АНДРЕЙ ОВЧАРЕНКО

Ajax

НА ПРИМЕРАХ

ОСНОВЫ JavaScript, HTML, CSS И DOM

ОБМЕН ДАННЫМИ С СЕРВЕРОМ В ФОРМАТЕ XML И JSON

РАБОТА С APACHE, PHP И MySQL

ЗАЩИТА ПРИЛОЖЕНИЙ, АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

СООТВЕТСТВИЕ СТАНДАРТАМ И КРОССБРАУЗЕРНОСТЬ

+CD

bhv®

Андрей Овчаренко

Ajax
НА ПРИМЕРАХ

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.068
ББК 32.973.26-018.1
О-35

Овчаренко А. В.

О-35 Ajax на примерах. — СПб.: БХВ-Петербург, 2009. — 432 с.: ил. + CD-ROM

ISBN 978-5-9775-0299-3

На практических примерах рассмотрены эффективные приемы разработки динамических Web-приложений, построенных по технологии Ajax. Каждая глава посвящена разработке законченного компонента пользовательского интерфейса Web-приложения. Даны необходимые для быстрого старта сведения по HTML и CSS, XML и DOM Level 1, PHP и MySQL, а также примеры совместного их применения. Большое внимание уделено программированию на языке JavaScript и асинхронному обмену данными между клиентом и сервером при помощи объекта XMLHttpRequest в формате XML и JSON. Подробно описан процесс разработки компонентов пользовательского интерфейса: "Аккордеон", панель с закладками, слайд-шоу, выпадающее меню, плавающие окна и др. Рассмотрены вопросы доступа к базам данных MySQL, управления учетными записями пользователей, защиты данных, аутентификации и авторизации, кроссбраузерности разрабатываемых приложений и др. На прилагаемом к книге компакт-диске содержится полный программный код компонентов и приложений, рассмотренных в книге.

Для Web-программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 34,83.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

ISBN 978-5-9775-0299-3

Оглавление

Введение

Уважаемые читатели. Вы держите в руках книгу, которая полностью посвящена разработке Web-приложений с использованием технологии *Ajax*. Возможно, вы уже сталкивались с этой технологией или даже являетесь гуру в области Ajax. Возможно, вы никогда даже не слышали о существовании технологии Ajax. Автор сделал все, чтобы предлагаемая вашему вниманию книга была интересной, а главное — полезной. Автор книги — практический программист, который уже не один год использует технологию Ajax в своей ежедневной деятельности. Поэтому все примеры программного кода книги обкатаны в реальных приложениях и совмещают простоту, надежность и практическую направленность.

Для тех, кто уже работал с технологией Ajax, вводная часть может быть на этом закончена, и можно переходить к непосредственно интересующей вас главе книги. Главы книги относительно независимы, хотя ключевой является *глава 2*, и ее автор рекомендует прочитать вне зависимости от уровня вашей подготовки. Такое пожелание продиктовано тем, что в последующих главах используются утилитные функции, которые разработаны в *главе 2*. Для тех же, кто впервые сталкивается с технологией Ajax, будет удобнее изучать главы в последовательном порядке, не пропуская, между прочим, и введения. Итак, мы — начинаем.

Ajax — это аббревиатура, которая расшифровывается как *асинхронный JavaScript и XML*. Второе (вернее, истинное) значение этого слова — имя древнегреческого героя, о чем уже не раз писали авторы литературы по Ajax. Но постараюсь и тут дать вам немного новой информации. На самом деле Аяксов было двое — *Большой Аякс* и *Малый Аякс*. Согласно мифу, они были друзьями. Поэтому лет, этак, 100 назад, когда гимназисты получали классическое образование, выражение "два Аякса" было у всех на слуху и означало примерно то же, что сейчас означают выражения неразлейвода, Фома и Ерема, Лель и Полель, два сапога пара. В этом (в том, что Аяксов было двое) можно усмотреть очень глубокий смысл, который и не снился авторам самой

технологии и термина Ajax. До появления технологии Ajax Web-приложение выполнялось преимущественно на стороне Web-сервера. Web-браузер в таком приложении играл роль пассивного монитора, который отображал полученный с Web-сервера HTML-документ. В противоположность классическому Web-приложению, Ajax-приложение выполняется как на Web-сервере, так и на Web-клиенте, т. е. в Web-браузере. Говоря образно, Web-сервер можно ассоциировать с Большим Аяксом, а Web-браузер — с Малым Аяксом.

Надо сказать, что наименование (аббревиатура) Ajax носит не столько смысловой, сколько маркетинговый характер. Когда я впервые увидел книгу, в названии которой присутствовало слово Ajax, то если меня и привлекло это название, то благодаря наличию букв **j** и **x**. Первая из них ассоциировалась с передовыми интернет-технологиями, а вторая — со всем хорошим, широким (и, в то же время, немного таинственным), что только может придумать человек. Наличие в названии сразу двух букв **a** (заметьте, как и Аяксы, одна большая и одна малая) только усиливало положительный эффект, показывая, что технология активная и находится на первом месте среди равных. Но только когда я пролистал (благо была возможность) полкниги, я, наконец, понял, что технология Ajax — то самое, что мне необходимо прямо сейчас и чего мне все это время не хватало при разработке Web-приложений.

На самом деле Ajax-технология — это не столько новая технология, сколько новый подход к разработке Web-приложений, использующий возможности, которые присутствуют в Web-браузере с конца 90-х годов прошлого века. Средства Web-браузера, которые используются в Ajax-технологии, принято называть инфраструктурой Ajax. Инфраструктуру Ajax составляют:

- документы HTML;
- объектная модель документа DOM (Document Object Model);
- каскадные таблицы стилей CSS (Cascade Stylesheets);
- язык программирования JavaScript;
- объект `XMLHttpRequest` Web-браузера.

Первые четыре составляющие инфраструктуры Ajax широко известны как *DHTML* (динамический HTML). Несмотря на широкие возможности технологии DHTML и публикацию близких к этой технологии спецификаций *DOM Level 1* и *CSS*, технология DHTML некоторое время оставалась в тени и не имела той популярности среди разработчиков Web-приложений, которая была бы достойна ее богатым возможностям и которую она начинает приобретать в настоящее время. Для этого есть, по крайней мере, две причины. Во-первых, достаточно долго Web-браузеры обеспечивали весьма слабую совместимость, позволяя кросс-браузерно отображать только самые простые

HTML-документы. Код более сложных HTML-документов и Web-приложений, которые работали с объектной моделью документа DOM при помощи JavaScript, разрабатывался или для конкретной версии Web-браузера, или использовал различные приемы, похожие на трюки, для достижения кросс-браузерной работы приложения.

В настоящее время, опубликованы спецификации *HTML 4.01*, *DOM Level 1*, *CSS2.1*, которые поддерживаются всеми основными производителями Web-браузеров. Со времени публикации этих спецификаций в конце 90-х годов прошлого столетия должно было пройти еще время, чтобы разработчики Web-браузеров обеспечили поддержку этих спецификаций, и большинство пользователей Web-приложений установили у себя на компьютерах новые версии Web-браузеров. На момент написания книги можно быть практически уверенным в том, что большинство пользователей имеют на своих компьютерах Web-браузеры, работающие с технологией DHTML и Ajax. Хотя для разработки кросс-браузерных приложений по-прежнему необходимо прикладывать дополнительные усилия, можно сказать, что прогресс в обеспечении кросс-браузерности достигнут колоссальный. И это создало предпосылки для широкого использования технологии DHTML разработчиками Web-приложений.

> ***Замечание***
>
> В настоящее время опубликованы и более новые спецификации HTML, DOM и CSS. Есть и проект новой спецификации JavaScript (ECMAScript), который, по мнению авторов, должен стать языком строго типизированным, поддерживать пакеты, классы и пространства имен.

Но была и вторая (после несовместимости Web-браузеров) причина, которая мешала широкому использованию технологии DHTML. Дело в том, что технология DHTML не предусматривала средств для обмена данными между Web-браузером и Web-сервером. Как уже стало ясно сейчас, с высоты достижений технологии Ajax, только совместное использование DHTML и фонового (без перезагрузки основного HTML-документа) обмена данными между Web-браузером и Web-сервером позволяют создавать по-настоящему динамические Web-приложения. Может показаться почти фантастическим, что вместе с появлением технологии DHTML в тот же период, был разработан объект `XMLHttpRequest` (пятый элемент в инфраструктуре Ajax), который также стал доступен для разработчиков. При помощи объекта `XMLHttpRequest` как раз и можно организовать в DHTML-приложении фоновый обмен данными между Web-браузером и Web-сервером. Но все составляющие инфраструктуры Ajax какое-то время не могли встретиться в рамках одного приложения и существовали сами по себе. В это время довольно часто для фонового обмена данными Web-браузера с Web-сервером использовались HTML-элементы

IFRAME или SCRIPT. Но по-настоящему удобным обмен данными между Web-браузером и Web-сервером стал с использованием асинхронных запросов при помощи объекта XMLHttpRequest.

Давайте разберемся, почему возникла технология Ajax и почему она возникла именно сейчас? Сравнительно длительный период (для нашего стремительного времени) развитие Web-приложений шло в направлении усложнения серверного программирования. Сначала Web-сервер просто доставлял HTML-документ в Web-браузер пользователя. Потом появилась возможность отсылать с Web-браузера на Web-сервер простые данные при помощи технологии CGI. Вскоре оказалось, что CGI-приложение может не только принимать и обрабатывать данные, отправленные Web-браузером, но и возвращать в Web-браузер динамически сгенерированный HTML-документ. Затем появилась возможность управлять сессиями, создавать серверные объекты. За этим великолепием серверных технологий в тени остался пользователь, который работал с Web-браузером. Независимо от того, какая технология работала на Web-сервере, пользователю доставлялась все та же HTML-страница. И работа пользователя с Web-браузером практически не изменялась при очередном переходе на все более и более прогрессивную серверную технологию. И это выглядело довольно неестественно, потому что Web-браузер уже в конце 90-х годов прошлого века позволял создавать Web-приложения, которые активно моги бы использовать средства программирования на стороне Web-браузера. Налицо было противоречие между все усложняющимися серверными технологиями и их результатом, который, с точки зрения пользователя, раскрывшего Web-браузер, застыл на уровне середины 90-х годов.

Программирование на стороне Web-браузера не только не развивалось, но даже считалось недопустимым при разработке серьезного Web-приложения. Отчасти, такое мнение имеет под собой почву. Выполнение ответственных бизнес-операций с использованием программирования на стороне клиента Web-браузера было бы достаточно рискованным решением. Это связано с тем, что Web-сервер не может контролировать среду выполнения клиентского приложения, как это могут делать некоторые серверы приложений, и этим может воспользоваться злоумышленник. Выполнение некоторых бизнес-операций, таких как вычисление стоимости заказа или оплата счета в интернет-магазине, допустимо только на сервере. Некоторые разработчики даже рекомендуют это делать не на Web-сервере, а использовать еще дополнительный уровень — сервер приложений. Но управлять поведением элементов пользовательского интерфейса при помощи полной перерисовки HTML-документа, полученного новым запросом с Web-сервера, как это делалось в классических Web-приложениях, — слишком тяжеловесное решение и с точки зрения Web-разработчика, и с точки зрения пользователя.

Как бы то ни было, инертность мышления Web-разработчиков некоторое время сдерживала развитие технологии DHTML и Ajax, но это не могло продолжаться вечно. Слово Ajax было произнесено, и приложения, построенные с использованием этой технологии, позволили сделать то, что еще совсем недавно никто не мог ожидать от Web-приложений.

Я уже говорил, что технология Ajax использует для реализации обмена данными с Web-сервером объект `XMLHttpRequest`. Что же представляет собой этот объект? Его можно назвать браузером в браузере или невидимым браузером. Этот объект может в фоновом режиме (без перезагрузки основного HTML-документа) отправить запрос на Web-сервер и получить ответ в виде текста или XML-документа. Полученный с Web-сервера текст или XML-документ не отображается в Web-браузере непосредственно, как при обычном запросе Web-браузера к Web-серверу. Ответ Web-сервера доступен только из языка программирования JavaScript (или другого скриптового языка, поддерживаемого Web-браузером), и это открывает новые возможности для Web-программирования. Если вам сейчас это не совсем понятно — не огорчайтесь. Ведь именно изучением того, как это сделать на практике, мы и будем с вами заниматься на протяжении всей книги.

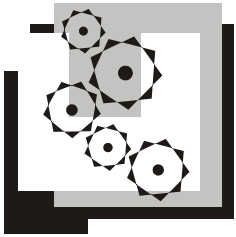
Возможности объекта `XMLHttpRequest` гораздо шире, чем это можно было бы предположить из названия объекта. Как и название Ajax, название объекта `XMLHttpRequest` может поначалу увести от понимания его прямого назначения. Разберем этот момент более подробно:

- XML — на самом деле, объект `XMLHttpRequest` может работать как с XML-документами, так и с текстом, например `text/html` или `text/plain`;
- Http — объект `XMLHttpRequest` может работать как с протоколом HTTP, так и с другими протоколами, которые поддерживает Web-браузер, например с протоколами HTTPS, file;
- Request — объект `XMLHttpRequest` может отправлять (методом `send()`) запрос (request) к Web-серверу. Кроме того, объект может получать ответ (response) Web-сервера (свойства `responseText` и `responseXML`).

Я подчеркиваю этот момент, потому что, исходя из названия объекта, у вас может сложиться впечатление, что технология Ajax предназначена исключительно для рафинированных теоретиков, имеющих дело с генерацией и разбором XML-документов. Нет, Ajax-приложение имеет дело с динамической загрузкой контента с Web-сервера в самых разнообразных, удобных и нужных вам вариантах. Это может быть и загрузка фрагментов HTML-документа, что с успехом используется в тех случаях, когда традиционно применялись фреймы. Это может быть загрузка программного кода JavaScript и, наконец, работа с XML-документами.

Теперь рассмотрим вопрос, чем Ajax-приложение отличается от классического Web-приложения. Ajax — это аббревиатура от *асинхронный JavaScript и XML*. Как это иногда бывает, Ajax — это еще и яркое запоминающееся название, которое служит продвижению этой замечательной технологии. Как бы то ни было, не стоит искать ответы на все вопросы только в названии. Если постараться сформулировать то, что выделяет Ajax-приложения в мире Интернета — так это будет новый удобный или, как принято говорить, *богатый* пользовательский интерфейс. В свою очередь, появление нового *богатого интерфейса* позволило расширить круг решаемых задач и расширить сферу применения Web-приложений. Ajax-приложение — это приложение, которое удобно для пользователя и обеспечивает функциональность, ставшую уже привычной для настольного (desktop) приложения. Несмотря на то, что технология Ajax связана с применением конкретных средств, которые мы называем инфраструктурой Ajax, основное в этой технологии — не используемые средства, а расширение границ применения Web-приложений, новое ощущение комфорта, который получает пользователь при работе с Web-приложением.

Я рад, что издательство "БХВ-Петербург" решило внести вклад в развитие технологии Ajax и поручило мне написать книгу, которую имею честь предложить вашему вниманию. В процессе работы над книгой я получал всяческую поддержку от коллектива издательства, за что хочу выразить свою благодарность. Той скорости решения вопросов, которые неизбежно возникали у меня в процессе подготовки книги, могли бы позавидовать даже железнодорожники. В связи с этим я особо хотел бы поблагодарить сотрудников издательства "БХВ-Петербург" Е. Е. Рыбакова, Г. Л. Добина и Александру, которые, несмотря на свою сверхзагруженность, всегда находили время, чтобы оперативно ответить на мои вопросы. Когда вы, уважаемые читатели, будете читать эту книгу, вспомните со словами благодарности о замечательном редакторе издательства "БХВ-Петербург" Анне Сергеевне Кузьминой. Просто текст отличается от хорошего текста тем что слова появляются вовремя и кстати, и еще чем-то неуловимым, о чем знают только мастера, такие как Анна Сергеевна. Спасибо Вам, уважаемая Анна Сергеевна! А также хочу поблагодарить своего друга и наставника Б. С. Свитского за предоставления широкого пространства для приложения Ajax-технологии и, особенно, за удачный перевод на русский язык моей любимой английской пословицы о том, что *не бывает обстоятельств столь благоприятных, которыми глупый человек не воспользовался бы себе во вред, и не бывает обстоятельств столь неблагоприятных, из которых умный человек не мог бы извлечь пользу. (Нет худа без добра)*



Глава 1

Создание компонента "Аккордеон"

В 80-х годах XX века было принято начинать обучение программированию с написания программы "Hello, world", которая печатала на черном экране монохромного дисплея два слова: "Hello World!" В 90-е годы программирование перешло на объектно-ориентированные рельсы, и программисты начали мыслить объектами. Поэтому первая программа начинающего программиста создавала объект `Dog` и вызывала единственный метод этого объекта: `Dog.bark()`. Осторожные программисты не забывали реализовать более полезный метод `Dog.foo()`.

Есть свои традиции и у Ajax-программистов. Первый компонент, который создают Ajax-программисты, называется *"Аккордеон"*. Такое музыкальное имя компонент "Аккордеон" получил благодаря своему внешнему сходству с этим замечательным инструментом. Когда компонент "Аккордеон" находится в свернутом состоянии, на экране отображаются только заголовки разделов. Если пользователь щелкнет по заголовку кнопкой мыши, разворачивается полный текст раздела — и мехи "Аккордеона" как бы растягиваются. Если пользователь щелкнет кнопкой мыши по другому заголовку — текущий раздел сворачивается и разворачивается новый раздел, соответствующий выбранному заголовку. "Аккордеон" играет!

Сложно сказать, почему именно этот компонент так полюбился Ajax-программистам. Нам же разработка компонента "Аккордеон" позволит сконцентрироваться на очень важных моментах в технике Ajax-программирования. В частности, на работе с *HTML-элементами* `DIV` и `SPAN`, *каскадными таблицами стилей CSS* и *объектной моделью документа DOM*. Кроме того, совсем неплохо на первом же уроке создать полнофункциональный компонент, который можно применить на практике.

Следующие два раздела посвящены использованию CSS в HTML-документах, после чего мы сможем приступить к разработке компонента "Аккордеон" во всеоружии.

Код готового компонента "Аккордеон" и некоторых примеров, рассматриваемых в данной главе, вы можете найти в папке \bhvajax\accordion на прилагаемом к книге компакт-диске.

1.1. HTML-элементы *DIV* и *SPAN* — основа построения современного HTML-документа

Основу построения современного *HTML-документа* составляют *элементы* `DIV` (от англ. *division* — раздел) и `SPAN` (от англ. *span* — фрагмент). Иногда вместо термина *"элемент"* не совсем точно употребляют термин *"тег"*, хотя тег и элемент — это совершенно разные понятия. *Открывающий тег* `<div>`, *закрывающий тег* `</div>` и часть документа, которая ограничена этими тегами, или *тело элемента* — называют элементом `DIV`. Тело элемента может содержать другие HTML-элементы и текст.

Элемент `DIV` отображается на экране в виде прямоугольного блока. Его свойство стиля `style.display` по умолчанию имеет значение `block`. Элемент `SPAN` отображается построчно, т. е. заполняет родительский блочный элемент построчно, как это делает печатная машинка на листе бумаги. Его свойство стиля `style.display` по умолчанию имеет значение `inline`. Для того чтобы ярче представить, о чем идет речь, создадим документ с использованием элементов `DIV` и `SPAN` (листинг 1.1).

Листинг 1.1. Использование HTML-элементов `DIV` и `SPAN`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Test CSS</title>
  </head>
  <body>
    <h1>Заголовок первого уровня.</h1>
    <p>
      Абзацы P выделяются при форматировании Web-браузером.
    </p>
    <p>
      Часто применяют выделение текста <b>полужирным шрифтом</b>
      и <i>курсивом</i>.
    </p>
```

```
    <div>
      Элемент DIV может выступать в качестве заголовка.</div>
    <div>
      Подобно абзацу P, элемент DIV отображается как блочный элемент.
    </div>
    <div>
      А этот текст не будет выделяться <span>полужирным шрифтом</span>
      и <span>курсивом</span>.
    </div>
    <div>
      А для этого элемента стиль может быть задан
      <span id="mySpan">по идентификатору id=mySpan</span>.
    </div>
  </body>
</html>
```

Документ можно создать в любом текстовом редакторе, например Блокноте, и сохранить в файле с расширением html, например divandspan.html. Теперь сохраненный в файле HTML-документ можно открыть любым Web-браузером, дважды щелкнув по нему кнопкой мыши или используя меню Web-браузера **Файл | Открыть файл**.

Разберем приведенный пример. Открывает и закрывает HTML-документ открывающий тег `<html>` и закрывающий тег `</html>`. Любой HTML-документ всегда содержит ровно один корневой элемент `HTML`. Если в документе отсутствуют теги `<html>` и `</html>`, Web-браузер сформирует элемент `HTML` автоматически, как это предусмотрено спецификацией HTML. Элемент `HTML` содержит ровно два дочерних элемента: `HEAD` и `BODY`, которые так же будут обязательно добавлены Web-браузером, если они отсутствуют в тексте HTML-документа. Элемент `HEAD` содержит специальную информацию об HTML-документе в целом. Собственно текст документа содержится в теле элемента `BODY`, между открывающим тегом `<body>` и закрывающим тегом `</body>`.

> ***Замечание***
>
> Важным отличием элемента `HEAD` от элемента `BODY` является то, что все ресурсы, на которые ссылается документ в элементе `HEAD`, будут загружены с Web-сервера в Web-браузер до начала отображения элемента `BODY`. В качестве таких ресурсов чаще всего выступают скрипты JavaScript и файлы с определением стилей документа CSS. Крайне важно загружать файлы с определением стилей CSS именно в элементе `HEAD` HTML-документа. Скрипты JavaScript не обязательно помещать в элемент `HEAD`. Более того, некоторые скрипты можно размещать только в `HEAD` или только в `BODY`. В `BODY` помещают скрипты, которые манипулируют HTML-элементами `BODY` HTML-документа.

Как вы заметили, Web-браузер допускает некоторые вольности в написании HTML-документа. Это закреплено в спецификации HTML. Кроме того, Web-браузер может исправлять некоторые некритичные ошибки разработчика HTML-документа. В частности, в любом документе обязательно будет присутствовать корневой элемент `HTML`. Ссылку на элемент `HTML` можно получить из свойства `document.documentElement`, где `document` — это предопределенный объект Web-браузера, представляющий документ как объект Web-браузера, а `document.documentElement` — это свойство объекта `document`, которое представляет документ как корневой элемент `HTML` объектной модели документа (DOM).

Как мы уже выяснили, элемент `HTML` содержит ровно два дочерних элемента — `HEAD` и `BODY`. Некоторые HTML-элементы могут присутствовать только в теле элемента `HEAD`. Это элементы `STYLE`, `META` и `LINK`. Некоторые HTML-элементы могут присутствовать как в теле элемента `HEAD`, так и в теле элемента `BODY` — это элементы `SCRIPT` и `OBJECT`. Все прочие типы HTML-элементов и текст могут располагаться только в теле элемента `BODY`.

Разметка текста в листинге 1.1 дана в двух вариантах. Первый вариант — с использованием так называемых типографских тегов, отвечающих за форматирование текста: `H1` — заголовок первого уровня, `P` — абзац (paragraf), `I` — курсив (italic), `B` — полужирный шрифт (bold). Второй вариант разметки — с использованием всего двух тегов `DIV` и `SPAN`.

При помощи типографских тегов можно красиво оформить HTML-документ с учетом самых тонких требований дизайна. Для чего же, в таком случае, понадобилось вводить в спецификацию HTML еще два новых элемента `DIV` и `SPAN`? Потребность в двух новых элементах возникла с появлением технологии *каскадные таблицы стилей CSS*. Каскадные таблицы стилей применяются для форматирования документа и для придания ему динамичности. Вместе с *объектной моделью документа DOM* и *языком программирования JavaScript*, каскадные таблицы стилей CSS стали основой технологии *динамический HTML (DHTML)*, а впоследствии одной из составляющих *инфраструктуры Ajax*. Для того чтобы старая модель форматирования документа при помощи типографских тегов была совместима с новой моделью CSS, были введены два совершенно новых элемента: `DIV` и `SPAN`. Их внешний вид полностью определяется каскадными таблицами стилей. Без использования CSS элементы `DIV` и `SPAN` отображаются как обычный неформатированный текст.

Большинство типографских тегов (такие как `H1`, `P`, `B`, `I`) совместимы и широко употребляются с новой моделью форматирования CSS. Исключение составляют теги `BASEFONT`, `CENTER`, `FONT`, `S`, `STRIKE` и `U`, которые объявлены *нежелательными (deprecated)*. Кроме того, нежелательными объявлены и без того редко используемые теги `DIR`, `ISINDEX` и `MENU`.

1.2. CSS — каскадные таблицы стилей

Каскадные таблицы стилей CSS — ключевая технология при разработке пользовательских интерфейсов Ajax-приложений. Именно на технологии CSS построены динамические эффекты DHTML и Ajax. Меню появляются и скрываются, окна (не окна Web-браузера, а рисованные окна, реализованные при помощи HTML-элемента `DIV`) открываются, перетаскиваются и бросаются (Drag-and-Drop), распахиваются и сворачиваются, списки раскрываются и закрываются — все это основано на использовании каскадных таблиц стилей CSS.

В данном разделе будут рассмотрены основы CSS, достаточные для начала работы с технологией Ajax. Кроме того, предлагаемый раздел будет служить в качестве краткого справочника для работы с CSS на протяжении всей книги. Поэтому некоторый материал будет дан подробнее, чем это следовало бы сделать при первом знакомстве. К счастью, технология CSS компактна, поэтому даже если вы никогда с ней не работали — очень скоро будете хорошо в ней ориентироваться.

Суть технологии CSS заключается в том, что стиль отображения HTML-элемента в окне Web-браузера определяется значением свойств атрибута `style`, который может быть задан для каждого HTML-элемента или для множества элементов, принадлежащих к одному стилевому классу. Атрибуту `style` HTML-элемента можно задавать следующие свойства: цвет текста `style.color`, цвет или рисунок фона `style.background`, шрифт `style.font`, внешний вид рамки `style.border`, отступы текста от рамки внутри элемента `style.padding` и отступы от соседних элементов `style.margin`, размеры элемента (`style.width`, `style.height`), координаты (`style.top`, `style.left`), способ отсчета координат (`style.position`: `static`, `relative` или `absolute`), способ отображения текста, выходящего за рамки элемента (`style.ovreflow`: `visible`, `scroll`, `hidden` или `auto`), способ врезки элемента в текст документа (`style.display`: `inline`, `block`, `none` и др.), видимость элемента (`style.visibility`: `inherit`, `visible` или `hidden`) и некоторые другие.

Определять стиль для каждого элемента HTML-документа заданием всех свойств атрибута `style` было бы слишком утомительно. Поэтому в технологии CSS введены *стилевые правила*. Стилевое правило может определять свойства стиля сразу для всех HTML-элементов одного типа, имеющих одинаковое имя тега. Например, для всех элементов `H1` (заголовков первого уровня) или для всех элементов `P` (абзац). Но чаще стилевые правила связывают не с именем тега элемента, а с именем *стилевого класса*. Имя стилевого

класса задается в атрибуте HTML-элемента `class`. Например, для всех элементов, у которых атрибут `class` равен `mystyleclass`.

Можно задать и более узкое правило: одновременно по имени тега элемента и по имени стилевого класса. Например, только для элементов `DIV` с атрибутом `class` равным `mystyleclass`. Наконец, можно задать самое узкое правило, которое будет распространяться на один-единственный HTML-элемент с конкретным идентификатором, который задается в атрибуте `id` HTML-элемента. Например, для элемента с атрибутом `id` равным `myelement`.

Наиболее рационально определять стилевые правила для HTML-документа в отдельном файле с расширением css (cascade stylesheet). Формат определения стилевого класса в css-файле следующий:

```
имя_HTML_тега {
  имя_свойства1: значение_свойства1;
  имя_свойства2: значение_свойства2;
  ...
}
имя_HTML_тега.имя_стилевого_класса {
  имя_свойства1: значение_свойства1;
  имя_свойства2: значение_свойства2;
  ...
}
.имя_стилевого_класса {
  имя_свойства1: значение_свойства1;
  имя_свойства2: значение_свойства2;
  ...
}
#идентификатор_элемента {
  имя_свойства1: значение_свойства1;
  имя_свойства2: значение_свойства2;
  ...
}
```

В css-файле записывается *селектор* правила, в качестве которого выступает имя тега, имя стилевого класса или идентификатор элемента, заданный в атрибуте `id`. За селектором следует стилевое правило, которое выделяется фигурными скобками. Список свойств в определении стилевого правила разделяется точкой с запятой, а имя стилевого свойства и значение стилевого свойства — двоеточием.

Например, если вы решили, что все заголовки первого уровня `H1` должны отображаться буквами размером `14pt` цвета индиго, стилевой класс необходимо определить так:

```
H1 {
  color: indigo;
  font: 14pt
}
```

Чтобы Web-браузер прочитал определение стилей из css-файла, необходимо в элемент `HEAD` документа добавить элемент `LINK`, как это показано в примере:

```
<html>
<head>
<title>Test CSS</title>
<link rel="stylesheet" type="text/css" href="divandspan.css">
</head>
<body>
...
</body>
</html>
```

Для того чтобы связать стилевые классы, определенные в css-файле, с HTML-элементами, служат атрибуты HTML-элементов `class` и `id`. Все атрибуты HTML-элемента записываются в открывающем теге HTML-элемента в форме *имя_арибута="значение_атрибута"*. Значение атрибута рекомендуется всегда брать в двойные или одинарные кавычки. Если вы всегда будете заключать атрибуты HTML-элементов в двойные кавычки — это сделает ваш текст более наглядным.

Вернемся к примеру и применим полученные знания CSS (листинг 1.2).

Листинг 1.2. Применение каскадных таблиц стилей CSS

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Test CSS</title>
    <link rel="stylesheet" type="text/css" href="divandspan.css">
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
```

```
    <p>
      Абзацы P выделяются при форматировании Web-браузером
    </p>
    <p>
      Часто применяют выделение текста <b>полужирным шрифтом</b>
      и <i>курсивом</i>.
    </p>
    <div class="header1">
      Элемент DIV в стиле заголовка первого уровня
    </div>
    <div class="paragraf1">
      Подобно абзацу P, элемент DIV отображается как блочный элемент.
    </div>
    <div>
      Другой способ выделения текста
      <span class="bold1">полужирным шрифтом</span>
      и <span class="italic1">курсивом</span>.
    </div>
    <div>
      А для этого элемента стиль задан
      <span id="mySpan">по идентификатору id=mySpan</span>
    </div>
  </body>
</html>
```

Приведенный в листинге 1.2 пример ссылается в HTML-элементе `LINK` на файл с определением стилей divandspan.css. Путь к файлу задан относительно текущей папки `src="divandspan.css"`, поэтому файл divandspan.css следует расположить в той же папке, что и файл с текстом примера divandspan.html. Заметим, что в промышленных приложениях разрабатываются css-файлы единые для всего приложения, и располагаются они в отдельной папке Web-сервера, которую часто называют \css.

Разработаем правила для отображения документа divandspan.html и поместим эти правила в файл divandspan.css. Пусть все элементы `H1` будут отображаться наклонным полужирным шрифтом размера `14pt` без засечек цвета индиго. Это правило для всех элементов `H1` будет определено в файле divandspan.css так:

```
H1 {
  color: indigo;
  font: italic bold 28pt/150% sans-serif
}
```

Для элементов стилевого класса `header1` определим точно такое же правило. Точка перед `.header1` в определении правила обязательна и обозначает, что `.header1` — это не имя HTML-элемента, а имя стилевого класса:

```
.header1 {
  color: indigo;
  font: italic bold 28pt/150% sans-serif
}
```

Два идентичных правила (как в нашем случае) можно записать в сокращенной форме через запятую:

```
H1, .header1 {
  color: indigo;
  font: italic bold 28pt/150% sans-serif
}
```

Запятая между `H1` и `.header1` обязательна. Без запятой такая запись имела бы совершенно другой смысл, и применялась бы к HTML-элементам с атрибутом `class` равным `header1` (т. е. стилевого класса `header1`), расположенным в теле элемента `H1`.

Для отображения элементов стилевого класса `paragraf1` определим стилевое правило: белые буквы на зеленом фоне в синей рамке:

```
.paragraf1 {
  color: white;
  background-color: green;
  border: solid 2px blue;
  font: 14pt/150% sans-serif
}
```

Стилевой класс `bold1` будет форматировать текст полужирным шрифтом без использования тегов `<b> </b>`:

```
.bold1 {
  font: bold 14pt/150% sans-serif
}
```

Стилевой класс `italic1` будет форматировать текст наклонным шрифтом без использования тегов `<i> </i>`:

```
.bold1 {
  font: bold 14pt/150% sans-serif
}
```

Наконец, чтобы закрепить работу с правилами для HTML-элементов, заданных по идентификатору элемента (атрибут `id`), определим правило для HTML-элемента с идентификатором `id` равным `mySpan`:

```
#mySpan {
  border-color: orange
}
```

Обратите внимание, в определении стиля для элемента с идентификатором `mySpan` используется не свойство `border`, а свойство `border-color`. Свойство `border` применяется как сокращенная форма записи сразу трех свойств: `border-style`, `border-color` и `border-width`. Если бы в файле divandspan.css вместо правила `border-color: orange` было определено правило `border: orange`, это бы означало, что неявно определяется правило `border-color: orange; border-style: none; border-width: 0px`, и элемент был бы отображен без рамки.

Свойство стиля `border` будет нами использоваться достаточно широко. При помощи этого свойства можно легко создавать динамические эффекты, например "подсвечивание" HTML-элемента, "нажатие кнопки". Разберем особенности применения свойства стиля `border` до конца. Это позволит нам в дальнейшем ссылаться на данный материал как на краткий справочник.

До сих пор мы применяли одинаковые стили ко всем четырем рамкам элемента: верхней, правой, нижней и левой. Стиль каждой из рамок можно определять раздельно при помощи свойств `border-top`, `border-right`, `border-bottom` и `border-left`:

```
#mySpan {
  border-top: solid orange 2px
}
```

Или, наконец, в самой полной форме:

```
#mySpan {
  border-top-color: orange
}
```

Допускается вариант сокращения, если вы хотите задать разные атрибуты для всех четырех рамок в одном правиле:

```
#mySpan {
  border-color: orange blue red yellow
}
```

Такая запись означает, что верхняя граница будет цвета `orange`, правая `blue`, нижняя `red` и левая `yellow`. Порядок перечисления границ ведется с верхней

границы по движению часовой стрелки. Это общее правило для всех аналогичных атрибутов CSS.

Значения свойства `border-style` определяет внешний вид линии, которой будет прорисована рамка. По умолчанию `border-style` линии равен `none`, т. е. никакой линии. Это означает, что для отображаемых линий свойство `border-style` всегда должно быть задано явно. Возможные значения свойства `border-style` следующие:

- `None` — значение по умолчанию, без рамки, принудительно устанавливает значение `border-width`, равное нулю;
- `Hidden` — скрытая, аналогично `none`, но имеет больший приоритет при наследовании стилей, что важно в некоторых сложных случаях наследования;
- `Solid` — сплошная линия (просто линия);
- `Double` — сплошная двойная линия;
- `Dotted` — точечная линия;
- `Dashed` — штриховая линия;
- `Groove` — вдавленная линия;
- `Ridge` — выпуклая линия;
- `Inset` — ступенька вверх;
- `Outset` — ступенька вниз.

Стили рамки `inset` и `outset` часто используют для рисования кнопок, которые можно нажимать и отпускать. Стиль `inset` (ступенька вверх) — обычное состояние кнопки. Стиль `outset` (ступенька вниз) — кнопка в нажатом состоянии.

Свойства стиля можно задавать непосредственно в атрибуте `style`. Я долгое время путал атрибуты `style` и `class`, потому что, на мой вкус, эти имена не совсем точно отражают их функциональность. Особенно это касается имени `class`, которое является зарезервированным словом в JavaScript и всегда рождает массу ненужных ассоциаций в головах объектно-ориентированных программистов. Подчеркнем разницу атрибутов `class` и `style`. Атрибут `class` содержит имя стилевого класса, который определен в css-файле и загружен в Web-браузер посредством элемента `LINK`. Атрибут `style` содержит строку с непосредственным определением параметров своего стилевого оформления. Например:

```
<span style="font: italic bold 14pt/150% sans-serif">
  Определение стиля в атрибуте style
</span>
```

Синтаксис определения стиля в css-файле и в атрибуте `style` одинаковый. Но в атрибуте `style` не следует брать определение стилевых правил в фи-

гурные скобки. Напомним, что подобно всем атрибутам, атрибут `style` следует взять в двойные или одинарные кавычки (апострофы).

Есть еще один способ определения стилевых классов, который мы не будем использовать в своих примерах. Определять стилевые классы можно в элементе `STYLE` непосредственно в `HEAD` HTML-документа. При этом теряется одно из главных преимуществ технологии CSS — отделение стилей оформления документа от текста документа (поэтому я даже не даю примера использования этой возможности).

Мы завершили обзор технологии CSS каскадных таблиц стилей. Теперь ответим на вопрос: почему таблицы стилей называют каскадными таблицами стилей? Дело в том, что на каждый элемент действует одновременно несколько стилевых правил, которые могут быть:

- унаследованы от родительского элемента, заданы в файле CSS или заданы непосредственно в атрибуте `style` элемента;
- взяты из профиля пользователя, настроек Web-браузера и операционной системы.

Из всех этих правил будет выбрано единственное правило с наибольшим приоритетом, которое и будет использовано для отображения элемента в окне Web-браузера. Такая логика определения стиля элемента называется наследованием и каскадированием. Правила наследования и каскадирования отвечают на вопрос, какое значение будет присвоено свойству стиля элемента при отображении HTML-документа, если на него действует сразу несколько стилевых правил.

Подведем итоги. Применение каскадных таблиц стилей позволяет вынести описание стилей оформления HTML-документа в отдельный файл с расширением css, ссылка на который должна содержаться в теге `LINK`:

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

Для привязки HTML-элемента к стилевому классу, определенному в css-файле, используется атрибут HTML-элемента `class`. Кроме того, стилевой класс можно определить для HTML-элементов по имени тега элемента или для конкретного элемента по его идентификатору, заданному в атрибуте `id`.

1.3. Разработка каскадных таблиц стилей для компонента "Аккордеон"

Сейчас мы приступаем к разработке компонента "Аккордеон". Преимущества использования технологии CSS заключаются в том, что разработку стилевого оформления компонента можно проводить независимо от разработки кода

компонента. Не имеет смысла задумываться, с чего начинать: с разработки кода или с разработки стилевого оформления. Разработку кода компонента и стилевого оформления можно вести параллельно, предварительно согласовав имена стилевых классов. Но формат книги требует последовательного изложения.

Компонент "Аккордеон" состоит из двух типов подкомпонентов: *заголовков разделов* и *панелей разделов*. Заголовки разделов всегда отображены в окне Web-браузера, если только отображен компонент "Аккордеон". Из панелей разделов всегда отображена только одна панель. Назовем такую видимую панель *активной* или *текущей панелью*. Все *неактивные панели* скрыты и представлены в окне Web-браузера только своими заголовками (рис. 1.1).

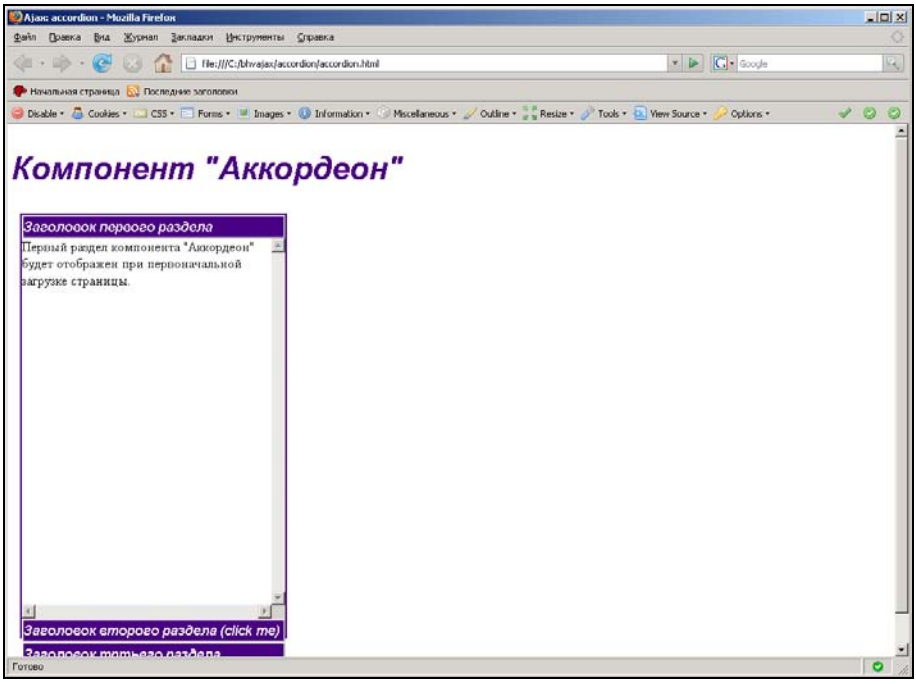


Рис. 1.1. Внешний вид компонента "Аккордеон"

Предварительно можно предположить, что для стилевого оформления компонента должны быть определены следующие стилевые классы:

- стиль компонента `class="accordionComponent"`;
- стиль активного заголовка `class="accordionCurrentTitle"`;
- стиль неактивного заголовка `class="accordionOtherTitle"`;

- стиль активной панели `class="accordionCurrentPane"`;
- стиль неактивной панели `class="accordionOtherPane"`.

Определения стилей поместим в файл accordion.css.

Я предложил начать разработку компонента "Аккордеон" с разработки стилей документа, чтобы вы на практике могли убедиться в удобстве технологии CSS. Если стилевое оформление документа отделено от самого документа, вы можете поручить разработку стилей специалистам в области художественного дизайна, можете вести работу параллельно, можете разработать несколько стилей для одного компонента или можете использовать один стиль для разных компонентов, чтобы все компоненты выглядели как единое целое. Согласитесь, заманчивая перспектива!

Для стилевого класса компонента "Аккордеон" зададим высоту `500px` и ширину `30%` в процентах от размеров родительского HTML-элемента (в нашем случае родительским элементом компонента "Аккордеон" будет HTML-элемент `BODY`). Обязательно присвоим значение свойству стиля `overflow: visible`, т. е. при переполнении размер элемента будет увеличен. Это существенно упрощает поставленную задачу, т. к. избавляет нас от необходимости вычислять высоту и ширину компонента в зависимости от многих заранее непредвиденных обстоятельств: типа Web-браузера, разрешения экрана, размеров родительского элемента, объема текста разделов компонента "Аккордеон", количества разделов. Более точные методы нам еще предстоит освоить, но сейчас мы создаем наш первый компонент. В файле accordion.css определение стилевого класса компонента "Аккордеон" будет следующим:

```
.accordionComponent{
  width: 30%;
  margin: 10px;
  border: solid indigo;
  border-width: 2px 2px 2px 2px;
  overflow: visible;
  height: 500px
}
```

Стиль заголовков разделов может быть произвольным. Но обязательно следует предусмотреть рамку (о рамках мы уже знаем все), иначе в свернутом состоянии заголовки "Аккордеона" будут сливаться. Заголовки активных и неактивных разделов в предлагаемом варианте будут иметь одинаковое стилевое оформление. Но, возможно, у вас появится желание сделать их различными. Можете прямо сейчас поэкспериментировать с цветами фона и стилевым оформлением рамок. Например, может быть, вы захотите задать для

активного заголовка рамку `inset`, а для неактивного заголовка рамку `outset`. Мне же кажутся более подходящими для данного случая такие стилевые правила:

```
.accordionCurrentTitle{
  background: indigo;
  color: white;
  font: italic bold 12pt/150% sans-serif;
  border: outset 2px
}
.accordionOtherTitle{
  background: indigo;
  color: white;
  font: italic bold 12pt/150% sans-serif;
  border: outset 2px
}
```

Для стилевого класса активной панели следует учесть несколько важных моментов. Значение свойства `display: block` означает, что панель будет отображена в окне Web-браузера. Значение свойства `overflow: scroll` означает, что при отображении текста, который не помещается на панели — будут сформированы полосы прокрутки. И, наконец, свойство `height: 90%`, заданное в процентном отношении от высоты родительского компонента "Аккордеон", позволяет логично отобразить разделы в случае, если текст раздела не заполняет все свободное пространство панели. Для таких разделов высота панели будет принудительно увеличена до `90%` от высоты компонента "Аккордеон". Это позволяет более логично отобразить панель, которая содержит короткий текст. Без указания `height: 90%`, панели компонента "Аккордеон" сжимались бы по размеру текста и имели бы различную высоту в зависимости от размера текста раздела. Сейчас мы используем примитивные, но надежные методы подгонки размеров компонентов, ведь это наш первый компонент. Не огорчайтесь, уже в этой главе мы применим более точные методы подгонки при помощи анализа свойств элементов `clientHeight` и `offsetHeight` средствами JavaScript.

Стилевой класс для активной панели в файле accordion.css определен так:

```
.accordionCurrentPane{
  border: none;
  overflow: scroll;
  display: block;
  height: 90%
}
```

Единственным свойством стилевого класса неактивной панели является `display: none`, что означает: панель невидима в окне Web-браузера и не занимает места в этом окне.

```
.accordionOtherPane{
    display: none
}
```

> ***Замечание***
>
> Свойство стиля `display` немного похоже на свойство стиля `visibility`. Свойство стиля `visibility` может принимать значение `inherit` (унаследовано от родительского элемента), `visible` и `hidden`. Если свойству стиля `visibility` присвоить значение `hidden`, элемент становится невидимым, но продолжает занимать место на экране. Нам необходимо сделать элемент неактивной панели невидимым и освободить занимаемое им место для отображения активной панели. Поэтому мы используем свойство стиля `display` со значением `none`.

Все определения стилей сохраним в файле accordion.css, который можно найти на компакт-диске. Продолжим разработку компонента.

1.4. Разработка HTML-документа для компонента "Аккордеон"

Компонент "Аккордеон" будет представлен в документе в виде HTML-элемента `DIV`. Заголовки разделов и панели разделов будут также представлены элементами `DIV`, которые содержатся в главном элементе `DIV` компонента "Аккордеон" и расположены в следующем порядке:

```
<div title="Компонент 'Аккордеон'">
<div>Заголовок раздела №1</div>
<div>Панель раздела №1</div>
<div>Заголовок раздела №2</div>
<div>Панель раздела №2</div>
...
</div>
```

Для работы с компонентом создадим файл accordion.html (см. на компакт-диске). Прошу вас обратить особое внимание на заголовок документа:

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

Существует несколько схем `DTD` для HTML-документов. Мы будем использовать нестрогий тип `Transitional`. В практической работе лучше использо-

вать более строгую схему `Strict`. Иногда разработчики совсем не используют объявление типа документа в HTML-документах. Это неправильно. Дело в том, что поведение документов с явно объявленным типом является более кросс-браузерным и может существенно отличаться от поведения документа без явно объявленного типа. Если вы пишите документы сложнее, чем простой текст, явное объявление типа документа вам просто необходимо. Такой документ требует большей аккуратности разработчика. Если вы разработали документ без явного объявления типа, а затем решили добавить описание типа в готовый документ — будьте готовы к тому, что всю работу придется проделать заново. Поэтому явное объявление типа документа должно присутствовать с первых этапов разработки документа.

Теперь вы можете раскрыть Web-браузером файл accordion.html и... конечно заняться отладкой компонента, который включает на данном этапе два файла: accordion.css и accordion.html. Совершенно невероятно, чтобы все получилось сразу. Не стоит отчаиваться. Наконец начинается интересная работа. Проверьте наличие открывающих и закрывающих фигурных скобок в определениях стилей в файле accordion.css. Далее убедитесь в наличии двоеточий между именем и значением атрибута, а также точки с запятой после каждой пары *`имя_атрибута`*`:` *`значение_атрибута`*`;` в файле accordion.css. Теперь раскройте файл accordion.html. Здесь следует проверить парность открывающих и закрывающих тегов элементов, наличие наклонной черты в закрывающем теге перед именем тега элемента. Проверьте, все ли строковые атрибуты элементов правильно закрыты. Открывающая и закрывающая кавычки строкового атрибута должны быть одинаковыми — обе двойные или обе одинарные. Очень часто строковую константу просто забывают закрыть, и весь текст документа оказывается внутри одной большой строковой константы. Если вы добились правильного отображения документа — можно продолжить разработку компонента.

Наш компонент "Аккордеон" отображен Web-браузером: как и предполагалось, видны заголовки всех разделов и содержимое панели активного раздела. Все неактивные панели скрыты. Но нет возможности отобразить содержимое скрытых панелей.

Как же заставить компонент "Аккордеон" работать? Очевидное решение — по щелчку кнопкой мыши на заголовке раздела присвоить атрибуту `class` текущего раздела имя стилевого класса `accordionOtherPane`, а атрибуту `class` следующего выбранного раздела — имя стилевого класса `accordionCurrentPane`. Для этого мы должны привлечь возможности программирования на JavaScript.