HIGHTECH

БЕР БИБО
ИЕГУДА КАЦ

jQuery

Подробное руководство по продвинутому JavaScript

Символ®

jQuery
in Action

Bear Bibeault, Yehuda Katz

HIGH TECH

jQuery

Подробное руководство по продвинутому JavaScript

Бер Бибо, Иегуда Кац

Санкт-Петербург — Москва
2009

Серия «High tech»

Бер Бибо, Иегуда Кац

jQuery. Подробное руководство по продвинутому JavaScript

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Научный редактор	*Б. Попов*
Редактор	*Т. Темкина*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Бер Бибо, Иегуда Кац

jQuery. Подробное руководство по продвинутому JavaScript. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 384 с., ил.

ISBN-10: 5-93286-135-5

ISBN-13: 978-5-93286-135-6

Издание представляет собой введение в jQuery – мощную платформу для разработки веб-приложений. Ее уникальная способность составлять «цепочки» из команд позволяет выполнять несколько последовательных операций над элементами страницы. В книге подробно описано как выполнять обход документов HTML, обрабатывать события, воспроизводить анимацию и добавлять поддержку технологии Ajax в свои веб-страницы. Изучение каждой новой концепции закрепляется на практических примерах. Рассмотрены вопросы взаимодействия jQuery с другими инструментами и платформами и методы создания модулей расширения для этой библиотеки.

Книга предназначена для разработчиков, знакомых с языком JavaScript и технологией Ajax и стремящихся создавать краткий и понятный программный код – сократить его в несколько раз позволит грамотное использование библиотеки jQuery.

ISBN-10: 5-93286-135-5
ISBN-13: 978-5-93286-135-6
ISBN: 1-933988-35-5 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 30.10.2008. Формат 70x100 1/16. Печать офсетная.
Объем 24 печ. л. Тираж 2000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

В этой главе:

- Для чего нужна библиотека jQuery
- Что такое «ненавязчивый JavaScript»
- Основы jQuery
- jQuery и другие библиотеки JavaScript

1

Введение в jQuery

Большую часть своей жизни считавшийся среди серьезных веб-разработчиков «игрушечным», язык JavaScript обрел былой авторитет на волне интереса к полнофункциональным интернет-приложениям и технологиям Ajax. Языку пришлось очень быстро повзрослеть, так как разработчики клиентских сценариев отказались от приема копирования и вставки программного кода JavaScript в пользу полноценных библиотек, которые позволяют решать сложные проблемы, связанные с различиями между броузерами разных типов, и реализуют новые и улучшенные парадигмы разработки веб-приложений.

Несколько запоздало появившаяся в мире библиотек JavaScript jQuery покорила сообщество веб-разработчиков, быстро завоевав поддержку крупных сайтов, таких как MSNBC, и была высоко оценена такими открытыми проектами, как SourceForge, Trac и Drupal.

В отличие от других инструментов, сконцентрированных на применении сложных методик JavaScript, jQuery стремится изменить представление веб-разработчиков о принципах создания полнофункциональных интернет-приложений. Вместо того чтобы тратить время на жонглирование непростыми возможностями языка JavaScript, разработчики могли бы, применив знание каскадных таблиц стилей (Cascading Style Sheets, CSS), расширенного языка разметки гипертекста (eXtensible Hypertext Markup Language, XHTML) и старого доброго JavaScript, напрямую манипулировать элементами страницы, воплотив мечту о быстрой разработке веб-приложений.

В этой книге мы во всех подробностях рассмотрим, что может предложить jQuery нам как авторам полнофункциональных интернет-приложений для страниц. Для начала узнаем, что в действительности может дать jQuery при разработке веб-страниц.

1.1. Почему jQuery?

Потратив некоторое время на попытки привнести динамическую функциональность в ваши страницы, вы обнаружите, что постоянно следуете одному и тому же шаблону: сначала отбирается элемент или группа элементов, а затем над ними выполняются некоторые действия: вы могли бы скрывать или показывать элементы, добавлять к ним класс CSS, создавать анимационные эффекты или изменять атрибуты.

С обычным JavaScript для решения каждой из задач потребуются десятки строк программного кода. Создатель jQuery разработал свою библиотеку именно для того, чтобы сделать наиболее общие задачи тривиальными. Например, чтобы создать таблицу с разным цветом фона для четных и нечетных строк, дизайнеру потребуется написать до 10 строк кода на языке JavaScript. А вот как тот же эффект достигается с помощью jQuery:

```
$("table tr:nth-child(even)").addClass("striped");
```

Не волнуйтесь, если сейчас эта строка кажется вам странной. Вскоре вы поймете, как она работает, и сами будете использовать короткие, но мощные инструкции jQuery, чтобы добавить живости своим страницам. Давайте коротко рассмотрим, как работает этот фрагмент кода.

Эта инструкция отыскивает все четные строки (элемент `<tr>`) во всех таблицах на странице и добавляет к ним класс CSS `striped`. Применяя желаемый цвет фона к этим строкам через правило CSS класса `striped`, единственная строка JavaScript может улучшить эстетику всей страницы.

В результате таблицы на странице будут выглядеть, как показано на рис. 1.1.

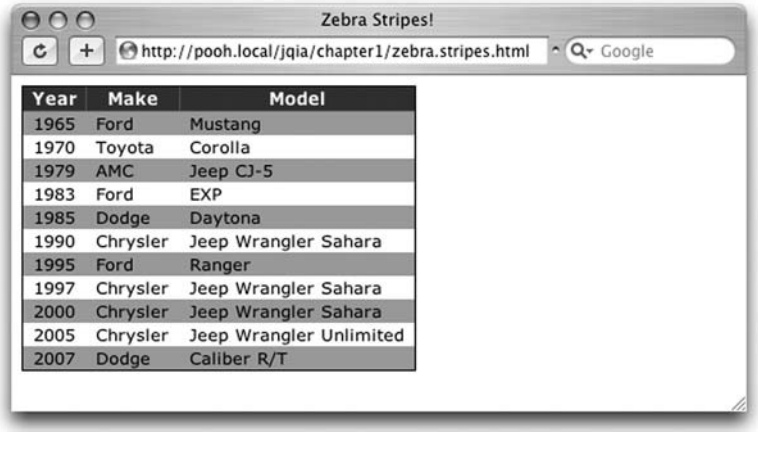

Рис. 1.1. *Библиотека jQuery позволяет добавить расцветку строк таблицы с помощью единственной инструкции!*

Истинная мощь этой инструкции jQuery заложена в *селекторе* – выражении идентификации элементов, позволяющем идентифицировать и отбирать нужные элементы страницы. В данном случае – каждый четный элемент `<tr>` во всех таблицах. Полный исходный код этой страницы вы найдете среди загружаемых примеров этой книги, в файле *chapter1/zebra.stripes.html*.

Позднее мы узнаем, как легко создавать эти селекторы, но сначала посмотрим, как изобретатели jQuery представляют себе эффективное использование JavaScript в наших страницах.

1.2. Ненавязчивый JavaScript

Помните суровые времена до появления CSS, когда мы были вынуждены смешивать в HTML-страницах стилевую разметку со структурой документа?

Эти воспоминания почти наверняка заставят содрогнуться любого, кто создавал тогда страницы. Добавление CSS в арсенал инструментов веб-разработчика позволяет отделить стилистическую информацию от структуры документа и проводить на заслуженный отдых такие теги, как `<font>`. Отделение стиля от структуры не только упрощает управление документами, но и придает им гибкость полного изменения стиля отображения страницы простой заменой таблицы стилей.

Немногие из нас захотели бы вернуться в прошлое, когда стили отображения применялись непосредственно к HTML-элементам. И все же до сих пор очень популярна разметка вроде этой:

```
<button
  type="button"
  onclick="document.getElementById('xyz').style.color='red';">
    Click Me
</button>
```

Здесь видно, что стиль элемента-кнопки, включая шрифт надписи на ней, определяется не тегом `<font>` или другой нежелательной разметкой стиля, а задается правилами CSS, действующими в пределах страницы. В этом объявлении не смешивается разметка стиля и структура, однако здесь налицо смешение *поведения* и структуры за счет включения кода JavaScript, выполняемого по щелчку на кнопке, в код разметки элемента `<button>` (в данном случае щелчок на кнопке вызывает окрашивание в красный цвет некоторого элемента объектной модели документа (Document Object Model, DOM) с именем `xyz`).

По тем же причинам, по которым желательно отделять стиль от структуры HTML-документа, точно так же (если не более) желательно было бы отделить *поведение* элементов от структуры.

Такого рода отделение известно под названием *ненавязчивый JavaScript*, и разработчики jQuery сосредоточили свои усилия на том, чтобы

помочь авторам страниц следовать ему. С точки зрения концепции ненавязчивого JavaScript и библиотеки jQuery наличие выражений или инструкций на языке JavaScript в теге `<body>` HTML-страниц, в атрибутах HTML-элементов (например, `onclick`) либо в блоках сценариев в теле страницы считается неправильным.

Вы можете спросить: «Как же пользоваться кнопкой без атрибута `onclick`?» Рассмотрим такое определение кнопки:

```
<button type="button" id="testButton">Click Me</button>
```

Оно гораздо проще! Но теперь кнопка просто *ничего не делает*.

Вместо того чтобы встраивать определение поведения кнопки в ее разметку, мы поместим его в блок сценария в разделе `<head>` страницы, за пределами тела документа, как показано ниже:

```
<script type="text/javascript">
  window.onload = function() {
    document.getElementById('testButton').onclick = makeItRed;
  };

  function makeItRed() {
    document.getElementById('xyz').style.color = 'red';
  }
</script>
```

В обработчике события страницы `onload` мы связываем функцию `makeItRed()` с атрибутом `onclick` элемента-кнопки. Обработчик события `onload` (а не встроенный код) используется потому, что нам нужно, чтобы элемент-кнопка уже *существовал* к тому моменту, как мы попытаемся манипулировать им. (В разд. 1.3.3 будет показано, что jQuery предоставляет лучший способ размещения такого программного кода.)

Если в этом примере что-то покажется вам непонятным, не пугайтесь! В приложении приведен обзор понятий JavaScript, которые помогут вам эффективно использовать jQuery. Кроме того, в оставшейся части главы мы узнаем, как jQuery позволяет писать код, аналогичный предыдущему, более короткий, более простой и одновременно более гибкий.

Ненавязчивый JavaScript – это мощная методика, позволяющая разделить обязанности в веб-приложениях, но за это приходится платить. Возможно, вы уже обратили внимание, что для достижения поставленной цели нам потребовалось добавить немного больше строк кода, чем в случае, когда код JavaScript помещался непосредственно в код разметки. Ненавязчивый JavaScript не только может привести к увеличению объема программного кода, но также требует определенной дисциплины и применения хорошо зарекомендовавших себя шаблонов программирования клиентских сценариев.

Но в этом нет ничего плохого – все, что побуждает нас с заботой и уважением писать свой клиентский код, обычно размещаемый на сервере, нам только на пользу. Но без jQuery для этого пришлось бы *выполнить* лишнюю работу.

Как уже говорилось, разработчики jQuery сосредоточили свои усилия на том, чтобы упростить нам применение методик ненавязчивого JavaScript при программировании страниц, не платя за это лищним программным кодом. Мы считаем, что эффективное использование jQuery позволяет нам добиться больших возможностей за счет меньшего объема сценариев.

А теперь спокойно рассмотрим, как библиотека jQuery позволяет расширять функциональность страниц без лишних усилий и головной боли.

1.3. Основы jQuery

Суть jQuery в том, чтобы отбирать элементы HTML-страниц и выполнять над ними некоторые операции. Если вы знакомы с CSS, то хорошо понимаете, насколько удобны селекторы, которые описывают группы элементов, объединенные по каким-либо атрибутам или по местоположению в документе. Благодаря jQuery вы сможете, используя свои знания, значительно упростить код JavaScript.

Библиотека jQuery в первую очередь обеспечивает непротиворечивую работу программного кода во всех основных типах броузеров, решая такие сложные проблемы JavaScript, как ожидание загрузки страницы перед тем, как выполнить какие-либо операции.

На тот случай, если в библиотеке обнаружится недостаток функциональности, разработчики предусмотрели простой, но весьма действенный способ ее расширения. Многие начинающие программисты jQuery обнаруживают эту гибкость на практике, расширяя возможности jQuery в первый же день.

Но для начала давайте посмотрим, как знание CSS помогает создать краткий, но мощный программный код.

1.3.1. Обертка jQuery

С введением CSS в веб-технологии для отделения представления от содержимого понадобился способ, позволяющий ссылаться на группы элементов страницы из внешних таблиц стилей. В результате был разработан метод, основанный на использовании *селекторов*, которые представляют элементы на основе их атрибутов или местоположения в HTML-документе.

Например, селектор

```
p a
```

ссылается на все ссылки (элементы <a>), вложенные в элементы <p>. Библиотека jQuery использует те же самые селекторы и поддерживает не только обычные селекторы, применяемые сегодня в CSS, но и другие, еще не полностью реализованные в большинстве броузеров. Селектор

`nth-child` из примера с полосатой таблицей, приведенного выше, – отличный пример применения селектора, определенного в спецификации CSS3.

Синтаксис отбора группы элементов прост:

```
$(selector)
```

или

```
jQuery(selector)
```

Функция `$()`, на первый взгляд, необычна, но большинство пользователей быстро начинают применять ее благодаря ее краткости.

Например, чтобы получить группу ссылок, вложенных в элементы `<p>`, можно использовать следующий код:

```
$("p a")
```

Функция `$()` (псевдоним функции `jQuery()`) возвращает специальный объект JavaScript, который содержит массив элементов DOM, соответствующих указанному селектору. У этого объекта много удобных предопределенных методов, способных воздействовать на группу элементов.

На языке программирования такого рода конструкция называется *оберткой* (*wrapper*), потому что она «обертывает» отобранные элементы дополнительной функциональностью. Мы будем использовать термин *обертка jQuery*, или *обернутый набор*, ссылаясь на группы элементов, управлять которыми позволяют методы, определенные в jQuery.

Предположим, нам требуется реализовать постепенное исчезновение всех элементов `<div>` с классом CSS `notLongForThisWorld`. jQuery позволяет сделать это так:

```
$("div.notLongForThisWorld").fadeOut();
```

Особенность многих из этих методов, часто называемых *командами* jQuery, состоит в том, что по завершении своих действий (например, действия, обеспечивающего постепенное исчезновение) они возвращают ту же самую группу элементов, готовую к выполнению другой операции. Предположим, что после того, как элементы исчезнут, к ним нужно добавить класс CSS `removed`. Записать это можно так:

```
$("div.notLongForThisWorld").fadeOut().addClass("removed");
```

Такую *цепочку* (*chain*) команд jQuery можно продолжать до бесконечности. Вы без труда сможете отыскать в Интернете примеры цепочек jQuery, состоящих из десятков команд. А поскольку каждая функция работает сразу со всеми элементами, соответствующими указанному селектору, вам не потребуется выполнять обход массива элементов в цикле. Все нужное происходит за кулисами!

Но даже при том, что группа отобранных элементов представлена довольно сложным объектом JavaScript, в случае необходимости мы мо-

жем обращаться к ней как к обычному массиву элементов. В итоге следующие две инструкции дают идентичные результаты:

```
$("#someElement").html("Добавим немного текста");
```

или

```
$("#someElement")[0].innerHTML =
  "Добавим немного текста";
```

Поскольку здесь мы использовали селектор по атрибуту ID, ему будет соответствовать один элемент. В первом случае используется метод библиотеки jQuery – `html()`, который замещает содержимое элемента DOM некоторой HTML-разметкой. Во втором случае с помощью jQuery извлекается массив элементов, выбирается первый элемент массива с индексом `0` и затем его содержимое замещается с помощью самого обычного метода JavaScript.

Если мы захотим получить тот же результат с помощью селектора, который может отобрать несколько элементов, то можно использовать любой из двух следующих способов, дающих идентичные результаты:

```
$("div.fillMeIn")
  .html("Добавим немного текста в группу элементов");
```

или

```
var elements = $("div.fillMeIn");
for(i=0;i<elements.length;i++)
  elements[i].innerHTML =
    "Добавим немного текста в группу элементов";
```

С увеличением сложности поставленных задач способность jQuery создавать цепочки команд по-прежнему способствует уменьшению числа строк программного кода, необходимого для получения желаемых результатов. Библиотека jQuery поддерживает не только селекторы, которые вы уже знаете и любите, но и более сложные селекторы, определенные как часть спецификации CSS, и даже некоторые нестандартные селекторы.

Вот несколько примеров.

```
$("p:even");
```

Этот селектор отбирает все четные элементы `<p>`.

```
$("tr:nth-child(1)");
```

Этот селектор отбирает первые строки во всех таблицах.

```
$("body > div");
```

Этот селектор отбирает элементы `<div>`, являющиеся прямыми потомками элемента `<body>`.

```
$("a[href$=pdf]");
```

Этот селектор отбирает ссылки на файлы PDF.

```
$("body > div:has(a)")
```

Этот селектор отбирает элементы `<div>`, которые являются прямыми потомками элемента `<body>` и содержат ссылки.

Мощная штука!

Для начала вы можете использовать свое знание CSS, а затем изучите более сложные селекторы, поддерживаемые библиотекой. Очень подробно селекторы будут рассматриваться в разд. 2.1, а их полный перечень вы найдете по адресу *http://docs.jquery.com/Selectors*.

Выбор элементов DOM для выполнения операций – это самое обычное дело для наших страниц, но в некоторых случаях требуется выполнить какие-либо действия, никак не связанные с элементами DOM. Давайте коротко рассмотрим, что еще может предложить jQuery помимо манипулирования элементами.

1.3.2. Вспомогательные функции

Даже при том, что обертывание элементов для выполнения операций над ними – основное применение функции `$()` в библиотеке jQuery, это не единственная ее возможность. Одна из ее дополнительных возможностей – выступать в качестве префикса *пространства имен* (*namespace*) для вспомогательных функций общего назначения. Обертка jQuery, получаемая в результате вызова `$()` с селектором, предоставляет авторам страниц максимум возможностей, поэтому большинство авторов страниц использует другие возможности, предоставляемые вспомогательными функциями, сравнительно редко. Мы не будем подробно рассматривать эти функции до главы 6, где описаны подготовительные действия для создания подключаемых модулей jQuery. Но некоторые из этих функций *встретятся* вам в следующих разделах, поэтому мы опишем их здесь.

Поначалу способ обращения к этим функциям может казаться немного странным. Давайте рассмотрим пример использования вспомогательной функции, которая усекает строки. Вызов этой функции выглядит так:

```
$.trim(someString);
```

Если префикс `$.` кажется вам странным, запомните, что `$` – это обычный идентификатор JavaScript, такой же, как и любой другой. Вызов той же самой функции с использованием идентификатора `jQuery` выглядит более знакомым:

```
jQuery.trim(someString);
```

Здесь совершенно очевидно, что функция `trim()` принадлежит пространству имен `jQuery` с псевдонимом `$`.

Примечание

В документации эти элементы называются вспомогательными *функциями*, но совершенно очевидно, что в действительности они являются *методами* функции `$()`. Не углубляясь в технические детали, мы также будем называть эти методы *вспомогательными функциями*, чтобы терминология соответствовала электронной документации.

Одну такую вспомогательную функцию, позволяющую расширять jQuery, мы рассмотрим в разд. 1.3.5, а другую, позволяющую jQuery мирно сосуществовать с другими клиентскими библиотеками, – в разд. 1.3.6. Но для начала рассмотрим еще один важный аспект функции `$`.

1.3.3. Обработчик готовности документа

Методика ненавязчивого JavaScript позволяет отделить поведение элементов от структуры документа, поэтому мы будем манипулировать с элементами страницы за пределами разметки документа, где создаются эти элементы. Для этого нам нужен механизм, позволяющий дождаться окончания загрузки элементов DOM страницы и только после этого выполнить необходимые операции. В примере с полосатой таблицей мы должны дождаться момента, когда будет загружена вся таблица, и только потом изменить цвет фона ее строк.

Традиционно для этих целей используется обработчик `onload` объекта `window`, который выполняет инструкции после того, как страница будет загружена целиком. Обычно он вызывается так:

```
window.onload = function() {
  $("table tr:nth-child(even)").addClass("even");
};
```

Тем самым программный код раскрашивания таблицы вызывается только после того, как документ полностью загружен. К сожалению, броузер задерживает выполнение обработчика `onload` не только до момента создания полного дерева DOM, но также ждет, пока будут загружены все изображения и другие внешние ресурсы и страница отобразится в окне броузера. В результате посетитель может заметить задержку между тем моментом, когда он впервые увидит страницу, и тем, когда будет выполнен сценарий `onload`.

Хуже того, если изображение или другой ресурс загружается достаточно долго, посетитель вынужден ждать окончания его загрузки, прежде чем дополнительные особенности поведения элементов станут доступными. Во многих реальных применениях это могло бы обречь идею ненавязчивого JavaScript на неудачу.

Намного лучший подход заключается в том, чтобы задерживать запуск сценариев, обеспечивающих дополнительные особенности поведения, *только* до момента окончания загрузки структуры документа,

когда HTML-код страницы будет преобразован броузером в дерево DOM. Добиться этого независимым от типа броузера способом достаточно сложно, но библиотека jQuery предоставляет простой способ запуска программного кода сразу после загрузки дерева DOM, но до загрузки внешних изображений. Формальный синтаксис определения такого кода (из примера с таблицей):

```
$(document).ready(function() {
  $("table tr:nth-child(even)").addClass("even");
});
```

Сначала производится обертывание экземпляра документа в функцию `jQuery()`, а затем применяется метод `ready()`, которому функция передается для исполнения после того, как документ станет доступным для манипуляций.

Этот синтаксис мы назвали *формальным* потому, что гораздо чаще используется сокращенная форма его записи:

```
$(function() {
  $("table tr:nth-child(even)").addClass("even");
});
```

Вызывая функцию `$()`, мы тем самым предписываем броузеру дождаться, пока дерево DOM (и только дерево DOM) будет полностью загружено, прежде чем выполнить этот код. Более того, в одном и том же HTML-документе мы можем применять этот прием многократно, и броузер выполнит все указанные функции в порядке следования объявлений. Напротив, методика на базе обработчика `onload` объекта `window` позволяет указать единственную функцию. Это ограничение чревато трудноуловимыми ошибками, если какой-либо сторонний сценарий использует механизм `onload` для собственных нужд (что никак нельзя признать хорошей практикой).

Мы познакомились со вторым назначением функции `$()`. Давайте посмотрим, что еще она может нам предложить.

1.3.4. Создание элементов DOM

Совершенно очевидно, что авторы jQuery избегали вводить глобальные идентификаторы в пространство имен JavaScript, сделав функцию `$()` (которая является обычным псевдонимом функции `jQuery()`) достаточно универсальной для выполнения множества операций. Итак, у этой функции есть еще одна особенность, которую мы хотим исследовать.

Передавая функции `$()` строку с кодом разметки HTML-элементов дерева DOM, мы можем создавать эти элементы на лету. Например, так можно создать новый элемент-абзац:

```
$("<p>Hi there!</p>")
```

Но в создании ни с чем не связанных элементов DOM (или иерархии элементов) нет никакого смысла. Обычно после создания иерархии элементов задействуются другие функции jQuery для манипулирования деревом DOM.

Давайте исследуем в качестве примера листинг 1.1.

Листинг 1.1. Создание элементов HTML на лету

```
<html>
  <head>
    <title>Follow me!</title>
    <script type="text/javascript" src="../scripts/jquery-1.2.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $("<p>Hi there!</p>").insertAfter("#followMe");
      });
    </script>
  </head>

  <body>
    <p id="followMe">Follow me!</p>
  </body>
</html>
```

❶ Обработчик готовности документа, создающий элементы HTML

❷ Существующий элемент, за которым будет вставлен новый

В этом примере в теле документа определяется существующий HTML-элемент абзаца с именем `followMe` ❷. В сценарии, который находится в разделе `<head>`, определяется сценарий обработчика события готовности документа ❶. Этот сценарий добавляет вновь создаваемый абзац в дерево DOM сразу вслед за существующим элементом:

```
$("<p>Hi there!</p>").insertAfter("#followMe");
```

Результат показан на рис. 1.2.

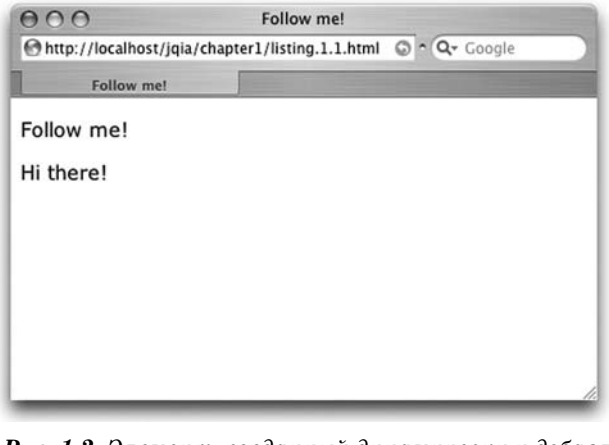

***Рис. 1.2.** Элемент, созданный динамически и добавленный в документ*

Полный комплект функций манипулирования деревом DOM мы рассмотрим в главе 2, где вы увидите, что jQuery предоставляет много функций манипулирования DOM, позволяющих придать документу желаемую структуру.

Теперь, когда вы познакомились с базовым синтаксисом jQuery, давайте взглянем на одну из самых мощных особенностей этой библиотеки.

1.3.5. Расширение jQuery

Функция-обертка jQuery обеспечивает доступ к многим полезным функциям, которые мы постоянно будем использовать в страницах. Но ни одна библиотека не в состоянии удовлетворить все потребности без исключения. Можно даже утверждать, что ни одна библиотека не должна стремиться удовлетворить все возможные потребности, иначе появится огромная неуклюжая масса программного кода, содержащая редко используемые функции, которые только мешают работе!

Понимая это, авторы библиотеки jQuery прилагают усилия для выявления функциональных особенностей, востребованных большинством авторов страниц, чтобы включать в библиотеку только такие особенности. Ввиду того что у каждого автора страниц могут быть собственные неповторимые потребности, библиотека jQuery была создана легко расширяемой.

Но зачем расширять jQuery – ведь для восполнения недостатка функциональности можно писать автономные функции!

Все просто: расширяя библиотеку, мы можем использовать ее мощные особенности, в частности возможность выбора элементов.

Рассмотрим конкретный пример: в библиотеке jQuery отсутствует предопределенная функция, которая деактивировала бы элементы форм. Если во всех приложениях используются формы, пригодилась бы возможность применять, к примеру, такой синтаксис:

```
$("form#myForm input.special").disable();
```

К счастью, архитектура jQuery позволяет легко расширять имеющийся набор функций, расширяя обертку, возвращаемую вызовом `$()`. Рассмотрим базовую идиому, позволяющую этого достигнуть:

```
$.fn.disable = function() {
  return this.each(function() {
    if (typeof this.disabled != "undefined") this.disabled = true;
  });
}
```

Здесь много новых синтаксических конструкций, но не стоит из-за них сильно волноваться. Волнения сами собой улягутся, когда вы прочитаете следующие несколько глав. Эту базовую идиому вы еще много раз примените на практике.

Во-первых, конструкция `$.fn.disable` означает, что мы расширяем обертку `$` функцией с именем `disable`. Внутри этой функции коллекцию обернутых элементов DOM, над которыми будет выполняться операция, представляет идентификатор `this`.

Во-вторых, метод `each()` в этой обертке вызывается для обхода всех элементов коллекции. Подробнее этот и подобные ему методы рассматриваются в главе 2. Внутри функции, передаваемой методу `each()`, ключевое слово `this` является ссылкой на конкретный элемент DOM в текущей итерации. Пусть вас не смущает тот факт, что внутри вложенной функции ссылка `this` указывает на различные объекты. Когда вы напишете несколько функций расширения, это станет привычным и естественным.

Для каждого элемента выполняется проверка – есть ли у текущего элемента атрибут `disabled`, и если такой атрибут имеется, ему присваивается значение `true`. Результат метода `each()` (обертка) возвращается в вызывающую программу, чтобы обеспечить возможность составления цепочек нашим новым методом `disable()`, как это делают многие методы библиотеки jQuery. После этого можно написать такую инструкцию:

```
$("form#myForm input.special").disable().addClass("moreSpecial");
```

С точки зрения кода нашей страницы все выглядит так, как если бы наш новый метод `disable()` был встроен непосредственно в библиотеку! Этот прием обладает такой мощью, что большинство пользователей, начинающих изучать jQuery, обнаруживают в себе способность создавать небольшие расширения для jQuery, как только начали применять библиотеку.

Более того, наиболее инициативные пользователи расширили возможности jQuery наборами полезных функций, которые называются *подключаемыми модулями*, *модулями расширения* (*plugins*). Мы еще не раз поговорим об этом способе расширения jQuery, а в главе 9 представим вашему вниманию свободно распространяемые официальные модули расширения.

Прежде чем углубиться в тонкости использования jQuery, чтобы вдохнуть жизнь в наши страницы, любопытный спросит, можно ли применять jQuery совместно с другими библиотеками, такими как Prototype, ведь в них, как известно, тоже есть сокращение `$`. Ответ на этот вопрос вы найдете в следующем разделе.

1.3.6. Сочетание jQuery с другими библиотеками

jQuery предоставляет в распоряжение программиста набор мощных инструментов, способных удовлетворить насущные потребности большинства авторов страниц, но даже при этом иногда в странице требуется задействовать возможности нескольких библиотек JavaScript. Такие ситуации могут возникать, например, во время перевода приложения

на использование библиотеки jQuery или когда необходимо использовать в наших страницах другую библиотеку совместно с jQuery.

Разработчики jQuery четко понимают потребности пользователей в своем сообществе и не стремятся блокировать применение других библиотек, создавая все условия для обеспечения мирного сосуществования jQuery с другими библиотеками в ваших страницах.

В первую очередь, по общепринятой рекомендации, они стремятся избегать «загрязнения» глобального пространства имен идентификаторами, которые могут вызывать конфликты не только с другими библиотеками, но, возможно, и с именами из ваших собственных сценариев. Идентификатор `jQuery` и его псевдоним `$` – это единственные имена, внедряемые в глобальное пространство имен. Определение вспомогательных функций, о которых говорилось в разд. 1.3.2, как части пространства имен `jQuery` – это прекрасный пример такой заботы.

Весьма маловероятно, чтобы какая-то другая библиотека имела достаточно веские причины объявить глобальный идентификатор `jQuery`, но здесь есть и более удобный идентификатор – псевдоним `$`. Другие библиотеки JavaScript, в частности библиотека Prototype, используют имя `$` для своих собственных целей. А поскольку применение идентификатора `$` в таких библиотеках является ключом к их функциональности, это чревато серьезным конфликтом.

Авторы jQuery постарались избежать этого конфликта с помощью вспомогательной функции, которая так и называется `noConflict()`. В любой момент, как только будут загружены конфликтующие библиотеки, можно вызвать функцию

```
jQuery.noConflict();
```

которая установит значение идентификатора `$` в соответствие с требованиями библиотеки, отличной от jQuery.

Тонкости использования этой вспомогательной функции мы подробнее рассмотрим в разд. 7.2.

1.4. Итоги

В этом кратком введении в jQuery мы рассмотрели множество подготовительных сведений, чтобы легко и быстро приступить к разработке полнофункциональных интернет-приложений.

Библиотека jQuery будет полезна для любой страницы, которая должна выполнять какие-либо действия кроме простейших операций JavaScript и позволит авторам страниц применять методику ненавязчивого JavaScript. При таком подходе поведение элементов отделяется от структуры документа, так же как CSS позволяет отделить информацию о представлении от структуры, за счет чего улучшается организация страниц и увеличивается гибкость программного кода.

jQuery вводит в пространство имен JavaScript всего два новых идентификатора – функцию `jQuery` и ее псевдоним `$`, тем не менее, через эту функцию библиотека предоставляет массу новых возможностей, что делает ее весьма гибким инструментом. Любая операция, выполняемая этой функцией, основана на ее параметрах.

Как показано выше, функция `jQuery()` служит для:

- отбора и обертывания элементов DOM, участвующих в операции;
- исполнения роли пространства имен для глобальных вспомогательных функций;
- создания элементов DOM из кода разметки HTML;
- определения программного кода, выполняемого после того, как дерево DOM будет готово к манипуляциям.

Библиотека jQuery ведет себя на странице как добропорядочный гражданин, не только минимизируя свое вторжение в глобальное пространство имен, но и обеспечивая возврат официального значения идентификатора `$`, еще больше минимизируя вторжение в глобальное пространство имен в случаях, когда это имя может вызвать конфликт, а именно, когда какая-нибудь другая библиотека, например Prototype, задействует идентификатор `$` для своих нужд. Как вам *такое* отношение к пользователю?

Последнюю версию jQuery можно получить на сайте проекта по адресу *http://jquery.com/*. Версия библиотеки jQuery, с которой тестировался программный код примеров из этой книги (1.2.1), включена в состав загружаемого пакета с примерами.

В следующих главах мы рассмотрим все, что может предложить jQuery нам как авторам полнофункциональных интернет-приложений для страниц. Путешествие, которые мы начнем в следующей главе, покажет, как вдохнуть жизнь в страницы путем манипулирования деревом DOM.