положение названий пунктов горизонтальной части меню не центрировано, отступы названий пунктов вертикальных подменю слишком велики и прямоугольник подменю немного сдвинут вправо относительно границы между пунктами горизонтального меню. В листинге 9.5 эти недостатки устранены (рис. 9.13). Для центрирова-

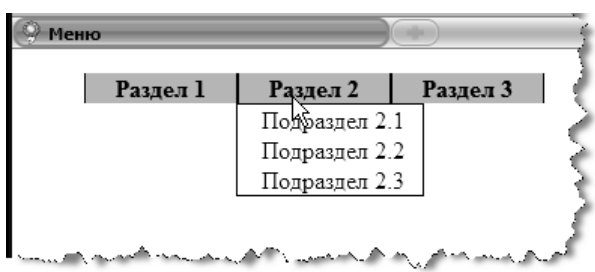


Рис. 9.13. Внешний вид меню листинга 9.5

ния названий пунктов горизонтальной части меню применен параметр `text:align: center`. Левый отступ для элементов подменю задан параметром `padding:0 0 0 3px`. Для коррекции положения подменю применена пара параметров: `position:relative; left:-2px`.

Листинг 9.5. Раскрывающееся двухуровневое меню (вариант 2)

```
 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Меню</title>
<style type="text/css">
#mylist ul {
  list-style-type:none; /* маркеры не нужны */
  cursor:default
}
#mylist ul li.opening { /* горизонтальное меню*/
  float:left;
  width:100px;
  border-left:solid 1px;
  border-right:solid 1px;
  text-align:center
}
  /* Появление/скрытие: */
#mylist ul li ul    {
  display:none;
  border:solid 1px;
  width:120px;
  background:yellow;
  font-weight:normal;
  padding:0 0 0 3px;
  position:relative;
  left:-2px /* т. к. граница между горизонтальными опциями 2px*/
}
#mylist ul li:hover ul{
  display:block
}

.opening { /* пункт главного меню*/
  background:#00ffff;
  font-weight:bold;
}
</style>
</head>
<body>
<div id="mylist">
  <ul >
    <li class="opening">Раздел 1
```

```
      <ul>
        <li>Подраздел 1.1</li>
        <li>Подраздел 1.2 </li>
       </ul>
    </li>

    <li class="opening">Раздел 2
      <ul>
        <li>Подраздел 2.1</li>
        <li>Подраздел 2.2</li>
        <li>Подраздел 2.3</li>
      </ul>
    </li>
    <li class="opening">Раздел 3</li>
   </ul>
</div>
</body>
</html>
```

9.6. Выравнивание элементов списка

Элементы многоуровневого списка позиционируются относительно друг друга с некоторыми отступами. Маркеры также занимают некоторое предопределенное по умолчанию положение. Отступы и положение маркеров можно задавать с помощью параметров CSS.

На рис. 9.14 приведен пример двухуровневого списка без каких-либо параметров CSS, кроме задающих границы блоков. Это сделано только для наглядности вложенности элементов: границы `<ul>` отображаются штриховой линией, а `<li>` — сплошной. Обратите внимание на различия в определениях блоков браузерами Internet Explorer и Firefox. Opera, Safari и Chrome создают блоки для элементов списка так же, как и Firefox.

Видимые отступы элементов списка определяются параметрами `margin` и `padding`. Если они не указаны явно в таблице стилей, то устанавливаются браузером автоматически. При этом Internet Explorer устанавливает отступы заданием значения параметра `margin-left`, а остальные браузеры — `padding-left` (рис. 9.15). Поэтому если вы попытаетесь задать отступ посредством только какого-нибудь одного параметра, то получите разные результаты для Internet Explorer и других браузеров. Чтобы отображение было одинаковым, необходимо задать значения как для `margin`, так и для `padding`.

Положение маркера относительно элемента списка задается параметром `list-style-position`, принимающим значения:

- `inside` — маркер располагается внутри блока элемента и обтекается содержимым последнего;
- `outside` — маркер располагается вне блока элемента (значение по умолчанию).

На рис. 9.16 показан список при различных значениях параметра `list-style-position`. Для наглядности элементам списка заданы видимые границы шириной 1 px. Соответствующий код приведен в листинге 9.6.

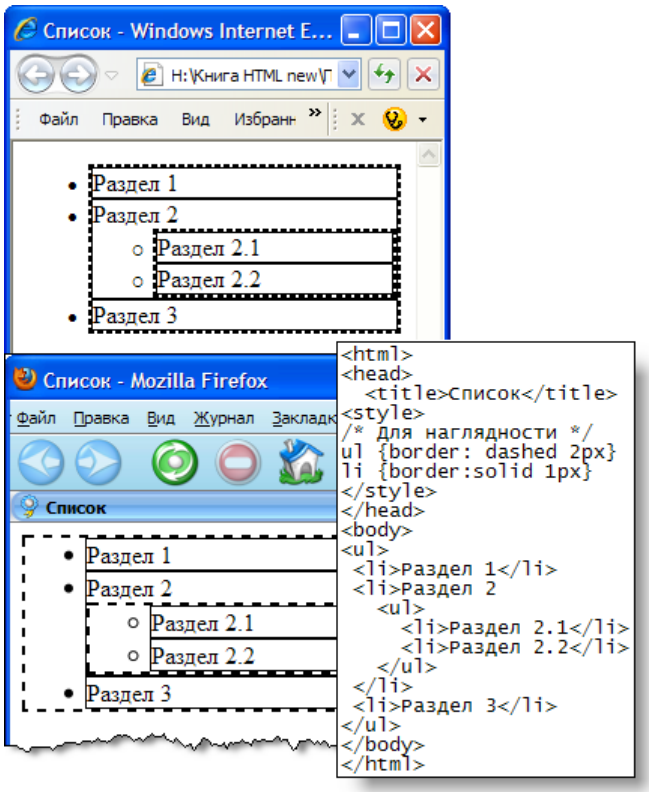


Рис. 9.14. Блоки элементов двухуровневого списка, создаваемые по умолчанию, в Internet Explorer и Firefox

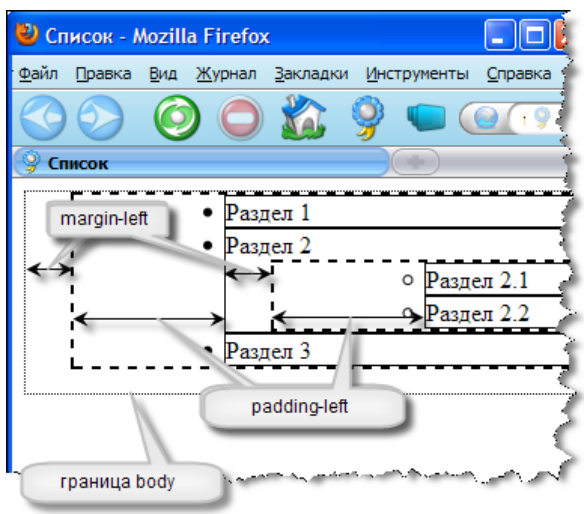


Рис. 9.15. Параметры, определяющие отступы элементов списка

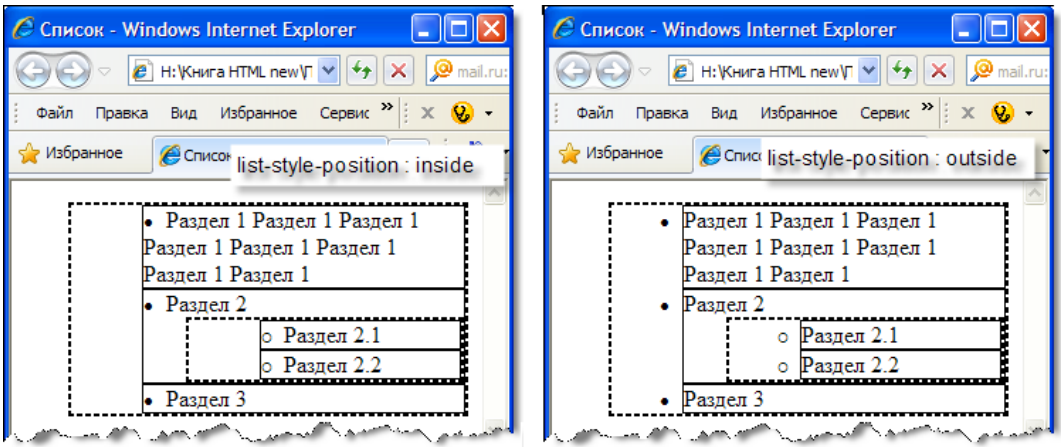


Рис. 9.16. Положение маркера относительно элемента списка

Листинг 9.6. Положение маркера относительно элемента списка

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
  <title>Список</title>
<style type="text/css">
/* Границы для наглядности */
ul {
  border: dashed 2px;
  margin-left:30px;
  padding-left:50px;
  list-style-position:outside; /*другое значение: inside*/
}
li {border:solid 1px;}
</style>
</head>
<body>
<ul>
 <li>Раздел 1 Раздел 1 Раздел 1
     Раздел 1 Раздел 1 Раздел 1
     Раздел 1 Раздел 1
 </li>
 <li>Раздел 2
   <ul>
     <li>Раздел 2.1</li>
     <li>Раздел 2.2</li>
   </ul>
 </li>
 <li>Раздел 3</li>
```

```
</ul>
</body>
</html>
```

9.7. Список определений

Как известно, определение состоит из двух частей: определяемого термина или понятия и собственно его определения или описания. Список определений задается контейнерным тегом `<dl>` (от англ. *definition list*), в который для каждого элемента (определения) вставляется пара контейнерных тегов: `<dt>` — для определяемого термина (от англ. *definition term*) и `<dd>` — для его описания (от англ. *definition description*). Браузер отображает определяемый термин в отдельной строке, а его описание — ниже и с отступом. В листинге 9.7 приведен пример кода со списком определений, а на рис. 9.17 — его вид в окне браузера.

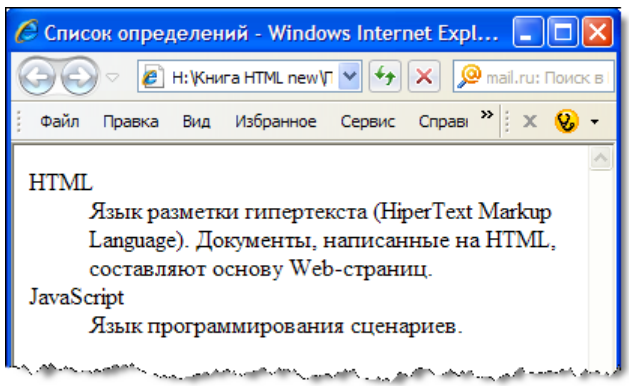


Рис. 9.17. Пример списка определений

Листинг 9.7. Список определений

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
  <head><title>Список определений</title></head>
<body>
  <dl>
    <dt>HTML</dt>
    <dd>Язык разметки гипертекста (Hyper Text Markup Language).
      Документы, написанные на HTML, составляют основу Web-страниц.
    </dd>
    <dt>JavaScript</dt>
    <dd>Язык программирования сценариев.</dd>
  </dl>
</body>
</html>
```

Разумеется, столь нехитрое форматирование можно организовать и другими средствами как HTML, так и CSS. Однако не забывайте, что теги в (X)HTML должны использоваться главным образом для структуризации документа, а их визуализация — дело вторичное. На практике нередко приходится создавать списки в структуре определений. В таком случае уместно применять соответствующие теги безотносительно к тому, как они отображаются браузерами по умолчанию, т. е. без специальных определений CSS. При необходимости вы можете задать собственные стили как для тегов списка, так и для их содержимого.

ГЛАВА 10

Таблицы

Таблицы в (X)HTML-документах применяют довольно часто с двумя основными целями. Во-первых, для представления данных с естественной табличной организацией (например, прайс-листы, сведения о спортивных состязаниях, план-график выполнения работ и т. п.). Во-вторых, таблицы все еще широко используются как средство разметки пространства Web-страницы, хотя эту задачу лучше решать с помощью тегов `<div>` и CSS.

10.1. Табличные теги

Данные в таблице организованы в виде прямоугольной сетки, которая состоит из вертикальных колонок (столбцов) и горизонтальных рядов (строк). Таблицы состоят из пяти основных частей: названия, верхней части (шапки), заголовков столбцов, ячеек с данными и нижней части (подвала). Впрочем, таблица может и не иметь названия, заголовков столбцов, верхней и нижней частей. Для задания таблицы достаточно определить ее ячейки, в которых будут находиться данные.

Все теги формирования таблицы, кроме `<col/>`, контейнерные, они содержат:

- `<table>` — все другие теги таблицы;
- `<caption>` — название таблицы; если применяется, то только непосредственно после открывающей части тега `<table>`;
- `<thead>` — набор строк верхней части таблицы; если применяется, то только один раз перед элементом `<tbody>`;
- `<tfoot>` — набор строк, располагающихся ниже основной части таблицы (подвал); если применяется, то только один раз перед элементом `<tbody>`;
- `<tbody>` — набор строк основной части таблицы; может применяться несколько раз для логического выделения группы строк, задаваемых тегами `<tr>`;
- `<tr>` — ячейки, составляющие одну строку таблицы; в теге `<tr>` размещаются теги ячеек `<td>` и `<th>`;
- `<td>` — данные, содержащиеся в ячейке таблицы;
- `<th>` — данные, содержащиеся в ячейке, используемой как заголовок столбца;

- <col/> — параметры выравнивания столбцов; это неконтейнерный тег;
- <colgroup> — группы столбцов.

Для формирования таблицы тег <table> обязателен. Остальные теги присутствуют в нем по желанию разработчика. В большинстве случаев можно обойтись только тегами <tr> для строк и <td> для ячеек.

Для задания названия таблицы предназначен тег <caption>. По умолчанию название располагается над прямоугольником таблицы и центрируется относительно его по горизонтали. Управлять расположением элемента <caption> можно посредством стилевого параметра caption-side, который принимает значения top и bottom, Firefox воспринимает еще и значения right и left. Данный параметр назначается элементу <table>.

Каждый ряд (строка) ячеек задается тегом <tr>. Если ряд должен содержать заголовки столбцов таблицы, то добавляют теги <th>. Для размещения данных в ячейках служит тег <td>. Теги заголовков столбцов и ячеек для данных должны располагаться в теге ряда <tr>.

Таблица — это элемент блочного типа. По умолчанию она представляется в окне браузера без рамки. Размеры таблицы в целом и ячеек вычисляются браузером исходя из объема содержимого (данных). Названия заголовков в тегах <th> центрируются в ячейках по горизонтали и вертикали и отображаются полужирным шрифтом, а данные в тегах <td> центрируются по вертикали, выравниваются по левому краю ячеек и отображаются обычным шрифтом.

В листинге 10.1 приведен код таблицы, содержащей все перечисленные теги, а на рис. 10.1 — ее вид в окне браузера.

Листинг 10.1. Таблица

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Таблица</title></head>
<body>
<table>
  <caption>Список сотрудников</caption>
  <thead>
    <tr><td>Надзаголовок</td></tr>
  </thead>
  <tfoot>
    <tr><td>Подзаголовок</td></tr>
  </tfoot>
  <tbody>
    <tr>
      <th>Фамилия</th><th>Должность</th><th>Оклад</th>
    </tr>
    <tr>
      <td>Иванов</td><td>Директор</td><td>10000</td>
    </tr>
```

```
    <tr>
      <td>Петров</td><td>Бухгалтер</td><td>5000</td>
    </tr>
    <tr>
      <td>Сидоров</td><td>Охранник</td><td>3000</td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

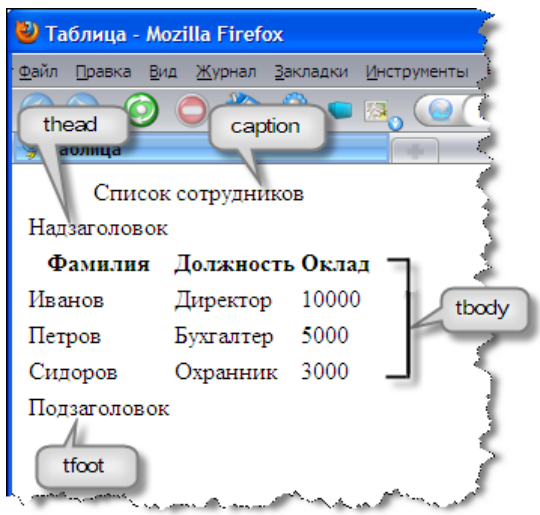


Рис. 10.1. Пример таблицы (листинг 10.1)

При создании таблицы обычно решают две основные задачи:

- задают геометрическую структуру (число рядов и ячеек, размеры, расстояния между ними и т. п.);
- определяют внешний вид и форматирование данных в ячейках (рамки, фон, выравнивание данных в ячейках).

Основу таблицы формируют HTML-тегами, а далее ее приводят к надлежащему виду с помощью соответствующих атрибутов тегов и параметров CSS, которые мы рассмотрим в следующих разделах.

10.2. Рамки таблицы

Как уже отмечалось, по умолчанию таблица отображается без рамок (видимых границ). Задание рамок относится к задаче внешнего оформления таблицы. Однако их видимость делает геометрическую структуру таблицы более наглядной. Даже если наличие рамок таблицы не соответствует вашему дизайнерскому плану, их все же полезно отобразить, по крайней мере, на этапе проектирования и отладки кода. Позже их легко можно удалить. Поэтому начнем с определения рамок.

Чтобы отобразить внешнюю рамку таблицы, а также индивидуальные рамки ячеек, в теге `<table>` можно указать целочисленное значение атрибута `border`, определяющее ее толщину (в пикселах), например, `<table border="5">`. При этом указанную толщину будет иметь только внешняя рамка таблицы, а толщина рамок ячеек останется равной 1 px. Каждая ячейка имеет собственную рамку, а между рамками соседних ячеек предусмотрен некоторый зазор, поэтому разделительные линии кажутся двойными. Если атрибут `border` не указывать или задать ему нулевое значение, то таблица отобразится без рамок. На рис. 10.2 показаны примеры таблицы с различными значениями атрибута `border`. Внешняя рамка отображается в оттенках серого цвета, причем разных для различных частей рамки, что можно заметить при достаточно большой ее толщине. Однако собственно оттенки могут различаться в разных браузерах.

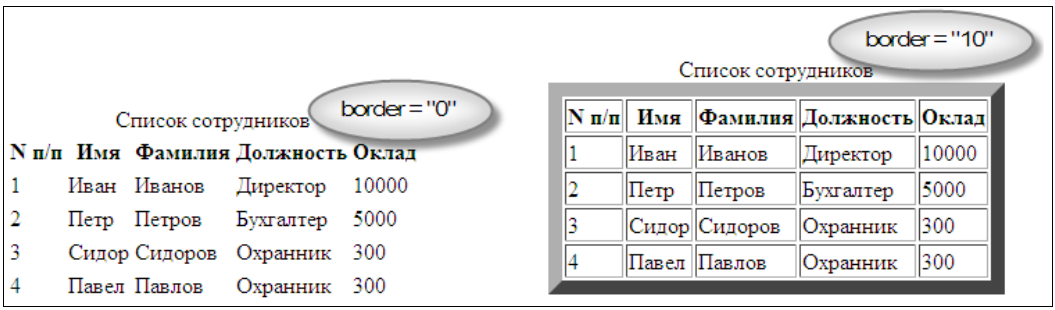


Рис. 10.2. Влияние атрибута `border` на видимость и толщину рамок таблицы

Отображением рамок можно управлять с помощью атрибутов `frame` и `rules`. Хотя это нестандартные атрибуты, их поддерживают ведущие браузеры.

С помощью атрибута `frame` можно дополнительно указать, какие части внешней рамки следует отображать, а какие нет. Данный атрибут действует только при указанном в `<table>` атрибуте `border` с ненулевым значением, например,

```
<table border="3" frames="box">
```

Значения атрибута `frame`:

- `box` — отображать все четыре стороны рамки;
- `above` — отображать только верхнюю часть рамки;
- `below` — отображать только нижнюю часть рамки;
- `hsides` — отображать горизонтальные части рамки сверху и снизу;
- `vsides` — отображать только левую и правую вертикальные части рамки;
- `lhs` — отображать только левую часть рамки;
- `rhs` — отображать только правую часть рамки;
- `void` — не отображать внешнюю рамку.

На рис. 10.3 показано несколько примеров использования атрибута `frame`.

Отображением внутренних разделительных линий (рамок ячеек) управляет атрибут `rules`, который действует только при указанном в `<table>` атрибуте `border` с ненулевым значением, например,

```
<table border="3" rules="cols">
```

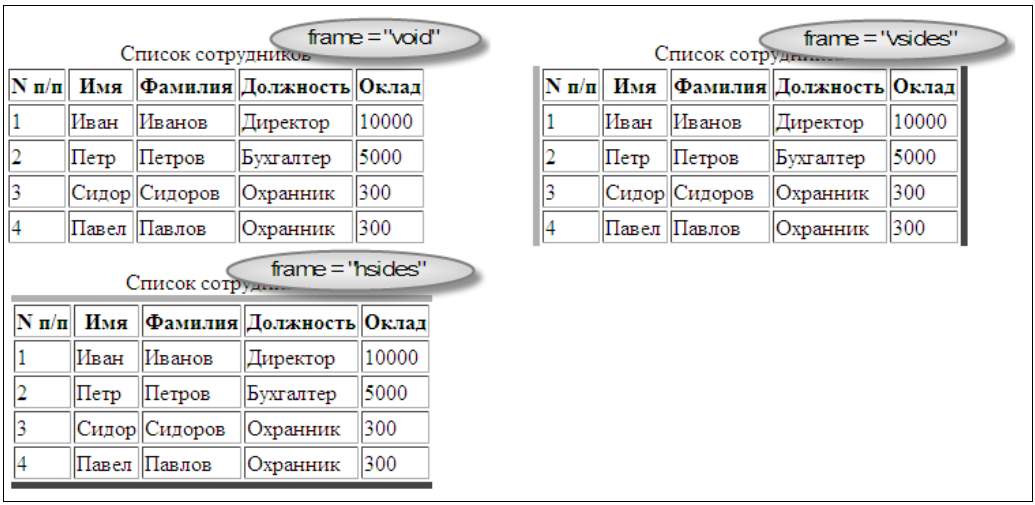


Рис. 10.3. Применение атрибута `frame` для управления отображением внешних рамок

Значения атрибута `rules`:

- `all` — отображать все вертикальные и горизонтальные линии;
- `rows` — отображать только горизонтальные линии между строками;
- `cols` — отображать только вертикальные линии между столбцами;
- `none` — не отображать разделительные линии.

На рис. 10.4 показано несколько примеров использования атрибута `rules`.

Обратите внимание, что при использовании атрибута `rules` со значением, отличным от `none`, внешняя рамка имеет несколько иной вид. При этом внутренние разделительные линии одинарные, а не двойные.

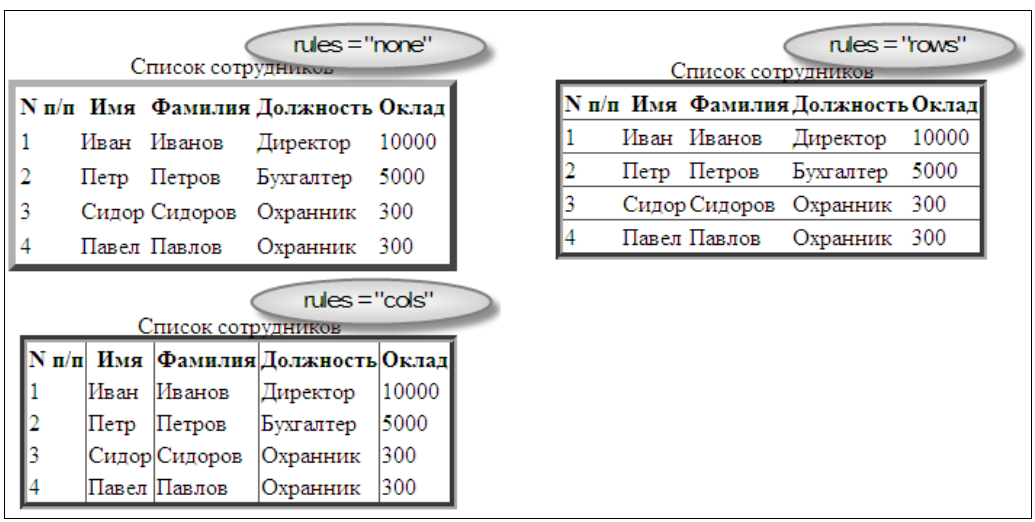


Рис. 10.4. Применение атрибута `rules` для управления отображением внутренних рамок

Примечание

Internet Explorer 6.0, а также версии 7 и 8 в режиме предоставления совместимости отображают двойные разделительные линии.

Для обеспечения межбраузерной совместимости управлять отображением рамок таблицы рекомендуется с помощью параметров CSS. В HTML 5 рассмотренные атрибуты рамок вообще не предусмотрены. Чтобы отобразить рамку, достаточно применить стилевой параметр `border` или более конкретные параметры этой же группы (`border-top`, `border-left`, `border-style` и т. п.), которые задают границу любого элемента, в том числе и `<table>`, `<td>` и `<th>`.

Установить двойные или одинарные разделительные линии можно с помощью стилевого параметра `border-collapse`, принимающего следующие значения:

- `collapse` — установить одинарные разделительные линии (т. е. для смежных ячеек отображать одну общую рамку);
- `separate` — установить двойные разделительные линии (т. е. для каждой ячейки отображать отдельную рамку).

Задание параметра `border-collapse` само по себе еще не приводит к отображению рамок. Чтобы рамки появились, необходимо еще указать их толщину, например `border: solid 1px`. На рис. 10.5 приведено несколько примеров управления отображением рамок посредством CSS.

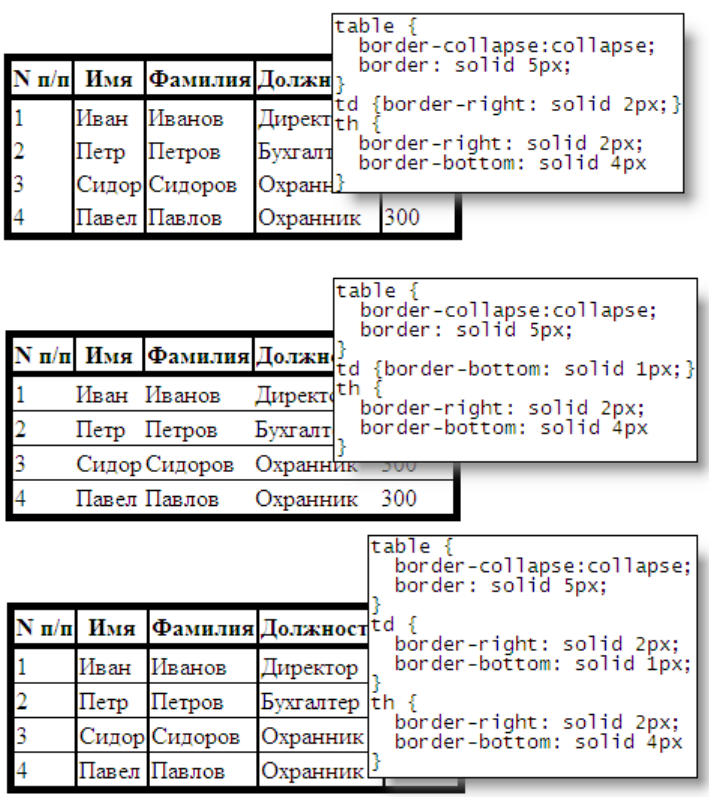


Рис. 10.5. Управление отображением рамок посредством CSS

Если задан параметр `border-collapse:separate` (установить собственные рамки или границы ячеек), то с помощью атрибута `cellspacing` тега `<table>` можно указать расстояние между границами ячеек, например,

```
<table cellspacing="10">
```

Для этой же цели служит стилевой параметр `border-spacing`, принимающий одно или два числовых значения с указанием размерности. Если указано единственное значение, то оно определяет расстояние между границами и по горизонтали, и по вертикали. Если указаны два значения (например, `border-spacing:10px 15px;`), то первое из них задает расстояние по горизонтали, а второе — по вертикали.

Попытка применить к ячейкам стилевой параметр `margin` (поле вокруг ячейки) с целью задать расстояние между их границами результата не даст.

> ***Примечание***
>
> Internet Explorer версий 7, 8 в режиме предоставления совместимости и более ранние версии не воспринимают параметр `border-spacing`.

10.3. Размеры таблицы

Ширину и высоту таблицы в целом и ее ячеек в частности можно регламентировать атрибутами `width` и `height` для соответствующих тегов. Атрибуты принимают целочисленные значения без указания размерности (подразумеваются пикселы), а также с указанием `%` или `auto` (размер определяется содержимым с учетом других факторов).

Пример:

```
<table width="80%" height="400">.
```

Возможно также применение стилевых параметров `width` и `height` с указанием единиц измерения. Для ячеек значения `width` в процентах задают их ширину относительно ширины таблицы, а для таблицы — относительно ширины содержащего ее контейнера (если это `<body>`, то относительно ширины клиентской области окна браузера).

Если размеры таблицы не указаны, то браузер вычисляет их в зависимости от содержимого ячеек и, возможно, с учетом других параметров. Даже если размеры таблицы заданы, то все равно нет абсолютной гарантии, что они будут в точности выдержаны браузером. В спецификации HTML 4.01 не определено, как браузер должен выводить на экран таблицу. Спецификация CSS 2 также не накладывает на алгоритм вывода таблицы ограничений, если только не задан стилевой параметр `text-layout:fixed`.

Возможные значения параметра `text-layout`:

- `auto` — алгоритм автоматического вывода таблицы, например, ширина таблицы определяется шириной ее столбцов, но браузеры вольны использовать свои "фирменные" алгоритмы, которые обычно сначала загружают всю таблицу с данными, анализируют ее и только затем отображают; данное значение принято по умолчанию;
- `fixed` — алгоритм фиксированного вывода; горизонтальный вывод таблицы зависит от ширины таблицы, ширины столбцов, а также от рамок и заполнения ячеек, но не зависит от содержимого ячеек (данных); таким образом, данный алгоритм сразу начинает вывод таблицы без предварительного анализа ее содержимого; если ширина

таблицы задана параметром `width` со значением `auto`, то используется алгоритм автоматического вывода; данный алгоритм работает быстрее, чем автоматический вывод;

- `inherit` — алгоритм вывода такой же, что и у родительской таблицы.

Здесь мы не будем приводить подробные описания довольно непростых алгоритмов вывода таблицы, а ограничимся лишь рассмотрением их сути.

Если вам необходимо возможно полнее контролировать размеры таблицы и к тому же отображать ее как можно быстрее, то следует явно задать ее размеры и применить алгоритм фиксированного вывода. Если задать ширину всех ячеек первого ряда, то алгоритм фиксированного вывода построит таблицу с учетом этих данных. При этом все столбцы будут иметь указанную ширину. Однако реальная ширина может отличаться от указанной параметром `width`. Она может быть скорректирована в зависимости от содержимого ячеек, ширины клиентской области окна браузера и других обстоятельств. Так, ячейка может быть расширена в случае переполнения, чтобы все ее содержимое было видно или же, наоборот, ширина ячейки сохраняется, а содержимое усекается. Например, Internet Explorer версий 7 и 8 в режиме предоставления совместимости в данном случае усекает содержимое ячеек, а остальные браузеры расширяют ячейки. Если ширина клиентской области браузера меньше ширины таблицы, то ячейки могут быть несколько сужены. Internet Explorer версий 7 и 8 в режиме предоставления совместимости не делает этого, а остальные браузеры уменьшают ширину ячеек.

Рассмотренное поведение ячеек характерно в случае применения параметра `table-layout:fixed` и явного задания ширины столбцов, отличной от `auto`. Если при этом задать еще и ширину таблицы в целом, отличную от `auto`, то реальная ширина таблицы и столбцов определится с учетом соотношения значений параметра `width` для этих элементов. Если значение `width` для таблицы (элемента `<table>`) больше суммы значений этого же параметра для столбцов, то реальная ширина таблицы будет равна значению `width`, а в противном случае — сумме значений `width` для столбцов. При этом еще учитывается ширина рамок. Когда заданы отличные от `auto` значения параметра `width` для таблицы и столбцов, ширина столбцов уже не зависит от содержимого ячеек и ширины клиентской области браузера, но при переполнении ячейки содержимое выходит за ее пределы, накладываясь на соседнюю ячейку. В Internet Explorer версий 7 и 8 в режиме предоставления совместимости вместо наложения переполняющего содержимого происходит его усечение. Впрочем, отображением содержимого, переполняющего ячейку, можно управлять стилевым параметром `overflow` со значениями `visible` (показать все) или `hidden` (усечь).

Высота таблицы зависит от количества рядов и высоты ячеек, а также от значения атрибута или стилевого параметра `height`. Высота ячейки, в свою очередь, зависит от высоты содержимого. Она всегда не меньше той величины, при которой содержимое полностью помещается в ячейке по высоте. Вы можете для ячейки задать атрибут или стилевой параметр `height`. Однако реальная высота ячейки вычисляется как наибольшее из значений `height` и высоты содержимого. Если `height` имеет значение `auto`, то выбирается высота содержимого. Такой выбор выполняется для всех ячеек одного и того же ряда и максимальная из полученных величин становится реальной высотой всех ячеек данного ряда. Например, если для ячейки указан параметр `height:30px`, а в ней расположена картинка высотой 50 px, то данная ячейка в действительности будет не ниже 50 px.

Таким образом (с учетом толщины рамок) вычисляется высота ряда. Реальная высота таблицы в целом определяется как наибольшее из суммы высот рядов и значения `height`, указанного для таблицы (элемента `<table>`).

Обобщая и упрощая ситуацию, можно сказать, что значение `height`, отличное от `auto`, может влиять на реальную высоту ячейки, если только оно не меньше минимальной величины, необходимой для размещения содержимого целиком. Этого еще недостаточно. Необходимо также, чтобы все ячейки в одном и том же ряду имели одинаковую высоту. Так что реальная высота ячейки может быть больше, чем указанное для нее значение параметра `height`. Далее, если сумма высот всех рядов окажется меньше значения параметра `height` для таблицы в целом, то разница равномерно распределится между всеми рядами или будет прибавлена к высоте только первого ряда. Так или иначе, реальная высота ячейки может стать еще больше.

В листинге 10.2 в качестве примера приведен код таблицы, а на рис. 10.6 — ее вид в окне браузера при различных значениях параметра `table:layout`. Данная таблица содержит текстовые данные, а в одной ячейке расположено еще и графическое изображение размером 128×96 px. Размеры таблицы и ячеек заданы стилевыми параметрами `width` и `heigh` для элементов `<table>` и `<th>` соответственно.

При фиксированном алгоритме вывода таблицы (`table:layout:fixed`) браузер пытается выдержать указанную ширину ячеек и таблицы. Однако в рассматриваемом примере размеры заданы неудачно и поэтому возникает переполнение ячеек, т. е. содержимое выходит за их границы. В случае автоматического вывода (`table-layout:auto`) ячейки увеличиваются так, чтобы они не переполнялись.

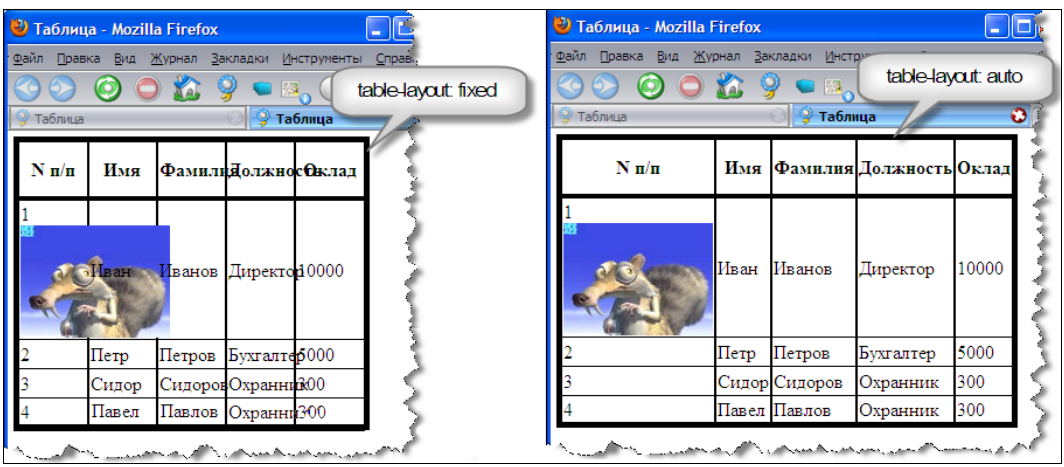


Рис. 10.6. Таблица при фиксированном и автоматическом алгоритмах вывода (листинг 10.2)

Листинг 10.2. Задание размеров таблицы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Таблица</title>
<style type="text/css">
table {
  table-layout:fixed;      /* auto*/
  border-collapse:collapse;
  border: solid 5px;
  width:300px;
  height:200px;
}
th {
  border-right: solid 2px;
  border-bottom: solid 4px;
  width:30px;
  height:50px
}
td {
  border-right: solid 2px;
  border-bottom: solid 1px;
  height:25px;
}
</style>
</head>
<body>
<table  >
    <tr>
      <th>N п/п</th>
      <th>Имя</th><th>Фамилия</th><th>Должность</th><th>Оклад</th>
    </tr>
    <tr>
      <td>1<img src="белка.jpg"/></td>
      <td>Иван</td><td>Иванов</td><td>Директор</td><td>10000</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Петр</td><td>Петров</td><td>Бухгалтер</td><td>5000</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Сидор</td><td>Сидоров</td><td>Охранник</td><td>300</td>
    </tr>
   <tr>
      <td>4</td>
      <td>Павел</td><td>Павлов</td><td>Охранник</td><td>300</td>
    </tr>
</table>
</body>
</html>
```

Чтобы исключить наложение данных соседних ячеек при фиксированном алгоритме вывода, следует правильно подобрать размеры ячеек. Кроме того, можно применить

параметр `overflow:hidden`, чтобы усечь не вмещающееся содержимое. Заметим, что `overflow` со значениями `scroll` и `auto` для ячеек не создает полосы прокрутки. На рис. 10.7 показана таблица с лучше подобранными размерами и фиксированным алгоритмом вывода. Данный вариант определения размеров далек от идеального, но достаточно поучителен. Обратите внимание, что реальная ширина таблицы в целом явно больше заданного для нее значения 300 px и близка к сумме значений `width` для столбцов (80 × 5 = 400). Если значение параметра `width` для таблицы задать больше суммарной ширины столбцов, то оно и определит реальную ширину таблицы. При этом ширина столбцов также пропорционально увеличится. Приблизительность оценок размеров обусловлена тем, что действительные размеры учитывают еще и толщину рамок, а также небольшие размеры отступов и полей, принятые по умолчанию.

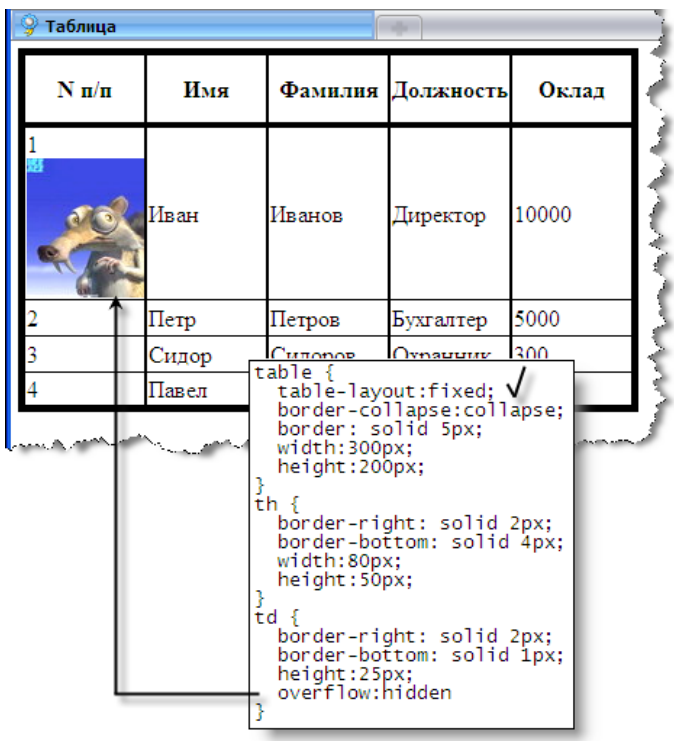


Рис. 10.7. Фиксированный вывод таблицы с заданной шириной столбцов. Ширина таблицы приблизительно равна суммарной ширине столбцов

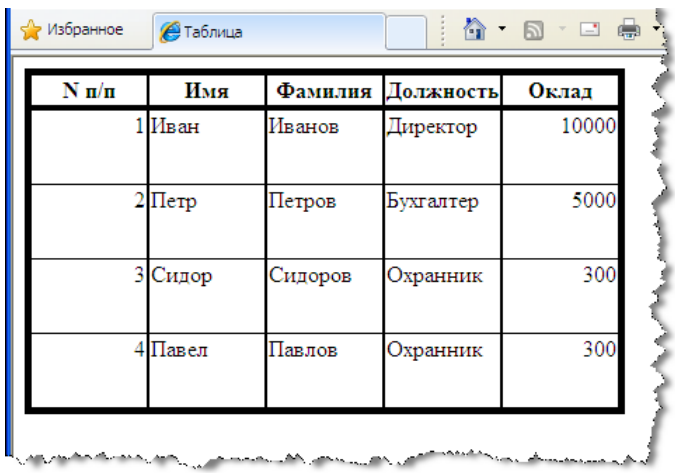
10.4. Выравнивание содержимого ячеек таблицы

Как уже отмечалось, по умолчанию содержимое ячеек данных (элементов `<td>`) выравнивается горизонтально по левому их краю, а вертикально центрируется; содержимое заголовков столбцов (элементов `<th>`) центрируется и горизонтально, и вертикально. Тем не менее часто требуется специфическое выравнивание содержимого ячеек. Например, числа обычно выравнивают по правому краю ячеек, а текст нередко требуется выровнять вертикально по верхнему краю ячейки.

Для горизонтального выравнивания содержимого служит стилевой параметр `text-align`. Он принимает значения `left` (по левому краю), `right` (по правому краю), `center` (по центру) и `justify` (одновременно по обоим краям) и применяется не только к текстовому содержимому.

Вертикальное выравнивание обеспечивает стилевой параметр `vertical-align`, который принимает значения `top` (по верхнему краю), `bottom` (по нижнему краю), `middle` (посередине) и `baseline` (по базовой линии) и др. (см. *разд. 8.4.2*).

В листинге 10.3 представлен код таблицы, в котором первый и последний столбцы с числами выровнены горизонтально по правому краю, а остальные столбцы — по центру. Вертикальное выравнивание содержимого всех ячеек выполнено по верхнему краю. Для горизонтального выравнивания по правому краю содержимого ячеек первого и последнего столбцов задан класс `.right {text-align:right}`, на который ссылаются все элементы `<td>` указанных столбцов. К сожалению, задание параметра `text-align` для ячеек только первого ряда таблицы не приведет к форматированию всех ячеек соответствующих столбцов. Вертикальное выравнивание одинаково для всех ячеек и поэтому оно задано правилом `vertical-align:top` с селектором `td`. Однако, в отличие от `text-align`, параметр `vertical-align` можно применить только к ячейкам первого ряда, чтобы выполнить вертикальное выравнивание всех ячеек соответствующих столбцов. Вид данной таблицы в окне браузера показан на рис. 10.8.



N п/п	Имя	Фамилия	Должность	Оклад
1	Иван	Иванов	Директор	10000
2	Петр	Петров	Бухгалтер	5000
3	Сидор	Сидоров	Охранник	300
4	Павел	Павлов	Охранник	300

Рис. 10.8. Выравнивание содержимого всех ячеек таблицы по верхнему краю, а первого и последнего столбцов — еще и по правому краю (листинг 10.3)

Листинг 10.3. Выравнивание содержимого ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Таблица</title>
<style type="text/css">
table {
  table-layout:fixed;
```

```
  border-collapse:collapse;
  border: solid 5px;
  width:400px;
}
th {
  border-right: solid 2px;
  border-bottom: solid 4px;
  width:80px;
}
td {
  border-right: solid 2px;
  border-bottom: solid 1px;
  height:50px;
  vertical-align:top      /* выравнивание по верхнему краю */
}
.right {text-align:right} /* выравнивание по правому краю*/

</style>
</head>
<body>
<table>
    <tr>
      <th>N п/п</th>
      <th>Имя</th><th>Фамилия</th><th>Должность</th><th>Оклад</th>
    </tr>
    <tr>
      <td class="right">1</td>
      <td>Иван</td><td>Иванов</td><td>Директор</td>
      <td class="right">10000</td>
    </tr>
    <tr>
      <td class="right">2</td>
      <td>Петр</td><td>Петров</td><td>Бухгалтер</td>
      <td class="right">5000</td>
    </tr>
    <tr>
      <td class="right">3</td>
      <td>Сидор</td><td>Сидоров</td><td>Охранник</td>
      <td class="right">300</td>
    </tr>
   <tr>
      <td class="right">4</td>
      <td>Павел</td><td>Павлов</td>
      <td>Охранник</td>
      <td class="right">300</td>
    </tr>
</table>
</body>
</html>
```

Для задания отступов от границ ячеек (расстояний между границами и содержимым) можно использовать параметр `padding`.

10.5. Задание параметров столбцов

Нередко требуется задать одинаковые параметры для всех ячеек одного и того же столбца таблицы, но при этом у двух или более столбцов должны быть разные параметры. Иначе говоря, часто возникает задача специфического форматирования ячеек столбцов. Чтобы задать параметры для всех ячеек столбца, в большинстве случаев достаточно назначить их только соответствующей ячейке первого ряда. Это можно сделать для параметров `width`, `vertical-align`, `padding` и многих других. Однако очень важный параметр `text-align` (горизонтальное выравнивание) будет действовать только на одну ячейку, к которой он применен. В подобных случаях приходится задавать параметр для каждой ячейки столбца, так как, например, в листинге 10.3.

Для задания параметров столбцов имеется нестандартный, но поддерживаемый всеми основными браузерами тег `<col/>`, вставляемый в контейнер `<table>` перед `<tbody>`. Он не создает в документе видимого элемента, а служит только для того, чтобы посредством специальных атрибутов или ссылки на таблицу стилей отформатировать все ячейки столбца. Иначе говоря, тегу `<col/>` можно назначить стилевые параметры или атрибуты форматирования, относящиеся к одному и тому же столбцу. В контейнер `<table>` можно вставить один или несколько тегов `<col/>`, каждый из которых соответствует столбцу в порядке упоминания в HTML-коде. Так, первый тег `<col/>` соответствует первому столбцу, второй тег `<col/>` — второму столбцу и т. д.

В листинге 10.4 приведен фрагмент кода, иллюстрирующий применение тега `<col/>`. Здесь форматируются первый и пятый столбцы, поэтому тег `<col/>` упоминается пять раз. Первый `<col/>` относится к первому столбцу, второй `<col/>` — ко второму и т. д. Первый и пятый `<col/>` ссылаются на соответствующие классы стилевых параметров, а для остальных тегов специфические правила форматирования не заданы, но они все равно присутствуют, чтобы можно было упомянуть пятый `<col/>` со ссылкой на специфические правила форматирования.

Листинг 10.4. Пример использования тега <col>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Таблица</title>
<style type="text/css">
.f1 {width:30px}  /* 1-й столбец *.
.f5 {width:50px}  /* 5-й столбец */
</style>
</head>
<body>
<table>
   <col class="f1" />
   <col/>
```

```
   <col/>
   <col/>
   <col class="f5"/>
    <tr>
      <th >N п/п</th>
      <th>Имя</th>
      <th>Фамилия</th>
      <th>Должность</th>
      <th>Оклад</th>
    </tr>
    <tr>
      <td >1</td>
      <td>Иван</td>
      <td>Иванов</td>
      <td>Директор</td>
      <td>10000</td>
    </tr>
    <!--здесь другие строки таблицы -->
</table>
</body>
</html>
```

Другой пример использования тегов `<col/>` приведен в *разд. 10.7*.

В теге `<col/>` можно указать следующие атрибуты, используемые вместо аналогичных стилевых параметров:

- `align` — горизонтальное выравнивание содержимого (аналог стилевого параметра `text-align`);
- `valign` — вертикальное выравнивание содержимого (аналог стилевого параметра `vertical-align`);
- `width` — ширина столбца; указывается число, возможно с символом % (аналог стилевого параметра `width`);
- `span` — количество объединяемых смежных столбцов; значение по умолчанию равно единице (т. е. столбец не объединяется ни с каким другим).

Для задания общих параметров для группы смежных столбцов служит контейнерный тег `<colgroup>`, в который включают теги `<col/>`, соответствующие столбцам группы. Листинг 10.5 иллюстрирует пример группировки первых двух и следующих трех столбцов.

Листинг 10.5. Пример группировки столбцов таблицы

```
<colgroup class="gr1">
  <col/>
  <col/>
</colgroup>
<colgroup class="gr2">
  <col/>
```

```
  <col/>
  <col/>
</colgroup>
```

Тег <colgroup>, так же как и <col>, нестандартный. Эти теги от Microsoft теперь кажутся архаичными, поскольку их можно заменить соответствующими правилами CSS. Но поскольку во многих случаях эти теги предоставляют некоторые удобства при форматировании столбцов, их поддерживают все основные браузеры.

> ***Примечание***
>
> Internet Explorer версий 7 и 8 в режиме предоставления совместимости, а также более ранние версии применяют параметр text-align ко всему столбцу при назначении его как первой ячейке ряда, так и соответствующему тегу <col>. Другие браузеры, а также Internet Explorer 7, 8 в обычном режиме не распространяют действие параметра text-align на весь столбец подобным образом. С другой стороны, атрибут align (аналог стилевого параметра text-align) тега <col/> работает только в Internet Explorer версий 7 и 8 в режиме предоставления совместимости, а также в более ранних версиях. Таким образом, при создании совместимого кода некоторые неудобства горизонтального выравнивания содержимого ячеек одного столбца сохраняются как при использовании тегов <col/>, так и без них.

10.6. Сложные таблицы

10.6.1. Расширение ячеек

Рассмотренные ранее таблицы просты в том смысле, что представляют собой прямоугольную сетку, клетки (ячейки) которой образуются в результате пересечения *m* вертикальных столбцов с *n* горизонтальными рядами. Усложнение таблицы возникает, когда объединяются несколько смежных ячеек простой таблицы. Как объединить две или более ячеек? В HTML это делается путем расширения некоторой ячейки за счет пространства, занимаемого в таблице соседними ячейками. В данной операции соседними могут быть ячейки справа и снизу. Соответственно, расширение данной ячейки может быть горизонтальным (захват пространства ячеек справа) и вертикальным (захват пространства ячеек снизу).

Для расширения ячейки в тегах <td> и <th> служат атрибуты:

- colspan — расширение ячеек по горизонтали (столбцам);
- rowspan — расширение ячеек по вертикали (строкам).

Эти атрибуты принимают целочисленные значения, которые указывают, на сколько ячеек требуется расширить данную ячейку. Расширяемая ячейка захватывает пространство смежных ячеек, вытесняя их вправо. Например, таблица на рис. 10.8 имеет повторяющиеся данные в последних двух строках и столбцах, а потому ячейки с одинаковыми данными можно объединить путем расширения первой из объединяемых ячеек. С этой целью следует для ячеек третьего ряда и последних двух столбцов указать атрибут rowspan="2" — вертикальное объединение двух ячеек, или, другими словами, вертикальное расширение на две ячейки. Кроме того, в исходном коде необходимо удалить ячейки, пространство которых захватывают расширяемые ячейки, поскольку иначе нарушится структура таблицы. В данном примере следует удалить из исходной таблицы ячейки четвертого ряда столбцов "Должность" и "Оклад". В листинге 10.6 приведен код, а на рис. 10.9 — вид таблицы в окне браузера.

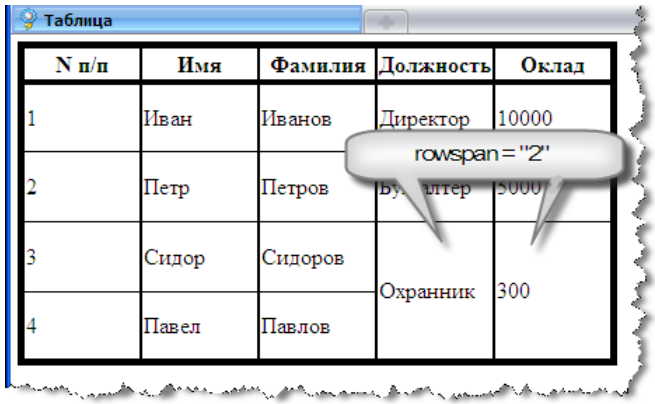


Рис. 10.9. Вертикальное расширение ячеек посредством атрибута `rowspan` (листинг 10.6)

Листинг 10.6. Вертикальное расширение ячеек

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Таблица</title>
<style type="text/css">
table {
  table-layout:fixed;
  border-collapse:collapse;
  border: solid 5px;
  width:400px;
}
th {
  border-right: solid 2px;
  border-bottom: solid 4px;
  width:80px;
}
td {
  border-right: solid 2px;
  border-bottom: solid 1px;
  height:50px;
}
</style>
</head>
<body>
<table>
   <tr>
      <th>N п/п</th>
      <th>Имя</th>
      <th>Фамилия</th>
      <th>Должность</th>
      <th>Оклад</th>
    </tr>
```

```
    <tr>
      <td>1</td>
      <td>Иван</td>
      <td>Иванов</td>
      <td>Директор</td>
      <td>10000</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Петр</td>
      <td>Петров</td>
      <td>Бухгалтер</td>
      <td>5000</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Сидор</td>
      <td>Сидоров</td>
      <td rowspan="2">Охранник</td>
      <td rowspan="2">300</td>
    </tr>
   <tr>
      <td>4</td>
      <td>Павел</td>
      <td>Павлов</td>
   </tr>
</table>
</body>
</html>
```

Теперь рассмотрим построение таблицы с четырьмя ячейками, структура которой показана на рис. 10.10. Подобные таблицы довольно часто используют в качестве каркаса Web-страниц. Очевидно, данная таблица содержит три ряда, причем ячейки первого и последнего рядов расширены горизонтально на две ячейки (`colspan="2"`). Во втором ряду находятся две ячейки разной ширины. Задание параметра `width` для каждой из них не даст нужного результата, если применяется алгоритм фиксированного вывода (`table-layout:fixed`). Этот алгоритм хорошо работает, если заданы размеры ячеек первого ряда. Но в нашем примере этого нельзя сделать, поскольку в первом ряду задана единственная ячейка, хотя она и расширяется на две. При `table-layout:fixed` две ячейки во втором ряду будут иметь одинаковую ширину. Чтобы сделать их разными по ширине, необходимо прежде задать алгоритм автоматического вывода таблицы (`table-layout:auto`), при котором произойдет более глубокий анализ содержимого тега `<table>`. Код, создающий требуемую таблицу, представлен в листинге 10.7. Видимые границы между ячейками таблицы установлены только ради наглядности.

Листинг 10.7. Таблица для каркаса Web-страниц

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Таблица</title>
<style type="text/css">
table {
  table-layout:auto;
  border-collapse:collapse;
  border: solid 1px;
  width:300px;
}
td {border-right: solid 1px;border-bottom:solid 1px}
#header {height:50px}
#box {height:200px}
#column1 {width:100px}
#footer  {height:30px}
</style>
</head>
<body>
<table  >
    <tr id="header">
      <td colspan="2">header (Заголовок)</td>
    </tr>
    <tr id="box">
      <td id="column1">column1 (Колонка 1)</td>
      <td id="column2">column2 (Колонка 2)</td>
    </tr>
    <tr id="footer">
      <td colspan="2">footer (Подвал)</td>
    </tr>
</table>
</body>
</html>
```

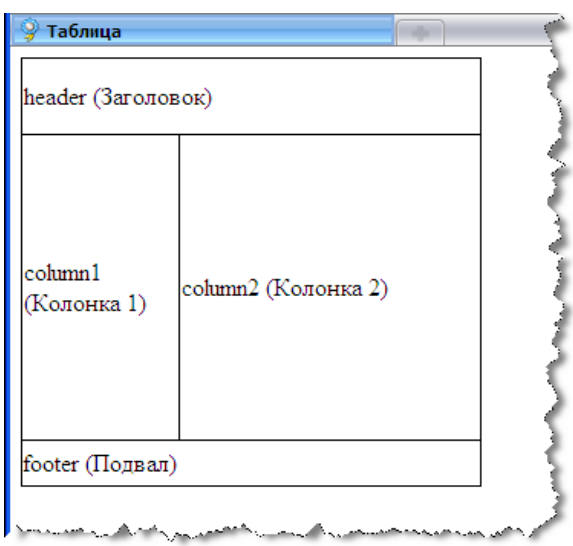


Рис. 10.10. Таблица, нередко используемая в качестве каркаса Web-страниц (листинг 10.7)

Для горизонтального расширения столбцов кроме атрибута `colspan` во всех тегах `<td>` и `<th>` расширяемого столбца можно указать атрибут `span` тега `<col>`, например, `<col span="3"/>` (см. *разд. 10.5*).

10.6.2. Прокручиваемая таблица

Еще более сложные по структуре таблицы можно получить, вставляя в ячейки другие таблицы. В качестве примера рассмотрим создание прокручиваемой таблицы, содержащей много рядов. Разумеется, такую таблицу можно заключить в контейнер `<div>` с необходимой высотой и стилевым параметром `overflow:auto`, задающим полосы прокрутки в случае необходимости. Однако в этом случае прокручиваться будет вся таблица, включая заголовки столбцов. Тем не менее желательно, чтобы положение заголовков оставалось фиксированным, а прокручивались только ряды с данными. Данную задачу можно решить следующим способом.

Создадим таблицу с заголовками столбцов и с единственным рядом данных, содержащим единственную ячейку, расширенную на количество столбцов. В эту ячейку вставим прокручиваемый контейнер `<div>`, а в него — таблицу, содержащую только ряды с данными. Далее останется только согласовать размеры внешней и внутренней таблиц.

В листинге 10.8 приведен код прокручиваемой таблицы, а на рис. 10.11 — ее вид в окне браузера. В данном примере таблица содержит пять столбцов одинаковой ширины 80 px, а параметр `width` таблицы устанавливается равным 80 × 5 = 400 px. Ширина контейнера `<div>` с другой таблицей несколько больше (420 px), чтобы полоса вертикальной прокрутки оказалась прижатой к правому краю внешней таблицы. При этом в Internet Explorer, Firefox и Opera полоса прокрутки будет расположена внутри таблицы, а в Safari и Chrome — снаружи. В данном примере важную роль играет параметр `text-layout:fixed`. Параметры прокручиваемого элемента `<div>`, содержащего таблицу с данными, указаны в классе `.scrolling`, в котором за возможность вертикального прокручивания таблицы отвечает параметр `overflow-y:auto`.

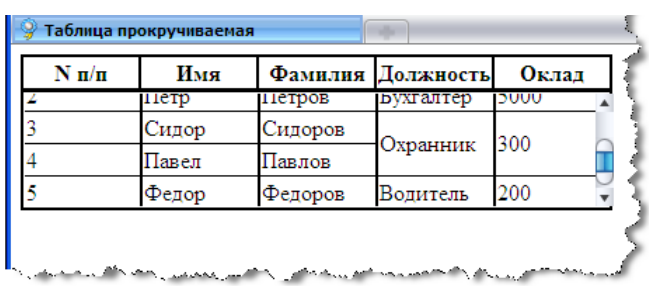


Рис. 10.11. Прокручиваемая таблица (листинг 10.8)

Листинг 10.8. Прокручиваемая таблица

```
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Таблица прокручиваемая</title>
<style type="text/css">
table {
  table-layout:fixed;
  border-collapse:collapse;
  border: solid 2px;
  width:400px
}
td {
  border-right: solid 2px;
  border-bottom:solid 1px;
}
th {
  border-right: solid 2px;
  border-bottom: solid 3px;
  width:80px
}
.scrolling { /* прокручиваемый блок*/
  width:420px;
  height: 80px;
  overflow-y: auto;
  overflow-x: hidden
}
table table {border:0}
table table td {width:80px}
</style>
</head>
<body>
<table>
  <tr>
      <th>N п/п</th>
      <th>Имя</th>
      <th>Фамилия</th>
      <th>Должность</th>
      <th>Оклад</th>
  </tr>
  <tr>
   <td colspan="5">
    <div class="scrolling">
    <table>
    <tr>
      <td >1</td>
      <td>Иван</td>
      <td>Иванов</td>
      <td>Директор</td>
      <td>10000</td>
    </tr>
    <tr>
      <td>2</td>
```

```
      <td>Петр</td>
      <td>Петров</td>
      <td>Бухгалтер</td>
      <td>5000</td>
    </tr>
    <!-- Здесь находятся другие ряды данных -->
   </table>
   </div>
  </td>
 </tr>
</table>
</body>
</html>
```

10.7. Декорирование таблицы

По умолчанию фон таблицы прозрачен, но его можно задать стилевым параметром `background` и другими параметрами этой же группы (`background-color`, `background-image` и т. п.). Фон можно установить для таблицы в целом (`<table>`), для отдельных ее ячеек (`<td>` и `<th>`) и рядов (`<tr>`). Стилевые параметры для задания фона элементов подробно рассматривались в *главе 5*.

Цвет и стиль рамок (границ) таблицы в целом и ее ячеек задает параметр `border` и другие параметры этой же группы (`border-color`, `border-style` и т. п.). При этом можно оформить отдельно каждую из четырех сторон прямоугольной рамки. Например, для задания нижней части рамки в виде штриховой красной линии толщиной 2 px можно использовать параметр `border-bottom: dashed 2px red` или же группу таких параметров:

```
border-bottom-style:dashed;
border-bottom-color:red;
border-bottom-width:2px;
```

Более подробно стилевые параметры границ элементов рассматривались в *разд. 3.7*.

На рис. 10.12 показан пример таблицы с выделением ряда и ячейки, на которых находится указатель мыши. Выделение элементов обеспечивается стилевыми параметрами для селекторов `tr:hover` и `td:hover`. Браузеры Internet Explorer, Firefox и Opera показывают данный эффект с сохранением внешней рамки таблицы, а Safari и Chrome не отображают внешнюю рамку на участке, соответствующем выделенному ряду.

Нередко ряды или столбцы таблиц через один выделяют цветом. В CSS можно создать класс с параметрами выделения, а затем в каждом элементе `<tr>`, требующем выделения, посредством атрибута `class` указать на него ссылку. Для выделения столбцов можно применить теги `<col/>` со ссылкой на класс параметров выделения (рис. 10.13).

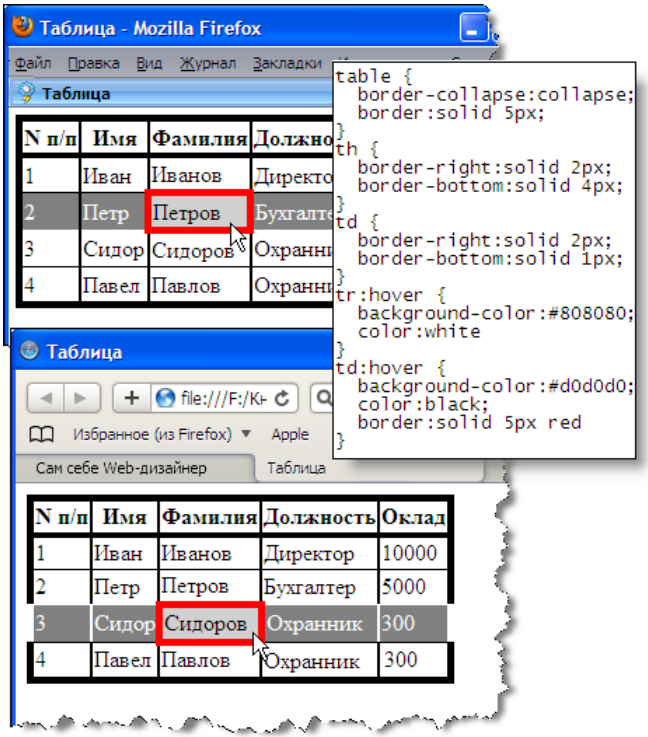


Рис. 10.12. Выделение ряда и ячейки таблицы под указателем мыши в браузерах Firefox (наверху) и Safari (внизу)

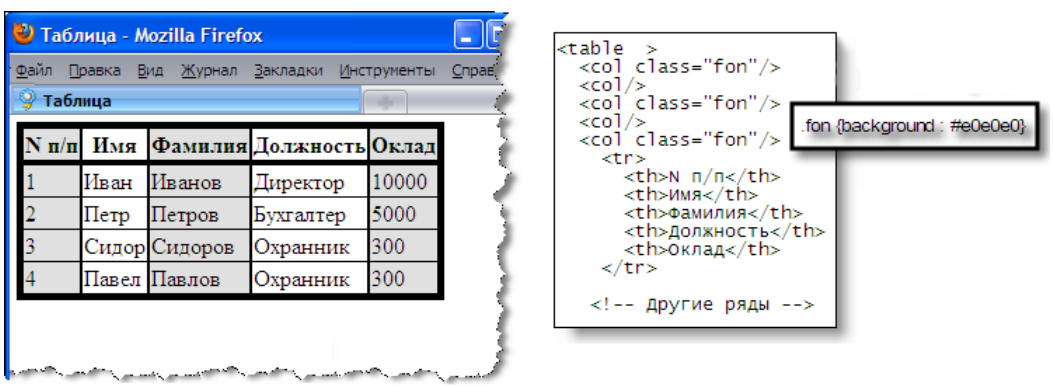


Рис. 10.13. Выделение столбцов

ГЛАВА 11

Формы и элементы пользовательского интерфейса

На многих Web-страницах встречаются элементы пользовательского интерфейса: поля ввода данных, кнопки, переключатели и др. Внешний вид этих элементов обычно соответствует их назначению. Так, в поле ввода данных можно ввести символы посредством клавиатуры, а на кнопку можно навести указатель мыши и щелкнуть ее левой клавишей. Что при этом произойдет? Введенные данные будут сохранены, но лишь до тех пор, пока пользователь находится на выбранной странице; кнопка визуально "утопится", создав впечатление, что она среагировала на щелчок мышью. Это — минимум из того, что можно получить. Действия пользователя можно как-то обработать с помощью сценариев, написанных, например, на языке JavaScript. Но что можно сделать без сценариев? Средства (X)HTML позволяют отправить данные на сервер для того, чтобы серверное программное обеспечение их как-то обработало или, в простейшем случае, просто сохранило на своем, удаленном серверном диске. Для выполнения этой процедуры со стороны клиента (документа, загруженного в браузер) ничего другого, кроме (X)HTML, не требуется.

Для обеспечения пересылки данных, введенных пользователем в соответствующие элементы на Web-странице и/или сгенерированных программно с помощью сценариев, достаточно создать так называемую форму. В (X)HTML-документе для этого предусмотрен контейнерный тег `<form>`, внутри которого можно расположить один или несколько тегов, создающих элементы пользовательского интерфейса — поля ввода данных, переключатели и кнопки. При соблюдении некоторых простых правил создания формы данные, находящиеся в ее элементах, можно передать серверу.

Элементы пользовательского интерфейса могут существовать и вне формы. Например, вы можете применить кнопку для инициализации ссылки, а поле ввода данных — для организации диалога с пользователем. В последнем случае вам потребуется некоторый сценарий (программа на JavaScript) для обработки введенных данных.

11.1. Поля ввода, кнопки и переключатели: тег *<input>*

С помощью неконтейнерного внутристрочного тега `<input>` можно создавать элементы различных типов: поля ввода данных, кнопки и переключатели (флажки и радиокнопки). Тип элемента определяется атрибутом `type`. В зависимости от типа элемента могут

использоваться те или иные дополнительные атрибуты. Некоторые типы предусмотрены специально для передачи на сервер данных, ассоциированных с элементами. Например, кнопка специального типа `submit` инициализирует передачу данных, если эта кнопка и элементы, содержащие данные, находятся в одном контейнерном теге формы `<form>` с соответствующими значениями атрибутов (см. *разд. 11.6*). Данные, ассоциированные с элементами, — это значения атрибута `value` тегов, задающих элементы пользовательского интерфейса.

На рис. 11.1 показано, как выглядят в окне браузера элементы пользовательского интерфейса, заданные тегом `<input>` при различных значениях атрибута `type`.

Атрибут `type` тега `<input>` может принимать следующие значения:

- `text` — текстовое поле ввода данных; дополнительные атрибуты:
  - `maxlength` — максимальное число вводимых символов (по умолчанию — не ограничено);
  - `size` — ширина поля, выраженная в количестве одновременно видимых символов;
  - `value` — значение (содержимое поля);

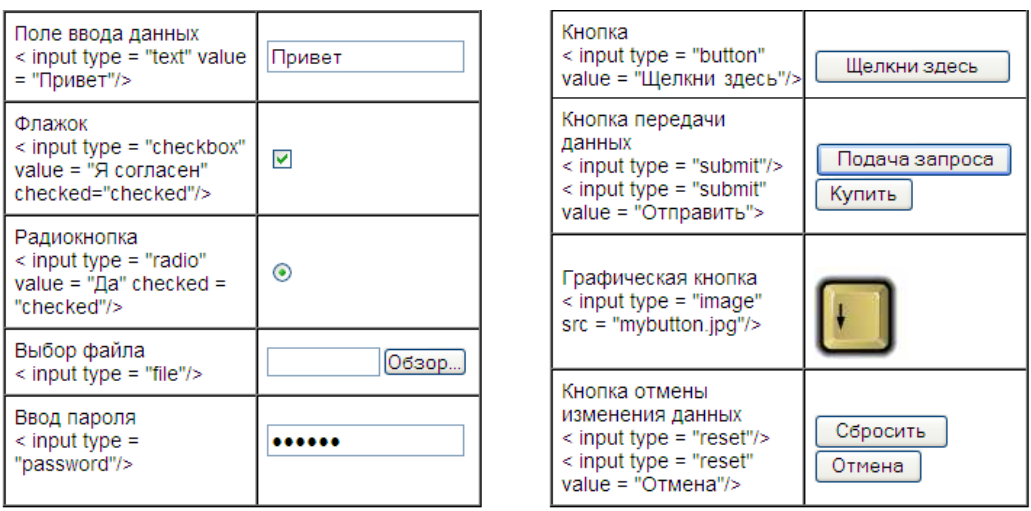


Рис. 11.1. Элементы, заданные тегом `<input>`

- `checkbox` — переключатель типа флажок; дополнительные атрибуты:
  - `value` — значение, ассоциированное с флажком;
  - `checked="checked"` — если указан, то флажок установлен;
- `radio` — селекторный переключатель (радиокнопка); дополнительные атрибуты:
  - `value` — значение, ассоциированное с переключателем;
  - `checked="checked"` — если указан, то переключатель установлен (выбран);
- `file` — текстовое поле ввода данных с кнопкой **Обзор**, щелчок на которой открывает диалоговое окно выбора файла; полное имя выбранного файла отображается в

поле ввода. Дополнительные атрибуты `maxlength`, `size` и `value` такие же, что и для обычного поля ввода данных, задаваемого атрибутом `type="text"`;

- `password` — текстовое поле ввода данных, в котором все символы отображаются в виде "звездочки" (`*`) или закрашенного кружка. Дополнительные атрибуты `maxlength`, `size` и `value` те же, что и для обычного поля ввода данных, задаваемого атрибутом `type="text"`. Обычно в такое поле вводят пароль. Однако, хотя вводимые в поле типа `password` символы и маскируются "звездочками", передача их на сервер не шифруется;
- `button` — кнопка; дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке нет надписи;
- `submit` — кнопка, щелчок на которой приводит к отправке данных, содержащихся в форме, на сервер; дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке находится надпись **Подача запроса**, **Отправить запрос**, **Начали** или **Submit** в зависимости от браузера. Передача данных произойдет только при определенных условиях (см. *разд. 11.6*);
- `image` — графическое изображение, играющее такую же роль, что и кнопка типа `submit`; адрес графического файла указывается как значение атрибута `src`;
- `reset` — кнопка, щелчок на которой отменяет все изменения в элементах, содержащихся в форме (восстанавливает исходные значения); дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке находится надпись **Сброс**, **Сбросить** или **Reset** в зависимости от браузера;
- `hidden` — невидимый элемент; пользователь не может изменить данные, сохраняемые как значение атрибута `value`. Если этот элемент находится в форме, то его данные можно передать на сервер. Данные (значение атрибута `value`) невидимого элемента могут быть сформированы сценарием.

В HTML 5 для атрибута `type` предполагается ввести новые значения, специфицирующие вводимые значения: `number`, `range`, `email`, `url`, `time` и др. Например, элемент `<input type="number" min="2" max="10"/>` определяет поле ввода чисел в диапазоне от 2 до 10; в элемент `<input type="email" value="`*список_адресов*`"/>` можно вводить строки, имеющие формат адреса электронной почты. Применение новых типов элемента `<input>` позволит контролировать введенные пользователем данные без помощи скриптов.

Кроме рассмотренных, тег `<input>` имеет следующие специфические атрибуты:

- `name` — имя (идентификатор) элемента. Если предполагается использовать элемент в составе формы для передачи его данных (значения атрибута `value`) на сервер, то указание атрибута `name` обязательно;
- `disabled="disabled"` — делает элемент недоступным пользователю;
- `readonly="readonly"` — делает элемент недоступным пользователю;
- `accesskey` — определяет клавишу быстрого доступа к элементу; значение атрибута — строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, `<input type="button" value="OK" accesskey="Y">`;

- `tabindex` — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно указывается, когда в форме присутствует несколько элементов интерфейса, а порядок ввода данных в них важен.

В HTML 5 предполагается ввести ряд новых атрибутов, например, `autocomplete` (установка режима автозаполнения поля), `autofocus` (автоматическая установка фокуса на элементе), `form` (идентифицирует форму, с которой ассоциируется элемент), `required` (определяет, что поле обязательно должно быть заполнено), `pattern` (определяет регулярное выражение, которому должны соответствовать вводимые данные) и др.

В отличие от других рассмотренных элементов, селекторные переключатели (радиокнопки) не используются в одиночку. Два или более таких переключателей объединяют в группу, чтобы они действовали совместно. В одной группе из нескольких радиокнопок только одна из них может находиться в установленном состоянии. Щелчок на другой, неотмеченной радиокнопке из той же группы устанавливает ее и одновременно сбрасывает ранее отмеченную кнопку. Чтобы организовать группу радиокнопок, достаточно сделать одинаковыми их имена (значения атрибута `name`). В одном и том же документе может быть несколько групп радиокнопок. Обычно (но не обязательно) радиокнопки из одной группы находятся рядом друг с другом.

> ***Примечание***
>
> Тег `<input>`, как и все другие, имеет необязательный атрибут `id`, значением которого является строка, содержащая идентификатор элемента (подобно имени переменной). Идентификатор обеспечивает в сценариях доступ к объекту, соответствующему этому элементу документа. Атрибут `name` необходим для организации отправки данных элемента (значения атрибута `value`) на сервер. В сценарии можно получить доступ к элементу и по значению `name`, но иначе, чем по значению `id`. Поэтому, чтобы не запутаться, значения атрибутов `id` и `name` устанавливают одинаковыми, хотя это и не обязательно. Это же замечание касается тега `<form>` и других тегов, предназначенных для задания элементов формы (`<select>` и `<textarea>`).

11.2. Кнопка: тег *<button>*

Создать кнопку можно не только тегом `<input type="button"...>`, но и с помощью контейнерного внутристрочного тега `<button>`. Поскольку это контейнерный тег, в него допустимо вставлять другие элементы, например обычный текст и заголовки, графические изображения и даже таблицы (рис. 11.2).

Кнопка автоматически принимает размеры, достаточные для размещения на ней элементов, заданных внутри тега `<button>`. Если надписи и графические изображения невелики, то различные браузеры отображают кнопку примерно одинаково. При объемном содержимом тега `<button>`, а также при задании размеров кнопки с помощью таблицы стилей различия в отображении кнопки разными браузерами становятся весьма заметными.

В листинге 11.1 приведен HTML-код, создающий две кнопки посредством тега `<button>` с одинаковым содержимым (текстом и картинкой). Первая кнопка не имеет параметров CSS, и ее размеры устанавливаются автоматически, а для второй заданы размеры, цвет фона и размер шрифта. Вид этих кнопок в браузере Firefox показан на рис. 11.3. Internet Explorer 8.0 в стандартном режиме и Firefox отображают кнопки одинаково, причем так

же, как и браузеры Safari и Chrome. Opera показывает не вписывающуюся в границы кнопки картинку полностью. Впрочем, различия легко устраняются, если задать для кнопки параметр `overflow:hidden` или, наоборот, `overflow:visible`. Однако Internet Explorer 8.0 в режиме предоставления совместимости никак не отреагирует на это. Если же задать параметр `overflow:auto`, то Internet Explorer в стандартном режиме, Safari и Chrome покажут кнопку с фиксированными размерами одинаково с полосами прокрутки, а Firefox, Opera и Internet Explorer в режиме предоставления совместимости — нет.

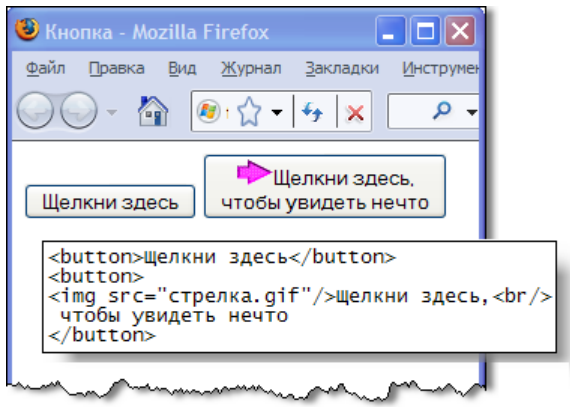


Рис. 11.2. Примеры кнопок

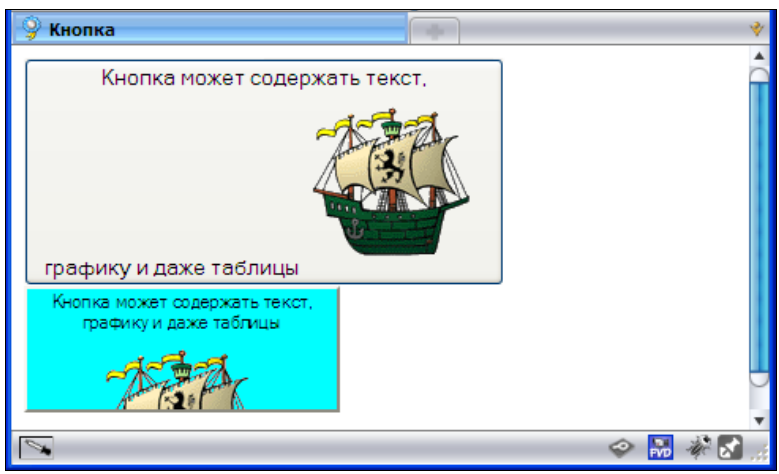


Рис. 11.3. Отображение документа листинга 11.1 в Firefox

Листинг 11.1. Создание кнопки

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd"><html>
<head><title>Кнопка</title>
<style type="text/css">
```

```
  #btn {
  width:200px;height:80px;
  background:#00ffff;font-size:8pt
}
</style>
</head>
<body>
<button>
Кнопка может содержать текст,<br/>
графику и даже таблицы<img src="корабль.gif"/>
</button>
<br/>
<button id="btn">
Кнопка может содержать текст,<br/>
графику и даже таблицы<img src="корабль.gif"/>
</button>
</body>
</html>
```

Чтобы добиться примерно одинакового вида больших кнопок в различных браузерах, потребуются некоторые усилия при задании подходящих значений параметров CSS. Впрочем, типичный вариант кнопки (короткая надпись и, возможно, маленькое графическое изображение) отображается почти одинаково всеми основными браузерами.

Как уже отмечалось, в контейнерный тег `<button>` можно вставлять другие теги, но, к сожалению, не все. Так, вы можете вставлять заголовки (`<h1>`, `<h2>`,...), абзацы (`<p>`), переходы на новую строку (`<br/>`), изображения (`<img.../>`) и таблицы (`<table>`). Элементы пользовательского интерфейса, например, не следует вставлять в кнопку потому, что большинство браузеров не отобразят такое "чудовищное" построение. Однако Safari и Chrome сделают это, будучи наиболее последовательными в интерпретации контейнерных тегов. На рис. 11.4 показана в окне Safari сложная кнопка (`<button>`), со-

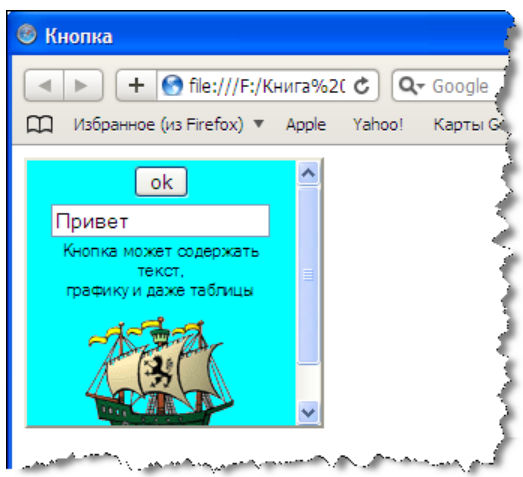


Рис. 11.4. Отображение сложной кнопки браузером Safari

держащая текст, графическое изображение и два элемента пользовательского интерфейса (<input>).

Основные атрибуты тега <button>:

- accesskey — определяет клавишу быстрого доступа к кнопке; значение атрибута — строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, <button accesskey="N">Щелкни здесь</button>;
- disabled="disabled" — делает кнопку недоступной пользователю;
- type — тип кнопки, принимает значения:
  - button — обычная кнопка;
  - reset — кнопка для отмены всех изменений и восстановления исходных значений всех элементов формы, в которой данная кнопка находится;
  - submit — кнопка для отправки данных в элементах формы, в которой данная кнопка находится;
- name — имя (идентификатор) элемента. Если предполагается использовать элемент в составе формы для передачи его данных (значения атрибута value) на сервер, то указание атрибута name обязательно;
- value — значение, ассоциированное с данной кнопкой (передается на сервер при отправке данных формы).

11.3. Комбинированный раскрывающийся список: тег <*select*>

Тег <select> позволяет создать раскрывающийся комбинированный список, из которого можно выбрать элемент. Это контейнерный тег, внутри которого находятся теги <option>, предназначенные для задания элементов (опций) списка.

Тег <select> имеет множество атрибутов, среди которых отметим основные:

- accesskey — определяет клавишу быстрого доступа к раскрывающемуся списку; значение атрибута — строка, содержащая букву или цифру. Если нажать на указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, <select accesskey="S">;
- name — имя (идентификатор) списка;
- disabled="disabled" — делает список недоступным пользователю;
- multiple="multiple" — обозначает возможность выбора из списка одновременно нескольких элементов;
- size — число одновременно видимых элементов списка; по умолчанию — 1. Если это число больше единицы, то список снабжается полосой прокрутки;
- tabindex — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно используется, когда в форме присутствуют несколько элементов интерфейса, а порядок ввода данных в них важен.

Атрибуты тега `<option>`:

- `selected="selected"` — обозначает выбранный (выделенный) элемент списка;
- `value` — значение, ассоциированное с элементом списка.

Текстовое содержимое элемента списка указывается сразу после тега `<option>`.

Листинг 11.2 содержит три комбинированных раскрывающихся списка (рис. 11.5).

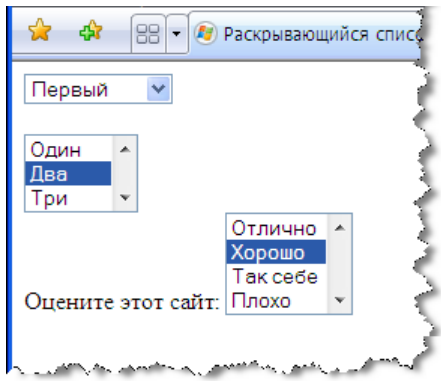


Рис. 11.5. Раскрывающиеся списки (листинг 11.2)

Листинг 11.2. Комбинированные раскрывающиеся списки

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd"><html>
<head><title>Раскрывающийся список</title><head>
<body>
  <select name="List1">
    <option value="1"/>Первый
    <option value="2"/>Второй
    <option value="3"/>Третий
    <option value="4"/>Четвертый
  </select>
  <br/><br/>
  <select name="List2" size=3>
    <option value="Один"/>Один
    <option selected="selected" value="Два"/>Два
    <option value="Три"/>Три
    <option value="четыре"/>Четыре
  </select>
  <br/>
  Оцените этот сайт:
  <select name="List3" size=4>
    <option value="5"/>Отлично
    <option selected="selected" value="4"/>Хорошо
    <option value="3"/>Так себе
    <option value="0"/>Плохо
```

```
  </select>
</body>
</html>
```

На рис. 11.6 показаны раскрывающиеся списки, созданные кодом листинга 11.2, но с добавлением таблицы стилей.

Атрибут `value` предназначен для хранения значений, ассоциированных с элементами списка. Элемент списка на экране и ассоциированное с ним значение могут быть как одинаковыми, так и различными. Если раскрывающийся список находится в форме (тег `<form>`), то при отправке данных формы на сервер передается значение атрибута `value` выбранного элемента.

Создать иерархические списки можно с помощью контейнерного тега `<optgroup>`, в который заключают теги `<option>` элементов списка, объединяемых в одну группу. Атрибут `label` тега `<optgroup>` принимает в качестве значения текстовую строку, отобра-

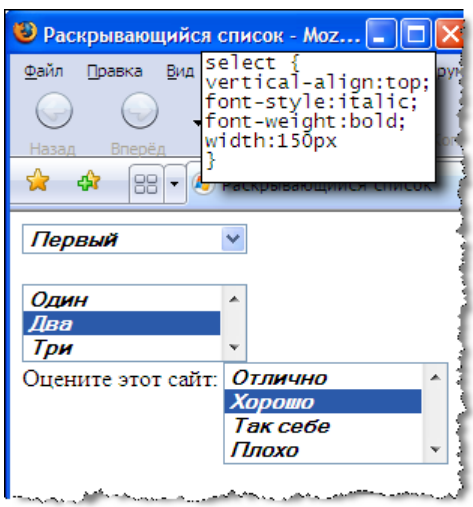


Рис. 11.6. Раскрывающиеся списки, отформатированные с помощью таблицы стилей

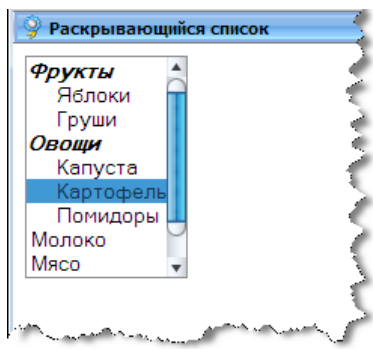


Рис. 11.7. Двухуровневый раскрывающийся список (листинг 11.3)

жаемую в списке жирным курсивом (Opera отображает обычным шрифтом, но белым на черном фоне). Это невыбираемый элемент списка, выполняющий роль заголовка группы элементов. При этом выбираемые элементы группы, заданные посредством тега `<option>`, имеют отступ слева.

(X)HTML-код, приведенный в листинге 11.3, создает двухуровневый раскрывающийся список (рис. 11.7).

Листинг 11.3. Двухуровневый раскрывающийся список

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
```

```
<head><title>Раскрывающийся список</title><head>
<body>
  <select name="mylist" size=9>
    <optgroup label="Фрукты">
      <option value="1"/>Яблоки
      <option value="2"/>Груши
    </optgroup>
    <optgroup label="Овощи">
      <option value="3"/>Капуста
      <option value="4" selected="selected"/>Картофель
      <option value="5"/>Помидоры
    </optgroup>
    <option s value="6"/>Молоко
    <option value="7"/>Мясо
    <option value="8"/>Пиво
  </select>
</body>
</html>
```

Выбор пользователем элемента из раскрывающегося списка можно обработать с помощью сценария.

11.4. Текстовая область: тег *<textarea>*

Текстовая область — это прямоугольное поле с полосами прокрутки, внутри которого можно разместить многострочный текст. Его можно сделать как редактируемым, так и доступным только для просмотра. Для создания текстовой области применяют контейнерный внутристрочный тег `<textarea>`. Важная особенность этого тега в том, что все теги, заключенные в него, не выполняются браузером, а отображаются как обыкновенный текст.

Основные атрибуты тега `<textarea>`:

- `accesskey` — определяет клавишу быстрого доступа к текстовой области; значение атрибута — строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с <Alt>, то элемент получит фокус. Например, `<textarea accesskey="T">`;
- `cols` — число символьных позиций по горизонтали для отображения текста;
- `rows` — число строк, одновременно видимых в окне текстовой области;
- `readonly="readonly"` — устанавливает текстовую область в режим просмотра (только для чтения); если этот атрибут опустить, то текстовая область будет работать как поле ввода и редактирования данных;
- `name` — имя (идентификатор) текстовой области;
- `disabled="disabled"` — делает область недоступной пользователю;
- `tabindex` — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно необходимо, когда в форме присутствуют несколько элементов интерфейса, а порядок ввода данных в них важен;

- `wrap` — определяет режим автоматического переноса символов на другую строку; возможные значения: `physical`, `virtual` и `off` (выключено). По умолчанию включен автоматический перенос посимвольно.

В примере из листинга 11.4 создаются две текстовые области. Первая из них содержит обыкновенный текст, а вторая — HTML-теги, которые только отображаются, но не интерпретируются браузером (рис. 11.8).

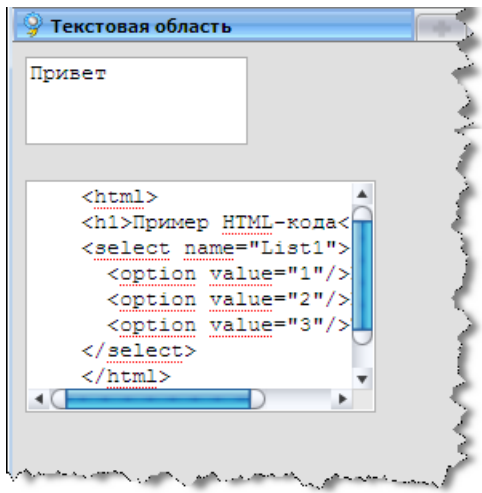


Рис. 11.8. Текстовые области

Листинг 11.4. Текстовые области

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
  <head><title>Текстовая область</title><head>
  <body style="background:#e0e0e0">
    <textarea cols=15 rows=5>
        Привет
    </textarea>
    <br/><br/>
    <textarea cols=25 rows=8 wrap=off>
    <html>
    <h1>Пример HTML-кода</h1>
    <select name="List1">
      <option value="1"/>Первый
      <option value="2"/>Второй
      <option value="3"/>Третий
    </select>
    </html>
    </textarea>
  <body>
</html>
```

Интересно, что Safari и Chrome, в отличие от других браузеров, позволяют изменять размеры текстовой области с помощью мыши.

11.5. Декорации элементов интерфейса

Для декорирования элементов пользовательского интерфейса можно использовать контейнерные теги `<fieldset>`, `<legend>` и `<label>`.

Тег `<fieldset>` служит для создания выделенного рамкой блока элементов интерфейса. В нем может находиться тег `<legend>`, содержимое которого отображается в верхней части блока (на рамке). Обычно это текстовый заголовок группы элементов, хотя можно использовать графические изображения и другие элементы. В примере из листинга 11.5 блок содержит три радиокнопки (рис. 11.9).

Листинг 11.5. Блок радиокнопок

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Блок</title></head>
  <fieldset>
    <legend>Выберите продукт</legend>
    <input type=radio name=product value="auto"/>Автомобиль
      <img src="auto.gif"/><br/>
    <input type=radio name=product value="flowers"
      checked="checked">Букет
      <img src="flowers.jpg"><br>
    <input type=radio name=product value="nothing"/>Ничего не надо
  </fieldset>
<html>
```

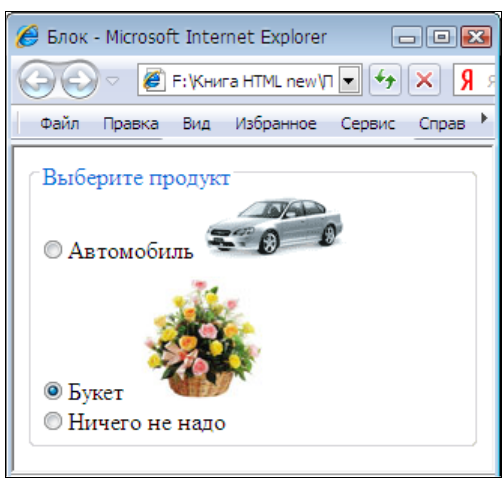


Рис. 11.9. Блок радиокнопок (листинг 11.5)

Размещение радиокнопок в одном блоке еще не делает их селекторными переключателями, среди которых только один может быть в установленном состоянии. Для этого, как уже отмечалось ранее, требуется, чтобы они имели одинаковые значения атрибута `name`.

Элемент `<fieldset>` по умолчанию занимает всю ширину клиентской области окна браузера, а высота блока зависит от количества расположенных в нем по вертикали элементов и их размеров. Установить размеры блока, стиль, толщину и цвет рамки по своему усмотрению можно с помощью параметров CSS.

Содержимое тега `<legend>` по умолчанию располагается на верхней части рамки слева. Изменить его положение с помощью параметра `text-align` можно пока только в Safari и Chrome. Для всех браузеров сработает атрибут `align` с возможными значениями `left`, `right` и `center`.

Обычно элементы пользовательского интерфейса сопровождают поясняющие надписи (метки). Это можно сделать, расположив текст надписи до или после тега элемента. Вместе с тем, контейнерный тег `<label>` позволяет автоматически расположить надпись рядом с элементом.

Пример

```
<label>
  Введите Ваше имя
  <input type="text"/>
</label>
```

Тег `<label>` обладает атрибутом `for`, который дает возможность указать, к какому элементу относится надпись. В качестве значения атрибут `for` принимает значение атрибута `id` элемента. При этом теги элементов могут не вкладываться в соответствующие им теги `<label>`.

Пример

```
<input type="radio" id="r1" name="product" value="auto"/>
<label for="r1">Автомобиль</label><br/>
<input type="radio" id="r2" name=product value ="flowers"
       checked=" checked "/>
<label for="r2">Цветы</label>
```

В теге `<label>` можно указать атрибут `accesskey` для задания клавиши с латинской буквой, например,

```
<label for="r1" accesskey="a">Автомобиль</label>.
```

При нажатии на эту клавишу в сочетании с клавишей <Alt> должен произойти переход к соответствующему элементу пользовательского интерфейса. Однако это работает только в Internet Explorer, Safari и Chrome.

11.6. Форма: тег *<form>*

Тег `<form>` используется в качестве контейнера для элементов пользовательского интерфейса (`<input>`, `<select>` , `<textarea>` и `<button>`), образуя так называемую *форму*. Благодаря специальным атрибутам этого тега с помощью кнопки типа `submit` (`<input type="submit"/>`) можно передать на сервер данные, содержащиеся в элементах формы.

Атрибуты тега `<form>`:

- `action` — URL-адрес сервера или конкретного серверного приложения (CGI-программы), обрабатывающего переданные ему данные формы. Для передачи данных этот атрибут обязателен: если он не указан или его значение пусто, данные формы не будут отправлены никаким способом.

  Примеры:

  - `<form action="/cgi-bin/">` — относительный URL-адрес; поскольку файл программы не задан, то предполагается первая программа в указанной папке;
  - `<form action="http://mysite.com/scripts/myprogram.php">` — URL-адрес файла программы, написанной на языке PHP;
  - `<form action="http://www.anyserver.ru/mypage.asp">` — URL-адрес asp-файла (Active Server Page) программы, написанной на JScript или VBScript;
- `method` — метод передачи данных на сервер: `get` или `post`;
- `enctype` — метод кодирования данных, передаваемых на сервер;
- `name` — имя формы;
- `target` — строка, содержащая имя (значение атрибута `name`) окна или фрейма, в котором следует показать ответ сервера; возможны также и следующие значения:
  - `_parent` — родительский фрейм, а если фреймов нет, то данный параметр работает как `_self`;
  - `_blank` — новое окно браузера;
  - `_top` — отменяет все фреймы и загружает ответ сервера в главное окно браузера, а если фреймов нет, то данный параметр работает как `_self`;
  - `_self` — загружает ответ сервера в текущее окно или фрейм.

При отправке данных формы браузер генерирует запрос на Web-сервер в соответствии с протоколом HTTP — основным протоколом передачи во Всемирной паутине. HTTP-запрос состоит из трех частей:

- метод; URL-адрес запроса и, возможно, данные; версия протокола;
- заголовок запроса (служебная часть);
- тело запроса, содержащее передаваемые данные.

Метод запроса задается в теге `<form>` как значение атрибута `method`. Значение `get`, устанавливаемое по умолчанию, означает, что передаваемые данные будут присоединены к URL-адресу через разделитель в виде символа `?`. Методом `get` передают небольшие объемы данных. Значение `post` означает, что данные будут находиться в теле запроса. Этот метод применяется при передаче больших объемов данных и, в частности, сообщений электронной почты.

Передаваемые данные в запросе представляются последовательностью пар вида:

значение_атрибута_name=значение_атрибута_value

Для каждого элемента формы генерируется своя пара *имя=значение*. При использовании метода `get` такие пары для разных элементов разделяются амперсандом (`&`), например:

```
/cgi-bin/myprogram.php?product=auto&model=vaz2110&color=red HTTP/1.1
```

Атрибут `enctype` задает метод кодирования данных. Точнее, значением этого атрибута является так называемый MIME-тип (Multipurpose Internet Mail Extensions — многоцелевые расширения электронной почты для Интернета). Возможны следующие значения:

- `application/x-www-form-urlencoded` (значение по умолчанию);
- `multipart/form-data`;
- `text/plain`.

Для передачи данных по электронной почте рекомендуется для атрибута `enctype` указывать значение `"text/plain"`.

В листинге 11.6 в качестве примера приводится (X)HTML-код формы для отправки данных на сервер. Эта форма содержит поле ввода адреса отправителя, текстовую область для ввода собственно сообщения, кнопку сброса введенных данных и кнопку инициализации передачи данных формы (рис. 11.10). Здесь предполагается, что данные передаются методом `post` серверной программе (сценарию на языке PHP), расположенной по адресу `http://anyserver.ru/anyprogram.php`.

Листинг 11.6. Форма

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
  <head><title>Форма</title>
  <style type="text/css">
    form {
     width:300px;
     padding:20px;
     border-width:1px;
     border-style:solid;
     background:#e0e0e0
    }
  </style>
  </head>
  <body>
     <form action="http://anyserver.ru/anyprogram.php" method="post" >
     <h2>Сообщение автору</h2>
       Ваш E-mail:
      <input type="text" name="from" value="" size=30/>
      <br/>
      Сообщение:<br>
      <textarea name='text' cols=30 rows=8></textarea>
      <br/>
```

```
      <input type="reset"/>
      <input type="submit" name="Отправить" value="Отправить"/>
    </form>
  </body>
<html>
```

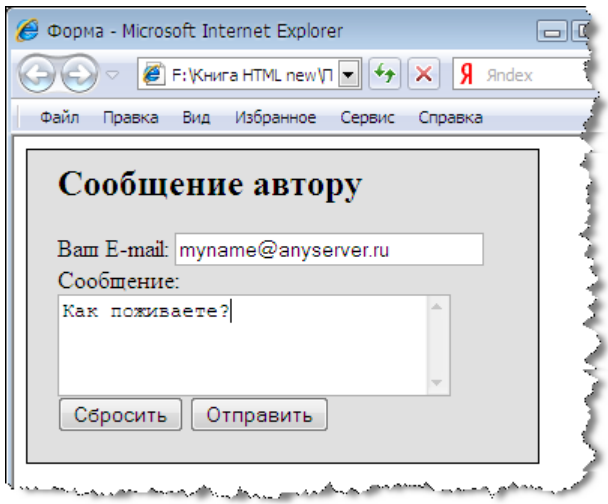


Рис. 11.10. Форма для отправки сообщений (листинг 11.6)

Серверный сценарий, получив данные, введенные посетителем страницы в форму, может распорядиться ими так или иначе в зависимости от назначения. Например, он может сохранить их на диске (в базе данных), переслать по электронной почте и т. п.

ГЛАВА 12

Вставка внешнего содержимого

Web-страница может содержать разнородную информацию: тексты, графические изображения, видео, звук, Flash-фильмы и т. д. В любом случае страница формируется посредством (X)HTML-кода. Однако только текст может содержаться в (X)HTML-коде непосредственно. Другие виды информации находятся во внешних файлах или программных компонентах и вставляются в Web-страницу посредством специальных тегов. Например, графическое растровое изображение, хранящееся в файле picture.jpg, вставляется в Web-страницу посредством тега `<img src="picture.jpg"/>`, который располагается где-то в (X)HTML-коде данной страницы. Интерпретатор (X)HTML-кода, встретив данный тег, попытается загрузить изображение, указанное в его атрибуте `src`, и в случае успеха отобразит его в окне браузера. Таким образом, интерпретатор замещает (X)HTML-теги, предназначенные для вставки внешнего содержимого, собственно содержимым из внешнего файла.

Внешняя информация в (X)HTML-документ вставляется в зависимости от ее особенностей посредством тегов `<img>`, `<iframe>`, `<object>`, `<embed>` и др., применению которых и посвящена данная глава.

12.1. Графические изображения

Графические изображения присутствуют в большинстве Web-страниц как в виде элементов внешнего оформления, так и в качестве внутреннего информационного содержимого (иллюстраций) публикуемых документов. Графические изображения бывают растровых и векторных форматов.

12.1.1. Растровая графика

Пикселы

Центральное понятие растровой графики — пиксел. Образно выражаясь, это атом в дискретном разложении графического изображения на квадратики. Каждый пиксел характеризуется цветом, а цвет — одним или несколькими числами. Пиксельная структура растрового изображения хорошо просматривается при достаточно большом его увеличении, установленном в программе просмотра (рис. 12.1).

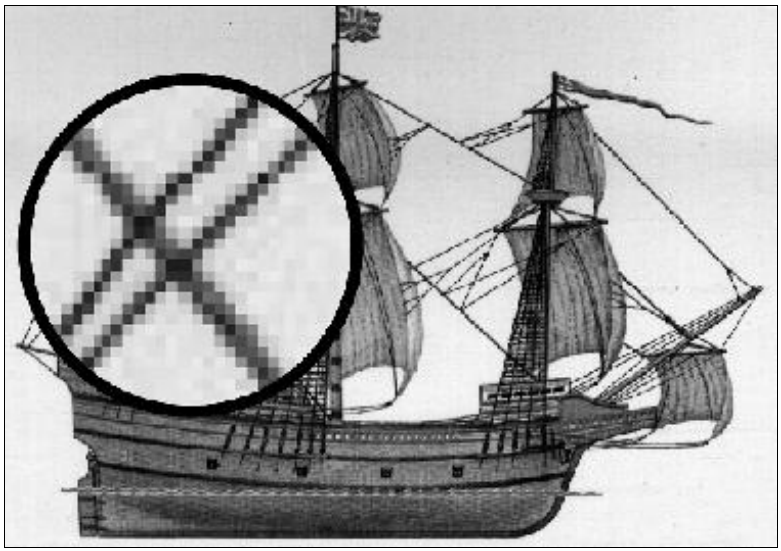


Рис. 12.1. Пиксельная структура растрового изображения

Очевидно, чем меньше размер пикселов, тем точнее можно представить изображение в дискретном виде. И наоборот, чем больше пикселы, тем грубее представляется исходное изображение или, иначе говоря, тем больше потери графической информации. Вместо потерь и точности передачи графической информации принято говорить о разрешении. *Разрешение* (resolution) определяется размерами пиксела: чем меньше пиксел, тем больше разрешение. С другой точки зрения, при фиксированных размерах изображения, измеряемых в метрах, сантиметрах, миллиметрах, дюймах и т. п., чем больше пикселов, которыми представляется это изображение, тем больше разрешение. Обобщая эти два взгляда на дискретное представление изображения, можно сказать, что разрешение есть величина, измеряемая как количество пикселов на единицу длины, т. е. плотность пикселов. Например, часто разрешение измеряют в *пикселах на дюйм* (pixels per inch, ppi), 1 дюйм = 25,4 мм.

Итак, разрешение и размер пиксела жестко взаимосвязаны:

Размер пиксела (мм) = 25,4 / Разрешение (ppi).

Например, при разрешении 300 ppi размер пиксела составит 0,085 мм (85 мкм).

При сохранении изображения в файле записывается и его разрешение. Таким образом, в компьютер попадает вся необходимая информация о размерах изображения.

Дисплей монитора состоит из множества элементов (точек), которые могут воспроизводить цвет. Эти точки нередко также называют пикселами, хотя это и неправильно. Говоря об отображении картинки на дисплее, мы имеем в виду отображение пикселов этой картинки на точки дисплея. Монитор можно настроить на отображение 640×480, 800×600, 1024×768, 1152×864, 1280×800, 1280×1024 и более пикселов по ширине и высоте соответственно. Это так называемое *установленное разрешение монитора*. Размеры вашей картинки в пикселах могут быть больше или меньше установленных размеров дисплея. Если они меньше или равны, то у дисплея достаточно точек, чтобы отобразить каждый пиксел картинки одной своей точкой. В этом случае картинка занимает

только часть дисплея или весь дисплей. Если увеличить масштаб вывода картинки, дисплей будет отображать один пиксел картинки несколькими своими точками. Скорее всего, в отображаемой картинке появятся размытость и зернистость. С увеличением масштаба отображения картинки разрешение (плотность пикселов, т. е. их количество на единицу длины) уменьшается. Если, наоборот, размеры картинки в пикселах больше установленного разрешения дисплея, то один пиксел дисплея должен будет отображать несколько пикселов картинки. Очевидно, что качество выводимой картинки ухудшится за счет потери деталей (то, что в картинке описывалось несколькими пикселами, теперь будет отображаться одной точкой дисплея). В этой ситуации высокое разрешение картинки оказалось бесполезным.

Нередко при создании и обработке изображения оперируют не его разрешением, а размерами. Однако если размеры определены в пикселах, то пересчитать их в разрешение не представляет особого труда. Например, изображение размером 216×216 пикселов и 3×3 дюйма имеет разрешение 216/3 = 72 ppi. Еще раз напомню, что разрешение — это всего лишь плотность пикселов. Вместе с тем, часто ширину и высоту изображения, выраженные в пикселах, называют горизонтальным и вертикальным разрешением. Так, например, о картинке размером 640×480 пикселов говорят, что она имеет разрешение 640 пикселов по горизонтали и 480 пикселов по вертикали.

Итак, мы рассмотрели понятия пиксела и разрешения как характеристики растровых изображений. Устройства ввода и вывода графической информации (мониторы, принтеры, сканеры, цифровые фотоаппараты) также характеризуются разрешением, которое принято измерять не в ppi, а в dpi (dots per inch — точках на дюйм). Размер точки этих устройств ограничен снизу их физическими и техническими особенностями. Следует различать понятия пиксела и точки и, следовательно, ppi и dpi. Если речь идет собственно об изображении, его хранении, обработке или вводе в компьютер, то употребляют понятия пиксела и ppi. Чтобы охарактеризовать способность некоторого устройства вывода воспроизвести графическую информацию, применяют понятия точки и dpi.

Нередко в фотокамерах, мобильных телефонах и сканерах устанавливают максимальное разрешение, получая изображения размером 1280×1024 px и более. Точнее, данные размеры характеризуют объем графической информации, снятой соответствующим устройством или программой. Такие размеры напрямую связаны с большим объемом памяти для хранения графической информации, но они позволяют вывести изображение в довольно крупном формате без потери качества. При публикации графических изображений в Web следует подумать об их оптимальном разрешении, поскольку область отображения даже на больших мониторах все же ограничена. Фотография достаточно хорошо воспринимается на расстоянии вытянутой руки, если имеет разрешение примерно 3—4 px/мм (т. е. типичные 72—96 px/дюйм). Такое разрешение при размерах 300—400 px позволяет растянуть фотографию до размеров 10×10 см без ухудшения качества. Фотографию размером 1280×1024 px можно отобразить без ухудшения качества на экране или бумаге размерами примерно 43×34—32×26 см. Очевидно, это не те размеры, которые пригодны для отображения картинок на Web-страницах. Типичные размеры 640×400 px соответствуют отображению картинки в области примерно 18×11 см, что более чем достаточно для Web-страниц, а также для книг и журналов. Таким образом, объем графической информации, содержащейся в картинках, для публикации на Web-страницах можно сократить, по крайней мере, до 640×400 px.

Цветовые модели

Как уже отмечалось, в растровом изображении каждый пиксел содержит информацию о своем цвете. Цвет представляется совокупностью чисел в соответствии с той или иной цветовой моделью.

Цветовая модель RGB используется при описании цветов, получаемых при смешении световых лучей. Она подходит для описания цветов, отображаемых мониторами, получаемых сканерами, фотокамерами, цветовыми фильтрами, но не печатающими устройствами.

Цвет в модели RGB представляется в виде суммы трех базовых цветов — красного (Red), зеленого (Green) и синего (Blue). На рис. 12.2 показано, какие цвета получаются при смешении базовых. Каждый базовый цвет характеризуется яркостью (интенсивностью), которая может принимать 256 дискретных значений от 0 до 255. Смешивая базовые цвета в различных пропорциях, можно получить 256 × 256 × 256 = 16 777 216 различных цветов. В частности, смешение базовых цветов в одинаковых пропорциях дает 256 оттенков серого, от черного до белого. Модель RGB еще называют аддитивной, поскольку по мере увеличения яркости базовых цветов увеличивается яркость результирующего цвета.

Все ли цвета, различимые человеческим глазом, можно представить моделью RGB? Другими словами, совпадают ли *цветовые диапазоны* человеческого глаза и модели RGB? Вообще говоря, нет. Однако более 16 млн цветов, представляемых в RGB, оказываются вполне достаточными для практических нужд. Вместе с тем, следует иметь в виду, что не любой цвет в RGB можно вывести на печать. Другими словами, цвета на экране вашего монитора могут выглядеть иначе при их выводе на печать, причем это отличие может оказаться принципиальным, а не только обусловленным низким качеством принтера или монитора.

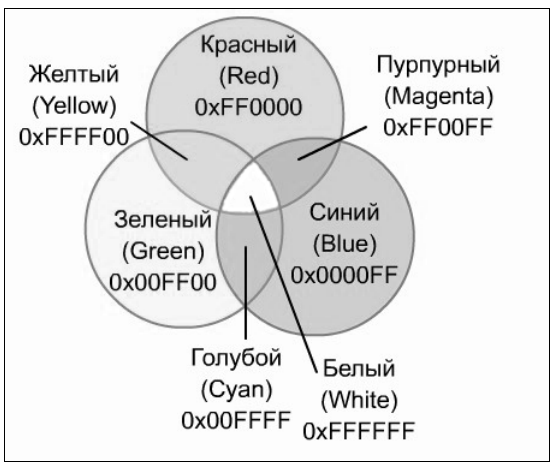


Рис. 12.2. Комбинация базовых цветов модели RGB

Если мониторы суммируют разноокрашенные лучи света, то печатающие устройства смешивают краски. Для описания результата комбинирования различных красок служит модель CMYK с тремя базовыми цветами: голубым или, точнее, бирюзовым

(Cyan), пурпурным (Magenta) и желтым (Yellow). Кроме того, применяется еще один, так называемый ключевой (Key) цвет, обычно это черный (blacK). На рис. 12.3 показана комбинация базовых цветов модели CMYK.

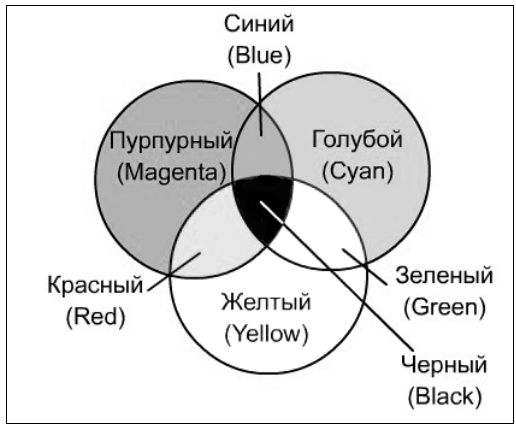


Рис. 12.3. Комбинация базовых цветов модели CMYK

Если картинки на мониторе мы воспринимаем в падающем на сетчатку глаза свете (монитор излучает свет), то изображения на бумаге воспринимаются в отраженном свете. Видимый отраженный свет получается в результате поглощения одной и отражения остальной части падающего света. Например, красная краска воспринимается нами как красная потому, что ее вещество поглощает зеленую и синюю составляющие падающего белого света и отражает красную. В частности, черный цвет возникает, если вещество поглощает весь падающий на него белый свет и ничего не отражает. Белый цвет получается потому, что вещество ничего не поглощает, а все отражает. Базовые цвета модели CMYK получают путем вычитания из белого базовых цветов модели RGB и в связи с этим CMYK называют субтрактивной моделью:

Cyan = RGB – R = GB (поглощает R и отражает G и B);

Yellow = RGB – B = RG (поглощает B и отражает R и G);

Magenta = RGB – G = RB (поглощает G и отражает R и B).

Смесь трех базовых цветов модели CMYK поглощает все базовые цвета RGB и поэтому теоретически должна быть черной. Однако на практике она получается грязно-коричневого цвета. Поэтому в цветовую модель CMYK включен еще и чистый черный цвет, который используется для создания темных оттенков, а также для печати чисто черных элементов изображения.

Основные цвета моделей RGB и CMYK взаимосвязаны (рис. 12.4). Каждый цвет расположен напротив дополняющего его и между цветами, с помощью которых он получен. Чтобы усилить какой-либо цвет, необходимо ослабить дополняющий цвет, расположенный на противоположной стороне круга. Например, чтобы усилить желтый (Yellow), нужно ослабить синий (Blue). На круге цветов желтый расположен между зеленым (Green) и красным (Red). Сложение этих цветов дает желтый (Yellow).

Следует отметить, что не все цвета модели CMYK можно представить в модели RGB и наоборот. В количественном отношении цветовой диапазон CMYK меньше, чем RGB.

Это обстоятельство имеет принципиальное значение, а не обусловлено только физическими особенностями монитора, печатающего устройства или красок и холста.

Кроме RGB и CMYK существуют и другие модели (HSB, HLS и Lab), не ориентированные на какое-либо устройство воспроизведения.

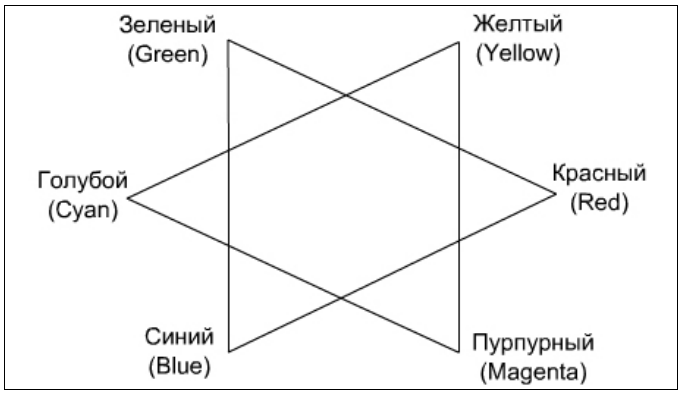


Рис. 12.4. Взаимосвязь базовых цветов RGB и CMYK

Модель HSB основана на трех параметрах: H — оттенок или тон (Hue), S — насыщенность (Saturation) и B — яркость (Brightness). Модель HSB лучше, чем RGB и CMYK, соответствует понятию цвета, которое используют маляры и профессиональные художники. Действительно, у них обычно есть несколько основных красок, а все другие получают добавлением к ним белой и черной. Таким образом, нужные цвета — это некоторая модификация основных: замесить погуще или развести пожиже, осветлить или затемнить. Хотя художники и смешивают краски, но это уже выходит за рамки модели HSB ("...есть многое на свете, друг Горацио, что и не снилось нашим мудрецам...").

Насыщенность характеризует чистоту цвета. Нулевая насыщенность соответствует серому цвету, а максимальная насыщенность — наиболее яркому варианту данного цвета. Можно считать, что изменение насыщенности связано с добавлением белой краски. То есть уменьшение насыщенности соответствует добавлению белой краски.

Яркость понимается как степень освещенности. При нулевой яркости цвет становится черным. Наибольшая яркость при максимальной насыщенности дает наиболее выразительный вариант данного цвета. Можно также считать, что яркость изменяется добавлением черной краски. Чем больше черной краски добавлено, тем меньше яркость.

В некоторых графических редакторах, например в Adobe FreeHand, используется модель HLS (Hue, Lightness, Saturation), которая похожа на HSB. В модели HLS, в отличие от HSB, яркость заменяет параметр L — освещенность (Lightness). Уменьшение освещенности приближает цвет к черному, а увеличение — к белому. Чистый спектральный цвет получается при освещенности 50%.

Ранее уже отмечалось, что модель RGB ориентирована в основном на особенности излучаемого света (монитор), а CMYK — поглощаемого света (принтер). Кроме того, цветовые диапазоны этих моделей не совпадают. Добавим, что RGB хорошо воспроизводит цвета в диапазоне от синего до зеленого и несколько хуже — желтые и оранже-

вые оттенки, а в модели CMYK не хватает очень многих оттенков. От всех этих недостатков свободна модель Lab. В рамках Lab работают многие профессионалы компьютерной графики.

Модель Lab основана на трех параметрах: L — яркость (Luminosity) и два цветовых параметра — a и b. Параметр a содержит цвета от темно-зеленого через серый до ярко-розового, b — от светло-синего через серый до ярко-желтого.

Параметр L еще называют освещенностью и даже светлотой. Следует отметить, что понятия яркости в моделях Lab и HSB не тождественны. Как и в RGB, смешение цветов из шкал a и b позволяет получить более яркие цвета. Уменьшить яркость результирующего цвета можно за счет параметра яркости L.

Модель Lab аппаратно независима, ее цветовой диапазон покрывает диапазоны RGB и CMYK. Графический редактор Adobe Photoshop, например, при переходе от режима RGB к CMYK использует Lab в качестве промежуточного этапа.

Внимание!

При подготовке графических изображений к публикации в Интернете окончательный результат следует представить в цветовой модели RGB.

Цветовая глубина

Цвет каждого пиксела описывается числами, задающими яркости базовых цветов или другие параметры. Так, в модели RGB каждому пикселу сопоставлена тройка чисел, соответствующих яркостям базовых цветовых составляющих (красной, зеленой и синей). В модели CMYK пиксел описывается четверкой чисел (яркости бирюзовой, пурпурной, желтой и черной составляющих). В моделях HSB и Lab — тройкой чисел, соответствующих значениям параметров этих моделей. Числа, которыми описывается цвет пикселов, еще называют цветовыми каналами.

Как известно, числа можно представлять в различных системах счисления. Обычно мы пользуемся десятичной системой, в которой числа записывают цифрами от 0 до 9. В программировании часто встречается шестнадцатеричная система счисления, в которой 16 цифр (0, 1, 2, ..., 9, a, b, c, d, e, f). Работа компьютеров основана на двоичной системе с двумя цифрами — 0 и 1. Двоичную цифру называют *битом*. Бит может принимать только одно из двух возможных значений. Количество битов, отводимое на каждый пиксел для представления цветовой информации, называют *цветовой глубиной* (color depth), или *битовой глубиной цвета* (bit depth).

Цветовая глубина определяет, как много цветов может быть представлено пикселом. Например, если цветовая глубина равна 1 биту, то пиксел может представлять только один из двух возможных цветов, скажем, белый или черный. Если цветовая глубина равна 8 битам, то количество возможных цветов равно $2^8 = 256$. При глубине цвета 24 бита количество цветов превышает 16 млн. Связь между битовой глубиной цвета и количеством цветов проста:

$$\text{Количество цветов} = 2^{\text{битовая глубина цвета}}.$$

Иногда под цветовой глубиной понимают максимальное количество цветов, которые можно представить. Очевидно: чем больше цветовая глубина, тем больше объем файла, содержащего описание всего изображения.

Изображения в системах RGB, CMYK, Lab и оттенках серого (gray scale) обычно содержат 8 битов на один цветовой канал. Поскольку в RGB и Lab три цветовых канала, глубина цвета в этих режимах $8 \times 3 = 24$ бита. В CMYK четыре канала и поэтому цветовая глубина $8 \times 4 = 32$ бита. В полутоновых изображениях только один канал, следовательно, его цветовая глубина равна 8 битам.

Во многих случаях описание изображения может быть более компактным, чем в режимах RGB, CMYK или Lab. Режим индексированных цветов (indexed color) обеспечивает компактное представление изображений, содержащих не более 256 цветов (т. е. имеющих глубину цвета не более 8 битов).

На практике изображения часто содержат не все воспринимаемые глазом цвета, а лишь небольшое их подмножество. Например, плакаты обычно раскрашены не более чем десятью цветами. Представим себе некоторую картинку. Разделим все различимые на ней цвета на несколько классов. Например, близкие цвета можно отнести к одному и тому же классу, а существенно различные — к разным. Совокупность всех таких классов образует палитру цветов данной картинки. Элементы палитры (классы цветов) можно пронумеровать или, иначе говоря, проиндексировать. Далее, составим таблицу, в которой каждому индексу сопоставим цвет из палитры (например, RGB-код). Тогда описание картинки должно содержать эту таблицу и последовательность индексов, соответствующих каждой точке картинки. Если элементов палитры меньше, чем исходных цветов, то при таком описании происходит потеря исходной графической информации. Если элементов палитры столько же, сколько исходных цветов, то описание точно передает исходную графическую информацию.

На практике довольно широко распространены палитры, содержащие 256 цветов (каждый цвет представляется одним байтом или восемью битами). Иногда применяются 16-цветные (4-битные) палитры. Индексированные цвета используются в графических файлах формата GIF, а этот формат чрезвычайно популярен в Web-дизайне.

Все графические изображения, создаваемые для Web, должны основываться либо на цветовой модели RGB, либо на подмножестве индексированных цветов, либо представляться в градациях серого цвета. Это общее правило. Однако в Интернете существует множество мониторов с низким разрешением цветопередачи — 256 цветов. Даже если ваша картинка создана в палитре из 256 цветов, это еще не означает, что она будет отображена в Web-браузере так же, как вы ее видите в своем графическом редакторе.

И это еще не все. В Web довольно часто встречается графика с индексированными цветами. В графических редакторах можно создавать изображения с различными палитрами цветов. Однако браузеры имеют свою собственную палитру, содержащую 216 цветов — результат пересечения множеств цветов, содержащихся в палитрах Windows и Mac OS. Если говорить по существу, Web-палитра включает в себя те 216 цветов, значения R, G и B которых делятся нацело на 51. Это означает, что яркость каждого базового цвета модели RGB можно установить равной 0, 51, 102, 153, 204 или 255 (т. е. 6 возможных значений). Таким образом, палитра Web содержит $6 \times 6 \times 6 = 216$ цветов. При выводе изображения на экран монитора, настроенный на отображение 256 цветов, браузер автоматически меняет все цвета на те, которые представлены в палитре Web. Это гарантирует, что посетители вашей страницы увидят то же, что и вы на своем мониторе.

12.1.2. Основные форматы растровой графики

В настоящее время в Web применяют в основном три формата растровых графических изображений: GIF, PNG и JPEG.

Формат GIF (Graphics Interchange Format — формат графического обмена) использует алгоритм сжатия без потерь LZW и предназначен для сохранения растровых изображений с количеством цветов не более 256.

В настоящее время существуют две версии формата — GIF87a и GIF89a. Расширение файлов этих форматов — gif. Первая версия была разработана в 1987 г., а вторая — в 1989 г. Формат GIF89a позволяет сохранять прозрачность (transparency) пикселов и поддерживает анимацию. Однако участки изображения не могут быть полупрозрачными: они либо полностью прозрачны, либо полностью непрозрачны. Формат GIF допускает чересстрочную (interlaced) запись графической информации, чтобы загрузка в браузер выглядела как постепенное повышение четкости изображения. Чересстрочная запись несколько увеличивает размер файла, но дает интересный визуальный эффект.

В формате GIF89a можно сохранить не только одно, но и несколько изображений, которые браузер показывает друг за другом с заданными частотой и временем задержки. В результате возникает эффект анимации картинки. В настоящее время существует множество редакторов для создания анимационных GIF-файлов.

В GIF-файлах хорошо сохранять контрастные изображения без плавных цветовых переходов и цветового шума, например, логотипы, баннеры, чертежи, схемы. Другими словами, чем меньше нюансов в изображении, тем больше степень сжатия. Для фотографий с высоким качеством цветопередачи формат GIF вряд ли годится из-за ограничения палитры цветов. Для таких изображений лучше подходят форматы PNG и JPEG *(см. далее)*.

GIF-формат популярен в Web-дизайне благодаря достаточно высокой степени сжатия без потерь, поддержке прозрачности пикселов, чересстрочности и анимации.

Формат PNG (Portable Network Graphics — переносимая графика для сети) был разработан с целью заменить формат GIF. Во-первых, потому, что фирма-разработчик GIF запатентовала его. Во-вторых, формат PNG должен был преодолеть недостатки GIF, связанные с ограничением числа цветов. PNG не запатентован и может использоваться бесплатно. Расширение файлов этого формата — png.

Формат PNG позволяет сохранять изображения с глубиной цвета 24 и даже 48 бит, он также дает возможность включать каналы масок для управления градиентной прозрачностью, но не поддерживает слои. Преобразование графической информации в формат PNG происходит без потерь. В Adobe Photoshop, например, предусмотрено сохранение изображений с глубиной цвета 8 (как в GIF) и 24 бита (как в JPEG). Файлы PNG обычно имеют больший размер, чем GIF- и JPEG-файлы с аналогичными изображениями. В Web-дизайне этот формат целесообразен для хранения небольших многоцветных изображений с мелкими деталями (например, переливающихся цветами кнопок с картинками).

Формат JPEG (Joint Photographic Experts Group — объединенная группа экспертов по фотографии) предназначен для компактного хранения многоцветных изображений с фотографическим качеством. Он был разработан в 1995 г. специально для нужд Интернета. Файлы этого формата имеют расширение jpg, jpe или jpeg.

В отличие от GIF и PNG, в формате JPEG применяется алгоритм сжатия с потерями информации, благодаря чему достигается очень большая степень сжатия (от единиц до сотен раз).

Формат JPEG не поддерживает прозрачность пикселов (альфа-канал). Он хорош для изображений с большим числом цветов и плавными цветовыми переходами (фотографий). Во многих случаях потери графической информации, возникающие при сохранении изображений в этом формате, оказываются вполне приемлемыми, незаметными для глаза.

12.1.3. Векторная графика

Информация об изображении в файлах растровых форматов заключена в описании пикселов. В так называемых векторных изображениях графическая информация представлена в виде описания более сложных объектов, таких как линии, овалы, прямоугольники и другие геометрические фигуры. Сложные объекты образуются из более простых (графических примитивов). Векторный графический объект создается на основе некоторого математического описания, определяющего изображение не как совокупность отдельных точек (пикселов). В векторном описании каждая линия изображения характеризуется положением, длиной и направлением, в котором она проводилась. Кроме того, описание линии содержит информацию о ее толщине, цвете и другие параметры.

Иначе говоря, векторное описание — параметрическое, используя которое программа визуализации может воспроизвести (нарисовать) соответствующий объект. Например, чтобы нарисовать отрезок прямой, достаточно указать координаты его начала и конца. Если линия замкнута (многоугольник, овал и т. п.), то она образует некоторый новый векторный объект — форму или, другими словами, фигуру. Область, охваченная замкнутой линией, может иметь или не иметь цвет. Поэтому ее еще называют областью заливки цветом. Внутренняя область фигуры может быть залита обычным, сплошным или, как иногда говорят, твердым цветом или в некоторой степени прозрачным, благодаря чему сквозь полупрозрачный объект переднего плана можно разглядеть задний план. Особую роль в окрашивании объектов имеют градиентные заливки, при которых один цвет плавно переходит в другой. В векторных объектах область заливки представляется не совокупностью пикселов заданного цвета, а параметрическим описанием (коды цвета и способ их распространения по области).

С точки зрения пользователя важно понимать следующие три обстоятельства:

- векторные изображения хранятся в файлах компьютера в виде совокупности описаний отдельных элементарных векторных изображений (графических примитивов), а не пикселов. В большинстве практически важных случаев это оказывается более экономным, чем хранение изображения в виде описаний всех его пикселов;
- создание изображений путем коррекции и композиции графических примитивов более удобно и производительно, чем раскрашивание отдельных пикселов;
- поскольку векторное изображение воспроизводится устройством вывода по его математическому описанию, увеличение масштаба практически не ухудшает качества картинки.

Итак, экономия в объеме, удобство создания и редактирования, а также масштабируемость без ухудшения качества — основные достоинства векторной графики. Главный ее недостаток — сложность создания изображений типа фотографий и произведений живописи.

Наиболее широко известные графические редакторы, позволяющие создавать векторные изображения, — CorelDRAW, Macromedia Flash (начиная с версии 9 этот пакет выходит под маркой Adobe), Adobe Illustrator и Inkscape.

В настоящее время существует довольно много векторных графических форматов, но лишь только некоторые из них могут применяться в Web-дизайне. Векторный формат SWF (Shockwave Flash) наиболее широко применяется в Web для представления векторной статической и, главным образом, анимационной графики (так называемых Flash-фильмов или роликов). Файлы данного формата с расширением swf обычно создают и редактируют с помощью редактора Flash, хотя и некоторые другие векторные редакторы позволяют сохранять разработанные в них изображения в swf-файлах.

Другой подход к созданию векторной графики основан на применении специальных языков, являющихся словарями данных языка XML. Microsoft разработала язык VML (Vector Markup Language — язык векторной разметки) и в 1998 г. предложила его спецификацию для стандартизации консорциуму W3C. Хотя данный язык не был принят в качестве официальной рекомендации, его поддерживают некоторые программы, в том числе Internet Explorer 5+ и Microsoft Office 2000+. Вместо VML консорциум W3C принял в качестве рекомендации язык SVG (Scalable Vector Graphic — масштабируемая векторная графика).

SVG является подмножеством XML и позволяет описывать двумерную статическую и анимационную как векторную, так и смешанную векторно-растровую графику. Графику на языке SVG можно описать в обычном текстовом редакторе посредством специальных тегов и атрибутов. Например, чтобы задать красный круг с центром в точке с координатами 250 и 300 px и радиусом 100 px, достаточно такого тега:

```
<circle cx="250px" cy="300px" r="100px" fill="red"/>
```

Таким образом, SVG — язык декларативного (а не процедурного) описания графики. Вместе с тем, он поддерживает управляемость графических элементов скриптами на языке JavaScript, а также интерактивность.

Применение для создания SVG-графики обычного текстового редактора предполагает знание языка SVG. Альтернативный подход заключается в использовании графического редактора, например, CorelDRAW, Adobe Illustrator или бесплатного Inkscape (**www.inkscape.org**). Текстовые файлы с описанием графики на языке SVG обычно имеют расширение svg. Возможно также их представление в сжатом виде, тогда расширение файлов — svgz.

В настоящее время SVG поддерживают браузеры Firefox, Opera, Safari, Chrome и Internet Explorer 9. Однако последний не поддерживает анимацию. Предполагается, что этот недостаток будет устранен в следующих версиях данного браузера. Internet Explorer 8 и более ранние версии не имеет встроенных средств для отображения SVG-графики и для него требуется установить соответствующий плагин, например, Adobe SVGViewer (**www.adobe.com/svg/viewer/install/**).

12.1.4. Вставка графики в (X)HTML-документ

Растровые графические изображения вставляют в (X)HTML-документ с помощью тегов `<img>`, `<iframe>` и `<object>`, а также в качестве фона элемента посредством стилевого параметра `background` или `background-image`. Применение изображения в качестве фона элемента документа подробно рассматривалось в *главе 5*. Тег `<img>` уже неоднократно упоминался в предыдущих главах. Здесь мы рассмотрим его более обстоятельно.

Применение тега *<img>*

Основные атрибуты внутристрочного неконтейнерного тега `<img>`:

- `src` — URL-адрес графического файла (обязательный атрибут);
- `alt` — альтернативный текст, который отображается прежде загрузки и отображения в браузере графического изображения. При отключенной в настройках браузера загрузке изображений данный текст позволяет пользователю составить представление о том, что должно быть отображено;
- `usemap` — значение атрибута `id` тега `<map>`, содержащего определения активных областей графической карты ссылок в клиентском варианте (см. *разд. 7.3*);
- `ismap="ismap"` — указывает, что изображение используется в качестве графической карты ссылок в серверном варианте.

Дополнительные атрибуты, предназначенные для форматирования изображения:

- `width`, `height` — ширина и высота прямоугольной области, в которой отображается графическое изображение. Возможны целочисленные значения без указания размерности (подразумеваются пикселы) и с указанием %. В последнем случае размеры задаются относительно контейнера, содержащего тег `<img>`. Если атрибуты `width` и `height` не указаны, то графическое изображение показывается в своих оригинальных размерах. Если указан только один из атрибутов, то значение второго вычисляется так, чтобы сохранить оригинальные пропорции изображения. При отличии значений `width` и `height` от оригинальных размеров изображения, последнее масштабируется (растягивается или сжимается) так, чтобы заполнить всю заданную прямоугольную область. При масштабировании растровых изображений возможно заметное ухудшение качества их отображения. Указание атрибутов `width` и `height` не влияет на объем, а следовательно, и на скорость загрузки графического файла;
- `border` — число, задающее толщину рамки, окаймляющей изображение; нулевое значение соответствует отсутствию рамки;
- `align` — горизонтальное выравнивание изображения относительно окружающего текста и других элементов; возможные значения: `bottom` (по умолчанию), `top`, `left`, `right`, `middle`;
- `hspace`, `vspace` — размеры соответственно горизонтальных и вертикальных отступов вокруг изображения. Целочисленные значения указывают без размерности (подразумеваются пикселы).

Атрибуты форматирования `align`, `hspace`, `vspace`, `width` и `height` можно и рекомендуется заменять соответствующими параметрами CSS. В HTML 5 атрибуты `align` и `hspace` не предусмотрены. Так, размеры задаются стилевыми параметрами `width` и `height` с указа-

нием размерности, отступы и граница (рамка) определяются параметрами соответственно `padding` и `border`. Если картинка должна соседствовать с текстом, то неплохо подумать о применении параметра `float`, задающего способ обтекания картинки текстом или, наоборот, текста картинкой.

Если размеры изображения указаны одновременно посредством и CSS, и атрибутов, то при удачной его загрузке во всех браузерах, кроме Opera, приоритет будет у стилевых параметров. Задание параметров `width` и/или `height`, даже если вам нужны оригинальные размеры изображения, несколько ускоряет формирование документа в окне браузера.

Что произойдет, если указанный в атрибуте `src` файл не найден? Результат зависит от браузера. Internet Explorer, Safari и Chrome отобразят прямоугольник с заданными размерами, специфической пиктограммой и альтернативным текстом. Opera покажет только прямоугольник с альтернативным текстом, если он задан (хотя бы пустой), а в противном случае — слово "Изображение". Firefox выведет только альтернативный текст, а элементу `<img>` назначит минимальные размеры, чтобы разместить этот текст. Если последний не определен, то Firefox установит нулевые размеры. При этом значения заданных параметров `width` и `height` будут проигнорированы, что может сказаться на расположении окружающих элементов. В других браузерах элемент `<img>` будет иметь заданные размеры, даже если изображение не удалось загрузить.

В листинге 12.1 приведен тестовый код вставки в (X)HTML-документ пяти элементов `<img>`. В первом теге атрибут `src` имеет пустое значение и, следовательно, никакое изображение не будет загружено. Второй тег `<img>` вставляет изображение в своих оригинальных размерах. Последующие теги `<img>` загружают ту же картинку соответственно в пропорционально уменьшенных, пропорционально увеличенных и искажающих размерах. Вид этого документа в браузерах Internet Explorer и Firefox показан на рис. 12.5.

Листинг 12.1. Вставка графических элементов посредством тега `<img>`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Графика</title>
<style type="text/css">
  .small {width:50px} /*маленькая*/
  .large {width:130px} /*большая*/
  .distort {width:150px;height:50px}/*искаженная*/
</style>
</head>
<body>
  <img  id="myimg" src="" alt="Картинка 1"/>
  <img  src="picture.gif" alt="Картинка 2"/>
  <img  src="picture.gif" class="small" alt="Картинка 3"/>
  <img  src="picture.gif" class="large" alt="Картинка 4"/>
  <img  src="picture.gif" class="distort" alt="Картинка 5"/>
</body>
</html>
```

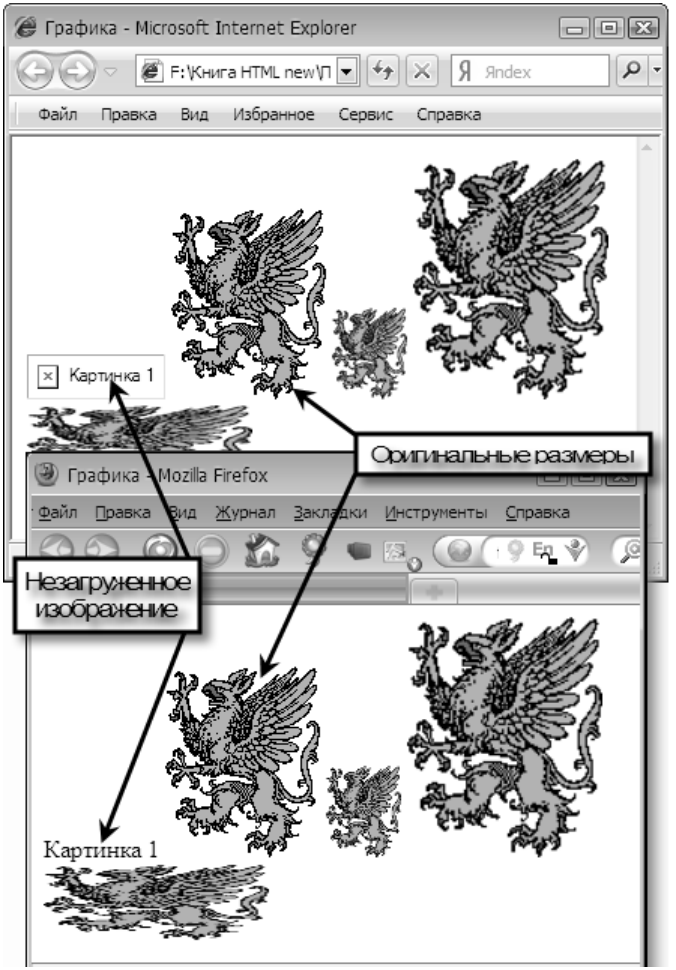


Рис. 12.5. Варианты отображения графического изображения посредством тега `<img>` в Internet Explorer и Firefox (листинг 12.1)

Применение тегов *<iframe>* и *<object>* для вставки изображений

Тег `<img>` — не единственно возможное средство вставки картинки в (X)HTML-документ. Изображение можно вставить и в элементы, создаваемые следующими внутристрочными контейнерными тегами:

- `<iframe src="`*URL-адрес файла*`"></iframe>`
- `<object data="`*URL-адрес файла*`" type="`*MIME-тип файла*`"></object>`

Данные теги контейнерные и в них можно вставить альтернативный текст или графическое изображение на случай, если браузер не воспринимает их или не может загрузить изображение. Примеры MIME-типов файлов с графическими изображениями: image/gif, image/jpeg, image/jpeg2000, image/png, image/x-png.

В случае применения тега `<iframe>` картинка отображается в отдельном окне (плавающем или внутреннем фрейме). Это окно по умолчанию имеет размеры 300×150 px, хотя

их можно задать по своему усмотрению с помощью атрибутов или CSS-параметров width и height. В браузерах Internet Explorer, Opera, Safari и Chrome изображение не масштабируется сообразно размерам фрейма, в котором при необходимости появляются полосы прокрутки. А Firefox, наоборот, масштабирует изображение в сторону уменьшения так, чтобы оно полностью поместилось во фрейме.

По умолчанию плавающий фрейм отображается с видимой границей. Во всех браузерах, кроме Internet Explorer, с помощью стилевого параметра border:0 или атрибута border="0" ее можно сделать невидимой. По умолчанию фон фрейма белый в Internet Explorer, светло-серый в Opera и прозрачный в Firefox, Safari и Chrome. При этом в Internet Explorer и Opera, в отличие от других браузеров, его нельзя установить с помощью стилевого параметра background.

Тег <iframe> имеет атрибут name, с помощью которого можно задать имя фрейма. Это имя можно использовать, например, в ссылках в качестве значения атрибута target, чтобы изображение или какой-нибудь другой ресурс загрузить в плавающий фрейм с данным именем:

```
<iframe name="myframe"></iframe>
<a href="mypicture.jpg" target="myframe">Показать картинку</a>
```

Тег <object> отображает графическое изображение в окне, подобном плавающему фрейму. Однако имеются отличия. Размеры окна по умолчанию устанавливаются так, чтобы картинка полностью в нем поместилась. При задании размеров картинка масштабируется (растягивается или сжимается), чтобы занять всю площадь окна подобно тому, как это происходит при использовании тега <img>.

По умолчанию окно элемента <object> отображается без видимых границ, но их можно задать стилевым параметром border. Фон окна по умолчанию прозрачный, но может быть установлен с помощью стилевого параметра background.

Все современные браузеры отображают графические изображения в окне элемента <object> одинаково. Однако Internet Explorer 8.0 в режиме предоставления совместимости показывает этот элемент, только если заданы его размеры. При этом картинка не масштабируется, а в случае необходимости появляются полосы прокрутки.

В листинге 12.2 приведен тестовый код вставки в (X)HTML-документ одной и той же картинки на прозрачном фоне размером 103×127 px посредством тегов <img>, <iframe> и <object>. Вид этого документа в браузерах Internet Explorer 8 и Firefox показан на рис. 12.6. Фон картинки в данном примере прозрачный.

Листинг 12.2. Вставка графических элементов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Графика</title>
<style type="text/css">
  body {background:#e0e0e0}
  iframe, object {width:150px; height:100px}
</style>
</head>
```

```
<body>
  <img id="myimg" src="picture.gif" alt="Картинка"/>
  <iframe src="picture.gif" >Картинка</iframe>
  <object data="picture.gif" type="image/gif">Картинка</object>
</body>
</html>
```

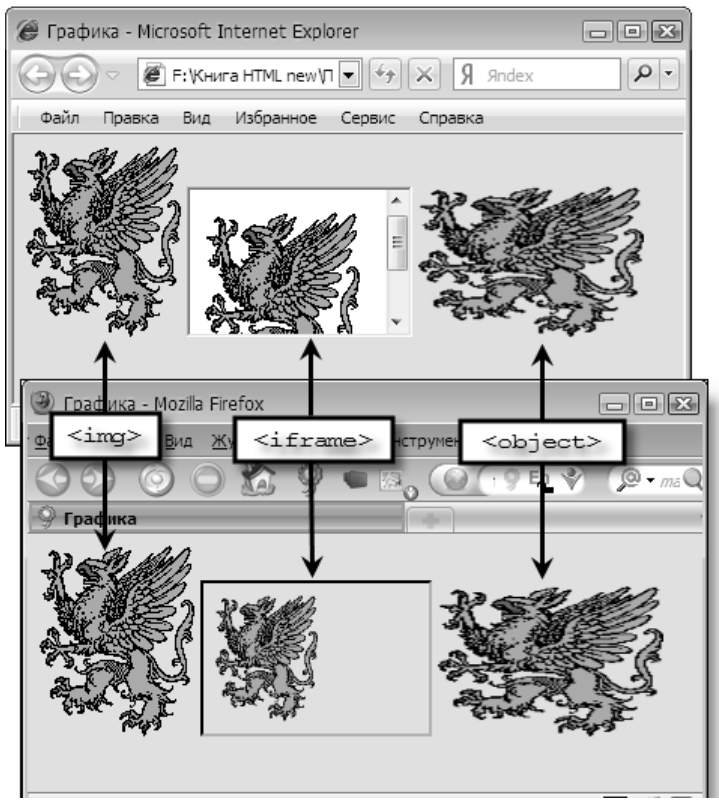


Рис. 12.6. Отображение графического изображения посредством тегов `<img>`, `<iframe>` и `<object>` в Internet Explorer и Firefox (листинг 12.2)

Пример простой фотогалереи

Проиллюстрируем применение тега `<iframe>` на примере создания простой фотогалереи. Web-страница с фотогалереей содержит несколько миниатюрных графических изображений. При щелчке на миниатюре соответствующее графическое изображение появляется в окне плавающего фрейма. В данном примере элементы, чувствительные к щелчку, — ссылки на файлы с графическими изображениями. Миниатюры — графические изображения, вставляемые посредством тегов `<img>` в теги ссылок `<a>`. Тег `<iframe>` снабжен атрибутом `name` (имя фрейма), значение которого присваивается атрибуту `target` всех тегов `<a>`. Обычно в качестве миниатюр используют изображения с малыми оригинальными размерами, но в данном примере для простоты их берут из тех же файлов, которые показываются в окне фрейма. (X)HTML-код фотогалереи представлен в листинге 12.3, а ее вид в браузерах Internet Explorer и Firefox — на рис. 12.7. Internet

Explorer, как и Opera, Safari и Chrome, не масштабирует изображение во фрейме и при необходимости снабжает последний полосами прокрутки. Однако Firefox изменяет размеры изображения так, чтобы с сохранением оригинальных пропорций оно вписалось во фрейм.

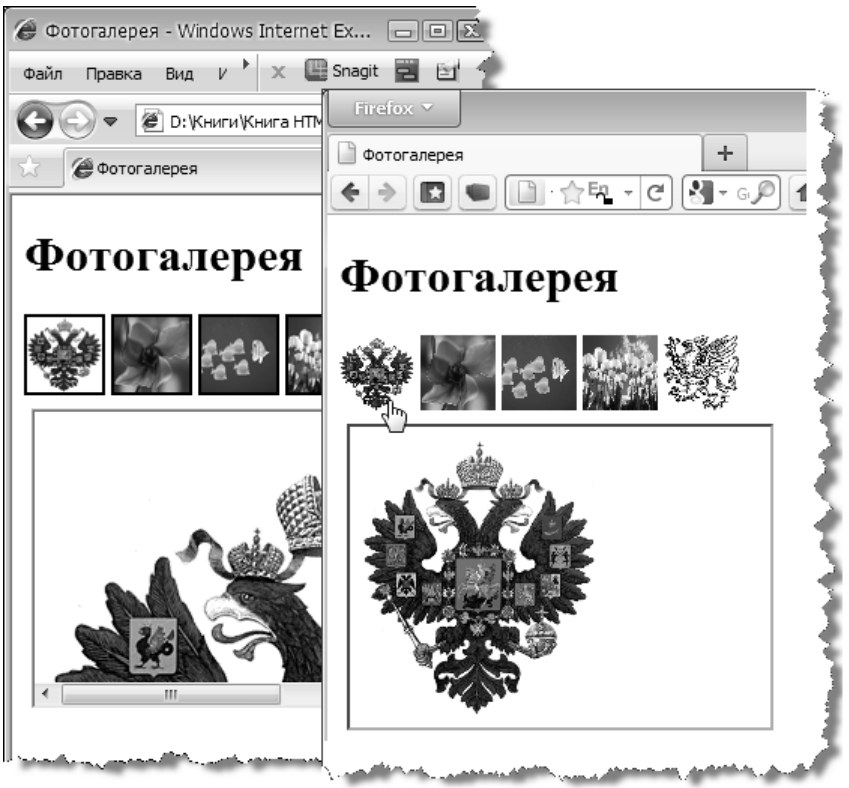


Рис. 12.7. Простая фотогалерея (листинг 12.3)

Листинг 12.3. Простая фотогалерея

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Фотогалерея</title>
<style type="text/css">
  .thumbnail {width:50px; height:50px; } /*миниатюра*/
  .frame {width:280px; height:200px; margin:5px} /*фрейм*/
</style>
</head>
<body>
<h1>Фотогалерея</h1>
<a href="picture1.jpg" target="frame">
    <img src="picture1.jpg" class="thumbnail">
</a>
```

```
<a href="picture2.jpg" target="frame">
    <img src="picture2.jpg" class="thumbnail">
</a>
<a href="picture3.jpg" target="frame">
    <img src="picture3.jpg" class="thumbnail">
</a>
<a href="picture4.jpg" target="frame">
    <img src="picture4.jpg" class="thumbnail">
</a>
<a href="picture5.jpg" target="frame">
    <img src=" picture5.jpg " class="thumbnail">
</a>
<br/>
<iframe name="frame" class="frame"></iframe>
</body>
</html>
```

Фотогалерею можно усложнить, если добавить к ней кнопки перехода к следующей и предыдущей миниатюрам, а также другую функциональность. Но для этого необходимы скрипты.

Вставка SVG

Как отмечалось ранее, SVG — масштабируемая векторная графика, описываемая на специальном XML-подобном языке и сохраняемая в обычном текстовом файле. SVG позволяет описывать собственно векторные изображения (линии и фигуры), форматированный текст, а также включать в документ и работать с растровыми изображениями: масштабировать и трансформировать их подобно векторным объектам. Кроме того, он дает возможность создавать анимационную графику, интерактивные приложения, управлять поведением и внешним видом с помощью JavaScript и CSS.

Браузеры Firefox, Opera, Safari, Chrome и Internet Explorer 9 поддерживают SVG встроенными средствами, а для Internet Explorer 8 нужен специальный плагин (расширение), среди которых наиболее широко известен Adobe SVGViewer 3.03 (**www.adobe.com/svg/viewer/install/**), хотя с 1 января 2009 г. компания Adobe прекратила его дальнейшую поддержку. Имеется также бесплатный RENESIS Player Plugin 1.1.1 (**http://www.examotion.com**), который, видимо, еще будет развиваться, поскольку версия 1.1.1 не поддерживает анимацию. Следует иметь в виду, что пока SVG поддерживается различными браузерами в разной степени. Например, Internet Explorer 9 не поддерживает анимацию. Поэтому SVG в Web-дизайне в настоящее время применяют с осторожностью.

SVG-документ можно не только непосредственно загрузить и отобразить в окне браузера, но и вставить в (X)HTML-документ. Для этого используются теги `<object>` или `<embed>` (листинг 12.4).

Листинг 12.4. Примеры вставки SVG-документа в HTML-документ

```
<object data="URL-адрес SVG-файла" type="image/svg+xml">
Для просмотра SVG необходим плагин.
 <a href="http://www.adobe.com/svg/viewer/install/">
```

```
  Загрузить Adobe SVGViewer
</a>
</object>
<embed src="URL-адрес SVG-файла" type="image/svg+xml" pluginspage=
"http://www.adobe.com/svg/viewer/install/" />
```

В данных тегах с помощью атрибутов `width` и `height` можно указать размеры области отображения содержимого SVG-файла на Web-странице, выделяемой для элемента `<object>` или `<embed>`. Несогласованность значений атрибутов `width` и `height` тегов `<svg>` и `<object>` или `<embed>` может привести к неполному отображению SVG-содержимого.

Применение тега `<embed>` для вставки SVG-документа представляется сейчас более надежным.

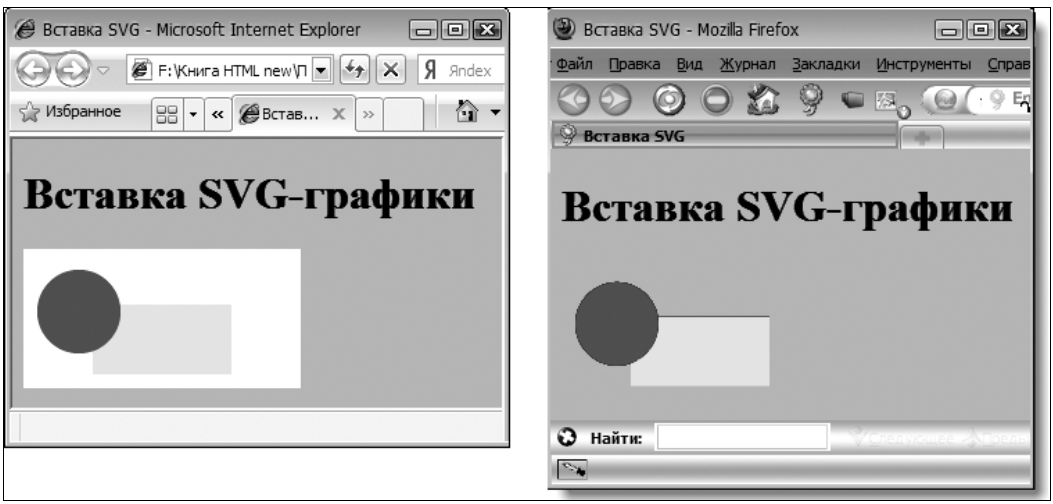


Рис. 12.8. Вставка SVG-документа в HTML-документ (листинг 12.5)

В листинге 12.5 приведен пример вставки SVG-документа в (X)HTML-документ с помощью тега `<embed>`, а на рис. 12.8 — его вид в Internet Explorer и Firefox. Для обеспечения межбраузерной совместимости следует позаботиться о фоне SVG-документа в нем самом.

Листинг 12.5. Пример вставки SVG-документа в (X)HTML-документ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Вставка SVG</title>
<style>
  body {background:grey}
</style
</head>
<body>
<h1>Вставка SVG-графики</h1>
```

```
<embed src="example.svg" type="image/svg+xml"
  pluginspage="http://www.adobe.com/svg/viewer/install/"
  width="200" height="100" />
</body>
</html>
```

12.2. Звук и видео

12.2.1. Основные форматы звуковых и видеофайлов

В настоящее время существует большое количество форматов аудио- и видеофайлов. Пользователя в основном интересует главный показатель — соотношение качества и объема соответствующей информации. Для публикации в Интернете очень важно сократить объем файла и в то же время сохранить приемлемое качество воспроизведения.

Наиболее популярные форматы аудиофайлов для Интернета (здесь и далее в скобках указаны типичные расширения файла):

MP3 (mp3), MPEG-2 (mpa), WMA (wma), AIFF (aiff, snd), MIDI (mid) и WAV (wav).

Наибольшее сжатие данных достигается за счет применения так называемых психоакустических моделей восприятия звуковой информации, учитывающих особенности человеческого слухового аппарата. В частности, ухо обычного человека не воспринимает слишком низкие (менее 16—20 Гц) и слишком высокие (более 16—20 кГц) частоты звуковых колебаний. Такие модели допускают некоторую потерю оригинальной звуковой информации с сохранением приемлемого качества ее воспроизведения. При этом достигается существенное сжатие данных. Например, в широко распространенных форматах MP3 и WMA для сжатия данных применяется некая психоакустическая модель, а формат WAV не предполагает никакого сжатия. В 2002 г. появился новый формат OGG/Vorbis (ogg), созданный некоммерческой организацией Xiphophorus, который призван конкурировать с MP3. Однако пока его поддерживают только Firefox , Chrome и Safari.

Из видеоформатов наиболее широко применяются следующие:

- MPEG (Moving Pictures Experts Group) — группа международных форматов, из которых наиболее популярны следующие два:
  - MPEG-2 (mpg, mpeg) — первоначально разработан для передачи цифрового видео широковещательного качества, используется в DVD и цифровом телевидении, в том числе и высокой четкости (HDTV);
  - MPEG-4 (mp4) — рассчитан на передачу небольших потоков видео для применения в видеотелефонах, мультимедийной электронной почте и т. п.;
- WMV (Windows Media Video) — формат, разработанный Microsoft (wmv);
- AVI (Audio Video Interleaved) — формат-контейнер, разработанный Microsoft, который может содержать данные других форматов, например, MPEG; не накладывает ограничений на тип кодеков (avi);
- 3GP (3rd Generation Phone) — формат файлов видео и аудио для мобильных телефонов третьего поколения (3gp);
- MOV (mov, qt) — основной формат Apple QuickTime;

- FLV (Flash Video) — формат от Adobe (ранее Macromedia), рассчитанный на передачу потокового видео через Интернет (flv);
- SWF (Shock Wave Format) — Adobe Flash-анимация (swf);
- OGG (ogg) — формат для потокового звука (Vorbis) и видео (Theora), созданный организацией Xiphophorus;
- WebM — новый формат (анонсирован в 2010 г.) потокового видео с использованием видеокодека VP8 и аудиокодека Vorbis, призванный заменить формат H.264. Разрабатывается Google вместе с Mozilla и Opera.

12.2.2. Вставка звука и видео в (X)HTML-документ

На Web-странице можно разместить текстовую или/и графическую ссылку на звуковой или видеофайл. В этом случае возможны два варианта действий: открыть файл в ассоциированном с ним проигрывателе или сохранить его на локальном носителе информации, например диске. Обычно так и поступают, когда имеют дело с большими по объему файлами непотокового звука и видео. Вместе с тем, возможность открыть файл непосредственно в заданном месте Web-страницы имеется. Данный вариант мы и рассмотрим.

Теги *<object>* и *<embed>*

Вставка в (X)HTML-документ звука и видео из файлов соответствующих форматов производится с помощью тегов `<object>` и `<embed>`. При этом в окне браузера открывается окно проигрывателя в зависимости от плагина, ассоциированного с типом загружаемого файла. Например, это может быть проигрыватель Apple QuickTime, Windows Media Player, Adobe Flash Player и др. Для звуковых файлов проигрыватель представляется панелью управления воспроизведением, а для видео — еще и окном отображения содержимого.

Стандартный универсальный тег вставки в документ внешнего содержимого — `<object>`. Однако его поддержка в разных браузерах пока отличается. Тег `<embed>` служит той же цели. Хотя он не описан в стандарте HTML 4, его поддерживают все браузеры примерно одинаково и, более того, его применение сейчас оказывается более надежным, чем использование тега `<object>`. Возможно, поэтому тег `<embed>` включен в спецификацию HTML 5.

Вот две основные схемы применения тегов `<object>` и `<embed>` для вставки аудио/видео:

```
<object data= "URL-адрес файла" type="MIME-тип файла">
  альтернативный текст и/или изображение
</object>
<embed src="URL-адрес файла" type="MIME-тип файла"/>
```

На практике нередко применяют оба тега, вкладывая `<embed>` в контейнер `<object>`:

```
<object data= "URL-адрес файла" type="MIME-тип файла">
  <embed src="URL-адрес файла" type="MIME-тип файла"/>
</object>
```

Данный способ позволяет загрузить файл посредством какого-нибудь одного элемента (`<object>` или `<embed>`), который воспринимается браузером.

Вот несколько примеров MIME-типов звуковых файлов: audio/aiff, audio/midi, audio/wav, audio/mpeg, audio/mp4, audio/x-ms-wma. Для видео наиболее широко применяются следующие MIME-типы: video/mpeg, video/x-mpeg, video/mp4, video/x-ms-wmv, video/quicktime и др.

Для элементов `<object>` и `<embed>` с помощью атрибутов или стилевых параметров `width` и `height` необходимо правильно задавать размеры, чтобы не было искажения или усечения видео, а также была видна панель управления воспроизведением. Если размеры не указывать, то результат зависит от браузера и плагина, который воспроизводит файл аудио/видео. Не исключено, что при незаданных размерах видео вообще не будет воспроизведено.

В листинге 12.6 приведен пример вставки звука формата MP3 и видеоформатов MPEG и WMV посредством тегов `<embed>` и `<object>`.

Листинг 12.6. Вставка звука и видео

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Мультимедиа</title>
<style type="text/css">
  #audio {width:100px;height:30px}
  #video1 {width:352px;height:288px}
  #video2 {width:200px;height:200px}
</style>
</head>
<body>
<embed   src="music.mp3"  type="audio/mp3"      id="audio"/>
<embed   src="video1.mpg" type="video/mpeg"     id="video1">
<object data="video2.wmv" type="video/x-ms-wmv" id="video2">
</object>
</body>
</html>
```

Чтобы воспроизвести звуковой файл в фоновом режиме (без показа панели управления), в теге `<embed>` предусмотрен атрибут `hidden`. Однако надежнее просто назначить нулевые размеры для элементов `<embed>` и `<object>`.

Вставка видео в форматах FLV (Flash-видео) и SWF (Flash-анимация) также осуществляется с помощью тегов `<object>` и `<embed>`, требует установленного плагина, например, Adobe Flash Player и имеет некоторую специфику, которую мы рассмотрим в *разд. 12.2.4* и *12.2.5* соответственно.

Теги *<audio>* и *<video>*

В HTML 5 для вставки звука и видео в Web-страницу предусмотрены внутристрочные контейнерные теги `<audio>` и `<video>`. Браузеры, поддерживающие данные теги, должны воспроизводить медиафайлы своими встроенными проигрывателями, не полагаясь на внешние плагины. В настоящее время теги вставки видео и аудио поддерживают Firefox, Opera, Chrome, Safari и Internet Explorer 9 (версия 8 их не поддерживает).

Вот простейшие примеры использования тегов <audio> и <video>.

Примеры

```
<audio src="mymusic.mp3" controls="controls">
  Ваш браузер не поддерживает тег audio
</audio>
<video src="mymovie.ogg" controls="controls" width="320" height="240">
  Ваш браузер не поддерживает тег audio
</video>
```

Вот некоторые атрибуты тегов <audio> и <video>:

- src — URL-адрес файла воспроизведения;
- controls="controls" — если указан, то отображается панель управления воспроизведением, а противном случае — нет;
- autoplay="autoplay" — если указан, то воспроизведение начинается автоматически по готовности, а в противном случае для запуска воспроизведения следует щелкнуть на соответствующей кнопке панели управления;
- autobuffer="autobuffer" — если указан, то по рекомендации W3C проигрыватель должен обеспечить некий эффект, создающий ощущение, что файл может быть воспроизведен;
- loop="loop" — если указан, то задает циклическое воспроизведение до тех пор, пока оно не будет остановлено с помощью соответствующей кнопки управления, а в противном случае воспроизведение выполняется один раз и возобновляется с помощью соответствующей кнопки управления;
- poster — принимает в качестве значения URL-адрес графического файла и может использоваться в теге <video> для отображения картинки, пока видео не готово к воспроизведению;
- preload = "preload" — если указан, то воспроизведение начнется как только страница будет загружена;
- width, height — числа, задающие ширину и высоту в пикселах окна проигрывателя видео.

В теги <audio> и <video> можно вставить тег <source> для определения одного или нескольких медиаресурсов. С помощью атрибутов указывают дополнительные параметры, помогающие браузеру определить, какой кодек следует использовать. С помощью нескольких тегов <source> задают варианты звука или видео для различных браузеров или/и кодеков.

Пример использования тега <source> иллюстрирует листинг 12.7.

Листинг 12.7. Пример использования тега <source>

```
<video controls="controls">
  <source src="http://anyserver.ru/myvideo.ogg"
          type="video/ogg; codecs= 'theora, vorbis'"/>
```

```
  <source src="http://anyserver.ru/myvideo.mp4"
          type="video/mp4; codecs='mp4v.20.8, mp4a.40.2'"/>
  Ваш браузер не поддерживает тег video
</video>
```

Кроме альтернативного текста в тег `<video>` можно вставить графическое изображение или Flash-ролик (см. *разд. 12.2.4*), которые будут отображаться, если браузер не поддерживает тег `<video>`.

12.2.3. Вставка FLV-видео

В настоящее время существуют Web-сервисы, такие как YouTube, RuTube, Google Video и др., которые предоставляют возможность просмотра и скачивания flv-файлов, а также публикации в Web ваших видеороликов. Видео в формате FLV (Flash Video) потоковое и, следовательно, начинает воспроизводиться до завершения загрузки flv-файла. Его дозагрузка происходит во время воспроизведения, которое обеспечивает плагин Adobe Flash Player (**http://get.adobe.com/flashplayer**). Файлы FLV можно также воспроизвести с помощью проигрывателей Adobe Media Player, GOM, Winamp, Media Player Classic и др.

Flv-файлы обычно создают путем конвертации файлов других форматов (AVI, MPEG, DivX и др.), полученных с помощью видеокамер и программ обработки видео. Общедоступные конвертеры нетрудно найти в Интернете, например Any Video Converter (**www.any-video-converter.com**).

Чтобы воспроизвести flv-файл на своей Web-странице, не используя услуги Web-сервисов и внешних проигрывателей, можно с помощью тега `<object>` или `<embed>` вставить FLV-проигрыватель и загрузить в него требуемый flv-файл. FLV-проигрыватель представляется в виде swf-файла, который можно создать посредством пакета Adobe (Macromedia) Flash. Вместе с тем, такие проигрыватели распространяются в Интернете как в окончательном виде, так и с возможностью доработки или настройки. Например, бесплатный FLV-проигрыватель JW FLV Player в виде swf-файла можно получить по адресу **www.longtailvideo.com**. Сайт **http://uppod.ru** предлагает в режиме онлайн сконструировать FLV-проигрыватель по своему вкусу и скачать его на свой сайт.

Итак, FLV-проигрыватель представляет собой swf-файл, который при желании можно переименовать по своему усмотрению. Если вы хотите показывать FLV-видео со своего сайта, то на нем следует разместить и swf-файл FLV-проигрывателя.

В листинге 12.8 показана вставка flv-файла mymovie.flv посредством тегов `<object>` и `<embed>`, причем `<embed>` вложен в `<object>`. Здесь FLV-проигрыватель представлен файлом flvplayer.swf, которому с помощью параметра `flashvars` передается имя (URL-адрес) flv-файла. Разумеется, имя swf-файла проигрывателя может быть и иным.

Объект, вставляемый в (X)HTML-документ, может иметь параметры. В теге `<object>` параметры задают посредством вложенных тегов вида

```
<param name="имя_параметра" value="значение_параметра">,
```

а в теге `<embed>` — с помощью атрибутов вида

```
имя_параметра ="значение_параметра"
```

В рассматриваемом случае задают следующие параметры:

- `flashVars` со значением `"file=mymovi.flv"` — определяет файл, который требуется воспроизвести;
- `allowFullScreen` со значением `true` — разрешает воспроизведение в полноэкранном режиме (`false` — запрещает).

В тег `<object>` вложен еще один тег `<param name="movie" value="flvplayer.swf"/>`, который необходим для Internet Explorer.

При любых ненулевых размерах (`width` и `height`) окна проигрывателя видео масштабируется так, чтобы оно вписывалось в него без искажения оригинальных пропорций.

Листинг 12.8. Вставка FLV-видео

```
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head>
<title>Мультимедиа</title>
<style type="text/css">
   embed,object {width:300px;height:200px}
</style>
</head>
<body>
<object data=" flvplayer.swf " type="application/x-shockwave-flash">
  <param name="movie" value="flvplayer.swf "/>
  <param name="flashVars" value="file=masterbook.flv">
  <param name="allowFullScreen" value="true"/>
  <embed src="flvplayer.swf" type="application/x-shockwave-flash"
         flashVars="file=masterbook.flv" allowFullScreen ="true"/>
</object>
</body>
</html>
```

Рассмотренный пример был проверен на FLV-проигрывателе JW FLV Player (**www.longtailvideo.com**). Проигрыватель проекта Uppod (**http://uppod.ru**) кроме swf-файла включает закодированные текстовые файлы стилей, определяющих его внешний вид и функциональные возможности. Так что, при его использовании параметр `flashVars` содержит как минимум ссылки на flv-файл и txt-файл стиля:

- `<param name="flashVars" value="st=`*файл_стилей*`&file=`*медиа_файл*`" />` (в теге `<object>`);
- `flashVars="st=`*файл_стилей*`&file=`*медиа_файл*`"` (в теге `<embed>`).

На сайте Uppod (**http://uppod.ru**) можно создать файл стилей, предусматривающий плейлист, а его содержимое определить как значение параметра `flashVars`.

Пример

```
'st=styles/mystyle.txt&amp;
pl={"playlist":[
{"file":"video/myvideo1.flv","comment":"1. Мой первый фильм"},
{"file":"video/myvideo2.flv","comment":"2. Мой второй фильм"},
{"file":"video/myvideo3.flv","comment":"3. Мой третий фильм"}]}'
```

Проигрыватель от Uppod обеспечивает много других возможностей, с которыми можно познакомиться на сайте разработчиков.

12.2.4. Вставка Flash-фильмов

Flash-фильмы (ролики) могут содержать статическую и анимационную масштабируемую графику, видео со звуковым сопровождением, а также элементы управления (пользовательский интерфейс). С помощью технологии Flash можно создавать страницы сайта целиком, хотя в настоящее время на Web-страницах чаще всего присутствуют лишь небольшие ролики рекламного и оформительского характера.

Фильмы (документы) Flash создают в специальной системе разработки приложений Adobe Flash (ранее — Macromedia Flash) и сохраняют, как правило, в файлах формата SWF с расширением swf. Эти файлы затем можно воспроизвести в проигрывателе, например, Adobe Media Player или вставить в (X)HTML-документ посредством тегов `<object>` и `<embed>`. В последнем случае они будут воспроизводиться в окне браузера с помощью плагина, например, Adobe Flash Player (**http://get.adobe.com/flashplayer**). В *разд. 12.2.3* мы уже рассматривали пример вставки swf-файла, который выполнял функцию FLV-проигрывателя.

В листинге 12.9 приведен простейший вариант вставки некоторого Flash-фильма из файла myflash.swf. При этом тег `<embed>` вложен в `<object>`, размеры окна заданы посредством CSS, хотя их можно определить и с помощью атрибутов `width` и `height`. Обратите внимание, что в теге `<object>` вложение тега `<param name="movie" value="myflash.swf" />` обязательно. Однако атрибут `data="myflash.swf"`, дублирующий информацию в данном теге `<param>`, может отсутствовать.

Листинг 12.9. Вставка Flash-фильма

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<head>
<title>Flash фильм</title>
<style type="text/css">
    .myflash {width:270px; height:200px}
</style>
</head>
<body >
<object data="myflash.swf" type="application/x-shockwave-flash"
      class="myflash">
```

```
  <param name="movie" value="myflash.swf" />
  <embed src=" myflash.swf" type="application/x-shockwave-flash"
      class="myflash"/>
</object>
</body>
</html>
```

Flash-фильм воспроизводится в области, заданной параметрами `width` и `height`, которая не имеет панели управления. Вместо нее можно использовать контекстное меню, открываемое щелчком на области фильма правой кнопкой мыши.

В системе Adobe Flash имеются средства публикации созданного фильма в различных форматах, в том числе и в HTML. Параметры фильма можно установить с помощью специального диалогового окна. При публикации в HTML-формате на диске вашего компьютера кроме swf-файла создается еще и html-файл, содержащий контейнерный тег `<object>` с вложенным тегом `<embed>` и всем необходимым, чтобы показать соответствующий swf-файл в окне браузера. Данный фрагмент (X)HTML-кода можно скопировать в нужное место своего (X)HTML-документа и при необходимости скорректировать, чтобы Flash-фильм демонстрировался с подходящими параметрами.

В теге `<object>` параметры задают посредством вложенных тегов вида

`<param name="`*имя_параметра*`" value="`*значение_параметра*`">`,

а в теге `<embed>` — с помощью атрибутов вида

имя_параметра`="`*значение_параметра*`"`.

Вот несколько наиболее важных параметров:

- `quality` — уровень качества воспроизведения; принимает значения:
  - `low` (низкое);
  - `medium` (среднее);
  - `high` (высокое);
  - `best` (наилучшее);
  - `autolow` (автонизкое), `autohigh` (автовысокое); при значении `autolow` (`autohigh`) воспроизведение начинается с низким (высоким) качеством и переходит к высокому (низкому) качеству, если проигрыватель определит, что оно поддерживается (не поддерживается) компьютером;
- `loop` — цикличность воспроизведения; значение `true` (по умолчанию) означает циклическое воспроизведение, а `false` — однократное;
- `menu` — значение `true` (по умолчанию) открывает доступ к командам управления воспроизведением фильма в контекстном меню, которое открывается щелчком правой кнопки мыши на области фильма; значение `false` удаляет команды управления из контекстного меню;
- `play` — режим воспроизведения; значение `true` (по умолчанию) означает начало воспроизведения сразу же по окончании загрузки; при значении `false` сначала отображается только первый кадр фильма, а чтобы продолжить его воспроизведение

следует выбрать соответствующую команду из контекстного меню или щелкнуть на кнопке в самом фильме, если такая предусмотрена его автором;

- `wmode` — режим окна; возможны следующие значения:
  - `opaque` — фон фильма непрозрачный, закрывающий элементы Web-страницы, которые расположены под областью воспроизведения; данное значение принято по умолчанию;
  - `transparent` — фон фильма прозрачный; элементы Web-страницы, расположенные под областью воспроизведения, видны;
- `scale` — определяет масштаб фильма при размещении его в области, заданной параметрами `width` и `height`; по умолчанию (если данный параметр не задан) фильм масштабируется так, чтобы он разместился в заданной области без искажения пропорций, при этом могут появиться поля (точнее, границы) около фильма; параметр `scale` может принимать следующие значения:
  - `noscale` — фильм не масштабируется, и если он не вписывается в размеры окна, то усекается;
  - `noborder` — фильм масштабируется так, чтобы он занял всю заданную область без искажения пропорций, но, возможно, с обрезкой краев;
  - `exactfit` — фильм масштабируется так, чтобы он занял всю заданную область, при этом возможно искажение пропорций;
- `bgcolor` — задает цвет фона окна фильма; принимает значения цвета; данный параметр действует, если параметр `wmode` не имеет значения `transparent`;
- `allowScriptAccess` — разрешение на выполнение внешних скриптов (сценариев); данный параметр следует задать, если Flash-фильм использует методы `fscommand()` и `getURL()`, обеспечивающие выполнение внешних скриптов; параметр `allowScriptAccess` принимает значения:
  - `none` — не разрешается выполнять внешние скрипты;
  - `someDomain` — разрешается выполнять внешние скрипты, если они расположены в том же домене, что и swf-файл с Flash-фильмом (по умолчанию);
  - `always` — разрешается выполнять любые внешние скрипты;
  - `flashVars` — принимает в качестве значения строку, содержащую определения переменных, передаваемых Flash-фильму; пример использования данного параметра рассмотрен в *разд. 12.2.3*.

В теге `<embed>` можно указать атрибут `pluginspage`, принимающий в качестве значения URL-адрес плагина для воспроизведения Flash-фильма, например,

```
plaginspage="http://get.adobe.com/flashplayer/"
```

Если на компьютере пользователя плагин не установлен, то произойдет переход на Web-страницу по указанному адресу, где можно получить недостающий плагин.

В листинге 12.10 приведен пример вставки Flash-фильма с типичными параметрами, содержащего единственную анимированную кнопку. Размеры области, занимаемой Flash-фильмом, заданы посредством CSS, а сам фильм имеет прозрачный фон. Вид данного примера в браузере показан на рис. 12.9.

Листинг 12.10. Вставка Flash-фильма с параметрами

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<head>
<title>Кнопка</title>
<style type="text/css">
  body {background: url("fon.jpg") repeat} /* фон страницы */
  .myflash {width:300px;height:200px} /* размеры области фильма */
</style>
</head>
<h1>Моя страница</h1>
<object data="button.swf" type="application/x-shockwave-flash"
   class="myflash">
<param name="movie" value="button.swf"/>
<param name="allowScriptAccess" value="sameDomain"/>
<param name="quality" value="high"/>
<param name="wmode" value="transparent"/>
<embed src="button.swf" type="application/x-shockwave-flash"
   class="myflash"
   allowScriptAccess="sameDomain"
   quality="high"
   wmode="transparent"
   pluginspage="http://get.adobe.com/flashplayer/"
/>
</object>
</body>
</html>
```

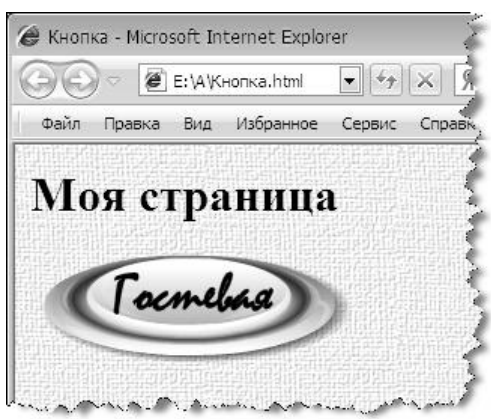


Рис. 12.9. Flash-фильм, содержащий кнопку (листинг 12.10)

12.3. Вставка (X)HTML-документов

В один (X)HTML-документ можно вставить другой. Например, Web-страница, сформированная некоторым (X)HTML-документом, содержит ссылки на другие (X)HTML-документы, а также место, в котором последние следует отображать.

Вставить (X)HTML-документ можно с помощью тега `<iframe>` (плавающий фрейм) с атрибутами `src="`*`URL-адрес файла`*`"` и `type="text/html"`. Тег `<iframe>` имеет атрибут `name`, значение которого можно задать в качестве значения атрибута `target` тега ссылки `<a>`, чтобы при ее активизации запрашиваемый документ отображался в указанном фрейме. Примеры использования плавающего фрейма рассматривались в *главе 6*.

Однако следует иметь в виду, что не любой (X)HTML-документ можно загрузить во фрейм. Например, главную страницу Яндекса (**www.yandex.ru**) загрузить во фрейм не удастся. При попытке это сделать она откроется в новой вкладке или окне браузера. Чтобы предотвратить загрузку документа во фрейм, достаточно снабдить его простым скриптом, но об этом будет рассказано в *разд. 15.1.3*.

12.4. Вставка элементов управления ActiveX

12.4.1. Что такое ActiveX

Многие современные приложения Windows создавались и сейчас создаются с использованием механизмов OLE (Object Linking and Embedding — связывание и встраивание объектов), основанных на компонентной объектной модели (Component Object Model, COM). ActiveX — дальнейшее развитие этой технологии, разработанной Microsoft специально для создания сетевых приложений. Технология базируется на DCOM (Distributed Component Object Model — распределенная компонентная объектная модель) и реализована в библиотеках ActiveX, которые и компактнее, и оперативнее библиотек OLE. При этом обеспечивается совместимость: любой компонент OLE будет работать и с библиотекой ActiveX. Программные компоненты, созданные по технологии ActiveX, могут взаимодействовать друг с другом по сети независимо от языка, на котором они написаны.

Элементы управления ActiveX — компоненты, работающие на компьютере-клиенте. По многим параметрам это нечто среднее между подключаемыми модулями расширений (плагинами) и апплетами (см. *разд. 12.5*). Элемент управления (control) представляет собой обычно небольшую программу, для которой браузер выделяет на Web-странице некоторый прямоугольный участок. В пределах этого участка элемент управления полностью отвечает за визуальное отображение и взаимодействие с пользователем.

Встретив в (X)HTML-документе ссылку на некий элемент управления ActiveX (тег `<object>` с соответствующими атрибутами), браузер сначала проверяет, установлен ли он на компьютере пользователя. Если такой элемент управления уже установлен, то браузер запускает его и передает ему необходимые данные. В противном случае браузер обращается к серверу по заданному адресу, перекачивает и устанавливает элемент управления, который теперь может использоваться не только браузером, но и другими приложениями, способными работать с так называемыми нестандартными элементами управления OLE (OLE Custom Controls, OCX).

Наибольший эффект применения элементов ActiveX обеспечивают сценарии. Однако в ряде случаев можно обойтись и без них.

Элементы ActiveX работают только под Windows. Любые модули расширения (плагины) в Internet Explorer это элементы ActiveX. Элементы ActiveX, вставленные в

(X)HTML-документы, "понимает" только Internet Explorer для Windows, а потому их применение в документах, публикуемых в Web, довольно ограничено.

Чтобы вставить элемент ActiveX в (X)HTML-документ, необходимо в теге `<object>` указать следующие атрибуты:

- `classid` — уникальный регистрационный идентификатор встраиваемого компонента (например, элемента управления ActiveX для Internet Explorer); собственно идентификатору предшествует ключевое слово `clsid`, например

  ```
  classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  ```

- `codebase` — URL-адрес, по которому браузер может взять файл (обычно — упакованный) компонента, чтобы установить его при отсутствии на компьютере пользователя. Вслед за URL-адресом указывается версия компонента в виде строки: `#Version=M1,M2,m1,m2`. Здесь `M1` и `M2` — максимальные, а `m1` и `m2` — минимальные мажорный и минорный номера версии, доступные на сервере. Встречая в HTML-документе ссылку на компонент, браузер проверяет его наличие на компьютере пользователя. Если компонента нет, то он загружается и устанавливается (с разрешения пользователя или без него в зависимости от настроек безопасности браузера). Если на компьютере пользователя установлена более новая версия, чем указанная атрибутом `codebase`, то компонент не загружается.

  Пример:

  ```
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab #version=8,0,0,0"
  ```

  Здесь указан URL-адрес упакованного cab-файла Flash-проигрывателя единственной версии 8.

- `codetype` — MIME-тип файлов, указанных атрибутом `codebase`. Например, для элементов управления ActiveX значением этого атрибута (если он указан) является `application/x-oleobject`.

В зависимости от конкретного элемента ActiveX возможны и другие атрибуты, а также параметры, задаваемые вложенными в `<object>` тегами `<param>`. Состав и возможные значения параметров следует узнавать у автора или поставщика элемента ActiveX.

12.4.2. Примеры элементов ActiveX

Flash-проигрыватель

В *разд. 12.2.4* уже рассматривалась вставка в (X)HTML-документ Flash-фильма с помощью тега `<object>`. В Internet Explorer этот тег инициирует подключение элемента ActiveX, выполняющего роль Flash-проигрывателя. В других браузерах подключается не ActiveX, а аналогичный плагин. Вместе с тем, для вставки Flash-проигрывателя и загрузки в него фильма (swf-файла) можно явно указать, какой элемент ActiveX следует выбрать. Это делается с помощью атрибута `classid` тега `<object>` (листинг 12.11).

Листинг 12.11. Пример вставки Flash-проигрывателя

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase=   "http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
 #version=8,0,0,0" width="550" height="400" >
```

```
<param name="movie" value="mymovie.swf" />
<param name="allowScriptAccess" value="sameDomain" />
<embed src="mymovie.swf" width="550" height="400"
       name="mymovie" type="application/x-shockwave-flash"
       allowScriptAccess="sameDomain"
       pluginspage="http://www.macromedia.com/go/getflashplayer"
/>
</object>
```

В данном примере тег `<object>` инициирует элемент ActiveX с идентификатором, указанным в атрибуте `classid`. В атрибуте `codebase` указывается требуемая версия (8). Если на компьютере пользователя такой элемент не установлен, то браузер попытается загрузить его с сайта по указанному в `codebase` адресу. Вложенный в `<object>` тег `<embed>` обеспечивает подключение Flash-проигрывателя в виде плагина в браузерах, не поддерживающих ActiveX.

Adobe SVGViewer

Плагин Adobe SVGViewer 3.03 для просмотра SVG-документов в Internet Explorer выполнен как элемент ActiveX. Если он установлен, то вставить svg-файл в (X)HTML-документ можно, например, как показано в листинге 12.12.

Листинг 12.12. Вставка svg-файла

```
<object classid="clsid:377B5106-3B4E-4A2D-8520-8767590CAC86"
    codebase="http://www.adobe.com/svg/viewer/install/"
    data="mysvg.svg" type="image/svg+xml"
    width="500" height="400">
<embed src="mysvg.svg" type="image/svg+xml"
    pluginspage="http://www.adobe.com/svg/viewer/install/"
/>
</object>
```

Календарь

Элемент управления ActiveX, реализующий календарь, встроен в Internet Explorer. Для отображения такого календаря на Web-странице (рис. 12.10) достаточно использовать следующий код:

```
<object classid="clsid:8E27C92B-1264-101C-8A2F-040224009C02"
        width="300" height="200"></object>
```

Обслуживание табличных данных в текстовых файлах

Данные с табличной организацией можно хранить в обычных текстовых файлах. При этом разделителями полей (столбцов) и строк могут быть различные символы, не используемые в представлении собственно данных. Файлы с табличными данными еще называют CSV-файлами (Comma Separated Values — значения, разделенные запятой), хотя разделителем полей может служить не только запятая.

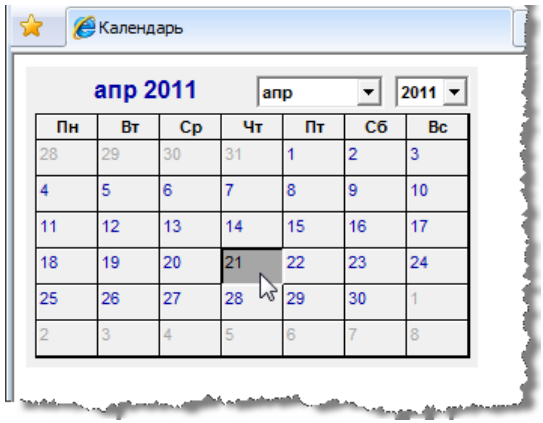


Рис. 12.10. Календарь, реализуемый элементом управления ActiveX

Создавать такие файлы можно вручную с помощью текстового редактора (например, Блокнота Windows). Кроме того, в большинстве систем работы с базами данных возможно преобразование специфических файлов в текстовые с заданными разделителями.

Для работы с табличными данными в текстовом файле подходит элемент ActiveX Tabular Data Control (TDC — управление табличными данными). В (X)HTML-документ элемент управления TDC вставляется с помощью тега `<object>` (листинг 12.13).

Листинг 12.13. Вставка элемента управления TDC

```
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        id="mydbcontrol">
  <param name="FieldDelim" value="|">
  <param name="DataURL" value="mydb.txt">
  <param name="UseHeader" value="true">
</object>
```

Здесь с помощью тегов `<param>` задаются параметры:

- `FieldDelim` — разделитель полей (ячеек); в примере значением этого параметра является вертикальная черта (`|`);
- `DataURL` — URL-адрес текстового файла с данными; в примере указано просто имя файла mydb.txt;
- `UseHeader` — определяет, содержит ли первая строка в текстовом файле имена полей.

Разделителем строк обычно служит невидимый символ перевода строки (клавиша <Enter>). Однако можно задать любой символ в качестве разделителя строк с помощью параметра:

```
<param name="RowDelim" value="&#ASCII-код;">
```

Значение атрибута `value` по умолчанию — `
`, т. е. ASCII-код символа перевода строки. Кроме перечисленных, элемент управления имеет и другие параметры.

В первой строке текстового файла (источника данных) через символьный разделитель следуют имена столбцов. Они не обязательно должны совпадать с теми заголовками столбцов, которые вы хотите видеть на экране. Главное их назначение в том, чтобы обеспечить доступ к данным. Это так называемые *имена полей* базы данных или, другими словами, *идентификаторы полей*. Правила их образования такие же, как и для имен переменных (первый символ — буква или знак подчеркивания, не должно быть пробелов и т. п.)

На рис. 12.11 показан текстовый файл, содержащий сведения о некоторых моих книгах, в котором первая строка хранит имена полей, а остальные — собственно данные (значения полей). Разделителем полей здесь является вертикальная черта (`|`).

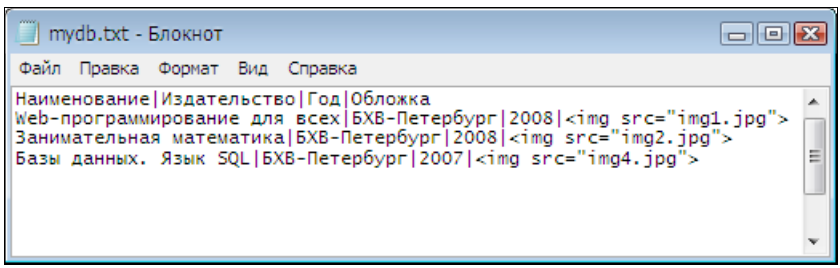


Рис. 12.11. Текстовый файл с табличными данными

Отображать данные на Web-странице должны элементы, определяемые HTML-тегами. Это могут быть теги таблицы, поля ввода данных, кнопки и др. Чтобы это стало возможным, необходимо привязать данные к элементам (X)HTML-документа.

Для привязки данных к элементам (X)HTML-документа необходимо в соответствующих тегах атрибуту `datasrc` присвоить значение, совпадающее со значением атрибута `id` тега `<object>`, установив перед ним символ `#`. Тем самым мы укажем, у какого объекта-поставщика следует взять данные.

С помощью атрибутов `datafld` задают имена полей из числа тех, что определены в первой строке текстового файла. В текстовом файле может содержаться не только обычная текстовая информация, но и HTML-коды. Чтобы эти коды интерпретировались, а не просто отображались, следует использовать атрибут `dataformatas="html"`. Это позволяет, например, вставлять в ячейки таблицы изображения, гиперссылки, кнопки и другие элементы. Например, чтобы отобразить в окне браузера данные из текстового файла mydb.txt (рис. 12.11) в виде таблицы, можно выполнить (X)HTML-код листинга 12.14. Вид этого документа в окне браузера показан на рис. 12.12.

Листинг 12.14. Отображение в таблице данных из текстового файла

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Табличные данные</title>
<style>
  table {
    border-collapse:collapse;
    border:solid 2px black
  }
```

```
  td, th {border: solid 1px black}
</style>
</head>
<body>
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        id="mydbcontrol">
    <param name="FieldDelim" value="|">
    <param name="DataURL"    value="mydb.txt">
    <param name="UseHeader"  value="true">
</object>
<table datasrc="#mydbcontrol">
    <caption>Мои книги</caption>
    <thead>
      <th>Название</th>
      <th>Издательство</th>
      <th>Год выпуска</th>
      <th>Обложка</th>
    </thead>
    <tr>
      <td><span datafld=Наименование></span></td>
      <td><span datafld=Издательство></span></td>
      <td><span datafld=Год></span></td>
      <td><span datafld="Обложка" dataformatas="html"></span></td>
    </tr>
</table>
</body>
</html>
```

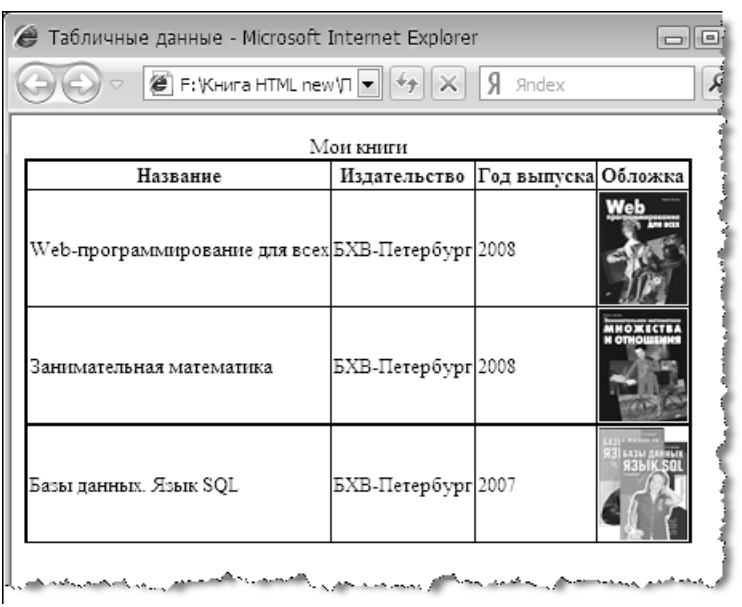


Рис. 12.12. Пример отображения в таблице данных из текстового файла (листинг 12.14)

В листинге 12.14 в теге `<table>` значение атрибута `datasrc` — ссылка на элемент ActiveX TDC (значение атрибута `id` в теге `<object>`). Внутри тегов данных `<td>`, в контейнерах `<span>`, значениями атрибутов `datafld` являются имена полей, как они определены в текстовом файле. Написание этих имен должно в точности соответствовать указанным в первой строке текстового файла данных. Однако порядок следования и количество выводимых на экран полей может быть произвольным. Например, можно вывести не все поля или, наоборот, продублировать некоторые из них. Кроме того, в теге `<table>` можно определить порядок отображения и состав отображаемых полей (столбцов).

Таким образом, при использовании TDC организуется взаимодействие трех компонентов: элемент управления TDC устанавливает связь с источником данных (текстовым файлом), элемент `<table>` связывается с TDC и источником данных, текстовый файл содержит собственно данные.

Элемент управления TDC имеет и другие параметры. В частности, есть параметры, с помощью которых можно установить фильтр (выборку) и сортировку данных:

```
<param name="FilterColumn"    value="имя_поля"
<param name="FilterCriterion" value="оператор_сравнения"
<param name="FilterValue"     value="образец"
```

С помощью указанных параметров задается логическое условие (критерий) фильтра следующего вида:

```
имя_поля оператор_сравнения образец
```

Например, отфильтровать из таблицы все строки, в которых поле `Издательство` имеет значение `"БХВ-Петербург"`, позволят следующие теги параметров:

```
<param name="FilterColumn"    value="Издательство"
<param name="FilterCriterion" value="=="
<param name="FilterValue"     value="БХВ-Петербург"
```

В качестве оператора сравнения допустимы символы `=` (неточное равенство), `==` (точное равенство), `!=` (неравенство), `>` (больше), `<` (меньше), `>=` (не меньше), `<=` (не больше).

Чтобы выключить (сбросить) фильтр и сделать доступными все строки таблицы, достаточно задать условие, которому заведомо удовлетворяют все строки таблицы. Например, следующие параметры сбрасывают текущий фильтр, поскольку пустая строка содержится в любой строке:

```
<param name="FilterCriterion" value="="
<param name="FilterValue"     value=""
```

По умолчанию фильтрация данных чувствительна к регистру, в котором набран образец. Управлять зависимостью от регистра можно с помощью параметра `CaseSensitive`:

```
<param name="CaseSensitive" value="значение">
```

Атрибут `value` может принимать значения `0` или `false` (не зависит) или `1` или `true` (зависит).

Для сортировки (упорядочения) данных предназначен следующий параметр:

```
<param name="SortColumn" value ="имя_поля">
```

Строки сортируются по значениям ASCII-кода значений заданного поля базы данных. Например, если мы хотим, чтобы названия книг располагались по алфавиту, необходимо записать:

```
<param name="SortColumn" value="Наименование">
```

По умолчанию сортировка осуществляется по увеличению значения ASCII-кода символов. Однако порядок можно изменить на противоположный, если указать параметр

```
<param name="SortAscending" value="0">
```

Отобразить данные, находящиеся во внешнем текстовом файле, могут и другие элементы (X)HTML-документа. Так, текстовую информацию можно показать в элементах, соответствующих тегам `<input type="text">` и `<textarea>`. Кнопки `<input type="button">` и `<button>` отображают текстовые данные в виде надписей. Если данные во внешнем файле представляют собой адрес графического файла, то его можно показать с помощью тега `<img>`. Во всех перечисленных случаях теги должны содержать атрибуты `datasrc` и `datafld`. При этом в указанных элементах будет отображаться информация из первой строки данных текстового файла. Чтобы показать данные из других строк, необходимо осуществить перемещение по файлу к требуемой строке. Для этого элемент управления TDC предоставляет свои методы и свойства, доступные в сценариях, но об этом будет рассказано в *разд. 15.3.1*.

В листинге 12.15 показано, как отобразить данные из рассмотренного ранее файла mydb.txt (рис. 12.11) в поле ввода и на кнопке. Содержимое поля ввода данных можно изменить вручную или с помощью скрипта. Однако изменения не сохранятся в текстовом файле (источнике данных).

Листинг 12.15. Отображение данных из текстового файла в поле ввода и на кнопках

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Табличные данные</title>
</head>
<body>
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        id="mydbcontrol">
    <param name=FieldDelim value="|"/>
    <param name=DataURL value="mydb.txt"/>
    <param name=UseHeader  value="true"/>
</object>
  <!-- Поле ввода -->
  <input type=text datasrc="#mydbcontrol" datafld="Наименование"
         size="30"/>
  <br/><br/>
  <!-- Кнопка -->
  <input type=button datasrc="#mydbcontrol" datafld="Наименование"/>
  <br/><br/>
```

```
  <!-- Кнопка -->
  <button datasrc="#mydbcontrol" datafld="Обложка"
          dataformatas="html">
  </button>
</body>
</html>
```

12.5. Вставка апплетов Java

12.5.1. Что такое апплет

Английское слово "applet" — уменьшительное от application (приложение). Так что, апплет — обычно небольшое приложение (программа), написанное на языке Java (не путать с JavaScript), выполняемое браузером или специальной программой просмотра. Апплет хранится на сервере, а в (X)HTML-документе с помощью специальных тегов задаются ссылка на него и параметры. При получении такого (X)HTML-документа браузер загружает и выполняет апплет, который, в свою очередь, может загружать и задействовать другие ресурсы. Как только необходимость в апплете пропадает (например, при переходе на другую страницу), он выгружается. Если элементы ActiveX и плагины устанавливаются на компьютере пользователя, то апплеты не устанавливаются, а передаются ему каждый раз, когда запрашиваются. К апплетам обычно прибегают, когда требуются особые графические или сетевые возможности. Однако в случае необходимости иметь особые графические средства лучше, пожалуй, использовать Flash, SVG или новый элемент HTML 5 `<canvas>`.

Исходный код апплетов пишут на объектно-ориентированном интерпретируемом и независимом от платформы языке программирования Java. Далее исходный код транслируется компилятором в специальный байтовый код, называемый J-кодом, который выполняется интерпретатором Java. Интерпретатор Java ориентирован на конкретную платформу (Windows, Macintosh, UNIX, Linux и др.), но сам J-код не зависит от платформы, что и обеспечивает переносимость программ на языке Java. J-код можно рассматривать как набор команд для некоторой виртуальной машины, реализуемой интерпретатором. Этот код может выполняться в любой среде, в которой работает виртуальная машина.

Основные элементы, из которых строится программа на языке Java, — классы, образующие иерархическую структуру. Класс представляет собой определение данных для решения некоторой задачи и методов их обработки. Исходный код на языке Java, описывающий класс, сохраняется в текстовом файле с расширением java. Этот файл компилируется в байтовый J-код с тем же именем, но расширением class. Такие файлы с J-кодом могут храниться на одном компьютере или быть распределены по нескольким компьютерам сети и динамически загружаться выполняющимся приложением (апплетом) по мере необходимости.

В принципе, на Java можно написать потенциально опасную для компьютера пользователя программу. Поэтому апплеты должны удовлетворять некоторым правилам безопасности, предъявляемым браузерами и Java. Так, апплет не может выполнять файловые операции чтения/записи на компьютере пользователя, устанавливать соединения с другими компьютерами сети, кроме компьютера, с которого он был загружен, запус-

кать другие программы на компьютере, на котором он сам выполняется, использовать библиотеки других языков (например, C++), изменять системные параметры.

Для проверки соблюдения апплетом правил безопасности имеется специальный класс, экземпляр которого выполняется виртуальной Java-машиной. При нарушении правил безопасности генерируется исключительная ситуация, на которую апплет реагирует определенным образом.

Программисты создают апплеты с помощью специальных средств разработки, например Java Development Kit (**http://www.javasoft.com**), и распространяют на платной или бесплатной основе. Например, интересные апплеты можно получить по адресу **http://www.wyka-warzecha.com**.

Ранее для вставки апплета применялся тег `<applet>`. Хотя он не входит в спецификации ни HTML 4, ни HTML 5, тем не менее поддерживается всеми современными браузерами. В настоящее время `<applet>` рекомендуется заменять тегом `<object>`. Здесь мы рассмотрим оба тега для вставки апплетов.

12.5.2. Вставка апплета посредством тега *<applet>*

Рассмотрим в качестве примера вставку в (X)HTML-код апплета ABCMenuMan.class, взятого с сайта **www.wyka-warzecha.com**. Этот апплет реализует меню (рис. 12.13), опции которого можно настроить по своему усмотрению.

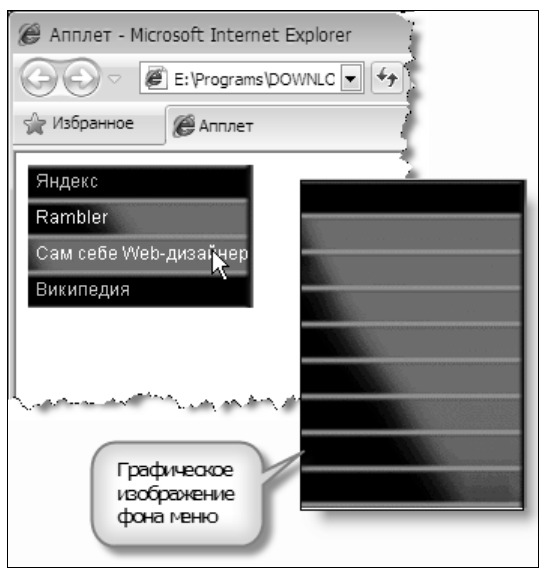


Рис. 12.13. Меню, реализованное с помощью апплета

С помощью тега `<applet>` данный апплет вставляется в (X)HTML-код так, как показано в листинге 12.16. Имя файла апплета указывают в атрибуте `code`. Параметры меню (имя графического файла для фона, число и названия опций и соответствующие им ссылки, ссылка на сайт автора и длительность цветового эффекта при переходе указателя мыши с одной опции на другую) задают посредством вложенных в `<applet>` тегов `<param>`. Для

настройки меню можно удалить или добавить соответствующие теги `<param>`, а также изменить значения их атрибутов `value`. При необходимости следует скорректировать количество опций (`<param name="noItems" value="4">`), а также размеры области отображения (`width` и `height`).

> ***Примечание***
>
> Данный апплет устроен так, что при удалении или изменении тега `<param name="copyright" value="http://www.wyka-warzecha.com"/>` он не будет выполняться.

За последним тегом `<param>` вставлен альтернативный текст и ссылка на сайт, с которого можно загрузить Java-плагин, если он не установлен на компьютере пользователя.

Листинг 12.16. Применение тега <applet>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Апплет</title></head>
<body>
<applet code="ABCMenuMan.class" width="150" height="95">
  <param name="copyright" value="http://www.wyka-warzecha.com"/>
  <param name="timingValue" value="75"/>
  <param name="noItems" value="4"/>
  <param name="aniMenu" value="menu.gif"/><!-- Фон меню -->

  <param name="title1" value="_Яндекс">
  <param name="title2" value="_Rambler">
  <param name="title3" value="_Сам себе Web-дизайнер">
  <param name="title4" value="_Википедия">

  <param name="link1" value="http://www.yandex.ru">
  <param name="link2" value="http://www.rambler.ru">
  <param name="link3" value="http://dunaevv1.narod.ru">
  <param name="link4" value="http://ru.wikipedia.org/wiki/">
Ваш браузер не поддерживает Java.
<a href="http://www.java.com/ru">
Скачайте последнюю версию Java-плагина
</a>
</applet>
</body>
</html>
```

12.5.3. Вставка апплета посредством тега *<object>*

При вставке апплета с помощью тега `<object>` существуют различные варианты для Internet Explorer и других браузеров, таких как Firefox, Opera, Safari и Chrome. Так, для Internet Explorer необходимо явно указать элемент ActiveX, выполняющий роль Java-плагина посредством его уникального идентификатора:

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93">
<param name="code" value="файл_апплета" />
...
</object>
```

Синтаксис для остальных браузеров:

```
<object classid="java:файл_апплета">
...
</object>
```

Параметры апплета передаются с помощью тегов <param>, как и в случае применения тега <applet>. За последним тегом <param> обычно вставляют альтернативный текст и ссылку на сайт, с которого можно загрузить Java-плагин, если он не установлен на компьютере пользователя.

Чтобы обеспечить межбраузерную совместимость кода, достаточно в тег <object> для Internet Explorer вложить тег <object> для других браузеров. Однако при этом придется продублировать набор тегов <param>. В листинге 12.17 приведен код вставки по такой схеме апплета, рассмотренного в *разд. 12.5.2*.

Листинг 12.17. Вариант вставки апплета с помощью тега <object>

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Апплет</title></head>
<body>
<object classid="java:ABCMenuMan.class" width="150" height="95">
  <param name="copyright" value="http://www.wyka-warzecha.com"/>
  <param name="timingValue" value="75"/>
  <param name="noItems" value="4"/>
  <param name="aniMenu" value="menu.gif"/>

  <param name="title1" value="_Яндекс">
  <param name="title2" value="_Rambler">
  <param name="title3" value="_Сам себе Web-дизайнер">
  <param name="title4" value="_Википедия">

  <param name="link1" value="http://www.yandex.ru">
  <param name="link2" value="http://www.rambler.ru">
  <param name="link3" value="http://dunaevv1.narod.ru">
  <param name="link4" value="http://ru.wikipedia.org/wiki/">

  <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        width="150" height="95" >
    <param name="code" value="ABCMenuMan.class" />
    <param name="copyright" value="http://www.wyka-warzecha.com"/>
    <param name="timingValue" value="75"/>
```

```
    <param name="noItems" value="4"/>
    <param name="aniMenu" value="menu.gif"/>

    <param name="title1" value="_Яндекс">
    <param name="title2" value="_Rambler">
    <param name="title3" value="_Сам себе Web-дизайнер">
    <param name="title4" value="_Википедия">

    <param name="link1" value="http://www.yandex.ru">
    <param name="link2" value="http://www.rambler.ru">
    <param name="link3" value="http://dunaevv1.narod.ru">
    <param name="link4" value="http://ru.wikipedia.org/wiki/">
  </object>
Ваш браузер не поддерживает Java.
<a href="http://www.java.com/ru">
Скачайте последнюю версию Java-плагина
</a>
</object>
</body>
</html>
```

ГЛАВА 13

Что такое JavaScript

13.1. Из истории

Современное состояние чего-либо, и в частности языков и технологий программирования, легче понять в исторической ретроспективе. Сначала HTML-документы содержали лишь команды (теги) форматирования текста, ссылки на другие документы и вставки графических изображений. Нередко мы поначалу ограничиваем себя слишком скромными целями, то ли от неуверенности в собственных силах, то ли от непонимания перспективы. При этом мы говорим: хорошо бы реализовать хотя бы это немногое, а там посмотрим. И в самом деле, на первых порах этого немногого оказалось вполне достаточно для публикации пользователями Всемирной сети своих творений, содержащих самое главное — тексты с иллюстрациями и ссылки на другие документы того же рода. Таким образом, реализовалась возможность практически неограниченного распространения чего бы то ни было.

Вспомните недавнее прошлое, когда была лишь одна возможность публикации, — напечатать сначала на механической пишущей машинке, а затем попытаться разместить свой текст на бумажном носителе, выпускаемом каким-либо официальным издательством. На издание небольшой статьи уходило несколько месяцев. Однако, как почти всегда и бывает, освоенное не только порождает удовлетворенность или, напротив, разочарование достигнутым состоянием, но и вызывает новые безудержные желания (вспомните сказку о рыбаке и золотой рыбке). HTML был прост и доступен даже тем, кто не имел никакого опыта программирования и, тем более, создания программных проектов, т. е. программ, предназначенных для более или менее широкой потребительской аудитории. Простота и доступность HTML сыграли свою главную роль — обеспечить доступ к всемирным информационным ресурсам всех желающих. Лиха беда начало!

Вскоре выяснилось, что добавление к уже существующему еще чего-то простого и в то же время мощного может раскрыть ворота в мир почти неограниченного могущества. Где же была предполагаемая точка атаки и последующей экспансии? Первоначальные HTML-документы содержали лишь команды представления на экране некоторых информационных ресурсов первостепенной важности: текстов, графики, а затем аудио и видео. Тогда можно было лишь надлежащим образом разместить информацию на страницах, но управлять ею или обрабатывать динамически (т. е. во время экспозиции) было невозможно. Решить эти и многие другие задачи позволили языки программирова-

ния, отличные от HTML и похожие на уже известные, посредством которых пишут программы для обычных компьютерных приложений. Таким образом, необходимо было обеспечить вставку в HTML-код фрагментов на другом языке программирования.

В 1995 г. компания Netscape впервые предложила язык LiveScript для своего браузера Netscape Navigator 2.0. Главное назначение этого языка было весьма скромным — проверка и обработка данных, введенных пользователем в форму на Web-странице, чтобы исключить передачу на сервер заведомо неправильных или пустых значений. Вскоре он был переименован: префикс Live был заменен на Java. Видимо, это произошло по коммерческим соображениям. По крайней мере, JavaScript не является какой бы то ни было упрощенной версией известного мощного, но совсем другого языка Java. Итак, Netscape ввела в обиход язык программирования для своего браузера. Однако в это время уже существовали и развивались браузеры других конкурирующих фирм. Естественно, они не могли остаться в стороне.

Вскоре за Netscape корпорация Microsoft представила свою редакцию JavaScript под немного отличающимся названием JScript для своего браузера Internet Explorer 3.0. Название JScript было выбрано, видимо, чтобы избежать проблем с авторскими правами и лицензированием. Тем не менее идея родилась и воплотилась. Так началось развитие скриптового языка (т. е. языка сценариев), который теперь называют JavaScript, а название JScript употребляют, когда хотят подчеркнуть, что имеется в виду язык JavaScript с множеством фирменных расширений, поддерживаемых преимущественно браузерами Microsoft Internet Explorer. Microsoft немало потрудилась, чтобы обеспечить программисту как можно больше возможностей и удобств. Поэтому браузеры других фирм были вынуждены поддерживать в порядке исключения кое-что из созданного Microsoft. И в самом деле, нельзя же полностью игнорировать массу программного кода, написанного исключительно для широко распространенного браузера Microsoft Internet Explorer. Заметим попутно, что браузер Opera стал поддерживать JavaScript, начиная с версии 3.0.

Со временем все основные производители браузеров создали собственные расширения языка JavaScript, позволяющие решать существенно более сложные и интересные задачи, чем первоначально поставленные. Этапы развития JavaScript фиксировались в виде его версий, которые были тесно связаны с типами и версиями браузеров. Действительно, любой язык жив, пока существует система, способная его интерпретировать. Такой системой для языков программирования является компилятор или интерпретатор. Сейчас мы не будем выяснять, в чем разница между ними. Достаточно лишь указать, что скриптовые языки воспринимаются интерпретаторами, а интерпретатор JavaScript встроен в Web-браузер, который в принципе может выполнять скрипты (сценарии). Современные браузеры это могут делать, но пока с некоторыми различиями.

Версии и редакции (клоны) языка JavaScript тесно связаны с поддерживающими их браузерами различных фирм. Браузеры содержали в себе встроенные интерпретаторы этого языка, а их разработчики могли видоизменять эти языки и расширять их как угодно. Однако произвол такого творчества, разумеется, ограничен. Всемирная паутина по своему замыслу должна предоставлять пользователю информационные ресурсы инвариантно относительно браузеров. Поэтому творцам языка приходилось так или иначе стремиться к некоторому "общему знаменателю". Тем не менее разночтение все же возникло и породило весьма запутанную проблему совместимости программных кодов (сценариев), написанных для различных браузеров.

Разумеется, синтаксис и основные объекты языка остаются незыблемыми, поддерживаемыми всеми современными браузерами, которые в принципе способны выполнять сценарии для Web-страниц. Однако в языке предусмотрены еще и объекты браузера и документа, состав и/или реализация которых варьируют от версии к версии, а также зависят от типа браузера. Именно эти объекты в основном и являются причиной несовместимости браузеров различных производителей.

Сначала Netscape и Microsoft использовали одинаковые стратегии развития своих браузеров: в конкурентной борьбе они создавали свои фирменные расширения первоначальной версии языка JavaScript, которые, к сожалению, не были совместимыми. На этом пути были созданы браузеры Netscape Navigator 4.*x* и Microsoft Internet Explorer 4.*x*. Между JavaScript от Netscape и JScript от Microsoft было много общего, но имелись и существенные различия, хотя и Netscape, и Microsoft вместе с другими крупнейшими производителями программного обеспечения участвовали в работе организации W3C, цель которой — выработка рекомендаций и стандартов для Web. Казалось бы, они вполне могли бы договориться об "общем знаменателе", но этого не произошло.

В дальнейшем стратегии Netscape и Microsoft существенно разошлись. Компания Netscape в своих браузерах версий 6.*x* и 7.*x* отказалась от создания, поддержки в полной мере уже существующих и наращивания фирменных дополнительных возможностей JavaScript в пользу приверженности стандартам, при этом она также отказалась от принципа сохранения обратной совместимости своих новых продуктов. Иначе говоря, новые браузеры Netscape не поддерживали в полной мере сценарии, написанные для предыдущих версий. И это в условиях, когда уже существовало огромное количество кода, написанного для прежних версий браузеров Netscape! В результате, что нетрудно было предвидеть, линейка браузеров Netscape Navigator 4.*x* стала терять популярность среди пользователей и в настоящее время почти исчезла. Вместе с тем, в 1998 г. Netscape представила исходный код браузера Mozilla. Это был важный этап реализации новой стратегии, часть которой воплотилась в браузерах Netscape 6*x*, 7*x*. Название "Mozilla" сначала использовалось в качестве внутреннего имени программного кода браузеров компании Netscape. В настоящее время оно относится к браузеру как продукту, платформе и организации. Mozilla — это одновременно наименование и кроссплатформенного браузера с открытым кодом, и платформы — каркаса, на базе которого разработаны как сам браузер, так и другие кроссплатформенные приложения, и организации — независимой группы разработчиков, занимающейся поддержкой и усовершенствованием данных программных продуктов. Открытый программный код Mozilla теперь может быть использован в качестве основы (с соблюдением определенных лицензионных соглашений) для создания любых собственных браузеров теми, кто возьмется за это дело. Так, например, Mozilla, Mozilla SeaMonkey и Mozilla Firefox — конкретные браузеры, созданные на базе одного и того же программного кода Mozilla, но являющиеся разными продуктами, поскольку имеют некоторые отличия.

Microsoft, напротив, продолжила создавать расширения языка, дающие опытному разработчику большие возможности, а неопытному — автоматическое исправление синтаксических ошибок и отклонений от соблюдения стандартов. Так что для Internet Explorer можно позволить себе некоторую небрежность в написании программных кодов: браузер многое исправит сам. При этом, что очень важно, Microsoft сохранила для своих браузеров Internet Explorer обратную совместимость и в то же время обеспечила поддержку современных стандартов, хотя и не на 100%. Если вы хотите создавать Web-

документы, инвариантные относительно всех современных браузеров, то пишите тексты программ с соблюдением стандартов, не прибегая к дополнительным фирменным возможностям браузеров. А при разработке приложений исключительно для пользователей Internet Explorer (например, для интрасети) ничто не мешает вам применить его фирменные расширения языка JScript.

Теперь немного о собственно стандартах. Рано или поздно, во избежание кризиса "вавилонского столпотворения", причиной которого было, как хорошо известно, разноязычие строителей, приходится задуматься о соответствии пестрого разнообразия языков каким-нибудь общим правилам.

Чтобы создать основу для обеспечения межбраузерной совместимости в отношении языка JavaScript, была выработана стандартная спецификация ECMAScript (ECMA-262) для языка сценариев, встраиваемых в Web-страницы. В настоящее время известны три редакции (edition) данного стандарта. Все современные браузеры теперь объявляют полную поддержку стандарта ECMA-262 Edition 3, но некоторые из них, как уже отмечалось ранее, имеют и дополнительные возможности (расширения). Больше всех фирменных (нестандартных) возможностей предоставляет, пожалуй, браузер Microsoft Internet Explorer, но и другие делают что-то в этом направлении. Так что разработчикам Web-приложений следует быть внимательным и готовым к тестированию своей продукции на нескольких наиболее популярных браузерах, таких как Mozilla Firefox, Opera, Google Chrome, Microsoft Internet Explorer (версии 8 для Windows XP/Vista и версии 9+ для Windows 7+) и Apple Safari.

13.2. Общая характеристика языка

В данном разделе приводятся наиболее общие сведения о JavaScript, которые могут оказаться полезными для тех, кто уже имеет опыт работы с каким-нибудь другим языком.

Отличительные особенности JavaScript:

- исходный программный код интерпретируется, а не компилируется;
- операторы разделяются точкой с запятой. Вместе с тем, за завершенным (т. е. полным) оператором, расположенным в одной строке, разделитель можно не указывать: в этом случае переход на другую строку работает так же, как и точка с запятой. За последним оператором в блоке кода (например, внутри фигурных скобок {}) точку с запятой можно не указывать;
- язык регистрозависимый. Новички об этом нередко забывают, возможно, потому, что HTML, с которым JavaScript обычно используется совместно, не зависит от регистра (имена тегов и атрибутов HTML можно писать как строчными, так и прописными буквами);
- внутристрочный комментарий (неинтерпретируемый, игнорируемый текст) выделяется символом //, а многострочный — парой символов /* и */;
- к данным применяется слабый (динамический) контроль типов. В операторах с разнотипными данными последние автоматически приводятся к требуемому типу. Типы данных могут быть примитивными и составными. Примитивные типы содержат простые однородные значения, такие данные можно передавать функциям в качест-

ве параметров по значению, а не по ссылке. Составные типы содержат разнородные данные (в том числе и составные), их передают функциям только по ссылке;

- язык объектно-ориентированный, однако, он основан на прототипах, а не на классах. Имеются четыре типа объектов: встроенные объекты, объекты браузера (интерпретирующей системы), объекты документа и объекты пользователя (программиста). Синтаксис доступа к объектам точечный, а не стрелочный. Например, для получения значения свойства `property` объекта `anyobject`, который является подобъектом объекта `mainobject`, следует использовать выражение `mainobject.anyobject.property`; для вызова метода `anymethod()` этого же объекта применяется выражение `mainobject.anyobject.anymethod()`. Объект `mainobject` может быть, в свою очередь, подобъектом объекта более высокого уровня. Тогда последний нужно указать в качестве префикса в цепочке объектов, находящихся на пути к интересующему нас свойству или методу;
- ввод/вывод в основном ограничен взаимодействием с документами и пользователями. По умолчанию предполагается, что доступ к локальной файловой системе запрещен. Однако браузеры могут предоставлять специальные объекты, с помощью которых обеспечивается работа с файловой системой пользователя, хотя и с выдачей предупреждений об опасности выполнения файловых операций.

13.3. Вставка сценариев в (X)HTML-документ

Сценарии (программы, скрипты) на языке JavaScript взаимодействуют с объектами (X)HTML-документа (Web-страницы). Для этого они собственно и создаются. Чтобы реализовать такое взаимодействие, их необходимо, прежде всего, внедрить (вставить) в текущий (X)HTML-документ.

Сценарии можно вставить в (X)HTML-документ несколькими способами, но чаще всего их размещают внутри контейнерного тега `<script>`, т. е. между дескрипторами `<script>` и `</script>`. Сам контейнер `<script>` находится в (X)HTML-документе. Программный код сценария можно писать непосредственно в (X)HTML-документе, а также в отдельных текстовых файлах, которые вызываются из него. Проще всего размещать сценарий непосредственно в (X)HTML-документе, именно так обычно и поступают. Мы рассмотрим оба варианта.

Встречая тег `<script>` в загруженном (X)HTML-документе, браузер "понимает", что за ним начинается код сценария. Заключительный тег `</script>` указывает браузеру, что код сценария закончился. Все, что находится вне этих тегов, браузер воспринимает как (X)HTML-код. Контейнер `<script>` может располагаться в любом месте (X)HTML-документа, и даже не один раз. От его расположения в теле (X)HTML-документа иногда может зависеть функционирование всей Web-страницы, но об этом будет сказано далее.

Атрибуты тега `<script>`:

- `language` — указывает язык сценария; возможные значения:
  - `"JavaScript"`, `"JScript"`. Однако второе значение правильно воспринимается только браузером Internet Explorer. Для сценариев, рассчитанных на различные браузеры, следует указывать `language = "JavaScript"` *или* `"javascript"`;

- `"VBScript"`, `"VBS"`. Эти значения воспринимаются только браузером Internet Explorer и применяются в случае сценариев на языке Visual Basic Script.

Примечание

Если атрибут `language` не указан, то Internet Explorer подразумевает `JScript`. Вместе с тем, любая опечатка в значении атрибута `language` может привести к игнорированию браузером всего содержимого тега `<script>`.

Атрибут `language` не соответствует стандарту W3C, хотя он широко используется на практике. Его рекомендуется заменять атрибутом `type` (*см. далее*).

- `type` — указывает MIME-тип используемого языка. Для JavaScript следует задавать значение `"text/javascript"`.

Примечание

В теге `<script>` возможно указание одновременно и атрибута `language`, и `type`. Однако настоятельно рекомендуется указывать атрибут `type`.

- `src` — указывает имя или URL-адрес текстового файла, содержащего код сценария. Этот атрибут необходим в том случае, если сценарий расположен не непосредственно в (X)HTML-документе, а в отдельном файле. Расширение файла со сценарием может быть каким угодно, но обычно используют js:

```
<script type = "text/javascript" src = "myscripts.js"></script>.
```

Примечание

Если сценарий располагается в отдельном файле, то в нем, разумеется, теги `<script>` и `</script>` не пишут.

Сценарий, загруженный из внешнего файла, можно представить себе просто как его вставку в (X)HTML-документ.

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги `<script>` и `</script>`, а все, что находится между ними, интерпретируют как текстовое содержимое (X)HTML-документа. Результат при этом может быть самым неожиданным. Чтобы уменьшить вероятность отображения кода сценария в окне старого браузера, следует заключить его в дескрипторы комментария `<!--` и `-->`. Новые браузеры, поддерживающие сценарии, будут опускать эти символы, выполняя код сценария, а старые браузеры (не понимающие сценарии), наоборот, проигнорируют код сценария. Пример такого подхода приведен в листинге 13.1.

Листинг 13.1. Пример обеспечения совместимости при вставке сценария

```
<script type = "text/javascript">
 <!--
  // код сценария
 //-->
</script>
```

Обратите внимание на заключительные символы тега комментария, перед которыми стоят две косые черты. При их отсутствии браузер будет пытаться интерпретировать символы `-->` как окончание комментария (X)HTML, а при наличии он просто их проигнорирует.

В браузерах, потенциально поддерживающих сценарии, выполнение последних можно отключить. Кроме того, существуют браузеры, принципиально не поддерживающие сценарии. В данной ситуации желательно хотя бы вывести сообщение о том, что произошло. С этой целью в (X)HTML-документе используют контейнерный тег <noscript>, в котором размещают, например, текст предупреждающего сообщения или ссылку на бесскриптовую версию страницы. Все браузеры, поддерживающие сценарии, проигнорируют содержимое тегов <noscript>, кроме тех случаев, когда поддержка сценариев отключена. Браузеры, принципиально не поддерживающие сценарии, напротив, содержимое тега <script> (если оно заключено в теги <!-- и //-->) опустят, а тега <noscript> — отобразят. Пример приведен в листинге 13.2.

Листинг 13.2. Применение тега <noscript>

```
<html>
<head>
<title>Пример применения тега noscript</title>
</head>
<script type="text/javascript">
<!--
   alert("Поддержка javascript включена");
//-->
</script>
<noscript>
   <h2>Ваш браузер не поддерживает javascript
   или его поддержка отключена</h2>
</noscript>
</html>
```

Здесь был использован встроенный метод alert(*сообщение*) для вывода сообщений в диалоговом окне.

> ***Внимание!***
>
> В примерах, приводимых в настоящей книге, мы не будем указывать символы комментария и теги <noscript> для адаптации к старым браузерам из соображений экономии времени и места.

В отдельных файлах обычно размещают библиотеки функций (определения функций), а также сценарии, использующиеся в нескольких (X)HTML-документах одного или нескольких сайтов. Пример сценария из нескольких разделов иллюстрирует листинг 13.3.

Листинг 13.3. Пример сценария, состоящего из нескольких разделов

```
<html>
<head>
<title>Пример</title>
</head>
 <script type="text/javascript">
```

```
    function myfunc(){
     ...
    }
 </script>
 <script type="text/javascript" src = "prog/myprog1.js"></script>
 <script type="text/javascript" src = "myprog2.js"></script>
...
</html>
```

Здесь в (X)HTML-документе расположены три раздела (секции) сценария, два из которых загружаются из отдельных файлов. В первом разделе сценария определена некоторая функция.

Сценарий можно также писать в виде строки операторов, разделенных точкой с запятой. Такая строка, взятая в кавычки, служит в качестве значения атрибута-события:

```
<img src="mypicture.gif" onclick = "alert('Привет!')"/>.
```

Вызовы функций и их определения могут следовать в произвольном порядке, только если они расположены в одном и том же контейнере `<script>`. Если вы размещаете их в разных контейнерах `<script>`, то необходимо, чтобы определение функции предшествовало ее вызову. Аналогично, если определения функций находятся в отдельном файле, его необходимо загрузить в (X)HTML-документ раньше вызовов этих функций. В листинге 13.4 приведены примеры правильного и неправильного расположения сценариев в (X)HTML-документе.

Листинг 13.4. Расположение сценария в HTML-документе

Правильно

```
<html>
 ...
 <script type="text/javascript">
  function myfunc(){
   ...
  }
 </script>
 ...
 <script type="text/javascript">
  ...
  myfunc()
  ...
 </script>
 ...
</html>
```

Неправильно

```
<html>
 ...
 <script type="text/javascript">
  ...
  myfunc()
  ...
 </script>
 ...
 <script type="text/javascript">
  function myfunc(){
   ...
  }
 </script>
 ...
</html>
```

При попытке загрузить и выполнить неправильный вариант (X)HTML-кода появится диалоговое окно с сообщением "Ошибка: Предполагается наличие объекта". Браузер интерпретирует HTML-теги последовательно. Так, встретив выражение вызова функции `myfunc()` в одном контейнере `<script>`, браузер еще не имеет в памяти определения этого объекта (функции), расположенного в другом контейнере `<script>`. Если же опре-

деление функции и ее вызов находятся в одном и том же контейнере <script>, то его содержимое сначала загружается в память, а затем анализируется. Когда интерпретатор встречает вызов функции, то он ищет ее определение в памяти и в случае удачи выполняет код этой функции. Следующие два варианта, приведенные в листинге 13.5, правильные, потому что определение функции и ее вызов находятся в одной секции сценария (в одном контейнере <script>).

Листинг 13.5. Варианты расположения сценария

Вариант 1

```
<html>
 ...
 <script type="text/javascript">
  ...
  function myfunc(){
   ...
  }
  ...
  myfunc()
  ...
 </script>
 ...
</html>
```

Вариант 2

```
<html>
 ...
 <script type="text/javascript">
  ...
  myfunc()
  ...
  function myfunc(){
   ...
  }
  ...
 </script>
 ...
</html>
```

Если необходимо, чтобы сценарий оказался в браузере прежде, чем загрузятся элементы (X)HTML-документа, то его следует расположить в верхней части (X)HTML-кода, а еще лучше в контейнере <head> (в заголовке документа).

Если требуется, чтобы сценарий начал выполняться после загрузки всех элементов (X)HTML-документа, то возможны следующие два варианта.

1. Расположить сценарий в конце (X)HTML-документа.
2. Указать атрибут-событие onload в контейнерном теге <body>, который задает основную часть (X)HTML-документа. В данном случае значением атрибута onload обычно является строка, содержащая имя функции. Определение этой функции, как правило, содержится в контейнере <script>, размещенном в контейнере заголовка (X)HTML-документа <head>. Обратите внимание, что при использовании атрибута-события onload в теге <body> обработчик этого события выполняется по завершении загрузки всех элементов, определенных в контейнере <body>, а не в начале или процессе их загрузки. Пример такого подхода иллюстрирует листинг 13.6. Другой вариант организации выполнения скрипта после загрузки (X)HTML-документа представлен в листинге 13.7.

Листинг 13.6. Обработка сценария после загрузки (X)HTML-кода (вариант 1)

```
<html>
 <head>
  <script type="text/javascript">
```

```
    function myfunc(){
    ...
    }
  </script>
 </head>
 ...
 <body onload ="myfunc()">
  <!-- теги тела документа -->
 </body>
 ...
</html>
```

Листинг 13.7. Обработка сценария после загрузки (X)HTML-кода (вариант 2)

```
<html>
 <head>
  <script type="text/javascript">
    window.onload=function(){
    /* здесь код сценария */...
    }
  </script>
 </head>
 ...
</html>
```

Иногда требуется выполнить небольшой скрипт при щелчке на ссылке. В таком случае строку выражений скрипта, разделенных точкой с запятой, можно указать в качестве значения атрибута `href` тега `<a>`. При этом необходим префикс `javascript:`, как в следующем примере:

```
<a href="javascript:var x=5;alert(x+10)">Щелкни здесь</a>
```

При инициализации данной ссылки будет выведено диалоговое окно с числом 15.

13.4. Специальные термины и понятия

У программистов имеется профессиональный жаргон, с помощью которого они могут обмениваться мыслями более лаконично.

Вот некоторые термины.

- *Регистровая зависимость* — зависимость выражений языка или их частей (например, имен переменных, функций и т. д.) от того, строчными или прописными буквами они написаны. Так, в регистрозависимых языках переменные `myvar`, `MYVAR` и `myVar` различны, а в регистронезависимых это одна и та же переменная. Например, язык JavaScript полностью регистрозависимый, а PHP — регистрозависимый, но не в полной мере.
- *Идентификатор* (identifier) — имя переменной, функции, объекта и др.; идентификатор не должен быть ключевым или зарезервированным словом (*см. далее*). Про-

граммист обычно выбирает идентификаторы по своему усмотрению, но в соответствии с определенными правилами. В большинстве случаев идентификаторы состоят из печатных символов (букв, цифр и символа подчеркивания), причем идентификатор не должен начинаться с цифры и содержать символы пробела. Примеры: `x`, `myvar`, `userName`, `user_name`, `var12`.

- *Ключевое слово* (keyword) — слово, являющееся частью языка. Ключевые слова недопустимы в качестве идентификаторов. Примеры: `function`, `if`, `while`, `var`.
- *Зарезервированное слово* (reserved word) — слово, которое не рекомендуется использовать в качестве идентификатора. Зарезервированные слова могут оказаться ключевыми в более поздних версиях языка.
- *Оператор* (statement) — команда, предписывающая компьютеру выполнить некоторое действие; операторы обычно приводят к изменению состояния окружения (например, переменной или определения). Примеры: `x = x + 5`; `function myfunc(x,y) {return x*y}`.
- *Литерал* (literal) — значение, содержащееся непосредственно в тексте программы. Примеры: `5`, `"Привет всем!"`, `[3,7,9,25]`, `false`.
- *Лексема* (token) — наименьшая и неделимая единица языка. Примеры: все идентификаторы, ключевые слова, а также литералы типа `5.2` и `"Привет всем!"`.
- *Знак операции* (operator) — лексемы, представляющие собой встроенные операторы языка. Примеры: `=`, `+`, `-`, `*`, `/` (оператор присваивания и арифметические операторы). Операции нередко называют операторами.
- *Выражение* (expression) — группа лексем (обычно литералов и идентификаторов) в сочетании со знаками операций, для которых можно вычислить значение. Примеры: `3.141`; `"Привет!"`; `(2 + x)*5`; `myfunc()`, `x = y + myfunc()`.

В следующих главах будут использоваться понятия, присущие большинству современных языков программирования: переменные, типы данных, массивы, функции, классы и объекты, операторы и выражения. Вот их краткие определения:

- *переменные* — это контейнеры для хранения данных, называемых значениями переменных; переменная имеет имя или идентификатор, чтобы отличить ее от других переменных и обращаться к ней по имени для получения содержащегося в ней значения;
- *значения* — могут быть различных типов: числового, символьного (строкового), логического и др. Тип — множество допустимых значений — может быть задан явным перечислением элементов либо посредством формулировки некоторого ограничения или правила. Например, логический тип данных есть множество из двух элементов (true и false), интерпретируемых как "истина" и "ложь" соответственно. Множество чисел бесконечно и не может быть задано перечислением своих элементов. Поэтому числовой тип определяют неким правилом формирования относящихся к нему элементов: последовательность цифр, перед которой может быть указан знак, а разделителем целой и дробной частей служит точка;
- *переменная* — принимает значение или, другими словами, в переменную записывается значение с помощью оператора присваивания. Доступ к значению переменной осуществляют по ее имени (идентификатору);

- *массив* — упорядоченный набор (последовательность данных). Можно сказать, что конкретный массив это значение составного типа, который также называется массивом;
- *функция* — это поименованный блок программного кода (тело функции), который можно вызвать для выполнения из основной (внешней по отношению к коду функции) программы;
- *объект (класс)* — описание объекта в виде набора его свойств, которые аналогичны обычным переменным и функциям. Часто свойства-переменные называют просто свойствами, а свойства-функции — методами;
- *операторы* — предназначены для задания команд, которые должен выполнить компьютер. Выражения содержат один или несколько операторов. Наиболее часто встречается оператор присваивания значения переменной. Для управления ходом вычислений служат условные операторы и циклы. Если бы не эти специальные операторы, программа выполнялась бы только в одном направлении — сверху вниз, в порядке упоминания выражений в листинге (исходном тексте).

Освоение языка совершенно не эффективно без практического программирования, как невозможно научиться говорить на иностранном языке вне языковой среды. Поэтому настоятельно рекомендуется самостоятельно выполнить предлагаемые в книге примеры.

ГЛАВА 14

Объектная модель браузера и документа

Браузер и загруженный в него (X)HTML-документ программно представляются в виде совокупности объектов, свойства и методы которых доступны из сценариев на языке JavaScript. Мы можем прочитать значения свойств, а в большинстве случаев и изменить их без перезагрузки документа. Чтобы сделать это, необходимо иметь сведения об объектной модели, т. е. о том, какие объекты представляют браузер и загруженный в него документ, какими свойствами и методами они обладают. Кроме того, необходимо знать, как получить к ним доступ. В данной главе я лишь намечу пути, как добраться до необходимых сведений.

14.1. Общие сведения

Браузер представлен в объектной модели объектом `window`, который содержит в себе еще несколько дочерних объектов, в том числе `document` (рис. 14.1). Объектная модель браузера (т. е. объект `window`) не стандартизирована консорциумом W3C и находится "на откупе" производителей браузеров. Тем не менее имеется некий общий набор ее объектов, который поддерживается всеми основными браузерами примерно одинаково. На рис. 14.1 показаны несколько таких общезначимых подобъектов объекта `window`.

Объект `window` находится на самом верхнем уровне иерархии, он имеет свои собственные свойства и методы, а также содержит несколько объектов, важнейшим из которых является `document`. Последний, в отличие от других, стандартизирован W3C.

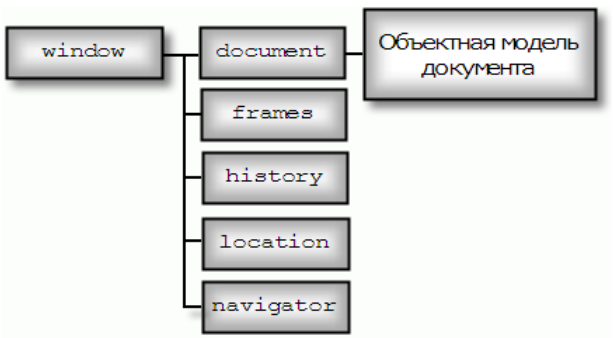


Рис. 14.1. Основные объекты браузера

Объект `frames` представляет коллекцию (массив) фреймов. Каждый фрейм является подокном главного окна браузера и как объект устроен аналогично `window`. В частности, он может содержать объект `document`, соответствующий загруженному в него документу.

Объект `history` содержит список последних URL-адресов, к которым обращался пользователь. Данный объект представляет собой то, что называют журналом браузера.

Объект `location` содержит сведения об URL-адресе загруженного в браузер документа.

В объекте `navigator` находится информация об основных характеристиках браузера, в том числе о его типе и версии.

Объект `document` содержит информацию об элементах загруженного в него документа, таких как ссылки, графические изображения, фрагменты текста и др. Для каждого элемента в загруженном (X)HTML-документе браузер автоматически создает соответствующий объект. Кроме них создаются и другие объекты. Общая схема объекта `document` сложилась к 1996 г. и была воплощена в первых версиях браузеров Internet Explorer и Netscape. На рис. 14.2 показаны основные (но далеко не все) объекты, составляющие объектную модель документа. Объекты, около имен которых указана пара квадратных скобок, называют еще коллекциями — массивами, содержащими другие объекты. Например, `forms[]` — коллекция всех объектов, представляющих формы в текущем (загруженном) документе, а `links[]` — коллекция ссылок. Далее разработчики Microsoft и Netscape усовершенствовали объектную модель, добавив новые возможности и реали-

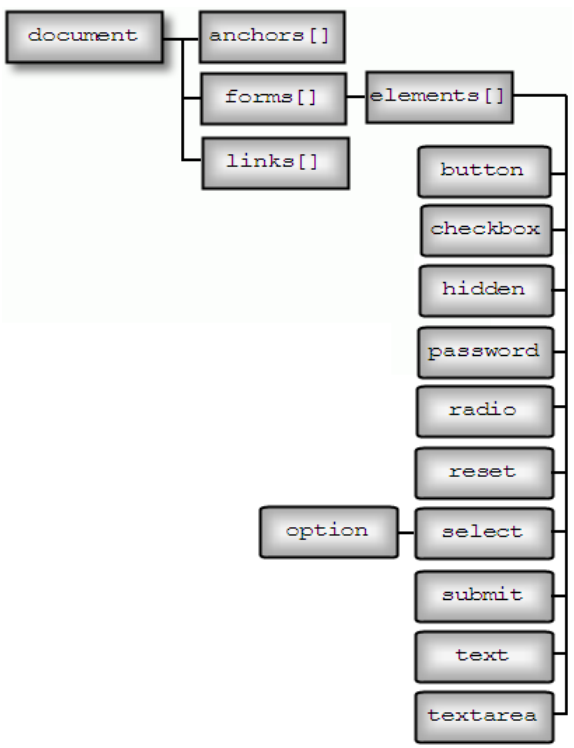


Рис. 14.2. Основные объекты документа

зовав их в браузерах Internet Explorer 4+ и Netscape 4*x*. При этом модели, реализованные в разных браузерах, несколько отличались друг от друга, что порождало проблему межбраузерной несовместимости скриптов.

Чтобы преодолеть несовместимость браузеров, консорциум W3C предложил стандарт, связывающий (X)HTML- и XML-документы с иерархией объектов, доступной программисту посредством скриптов. Именно этот стандарт получил название модель DOM (Document Object Model — объектная модель документа, см. **www.w3.org/DOM/**). Когда говорят об объектах, свойствах и методах DOM, имеют в виду соответствие последних стандарту DOM от W3C. Модель DOM обеспечивает программный интерфейс приложения (API — Application Programming Interface), обеспечивающий скриптам доступ к любому элементу Web-страницы, включая теги, атрибуты и стилевые параметры.

Консорциум W3C, учитывая уже достигнутое Netscape и Microsoft, и в то же время желая обеспечить межбраузерную совместимость, определил три уровня DOM:

1. DOM 0 — данный уровень практически полностью соответствует моделям, реализованным в Netscape 3.0 и Internet Explorer 3.0. Это классическая или традиционная объектная модель JavaScript. Все современные браузеры поддерживают данный уровень DOM.
2. DOM 1 — обеспечивает работу со всеми элементами документа посредством стандартного набора функций. Здесь все части страницы доступны для чтения и изменения. Современные браузеры поддерживают данный уровень DOM.
3. DOM 2 — обеспечивает более совершенные возможности доступа к элементам документа, объединяя модели нулевого и первого уровней и добавляя доступ к таблицам стилей. Данная форма DOM поддерживает улучшенную модель событий и некоторые другие расширения. Все современные браузеры объявляют поддержку DOM 2, но де-факто осуществляют ее не полностью, в разной степени.

С другой точки зрения DOM разделяют на пять частей:

1. DOM Core (ядро DOM) — определяет типовую модель в виде дерева для просмотра и изменения документа с элементами разметки.
2. DOM HTML — определяет расширение ядра DOM для работы с HTML.
3. DOM CSS — определяет интерфейсы для программного управления правилами CSS.
4. DOM Events (события DOM) — добавляет в DOM средства обработки событий.
5. DOM XML — расширение ядра DOM для работы с XML-документами для обеспечения работы с разделами CDATA, инструкциями, пространством имен и др.

Все современные браузеры поддерживают стандарт DOM от W3C того или иного уровня, но пока не в полной мере. Однако основная тенденция развития браузеров состоит в том, чтобы следовать этому стандарту в возможно большей степени. Вместе с тем, браузеры продолжают поддерживать и некоторые нестандартные фирменные расширения объектной модели. Последнее в наибольшей степени относится к Microsoft Internet Explorer. Многие интересные идеи Microsoft воплощала в своем ранее почти беспредельно господствовавшем браузере, надеясь на их укоренение либо в будущих стандартах, либо де-факто. Но в жизни все оказалось несколько иначе: Internet Explorer теперь уступает позиции на рынке, появляются стандарты, не утверждающие де-юре достижения Microsoft, укрепляют свои позиции новые браузеры, которые следуют стандартам.

Тем не менее, поскольку в настоящее время существует очень много сценариев, корректно выполняющихся в Internet Explorer, другие браузеры также сохранили поддержку некоторых нестандартных элементов объектной модели.

Загружая (X)HTML-документ, браузер представляет его в форме дерева соответствующих программных объектов, которые доступны посредством клиентских сценариев на JavaScript. На рис. 14.3 показано такое дерево для примера простого документа. Корневой элемент `<html>` содержит элементы `<head>` и `<body>`. Элемент `<head>` содержит `<title>`, который в свою очередь содержит текстовый блок "Пример". Элемент `<body>` включает три элемента: `<h1>`, комментарий и абзац `<p>`. Элемент `<h1>` заголовка первого уровня содержит текстовый блок "Заголовок", а элемент `<p>` абзаца — текстовые блоки "Это абзац" и "для примера", а также элемент `<b>` выделения жирным шрифтом, который в свою очередь включает текстовый блок "текста".

Между узлами дерева существуют отношения родства (родительский элемент — дочерний элемент). Так, в документе на рис. 14.3 родительским элементом для `<p>` явля-

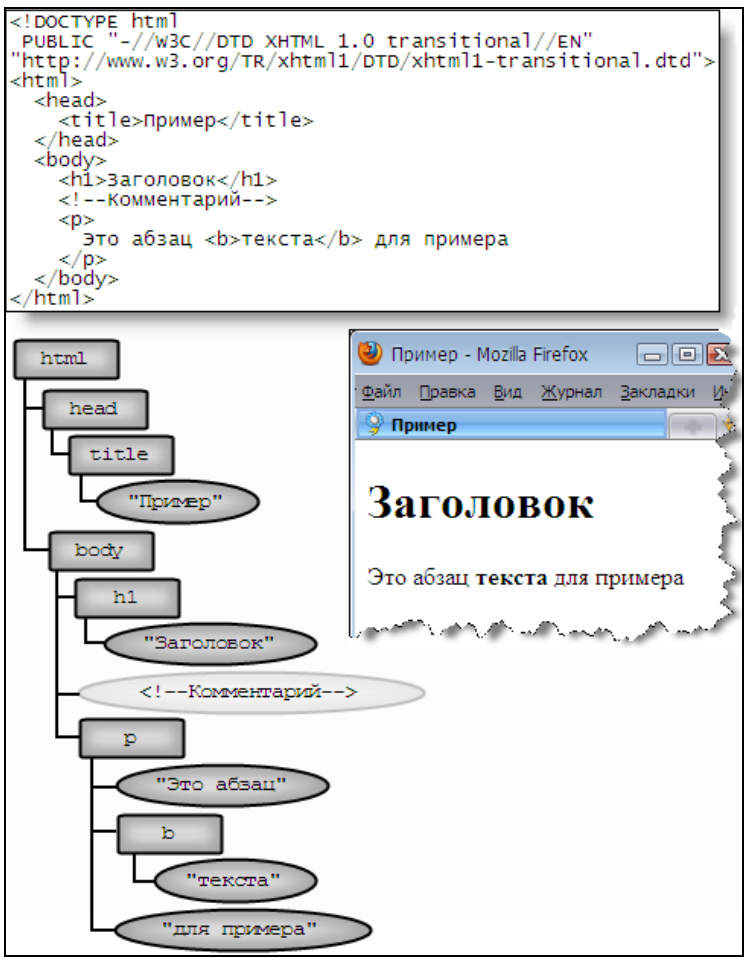


Рис. 14.3. Пример дерева простого документа

ется <body>, а сам <p> дочерний по отношению к <body>. Элемент <body>, кроме <h1> и <p>, имеет еще элемент комментария. Элемент абзаца <p>, в свою очередь, имеет три дочерних элемента: текстовый блок "Это абзац", элемент выделения жирным шрифтом <b> и текстовый блок "для примера". Текстовый блок "текста" дочерний элемент <b>, но не прямой потомок элемента <p>, т. е. текстовый блок внутри тега <b> не есть дочерний элемент абзаца <p>.

14.2. Доступ к объектам

Объекты образуют древовидную структуру, в которой они позиционируются по уровням иерархии так, что одни из них выше других, или, иначе говоря, одни содержат другие в качестве вложенных (подобъектов). Впрочем, вложенные объекты можно рассматривать и как свойства содержащих их объектов. Другими словами, объект может иметь свойство, которое само является объектом. Для доступа к свойствам какого-нибудь объекта необходимо указать всю последовательность объектов, лежащих на одной ветви дерева, ведущей к данному объекту:

объект1.объект2. ... объектN.свойство

Здесь *свойство* является свойством объекта *объектN*, которое может быть как свойством-переменной (собственно свойством), так и свойством-функцией (методом). Для методов после их имени указывается пара круглых скобок, в которых может быть указан список параметров, если он предусмотрен для данного метода.

Объект самого верхнего уровня (корневой объект) — `window` и в силу своего особого положения его можно не указывать в цепочке объектов. Например, объект `location` — дочерний объект `window`. Если мы хотим получить значение свойства `href` объекта `location` (URL-адрес загруженного в браузер документа), то следует написать `window.location.href` или короче: `location.href`. Уже встречавшиеся в данной книге методы `alert()`, `confirm()` и `prompt()` являются методами объекта `window` и при их вызове последний обычно не указывают.

Объект `document` дочерний по отношению к `window`. Поэтому для обращения к свойствам загруженного в браузер (X)HTML-документа префикс `window` можно не указывать, а начинать цепочку сразу с объекта `document`. Например, цвет фона (X)HTML-документа сохраняется в свойстве `bgColor` объекта `document`, а для получения его значения достаточно написать `document.bgColor`.

Для доступа к элементам (X)HTML-документа, заданным тегами, в DOM предусмотрены следующие методы объекта `document`:

- `getElementById(`*значение_id*`)` — возвращает объект элемента по значению его атрибута `id`, указанному в качестве строкового параметра; например, `document.getElementById("myimg")`; данный метод позволяет получить доступ к любому объекту документа независимо от его расположения в иерархии объектной модели;
- `getElementsByTagName(`*список_тегов*`)` — возвращает коллекцию (массив) всех объектов элементов, заданных тегами, имена которых перечислены через запятую в строковом параметре; например, `document.getElementsByTagName("h1, h2, h3")` возвращает коллекцию объектов, соответствующих элементам <h1>, <h2> и <h3>; однако ведущие

браузеры сейчас поддерживают список только из одного имени тега; для получения коллекции всех элементов документа можно указать в качестве параметра *;

- `getElementsByName(`*значение_name*`)` — возвращает коллекцию объектов элементов по значению атрибута `name`, указанному в качестве строкового параметра; например, `document.getElementsByName("myform")`.

> ***Примечание***
>
> При записи названий перечисленных методов часто допускают опечатку. Обратите внимание, что следует писать `get`**`Element`**`ById` (получить элемент по `id`), поскольку данный метод возвращает единственный элемент, и писать `get`**`Elements`**`ByTagName` и `get`**`Elements`**`ByName` (получить элементы по имени тега или имени соответственно), поскольку эти методы возвращают не элемент, а массивы элементов.

Пусть некоторый элемент имеет атрибут `id="myelem"`, тогда для доступа к его объекту можно написать `document.getElementById("myelem")` или более громоздко `document.getElementsByTagName("*")["myelem"]`.

Доступ к объектам можно получить, используя и традиционные коллекции объектной модели, такие как `forms`, `links`, `images` и др. Коллекции, указанные на рис. 14.2, поддерживают все ведущие браузеры. Но есть и другие. Так, начиная с версии 4.0, браузер Internet Explorer поддерживает удобную коллекцию `all`, обеспечивающую доступ к любому элементу документа по значению атрибута `id` или по целочисленному индексу. Индекс указывает номер элемента в порядке его упоминания в документе, а нумерация начинается с нуля. Это нестандартная коллекция, но с ее применением было написано очень много сценариев. Поэтому кроме Internet Explorer ее продолжают поддерживать и другие браузеры, в частности Opera и Safari. Однако Firefox последних версий и Chrome не поддерживают `all`. В связи с этим для обеспечения межбраузерной совместимости данную коллекцию применять не рекомендуется. Впрочем, если вы не хотите переписывать свои старые скрипты с `all`, то перед их вызовом можно применить следующую почти эквивалентную замену:

```
if (!document.all) document.all=document.getElementsByTagName("*");
```

Тогда к элементу с `id="myelem"` можно обратиться как `document.all["myelem"]`.

К форме можно обратиться так же, как и к другим элементам документа, через указанные ранее методы DOM. Вместе с тем, можно использовать и коллекцию `form`, передавая ей в качестве параметра значение атрибута `id` или индекс:

```
document.forms["myform"];
```

Кроме того, можно обращаться к форме и непосредственно по ее имени, т. е. значению атрибута `name`. Например, для обращения к элементу `<form name="myform" ...>` подойдет такая запись:

```
document.myform;
```

Аналогичным образом можно обращаться к элементам форм, таким как `<input>` и `<textarea>`, которые также имеют атрибут `name`. Доступ к элементам формы по значению атрибута `name` — старинный способ, придуманный еще тогда, когда не было атрибута `id`.

На рис. 14.4 показаны два варианта обращения к элементу `<input>`, находящемуся в форме. При этом читается его свойство `value` (значение текстового поля). В первом

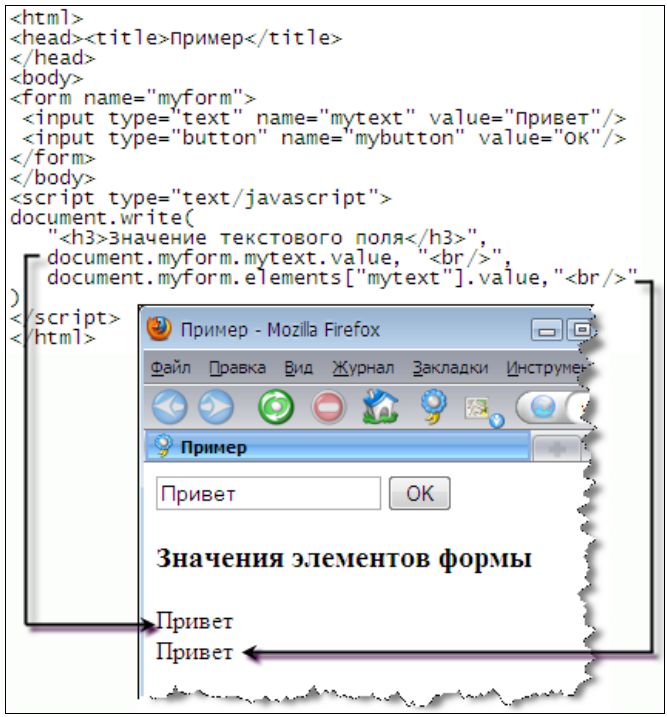


Рис. 14.4. Два варианта обращения к элементу формы

варианте обращение к форме и, далее, ее элементу происходит по значению атрибута `name`, а во втором варианте кроме `name` еще задействована коллекция `elements`.

> ***Внимание!***
>
> Все элементы могут иметь атрибут `id` (уникальный идентификатор), но некоторые из них (например, формы и их элементы, окна и фреймы) могут иметь атрибут `name` (имя). Атрибуты `id` и `name` не вполне эквивалентны. Во избежание недоразумений рекомендуется задавать для них одинаковые значения там, где требуется и то, и другое.

Если в примере на рис. 14.4 для элементов формы задать атрибуты `id` со значениями, совпадающими с соответствующими значениями атрибутов `name`, то получить, например, значение свойства `value` текстового поля можно и с помощью стандартного метода DOM:

```
document.getElementById("mytext").value;
```

На рис. 14.5 показан пример, в котором с помощью скрипта читаются и выводятся в окно браузера свойства элементов `<input>`. Здесь сначала с помощью метода `getElementsByTagName("input")` получаем коллекцию всех объектов, соответствующих элементам `<input>`, а затем в цикле (`for`) для каждого объекта данной коллекции читаем и выводим его свойства, такие как имя тега и значения атрибутов `id`, `name`, `type` и `value`.

А какими свойствами вообще может обладать объект, соответствующий тому или иному элементу (X)HTML-документа? Для получения ответа следует обратиться к справочнику или же с помощью скрипта вывести в окно браузера список всех свойств интересующего нас элемента. Чтобы расшифровать этот список, все равно придется обра-

титься к документации, но свойства, соответствующие атрибутам элементов, в этом списке нетрудно найти и понять. В листинге 14.1 приведен скрипт, который выводит в окно браузера все свойства для элемента с указанным значением атрибута `id`. Здесь применен объектный оператор `for...in...`, с помощью которого можно циклически прочитать все свойства объекта, записывая их имена в переменную (в листинге — `prop`).

```
<html>
<head><title>Пример</title>
</head>
<body>
<form name="myform" id="myform">
 <input type="text" name="mytext" id="mytext"
        value="Привет"/>
 <input type="button" name="mybutton" id="mytext"
        value="OK"/>
</form>
</body>
<script type="text/javascript">
document.write("<h3>Свойства элементов input</h3>");
var aElem=document.getElementsByTagName('input');
for(i=0;i<aElem.length;i++){
  document.write(
    "<p>",
    "Тег ",    aElem[i].tagName,
    " id=",    aElem[i].id,
    " name=",  aElem[i].name,   "<br/>",
    " type=",  aElem[i].type,   "<br/>",
    " value=",aElem[i].value,
    "</p>"
  )
}
</script>
</html>
```

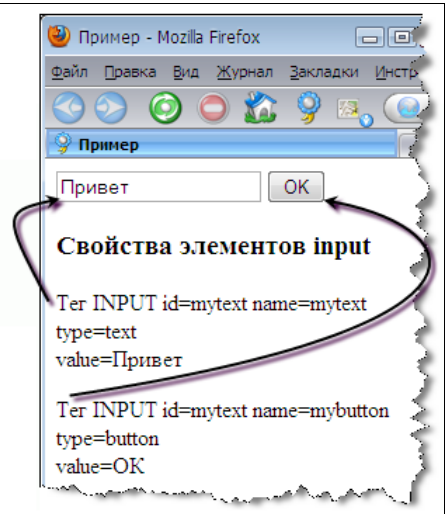


Рис. 14.5. Доступ к свойствам элементов с использованием метода `getElementsByTagName()`

Листинг 14.1. Создание списка свойств элемента

```
 <script type="text/javascript">
document.write("<h3>Свойства элемента</h3>");
/* вместо 'myelement' подставьте значение атрибута id
   интересующего элемента */
var elem=document.getElementById('myelement');
var prop,i;
for (prop in elem){ // запись имен свойств объекта elem в переменную prop
   document.write(prop,": " , elem[prop], "<br/>")
}
</script>
```

Чтобы получить список свойств объекта `document`, достаточно применить такой код:

```
for (prop in document){
   document.write("<b>",prop,":</b> " ,document[prop],"<br/>")
}
```

Аналогично получают списки свойств для других объектов, таких как `window`, `location`, `navigator` и т. д. Основные свойства важнейших объектов будут рассмотрены в *разд. 14.5*, а доступ к свойствам элементов (X)HTML-документа, заданных тегами, — в следующем разделе.

14.3. Доступ к свойствам элементов документа

Элементы (X)HTML-документа, создаваемые посредством тегов, имеют свойства, задаваемые атрибутами тегов и/или параметрами CSS. С помощью скриптов можно прочитать значения этих свойств и даже изменить их. Более того, можно поменять и содержимое тега.

14.3.1. Доступ к атрибутам

Для доступа к свойствам элементов (X)HTML-документов, которые задаются атрибутами соответствующих тегов, можно использовать следующие методы DOM:

- `getAttribute(`*атрибут*`)` — возвращает значение атрибута, указанного в строковом параметре, если он установлен, и `null` — в противном случае;
- `setAttribute(`*атрибут*`, `*значение*`)` — устанавливает для атрибута значение; последние указываются в качестве строковых параметров.

Данные методы применяются к объекту элемента, который является подобъектом объекта `document`. Объект элемента документа можно получить, например, методом `document.getElementById("`*значение_id*`")`. Чтобы установить для данного элемента новое значение его атрибута, достаточно применить такое выражение:

```
document.getElementById("значение_id").setAttribute(атрибут, значение)
```

Пусть, например, в (X)HTML-документе имеется тег, задающий графическое изображение:

```
<img id="myimg" src="picture1.jpg"/>
```

Тогда следующий скрипт заменит картинку из файла picture1.jpg на изображение из файла picture2.jpg:

```
document.getElementById("myimg").setAttribute("src", "picture2.jpg");
```

В следующем примере смена картинки произойдет, только если текущее изображение взято из файла picture1.jpg:

```
if (document.getElementById("myimg").getAttribute("src")=="picture1.jpg")
  document.getElementById("myimg").setAttribute("src", "picture2.jpg");
```

Объект элемента документа, кроме методов `getAttribute()` и `setAttribute()`, имеет еще несколько полезных свойств, из которых мы упомянем лишь несколько наиболее важных:

- `tagName` — свойство только для чтения, значением которого является имя тега данного элемента;
- `hasAttribute(`*атрибут*`)` — возвращает логическое значение, в зависимости от того, имеется ли в данном элементе некоторое установленное значение указанного в параметре атрибута или нет;
- `removeAttribute(`*атрибут*`)` — удаляет из элемента указанный в параметре атрибут.

Перечисленные методы работают во всех современных браузерах, поскольку они соответствуют стандарту DOM. Хотя они достаточно понятны, при их практическом применении важно учитывать следующие обстоятельства.

Во-первых, каждому элементу (X)HTML-документа можно установить любой атрибут, в том числе и не присущий ему по спецификации. По спецификации для каждого элемента определен некоторый набор атрибутов, которые ему присущи, т. е. которые соответствуют спецификации данного элемента (тега) и влияют на него, по крайней мере, при отображении в браузере. Например, элементу `<img>` по спецификации присущи атрибуты `src`, `width`, `height` и др., но атрибуты `left` и `top` ему не присущи. Если мы задаем значение атрибута, присущего данному элементу, то это как-то отразится внешним образом на последнем. Если же мы устанавливаем значение атрибута, не присущего данному элементу (в том числе и произвольно назначенного), то это значение сохранится в атрибуте как в переменной, но никак не отразится внешне при отображении элемента.

Во-вторых, атрибуты могут быть присущи элементам, но не установлены, т. е. не упомянуты в теге. В данном случае метод `getAttribute(`*`атрибут`*`)` вернет значение `null`. Но с помощью метода `setAttribute(`*`атрибут`*`, `*`значение`*`)` можно создать атрибут, даже если он явно не упоминался в теге, и установить для него какое-нибудь значение.

Кроме методов `getAttribute(`*`атрибут`*`)` и `setAttribute(`*`атрибут`*`, `*`значение`*`)` обращаться к свойствам, ассоциированным с атрибутами, можно и короче, через имена соответствующих свойств объекта:

- `document.getElementById(`*`значение_id`*`).`*`свойство`* — чтение значения свойства;
- `document.getElementById(`*`значение_id`*`).`*`свойство = новое_значение`* — установка нового значения свойства.

Например, установить значение атрибута `src` элемента, задаваемого тегом `<img id="myimage"....>`, можно посредством такого выражения:

```
document.getElementById("myimg").src="anypicture.jpg";
```

Для установки текста всплывающей подсказки, появляющейся при наведении указателя мыши на графическое изображение, достаточно применить такое выражение:

```
document.getElementById("myimg").title="Мой портрет";
```

В большинстве случаев имена атрибутов и соответствующих свойств объекта элемента (X)HTML-документа совпадают, но имеются и некоторые различия. Например, атрибут `class` указывает на определенный набор правил CSS. Чтобы установить некоторый набор правил CSS для элемента, можно воспользоваться одним из двух выражений:

```
document.getElementById("myelem").setAttribute("class", "myclass");
document.getElementById("myelem").className="myclass";
```

Здесь в первом выражении устанавливается некоторое значение для атрибута `class`, а во втором — то же значение для свойства `className` объекта элемента. Иначе говоря, атрибут имеет имя `class`, а свойство объекта, ему соответствующее, — `className`. Оба выражения устанавливают одно и то же значение для атрибута `class`, но различными способами и с использованием различных имен свойства, соответствующего имени класса CSS.

14.3.2. Доступ к свойствам CSS

Средства работы с CSS специфицированы в модели DOM 2 и поддерживаются браузерами Firefox, Opera, Safari и Chrome, но в Internet Explorer вместо некоторых из них

имеются фирменные аналогичные возможности. Стилевые параметры, как известно, задаются с помощью атрибута `style` для конкретного элемента, а также посредством правил с селекторами, записанных в контейнере `<style>` или во внешнем файле, ссылка на который указывается в теге `<link>`. Доступ к ним осуществляется по-разному.

Параметры CSS, определенные в атрибуте *style*

Доступ к стилевым параметрам, заданным для элемента посредством атрибута `style`, обеспечивает объект `style`, который является свойством объекта (X)HTML-элемента:

```
document.getElementById("myelem").style.параметр=значение;
```

Например, пусть в (X)HTML-документе задан элемент параграфа с красным текстом и размером шрифта 10 px:

```
<p id="mytext" style="color:red; font-size:10px">Привет!</p>
```

Тогда следующие выражения установят соответственно зеленый цвет текста и размер шрифта 14 px:

```
document.getElementById("mytext").style.color="green";
document.getElementById("mytext").style.fontSize="14px";
```

> ***Внимание!***
>
> В CSS имена многих стилевых параметров содержат дефис (например, `font-size`, `border-top-color`, `z-index` и т. п.). В JavaScript аналогичные свойства записывают без дефисов одним словом, подобно именам переменных, в так называемом стиле interCap, когда составляющие имени (кроме первого) начинаются с прописной буквы (например, `fontSize`, `borderTopColor`, `zIndex` и т. п.). Данное правило соблюдается для всех свойств CSS, за исключением `float`, для которого в JavaScript следует писать `cssFloat`, поскольку слово "float" зарезервированное.

Обратите внимание, что в объекте `style` имеются только те стилевые параметры, которые были установлены посредством атрибута `style`. Так, если для некоторого элемента параметр `color` был задан правилом внутри тега `<style>`, а в атрибуте `style` нет, то выражение

```
document.getElementById(значение_id).style.color
```

вернет пустое значение. С другой стороны, выражение

```
document.getElementById(значение_id).style.color="red"
```

установит для элемента красный цвет, но это никак не отразится на правилах CSS, заданных посредством тега `<style>`.

Параметры CSS, определенные в теге *<style>* или во внешнем файле

Теперь рассмотрим доступ к правилам стилей, заданных в (X)HTML-документе посредством элементов `<style>` (внедрение CSS) или `<link>` (связывание CSS). Правила в данных элементах имеют селекторы, из которых важное место занимают классы. Напомню, что класс CSS — это поименованный набор стилевых параметров, на который

элемент может ссылаться с помощью атрибута `class` (значение атрибута `class` — имя класса CSS).

Чтобы назначить элементу (X)HTML-документа с идентификатором `id="myelem"` определенный в CSS класс с именем, например `anyclass`, можно воспользоваться любым из следующих выражений:

```
document.getElementById("myelem").setAttribute("class", "anyclass");
document.getElementById("myelem").className="anyclass";
```

Если требуется назначить класс `anyclass` стилевых параметров для всех элементов, заданных, например, тегом `<p>` (абзац), то достаточно выполнить любой из двух скриптов:

```
/* Вариант 1 */
var elem=document.getElementsByTagName("p");
for(i=0; i<elem.length;i++)
{elem[i].setAttribute("class","anyclass")};

/* Вариант 2 */
var elem=document.getElementsByTagName("p");
for(i=0; i<elem.length; i++){elem[i].className="anyclass"};
```

Таблицы стилей для данного (X)HTML-документа можно сформировать посредством нескольких тегов `<style>` и/или `<link>`. Доступ к любой из них осуществляется посредством коллекции `styleSheets[i]`, где `i` — индекс таблицы стилей в порядке упоминания в документе (нумерация начинается с нуля). Так, если с документом связана единственная таблица стилей, то доступ к ней обеспечивается выражением `document.styleSheets[0]`.

Коллекция `styleSheets[]` имеет свойства, из которых можно выделить наиболее полезные:

- `disabled` — логическое значение, указывающее, активна или неактивна данная таблица (можно установить, для отключения или подключения таблицы стилей);
- `href` — содержит значение атрибута `href` тега `<link>` для связывания документа с внешней таблицей стилей. В соответствии со спецификацией DOM нельзя изменять ссылку на внешнюю таблицу стилей, как и правила последней.

Коллекция `styleSheets[]` содержит объекты таблиц стилей. Доступ к правилам CSS в соответствии со стандартом DOM 2 обеспечивается посредством подколлекции `cssRules[j]`, где `j` — индекс правила в порядке упоминания в таблице (нумерация начинается с нуля). Однако для Internet Explorer вместо `cssRules[]` используется нестандартная коллекция `rules[]`.

Итак, для получения доступа к отдельному правилу CSS следует использовать выражение:

- `document.styleSheets[i].cssRules[j]` — обращение в соответствии со стандартом DOM;
- `document.styleSheets[i].rules[j]` — обращение в Internet Explorer.

Получив доступ к конкретному правилу, можно воспользоваться следующими свойствами:

- `style` — обеспечивает доступ к отдельному стилевому параметру путем дальнейшего указания его имени; с помощью данного свойства можно изменить существующий или установить новый стилевой параметр, например, `document.styleSheets[i].cssRules[j].style.color="red"`;
- `selectorText` — селектор правила (набора стилевых параметров).

В листинге 14.2 приведен код, иллюстрирующий некоторые возможности обращения к таблицам CSS. В данном документе посредством тегов `<style>` заданы две таблицы стилей с одинаковыми наборами селекторов класса, а также элемент `<p>`, ссылающийся на класс `.myclass2`. В силу каскадности таблиц стилей, по умолчанию должно использоваться правило из второй таблицы. Однако в данном примере с помощью метода `confirm()` пользователю предоставляется возможность выбора одной таблицы стилей, дезактивировав другую. В окно браузера выводятся селектор первого правила и значение параметров `font-size` и `font-family` из первой и второй таблиц соответственно. Чтобы данный скрипт работал и в Internet Explorer, пришлось принять особые меры. Вид документа в браузере показан на рис. 14.6.

Листинг 14.2. Пример обращения к CSS

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример</title>
<style type="text/css" >
  .myclass1 {font-size:15px;font-family:arial}
  .myclass2 {font-size:25px;font-family:times}
</style>
<style type="text/css">
  .myclass1 {font-size:5px;font-family:arial;}
  .myclass2 {font-size:10px;font-family:times}
</style>
</head>
<body>
<p class="myclass2">Это текст для эксперимента</p>
</body>
<script type="text/javascript">
/* Вывод некоторых свойств CSS в окно браузера */
if(document.styleSheets[0].rules) // для IE
   document.styleSheets[0].cssRules=document.styleSheets[0].rules;
document.write(
    "Таблица CSS 0","<br/>",
    document.styleSheets[0].cssRules[0].selectorText,
    " font-size:",
    document.styleSheets[0].cssRules[0].style.fontSize,
    "<br/>"
);
if(document.styleSheets[1].rules) // для IE
   document.styleSheets[1].cssRules=document.styleSheets[1].rules;
```

```
document.write(
    "Таблица CSS 1","<br/>",
    document.styleSheets[1].cssRules[0].selectorText,
    " font-family:",
    document.styleSheets[1].cssRules[0].style.fontFamily,
    "<br/>"
);
/* Выбор таблицы CSS: */
if(confirm(
   "Выберите таблицу стилей:\n OK - стиль 1, Отмена — стиль 2"))
   document.styleSheets[1].disabled=true
else
   document.styleSheets[0].disabled=true;
</script>
</html>
```

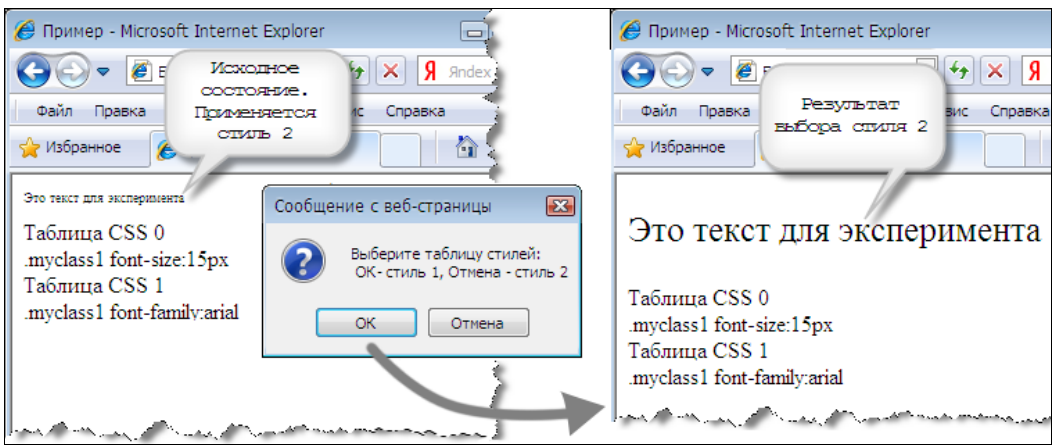


Рис. 14.6. Вид документа листинга 14.2

Для добавления и удаления правил CSS предназначены следующие методы объекта `styleSheet`:

- `insertRule(`*правило*`,` *индекс*`)` — добавляет указанное правило в коллекцию `cssRules[]` на место, указанное индексом (метод DOM 2); например,

  ```
  docyment.styleSheets[0].insertRule(
                                "h1 {color:red; font-size:10px}");
  ```

  чтобы добавить правило в конец коллекции `styleSheets[]`, можно в качестве индекса использовать значение свойства `length` объекта `cssRules`, например, `document.styleSheets[0].cssRules.length`.

  Для Internet Explorer применяется аналогичный метод `addRule(`*селектор*`,` *значение*`,` *индекс*`)`; если индекс не указан, то правило добавляется в конец коллекции `rules[]` например,

  ```
  document.styleSheets[0].addRule("h1","color:red; font-size:10px");
  ```

- `deleteRule(`*индекс*`)` — удаляет правило, имеющее указанный индекс, из коллекции `cssRules[]` (метод DOM 2).

  Для Internet Explorer применяется аналогичный метод `removeRule(индекс)`, удаляющий правило с указанным индексом из коллекции `rules[]`.

14.3.3. Доступ к содержимому элемента

Модель DOM предлагает средства для просмотра содержимого, создания новых и удаления уже имеющихся узлов дерева, представляющего документ, такие как `createElement()`, `createTextNode()`, `appendChild()` и др. С помощью этих методов можно динамически изменить существующий документ вплоть до полной неузнаваемости, однако их применение нельзя назвать простым и удобным. В данной книге они не будут рассматриваться.

Еще до появления стандарта DOM существовало очень полезное и удобное свойство `innerHTML`. Это свойство объекта элемента документа, и для контейнерных тегов его значением является строка, содержащая HTML-код, находящийся внутри контейнера. Например, для элемента

```
<p id="mypar">Это <b>просто</b> текст</p>
```

значением `document.getElementById("mypar").innerHTML` является строка

```
"Это <b>просто</b> текст ".
```

Значение свойства `innerHTML` можно не только прочитать, но и изменить, при этом соответствующий элемент тут же модифицируется без перезагрузки документа. Например, выражение

```
document.getElementById("mypar").innerHTML = "<img src='picture.jpg'/>"
```

заменит содержимое контейнера `<p>` элементом графического изображения.

Таким образом, можно динамически изменять HTML-код фрагментов документа. Свойство `innerHTML` не соответствует стандартам W3C, но в силу своей полезности и удобства применения его поддерживают все основные браузеры.

В листинге 14.3 приведен код, иллюстрирующий возможности применения свойства `innerHTML`. Скрипт выводит в диалоговое окно пронумерованный список значений `innerHTML` для всех элементов документа (коллекции `document.getElementsByTagName("*")`). Точнее, выводятся порядковый номер, элемент коллекции, имя тега и значение `innerHTML`. Результат работы данного скрипта в браузере Firefox показан на рис. 14.7. Для Opera, Safari и Chrome все выглядит аналогично, а в Internet Explorer 8 несколько по-другому из-за иного состава коллекции всех элементов. В частности, в Internet Explorer в данную коллекцию в качестве первого элемента включен элемент `<html>`, сам содержащий все элементы документа.

Листинг 14.3. Значения свойства `innerHTML` для всех элементов документа

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Свойство innerHTML</title></head>
<body>
<a href = "http://www.yandex.ru">Яндекс</a>
<br/>
<img src="picture.jpg" height="200"/>
</body>
<script "type=text/javascript">
 var xstr="";
 var elements=document.getElementsByTagName("*");
 for (var i=1; i<elements.length;i++){
   elem=elements[i];
   xstr+=i+". "+elem+" "+elem.tagName+": \n"+elem.innerHTML+"\n"
 }
 alert(xstr); // вывод значения innerHTML
</script>
</html>
```

Браузеры Internet Explorer, Opera, Chrome и Safari, в отличие от Firefox, кроме `innerHTML` поддерживают еще три аналогичных свойства: `outerHTML`, `innerText` и `outerText`.

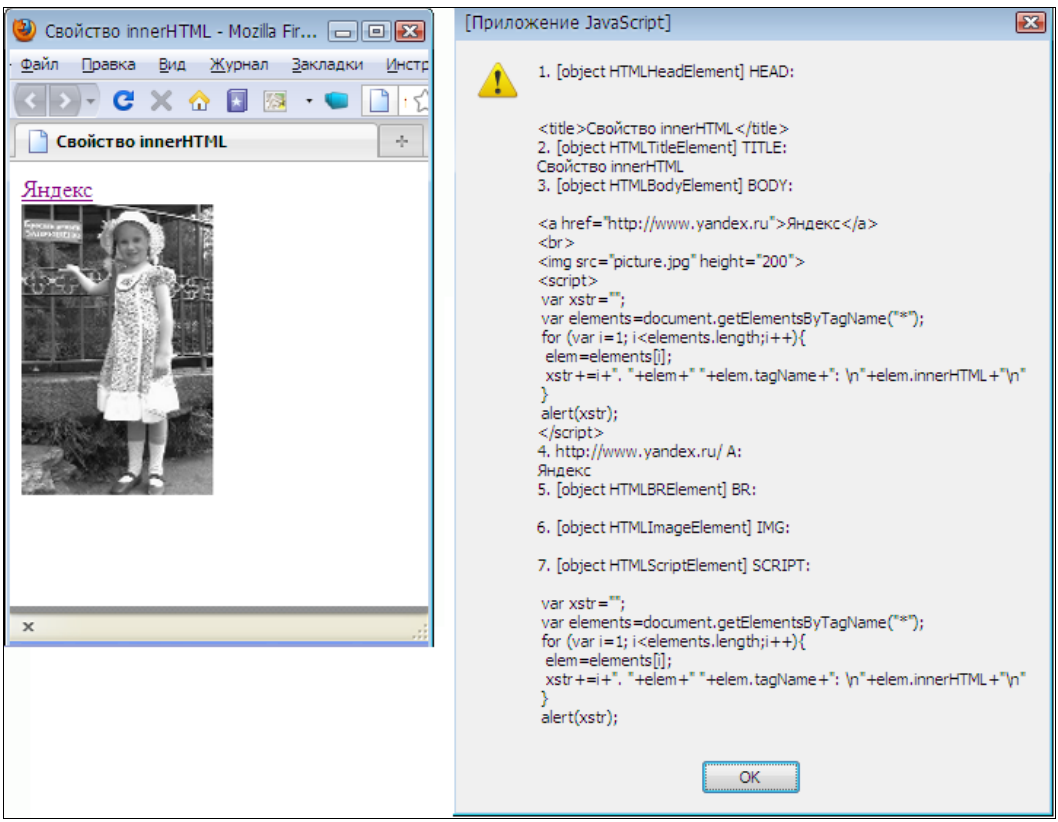


Рис. 14.7. Результат работы скрипта из листинга 14.3 в Firefox

Свойство `outerHTML` отличается от `innerHTML` тем, что содержит весь HTML-код элемента, включая также внешние открывающий и закрывающий теги элемента.

Значение свойства `innerText` — все текстовое содержимое, заключенное между открывающим и закрывающим тегами элемента. Если в эту строку входят HTML-теги, то они игнорируются. Таким образом, `innerText` следует понимать как весь внутренний текст, содержащийся в контейнере.

В отличие от `innerText`, свойство `outerText` включает в себя данные, содержащиеся и в открывающем, и в закрывающем тегах элемента. Следовательно, `outerText` нужно понимать как весь текст, содержащийся в контейнере, включая и его внешние теги.

При вставке в (X)HTML-документ с помощью тега `<object>` элементов управления ActiveX иногда требуется информация, которая обычно задается в тегах `<param>`. Конечно, подобную информацию правильнее было бы получить из описания от производителя элемента ActiveX, но можно попытаться добыть ее и с помощью свойства `innerHTML` или `outerHTML`. На рис. 14.8 показан пример вывода полного содержимого тега `<object>` для элемента управления Tabular Data Control, обеспечивающего работу с табличными данными в текстовом файле (см. *разд. 12.4.2*). Хотя в контейнере `<object>` теги `<param>` явно не указывались, они все же присутствуют в коде элемента. Рассматривая их, нетрудно понять, что, например, параметр `SortColumn` задает столбец, по значениям которого сортируются строки таблицы, а параметр `FilterCriterion` определяет критерий фильтрации данных.

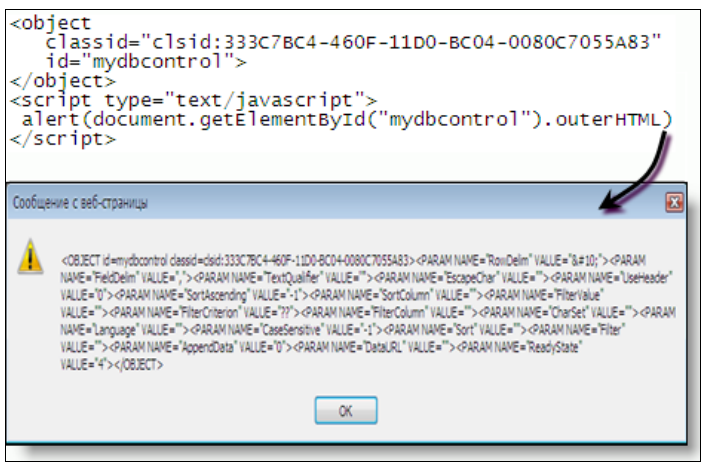


Рис. 14.8. Пример использования свойства `outerHTML` в Internet Explorer

14.4. Обработка событий

Одно из важнейших назначений сценариев (скриптов) — обработка событий, возникших как в результате действий пользователя (манипуляции мышью и клавиатурой), так и без его прямого участия (например, окончание загрузки документа).

Прежде всего, необходимо понять трехместное отношение "событие (`event`) — контекст (`context`) — обработчик (`handler`)". Событие возникает в результате каких-то дей-

ствий пользователя (например, щелчок мышью) и/или по другим причинам. Любое событие происходит при каких-то обстоятельствах (т. е. в контексте), например, щелчок мышью может произойти, когда ее указатель находился на графическом изображении, кнопке или еще где-то в области отображения документа. Событие может быть как-то обработано сценарием, чтобы браузер среагировал на него определенным образом.

Для некоторых событий в определенных контекстах предусмотрены встроенные обработчики, или, другими словами, обработчики по умолчанию. Например, щелчок (событие) на ссылке (контекст) вызывает обработчик по умолчанию, который загружает в браузер документ с указанным в ссылке адресом. Аналогично, щелчок на кнопке типа `submit` (т. е. на элементе `<input type="submit"/>`) приводит к отправке данных, содержащихся в элементах формы. При этом сценарий отправки можно не писать. Однако для обработки щелчка на обыкновенной кнопке, графическом изображении или обыкновенной текстовой строке требуется написать специальный скрипт — обработчик данного события в определенном контексте. Специальными обработчиками можно заменить обработчики по умолчанию. Например, вы можете создать свой обработчик щелчка на кнопке типа `submit`, который прежде отправки проверяет корректность данных, введенных пользователем в форму.

В настоящее время существует несколько моделей событий, но можно выделить так называемую базовую модель, поддерживаемую приблизительно одинаково всеми браузерами. Она проста и в то же время достаточно гибка. В большинстве случаев ничего большего и не требуется. Фирменные модели, а также модель DOM 2 совместимы с базовой. Так что, можно оставаться в рамках последней при создании сценариев обработки событий даже для самых новых браузеров. Фирменные модели и модель DOM 2 дают больше возможностей для обработки событий, но их использование в полной мере сопряжено с проблемами межбраузерной совместимости. Поэтому в данной книге мы ограничимся рассмотрением основных средств преимущественно базовой модели.

14.4.1. Привязка обработчиков событий

Во многих, если не в большинстве, случаях событие и его обработчик связываются с так называемым целевым объектом — (X)HTML-элементом, в котором это событие происходит. Целевой объект еще называют приемником события. Привязку события и его обработчика к элементу можно выполнить несколькими способами:

- применением специальных атрибутов, имена которых указывают на соответствующие события;
- регистрацией обработчиков событий в сценарии;
- применением средств модели событий DOM 2.

Первые два способа работают во всех браузерах, а последний, к сожалению, не работает в Internet Explorer 8 и более ранних версиях.

Атрибуты-события

Первый способ состоит в применении так называемых атрибутов-событий, имена которых образуются из английского названия события с добавлением префикса "on". Например, `onclick` (click — щелчок, onclick — при щелчке), `onmouseover` (при наведении

указателя мыши), onsubmit (при отправке данных формы на сервер) и т. д. При этом используется следующая схема:

```
<тег ... атрибут-событие="код_обработчика">
```

Программный код обработчика представляет собой одно или несколько выражений, разделенных точкой с запятой. При большом объеме кода его обычно оформляют в виде функции. Например, обработчик щелчка на графическом изображении можно привязать к элементу изображения следующими способами:

```
<img id="myimg" src="picture.jpg"
     onclick="alert('Не щелкай на мне')"/>
<img id="myimg" src="picture.jpg" onclick="myfunc()"/>
```

Здесь myfunc() — некоторая определенная функция, выполняющая обработку события.

В одном и том же элементе можно использовать одновременно несколько различных атрибутов-событий, назначая им разные или одинаковые обработчики. Например, следующий элемент графического изображения будет реагировать на щелчок выполнением функции myfunc1(), а на наведение указателя мыши — функции myfunc2():

```
<img id="myimg" src="picture.jpg"
     onclick="myfunc1()" onmouseover="myfunc2()"/>
```

Отключение обработчика события производится в сценарии с помощью выражения вида:

```
объект.атрибут-событие = null;
```

Например, следующее выражение отключает обработчик щелчка на элементе с атрибутом id="myelement":

```
document.getElementById("myelement").onclick=null;
```

Регистрация обработчика события

Второй способ привязки события и его обработчика к элементу называется регистрацией обработчика. Он реализуется в сценарии и основан на том, что обработчик доступен в виде метода объекта, к которому он привязан. При этом используется следующая схема:

```
объект.атрибут-событие = имя_функции;
```

Здесь *имя_функции* указывается без последующих круглых скобок и не заключается в кавычки, а *атрибут-событие* является свойством объекта, ассоциированным с атрибутом (см. *разд. 14.3.1*). Такого рода свойства записывают в нижнем регистре (например, onclick, но не onClick).

В листинге 14.4 приведен пример, в котором для графического изображения регистрируются обработчики наведения и ухода указателя мыши.

Листинг 14.4. Регистрация обработчиков событий

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>События</title></head>
<body>
<img id="myimg" src="picture.jpg"/>
</body>
<script type="text/javascript">
/* Регистрация обработчиков событий: */
document.getElementById("myimg").onmouseover=overimg;
document.getElementById("myimg").onmouseout =outimg;
/* Обработчики: */
function overimg(){
  document.getElementById("myimg").src="picture2.jpg"
}
function outimg(){
  document.getElementById("myimg").src="picture1.jpg"
}
</script>
</html>
```

При регистрации обработчика события можно сразу использовать анонимное определение функции (см. *разд. 14.5.3*):

```
объект.атрибут-событие = function(параметры){код};
```

Например, обработчик наведения указателя мыши на графическое изображение (см. листинг 14.4) можно зарегистрировать еще и так:

```
document.getElementById("myimg").onmouseover=function() {
      document.getElementById("myimg").src="picture2.jpg"};
```

В данном случае код обработчика можно записать еще короче, используя ключевое слово `this`, указывающее на то, что свойство `src` относится к тому же объекту, для которого задается данный обработчик:

```
document.getElementById("myimg").onmouseover=function(){
                                this.src="picture2.jpg"};
```

О ключевом слове `this` для ссылки на объект подробнее будет рассказано в *разд. 14.4.2*.

Обработчик события необходимо зарегистрировать после того, как будет создан объект, соответствующий элементу, к которому он привязывается. В противном случае возникнет ошибка. Во избежание этого можно разместить сценарий с регистрацией обработчика после задания элемента. В листинге 14.4 весь сценарий находится в конце документа, так что он выполнится после создания объектов DOM, соответствующих (X)HTML-элементам документа. Другой вариант состоит в том, чтобы назначить обработчики событий в обработчике события `onload` элемента `<body>`. Это событие происходит по окончании загрузки документа в браузер. Данный подход реализован в листинге 14.5, в котором весь сценарий расположен в контейнере `<head>`, а регистрация обработчиков `overimg()` и `outimg()` определена в теле функции `init()` — обработчике события `onload` элемента `<body>`.

Листинг 14.5. Регистрация обработчиков событий в обработчике события `onload`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>События</title>
  <script type="text/javascript">
    function init(){ // обработчик события onload
    /* Регистрация обработчиков событий: */
      document.getElementById("myimg").onmouseover=overimg;
      document.getElementById("myimg").onmouseout =outimg;
    }
    /* Обработчики: */
    function overimg(){
      document.getElementById("myimg").src="picture2.jpg"
    }
    function outimg(){
      document.getElementById("myimg").src="picture1.jpg"
    }
  </script>
</head>
<body onload="init()">
<img id="myimg" src="picture.jpg"/>
</body>
</html>
```

Вместо привязки обработчика `init()` события `onload` к элементу `<body>` можно зарегистрировать этот обработчик так:

```
window.onload=init;
```

Преимущество регистрации обработчиков перед использованием атрибутов — возможность отделить собственно (X)HTML-код от средств обеспечения его функциональности. Кроме того, регистрация удобна в случаях, кода предполагается динамическое добавление, удаление или изменение обработчиков событий.

Иллюстрация возможности динамического создания обработчика события приведена в листинге 14.6. (X)HTML-документ содержит графическое изображение, текстовую область и кнопку. В текстовую область можно ввести код обработчика (тело определения анонимной функции). Щелчок на кнопке создаст выражение регистрации события `onmouseover` для графического изображения и выполнит его (если, конечно, не будет ошибок). А именно к содержимому текстовой области добавляется окаймление, чтобы оно стало анонимным определением функции, назначенным в качестве обработчика наведения указателя мыши на графическое изображение. Полученная таким образом строка символов кода в качестве параметра передается функции `eval()`, которая и выполняет выражение регистрации обработчика. Если вы введете в текстовую область, например, строку `this.width="200"; this.height="100"` и щелкнете на кнопке, то при наведении указателя мыши на картинку последняя примет указанные размеры. Если теперь вместо прежней вы введете строку `this.src="otherpicture.jpg"`, то после щелчка на кнопке и наведении указателя мыши на картинку браузер попытается за-

менить ее другой из файла otherpicture.jpg. Обратите внимание, что кавычки в текстовой области должны быть двойными.

Листинг 14.6. Динамическое создание обработчика события

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>События</title></head>
<body>
<img id="myimg" src="picture.jpg"/>
<div>
  <textarea id="code" cols="40" rows="6"></textarea>
  <button id="mybtn">Создать обработчик</button>
</div>
</body>
<script type="text/javascript">
/* Регистрация обработчика щелчка на кнопке: */
document.getElementById("mybtn").onclick=mybtnclick;
function mybtnclick(){ // обработчик щелчка на кнопке
  var mycode=document.getElementById("code").value;
  if (mycode)
     eval('document.getElementById("myimg").onmouseover=function(){'+
         mycode+'}')
  else alert("Нет кода для выполнения");
}
</script>
</html>
```

Для отключения обработчика события можно просто выполнить выражение вида:

```
объект.атрибут-событие = null;
```

Средства DOM

В модели DOM 2 для привязки обработчиков событий есть метод addEventListener() (добавить приемник события), применение которого имеет следующий синтаксис:

```
объект.addEventListener("событие", обработчик, фаза);
```

Здесь объект — узел объектной модели, к которому привязывается обработчик события. Данный метод принимает параметры:

- *событие* — отслеживаемое событие; задается как атрибут-событие, но без приставки "on", например click, mouseover и т. д.; название события заключается в кавычки;
- *обработчик* — функция, которая вызывается, когда происходит указанное событие; имя функции указывается без кавычек;
- *фаза* — логическое значение, определяющее, когда должен вызываться обработчик события — в фазе захвата (true) при его нисхождении по объектной модели к целевому объекту или в фазе возврата (false) при его всплывании от целевого объекта

вверх по иерархии объектов; о фазах распространения событий по объектной модели см. *разд. 14.4.5*.

Метод `removeEventListener("событие", обработчик, фаза)` с теми же параметрами служит для удаления обработчика.

Пример использования метода `addEventListener()` приведен в листинге 14.7.

Листинг 14.7. Привязка обработчиков событий методом `addEventListener()`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>События</title>
<script type="text/javascript">
/* Обработчики: */
function overimg(){
  document.getElementById("myimg").src="picture2.jpg"
}
function outimg(){
  document.getElementById("myimg").src="picture1.jpg"
}
</script>
</head>
<body>
<img id="myimg" src="picture.jpg"/>
</body>
<script type="text/javascript">
/* Регистрация обработчиков событий: */
document.getElementById("myimg").addEventListener(
                                "mouseover",overimg, true);
document.getElementById("myimg").addEventListener(
                                 "mouseout",outimg, false);
</script>
</html>
```

Применение методов DOM позволяет привязать к объекту сразу несколько обработчиков одного и того же события. В этом случае при наступлении события вызываются все обработчики, хотя порядок их вызова может быть произвольным. Достичь этого рассмотренными ранее способами нельзя. Даже если вы формально используете несколько одинаковых атрибутов-событий с разными обработчиками или укажете несколько регистраций разных обработчиков для одного и того же события, то в действительности будет задана та пара событие-обработчик, которая упомянута последней. Далее, применение метода DOM позволяет обработать событие в фазе захвата, когда оно еще только спускается к целевому объекту (см. *разд. 14.4.5*). Имеются и другие полезные возможности.

Примечание

К сожалению, браузер Internet Explorer версии 8 не поддерживает методы `addEventListener()` и `removeEventListener()`, что пока ограничивает их применение.

14.4.2. Область видимости обработчиков событий

Что видно из сценария или, другими словами, каков контекст сценария? Контекстом выполнения сценария является объект `window`, в котором находится его текст. Вместе с тем, скрипт можно включить в текст обработчика события, и в этом случае он имеет контекст того объекта, к которому данный обработчик привязан. Чтобы в тексте обработчика оперировать именно свойствами привязанного объекта, достаточно применить ключевое слово `this` в качестве ссылки на последний.

В *разд. 14.4.1* уже приводился пример использования `this`. На рис. 14.9 показан еще один пример, в котором щелчок на поле ввода, находящемся в форме, выводит в диалоговое окно значение атрибута `name` именно поля, а не формы или еще чего-нибудь.

Важно отметить, что только скрипт в значении атрибута-события имеет указанную область видимости, а остальной программный код будет иметь обычный контекст (рис. 14.10).

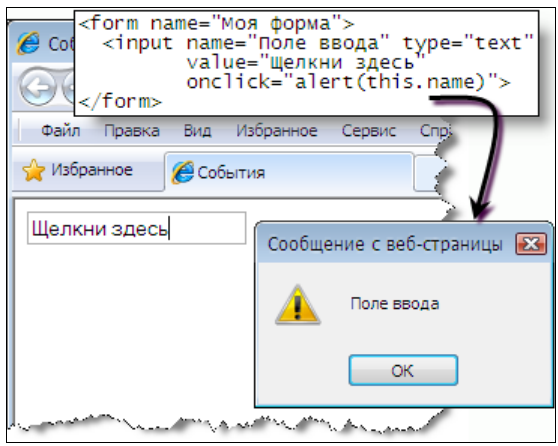


Рис. 14.9. Областью видимости обработчика щелчка на поле ввода является объект поля ввода

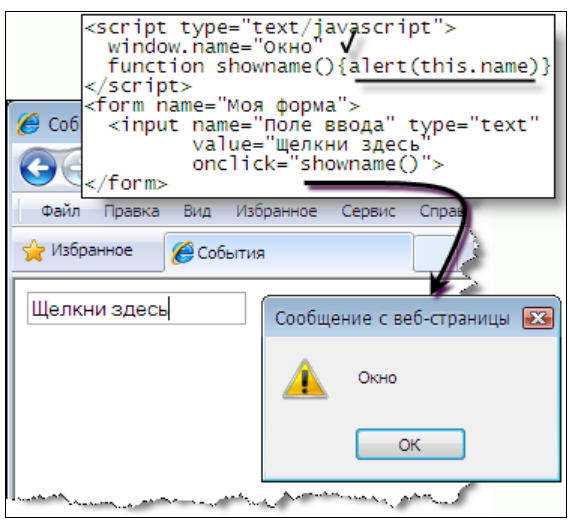


Рис. 14.10. Областью видимости обработчика щелчка на поле ввода является объект окна `window`

Заметим, что если бы мы не определили window.name, то при щелчке на поле ввода в диалоговом окне появилось бы пустое сообщение, а не значение атрибута name тега <form>.

14.4.3. Изменение поведения элементов по умолчанию

Отдельные элементы для некоторых событий имеют обработчики по умолчанию. Например, при щелчке на ссылке по умолчанию осуществляется переход на указанную страницу, на кнопке типа submit — данные формы передаются на сервер, на кнопке типа reset — все данные формы очищаются.

Поведение по умолчанию можно при желании отменить или изменить. Например, может потребоваться, чтобы при щелчке на ссылке выполнилась некоторая функция, а не попытка загрузки какого-то документа, а при щелчке на кнопке submit — предварительная проверка наличия в форме каких-либо данных или подтверждения намерения передать их.

Для изменения поведения элементов по умолчанию достаточно сделать так, чтобы обработчик события вернул значение false. Вот два типичных примера.

- Щелчок на ссылке должен приводить к вызову некоторой функции, а не загружать в браузер документ:

```
<a href="" onclick="myfunc(); return false">Щелкни здесь</a>
```

  Здесь в коде обработчика события onclick последнее выражение явным образом возвращает значение false и, следовательно, переход по URL-адресу ссылки (даже если он не пуст) не сработает, а будет вызвана функция myfunc(), которая что-то делает по усмотрению программиста. Впрочем, выражение return false, определяющее возвращаемое значение, можно вставить и в тело функции myfunc().

- Щелчок на кнопке типа submit в форме требует подтверждения о необходимости отправить данные, введенные пользователем:

```
<input type="submit"
onclick=
"return confirm('Вы действительно хотите передать данные на сервер?')"
value="Передать данные"/>
```

  Здесь метод confirm() объекта window возвращает true или false в зависимости от того, на какой кнопке выведенного диалогового окна щелкнул пользователь. Отправка данных произойдет, если метод confirm() вернет true, т. е. если пользователь щелкнет на кнопке **OK** (и если к тому же указан адрес и метод передачи данных в соответствующих атрибутах тега <form>, задающего форму); в противном случае браузер не будет отправлять данные.

В браузере Internet Explorer 4+ для описанной цели можно еще использовать свойство returnValue объекта событий Event, содержащего информацию о произошедшем событии. Так, для предотвращения поведения элементов по умолчанию можно в обработчике события написать event.returnValue=false.

Итак, модель DOM 2 в своей базовой части позволяет отменить действие по умолчанию, связанное с событием, просто возвратив значение false из обработчика. Этот способ поддерживают все браузеры. Вместе с тем, DOM 2 для этой цели предлагает и соб-

ственный метод `preventDefault()` объекта `Event`, выходящий за рамки базовой модели событий и, следовательно, он не работает в Internet Explorer. В листинге 14.8 приведен пример, в котором отменяются действия по умолчанию, связанные со щелчками, где бы они ни происходили. В частности, щелчки на ссылках не будут выполнены.

Листинг 14.8. Отмена действий по умолчанию, связанных со щелчками на документе

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Обработка событий</title></head>
<body>
<a href="http://www.yandex.ru"Ю>Яндекс</a>
<a href="http://www.rambler.ru"Ю>Rambler</a>
</body>
<script type="text/javascript">
/* Только для браузеров, поддерживающих DOM 2 */
function preventClicks(event){ // предотвратить умолчания для щелчков
  event.preventDefault(); // метод DOM 2
}
/* Регистрация обработчика щелчков на документе */
document.addEventListener("click", preventClicks, true) </script>
</html>
```

Здесь обработчику `preventClicks(event)` передается в качестве параметра экземпляр объекта события (подробнее об этом см. *разд. 14.4.6*). Обработчик использует его для того, чтобы вызвать метод `preventDefault()` для отмены текущего события (в данном случае щелчка).

Заметим, что применение метода `preventDefault()` не отменяет процесс распространения события по иерархии объектов. Поэтому, если в вышестоящем элементе определен какой-нибудь обработчик для того же события, то он будет выполнен.

14.4.4. Программный вызов обработчика события

Обработчики событий можно вызвать программно, а не только манипулируя мышью и клавиатурой. Иначе говоря, можно выполнить программный код обработки события, которое не произошло из-за действий пользователя или по другим причинам. Вызов обработчика того или иного события обеспечивает метод объекта, название которого совпадает с названием атрибута-события. Например, вызов обработчика щелчка осуществляет метод `onclick()`.

В следующем примере (листинг 14.9) обработчик события `onclick` срабатывает сразу же при загрузке документа, даже если пользователь не щелкал мышью на кнопке.

Листинг 14.9. Программный вызов обработчика события

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Обработка событий</title></head>
<body>
<button id="mybtn" onclick="alert('Щелчок!')" >
Щелкни здесь
</button>
</body>
<script type="text/javascript">
document.getElementById("mybtn").onclick() // вызов обработчика
</script>
</html>
```

При загрузке данного кода в браузер сразу же будет выведено диалоговое окно с сообщением "Щелчок!", хотя пользователь еще не успел щелкнуть на кнопке.

Разумеется, в рассмотренном примере мы могли бы не определять обработчик с помощью атрибута-события, а зарегистрировать его в секции `<script>`, где и происходит его вызов:

```
document.getElementById("mybtn").onclick="alert('Щелчок!')";// регистрация
document.getElementById("mybtn").onclick(); // вызов обработчика
```

Для некоторых элементов обработчик можно вызвать программно не напрямую, как в предыдущем примере, а путем генерации соответствующих событий. События генерируются методами, названия которых подобны названиям атрибутов-событий, но не имеют префикса "on". Например, генерацию щелчка обеспечивает метод `click()`. Программно сгенерированное событие влечет за собой вызов соответствующего обработчика, если, конечно, он определен.

В следующем примере (листинг 14.10) в сценарии генерируется событие, соответствующее щелчку на кнопке формы, и выполняется его обработчик, выводящий диалоговое окно с сообщением "Щелчок!". Это происходит сразу после загрузки документа без вмешательства пользователя.

Листинг 14.10. Программная генерация события

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Обработка событий</title></head>
<body>
<form name="myform" id="myform">
 <input id="mytext" type = "text" value="Привет"/>
 <input id="mybutton" type = "button" value = "Щелкни здесь"
     onclick = "alert('Щелчок!')"/>
</form>
</body>
<script "type=text/javascript">
document.getElementById("mybutton").click() // генерация события
</script>
</html>
```

Иногда событие удобнее сгенерировать, а не просто непосредственно вызвать его обработчик. Например, для ввода данных в текстовое поле (элемент `<input>`) требуется сначала установить на нем фокус ввода или, другими словами, установить в нем курсор. После этого пользователь может вводить данные посредством клавиатуры. При загрузке документа с текстовыми полями курсор (фокус ввода) не устанавливается ни в одном из них, а потому пользователь должен щелчком мыши или клавишей табуляции активизировать нужное поле ввода прежде, чем вводить собственно данные. В большинстве случаев так поступать весьма неудобно: желательно сразу активизировать какое-нибудь поле ввода. Это можно сделать программно, сгенерировав событие установки фокуса ввода на нужном элементе с помощью метода `focus()`. В следующем примере (листинг 14.11) страница содержит три текстовых поля, и по окончании ее загрузки в браузер курсор сразу же устанавливается во втором поле методом `focus()`.

Листинг 14.11. Установка фокуса ввода

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Обработка событий</title></head>
<body>
<input type="text" id="inp1"/>
<input type="text" id="inp2"/>
<input type="text" id="inp3"/>
</body>
<script "type=text/javascript">
 window.onload=function(){
    document.getElementById("inp2").focus(); // фокус на второе поле
 }
</script>
</html>
```

Здесь мы обращаемся к объекту `document.getElementById("inp2")`, соответствующему второму полю ввода, и применяем к нему метод `focus()` генерации события установки фокуса. В результате во втором поле появляется курсор и, следовательно, оно готово к вводу данных с клавиатуры.

Не все события можно сгенерировать программно. Перечень таких событий зависит от браузера. В табл. 14.1 приведены сведения о событиях и элементах документа, для которых это возможно во всех браузерах.

***Таблица 14.1.** События, которые можно сгенерировать программно*

Методы генерации событий	HTML-элементы
`click()` — щелчок	`<input>` со значениями атрибута `type`: `"button"`, `"checkbox"`, `"radio"`, `"reset"`, `"submit"`; `<a>`
`blur()` — потеря элементом фокуса ввода	`<a>`, `<input>`, `<select>`, `<textarea>`

Таблица 14.1 *(окончание)*

Методы генерации событий	HTML-элементы
`focus()` — получение элементом фокуса ввода	`<a>`, `<input>`, `<select>`, `<textarea>`
`select()` — выбор текста (обычно выделением его с помощью мыши)	`<input>` со значениями атрибута `type`: `"text"`, `"password"`, `"file"`; `<textarea>`
`reset()` — возвращение формы в исходное состояние	`<form>`
`submit()` — отправка данных формы на сервер	`<form>`

14.4.5. Прохождение событий

Кроме рассмотренной базовой модели событий, существуют фирменные модели, реализованные в браузерах Netscape Navigator 4+ и Internet Explorer 4+, а также стандартная модель событий DOM 2. Указанные фирменные модели исторически предшествуют модели событий DOM 2, но они легли в основу последней. В отличие от базовой модели, фирменные модели и DOM 2 предоставляют возможность использования объекта `Event`, содержащего сведения о контексте произошедшего события, а также предусматривают распространение событий по иерархии объектов документа. Браузеры Netscape уже практически исчезли, браузеры на базе Mozilla поддерживают DOM 2, а Internet Explorer 4+ — только свою фирменную модель. Модель DOM 2 объединила в себе принципы моделей Netscape и Internet Explorer и существенно расширила соответствующие возможности. В данном разделе мы рассмотрим распространение событий, а объект `Event` — в следующем.

Элементы (X)HTML, как известно, образуют иерархическую структуру вложенности. Например, текстовые поля и кнопки могут находиться в форме, а форма — в теле документа, т. е. в элементе `<body>`. Щелкая мышью на кнопке внутри формы, мы одновременно "щелкаем" и на самой форме, и на документе в целом. Иначе говоря, событие, возникшее в некотором месте, распространяется далее по объектам. В различных браузерах модели распространения событий были реализованы по-разному. Например, в Netscape 4+ анализ события начинается на вершине иерархии (т. е. на объекте самого высокого уровня) и "спускается вниз" к целевому объекту. В Internet Explorer 4+ распространение события от целевого объекта распространяется вверх по иерархии. Говорят, что событие "всплывает" подобно пузырьку с воздухом в воде. В модели событий DOM 2 предусмотрен как "спуск", так и "всплывание" событий: сначала событие спускается от объекта `document` к целевому объекту, а затем всплывает от целевого объекта к `document`. При этом выполнение его обработчиков происходит в фазе всплывания, но в фазе спуска можно захватить событие и что-то успеть сделать до фазы его всплытия и выполнения обработчиков. В данной книге мы не будем рассматривать захваты событий, а остановимся лишь на фазе их всплытия, когда выполняются заданные обработчики событий и когда можно прекратить дальнейшее всплытие последних. Данная тема актуальна для всех браузеров.

В листинге 14.12 в качестве примера приведен XHTML-код документа, в котором кнопка (<button>) находится внутри контейнера, заданного тегом . Контейнер в свою очередь расположен в контейнере <body>. Внешний вид данного документа показан на рис. 14.11. Все перечисленные элементы документа реагируют на щелчок мышью, выводя диалоговое окно с сообщением о том, какой элемент принял данное событие. Обработчики щелчка на данных элементах определены посредством атрибута-события onclick. Кроме того, для объекта document также определен аналогичный обработчик, но не через атрибут-событие, а путем его регистрации в сценарии. Если щелкнуть на кнопке (самом нижнем в иерархии элементе), то на щелчок последовательно отреагируют все элементы: сначала <button>, затем , <body> и, наконец, document. Если щелкнуть вне границ элемента , то сработают обработчики для <body> и document.

В данном примере для различных элементов документа определены обработчики одного и того же события onclick, которое может инициировать пользователь щелчком левой кнопкой мыши. Пользователь может щелкнуть на любом из этих элементов. Если он сделает это, то соответствующий элемент станет целевым и выполнится обработчик, привязанный к нему. Далее событие продолжит свое всплывание по иерархии объектов. Если на этом пути в каком-либо элементе оно встретит определенный для себя обработчик, то последний также будет выполнен, но во вторую очередь. Данный процесс продолжается аналогичным образом вплоть до достижения наивысшего объекта иерархии — document. Рассматриваемый пример выполняется почти одинаково во всех современных браузерах.

> ***Примечание***
>
> Интересно, что Firefox , в отличие от других браузеров, объект элемента <body> считает находящимся выше, чем document. В типичных ситуациях с целью обеспечения межбраузерной совместимости надежно считать <body> самым верхним элементом документа (в самом деле, <body> представляет основное содержимое документа). Иначе говоря, обработчики событий документа вообще, видимо, лучше привязывать к <body>, а не к document. Однако можно найти ситуации, когда различия между document и объектом элемента <body> весьма существенны. Чтобы лучше понять, о чем идет речь, попробуйте "пощелкать" внизу страницы в браузере, когда все элементы расположены в верхней ее части.

Листинг 14.12. Иллюстрация прохождения события по иерархии объектов

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Прохождение событий</title>
<script type="text/javascript">
   document.onclick=function(){alert('Щелчок на document')}
</script>
</head>
<body onclick="alert('Щелчок на body')">
  Щелкните где-нибудь
  <div onclick="alert('Щелчок на div ')"
      style="background-color:#00ffff; width:200px; height:100px">
    элемент div
```

```
    <button onclick="alert('Щелчок на button')"
            style="position:relative; top:50px; left:10px">
      Кнопка
    </button>
  </div>
</body>
</html>
```

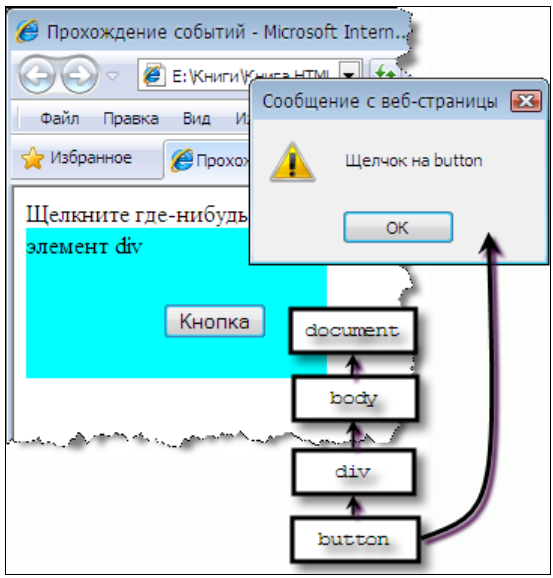


Рис. 14.11. Вид документа из листинга 14.11

Процесс всплытия события по иерархии объектов можно прекратить. Например, можно обработать щелчок на кнопке и прекратить дальнейшее распространение события `onclick`, чтобы содержащие кнопку элементы `<div>` и `<body>` не реагировали на него своими собственными обработчиками. Для этого имеется специальное свойство объекта `Event` с именем `cancelBubble` (прекратить пузырьки). Данное свойство не входит в стандарт DOM 2, но поддерживается Internet Explorer и браузерами на базе Mozilla.

По умолчанию свойство `cancelBubble` имеет значение `false`, т. е. всплывание события разрешено. Если в обработчике события присвоить данному свойству значение `true`, то по окончании обработки событие не всплывет и, следовательно, обработчики данного события для объектов верхних уровней не будут выполнены.

Листинг 14.13 аналогичен листингу 14.12 и отличается лишь тем, что в код обработчика события `onclick` для элемента `<div>` добавлено выражение

```
event.cancelBubble='true';
```

Если вы щелкнете на кнопке, то выполнится соответствующий обработчик, а затем и обработчик щелчка для элемента `<div>`. Однако вышестоящие по иерархии элемент `<body>` и объект `document` не отреагируют на щелчок своими обработчиками, потому что после `<div>` всплывание события прекратилось.

Листинг 14.13. Иллюстрация прекращения всплывания события

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Прохождение событий</title>
<script type="text/javascript">
   document.onclick=function(){alert('Щелчок на document')}
</script>
</head>
<body onclick="alert('Щелчок на body')">
  Щелкните где-нибудь
  <div onclick="alert('Щелчок на div '); event.cancelBubble='true'"
      style="background-color:#00ffff; width:200px; height:100px">
    элемент div
    <button onclick="alert('Щелчок на button')"
            style="position:relative; top:50px; left:10px">
      Кнопка
    </button>
  </div>
</body>
</html>
```

Модель событий DOM 2 предлагает еще и собственный метод `stopPropagation()` объекта `Event`, который, однако, неприменим в Internet Explorer. Применение данного метода в обработчике события для некоторого элемента означает, что после выполнения всех обработчиков событий, связанных с данным элементом, событие прекратит движение по иерархии объектов.

14.4.6. Информация о событии: объект *Event*

Объект событий `Event` предназначен для сохранения информации о произошедшем событии. Например, для щелчка в нем сохраняется ссылка на целевой объект, информация о координатах щелчка и др. При возникновении события браузер создает экземпляр объекта `Event`, который существует до тех пор, пока данное событие не будет обработано. Все экземпляры объектов инициированных событий обрабатываются в порядке их наступления.

Доступ к *Event*

`Event` является объектом браузера и доступен обработчикам событий по-разному в зависимости от типа браузера. В Internet Explorer экземпляр `Event` устанавливается как свойство объекта `window` с названием `event` и поэтому к нему можно обращаться как `window.event` или просто по имени `event` в любом сценарии. В браузерах, поддерживающих DOM 2 (в их числе Firefox, Opera, Safari и Chrome), доступ к объекту `Event` осуществляется в двух вариантах:

- в обработчиках, заданных через атрибуты-события, можно обращаться просто по имени event; в данном случае доступ к экземпляру `Event` формально одинаков во всех браузерах;

- обработчикам, привязанным с помощью методов DOM или путем установки соответствующих свойств в сценарии, экземпляр `Event` передается в виде параметра; в данном случае доступ к `Event` организуется иначе, чем в Internet Explorer.

Рассмотрим варианты доступа к экземпляру объекта `Event` более подробно.

В простейшем случае привязки обработчика через атрибут-событие во всех браузерах можно использовать обращение по имени `event`, например,

```
<body onclick="var x=event.screenX; alert('x='+x)">
```

Здесь код обработчика представлен непосредственно как значение атрибута-события `onclick`. При щелчке на элементе `<body>` в диалоговом окне появится сообщение о горизонтальной координате щелчка относительно экрана.

Привязанный посредством атрибута-события обработчик обычно определяют в виде функции. В этом случае экземпляр объекта события передают обработчику в качестве параметра как `event`. Соответствующий пример приведен в листинге 14.14. Данный код сработает во всех браузерах.

Листинг 14.14. Передача объекта `Event` обработчику (вариант 1)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Объект Event</title>
<script type="text/javascript">
  function myhandler(evt){  // обработчик щелчка
     alert("x="+evt.screenX)
  }
</script>
</head>
<body onclick="myhandler(event)">
Щелкните где-нибудь
</body>
</html>
```

В браузерах, поддерживающих модель событий DOM 2, при вызове обработчика ему всегда неявно передается соответствующий экземпляр объекта `Event`. Поэтому в таких браузерах сработает код, представленный в листинге 14.15. В данном примере используется регистрация обработчика в сценарии, а сам обработчик определен в виде функции с параметром `evt` (возможно любое имя переменной). Заметьте, что при регистрации обработчика `myhandler` явно не указывается, что он принимает параметр, но в определении функции-обработчика это делается (`function myhandler(evt) {...}`). Однако данный вариант не работает в Internet Explorer, даже если вы укажете в качестве параметра ключевое слово `event`.

Листинг 14.15. Передача объекта `Event` обработчику (вариант 2)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Объект Event</title></head>
<body>
Щелкните где-нибудь
</body>
<script type="text/javascript">
  function myhandler(evt){ // обработчик щелчка, не работает в IE
    alert("x="+evt.screenX)
  }
  document.body.onclick=myhandler // регистрация обработчика
</script>
</html>
```

Чтобы получить вариант, который работоспособен во всех браузерах, включая Internet Explorer, достаточно в начале кода обработчика `myhandler(evt)` вставить следующую строку:

```
if (!evt) {if(window.event) evt=window.event; else return true};
```

Здесь проверяется, не является ли переданное обработчику значение пустым. Если оно пусто (объект события не был передан в качестве параметра), то проверяется доступность свойства `event` объекта `window` (для Internet Explorer). Если оно доступно, то назначается переменной `evt`, и далее код обработчика оперирует этим значением, а в противном случае происходит выход из обработчика с возвратом значения `true`, чтобы не препятствовать выполнению обработчика по умолчанию, если он определен для данного элемента документа. Если вернуть `false`, то будет отменена обработка события по умолчанию.

Наиболее важные применения объекта `Event` в сценариях — отмена или изменение поведения элементов по умолчанию, а также предотвращение дальнейшего распространения события по иерархии объектов. Эти задачи рассмотрены в *разд. 14.4.3* и *14.4.5* соответственно.

Кто является целевым объектом?

Нередко возникают задачи, когда обработчик должен выполнять те или иные действия в зависимости от целевого объекта. В таких случаях один обработчик, привязанный к одному элементу верхнего уровня, может обрабатывать событие для дочерних нижестоящих элементов, предварительно выяснив, какой из них является целевым. При этом поведение обработчика также может быть задано в зависимости от целевого объекта. Например, вы можете привязать обработчик щелчка к элементу `<body>`, а чувствительными к щелчку неким особым образом сделать все или только некоторые элементы документа.

Для определения целевого объекта в модели событий DOM 2 предусмотрено свойство `target` объекта `Event`, а в Internet Explorer — `srcElement`. Поскольку эти ссылки на целевой объект различны, в обработчике необходимо предусмотреть адаптацию к браузеру.

В следующем примере (листинг 14.16) документ содержит абзац, ссылку и кнопку. В сценарии зарегистрирован обработчик `myhandler` щелчка, привязанный к элементу `<body>`. В обработчике, которому передается экземпляр объекта `Event`, определяется, какой элемент принял щелчок в качестве целевого объекта, и в диалоговом окне выво-

дится значение его атрибута id. Последнее делается лишь для демонстрации того, что мы правильно определили целевой объект. На практике вместо этого можно использовать операторы условного перехода с блоками кода, определяемыми для различных целевых объектов. Данный код сработает в любом браузере, поскольку в нем учтены особенности как передачи экземпляра объекта Event, так и обращения к целевому объекту.

Листинг 14.16. Определение целевого объекта в обработчике

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Объект Event</title></head>
<body id="mybody">
<p id="mytext">Щелкните где-нибудь</p>
<a id="myref" href="http://www.yandex.ru">Яндекс</a>
<button id="mybutton">Кнопка</button>
</body>
<script type="text/javascript">
function myhandler(evt){ // обработчик
  /* Для IE: */
  if (!evt) {if(window.event) evt=window.event; else return true};
  var elem; // переменная для целевого объекта
  if(evt.srcElement) elem=evt.srcElement // для IE
  else elem=evt.target;                  // для других браузеров
  /* Вместо alert() может быть и другой код */
  alert("Вы щелкнули на элементе id="+elem.id)
}
document.body.onclick=myhandler // регистрация обработчика щелчков
</script>
</html>
```

Обратите внимание, что если в листинге 14.16 вместо регистрации обработчика в сценарии реализовать привязку к <body> с помощью атрибута onclick="myhandler(event)", то в функции myhandler(evt) можно применить более короткий код для адаптации к браузеру:

```
var elem =evt.srcElement;  // для IE
if(!elem) elem=evt.target; // для других браузеров
```

Какое событие произошло?

В предыдущем подразделе мы рассмотрели создание одного обработчика, который может по-разному реагировать на одно и то же событие в зависимости от целевого объекта. Аналогично можно создать один обработчик для различных событий. Например, вы можете к одному элементу для различных событий привязать один и тот же обработчик, поведение которого определяется типом события.

Тип события в моделях DOM 2 и Internet Explorer представляется свойством type объекта Event. Это строка, содержащая название события, но без префикса "on", например, click, mouseover, mouseout и т. п.

В листинге 14.17 для абзаца и событий `onclick`, `onmouseover` и `onmouseout` зарегистрирован один и тот же обработчик `myhandler`. В данном примере обработчик просто выводит в диалоговое окно сообщение о типе произошедшего события.

Листинг 14.17. Определение типа события в обработчике

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Объект Event</title></head>
<body>
<p id="mytext">Щелкните где-нибудь</p>
</body>
<script type="text/javascript">
function myhandler(evt){ // обработчик
  /* Для IE: */
  if (!evt) {if(window.event) evt=window.event; else return true};
  /* Вместо alert() может быть и другой код */
  alert("Произошло событие "+evt.type)
}
/* Регистрация обработчиков событий */
document.getElementById("mytext").onclick=myhandler;
document.getElementById("mytext").onmouseover=myhandler;
document.getElementById("mytext").onmouseout=myhandler
</script>
</html>
```

Основные свойства объекта *Event*

Как уже отмечалось, объект `Event` содержит информацию о произошедшем событии, которая может использоваться при его обработке. Некоторые из важнейших свойств (например, `target`, `srcElement`, `type`) уже были рассмотрены в предыдущих подразделах. Список всех свойств объекта `Event` существенно зависит от браузера. Далее будут указаны основные свойства, которые поддерживаются всеми ведущими браузерами, и лишь некоторые специфические свойства.

Важно понимать, что не для каждого события в экземпляре объекта `Event` определяются все свойства. Например, экземпляр `Event`, соответствующий событиям клавиатуры, не имеет свойств с информацией о положении указателя мыши.

В табл. 14.2 приведены основные свойства объекта `Event`, которые доступны во всех наиболее популярных браузерах.

Таблица 14.2. *Основные свойства объекта* `Event`

Свойство	Описание
`altKey`	Логическое значение, указывающее, была ли во время события нажата клавиша <Alt>
`ctrlKey`	Логическое значение, указывающее, была ли во время события нажата клавиша <Ctrl>

Таблица 14.2 *(окончание)*

Свойство	Описание
`shiftKey`	Логическое значение, указывающее, была ли во время события нажата клавиша <Shift>
`keyCode`	Целое число, представляющее значение Unicode соответствующей клавиши
`button`	Целое число, указывающее, какая кнопка мыши была нажата в момент события. Наборы возможных значений отличаются в IE и других браузерах
`cancelBubble`	Логическое значение, указывающее, должно ли событие прекратить дальнейшее "всплывание" в иерархии объектов после его обработки текущим обработчиком
`clientX, clienY`	Координаты соответственно `x` и `y` указателя мыши в пикселах относительно клиентской области окна браузера
`screenX, screenY`	Координаты позиции, в которой произошло событие (в пикселах относительно всего экрана)
`srcElement`	Ссылка на объект элемента, являющегося целевым (IE4+). Для других браузеров следует использовать свойство `target`
`target`	Ссылка на целевой объект элемента, в котором произошло событие. Для IE следует применить свойство `srcElement`
`type`	Строка, указывающая тип события (например, "click")

В браузерах на базе Mozilla и поддерживающих модель событий DOM 2 объект `Event` имеет дополнительные свойства, среди которых укажем лишь несколько:

- `bubbles` — логическое значение, указывающее, всплывает ли событие;
- `cancelable` — логическое значение, указывающее, можно ли отменить событие;
- `detail` — количество щелчков кнопкой мыши;
- `timeStamp` — время наступления события в миллисекундах от начала эпохи (в Chrome это строка, содержащая дату и время, в Opera — 0);
- `view` — ссылка, указывающая на окно или фрейм, содержащий объект, в котором произошло событие;
- `eventPhase` — число, указывающее фазу, в которой в настоящий момент находится данное событие (1 — фаза захвата или спуска, 2 — в целевом объекте, 3 — фаза всплывания);
- `which` — число, указывающее для событий мыши использовавшуюся кнопку (1 — левая, 2 — средняя и 3 — правая), а для событий клавиатуры это значение Unicode нажатой клавиши.

Вот несколько свойств объекта `Event`, поддерживаемых Internet Explorer:

- `offsetX`, `offsetY` — координаты `x` и `y` в пикселах указателя мыши относительно целевого объекта (поддерживается также Opera, Safari и Chrome);
- `fromElement` — ссылка на объект, из которого исходит событие мыши (поддерживается также Safari и Chrome);
- `toElement` — ссылка на объект, в направлении которого перемещается указатель мыши (поддерживается также Safari и Chrome);

- `propertyName` — строка, содержащая имя свойства, вызвавшего событие `onpropertychange` (изменение значения свойства);
- `repeat` — логическое значение, указывающее, была ли клавиша в режиме непрерывного повтора в момент события `onkeydown` (нажатие на клавишу).

В листинге 14.18 приведен код, который выводит в окно браузера список свойств экземпляра объекта `Event`, создаваемого при щелчке мышью или нажатии на клавишу. Список свойств зависит как от произошедшего события, так и от браузера. Однако в Internet Explorer 8 приведенный скрипт не работает.

Листинг 14.18. Вывод списка свойств экземпляра объекта `Event`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Объект Event</title></head>
<body>
<p>Щелкните где-нибудь или нажмите на какую-нибудь клавишу</p>
</body>
<script type="text/javascript">
function myhandler(evt){ // обработчик
  /* Для IE: */
  if (!evt) {if(window.event) evt=window.event; else return true};
  var prop,i;
  for (prop in evt){
   document.write("<b>",prop,":</b> " ,evt[prop],"<br/>")
  }
}
/* Здесь можно использовать регистрацию обработчика для других событий */
document.body.onclick=myhandler  /* регистрация обработчика щелчков */
document.body.onkeydown=myhandler;/* регистрация обработчика нажатий
                                     на клавиши */
</script>
</html>
```

14.4.7. Основные события

Списки событий, поддерживаемых различными браузерами, огромны и имеют много отличий. Далеко не все события, предусмотренные в Internet Explorer, поддерживаются другими браузерами и входят в модель DOM 2. Тем не менее можно выделить группу событий, поддерживаемую основными современными браузерами примерно одинаковым образом. Для решения большинства задач создания Web-сайтов таких событий вполне достаточно.

При создании обработчика того или иного события следует обращать внимание на то, в каких объектах оно может возникать, может ли оно всплывать и отменяться.

В табл. 14.3 приведены сведения об основных событиях, поддерживаемых ведущими браузерами.

Таблица 14.3. *Основные события*

Событие	Описание события	Объекты
onblur	Потеря элементом фокуса ввода, когда пользователь активизирует другой элемент, например, щелчком или с помощью клавиши табуляции. Всплывание — нет. Отмена — нет	window, <body>, <a>, <area>, <button>, <input>, <label>, <select>, <textarea>, <applet>, , <embed>, <hr>, <img>, <marquee>, <object>, , <table>, <td>, <tr>
onchange	Потеря фокуса ввода полем формы, когда его значение было изменено. Иначе говоря, изменение содержимого поля завершено. Всплывание — да. Отмена — нет	<input>, <select>, <textarea>
onclick	Щелчок на элементе. Всплывание — да. Отмена — дат	Большинство элементов
ondblclick	Двойной щелчок на элементе. Всплывание — да. Отмена — да	Большинство элементов
onfocus	Получение фокуса ввода элементом. Всплывание — нет. Отмена — нет	window, <body>, <a>, <area>, <button>, <input>, <label>, <select>, <textarea>
onkeydown	Нажатие на клавишу, когда элемент имеет фокус ввода. Всплывание — да. Отмена — да	Большинство элементов
onkeypress	Нажатие на клавишу и освобождение ее, когда элемент имеет фокус ввода. Всплывание — да. Отмена — да	Большинство элементов
onkeyup	Освобождение ранее нажатой клавиши, когда элемент имеет фокус ввода. Всплывание — да. Отмена — нет	Большинство элементов
onload	Окончание загрузки объекта (обычно окна или фрейма) в браузер. Всплывание — нет. Отмена — нет	window,<body>, <frameset>
onmousedown	Нажатие на кнопку мыши, когда элемент имеет фокус ввода. Всплывание — да. Отмена — да	Большинство элементов

Таблица 14.3 *(окончание)*

Событие	Описание события	Объекты
onmousemove	Перемещение указателя мыши над элементом. Всплывание — да. Отмена — да	Большинство элементов
onmouseout	Смещение указателя мыши за пределы элемента. Всплывание — да. Отмена — да	Большинство элементов
onmouseover	Перемещение указателя мыши на элемент. Всплывание — да. Отмена — да	Большинство элементов
onmouseup	Освобождение ранее нажатой кнопки мыши, когда элемент имеет фокус ввода. Всплывание — да. Отмена — да	Большинство элементов
onreset	Возвращение формы в исходное состояние (например, нажатием на кнопку типа reset). Всплывание — да. Отмена — нет	<form>
onresize	Изменение размеров окна. Всплывание — да. Отмена — нет	window, <body>
onscroll	Прокрутка окна. Всплывание — да. Отмена — нет	window, <body>
onsubmit	Отправка данных формы на сервер (например, нажатием на кнопку типа submit). Всплывание — да. Отмена — да	<form>
onselect	Выбор пользователем текста (обычно выделением его с помощью мыши). Всплывание — да. Отмена — нет	<input>, <textarea>
unload	Выгрузка текущего документа из окна или фрейма. Всплывание — нет. Отмена — нет	window, <body>, <frameset>

Как уже отмечалось в *разд. 14.4.1*, привязка обработчиков событий может производиться с помощью атрибутов-событий, регистрацией обработчиков в сценарии и применением методов DOM. Обработчики событий onload, onunload, onresize и onscroll

можно привязать к элементу <body> любым из перечисленных способов или к объекту window окна браузера путем регистрации, например,

```
window.onscroll=myhandler;
```

Вместе с тем, для событий onresize и onscroll более надежной представляется привязка к window путем регистрации.

Кроме перечисленных в табл. 14.3, браузеры поддерживают и другие события. Например, Internet Explorer 5+ поддерживает, среди прочих, еще и такие полезные события:

- oncopy — попытка копирования выделенного содержимого объекта в буфер обмена;
- oncut — попытка вырезания выделенного содержимого объекта в буфер обмена;
- onpaste — попытка вставки содержимого буфера обмена в документ;
- ondrag — существует во время перетаскивания объекта мышью;
- ondragend — возникает по завершении перетаскивания объекта при отпускании кнопки мыши;
- ondragstart — возникает при начале перетаскивания выделенной области.

Последние три события поддерживают также и другие браузеры, но по-разному. Так, Firefox и Opera не поддерживают в объекте Event свойства, связанные с координатами указателя мыши.

14.5. Основные объекты браузера и документа

Данный раздел посвящен основным свойствам важнейших объектов браузера и загруженного в него документа: window, screen, location, history, navigator и document. Разумеется, это далеко не полный список объектов, но они наиболее важны с практической точки зрения, поскольку находятся на самых верхних уровнях иерархии объектной модели. Приводимые списки свойств и методов поддерживаются всеми браузерами. Тем не менее браузеры могут поддерживать и другие, свои фирменные свойства и методы.

Получить список всех свойств того или иного объекта, поддерживаемых вашим браузером, можно с помощью сценария вида:

```
for (var prop in объект){
   document.write("<b>",prop,":</b> " , объект[prop],"<br/>")}
```

Здесь prop — просто переменная, принимающая значения свойства интересующего нас объекта *объект* (см. *разд. 14.10.3*).

Например, для вывода в окно браузера свойств объекта window достаточно загрузить (X)HTML-документ, содержащий такой скрипт:

```
for (var prop in window){
   document.write("<b>",prop,":</b> " , window[prop],"<br/>")}
```

14.5.1. Объект *window*

window (окно) — объект высшего уровня, представляющий свойства окна браузера или фрейма. Фреймы, если они есть, сами являются объектами window. Доступ к фрейму из

главного окна осуществляется через коллекцию `window.frames[]` по его имени (атрибут `name`) или индексу.

Поскольку объект `window` всегда рассматривается в контексте JavaScript клиента, его свойства и методы доступны без указания префикса `window`. Например, вместо `window.alert("Привет!")` можно писать короче: `alert("Привет!")`. Однако в обработчиках событий данное правило может иметь исключения, поскольку любое свойство объекта `document` с тем же именем, что и у объекта `window`, маскирует соответствующее свойство `window`.

Свойства объекта *window*

Основные свойства объекта `window`:

- `parent` — возвращает ссылку на окно (фрейм), являющееся родительским для текущего окна;
- `self` — возвращает ссылку на текущее окно;
- `top` — возвращает ссылку на главное окно;
- `name` — название окна (значение атрибута `name` окна или фрейма);
- `opener` — ссылка на окно, в котором было создано текущее окно;
- `closed` — логическое значение, указывающее, закрыто ли окно;
- `status` — текст, отображаемый в строке состояния браузера в текущий момент времени;
- `defaultStatus` — текст по умолчанию в строке состояния браузера;
- `returnValue` — позволяет определить возвращенное значение для события или диалогового окна (поддерживается только Internet Explorer 4+);
- `document` — ссылка только для чтения на объект `document` документа, загруженного в данное окно;
- `history` — ссылка только для чтения на объект окна `history` (см. *разд. 14.5.4*);
- `location` — ссылка только для чтения на объект окна `location` (см. *разд. 14.5.3*);
- `navigator` — ссылка только для чтения на объект окна `navigator` (см. *разд. 14.5.5*);
- `screen` — ссылка только для чтения на глобальный объект `screen` (см. *разд. 14.5.2*);
- `frames[]` — коллекция фреймов, содержащихся в окне;
- `length` — количество фреймов в текущем окне.

Браузер Internet Explorer 4+ поддерживает дополнительные свойства, из которых мы отметим два наиболее важных:

- `event` — ссылка только для чтения на глобальный объект `event`, содержащий информацию о событии;
- `returnValue` — логическое значение, которое будет возвращено окном, созданным методом `showModalDialog()`.

Свойство `parent` позволяет обратиться к объекту, находящемуся в иерархии на одну ступень выше, например, к окну, содержащему коллекцию фреймов, в которой нахо-

дится наш фрейм. Для перемещения на две ступени вверх мы должны записать `parent.parent` и т. д.:

```
parent.window.frames[0]
parent.parent.window.frames[0]
```

Чтобы обратиться к самому главному окну, т. е. к окну браузера, следует использовать свойство `top`. Однако `top` не может обращаться к главному фрейму вашей системы разбиения на фреймы.

Свойство `name` соответствует имени фрейма, которое мы задаем в теге `<frameset>`.

Свойство `status` полезно для вывода сообщений во время работы сценария, например, при отладке:

```
window.status = "Сейчас работает сценарий"
```

Методы объекта *window*

Основные методы объекта `window`:

- `open()` — открывает новое окно (или вкладку) браузера;
- `close()` — закрывает текущее окно браузера;
- `alert()` — показывает окно предупреждения с сообщением и кнопкой **OK**;
- `prompt()` — показывает окно приглашения (запроса) с сообщением, текстовым полем и кнопками **OK** и **Отмена** (Cancel);
- `confirm()` — показывает окно подтверждения с сообщением и кнопками **OK** и **Отмена** (Cancel);
- `navigate()` — загружает в данное окно страницу с указанным адресом;
- `blur()` — убирает фокус с текущей страницы; соответствующее событие — `onblur`;
- `focus()` — устанавливает окно в фокус; соответствующее событие — `onfocus`;
- `scroll(x,y)` — прокручивает содержимое окна так, чтобы позиция, указанная в параметрах `x` и `y`, переместилась в верхний левый угол окна (не рекомендуется использовать);
- `scrollBy(x,y)` — прокручивает содержимое окна на указанные в параметрах количества пикселов по горизонтали и вертикали;
- `scrollTo(x,y)` — прокручивает содержимое окна в позицию, указанную в параметрах;
- `moveBy(x,y)` — смещает окно на указанные в параметрах количества пикселов по горизонтали и вертикали;
- `moveTo(x,y)` — помещает окно в указанные в параметрах координаты по горизонтали и вертикали (в пикселах);
- `resizeBy(ширина,высота)` — изменяет размеры окна на указанные в параметрах количества пикселов (отрицательные значения соответствуют сжатию);
- `resizeTo(ширина,высота)` — устанавливает размеры окна в соответствии с указанными в параметрах значениями пикселов;

- `setInterval(код, период)` — запускает код (функцию) периодически через заданный интервал (в миллисекундах);
- `setTimeout(код, задержка)` — запускает код (функцию) через заданный интервал (в миллисекундах) после вызова данного метода;
- `clearInterval()` — обнуляет таймер, заданный методом `setInterval()`;
- `clearTimeout()` — обнуляет таймер, заданный методом `setTimeout()`;
- `print()` — открывает диалоговое окно печати.

Браузер Internet Explorer 4+ поддерживает дополнительные методы, из которых мы отметим четыре наиболее важных:

- `showHelp()` — показывает окно подсказки как диалоговое;
- `showModalDialog()` — показывает новое окно как диалоговое модальное (т. е. остающееся впереди других, пока не будет закрыто);
- `showModalessDialog()` — показывает новое окно как диалоговое немодальное;
- `execScript(выражение, язык)` — выполняет код сценария на указанном языке (по умолчанию JScript в Internet Explorer).

14.5.2. Объект *screen*

Объект `screen` (экран) содержит информацию о возможностях дисплея пользователя и может применяться, например, для определения размеров создаваемых окон и страниц, а также выбора характеристик передаваемой графики. Адаптируемый к дисплею пользователя дизайн сайта делается на основе анализа свойств объекта `screen`. Данный объект является дочерним по отношению к `window`. Далее приведены свойства `screen`, которые одновременно поддерживают Internet Explorer, Firefox, Opera, Safari и Chrome:

- `availWidth` — возвращает ширину экрана пользователя в пикселах за вычетом элементов оформления окна (таких как панели инструментов Windows), создаваемых операционной системой;
- `availHeight` — возвращает высоту экрана пользователя в пикселах за вычетом элементов оформления окна, создаваемых операционной системой;
- `width` — возвращает ширину экрана пользователя (в пикселах);
- `height` — возвращает высоту экрана пользователя (в пикселах);
- `colorDepth` — возвращает битовую глубину цвета (количество битов на пиксел для хранения цветовой информации.

Методов у объекта `screen` нет.

14.5.3. Объект *location*

Объект `location` (расположение) содержит информацию об URL-адресе в целом и отдельных его компонентах для текущего документа, а также методы, позволяющие обновлять страницы. Данный объект дочерний по отношению к `window`.

Свойства объекта *location*

Далее приведен список свойств объекта `location`:

- `href` — полный URL-адрес в виде строки;
- `hash` — строка, следующая в URL за символом `#`, с помощью которого указывается, к какому якорю внутренней ссылки следует переместиться при загрузке документа;
- `host` — часть URL "*хост*:*порт*"; значение порта содержится лишь тогда, когда оно явно было указано в URL-адресе;
- `hostname` — имя хоста или, иначе, сетевое имя сервера; для большинства Web-сайтов имя сервера включает в себя не только имя домена, но и префикс `www`, но в имени хоста номер порта не содержится, даже если он определен в URL;
- `pathname` — часть URL, соответствующая пути к объекту или файлу, всегда содержит по меньшей мере `/`;
- `port` — номер порта в URL;
- `protocol` — начальная часть, определяющая протокол, за которой следует двоеточие, например, "`http:`";
- `search` — строка запроса или данные URL, следующие после имени файла (включая разделительный символ `?`).

Например, если вы загрузили страницу с адресом **http://www.cityline.spb.ru**, то значением `location.href` будет эта строка.

Присваивая свойству `href` новое значение, мы можем изменять показываемую в браузере страницу, например:

```
window.location.href="http://www.rambler.ru"
```

Методы объекта *location*

Перечислим методы объекта `location`:

- `assign("`*URL*`")` — загружает документ с указанным URL-адресом; этот метод эквивалентен изменению свойства `window.location.href` с помощью оператора присваивания;
- `reload(`*флаг*`)` — обновляет (перезагружает) текущий документ; если логический параметр *флаг* имеет значение `true`, то загрузка должна произойти не из кэша;
- `replace("`*URL*`")` — загружает документ с указанным в параметре URL-адресом и заменяет URL-адрес текущего документа в журнале браузера (эквивалентен `location.href`).

Некоторые Web-сайты могут содержать страницы, которые не предназначены для посещения пользователем и занесения их в список посещенных страниц. Например, перемещение по сайту может привести пользователя на некоторые промежуточные страницы, которые ему больше не понадобятся. При этом желательно, чтобы пользователь не мог вернуться к ним снова с помощью кнопки **Назад** (Back). В этом случае применяется метод `location.replace("`*URL*`")` для перехода к указанной странице без занесения его в список документов, переход к которым осуществляется щелчком на этой кнопке.

14.5.4. Объект *history*

Объект `history` (история) содержит информацию об адресах страниц, которые браузер отобразил во время текущего сеанса. Вы можете передвигаться по этому списку с помощью сценария и загружать соответствующие страницы, имитируя щелчки на кнопках **Назад** и **Вперед**. Данный объект дочерний по отношению к `window`.

Свойства объекта *history*

Свойство `length` — количество элементов в списке посещенных страниц. В сценарии это свойство позволяет проконтролировать выход за пределы списка (массива). Другие свойства, такие как `current`, `next` и `previous` (соответственно текущий, следующий и предыдущий URL-адрес журнала), для доступа требуют привилегии `UniversalBrowserRead`. Здесь мы их не будем рассматривать.

Методы объекта *history*

Методы таковы:

- `back()` — загружает предыдущую страницу из списка журнала;
- `forward()` — загружает следующую страницу из списка журнала;
- `go()` — загружает страницу с относительным номером `n` из списка (от нуля по `history.length - 1`) или с указанным URL-адресом.

В следующем примере сначала загружается первая страница, а затем — пятая, причем предварительно проверяется, существует ли она в списке уже посещенных страниц:

```
window.history.go(1)
if (window.history.length>4)
   window.history.go(5)
```

Очевидно, метод `history.go(-1)` эквивалентен `history.back()`, а `window.history.go(1)` — методу `history.forward()`.

14.5.5. Объект *navigator*

Объект `navigator` (навигатор) содержит информацию о производителе браузера, его версии и возможностях, используемом языке, а также об операционной системе. Состав свойств данного объекта зависит от браузера, и здесь приводятся лишь некоторые из них, которые поддерживаются всеми основными браузерами. Данный объект дочерний по отношению к `window`.

Свойства объекта *navigator*

Далее перечислены свойства объекта `navigator`:

- `appCodeName` — название кода браузера; например, для Internet Explorer, Firefox, Opera, Safari и Chrome данное свойство имеет значение "Mozilla";
- `appName` — название браузера; например, Internet Explorer показывает значение данного параметра как "Microsoft Internet Explorer", Opera — как "Opera", а Firefox, Safari и Chrome устанавливают его как "Netscape"; таким образом, данный параметр не подходит для идентификации браузера пользователя;

- appVersion — версия браузера, например, "5.0 (Windows; ru)";
- cookieEnabled — логическое значение; определяет возможность использования в браузере cookies со стороны клиента;
- userAgent — название браузера, посылаемое им на сервер с помощью HTTP-протокола; например, "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.9.2) Gecko/20100115 MRA 5.5 (build 02842) AdCentriaIM/1.7 Firefox/3.6";
- platform — информация об операционной системе, для которой скомпилирован браузер, например "Win32", "MacPPC", "SunOS".

Для хранения сведений о языке браузера предусмотрено свойство languge, поддерживаемое Firefox, Opera, Safari и Chrome, но не Internet Explorer 8. В последнем есть свойства systemLanguge (язык операционной системы клиента) и userLanguage (язык пользователя, определенный настройками операционной системы).

Иногда требуется установить, каков браузер пользователя. Например, это может понадобиться, если вы решили создать версии своего документа, адаптированные к различным браузерам. Для выделения браузера Internet Explorer среди других достаточно воспользоваться свойством appName:

```
if (navigator.appName=="Microsoft Internet Explorer")
      location.replace("mypage1.htm")
else
      location.replace("mypage2.htm");
```

Для выделения любого браузера из заданного множества и определения его краткого названия (например, Firefox, Opera и т. д.) лучше воспользоваться свойством userAgent, а не appName. Однако следует иметь в виду, что ни формат, ни собственно содержание значения свойства userAgent не стандартизированы, а поэтому необходимо проанализировать реальные значения для различных браузеров и их версий прежде, чем написать соответствующий сценарий. В листинге 14.19 приведен один из возможных вариантов сценария, устанавливающего и сохраняющего в переменной краткое название одного из пяти наиболее популярных браузеров пользователя.

Листинг 14.19. Определение названия браузера

```
if (navigator.userAgent.indexOf("Opera")!=-1)
  brName="Opera";
else if (navigator.userAgent.indexOf("Chrome") !=-1)
     brName="Chrome";
   else if (navigator.userAgent.indexOf("Safari") !=-1)
        brName="Safari";
      else if (navigator.userAgent.indexOf("Firefox") !=-1)
         brName="Firefox";
       else if (navigator.userAgent.indexOf("MSIE") !=-1)
            brName="Internet Explorer";
          else brName="Неизвестный";
```

Здесь с помощью метода indexOf() определяется, содержится ли название браузера в значении свойства userAgent. Значение свойства userAgent как для Apple Safari, так и

для Google Chrome содержит слово "Safari". Поэтому в приведенном сценарии важно, что Chrome проверяется раньше, чем Safari.

Коллекции объекта *navigator*

Перечислим коллекции объекта `navigator`:

- `mimeTypes` — коллекция всех типов документов и файлов, поддерживаемых браузером;
- `plugins` — всех внедряемых объектов на странице.

Методы объекта *navigator*

Методы объекта `navigator` таковы:

- `taintEnabled()` — возвращает значение `false`; метод включен для совместимости с Netscape Navigator;
- `javaEnabled()` — логическое значение; сообщает, возможен ли в данном браузере запуск кода сценария на языке JavaScript.

14.5.6. Объект *document*

Объект `document` (документ) — центральный в иерархической объектной модели, он представляет всю информацию о загруженном в браузер (X)HTML-документе с помощью коллекций и свойств.

Свойства объекта *document*

Основные свойства объекта `document`:

- `alinkColor` — цвет активных ссылок;
- `bgColor` — цвет фона документа;
- `body` — ссылка только для чтения на неявный основной объект документа, определенный тегом `<body>` или `<frameset>`;
- `cookie` — строка cookie-записи. Присвоение нового значения этому свойству приводит к записи cookie на диск после закрытия браузера;
- `dir` — строка, указывающая направление обработки текста в документе;
- `doctype` — ссылка на объект `DocumentType` документа;
- `documentElement` — ссылка на объект корневого узла документа в иерархии объектов;
- `domain` — строка, содержащая имя домена, откуда был получен документ;
- `fgColor` — цвет текста;
- `height` — высота в пикселах содержимого документа, включая части вне видимости (поддерживается Firefox, Safari, Chrome);
- `implementation` — объект с методом `hasFeature(`*`возможность`*`, `*`уровень`*`)`, возвращающим логическое значение, которое указывает, поддерживает ли браузер указанную возможность на указанном уровне DOM; допустимые значения параметра: `CSS`, `Events`,

HTML, HTMLEvents, MouseEvents, Range, StyleSheets, Views, XML; параметр уровень значения 1.0, 2.0 и т. п.;

- lastModified — строка только для чтения, содержащая дату последнего изменения документа;
- linkColor — цвет еще не посещенных гиперссылок;
- readyState — определяет текущее состояние загружаемого объекта;
- referrer — строка только для чтения, содержащая URL-адрес документа, который ссылается на данный. Если документ не получен с сервера (загружен непосредственно), то значение данного свойства не устанавливается;
- selection — ссылка только для чтения на дочерний для document объект selection, соответствующий выделенному в текущий момент тексту (поддерживается только Internet Explorer 4+ и Opera);
- title — содержимое тега <title>;
- vlinkColor — цвет посещенных ссылок;
- width — ширина в пикселах содержимого документа, включая части вне видимости (поддерживается Firefox, Safari, Chrome).

Коллекции объекта *document*

Перечислим некоторые основные коллекции объекта document:

- all — коллекция всех элементов в основной части документа; это нестандартная коллекция, поддерживаемая Internet Explorer, Opera, Safari; в *разд. 14.2* рассматривается, как обойтись без коллекции all;
- anchors — коллекция "якорей" (закладок, анкеров) в документе, т. е. элементов, задаваемых тегами вида ...;
- applets — коллекция объектов в документе, включая встроенные элементы управления, графические элементы, апплеты, внедренные и другие объекты;
- embeds — коллекция внедренных объектов в документе;
- forms — коллекция форм на странице;
- frames — коллекция фреймов, определенных в теге <frameset>;
- images — коллекция графических элементов (изображений) на странице;
- links — коллекция ссылок и блоков <area> на странице;
- styleSheets — коллекция объектов styleSheets в документе (элементов <style>).

Методы объекта *document*

Приведем список основных методов объекта document:

- clear() — очищает выделенный участок;
- close() — завершает выходной поток документа;
- getElementById() — возвращает элемент со значением атрибута id, указанным в параметре, или null, если такого элемента не существует;

- `getElementsByName()` — возвращает коллекцию элементов со значением атрибута `name`, указанным в параметре;
- `getElementsByTagName()` — возвращает коллекцию элементов, определяемых списком имен тегов, указанным в параметре; если значение параметра есть `*`, то возвращается коллекция всех элементов документа;
- `open()` — открывает документ для записи, очищая его, если имеется содержимое;
- `write(`*список строк*`)` — записывает в документ текст и код HTML, указанные в качестве одного или нескольких строковых параметров;
- `writeln(`*список строк*`)` — записывает в документ текст и код HTML, заканчивающийся символом перехода на новую строку.

ГЛАВА 15

Работа с основными объектами посредством JavaScript

15.1. Управление окнами и фреймами

15.1.1. Создание окон

Объект `window` представляет окно браузера или фрейм. С помощью сценария можно создавать и удалять окна, загружать в них документы и записывать данные.

Окно создается методом `open()` объекта `window`. Синтаксис:

```
window.open(URL-адрес, имя, опции, замена);
```

Префикс `window` можно не указывать.

Данный метод принимает параметры:

- *URL-адрес* — строка с URL-адресом документа, который следует загрузить в открываемое окно; если вы не хотите загружать в окно никакой документ, то укажите пустую строку `""`;
- *имя* — строка с именем окна, которое позже может стать значением атрибута `target` в тегах `<a>` ссылок; задается по тем же правилам, что и имя переменной; если вы не будете ссылаться на данное окно, то укажите пустую строку `""`;
- *опции* — строка, в которой через запятую указаны дополнительные параметры оформления окна (*см. далее*); можно не указывать данный и следующий параметры, если вы полагаетесь на параметры создаваемого окна, принятые по умолчанию; однако следует иметь в виду, что умолчания могут быть разными для различных браузеров;
- *замена* — необязательное логическое значение, указывающее, должен ли документ по заданному URL-адресу заменить содержимое окна (применимо к уже существующим окнам).

Метод `open()` возвращает ссылку на объект открытого окна (экземпляр `window`). Это идентификатор окна, который можно сохранить в переменной, чтобы затем использовать, например, для закрытия данного окна. Вот пример открытия окна:

```
mywindow=open("http://www.yandex.ru","Yandex");
```

Здесь `mywindow` — произвольная переменная, в которой сохраняется идентификатор открытого окна.

Метод `close()` позволяет закрыть окно. Синтаксис:

```
идентификатор_окна.close();
```

Здесь *идентификатор_окна* — ссылка на объект открытого ранее окна, возвращаемая методом `open()`.

Заметим, что вызов `window.close()` означает попытку закрыть главное окно, которая в зависимости от браузера может выполниться, заблокироваться или вызвать диалоговое окно подтверждения. Это связано с политикой безопасности JavaScript, которая может быть той или иной в зависимости от браузера. Так, кажется вполне разумным, что клиентский скрипт не может закрыть браузер, но некоторые браузеры допускают эту возможность.

Для корректного применения метода `close()` следует прежде убедиться, что окно существует и не закрыто (свойство `closed`):

```
if (mywindow&&!mywindow.closed) mywindow.close();
```

Если вы хотя бы однажды создали окно, но потом закрыли его, то оно продолжает существовать. Именно поэтому следует сначала проверить его существование и то, что оно не закрыто, а только затем выполнить метод `close()`, закрывающий существующее и незакрытое окно.

При создании окна методом `open()` с помощью третьего параметра можно указать в строке множество дополнительных параметров, разделенных запятыми. Например, вы можете задать размеры и позицию окна, наличие полос прокрутки и панели меню и др. Каждый дополнительный параметр задается как *имя=значение*, причем *значение* в кавычки не заключается, например, `"width=300, height=200, scrollbars=yes"`.

Среди довольно большого количества дополнительных параметров наиболее инвариантны относительно браузеров и в то же время популярны следующие:

- `width`, `height` — ширина и высота окна в пикселах;
- `screenX`, `screenY` (`left`, `top`) — горизонтальная и вертикальная координаты относительно левого верхнего угла экрана в пикселах; для Internet Explorer и Opera следует использовать только `left` и `top`, а для Firefox, Safari и Chrome допустимы и то, и другое;
- `menubar` — разрешение/запрет отображения панели меню; работает в Internet Explorer и Firefox, а в Opera, Safari и Chrome панель меню не отображается в любом случае;
- `toolbar` — разрешение/запрет отображения панели инструментов; работает в Internet Explorer и Firefox, а в Opera, Safari и Chrome панель инструментов не отображается в любом случае;
- `scrollbars` — разрешение/запрет отображения полос прокрутки; работает в Internet Explorer, а в Firefox, Opera, Safari и Chrome полосы прокрутки появляются при необходимости независимо от данного параметра;
- `resizable` — разрешение/запрет изменения размеров окна пользователем; работает в Internet Explorer, а в Firefox, Opera, Safari и Chrome изменение размеров возможно независимо от данного параметра.

Логические значения для данных параметров указывают как `yes` и `no` или как 1 и 0 соответственно. Например, записи `"menubar=yes"` и `"menubar=1"` эквивалентны. Вместе с тем, для обратной совместимости скриптов рекомендуется использовать `yes` и `no`.

После создания окна методом `open()` вы можете в сценарии установить для него любое из изменяемых свойств или применить любой метод объекта `window` (см. *разд. 14.5.1*). Допустим, вы создали окно с идентификатором `mywindow` и загрузили в него сайт **www.yandex.ru** следующим образом:

```
mywindow=open("http://www.yandex.ru",
              "Yandex","width=400, height=400")
```

Вот несколько примеров управления данным окном:

- `mywindow.moveBy(-10,20)` — сдвиг созданного окна с идентификатором `mywindow` на 10 px влево и на 20 px вниз относительно исходного позиционирования;
- `mywindow.moveTo(100,200)` — позиционирование созданного окна с идентификатором `mywindow` в точку 100 px по горизонтали и 200 px по вертикали относительно экрана;
- `mywindow.scrollBy(0,20)` — прокрутка содержимого окна на 20 px вниз;
- `mywindow.scrollTo(100,200)` — прокрутка содержимого окна к позиции 100 px и 200 px.

15.1.2. Взаимодействие окон

Исходное окно и окна, созданные из него, могут взаимодействовать друг с другом. Для этого необходимо обращаться к окну по идентификатору экземпляра объекта `window`. К главному окну можно обращаться, используя название `window` объекта окна или даже опуская его. Рассмотрим несколько примеров.

В созданное окно можно записать данные, используя метод `document.write()`, указывая в качестве префикса идентификатор окна, например,

```
mywindow.document.write("<h1>Привет!</h1>").
```

Код в листинге 15.1 приведен для иллюстрации возможности записи в создаваемое окно. В (X)HTML-документе, загружаемом в главное окно, имеется текстовая область и две кнопки (рис. 15.1). В текстовую область можно ввести как обычный текст, так и (X)HTML-код. Щелчок на кнопке **Записать** создает новое окно (если его еще нет) и записывает в него содержимое текстовой области. Щелчок на кнопке **Закрыть** закрывает новое окно. Если окно открыто, то при каждом щелчке на кнопке **Записать** содержимое текстовой области добавляется к содержимому в новом окне. Чтобы содержимое окна переписывалось заново, достаточно после каждой записи закрывать отображаемый в нем документ: `mywindow.document.close()`. В листинге 15.1 данное выражение отключено символом комментария.

Листинг 15.1. Запись в окно

```
<!DOCTYPE HTML
PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Запись в окно</title>
</head>
<body id="mybody">
```

```
<textarea id="mytext" rows=6 cols=20 value=""></textarea>
<div>
<button onclick="sendtext()">Записать</button>
<button onclick="closewin()">Закрыть </button>
</div>
</body>
<script type="text/javascript">
var mywindow;
function sendtext(){ // запись в окно
  if (!mywindow||mywindow.closed) // если не существует или закрыто
     mywindow=open('','Мое_окно',
     "width=200,height=200,top=100,left=200, scrollbars=yes");

     mywindow.document.write(document.getElementById("mytext").value);
//     mywindow.document.close(); // закрыть документ
     mywindow.focus(); // фокус на новое окно
}
function closewin(){ // закрыть окно
  if (mywindow&&!mywindow.closed)
      mywindow.close();
}
</script>
</html>
```

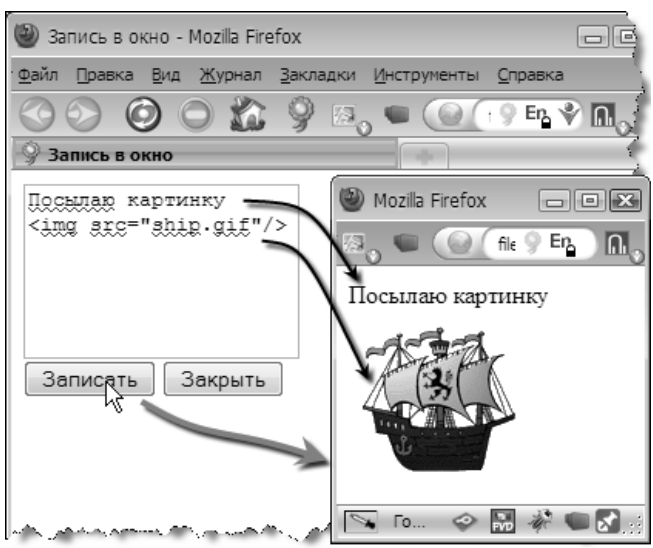


Рис. 15.1. Запись данных из главного окна в дочернее (листинг 15.1)

Нетрудно понять, что с помощью скрипта в документе главного окна можно сформировать (X)HTML-код и записать его в дочернее окно.

Из дочернего окна можно обратиться к родительскому, используя свойство `window.opener` для получения ссылки на объект `window`, создавший данное окно. Так,

если в предыдущем примере (листинг 15.1, рис. 15.1) в текстовую область в главном окне вы введете строку

```
<script>window.opener.document.write('Привет из нового окна')</script>
```

и щелкнете на кнопке **Записать**, то данный скрипт выполнится в новом окне и перезапишет содержимое главного окна, а именно выведет в нем текст "Привет из нового окна".

Иначе говоря, при обращении к содержимому родительского окна следует явно указывать последнее. Это можно сделать двояко, указав в качестве префикса `window.opener` или ключевое слово `parent` (родитель).

Скрипты в документе, загруженном в дочернее окно, могут изменять свойства элементов документа родительского или других дочерних окон. Чтобы обратиться к свойству элемента документа в родительском окне из дочернего, можно использовать такое выражение:

```
window.opener.document.getElementById(id).свойство;
```

Для обращения к созданному окну из сестринского окна следует сначала обратиться к родительскому, которое "знает" все свои дочерние окна:

```
window.opener.дочернее_окно.document.getElementById(id).свойство;
```

Обычно открытие дополнительных окон вызывает раздражение посетителей сайта. Не все такие окна бесполезны, однако не следует злоупотреблять их количеством. Политика безопасности JavaScript предполагает, что вы не можете получить доступ к окнам, которые не локальны относительно вашего сайта, но и другие сайты не могут иметь доступа к вашим окнам.

15.1.3. Работа с фреймами

Взаимодействие фреймов

Документ может содержать фреймы, созданные тегами `<iframe>`. В эти фреймы могут быть загружены документы, которые сами содержат фреймы, и тогда образуется более сложная иерархия фреймов.

При работе с фреймами посредством скриптов важно понять, что каждому фрейму соответствует свой экземпляр объекта `window`. Именно поэтому каждый фрейм ведет себя подобно окну браузера. Иерархии фреймов соответствует иерархия экземпляров объекта `window`.

При задании фреймовой структуры главное окно браузера получает статус главного фрейма, а все фреймы, заданные посредством тегов `<iframe>`, оказываются дочерними фреймами по отношению к неявно созданному главному фрейму.

Все фреймы имеют свои собственные объекты `document`. Из сценария, расположенного в документе одного фрейма, можно получить доступ к документу, загруженному в другой фрейм. При этом следует указать фрейм, доступ к содержимому которого требуется получить.

Простейший случай — применение ссылок в одном фрейме для загрузки запрашиваемых документов в другой фрейм. Теги `<iframe>`, задающие фреймы, имеют атрибут `name` (имя фрейма). Значение данного атрибута можно использовать в атрибуте `target` тега

ссылки, чтобы указать в какой фрейм следует загрузить запрашиваемый документ. При этом не важно, в каком из фреймов находится сама ссылка.

Другое дело, когда требуется получить доступ к фрейму из сценария, расположенного в другом фрейме. В данном случае следует учитывать иерархию фреймов. При этом возможны три основных типа обращения:

- ❒ из родительского фрейма к дочернему;
- ❒ из дочернего фрейма к родительскому;
- ❒ из дочернего к сестринскому.

Допустим, в документе (назовем его главным) находятся два фрейма с именами `frame1` и `frame2` (рис. 15.2). Главному окну браузера соответствует объект `window`, который является родительским фреймом для двух своих дочерних фреймов: `frame1` и `frame2`. В данном случае обращение из сценария в главном документе (т. е. из родительского фрейма) к дочернему происходит просто по имени последнего. Например, в сценарии, расположенном в главном документе, можно указать, какой документ следует загрузить в дочерний фрейм `frame1`:

```
frame1.location.href="http://www.anyserver.ru";
```

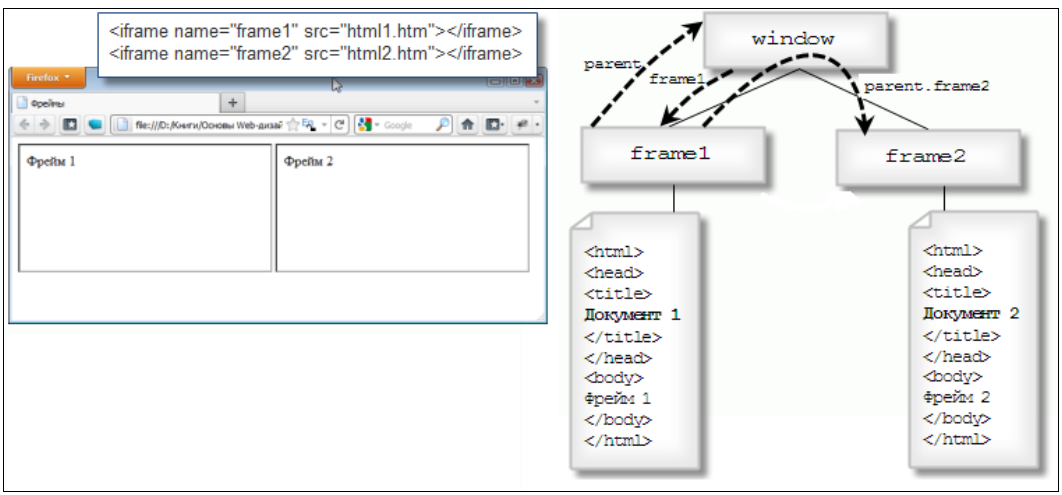


Рис. 15.2. Взаимодействие фреймов

Фрейм, созданный тегом `<iframe>`, является элементом документа и, следовательно, к нему можно обращаться еще и с помощью метода `document.getElementById(id)`.

Чтобы обратиться из дочернего фрейма к родительскому, следует указать ключевое слово `parent` (родитель) в качестве префикса. Например, если в дочернем фрейме выполнить выражение

```
parent.location.href="http://www.anyserver.ru";
```

то браузер попытается загрузить документ по указанному адресу в главное окно, вытеснив вашу фреймовую структуру.

Дочерний фрейм ничего "не знает" о своих сестрах. Поэтому, чтобы из него получить доступ к другому дочернему фрейму, необходимо это сделать через родительский

фрейм, который "знает" все о своих потомках. Для этого перед именем фрейма следует указать префикс `parent`. Например, в сценарии фрейма `frame1` инициировать загрузку документа в сестринский фрейм документа можно так:

```
parent.frame2.location.href="http://www.anyserver.ru";
```

Как уже отмечалось, каждый фрейм имеет свой объект `document`. Для получения доступа к его элементам необходимо перед словом `document` указать, к какому фрейму относится данный документ. Например, чтобы из сценария в первом фрейме изменить цвет фона документа во втором фрейме (с именем `frame2`), следует написать так:

```
parent.frame2.document.bgColor="blue";
```

Мы рассмотрели случай, когда фреймы являются дочерними по отношению к главному окну, т. е. прямыми потомками последнего. Однако можно представить себе ситуацию, когда во фрейм первого поколения загружается документ со своей собственной фреймовой структурой (рис. 15.3). Такие фреймы будут принадлежать ко второму поколению. Для получения доступа из фрейма первого поколения к фрейму второго поколения следует сначала подняться к общему родительскому элементу, а затем спуститься к целевому фрейму:

```
parent.имя_фрейма_поколения1.имя_фрейма_поколения2....
```

Для получения доступа, наоборот, из фрейма второго поколения к фрейму первого поколения также необходимо сначала подняться к общему родителю, но это достигается повторением слова `parent` дважды:

```
parent.parent.имя_фрейма_поколения1....
```

Здесь первое слово `parent` указывает на объект `window`, являющийся родителем фрейма второго поколения, а последовательность `parent.parent` — на объект `window` более высокого уровня, который является родителем для фреймов первого поколения и прародителем для фрейма второго поколения.

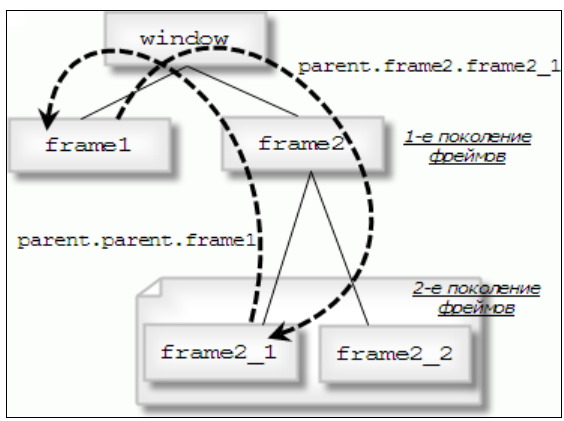


Рис. 15.3. Взаимодействие фреймов разных поколений

Как мы видели, для обращения к вышестоящим фреймам ключевое слово `parent` используется один или несколько раз в зависимости от того, насколько уровней выше находится интересующий нас фрейм. Сразу обратиться к фрейму, находящемуся на вершине иерархии, можно с помощью ключевого слова `top`. Однако при этом следует

учесть то обстоятельство, что ваш фреймовый документ может быть загружен во фрейм другого сайта. В таком случае указанный вами объект `top` окажется совсем иным, и ссылки с `top` не будут правильно работать. Поэтому для ссылок на вышестоящий фрейм (окно) рекомендуется `parent`.

Мы обращались к фреймам по их именам (значениям атрибута `name` тега `<iframe>`) непосредственно в цепочке объектов. Вместе с тем, возможно и применение коллекции `frames[]`, в которой фрейм указывается индексом или значением атрибута `name`, например,

```
parent.parent.frames["frame1"].document.bgColor="#00ffff"
```

Скрипты, обращающиеся к документам во фреймах, должны срабатывать не раньше, чем последние будут полностью загружены. В противном случае возникнет ошибка типа "Такой-то объект не определен". Во избежание этого следует предпринимать специальные меры. Например, можно помещать скрипты, воздействующие на содержимое других фреймов, в обработчики события `onload` или событий, инициированных пользователем (`onclick`, `onmouseover` и т. п.).

В листинге 15.2 приведен пример (X)HTML-кода документа с плавающим фреймом и тремя кнопками, щелчки на которых устанавливают содержимое фрейма, причем тремя различными способами. Так, Rambler загружается путем установки свойства `src` плавающего фрейма, к которому мы обратились как к элементу документа. В остальных двух случаях мы обращаемся к фрейму как к подобъекту `window`. Причем в первом случае мы прямо указали имя `myframe` фрейма, а во втором — воспользовались коллекцией `frames['myframe']`, указав в качестве аргумента имя фрейма. В данном примере значения атрибутов `id` и `name` тега `<iframe>` совпадают, однако в действительности только в первом случае (загрузка Rambler) используется `id`, а в остальных двух — `name`. Поскольку элемент `<iframe>` находится в документе главного окна браузера, префикс `window` в обращении к нему можно опустить. Если же скрипт должен воздействовать на плавающий фрейм, находящийся в другом фрейме (окне), необходимо выстроить к нему путь в виде цепочки объектов фреймов (окон).

Листинг 15.2. Обращение к плавающему фрейму

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Плавающий фрейм</title></head>
<body>
<iframe name="myframe" id="myframe"
        src="" width="400" height="300">
</iframe>
<div>
<input type="button" value="Rambler" onclick=
       "document.getElementById('myframe').src='http://www.rambler.ru'"/>

<input type="button" value="Google" onclick=
       "myframe.location.href='http://www.google.ru'"/>
```

```
<input type="button" value="Сам себе Web-дизайнер"  onclick=
       "frames['myframe'].location.href='http://dunaevv1.narod.ru'"/>
</div>
</body>
</script>
</html>
```

Предотвращение загрузки в чужой фрейм

Кроме ссылки `parent` на фрейм вышестоящего уровня имеются ссылки `top` и `self`. Ключевое слово `top` — ссылка на объект окна (фрейм) самого верхнего уровня иерархии, а `self` — на данный, текущий или собственный объект окна (фрейма). Сейчас мы рассмотрим возможные роли указанных ссылок на примере решения практической задачи.

Если вы не желаете, чтобы ваш сайт отображался во фреймах чужого сайта, то необходимо, чтобы ваш документ самого верхнего уровня проверял, в какое окно он загружен. Это должно быть окно самого верхнего уровня (`top`) или родительское окно (`parent`). Если это действительно так, то значения ссылок `top` и `self` совпадают. При несовпадении значений `top` и `self` документ следует перезагрузить заново, но уже в окно верхнего уровня. Скрипт, выполняющий эту работу, необходимо разместить в начале документа, например, в секции `<head>`. Вот его код:

```
<script type="text/javascript">
  if(top!=self) top.location=location;
</script>
```

Обратите внимание, что, например, главную страницу сайта Яндекс не удается загрузить во фрейм своего сайта.

Проверка загрузки всех фреймов

При посещении вашего сайта с фреймовой структурой пользователь может сохранить ссылку на документ какого-нибудь одного фрейма, в то время как все средства навигации, реквизиты и элементы оформления находятся в других фреймах. Иначе говоря, посетитель вашего сайта имеет возможность загружать лишь отдельные его компоненты, что может не понравиться вам как автору. Следующий простой скрипт проверяет, загружается ли Web-страница в своем наборе фреймов, сравнивая URL-адреса окна `top` текущего окна. Если они совпадают, значит, загружен документ лишь из одного фрейма, и необходимо загрузить весь набор фреймов, определенный в установочном файле:

```
<script type="text/javascript">
if (top.location.href==location.href)
    top.location.href="URL-адрес файла с описанием фреймов"
</script>
```

Данный скрипт следует разместить в начале каждого (X)HTML-документа вашего сайта с фреймовой структурой. Однако следует иметь в виду, что в результате выполнения этого скрипта во фреймах отобразятся документы, заданные в установочном файле, и среди них может не оказаться того, который хотел видеть пользователь. Простое по идее, но громоздкое по реализации решение данной проблемы состоит в том, чтобы заготовить множество установочных файлов. Каждый такой установочный файл отличается тем, что задает отображение частного документа сайта в одном из своих фрей-

мов. При этом рассмотренный скрипт загружает тот установочный файл, который обеспечивает отображение документа, запрошенного пользователем.

15.1.4. Окно *PopUp* в Internet Explorer

Internet Explorer 5.5+ имеет фирменный объект `popup`, представляющий всплывающее окно без элементов управления, которое можно позиционировать относительно всего экрана (рабочего стола). Объект `popup` является подобъектом `window` и создается методом `createPopup()`, но сначала окно невидимо и не имеет содержимого. С помощью сценария в данное окно можно вывести информацию, отобразить его методом `show()` и скрыть методом `hide()`.

Свойства и методы объекта `popup`:

- `document` — ссылка на объект `document` всплывающего окна;
- `isOpen` — логическое значение, указывающее, открыто ли окно;
- `hide()` — скрывает окно;
- `show(x, y, ширина, высота, объект)` — отображает окно на экране в позиции `x`, `y` с указанными размерами в пикселах; необязательный параметр *объект* — ссылка на объект, относительно которого рассматриваются координаты `x`, `y`.

Если всплывающее окно открыть, а затем щелкнуть где-нибудь вне его на экране, то оно закроется.

В листинге 15.3 (рис. 15.4) приведен пример, в котором создается объект `popup()` и определяются стилевые параметры для его содержимого. Открывает данное окно и наполняет его содержимым функция `pop()`, которая привязана в качестве обработчика события `onmouseover` (наведение указателя мыши) к слову "здесь". Наполнение окна производится с помощью свойства `innerHTML`, принимающего в качестве значения строку, содержащую тег `<img>` и текст.

Если пятым параметром в методе `show` указать, например, `document.body`, то всплывающее окно будет позиционироваться относительно элемента `<body>` документа в окне браузера, а не относительно экрана (если опустить данный параметр).

Листинг 15.3. Всплывающее окно

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Всплывающее окно</title>
<!-- Только для IE -->
<script type="text/javascript">
  var mypopup=createPopup(); // создаем объект окна
  var popBody=mypopup.document.body; // ссылка на тело документа в окне
/* Стиль содержимого окна: */
  popBody.style.border="solid 2px red";
  popBody.style.background="#ffffd0";
  popBody.style.padding="5px";
```

```
/* Содержимое окна: */
var xcontent="<img src='book.jpg' style='float:left'/>";
xcontent+= "Занимательная математика. Множества и отношения";
  function pop(xcontent){            // наполнение содержимым и показ окна
    if(mypopup.isOpen) return null; // если открыто, то ничего не делаем
    popBody.innerHTML=xcontent;     // формируем содержимое
    mypopup.show(120,100,300,200)   // показываем окно
  }
</script>
</head>
<body>
Подробности можно увидеть
 <span style="font-style:italic"
       onmouseover ="pop(xcontent)">здесь</span>
</body>
</html>
```

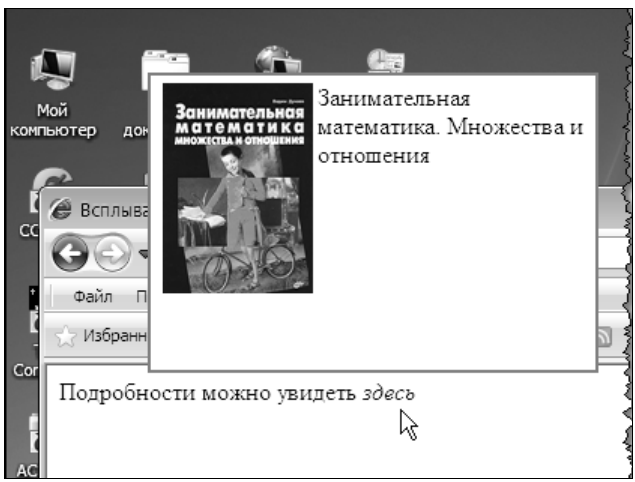


Рис. 15.4. Всплывающее окно (листинг 15.3)

15.2. Работа с таблицами

Как объект документа, таблица имеет две коллекции, посредством которых осуществляется доступ к ее содержимому:

- `rows` — коллекция строк; содержит все строки таблицы, включая разделы `<thead>` и `<tfoot>`;
- `cells` — коллекция ячеек одной строки; содержит элементы таблицы, созданные посредством тегов `<th>` и `<td>`.

К элементам коллекций можно обращаться по числовому индексу, а также по значению атрибута `id`. Ячейки в коллекции элементов одной строки (`cells`) нумеруются слева направо, начиная с нуля.

Примечание

В Internet Explorer объект таблицы имеет свою коллекцию всех ячеек `cells`, а коллекция строк `rows` — свою. Ячейки в коллекции `cells` всей таблицы имеют сквозную нумерацию: слева направо и сверху вниз.

Например, для доступа к *j*-й ячейке *i*-й строки таблицы можно записать такое выражение:

```
document.getElementById("mytab").rows[i].cells[j].
```

В листинге 15.4 приведен пример различных способов обращения к ячейкам таблицы из сценария. Собственно содержимое ячеек получается как значение свойства `innerHTML` — строки HTML-кода, находящегося внутри контейнерного тега элемента. Значение свойства `innerHTML` можно изменять и, следовательно, модифицировать содержимое ячеек. В данном примере читается значение нулевой ячейки первой строки (второй сверху), а затем заменяется строкой "Федоров". Для браузера Internet Explorer должно сработать обращение к ячейке через общую коллекцию `cells`, а в других браузерах соответствующий код будет проигнорирован. Вид рассматриваемого документа в окне Internet Explorer и Firefox показан на рис. 15.5.

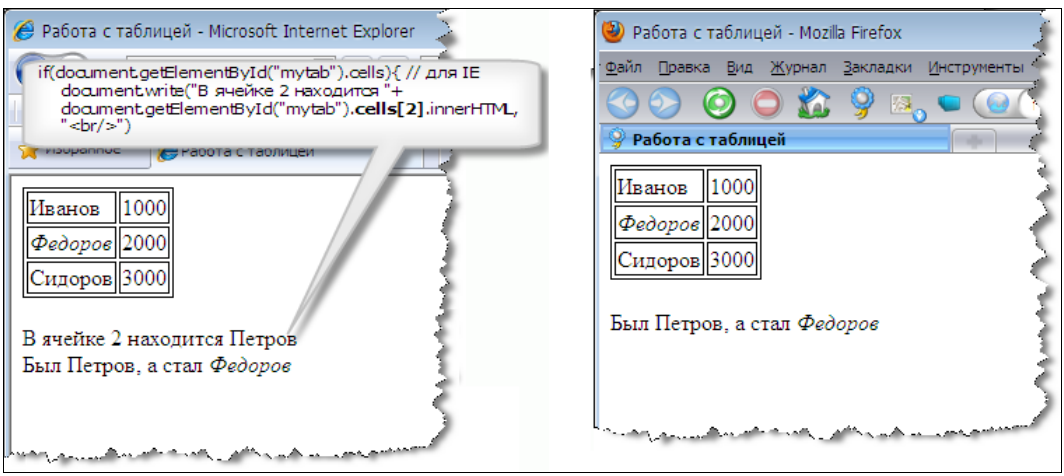


Рис. 15.5. Вид документа из листинга 15.4 в Internet Explorer и Firefox

Листинг 15.4. Пример работы с таблицей

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Работа с таблицей</title>
<style type="text/css">
table, td {border: solid 1px black}
</style>
</head>
<body>
<table id="mytab">
<tr><td>Иванов</td><td>1000</td></tr>
```

```
<tr><td id="x">Петров</td><td>2000</td></tr>
<tr><td>Сидоров</td><td>3000</td></tr>
</table>
<br/>
</body>
<script type="text/javascript">
if(document.getElementById("mytab").cells){// для IE
    document.write("В ячейке 2 находится "+
    document.getElementById("mytab").cells[2].innerHTML,"<br/>")
};
document.write("Был "+
document.getElementById("mytab").rows[1].cells[0].innerHTML);
document.write(", а стал "+
(document.getElementById("mytab").rows[1].cells['x'].innerHTML=
"<em>Федоров</em>"));
</script>
</html>
```

Метод `insertRow(`*индекс_строки*`)` добавляет новую строку в таблицу; параметр указывает позицию строки в таблице. Например, при значении 0 добавляемая строка становится первой, а при значении `document.getElementById("mytab").rows.length` — последней. Вместе с тем, для добавления новой строки в конец таблицы достаточно в качестве индекса указать –1. Вы можете вставить строку внутри существующей таблицы, добавить ее в начало или конец, но нельзя добавлять ее далеко за пределами уже определенных строк. Например, если таблица содержит пять строк (с индексами от 0 до 4), то в методе `insertRow(`*индекс*`)` недопустимы индексы 6, 7 и т. д. Данный метод возвращает ссылку на вновь созданную строку, которая затем служит для вставки ячеек.

Ячейки вставляются в строку с помощью метода `insertCell(`*индекс_ячейки*`)`. Данный метод возвращает ссылку на созданную ячейку, которая в дальнейшем служит для задания содержимого ячейки. Ячейки вставляются в строку по порядку без пропусков, начиная с нулевой. Так, нельзя создать ячейку с индексом 3, если ячейки с индексами 0, 1 и 2 еще не созданы.

> ***Примечание***
>
> В Internet Explorer при добавлении новой строки вы можете последовательно создать любое количество ячеек, даже превышающее их число в уже имеющихся строках. В других браузерах это не так.

Таким образом, новая строка создается в три этапа: сначала к таблице добавляется новая строка, затем в нее вставляются ячейки, после чего они заполняются значениями. Применение указанных методов в сценарии сразу же отражается на виде таблицы в окне браузера. Листинг 15.5 иллюстрирует пример добавления строки к существующей таблице и заполнения ее ячеек значениями.

Листинг 15.5. Добавление строки и заполнение ее ячеек данными

```
var newrow=document.getElementById("mytab").insertRow(-1); /* добавляем
                                        новую строку в конец таблицы */
newcell=newrow.insertCell(0);           // вставляем ячейку с индексом 0
```

```
newcell.innerHTML="Ипполит";            // заполняем ячейку значением
newcell=newrow.insertCell(1);           // вставляем ячейку с индексом 1
newcell.innerHTML="<img src='pict.jpg'>"; // вставляем картинку
```

Для удаления строки таблицы служит метод `deleteRow(`*`индекс_строки`*`)`. Параметр указывает номер удаляемой строки:

```
if (document.getElementById("mytab").rows.length>2)
    document.getElementById("mytab").deleteRow(2);
```

Индекс ячейки в текущей строке таблицы содержится в свойстве `cellIndex`, которое может оказаться полезным при передаче значений параметра методам `insertCell()` и `deleteCell()`.

В листинге 15.6 приведен пример создания таблицы посредством сценария. С помощью разметки задается пустая таблица, а затем скрипт вставляет в нее строки и ячейки (рис. 15.6).

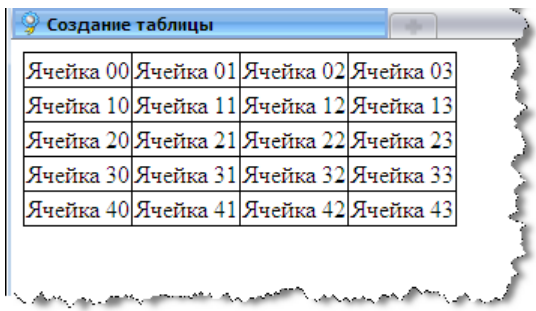


Рис. 15.6. Таблица, созданная с помощью сценария (листинг 15.6)

Листинг 15.6. Создание таблицы с помощью сценария

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Создание таблицы</title>
<style type="text/css">
table,td {border: solid 1px black;border-collapse:collapse}
</style>
</head>
<body>
<table id="mytab"></table>
</body>
<script type="text/javascript">
var nrows=5, ncells=4, newrow, newcell, i, j;
for(i=0;i<nrows;i++) {
  /* добавляем новую строку: */
  newrow=document.getElementById("mytab").insertRow(i);
```

```
  for(j=0;j<ncells;j++){
    newcell=newrow.insertCell(j);      // вставляем ячейку
    newcell.innerHTML="Ячейка "+i+newcell.cellIndex;
  }
}
</script>
</html>
```

15.3. Работа с табличными данными в текстовых файлах

Данные с табличной организацией могут храниться в обычных текстовых файлах (csv-файлах) и представляться на Web-странице с помощью элемента управления ActiveX Tabular Data Control (TDC — управление табличными данными), который рассматривался в *разд. 12.4.2*. Напомню, что данный элемент относится к фирменным расширениям Internet Explorer для Windows и другие браузеры его не поддерживают. О решении данной задачи, инвариантном относительно браузеров, рассказано в *разд. 15.3.2*.

15.3.1. Применение ActiveX Tabular Data Control

В (X)HTML-документ элемент управления ActiveX Tabular Data Control вставляется с помощью тега `<object>`, например, так:

```
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        id="mydbcontrol">
  <param name="FieldDelim" value="|">
  <param name="DataURL" value="mydb.txt">
  <param name="UseHeader" value="true">
</object>
```

Здесь `mydb.txt` — текстовый файл, содержащий данные, разделенные символом `"|"`.

В данном разделе мы рассмотрим применение методов элемента TDC, которые обеспечивают перемещение по строкам, сортировку и фильтрацию табличных данных.

Перемещение по строкам таблицы

Элемент управления TDC предоставляет объект `recordset`, который поддерживает методы перемещения по строкам (записям) таблицы (по записям базы данных). Синтаксис следующий:

id_объекта`.recordset.`*метод*

Например, `mydb.recordset.moveNext()` перемещает указатель текущей записи на следующую запись.

Методы перемещения по записям базы данных:

- `movePrevios()` — к предыдущей записи;
- `moveNext()` — к следующей записи;

- `moveFirst()` — к первой записи;
- `moveLast()` — к последней записи.

Перед вызовом методов `moveNext()` и `movePrevios()` следует проверять значения свойств `eof` (конец файла данных) и `bof` (начало файла данных). Так, если текущей является последняя строка таблицы, то нельзя вызывать метод `moveNext()`. Аналогично, если первая строка текущая, то недопустим `movePrevios()`. Разумеется, методы перемещения можно применять в сценариях обработки событий, таких как, например, щелчок на кнопке. Это необходимо, если на экран выводятся не все записи (строки) таблицы, а только одна запись (например, с помощью полей ввода). Перемещение по строкам таблицы осуществляется при поиске данных в соответствии с некоторым критерием и при их обработке.

Если использовать поля ввода (скажем, при создании формы для просмотра и коррекции данных), то привязку данных к ним можно выполнить с помощью следующего фрагмента (X)HTML-кода:

```
<tr><td>
  <input type="text" datasrc=#mydbcontrol datafld="наименование">
</td></tr>
```

Здесь `mydbcontrol` — значение атрибута `id` тега `<object>`, с помощью которого вставлен элемент управления TDC.

Обратите внимание, что атрибуты привязки данных `datasrc` и `datafld` совместно используются в одном и том же теге.

Рассмотрим следующий пример. Пусть данные находятся в текстовом файле mydb.txt, показанном на рис. 12.11. Создадим страницу, в которой отображаются данные только из одной строки таблицы. Для инициации перемещения по строкам будут служить кнопки, к которым привяжем соответствующие обработчики щелчка. В листинге 15.7 приведен пример (X)HTML-кода, который выводит на страницу одну запись базы данных с кнопками навигации, а на рис. 15.7 — показан результат.

Листинг 15.7. Форма с данными и кнопками навигации

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
  <head><title>Пример формы БД</title></head>
  <body>
  <!-- Элемент TDC -->
    <object id="mydbcontrol"
            classid="clsid:333c7bc4-460f-11d0-bc04-0080c7055a83">
      <param name="FieldDelim" value="|">
      <param name="DataURL"    value="mydb.txt">
      <param name="UseHeader"  value=true>
    </object>

    <!-- Поля с данными -->
    <table width="75%">
```

```
      <tr><th>Наименование</th>
      <td>
        <input type="text"
               datasrc=#mydbcontrol datafld="Наименование" >
      </td></tr>
      <tr><th>Издательство</th>
        <td><input type="text"
                   datasrc=#mydbcontrol datafld="Издательство">
        </td></tr>
      <tr><th>Год</th>
        <td><input type="text"
                   datasrc="#mydbcontrol" datafld="Год">
        </td></tr>
      <tr><th>Обложка</th>
        <td><span datasrc=#mydbcontrol datafld="Обложка"
                  dataformatas="html"></span>
        </td></tr>
    </table>

    <!-- Кнопки перемещения по записям -->
    <input name=cmdFirst type="button" value="<<" title="В начало"
           onclick="First()">
    <input name=cmdPrevious type="button" value="<" title="Назад"
           onclick ="Previous()">
    <input name=cmdNext type="button" value=">" title="Вперед"
           onclick="Next()">
    <input name=cmdLast type="button" value=">>" title="В конец"
           onclick="Last()">
  </body>
  <script type="text/javascript">
    function First() {                          // к первой записи
      mydbcontrol.recordset.moveFirst()
    }

    function Previous() {                       // к предыдущей записи
      if (! mydbcontrol.recordset.bof)
        mydbcontrol.recordset.movePrevious()
    }

    function Next() {                           // к следующей записи
      if (! mydbcontrol.recordset.eof)
        mydbcontrol.recordset.moveNext()
    }

    function Last() {                           // к последней записи
      mydbcontrol.recordset.moveLast()
    }
  </script>
</html>
```

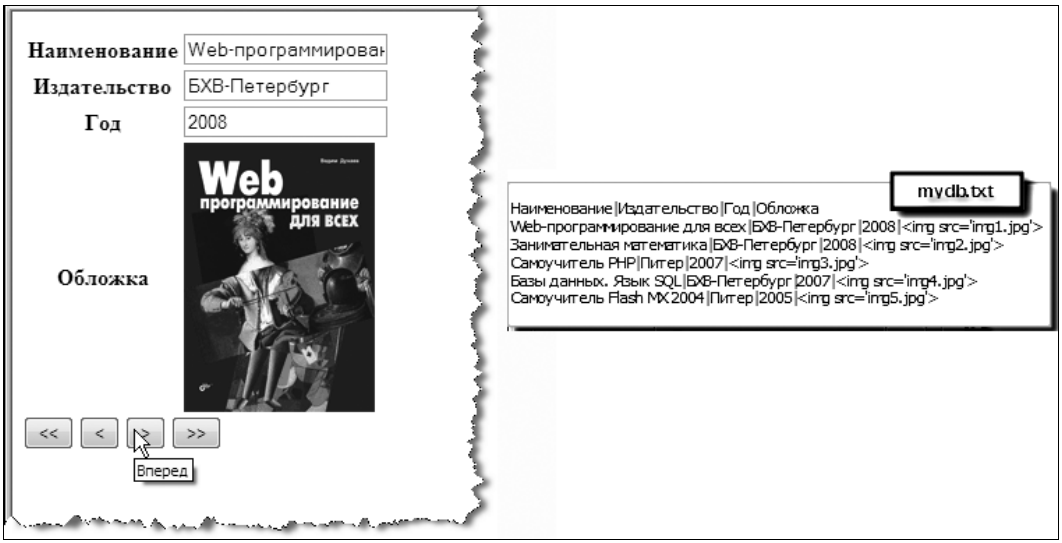


Рис. 15.7. Форма с данными и кнопками
для перемещения по строкам таблицы (листинг 15.7)

Внимание!

Обратите внимание, что четвертое поле с именем `Обложка` содержит изображение, и поэтому мы показываем его не с помощью элемента `<input>`, а с помощью контейнера `<span>` с атрибутом `dataformatas="html"`.

Сортировка данных таблицы

Рассмотрим сортировку строк таблицы по значениям того или иного столбца. В нашем примере для сортировки необходимо просто щелкнуть на заголовке нужного столбца таблицы. Для этого, кроме элемента управления TDC и таблицы, нам понадобится сценарий, содержащий функцию, обрабатывающую щелчок (листинг 15.8). Сначала база данных показывается без всякой сортировки. При щелчке на заголовке таблицы вызывается функция сортировки `sort(field)`, которой передается имя столбца. Эта функция задает значение `field` свойству `SortColumn` элемента TDC и переустанавливает элемент `<object>` переприсваиванием его HTML-кода с помощью свойства `outerHTML`:

```
document.getElementById("mydbcontrol").outerHTML =
            document.getElementById("mydbcontrol").outerHTML;
```

В результате таблица перерисовывается в соответствии с новым порядком следования строк.

Листинг 15.8. Сортировка данных таблицы

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Пример таблицы с сортировкой</title></head>
```

```
<body>
<object id="mydbcontrol"
        classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name="FieldDelim"    value="|">
  <param name="DataURL"       value="mydb.txt">
  <param name="UseHeader"     value=true>
  <param name="SortColumn"    value="">
  <param name="SortAscending" value=1>
</object>

<table datasrc=#mydbcontrol border=5>
  <thead>
    <th onclick="sort('Наименование')">Наименование</th>
    <th onclick="sort('Издательство')">Издательство</th>
    <th onclick="sort('Год')">Год</th>
    <th>Обложка</th>
  </thead>
  <tr>
    <td><span datafld="Наименование"></span></td>
    <td><span datafld="Издательство"></span></td>
    <td><span datafld="Год"></span></td>
    <td><span datafld="Обложка" dataformatas="html"> </span></td>
  </tr>
</table>
</body>
<script type="text/javascript">
  function sort(field) {     /* сортировка по значениям столбца
                                field */
    document.getElementById("mydbcontrol").SortColumn=field;
    /* Переинициализация TDC: */
    document.getElementById("mydbcontrol").outerHTML=
             document.getElementById("mydbcontrol").outerHTML;
  }
</script>
</html>
```

В представленном примере сортировать данные можно только по первым трем текстовым полям. Можно усовершенствовать этот код. Например, сделать так, чтобы при двойном щелчке на заголовке столбца таблицы сортировка происходила в противоположном порядке. Для этого необходимо в обработчике установить нулевое значение для свойства `SortAscending`:

```
document.getElementById("mydbcontrol").SortAscending=0;
```

> ***Примечание***
>
> В предыдущих изданиях, а также в других моих книгах для задания новых параметров элемента управления TDC я формировал строку, содержащую соответствующие теги `<param>`, а затем использовал свойство `outerHTML`, чтобы новые параметры вступили в силу. Это работало в Internet Explorer 6.0. В версии 8.0, как выяснилось, данный прием вызывает ошибку.

Фильтрация данных таблицы

Для организации фильтрации данных из таблицы по заданному критерию используется подход, аналогичный рассмотренному ранее в случае сортировки. В приведенном далее примере (листинг 15.9) в окно браузера выводится таблица с данными и элементы, с помощью которых можно задать столбец и значение (образец) для формирования простого условия фильтра вида: *имя_столбца=значение*.

Имя столбца выбирается из раскрывающегося списка, а значение вводится с клавиатуры. Для установки фильтра следует щелкнуть на кнопке **Применить**, а для отмены фильтра служит кнопка **Отмена**. Если поле ввода значения пусто, то при щелчке на кнопке **Применить** фильтр будет сброшен (отменен). Заметим, что результат фильтрации в данном примере не зависит от регистра, в котором пользователь ввел значение.

Так же, как и при сортировке, результаты фильтрации отображаются с помощью свойства `outerHTML`.

Листинг 15.9. Фильтрация данных таблицы

```
<!DOCTYPE HTML
 PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<html>
<head><title>Фильтрация данных таблицы</title></head>
<body>
<h3>Фильтр:</h3>
    Имя столбца
    <!-- Раскрывающийся список имен столбцов -->
    <select name="fld">
      <option value="Наименование"/>Наименование
      <option value="Издательство"/>Издательство
      <option value="Год"/>Год
    </select>
    <br>
    Значение
    <! Поле ввода значения для фильтра и кнопка >
    <input name="inputvalue" value ="" type="text"/>
    <p>
    <button onclick="filter()">Применить</button>
    <button onclick="cancelfilter()">Отменить</button>
    </p>

<object id="mydbcontrol"
        classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <param name="FieldDelim"    value="|">
  <param name="DataURL"       value="mydb.txt">
  <param name="UseHeader"     value=true>
</object>

<table datasrc=#mydbcontrol border=5>
  <thead>
    <th>Наименование</th>
```

```
    <th>Издательство</th>
    <th>Год</th>
    <th>Обложка</th>
  </thead>
  <tr>
    <td><span datafld="Наименование"></span></td>
    <td><span datafld="Издательство"></span></td>
    <td><span datafld="Год"></span></td>
    <td><span datafld="Обложка" dataformatas="html"> </span></td>
    </tr>
</table>
</body>
<script type="text/javascript">
  function filter() {       // установка фильтра
    if (!inputvalue.value) return cancelfilter();/* если не введено
                                                    значение */
    document.getElementById("mydbcontrol").FilterColumn=fld.value;
    document.getElementById("mydbcontrol").FilterValue=inputvalue.value;
    document.getElementById("mydbcontrol").FilterCriterion='=';
    document.getElementById("mydbcontrol").CaseSensitive=false;
    document.getElementById("mydbcontrol").outerHTML=
            document.getElementById("mydbcontrol").outerHTML;
  }

  function cancelfilter(){ // отмена фильтра
    document.getElementById("mydbcontrol").FilterValue='';
    document.getElementById("mydbcontrol").outerHTML=
             document.getElementById("mydbcontrol").outerHTML;
  }
</script>
</html>
```

Данный код нетрудно усовершенствовать, добавив возможность выбирать оператор сравнения из раскрывающегося списка, а также устанавливать режим зависимости или независимости от регистра.

15.3.2. Применение объекта *XMLHttpRequest*

Для работы с табличными данными в текстовых файлах (csv-файлах) на стороне клиента без использования специфического только для Internet Explorer элемента управления ActiveX TDC (см. *разд. 15.3.1*) достаточно применить объект `XMLHttpRequest`, который подробно рассматривается в *разд. 15.7.1*. Посредством этого объекта можно загрузить в браузер csv-файл с табличными данными. Содержимое файла сохраняется в свойстве `responseText` объекта и далее его следует преобразовать в двумерный массив, используя для этого символы разделения строк и полей. Для сортировки и фильтрации строк можно применить функции, приведенные в листинге 15.10. Для отображения табличных данных на странице можно воспользоваться, например, свойством `innerHTML`, предварительно сформировав в сценарии строку, содержащую табличные теги с соответствующими элементами двумерного массива.

Листинг 15.10. Функции сортировки и фильтрации двумерных массивов

```
function asort(arr,fld,direct){
/* Сортировка двумерного массива:
   arr — двумерный массив
   fld — номер поля (столбца), по значениям которого требуется
         отсортировать строки (нумерация столбцов начинается с 0);
         если не указано, то 0
   direct — порядок сортировки (если пусто, то по возрастанию,
                                иначе — по убыванию)
   Возвращает отсортированный массив
*/
  if (!fld) fld=0;
  if (!direct) {direct=1;} else direct=-1;
  return arr.sort(comp)
  function comp(x,y){
    if (x[fld]<y[fld]) return -direct;
    if (x[fld]>y[fld]) return direct;
    if (x[fld]===y[fld]) return 0
  }
}
/*******************************************************/

function afilter(arr,criterion,count,field){
/* Фильтрация двумерного массива:
   arr - двумерный массив
   criterion — строка с выражением условия выбора строк,
               в которой поля обозначаются в виде fld[0], fld[1] и т. п.;
               например, "fld[0]=='2'||fld[1]=='Маша'"
   count — количество строк, не более которого следует отфильтровать;
           если пусто, то принимается число всех строк
   field — значение для обозначения поля в criterion;
           если не указано, то имеется в виду "fld" (на всякий случай)
   Возвращает массив отобранных строк
*/
  var fltr=[], curcount=0;
  if(!count) count=arr.length;
  if(!field) field="fld";
  var re=new RegExp(field,"g");
  criterion=criterion.replace(re,"arr[i]");
  for(var i=0;i<arr.length;i++){
     if (eval(criterion)){
       fltr=fltr.concat(arr[i]);
       curcount++;
       if(curcount>=count) break
     }
  }
  return fltr
}
```

15.4. Работа с формами

В объектной модели документа каждой форме соответствует свой объект, входящий в коллекцию `forms`. Форма предназначена главным образом для организации отправки на сервер всех данных (введенных пользователем и/или сгенерированных сценарием), имеющихся в элементах этой формы. Для отправки данных на сервер сценарий не обязателен. Чтобы отправить данные, достаточно в теге `<form>` атрибуту `action` присвоить надлежащее значение — URL-адрес получателя, а в самой форме разместить кнопку типа **Отправить** (`submit`). Щелчок на этой кнопке инициализирует отправку данных. Если атрибут `action` не указан или его значение пусто, данные формы не будут отправлены, даже если вы щелкнете на кнопке типа `submit`.

15.4.1. Проверка данных перед отправкой

Если перед отправкой данных формы требуется предварительно их проверить или еще что-нибудь сделать, то необходим сценарий. В примере из листинга 15.11 выводится форма (рис. 15.8) и проверяется, имеется ли символ `@` в поле ввода адреса электронной почты отправителя и не пусты ли другие поля ввода. Если символа `@` в адресе нет или хотя бы одно поле пусто, то отправка не производится. Функция `validator()` обрабатывает событие `onsubmit`, возникающее при щелчке на кнопке типа **Отправить** (`submit`), и возвращает значение `true`, если все условия выполнены, и `false` — в противном случае. Чтобы возвращаемое значение могло разрешить или запретить действие обработчика `onsubmit` по умолчанию, в теге `<form>` указан атрибут `onsubmit="return validator()"`. Форма содержит еще одно невидимое поле `<input name="browser" type="hidden" value="">`, значение которого (название браузера пользователя) формируется программно. Это сделано лишь для демонстрации возможности передачи на сервер данных, неконтролируемых пользователем.

Листинг 15.11. Проверка данных формы перед отправкой

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Отправка данных</title>
<meta http-equiv="content-type"
      content="text/html; charset=windows-1251"/>
</head>
</body>
<form id="myform" name="myform" method="post"
  action="http://www.anyserver.ru/program.php"
  onsubmit="return validator()"
>
  Ваше имя:
  <input name="username" id="username" type="text" value=""/>
  <br/><br/>
  Ваш e-mail:
  <input name="email" type="text" value=""/>
  <br/>
```

```
  Ваше сообщение:<br/>
  <textarea name="message" type="text" value=""></textarea>
  <br/>
  <input name="browser" type="hidden" value=""/>
  <br/>
  <input name="send" type="submit" value="Отправить"/>
</form>
</body>
<script type="text/javascript">
document.getElementById('username').focus(); /* устанавливаем курсор
                                                ввода */
function validator(){ // проверка данных формы
 var nousername=document.myform.username.value=="";
 var noemail=(document.myform.email.value.indexOf('@') == – 1);
 var nomsg=document.myform.message.value=="";
if (nousername || noemail || nomsg){
  errortext="Одно из полей не заполнено,";
  errortext+="\nили неверно указан адрес e-mail.";
  errortext+="\nСообщение не отправлено";
  alert(errortext);
  return false // отменить отправку
 } else { // название браузера в скрытом поле
  myform.browser.value=document.navigator.appName;
  return true // разрешить отправку
 }
}
</script>
</html>
```

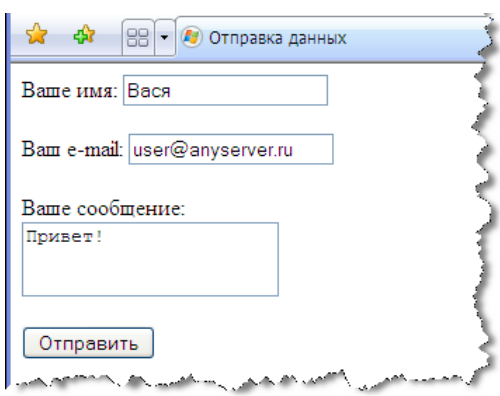


Рис. 15.8. Форма для отправки данных листинга 15.11

В данном примере правильность адреса электронной почты проверяется весьма примитивно. Для более тщательной проверки требуется либо более сложный сценарий, либо так называемые регулярные выражения. Язык регулярных выражений не прост, но его применение для поиска соответствия строк заданному шаблону, замены подстрок и т. п. очень эффективно. Более полную проверку правильности введенного

пользователем адреса электронной почты можно выполнить с помощью следующих двух выражений:

```
var re=new RegExp(/[0-9a-z_]+@[0-9a-z_^.]+.[a-z]{2,3}/i);
var noemail=document.myform.email.value.search(re)==-1;
```

Здесь сначала создается экземпляр объекта `RegExp` и устанавливается регулярное выражение (шаблон), в соответствии с которым введенная пользователем строка должна состоять из двух частей, разделенных символом `@` (имя пользователя и имя домена). В имени пользователя могут быть латинские буквы в любом регистре, цифры, знаки подчеркивания, минуса и точки. Последнее доменное имя (ru, com и т. п.) должно состоять из двух или трех латинских букв. Выражение вида

строка`.search(`*шаблон*`)`

определяет, соответствует ли *строка* указанному в методе `search()` шаблону; если нет, то метод возвращает `-1`.

Приведенные два выражения можно подставить в листинг 15.11 вместо строки

```
var noemail=(document.myform.email.value.indexOf('@') == – 1);
```

и получить более качественную проверку правильности введенного пользователем адреса электронной почты.

15.4.2. Баннер как форма

Баннер — небольшое графическое изображение (часто анимированное), которое что-то рекламирует. Нередко щелчок на баннере просто загружает в браузер некоторую страницу. В рассматриваемом случае это просто графическая ссылка, создаваемая HTML-элементом `<a href=...><img src=.../></a>`. Поскольку функции баннера реализуются тривиально, дизайнеры сосредотачиваются на его внешнем виде. Но не в этом главная сущность данного элемента для программиста. Баннер — конструкция, содержащая что-то видимое и обеспечивающая некоторые действия. С точки зрения функциональности баннер должен обеспечить передачу каких-то данных на сервер, который как-то их обработает. Для создания баннера очень подходит HTML-элемент `<form>`, т. е. контейнер для элементов пользовательского интерфейса, таких как поля ввода данных, а также кнопки и различного рода переключатели. В баннер, как правило, ничего не нужно вводить, а потому он, как форма, должен содержать только невидимые (скрытые) элементы, получаемые из обычных с помощью атрибута `type="hidden"`. Однако они могут содержать данные (значения атрибута `value`), например, URL-адрес страницы, сведения о браузере пользователя и др., необходимые серверному приложению для учета щелчков и, в частности, для начисления бонусов владельцу страницы с данным баннером. Элементы формы могут содержать предустановленные данные (значения атрибутов `value`) или сформированные программно путем присвоения свойству `value` требуемого значения.

Форма для баннера, кроме скрытых полей, должна содержать элемент

`<input type="image" src=`*url-адрес_изображения*`/>`.

Это графическое изображение, которое в форме играет ту же роль, что и кнопка типа `submit`. Иначе говоря, щелчок на ней приводит к передаче данных, содержащихся во всех элементах формы, в том числе и скрытых. Разумеется, файл графического изобра-

жения, указанный в атрибуте `src`, должен иметь надлежащее содержание, соблазняя посетителя страницы щелкнуть на его картинке. Но сейчас не об этом. Пример кода баннера приведен в листинге 15.12. При щелчке на картинке будет вызвана функция `send()`, которая запишет в скрытое поле адрес страницы с данным баннером. Поскольку картинка играет роль кнопки типа `submit`, произойдет отправка на сервер скрытых данных.

Листинг 15.12. Пример кода баннера

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Баннер</title></head>
<body>
<form name="myform"
 action="http://www.anyserver.ru/anyprogram.php" method="post"
 onsubmit="send()">
<input type="hidden" name="mydata"/>
<input type="image" src="banner.jpg"/>
</form>
</body>
<script type="text/javascript">
function send(){ // запись адреса страницы в скрытое поле
document.myform.mydata.value=window.location.href;
}
</script>
</html>
```

Таким образом, если форма содержит только скрытые элементы, то посетитель Web-страницы может и не догадаться, что, щелкнув на картинке или кнопке, он отправляет какие-то данные, сформированные программно.

15.4.3. Переходы между полями по клавише <Enter>

Если форма на Web-странице содержит несколько полей ввода, то переход между ними можно осуществить с помощью мыши или клавиши <Tab>. Однако в обычных приложениях то же самое можно сделать с помощью клавиши <Enter>. Казалось бы, переход к следующему полю формы при нажатии на клавишу <Enter> можно реализовать с помощью метода `focus()` по событию `onkeypress`. Но на Web-странице нажатие на клавишу <Enter> при активном поле формы приводит по умолчанию к отправке данных на сервер, как будто вы щелкнули на кнопке типа `submit`. Задача состоит в том, чтобы по клавише <Enter> осуществить переход к следующему полю, не передавая данные на сервер.

В следующем примере (листинг 15.13) форма содержит три поля ввода данных и обычную кнопку (`<input type="button"/>`). Если поле активно (курсор находится в поле), то при нажатии на клавишу <Enter> происходит активизация следующего поля. При этом

если активизируемый элемент является кнопкой **Отправить**, то инициируется передача данных формы на сервер. Разумеется, щелчок на кнопке **Отправить** также вызывает отправку данных. В рассматриваемом примере нажатие на клавишу <Enter> в последнем поле ввода приводит к передаче данных. Обратите внимание, что форма содержит обычную, а не специальную кнопку типа `submit`, а потому передача данных инициируется принудительно путем программной генерации события `onsubmit` с помощью метода `submit()`. О программном вызове обработчиков событий и программной генерации событий уже говорилось в *разд. 14.4.4*.

Листинг 15.13. Пример переходов между полями по клавише <Enter>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Форма</title></head>
<body onload='document.getElementById("inp1").focus()'>
<form id="myform"
 action="http://www.anyserver.ru/program.php" method="post" >
<input type="text" id="inp1" onkeypress='keypr(event,"inp2")'/>
<br/>
<input type="text" id="inp2" onkeypress='keypr(event,"inp3")'/>
<br/>
<input type="text" id="inp3" onkeypress='keypr(event,"subm")'/>
<br/>
<input type="button" id="subm" value="Отправить"
 onclick="document.myform.submit()"
 onkeypress='keypr(event,"subm")'/>
</form>
</body>
<script type="text/javascript">
function keypr(event,xid){ // обработка нажатия на клавишу
  /* event — объект события;
     xid — id элемента, на который следует установить фокус
  */
  if (event.keyCode==13) {              // если нажали на клавишу <Enter>
    document.getElementById(xid).focus();      // установить фокус на xid
    if (xid=="subm") document.myform.submit(); // отправить
  }
}
</script>
</html>
```

Примечание

Если для передачи данных в форму вставить не обычную кнопку, а кнопку типа `submit`, то возникнут проблемы, связанные с тем, что при нажатии на клавишу <Enter> автоматически вызывается обработчик события `onsubmit` по умолчанию, т. е. делается преждевременная попытка передать данные.

15.5. Работа с локальным хранилищем данных

Для работы некоторых Web-приложений требуется более или менее длительное хранение данных, которые могут использоваться при последующем посещении Web-страницы. Обычно данные сохраняют на диске сервера, но нередко это делается и на локальном диске компьютера пользователя (клиента). Для обеспечения хранения данных на стороне клиента существуют два механизма: cookie (печенье) и объект `localStorage` (локальное хранилище). Если cookie — старинный способ хранения данных, то `localStorage` — новый, спецификация которого существует пока лишь в черновом варианте (**http://dev.w3.org/html5/webstorage/**). Тем не менее его поддерживают все ведущие браузеры.

15.5.1. Cookie

Для хранения небольших (до 4—10К) объемов информации на диске компьютера пользователя в браузере предусмотрен так называемый механизм cookie (куки). Обычно он применяется для аутентификации пользователя (имя и пароль), сохранения параметров настройки его браузера, информации о предыдущем посещении сайта (сеансе), о предпочтениях пользователя. Например, можно сохранить на диске дату последнего посещения сайта данным пользователем. При загрузке Web-страницы это значение сравнивается с некоторой датой, установленной автором сайта в качестве даты обновления. Если вторая (авторская) дата более поздняя, чем первая, то на Web-странице появляется соответствующая отметка, например, текст или изображение с надписью "New" или "Обновлено!". Разумеется, сравнение дат и вывод на страницу отметки реализует сценарий. А вот еще один пример: сценарий проверяет, заходил ли на сайт, скажем, Вася Пупкин; если это произошло, то на страницу выводится персональное сообщение.

Cookie могут создаваться скриптами как на стороне сервера, так и на стороне клиента, но сохраняются всегда на компьютере последнего. При попытке открыть страницу сайта, создавшего cookie-запись, браузер передает ее на сервер в http-запросе данной страницы. Cookie это просто текстовые данные, они не могут содержать объекты, исполняемый код или вирусы и, следовательно, не могут нанести вред компьютеру, на котором сохраняются.

Записи cookie браузеры сохраняют в отдельных текстовых файлах, расположенных в специальной папке. Конкретное местоположение этой папки зависит от операционной системы и браузера.

Структура данных в cookie-файлах для различных браузеров может быть разной, но это не столь важно, поскольку, во-первых, не рекомендуется открывать и изменять эти файлы в текстовых редакторах, а во-вторых, браузеры используют одинаковый синтаксис чтения и записи cookie-данных, основанный на свойстве `document.cookie`.

Итак, данные в cookie-файлах организованы в виде записей. Каждую такую запись можно представить себе в виде строки, содержащей следующие элементы (поля):

- имя записи;
- содержание записи;

- срок хранения (годности) записи;
- домен сервера, который создал запись;
- сведения о необходимости установки безопасного HTTP-соединения для доступа к записям;
- путь расположения документов с URL, которым разрешен доступ к записям.

Зависимость записей от домена обеспечивает безопасность хранения так называемых невосстанавливаемых паролей (пар вида *имя_пользователя-пароль*), поскольку запись, созданную сервером одного домена, не может прочитать сервер с другим доменом.

Учитывая срок хранения, браузер автоматически удаляет просроченные записи, чтобы предотвратить чрезмерное разрастание объема cookie-файлов. Запись данных в cookie-файл с помощью JavaScript осуществляется выражением присваивания строки, содержащей cookie-данные, свойству `document.cookie`. При этом важно соблюдать формат строки (листинг 15.14).

Листинг 15.14. Формат cookie-строки

```
document.cookie="cookieName=данные
[; expires=строка_времени_GMT]
[; path=путь]
[; domain=домен]
[; secure]"
```

Здесь квадратные скобки указывают, что заключенное в них содержимое не обязательно (может быть опущено).

Рассмотрим элементы cookie-строки.

- *cookieName* — каждая cookie-запись должна иметь имя (подобное имени переменной) и строковое значение, которое может быть и пустой строкой. Например, если требуется сохранить слово "Вася" в cookie-записи с именем `User_Name`, то соответствующее выражение JavaScript будет иметь вид:

  ```
  document.cookie="User_Name=Вася".
  ```

  При выполнении этого выражения браузер попытается найти cookie-запись с таким именем. Если он не найдет ее в текущем домене, то создаст автоматически. Если запись с таким именем уже существует, то браузер заменит ее данные новыми. Элемент cookie-записи не должен содержать точек с запятыми, запятых, пробелов и кириллических символов. Чтобы заменить их соответствующими символами, строка с данными предварительно обрабатывается функцией `escape()`. Например, строка "User_Name=Вася" после кодирования функцией `escape()` будет выглядеть следующим образом:

  ```
  User_Name=%u0412%u0430%u0441%u044F
  ```

- `expires` — дата и время хранения (годности) cookie-записи должны быть представлены строкой и содержать данные по Гринвичу (GMT). Например, вычислить дату истечения срока хранения записи месяц спустя от текущей даты можно следующим образом:

```
var expdate=new Date();     // создаем объект даты
var monthFromNow=expdate.getTime() + (30*24*60*60*1000);
expdate.setTime(monthFromNow); // устанавливаем значение даты
```

После этого полученную дату следует привести к строковому формату GMT:

```
document.cookie="User_Name=Вася; expires=" + expdate.toGMTString();
```

Cookie-запись можно удалить и до истечения заданного срока хранения, установив новый срок, заведомо прошедший:

```
expdate=Thu, 01-Jan-70 00:00:01 GMT;
```

Отсутствие срока хранения означает для браузера, что данная cookie-запись временная и не подлежит записи в файл.

В cookie-файле дата и время годности хранятся в виде числового значения (секунды), но при установке cookie-записи они должны быть представлены в формате времени GMT.

- `path` — для cookie-записей на стороне клиента наиболее удобно использовать путь, принятый по умолчанию (в текущей папке). Однако можно создать копию cookie в другой папке, указав путь к ней в качестве значения этого параметра.
- `domain` — для синхронизации cookie-данных с определенным документом или/и группой документов браузер выясняет домен текущего документа и помещает в cookie-файл записи, соответствующие этому домену. Если пользователю требуется получить список всех cookie-записей, содержащихся в свойстве `document.cookie`, то он должен просмотреть все пары *имя-значение*, находящиеся в cookie-файле с именем домена текущего документа. Формат представления домена должен охватывать, по крайней мере, два уровня, например, `.rambler.ru`.
- `secure` — принимает логические значения (`true` или `false`). При создании cookie-записей на стороне клиента этот параметр опускается. Отсутствие данного параметра означает, что cookie-записи доступны для любого документа и программы, находящихся на сайте пользователя.

Теперь рассмотрим чтение и запись данных cookie. Данные cookie, которые можно получить с помощью сценария на JavaScript, представляют собой единственную строку — значение свойства `document.cookie`. Значения отдельных элементов (параметров) cookie выбирают на основе анализа содержимого этой строки методами объекта `String`. Кроме того, если две и более cookie-записи соответствуют одному и тому же домену, то в JavaScript их все равно записывают в одну строку и разграничивают точкой с запятой и пробелом. Для удобства работы с cookie полезно иметь несколько подходящих функций, коды которых мы приведем далее.

Рассмотрим сначала функцию `readCookie(name)`, читающую cookie-данные, соответствующие имени записи `name`, которое передается этой функции в качестве параметра. Данная функция (листинг 15.15) анализирует содержимое свойства `document.cookie`, чтобы выделить в нем имя записи и данные. Если это удастся, то функция возвращает декодированные данные, а в противном случае — `null`. В теле функции `readCookie()` содержится вызов еще одной, вспомогательной, функции `getCookieVal(j)`, возвращающей декодированное с помощью встроенной функции `unescape()` значение cookie-данных (листинг 15.16).

Листинг 15.15. Функция чтения cookie

```
function readCookie(name) {  // чтение cookie-данных записи
 var xname=name + "=";
 var xlen=xname.length;
 var clen=document.cookie.length;
 var i=0;
 while(i < clen){
  var j=i + xlen;
  if (document.cookie.substring(i, j) == xname)
   return getCookieVal(j); // вспомогательная функция
  i=document.cookie.indexOf(" ",1) + 1;
  if (i == 0) break;
 }
 return null
}
```

Листинг 15.16. Функция декодирования cookie

```
function getCookieVal(n){  /* вспомогательная функция, вызываемая
                              из readCookie() */
 var endstr=document.cookie.indexOf(";", n);
 if (endstr == -1) endstr=document.cookie.length;
 return unescape(document.cookie.substring(n, endstr))
}
```

Теперь рассмотрим функцию `writeCookie()`, позволяющую создать или обновить cookie-запись (листинг 15.17).

Листинг 15.17. Функция записи cookie

```
function writeCookie(name, value, expires, path, domain, secure) {
  /* запись cookie */
  if(navigator.cookieEnabled){ // если cookie доступны
   document.cookie =
    name + "=" + escape(value) +
    ((expires) ? "; expires=" + expires.toGMTString(): "") +
    ((path) ? "; path=" + path : "") +
    ((domain) ? "; domain=" + domain : "") +
    ((secure) ? "; secure" : "");
    return true
  } else return false
}
```

Эта функция принимает следующие параметры:

- `name` — строка, содержащая имя cookie-записи (обязательный параметр);
- `value` — строка, содержащая значение cookie (обязательный параметр);
- `expires` — объект даты (`Date`), содержащий срок хранения cookie-записи; если отсутствует, то после завершения работы браузера cookie-запись удаляется;

- `path` — строка, содержащая путь cookie-записи; если параметр не указан, то используется путь вызванного документа;
- `domain` — строка, содержащая домен нужной cookie-записи; если параметр не указан, то устанавливается домен вызванного документа;
- `secure` — логическое значение (`true` или `false`), определяющее необходимость безопасного HTTP-соединения.

Обратите внимание, что в теле функции `writeCookie()` происходит кодирование значения параметра `value` с помощью встроенной функции `escape()`. В функции `writeCookie()` проверяется возможность использования cookie в браузере (свойство `navigator.cookieEnabled`). Если такая возможность есть, то возвращается `true`, а в противном случае — `false`.

Для удаления cookie-записи можно вызывать функцию `deleteCookie()` (листинг 15.18).

Листинг 15.18. Функция удаления cookie

```
function deleteCookie(name, path, domain) {
 /* удаление cookie-записи */
 if (readCookie(name)) {
  document.cookie =
   name + "=" +
   ((path) ? "; path=" + path : "") +
   ((domain) ? "; domain="+domain : "") +
   "; expires=Thu, 01-Jan-70 00:00:01 GMT"
 }
}
```

Эта функция устанавливает дату срока хранения cookie-записи так, что запись будет удалена. Значение параметра `path` должно быть таким же, как при создании cookie-записи, или пустым (`null`), если при создании записи он не был определен. Так же задается значение параметра `domain`.

В рассмотренных кодах `escape()` и `unescape()` можно заменить более современными функциями (методами объекта `global`) `encodeURIComponent()` и `decodeURIComponent()` соответственно. Эти методы кодируют и декодируют все символы, кроме латинских букв, в том числе и такие, как ? и /.

Чтобы поэкспериментировать с чтением и созданием cookie-записей, можно воспользоваться кодом в листинге 15.19. При этом следует иметь в виду, что созданные или измененные cookie-записи будут записаны на диск только после закрытия браузера. До этого они существуют лишь в кэше (оперативной памяти). С другой стороны, cookie-записи загружаются в оперативную память и становятся доступными как значение свойства `document.cookie` только при запуске браузера.

Листинг 15.19. XHTML-код для эксперимента со сценарием, записывающим и читающим cookie-запись

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Работа с cookie</title></head>
 <script type="text/javascript">
/****** Определения функций для работы с cookie ***********************/
  function readCookie(name) { // чтение cookie-данных записи
   ... // см. листинг 15.15
  }
  function getCookieVal(n){  /* вспомогательная функция, вызываемая
                                из readcookie() */
   ... // см. листинг 15.16
  }
  function writeCookie(name, value, expires, path, domain, secure) {
  // запись cookie
   ... // см. листинг 15.17
  }
function deleteCookie(name, path, domain) { // удаление cookie
       // см. листинг 15.18
}
/****** Чтение и создание записи cookie ******************************/
  / * Задаем срок хранения — 1 год от текущей даты: */
  var d1=new Date();
  var d2=d1.getTime() + (365*24*60*60*1000);
  d1.setTime(d2);
  /* запись и чтение cookie: */
  if (writeCookie("User_Name","Вася Пупкин",d1)){
    document.write("document.cookie: ",document.cookie,"<br/>");
    document.write(
       "Содержимое записи User_Name: ",readCookie("User_Name"))
  } else
    alert("Для правильной работы страницы\nнеобходимо записать cookie"); </script>
</html>
```

15.5.2. Объект *localStorage*

Для преодоления ряда недостатков cookie служит специальный объект `localStorage`, являющийся подобъектом объекта `window`. Работа с ним аналогична работе с cookie. При этом допускается хранение данных объемом до 5M.

Объект `localStorage` имеет следующие методы:

- `setItem(`*имя_записи*, *содержание_записи*`)` — сохранение строки содержание_записи под именем имя_записи;
- `getItem(`*имя_записи*`)` — возвращает содержимое записи с указанным именем;
- `removeItem(`*имя_записи*`)` — удаляет указанную запись;
- `clear()` — очищает хранилище.

Следует иметь в виду, что сохраняемые данные имеют строковый тип. Если необходимы данные других типов, то следует воспользоваться методами преобразования типов (например, `parseInt()`, `parseFloat()` и др.).

Пример:

```
localStorage.setItem("myrecord", "Привет");  /* запись строки "Привет"
                                                под именем myrecord */
localStorage.getItem("myrecord");    // возвращает Привет
localStorage.removeItem("myrecord"); // удаляет запись myrecord
localStorage.getItem("myrecord");    // возвращает null
```

15.6. Работа с графическими изображениями

15.6.1. Объект элемента *<img>*

Графическим изображениям, вставляемым в (X)HTML-документы посредством тегов <img>, соответствуют объекты, которые собраны в коллекцию images объекта document. Данная коллекция соответствует стандарту DOM 1. К ее элементам можно обращаться по индексу и по значению атрибута name тегов <img>. Кроме того, можно использовать метод

```
document.getElementById(значение_id).
```

Допустим, например, в документе имеется единственное графическое изображение, заданное тегом . Тогда получить значение свойства src данного элемента можно следующими способами:

```
document.images[0].свойство;
document.images["myimg"].src;
document.images.myimg.src;
document.getElementById("myimg").src;
```

Здесь во втором и третьем выражениях используется значение "myimg" атрибута name, а в четвертом — атрибута id. В данном примере значения атрибутов name и id совпадают, но в общем случае они могут различаться. Чтобы не запутаться при различных способах обращения, рекомендуется задавать одинаковые значения для атрибутов id и name.

Убедиться в возможности браузера работать с графическими изображениями посредством сценария можно с помощью проверки доступности коллекции images:

```
if(document.images) {
// код для работы с изображениями
}
```

Все современные браузеры поддерживают коллекцию images и позволяют работать с изображениями посредством скриптов.

Особенность элемента документа, заданного тегом <img>, состоит в том, что указанное в нем графическое изображение отображается в браузере, а изменение свойств соответствующего ему объекта с помощью сценария сразу же отражается на внешнем виде изображения. Так, изменение значения свойства src приводит к смене картинки, а модификация свойств width и height — ее размеров. С помощью параметров CSS изображение можно сделать невидимым. Вместе с тем, изображение можно загрузить в браузер, но не отображать его сразу же. Предварительно загруженное изображение можно

как-то обработать прежде, чем отобразить в браузере. Об этом будет рассказано в следующих подразделах.

15.6.2. Объект *Image*

Кроме объектов элементов, задаваемых в (X)HTML-документе тегами <img>, имеется статический объект Image, который существует только в памяти браузера и сам по себе ничего не отображает. Его экземпляры создаются в сценарии для предварительной загрузки изображений, которые потом можно отобразить, а также для решения других задач. Как создаются и применяются экземпляры объекта Image, мы рассмотрим в *разд. 15.6.4* и *15.6.5*, а сейчас перечислим его свойства (такие же, что и для объектов элементов <img>).

Объект Image, кроме базовых свойств, которыми обладают все объекты документа (id, className, style, title и др.), имеет следующие специфические свойства:

- align — параметры выравнивания изображения, обычно left или right;
- alt — альтернативный текст, задаваемый атрибутом alt;
- border — ширина рамки около изображения, задаваемая в пикселах;
- complete — логическое значение, указывающее, завершена ли загрузка изображения (нестандартное свойство, но имеющее широкую поддержку);
- height — высота изображения в пикселах;
- hspace — ширина отступа слева и справа от изображения;
- longDesc — так называемое "длинное" описание изображения, задаваемое атрибутом longdesc; по замыслу это должен быть URL-адрес файла с подробным описанием изображения;
- lowsrc — URL-адрес изображения с низким разрешением, задаваемый атрибутом lowsrc;
- name — значение атрибута name;
- src — URL-адрес изображения;
- useMap — URL-адрес изображения, используемого в качестве клиентской графической карты ссылок, когда в теге <img> упоминается атрибут usemap;
- vspace — ширина отступа сверху и снизу от изображения;
- width — ширина изображения в пикселах.

Объект Image поддерживает все события мыши, а также события onload, onabort и onerror. Обработка последних может понадобиться при загрузке больших изображений. Так, событие onload возникает при завершении загрузки, onabort — когда пользователь прерывает загрузку с помощью кнопки браузера **Остановить**, а onerror — когда при загрузке возникает ошибка.

15.6.3. Управление свойствами изображения

Установить или изменить свойства изображения, такие как src, width, height и др., можно обычным способом:

```
document.getElementById("id_элемента_img").свойство=значение;
```

При установке нового значения свойства `src` (URL-адрес файла изображения) браузер посылает на сервер соответствующий запрос и пытается загрузить и отобразить графическое изображение без перезагрузки текущего документа. Иначе говоря, содержимое элементов `<img>` изменяется динамически.

В листинге 15.20 приведен пример документа с элементом `<img>` и формой, в которой можно выбрать графическое изображение и ввести значения его ширины и высоты (рис. 15.9). При щелчке на кнопке **Применить** вызывается функция `changeimg()`, которая устанавливает заданные в форме значения для соответствующих свойств изображения. Результат можно сразу же увидеть в окне браузера. Щелчок на кнопке **Отмена** возвращает содержимое формы в исходное состояние и отображает исходную картинку. Обратите внимание, что кнопка **Отмена** является обычной, а не кнопкой типа `reset`. Событие для восстановления исходного состояния формы генерируется программно в обработчике события `onclick`, который после этого вызывает функцию `changeimg()`, устанавливающую параметры изображения в соответствии с данными формы. Если бы мы использовали не обычную, а кнопку типа `reset` с обработчиком `onclick="changeimg()"`, то при щелчке на ней сработал бы обработчик по умолчанию (восстановление формы), а не `changeimg()`.

Листинг 15.20. Динамическая установка свойств графического изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Изменение свойств изображения</title>
<script  type="text/javascript">
 function changeimg(){ // изменение параметров изображения
  document.getElementById("myimg").src=document.myform.mysrc.value;
  document.getElementById("myimg").width=document.myform.mywidth.value;
  document.getElementById("myimg").height=document.myform.myheight.value;
}
</script>
</head>
<body>
<form id="myform" name="myform">
<!-- Ввод значений свойств изображения -->
 Файл:
 <select id="mysrc" name="mysrc" >
   <option value="picture1.jpg"/>Картинка1
   <option value="picture2.jpg"/>Картинка2
   <option value="picture3.jpg"/>Картинка3
 </select>
 <br/>
 Ширина:
 <input id="mywidth" name="mywidth" type="text" value="200" size="4"/>
 Высота:
 <input id="myheight" name="myheight" type="text" value="100" size="4"/>
 <br/>
 <input id="applay" name="applay" type="button" value="Применить"
        onclick="changeimg()"/>
```

```
 <input id="cancel" name="cancel" type="button" value="Отмена"
        onclick="document.myform.reset(); changeimg()"/>
</form>
 <img id="myimg" name="myimg" src="picture1.jpg" alt="Картинка"
      width="200" height="100"/>
</body>
</html>
```

Рис. 15.9. Динамическое изменение параметров графического изображения (листинг 15.20)

Некоторые свойства графического изображения (например, `width` и `height`) могут быть установлены не только с помощью атрибутов тега `<img>` или программным путем как свойства соответствующего объекта, но и посредством CSS. Параметры CSS имеют приоритет перед соответствующими параметрами, задаваемыми атрибутами и с помощью скриптов. Например, если в листинге 15.20 для `<img>` задать стилевой параметр `width`, то ширина всех картинок будет определяться именно его значением и сценарий в данном листинге не сможет ее изменить. Чтобы все же это сделать, необходимо в функции `changeimg()` заменить выражение

```
document.getElementById("myimg").width=document.myform.mywidth.value;
```

на такое:

```
document.getElementById("myimg").style.width=
                                  document.myform.mywidth.value+"px";
```

15.6.4. Предварительная загрузка изображений

Как уже говорилось, при использовании тега `<img src="`*URL-адрес_изображения*`"/>` происходит загрузка и отображение картинки с указанным URL-адресом. Однако загрузка в браузер очень большого изображения или большого количества картинок обычных размеров может потребовать значительного времени. При этом формирование документа на экране происходит визуально постепенно. Если к тому же сценарий попытает-

ся изменить параметры изображения до окончания его загрузки, то будет показана испорченная картинка.

С помощью объекта `Image` можно разделить процессы загрузки и отображения картинки. Сначала графические изображения загружаются в память, оставаясь невидимыми. В это время пользователь может рассматривать текстовую информацию. Затем, по окончании загрузки, картинки быстро отображаются в окне браузера.

Для предварительной загрузки картинки следует создать экземпляр объекта `Image` в памяти браузера:

```
imgObj = new Image(ширина, высота);
```

Необязательные параметры функции-конструктора объекта помогают браузеру определить размер выделяемой памяти. Если цель создания объекта — предварительная загрузка изображения, то параметры можно не указывать. В конечном итоге будет выделено столько памяти, сколько необходимо для сохранения загружаемого изображения.

Для созданного в памяти объекта изображения `myimg` с помощью свойства `src` можно задать имя или, в общем случае, URL-адрес графического файла:

```
imgObj.src="URL-адрес_изображения".
```

Данное выражение предписывает браузеру загрузить в кэш-память указанную картинку, но не выводить ее. После записи в кэш всех изображений и загрузки всего документа можно сделать их видимыми. Для этого свойству `src` элемента `<img>` нужно присвоить значение этого же свойства объекта изображения в кэш-памяти, например, так:

```
document.images[0].src=myimg.src;
```

Здесь слева от оператора присваивания указано свойство `src` первого элемента в документе, соответствующего тегу `<img>`, а справа — свойство `src` объекта изображения в кэш-памяти. Разумеется, в коллекции `images[]` можно указать не только индекс, но и значение атрибута `id`. Вместо коллекции можно применить и метод `document.getElementById("значение_id")`.

В листинге 15.21 приведен пример предварительной загрузки изображения. Свойство `src` для элемента `<img>` устанавливается по окончании загрузки документа обработчиком события `onload`, привязанным к объекту `window`. Обратите внимание, что в теге `<img>` атрибут `src` имеет пустое значение.

Листинг 15.21. Предварительная загрузка изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Предварительная загрузка изображения</title>
<script  type="text/javascript">
  var imgObj=new Image();   // создаем экземпляр объекта Image
  imgObj.src="picture1.jpg";// загружаем изображение
  window.onload=function(){ // регистрируем обработчик
    /* делаем изображение видимым: */
    document.getElementById("myimg").src=imgObj.src
  }
```

```
</script>
</head>
<body>
 <img id="myimg" src="" alt="Картинка" />
</body>
</html>
```

Теперь рассмотрим в качестве примера (X)HTML-документ, в котором отображается раскрывающийся список названий графических элементов и одно исходное изображение. Выбор элемента из списка приводит к отображению требуемой картинки. Все графические элементы из этого списка предварительно загружаются в кэш-память и поэтому быстро отображаются при выборе. Раскрывающийся список, как известно, создается с помощью контейнерного тега <select>, содержащего теги <option>. Соответствующий фрагмент (X)HTML-кода генерируется сценарием и записывается в текущий документ (листинг 15.22). Внешний вид данного документа показан на рис. 15.10.

Листинг 15.22. Выбор изображения из списка предварительно загруженных изображений

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
 <head><title>Предварительная загрузка изображений</title>
 <script type="text/javascript">
 var imgFile=new Array(  // массив имен графических файлов
              "picture1.jpg",
              "picture2.jpg",
              "picture3.jpg",
              "picture4.jpg"
             );
 var imgName=new Array(  // массив названий картинок
              "Картинка1",
              "Картинка2",
              "Картинка3",
              "Картинка4"
             );
 /* Создание объектов изображений и загрузка изображений в кэш: */
 var imgObj=new Array()   // массив объектов изображений
 for(i=0; i < imgFile.length; i++){
   imgObj[i]=new Image(); // создаем экземпляр объекта изображения
   imgObj[i].src=imgFile[i]  /* загрузка картинки в память
                               без отображения */
 }

 function imgshow(list){ // отображение картинки при выборе из списка
   var x=list.options[list.selectedIndex].value; /* выбранный номер
                                                    картинки */
   document.getElementById('img0').src=imgObj[x].src
 }
```

```
 /* Создание списка картинок */
 var clist="<select onchange='imgshow(this)'>"
 for(i=0; i<imgFile.length; i++){
   clist+= "<option value='" + i + "'/>" + imgName[i]
 }
 clist+= "</select>";
 document.write(clist)  // запись списка картинок в документ
 </script>
 </head>
<body>
 <!-- Исходная картинка -->
 <img id="img0" src="picture1.jpg" style="width:200px; height:150px"/>
</body>
</html>
```

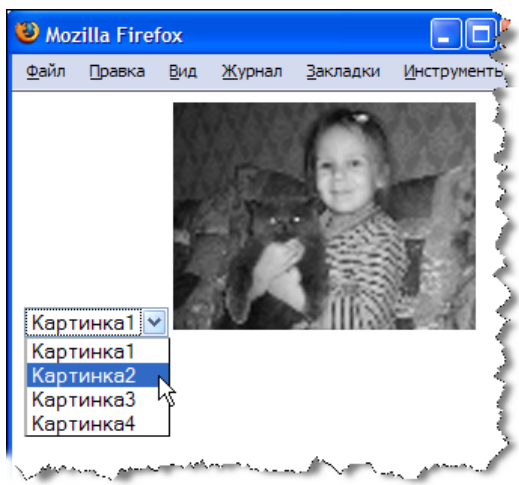


Рис. 15.10. Страница, сформированная кодом листинга 15.22

15.6.5. Нетипичные применения объекта *Image*

Типичное применение объекта `Image` — загрузка в кэш браузера графического изображения, URL-адрес которого указывается в свойстве `src`. Как только вы установите значение для свойства `src` объекта `Image`, браузер сразу же пошлет запрос по указанному адресу. Это произойдет, даже если в `src` указан URL-адрес не графического файла, а файла, например, с серверным сценарием (CGI-программы). Как известно, к URL-адресу файла можно дописать через символ `?` передаваемые данные, например, набор пар вида *имя=значение*, разделенных символом `&`. Серверный сценарий может как-то обработать полученные данные и вернуть какое-нибудь изображение или что-либо иное. Он может даже вернуть код ошибки, что совершенно не важно, если целью применения объекта `Image` является передача данных на сервер. Далее мы рассмотрим примеры использования `Image` не по своему прямому назначению.

Передача данных на сервер

В качестве примера практического использования объекта изображения для передачи данных на сервер можно привести создание счетчика посещений Web-страницы. При загрузке страницы в браузер необходимо отправить на сервер идентификатор счетчика, чтобы серверный сценарий мог увеличить текущее значение счетчика с данным идентификатором на единицу. Кроме того, сервер может вернуть графическое изображение счетчика, чтобы браузер отобразил его на странице. Вот как устанавливается в (X)HTML-документе счетчик Rambler:

```
<a href="http://top100.rambler.ru/top100/" >
  <img src="http://counter.rambler.ru/top100.cnt?123456"
       alt="Rambler's Top100" width=54 height=42 border=0/>
</a>
```

Здесь в атрибуте `src` тега `<img>` указан URL-адрес некоторой серверной программы, которой передается идентификатор счетчика — в данном случае `123456`. Серверная программа изменяет текущее значение счетчика, создает графическое изображение счетчика с данным значением и возвращает его, а браузер отображает полученную картинку. Тег `<img>` вставлен в тег ссылки на страницу Rambler's Top100, где можно посмотреть подробную статистику посещений вашей страницы. Доступ к статистике получают обычно с помощью пароля.

Если вы не хотите демонстрировать счетчик на своей странице, то вместо приведенного ранее (X)HTML-кода достаточно воспользоваться следующим скриптом:

```
var mycounter=new Image();
mycounter.src="http://counter.rambler.ru/top100.cnt?123456";
```

Как известно, стандарный способ передачи данных, введенных пользователем, — применение форм. После отправки данных в этом случае обычно происходит загрузка новой страницы, содержащей ответ сервера. Иногда требуется отправить данные и получить ответ без перезагрузки страницы. Например, небольшая область для ввода данных регистрации посетителя может находиться на основной странице, а после отправки данных на сервер страница не должна перезагружаться. При этом область с введенными данными может остаться на месте или быть заменена каким-нибудь элементом, например, картинкой или словами благодарности за регистрацию.

Другой пример — так называемое рейтинговое голосование, при котором на страницу выводится список элементов (например, товаров или услуг), каждому из которых предлагается дать оценку. Оценив элемент списка, посетитель страницы может сразу же отправить свою оценку на сервер. Если воспользоваться стандартным механизмом отправки данных формы, то при оценке каждого элемента будет происходить неприятная перезагрузка страницы. Если же данные отправлять лишь при оценке всех элементов, то есть риск, что пользователь просто забудет щелкнуть на кнопке типа `submit` и, следовательно, введенные им данные не будут переданы.

Реализовать идею отправки данных формы без перезагрузки страницы можно с помощью объекта `Image`. В листинге 15.23 приведен пример, в котором данные формы передаются на сервер при щелчке на обычной кнопке **Отправить**, а не на кнопке типа `submit`. Это сделано с тем, чтобы обойти механизм отправки данных, принятый для форм по умолчанию. В обработчике `sendData()` щелчка на обычной кнопке формируется строка, со-

держащая URL-адрес серверного сценария (в примере это `http://anyserver/php/myprogram.php`) и передаваемые ему параметры (содержимое элементов `<input>` формы). При этом мы кодируем данные посредством метода `encodeURIComponent()`, поскольку пользователь может ввести кириллические и другие недопустимые для URL символы. Эта строка устанавливается в качестве значения свойства `src` объекта `Image` и, следовательно, данные передаются по указанному адресу. Далее, не дожидаясь ответа сервера, мы просто заменяем содержимое контейнера `<form>` на строку "Спасибо за регистрацию". При этом исходная страница не перезагружается.

Листинг 15.23. Передача данных посредством объекта `Image`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Передача данных</title></head>
<body>
<form id="myform" action="" method="get">
  Логин:<input type="text" id="inp1" name="inp1"/>
  <br/>
  Пароль:<input type="password" id="inp2" name="inp2"/>
  <br/>
  <input type="button" id="send" value="Отправить"
         onclick="sendData()"/>
</form>
</body>
<script type="text/javascript">
function sendData(){ // Отправить данные
 var xadress="http://anyserver.ru/php/myprogram.php"+
      "?inp1="+encodeURIComponent(document.getElementById('inp1').value)+
      "&inp2="+encodeURIComponent(document.getElementById('inp2').value);
 var imgobj=new Image();
 imgobj.src=xadress; // передача данных
./* Замена содержимого формы текстом: */
 document.getElementById("myform").innerHTML="Спасибо за регистрацию"
}
</script>
</html>
```

В рассмотренном примере сценарий отправляет данные на сервер и никак не реагирует на его отклик. В качестве последнего серверный сценарий мог бы послать произвольное графическое изображение шириной и высотой, например, 1 px. Тогда клиентский сценарий мог бы произвести изменения на странице, только если это изображение действительно загрузилось в браузер. Но в этом случае необходимо еще предусмотреть возможную задержку загрузки. Такого рода проверка рассматривается в следующем подразделе.

Объект `Image` — не единственно возможное средство для обмена данными с сервером без перезагрузки страницы. Другой более функциональный способ — применение объекта `XMLHttpRequest`, которое будет рассмотрено в *разд. 15.7*.

Парольная защита страницы на стороне клиента

Защита паролем доступа к Web-странице обычно обеспечивается посредством серверных сценариев. Однако эту же задачу можно решить и с помощью клиентского сценария, применив объект `Image` для передачи на сервер введенного пароля и получения ответа.

Главная особенность решения задачи парольной защиты на стороне клиента состоит в том, что в сценарии проверки правильности пароля последний не должен упоминаться в явном виде. Поэтому предварительно создадим графический файл (например, в формате GIF, PNG или JPG). При этом не важно, каково его содержимое, но желательно, чтобы изображение было возможно меньших размеров, например, 1×1 пиксел. Сохраним данный файл на сервере в той же папке, в которой находится (X)HTML-документ с элементами для ввода и отправки пароля. Графический файл сохраняем под некоторым именем без расширения. Имя файла и будет паролем.

При загрузке страницы с предложением ввести пароль в текстовое поле (элемент `<input type="password"/>`) пользователь вводит набор символов, который присваивается атрибуту `src` объекта `Image`, если щелкнуть на кнопке **OK**. Если указанный таким образом графический файл существует, то он будет загружен в браузер, но не показан на странице. Далее остается проверить, равны ли ширина и высота загруженного изображения определенным величинам, например 1 px. Таким образом, выполняются две проверки: факта существования файла и размеров изображения. Кроме этого, необходимо предусмотреть, что загрузка файла с сервера может произойти с некоторой задержкой, точное значение которой предугадать невозможно. Поэтому следует предусмотреть несколько проверок окончания загрузки файла, выполняющихся через некоторый временной интервал.

В листинге 15.24 представлен документ, формирующий страницу для ввода пароля (рис. 15.11). При щелчке на кнопке **OK** вызывается функция `enterpw()`, которой в качестве параметра передается значение атрибута `id` элемента, содержащего пароль (в примере это `"pw"` поля ввода пароля). Данная функция создает экземпляр объекта `Image` и делает попытку загрузить в него файл с именем, совпадающим с введенным паролем.

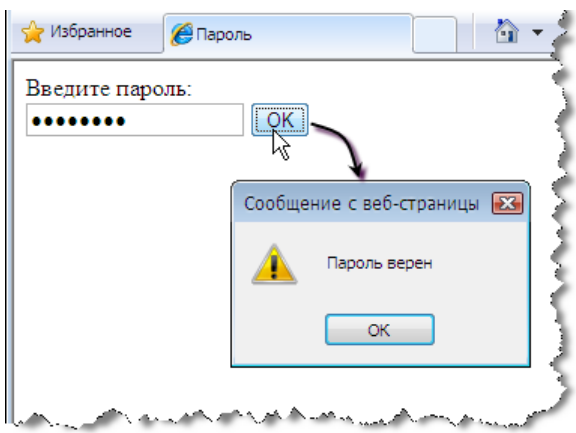


Рис. 15.11. Страница ввода пароля (листинг 15.24)

Далее, вызывается метод `setInterval(validator, 20)`, который периодически с интервалом 20 мс вызывает функцию `validator()`, выполняющую необходимые проверки и соответствующие действия. Функция `validator()` определена в теле функции `enterpw()`. Подробнее о методах управления вычислительными процессами во времени рассказано в *разд. 15.8*.

Число попыток загрузить файл не должно превышать 100 (время ожидания не более 2 с). С помощью свойства `complete` объекта `Image` проверяется окончание загрузки изображения. Если лимит попыток не исчерпан, а загрузка не завершена, то повторная проверка окончания загрузки повторится примерно через 20 мс. Если загрузка завершилась, то процесс периодических вызовов функции `validator()` прекращается методом `clearInterval()`, а затем проверяются размеры изображения. В рассматриваемом примере по результатам данной проверки выводится диалоговое окно с соответствующим сообщением. Разумеется, вы можете вместо этого для случая успешной проверки написать код загрузки той страницы, которая защищена данным паролем, например,

```
window.location.href="myspecialpage.htm".
```

Листинг 15.24. Парольная защита страницы

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пароль</title><head>
<body>
Введите пароль:<br>
<input type="password" id="pw" value="">
<button onclick="enterpw('pw')">OK</button>
</body>
<script type="text/javascript">
function enterpw(xid){ // обработка введенного слова и проверка пароля
  var imgobj=new Image();
  imgobj.src = document.getElementById(xid).value;// загрузка изображения
  validator.i=0; // счетчик вызовов validator;
  var timer=setInterval(validator,20); // периодический вызов validator
  return null

  function validator(){ // проверка пароля
    validator.i=validator.i+1;
    if (validator.i>100) {  // если слишком много вызовов
      clearInterval(timer); // останавливаем таймер
      alert("Сервер долго не отвечает.\nВозможно, неверный пароль");
    } else {
      if(imgobj.complete) { // если изображение загружено
         clearInterval(timer); // останавливаем таймер
         if((imgobj.width==1)&&(imgobj.height==1)){ /* если правильные
                                                       размеры */
            alert("Пароль верен");
            imgobj=null;
```

```
            // здесь могут быть другие операторы
         } else {
            document.getElementById(xid).value=""; // очищаем поле ввода
            alert("Пароль НЕ верен");
            // здесь могут быть другие операторы
         }
       }
    }
    return null
  }
}
</script>
</html>
```

Сценарий листинга 15.24 в случае, когда требуемый графический файл не существует, работает в Internet Explorer и Opera несколько иначе, чем в Firefox, Safari и Chrome. Первые два браузера в данном случае выдают сообщение "Сервер долго не отвечает. Возможно, пароль не верен", а остальные — "Пароль не верен". Это связано с различиями в установке браузерами значения свойства `complete` объекта `Image`. Так, при отсутствии файла Internet Explorer и Opera устанавливают этому свойству значение `false` (видимо, интерпретируя ситуацию как неудачную загрузку), а остальные браузеры устанавливают `true` (интерпретируя как завершение процесса загрузки, пусть и неудачной).

Защищенная страница должна иметь скрипт, определяющий, была ли она вызвана из той страницы, в которой вводился пароль. Иначе говоря, прежде чем показать *защищенную* страницу, следует еще проверить, была ли она вызвана из *защищающей* страницы. Это можно сделать, например, так:

```
var ref="URL страницы ввода пароля";
if(document.referrer!=ref) window.location.href=ref;
```

Можно ли взломать описанную парольную защиту? Разумеется, да. Советую подумать об этом самостоятельно в качестве упражнения. Тем не менее, предлагаемый здесь простой способ позволит оградить вашу страницу от подавляющего большинства тех, для кого она не предназначена.

Для некоторого улучшения качества защиты можно порекомендовать не располагать на странице ввода пароля картинок, находящихся в той же папке, а также добавить в сценарий проверку, не введены ли в качестве символов пароля слэши.

15.7. Взаимодействие с сервером: объект *XMLHttpRequest* и AJAX

Web-страница взаимодействует с сервером, как известно, посредством форм и таких элементов (X)HTML, которые имеют атрибуты `href`, и `src`, принимающие в качестве значений URL запрашиваемого ресурса. Обратиться через URL можно и к серверному сценарию (CGI-программе). При этом к URL-адресу можно дописать данные (параметры), чтобы передать их серверной программе (см. *разд. 6.5*). В *разд. 15.6.5* рассматривалось применение объекта `Image` для решения некоторых задач коммуникации с сервером, хотя его исконное назначение — работа с графическими изображениями. Вместе

с тем, есть специальный объект `XMLHttpRequest` для организации в клиентском сценарии взаимодействия с сервером. Технологию его применения теперь обычно называют AJAX.

15.7.1. Объект *XMLHttpRequest*

Современные браузеры поддерживают объект `XMLHttpRequest`, который позволяет организовать с помощью клиентского сценария взаимодействие с сервером без перезагрузки текущего (X)HTML-документа. Иначе говоря, с помощью данного объекта можно изменять содержимое страницы динамически. Более того, взаимодействие с сервером может происходить асинхронно, т. е. вы можете не дожидаться завершения обмена данными, приостанавливая работу других частей сценария. Другими словами, сценарий может продолжать выполняться, хотя запрос к серверу еще не завершен. Обмен данными может происходить и без видимых изменений страницы, в фоновом режиме. Ответ сервера сохраняется в объекте `XMLHttpRequest`, но автоматически нигде не отображается. При необходимости полученные данные можно обработать и отобразить другими средствами JavaScript.

Свойства объекта *XMLHttpRequest*

Объект `XMLHttpRequest` обладает следующими свойствами и методами:

- `readyState` — состояние запроса, представляемое целым числом: 0 (не инициализирован), 1 (загрузка), 2 (получены заголовки ответа), 3 (получена часть содержимого ответа), 4 (запрос завершен);
- `onreadystatechange` — свойство, значением которого является имя функции, вызываемой при изменении значения свойства `readyState` (состояния запроса);
- `status` — http-код состояния, возвращаемый сервером (например, "200");
- `statusText` — полная строка http-состояния, возвращенная сервером (например, "200OK");
- `responseText` — полный ответ сервера в виде строки;
- `responseXML` — объект `document`, представляющий собой ответ сервера, рассмотренный как XML-документ;
- `abort()` — отменяет асинхронный запрос;
- `getAllResponseHeaders()` — возвращает строку, содержащую все http-заголовки, имеющиеся в ответе сервера. Каждый http-заголовок представляется парой *имя/значение*, разделителем служит двоеточие, а строки заголовка разделяются парой символов *возврат каретки/переход на новую строку*;
- `getResponseHeader(`*`имя_заголовка`*`)` — возвращает строку, соответствующую указанному параметру *`имя_заголовка`* возвращенного сервером ответа (например, `requestResponseHeader("Set-cookie")`);
- `open(`*`метод`*`, `*`url-адрес`*`, `*`асинхронный`*`, `*`логин`*`, `*`пароль`*`)` — инициализирует (параметризует) запрос серверу. Параметр *`метод`* указывает метод передачи (например, `get` или `post`). Параметр *`url-адрес`* задает адрес, по которому будет направлен запрос. Последние три параметра не обязательны. Параметр *`асинхронный`* указывает режим об-

мена и принимает логические значения (`true` соответствует асинхронному режиму и установлено по умолчанию). Параметры *логин* и *пароль* необходимы, когда для указанного URL-адреса требуется http-аутентификация. Если для такого URL-адреса данные параметры не указаны, то пользователю будет предложено ввести их;

- `setRequestHeader(`*имя*`,` *значение*`)` — добавляет http-заголовок, заданный указанными параметрами *имя* и *значение*; например,

```
setRequestHeader(
"Cache-Control", "no-store, no-cache, must-revalidate")
```

- `send(`*содержимое*`)` — передает серверу запрос, параметры которого заданы методом `open()`. Параметр *содержимое* представляет собой строку вида "*имя_параметра1=значение&имя_параметра2=значение*..." для передачи запроса методом `post` или пустую строку или `null` при передаче методом `get`.

Передача данных

Чтобы воспользоваться объектом `XMLHttpRequest`, необходимо сначала создать его экземпляр. В различных браузерах это делается по-разному:

- `var req = new XMLHttpRequest()` — для всех современных браузеров;
- `var req = new ActiveXObject('Msxml2.XMLHTTP')` — для Internet Explorer версий, меньших, чем 7.0;
- `var req = new ActiveXObject('Microsoft.XMLHTTP')` — для самых ранних версий Internet Explorer.

Для обеспечения межбраузерной совместимости в сценарии следует предусмотреть все возможные варианты активизации объекта `XMLHttpRequest`, что мы рассмотрим позже.

Общий порядок активизации `XMLHttpRequest` выглядит следующим образом.

1. Создание экземпляра объекта `XMLHttpRequest`.
2. Установка обработчиков событий.
3. Открытие соединения с сервером.
4. Отправка запроса на сервер.

В листинге 15.25 приведен код, иллюстрирующий применение объекта `XMLHttpRequest`. Здесь инициализация обмена с сервером происходит щелчком на кнопке. При загрузке в браузер данного кода первоначально отображается текст и кнопка с надписью "Получить данные с сервера". Щелчок на этой кнопке приводит к взаимодействию с сервером посредством указанного объекта. Говоря точнее, выполняется следующая последовательность действий:

1. Создается экземпляр объекта `XMLHttpRequest`. Оператор `try...catch` обработки исключительных ситуаций используется при выборе способа создания экземпляра объекта. Для разнообразия возможность применения современного способа осуществляется проще — с помощью оператора `if`.
2. Пользовательская функция `Load()` является обработчиком события `onclick` для кнопки. Она устанавливает обработчик `receive()` события `onreadystatechange` (изменение состояния), открывает методом `open()` соединение с серверным сценарием, расположенным в файле myprogram.php, и отправляет запрос методом `send()`.

3. В случае успеха данной операции сервер должен ответить браузеру. Методу `open()` в рассматриваемом примере передаются три параметра: метод передачи (в данном примере `get`), URL-адрес файла с серверным сценарием и значение `true`, задающее асинхронный режим взаимодействия. Впрочем, `true` — значение по умолчанию. Методу `send()` можно передать в качестве параметра какие-то данные для сервера, если задан метод передачи `post`, а для метода `get` следует указать пустую строку или `null`.
4. При получении от сервера ответа возникает событие `onreadystatechange` и вызывается ранее назначенный обработчик — функция `receive()`, которая анализирует коды состояния объекта `XMLHttpRequest`, т. е. проверяет, успешно ли завершен запрос к серверу.
5. Если загрузка данных с сервера завершена (код 4) и не было ошибок (код 200), то функция `receive()` заменяет содержимое элемента `<div>` полученными от сервера данными, хранящимися в свойстве `responseText`. Это делается посредством свойства `innerHTML`.

Листинг 15.25. Пример использования объекта `XMLHttpRequest`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<head><title>Применение XMLHttpRequest</title>
<script type="text/javascript">
var req=false;
  if(window.XMLHttpRequest) { // поддерживается ли XMLHttpRequest ?
       req=new XMLHttpRequest(); // для IE7+ и других браузеров
  } else {
     try {
      req=new ActiveXObject('Msxml2.XMLHTTP'); // для IE версий, меньших 7
     } catch (e) {
         req=new ActiveXObject('Microsoft.XMLHTTP'); // для старых IE
     }
  }
if (!req) // если объект XMLHttpRequest не поддерживается
 alert("Объект XMLHttpRequest не поддерживается данным браузером");

function Load() { //
  if (!req) return; // если объект XMLHttpRequest не поддерживается
  /* назначаем обработчик событию готовности объекта: */
     req.onreadystatechange = receive;
  /* открываем объект для взаимодействия:
     передача методом get;
     серверный сценарий  в файле myprogram.php;
     режим — асинхронный:
  */
     req.open("get", 'myprogram.php',true); // открываем соединение
     req.send(null);                        // посылаем запрос серверу
}
```

```
function receive() { // получение данных от сервера
  if (req.readyState == 4){    // если запрос завершен
      if (req.status == 200) { // если запрос завершен без ошибок
     /* Отображаем ответ сервера в HTML-элементе документа */
        document.getElementById('content').innerHTML = req.responseText;
      } else {
          alert("Ошибка "+ req.status+":\n" + req.statustext);
      }
  }
}
</script>
</head>
<body>
Щелчок на кнопке приведет к загрузке в данный документ
ответа сервера.
<input type="button" value="Получить данные с сервера" onclick="Load()"/>
<br/>
Данные, принятые от сервера:
<div id="content" style="color:#ff0000"></div>
</body>
</html>
```

Опробовать приведенный пример можно на локальном компьютере с установленными Web-сервером (например, Microsoft Internet Information Services или Apache) и модулем PHP. Для этого в домашнем каталоге Web-сервера создайте папку, например, TEST и разместите в ней файл с кодом из листинга 15.25 (пусть это будет test.htm), а также файл myprogram.php с серверным сценарием на языке PHP. Для первого эксперимента серверный сценарий можно сделать очень простым:

```
<?php
echo 'В первых строках <i>своего письма</i> передаю всем <b>привет</b>';
?>
```

Данный сценарий возвращает браузеру строку текста, которая должна быть отображена в документе test.htm. Чтобы избежать проблем с кириллицей, файл myprogram.php с серверным сценарием сохраните в кодировке UTF-8.

Если в адресную строку браузера ввести **localhost/TEST/test.htm**, то в окне браузера должны появиться текст и кнопка. Щелчок на этой кнопке отобразит в нижней части документа строку текста, переданную сервером (рис. 15.12).

В сценарии листинга 15.25 был применен метод `get` запроса к серверу. Однако в Internet Explorer, в отличие от других браузеров, ответ сервера кэшируется, что препятствует обновлению данных на загруженной странице, даже если они были обновлены на сервере. Нередко можно услышать негодование разработчиков по данному вопросу. Тем не менее, данная политика Internet Explorer вполне разумна. Если некоторый запрос активизируется впервые, то ответ на него кэшируется: пара запрос-ответ сохраняется в памяти браузера. При последующем запросе проверяется, нет ли на него ответа в кэше. Если есть, то предлагается использовать именно его, а в противном случае — вновь обратиться к серверу. Как именно поступить, определяется характеристиками запроса.

В случае метода `get` запрос полностью характеризуется строкой, содержащей URL, содержащей адрес, а также, возможно, передаваемые данные. Если же в URL указан только адрес, то все запросы по данному URL будут формально одинаковы, и, следовательно, начиная со второго запроса, ответ на него следует поискать в кэше, а не обращаться к серверу. В данном случае браузеру Internet Explorer нет дела до того, что на сервере по тому же адресу данные уже изменились. Такая политика направлена на сокращение трафика.

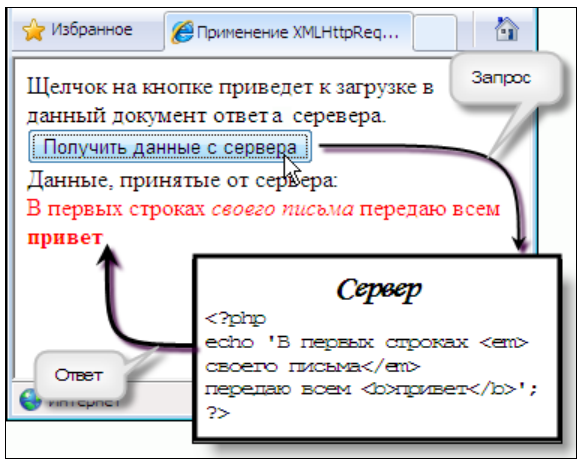


Рис. 15.12. Результат взаимодействия с сервером сценария из листинга 15.25

Простой метод избежать кэширования get-запроса состоит в том, чтобы к URL-адресу дописать в качестве передаваемых данных какое-нибудь значение, изменяющееся между вызовами метода `open()`, например, текущее время или случайное число. Вот примеры:

- `var curentTime=new Date();`
- `req.open("get", "myprogram.php?"+curentTime);`
- `req.open("get", "myprogram.php?"+Math.random())`

Запросы такого вида будут отличаться друг от друга при каждом вызове метода `open()` и, следовательно, ответы на них будут браться не из кэша браузера, а приниматься от сервера. В отличие от get-запросов, ответы на post-запросы не кэшируются, но мы их рассмотрим позже.

Клиенту можно передавать данные и других типов, например графические изображения, звук, видео, текстовые и HTML-файлы, а также сценарии на языке JavaScript. Это делается аналогичным способом. Так, передача сценария для выполнения браузеру позволяет разгрузить сервер.

Если в функции `receive()` в листинге 15.25 вместо выражения

```
document.getElementById('content').innerHTML = req.responseText
```

вставить выражение

```
eval(req.responseText)
```

то браузер сможет выполнить скрипт на языке JavaScript, полученный от сервера. Например, следующий серверный сценарий возвращает браузеру строку, содержащую три выражения на языке JavaScript:

```
<?php
echo 'alert("Привет!"); y=2+3; alert(y)';
?>
```

Получив данную строку, браузер выполнит ее с помощью встроенной функции `eval()`: выведет диалоговое окно с сообщением "Привет!", присвоит переменной значение 5 и отобразит диалоговое окно с этим значением.

Серверный сценарий может не только вернуть клиенту некоторые данные, но еще и передать скрипт, который должен их обработать и, при необходимости, отобразить в окне браузера.

Клиент может запросить текстовый csv-файл, содержащий данные с табличной структурой, и как-то их обработать на своей стороне, например, отсортировать, отфильтровать и отобразить на странице. Функции для обработки двумерных массивов приведены в листинге 15.26.

Листинг 15.26. Функции сортировки и фильтрации двумерных массивов

```
function asort(arr,fld,direct){
/* Сортировка двумерного массива:
   arr – двумерный массив
   fld — номер поля (столбца), по значениям которого требуется
         отсортировать строки (нумерация столбцов начинается с 0);
         если не указано, то 0
   direct — порядок сортировки (если пусто, то по возрастанию,
                                иначе — по убыванию)
   Возвращает отсортированный массив
*/
  if (!fld) fld=0;
  if (!direct) {direct=1;} else direct=-1;
  return arr.sort(comp)
  function comp(x,y){
    if (x[fld]<y[fld]) return -direct;
    if (x[fld]>y[fld]) return direct;
    if (x[fld]===y[fld]) return 0
  }
}
/*******************************************************/

function afilter(arr,criterion,count,field){
/* Фильтрация двумерного массива:
   arr – двумерный массив
   criterion — строка с выражением условия выбора строк,
               в которой поля обозначаются в виде fld[0], fld[1] и т. п.;
               например, "fld[0]=='2'||fld[1]=='Маша'"
   count — количество строк, не более которого следует отфильтровать;
           если пусто, то принимается число всех строк
```

```
   field — значение для обозначения поля в criterion;
           если не указано, то имеется в виду "fld" (на всякий случай)
   Возвращает массив отобранных строк
*/
  var fltr=[], curcount=0;
  if(!count) count=arr.length;
  if(!field) field="fld";
  var re=new RegExp(field,"g");
  criterion=criterion.replace(re,"arr[i]");
  for(var i=0;i<arr.length;i++){
     if (eval(criterion)){
       fltr=fltr.concat(arr[i]);
       curcount++;
       if(curcount>=count) break
     }
  }
  return fltr
}
```

Как уже отмечалось, передачу каких-либо данных серверу можно осуществить и методом `post`. Однако при использовании метода `post`, в отличие от `get`, необходимо установить тип содержимого запроса с помощью метода `setRequestHeader()`.

В листинге 15.27 приведен код сценария, который передает на сервер данные методом `post` и отображает полученный ответ. Здесь передаются две переменные (`name` и `pw`), представленные в виде строки `"name=Вася&pw=1234567"`. Обратите внимание на использование метода `setRequestHeader("Content-Type","application/x-www-form-urlencoded")`.

Листинг 15.27. Передача данных на сервер методом `post`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<head><title>Применение XMLHttpRequest</title>
<script type="text/javascript">
var req=false;
if(window.XMLHttpRequest) { // поддерживается ли XMLHttpRequest ?
      req=new XMLHttpRequest(); // для IE7+ и др.
} else {
   try {
     req=new ActiveXObject('Msxml2.XMLHTTP'); // для IE версий, меньших 7
   } catch (e) {
       req=new ActiveXObject('Microsoft.XMLHTTP'); // для старых IE
   }
}
if (!req) // если объект XMLHttpRequest не поддерживается
    alert("Объект XMLHttpRequest не поддерживается данным браузером");

function Load() {
    if (!req) return; // если объект XMLHttpRequest не поддерживается
    /* назначаем обработчик событию готовности объекта: */
     req.onreadystatechange = receive;
```

```
    /* открываем объект для взаимодействия:
       передача методом post;
       серверный сценарий  в файле myprogram.php;
       режим — асинхронный:
    */
     req.open("post", 'myprogram.php',true); // открываем соединение
    /* устанавливаем тип данных */
     req.setRequestHeader("Content-Type",
                          "application/x-www-form-urlencoded");
    /* данные, передаваемые на сервер: */
     var data="name=Вася&pw=1234567";
     req.send(data); // посылаем запрос серверу
}

function receive() { // получение данных от сервера
  if (req.readyState == 4){// если запрос завершен
     if (req.status == 200) { // если запрос завершен без ошибок
     /* Отображаем ответ сервера в HTML-элементе документа: */
        document.getElementById('content').innerHTML = req.responseText;
     } else {
        alert("Ошибка "+ req.status+":\n" + req.statustext);
     }
  }
}
</script>
</head>
<body>
Щелчок на кнопке приведет к загрузке в данный документ
ответа сервера.
<input type="button" value="Получить данные с сервера" onclick="Load()"/>
<br/>
Данные, принятые от сервера:
<div id="content" style="color:#ff0000"></div>
</body>
</html>
```

Чтобы проверить работу этого примера, на стороне сервера (в файле myprogram.php) можно использовать следующий сценарий на языке PHP:

```
<?php
$x=$_POST['name']."<br/>".$_POST['pw'];
echo $x;
?>
```

Данный сценарий просто возвращает браузеру значения переменных `name` и `pw`, переданные серверу методом `post` и хранящиеся на сервере в массиве `$_POST`.

Теперь рассмотрим случай, когда на сервере данные время от времени обновляются, а клиент должен их получать автоматически почти в реальном времени, без инициализации обмена с помощью кнопки и т. п. Для этой цели достаточно воспользоваться методом `setInterval(Load, задержка)`, который будет периодически с заданной задержкой

во времени вызывать функцию, например, `Load()`, осуществляющую взаимодействие с сервером. Подробнее о методах управления вычислительными процессами во времени рассказано в *разд. 15.8*.

В качестве метода передачи данных лучше использовать `post`, при котором браузеры не кэшируют запрошенные данные и тем самым не задерживают их обновление на загруженной странице. Кроме того, период обновления (задержка), устанавливаемый в качестве второго параметра метода `setInterval()`, не должен быть слишком малым. В противном случае могут возникнуть ошибки выполнения сценария. Точнее, временной интервал между вызовами функции `Load()` должен быть таким, чтобы подача запроса и получение ответа успели завершиться. При работе с сервером на локальном компьютере минимально необходимый интервал может составить от одной до нескольких секунд. При работе с удаленным сервером в практических приложениях данный интервал может принимать значения от нескольких секунд до нескольких минут и больше.

В листинге 15.28 приведен код документа, в котором периодически запрашиваются данные, возвращаемые серверным сценарием. Для тестирования данного кода в файле myprogram.php можно разместить серверный сценарий, возвращающий обновляемые данные, например, текущее системное время:

```
<?php
echo 'Время'.date("H:i:s");
?>
```

Вид документа листинга 15.28 в окне браузера показан на рис. 15.13. В данном примере временная задержка между вызовами функции `Load()` составляет 2 с. Попробуйте ради эксперимента установить меньшее значение, например, 1 с или, наоборот, большее — 3 или 10 с.

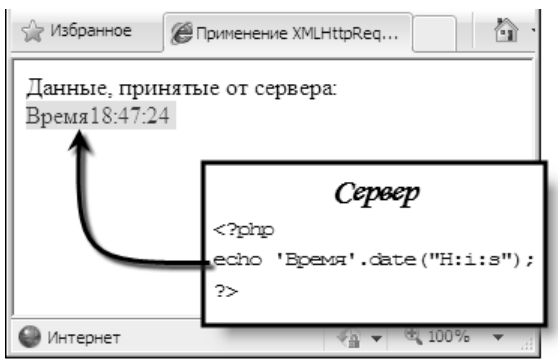


Рис. 15.13. Периодический доступ к серверу (листинг 15.28)

Листинг 15.28. Периодический опрос сервера

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 transitional//EN"
"http://www.w3.org/TR/HTML4.01/loose.dtd">
<head><title>Применение XMLHttpRequest</title>
<meta name="pragma" content="no-cache">
<meta name="EXPIRES:" content="Wed, 23 Feb 2005 08:25:00 GMT">
```

```
<script type="text/javascript">
var req=false;
if(window.XMLHttpRequest) { // поддерживается ли XMLHttpRequest ?
    req=new XMLHttpRequest(); // для IE7+ и др.
} else {
    try {
     req=new ActiveXObject('Msxml2.XMLHTTP'); // для IE версий, меньших 7
    } catch (e) {
     req=new ActiveXObject('Microsoft.XMLHTTP'); // для старых IE
    }
}
if (req){// если объект XMLHttpRequest поддерживается
    setInterval("Load()",2000); // периодический вызов Load() через 2 с
}else {
    alert("Объект XMLHttpRequest не поддерживается данным браузером")
}

function Load() { //
  if (!req) return;// если объект XMLHttpRequest не поддерживается
  /* назначаем обработчик событию готовности объекта: */
  req.onreadystatechange = receive;
  /* открываем объект для взаимодействия:
     передача методом post;
     серверный сценарий  в файле myprogram.php;
     режим — асинхронный:
     */
  req.open("post", 'mypogram.php',true); // открываем соединение
  req.setRequestHeader("Content-Type",
                       "application/x-www-form-urlencoded");
     req.send("");          // посылаем запрос серверу
}
function receive() { // получение данных от сервера
  if (req.readyState == 4){  // если запрос завершен
    if (req.status == 200) { // если запрос завершен без ошибок
      /* Отображаем ответ сервера в HTML-элементе документа */
      document.getElementById('content').innerHTML = req.responseText
    } else {
      alert("Ошибка "+ req.status+":\n" + req.statustext);
    }
  }
}
</script>
</head>
<body>
Данные, принятые от сервера:
<div id="content"
     style="color:#ff0000;bacground:yellow; width:150px">
</div>
</body>
</html>
```

При необходимости применить метод `get` в случае периодического опроса сервера следует учесть, что браузер Internet Explorer кэширует ответы сервера, а это не позволяет отслеживать изменения данных. Тем не менее преодолеть кэширование возможно, и мы уже рассматривали это при обсуждении листинга 15.25.

В широко известной JavaScript-библиотеке jQuery имеется, кроме прочего, набор функций для работы с сервером в асинхронном режиме. Официальный сайт, на котором можно получить данную библиотеку и документацию к ней (на английском языке), находится по адресу **http://docs.jquery.com/Main_Page**. Перевод этой документации на русский язык можно найти на сайте **http://jquery-docs.ru**. На указанных и многих других сайтах все, что относится к использованию объекта `XMLHttpRequest`, собрано под названием AJAX.

15.7.2. AJAX

Систематическое применение объекта `XMLHttpRequest` в клиентских сценариях для организации асинхронного информационного обмена между загруженным в браузер документом и сервером получило название AJAX (Asynchronous Javascript and XML — асинхронный JavaScript и XML). Главная цель технологии AJAX состоит в том, чтобы обеспечить динамическое обновление содержимого Web-страницы без ее полной перезагрузки, а также организовать взаимодействие с сервером, при котором можно рационально распределить вычислительные ресурсы между ним и клиентом. Именно широкое использование в настоящее время уже давно поддерживаемого браузерами объекта `XMLHttpRequest` обусловило появление модного ныне термина "AJAX".

Применение `XMLHttpRequest` имеет и недостатки. Во-первых, документы, сформированные динамически посредством сценария, оказываются недоступными для поисковых систем. Во-вторых, такие страницы не регистрируются браузером в журнале посещенных, а потому для возврата вы не можете воспользоваться кнопкой **Назад**. Имеются и другие эффекты, вызывающие нежелательные осложнения.

Итак, главное в AJAX — возможность динамического обновления Web-страниц без их перезагрузки. Однако этой цели можно достичь не только посредством `XMLHttpRequest`. Так, с помощью фреймов (обычных и плавающих), элементов `<img>`, `<object>` и `<embed>` можно динамически изменять содержимое частей страницы без перезагрузки ее в целом. В самом деле, загрузка данных в перечисленные элементы происходит без перезагрузки страницы, в которой эти элементы расположены. С помощью параметров CSS элементы `<iframe>`, `<img>`, `<object>` и `<embed>` можно сделать невидимыми, чтобы скрыть от глаз пользователя работу с сервером. Объект `Image` может использоваться для взаимодействия с сервером не только ради предварительной загрузки изображений (см. *разд. 15.6.5*). При этом загрузка в него данных происходит без видимых признаков. Так что технология AJAX задействует весь арсенал средств, обеспечивающих динамическое взаимодействие с сервером.

15.8. Управление во времени

Управление вычислительными процессами во времени обеспечивают методы объекта `window`: `setInterval()`, `clearInterval()`, `setTimeout()` и `clearTimeout()`. Вы можете периодически, через заданный интервал, вызывать одну или несколько функций, а также задерживать их вызов.

Для организации периодического выполнения некоторого выражения или функции служит метод `setInterval()` объекта `window`, имеющий следующий синтаксис:

`setInterval(`*код*`, `*период*`)`.

Первый параметр *код* возможен в двух вариантах:

- ссылка на функцию — имя функции без круглых скобок; например, `setInterval(myfunc,100)`;
- строка, содержащая код на языке JavaScript (в частности, вызов функции); например, `setInterval("myfunc()",100)`.

Второй параметр *период* — целое число, указывающее временну́ю задержку в миллисекундах перед последующими выполнениями кода, заданного в первом параметре.

> ***Примечание***
>
> В методе `setInterval()` с заданием в качестве первого параметра ссылки на выполняемую функцию можно еще указать третьим и последующими параметрами аргументы, передаваемые данной функции при каждом ее вызове. Однако это не будет работать в Internet Explorer. Поэтому для обеспечения межбраузерной совместимости при необходимости передать функции параметры следует использовать не ссылку на нее, а строковое представление.

Метод `setInterval()` возвращает некоторое целое число — идентификатор временного интервала, который может быть использован в дальнейшем для прекращения выполнения процесса, запущенного с помощью данного метода (см. далее метод `clearInterval()`).

Пусть, например, требуется, чтобы некоторая функция `myfunc()` выполнялась периодически через 0,5 с, тогда в сценарии следует записать следующее выражение:

```
var myint=setInterval("myfunc()", 500).
```

Здесь идентификатор интервала сохранен в переменной `myint`, чтобы в последующем иметь возможность остановить запущенный процесс периодического вызова функции `myfunc()`.

Выражение, переданное методу `setInterval()`, будет периодически выполняться сколь угодно долго. Если оно осуществляет, например, приращение координат какого-нибудь видимого элемента документа, то этот элемент станет перемещаться в окне браузера. Даже если он скроется за кромкой экрана, метод `setInterval()` все равно будет продолжать работать.

Для остановки запущенного временного процесса служит метод

`clearInterval(`*идентификатор*`)`,

который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом `setInterval()`, например:

```
var myproc=setInterval("myfunc(), 100");
if (confirm("Прервать процесс ?")) clearInterval(myproc);
```

В листинге 15.29 приведен пример реализации движения заголовка (элемента `<h1>`) по горизонтали на основе методов `setInterval()` и `clearInterval()`. Периодически вызываемая

функция `move()` осуществляет только приращение координаты `left` таким образом, чтобы она не была меньше 10 px и больше 500 px.

Листинг 15.29. Движение заголовка

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Применение setInterval()</title></head>
<body>
<h1 id="myzag" style="position:absolute; left:10px;top:50px">
Движение — это жизнь
</h1>
<input type="button" onclick="clearInterval(myint)" value="Остановить"/>
</body>
<script type="text/javascript">
var flag=false; // для задания направления
var myint=setInterval("move()",10) // запуск перемещения
function move(){
 var x=parseInt(document.getElementById("myzag").style.left);
 if (x==500) {flag=true};
 if (x==10) flag=false;
 if (flag) d=-1 else d=1;
 document.getElementById("myzag").style.left=(x+d)+"px";
}
</script>
</html>
```

Выполнить код с некоторой временной задержкой позволяет метод `setTimeout()`, реализующий так называемый таймер и имеющий следующий синтаксис:

`setTimeout(`*код*`, `*задержка*`)`.

Первый параметр представляет собой ссылку на функцию или строку, содержащую код (как и для метода `setInterval`). Второй параметр — целое число, указывающее временную задержку выполнения кода (в миллисекундах), заданного в первом параметре.

> ***Примечание***
>
> При записи в методе `setTimer()` в качестве первого параметра ссылки на функцию можно указать третьим и последующими параметрами аргументы, передаваемые данной функции при каждом ее вызове. Однако это не будет работать в Internet Explorer. Поэтому для обеспечения межбраузерной совместимости при необходимости передать функции параметры следует использовать не ссылку на нее, а строковое представление.

Метод `setTimeout()` возвращает некоторое целое число — идентификатор таймера, который может быть использован в дальнейшем для отмены задержки выполнения процесса, запущенного с помощью данного метода.

Пусть, например, требуется, чтобы некоторая функция `myfunc()` выполнялась спустя 1 с, тогда в сценарии необходимо записать следующее выражение:

```
var mytimer=setTimeout("myfunc()", 1000).
```

Примечание

Помните, что это выражение задерживает выполнение не всех последующих выражений сценария, а лишь функции `myfunc()`.

Для отмены задержки процесса, запущенного с помощью метода `setTimeout()`, предусмотрен метод

`clearTimeout(`*идентификатор*`)`,

который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом `setTimeout()`.

В следующем XHTML-документе имеются две кнопки (листинг 15.30). Щелчок на первой кнопке через 5 с открывает новое окно и загружает в него документ. Однако это действие можно отменить с помощью второй кнопки, если щелкнуть на ней, пока новое окно еще не открыто.

Листинг 15.30. Использование временной задержки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Применение setTimeout() и clearTimeout</title>
<script type="text/javascript">
 var myproc;
 function start(){
  return myproc=
         setTimeout("window.open('http://dunaevv1.narod.ru')", 5000)
 }
</script>
</head>
<body>
<button id="start" onclick="myproc=start()">
    Загрузить с задержкой 5 с
</button>
<button onclick="clearTimeout(myproc)">
    Отменить загрузку
</button>
</body>
</html>
```

Примечание

При работе с методами `setInterval()` и `setTimeout()` следует иметь в виду, что их вторые числовые параметры задают лишь приблизительные значения временных задержек.

С помощью метода `setTimeout()` можно организовать многократный вызов функции, причем с ускорением или замедлением. Для этого определение функции должно содержать вызов самой себя посредством метода `setTimeout()` (так называемый рекурсивный вызов) (листинг 15.31).

Листинг 15.31. Реализация многократного вызова функции

```
var t=100;dt=5;
myfunc();
function myfunc(){
    // здесь некоторый код;
    t+=dt; // временная задержка
    setTimeout("myfunc()",t); // рекурсивный вызов
}
```

В данном случае функция `myfunc()` будет вызываться многократно с нарастающим интервалом между вызовами, т. е. с замедлением. Если величину `dt` сделать отрицательной, то вызовы будут происходить с ускорением.

ГЛАВА 16

Примеры сценариев на JavaScript

Если в предыдущих главах скрипты приводились в качестве иллюстраций работы с различными объектами документа, то теперь мы рассмотрим примеры, требующие для своей реализации комплексного применения нескольких технологических средств. Обычно одну и ту же задачу можно решить несколькими способами. Выбор наилучшего из них часто определяется многими параметрами и не всегда однозначен.

16.1. Подсветка кнопки

Подсветка кнопки — некий визуальный эффект (например, изменение цвета или надписи), возникающий при наведении (и/или уходе) указателя мыши на изображение кнопки. Реализовать данную возможность легко, используя псевдокласс `:hover` из CSS. Требуемый эффект достигается путем изменения значений стилевых параметров `background` (фоновый цвет или изображение). Вместе с тем, это можно сделать и с помощью скриптов. Достаточно создать коды обработчиков для двух событий — `onmouseover` (наведение указателя) и `onmouseout` (уход указателя). Эти обработчики должны изменить значение соответствующего свойства элемента, представляющего кнопку. В листинге 16.1 приведен простейший пример изменения цвета кнопки от синего к голубому и наоборот. Обратите внимание на ключевое слово `this` (этот) для указания на объект, соответствующий элементу, для которого написан обработчик события.

Листинг 16.1. Подсветка кнопки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Подсветка кнопки</title></head>
<body>
<input type="button" value="Кнопка"
       style="background-color:#0000aa; color:#ffffff"
       onmouseover="this.style.background='#0000ff'"
       onmouseout ="this.style.background='#0000aa'"/>
</body>
</html>
```

Теперь рассмотрим создание более сложного эффекта: при наведении указателя мыши на кнопку графический фон последней перемещается и вновь возвращается в исходное положение. Если в качестве графического фона выбрать картинку с градиентной заливкой, то возникает эффект "цветового перелива". Разумеется, можно придумать и какое-либо иное внешнее проявление, а сейчас мы рассмотрим реализацию именно переливающихся цветов.

Создадим кнопку посредством контейнера `<div>` с заданными размерами, внутри которого расположим графическое изображение (`<img src=.../>`), несколько бóльших размеров. Контейнер `<div>` должен иметь стилевой параметр `overflow:hidden`, чтобы части графического изображения, выступающие за его границы, не были видны. При наведении указателя мыши на картинку последняя начинает постепенно перемещаться, пока не займет заданного положения. Для обеспечения постепенности перемещения нам пригодится метод `setInterval()` (см. *разд. 15.8*). Как только картинка достигнет конечной точки своего "путешествия", мы вернем ее в исходное положение. Схема данной идеи представлена на рис. 16.1, а ее реализация — в листинге 16.2.

В данном примере размеры `<div>` устанавливаются равными 150×20 px, а фонового графического изображения — 150×40 px, т. е. высота картинки в два раза больше высоты контейнера `<div>`. При наведении указателя мыши картинка перемещается сверху вниз. Скорость и плавность перемещения задаются с помощью второго параметра метода `setInterval()` и величины приращения значения стилевого параметра `top` (в данном примере — 10 мс и 1 px соответственно).

Листинг 16.2. Перемещение фонового изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Подсветка</title>
<style type="text/css">
div.hilight {
      border: solid 1px;
      position: relative;
      width: 150px;
      height: 20px;
      overflow: hidden; /* все, что не помещается, не видно */
      text-align: center;
}
</style>
</head>
<body>
<div class="hilight" >
<img id="bg" src="fon.jpg"
  onmouseover='myint=setInterval("move()",10)'
  style="position:relative;top:-20px;" />
<a href="http://anyserver.ru"
  style="position:relative;top:-42px;
  color:white;
```

```
  text-decoration:none">
  Кнопка
</a>
</div>
<script type="text/javascript">
var myint;
function move(){ // перемещение картинки
  document.getElementById('bg').style.top=
     (parseInt(document.getElementById('bg').style.top)+1)+"px";
  if (parseInt(document.getElementById('bg').style.top)>0) {
     clearInterval(myint); // прекращаем перемещение
       /* Возвращаем картинку в исходное положение */
     document.getElementById('bg').style.top="-20px";
  }
}
</script>
</html>
```

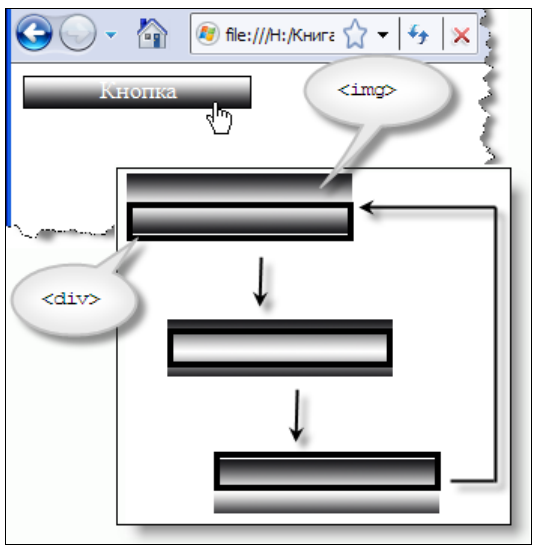


Рис. 16.1. Схема эффекта цветового перелива кнопки

В листинге 16.2 кнопка выполняла роль ссылки, поскольку контейнер `<div>` кроме картинки содержал тег `<a>`. Для кнопки текст ссылки обычно не подчеркивают, поэтому был использован стилевой параметр `text-decoration:none`. Применение ссылки автоматически решает задачу вида указателя мыши (курсора). Если же кнопка служит для вызова некоторого сценария и тег ссылки не применяется, то вид указателя мыши следует установить принудительно.

Например, для элемента `<img>` в обработчик события `onmouseover` в качестве первого выражения можно добавить следующее:

```
onmouseover='this.style.cursor="pointer";
```

Кроме того, в этот же элемент следует добавить еще один обработчик:

```
onmouseout='this.style.cursor="auto"'
```

В результате при наведении мыши на кнопку ее указатель примет форму руки, а при уходе — обычную форму по умолчанию.

16.2. Меню

Раскрывающееся или ниспадающее меню — часто встречающийся на Web-страницах элемент пользовательского интерфейса. Его создание с помощью CSS (благодаря псевдоклассу `:hover`) рассматривалось в *разд. 7.1.1* и *7.1.2*. В данном разделе мы займемся разработкой меню на основе скриптов.

16.2.1. Моментально раскрывающееся вертикальное меню

Сначала займемся вертикальным раскрывающимся меню, которое в исходном состоянии выглядит подобно кнопке (прямоугольник с надписью). При наведении указателя мыши на данный прямоугольник меню раскрывается, показывая опции (пункты) своего подменю. При уходе указателя мыши с прямоугольника раскрывшегося меню последнее сворачивается и принимает исходное состояние.

Проще всего сделать меню, которое раскрывается моментально, т. е. весь набор его опций (подменю) появляется одновременно. Для этого подойдут два контейнера `<div>`, из которых второй вложен в первый и в исходном состоянии невидим. Первый `<div>` содержит, например, некий текст и всегда отображается. Наведение на него указателя мыши (событие `onmouseover`) изменяет стилевой параметр видимости `display` подменю так, чтобы оно стало видимым. Уход указателя мыши (событие `onmouseout`) обрабатывается как изменение параметра `display` так, чтобы подменю исчезло для наблюдателя.

В листинге 16.3 приведен пример кода для описанного меню. Здесь подменю содержит ссылки, отображение которых регламентируется CSS. Так что, выделение опций подменю при наведении на них указателя мыши и при щелчках определяется стилевыми параметрами для тега `<a>`. Параметр `z-index` установлен заведомо большим, чтобы абсолютно позиционируемое меню располагалось поверх других элементов страницы. Разумеется, вы можете выбрать его значение, а также другие стилевые параметры исходя из собственных обстоятельств и целей. Вид рассмотренного меню в браузере показан на рис. 16.2.

Листинг 16.3. Моментально раскрывающееся вертикальное меню

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Меню</title>
<style type="text/css">
.mainmenu { /* главное меню */
     border:solid 1px #000000;
     position:absolute;
     top:15px;
```

```
      width:150px;
      background:#00aaff;
      text-align:center;
      font-weight:bold;
      color:white;
      z-index:1000;
}
.submenu { /* подменю */
      display:none; /* не видно */
      background:#00ffff;
      text-align:left
}
a {
      padding-left:10px;
      display:block;
      color:#000000;
      text-decoration:none
}
a:visited {color:#808080}
a:hover{color:#ffff00; background:blue}
a:active {color:#ffffff}
</style>
</head>
<body>
<div class="mainmenu"
      onmouseover='document.getElementById("sm").style.display="block"'
      onmouseout='document.getElementById("sm").style.display="none"'>
  Поисковики
  <div id="sm" class="submenu" >
    <a href="http://www.rambler.ru/">Rambler</a>
    <a href="http://www.yandex.ru/">Яндекс</a>
    <a href="http://www.google.com/">Google</a>
  </div>
</div>
</body>
</html>
```

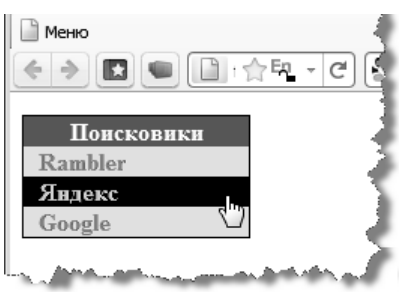


Рис. 16.2. Вид меню листинга 16.3

16.2.2. Плавно раскрывающееся меню

Опции плавно раскрывающегося меню появляются постепенно, с некоторой временной задержкой. Один из возможных способов реализации данного меню довольно прост и основан на использовании двух контейнеров `<div>`. Первый `<div>` имеет размеры, соответствующие исходному состоянию и достаточные для того, чтобы отобразить содержащийся в нем некоторый текст. Для данного `<div>` назначается стилевой параметр `overflow:hidden`, чтобы обеспечить невидимость подменю в исходном состоянии. Иначе говоря, первый `<div>` выглядит на экране как небольшой прямоугольник с надписью. Второй `<div>` (подменю) содержится в первом и позиционирован относительно него так, чтобы в исходном состоянии он был невидим (выходил за пределы первого `<div>`). При наведении указателя мыши на первый `<div>` обработчик события `onmouseover` постепенно увеличивает его размеры, а следовательно, расширяет и область видимости второго `<div>`. Постепенность обеспечивается методом `setInterval()` (см. *разд. 15.8*). При уходе указателя мыши с прямоугольника меню срабатывает обработчик события `onmouseout`, который плавно уменьшает размеры первого `<div>` до исходных значений. Код данного меню приведен в листинге 16.4.

Листинг 16.4. Плавно раскрывающееся меню

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Меню</title>
<style>
.mainmenu { /* главное меню */
     border:solid 1px #000000;
     position:absolute;
     width:170px;
     height:20px;
     background:#00aaff;
     overflow-y:hidden;
     text-align:center;
     color:white;
     z-index:1000;
}
.submenu { /* подменю */
     position:relative;
     top:10px;
     width:170px;
     height:100px;
     background:#00ddff;
}
.submenu span {
     display:block;
}
.submenu span:hover {background:#00bbee}
a {
     color:#000000;
```

```
     text-decoration:none;
}
a:visited {color:#cccccc}
a:hover{color:#ffff00}
a:active {color:#00ffff}
</style>
</head>
<body>
<div class="mainmenu" id="mm"
     onmouseover="menuDown('mm')"
     onmouseout="menuUp('mm')" style="height:20px">
  Меню
  <div class="submenu">
     <span><a href="http://www.rambler.ru">Rambler</a></span>
     <span><a href="http://www.yandex.ru">Яндекс</a></span>
     <span><a href="http://www.google.ru">Google</a></span>
  </div>
</div>
</body>
<script type="text/javascript">
var myint;  // для хранения идентификатора интервала
function menuDown(idmenu){ // раскрытие меню
  clearInterval(myint);
  myint=setInterval("move1('"+idmenu+"')",10);
}
function menuUp(idmenu){ // свертывание меню
  clearInterval(myint);
  myint=setInterval("move2('"+idmenu+"')",10);
}

function move1(idmenu){ // смещение вниз
  document.getElementById(idmenu).style.height=
      (parseInt(document.getElementById(idmenu).style.height)+2)+"px";
  if (parseInt(document.getElementById(idmenu).style.height)>100) {
      document.getElementById(idmenu).style.height="100px";
  }
}
function move2(idmenu){ // смещение вверх
  document.getElementById(idmenu).style.height=
      (parseInt(document.getElementById(idmenu).style.height)-2)+"px";
  if (parseInt(document.getElementById(idmenu).style.height)<20) {
      document.getElementById(idmenu).style.height="20px";
  }
}
</script>
</html>
```

В рассмотренном примере размеры (`width` и `height`) и приращение высоты для элемента `<div>`, а также временной параметр метода `setInterval()`, которые вместе определяют скорость и плавность раскрытия/свертывания меню, подбирались экспериментально.

Функции moveDown() и moveUp() принимают в качестве параметра значение атрибута id внешнего элемента , содержащего заголовок меню и внутренний с опциями подменю. Это позволяет использовать один и тот же код для обеспечения работы нескольких меню на странице. Однако в таком случае все подменю будут иметь одинаковую высоту (в примере 100 px). Для большей универсальности следует передавать функциям moveDown() и moveUp() в качестве параметра, помимо значения id, еще и высоту подменю. Соответствующие изменения в коде функций очевидны.

16.3. Раскрывающийся комбинированный список

Раскрывающийся комбинированный список, создаваемый посредством тегов <select> и <option>, уже рассматривался в *разд. 11.3*. Здесь мы остановимся на работе с данным элементом пользовательского интерфеса с помощью сценария.

В листинге 16.5 приведен пример, в котором из раскрывающегося списка можно выбрать сайт для последующего перехода на него (рис. 16.3). Сценарий определяет номер выбранной пользователем опции (свойство selectedIndex), а затем загружает в браузер документ по адресу, указанному в соответствующем атрибуте value.

В данном примере элемент <select>, задающий внутреннее содержимое (атрибут value тега <option>) и отображаемые на экране тексты опций списка, реагирует на изменение пользователем выбранной опции, осуществляемое щелчком на ней (событие onchange). Реакция состоит в вызове обработчика данного события — функции myselection(), которая определяет номер (индекс) выбранной опции посредством выражения

```
document.getElementById("mymenu").selectedIndex,
```

а затем по этому индексу определяет значение свойства value соответствующего элемента списка, представленного в объектной модели как элемент коллекции (массива) options. В рассматриваемом примере свойство value хранит URL-адрес документа, который требуется загрузить в браузер. Этот адрес присваивается свойству window.location.href, чего вполне достаточно для действительной загрузки (если, разумеется, такой документ существует и доступен). Очевидно, можно написать и другой сценарий обработки выбранной опции списка.

Листинг 16.5. Раскрывающийся список

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Раскрывающийся список</title></head>
<body>
Перейти на:
<select id="mymenu" onchange ="myselection()" >
 <option value="" selected="selected"/>Выберите сайт
 <option value="http://www.yandex.ru"/>Яндекс
 <option value="http://www.rambler.ru"/>Rambler
```

```
 <option value="http://www.google.ru"/>Google
 <option value="http://dunaevv1.narod.ru"/>Сам себе Web-дизайнер
</select>
</body>
<script type="text/javascript">
function myselection() {
  var i=document.getElementById("mymenu").selectedIndex;
  if (i>0)
     window.location.href=
               document.getElementById("mymenu").options[i].value;
  document.getElementById("mymenu").options[0].selected="selected";
}
</script>
</html>
```

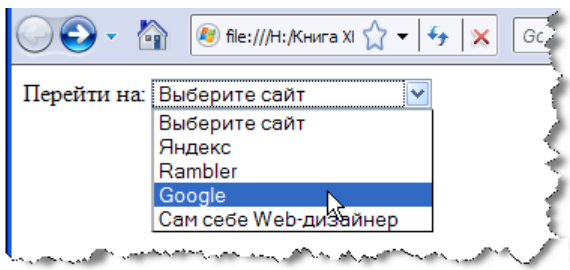


Рис. 16.3. Раскрывающийся список (листинг 16.5)

16.4. Иерархический раскрывающийся список

Иерархические раскрывающиеся списки, создаваемые с помощью тегов `<ul>` и `<li>` и CSS (псевдокласс `:hover`), рассматривались в *разд. 9.4*. Здесь мы решим ту же задачу, но с применением сценария.

В листинге 16.6 приведен код трехуровневого списка, показанного на переднем плане рис. 16.4. Узлы с изображением папок раскрываются и закрываются щелчком. Узлы с маркерами в виде кружков являются терминальными, т. е. нераскрывающимися. В данном примере выбор терминального узла никак не обрабатывается. Однако нетрудно внести в код дополнения, чтобы при щелчке на терминальном узле что-то происходило.

Разумеется, вы можете создать список с любым количеством уровней и с любыми маркерами его пунктов (опций). Раскрывающиеся узлы поддерживаются однотипными скриптами, которые переключают нужные правила CSS. Функция `displayitem(thisobj,xid)` изменяет картинки раскрывающегося пункта списка (передаваемого ей объекта элемента `<li>`) и управляет видимостью подсписка со значением атрибута `id`, равным `xid`. Обработчик `"event.cancelBubble=true"`, прерывающий распространение события, необходим, чтобы щелчок на терминальном пункте списка не приводил к свертыванию последнего. Подробнее о распространении событий см. *разд. 14.4.5*.

Листинг 16.6. Иерархический раскрывающийся список

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Раскрывающийся список</title>
<style type="text/css">
li {
  list-style-image:url("ch.gif");
  list-style-position:inside
}
ul.closed {
  display:none;
}
ul.open {
  display:block;
}
/* Для узлов с потомками: */
li.nodclosed {list-style-image:url("fldclose.gif")}
li.nodopen   {list-style-image:url("fldopen.gif")}

</style>
</head>
<body>
<ul>Мой список
  <li>Один</li>
  <li>Два</li>
  <li class="nodclosed" onclick="displayitem(this,'p3')">Три
   <ul class="closed" id="p3" onclick="event.cancelBubble=true">
    <li>1</li>
    <li>2</li>
    <li>3</li>
   </ul>
  </li>
  <li class="nodclosed" onclick="displayitem(this,'p4')">Четыре
   <ul class="closed" id="p4" onclick="event.cancelBubble=true">
    <li class="nodclosed" onclick="displayitem(this,'p41')">4.1
     <ul class="closed" id="p41" onclick="event.cancelBubble=true">
      <li>4.1.1</li>
      <li>4.1.2</li>
      <li>4.1.3</li>
     </ul>
    </li>
    <li>4.2</li>
   </ul>
  </li>
  <li>Пять</li>
  <li>Шесть</li>
</ul>
<script type="text/javascript">
function displayitem(thisobj,xid){ // показ/сокрытие подсписков
```

```
 thisobj.className=='nodopen'?
 thisobj.className='nodclosed':thisobj.className='nodopen';
 document.getElementById(xid).className =
   document.getElementById(xid).className=="closed"? "open": "closed";
}
</script>
</html>
```

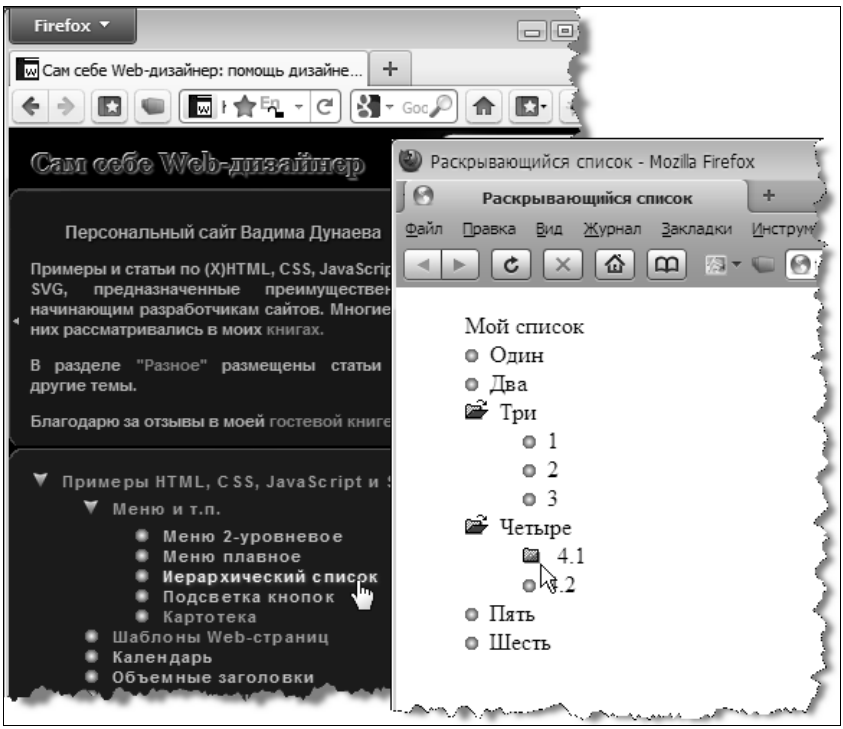


Рис. 16.4. Иерархический раскрывающийся список (листинг 16.6)

16.5. Эффект пишущей машинки

Постепенный (побуквенный) вывод на страницу текста, как будто его набирают на клавиатуре (имитация печати на пишущей машинке), можно создать с помощью метода `setInterval()`. Точнее, строку мы превращаем в массив одиночных символов, которые затем и выводим друг за другом с некоторой временной задержкой. Сценарий листинга 16.7 выводит в текстовую область (`<textarea>`) символы заданной строки с задержкой 0,1 с.

Листинг 16.7. Посимвольный вывод текста

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Пишущая машинка</title></head>
<body>
<textarea id="mytext" rows=8 cols=25></textarea>
</body>
<script type="text/javascript">
    var mystr="Привет, мои друзья!\nРад вам сообщить приятное известие."
    var astr=mystr.split("") /* разбиваем строку на массив символов */
    var typestr="";
    var xinterval=setInterval("wrtext()", 100); /* периодический вызов
                                                   функции wrtext() */
    var i=0;
    function wrtext(){    // вызывается с помощью метода setInterval()
     if (i < astr.length){
        typestr+= astr[i];                    // выводимая строка
        document.getElementById("mytext").value=typestr;  // вывод строки
        i++;
      }else clearInterval(xinterval)        // прекращаем вывод текста
    }
</script>
</html>
```

16.6. Отображение кода на странице

Иногда на Web-странице требуется вывести некоторый (X)HTML-код. Например, на своем сайте я привожу коды примеров, опубликованных в моих книгах. Чтобы браузер не интерпретировал, а лишь отображал код в виде обычного текста, достаточно заменить в нем символы `&`, `<` и `>` на `&`, `<` и `>` соответственно. Однако этот же код применяется в документе как рабочий, чтобы продемонстрировать его в действии. Разумеется, мне не хочется дублировать его с указанной заменой символов. В листинге 16.8 приводится код функции `showcode(xid)`, которая выполняет это автоматически для (X)HTML-кода элемента, значение атрибута `id` которого указано в параметре. Точнее, эта функция возвращает строку кода, в котором произведены указанные замены символов. Эта строка предназначена только для вывода, но не для выполнения. Кроме того, в начале данной строки кода добавляется тег `<pre>`, а в конце — `</pre>`, чтобы вывод в окно браузера осуществлялся в исходном текстовом формате. Для отображения этого кода в окне браузера можно воспользоваться методом `document.write()`.

Атрибутом `id` можно снабдить все теги документа, в том числе `<html>`, `<body>`, `<style>` и `<script>`. Таким образом, функция `showcode()` позволит вывести как весь код документа, так и лишь некоторые его фрагменты.

Обратите внимание, что в коде функции `showcode()` сначала производится замена символа `&` на `&`, а только затем замена остальных двух символов.

Листинг 16.8. Функция отображения кода (вариант 1)

```
function showcode(xid){ // код для отображения на странице
 if (!xid) return "";
 var x=document.getElementById(xid).innerHTML; // текущее значение кода
```

```
 x=replacestr(x, "&", "&amp;");
 x=replacestr(x,"<", "&lt;");
 return "<pre>"+replacestr(x,">","&gt;")+"</pre>";

 function replacestr(s1, s2, s3) { // замена подстрок
  if (s2 == "") s2=" "; /* заменяем пустую строку на
          строку с пробелом */
  var s="";    // обработанная часть строки
  while (true) {
   i=s1.indexOf(s2);   // индекс вхождения s2 в s1
   if (i >= 0) {
     s+= s1.substr(0, i) + s3;     /* обработанная часть
               строки */
     s1 = s1.substr(i + s2.length); /* оставшаяся часть
               строки */
   }else break;      // выход из цикла
  }
  return s + s1
 }
}
```

Код функции `showcode()` можно сократить и сделать более эффективным, если воспользоваться объектом `RegExp` регулярных выражений (листинг 16.9). В данной книге мы не рассматриваем его подробно, а лишь приводим пример использования.

Сначала создается экземпляр данного объекта с указанием в качестве параметров шаблона строки, которую следует заменить, и так называемого флага глобальности `"g"`, например, `var y = new RegExp("<", "g")`. Если флаг `"g"` опустить, то будет выполнена замена только первого вхождения строки, указанной в качестве шаблона. Затем применяется метод `replace ()` строкового объекта для замены одной подстроки на другую. Например, выражение `x.replace(y, "<")` заменяет в строке `x` вхождения подстрок, соответствующих регулярному выражению `y`, на строку `"<"`.

Листинг 16.9. Функция отображения кода (вариант 2)

```
function showcode(xid){ // код для отображения на странице
  if (!xid) return "";
  var x=document.getElementById(xid);
  if (!x) return "";
  x=x.innerHTML;
  var y=new RegExp("&","g");
  x=x.replace(y,"&amp;");
  y=new RegExp("<","g");
  x=x.replace(y,"&lt;");
  y=new RegExp(">","g");
  x=x.replace(y,"&gt;");
  return "<pre wrap='wrap'>"+x+"</pre>"
}
```

Листинг 16.10 и рис. 16.5, 16.6 иллюстрируют тестовый пример, в котором выводится содержимое тега `<html>`. Хотя различные браузеры отображают код по-разному, суть все же сохраняется.

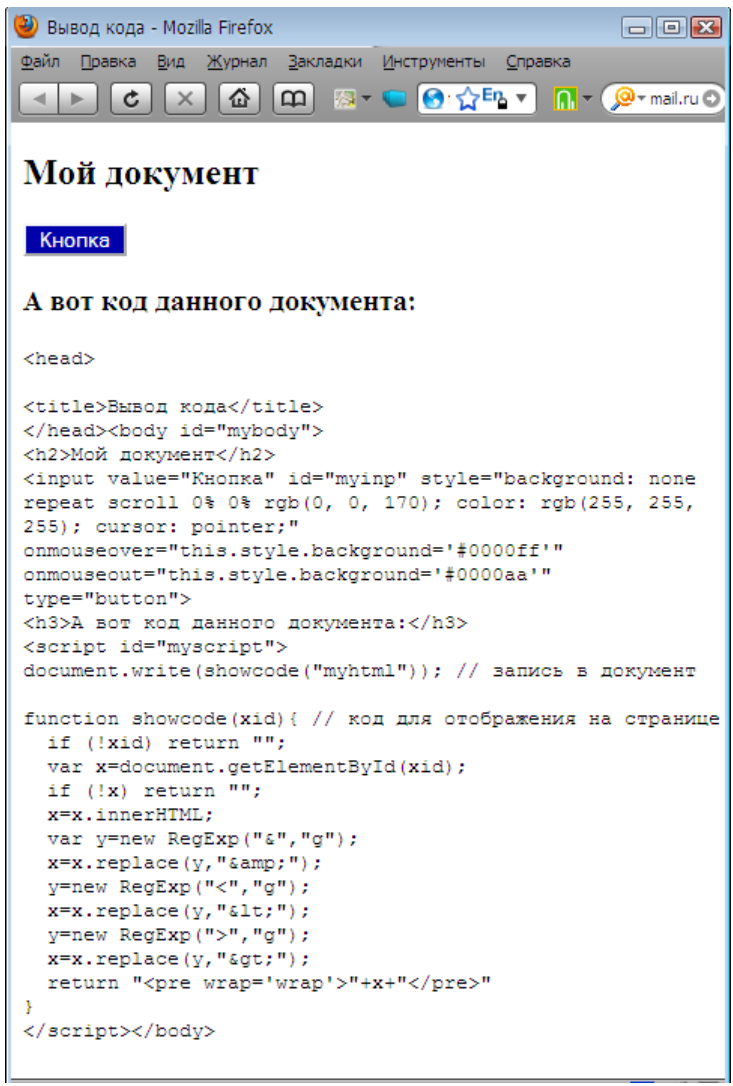


Рис. 16.5. Пример работы функции `showcode()` в Firefox

Листинг 16.10. Пример использования функции `showcode()`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html id="myhtml">
<head><title>Вывод кода</title></head>
<body id="mybody">
```

```
<h2>Мой документ</h2>
<input type="button" value="Кнопка" id="myinp"
 style="background:#0000aa; color:#ffffff; cursor:pointer"
 onmouseover="this.style.background='#0000ff'"
 onmouseout ="this.style.background='#0000aa'"/>
<h3>А вот код данного документа:</h3>
</body>
<script id="myscript">
document.write(showcode("myhtml")); // запись в документ

function showcode(xid){ // код для отображения на странице
  // см. листинг 16.9
}
</script>
</html>
```

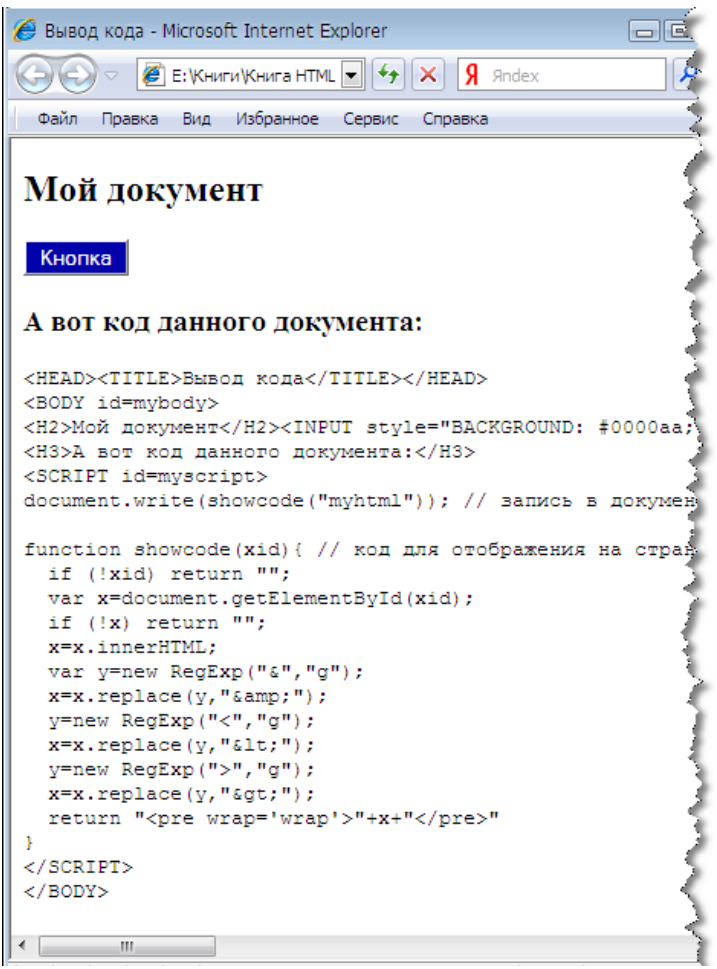


Рис. 16.6. Пример работы функции `showcode()` в Internet Explorer

16.7. Перемещение элементов мышью

Хотя в Web-приложениях задача перемещения видимых элементов документа с помощью мыши не очень актуальна, ее изучение дает нечто важное для понимания обработки событий в частности и Web-программирования вообще.

Очевидно, для перемещения элементов потребуется создать обработчики таких событий, как `onmousedown` (нажатие на кнопку мыши), `onmousemove` (перемещение указателя мыши) и `onmouseup` (освобождение ранее нажатой кнопки мыши). В самом деле, при перетаскивании элемента пользователь наводит на него указатель мыши, нажимает на левую кнопку и, удерживая ее, перемещает указатель, а в точке назначения отпускает кнопку. Однако не все элементы нужно и можно перемещать. Так, элементы `<input>` и `<textarea>` (поле и область ввода данных) реагируют на события мыши по-своему, например, щелчок устанавливает курсор, и мы не можем воспользоваться событиями мыши для других целей, в частности для управления перемещением. Хотя задача перемещения и этих элементов вполне разрешима, мы здесь не будем ее рассматривать. С другой стороны, не все элементы на странице нужно двигать с места на место, значит, разработчик должен как-то указать, что позволено клиенту перетаскивать, а что нет. В противном случае хаос может разрушить проект.

В предлагаемом здесь решении задачи элементы документа, для которых возможно перемещение, должны иметь атрибут `class="dragelement"`. Атрибут `class` задумывался для связи элемента с каскадными таблицами стилей. В данной задаче мы не используем его по своему прямому назначению. Сейчас он играет роль метки для тех элементов, которые мы разрешаем перемещать мышью. Разумеется, значение этого атрибута выбрано произвольно и может быть изменено разработчиком по своему усмотрению.

Далее, перемещение инициируется нажатием на кнопку мыши при наведении ее указателя на элемент, т. е. событием `onmousedown`. Обработчик этого события — функция `dragstart()`, привязанная не к конкретному перемещаемому элементу, а к контейнерному элементу `<body>`. Следовательно, событие `onmousedown` может восприниматься любым элементом документа. Однако обработчик `dragstart()` анализирует, в каком именно элементе произошло это событие (в частности, имеет ли он атрибут `class="dragelement"`) и, если все необходимые условия выполнены, начинает работу обработчик перемещения мыши — функция `drag()`, которая присваивает координатам элемента значения координат указателя мыши до тех пор, пока не будет освобождена кнопка мыши (событие `onmouseup`).

В листинге 16.11 в качестве примера приведен XHTML-код и сценарий, обеспечивающий перемещение элементов, которые имеют атрибут `class="dragelement"`. В рассматриваемом примере перемещаются два графических изображения и текст, заключенный в контейнер `<span>`. Внешний вид данного документа до и после нескольких перемещений элементов показан на рис. 16.7.

Листинг 16.11. Перемещение элементов мышью

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Перемещение мышью</title></head>
```

```
<body>
<img src="picture1.jpg" class="dragelement"
     style="position:absolute;top:150px;left:10px"/>
<img src="picture2.jpg" class="dragelement"
     style="position:absolute;top:100px;left:150px"/>
<span id="txt" class="dragelement">Перетащи меня</span>
<br/>
<input type="text" value="Привет"/>
</body>
<script type="text/javascript">
window.onload=function(){ // регистрируем обработчики событий
      document.body.onmousedown=dragstart;
      document.body.onmousemove=drag;
      document.body.onmouseup=dragend;
      id_el=null;
}
/*********************************************************************/
function dragstart(evt){    // начало перемещения элемента
  /* Получаем объект элемента, который нужно перетащить */
  var e;
  if (window.event) e=event.srcElement    // для IE
  else e=evt.target;                      // для других браузеров
  id_el=e.id;    // id перетаскиваемого элемента
  if (!id_el) {    // если нет id, то создаем его
    id_el="xid_"+parseInt((1+99999*Math.random()));
    e.id=id_el
  }
  /* Запрещенные для перемещения элементы: */
  if (
   e.tagName.toLowerCase()=="input"||e.tagName.toLowerCase()=="textarea")
  {
     e.focus();
     return false
  }

  if (document.getElementById(id_el)&&
    (document.getElementById(id_el).className=="dragelement"))
  {
    /* если нет позиционирования, то добавляем его: */
    if(!document.getElementById(id_el).style.position)
       document.getElementById(id_el).style.position="absolute";
    /* вид указателя мыши: */
    document.getElementById(id_el).style.cursor = 'move';
    /* увеличиваем z-index: */
    document.getElementById(id_el).style.zIndex++;
  }
  return false
}
```

```
/*********************************************************************/
function drag(event){    // перемещение элемента
  if (!id_el) return false;
  if (document.getElementById(id_el)&&
    (document.getElementById(id_el).className=="dragelement"))
  {
    var evt;
    /* для межбраузерной совместимости: */
    if (event) evt=event
    else evt=window.event;
   /* изменяем координаты элемента */
    var dx=20, dy=20;
    document.getElementById(id_el).style.top=(evt.clientY-dy)+"px";
    document.getElementById(id_el).style.left=(evt.clientX-dx)+"px";
  }
  return false
}
/*********************************************************************/
function dragend(){       // конец перемещения элемента
  if (document.getElementById(id_el)&&
     (document.getElementById(id_el).className=="dragelement"))
  {
    document.getElementById(id_el).style.cursor ="auto";
    id_el=null;
  }
  return false
}
</script>
</html>
```

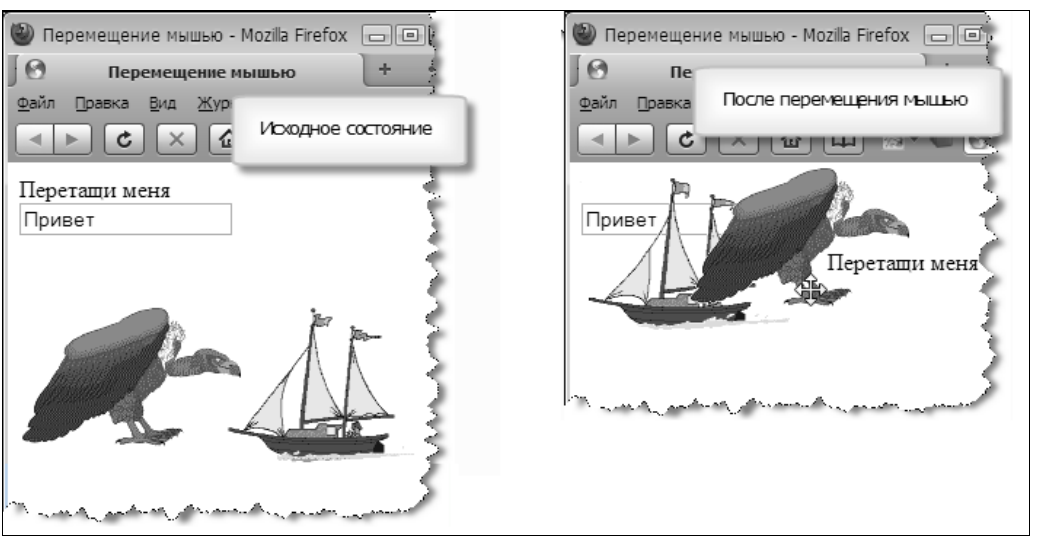


Рис. 16.7. Страница, сформированная кодом из листинга 16.11, до и после перемещения элементов

В функции dragstart(evt), которой передается экземпляр объекта события, сначала определяется, в каком именно элементе документа произошло событие. В дальнейшем обращение к объекту данного элемента происходит по значению атрибута id. Однако это значение может быть не установлено в (X)HTML-коде, т. е. в теге элемента отсутствует атрибут id. В таком случае свойство id устанавливается программно с помощью датчика случайных чисел (random()). Далее проверяется доступность объекта элемента и то, что у него есть атрибут class="dragelement" (соответствующее свойство объекта — className).

Перемещать можно элементы, среди параметров стиля (style) которых есть position. Иначе говоря, если этот параметр не определен, то вы не можете обращаться к параметрам top и left, которые задают соответственно вертикальную и горизонтальную координаты позиционирования элемента. Если у элемента не оказалось предварительно заданного стилевого параметра position, то он создается программно. Далее устанавливается вид указателя мыши "move" (перекрестие), соответствующий операции перетаскивания, и увеличивается текущее значение стилевого параметра z-index (свойство zIndex объекта style), чтобы перемещаемый элемент продвинулся ближе к переднему плану и мог расположиться поверх остальных.

Функция drag(event) обрабатывает событие onmousemove путем задания координатам элемента (стилевым параметрам left и top) значений координат указателя мыши относительно клиентской области браузера за вычетом dx и dy (в примере — 20 пикселов, эмпирически подобранное значение), чтобы указатель во время перемещения находился над элементом. Если перемещаемые элементы имеют установленные свойства width и height (ширина и высота), то для dx и dy можно назначить половины значений этих свойств соответственно.

Функция dragend() обрабатывает событие onmouseup, означающее завершение операции перетаскивания. Она приводит указатель мыши к обычному виду.

> ***Примечание***
>
> Для организации перемещения можно использовать обработчики событий ondragstart, ondrag, ondragend и т. п. Хотя эти события не предусмотрены в HTML 4, их поддерживают все ведущие браузеры. Вместе с тем, Firefox и Opera, в отличие от других браузеров, не сохраняют в объекте Event для данных событий координаты мыши, что затрудняет создание простого и универсального кода. События, связанные с перемещением, войдут в спецификацию HTML 5 и в будущем, возможно, будут поддерживаться одинаково всеми ведущими браузерами.

16.8. Движение по траектории

Задать движение элемента (например, графического изображения) по прямой линии — довольно простая задача. Для этого достаточно изменять значения текущих координат элемента на одну и ту же величину. Если это поручить некоторой функции, то, вызывая ее через определенный временной интервал с помощью метода setInterval(), можно получить движение элемента по линейной траектории. Более интересная задача состоит в организации движения по произвольной траектории, например по окружности или эллипсу.

16.8.1. Движение по произвольной кривой

Кривую линию на плоскости, как известно, можно задать с помощью двух выражений с одним и тем же независимым аргументом. Одно из этих выражений определяет закон изменения вертикальной координаты, а другое — горизонтальной. С помощью стандартной функции `eval()` можно вычислить значения этих выражений при конкретном значении аргумента и присвоить их стилевым параметрам `top` и `left` перемещаемого элемента. Функция (пусть это будет `move()`), которая все это выполняет, передается в качестве первого параметра методу `setInterval()`, периодически вызывающему ее через заданный интервал времени.

Функция инициализации движения `curvemove()` будет принимать четыре строковых и один числовой параметр. Строковые параметры содержат соответственно значение идентификатора `id` перемещаемого элемента, выражения для вертикальной и горизонтальной координат и имя независимого аргумента. Числовой параметр `ztime` определяет период времени, через который координаты элемента пересчитываются. По умолчанию независимый аргумент обозначается символом `"x"`, но можно выбрать и другой символ, например `"t"`. В листинге 16.12 приводятся определения функций `curvemove()` и `move()`.

Листинг 16.12. Функция движения по произвольной кривой

```
function curvemove(xid, yexpr, xexpr, ztime, namevar) {
  /*Движение по произвольной кривой.
    Параметры:
    1. xid — id движущегося объекта, строка
    2. yexpr — выражение для вертикальной координаты
    3. xexpr — выражение для горизонтальной координаты
    4. ztime — интервал времени между вызовами функции move(), мс
    5. namevar — имя аргумента в выражениях yexpr и xexpr
    Возвращает идентификатор временного интервала
  */
  if (!xid) return null;
  if (!yexpr) yexpr="x";
  if (!xexpr) xexpr="x";
  if (!ztime) ztime=100;          // интервал времени, мс
  if (!namevar) namevar="x";
     eval(namevar + "= 0")          // начальное значение аргумента
/* Если нет позиционирования, то добавляем его: */
  if(!document.getElementById(xid).style.position)
     document.getElementById(xid).style.position="absolute";
   return setInterval(
         "move('"+xid+"', '" + yexpr + "', '"+xexpr +"','"+namevar+ "')",
         ztime)
}

function move(xid, yexpr, xexpr, namevar) { // вызывается из curvemove()
  eval(namevar + "++");       // приращение аргумента на 1
  document.getElementById(xid).style.top = eval(yexpr)+"px";
  document.getElementById(xid).style.left = eval(xexpr)+"px";
}
```

Обратите особое внимание на создание переменной. Сначала имя этой переменной сохраняется в виде строкового значения в другой переменной, затем она используется при генерации строки, содержащей некоторое выражение. Далее строка с выражением передается в качестве параметра функции `eval()`, которая выполняет данное выражение. Чтобы изменения стилевых параметров `top` и `left` возымели действие, требуется наличие параметра `position`, который добавляется в сценарии при необходимости. Функция `curvemove()` возвращает идентификатор временного интервала, который можно использовать в качестве параметра метода `clearInterval()`, чтобы при необходимости остановить движение.

Пример использования функции `curvemove()` приведен в листинге 16.13.

Листинг 16.13. Пример использования функции движения по произвольной кривой

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Движение</title><head>
<body>
<h1 id="myobj1" style="position:relative">Заголовок</h1>
<br/>
<img src="picture2.jpg" id="myobj2" />
</body>
<script type="text/javascript">
function curvemove(xid, yexpr, xexpr, ztime, namevar) {
... // см. листинг 16.12
}
function move(xid, yexpr, xexpr, namevar) {
... // см. листинг 16.12
}
/* Примеры вызовов функций: */
curvemove("my1","100 + 100*Math.sin(0.05*x)/(0.05*x)","50+x", 50, "x");
curvemove("my2","100 + 2*time", "570 — time", 100,"time");
</script>
</html>
```

Здесь заголовок движется по волнистой линии с затухающей амплитудой, а графическое изображение — по прямой. Чтобы задать горизонтальное движение, можно воспользоваться, например, следующим вызовом функции:

```
curvemove("myobj1","100", "x");
```

В данном случае элемент с `id="myobj1"` будет перемещаться горизонтально со скоростью 10 px/c.

А вот примеры движения по эллипсу и дельтоиде соответственно:

- ```
  curvemove("myobj1",
            "100+80*Math.sin(6/25*x)",
            "100+60*Math.cos(6/25*x+1)");
  ```
  ```

```
❒ curvemove("myobj1",
          "120*Math.sin(x)-60*Math.sin(2*x)",
          "120*Math.cos(x)+60*Math.cos(2*x)");
```

16.8.2. Движение по эллипсу

Хотя эллиптическая траектория может быть создана функцией `curvemove()`, все же удобнее задавать ее геометрическими параметрами эллипса, такими как большая и малая полуоси, координаты центра и др. (рис. 16.8). В листинге 16.14 приведен код функции `ellipse()` с параметрами, задающими движение элемента страницы по эллипсу.

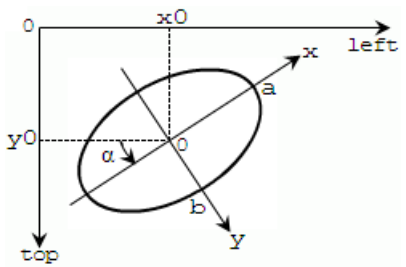


Рис. 16.8. Эллиптическая траектория

Листинг 16.14. Функция движения по эллипсу

```
function ellipse(xid,alpha,a,b,omega,x0,y0,ztime,dt){
/* Движение по эллипсу
 Параметры:
 1. xid — id движущегося объекта, строка
 2. alpha — угол поворота эллипса, град.
 3. a — большая полуось эллипса, пикселы
 4. b — малая полуось эллипса, пикселы
 5. omega — угловая скорость, град/с; знак задает направление вращения
 6. x0 — горизонтальная координата центра эллипса, пикселы,
 7. y0 — вертикальная координата центра эллипса, пикселы
 8. ztime — временная задержка, мс
 9. dt — начальная фаза, град.
 Возвращает идентификатор временного интервала
*/
if (!document.getElementById(xid)) return;
// проверка наличия параметров:
if (alpha==null) alpha=0;
if (a==null) a=0;
if (b==null) b=0;
if (omega==null) omega=0;
if (x0==null) x0=0;
if (y0==null) y0=0;
if (ztime==null)ztime=100;
if (dt==null) dt=0;
```

```
t=-ztime; /* чтобы начальное значение было 0,
             поскольку в move() уже есть приращение.
Многократный вызов функции move() с интервалом ztime, мс:
 */
/* Если нет позиционирования, то добавляем его: */
  if(!document.getElementById(xid).style.position)
     document.getElementById(xid).style.position="absolute";

return setInterval("move('"+xid+"',"+alpha+","+a+
","+b+","+omega+","+x0+","+y0+","+ztime+
","+dt+")",ztime)
}

function move(xid,alpha,a,b,omega,x0,y0,ztime,dt){
// пересчет координат, вызывается из ellipse()
t=t+ztime/1000; // приращение независимого аргумента
// x,y — координаты в собственной системе координат
var x=a*Math.cos((omega*t+dt)*Math.PI/180);
var y=b*Math.sin((omega*t+dt)*Math.PI/180);
var as=Math.sin(alpha*Math.PI/180);
var ac=Math.cos(alpha*Math.PI/180);
document.getElementById(xid).style.top=
                    (-x*as+y*ac+y0)+"px";
document.getElementById(xid).style.left=
                    (x*ac+y*as+x0)+"px";
}
```

Вот примеры вызова функции `ellipse()`:

```
ellipse("myobj1",60,100,30,10,170,300,200,0);
ellipse("myobj1",30,80,60,5,170,230,200,15);
ellipse("myobj1",60,100,50,25,200,210);
```

Скорость и плавность движения по эллиптической траектории определяется параметрами `ztime` (временной интервал между вызовами функции `move()`) и `omega` (угловая скорость).

Чтобы элемент перемещался по окружности, достаточно задать одинаковые значения большой и малой полуосей эллипса. А для движения по прямой линии циклически, туда и обратно, достаточно задать одну из полуосей эллипса равной нулю.

16.9. Рисование линий посредством * *

Для создания горизонтальной прямой линии в документе служит тег `<hr/>` с возможными атрибутами:

- `size` — толщина в пикселах;
- `width` — ширина в пикселах;
- `align` — способ выравнивания (`center`, `left`, `right`);
- `noshade="noshade"` — если указан, то создается сплошная черная полоса без тени.

Прямую горизонтальную и вертикальную линии можно создать и посредством тега `<div>` с указанием параметров `width`, `height`, `top`, `left`, `bacground-color`. Для наклонных прямых и произвольных кривых линий потребуется сценарий, который генерирует множество тегов `<div>`, надлежащим образом позиционированных друг относительно друга. Каждый из таких тегов соответствует точке кривой. Далее мы рассмотрим, как это сделать.

16.9.1. Прямая линия

Очевидно, горизонтальную или вертикальную линию можно создать одним тегом `<div>`, а для наклонной прямой их потребуется несколько. В листинге 16.15 приведен код функции `line()`, которая рисует прямые линии различной толщины и цвета. Данная функция возвращает строку HTML-кода для формирования линии и записывает ее в текущий документ, чтобы отобразить линию.

Листинг 16.15. Функция рисования прямой линии

```
function line(x1, y1, x2 ,y2, n, color){
/* Рисование прямых линий
   Параметры:
     x1, y1 – координаты начала линии
     x2, y2 – координаты конца линии
     n — толщина линии
     color — цвет линии
   Возвращает строку с HTML-кодом линии
 */
  var xstr="";            // строка тегов для записи в HTML-документ
  if (!n) n=1;
  if (!color ) color="#000000";
  if (x1==x2){ //если линия вертикальная
    xstr='<div style="position:absolute; height:'+
    Math.abs(y2 — y1)+'px; width:'+n+'px; top:'+
    Math.min(y1,y2)+'px; left:'+x1+'px; background:'+color+'"></div>';
    document.write(xstr); // запись в документ
    return xstr          // возвращение строки тегов
  }

  if (y1==y2){ //если линия горизонтальная
    xstr='<div style="position:absolute; width:'+
    Math.abs(x2 — x1)+'px; height:'+n+'px;
    top:'+y1+'px;left:'+
    Math.min(x1,x2)+'px;background:'+color+'"></div>'
    document.write(xstr);  // запись в документ
    return xstr           // возвращение строки тегов
  }
 /* Если прямая наклонная: */
  var xstr0='<div style="position:absolute; width:'+n+
 'px; height:'+n+'px; background:'+color+';';
  var x, k, direct;
```

```
/* Как развернуть линию, по горизонтали или вертикали? */
  var vertical = Math.abs(y2 — y1) > Math.abs(x2 — x1);
  if (vertical){
    direct=(y2 > y1);                  // направление прямой
    x=y1;
    k=(x2 — x1)/(y2 — y1)              // коэффициент наклона прямой
  }else{
    direct=(x2 > x1);                  // направление прямой
    x=x1;
    k=(y2 — y1)/(x2 — x1)              // коэффициент наклона прямой
  }
  while (true){
    if (!vertical){
      xstr += xstr0 + 'top:' + (y1 + k*(x — x1)) + 'px; left:' +
      x+'px"></div>'
      if (x == x2) break;
      if (direct) x++
      else x--
    }else{
      xstr += xstr0 + 'left:' + (x1+k*(x-y1)) + 'px; top:' +
      x + 'px"></div>'
      if (x == y2) break;
      if (direct) x++
      else x--
    }
  }
 document.write(xstr)   // запись в документ
 return xstr // возвращение строки тегов
}
```

Здесь определяется, к горизонтали или вертикали ближе наклонная линия, чтобы задать, вдоль какой из них производить развертку. Это позволяет устранить прерывистость отображения наклонной линии.

В листинге 16.16 и на рис. 16.9 приведен пример использования функции `line()`. Сценарий содержит определение функции `line()` и несколько его вызовов. Кроме того, эта же функция вызывается в цикле с изменением ее параметров. Цвет задает вспомогательная функция `to16()`, которая преобразует десятичное целое число от 0 до 255 в строку, содержащую соответствующее шестнадцатеричное двузначное число. В результате работы цикла получается треугольник, заполненный градиентом.

Листинг 16.16. Пример использования функции рисования прямых линий

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Прямые линии</title></head>
<script type="text/javascript">
function line(x1, y1, x2 ,y2, n, color){
 ... // см. листинг 16.15
}
```

```
/* Вызовы функции line() */
line(10,10,200,10);
line(10,10,200,100,2);
line(10,10,200,200);
line(10,10,150,200);
line(120,40,10,200,3);
line(70,20,250,20,8);

/* Построение треугольника, залитого градиентом: */
var h, xcolor;
for (i=1;i<90;i++){
     h=to16(40+2*i);
        xcolor="#"+h+h+h;
        line(170,120,350-i*4,250+i*2,6,xcolor)
}
function to16(n10){ // преобразование 10-ричного числа в 16-ричное
  var hchars="0123456789abcdef";
  if (n10>=255) return "ff";
  if (n10<=0) return "00";
  var i=n10%16;
  var j=(n10-i)/16;
  rezult=hchars.charAt(j);
  rezult+=hchars.charAt(i);
  return rezult;
}
</script>
</html>
```

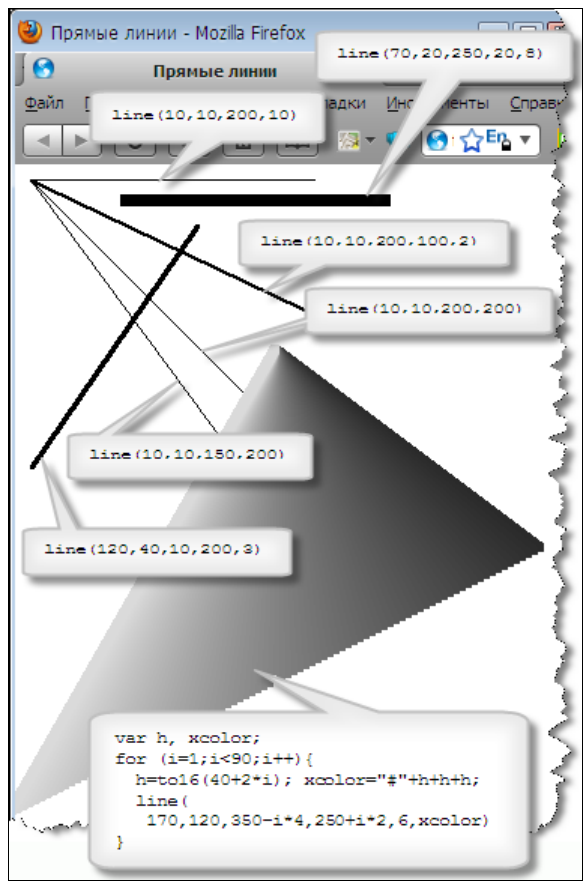


Рис. 16.9. Страница, сформированная кодом листинга 16.16

Примечание

Быть может, лучше сделать так, чтобы сама функция `line()` ничего не выводила на экран, а только возвращала строку, содержащую необходимый для отображения линии HTML-код. Тогда отображение линии с помощью метода `document.write()` можно выполнить где-нибудь в другом месте сценария.

16.9.2. Произвольная линия

Идеи рисования произвольной (кривой) линии и организации движения элемента (см. *разд. 16.8.1*) схожи.

В листинге 16.17 приведен код функции `curve()`, которой в качестве параметров передаются два выражения с независимым аргументом `x`. Первое из них определяет закон изменения вертикальной координаты, а второе — горизонтальной (в простейшем случае это может быть `"x"`). Далее, функции передаются координаты начала кривой, количество точек кривой, толщина, длина штриха и паузы (для рисования прерывистых линий), а также цвет.

Листинг 16.17. Функция рисования произвольной линии

```
function curve( yexpr, xexpr, x0, y0, t, n, s, color){
  /* Параметры:
     yexpr — выражение с переменной x для вертикальной координаты
     xexpr — выражение с переменной x для горизонтальной координаты
     x0, y0 — координаты начала кривой в пикселах
     t — количество точек кривой (значений переменной x)
     n — толщина линии в пикселах
     s — длина штриха и паузы в пикселах (0 - сплошная линия)
     color - цвет или "url('URL_картинки')"
     Возвращает строку с HTML-кодом линии
  */
  if (!yexpr) return null;
  /* Значения параметров по умолчанию */
  if (!xexpr) xexpr="x";
  if (!x0) x0=0;
  if (!y0) y0=0;
  if (!s) s=0;
  if (!t) t=0;
  if (!n) n=1;
  if (!color ) color="#000000";

  var x; // независимый аргумент
  var xstr0='<div style="position:absolute; width:'+n+
      'px;height:'+n+'px;background:'+color+'; top:';
  var xstr="";
  var i=0, draw=true;
  for(x=0; x <  t; x++){
    if (draw) // если рисовать
      xstr += xstr0 + (y0 + eval(yexpr)) + 'px; left:' +
              (x0 + eval(xexpr)) + 'px"></div>';
```

```
    if (i > s&&s > 0) { // если пауза
      draw=!draw;
      i=0
    }
    i++
  }
  document.write(xstr);          // запись в документ
  return xstr
}
```

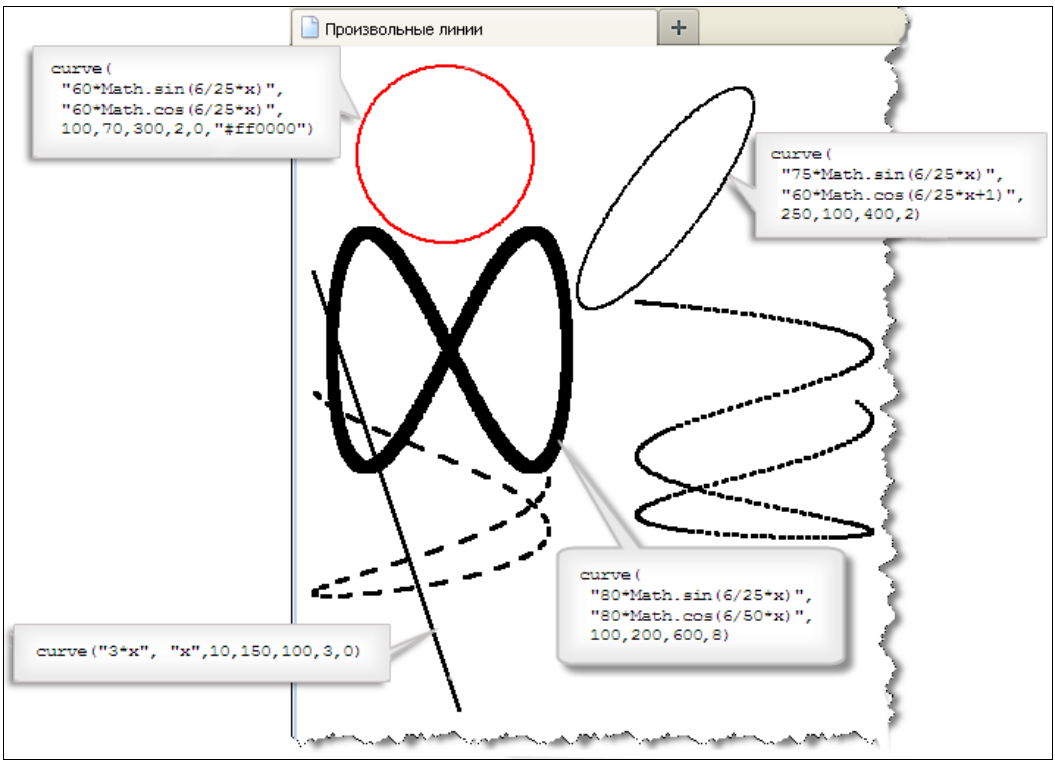


Рис. 16.10. Страница, сформированная кодом листинга 16.18

В листинге 16.18 и на рис. 16.10 приведен пример использования функции `curve()`.

Листинг 16.18. Пример использования функции рисования произвольных линий

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Произвольные линии</title></head>
<script type="text/javascript">
function curve( yexpr, xexpr, x0, y0, t, n, s, color){
... // см. листинг 16.17
}
```

```
curve(   // окружность
   "60*Math.sin(6/25*x)", "60*Math.cos(6/25*x)",
    100,70,300,2,0,"#ff0000");
curve(   // эллипс
   "75*Math.sin(6/25*x)", "60*Math.cos(6/25*x+1)",
    250,100,400,2);
curve("80*Math.sin(6/25*x)", "80*Math.cos(6/50*x)",
    100,200,600,8);
curve("80*Math.sin(6/250*x)", "80*Math.cos(6/75*x)",
    310,250,400,3);
curve("80*Math.sin(2/350*x)", "80*Math.cos(5/375*x)",
    90,290,700,3,10);
curve( // прямая
    "3*x", "x",
    10,150,100,3,0)</script>
</html>
```

Точки, составляющие линию, можно не только закрашивать, но и заполнять графическим изображением. Эта возможно благодаря тому, что значение последнего параметра функции `curve()` назначается свойству `background` для элементов `<div>`, посредством которых рисуется кривая. А свойство `background` может принимать значение в виде `"url('`*URL-адрес изображения*`')"`. На рис. 16.11 приведен соответствующий пример, в котором штриховая прямая и три так называемые фигуры Лиссажу нарисованы с помощью графических изображений (листинг 16.19).

Листинг 16.19. Рисование линий с помощью графических изображений

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Произвольные линии</title></head>
<script type="text/javascript">
function curve( yexpr, xexpr, x0, y0, t, n, s, color){
... // см. листинг 16.17
}

curve(
   "80*Math.sin(6/250*x)", "80*Math.cos(6/75*x)",
    80,25,150,200,11,
    "url('облако.png')");
curve("80*Math.sin(2/350*x)", "80*Math.cos(5/375*x)",
    220,400,760,70,0,
    "url('облако.png')");
curve( // прямая
    "0.7*x", "x",
    10,10,150,40,10,"url('градиент.png')")
curve( // дельтоида
   "100*Math.sin(x)-50*Math.sin(2*x)",
```

```
    "100*Math.cos(x)+50*Math.cos(2*x)",
     100,250,300,55,0,
     "url('градиент.png')");
</script>
</html>
```

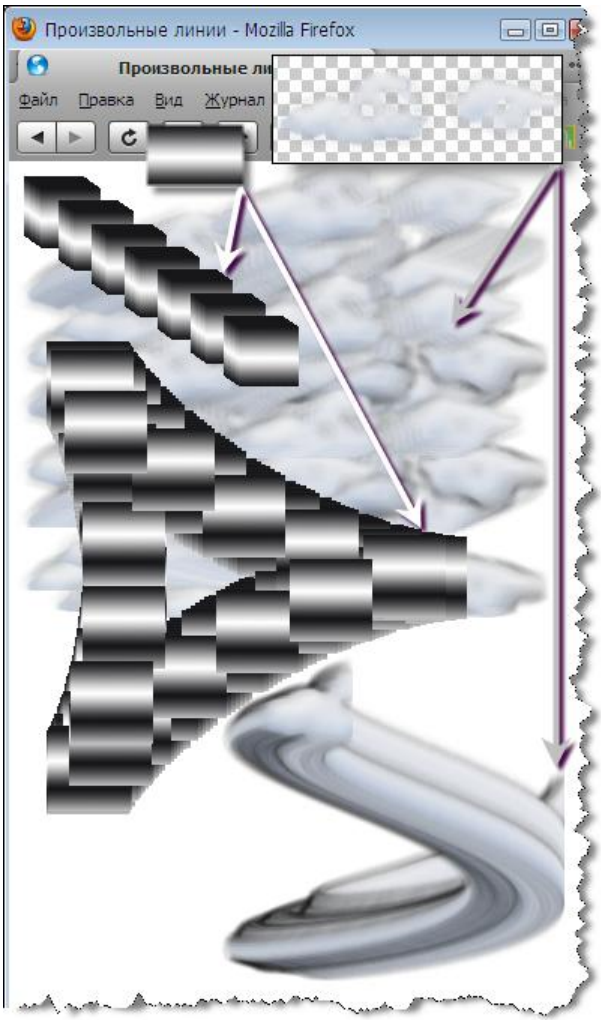


Рис. 16.11. Рисование произвольных линий с помощью графических изображений (листинг 16.19)

Примечания

В предлагаемом варианте длина кривой задается числом ее точек. Возможно, для практических целей лучше назначить какие-то другие параметры, определяющие ее продолжительность.

Можно сделать так, чтобы сама функция `curve()` ничего не выводила на экран, а только возвращала строку, содержащую необходимый для отображения линии HTML-код. Тогда отображение линии с помощью метода `document.write()` можно выполнить где-нибудь в другом месте сценария.

16.9.3. Графики зависимостей

Как известно, зависимости между двумя величинами можно задавать в виде таблиц, состоящих из двух столбцов, в одном из которых располагаются данные, соответствующие аргументу зависимости, а в другом — ее значению. Аргументы отображаются на графике вдоль горизонтальной координатной оси, а значения — вдоль вертикальной. Собственно зависимость обычно представляется либо точками на координатной плоскости, либо ломаной прямой, проходящей через эти точки, либо вертикальными прямыми (столбиками). Возможны, конечно, и другие способы графического представления данных (например, в виде круговой диаграммы). Здесь мы остановимся на задаче построения графиков в прямоугольной системе координат в виде ломаной прямой.

Таблицы с данными можно представить с помощью одного двумерного или двух одномерных массивов. Будем считать, что имеем дело с двумя одномерными массивами, в одном из которых содержатся аргументы, а во втором — значения зависимости. Например, в одном массиве хранятся даты, а в другом — соответствующие этим датам среднесуточные (или какие-нибудь другие) значения температуры. Зависимость между температурой и датой можно отобразить в виде ломаной линии, состоящей из отрезков прямых.

Легко догадаться, что для построения графика зависимости между элементами двух массивов в виде ломаной линии можно просто организовать в цикле вызов рассмотренной ранее функции `line()`. При этом следует заранее определить расстояние в пикселах между соседними точками по горизонтальной оси. Возникает задача масштабирования: перевод значений элементов массивов в пикселы экранных координат.

Прежде чем рассмотреть пример построения графика, немного модифицируем функцию `line()`, поскольку в том виде, как в листинге 16.15, она не совсем подходит для наших целей. Сделаем так, чтобы она просто возвращала строку тегов, описывающую линию, а не записывала ее в документ. Для этого необходимо в определении функции `line()` удалить все строки `document.write(xstr)`.

В листинге 16.20 приводится код сценария, отображающего в окне браузера график зависимости, заданной с помощью двух массивов (рис. 16.12).

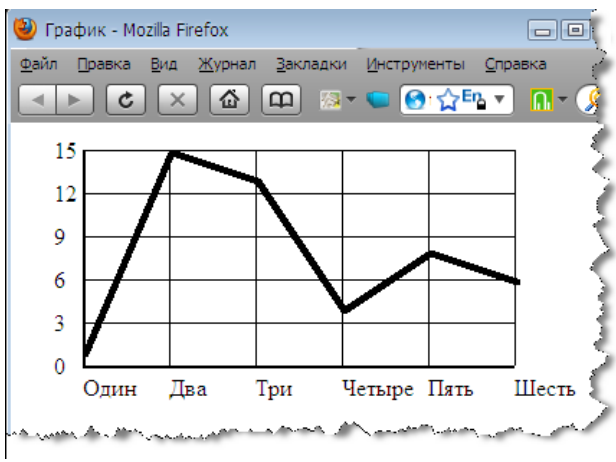


Рис. 16.12. График зависимости, сформированный кодом листинга 16.20

Листинг 16.20. График зависимости, заданной с помощью двух массивов

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>График</title></head>
<script type="text/javascript">
  /* График зависимости, заданной массивами */
  /* Массив элементов горизонтальной оси */
  var aX=new Array("Один", "Два", "Три", "Четыре", "Пять", "Шесть");
  /* Массив элементов вертикальной оси */
  var aY=new Array(1, 15, 13,  4,  8,  6);

  var ky=10, kx=60;          // коэффициенты масштабирования
  var x0=50, y0=200;         // отступы
  var xstr="";               // строка тегов, описывающая линии
  var i, x1,y1,x2,y2,x0,y0;
  for(i=0; i < aX.length -1; i++){      // линия графика
    x1=x0 + kx*i;
    y1=y0 — ky*aY[i];
    x2=x0 + kx*(i + 1);
    y2=y0 — ky*aY[i + 1];
    xstr+= line( x1, y1, x2, y2, 4);
  }

  for (i=0; i < aY.length; i++){
    /* Метки на вертикальной оси */
    xstr+= "<span style='position:absolute;top:"+
           (y0-i*30-10)+"px;left:" +(x0-20)+"px'>"+i*3+"</span>";
    if (i > 0){
      // горизонтальные линии:
      xstr+= line(x0, y0 — i*30, x0 + 5*kx, y0 — i*30, 1);
      // вертикальные линии:
      xstr+= line(x0 + kx*i, y0, x0 + kx*i, 50, 1);
    }
    /* Метки на горизонтальной оси */
    xstr+= "<span style='position:absolute;top:" + (y0 + 5) +
          "px; left:"+ (x0 + kx*i) +"px'>" +
    aX[i]  + "</span>";
  }
  xstr+= line(x0,y0,x0,50,2);           // вертикальная ось
  xstr+= line(x0,y0,x0 + 5*kx,y0,2);    // горизонтальная ось

  document.write(xstr);                 // запись в документ

function line(x1, y1, x2 ,y2, n,color){
... // см. листинг 16.15; модификация: удалить document.write(xstr)
}
</script>
</html>
```

16.9.4. Перерисовка линий

Иногда требуется время от времени перерисовывать линии. Например, если стрелка на изображении циферблата часов формируется с помощью сценария на основе функции `line()`, то для создания эффекта движения необходимо удалить ранее нарисованную стрелку и вывести новую, положение которой соответствует текущему времени. В другой ситуации может потребоваться перерисовать график некоторой зависимости при динамическом обновлении данных. В общем случае перерисовывание линии состоит из удаления ранее нарисованного ее изображения и создания новой линии.

Рассмотренные ранее коды функций `line()` и `curve()` не вполне приспособлены к решению новой задачи из-за того, что они выполняют запись в документ сгенерированной строки HTML-тегов. Вообще, это делать совсем не обязательно. Вполне достаточно, чтобы функции `line()` и `curve()` просто возвращали строку тегов, а записать ее в (X)HTML-документ можно и во внешней программе. В той же программе мы можем удалить и всю совокупность тегов, формирующих линию (когда это понадобится).

Итак, прежде всего, изменим определения функций `line()` и `curve()` (см. листинги 16.15 и 16.17) следующим образом: удалим в них все вхождения выражения `document.write(xstr)`. В результате функции не будут ничего рисовать, а станут лишь создавать и возвращать данные для последующего рисования. Теперь, чтобы нарисовать произвольную линию, следует в сценарии написать такую строку кода:

```
document.write(curve(параметры));
```

Поскольку мы должны быть готовы удалить линию из документа, то строку тегов, возвращаемую функциями `curve()` и `line()`, сначала заключим в какой-нибудь контейнер, например, `<div id=...>` с атрибутом `id`, а затем запишем в документ:

```
var cmycurve="<div id='mycurve'> " + curve(параметры) + "</div>"
document.write(cmycurve)          // запись в документ
```

Чтобы удалить линию из документа, достаточно заменить содержимое контейнерного тега `<div>` на пустую строку:

```
document.getElementById("mycurve").innerHTML="".
```

Здесь мы воспользовались свойством `innerHTML`, которое содержит весь HTML-код, заключенный в указанный контейнер. Присвоение этому свойству нового значения сразу же изменяет соответствующий элемент в документе. При этом сам контейнер `<div>` с пустым содержимым остается. В случае необходимости его можно наполнить новыми данными. Это типичный прием динамического изменения элементов (X)HTML-документов.

Чтобы вместо предыдущей линии нарисовать новую, достаточно записать в сценарии следующий код:

```
document.getElementById("mycurve").innerHTML=curve(другие_параметры);
```

В результате контейнер `<div id='mycurve'>` заполнится последовательностью тегов `<div...>`, формирующих новую линию, а браузер ее отобразит.

В *разд. 16.11.2* мы рассмотрим еще один пример реализации описанного приема.

16.10. Рисование посредством <*canvas*>

Элемент <canvas> (холст, подложка) предназначен для создания на Web-странице графических изображений посредством скриптов. С его помощью можно рисовать линии (в том числе и кривые Безье), фигуры, а также создавать анимацию. Многое из того, что позволяет SVG, доступно и посредством <canvas>.

Описание элемента <canvas> входит в спецификацию HTML 5.0 (WhatWG Web Application 1.0) от W3C (**http://www.w3.org/TR/html5**), которая пока имеет статус черновика. Тем не менее данный элемент поддерживается браузерами Firefox, Opera, Safari, Chrome и Internet Explorer 9. Internet Explorer 8.0, к сожалению, не имеет собственных средств поддержки элемента <canvas>, но для него есть общедоступная специальная JavaScript-библиотека (**http://code.google.com/p/explorercanvas**), позволяющая применять данный элемент, хотя и с некоторыми ограничениями и меньшим эффектом.

В Интернете можно найти множество примеров и учебных материалов по применению <canvas> (например, **https://developer.mozilla.org/en/Canvas_tutorial**). В данном разделе мы рассмотрим на примерах лишь некоторые основные его возможности.

16.10.1. Как вставить <*canvas*> в (X)HTML-документ

Тег <canvas> является внутристрочным и контейнерным, в котором можно разместить так называемое аварийное содержимое (например, текст или графическое изображение), отображаемое теми браузерами, которые данный элемент не поддерживают. Необязательные атрибуты `width` и `height` задают ширину и высоту в пикселах области страницы, в которой будет отображаться графическое содержимое элемента (подложки). Значениями по умолчанию являются соответственно 300 и 150. Для <canvas> можно задать стилевые параметры, например, `border`, `background`, `margin`, `position` и др. Вот пример задания элемента <canvas> в (X)HTML-документе:

```
<canvas width="400" height="200">Ваш браузер не поддерживает canvas
</canvas>
```

Если браузер поддерживает элемент <canvas>, то последний отображается как прозрачный прямоугольник без видимых границ. Иначе говоря, внешне сам по себе элемент <canvas> пуст и прозрачен, но занимает некоторое место на странице. Это только холст, на котором рисуют посредством скриптов.

Для обеспечения совместимости с Internet Explorer 8 следует обзавестись библиотечным файлом excanvas.js, который можно получить по адресу **http://code.google.com/p/explorercanvas**. Данный файл следует разместить на вашем сайте, а в документе, использующем <canvas>, вставить секцию <script>, обеспечивающую его загрузку:

```
<!--[if IE]>
    <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
```

Данную вставку лучше сделать в контейнере <head>.

В сценарии, который что-то рисует, прежде всего, необходимо получить доступ к так называемому контексту вывода (отрисовки) объекта элемента <canvas>. При этом желательно проверить, что таковой действительно существует. Вот типичная схема:

```
var canvas = document.getElementById('mycanvas'); /* 'mycanvas' —
                                          значение id тега <canvas> */
if (canvas.getContext){
  var ctx = canvas.getContext('2d');
  // здесь код для рисования
} else {
  // здесь код для случая, когда canvas не поддерживается
}
```

Методу `getContext('2d')`, обеспечивающему доступ к контексту вывода, передается параметр, указывающий тип контекста. Пока он имеет единственное значение `"2d"` — двумерная графика. В будущем, возможно, появится и значение `"3d"`, соответствующее трехмерной графике.

В листинге 16.21 приведен пример использования <canvas>, в котором сценарий рисует прямоугольник (рис. 16.13). Здесь для элемента <canvas> с помощью CSS указана граница. Прямоугольник внутри холста нарисован методом `strokeRect()`. Методы рисования мы рассмотрим в следующих подразделах.

Листинг 16.21. Пример использования <canvas>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример Canvas</title>
<style type="text/css">
    #mycanvas {border:solid 10px red} // граница canvas
</style>
<!--[if IE]>
   <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body>
<canvas id="mycanvas" width="150" height="150" >
 Ваш браузер не поддерживает canvas;
</canvas>
<script type="text/javascript">
var canvas = document.getElementById('mycanvas');
if (canvas.getContext){
  var ctx = canvas.getContext('2d');
  ctx.strokeRect(30,50,100,50); // рисуем контур прямоугольника
}
</script>
</body>
</html>
```

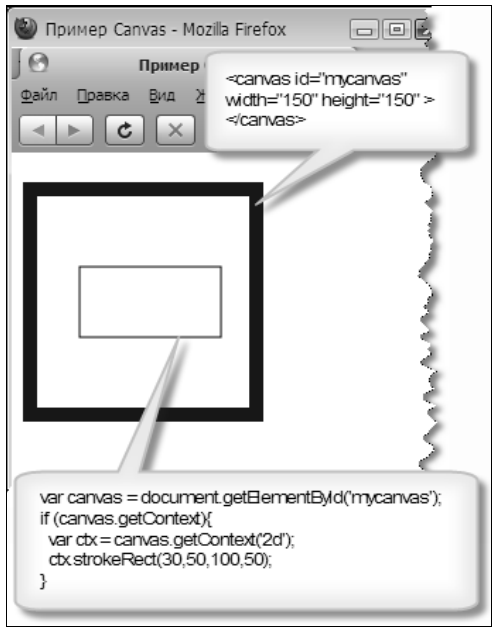


Рис. 16.13. Пример использования `<canvas>` (листинг 16.21)

Неплохо, особенно для совместимости с Internet Explorer 8, сценарий рисования запускать, предварительно убедившись, что загрузка документа завершилась. Для этого можно, например, оформить весь код рисования как обработчик события `window.onload`:

```
window.onload=function(){
              // здесь код рисования
            }
```

16.10.2. Фигуры и линии

Если вы знакомы с SVG, то понимание `<canvas>` не вызовет затруднений. Особенность `<canvas>` заключается в том, что рисование обеспечивается только посредством свойств и методов объекта контекста вывода:

```
document.getElementById('id_canvas').getContext('2d')
```

Данный объект поддерживает единственную примитивную фигуру — прямоугольник. Более сложные фигуры получаются с помощью линий — так называемых путей (path), дуг и кривых Безье. Фигуры могут иметь внешний контур (обводку) и/или внутреннюю область, называемую еще областью заливки.

Цвет контура и области заливки задаются свойствами, принимающими строковые значения в виде названия цвета (например, `"green"`), шестнадцатеричного значения (например, `" #00ff00"`), а также в виде `"rgb(r,g,b)"` и `"rgba(r,g,b,a)"`, где *r*, *g*, *b* — целые десятичные числа от 0 до 255, *a* — уровень непрозрачности от 0.0 до 1.

Координаты отсчитываются от верхнего левого угла холста и задаются, как и размеры, числами без указания размерности (предполагаются пикселы). Значения координат могут быть дробными.

Прямоугольник

Сначала рассмотрим рисование простого прямоугольника. Для этого предусмотрены три метода, принимающие в качестве параметров координаты верхнего левого угла, ширину и высоту:

- `strokeRect(x, y, width, height)` — рисует контур прямоугольника (внутренняя область прозрачна); по умолчанию цвет контура черный;
- `fillRect(x, y, width, height)` — рисует окрашенный прямоугольник (по умолчанию в черный цвет);
- `clearRect(x, y, width, height)` — очищает заданную параметрами прямоугольную область.

Для задания цвета контура и области заливки фигуры служат свойства соответственно `strokeStyle` и `fillStyle`. Толщину контура в пикселах определяет свойство `lineWidth`.

В листинге 16.22 приведен пример, в котором рисуются два прямоугольника, один из них представлен областью заливки, а другой только контуром (рис. 16.14). Для наглядности элементу `<canvas>` назначена граница в виде штриховой линии.

Листинг 16.22. Пример рисования прямоугольников

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример Canvas</title>
<style type="text/css">
  #mycanvas {border:dashed 1px} // штриховая граница canvas
</style>
<!--[if IE]>
   <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body>
<canvas id="mycanvas" width="200" height="150" >
 Ваш браузер не поддерживает canvas
</canvas>
</body>
<script type="text/javascript">
window.onload=function(){ // выполняется по окончании загрузки документа
  var canvas = document.getElementById('mycanvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');
    ctx.fillStyle="rgba(0,255,0,0.5)"; // цвет заливки
    ctx.fillRect(30,40,100,50);        // рисуем залитый прямоугольник
    ctx.strokeStyle="red";             // цвет контура
    ctx.strokeRect(50,60,100,50)       // рисуем прямоугольный контур
  }
}
</script>
</html>
```

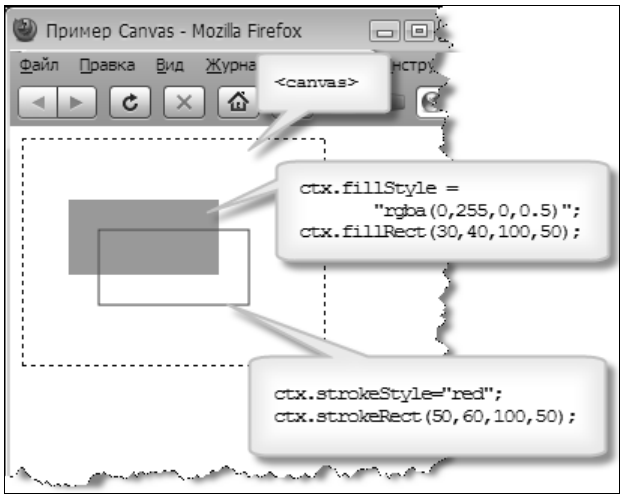


Рис. 16.14. Пример рисования прямоугольников (листинг 16.22)

Путь

Для рисования сложных фигур и линий применяются так называемые пути (path). Путь содержит информацию о контуре фигуры, но сам по себе не отображается. Для отображения фигуры, ограниченной заданным контуром, служит метод `fill()`, а для отображения только контура — метод `stroke()`.

Путь создается с помощью следующих методов:

- `beginPath()` — начинает создавать путь; последующие методы задают собственно контур (траекторию), состоящий из отрезков прямых, дуг и других линий;
- `closePath()` — завершает создание пути, замыкая контур прямой линией от текущей до начальной точки пути; применение данного метода не обязательно.

Итак, построение фигуры или линии начинается с применения метода `beginPath()`. Далее следует с помощью метода `moveTo(x,y)` указать на холсте начальную опорную точку пути и с помощью специальных методов задать его участки. Простейшим участком является отрезок прямой линии, задаваемый методом `lineTo(x, y)`. Данный метод задает отрезок прямой, от текущей опорной точки до точки с указанными в качестве параметров координатами, которая становится текущей опорной точкой. Участки более сложной формы задаются другими методами. Контур можно замкнуть, добившись совпадения текущей и начальной опорных точек, а также с помощью метода `closePath()`.

Отобразить фигуру и/или линию, заданную посредством пути, можно по-разному. Чтобы показать на холсте только контур, следует применить метод `stroke()`, для отображения только области заливки — метод `fill()`, а для того и другого — оба данных метода. Как уже отмечалось, по умолчанию цвет контура и заливки черный, а толщина линии — 1 пиксел. Чтобы установить данные параметры по своему усмотрению, достаточно воспользоваться свойствами `strokeStyle`, `fillStyle` и `lineWidth`. Установку значений этих свойств необходимо производить до применения методов отображения `stroke()` и `fill()`.

На рис. 16.15 приведены примеры построения пути и отображения соответствующих фигур и линий. Обратите внимание, что здесь не использовался метод `closePath()` замыкания пути, поэтому линия контура получилась незамкнутой. При этом область заливки сформировалась путем автоматического замыкания пути. Чтобы линия контура также оказалась замкнутой, достаточно применить `closePath()` до вызова методов отображения.

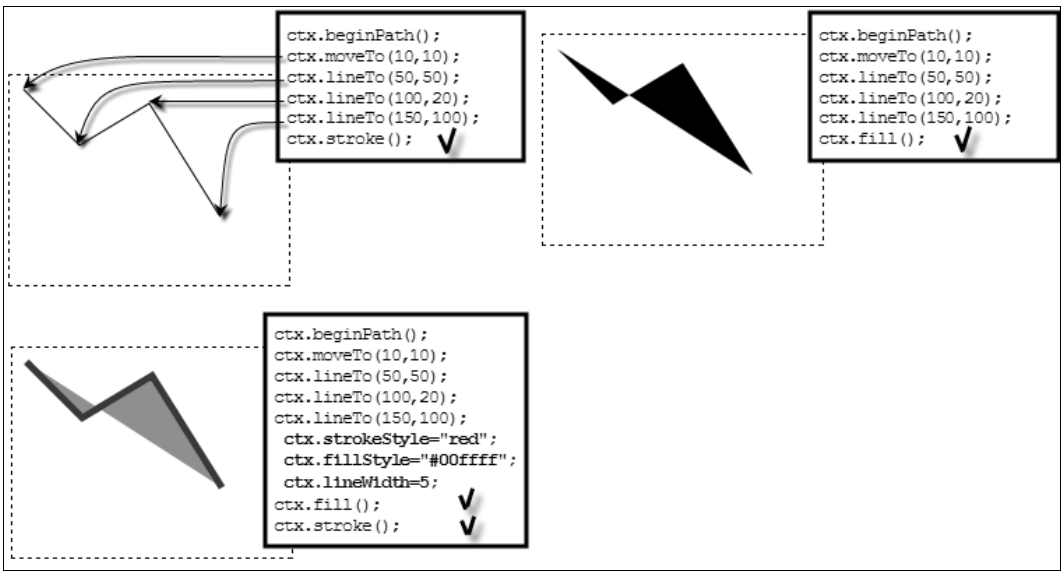


Рис. 16.15. Пример использования пути для рисования фигур

Линии

Сложную линию строят из более простых. Самую простую из них, создаваемый методом `lineTo()` отрезок прямой, мы уже рассмотрели. Теперь перейдем к кривым линиям.

Дуга окружности создается методом `arc(x, y, r, a1,a2)`, принимающим параметры:

- *x,y* — координаты центра окружности, пикселы;
- *r* — радиус окружности, пикселы;
- *a1* — начальный угол, радианы;
- *a2* — конечный угол, радианы;
- `anticlockwise` — направление рисования дуги (`true` — против часовой стрелки, `false` — по часовой стрелке).

Положительные значения углов `a1` и `a2` отсчитываются относительно горизонтали по часовой стрелке, а отрицательные — против. Чтобы перевести значение угла в градусах в радианы, необходимо умножить его на величину $\pi/2$.

На рис. 16.16 показаны схема и несколько примеров использования метода `arc()`.

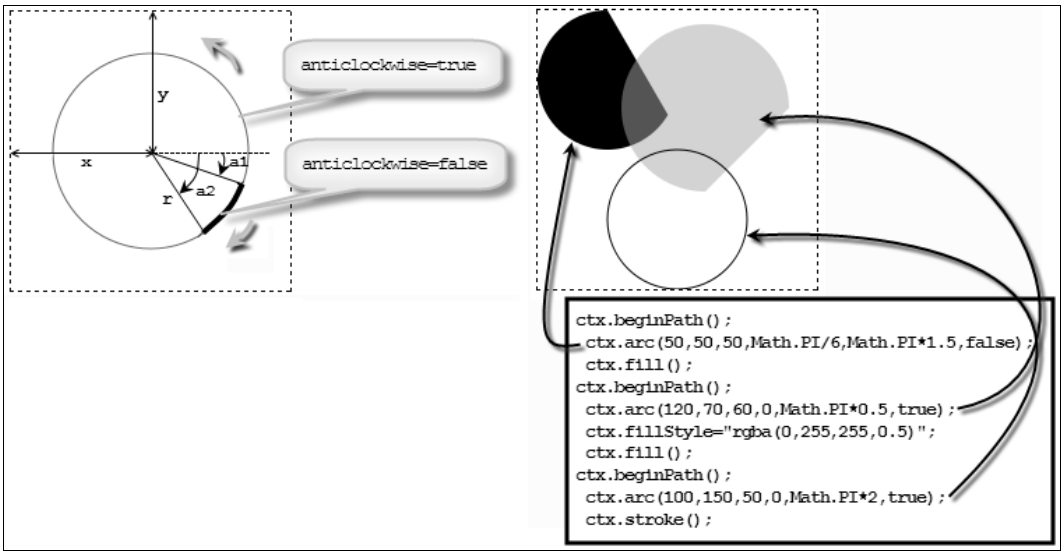


Рис. 16.16. Применение метода рисования дуг окружности

Еще более сложные линии получаются применением методов построения кривых Безье второго и третьего порядков (квадратичных и кубических кривых):

- `quadraticCurveTo(cpx, cpy, x, y)` — кривая второго порядка;
- `bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)` — кривая третьего порядка.

Данные методы принимают в качестве параметров координаты контрольных точек `cpx`, `cpy` (у кривой второго порядка их одна, а у кривой третьего порядка — две) и координаты `x, y` конца кривой. Начало кривой устанавливается либо методом `moveTo()`, либо ранее примененным методом создания кривой. Положение контрольных точек относительно начала и конца кривой определяют форму последней.

На рис. 16.17 показаны схема и пример использования метода `quadraticCurveTo()`.

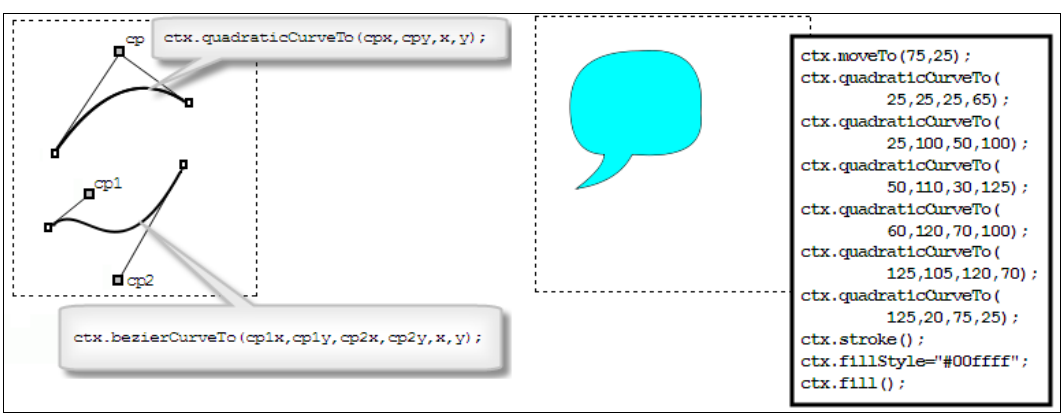


Рис. 16.17. Применение методов рисования кривых Безье

Панель с закругленными углами

Для создания панели с закругленными углами можно воспользоваться кривыми второго порядка. В листинге 16.23 приведен код функции, которая рисует такую панель, а на рис. 16.18 показано несколько примеров ее использования. Для создания правильной панели необходимо, чтобы радиус скругления был не больше полуширины и полувысоты прямоугольника. В противном случае можно получить весьма причудливую фигуру.

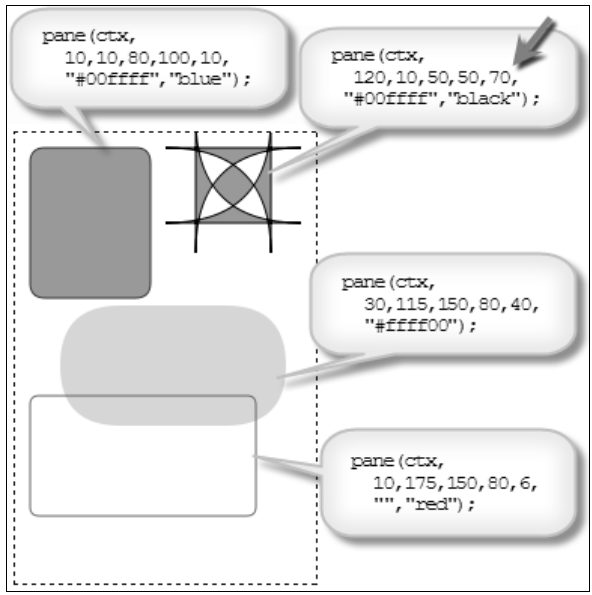


Рис. 16.18. Применение функции `pane()` (листинг 16.23)

Листинг 16.23. Функция рисования панели с закругленными углами

```
function pane(ctx,x,y,w,h,r,fillcolor,strokecolor){
/* Параметры:
   ctx — объект контекста вывода
   x,y — координаты верхнего левого угла
   w, h — ширина и высота
   r — радиус закругления углов
   fillcolor — цвет фона
   strokecolor — цвет контура
*/
 ctx.beginPath();
ctx.moveTo(x,y+r);
ctx.lineTo(x,y+h-r);
ctx.quadraticCurveTo(x,y+h,x+r,y+h);
ctx.lineTo(x+w-r,y+h);
ctx.quadraticCurveTo(x+w,y+h,x+w,y+h-r);
ctx.lineTo(x+w,y+r);
ctx.quadraticCurveTo(x+w,y,x+w-r,y);
```

```
ctx.lineTo(x+r,y);
ctx.quadraticCurveTo(x,y,x,y+r);
if (fillcolor) {
 ctx.fillStyle=fillcolor; //окрашивание
 ctx.fill();
}
if (strokecolor) {
  ctx.strokeStyle=strokecolor; //окантовка
  ctx.stroke();
}
}
```

16.10.3. Градиенты

Сначала создается объект линейного или радиального градиента, затем устанавливаются параметры цветовых переходов. Градиент применяется путем назначения его свойству `fillStyle`.

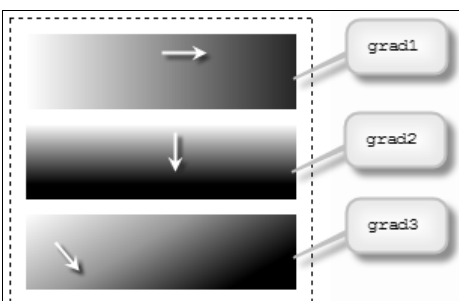
Объект градиента создается следующими методами:

- `createLinearGradient(`*x1*`,`*y1*`,`*x2*`,`*y2*`)` — линейный градиент; параметры (координаты) задают прямоугольную область и направление перехода цветов;
- `createRadialGradient(`*x1*`,`*y1*`,`*r1*`,`*x2*`,`*y2*`,`*r2*`)` — радиальный градиент; параметры задают координаты центров и радиусов кругов перехода цветов.

Опорные цвета градиента задаются методом `addColorStop(`*позиция*`, `*цвет*`)`. Параметр *позиция* принимает значения от 0.0 до 1.

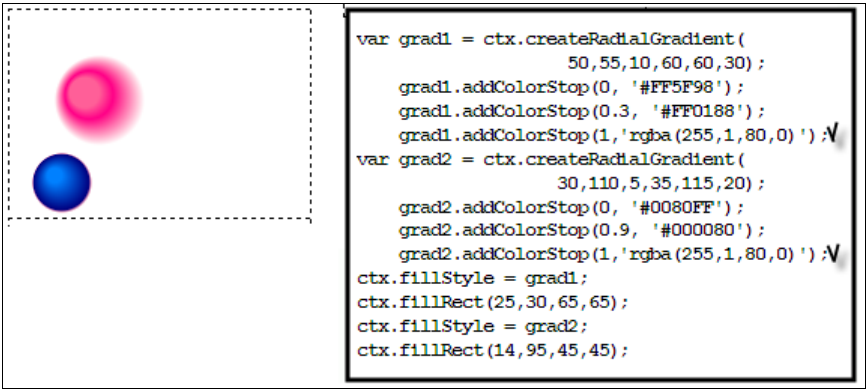
На рис. 16.19 показаны примеры заливки четырехугольника линейным градиентом. Стрелками указаны направления переходов цветов.

На рис. 16.20 показаны примеры заливки четырехугольника радиальным градиентом. Шаровидные формы получились благодаря тому, что последние опорные цвета в градиентах задавались как полностью прозрачные. Однако в Internet Explorer (с библиотекой excanvas.js от марта 2009 г.) вместо шариков получаются квадратики.




```
var grad1=ctx.createLinearGradient(10,10,200,10);
    grad1.addColorStop(0,"#ffffff");
    grad1.addColorStop(1,"#000080");
var grad2=ctx.createLinearGradient(10,70,10,120);
    grad2.addColorStop(0,"#ffffff");
    grad2.addColorStop(0.3,"#00ffff");
    grad2.addColorStop(0.8,"#000000");
var grad3=ctx.createLinearGradient(10,130,80,250);
    grad3.addColorStop(0,"#ffffff");
    grad3.addColorStop(0.4,"#00ffff");
    grad3.addColorStop(0.8,"#000000");
ctx.fillStyle = grad1;
ctx.fillRect(10,10,180,50);
ctx.fillStyle = grad2;
ctx.fillRect(10,70,180,50);
ctx.fillStyle = grad3;
ctx.fillRect(10,130,180,50);
```

Рис. 16.19. Примеры линейного градиента



```
var grad1 = ctx.createRadialGradient(
                    50,55,10,60,60,30);
    grad1.addColorStop(0, '#FF5F98');
    grad1.addColorStop(0.3, '#FF0188');
    grad1.addColorStop(1,'rgba(255,1,80,0)');
var grad2 = ctx.createRadialGradient(
                    30,110,5,35,115,20);
    grad2.addColorStop(0, '#0080FF');
    grad2.addColorStop(0.9, '#000080');
    grad2.addColorStop(1,'rgba(255,1,80,0)');
ctx.fillStyle = grad1;
ctx.fillRect(25,30,65,65);
ctx.fillStyle = grad2;
ctx.fillRect(14,95,45,45);
```

Рис. 16.20. Примеры радиального градиента

16.10.4. Трансформации

Трансформации (перенос, поворот, масштабирование) применяют ко всему контексту вывода (подложке), создавая новый контекст, наложенный на текущий. Вот методы трансформации:

- `translate(x, y)` — перенос в точку с указанными координатами;
- `rotate(угол)` — поворот на указанный угол в радианах; положительные значения угла отсчитываются относительно горизонтали по часовой стрелке, а отрицательные — против;
- `scale(kx, ky)` — масштабирование по горизонтали и вертикали с указанными коэффициентами;
- `transform(a,b,c,d,e,f)` — матричное преобразование с указанными коэффициентами; позволяет выполнить как перечисленные выше, так и более сложные преобразования, например наклон, отражение и др.;
- `setTransform(a,b,c,d,e,f)` — задает новое матричное преобразование вместо всех ранее установленных.

Прежде всего, следует иметь в виду, что результат трансформации накладывается на уже существующий контекст вывода. На рис. 16.21, слева, показан пример, в котором выводится прямоугольник, затем применяется поворот на 30° ($\pi/6$) и рисуется второй прямоугольник с теми же координатами и размерами, но с другим цветом. Обратите внимание, поворот не влияет на положение первого прямоугольника. Эта операция поворачивает подложку, и второй прямоугольник рисуется уже на ней, относительно ее системы координат. После этого снова применяется поворот на 30° и рисуется третий прямоугольник с такими же координатами и размерами. Однако он повернется на 30° относительно второго контекста и на 60° — относительно первого. При этом все прямоугольники видны, поскольку подложки по умолчанию прозрачны.

Справа на рис. 16.21 показан такой же пример, но с некоторой модификацией кода. А именно здесь были применены методы `save()` сохранения установок текущего контекста и `restore()` восстановления сохраненных ранее установок. В результате третий прямоугольник повернулся на 30° относительно исходного состояния и накрыл собой

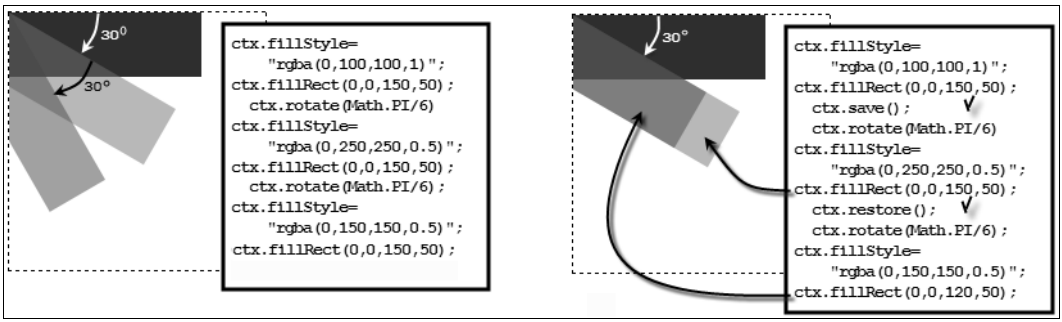


Рис. 16.21. Примеры использования метода `rotate()`

второй прямоугольник. Чтобы это было заметно, ширина третьего прямоугольника уменьшена по сравнению с шириной второго.

На рис. 16.22 показано применение операций переноса и масштабирования. При этом оба варианта кода дают одинаковый результат. В первом варианте последовательно применяются методы `translate()` и `scale()`, а во втором — метод `transform()` эквивалентного матричного преобразования. Для наглядности показан исходный квадрат размером 20×20 px, который позиционирован в начале координат исходного холста. Этот квадрат выводится после применения операций трансформации благодаря восстановлению ранее сохраненного контекста, соответствующего исходному состоянию.

Следует отметить, что результат последовательности нескольких операций трансформации зависит от порядка их применения. Так, если в примере на рис. 16.22 поменять местами методы `translate()` и `scale()`, то результат окажется иным.

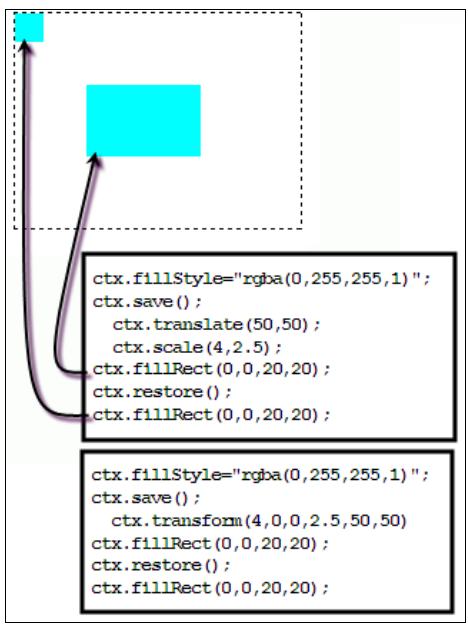


Рис. 16.22. Пример использования методов `translate()`, `scale()` и `transform()`

16.10.5. Импорт растровых графических изображений

Растровые графические изображения форматов JPEG, PNG и GIF можно импортировать в область отображения <canvas> и использовать совместно с рисованной графикой. Для этого применяется метод

drawImage(*image*, *x*, *y*, *width*, *height*),

принимающий в качестве параметров соответственно ссылку на объект графического изображения, координаты позиционирования на холсте, ширину и высоту прямоугольника, в котором картинка будет отображаться. Если последние два параметра не указаны, то картинка отображается в собственных размерах, а в противном случае она масштабируется. Возможны дополнительные параметры, но об этом позже.

Как уже отмечалось, метод drawImage() в качестве первого параметра принимает ссылку на объект, содержащий графическое изображение. При этом само графическое изображение уже должно быть загружено. Если требуется картинка из внешнего файла, то загрузить ее в браузер можно посредством объекта Image (см. *разд. 15.6.4*) следующим образом:

```
var myimg=new Image();     // объект изображения
myimg.src="URL-адрес графического изображения";  // загрузка изображения
```

Здесь переменная myimg содержит ссылку на объект графического изображения, которая может быть передана в качестве параметра методу drawImage().

Можно также использовать ссылки на объекты, уже содержащие изображения, которые соответствуют элементам документа <img> или <canvas>. Для получения ссылок на такие объекты достаточно вызвать метод

```
document.getElementById("значение id ");
```

В листинге 16.24 приведен (X)HTML-код со сценарием, осуществляющим импорт графического изображения в элемент <canvas> посредством объекта Image, а на рис. 16.23 показан его вид в браузере. Границы холста показаны штриховой линией.

Листинг 16.24. Импорт растрового изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример Canvas</title>
<style type="text/css">
    #mycanvas {border:dashed 1px} // граница canvas
</style>
<!--[if IE]>
   <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body>
<canvas id="mycanvas" width="300" height="200" >
 Ваш браузер не поддерживает canvas
</canvas>
</body>
```

```
<script type="text/javascript">
window.onload=function(){
  var canvas = document.getElementById('mycanvas');
  if (canvas.getContext){
     var ctx = canvas.getContext('2d');
     var myimg=new Image(); // объект изображения
     myimg.onload=function(){ // вывод изображения
                    ctx.drawImage(myimg,10,10);
                  }
     myimg.src="picture.jpg";  // загрузка изображения
  }
}
</script>
</html>
```

Рис. 16.23. Импорт растрового изображения в элемент `<canvas>`

Метод `drawImage()` имеет еще четыре дополнительных параметра, с помощью которых можно задать четырехугольную область исходного изображения, которую требуется отобразить на холсте. Иначе говоря, есть возможность выбора фрагмента картинки (клипа).

Для отображения клипа применяется следующий синтаксис метода `drawImage()`:

```
drawImage(image,x1,y1,width1,height1, x2,y2,width2,height2)
```

Здесь параметры `x1,y1,width1,height1` определяют клип изображения-источника, а `x2,y2,width2,height2` — область, занимаемую этим клипом на холсте (рис. 16.24).

В листинге 16.25 приведен (X)HTML-код документа, вид которого в браузере показан на рис. 16.24. Данный документ содержит элементы `<img>` и `<canvas>`. С помощью сценария из картинки элемента `<img>` вырезается клип и отображается в элементе `<canvas>`. Разумеется, клип можно получить аналогичным образом и из объекта `Image`, а также из содержимого другого элемента `<canvas>`.

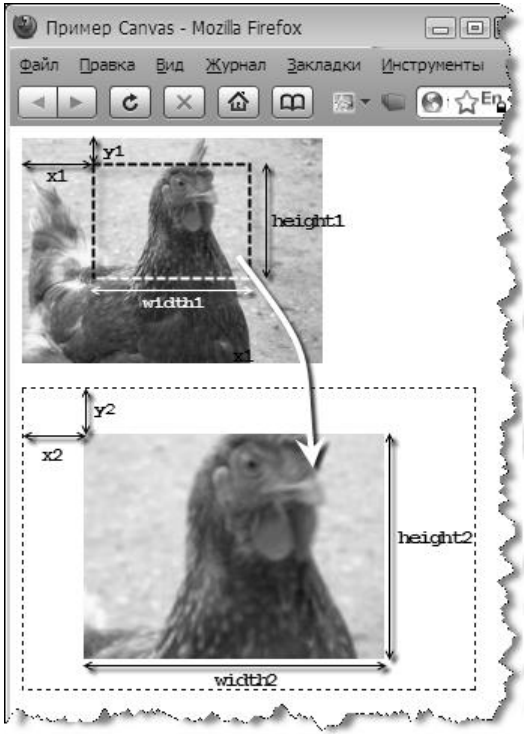


Рис. 16.24. Отображение на холсте клипа графического изображения

Листинг 16.25. Отображение клипа графического изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример Canvas</title>
<style type="text/css">
#mycanvas {border:dashed 1px}
</style>
<!--[if IE]>
   <script type="text/javascript" src="excanvas.js">
   </script>
<![endif]-->
</head>
<body>
<img id="img1" src="picture.jpg"/><p>
<canvas id="mycanvas" width="300" height="200" >
 Ваш браузер не поддерживает canvas
</canvas>
</p>
</body>
<script type="text/javascript">
```

```
window.onload=function(){
  var canvas = document.getElementById('mycanvas');
  if (canvas.getContext){
     var ctx = canvas.getContext('2d');
     ctx.drawImage(
        document.getElementById("img1"),
        50,20,100,75, 40,30,200,150);
  }
}
</script>
</html>
```

Примечание

В Internet Explorer при использовании библиотеки excanvas.js (март 2009 г.) не удается отобразить клип. Кроме того, при обычном импортировании изображения с помощью объекта `Image` (листинг 16.24) установку значения для `myimg.src` следует записать после определения обработчика `myimg.onload`. Это важно для Internet Explorer, хотя для других браузеров безразлично, где будет записана установка `myimg.src`, до или после `myimg.onload`.

16.10.6. Анимация

Для создания эффектов изменения содержимого элемента `<canvas>` во времени применяют методы `setInterval()`, `clearInterval()`; `setTimeout()` и `clearTimeout()` (см. *разд. 15.8*).

В листинге 16.26 приведен пример, в котором квадратик движется по волнистой траектории (синусоиде), которая рисуется вслед за ним (рис. 16.25). Получается что-то вроде осциллограммы. При этом квадратик, достигнув границы холста, меняет направление движения на противоположное.

Листинг 16.26. Анимация рисования линии и движения фигуры

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Пример Canvas</title>
<style type="text/css">
#mycanvas {border:dashed 1px;}
</style>
<!--[if IE]>
   <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body>
<canvas id="mycanvas" width="300" height="100" >
 Ваш браузер не поддерживает canvas
</canvas>
</body>
<script type="text/javascript">
window.onload=function(){
  var canvas = document.getElementById('mycanvas');
```

```
  if (canvas.getContext){
     ctx = canvas.getContext('2d');
     ctx.beginPath();  // открываем путь
     ctx.moveTo(2,70); // начальная точка
     x=0;
     dx=1;
     k=1;
     setInterval("draw()",100) // периодически вызываем draw()
  }
}
function draw() {
  if (x>=300) k=-1; // переключатель движения налево
  if (x<=0) k=1;    // переключатель движения направо
  ctx.clearRect(0,0,300,120); // очищаем область
  x=x+k*dx; // горизонтальная координата
  var y=50+25*Math.cos(20*Math.PI*x/400);// вертикальная координата
  ctx.lineTo(x,y); // добавляем линию к пути
  ctx.stroke();    // рисуем линию
  ctx.fillStyle="red";
  ctx.fillRect(x,y,10,10); // рисуем прямоугольник
}
</script>
</html>
```

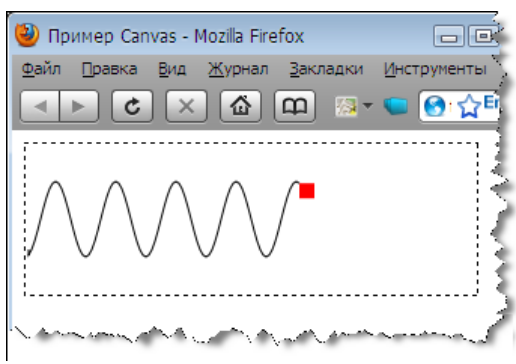


Рис. 16.25. Анимация рисования линии и движения фигуры (листинг 16.26)

В следующем примере две копии одного и того же графического изображения самолета (на прозрачном фоне) вращаются по окружности (рис. 16.26). Чтобы задать подходящий радиус окружности, здесь применяется операция переноса. Собственно вращение обеспечивается операцией поворота. Фон холста в виде графического изображения неба задан стилевым параметром `background` для элемента `<canvas>`. Соответствующий код приведен в листинге 16.27.

Листинг 16.27. Движение графического изображения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Canvas для работы с графикой</title>
<!--[if IE]>
  <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body >
<canvas id="mycanvas" width="400" height="300"
        style="background:url('sky.jpg')" >
 Ваш браузер не поддерживает &lt;canvas&gt;
</canvas>
</body>
<script id="myscript">
window.onload=function(){ // вращение картинки
  ctx = document.getElementById('mycanvas').getContext('2d');
  img = new Image();
  img.onload = function(){
    ctx.translate(150,200);
    setInterval("draw(-10)",100) // периодически вызываем draw
  }
  img.src="airplane.gif";
}
function draw(a) {
  ctx.clearRect(-20,-20,400,300);
  ctx.drawImage(img,40,30);   // 1-я копия с исх. размерами
  ctx.drawImage(img,0,0,40,40)// 2-я копия 40x40px
  ctx.translate(10,15);       // перемещаем холст
  ctx.rotate(Math.PI*a/180);  // поворачиваем на a град.
}
</script>
</html>
```

Рис. 16.26. Движение графического изображения (листинг 16.27)

Примечание

В Internet Explorer 8.0 при использовании библиотеки excanvas.js (март 2009 г.) код листинга 16.27 работает только в режиме предоставления совместимости. В других браузерах проблем не возникает.

16.10.7. Композиция графики

При позиционировании изображений случается, что одно из них перекрывает, полностью или частично, другое. По умолчанию изображение, выводимое позже, накладывается на изображения, выведенные скриптом ранее. Такое правило в большинстве случаев оказывается вполне естественным. Вместе с тем, можно установить различные типы композиции изображений, определяющие, какое изображение должно получиться в результате.

Тип композиции задается свойством `globalCompositeOperation`, которое может принимать строковые значения: `source-over` (по умолчанию), `source-in`, `source-out`, `source-atop`, `destination-over`, `destination-in`, `destination-out`, `destination-atop`, `lighter`, `darker`, `copy`, `xor`.

Я не буду подробно комментировать, что именно происходит при установке того или иного типа композиции, а просто покажу на примере. В листинге 16.28 приведен (X)HTML-код с шестнадцатью элементами `<canvas>` и сценарием, который выводит в них пересекающиеся квадрат и круг при различных типах композиции. В коде сначала записан вывод квадрата, а затем круга, но отображение их на холсте определяется типом композиции. На рис. 16.27 показано, как выглядят результаты композиции в Firefox 4.0. В других браузерах некоторые типы композиции отображаются иначе.

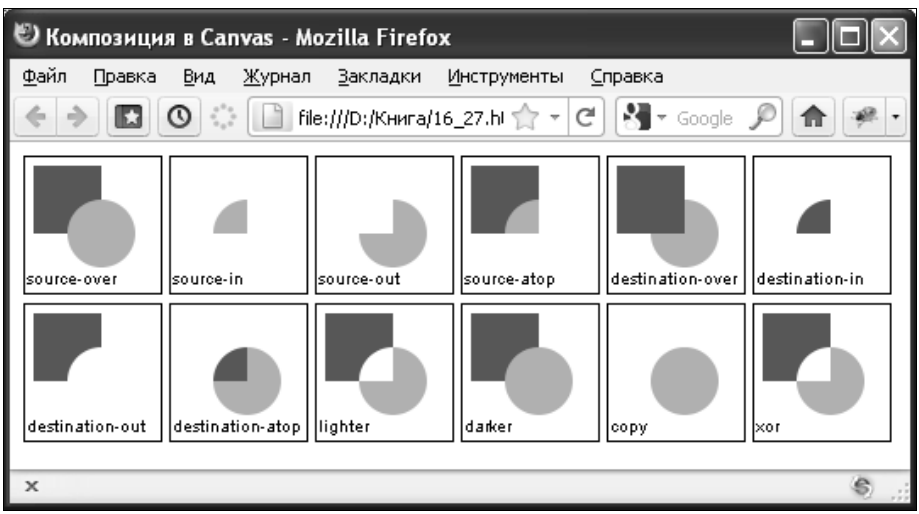


Рис. 16.27. Типы композиции графических изображений (листинг 16.28)

Листинг 16.28. Применение различных типов композиции

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<head><title>Композиция в Canvas</title>
<style type="text/css">
  canvas {border: solid 1px;}
</style>
<!--[if IE]>
  <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body >
<canvas id="canv0"  width="80" height="80"></canvas>
<canvas id="canv1"  width="80" height="80"></canvas>
<canvas id="canv2"  width="80" height="80"></canvas>
<canvas id="canv3"  width="80" height="80"></canvas>
<canvas id="canv4"  width="80" height="80"></canvas>
<canvas id="canv5"  width="80" height="80"></canvas>
<canvas id="canv6"  width="80" height="80"></canvas>
<canvas id="canv7"  width="80" height="80"></canvas>
<canvas id="canv8"  width="80" height="80"></canvas>
<canvas id="canv9"  width="80" height="80"></canvas>
<canvas id="canv10" width="80" height="80"></canvas>
<canvas id="canv11" width="80" height="80"></canvas>
</body>
<script type="text/javascript">
window.onload=draw;
function draw(){
  var compositeTypes = [
    'source-over','source-in','source-out','source-atop',
    'destination-over','destination-in',
    'destination-out','destination-atop',
    'lighter','darker','copy','xor'
  ];
/* Вывод в элементы canvas прямоугольника и круга */
  for (i=0;i<compositeTypes.length;i++){
    var ctx = document.getElementById('canv'+i).getContext('2d');
    /* прямоугольник */
    ctx.fillStyle = "#00bbff";
    ctx.fillRect(5,5,40,40);
    /* установка типа композиции */
    ctx.globalCompositeOperation = compositeTypes[i];
    /* круг */
    ctx.fillStyle = "#ff5500";
    ctx.beginPath();
    ctx.arc(45,45,20,0,Math.PI*2,true);
    ctx.fill();

    /* название типа композиции */
    if (ctx.fillText) {
       ctx.globalCompositeOperation = compositeTypes[0];
```

```
        ctx.fillStyle="#000000";
        ctx.fillText(compositeTypes[i],1,75)
      }
    }
}
</script>
</html>
```

16.10.8. Текст

Вставка текста в холст осуществляется методом

`fillText(`*текст*`, x, y, maxwidth)`,

где последний необязательный параметр задает максимальную ширину строки текста. Для вывода только контура теста применяется метод

`strokeText(`*текст*`, x, y, maxwidth)`.

Форматирование текста задают свойства `font`, значением которого является строка параметров CSS (см. *разд. 8.1*), и `textAlign` (выравнивание), принимающим в качестве значения строки вида: `"start"`, `"end"`, `"left"`, `"right"` и `"center"`. Имеются и другие свойства (например, `texBaseline`), которые здесь не рассматриваются.

Текст, указанный в качестве параметра метода вывода, отображается как одна строка и часть, которая не помещается в элементе `<canvas>`, просто отсекается. Параметр `maxwidth` задает предел по ширине выводимого текста, так что слишком длинная строка сжимается при отображении. Однако это работает только в Firefox, Opera и Internet Explorer 9, а в Safari 5 и Chrome 10 — нет. Выводить текст в несколько строк следует вручную, предварительно разбив его на части, каждую из которых придется позиционировать на холсте отдельно.

В листинге 16.29 приводится код вывода текста с различными параметрами шрифта. Внешний вид данного документа в браузере Firefox показан на рис. 16.28.

Листинг 16.29. Вывод текста

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Тексты в Canvas</title>
<style type="text/css">
  canvas {border: solid 1px;}
</style>
<!--[if IE]>
  <script type="text/javascript" src="excanvas.js"></script>
<![endif]-->
</head>
<body >
<canvas id="canv0"  width="200" height="200"></canvas>
<canvas id="canv1"  width="200" height="200"></canvas>
```

```
<canvas id="canv2"  width="200" height="200"></canvas>
<canvas id="canv3"  width="200" height="200"></canvas><br/>
</body>
<script type="text/javascript">
window.onload=draw;
function draw(){
  var mytext="Большой привет!" // строка текста
/* Массив наборов параметров шрифта */
  var myfonts = [
      "12px Arial",
      "20px Times",
      "italic bold 40pt serif",
      "bold 20pt Tahoma"
     ];
  for (i=0;i<myfonts.length;i++){
    var ctx = document.getElementById('canv'+i).getContext('2d');
    if (ctx.fillText) { // если доступен
       ctx.fillStyle="#000000";
       ctx.font=myfonts[i];              // установка шрифта
       ctx.fillText(mytext,5,60,190);    // вывод текста
       ctx.strokeText(mytext,5,120,190); // вывод контура символов текста
       /* параметры шрифта */
       ctx.fillText(myfonts[i],5,185,190);
    }
  }
}
</script>
</html>
```

Примечание

Методы вывода текста не работают в Internet Explorer 8 с excanvas.js от марта 2009 г.

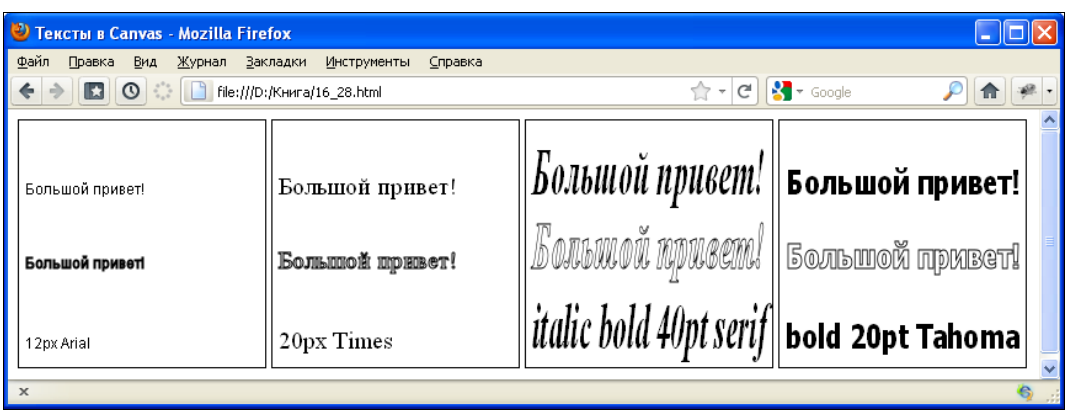


Рис. 16.28. Вывод текста в элементы <canvas> (листинг 16.29)

16.11. Дата и время

Получить данные о дате и времени позволяет встроенный объект JavaScript `Date`. Рассмотрим несколько примеров.

16.11.1. Отображение даты и времени в виде текста

Иногда требуется отобразить на странице текущую дату и/или время в виде текстовой строки. Решим эту задачу при условии, что дата и время должны периодически обновляться в соответствии с естественным течением времени.

В листинге 16.30 сценарий отображает в окне браузера текущие дату и время в виде текстовой строки, например, "26 апреля 2011 г. 11:50:15". Строка с датой и временем формируется функцией `clock()` и присваивается свойству `innerHTML` элемента `<div id="date">`. Как только мы обновим содержимое данного контейнерного тега с помощью свойства `innerHTML`, это сразу же вызовет соответствующие изменения элемента на странице. Чтобы данные о дате и времени обновлялись периодически, функция `clock()` передается в качестве параметра методу `setInterval()`, который вызывает ее с заданным периодом. Второй числовой параметр этого метода, напомню, задает временной интервал в миллисекундах вызова указанной функции.

Листинг 16.30. Отображение даты и времени в виде текстовой строки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Дата и время</title></head>
<body>
<div id="date"></div>
</body>
<script type="text/javascript">

function clock(){
  aMonth=[
    "января","февраля","марта","апреля","мая","июня",
    "июля","августа","сентября","октября","ноября","декабря"
  ];
var date=new Date(); // экземпляр объекта даты
var sec=date.getSeconds();   // секунды
var min=date.getMinutes();   // минуты
var hour=date.getHours();    // часы
var day=date.getDate();      // число
var month=date.getMonth();   // месяц
var year=date.getFullYear(); // год
var xstr=day+" "+aMonth[month]+" "+year+"г. "+
hour+":"+min+":"+sec; // строка с датой и временем
/* Отображаем в элементе <div id="date">: */
document.getElementById("date").innerHTML=xstr; // запись в документ
}
```

```
setInterval("clock()",1000); // обновляем каждую секунду
</script>
</html>
```

Для браузеров Internet Explorer и Firefox (но не для Safari и Chrome) код функции `clock()` можно упростить, задействовав метод `toLocaleString()`, который возвращает строку с датой и временем в локализованном формате (например, на русском языке) операционной системы:

```
function clock(){
  var date=new Date(); // экземпляр объекта даты
  document.getElementById("date").innerHTML=date.toLocaleString()
}
```

16.11.2. Часы

В *разд. 16.9* рассказывалось о том, как можно нарисовать линию, а затем заменить ее другой, т. е. перерисовать. Сейчас мы рассмотрим воплощение этой нехитрой идеи на примере создания стрелочных часов. Вместе с тем, стрелки можно создать с помощью функции `line()`, представленной в листинге 16.15, но несколько модифицированной: все вхождения выражения `document.write(xstr)` необходимо удалить. Таким образом, модифицированная функция `line()` сама ничего не отображает в окне браузера, а только возвращает строку тегов, которые затем используются для отображения линии во внешнем сценарии. Циферблат часов, разумеется, мы возьмем из подходящего графического файла, заимствованного или созданного самостоятельно с помощью графического редактора. В данном примере он имеет размеры 270×270 px. Изображение циферблата определим в качестве фона элемента `<div>`, для которого зададим размеры, равные размерам фоновой картинки. Стрелки, рисуемые посредством функции `arrows()` (вызывающей `line()`), поместим в тот же элемент `<div>`, предварительно определив координаты точки их начала (центр изображения циферблата). Здесь они равны половине размеров квадратного графического изображения, т. е. 135 px. Положение стрелок (координаты их концов) вычисляются с помощью методов объекта `Date`, а также несложных тригонометрических расчетов с использованием объекта `Math`.

В листинге 16.31 приведен пример кода создания стрелочных часов, а их внешний вид показан на рис. 16.29.

Листинг 16.31. Стрелочные часы

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Часы</title>
<style type="text/css">
  body {padding:0;margin:0;background:#d0d0d0}
  #line {width:270px;height:270px;background:url("clock.gif")}
</style>
</head>
```

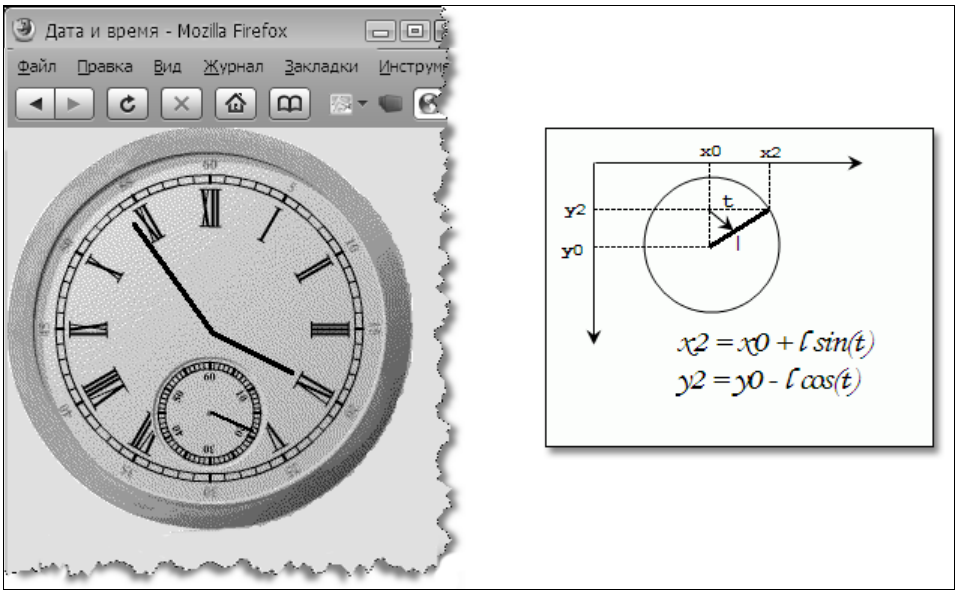


Рис. 16.29. Стрелочные часы (листинг 16.31)

```
<body>
<div id="date"></div>
</body>
<script type="text/javascript">
setInterval("arrows()",1000); // периодический вызов arrows()

function arrows(){
  var x0=135,y0=189, l=30;     // начало и длина секундной стрелки
  var date=new Date();
  var sec=date.getSeconds();
  var x2=parseInt(l*Math.sin(Math.PI/30*sec))+x0;
  var y2=y0-parseInt(l*Math.cos(Math.PI/30*sec));
 var xtime=line(x0,y0,x2,y2,2);
/***********************************/
  y0=135, l=90; // длина минутной стрелки

  var min=date.getMinutes();
  x2=parseInt(l*Math.sin(Math.PI/30*min))+x0;
  y2=y0-parseInt(l*Math.cos(Math.PI/30*min));
  xtime=xtime+line(x0,y0,x2,y2,3);
/***********************************/
  l=60;// длина часовой стрелки
  var hour=date.getHours();
  if (hour>12) hour=hour-12;
  hour=hour+min/60;
  x2=Math.round(l*Math.sin(Math.PI/6*hour))+x0;
  y2=y0-Math.round(l*Math.cos(Math.PI/6*hour));
  xtime=xtime+line(x0,y0,x2,y2,3);
```

```
  <!-- Запись в документ: -->
  document.getElementById("date").innerHTML=xtime;
}

function line(x1, y1, x2 ,y2, n,color){
... // см. листинг 16.15; модификация: удалить document.write(xstr)
}

</script>
</html>
```

16.11.3. Вечный календарь

В данном разделе мы продолжим рассмотрение способов работы с датой: создадим "вечный" календарь, который показывает раскладку чисел и дней недели на любой месяц в любом году.

В данной задаче есть два основных момента. Во-первых, необходимо знать алгоритм определения високосных годов. Во-вторых, предстоит поработать с отображением полученной информации о датах в привычном табличном формате.

Создадим сначала разметку для отображения календаря, т. е. только (X)HTML-код без скриптов (листинг 16.32, рис. 16.30).

Листинг 16.32. Разметка календаря

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Календарь</title>
  </head>
  <body>
  <!-- Разметка календаря -->
  <div id="calcontrol" style="text-align:center">
        <select id="month" onchange="selectdate()">
          <option/>Январь
          <option/>Февраль
          <option/>Март
          <option/>Апрель
          <option/>Май
          <option/>Июнь
          <option/>Июль
          <option/>Август
          <option/>Сентябрь
          <option/>Октябрь
          <option/>Ноябрь
          <option/>Декабрь
        </select>
       <input id="year" type=text size=4 maxlength=4
              onchange="selectdate()"
        />
```

```
      <input type="button" id="go" value=" OK "
            onclick="selectdate()"
            title="Если изменили год, щелкните здесь"
       />

      <div id="calbuttons">
       <textarea id="calpage"
         style="font-family:monospace; font-size:10pt;
               text-align:center"
         wrap=no rows=8 cols=24 >
      </textarea>
      <div>
      <input type=button id="prevyear" value="<<"
            onclick="setprevyear()"
            title="год -"
      />
      <input type=button id="prevyear" value="< "
            onclick="setprevmonth()"
            title="месяц -"
      />
      <input type=button id="prevyear2" value=""
            onclick="settoday()"
            title="Щелкните, чтобы показать календарь на текущую дату"
      />
      <input type=button id="prevyear" value=" >"
            onclick="setnextmonth()"
            title="месяц +"
      />
      <input type=button id="prevyear" value=">>"
            onclick="setnextyear()"
            title="год +"
      />
     </div>
    </div>
  </body>
</html>
```

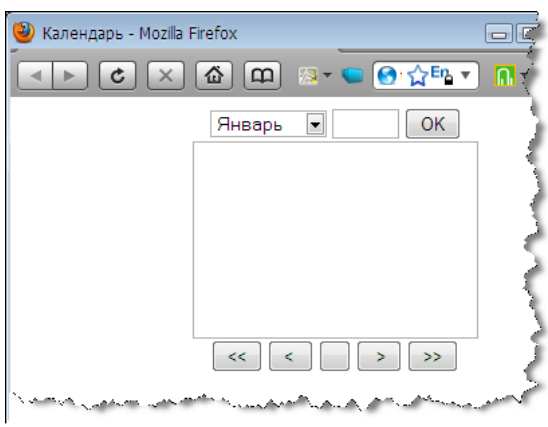


Рис. 16.30. Разметка календаря

Теперь осталось дописать код сценария, который обеспечит функциональность календарю (листинг 16.33, рис. 16.31). Функция `visyear()` определяет, является ли год високосным по следующему алгоритму: год високосный, если только его номер делится нацело на 4 и к тому же верно хотя бы одно из двух высказываний: он не делится нацело на 100; он делится нацело на 400. Например, год 1400 не високосный, т. к. его номер не делится нацело на 400; год 2012 високосный, поскольку его номер делится нацело на 4 и не делится без остатка на 100.

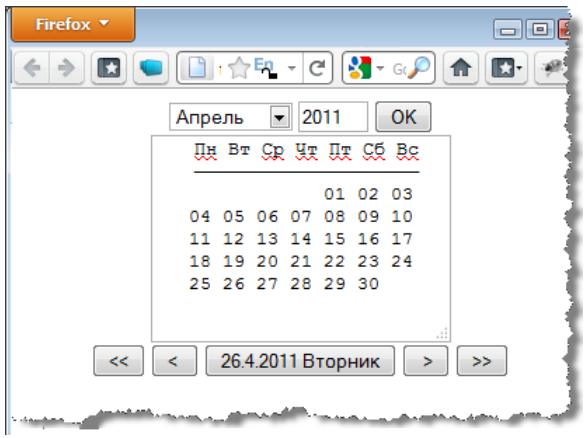


Рис. 16.31. Внешний вид календаря

Листинг 16.33. Календарь

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Календарь</title>
  </head>
<body>
  <!-- Здесь код разметки календаря, см. листинг 20.32 -->
</body>
<script type="text/javascript">
    /* Запускаем календарь по окончанию загрузки страницы: */
    window.onload=settoday;
/*******************************************************/
    function settoday() {      // установка текущей даты
      var now  = new Date();
      var day  = now.getDate();
      var month= now.getMonth();
      var year = now.getYear();
      var anameday=new Array("Воскресенье", "Понедельник",
          "Вторник", "Среда", "Четверг", "Пятница", "Суббота")
      var nameday=anameday[now.getDay()] // название дня недели
      if (year < 2000) year = year + 1900;
```

```
      document.getElementById("month").selectedIndex=month;
      document.getElementById("year").value=year;
      displaycalendar(month, year);
      var nowdate=day+"." + parseInt(month+1) + "." + year
      document.getElementById("prevyear2").value=nowdate + " " + nameday
    }
/*******************************************************/
    function isfourdigityear(year) {
      if (year.length != 4) {
        alert ("Год должен содержать 4 цифры");
        document.getElementById("year").select();
        document.getElementById("year").focus();
      } else  return true;
    }
/*******************************************************/
    function selectdate() {
      var year =document.getElementById("year").value;
      if (isfourdigityear(year)) {
        var day  =0;
        var month=document.getElementById("month").selectedIndex;
        displaycalendar(month, year);
      }
    }
/*******************************************************/
    function setprevyear() {   // предыдущий год
      var year =document.getElementById("year").value;
      if (isfourdigityear(year)) {
        var day  =0;
        var month=document.getElementById("month").selectedIndex;
        year--;
        document.getElementById("year").value=year;
        displaycalendar(month, year);
      }
    }
/*******************************************************/
    function setprevmonth() {  // предыдущий месяц
      var year=document.getElementById("year").value;
      if (isfourdigityear(year)) {
        var day  =0;
        var month=document.getElementById("month").selectedIndex;
        if (month == 0) {
          month=11;
          if (year > 1000) {
            year--;
            document.getElementById("year").value=year;
          }
        } else { month-- }
        document.getElementById("month").selectedIndex=month;
        displaycalendar(month, year);
      }
    }
```

```
/*******************************************************/
    function setnextmonth() {  // следующий месяц
      var year =document.getElementById("year").value;
      if (isfourdigityear(year)) {
        var day  =0;
        var month=document.getElementById("month").selectedIndex;
        if (month == 11) {
          month=0;
          year++;
          document.getElementById("year").value=year;
        } else { month++ }
        document.getElementById("month").selectedIndex=month;
        displaycalendar(month, year);
      }
    }
/*******************************************************/
    function setnextyear() {   // следующий год
      var year=document.getElementById("year").value;
      if (isfourdigityear(year)) {
        var day=0;
        var month=document.getElementById("month").selectedIndex;
        year++;
        document.getElementById("year").value=year;
        displaycalendar(month, year);
      }
    }
/*******************************************************/
    function displaycalendar(month, year) {   // показать календарь
      month=parseInt(month);
      year=parseInt(year);
      var i=0;
      var days=getdaysinmonth(month+1,year);
      var firstofmonth=new Date (year, month, 1);
      var startingpos=firstofmonth.getDay()-1;
      if (startingpos<0) startingpos=6
      days += startingpos;
      document.getElementById("calpage").value =
      " Пн Вт Ср Чт Пт Сб Вс";
      document.getElementById("calpage").value +=
      "\n ————————————————————";
      for (i =0; i < startingpos; i++) {
        if ( i%7 == 0 ) document.getElementById("calpage").value+="\n ";
        document.getElementById("calpage").value += "   ";
      }
      for (i=startingpos; i < days; i++) {
        if ( i%7 == 0 ) document.getElementById("calpage").value+= "\n ";
        if (i-startingpos+1 < 10)
          document.getElementById("calpage").value += "0";
        document.getElementById("calpage").value += i-startingpos+1;
```

```
        document.getElementById("calpage").value += " ";
      }
      for (i=days; i < 42; i++)  {
        if ( i%7 == 0 ) document.getElementById("calpage").value+= "\n ";
        document.getElementById("calpage").value += "   ";
      }
      document.getElementById("go").focus();
    }
/*****************************************************/
    function getdaysinmonth(month,year)  { // получить день в месяце
      var days;
      if (month==1 || month==3 || month==5 || month==7 ||
           month==8 || month==10 || month==12)  days=31;
      else if (month==4 || month==6 || month==9 || month==11) days=30;
      else if (month==2) {
        if (visyear(year)) { days=29; }    // если год високосный
        else { days=28; }
      }
      return days;
    }
/*****************************************************/
    function visyear (year) {  // високосные года
      if (((year % 4)== 0) && ((year % 100)!= 0) || ((year % 400)== 0)) {
        return true;             // високосный год
      } else  return false;    // невисокосный год
    }
  </script>
</html>
```

16.12. Виджет

Виджет (widget) — это элемент управления интерфейса пользователя, имеющий видимое представление. Всплывающее окно (например, с подсказкой, картинками, кнопками, ссылками и т. п.) является примером виджета. Текстовые подсказки, всплывающие при наведении указателя мыши на элемент Web-страницы, обычно реализуют с помощью атрибута `title` тега соответствующего элемента. Такие подсказки появляются и исчезают мгновенно при соответственно наведении и уходе указателя мыши.

Рассмотрим более интересный виджет. А именно при наведении указателя мыши на элемент страницы виджет начинает медленно появляться, если указатель задержался на элементе некоторое время. Далее, полностью проявившись, виджет сохраняется в таком состоянии пока указатель мыши находится на элементе или не исчерпан лимит времени экспозиции. При уходе мыши с элемента или истечении времени экспозиции виджет медленно исчезает.

Алгоритм работы виджета в виде графа переходов его состояний представлен на рис. 16.32. В виде овалов изображены состояния с их именами, а команды управления переходами состояний представлены стрелками, помеченными именами команд. **Начало** — исходное состояние виджета, в котором он невиден (полностью прозрачен). На-

ведение указателя мыши на картинку в начальном состоянии (команда **mover**) переводит его в состояние **Пауза**, в котором он остается невидимым, но запускается таймер паузы. Истечению заданного времени паузы соответствует команда **timeout**, которая переводит виджет в состояние **Восход**. Здесь запускается процесс постепенного увеличения видимости виджета, по завершении которого запускается таймер экспозиции. По истечении времени экспозиции выдается команда **timeout**, переводящая виджет в состояние **Заход**, в котором видимость его постепенно уменьшается. Став полностью прозрачным, виджет возвращается в начальное состояние. Кроме команды **timeout** на переходы воздействует команда **mout**, соответствующая уходу указателя мыши с картинки. Если какая-то команда для состояния не указана, то предполагается, что она не изменяет данного состояния.

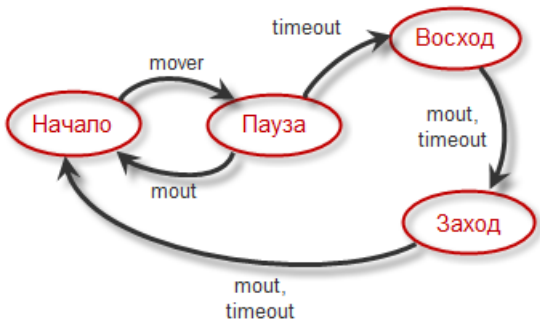


Рис. 16.32. Алгоритм работы виджета

Собственно всплывающую подсказку реализуем как элемент `<div>`, содержимое которого в простейшем случае является некоторым текстом. Вообще говоря, мы можем вставить туда еще и картинки, и другое более затейливое содержимое. Например,

```
<div id="widget1" class="widget" >
  Это всплывающая подсказка.
</div>
```

Внешний вид данного элемента определим с помощью CSS. При этом позиционируем его абсолютно (`position:absolute`) с заведомо большим значением параметра `z-index`, например так, как показано в листинге 16.34.

Листинг 16.34. Параметры CSS для виджета

```
.widget { // стилевые параметры всплывающей подсказки
   opacity:0;
   border:solid 1px;
   width:200px;height:100px;
   position:absolute;
   z-index:1000;
   background:yellow;
   padding:5px;

   box-shadow:2px 2px 1px;
   -webkit-box-shadow:2px 2px 1px;
```

```
   border-radius:8px;
  -webkit-border-radius:8px;
}
```

Переходы между состояниями и действия в различных состояниях виджета определяются посредством скрипта, приведенного в листинге 16.35.

Листинг 16.35. Сценарий, обеспечивающий функциональность виджета

```
var t1, t2, t3; // идентификаторы таймеров
var element1=document.getElementById("widget1");// объект виджета
/*******************************************************/
inst=function(event, element){ // координаты виджета
  element.style.opacity=0;
  element.style.top =event.clientY+2+'px';
  element.style.left=event.clientX+2+'px';
}
/* Массив переходов состояний автомата ********************/
var trans=[];
trans['Начало']=[];
  trans['Начало']["mover"]="Пауза";

trans['Пауза']=[];
  trans["Пауза"]["mout"]="Начало";
  trans["Пауза"]["timeout"]="Восход";

trans['Восход']=[];
  trans["Восход"]["mout"]="Заход";
  trans["Восход"]["timeout"]="Заход";

trans['Заход']=[];
  trans["Заход"]["mout"]="Начало";

/* Массив выходов (действий в состояниях) автомата ****/
var out=[]; /* здесь t1,t2,t3, element1 — переменные,
               соответствующие конкретному виджету */
  out["Начало"]=function() {
       clearInterval(t1);
       clearInterval(t2);
       clearTimeout(t3);
       element1.style.display="none";
  }

  out["Пауза"]=function() {
       element1.style.display="block";
       setTimeout("auto.input('timeout')",1000)
  }
  out["Восход"]=function() {
       clearInterval(t1);
       clearInterval(t2);
       clearTimeout(t3);
       t1=setInterval('timer1(element1)',100)
  }
```

```
out["Заход"]=function() {
       clearInterval(t1);
       clearInterval(t2);
       clearTimeout(t3);
       t2=setInterval('timer2(element1)',150)
  }

/* Вспомогательные функции, указанные в out **************/
function timer1(element){ // восход
  clearInterval(t2);
  var Dmax=1, d=0.05;
  if(parseFloat(element.style.opacity)>=Dmax){
    clearInterval(t1);
    element.style.opacity=Dmax;
    t3=setTimeout("auto.input('timeout')",10000) // экспозиция и заход
    return
  }
  element.style.opacity=parseFloat(element.style.opacity)+d
}

function timer2(element){ // заход
  clearInterval(t1);
  var Dmax=1, d=0.05;
  if(parseFloat(element.style.opacity)<=0){
    clearInterval(t2);
    element.style.opacity=0;
    auto.input("mout"); // в Начало
    return
  }
  element.style.opacity=parseFloat(element.style.opacity)-d;
}
/* Движок (автомат) ***************************************/
function automat(trans,out,stat){
  this.trans=trans; // массив переходов
  this.out=out;     // массив выходов
  this.stat=stat;   // текущее состояние
  this.input=function(inp){  // ввод команд
      if (inp&&this.trans[this.stat][inp]){
        this.stat=this.trans[this.stat][inp];
        this.output=out[this.stat]; // выход
        this.output()  // вычисление выхода
      }
      return this.stat
  }
  this.output=out[this.stat];// выход
}
/*****************************************************/
var auto=new automat(trans,out,"Начало") /* инициализация автомата */
```

Элемент, к которому привязывается виджет, может быть каким угодно. Так, если это графическое изображение, то для привязки к нему виджета можно применить такой код:

```
<img src="URL файла" alt=""
   onmouseover= "inst(event, объект виджета); auto.input('mover')"
   onmouseout = "auto.input('mout')"
/>
```

Обработчики событий `onmouseover` и `onmouseout` здесь привязаны к элементу `<img>` с помощью соответствующих атрибутов, хотя это можно сделать и по-другому. Вот что делается обработчиками данных событий:

- При наведении указателя мыши (onmouseover) на картинку вызывается функция `inst()`, которой передаются объект события `event` и объект виджета (элемента `<div>`) (например, `document.getElementById('widget1')`). Эта функция устанавливает координаты виджета, соответствующие положению указателя мыши (мы хотим, чтобы виджет появлялся рядом с указателем мыши в момент наведения на картинку, *см. далее*). После выполнения функции `inst()` обработчик выдает команду `mover` на вход автомата (экземпляр `auto` объекта `automat`, *см. далее*).
- При уходе указателя мыши (`onmouseover`) на вход автомата подается команда `mout`, которая из любого состояния возвращает его в начальное состояние.

Чтобы обработчики событий могли работать надлежащим образом, необходимо инициализировать объект автомата (так называемый движок автомата), т. е. создать его экземпляр. В данном примере это делается в скрипте с помощью выражения:

```
var auto=new automat(trans,out,"Начало")
```

Данный скрипт задает длительность паузы 1 с, а экспозиции — 10 с. Разумеется, можно установить иные значения. Длительность паузы определяется вторым параметром метода `setTimeout()` объекта `window`, который задерживает выполнение кода, указанного в качестве первого параметра. Так, в состоянии **Пауза** выход автомата (`out["Пауза"]`) содержит выражение `setTimeout("auto.input('timeout')",1000)`, которое выдает на автомат команду **timeout** с задержкой 1000 мс (т. е. 1 с). Плавность и длительность восхода и захода (мс) определяются вторыми параметрами методов `setInterva()`, а также значениями приращения CSS-параметра, определяющего прозрачность (`opacity`). Не следует чрезмерно уменьшать периодичность вызова кода, указанного в качестве первого параметра метода `setInterval()`. В конечном счете конкретные значения подбираются экспериментально.

Сценарий листинга 16.35 обеспечивает корректную работу только одного виджета. Если вам требуется на странице несколько подобных виджетов, привязанных к различным элементам, то необходимо инициализировать соответствующее количество автоматов с одинаковыми переходами состояний (массивами `trans`), но разными выходами (массивами `out`). Элементы различных массивов выходов `out` для разных виджетов идентичны. Отличными в них должны быть имена переменных для идентификаторов таймеров (в примере это t1, t2 и t3), а также имена переменных для объектов виджета (в примере объект виджета сохраняется в переменной `element1`) и экземпляра автомата (в примере это `auto`).

Таким образом, при создании нескольких виджетов на основе описываемой технологии вам придется в скрипте сделать следующее:

- Создать для каждого виджета свой экземпляр (переменную) объекта `automat`, передав ему в качестве параметров один и тот же массив `trans` переходов, но специфический массив `out` выходов. Так, в приведенном здесь примере единственного виджета мы использовали выражение:

```
var auto=new automat(trans,out,"Начало")
```

  В случае нескольких виджетов потребуется в скрипте указать несколько подобных выражений, используя переменные, например, `auto1`, `auto2` и т. д.

- Упомянутая выше специфика элементов массива `out` выходов заключается в использовании специфических имен трех переменных для таймеров и объекта виджета (элемента `<div>`). В приведенном здесь примере единственного виджета используются переменные `t1`, `t2` и `t3` для таймеров и `element1=document.getElementById("widget1")` — для объекта, соответствующего виджету, т. е. элементу `<div id="widget1" class="widget" >`. В случае нескольких виджетов придется определить соответствующее количество массивов выходов (например, `out1`, `out2` и т. д.), в каждом из которых указать свои переменные, имеющие аналогичный смысл, что и `t1`, `t2`, `t3` и `element1`. Разумеется, где-то в начале скрипта следует объявить тройки переменных для таймеров и переменные вида `elementi = document.getElementById(`*значение атрибута id элемента виджета*`)` для объектов виджетов.

> ***Примечание***
>
> Приведенный выше код работает во всех современных браузерах, за исключением IE 8 и более ранних версий. Дело в том, что IE не поддерживает стандартный CSS-параметр `opacity`, задающий уровень прозрачности. Кроме того, имеются различия в объекте события `event`. Для IE вместо параметра `opacity` следует использовать фирменное расширение css от Microsoft — графический фильтр filters['alpha'].opacity, принимающий целочисленные значения от 0 до 100. Вместо `event` достаточно использовать window.event.

Работу рассмотренного виджета можно посмотреть на сайте по адресу **http://dunaevv1.narod.ru**.

ПРИЛОЖЕНИЕ 1

Перечень тегов HTML 5

Здесь приведен перечень тегов и их атрибутов в соответствии с черновой спецификацией W3C HTML 5. Нерекомендованные теги и атрибуты, соответствующие только HTML 4, не указаны.

Базовые атрибуты:

accesskey, class, contenteditable, contextmenu, dir, draggable, hidden, id, lang, spellcheck, style, tabindex, title.

Атрибуты-события:

onabort, onblur, oncanplay, oncanplaythrough, onchange, onclick, oncontextmenu, ondblclick, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, ondurationchange, onemptied, onended, onerror, onfocus, onformchange, onforminput, oninput, oninvalid, onkeydown, onkeypress, onkeyup, onload, onloadeddata, onloadedmetadata, onloadstart, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel, onpause, onplay, onplaying, onprogress, onratechange, onreadystatechange, onscroll, onseeked, onseeking, onselect, onshow, onstalled, onsubmit, onsuspend, ontimeupdate, onvolumechange, onwaiting.

Базовые атрибуты и атрибуты-события здесь называются общими.

Тег	Атрибуты	Версия	Назначение
<!-- -->	нет	4/5	Комментарий
<!DOCTYPE>	нет	4/5	Тип документа
<a>	Общие, href, hreflang, media, name, ping, rel, target, type	4/5	Гиперссылка
<abbr>	Общие	4/5	Аббревиатура
<address>	Общие	4/5	Адрес
<area>	Общие, alt, cords, href, hreflang, media, ping, rel, shape, target, type	4/5	Область графической карты ссылок
<article>	Общие	5	Статья, независимая часть документа

(продолжение)

Тег	Атрибуты	Версия	Назначение
<aside>	Общие	5	Содержимое, косвенно связанное с основным
<audio>	Общие, autoplay, controls, loop, preload, src	5	Звук
<b>	Общие	4/5	Выделение текста полужирным шрифтом
<base>	Общие, href, target	4/5	Базовый URL для всех ссылок на странице
<bdo>	Общие, dir	4/5	Направление отображения текста
<blockquote>	Общие, cite	4/5	Выделение длинных цитат
<body>	Общие, onafterprint, onbeforprint, onbeforunload, onhaschange, onmessage, onoffline, ononline, onpopstate, onredo,onresize, onstorage, onundo, onunload	4/5	Основная часть документа
 	Общие	4/5	Переход на новую строку
<button>	Общие, autofocus, disabled, name, type, value	4/5	Кнопка
<canvas>	Общие, height, width	5	Область для отображения динамической графики
<caption>	Общие	4/5	Заголовок таблицы
<cite>	Общие	4/5	Цитата
<code>	Общие	4/5	Программный код
<col>	Общие, span	4/5	Параметры табличных столбцов
<colgroup>	Общие, span	4/5	Группа табличных столбцов
<command>	Общие, checked, default, disabled, hidden, icon, label, radiogroup, type	5	Командная кнопка
<datalist>	Общие	5	Включает набор элементов, которые представляют предопределенные опции для других элементов управления
<dd>	Общие	4/5	Содержание определения
<del>	Общие, cite, datetime	4/5	Удаленный текст
<details>	Общие, open	5	Дополнительные сведения об элементе, предоставляемые по требованию
	Общие	4/5	Раздел документа. Универсальный блочный контейнерный элемент

(продолжение)

Тег	Атрибуты	Версия	Назначение
<dfn>	Общие	4/5	Определяемый термин (первое упоминание в документе)
<dl>	Общие	4/5	Список определений
<dt>	Общие	4/5	Определяемый термин
	Общие	4/5	Выделение текста
<embed>	Общие, height, src, type, width	5	Внедрение внешнего содержимого
<fieldset>	Общие, name, disabled, form	4/5	Группа элементов формы
<figure>	Общие	5	Группа разнородного содержимого
<footer>	Общие	5	Подвал секции или страницы
<form>	Общие, action, accept-charset, enctype, method, name, novalidate, target	4/5	Форма
<h1>,…,<h6>	Общие	4/5	Заголовки
<head>	Общие	4/5	Информация о документе
<hgroup>	Общие	5	Группа заголовков
<header>	Общие	5	Заголовок секции или страницы
<hr>	Общие	4/5	Горизонтальная полоса
<html>	Общие, manifest	4/5	HTML-документ
<i>	Общие	4/5	Выделение текста курсивом
<iframe>	Общие, height, name, sandbox, seamless, src, width	4/5	Внутренний (плавающий) фрейм
<img>	Общие, alt, border, height, ismap, src, usemap, width	4/5	Графическое изображение
<input>	Общие, accept, alt, autocomplete, autofocus, checked, disabled,inputmode, list, max, min, maxlength, name, pattern, readonly, replace, required, src, step, template, type, value	4/5	Поле ввода данных
<ins>	Общие, cite, datetime	4/5	Вставленный текст
<kbd>	Общие	4/5	Текст, набираемый с помощью клавиатуры
<keygen>	Общие	5	Элемент управления для генерации и отправки ключей
<label>	Общие, for, form	4/5	Метка, связанная с элементом формы
<legend>	Общие	4/5	Заголовок группы элементов формы

(продолжение)

Тег	Атрибуты	Версия	Назначение
	Общие, value	4/5	Элемент списка
<link>	Общие, href, hreflang, media, rel, sizes, type	4/5	Устанавливает связь с внешним ресурсом
<mark>	Общие	5	Помеченный текст
<map>	Общие, name	4/5	Карта ссылок
<menu>	Общие, label, type	5	Меню
<meta>	Общие, charset, content, http-equiv, name	4/5	Служебная информация
<meter>	Общие, high, low, max, min, optimum, value	5	Указание шкалы измерения
<nav>	Общие	5	Группа навигационных ссылок
<noscript>	Общие	4/5	Содержимое для браузеров, не поддерживающих скрипты
<object>	Общие, data, form, height, name, type, usemap, width	4/5	Внедрение внешнего содержимого
	Общие, start, reversed	4/5	Упорядоченный список
<optgroup>	Общие, disabled, label	4/5	Группа опций
<option>	Общие, disabled, label, selected, value	4/5	Опция
<output>	Общие, for, form, name	5	Результат вычислений в форме
<p>	Общие	4/5	Параграф
<param>	Общие, name, value	4/5	Параметры внедренных объектов
<pre>	Общие	4/5	Предварительно отформатированный текст
<progress>	Общие, max, value	5	Индикатор прогресса выполнения задачи
<q>	Общие, cite	4/5	Выделение коротких цитат
<rp>	Общие	5	Пояснение, что Ruby-текст не может быть отображен
<rt>	Общие	5	Компонент Ruby-текста
<ruby>	Общие	5	Описание Ruby-аннотаций, используемых в странах восточной Азии
<samp>	Общие	4/5	Пример программного кода
<script>	Общие, async, defer, src, type	4/5	Скрипт (сценарий)
<section>	Общие	5	Секция документа
<select>	Общие, disabled, form, multiple, name	4/5	Список, из которого можно выбрать элементы

(окончание)

Тег	Атрибуты	Версия	Назначение
<small>	Общие	4/5	Сопутствующий текст (оговорки)
<source>	Общие, media, src, type	5	Описание источников информации для элементов <audio> и <video>
	Общие	4/5	Универсальный внутристрочный контейнерный элемент
	Общие	4/5	Выделение особо важного текста
<style>	Общие, media, type, scoped	4/5	Описание правил CSS
<sub>	Общие	4/5	Подстрочный текст
<summary>	Общие	5	Резюме, заголовок или легенда для элемента <details>
<sup>	Общие	4/5	Надстрочный текст
<table>	Общие, summary	4/5	Таблица
<tbody>	Общие	4/5	Группа строк таблицы
<td>	Общие, colspan, rowspan, headers	4/5	Ячейка таблицы
<textarea>	Общие, autofocus, cols, disabled, form, name, placeholder, readonly, required, rows, wrap	4/5	Текстовая область ввода данных
<tfoot>	Общие	4/5	Группа заключительных строк таблицы
<th>	Общие, colspan, headers, rowspan, scope	4/5	Ячейка для заголовка столбца таблицы
<thead>	Общие	4/5	Группа заголовков таблицы
<time>	Общие, datetime, pubdate	5	Дата и/или время
<title>	Общие	4/5	Название документа
<tr>	Общие	4/5	Строка таблицы
	Общие	4/5	Неупорядоченный список
<var>	Общие	4/5	Представление переменных математических формул или программ
<video>	Общие, autoplay, controls, height, loop, poster, preload, src, width	5	Видео

ПРИЛОЖЕНИЕ 2

Перечень параметров CSS

Здесь представлен перечень основных параметров CSS 2. Полный перечень параметров CSS 3 значительно больше и находится в разработке. Поэтому отметим из него лишь те параметры, которые рассматриваются в данной книге и поддерживаются, по крайней мере, несколькими ведущими браузерами:

- границы и тень — `border-image`, `border-radius`, `box-shadow`;
- мультиколоночное позиционирование — `columns`, `column-gap`, `column-rule`;
- прозрачность — `opacity`.

> ***Примечание***
>
> Свойства, допускающие сокращенную запись нескольких параметров как одно значение, выделены полужирным шрифтом. Наследуемые свойства помечены символом +.

Позиционирование

`display`	`clear`
`position`	`z-index`
`top`	`direction +`
`right`	`unicode-bidi`
`bottom`	`overflow`
`left`	`clip`
`float`	`visibility`

Размеры

`width`	`min-height`
`min-width`	`max-height`
`max-width`	`vertical-align`
`height`	

Цвет и фон

color +	background-repeat
background	background-image
background-color	background-position
background-attachment	

Текст

text-indent +	word-spacing +
text-align +	text-transform +
text-decoration	white-space +
text-shadow	line-height +
letter-spacing +	

Шрифты

font +	font-weight +
font-family +	font-stretch +
font-style +	font-size +
font-variant +	font-size-adjust +

Блоки (поля, отступы и границы)

margin	**border-color**
margin-top	border-top-color
margin-right	border-right-color
margin-bottom	border-bottom-color
margin-left	border-left-color
padding	**border-style**
padding-top	border-top-style
padding-right	border-right-style
padding-bottom	border-bottom-style
padding-left	border-left-style
border	**border-width**
border-top	border-top-width
border-bottom	border-right-width
border-right	border-bottom-width
border-left	border-left-width

Таблицы

caption-side +	border-spacing +
table-layout	empty-cells +
border-collapse +	speak-header +

Печать

size	page-break-inside +
marks	page +
page-break-before	orphans +
page-break-after	widows +

Интерфейс

cursor +	outline-style
outline	outline-color
outline-width	

Звук

volume +	elevation
speak +	speech-rate
pause	voice-family
pause-before	pitch
pause-after	pitch-range
cue	stress
cue-before	richness
cue-after	speak-punctuation
play-during	speak-numeral
azimuth	

Прочее

content	list-style-type +
quotes +	list-style-image +
counter-reset	list-style-position +
counter-increment	marker-offset
list-style +	

Предметный указатель

@

@import 59
@media 60, 65

A

ActiveX 261
AJAX 381, 391

C

Cascading Style Sheets (CSS) 56
Component Object Model (COM) 261
Cookie 363
CSS 36
CSV-файл 263, 350, 356, 386

D

Distributed Component Object Model (DCOM) 261
DTD 41

E

ECMAScript 277

F

Flash-фильмы 257

H

Handler 302
HTML 5.0 429

J

jQuery 391

L

LiveScript 275

M

MIME-тип 253

O

Object Linking and Embedding (OLE) 261
OLE Custom Controls (OCX) 261

S

SVG 242, 429

T

TDC 264, 350

U

URL 36, 133, 148

V

VML 242

W

Web-палитра 239

X

XML 37

А

Альфа-канал 111
Атрибут 36, 53
◊ accesskey 222, 228
◊ action 229
◊ align 207, 228
◊ autobuffer 254
◊ autoplay 254
◊ border 196
◊ cellspacing 199
◊ class 54, 61, 62
◊ code 270
◊ colspan 208
◊ controls 254
◊ coords 145
◊ datafld 265
◊ dataformatas 265
◊ datasrc 265
◊ disabled 222
◊ enctype 229, 230
◊ for 228
◊ frame 196
◊ href 50, 133
◊ id 54, 61
◊ label 224
◊ lang 44
◊ language 278
◊ media 60
◊ method 229
◊ name 147, 247
◊ nohref 145
◊ poster 254
◊ preload 254
◊ rowspan 208
◊ rules 196
◊ shape 145
◊ span 207
◊ style 54, 60
◊ target 133, 229
◊ title 54
◊ type 217
◊ usemap 145
◊ valign 207
◊ атрибут-событие 54
◊ базовый 54
◊ языковой 55
Атрибут-событие 303

Б

Баннер 360
Безье 431, 435

В

Выражение 284

Г

Гиперссылки *См.* Ссылка
Графическая карта ссылок 144

Д

Дескриптор
◊ <!-- ... //--> 280
◊ <!-- ...--> 45
◊ <!--[if IE]> ... <![endif]--> 46
◊ <!DOCTYPE…> 41

З

Закладка 146
Зарезервированное слово 284
Знак операции 284

И

Идентификатор 283
Индексы 163
Исключительная ситуация 382

К

Календарь 453
Капитель 154
Каскадные таблицы стилей 56
Ключевое слово 284
Кнопка 219, 396
Коллекция 287
◊ all 291
◊ anchors 334
◊ applets 334
◊ cells 346
◊ cssRules 297
◊ elements 292
◊ embeds 334
◊ forms 287, 334, 358

◊ frames 287, 334, 343
◊ images 334, 369
◊ links 287, 334
◊ mimeTypes 333
◊ options 403
◊ plugins 333
◊ rows 346
◊ rules 297
◊ styleSheets 297, 334
Комментарий
◊ в (X)HTML 45
◊ в CSS 64
Компоновка страницы 117
Контекст вывода 430
Красная строка 159
Куки *См.* Cookie
Курсор 398

Л

Лексема 284
Литерал 284

М

Меню 134, 184, 399
◊ двухуровневое 139
◊ моментально раскрывающееся 399
◊ на основе списка 184
◊ плавно раскрывающееся 401
◊ простое 134
Метод
◊ abort() 381
◊ addColorStop() 437
◊ addEventListener() 307
◊ addRule() 299
◊ alert() 328
◊ arc() 434
◊ assign() 330
◊ back() 331
◊ beginPath() 433
◊ bezierCurveTo() 435
◊ blur() 328
◊ clearInterval() 329, 391
◊ clearRect() 432
◊ clearTimeout() 329, 391
◊ close() 328, 337
◊ closePath() 433
◊ confirm() 328
◊ createLinearGradient() 437
◊ createPopup() 345
◊ createRadialGradient() 437
◊ deleteRow() 349
◊ deleteRule() 300
◊ drawImage() 440, 441
◊ execScript() 329
◊ fill() 433
◊ fillRect() 432
◊ fillText() 448
◊ focus() 313, 328, 361
◊ forward() 331
◊ get 229
◊ getAllResponseHeaders() 381
◊ getAttribute() 294
◊ getContext('2d') 430
◊ getElementById() 290, 334
◊ getElementsByName() 291, 335
◊ getElementsByTagName() 290, 335
◊ getResponseHeader() 381
◊ go() 331
◊ hasAttribute() 294
◊ insertCell() 348
◊ insertRow() 348
◊ insertRule() 299
◊ javaEnabled() 333
◊ lineTo() 434
◊ navigate() 328
◊ open() 328, 335, 381
◊ post 229
◊ preventDefault() 311
◊ print() 329
◊ quadraticCurveTo() 435
◊ reload() 330
◊ removeAttribute() 294
◊ removeEventListener() 308
◊ removeRule() 300
◊ replace () 408
◊ replace() 330
◊ resizeTo() 328
◊ restore() 438
◊ rotate() 438
◊ save() 438
◊ scale() 438
◊ scrollBy() 328
◊ scrollTo() 328
◊ search() 360
◊ send() 382
◊ setAttribute() 294
◊ setInterval() 329, 391

Метод *(прод.)*
◊ setRequestHeader() 382
◊ setTimeout() 329, 391
◊ setTransform() 438
◊ showHelp() 329
◊ showModalDialog() 329
◊ showModalessDialog() 329
◊ stopPropagation() 317
◊ stroke() 433
◊ strokeRect() 432
◊ strokeText() 448
◊ submit() 362
◊ taintEnabled() 333
◊ transform() 438
◊ translate() 438
◊ write() 335
◊ writeln() 335
◊ запроса 229
◊ open() 336
Модуль расширения *См.* Плагин

Н

Насыщенность 237
Нормальный поток 83

О

Область
◊ активная 144
◊ горячая 144
Объект
◊ date 450
◊ document 333
◊ event 317
◊ history 331
◊ image 370, 373, 440
◊ localStorage 368
◊ location 329
◊ navigator 331
◊ popup 345
◊ recordset 350
◊ RegExp 360, 408
◊ screen 329
◊ window 326, 336
◊ XMLHttpRequest 356, 381
◊ целевой 303
Объектная модель 286
◊ DOM 288
◊ базовая 303
◊ браузера 286
◊ документа 287
Объемный заголовок 164
Окно
◊ PopUp *См.* всплывающее
◊ всплывающее 345
Оператор 284
Отключение обработчика 304, 307

П

Панель
◊ раскрывающаяся 105
◊ с тенью 114
Пароль 218
Парсер 38
Переключатель 216
Переполнение контейнера 97
Пиксел 232
Поле ввода данных 216
Преобразование регистра 166
Привязка события и обработчика 303
Приоритет параметров CSS 65
Прозрачность 107, 111
Прямой поток *См.* Нормальный поток
Псевдо-URL 151
Псевдокласс 134

Р

Радиокнопка 216
Разрешение 233
Расстояние
◊ между буквами 161
◊ между словами 160
◊ между строками 159
Расширение ячеек 208
Регистрация обработчика 304
Регистровая зависимость 283
Регулярное выражение 359, 408
Режим индексированных цветов 239

С

Свойство 319, 333
◊ appCodeName 331
◊ appName 331
◊ appVersion 332
◊ availHeight 329
◊ availWidth 329
◊ bgColor 333

◊ body 333
◊ cancelBubble 316
◊ cellIndex 349
◊ className 295
◊ closed 327, 337
◊ colorDepth 329
◊ complete 370, 379
◊ cookie 333, 363
◊ cookieEnabled 332, 367
◊ defaultStatus 327
◊ dir 333
◊ doctype 333
◊ document 327
◊ documentElement 333
◊ domain 333
◊ event 327
◊ fgColor 333
◊ fillStyle 432
◊ globalCompositeOperation 446
◊ hash 330
◊ history 327
◊ host 330
◊ hostname 330
◊ href 330
◊ implementation 333
◊ innerHTML 300, 345, 347, 356, 383
◊ innerText 301
◊ lastModified 334
◊ lineWidth 432
◊ linkColor 334
◊ location 327
◊ longDesc 370
◊ name 327
◊ navigator 327
◊ onreadystatechange 381
◊ opener 327
◊ outerHTML 301, 353
◊ outerText 301
◊ parent 327
◊ pathname 330
◊ platform 332
◊ port 330
◊ prompt() 328
◊ readyState 334, 381
◊ referrer 334
◊ responseText 381
◊ responseXML 381
◊ returnValue 310, 327
◊ screen 327
◊ search 330
◊ selectedIndex 403
◊ selection 334
◊ self 327
◊ srcElement 319, 322
◊ status 327, 381
◊ statusText 381
◊ strokeStyle 432
◊ style 296
◊ tagName 294
◊ target 319, 322
◊ title 334
◊ top 327
◊ type 320
◊ userAgent 332
◊ vlinkColor 334
Сегментированная графика
См. Графическая карта ссылок
Селектор 60
Слой 93
Специальные символы 158
Специфичность селектора 66
Список
◊ иерархический 224
◊ раскрывающийся 222, 403
Ссылка
◊ внутренняя 133, 146
◊ графическая 143
◊ относительная 50
◊ текстовая 134
Ссылка (гиперссылка)
◊ указатель 133
Ссылки 133
Стилевые параметры 56
◊ bacground-position 108
◊ background 108
◊ background-attachment 108
◊ background-color 107
◊ background-image 107
◊ background-position 111
◊ background-repeat 108
◊ border 70
◊ border-collapse 198
◊ border-image 75
◊ border-radius 77
◊ border-spacing 199
◊ border-style 73
◊ border-top-style, border-right-style, border-left-style, border-bottom-style 73
◊ bottom 86

Свойство *(прод.)*
◊ box-shadow 115
◊ caption-side 194
◊ clear 94
◊ clip 98
◊ column-gap 166
◊ column-rule 166
◊ columns 166
◊ content 168, 176
◊ counter-increment 169, 176
◊ counter-reset 169, 176
◊ cursor 139
◊ display 99
◊ filter 111
◊ float 94
◊ font 155
◊ font-family 153
◊ font-size 155
◊ font-style 154
◊ font-variant 154
◊ font-weight 155
◊ height 78, 100
◊ left 86
◊ letter-spacing 161
◊ line-height 159
◊ list-style-image 173
◊ list-style-position 188
◊ list-style-type 174
◊ margin 70
◊ max-height 100
◊ max-width 100, 101
◊ min-height 100, 101
◊ min-width 100, 101
◊ opacity 111
◊ overflow 97
◊ padding 70
◊ position 87
◊ right 86
◊ text-align 159, 204
◊ text-decoration 135, 161
◊ text-indent 159
◊ text-layout 199
◊ text-shadow 165
◊ text-transform 166
◊ top 86
◊ vertical-align 159, 163, 204
◊ visibility 99
◊ white-space 167
◊ width 78, 100
◊ word-spacing 160
◊ z-index 93
Счетчик посещений 376

T

Таблица 193, 346
◊ перемещение по строкам 350
◊ прокручиваемая 212
◊ сложная 208
◊ сортировка строк 353
◊ фильтрация данных 355
Тег 33
◊ <a> 133
◊ <address> 156
◊ <applet> 270
◊ <area> 144
◊ <audio> 253
◊ <b> 157
◊ <base> 50
◊ <blockquote> 157
◊ <body> 37, 45
◊ <button> 219
◊ <canvas> 429
◊ <caption> 193
◊ 102
◊ <cite> 157
◊ <code> 157
◊ <col/> 194, 206
◊ <colgroup> 194, 207
◊ <dd> 191
◊ 34, 83
◊ <dl> 172, 191
◊ <dt> 191
◊ 156
◊ <embed> 252
◊ <fieldset> 227
◊ <form> 217, 229
◊ <head> 45, 46
◊ <hr/> 418
◊ <i> 157
◊ <iframe> 245, 340
◊ <img> 145, 243
◊ <input> 216
◊ <kbd> 157
◊ <label> 227
◊ <legend> 227
◊ 172, 404
◊ <link> 51, 59
◊ <map> 144

◊ <meta> 47
◊ <noscript> 280
◊ <object> 245
◊ 172
◊ <optgroup> 224
◊ <option> 223, 224, 403
◊ <p> 158
◊ <pre> 166, 407
◊ <samp> 157
◊ <script> 278
◊ <select> 222, 403
◊ <source> 254
◊ 83
◊ <style> 59
◊ <sub> 163
◊ <sup> 163
◊ <table> 193
◊ <tbody> 193
◊ <td> 193
◊ <textarea> 225
◊ <tfoot> 193
◊ <th> 193
◊ <thead> 193
◊ <title> 45, 46
◊ <tr> 193
◊ 172, 404
◊ <var> 157
◊ <video> 253
◊ блочный 83
◊ внутристрочный (in-line) 83
Текстовая область 225
Тень 113, 116, 165
Тип
◊ документа (DTD) 41
◊ события 320

У

Установка фокуса ввода 313

Ф

Флажок 216
Форма 229, 358
Формат
◊ 3GP 251
◊ AIFF 251
◊ AVI 251
◊ FLV 252, 255
◊ GIF 240
◊ JPEG 240
◊ MIDI 251
◊ MOV 251
◊ MP3 251
◊ MPEG 251
◊ OGG/ Vorbis 251
◊ PNG 240
◊ SWF 242, 252, 257
◊ WAV 251
◊ WMA 251
◊ WMV 251
Фрейм плавающий 245
Функция
◊ decodeURIComponent() 367
◊ deleteCookie() 367
◊ encodeURIComponent() 367
◊ escape() 364
◊ eval() 306, 386, 415
◊ readCookie() 365
◊ unescape() 365
◊ writeCookie() 366
◊ анонимное определение 305

Х

Холст 429

Ц

Цветовая глубина 238
Цветовая модель
◊ CMYK 235
◊ HLS 237
◊ HSB 237
◊ Lab 238
◊ RGB 235
Центрирование 102

Э

Эквиваленты символов 158
Элемент управления ActiveX 261
◊ Adobe SVGViewer 263
◊ Flash-проигрыватель 262
◊ Tabular Data Control 264, 350
◊ календарь 263

Я

Якорь 147
Яркость 237