3
DOM Traversal Methods

In addition to the selector expressions described in Chapter 2, *Selector Expressions*, jQuery has a variety of **DOM traversal methods** that help us select elements in a document. These methods offer a great deal of flexibility, even allowing us to act upon multiple sets of elements in a single chain as follows:

```
$('div.section').addClass('lit').eq(1).addClass('profound');
```

At times, the choice between a selector expression and a corresponding DOM traversal method is simply a matter of taste. However, there is no doubt that the combined set of expressions and methods makes for an extremely powerful toolset for acting on any part of the document we desire.

The jQuery function

The following function underpins the entire jQuery library. It serves as an "object factory," which allows us to create the jQuery objects that all of the other methods are attached to. The function is named `jQuery()`, but as with all uses of that identifier throughout the library, we typically use the alias `$()` instead.

$()

Create a new jQuery object matching elements in the DOM.

```
$(selector[, context])
$(element)
$(elementArray)
$(object)
$(html)
```

Parameters (first version)

- `selector`: A string containing a selector expression
- `context` (optional): The portion of the DOM tree within which to search

Parameters (second version)

- `element`: A DOM element to wrap in a jQuery object

Parameters (third version)

- `elementArray`: An array containing a set of DOM elements to wrap in a jQuery object

Parameters (fourth version)

- `object`: An existing jQuery object to clone

Parameters (fifth version)

- `html`: A string containing an HTML snippet describing new DOM elements to create

Return value

The newly constructed jQuery object.

Description

In the first formulation of this function, `$()` searches through the DOM for any elements that match the provided selector and creates a new jQuery object that references these elements.

```
$('div.foo');
```

In Chapter 2, *Selector Expressions*, we explored the range of selector expressions that can be used within this string.

Selector context

By default, selectors perform their searches within the DOM starting at the document root. However, an alternative context can be given for the search by using the optional second parameter to the `$()` function. For example, if we wish to do a search for an element within a callback function, we can restrict that search like this:

```
$('div.foo').click(function() {
  $('span', this).addClass('bar');
});
```

As we've restricted the `span` selector to the context of `this`, only spans within the clicked element will get the additional class.

Internally, selector context is implemented with the `.find()` method, so `$('span', this)` is equivalent to `$(this).find('span')`.

Using DOM elements

The second and third formulations of this function allow us to create a jQuery object using a DOM element(s) that we have already found in some other way. A common use of this facility is to call jQuery methods on an element that has been passed to a callback function through the `this` keyword.

```
$('div.foo').click(function() {
  $(this).slideUp();
});
```

This example causes elements to hide with a sliding animation when clicked. Because the handler receives the clicked item in the `this` keyword as a bare DOM element, the element must be wrapped in a jQuery object before we can call jQuery methods on it.

When XML data is returned from an AJAX call, we can use the `$()` function to wrap it in a jQuery object that we can easily work with. Once this is done, we can retrieve individual elements of the XML structure using `.find()` and other DOM traversal methods.

Cloning jQuery objects

When a jQuery object is passed as a parameter to the `$()` function, a clone of the object is created. This new jQuery object references the same DOM elements as the initial one.

Creating new elements

If a string is passed as the parameter to `$()`, jQuery examines the string to see if it looks like HTML. If not, the string is interpreted as a selector expression, as previously explained. However, if the string appears to be an HTML snippet, jQuery attempts to create new DOM elements as described by the HTML. Then a jQuery object that refers to these elements is created and returned. We can perform any of the usual jQuery methods on this object:

```
$('<p>My <em>new</em> paragraph</p>').appendTo('body');
```

When the parameter has multiple tags in it, as it does in this example, the actual creation of the elements is handled by the browser's `innerHTML` mechanism. Specifically, jQuery creates a new `<div>` element and sets the `innerHTML` property of the element to the HTML snippet that was passed in. When the parameter has a single tag, such as `$('<img />')` or `$('<a>hello</a>')`, jQuery creates the element using the native JavaScript `createElement()` function.

To ensure cross-platform compatibility, the snippet must be well formed. Tags that can contain other elements should be paired with a closing tag as follows:

```
$('<a></a>');
```

Alternatively, jQuery allows XML-like tag syntax (with or without a space before the slash) such as this:

```
$('<a/>');
```

Tags that cannot contain elements may or may not be quick-closed.

```
$('<img />');
$('<input>');
```

Filtering methods

These methods remove elements from the set matched by a jQuery object.

.filter()

Reduce the set of matched elements to those that match the selector or pass the function's test.

```
.filter(selector)
.filter(function)
```

Parameters (first version)

- `selector`: A string containing a selector expression to match elements against

Parameters (second version)

- `function`: A function used as a test for each element in the set

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.filter()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; all elements matching the selector will be included in the result.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
  <li>list item 6</li>
</ul>
```

We can apply this method to the set of list items like this:

```
$('li').filter(':even').css('background-color', 'red');
```

The result of this call is a red background for the items `1`, `3`, and `5`, as they match the selector. (Recall that `:even` and `:odd` use 0-based indexing.)

Using a filter function

The second form of this method allows us to filter elements against a function rather than a selector. If the function returns `true` for an element, the element will be included in the filtered set; otherwise, it will be excluded. Suppose we have a somewhat more involved HTML snippet as follows:

```
<ul>
  <li><strong>list</strong> item 1 -
    one strong tag</li>
  <li><strong>list</strong> item <strong>2</strong> -
    two <span>strong tags</span></li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
  <li>list item 6</li>
</ul>
```

In such a case, we can select the list items and then filter them based on their contents:

```
$('li').filter(function(index) {
  return $('strong', this).length == 1;
}).css('background-color', 'red');
```

This code will alter only the first item in the list as it contains exactly one `<strong>` tag. Within the filter function, `this` refers to each DOM element in turn. The parameter passed to the function tells us the index of that DOM element within the set matched by the jQuery object.

We can also take advantage of the `index` parameter passed through the function, which indicates the 0-based position of the element within the unfiltered set of the matched elements.

```
$('li').filter(function(index) {
  return index % 3 == 2;
}).css('background-color', 'red');
```

This alteration to the code will cause items `3` and `6` to be highlighted, as it uses the modulus operator (`%`) to select every item with an `index` value that, when divided by `3`, has a remainder of `2`.

.not()

Remove elements from the set of matched elements.

```
.not(selector)
.not(elements)
.not(function)
```

Parameters (first version)

- `selector`: A string containing a selector expression to match elements against

Parameters (second version)

- `elements`: One or more DOM elements to remove from the matched set

Parameters (third version)

- `function`: A function used as a test for each element in the set

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items.

```
$('li').not(':even').css('background-color', 'red');
```

The result of this call is a red background for items `2` and `4`, as they do not match the selector. (Recall that `:even` and `:odd` use 0-based indexing.)

Removing specific elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an ID applied to one of its items as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li id="notli">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can fetch the third item in the list using the native JavaScript `getElementById()` function, and then remove it from a jQuery object.

```
$('li').not(document.getElementById('notli'))
  .css('background-color', 'red');
```

This statement changes the color of items `1`, `2`, `4`, and `5`. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function returns `true` are excluded from the filtered set; all other elements are included.

.has()

Reduce the set of matched elements to those that have a descendant that matches the selector.

```
.has(selector)
```

Parameters

- `selector`: A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.has()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against the descendants of the matching elements; the element will be included in the result if any of its descendant elements matches the selector.

Consider a page with a nested list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2
    <ul>
      <li>list item 2-a</li>
      <li>list item 2-b</li>
    </ul>
  </li>
  <li>list item 3</li>
  <li>list item 4</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').has('ul').css('background-color', 'red');
```

The result of this call is a red background for item `2`, as it is the only `<li>` that has a `<ul>` among its descendants.

.eq()

Reduce the set of matched elements to the one at the specified index.

```
.eq(index)
```

Parameters

- `index`: An integer indicating the 0-based position of the element

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.eq()` method constructs a new jQuery object from one of the matching elements. The supplied index identifies the position of this element in the set.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').eq(2).css('background-color', 'red');
```

The result of this call is a red background for item `3`. Note that the supplied index is 0-based, and refers to the position of the element within the jQuery object, not within the DOM tree.

If a negative number is provided, this indicates a position starting from the end of the set, rather than the beginning. For example:

```
$('li').eq(-2).css('background-color', 'red');
```

The result of this call is a red background for item `4`, as it is second from the end of the set.

.first()

Reduce the set of matched elements to the first in the set.

```
    .first()
```

Parameters

None

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.first()` method constructs a new jQuery object from the first matching element.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').first().css('background-color', 'red');
```

The result of this call is a red background for item `1`.

.last()

Reduce the set of matched elements to the final one in the set.

```
.last()
```

Parameters

None

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.last()` method constructs a new jQuery object from the first matching element.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').last().css('background-color', 'red');
```

The result of this call is a red background for the final list item .

.slice()

Reduce the set of matched elements to a subset specified by a range of indices.

```
.slice(start[, end])
```

Parameters

- `start`: An integer indicating the 0-based position after which the elements are selected
- `end` (optional): An integer indicating the 0-based position before which the elements stop being selected; if omitted, the range continues until the end of the set

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.slice()` method constructs a new jQuery object from a subset of the matching elements. The supplied `start` index identifies the position of one of the elements in the set. If `end` is omitted, all of the elements after this one will be included in the result.

Consider a page with a simple list as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').slice(2).css('background-color', 'red');
```

The result of this call is a red background for the items `3`, `4`, and `5`. Note that the supplied index is 0-based, and refers to the position of elements within the jQuery object; not within the DOM tree.

The `end` parameter allows us to limit the selected range even further. For example:

```
$('li').slice(2, 4).css('background-color', 'red');
```

Now only items `3` and `4` are selected. The index is once again 0-based. The range extends up to, *but doesn't include*, the specified index.

Negative indices

The jQuery `.slice()` method is patterned after the JavaScript `.slice()` method for arrays. One of the features that it mimics is the ability for negative numbers to be passed as either the `start` or `end` parameter. If a negative number is provided, this indicates a position starting from the end of the set, rather than the beginning.
For example:

```
$('li').slice(-2, -1).css('background-color', 'red');
```

This time only the list item `4` turns red, as it is the only item in the range between the second from the end `(-2)` and the first from the end `(-1)`.

Tree traversal methods

These methods use the structure of the DOM tree to locate a new set of elements.

.find()

> Get the descendants of each element in the current set of matched elements filtered by a selector.
>
> `.find(selector)`

Parameters

- `selector`: A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.find()` method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The method accepts a selector expression of the same type that we can pass to the `$()` function. The elements will be filtered by testing whether they match this selector.

Consider a page with a basic nested list as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
```

```
          <li class="item-c">C</li>
        </ul>
      </li>
      <li class="item-iii">III</li>
    </ul>
```

If we begin at item `II`, we can find list items within it as follows:

```
$('li.item-ii').find('li').css('background-color', 'red');
```

The result of this call is a red background on items `A`, `B`, `1`, `2`, `3`, and `C`. Even though item `II` matches the selector expression, it is not included in the results; only descendants are considered candidates for the match.

As previously discussed in the section *The jQuery Function*, selector context is implemented with the `.find()` method. Therefore, `$('li.item-ii').find('li')` is equivalent to `$('li', 'li.item-ii')`.

Unlike in the rest of the tree traversal methods, the selector expression is *required* in a call to `.find()`. If we need to retrieve all of the descendant elements, we can pass in the universal selector `'*'` to accomplish this.

.children()

Get the children of each element in the set of matched elements, optionally filtered by a selector.

```
.children([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.children()` method allows us to search through the immediate children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

If we begin at the `level-2` list, we can find its children.

```
$('ul.level-2').children().css('background-color', 'red');
```

The result of this call is a red background behind the items `A`, `B`, and `C`. As we do not supply a selector expression, all of the children are part of the returned jQuery object; if we had supplied one, only the matching items among these three would be included.

.parents()

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

```
    .parents([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.parents()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether or not they match it.

Consider a page with a basic nested list as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

If we begin at item `A`, we can find its ancestors as follows:

```
$('li.item-a').parents().css('background-color', 'red');
```

The result of this call is a red background for the `level-2` list, the item `II`, and the `level-1` list (and on up the DOM tree all the way to the document's root element, which is typically `<html>`). As we do not supply a selector expression, all of the ancestors are part of the returned jQuery object. If we had supplied one, only the matching items among these would be included.

.parentsUntil()

Get the ancestors of each element in the current set of matched elements up to, but not including, the element matched by the selector.

```
.parentsUntil(selector)
```

Parameters

- `selector`: A string containing a selector expression to indicate where to stop matching ancestor elements

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.parentsUntil()` method traverses through the ancestors of these elements until it reaches an element matched by the selector passed in the method's argument. The resulting jQuery object contains all of the ancestors up to, but not including, the one matched by the `.parentsUntil()` selector. Consider a page with a basic nested list as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

If we begin at item `A`, we can find its ancestors up to but not including `<li class="level-1">` as follows:

```
$('li.item-a').parentsUntil('.level-1')
  .css('background-color', 'red');
```

The result of this call is a red background for the `level-2` list and the item `II`.

If the `.parentsUntil()` selector is not matched, or if no selector is supplied, the returned jQuery object contains all of the previous jQuery object's ancestors. For example, let's say we begin at item `A` again, but this time we use a selector that is not matched by any of its ancestors:

```
$('li.item-a').parentsUntil('.not-here')
  .css('background-color', 'red');
```

The result of this call is a red `background-color` style applied to the `level-2` list, the item `II`, the `level-1` list, the `<body>` element, and the `<html>` element.

.parent()

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

```
.parent([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.parent()` method allows us to search through the parents of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

If we begin at item `A`, we can find its parents.

```
$('li.item-a').parent().css('background-color', 'red');
```

The result of this call is a red background for the `level-2` list. As we do not supply a selector expression, the parent element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

.closest()

Get the first element that matches the selector, beginning at the current element and progressing up through the DOM tree.

```
.closest(selector[, context])
```

Parameters

- `selector`: A string containing a selector expression to match elements against
- `context` (optional): A DOM element within which a matching element may be found

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.closest()` method allows us to search through these elements and their ancestors in the DOM tree and construct a new jQuery object from the matching elements.

The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

.closest()	.parents()
Begins with the current element	Begins with the parent element
Travels up the DOM tree until it finds a match for the supplied selector	Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collection; it then filters that collection based on a selector if one is supplied
The returned jQuery object contains zero or one element	The returned jQuery object contains zero, one, or multiple elements

Consider a page with a basic nested list as follows:

```
<ul id="one" class="level-1">
  <li class="item-i">I</li>
  <li id="ii" class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

Suppose we perform a search for `<ul>` elements starting at item `A`.

```
$('li.item-a').closest('ul')
  .css('background-color', 'red');
```

This will change the color of the `level-2 <ul>`, as it is the first `<ul>` encountered when traveling up the DOM tree.

Suppose we search for an `<li>` element instead:

```
$('li.item-a').closest('li')
  .css('background-color', 'red');
```

This will change the color of list item `A`. The `.closest()` method begins its search with the element itself before progressing up the DOM tree and stops when item `A` matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
Var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII)
  .css('background-color', 'red');
$('li.item-a').closest('#one', listItemII)
  .css('background-color', 'green');
```

This will change the color of the `level-2 <ul>`, because it is both the first `<ul>` ancestor of list item `A` and a descendant of list item `II`. It will not change the color of the `level-1 <ul>`, however, because it is not a descendant of list item `II`.

.offsetParent()

Get the closest ancestor element that is positioned.

```
.offsetParent()
```

Parameters

None

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.offsetParent()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object wrapped around the closest positioned ancestor. An element is said to be **positioned** if its CSS `position` attribute is `relative`, `absolute`, or `fixed`. This information is useful for calculating offsets for performing animations and placing objects on the page.

Consider a page with a basic nested list with a positioned element as follows:

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii" style="position: relative;">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

If we begin at item `A`, we can find its positioned ancestor.

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

This will change the color of list item `II`, which is positioned.

.siblings()

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

```
    .siblings([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.siblings()` method allows us to search through the siblings of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the third `item`, we can find its siblings as follows:

```
$('li.third-item').siblings().css('background-color', 'red');
```

The result of this call is a red background behind items `1`, `2`, `4`, and `5`. As we do not supply a selector expression, all of the siblings are part of the object. If we had supplied one, only the matching items among these four would be included.

The original element is not included among the siblings, which is important to remember when we wish to find all of the elements at a particular level of the DOM tree.

.prev()

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

```
.prev([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.prev()` method allows us to search through the predecessors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the element that comes just before it.

```
$('li.third-item').prev().css('background-color', 'red');
```

The result of this call is a red background behind item `2`. As we do not supply a selector expression, this preceding element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

.prevAll()

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

```
    .prevAll([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.prevAll()` method allows us to search through the predecessors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements that come before it.

```
$('li.third-item').prevAll().css('background-color', 'red');
```

The result of this call is a red background behind items `1` and `2`. As we do not supply a selector expression, these preceding elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

.prevUntil()

Get all preceding siblings of each element up to, but not including, the element matched by the selector.

```
.prevUntil(selector)
```

Parameters

- `selector`: A string containing a selector expression to indicate where to stop matching previous sibling elements

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.prevUntil()` method allows us to search through the predecessors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all previous siblings up to, but not including, the one matched by the `.prevUntil()` selector.

If the selector is not matched or is not supplied, all previous siblings will be selected; in these cases it selects the same elements as the `.prevAll()` method does when no filter selector is provided.

Consider a page with a simple definition list as follows:

```
<dl>
  <dt>term 1</dt>
  <dd>definition 1-a</dd>
  <dd>definition 1-b</dd>
  <dd>definition 1-c</dd>
  <dd>definition 1-d</dd>

  <dt id="term-2">term 2</dt>
  <dd>definition 2-a</dd>
  <dd>definition 2-b</dd>
  <dd>definition 2-c</dd>

  <dt>term 3</dt>
  <dd>definition 3-a</dd>
  <dd>definition 3-b</dd>
</dl>
```

If we begin at the second term, we can find the elements that come before it until a preceding `<dt>`.

```
$('#term-2').prevUntil('dt').css('background-color', 'red');
```

The result of this call is a red background behind definitions `1-a`, `1-b`, `1-c`, and `1-d`.

.next()

Get the immediately following sibling of each element in the set of matched elements, optionally filtered by a selector.

```
.next([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.next()` method allows us to search through the successors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether or not they match it.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the `third-item`, we can find the element that comes just after it as follows:

```
$('li.third-item').next().css('background-color', 'red');
```

The result of this call is a red background behind item `4`. As we do not supply a selector expression, this following element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

.nextAll()

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

```
.nextAll([selector])
```

Parameters

- `selector` (optional): A string containing a selector expression to match elements against

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.nextAll()` method allows us to search through the successors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the `third-item`, we can find the elements that come after it.

```
$('li.third-item').nextAll().css('background-color', 'red');
```

The result of this call is a red background behind items `4` and `5`. As we do not supply a selector expression, these following elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

.nextUntil()

Get all following siblings of each element up to, but not including, the element matched by the selector.

```
.nextUntil(selector)
```

Parameters

- `selector`: A string containing a selector expression to indicate where to stop matching following sibling elements

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.nextUntil()` method allows us to search through the successive siblings of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all following siblings up to, but not including, the one matched by the `.nextUntil()` selector.

If the selector is not matched or is not supplied, all following siblings will be selected; in these cases it selects the same elements as the `.nextAll()` method does when no filter selector is provided.

Consider a page with a simple definition list as follows:

```
<dl>
  <dt>term 1</dt>
  <dd>definition 1-a</dd>
  <dd>definition 1-b</dd>
  <dd>definition 1-c</dd>
  <dd>definition 1-d</dd>

  <dt id="term-2">term 2</dt>
  <dd>definition 2-a</dd>
  <dd>definition 2-b</dd>
  <dd>definition 2-c</dd>

  <dt>term 3</dt>
  <dd>definition 3-a</dd>
  <dd>definition 3-b</dd>
</dl>
```

If we begin at the second term, we can find the elements that come after it until the next `<dt>`.

```
$('#term-2').nextUntil('dt').css('background-color', 'red');
```

The result of this call is a red background behind definitions `2-a`, `2-b`, and `2-c`.

Miscellaneous traversal methods

These methods provide other mechanisms for manipulating the set of matched DOM elements in a jQuery object.

.add()

Add elements to the set of matched elements.

```
.add(selector[, context])
.add(elements)
.add(html)
```

Parameters (first version)

- `selector`: A string containing a selector expression to match additional elements against
- `context` (optional): The portion of the DOM tree within which to search

Parameters (second version)

- `elements`: One or more elements to add to the set of matched elements

Parameters (third version)

- `html`: An HTML fragment to add to the set of matched elements

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.add()` method constructs a new jQuery object from the union of those elements and the ones passed into the method. The argument to `.add()` can be pretty much anything that `$()` accepts, including a jQuery selector expression, references to DOM elements, or an HTML snippet.

Consider a page with a simple list along with a couple paragraphs as follows:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
</ul>
<p>a paragraph</p>
```

```
<div>
  <p>paragraph within a div</p>
</div>
```

We can select the list items and then the paragraphs by using either a selector or a reference to the DOM element itself as the `.add()` method's argument.

```
$('li').add('p').css('background-color', 'red');
```

or

```
$('li').add(document.getElementsByTagName('p')[0])
  .css('background-color', 'red');
```

The result of this call is a red background behind all three list items and two paragraphs.

We can specify a context within which to search for the elements that we wish to add:

```
$('li').add('p', 'div').css('background-color', 'red');
```

Now the result is a red background behind all three list items, but only the second paragraph.

Using an HTML snippet as the `.add()` method's argument (as in the third version), we can create additional elements on the fly and add those elements to the matched set of elements. Let's say, for example, that we want to alter the background of the list items along with a newly created paragraph.

```
$('li').add('<p id="new">new paragraph</p>')
  .css('background-color', 'red');
```

Although the new paragraph has been created and its background color changed, it still does not appear on the page. To place it on the page, we could add one of the **insertion methods** to the chain.

[See Chapter 4, *DOM Manipulation Methods*, for more information about the insertion methods.]

.is()

Check the current matched set of elements against a selector and return `true` if at least one of these elements matches the selector.

```
.is(selector)
```

Parameters

- `selector`: A string containing a selector expression to match elements against

Return value

A Boolean indicating whether an element matches the selector.

Description

Unlike the rest of the methods in this chapter, `.is()` does not create a new jQuery object. Instead, it allows us to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose we have a list, with two of its items containing a child element as follows:

```
<ul>
  <li>list <strong>item 1</strong></li>
  <li><span>list item 2</span></li>
  <li>list item 3</li>
</ul>
```

We can attach a click handler to the `<ul>` element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked.

```
$('ul').click(function(event) {
  if ($(event.target).is('li') ) {
    $(event.target).css('background-color', 'red');
  }
});
```

Now, when the user clicks on the word **list** in item 1, or anywhere on item 3, the clicked list item will be given a red background. However, when the user clicks on **item 1** in the first item or anywhere in the second item, nothing will happen because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

.end()

End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

```
.end()
```

Parameters

None

Return value

The previous jQuery object.

Description

Most of the methods in this chapter operate on a jQuery object and produce a new one that matches a different set of DOM elements. When this happens, it is as if the new set of elements is pushed onto a **stack** that is maintained inside the object. Each successive filtering method pushes a new element set onto the stack. If we need an older element set, we can use `.end()` to pop the sets back off of the stack.

Suppose we have a couple short lists on a page as follows:

```
<ul class="first">
   <li class="foo">list item 1</li>
   <li>list item 2</li>
   <li class="bar">list item 3</li>
</ul>
<ul class="second">
   <li class="foo">list item 1</li>
   <li>list item 2</li>
   <li class="bar"></ul>
```

The `.end()` method is useful primarily when exploiting jQuery's **chaining** properties. When not using chaining, we can usually just call up a previous object by a variable name so that we don't need to manipulate the stack. With `.end()`, though, we can string all of the method calls together.

```
$('ul.first').find('.foo').css('background-color', 'red')
  .end().find('.bar').css('background-color', 'green');
```

This chain searches for items with the `foo` class within the first list only and turns their backgrounds red. Then `.end()` returns the object to its state before the call to `.find()`. So the second `.find()` looks for `'.bar'` inside `<ul class="first">`, not just inside that list's `<li class="foo">`, and turns the matching elements' backgrounds green. The net result is that items `1` and `3` of the first list have a colored background, while none of the items from the second list do.

A long jQuery chain can be visualized as a structured code block with filtering methods providing the openings of nested blocks and `.end()` methods closing them:

```
$('ul.first').find('.foo')
  .css('background-color', 'red')
.end().find('.bar')
  .css('background-color', 'green')
.end();
```

The last `.end()` is unnecessary, as we are discarding the jQuery object immediately thereafter. However, when the code is written in this form, the `.end()` provides visual symmetry and closure, making the program more readable at least in the eyes of some developers.

.andSelf()

Add the previous set of elements on the stack to the current set.

```
.andSelf()
```

Parameters

None

Return value

The new jQuery object.

Description

As previously described in the *Description* for *.end()*, jQuery objects maintain an internal stack that keeps track of changes to the matched set of elements. When one of the DOM traversal methods is called, the new set of elements is pushed onto the stack. If the previous set of elements is desired as well, `.andSelf()` can help.

Consider a page with a simple list as follows:

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements that come after it.

```
$('li.third-item').nextAll().andSelf()
  .css('background-color', 'red');
```

The result of this call is a red background behind items `3`, `4` and `5`. First, the initial selector locates item `3`, initializing the stack with the set containing just this item. The call to `.nextAll()` then pushes the set of items `4` and `5` onto the stack. Finally, the `.andSelf()` invocation merges these two sets together, creating a jQuery object that points to all three items.

.map()

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

```
.map(callback)
```

Parameters

- `callback`: A function object that will be invoked for each element in the current set

Return value

The new jQuery object.

Description

The `.map()` method is particularly useful for getting or setting the value of a collection of elements. Consider a form with a set of checkboxes as follows:

```
<form method="post" action="">
  <fieldset>
    <div>
      <label for="two">2</label>
      <input type="checkbox" value="2" id="two" name="number[]">
    </div>
    <div>
      <label for="four">4</label>
      <input type="checkbox" value="4" id="four" name="number[]">
    </div>
    <div>
      <label for="six">6</label>
      <input type="checkbox" value="6" id="six" name="number[]">
    </div>
    <div>
      <label for="eight">8</label>
      <input type="checkbox" value="8" id="eight" name="number[]">
    </div>
  </fieldset>
</form>
```

We can select all of the checkboxes by setting their `checked` property to `true`.

```
$(':checkbox').map(function() {
 return this.checked = true;
});
```

We can get the sum of the values of the checked inputs as follows:

```
var sum = 0;
$(':checked').map(function() {
 return sum += (this.value * 1);
});
```

We can get a comma-separated list of checkbox IDs.

```
$(':checkbox').map(function() {
  return this.id;
}).get().join(',');
```

The result of this call is the `two,four,six,eight` string.

.contents()

Get the children of each element in the set of matched elements, including text nodes.

```
.contents()
```

Parameters

None

Return value

The new jQuery object.

Description

Given a jQuery object that represents a set of DOM elements, the `.contents()` method allows us to search through the immediate children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.contents()` and `.children()` methods are similar, except that the former includes text nodes as well as HTML elements in the resulting jQuery object.

Consider a simple `<div>` with a number of text nodes, each of which is separated by two line break elements (`<br />`) as follows:

```
<div class="container">
  Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. <br /><br
/>
  Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.<br /><br />
   Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur.
   </div>
```

We can employ the `.contents()` method to help convert this block of text into three well-formed paragraphs.

```
$('.container').contents().filter(function() {
  return this.nodeType == 3;
})
  .wrap('<p></p>')
.end()
.filter('br')
  .remove();
```

This code first retrieves the contents of `<div class="container">` and then filters it for text nodes, which are wrapped in paragraph tags. This is accomplished by testing the `.nodeType` property of the element. This DOM property holds a numeric code indicating the node's type—text nodes use the code `3`. The contents are again filtered, this time for `<br />` elements, and then these elements are removed.