ГЛАВА 2

Введение в HTML

На протяжении всей книги мы будем постоянно работать с HTML-документами. Материал данной главы необходим для понимания того, что мы будем делать в остальной части книги. Это не руководство по HTML, а скорее описание ключевых характеристик HTML, которые составят основу для обсуждения в последующих главах.

Последняя версия HTML, известная под названием *HTML5*[1], уже сама по себе могла бы служить отдельной темой для изучения. Она содержит более ста элементов, каждый из которых имеет свое назначение и обладает собственной функциональностью. Для понимания принципов работы jQuery достаточно даже элементарных знаний HTML.

Базовый HTML-документ

Для начала посмотрим, как выглядит HTML-документ, чтобы вы получили первое представление о базовой структуре и иерархических принципах построения, которым подчиняется любой HTML-документ. Простой пример такого документа приведен в листинге 2.1. Этот документ используется в данной главе для того, чтобы познакомить вас с основными концепциями HTML.

Листинг 2.1. Пример простого HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-2.1.0.js" type="text/javascript"></script>
    <style>
        h1 {
            width: 700px; border: thick double black;
            margin-left: auto; margin-right: auto;
            text-align: center; font-size: x-large; padding: .5em;
            color: darkgreen; background-image: url("border.png");
            background-size: contain; margin-top: 0;
        }
        .dtable {display: table;}
        .drow {display: table-row;}
```

[1] Отсутствие пробела перед номером версии в названии спецификации, разрабатываемой группой WHATWG, не случайно. Подробный и увлекательный рассказ обо всех перипетиях развития HTML5 и о характере взаимного сотрудничества консорциума W3C и WHATWG можно найти по адресу http://www.habrahabr.ru/blogs/webstandards/103256/. — *Примеч. ред.*

```
        .dcell {display: table-cell; padding: 10px;}
        .dcell > * {vertical-align: middle}
        input {width: 2em; text-align: right;
               border: thin solid black; padding: 2px;}
        label {width: 5em; padding-left: .5em;
               display: inline-block;}
        #buttonDiv {text-align: center;}
        #oblock {display: block; margin-left: auto;
                 margin-right: auto; width: 700px;}
    </style>
</head>
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="aster.png"/>
                        <label for="aster">Астры:</label>
                        <input name="aster" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
```

```
    </form>
</body>
</html>
```

Несмотря на небольшой размер и простоту, этот документ позволяет выяснить ряд наиболее важных моментов, связанных с использованием HTML. Вид данного документа при его просмотре в окне браузера представлен на рис. 2.1[2].

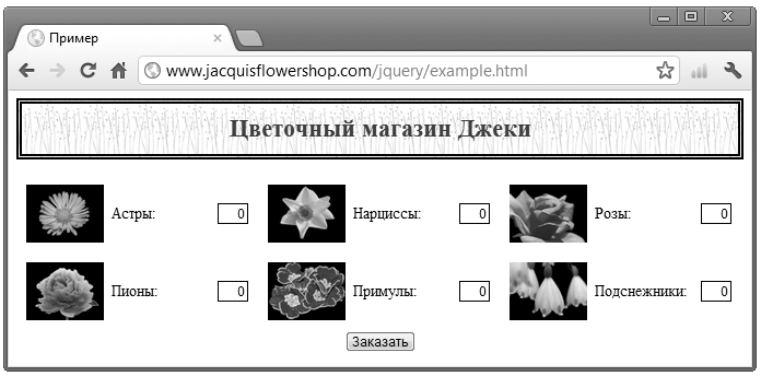

Рис. 2.1. Вид HTML-документа в окне браузера

Структура элементов HTML

В основе HTML лежит понятие *элемента*. Посредством элементов браузеру передается информация о типе содержимого каждой из частей HTML-документа. В качестве примера рассмотрим следующий элемент, взятый из приведенного выше примера:

```
...
<h1>Цветочный магазин Джеки</h1>
...
```

Этот элемент состоит из трех частей: открывающего (начального) дескриптора, закрывающего (конечного) дескриптора и содержимого (рис. 2.2).

Рис. 2.2. Структура простого HTML-элемента

Здесь `h1` — это *имя* данного элемента (также говорят *имя дескриптора*), которое указывает браузеру на то, что содержимое, заключенное между дескрипторами, должно интерпретироваться как заголовок верхнего уровня. Открывающий дескриптор создается путем заключения имени

[2] Для выполнения приведенных в книге примеров копируйте содержимое соответствующего листинга в файл `example.com`, который должен находиться в подкаталоге `jquery` корневого каталога вашего веб-сервера и в котором должны храниться также все необходимые вспомогательные файлы изображений. Кроме того, добавьте в файл `hosts` (в Windows — `C:\Windows\system32\drivers\etc\hosts`, в Linux — `/etc/hosts`) запись `127.0.0.1 www.jacquisflowershop.com` (требуются права администратора). — *Примеч. ред.*

дескриптора в две угловые скобки (< и >). Точно так же создается и закрывающий дескриптор, только в этом случае вслед за левой угловой скобкой (<) дополнительно ставится косая черта (/).

Атрибуты

Браузеру можно предоставлять дополнительную информацию об элементах, снабжая их *атрибутами*. В качестве примера в листинге 2.2 показано, как выглядит один из элементов нашего образца HTML-документа, имеющий атрибут.

Листинг 2.2. Объявление атрибута

```
...
<label for="aster">Астры:</label>
...
```

Здесь для элемента `label` определен атрибут `for`. Чтобы атрибут был более заметен, он выделен в тексте. Атрибуты всегда указываются в открывающем дескрипторе. У каждого атрибута есть *имя* и *значение*. В данном случае имя атрибута — `for`, а значение — `aster`. Не все атрибуты имеют значения; уже сам факт наличия некоторых атрибутов указывает браузеру на то, что вы хотите связать с данным элементом определенный тип поведения. Пример элемента, которому присвоен атрибут такого типа, приведен в листинге 2.3.

Листинг 2.3. Объявление атрибута, не требующего указания значения

```
...
<input name="snowdrop" value="0" required>
...
```

В данном элементе определены три атрибута. Первым двум из них, `name` и `value`, присвоены значения, как это было сделано в предыдущем примере. (Имена этих атрибутов — `name` и `value` — не должны сбивать вас с толку: `snowdrop` — это значение атрибута `name`, а `0` — это значение атрибута `value`.) Третий атрибут представлен единственным словом — `required` (“требуется”). В данном случае мы имеем дело с атрибутом, задавать значение которого необязательно, хотя его и можно определить, указав значение, совпадающее с именем атрибута (`required="required"`), или использовав в качестве значения пустую строку (`required=""`).

Атрибуты `id` и `class`

В этой книге особое значение для нас будут иметь два атрибута: `id` и `class`. Одной из наиболее распространенных задач, с которыми приходится сталкиваться в процессе работы с jQuery, является определение местоположения одного или нескольких элементов в документе для последующего выполнения над ними некоторой операции. Атрибуты `id` (идентификатор) и `class` (класс) значительно облегчают поиск нужных элементов в документе.

Использование атрибута `id`

Атрибут `id` используется в качестве уникального идентификатора элемента в документе. В документе не должно быть двух элементов, имеющих одинаковые значения `id`. Пример простого документа, в котором используется атрибут `id`, представлен в листинге 2.4.

Листинг 2.4. Использование атрибута `id`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
```

```
</head>
<body>
    <h1 id="mainheader">Добро пожаловать в цветочный магазин
        Джеки!</h1>
    <h2 id="openinghours">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 id="holidays">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В этом документе значения атрибута `id` определены для трех элементов. Значением атрибута `id` элемента `h1` является `mainheader`, элемента `h2` — `openinghours`, элемента `h3` — `holidays`. Атрибут `id` обеспечивает возможность поиска конкретных элементов в пределах документа.

Использование атрибута `class`

Атрибут `class` используется для произвольного группирования элементов. Один и тот же класс может быть присвоен многим элементам, а любой элемент может принадлежать одновременно нескольким классам, как показано в листинге 2.5.

Листинг 2.5. Использование атрибута `class`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <h1 id="mainheader" class="header">Добро пожаловать в
        цветочный магазин Джеки!</h1>
    <h2 class="header info">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 class="info">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В листинге 2.5 элемент `h1` принадлежит классу `header`, элемент `h2` — классам `header` и `info`, а элемент `h3` — только классу `info`. Как видно из листинга, один и тот же элемент может входить сразу в несколько классов, имена которых указываются в виде списка с использованием пробела в качестве разделителя.

Содержимое элементов

Элементы могут включать не только текст, но и другие элементы. Ниже приведен пример элемента, в котором находятся другие элементы.

```
...
<div class="dcell">
    <img src="rose.png"/>
    <label for="rose">Розы:</label>
    <input name="rose" value="0" required>
</div>
...
```

Здесь в элемент `<div>` помещены три других элемента: `img`, `label` и `input`. Возможно несколько уровней *вложения* элементов, а не только один, как в данном примере. Вложение

(подчинение) элементов — ключевая концепция HTML, поскольку она подразумевает распространение функциональности внешнего элемента на элементы, содержащиеся в нем (к обсуждению этой темы мы еще вернемся). Допускается смешивание текстового содержимого с другими элементами, как показано в следующем примере:

```
...
<div class="dcell">
    Это текстовое содержимое
    <img src="rose.png"/>
    Это дополнительный текст!
    <input name="rose" value="0" required>
</div>
...
```

Пустые элементы

Не все элементы могут иметь содержимое. Элементы, у которых не может быть содержимого, называются *пустыми* и записываются без отдельного закрывающего дескриптора. Вот как выглядит пустой элемент.

```
...
<img src="rose.png"/>
...
```

Пустой элемент определяется с помощью единственного дескриптора, в котором перед закрывающей угловой скобкой (>) помещается символ косой черты (/). Строго говоря, символ / должен отделяться от последнего символа последнего атрибута пробелом, как показано ниже.

```
...
<img src="rose.png" />
...
```

Однако в том, что касается интерпретации HTML-кода, браузеры весьма терпимы, так что символ пробела можно смело опускать. Пустые элементы часто используют, если элемент ссылается на внешний ресурс. В данном случае элемент `img` используется для ссылки на внешний файл изображения с именем `rose.png`.

Структура документа

Существуют ключевые элементы, которые определяют базовую структуру любого HTML-документа. Таковыми являются элементы `DOCTYPE`, `html`, `head` и `body`. В листинге 2.6 показано, как эти элементы соотносятся с остальной частью содержимого, которая для краткости опущена.

Листинг 2.6. Базовая структура HTML-документа

```
<!DOCTYPE html>
<html>
<head>
    ...содержимое элемента head...
</head>
<body>
    ...содержимое элемента body...
</body>
</html>
```

Каждый из этих элементов выполняет в HTML-документе свои специфические функции. Элемент DOCTYPE сообщает браузеру о том, что данный документ является HTML-документом, точнее — документом, соответствующим стандарту HTML5. Более ранние версии HTML требуют указания дополнительной информации. Вот так, например, выглядит элемент DOCTYPE для документа HTML4.

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

Элемент html обозначает область документа, в которой находится HTML-содержимое. Этот элемент всегда содержит два других ключевых структурных элемента: head и body. Как подчеркивалось в начале главы, подробное изучение всех HTML-элементов не входит в наши планы. Число существующих элементов настолько велико, что для их полного описания в книге по HTML5 потребовалось более тысячи страниц. Учитывая это обстоятельство, я ограничусь лишь кратким описанием используемых элементов, чего, однако, будет вполне достаточно для того, чтобы вы хорошо понимали смысл документов, с которыми придется работать. В табл. 2.1 перечислены элементы, используемые в нашем образце документа (часть из них будет описана более подробно далее).

Таблица 2.1. HTML-элементы, входящие в образец документа

Элемент	Описание
DOCTYPE	Указывает тип содержимого текущего документа
body	Обозначает область документа, содержащую другие элементы (более подробно описывается далее)
button	Обозначает кнопку; часто используется для отправки на сервер элемента form
div	Универсальный элемент-контейнер; часто используется для дополнительного структурирования документа с целью улучшения его представления
form	Обозначает HTML-форму, позволяющую получать данные от пользователя и отправлять их на сервер для последующей обработки
h1	Обозначает заголовок
head	Обозначает область документа, содержащую метаданные (более подробно описывается далее)
html	Обозначает область документа, содержащую HTML-код (обычно это весь документ)
img	Обозначает изображение
input	Обозначает поле ввода, используемое для получения единичной порции данных от пользователя, и обычно является частью HTML-формы
script	Обозначает сценарий (обычно на языке JavaScript), который должен выполняться как часть документа
style	Обозначает область документа, содержащую параметры каскадных стилевых таблиц CSS (см. главу 3)
title	Обозначает название текущего документа; используется браузером для задания заголовка окна (или вкладки), в котором отображается содержимое документа

Элементы метаданных

Элемент `head` предназначен для хранения метаданных документа, иными словами — одного или нескольких элементов, которые описывают содержимое документа или воздействуют на него, но сами не отображаются браузером. В раздел `head` нашего образца документа включены три элемента метаданных: `title`, `script` и `style`. Самый простой из них — это элемент `title`. Его содержимое выводится браузером в качестве заголовка окна или вкладки, и его необходимо указывать в любом HTML-документе. Два остальных элемента играют более важную роль в этой книге, и их смысл будет объяснен более подробно в последующих разделах.

Элемент `script`

С помощью элемента `script` можно включать в свой код сценарии JavaScript. Именно работе с этим элементом мы будем уделять больше всего внимания, как только перейдем к углубленному рассмотрению jQuery. В нашем образце документа имеется один элемент `script` (листинг 2.7).

Листинг 2.7. Элемент `script` из образца документа

```
...
<script src="jquery-2.1.0.js" type="text/javascript"></script>
...
```

Определяя для элемента `script` атрибут `src`, вы сообщаете браузеру, что хотите загрузить код JavaScript, содержащийся в другом файле. В данном случае этим файлом является основная библиотека jQuery, которую браузер загрузит из файла `jquery-2.1.0.js`. Один HTML-документ может содержать несколько элементов `script`, и при необходимости можно включить код сценария JavaScript непосредственно в документ, поместив его между открывающим и закрывающим дескрипторами элемента `script`, как показано в листинге 2.8.

Листинг 2.8. Включение встроенного кода JavaScript в документ с помощью элемента `script`

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-2.1.0.js" type="text/javascript"></script>
    <script type="text/javascript">
        $(document).ready(function() {
            $("#mainheader").css("color", "red");
        });
    </script>
</head>
<body>
    <h1 id="mainheader" class="header">Добро пожаловать в
        цветочный магазин Джеки!</h1>
    <h2 class="header info">Мы открыты с 10 до 18 часов ежедневно
        без выходных</h2>
    <h3 class="info">(закрыт в официальные праздничные дни)</h3>
</body>
</html>
```

В этом примере имеются два элемента `script`. Первый из них импортирует в документ библиотеку jQuery, тогда как второй содержит простой сценарий, в котором используется базовая функциональность jQuery. Не старайтесь сейчас понять, как работает второй сценарий. В главе 5

мы приступим к систематическому изучению возможностей библиотеки jQuery. Элементы `script` могут включаться в элементы `head` и `body` HTML-документа. Как правило, в данной книге я буду помещать элементы `script` в элементы `head`, но это всего лишь дело вкуса.

■ **Совет.** Порядок следования элементов `script` имеет значение. Прежде чем использовать библиотеку jQuery, ее необходимо загрузить.

Элемент `style`

Элемент `style` обеспечивает один из нескольких возможных способов, с помощью которых в документ могут включаться свойства каскадных таблиц стилей (Cascading Style Sheets — CSS). Если говорить коротко, то свойства CSS позволяют управлять внешним видом документа при его отображении в браузере. Элемент `style`, входящий в состав образца документа, представлен вместе со своим содержимым в листинге 2.9.

Листинг 2.9. Использование элемента `style`

```
...
<style>
    h1 {
        width: 700px; border: thick double black;
        margin-left: auto; margin-right: auto;
        text-align: center; font-size: x-large; padding: .5em;
        color: darkgreen; background-image: url("border.png");
        background-size: contain; margin-top: 0;
    }
    .dtable {display: table;}
    .drow {display: table-row;}
    .dcell {display: table-cell; padding: 10px;}
    .dcell > * {vertical-align: middle}
    input {width: 2em; text-align: right;
           border: thin solid black; padding: 2px;}
    label {width: 5em;  padding-left: .5em;
           display: inline-block;}
    #buttonDiv {text-align: center;}
    #oblock {display: block; margin-left: auto;
             margin-right: auto; width: 700px;}
</style>
...
```

Браузер поддерживает набор свойств, значения которых используются для управления внешним видом каждого элемента. Элемент `style` позволяет выбирать элементы и изменять значения одного или нескольких таких свойств. Более подробно эта тема рассматривается в главе 3.

Как и элемент `script`, элемент `style` можно включать в элементы `head` и `body`, но в книге я всегда помещаю его в раздел `head`, как это сделано в образце документа. Такой подход также является делом вкуса; лично я предпочитаю, чтобы мои стили располагались в документе отдельно от его основного содержимого.

Элементы содержимого

В элемент `body` помещается *содержимое* HTML-документа. Под этим подразумеваются те элементы, которые браузер будет отображать для пользователя и на которые воздействуют такие элементы метаданных, как `script` и `style`.

Разделение семантики и представления

Одно из главных новшеств спецификации HTML5 носит философский характер: в этой спецификации большое значение придается отделению семантики элемента от его воздействия на способ представления содержимого. Это глубокая идея. Элементы HTML используются для структуризации содержимого и наделения его функциональностью, а также для последующего управления способом представления этого содержимого путем применения стилей CSS к элементам. Далеко не все HTML-документы нуждаются в отображении (такие документы могут использоваться не только браузерами, но и автоматизированными программами), так что отделение представления HTML-документов от их содержимого упрощает их обработку и извлечение содержащейся в них информации автоматизированными методами.

Каждый HTML-элемент имеет определенный смысл. Например, элемент `article` используется для обозначения самостоятельной части содержимого, пригодной для синдикации, т.е. независимого распространения или многократного использования, тогда как элемент `h1` — для обозначения заголовка раздела содержимого.

Эта концепция составляет основу HTML. Элементы служат для обозначения типа соответствующего содержимого. Люди способны делать весьма точные заключения о смысле отдельных частей документа, исходя из контекста. Так, вы без труда определите, что заголовок некоторого раздела страницы подчинен предыдущему заголовку, если последний напечатан шрифтом большего размера.

Компьютерам до этого пока еще далеко, и именно поэтому им облегчают задачу, помещая различные разделы содержимого в отдельные элементы для обозначения того, как они соотносятся между собой. Пример документа, в котором элементы используются для придания отдельным составляющим определенного структурного и смыслового значения, представлен в листинге 2.10.

Листинг 2.10. Использование HTML-элементов для придания структурного и смыслового значения содержимому

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
</head>
<body>
    <article>
        <header>
            <hgroup>
                <h1>Новая служба доставки</h1>
                <h2>Цвет и красота у ваших дверей</h2>
            </hgroup>
        </header>
        <section>
            Мы рады сообщить о начале предоставления новой
            услуги - доставки заказанных вами цветов прямо на дом.
            В радиусе 20 миль от магазина доставка осуществляется
            бесплатно, доплата за каждую дополнительную милю
            составляет $1. Мы гарантируем, что наши цветы вам
            понравятся. Дополнительную бесплатную консультацию
            вы можете получить по телефону.
        </section>
        <section>
            Новая услуга будет оказываться начиная
            со <b>среды</b>. Первым 50 покупателям предоставляется
            скидка $10.
```

```
        </section>
        <footer>
            <nav>
                Дополнительная информация:
                <a href="http://jacquisflowershop.com">
                    Узнайте больше о продукции</a>
            </nav>
        </footer>
    </article>
</body>
</html>
```

Каких-либо жестких правил, регламентирующих применение элементов `section` и `article`, не существует, однако весьма желательно, чтобы вы систематически применяли их в своих документах для семантической разметки содержимого. Эти элементы не предоставляют браузеру никакой информации относительно того, как должно отображаться содержимое, что отражает саму суть принципа разделения содержания и представления документа. В отношении большинства HTML-элементов браузеры руководствуются *соглашениями о стилях*, определяющими, как именно должны отображаться элементы, если их представление не было изменено стилями CSS, однако подразумевается, что именно широкое использование стилей CSS будет обеспечивать внешний вид документа. Это можно сделать как с помощью элемента `style`, так и средствами библиотеки jQuery, которая обеспечивает простой способ решения данной задачи с помощью элемента `script`.

Некоторые из элементов, определенных в спецификации HTML 4, создавались тогда, когда идея о необходимости разделения содержимого и представления документа еще не созрела, в результате чего иногда возникают двусмысленные ситуации. В качестве примера можно привести элемент `b`. До появления спецификации HTML5 элемент `b` сообщал браузеру о том, что содержимое, заключенное между его начальным и конечным дескрипторами, следует отображать полужирным начертанием. В спецификации HTML5, которая не поощряет использования элементов, управляющих исключительно внешним видом содержимого, дается новое определение этого элемента.

> *Элемент `b` представляет фрагмент текста, некоторым образом выделенный относительно окружающего его содержимого, однако, в отличие, например, от выделения ключевых слов в рефератах документов или названий продуктов в обзорах, такое выделение, общепринятым типографским способом реализации которого является использование полужирного начертания для текста, не предполагает придания данному тексту какого-либо дополнительного логического акцента или подчеркивания степени его важности.*
>
> HTML: The Markup Language, w3c.org

С помощью этой довольно витиеватой формулировки нам просто пытаются сказать, что элемент `b` инструктирует браузер о том, что данный текст следует выделить полужирным начертанием. Элемент `b` не несет никакой семантической нагрузки, и речь здесь идет исключительно о способе представления текста. Туманность этого определения позволяет сделать один важный попутный вывод относительно стандарта HTML5: он все еще находится в стадии становления. Нам *хотелось бы*, чтобы принцип отделения элементов от их представления соблюдался в полной мере, но реальность такова, что одновременно требуется обеспечить совместимость стандарта с бесчисленным количеством уже имеющихся документов, написанных с использованием более ранних версий HTML, и поэтому мы вынуждены идти на определенный компромисс.

Элементы form и input

Одним из наиболее интересных элементов в теле образца документа является элемент `form`, представляющий собой механизм, с помощью которого можно получать от пользователей данные для их отправки на сервер. Как вы сами убедитесь в части III, библиотека jQuery предоставляет великолепную поддержку для работы с формами, что обеспечивается как средствами ядра библиотеки, так и некоторыми широко используемыми плагинами. Элемент `body`, входящий в состав образца документа, представлен вместе со своим содержимым в листинге 2.11.

Листинг 2.11. Содержимое образца документа

```
...
<body>
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="aster.png"/>
                        <label for="aster">Астры:</label>
                        <input name="aster" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
```

```
        </div>
    </form>
</body>
...
```

Везде, где встречается элемент `form`, обычно присутствует и элемент `input`. Он используется для получения определенной порции данных от пользователя. Элемент `input`, входящий в образец документа, представлен в листинге 2.12.

Листинг 2.12. Использование элемента `input`

```
...
<input name="snowdrop" value="0" required>
...
```

В данном случае элемент `input` получает от пользователя значение элемента данных `snowdrop`, инициализированного нулевым значением. Атрибут `required` сообщает браузеру о том, что пользователь не должен иметь возможности отправить форму на сервер, не предоставив предварительно значение элемента данных. Эта новая возможность, которая носит название *валидации* (проверки допустимости) формы, была впервые предусмотрена в HTML5, но, по правде говоря, валидация данных, обеспечиваемая jQuery, гораздо более эффективна, как это будет продемонстрировано в части III.

С формами тесно связан элемент `button`, который часто используется для отправки формы на сервер (а также может быть использован для сброса формы в исходное состояние). Фрагмент примера документа, в котором определен элемент `button`, приведен в листинге 2.13.

Листинг 2.13. Использование элемента `button`

```
...
<button type="submit">Заказать</button>
...
```

Указав для атрибута `type` значение `submit`, мы сообщаем браузеру о том, что щелчок на кнопке должен приводить к отправке формы на сервер. Содержимое элемента `button` отображается в браузере поверх соответствующего элемента управления, как показано на рис. 2.3.

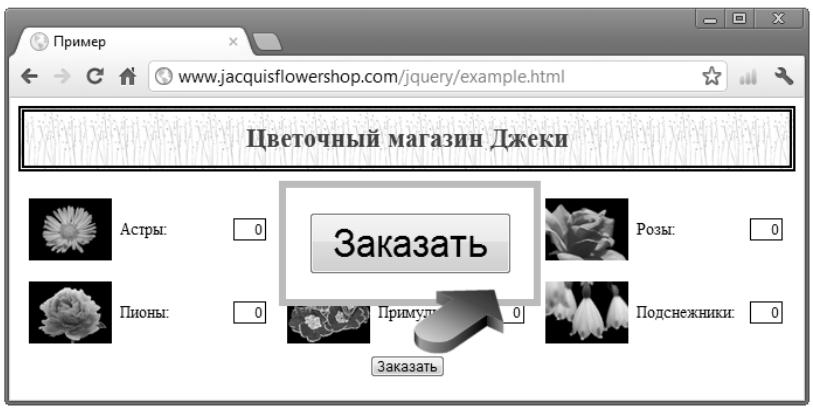

Рис. 2.3. Использование содержимого элемента `button`

Структурные элементы

Вы должны были заметить, что в теле образца документа встречается множество элементов `div`. Этот элемент не несет в себе определенного семантического смысла и часто используется для управления компоновкой содержимого веб-страницы. В данном случае элемент `div` используется для *табличной компоновки* содержимого, так что элементы, содержащиеся в элементах `div`, отображаются для пользователя в виде макетной сетки. Компоновка страницы осуществляется за счет применения к элементам `div` стилей CSS, содержащихся в элементе `style`. Краткие вводные сведения о каскадных таблицах стилей CSS, интенсивно используемых в книге, содержатся в главе 3.

Элементы для работы с внешними ресурсами

Существует ряд элементов, позволяющих включать в документы внешние ресурсы. Хорошим примером может служить элемент `img`, который используют для добавления изображений в документ. В нашем документе он используется для включения в состав содержимого изображений цветов, предлагаемых к продаже, как показано в листинге 2.14.

Листинг 2.14. Использование элемента `img` для добавления ссылки на изображение, хранящееся во внешнем файле

```
...
<img src="snowdrop.png"/>
...
```

Для указания изображения используется атрибут `src`. В данном случае нужное изображение содержится в файле `snowdrop.png`. Мы используем здесь *относительный* URL-адрес, т.е. адрес, определенный относительно URL документа, содержащего данный элемент.

Альтернативой относительным URL-адресам являются *абсолютные* URL-адреса (называемые также *полностью определенными, уточненными* или *полными* URL-адресами). Этот термин относится к URL-адресам, в которых указаны все базовые компоненты, как показано на рис. 2.4. (На рисунке указан также порт, но если он опущен, то браузер будет использовать порт, заданный по умолчанию для данной схемы. Для схемы `http` таковым является порт 80.)

Рис. 2.4. Базовая структура URL

Указание полных URL-адресов для всех требуемых ресурсов может оказаться довольно утомительным занятием, и именно поэтому столь полезны относительные URL-адреса. Указывая `snowdrop.png` в качестве значения атрибута `src` элемента `img`, мы сообщаем браузеру, что он может найти изображения в том же каталоге, в котором находится документ, содержащий элемент `img`. В табл. 2.2 представлены допустимые относительные URL-адреса, а также созданные на их основе абсолютные URL-адреса. Здесь везде предполагается, что загружаемый документ находится по следующему адресу:

```
http://www.jacquisflowershop.com/jquery/example.html
```

Таблица 2.2. Форматы относительных URL-адресов

Относительный URL-адрес	Полный адрес
`snowdrop.png`	`http://www.jacquisflowershop.com/jquery/snowdrop.png`
`/snowdrop.png`	`http://www.jacquisflowershop.com/snowdrop.png`
`/`	`http://www.jacquisflowershop.com/`
`//www.mydomain.com/index.html`	`http://www.mydomain.com/index.htm`

Последний из приведенных в таблице примеров редко используется на практике, поскольку обеспечиваемая им экономия времени при ручном вводе информации незначительна, но он может оказаться полезным, если вы хотите быть уверены в том, что при запросе ресурсов используется та же схема, что и при извлечении основного документа. Это позволяет избежать проблем в тех случаях, когда одна часть содержимого запрашивается по зашифрованному соединению (с использованием схемы `https`), а другая — по незашифрованному (с использованием схемы `http`). Некоторые браузеры, особенно Internet Explorer, плохо воспринимают смешивание безопасного и небезопасного содержимого, и в тех случаях, когда это происходит, выводят для пользователя соответствующее сообщение.

■ **Предупреждение.** Для навигации относительно того каталога, в котором хранится основной HTML-документ на веб-сервере, допустимо использование двух следующих подряд символов точки (`..`). Я не рекомендую применять этот прием хотя бы по той причине, что из соображений безопасности запросы, содержащие эти символы, будут отвергаться многими веб-серверами.

Иерархия элементов

В HTML-документе элементы естественным образом образуют иерархическую структуру. Элемент `html` содержит элемент `body`, который включает в себя элементы содержимого, каждый из которых содержит другие элементы, и т.д.

Знание этой иерархии приобретает особое значение, если вы пытаетесь осуществлять навигацию по документу с помощью стилей CSS (о чем говорится в главе 3) или использовать средства библиотеки jQuery для поиска элементов в документе (что подробно обсуждается в части II).

Наиболее важную часть этой иерархии составляют отношения между элементами. Чтобы упростить описание этих отношений, на рис. 2.5 представлено иерархическое дерево для ряда элементов, содержащихся в документе сайта цветочного магазина.

На этом рисунке изображена лишь часть иерархии элементов нашего документа, которой достаточно для того, чтобы можно было увидеть, что отношения между элементами непосредственно зависят от способа вхождения одних элементов в другие. Существуют различные виды отношений, описанию которых посвящены следующие разделы.

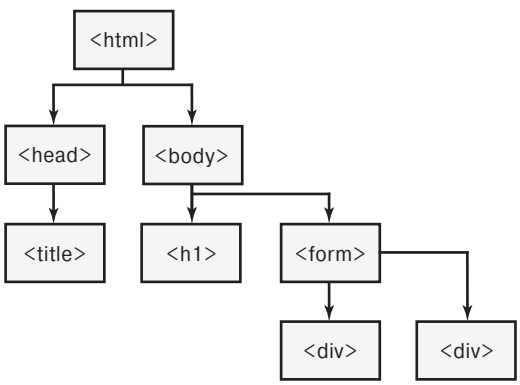

Рис. 2.5. Часть иерархического дерева документа

Отношения “родители–дети”

О существовании отношений “родители–дети” говорят в тех случаях, когда один элемент содержится непосредственно в другом. На приведенном выше рисунке элемент `form` является *дочерним* (child) по отношению к элементу `body`. Это утверждение можно обратить: элемент `body` является *родительским* (parent) по отношению к элементу `form`. У одного элемента может быть несколько дочерних элементов, но родительский элемент может быть у каждого элемента только один. В рассматриваемом примере элемент `body` имеет два дочерних элемента (`form` и `h1`) и является родительским по отношению к каждому из них.

Отношения “родители–дети” существуют только между элементами, одни из которых непосредственно вложены в другие. Так, например, элементы `div` являются дочерними элементами `form`, но не `body`.

Различают некоторые разновидности дочерних отношений. *Первый дочерний элемент* — это тот из дочерних элементов данного элемента, который первым встречается в документе. Например, элемент `h1` является первым дочерним элементом по отношению к элементу `body`. Соответственно *последний дочерний элемент* — это дочерний элемент, который встречается последним. Так, элемент `form` является последним дочерним элементом по отношению к элементу `body`. Также можно говорить об *n-м дочернем элементе* (первому дочернему элементу соответствует n=1).

Отношения “предки–потомки”

К *потомкам* элемента относятся его дочерние элементы, дочерние элементы его дочерних элементов и т.д. Фактически любой элемент, непосредственно или косвенно содержащийся в некотором элементе, является потомком последнего. Например, потомками элемента `body` являются элементы `h1`, `form` и оба элемента `div`, а все изображенные на рисунке элементы являются потомками элемента `html`.

Антиподами потомков являются *предки* элемента, к которым относятся его родительский элемент, родительский элемент его родительского элемента и т.д. Например, предками элемента `form` являются элементы `body` и `html`. Оба элемента `div` имеют один и тот же набор предков: `form`, `body` и `html`.

Сестринские отношения

Сестринскими (sibling) называют отношения, которые существуют между элементами, имеющими общего родителя. В образце документа элементы `h1` и `form` — сестринские, поскольку элемент `body` является родительским по отношению к ним обоим. Работая с сестринскими элементами, мы будем употреблять выражения *предыдущий сестринский элемент* и *следующий сестринский элемент*. Таковыми являются сестринские элементы, которые встречаются в документе соответственно до или после текущего элемента. Не у всех элементов имеются как предыдущий, так и следующий сестринские элементы. Первый и последний дочерние элементы могут иметь соседний элемент лишь одного из этих типов.

Объектная модель документа

В процессе загрузки и обработки HTML-документа браузер создает *объектную модель документа* (Document Object Model — DOM). DOM — это модель, в которой каждый элемент документа представляется объектом JavaScript, но одновременно это и механизм, обеспечивающий возможность взаимодействия с содержимым HTML-документа программными средствами.

■ **Примечание.** В принципе DOM может использоваться с любым языком программирования, реализованным в браузере. В основных типах браузеров преобладает язык JavaScript, и поэтому я не провожу различий между DOM как абстрактной идеей и DOM как коллекцией соответствующих объектов JavaScript.

Одной из причин, по которым вы должны хорошо представлять себе отношения между элементами, описанные в предыдущем разделе, является то, что те же соотношения сохраняются и в DOM. Отсюда следует, что природа и структура документа могут быть исследованы путем обхода узлов DOM-дерева с помощью JavaScript.

■ **Совет.** Использование DOM означает использование JavaScript. Если вам требуется освежить свои знания по JavaScript, обратитесь к главе 4.

Далее демонстрируются некоторые базовые возможности DOM. В остальной части книги для доступа к объектам DOM будет использоваться jQuery, но в данном разделе я познакомлю вас с некоторыми возможностями встроенной поддержки и, в частности, продемонстрирую, насколько элегантнее подход, предлагаемый jQuery.

Использование DOM

В JavaScript объектом, который определяет базовую функциональность, доступную в DOM для всех типов элементов, является объект `HTMLElement`. Объект `HTMLElement` определяет свойства и методы, являющиеся общими для всех типов HTML-элементов, включая свойства, приведенные в табл. 2.3.

Таблица 2.3. Базовые свойства объекта `HTMLElement`

Свойство	Описание	Тип возвращаемого значения
`className`	Возвращает или задает список классов, которым принадлежит данный элемент	`string`
`id`	Возвращает или задает значение атрибута `id`	`string`
`lang`	Возвращает или задает значение атрибута `lang`	`string`
`tagName`	Возвращает имя дескриптора (указывающее тип элемента)	`string`

Общее число доступных свойств гораздо больше. Их точное количество зависит от используемой версии HTML. Но указанных четырех свойств вполне достаточно для того, чтобы продемонстрировать базовые возможности DOM.

Для представления уникальных характеристик элементов каждого типа DOM использует объекты, получаемые путем наследования объекта `HTMLElement`. Например, объект `HTMLImageElement`, используемый для представления в DOM элементов `img`, определяет свойство `src`, которому соответствует атрибут `src` элемента `img`. Я не буду углубляться в детальное описание всех объектов, соответствующих конкретным элементам, а лишь замечу, что, как правило, можно рассчитывать на то, что для интересующего вас атрибута всегда найдется соответствующее свойство объекта DOM.

Доступ к DOM осуществляется через глобальную переменную `document`, возвращающую объект `Document`. Объект `Document` представляет отображаемый в браузере HTML-документ и определяет ряд методов, обеспечивающих поиск объектов в DOM (табл. 2.4).

Таблица 2.4. Методы объекта `Document`, предназначенные для поиска элементов

Метод	Описание	Тип возвращаемого значения
`getElementById(<id>)`	Возвращает элемент с указанным значением `id`	`HTMLElement`
`getElementsByClassName(<класс>)`	Возвращает элементы с указанным значением класса	`HTMLElement[]`
`getElementByTagName(<дескриптор>)`	Возвращает элементы указанного типа	`HTMLElement[]`
`querySelector(<селектор>)`	Возвращает первый найденный элемент, соответствующий указанному селектору CSS	`HTMLElement`
`querySelectorAll(<селектор>)`	Возвращает все элементы, соответствующие указанному селектору CSS	`HTMLElement[]`

Вновь повторюсь, что здесь приведены лишь методы, которые используются в данной книге. В двух последних методах, представленных в таблице, используются селекторы CSS, описанные в главе 3. Пример использования объекта `Document` для поиска элементов определенного типа в документе приведен в листинге 2.15.

Листинг 2.15. Поиск элементов в DOM

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример</title>
    <script src="jquery-2.1.0.js" type="text/javascript"></script>
    <style>
        h1 {
            width: 700px; border: thick double black;
            margin-left: auto; margin-right: auto;
            text-align: center; font-size: x-large; padding: .5em;
            color: darkgreen; background-image: url("border.png");
            background-size: contain; margin-top: 0;
        }
        .dtable {display: table;}
        .drow {display: table-row;}
        .dcell {display: table-cell; padding: 10px;}
        .dcell > * {vertical-align: middle}
        input {width: 2em; text-align: right;
               border: thin solid black; padding: 2px;}
        label {width: 5em;  padding-left: .5em;
               display: inline-block;}
        #buttonDiv {text-align: center;}
        #oblock {display: block; margin-left: auto;
                 margin-right: auto; width: 700px;}
    </style>
</head>
<body>
```

```
    <h1>Цветочный магазин Джеки</h1>
    <form method="post">
        <div id="oblock">
            <div class="dtable">
                <div class="drow">
                    <div class="dcell">
                        <img src="aster.png"/>
                        <label for="aster">Астры:</label>
                        <input name="aster" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="daffodil.png"/>
                        <label for="daffodil">Нарциссы:</label>
                        <input name="daffodil" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="rose.png"/>
                        <label for="rose">Розы:</label>
                        <input name="rose" value="0" required>
                    </div>
                </div>
                <div class="drow">
                    <div class="dcell">
                        <img src="peony.png"/>
                        <label for="peony">Пионы:</label>
                        <input name="peony" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="primula.png"/>
                        <label for="primula">Примулы:</label>
                        <input name="primula" value="0" required>
                    </div>
                    <div class="dcell">
                        <img src="snowdrop.png"/>
                        <label for="snowdrop">Подснежники:</label>
                        <input name="snowdrop" value="0" required>
                    </div>
                </div>
            </div>
        </div>
        <div id="buttonDiv">
            <button type="submit">Заказать</button>
        </div>
    </form>
    <script>
        var elements = document.getElementsByTagName("img");
        for (var i = 0; i < elements.length; i++) {
            console.log("Элемент: " + elements[i].tagName +
                " " + elements[i].src);
        }
    </script>
</body>
</html>
```

В этом примере элемент `script` добавляется в конце элемента `body`. Когда браузер встречает элемент `script`, он сразу же приступает к выполнению операторов JavaScript, не дожидаясь окончания загрузки и обработки остальной части документа. Это становится проблемой, когда вы работаете с DOM, потому что вы пытаетесь выполнить поиск элементов посредством объекта `Document` еще до того, как в модели созданы интересующие вас объекты. Чтобы избежать этого, я поместил элемент `script` в конце документа. Библиотека jQuery предлагает элегантный способ решения этой проблемы, описанный в части II.

Для поиска в документе всех элементов `img` в сценарии используется метод `getElementsByTagName()`. Этот метод возвращает массив объектов, которые перебираются в цикле для вывода на консоль значений свойств `tagName` и `src` каждого объекта. Выводимая на консоль информация имеет следующий вид.

```
Элемент: IMG http://www.jacquisflowershop.com/jquery/aster.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/daffodil.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/rose.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/peony.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/primula.png
Элемент: IMG http://www.jacquisflowershop.com/jquery/snowdrop.png
```

Изменение DOM

Объекты DOM *активны* в том смысле, что изменение значения свойства объекта оказывает влияние на документ, отображаемый в браузере. Сценарий, позволяющий продемонстрировать этот эффект, приведен в листинге 2.16. (Чтобы избежать ненужного повторения большей части документа, в листинге представлен лишь элемент `script`. Остальная часть документа остается той же, что и в предыдущем примере.)

Листинг 2.16. Изменение свойства DOM-объекта

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        elements[i].src = "snowdrop.png";
    }
</script>
...
```

В этом сценарии значение атрибута `src` для всех элементов `img` устанавливается равным `snowdrop.png`. Результат выполнения сценария показан на рис. 2.6.

Изменение стилей

DOM можно использовать для изменения значений свойств CSS. (Краткое введение в CSS содержится в главе 3.) В программном интерфейсе DOM предусмотрена основательная поддержка CSS, но проще всего это можно сделать, используя свойство `style` объекта `HTMLElement`. Свойства объекта, возвращаемого свойством `style`, соответствуют свойствам CSS (я осознаю, что слово “свойство” встречается в этом коротком предложении слишком уж часто, за что приношу свои извинения).

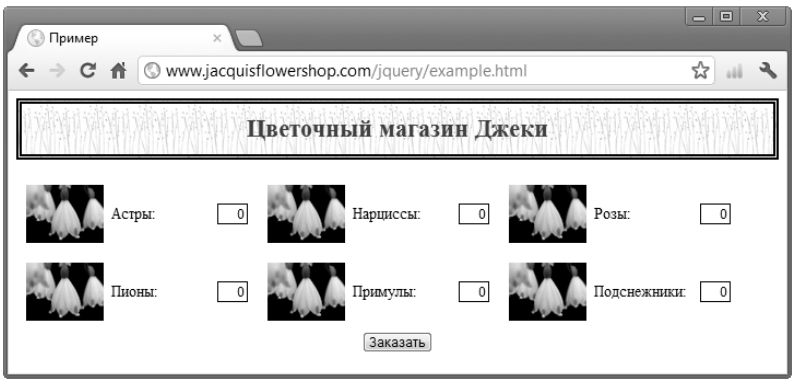

Рис. 2.6. Использование DOM для изменения HTML-документа

Принятые в CSS правила присвоения имен свойствам несколько отличаются от тех, которые используются в объекте, возвращаемом свойством `style`. Например, свойству CSS `background-color` соответствует свойство `style.backgroundColor` этого объекта. Управление стилями документа с помощью DOM продемонстрировано в листинге 2.17.

Листинг 2.17. Использование DOM для изменения стилей элементов

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        if (i > 0) {
            elements[i].style.opacity = 0.5;
        }
    }
</script>
...
```

Этот сценарий изменяет прозрачность, точнее — значение параметра непрозрачности (`opacity`), всех элементов `img` в документе, кроме первого. Один элемент `img` оставлен в неизменном виде, чтобы разница во внешнем виде элементов больше бросалась в глаза (рис. 2.7).

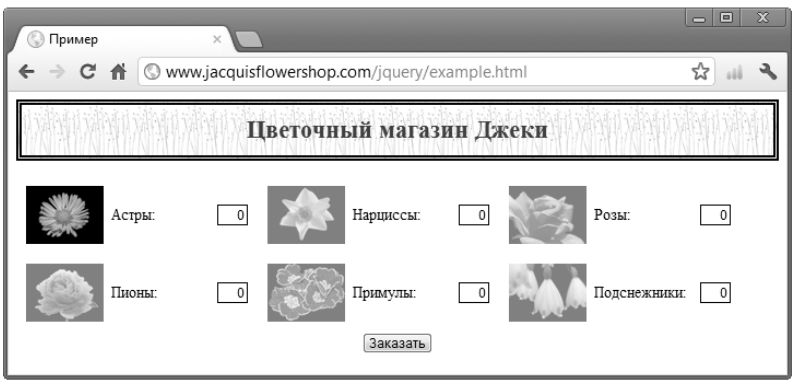

Рис. 2.7. Использование JavaScript для изменения значений свойств CSS

Обработка событий

События — это посылаемые браузером сигналы, уведомляющие об изменении состояния одного или нескольких объектов DOM. Для различных типов изменения состояний предусмотрены различные события. Например, после щелчка на кнопке запускается событие `click`, а после отправки формы — событие `submit`. Многие события взаимосвязаны. Так, событие `mouseover` запускается при наведении указателя мыши на элемент, а событие `mouseout` — при выходе указателя мыши за пределы элемента. Можно отреагировать на события нужным образом, связав с событием для DOM-элемента некоторую *функцию-обработчик* JavaScript. Соответствующий пример приведен в листинге 2.18.

Листинг 2.18. Обработка события

```
...
<script>
    var elements = document.getElementsByTagName("img");
    for (var i = 0; i < elements.length; i++) {
        elements[i].onmouseover = handleMouseOver;
        elements[i].onmouseout = handleMouseOut;
    }

    function handleMouseOver(e) {
        e.target.style.opacity = 0.5;
    }

    function handleMouseOut(e) {
        e.target.style.opacity = 1;
    }
</script>
...
```

В этом сценарии определены два обработчика событий, которые задаются значениями свойств `onmouseover` и `onmouseout` DOM-объектов `img`. В результате работы сценария изображения, представленные в документе, будут становиться частично прозрачными при наведении на них указателя мыши и восстанавливать свой обычный вид при перемещении указателя мыши за пределы области изображения. Пока что мы не будем обсуждать механизм обработки событий, используемый в DOM, поскольку подробному рассмотрению поддержки событий в jQuery посвящена глава 9. Тем не менее вам будет полезно уже сейчас получить предварительное представление об объекте `Event`, который передается обработчикам событий. Наиболее важные свойства этого объекта приведены в табл. 2.5.

Таблица 2.5. Функции и свойства объекта `Event`

Имя	Описание	Тип возвращаемого значения
`type`	Имя события, например `mouseover`	`string`
`target`	Целевой элемент события	`HTMLElement`
`currentTarget`	Элемент, приемники событий которого в данный момент активизируются	`HTMLElement`
`eventPhase`	Текущая фаза жизненного цикла события	`number`

Окончание табл. 2.5

Имя	Описание	Тип возвращаемого значения
`bubbles`	Возвращает значение `true`, если событие является всплывающим; в противном случае — значение `false`	`boolean`
`cancelable`	Возвращает значение `true`, если для события предусмотрено действие по умолчанию, которое можно отменить; в противном случае — значение `false`	`boolean`
`stopPropagation()`	Предотвращает распространение события по дереву элементов после запуска приемников событий для текущего элемента	`void`
`stopImmediatePropagation()`	Немедленно прекращает распространение события вверх или вниз по дереву элементов. Незапущенные приемники событий для текущего элемента игнорируются	`void`
`preventDefault()`	Предотвращает выполнение браузером действия по умолчанию, связанного с событием	`void`
`defaultPrevented`	Возвращает `true`, если перед этим вызывалась функция `preventDefault()`	`boolean`

В последнем примере элемент, к которому относится событие, определяется с помощью свойства `target`. Некоторые другие члены объекта `Event` связаны с *потоком события* и *действиями по умолчанию*, о чем пойдет речь (очень кратко) в двух следующих разделах. В данной главе лишь закладывается фундамент для более глубокого обсуждения этой темы в последующих главах.

Поток события

Жизненный цикл события включает три фазы: *захват события* (capture), *передача события цели* (target) и *всплытие события* (bubbling). Когда запускается событие, браузер определяет элемент, к которому оно относится, — так называемый *целевой элемент* (цель) события. Далее браузер определяет все элементы, располагающиеся в иерархической структуре DOM-дерева между элементом `body` и целевым элементом события, и выясняет в отношении каждого из них, имеют ли они обработчики событий, которые должны уведомляться о событиях, относящихся к подчиненным элементам (потомкам). Любой такой обработчик будет запускаться браузером до запуска обработчиков целевого объекта события. (О том, как выполнить запрос уведомления о событиях подчиненных элементов, говорится в главе 9.)

За фазой захвата следует фаза передачи события целевому элементу — простейшая из трех фаз. После завершения фазы захвата браузер запускает приемники событий данного типа, которые были добавлены в целевой элемент.

По завершении фазы передачи события целевому элементу браузер начинает последовательно просматривать всю цепочку элементов более высокого уровня (предков), продвигаясь в направлении элемента `body`. В ходе этого процесса браузер проверяет, существуют ли для каждого из просматриваемых элементов приемники событий данного типа, для которых захват событий отключен (подробнее об этом говорится в главе 9). Всплытие поддерживается не всеми событиями. Используя свойство `bubbles`, можно проверить, будет ли событие всплывать. Если значение этого свойства равно `true`, событие будет всплывать, в противном случае — не будет.

Действия браузера по умолчанию

Некоторые события определяют действие по умолчанию, которое будет выполняться в случае запуска события. Например, действием по умолчанию для события `click` является загрузка содержимого, местонахождение которого задано URL-адресом в атрибуте `href` элемента, на котором выполнен щелчок. Если для события предусмотрено действие по умолчанию, то значением свойства `cancelable` этого события будет `true`. Выполнение действия по умолчанию можно отменить, вызвав метод `preventDefault()`. Заметьте, что вызов метода `preventDefault()` не приводит к прекращению процесса прохождения события через фазы захвата события, его передачи целевому объекту и всплытия. Эти фазы по-прежнему будут выполняться, но по окончании фазы всплытия браузер не выполнит действие, предусмотренное по умолчанию. Можно определить, была ли вызвана для события функция `preventDefault()` одним из предшествующих обработчиков событий, получив значение свойства `defaultPrevented`. Если данное значение окажется равным `true`, то это будет означать, что функция `preventDefault()` вызывалась.

Резюме

В этой главе был приведен лишь краткий обзор принципов работы HTML, поскольку подробное описание каждого из более чем 100 элементов, которые могут входить в состав HTML-документа, в рамках данной книги не представляется возможным. Было показано, как создается и структурируется простой HTML-документ, как в результате вложения в элементы не только текстового содержимого, но и других элементов естественным образом возникает иерархия элементов и как в этой иерархии устанавливаются отношения между элементами, в соответствии с которыми элементы могут выступать один по отношению к другому в роли родительских, дочерних и сестринских элементов, а также элементов-предков и элементов-потомков. Также было показано, как работает классический DOM API и как с его помощью можно выбирать элементы и обрабатывать событий. Это было сделано намеренно в целях сравнения обеих методик, поскольку, в чем вы сами неоднократно убедитесь на протяжении всей книги, одной из главных причин использования jQuery является то, что эта библиотека эффективно скрывает технические детали DOM API и значительно упрощает и облегчает работу с HTML-элементами и представляющими их объектами JavaScript.