THIRD EDITION

SAMPLE CHAPTER

jQuery IN ACTION

Bear Bibeault
Yehuda Katz
Aurelio De Rosa

FOREWORDS BY Dave Methvin
John Resig

MANNING

jQuery in Action, Third Edition

by Bear Bibeault
Yehuda Katz
Aurelio De Rosa

Chapter 5

brief contents

5 Bringing pages to life with jQuery

This chapter covers

- Manipulating element class names
- Setting the content of DOM elements
- Getting and setting form element values
- Cloning DOM elements
- Modifying the DOM tree by adding, moving, or replacing elements

Today web developers and designers know better than those of 10 years ago (or even themselves 10 years ago) and use the power given to them by DOM scripting to *enhance* a user's web experience, rather than showcase annoying tricks made of blinking texts and animated GIFs. Whether it's to incrementally reveal content, create input controls beyond the basic set provided by HTML, or give users the ability to tune pages to their own liking, DOM manipulation has allowed many web developers to amaze (not annoy) their users.

On an almost daily basis, many of us come across web pages that do something that makes us say, "Hey! I didn't know you could do that!" And being the commensurate professionals that we are (not to mention being insatiably curious about

such things), we immediately start looking at the source code to find out how they did it. That's the beauty of the web, where you can see other developers' code at any time, isn't it?

But rather than having to code up all that script ourselves, we find that jQuery provides a robust set of tools to manipulate the DOM, making those types of "Wow!" pages possible with a surprisingly small amount of code. Whereas the previous chapter introduced you to working with properties, attributes, and data with jQuery, this chapter discusses how to perform operations on DOM elements to bring them to life and to bring that elusive "Wow!" factor to your pages.

5.1 Changing element styling

In the previous chapter, we mentioned that the `className` property is an example of a case where markup attribute names differ from property names. But, truth be told, class names are a bit special in other respects as well and are handled as such by jQuery. This section describes a better way to deal with class names than by directly accessing the `className` property or using the jQuery's `attr()` method.

When you want to change the styling of an element, there are two options used more often than others. The first is that you can add or remove a class, causing a restyle of the element based on its new or removed class. The other is that you can operate on the DOM element by applying styles directly.

Let's start by taking a look at how jQuery makes it simple to perform changes to an element's style via classes.

5.1.1 Adding and removing class names

The `class` attribute of HTML elements is crucially important to the creation of interactive interfaces. In HTML, the `class` attribute is used to supply these names as a space-separated string. You can have as many spaces as you want, but people usually use one. For example, you may have

```
<div class="some-class my-class    another-class"></div>
```

Unfortunately, rather than manifesting themselves as an array of names in the DOM element's corresponding `className` property, the class names appear as that same space-delimited string. How disappointing, and how cumbersome! This means that whenever you want to add class names to or remove class names from an element that already has class names, you need to parse the string to determine the individual names when reading it and be sure to restore it to a valid space-separated format when writing it.

Taking the cue from jQuery and other similar libraries, HTML5 has introduced a better way to manage this task through an API called `classList`. The latter has more or less the same methods exposed by jQuery, but unfortunately, unlike their jQuery counterparts, these native methods can work on only one element at a time. If you want to add a class to a set of elements, you have to iterate over them. In addition,

being a new introduction, it isn't supported by older browsers, most notably Internet Explorer 6–9. To better understand this difference, consider this code written in pure JavaScript that selects all elements having class `some-class` and adds the class `hidden`:

```
var elements = document.getElementsByClassName('some-class');
for(var i = 0; i < elements.length; i++) {
    elements[i].classList.add('hidden');
}
```

The previous snippet is compatible only with modern browsers, including Internet Explorer 10 and above. Now compare it with its jQuery equivalent:

```
$('.some-class').addClass('hidden');
```

The jQuery version is not only shorter but is also compatible starting with Internet Explorer 6 (depending on the version of jQuery you'll use, of course)!

> **NOTE** The list of class names is considered *unordered*; that is, the order of the names within the space-delimited list has no semantic meaning.

Although it's not a monumental activity to write code that handles the task of adding and removing class names from a set of elements, it's always a good idea to abstract such details behind an API that hides the mechanical details of such operations. Luckily, you don't have to develop your own code because jQuery has already done that for you.

Adding class names to all the elements of a set is an easy operation with the following `addClass()` method that was used in the previous snippet.

Method syntax: addClass

`addClass(names)`

Adds the specified class name(s) to all the elements in the set. If a function is provided, every element of the set is passed to it, one at a time, and the returned value is used as the class name(s).

Parameters

`names`	(String|Function) Specifies the class name, or a space-delimited string of names, to be added. If a function, the function is invoked for each element, with that element set as the function context (`this`). The function is passed two values: the element index and the element's current class value. The function's returned value is used as the new class name or names to add to the current value.

Returns

The jQuery collection.

Removing class names is just as straightforward with the following `removeClass()` method.

Method syntax: removeClass

`removeClass(names)`

Removes the specified class name(s) from each element in the jQuery collection. If a function is provided, every element of the set is passed to it, one at a time, and the returned value is used to remove the class name(s).

Parameters

`names`	(String|Function) Specifies the class name, or a space-delimited string of names, to be removed. If a function, the function is invoked for each element, setting that element as the function context (`this`). The function is passed two values: the element index and the class value prior to any removal. The function's returned value is used as the class name or names to be removed.

Returns

The jQuery collection.

To see when the `removeClass()` method is useful, let's say that you have the following element in a page:

```
<p id="text" class="hidden">A brief description</p>
```

You can remove the hidden class with this simple statement:

```
$('#text').removeClass('hidden');
```

Often, you may want to switch a set of styles that belong to a class name back and forth, perhaps to indicate a change between two states or for any other reasons that make sense with your interface. jQuery makes this easy with the `toggleClass()` method.

Method syntax: toggleClass

`toggleClass([names][, switch])`

Adds the specified class name(s) on elements that don't possess it, or removes the name(s) from elements that already possess the class name(s). Note that each element is tested individually, so some elements may have the class name added and others may have it removed.
If the `switch` parameter is provided, the class name(s) is always added to elements without it if `switch` is `true`, and removed from those that have it if it's `false`.
If the method is called without parameters, all the class names of each element in the set will be removed and eventually restored with a new call to this method.
If only the `switch` parameter is provided, all the class names of each element in the set will be kept or removed on that element based on the value of `switch`.
If a function is provided, the returned value is used as the class name or names, and the action taken is based on the `switch` value.

Parameters

`names`	(String|Function) Specifies the class name, or a space-delimited string of names, to be toggled. If a function, the function is invoked for each element, with that element set as the function context (`this`). The function is passed two values: the element index and the class value of that element. The function's returned value is used as the class name or names to toggle.
`switch`	(Boolean) A control expression whose value determines if the class will only be added to the elements (`true`) or only removed (`false`).

Returns

The jQuery collection.

As you can see, the `toggleClass()` method gives you many possibilities. Before moving forward with other methods, let's see some examples.

One situation where the `toggleClass()` method is most useful is when you want to switch visual renditions between elements quickly and easily, usually based on some other elements. Imagine that you want to develop a simple share widget where you have a button that, when clicked, shows a box containing the social media buttons to share the link of the page. If the button is clicked again, the box should be hidden.

Using jQuery—and jQuery's `click()` method that we'll cover in chapter 6—you can easily create this widget:

```
$('.share-widget').click(function() {
    $('.socials', this).toggleClass('hidden');
});
```

The complete code for this demo is available in the file chapter-5/share.widget.html. Before you're disappointed, we want to highlight that the demo doesn't provide any actual sharing functionality, only placeholder text. The resulting page in its two states (box hidden and displayed) is shown in figures 5.1a and figure 5.1b, respectively.

Figure 5.1a The presence of the `hidden` class is toggled whenever the user clicks the button, causing the box to be shown or hidden. In its initial state the box is hidden.

Figure 5.1b When the Share button is clicked, it toggles the `hidden` class. This figure shows the box when displayed.

Toggling a class based on whether the elements already possess the class or not is a common operation, but so is toggling the class based on some other arbitrary condition. Consider the following code:

```
if (aValue === 10) {
   $('p').addClass('hidden');
} else {
   $('p').removeClass('hidden');
}
```

For this common situation, jQuery provides the `switch` parameter we discussed in the description of the method. You can shorten the previous snippet of code as reported here:

```
$('p').toggleClass('hidden', aValue === 10);
```

In this case, the class `hidden` will be added to all the paragraphs selected if the variable `aValue` is strictly equal to 10; otherwise it'll be removed.

As our last example, imagine that you want to add a class on a per-element basis based on a given condition. You might want to add to all the odd-positioned elements in the set a class called `hidden` while keeping all the classes of the even-positioned elements unchanged. You can achieve this goal by passing a function as the first argument of `toggleClass()`:

```
$('p').toggleClass(function(index) {
    return (index % 2 === 0) ? 'hidden' : '' ;
});
```

Sometimes you need to determine whether an element has a particular class to perform some operations accordingly. With jQuery, you can do that by calling the `hasClass()` method:

```
$('p:first').hasClass('surprise-me');
```

This method will return `true` if any element in the set has the specified class, `false` otherwise. The syntax of this method is as follows.

Method syntax: hasClass

`hasClass(name)`

Determines if any element of the set possesses the passed class name

Parameters

`names`	(String) The class name to be searched

Returns

Returns `true` if any element in the set possesses the passed class name, `false` otherwise

Recalling the `is()` method from chapter 3, you could achieve the same goal with

```
$('p:first').is('.surprise-me');
```

But arguably, the `hasClass()` method makes for more readable code, and internally `hasClass()` is a lot more efficient.

Manipulating the stylistic rendition of elements via CSS class names is a powerful tool, but sometimes you want to get down to the nitty-gritty styles themselves as declared directly on the elements. Let's see what jQuery offers you for that.

5.1.2 Getting and setting styles

Modifying the class of an element allows you to choose which predetermined set of defined style sheet rules should be applied. But sometimes you only want to set the value of one or very few properties that are unknown in advance; thus a class name doesn't exist. Applying styles directly on the elements (via the `style` property available on all DOM elements) will automatically override the style defined in style sheets (some exceptions such as `!important` apply, but we aren't going to cover CSS specificity in

detail here), giving you more fine-grained control over individual elements and their styles.

The jQuery `css()` method allows you to manipulate these styles, working in a similar fashion to `attr()`. You can set an individual CSS style by specifying its name and value, or a series of styles by passing in an object.

Method syntax: css

`css(name, value)`
`css(properties)`

Sets the named CSS style property or properties to the specified value for each matched element.

Parameters

`name`	(String) The name of the CSS property to be set. Both the CSS and DOM formatting of multiple-word properties (for example `background-color` versus `backgroundColor`) are supported. Most of the time you'll use the first format version.
`value`	(String|Number|Function) A string, number, or function containing the property value. If a number is passed, jQuery will convert it to a string and add "px" to the end of that string. If you need a different unit, convert the value to a string and add the appropriate unit before calling the method. If a function is passed as this parameter, it will be invoked for each element of the collection, setting the element as the function context (`this`). The function is passed two values: the element index and the current value. The returned value serves as the new value for the CSS property.
`properties`	(Object) Specifies an object whose properties are copied as CSS properties to all elements in the set.

Returns

The jQuery collection.

The `value` argument can also be a function in a similar fashion to the `attr()` method. This means that you can, for instance, expand the width of all elements in the set by 20 pixels times the index of the element as follows:

```
$('.expandable').css('width', function(index, currentWidth) {
   return parseInt(currentWidth, 10) + 20 * index;
});
```

In this snippet you need to pass the current value to `parseInt()` because the width of an element is returned in pixels and as a string (for example, `"50px"`). Without a conversion, the sum will act as a concatenation of strings resulting in a value like `"50px20"` (if the value of `index` is `1`).

In case you want to expand the width of all the elements by 20 pixels, jQuery offers you a nice shortcut. Instead of writing a function, you can write

```
$('.expandable').css('width', '+=20');
```

A similar shortcut is available if you want to subtract a given amount of pixels:

```
$('.expandable').css('width', '-=20');
```

One interesting side note—and yet another example of how jQuery makes your life easier—is that the normally problematic `opacity` property will work perfectly across browsers (even older ones) by passing in a value between `0.0` and `1.0`; no more messing with the old IE alpha filters!

Now let's see an example of use of the second signature of the `css()` method:

```
$('p').css({
   margin: '1em',
   color: '#FFFFFF',
   opacity: 0.8
});
```

This code will set the values specified to all the elements in the set. But what if you want to create a descending opacity effect with your elements?

As in the shortcut version of the `attr()` method, you can use functions as values to any CSS property in the property's parameter object, and they will be called on each element in the set to determine the value that should be applied. You can achieve this task by using a function as the value of `opacity` instead of a fixed number:

```
$('p').css({
   margin: '1em',
   color: '#1933FF',
   opacity: function (index, currentValue) {
      return 1 - ((index % 10) / 10);
   }
});
```

An example of a page using this code can be found in file chapter-5/descending.opacity.html and as a JS Bin (http://jsbin.com/cuhexe/edit?html,js,output).

Lastly, let's discuss how you can use `css()` with a name or an array of names passed in to retrieve the computed style of the property or properties associated with that name(s) of the first element in the jQuery object. When we say *computed* style, we mean the style after all linked, embedded, and inline CSS has been applied.

Method syntax: css

`css(name)`

Retrieves the computed value or values of the CSS property or properties specified by `name` for the first element in the set

Parameters

`name`	(String|Array) Specifies the name of a CSS property or array of CSS properties whose computed value is to be returned

Returns

The computed value as a string or an object of property-value pairs

This variant of the `css()` method always returns values as a string, so if you need a number or some other type, you'll need to parse the returned value using `parseInt()` or `parseFloat()` depending on the situation.

To understand how the getter version of `css()` works when you pass an array of names, let's see an example. The goal is to print on the console the property and its corresponding value of an element having `special` as its class for the following properties: `font-size`, `color`, and `text-decoration`. To complete the task, you have to write this:

```
var styles = $('.special').css([
   'font-size', 'color', 'text-decoration'
]);

for(var property in styles) {
   console.log(property + ': ' + styles[property]);
}
```

Retrieves the object with the property-value pairs

Loops over the object

This code can be found in the file chapter-5/css.and.array.html and as a JS Bin (http://jsbin.com/mimixu/edit?html,css,js,console,output). Loading the page (or the JS Bin) in your browser, you can see how the values printed are the result of the combination of all the styles defined in the page. For example, the value printed for `font-size` isn't `"20px"` but `"24px"`. This happens because the value defined for the `special` class (`24px`) has more specificity than the one defined for the `div` elements (`20px`).

The `css()` method is another example of how jQuery solves a lot of cross-browser incompatibilities for you. To achieve this goal using native methods, you should use `getComputedStyle()` in all the versions of Chrome, Firefox, Opera, Safari, and Internet Explorer starting from version 9 and use the `currentStyle` and `runtimeStyle` properties in Internet Explorer 8 and below.

Before moving on, we want to highlight two important facts. The first fact is that different browsers may return CSS color values that are logically but not textually equal. For example, if you have a declaration like `color: black;` some browsers may return `#000`, `#000000`, or `rgb(0, 0, 0)`. The second is that the retrieval of shorthand CSS properties such as `margin` or `border` is not guaranteed by jQuery (although it works in some browsers).

For a small set of CSS values that are commonly accessed, jQuery provides convenience methods that access these values and convert them to the most commonly used types.

GETTING AND SETTING DIMENSIONS

When it comes to CSS styles that you want to set or get on your pages, is there a more common set of properties than the element's width or height? Probably not, so jQuery makes it easy for you to deal with the dimensions of the elements as numeric values rather than strings.

Specifically, you can get (or set) the width and height of an element as a number by using the convenient `width()` and `height()` methods. You can set the width or height as follows.

Method syntax: width and height

`width(value)`
`height(value)`
Sets the width or height of all elements in the matched set.

Parameters

`value`	(Number|String|Function) The value to be set. This can be a number of pixels or a string specifying a value in units (such as `px`, `em`, or `%`). If no unit is specified, `px` is the default. If a function is provided, the function is invoked for each element in the set, passing that element as the function context (`this`). The function is passed two values: the element index and the element's current value. The function's returned value is used as the new value.

Returns

The jQuery collection.

Keep in mind that these are shortcuts for the `css()` method, so

```
$('div').width(500);
```

is identical to

```
$('div').css('width', 500);
```

You can also retrieve the width or height as follows.

Method syntax: width and height

`width()`
`height()`
Retrieves the width or height of the first element of the jQuery object

Parameters

`none`

Returns

The computed width or height as a number in pixels; `null` if the jQuery object is empty

The getter version of these two methods is a bit different from its `css()` counterpart. `css()` returns a string containing the value and the unit measure (for example, `"40px"`), whereas `width()` and `height()` return a number, which is the value without the unit and converted into a `Number` data type. If your style defines the width or height using units different from pixels (`em`, `%`, and so on), jQuery will still return the value relative to the width or height of the element in pixels.

jQuery 3: Bug fixed

jQuery 3 fixes a bug of the `width()`, `height()`, and all the other related methods. These methods will no longer round to the nearest pixel, which made it hard to position elements in some situations. To understand the problem, let's say that you have three elements with a width of 33% inside of a container element that has a width of 100px:

```
<div class="wrapper">
    <div>Hello</div>
    <div>Hi</div>
    <div>Bye</div>
</div>
```

Prior to jQuery 3, if you tried to retrieve the width of one of the three children elements as follows

```
$('.wrapper div:first').width();
```

you'd obtain the value `33` as the result because jQuery rounds the value 33.33333. In jQuery 3 this bug has been fixed, so you'll obtain more accurate results.

The fact that the width and height values are returned from these functions as numbers isn't the only convenience that these methods bring to the table. If you've ever tried to find the width or height of an element by looking at its `style.width` or `style.height` properties, you were confronted with the sad truth that these properties are only set by the corresponding `style` attribute of that element; to find out the dimensions of an element via these properties, you have to set them in the first place. Not exactly a paragon of usefulness!

The `width()` and `height()` methods, on the other hand, compute and return the size of the element. Knowing the precise dimensions of an element in simple pages that let their elements lay out wherever they end up isn't usually necessary, but knowing such dimensions in highly interactive scripted pages is crucial to being able to correctly place active elements, such as context menus, custom tool tips, extended controls, and other dynamic components.

Let's put them to work. Figure 5.2 shows a sample page that was set up with two primary elements: a `div` serving as a test subject that contains a paragraph of text (with a border and background color for emphasis) and a second `div` in which to display the dimensions. To write the dimensions in the second `div`, we'll use the `html()` method that we'll cover shortly.

The dimensions of the test subject aren't known in advance because no style rules specifying dimensions are applied. The width of the element is determined by the width of the browser window, and its height depends on how much room will be needed to display the contained text. Resizing the browser window will cause both dimensions to change.

In our page, we define a function that will use the `width()` and `height()` methods to obtain the dimensions of the test subject `div` (identified as `test-subject`) and display the resulting values in the second `div` (identified as `display`):

```
function displayDimensions() {
   $('#display').html(
      $('#test-subject').width() + 'x' + $('#test-subject').height()
   );
}
```

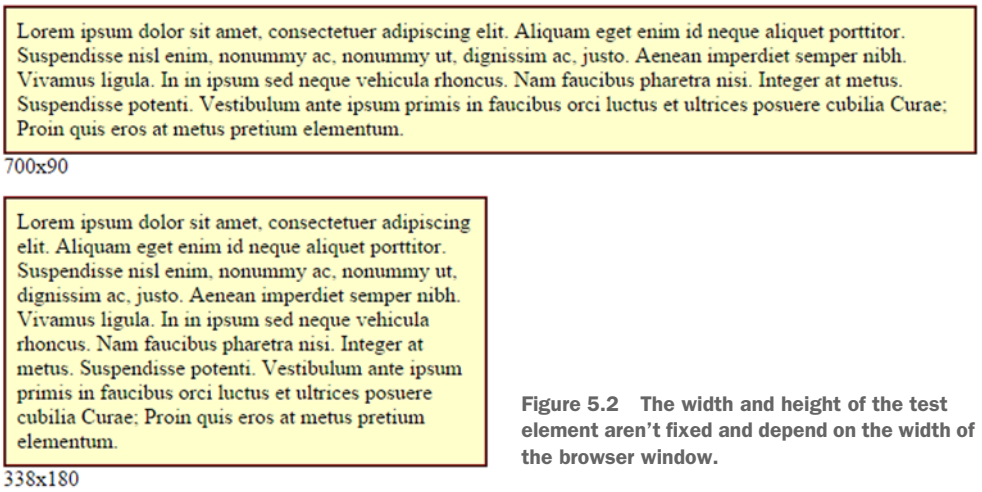

Figure 5.2 The width and height of the test element aren't fixed and depend on the width of the browser window.

We call this function as the last statement of our script, resulting in the initial values being displayed, as shown in the upper portion of figure 5.2.

We also add a call to the same function in a resize handler on the window that updates the display whenever the browser window is resized (you'll learn how to do this in chapter 6), as shown in the lower portion of figure 5.2. The full code of this page is shown in the following listing and can be found in the file chapter-5/dimensions.html.

Listing 5.1 Dynamically tracking and displaying the dimensions of an element

```
<!DOCTYPE html>
<html>
   <head>
      <title>Dynamic Dimensions Example - jQuery in Action, 3rd edition</
     title>
      <link rel="stylesheet" href="../css/main.css"/>
      <style>
         #test-subject
         {
            background-color: #FFFFCC;
            border: 2px ridge maroon;
            padding: 0.5em;
         }
      </style>
   </head>
   <body>
      <div id="test-subject">
         Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
         Aliquam eget enim id neque aliquet porttitor. Suspendisse
         nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
         Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
         sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
         Integer at metus. Suspendisse potenti. Vestibulum ante
```

Declares test subject with dummy text

```
        ipsum primis in faucibus orci luctus et ultrices posuere
        cubilia Curae; Proin quis eros at metus pretium elementum.
    </div>
    <div id="display"></div>                    <-- Displays dimensions in this area

    <script src="../js/jquery-1.11.3.min.js"></script>
    <script>
        function displayDimensions() {          <-- Defines a function that displays width and height of test subject
            $('#display').html(
                $('#test-subject').width() + 'x' +
                $('#test-subject').height()
            );
        }

        $(window).resize(displayDimensions);    <-- Establishes resize handler that invokes display function
        displayDimensions();                    <-- Invokes the function to show the initial values
    </script>
  </body>
</html>
```

In addition to the convenient `width()` and `height()` methods, jQuery also provides similar methods for getting more particular dimension values, as described in table 5.1.

Table 5.1 Additional jQuery dimension-related methods

Method	Description
`innerHeight()`	Returns the inner height of the first matched element, which excludes the border but includes the padding. The value returned is of type `Number` unless the jQuery object is empty, in which case `null` is returned. If a number is returned, it refers to the value of the inner height in pixels.
`innerHeight(value)`	Sets the inner height of all the matched elements with the value specified by `value`. The type of value can be `String`, `Number`, or `Function`. The default unit used is `px`. If a function is provided, it's called for every element in the jQuery object. The function is passed two values: the index position of the element within the jQuery object and the current inner height value. Within the function, `this` refers to the current element within the jQuery object. The returned value of the function is set as the new value of the inner height of the current element.
`innerWidth()`	Same as `innerHeight()` except it returns the inner width of the first matched element, which excludes the border but includes the padding.
`innerWidth(value)`	Same as `innerHeight(value)` except the value is used to set the inner width of all the matched elements.
`outerHeight ([includeMargin])`	Same as `innerHeight()` except it returns the outer height of the first matched element, which includes the border and the padding. The `includeMargin` parameter causes the margin to be included if it's `true`.
`outerHeight(value)`	Same as `innerHeight(value)` except the value is used to set the outer height of all the matched elements.

Table 5.1 Additional jQuery dimension-related methods *(continued)*

Method	Description
`outerWidth([includeMargin])`	Same as `innerHeight()` except it returns the outer width of the first matched element, which includes the border and the padding. The `includeMargin` parameter causes the margin to be included if it's `true`.
`outerWidth(value)`	Same as `innerHeight(value)` except the value is used to set the outer width of all the matched elements.

You're not finished yet; jQuery also gives you easy support for positions and scrolling values.

POSITIONS AND SCROLLING

jQuery provides two methods for getting the position of an element. Both of these methods return a JavaScript object that contains two properties: `top` and `left`, which indicate the top and left values of the element.

The two methods use different origins from which their relative computed values are measured. One of these methods, `offset()`, returns the position relative to the document.

Method syntax: offset

`offset()`

Returns the current coordinates (in pixels) of the first element in the set, relative to the document.

Parameters

`none`

Returns

An object with `left` and `top` properties as numbers depicting the position in pixels relative to the document.

This method can also be used to set the current coordinates of one or more elements.

Method syntax: offset

`offset(coordinates)`

Sets the current coordinates (in pixels) of all the elements in the set, relative to the document.

Parameters

`coordinates`	(Object|Function) An object containing the properties `top` and `left`, which are numbers indicating the new top and left coordinates for the elements in the set. If a function is provided, it's invoked for each element in the set, passing that element as the function context (`this`) and passing two values: the element index and the object containing the current values of `top` and `left`. The function's returned object is used to set the new values.

Returns

The jQuery collection.

The other method, `position()`, returns values relative to an element's closest offset parent. The *offset parent* of an element is the nearest ancestor that has an explicit positioning rule of `relative`, `absolute`, or `fixed` defined. The syntax of `position()` is as follows.

Method syntax: position

`position()`

Returns the position (in pixels) of the first element in the set relative to the element's closest offset parent

Parameters

`none`

Returns

An object with `left` and `top` properties as numbers depicting the position in pixels relative to the closest offset parent

Both `offset()` and `position()` can only be used for visible elements.

In addition to element positioning, jQuery gives you the ability to get and set the scroll bar position of an element. Table 5.2 describes these methods that work with both visible and hidden elements.

Table 5.2 The jQuery scroll bar control methods

Method	Description
`scrollLeft()`	Returns the horizontal position of the scroll bar of the first matched element. The value returned is of type `Number` unless the jQuery object is empty, in which case `null` is returned. If a number is returned, it refers to the value of the position in pixels.
`scrollLeft(value)`	Sets the horizontal position of the scroll bar for all matched elements of `value` pixels. This method returns the jQuery set it has been called upon.
`scrollTop()`	Same as `scrollLeft()` except it returns the vertical position of the scroll bar of the first matched element.
`scrollTop(value)`	Same as `scrollLeft(value)` except the value is used to set the vertical position of the scroll bar for all the matched elements.

Now that you've learned how to get and set the horizontal and vertical position of the scroll bar of elements using jQuery, let's see an example.

Imagine that you have an element with an ID of `elem` shown in the middle of your page and that after one second you want to move it to the top-left corner of the document. The point we've described has as its coordinates [0, 0], which means that to move it there you have to set both `left` and `top` to `0`. To achieve the goal just described, all you need is these few lines of code:

```
setTimeout(function() {
   $('#elem').offset({
```

```
      left: 0,
      top: 0
   });
}, 1000);
```

Let's now discuss different ways of modifying an element's contents.

5.2 Setting element content

When it comes to modifying the contents of elements, there are a lot of different methods you can employ, depending on the type of the text you want to inject. If you're interested in setting a text whose content should not be parsed as markup, you can use properties like `textContent` or `innerText`, depending on the browser.

Once again jQuery saves you from these browser incompatibilities by giving you a number of methods that you can employ.

5.2.1 Replacing HTML or text content

First up is the simple `html()` method, which allows you to retrieve the HTML content of an element when used without parameters or, as you've seen with other jQuery methods, to set the content of all the elements in the set when used with a parameter.

Here's how to get the HTML content of an element.

Method syntax: html

`html()`

Obtains the HTML content of the first element in the matched set,

Parameters

`none`

Returns

The HTML content of the first matched element.

And here's how to set the HTML content of all the matched elements.

Method syntax: html

`html(content)`

Sets the passed HTML fragment as the content of all matched elements.

Parameters

`content`	(String|Function) The HTML fragment to be set as the element's content. If a function, the function is invoked for each element in the set, setting that element as the function context (`this`). The function is passed two values: the element index and the existing content. The function's returned value is used as the new content.

Returns

The jQuery collection.

Let's say that in your page you have the following element:

```
<div id="message"></div>
```

You're running a function you've developed, and once it ends you need to show a message that contains some content to your user. You can perform this task with a statement like the following:

```
$('#message').html('<p>Your current balance is <b>1000$</b></p>');
```

This statement will cause your previous element to be updated as reported here:

```
<div id="message"><p>Your current balance is <b>1000$</b></p></div>
```

In this case, the tags passed to the method will be processed as HTML. The total balance, for instance, will be shown in bold.

In addition to setting the content as HTML, you can also set or get only the text contents of elements. The `text()` method, when used without parameters, returns a string that's the concatenation of all the texts in the matched set. For example, let's say you have the following HTML fragment:

```
<ul id="the-list">
  <li>One</li><li>Two</li><li>Three</li><li>Four</li>
</ul>
```

The statement

```
var text = $('#the-list').text();
```

results in the variable `text` being set to `OneTwoThreeFour`. Note that if there are white spaces or new lines in between elements (for example between a closing `</li>` and an opening `<li>`) they'll be included in the resulting string.

The syntax of this method is as follows.

Method syntax: text

`text()`

Retrieves the combined text contents of each element in the set of matched elements, including their descendants.

Parameters

`none`

Returns

A string of all the text contents.

You can also use the `text()` method to set the text content of the elements in the jQuery object. The syntax for this format is as follows.

Method syntax: text

`text(content)`

Sets the text content of all elements in the set to the passed value. If the passed text contains angle brackets (< and >) or the ampersand (&), these characters are replaced with their equivalent HTML entities.

Parameters

`content`	(String|Number|BooleanFunction) The text content to be set into the elements in the set. When the value is of type `Number` or `Boolean`, it'll be converted to a `String` representation. Any angle bracket characters are escaped as HTML entities. If a function, it's invoked for each element in the set, setting that element as the function context (`this`). The function is passed two values: the element index and the existing text. The function's returned value is used as the new content.

Returns

The jQuery collection.

Changing the inner HTML or text of elements using these methods will replace the contents that were previously in the elements, so use these methods carefully. jQuery isn't limited to these methods only, so let's take a look at the others.

5.2.2 Moving elements

Manipulating the DOM of a page without the necessity of a page reload opens a world of possibilities for making your pages dynamic and interactive. You've already seen a glimpse of how jQuery lets you create DOM elements on the fly. These new elements can be attached to the DOM in a variety of ways, and you can also move (and copy and move) existing elements.

To add content to the end of existing content, the `append()` method is available.

Method syntax: append

`append(content[, content, ..., content])`

Appends the passed argument(s) to the content of all matched elements. This method accepts an arbitrary number of arguments with a minimum of one.

Parameters

`content`	(String|Element|jQuery|Array|Function) A string, a DOM element, an array of DOM elements, or a jQuery object to be appended. If a function is provided, the function is invoked for each element in the set, setting that element as the function context (`this`). The function is passed two values: the element index and the existing contents of that element. The function's returned value is used as the content to append.

Returns

The jQuery collection.

Let's see an example of the use of this method. Consider the following simple case:

```
$('p').append('<b>some text<b>');
```

This statement appends the HTML fragment created from the passed string to the end of the existing content of all p elements on the page.

A more complex use of this method identifies existing elements of the DOM as the items to be appended. Consider the following:

```
$('p.append-to').append($('a.append'));
```

This statement moves all a elements with the class append to the end of the content of all p elements having class append-to. If there are multiple targets (the elements of the jQuery object the append() method is called upon) for the operation, the original element is cloned as many times as is necessary and then appended. In all cases, the original is removed from its initial location.

This operation is semantically a *move* if one target is identified; the original source element is removed from its initial location and appears at the end of the target's list of children.

Consider the following HTML code:

```
<a href="http://www.manning.com" class="append">Text</a>
<p class="append-to"></p>
```

By running the previous statement, you'll end up with the following markup:

```
<p class="append-to">
   <a href="http://www.manning.com" class="append">Text</a>
</p>
```

The operation can also be a copy-and-move operation if multiple targets are identified, creating enough copies of the original so that each target can have one appended to its children.

In place of a full-blown set, you can also reference a specific DOM element, as shown:

```
$('p.appendToMe').append(someElement);
```

Another example of use for this method is the following:

```
$('#message').append(
   '<p>This</p>',
   [
      '<p>is</p>',
      $('<p>').text('my')
   ],
   $('<p>text</p>')
);
```

In this code you can see how the append() method can manage multiple arguments and each argument can be of a different type (a string, an array, and a jQuery object). The result of running it is that you'll have an element having an ID of message with four paragraphs that compose the sentence "This is my text."

Although it's a common operation to add elements to the end of an element's content—you might be adding a list item to the end of a list, a row to the end of a table, or

adding a new element to the end of the document body—you might also need to add a new or existing element to the start of the target element's contents.

When such a need arises, the `prepend()` method will do the trick.

Method syntax: prepend

`prepend(content[, content, ..., content])`

Prepends the passed argument(s) to the content of all matched elements. This method accepts an arbitrary number of arguments with a minimum of one.

Parameters

`content`	Same as the `content` parameter of `append()` except the argument(s) are prepended to the content of each element in the set of matched elements.

Returns

The jQuery collection.

Sometimes you might wish to place elements somewhere other than at the beginning or end of an element's content. jQuery allows you to place new or existing elements anywhere in the DOM by identifying a target element that the source elements are to be placed before or after.

Not surprisingly, the methods are named `before()` and `after()`. Their syntax should seem familiar by now.

Method syntax: before

`before(content[, content, ..., content])`

Inserts the passed argument(s) into the DOM as siblings of the target elements, positioned before the targets. The target elements in the set must already be part of the DOM. This method accepts an arbitrary number of arguments with a minimum of one.

Parameters

`content`	Same as the `content` parameter of `append()` except the argument(s) is inserted before each element in the set of matched elements.

Returns

The jQuery collection.

Method syntax: after

`after(content[, content, ..., content])`

Inserts the passed argument(s) into the DOM as siblings of the target elements positioned after the targets. The target elements in the set must already be part of the DOM. This method accepts an arbitrary number of arguments with a minimum of one.

Parameters

`content`	Same as the `content` parameter of `append()` except the argument(s) is inserted after each element in the set of matched elements.

Returns

The jQuery collection.

These operations are crucial to manipulating the DOM effectively in your pages, so we've provided a Move and Copy Lab Page so that you can play around with these operations until you thoroughly understand them. This lab is available at chapter-5/lab.move.and.copy.html, and its initial display is as shown in figure 5.3.

The left pane of this Lab contains three images that can serve as sources for your move/copy experiments. Select one or more of the images by checking their corresponding check boxes.

Targets for the move/copy operations are in the right pane and are also selected via check boxes. Controls at the bottom of the pane allow you to select one of the four operations to apply: append, prepend, before, or after. (Ignore "clone" for now; we'll attend to that later.)

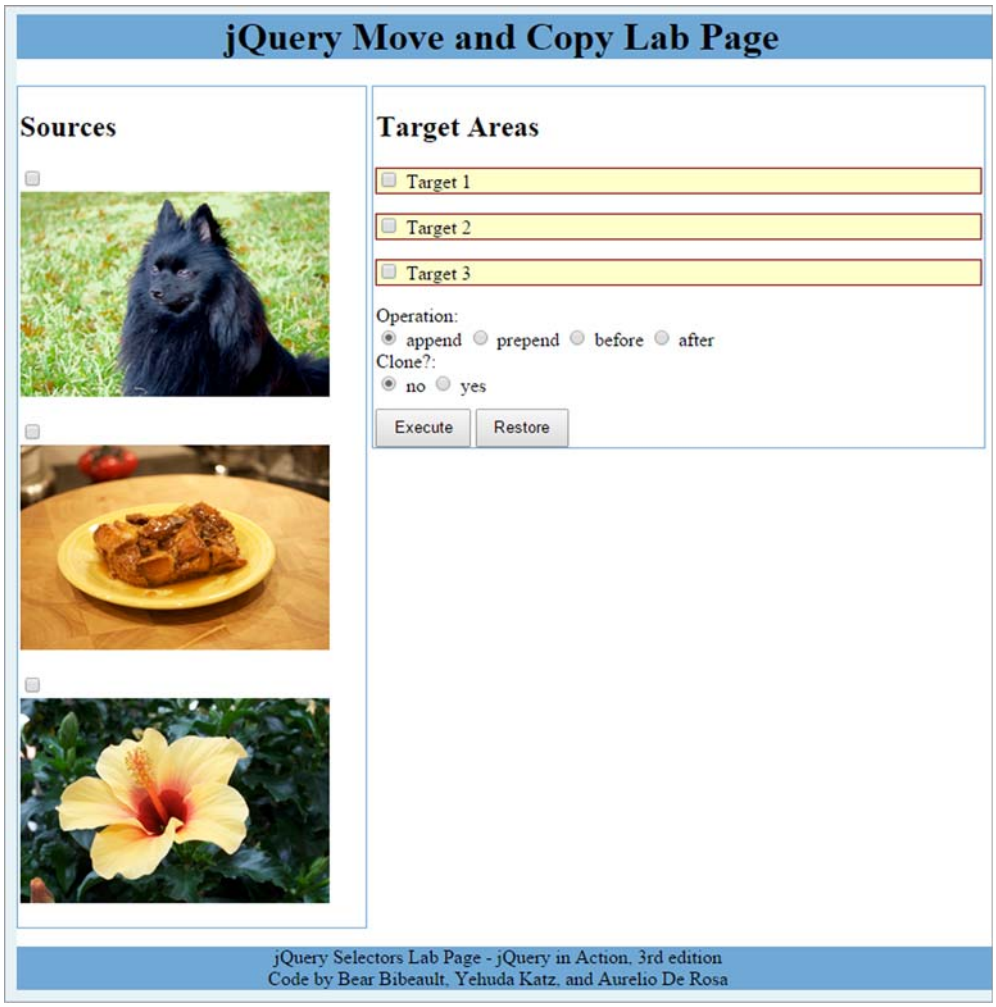

Figure 5.3 The Move and Copy Lab Page will let you inspect the operation of the DOM manipulation methods.

The Execute button causes any source images you've selected to be applied to a set of the selected targets using the specified operation. When you want to put everything back into place so you can run another experiment, use the Restore button.

Let's run an append experiment. Select the dog image and then select Target 2. Leaving the append operation selected, click Execute. The result of this operation is shown in figure 5.4.

Figure 5.4 Cozmo has been added to the end of Target 2 as a result of the append operation.

Use the Move and Copy Lab Page to try various combinations of sources, targets, and the four operations until you have a good feel for how they operate.

Sometimes it might make the code more readable if you could reverse the order of the elements passed to these operations. If you want to move or copy an element from one place to another, a possible approach would be to wrap the source elements (rather than the target elements) and to specify the targets in the parameters of the method. Well, jQuery lets you do that by providing analogous operations to the four that we just examined, reversing the order in which sources and targets are specified. They are `appendTo()`, `prependTo()`, `insertBefore()`, and `insertAfter()`, and are described in table 5.3.

Table 5.3 Additional methods to move elements in the DOM

Method	Description
`appendTo(target)`	Inserts every element in the set of matched elements to the end of the content of the specified target(s). The argument provided (`target`) can be a string containing a selector, an HTML string, a DOM element, an array of DOM elements, or a jQuery object. The method returns the jQuery object it was called upon.
`prependTo(target)`	Same as `appendTo(target)` except the elements in the set of matched elements are inserted at the beginning of the content of the specified target(s).
`insertBefore(target)`	Same as `appendTo(target)` except the elements in the set of matched elements are inserted before the specified target(s).
`insertAfter(target)`	Same as `appendTo(target)` except the elements in the set of matched elements are inserted after the specified target(s).

Wow, that's a lot of stuff to learn all at once. To help you digest this bunch of new methods, we'll show you a couple of examples.

EXAMPLE #1 - MOVING ELEMENTS

Let's say you have the following HTML code in a page:

```
<div id="box">
   <p id="description">jQuery is so awesome!</p>
   <button id="first-btn">I'm a button</button>
   <p id="adv">jQuery in Action rocks!</p>
   <button id="second-btn">Click me</button>
</div>
```

This code will be rendered by Chrome as shown in figure 5.5a.

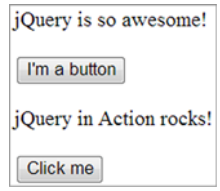

Figure 5.5a The HTML code rendered by Chrome

Your first goal will be to move all the buttons before the first paragraph, the one having `description` as its ID. To perform this task you can use the `insertBefore()` method:

```
$('button').insertBefore('#description');
```

Because you're a good and observant reader, you're thinking "Hey, I can do that using the `before()` method by just switching the selectors and wrapping the target one with the `$()` function!" Congratulations, you're right! The previous statement can be equivalently turned into this:

```
$('#description').before($('button'));
```

Figure 5.5b The HTML snippet rendered by Chrome after the execution of the statement

Once executed, regardless of which of the two previous statements you run, the page will be updated as shown in figure 5.5b.

You can see this code in action accessing the related JS Bin (http://jsbin.com/ARedIWU/edit?html,js,output) or the file

chapter-5/moving.buttons.html (as soon as the page is loaded, the buttons will be moved). Now let's see a slightly more complex example.

EXAMPLE #2 - COPYING AND MERGING CONTENT

Imagine you have the same markup as the previous example and you want to create a new paragraph having the content equal to the union of the two paragraphs inside the `div` and put it right after the `div` itself. The content of this newly created paragraph will be "jQuery is so awesome! jQuery in Action rocks!" You can do that by combining two of the methods we explained in this chapter: `text()` and `after()`. The code that implements this request is listed here:

```
var $newParagraph = $('<p></p>').text(
   $('#description').text() + ' ' + $('#adv').text()
);
$('#box').after($newParagraph);
```

These two examples should prove how the more methods you discover, the more power you have in your hands. And we're not finished yet! There's one more thing we need to address before we move on. Sometimes, rather than inserting elements *into* other elements, you want to do the opposite. Let's see what jQuery offers for that.

5.2.3 Wrapping and unwrapping elements

Another type of DOM manipulation that you'll often need to perform is to wrap an element (or a set of elements) in some markup. You might want to wrap all links of a certain class inside a `<div>`. You can accomplish such DOM modifications by using jQuery's `wrap()` method.

Method syntax: wrap

`wrap(wrapper)`

Wraps the elements in the jQuery object with the argument provided.

Parameters

`wrapper`	(String|Element|jQuery|Function) A string containing the opening tag (and optionally the closing tag) of the element with which to wrap each element of the matched set. The argument can also be an element, a jQuery object, or a selector specifying the elements to be cloned and serve as the wrapper. If a selector matching more than one element or a jQuery object is passed, only the first element is used as the wrapper. If a function is provided, the function is invoked for each element in the set, passing that element as the function context (`this`) and passing one parameter to the function that's the element index. The function's returned value, which can be an HTML fragment or a jQuery object, is used to wrap the current element of the set.

Returns

The jQuery collection.

To understand what this method does, let's say that you have the following markup:

```
<a class="surprise">A text</a>
<a>Hi there!</a>
```

```
<a class="surprise">Another text</a>
```

To wrap each link with class `surprise` with a `<div>` having class `hello`, you could write

```
$('a.surprise').wrap('<div class="hello"></div>');
```

The result of running such a statement would be the following:

```
<div class="hello"><a class="surprise">A text</a></div>
<a>Hi there!</a>
<div class="hello"><a class="surprise">Another text</a></div>
```

If you wanted to wrap the link in a clone of a hypothetical first `div` element on the page, you could write

```
$('a.surprise').wrap($('div:first'));
```

or alternatively

```
$('a.surprise').wrap('div:first');
```

Remember that in this case the content of the `div` will also be cloned and used to surround the `a` elements.

When multiple elements are collected in a jQuery object, the `wrap()` method operates on each one individually. If you'd rather wrap all the elements in the jQuery object as a unit, you could use the `wrapAll()` method instead.

Method syntax: wrapAll

`wrapAll(wrapper)`
Wraps the elements of the matched set, as a unit, with the argument provided

Parameters

`wrapper`	Same as the `wrapper` parameter of `wrap()`

Returns
The jQuery collection

jQuery 3: Bug fixed

jQuery 3 fixes a bug of the `wrapAll()` method that occurred when passing a function to it. Prior to jQuery 3, when passing a function to `wrapAll()`, it wrapped the elements of the matched set individually instead of wrapping them as a unit. Its behavior was the same as passing a function to `wrap()`.

In addition to fixing this issue, because in jQuery 3 the function is called only once, it isn't passed the index of the element within the jQuery object. Finally, the function context (`this`) now refers to the first element in the matched set.

Sometimes you may not want to wrap the elements in a matched set but rather their *contents*. For just such cases, the `wrapInner()` method is available.

Method syntax: wrapInner

`wrapInner(wrapper)`
Wraps the contents, including text nodes, of the elements in the matched set with the argument provided

Parameters

`wrapper`	Same as the `wrapper` parameter of `wrap()`

Returns
The jQuery collection

The converse operation of `wrap()`, which is removing the parent of a child element, is possible with the `unwrap()` method.

Method syntax: unwrap

`unwrap()`
Removes the parent element of the elements in the set. The child element, along with any siblings, replaces the parent element in the DOM.

Parameters

`none`

Returns
The jQuery collection.

jQuery 3: Parameter added

jQuery 3 adds an optional `selector` parameter to `unwrap()`. You can pass a string containing a selector expression to match the parent element against. In case there is a match, the child elements are unwrapped; otherwise, the operation isn't performed.

Before moving on, let's see an example of the use of these methods.

HOW TO WRAP LABEL-INPUT AND LABEL-TEXTAREA PAIRS OF A FORM

Imagine that you have the following contact form:

```
<form id="contact" method="post">
   <label for="name">Name:</label>
   <input name="name" id="name" />
   <label for="email">Email:</label>
   <input name="email" id="email" />
   <label for="subject">Subject:</label>
   <input name="subject" id="subject" />
   <label for="message">Message:</label>
```

```
   <textarea name="message" id="message"></textarea>
   <input type="submit" value="Submit" />
</form>
```

You want to wrap every `label-input` or `label-textarea` pair in a `<div>` having the class `field`. You define this class as follows:

```
.field
{
   border: 1px solid black;
   margin: 5px 0;
}
```

Wrapping the pairs means that you don't want to wrap each element inside the form individually. To achieve this goal, you'll need to use some of the knowledge you've acquired throughout the first chapters of the book. But don't worry! This can be a good test to see if you've digested the concepts explained so far or if you need a quick refresher. One of the possible solutions is the following:

Selects all the form's <input>s and <textarea>s and processes them individually

```
$('input, textarea', '#contact').each(function(index, element) {
    var $this = $(this);
    $this
       .add($this.prev('label'))
       .wrapAll('<div class="field"></div>');
});
```

Caches the set containing the current element only

**Wraps the pair inside the **

Adds to the set the preceding sibling element only if it's a <label>

To perform the exercise, you need to select all the `<input>`s and the `<textarea>`s inside the `<form>` (for the sake of brevity we're ignoring other tags such as `<select>`) and find a way to tie them with the related `<label>`. Then you have to process each pair individually. To do that, after selecting the elements you need, use the `each()` method we covered in chapter 3. Inside the anonymous function passed to it, wrap the current element using `$()` and then store it in a variable because you're going to use it twice. Then add to the element in the set the preceding sibling element only if it's a `label` element. At this point, you have a set with the two elements you need, so you wrap this set with the `<div>` as required. Yeah, mission accomplished!

Figure 5.6a shows the `<form>` before running the code and figure 5.6b shows it after.

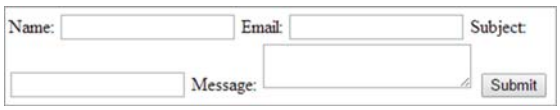

Figure 5.6a The form before executing the code

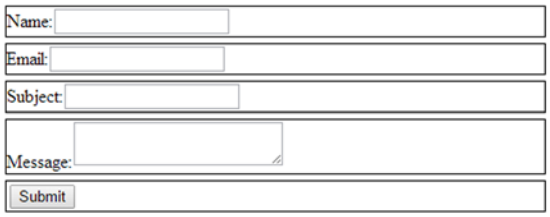

Figure 5.6b The form after executing the code. Note how each `label-input` or `label-textarea` pair is surrounded by a black border, proving that they've been wrapped by the `<div>` as required.

In case you want to play further with this example, it's available as a JS Bin (http://jsbin.com/IrUhEfAg/edit?html,css,js,output) and in the file chapter-5/wrapping.form.elements.html.

So far you've learned how to perform many operations. It's now time to learn how to remove elements from the DOM.

5.2.4 Removing elements

Sometimes you might need to remove elements that are no longer needed. If you want to remove a set of elements and all their content, you can use the `remove()` method, whose syntax is as follows.

Method syntax: remove

`remove([selector])`

Removes all elements and their content in the set from the page, including event listeners attached and any data stored

Parameters

`selector`	(String) An optional selector that further filters which elements of the set are to be removed

Returns

The jQuery collection

Note that as with many other jQuery methods, the set is returned as the result of this method. The elements that were removed from the DOM are still referenced by this set (and hence are not yet eligible for garbage collection) and can be further operated upon using other jQuery methods including `appendTo()`, `prependTo()`, `insertBefore()`, `insertAfter()`. But any data stored or event listener added to the removed element is lost.

If you want to remove the elements from the DOM but retain any bound events and data (that you might have added using the `data()` method), you can use `detach()`.

Method syntax: detach

`detach([selector])`

Removes all elements and their content in the set from the page DOM, retaining any bound events and jQuery data

Method syntax: detach *(continued)*

Parameters

`selector`	(Selector) An optional selector string that further filters which elements of the set are to be detached

Returns

The jQuery collection.

The `detach()` method is the preferred means of removing an element that you'll want to put back into the DOM at a later time with its events and data intact. A typical situation pulls the element from the DOM, applies several changes to it, and then pushes it again in the DOM. Doing so will improve the performance of your code because modifying a detached element is faster than applying all the changes to one or more elements that are currently in the DOM.

To completely empty DOM elements of their contents but retain the elements themselves, you can use the `empty()` method. Its syntax is as follows.

Method syntax: empty

`empty()`

Removes the content of all DOM elements in the matched set.

Parameters

`none`

Returns

The jQuery collection.

This method is useful when you deal with injecting external content fetched using Ajax. Let's say that you fetched some new content and now you need to add it inside your page in a `<div>` having `content` as its ID. You can perform this task with the following code:

```
var newContent = '<p>Wow, this new content is awesome!</p>';
$('#content')
    .empty()
    .html(newContent);
```

Content ideally fetched from an external resource

Retrieves the element

Removes its content

Injects the fetched content inside it as HTML

Remember to pay attention when you inject external content inside your page using the `html()` method because you may be exposed to attacks such as XSS (cross-site scripting) and CSRF (cross-site request forgery).

Removing elements is nice, but sometimes you need to clone elements.

5.2.5 Cloning elements

One more way that you can manipulate the DOM is to make copies of elements to attach elsewhere in the tree. jQuery provides a handy wrapper method for doing so with its `clone()` method.

Method syntax: clone

`clone([copyHandlersAndData[, copyChildrenHandlersAndData]])`

Creates a deep copy of the elements in the jQuery collection and returns a new jQuery collection that contains them. The elements and any children are copied. Event handlers and data are optionally copied depending on the setting of the `copyHandlersAndData` parameter.

Parameters

`copyHandlersAndData`	(Boolean) If `true`, event handlers and data are copied. If `false` or omitted, handlers and data aren't copied.
`copyChildrenHandlersAndData`	(Boolean) If `true`, copies the event handlers and the data for all the children of the cloned elements. If omitted, if the first parameter is provided, the same value is used; otherwise `false` is assumed. If `false`, the event handlers and the data aren't copied.

Returns

The newly created jQuery collection.

Making copies of existing elements with `clone()` isn't useful unless you do something with the copies. Generally, once the set containing the clones is generated, another jQuery method is applied to stick them somewhere in the DOM. For example,

```
$('img').clone().appendTo('fieldset.photo');
```

makes copies of all `img` elements and appends them to all `fieldset` elements with the class name `photo`.

A slightly more interesting example is as follows:

```
$('ul').clone(true).insertBefore('#here');
```

This method's chain performs a similar operation, but the targets of the cloning operation—all `ul` elements—are copied *including* their data and event handlers. In addition, because the `clone()` method clones children, too, and it's likely that any `ul` element will have a number of `li` children, you're sure that no information is lost. Because you omit the second argument but specify the first, the data and event handlers of all the children are also copied.

Before moving to another topic, let's discuss one last example. Imagine that you have a set of links with an image inside them. Both links and images have some data and events handlers attached. You want to copy all of them and place the copies after all the elements inside the first `div` of the page. In addition, you want to retain only the data and the handlers of the links, not those of the images inside. This task is performed with the following statement:

```
$('a').clone(true, false).appendTo('div:first');
```

This statement shows you the use of the optional parameters discussed in the description of the method.

In order to see the clone operation in action, return to the Move and Copy Lab Page. Just above the Execute button is a pair of radio buttons that allows you to specify a cloning operation as part of the main DOM manipulation operation. When the Yes radio button is selected, the sources are cloned before the `append()`, `prepend()`, `before()`, and `after()` methods are executed.

Repeat some of the experiments you conducted earlier with cloning enabled, and note how the original sources are unaffected by the operations.

You can insert, remove, and copy. Using these operations in combination, it would be easy to perform higher-level operations such as *replace*. But guess what? You don't need to!

5.2.6 Replacing elements

For those times when you want to replace existing elements with new ones, or to move an existing element to replace another, jQuery provides the `replaceWith()` method.

Method syntax: replaceWith

`replaceWith(content)`

Replaces each matched element with the specific content.

Parameters

`content`	(String|Element|Array|jQuery|Function) A string containing an HTML fragment to become the replaced content, or a DOM element, an array of DOM elements, or a jQuery object containing the elements to be moved to replace the existing elements. If a function, the function is invoked for each element in the set, setting that element as the function context (`this`) and passing no parameters. The function's returned value is used as the new content.

Returns

A jQuery collection containing the replaced elements.

To understand what this method does, let's discuss an example. Imagine that you have the following markup:

```
<img src="image1.jpg" alt="A ball" />
<img src="image2.jpg" alt="A blue bird" />
<img src="image3.jpg" />
```

You want to replace all the images that have an `alt` attribute, one at a time, with `span` elements. The latter will have as their text the value of the `alt` attribute of the image being replaced. Employing `each()` and `replaceWith()`, you could do it like this:

```
$('img[alt]').each(function(){
  $(this).replaceWith('<span>' + $(this).attr('alt') + '</span>');
});
```

The `each()` method lets you iterate over each matched element, and `replaceWith()` is used to replace the images with generated `span` elements. The resulting markup is shown here:

```
<span>A ball</span>
<span>A blue bird</span>
<img src="image3.jpg" />
```

This example shows once again how easy it is to work with jQuery to manipulate the DOM.

The `replaceWith()` method returns a jQuery set containing the elements that were removed from the DOM, in case you want to do something other than just discard them. As an exercise, consider how you'd augment the example code to reattach these elements elsewhere in the DOM after their removal.

When an existing element is passed as the argument to `replaceWith()`, it's detached from its original location in the DOM and reattached to replace the target elements. If there are multiple such targets, the original element is cloned as many times as needed.

At times, it may be convenient to reverse the order of the elements as specified by `replaceWith()` so that the replacing element can be specified using the matching selector. You've already seen such complementary methods, such as `append()` and `appendTo()`, that let you specify the elements in the order that makes the most sense for your code.

Similarly, the `replaceAll()` method mirrors `replaceWith()`, allowing you to perform a similar operation. But in this case the elements to be replaced are defined by the selector passed as arguments and thus are not those the method is called upon.

Method syntax: replaceAll

`replaceAll(target)`

Replaces each element matched by the passed `target` with the set of matched elements to which this method is applied

Parameters

`target`	(String|Element|Array|jQuery) A selector string expression, a DOM element, an array of DOM elements, or a jQuery collection that specifies the elements to be replaced

Returns

A jQuery collection containing the inserted elements

Like `replaceWith()`, `replaceAll()` returns a jQuery collection. But it doesn't contain the replaced elements but rather the *replacing* elements. The replaced elements are lost and can't be further operated upon. Keep this in mind when deciding which replace method to employ.

Based on the description of the `replaceAll()` method, you can achieve the same goal of the previous example by writing the following:

```
$('img[alt]').each(function(){
  $('<span>' + $(this).attr('alt') + '</span>').replaceAll(this);
});
```

Note how you invert the argument passed to `$()` and `replaceAll()`.

Now that we've discussed handling general DOM elements, let's take a brief look at handling a special type of element: the form elements.

5.3 Dealing with form element values

Because form elements have special properties, jQuery's core contains a number of convenience functions for activities such as these:

- Getting and setting their values
- Serializing them
- Selecting elements based on form-specific properties

What's a form element?

When we use the term *form element*, we're referring to the elements that can appear within a form, possess `name` and `value` attributes, and whose values are sent to the server as HTTP request parameters when the form is submitted.

Let's take a look at one of the most common operations you'll want to perform on a form element: getting access to its value. jQuery's `val()` method takes care of the most common cases, returning the `value` attribute of a form element for the first element in the jQuery object. Its syntax is as follows.

Method syntax: val

`val()`

Returns the current value of the first element in the jQuery collection. If the first element is a `<select>` and no option is selected, the method returns `null`. If the element is a multiselect element (a `<select>` having the `multiple` attribute specified) and at least one option is selected, the returned value is an array of all selected options.

Parameters

`none`

Returns

The fetched value or values.

This method, although quite useful, has a number of limitations of which you need to be wary. If the first element in the set isn't a form element, an empty string is returned, which some of you may find misleading. This method doesn't distinguish between the checked or unchecked states of check boxes and radio buttons and will return the value of check boxes or radio buttons as defined by their `value` attribute, regardless of whether they're checked or not.

For radio buttons, the power of jQuery selectors combined with the `val()` method saves the day. Consider a form with a radio group (a set of radio buttons with the same `name`) named `radio-group` and the following expression:

```
$('input[type="radio"][name="radio-group"]:checked').val();
```

This expression returns the value of the single checked radio button (or `undefined` if none are checked). That's a lot easier than looping through the buttons looking for the checked element, isn't it?

Because `val()` considers only the first element in a set, it's not as useful for check box groups where more than one control might be checked. But jQuery rarely leaves you without recourse. Consider the following:

```
var checkboxValues =
   $('input[type="checkbox"][name="checkboxgroup"]:checked').map(function() {
      return $(this).val();
   })
   .toArray();
```

Although the `val()` method is great for obtaining the value of any single form control element, if you want to obtain the complete set of values that would be submitted through a form submission, you'll be much better off using the `serialize()` or `serializeArray()` methods (which you'll see in chapter 10).

Another common operation you'll perform is to *set* the value of a form element. The `val()` method is also used for this purpose by supplying a value. Its syntax is as follows.

Method syntax: val

`val(value)`

Sets the passed value as the `value` of all matched elements. If an array of values is provided, it causes any check boxes, radio buttons, or options of `select` elements in the set to become checked or selected if their `value` properties match any of the values passed in the `values` array.

Parameters

`value`	(String|Number|Array|Function) Specifies the value that is to be set as the `value` property of each element in the set. An array of values will be used to determine which elements are to be checked or selected. If a function, the function is invoked for each element in the set, with that element passed as the function context (`this`), and two values: the element index and the current value of the element. The value returned from the function is taken as the new value to be set.

Returns

The jQuery collection.

As the description of the method specifies, the `val()` method can be used to cause check box or radio elements to become checked or to select options within a `<select>` element. Consider the following statement:

```
$('input[type="checkbox"], select').val(['one', 'two', 'three']);
```

It'll search all the `checkboxes` and `selects` on the page for values that match any of the input strings: `"one"`, `"two"`, or `"three"`. Any element found that matches will become checked or selected. In case of a `select` element without the `multiple` attribute defined, only the first value to match is selected. In our previous code only an option having a value of `one` is selected because in the array passed to `val()` the string `"one"` comes before the strings `"two"` and `"three"`. This makes `val()` useful for much more than the text-based form elements.

5.4 Summary

With the techniques learned in this chapter, you're able to copy elements, move them, replace them, or even remove them. You can also append, prepend, or wrap any element or set of elements on the page. In addition, we discussed how to manage the values of form elements, all leading to powerful yet succinct logic.

With that behind you, you're ready to look at some more advanced concepts, starting with the typically messy job of handling events in your pages.

WEB DEVELOPMENT

jQuery IN ACTION Third Edition

Bear Bibeault • Yehuda Katz • Aurelio De Rosa

Free eBook
SEE INSERT

Thanks to jQuery, no one remembers the bad old days when programmers manually managed browser inconsistencies, CSS selectors support, and DOM navigation, and when every animation was a frustrating exercise in raw JavaScript. The elegant, intuitive jQuery library beautifully manages these concerns, and jQuery 3 adds even more features to make your life as a web developer smooth and productive.

jQuery in Action, Third Edition, is a fast-paced guide to jQuery, focused on the tasks you'll face in nearly any web dev project. In it, you'll learn how to traverse the DOM, handle events, perform animations, write jQuery plugins, perform Ajax requests, and even unit test your code. Its unique Lab Pages anchor each concept in real-world code. This expanded Third Edition adds new chapters that teach you how to interact with other tools and frameworks and build modern single-page web applications.

What's Inside

- Updated for jQuery 3
- DOM manipulation and event handling
- Animations and effects
- Advanced topics including Unit Testing and Promises
- Practical examples and labs

Readers are assumed to have only beginning-level JavaScript knowledge.

Bear Bibeault is coauthor of *Secrets of the JavaScript Ninja*, *Ajax in Practice*, and *Prototype and Scriptaculous in Action*. **Yehuda Katz** is an early contributor to jQuery and cocreator of Ember.js. **Aurelio De Rosa** is a full-stack web developer and a member of the jQuery content team.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/jquery-in-action-third-edition

"Does a great job of showing how all the parts of jQuery fit together and demonstrates important concepts."
—From the Foreword by Dave Methvin, President jQuery Foundation

"The best-thought-out and researched piece of literature on the jQuery library."
—From the Foreword by John Resig, Creator of jQuery

"For three editions now, this is the only jQuery book I recommend to my clients, period."
—Christopher Haupt Mobirobo Inc.

ISBN 13: 978-1-617292-07-1
ISBN 10: 1-617292-07-9
54499
9 781617 292071

MANNING $44.99 / Can $51.99 [INCLUDING eBOOK]