3 Операции с коллекцией jQuery

В этой главе:

- создание и введение новых HTML-элементов в DOM;
- манипулирование коллекцией jQuery;
- итерирование по элементам коллекции jQuery.

Из этой главы вы узнаете, как создавать новые элементы DOM, используя чрезвычайно гибкую функцию `jQuery()`. При работе с библиотекой довольно часто возникает необходимость создавать новые элементы, и jQuery это позволяет. Вам понадобится такая возможность особенно тогда, когда мы начнем обсуждать, как вводить внешние данные в веб-страницу с помощью форматов JSON и XML и методов jQuery в работе с Ajax.

Кроме того, вы изучите другие методы, отличающиеся от `jQuery()`. Они будут разделены на две части. Сперва мы опишем методы, которые, начиная с коллекции jQuery, принимают селектор в качестве параметра для создания нового набора элементов. Например, вы увидите, как, исходя из набора, создать новый, содержащий все дочерние элементы исходного набора, дополнительно фильтрованные с помощью селектора, переданного в качестве аргумента. Затем мы рассмотрим методы, строго не связанные с селекторами, но позволяющие вам итерировать по элементам в наборе или проверить их. Начнем!

3.1. Добавление нового HTML-кода

Во многих случаях вам понадобится создать новые фрагменты HTML и вставить их в страницу. Такие динамические элементы могут быть как простыми, например дополнительный текст, который вы хотите отобразить, так и сложными, такими как создание таблицы в базе данных с результатами, полученными от сервера. Типичная ситуация, в которой вам может пригодиться данная функция, — когда нужно получить внешние данные, как правило, в виде JSON или XML, с помощью Ajax.

Используя jQuery, создать динамические элементы просто. Вы легко можете добавить динамически созданный объект jQuery, содержащий элементы DOM,

передавая функции `$()` строку с HTML-разметкой для этих элементов. Рассмотрим следующую строку:

```
$('<div>Привет</div>');
```

Этот код создает новый объект jQuery, содержащий элемент `div`, готовый к добавлению на страницу (на данный момент он не введен в DOM). Любой метод jQuery, который вы можете запустить для набора существующих элементов, может быть запущен и для свежесозданных HTML-фрагментов. На первый взгляд не впечатляет, но, когда будете работать одновременно с обработчиками событий, Ajax и эффектами (а это вам предстоит в последующих главах), поймете, насколько это действенный инструмент на самом деле.

Обратите внимание: если хотите создать пустой элемент `div`, то можете использовать краткую запись:

```
$('<div>');
```

Она идентична записям `$('<div></div>')` и `$('<div />')`, хотя настоятельно рекомендуем применять хорошо сформированную разметку и включать открытые и закрытые теги для любых типов элементов, которые могут содержать другие элементы. С точки зрения производительности эти три варианта эквивалентны, что становится понятным, если посмотреть на тест, показанный на рис. 3.1 (актуальный тест — на http://jsperf.com/jquery-create-markup/4).

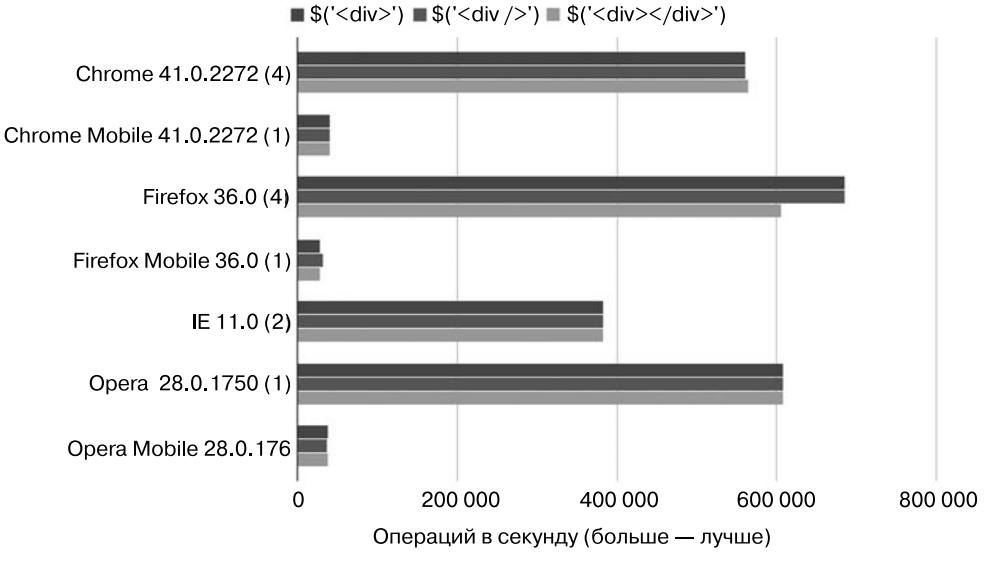


Рис. 3.1. Эталонный тест, сравнивающий три способа создания нового элемента с помощью jQuery(). Он доказывает, что эти способы эквивалентны по производительности практически в каждом браузере

Такие простые HTML-элементы создавать очень легко, а благодаря возможностям цепочки методов jQuery вводить более сложные элементы не намного

сложнее. Вы можете применить любой метод библиотеки к коллекции jQuery, содержащей вновь созданный элемент. Вы можете также присваивать атрибуты элементу с помощью метода `attr()` (мы разберем это в следующей главе), но у jQuery еще более эффективное средство.

В предыдущей главе мы познакомили вас с параметром `context` функции `$()`. При создании нового элемента с функцией `$()` вы используете данный параметр, чтобы указать атрибуты и их значения для элемента, создаваемого вами в виде объекта JavaScript. Свойства такого объекта служат в качестве имен атрибутов, которые будут применены к элементу, а значения соответственно в качестве значений атрибутов.

Предположим, вы хотите создать элемент `img` в комплекте с несколькими атрибутами и сделать его доступным для загрузки щелчком кнопкой мыши. Взгляните на код в листинге 3.1.

Листинг 3.1. Динамически создаваемый полнофункциональный элемент img

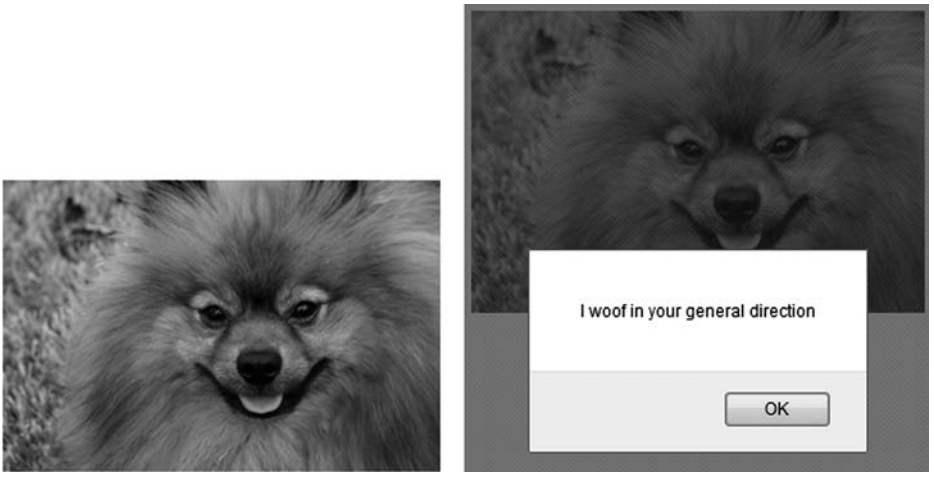
```
$('<img>',                          ← 1 Создает базовый элемент img
  {
    src: 'images/little.bear.png',     2 Присваивает различные атрибуты
    alt: 'Маленький Медведь',
    title:'Гав на вас!',
    click: function() {                3 Устанавливает обработчик для щелчка кнопкой мыши
      alert($(this).attr('title'));
    }
  })
  .appendTo('body');                ← 4 Добавляет элемент в DOM, в конце элемента body
```

В листинге один и тот же оператор jQuery создает базовый элемент `img` ❶; наделяет его важными атрибутами, используя второй параметр, такой как его источник (`source`), альтернативный текст (`alt`) и всплывающее название (`title`) ❷; присоединяет его к дереву DOM (в качестве дочернего элемента `body`) ❹. В показанном примере вы добавляете элемент в DOM с помощью метода `appendTo()`. Мы его еще не рассматривали, но он добавляет элементы в коллекцию jQuery — в этом случае только свежесозданное изображение — к элементу, указанному в аргументе (в нашем примере — элементу `body`).

Мы здесь немного схитрили. В примере вы также применяете второй параметр, чтобы установить обработчик событий, выдающий сообщение (текст которого получен из атрибута `title` изображения), когда мы щелкнем кнопкой мыши на изображении ❸.

Независимо от того, как вы составите код, это довольно громоздкий оператор. Он занимает несколько строк, имеет логические отступы для удобного чтения — но и делает немало. Такие операторы не редкость на страницах с поддержкой jQuery, и если вам это кажется чересчур сложным, то не волнуйтесь. Мы рассмотрим каждый метод, используемый в данном операторе, в нескольких последующих главах. И вскоре вы легко сможете писать составные операторы, подобные этому.

На рис. 3.2 показан результат этого кода, как при первой загрузке страницы *(а)*, так и после щелчка на изображении *(б)*. Полный код для примера можно найти в файле `chapter-3/listing-3.1.html`, предоставленном с книгой.



а *б*

Рис. 3.2. Пример динамически создаваемого изображения: *а* — создание сложных элементов (включая изображение, генерирующее сообщение по щелчку кнопкой мыши) — просто как дважды два; *б* — динамически создаваемое изображение обладает всеми необходимыми стилями и атрибутами, включая обработчик щелчка кнопкой мыши, который создает сообщение

До сих пор вы применяли методы для всего набора соответствующих элементов, но в дальнейшем вам понадобится совершить с ними некие операции, прежде чем пускать в действие.

3.2. Управление коллекцией jQuery

После получения набора jQuery, сделанного из существующих элементов DOM с селекторами либо созданного в виде новых элементов с использованием HTML-фрагментов (а возможно, и того и другого), вы готовы манипулировать этими элементами, задействуя эффективный набор методов библиотеки. Мы начнем рассматривать их в следующей главе, но что, если вы хотите еще большего от набора jQuery, с которым хотите работать? В этом разделе мы рассмотрим множество способов его улучшения, расширения или фильтрации.

Чтобы помочь вам, мы добавили еще одну страницу в загружаемом коде проекта для этой главы: jQuery Operations Lab (`chapter-3/lab.operations.html`). Она показана на рис. 3.3 и во многом напоминает страницу Selectors Lab, к которой мы обращались в главе 2.

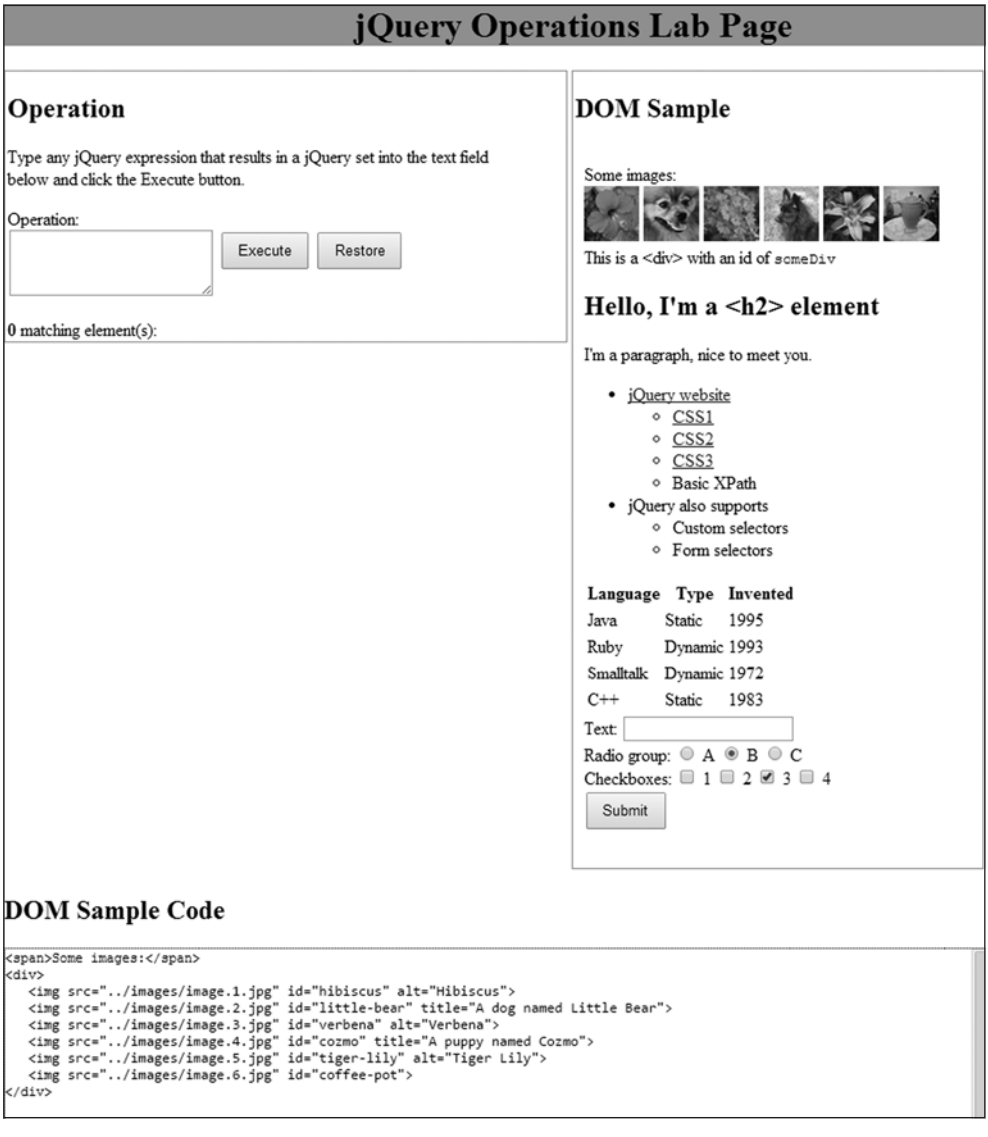


Рис. 3.3. Страница jQuery Operations Lab позволит вам создавать коллекции jQuery в режиме реального времени, чтобы помочь увидеть, как можно создавать и управлять коллекциями

Эта новая страница не только выглядит, как Selectors Lab, но и работает аналогичным образом. Но здесь вместо того, чтобы вводить селектор, можно ввести любую полную операцию jQuery, которая приведет к коллекции jQuery. Операция выполняется в контексте DOM Sample, и, как в случае с Selectors Lab, ее результаты отображаются на экране.

ПРИМЕЧАНИЕ

Эта лабораторная страница загружает элементы, на которые она действует, внутри iframe. Из-за ограничений безопасности некоторых браузеров данная операция может завершиться неудачей. Чтобы этого избежать, вы можете либо запустить страницу в работу с помощью таких веб-серверов, как Apache, Tomcat или IIS, либо поискать конкретное решение для вашего браузера. Например, для браузеров на основе WebKit можно запустить их через интерфейс командной строки (CLI), установив флажок --allow-file-access-fromfiles. Важно, что команда создает новый процесс, поэтому должна открываться не новая вкладка, а новое окно.

Страница jQuery Operations Lab позволяет ввести любое выражение, которое приводит к набору jQuery. Из-за особенностей поэтапной работы библиотеки данное выражение может также включать в себя методы jQuery. Это позволяет странице принести реальную пользу при изучении операций jQuery.

Имейте в виду: вам необходимо ввести правильный синтаксис, а также выражения, которые создают набор jQuery. В противном случае вы столкнетесь с ошибками JavaScript.

Загрузите лабораторную страницу в своем браузере и введите этот текст в поле Operation (Операция):

```
$('img').hide();
```

Затем нажмите кнопку Execute (Выполнить). Операция выполняется в контексте примера DOM, и вы увидите, как изображения исчезают из примера.

После любой операции можно вернуть пример DOM в исходное состояние, нажав кнопку Restore (Восстановить). Хотя мы этого еще не рассматривали, метод `hide()` принадлежит jQuery, и мы еще уделим ему внимание. На данный момент нужно знать: данная функция позволяет скрыть все элементы в наборе. Мы использовали ее с целью продемонстрировать вам конкретный пример того, что вы можете сделать на новой странице Operations Lab. Вы увидите ее в действии, когда проработаете материал в последующих разделах, и, возможно, она пригодится вам в дальнейшем, чтобы проверить различные операции jQuery.

3.2.1. Определение размера набора

Мы уже упоминали ранее, что набор элементов jQuery во многом работает как массив. Это сходство включает в себя также и свойство `length`, совсем как в массивах JavaScript, — оно определяет количество элементов в коллекции jQuery.

Допустим, вы хотите узнать количество всех абзацев на странице и вывести получившееся значение на экран. Можно написать следующую инструкцию:

```
alert($('p').length);
```

Итак, теперь известно, сколько у вас элементов. Что если вы хотите обратиться к ним напрямую?

3.2.2. Получение элементов из набора

После того как у вас появляется набор jQuery, вы часто используете методы jQuery для выполнения с ним какой-либо операции в целом. Возможны случаи, когда вы хотите получить прямую ссылку на элемент или элементы для совершения с ними операций JavaScript. Посмотрим на некоторые из способов, благодаря которым это возможно.

Получение элементов по индексу

Поскольку библиотека позволяет работать с коллекцией jQuery как с массивом JavaScript, можно использовать простую индексацию массива для получения любого элемента в списке по его позиции. Например, чтобы получить первый элемент в множестве всех `<img>` с атрибутом `alt` на странице, можно написать:

```
var imgElement = $('img[alt]')[0];
```

Самые наблюдательные из вас могли заметить, что мы не ставим знак доллара (`$`) перед именем переменной (`imgElement`). Мы о нем не забыли. Данный набор jQuery содержит массив элементов DOM, так что если вы извлекаете один элемент, то это будет не набор jQuery из одного элемента, а простой элемент DOM.

Если вы предпочитаете использовать метод, а не индексацию массивов, то jQuery определяет для этой цели метод `get()`.

Синтаксис метода: get

`get([index])`

Получает одно или все соответствующие элементы в наборе. Если параметр не указан, то все элементы в объекте jQuery будут возвращены как массив JavaScript. Если предоставляется параметр `index`, то возвращается индексированный элемент. `index` может быть отрицательным, в этом случае отсчет будет осуществляться от конца соответствующего набора.

Параметры

`index` (Число) Индекс отдельного элемента, который будет возвращаться. Если он не указан, то весь набор будет возвращен как массив. Если задано отрицательное число, то отсчет идет с конца набора. Если индекс вне границ массива, то есть больше или равен количеству элементов, то метод вернет значение `undefined`.

Возвращает

Элемент DOM, или массив элементов DOM, или `undefined`.

Фрагмент:

```
var imgElement = $('img[alt]').get(0);
```

эквивалентен предыдущему примеру, где использовалась индексация массивов.

Метод `get()` также принимает отрицательный индекс. `get(-1)` возвращает последний элемент в наборе, `get (-2)` — предпоследний и т. д. Вдобавок к получению одного элемента `get()` также может вернуть массив всех элементов в наборе, если он применяется без параметра.

Иногда может понадобиться объект jQuery, содержащий конкретный элемент, а не этот элемент сам по себе. Довольно странно (хотя синтаксически корректно) выглядела бы запись, подобная следующей:

```
$($('p').get(2))
```

Для этой цели у jQuery есть метод `eq()`. Он имитирует действие фильтра селектора `:eq()`, который мы рассматривали в предыдущей главе. Чтобы увидеть их различия в условиях кода, предположим, что вы хотите выбрать второй элемент в наборе, содержащий все `<div>` на странице. Вот как можно выполнить эту задачу:

```
var $secondDiv = $('div').eq(1); var $secondDiv = $('div:eq(1)');
```

Выбор с использованием метода eq()

Выбор с использованием фильтра :eq()

Разница между операторами минимальна, но для большей производительности (подробнее будем рассматривать в главе 15) лучше придерживаться первой формы (метод `eq()`). Как правило, мы предлагаем пропускать методы через фильтры, так как это обычно приводит к улучшению продуктивности.

Теперь, когда мы отметили разницу между методом и фильтром, детально рассмотрим первый.

Синтаксис метода: eq

`eq(index)`

Получает индексированный элемент в наборе и возвращает новый набор, содержащий только этот элемент.

Параметры

`index`	(Число) Индекс возвращаемого единичного элемента. Отрицательный индекс может быть указан для выбора элемента, начиная с конца набора.

Возвращает

Коллекцию jQuery, содержащую один элемент или ноль.

Получение первого элемента набора — настолько распространенная операция, что существует удобный метод, который делает ее еще проще, — метод `first()`.

Синтаксис метода: first

`first()`

Получает первый элемент в наборе и возвращает новый набор, содержащий только этот элемент. Если набор пустой — возвращается также пустой набор.

Параметры

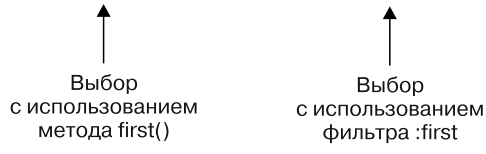
Отсутствуют.

Возвращает

Коллекцию jQuery, содержащую один элемент или ноль.

У метода `first()` есть аналог в фильтре `:first`. Мы хотим еще раз показать пример из двух вариантов. Если вы хотите получить первый абзац страницы, то можно написать одну из следующих инструкций:

```
var $firstPar = $('p').first(); var $firstPar = $('p:first');
```



Неудивительно, что разница с точки зрения кода минимальна, но метод `first()` здесь предпочтительнее, чем фильтр `:first`.

Как вы уже могли ожидать, существует соответствующий метод для получения и последнего элемента в наборе, который является аналогом фильтра `:last`.

Синтаксис метода: last

`last()`

Получает последний элемент в наборе и возвращает новый набор, содержащий только этот элемент. Если набор пустой — возвращается также пустой набор.

Параметры

Отсутствуют.

Возвращает

Коллекцию jQuery, содержащую один элемент или ноль.

Если вы хотите попрактиковаться с этими методами, то можете использовать страницу jQuery Operations Lab. Например, для получения первого элемента из списка, показанного на странице, можно написать:

```
$('li', '.my-list').first();
```

Теперь рассмотрим другой метод для получения массива элементов в наборе.

Получение всех элементов массива

Для получения всех элементов в объекте jQuery в виде JavaScript-массива из элементов DOM библиотека предоставляет метод `toArray()`.

Синтаксис метода: toArray

`toArray()`

Возвращает элементы набора как массив элементов DOM.

Параметры

Отсутствуют.

Возвращает

JavaScript-массив элементов DOM в этом наборе.

Рассмотрим такой пример:

```
var allLabeledButtons = $('label + button').toArray();
```

Команда собирает все `<button>` на странице, которым непосредственно предшествует `<label>`, в объект jQuery, а затем создает JavaScript-массив этих элементов, после чего присваивает его переменной `allLabeledButtons`.

Поиск индекса элементов

Притом что метод `get()` находит элемент по индексу, можно использовать обратную операцию `index()` для поиска индекса определенного элемента в наборе. Синтаксис метода `index()` выглядит следующим образом.

Синтаксис метода: index

`index([element])`

Находит указанный элемент в наборе и возвращает его порядковый индекс в наборе или находит порядковый индекс первого элемента в наборе среди других элементов того же уровня. Если элемент не найден, то возвращается значение `-1`.

Параметры

`element` (Селектор|Элемент|jQuery) Строка, содержащая селектор, ссылку на элемент или объект jQuery, порядковый номер которого необходимо определить. Если дается объект jQuery, то осуществляется поиск первого элемента в наборе. Непредоставление аргумента влечет возвращение индекса первого элемента в наборе в пределах списка родственных элементов одного уровня.

Возвращает

Порядковый номер указываемого элемента в наборе или его родственных элементов одного уровня или `-1`, если он не найден.

Чтобы помочь понять этот метод, предположим, что у вас есть следующий HTML-код:

```
<ul id="main-menu">
  <li id="home-link"><a href="/">Главная страница</a></li>
  <li id="projects-link"><a href="/projects">Проекты</a></li>
  <li id="blog-link"><a href="/blog">Блог</a></li>
  <li id="about-link"><a href="/about">О нас</a></li>
</ul>
```

Допустим, вы хотите узнать порядковый номер элемента списка (`<li>`), содержащий ссылку на блог, которым является элемент, имеющий идентификатор `blog-link`, в неупорядоченном списке с идентификатором `main-menu`.

ПРИМЕЧАНИЕ

Заполнять страницы таким количеством идентификаторов — не лучшая практика: в больших приложениях ими тяжело управлять и трудно дать гарантию, что там не будет дубликатов. Мы использовали их ради примера.

Можно получить это значение таким образом:

```
var index = $('#main-menu > li').index($('#blog-link'));
```

Основываясь на уже полученных знаниях о параметрах, принятых по методу `index()`, эту команду можно написать еще и так:

```
var index = $('#main-menu > li').index(document.getElementById('blog-link'));
```

Помните: индекс начинается с нуля. Индекс первого элемента — 0, второго — 1 и т. д. Таким образом, значение, которое вы получите, равняется `2`, потому что элемент является третьим в списке. Этот код доступен в файле `chapter-3/jquery.index.html`, предоставленном с книгой, а также в JS Bin (http://jsbin.com/notice/edit?html,js,console).

Метод `index()` вполне можно использовать для поиска индекса элемента в пределах его родительского элемента (то есть среди сестринских элементов). Посмотрим, что это значит, чтобы лучше понять. Родителем элемента списка с идентификатором `blog-link` является неупорядоченный список `main-menu`. Сестринскими будут элементы одного с `blog-link` уровня в дереве DOM, у которых один и тот же общий предок (неупорядоченный список). Учитывая нашу разметку, этими элементами являются все остальные элементы списка. Ссылки исключены, так как находятся внутри идентификатора `main-menu`, но не на одном и том же уровне, что и `blog-link`. Написание:

```
var index = $('#blog-link').index();
```

установит `index` опять к `2`.

Чтобы понять, чем интересен вызов `index()` без параметра, рассмотрим следующую разметку:

```
<div id="container">
  <p>Это текст</p>
  <img src="image.jpg" />
  <a href="/">Главная страница</a>
  <img src="anotherImage.jpg" />
  <p>Другой текст</p>
</div>
```

На этот раз разметка содержит несколько различных элементов. Допустим, вы хотите узнать порядковый номер первого элемента `img` в пределах его родительского элемента (`<div>` с идентификатором `container`). Вы можете написать:

```
var index = $('#container > img').index();
```

Значение `index` устанавливается как `1`, поскольку среди потомков `container` первый найденный `<img>` оказывается вторым элементом (он следует за `<p>`).

Помимо получения индекса элемента, библиотека дает возможность получить поднаборы набора, основанные на отношениях элементов коллекции jQuery к другим элементам в DOM. Посмотрим, как это сделать.

3.2.3. Получение наборов с использованием отношений

jQuery позволяет получить новые наборы из уже существующих на основе иерархических взаимоотношений элементов внутри DOM.

Допустим, у вас есть абзац, имеющий идентификатор `description`, и вы хотите узнать количество его предков, которые являются `<div>`. Основываясь на текущих знаниях о селекторах и методах, вы можете сказать, что это невозможно. На помощь приходит функция `parents()`. Рассмотрим следующий код:

```
var count = $('#description').parents('div').length;
```

Используя `parents()`, можно получить нужную информацию. Этот метод возвращает предков каждого элемента в текущий набор соответствующих элементов (который состоит из единственного абзаца, имеющего `description` в качестве своего идентификатора). Вы можете задать фильтры для предшественников с помощью селектора, как показано в примере. Поскольку в вашей коллекции jQuery находится только один элемент (мы предполагаем, что он существует на вашей странице), последует ожидаемый результат.

Что делать, если нужно узнать количество потомков вашего гипотетического абзаца? Это можно легко выяснить с помощью селекторов:

```
var count = $('#description > *').length;
```

Но погодите! Вы применяете все тот же универсальный селектор, который мы настоятельно не рекомендовали в предыдущем разделе? К сожалению, да. С точки зрения производительности лучше всего задать ту же самую команду, используя метод `children()`, как в таком примере:

```
var count = $('#description').children().length;
```

Однако этот метод не возвращает текстовые узлы. Что здесь можно сделать?

Для ситуаций, когда вы должны работать с текстовыми узлами, можно использовать метод `contents()`. Этот метод и `children()` отличаются тем, что первый из них не принимает никаких параметров. Возвращаясь к нашему примеру подсчета, вы можете написать:

```
var count = $('#description').contents().length;
```

Вы знаете, что подсчет элементов сам по себе не приносит большой пользы, и мы понимаем, что вам не терпится погрузиться в создание удивительных эффектов с помощью jQuery. Просим вас подождать еще чуть-чуть, пока мы не предоставим все необходимые сведения.

Метод `find()`, вероятно, один из наиболее часто используемых. Он позволяет осуществлять поиск через потомков элементов (с помощью поиска в глубину) в наборе и возвращает новый объект jQuery. Этот объект содержит все элементы,