Contents

1 Using jquery Core **3**

1.1 $ vs $() . . . 3
1.2 $(document).ready() . . . 3
1.3 Avoiding Conflicts with Other Libraries . . . 4
1.3.1 Putting jQuery Into No-Conflict Mode . . . 4
1.3.2 Including jQuery Before Other Libraries . . . 5
1.4 Attributes . . . 6
1.5 Selecting Elements . . . 6
1.5.1 Selecting Elements by ID . . . 6
1.5.2 Selecting Elements by Class Name . . . 6
1.5.3 Selecting Elements by Attribute . . . 6
1.5.4 Selecting Elements by Compound CSS Selector . . . 6
1.5.5 Does My Selection Contain Any Elements? . . . 6
1.5.6 Saving Selections . . . 7
1.5.7 Refining and Filtering Selections . . . 7
1.5.8 Selecting Form Elements . . . 7
1.6 Working with Selections . . . 8
1.6.1 Chaining . . . 8
1.7 Manipulating Elements . . . 9
1.7.1 Getting and Setting Information About Elements . . . 9
1.7.2 Moving, Copying, and Removing Elements . . . 9
1.7.3 Cloning Elements . . . 10
1.7.4 Removing Elements . . . 10
1.7.5 Creating New Elements . . . 11
1.7.6 Manipulating Attributes . . . 12

Chapter 1

Using jquery Core

1.1 $ vs $()

Most jQuery methods are called on jQuery objects; for example

```
$( "h1" ).remove();
```

these methods are said to be part of the $.fn namespace, or the "jQuery prototype" and are best thought of as **jQuery object methods**.

However, there are several methods that do not act on a selection; these methods are said to be part of the jQuery namespace, and are best thought of as **core jQuery methods**.

This distinction can be incredibly confusing to new jQuery users. Here's what you need to remember:

- Methods called on jQuery selections are in the *$.fn* namespace, and automatically receive and return the selection as *this*.
- Methods in the $ namespace are generally utility-type methods, and do not work with selections; they are not automatically passed any arguments, and their return value will vary.

1.2 $(document).ready()

A page can't be manipulated safely until the document is "ready."jQuery detects this state of readiness for you.
Code included inside *$(document).ready()* will only run **once the page Document Object Model (DOM) is ready for JavaScript code to execute**.
Code included inside *$(window).load(function() ...)* will run **once the entire page (images or iframes), not just the DOM, is ready**.

There is a shorthand for *$(document).ready()*

```
$(function() {
console.log( "ready!" );
});
```

You can also **pass a named function** to *$(document).ready()* instead of passing an anonymous function.

```
function readyFn( jQuery ) {
// Code to run when the document is ready.
}
$( document ).ready( readyFn );
// or:
$( window ).load( readyFn );
```

1.3 Avoiding Conflicts with Other Libraries

By default, jQuery uses $ as a shortcut for jQuery. Thus, if you are using another JavaScript library that uses the $ variable, you can run into conflicts with jQuery.

1.3.1 Putting jQuery Into No-Conflict Mode

In order to avoid these conflicts, you need to **put jQuery in no-conflict mode** immediately after it is loaded onto the page and before you attempt to use jQuery in your page.

```
<!-- Putting jQuery into no-conflict mode. -->
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
var $j = jQuery.noConflict();
// $j is now an alias to the jQuery function; creating the new alias
// is optional.
$j(document).ready(function() {
$j( "div" ).hide();
});
// The $ variable now has the prototype meaning, which is a shortcut for
// document.getElementById(). mainDiv below is a DOM element, not a jQuery
//object.
window.onload = function() {
var mainDiv = $( "main" );
}
</script>
```

In the code above, the $ will revert back to its meaning in original library. You'll still be able to use the full function name *jQuery* as well as the new alias *$j* in the rest of your application. The new alias can be named anything you'd like: jq, awesomeQuery, etc.
Finally, **if you don't want to define another alternative** to the full jQuery function name (you really like to use $ and don't care about using the other library's $ method), then there's still **another approach you might try**: simply **add the $ as an argument passed to your jQuery(document).ready() function**.
This is most frequently used in the case where you still want the benefits of really concise jQuery code, but don't want to cause conflicts with other libraries.

```
!-- Another way to put jQuery into no-conflict mode. -->
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
jQuery.noConflict();
jQuery( document ).ready(function( $ ) {
// You can use the locally-scoped $ in here as an alias to jQuery.
$( "div" ).hide();
});
// The $ variable in the global scope has the prototype.js meaning.
window.onload = function(){
var mainDiv = $( "main" );
}
</script>
```

1.3.2 Including jQuery Before Other Libraries

The code snippets above rely on jQuery being loaded after *prototype.js* is loaded. If you include jQuery **before** other libraries, you may use jQuery when you do some work with jQuery, but the $ will have the meaning defined in the other library. There is no need to relinquish the $ alias by calling *jQuery.noConflict()*.

```
<!-- Loading jQuery before other libraries. -->
<script src="jquery.js"></script>
<script src="prototype.js"></script>
<script>
// Use full jQuery function name to reference jQuery.
jQuery( document ).ready(function() {
jQuery( "div" ).hide();
});
// Use the $ variable as defined in prototype.js
window.onload = function() {
var mainDiv = $( "main" );
};
</script>
```

1.4 Attributes

The *.attr()* method acts **as both a getter and a setter**. As a setter, *.attr()* can accept either a key and a value, or an object containing one or more key/value pairs.

.attr() as a setter:

```
$( "a" ).attr( "href", "allMyHrefsAreTheSameNow.html" );
$( "a" ).attr({
title: "all titles are the same too!",
href: "somethingNew.html"
}); \\In this case I've used an object
```

.attr() as a getter:

```
$( "a" ).attr( "href" );// Returns the href for the first a element
\\in the document
```

1.5 Selecting Elements

1.5.1 Selecting Elements by ID

```
$( "#myId" ); // Note IDs must be unique per page.
```

1.5.2 Selecting Elements by Class Name

```
$( ".myClass" );
```

1.5.3 Selecting Elements by Attribute

```
$( "input[name='first_name']" );
// Beware, this can be very slow in older browsers
```

1.5.4 Selecting Elements by Compound CSS Selector

```
$( "#contents ul.people li" );
```

1.5.5 Does My Selection Contain Any Elements?

Once you've made a selection, you'll often want to know whether you have anything to work with. The best way to determine **if there are any elements is to test the selection's *.length* property**, which tells you how many elements were selected. If the answer is 0, the *.length* property will evaluate to false when used as a boolean value:

```
// Testing whether a selection contains elements.
if ( $( "div.foo" ).length ) {
...
}
```

1.5.6 Saving Selections

jQuery doesn't cache elements for you. If you've made a selection that you might need to make again, you should **save the selection in a variable** rather than making the selection repeatedly.

```
var divs = $( "div" );
```

Once the selection is stored in a variable, you can **call jQuery methods on the variable** just like you would have called them on the original selection.

A selection **only fetches the elements that are on the page at the time the selection is made**. If elements are added to the page later, you'll have to repeat the selection or otherwise add them to the selection stored in the variable. Stored selections don't magically update when the DOM changes.

1.5.7 Refining and Filtering Selections

Sometimes the selection contains more than what you're after. jQuery offers **several methods for refining and filtering selections**.

```
// Refining selections.
$( "div.foo" ).has( "p" ); // div.foo elements that contain <p> tags
$( "h1" ).not( ".bar" ); // h1 elements that don't have a class of bar
$( "ul li" ).filter( ".current" ); // unordered list items with class of current
$( "ul li" ).first(); // just the first unordered list item
$( "ul li" ).eq( 5 ); // the sixth
```

1.5.8 Selecting Form Elements

jQuery offers several pseudo-selectors that help **find elements in forms**. These are especially helpful because it can be difficult to distinguish between form elements based on their state or type using standard CSS selectors.

:button Using the *:button* pseudo-selector targets any *<button>* elements and elements with a *type="button"*:

```
$( "form :button" );
```

In order to get the best performance using *:button*, it's best to first select elements with a standard jQuery selector, **then use *.filter(":button")***.
The same concept applies to all selections on form elements.

:checkbox Using the *:checkbox* pseudo-selector targets any *<input>* elements with a *type="checkbox"*:

```
$( "form :checkbox" );
```

:checked Not to be confused with *:checkbox*, *:checked* targets **checked checkboxes**, but keep in mind that this selector works also for **checked radio buttons, and select elements** (for select elements only, use the *:selected* selector):

```
$( "form :checked" );
```

The *:checked* pseudo-selector works when used with checkboxes, radio buttons and selects.

Many More!! Look at the jQuery documentation for all the pseudo-selections available for forms. There are many of them!

1.6 Working with Selections

jQuery "overloads" its methods, so **the method used to set a value generally has the same name as the method used to get a value**. When a method is used to set a value, it's called a **setter**. When a method is used to get (or read) a value, it's called a **getter**. Setters affect all elements in a selection. Getters get the requested value only for the first element in the selection.

```
// The .html() method used as a setter:
$( "h1" ).html( "hello world" );
// The .html() method used as a getter:
$( "h1" ).html();
```

Setters return a jQuery object, allowing you to **continue calling jQuery methods** on your selection. Getters return whatever they were asked to get, so you **can't continue to call jQuery methods** on the value returned by the getter.

```
// Attempting to call a jQuery method after calling a getter.
// This will NOT work:
$( "h1" ).html().addClass( "test" );
```

1.6.1 Chaining

If you call a method on a selection and that method returns a jQuery object, you can continue to call jQuery methods on the object without pausing for a semicolon. **This practice is referred to as "chaining"**:

```
$( "#content" ).find( "h3" ).eq( 2 ).html( "new text for the third h3!" );
```

Chaining is extraordinarily powerful, and it is a feature that many libraries have adapted since it was made popular by jQuery. However, **it must be used with care**, extensive chaining can make code extremely difficult to modify or debug. There is no hard-and-fast rule to how long a chain should be just know that it's easy to get carried away.

1.7 Manipulating Elements

1.7.1 Getting and Setting Information About Elements

There are many ways to change an existing element. Among the most common tasks is **changing the inner HTML or attribute of an element**. jQuery offers simple, cross-browser methods for these sorts of manipulations. You can also get information about elements using many of the same methods in their getter incarnations. Here are a few methods you can use to get and set information about elements:

- .html() - Get or set the HTML contents.
- .text() - Get or set the text contents; HTML will be stripped.
- .attr() - Get or set the value of the provided attribute.
- .width() - Get or set the width in pixels of the first element in the selection as an integer.
- .height() - Get or set the height in pixels of the first element in the selection as an integer.
- .position() - Get an object with position information for the first element in the selection, relative to its first positioned ancestor. This is a getter only.
- .val() - Get or set the value of form elements.

Changing things about elements is trivial, but remember that **the change will affect all elements in the selection**. If you just want to change one element, be sure to **specify that in the selection before calling a setter method**.

```
// Changing the HTML of an element.
$( "#myDiv p:first" ).html( "New <strong>first</strong> paragraph!" );
```

1.7.2 Moving, Copying, and Removing Elements

While there are a variety of ways to move elements around the DOM, there are generally two approaches:

- Place the selected element(s) relative to another element.
- Place an element relative to the selected element(s).

For example, jQuery provides *.insertAfter()* and *.after()*. The *.insertAfter()* method places the selected element(s) after the element provided as an argument. The *.after()* method places the element provided as an argument after the selected element.

Several other methods follow this pattern: *.insertBefore()* and *.before()*, *.appendTo()* and *.append()*, and *.prependTo()* and *.prepend()*.

The method that makes the most sense will depend on what elements are selected, and whether you need to store a reference to the elements you're adding to the page. **If you need to store a reference, you will always want to take the first approach** - placing the selected elements relative to another element - as it returns the element(s) you're placing. In this case, .insertAfter(), .insertBefore(), .appendTo(), and .prependTo() should be the tools of choice.

```
// Moving elements using different approaches.
// Make the first list item the last list item:
var li = $( "#myList li:first" ).appendTo( "#myList" );
// Another approach to the same problem:
$( "#myList" ).append( $( "#myList li:first" ) );
// Note that there's no way to access the list item
// that we moved, as this returns the list itself.
```

1.7.3 Cloning Elements

Methods such as *.appendTo()* move the element, but sometimes **a copy of the element is needed instead**. In this case, use *.clone()* first:

```
// Making a copy of an element.
// Copy the first list item to the end of the list:
$( "#myList li:first" ).clone().appendTo( "#myList" );
```

If you need to copy related data and events, be sure to pass true as an argument to .clone().

1.7.4 Removing Elements

There are **two ways to remove elements from the page**: *.remove()* and *.detach()*. Use *.remove()* when you want to permanently remove the selection from the page. While *.remove()* does return the removed element(s), those elements will not have their associated data and events attached to them if you return them to the page.
Use *.detach()* if you need **the data and events to persist**. Like *.remove()*, it returns the selection, but it also maintains the data and events associated with the selection, so you can restore the selection to the page at a later time.
The *.detach()* method is **extremely valuable** if you are doing heavy manipulation on an element. In that case, it's beneficial to *.detach()* the element from the page, work on it in your code, then restore it to the page when you're done. This limits expensive "DOM touches" while maintaining the element's data and events. If you want to leave the element on the page but remove its contents, you can use *.empty()* to dispose of the element's inner HTML.

1.7.5 Creating New Elements

jQuery offers a trivial and elegant way to create new elements using the same $() method used to make selections:

```
// Creating new elements from an HTML string.
$( "<p>This is a new paragraph</p>" );
$( "<li class=\"new\">new list item</li>" );

// Creating a new element with an attribute object.
$( "<a/>", {
html: "This is a <strong>new</strong> link",
"class": "new",
href: "foo.html"
});
```

Note that the attributes object in the second argument above, **the property name class is quoted**, although the property names html and href are not. Property names generally do not need to be quoted unless they are reserved words (as class is in this case).
When you create a new element, **it is not immediately added to the page**. There are several ways to add an element to the page once it's been created.

```
// Getting a new element on to the page.
var myNewElement = $( "<p>New element</p>" );
myNewElement.appendTo( "#content" );
// This will remove the p from #content!
myNewElement.insertAfter( "ul:last" );
// Clone the p so now we have two.
$( "ul" ).last().after( myNewElement.clone() );
```

The created element **doesn't need to be stored in a variable**; you can call the method to add the element to the page directly after the $(). However, most of the time you'll want a reference to the element you added so you won't have to select it later.
You can also create an element as you're adding it to the page, but note that in this case you don't get a reference to the newly created element:

```
// Creating and adding an element to the page at the same time.
$( "ul" ).append( "<li>list item</li>" );
```

The syntax for adding new elements to the page is easy, so it's tempting to forget that there's **a huge performance cost for adding to the DOM repeatedly**. If you're adding many elements to the same container, you'll want to concatenate all the HTML into a single string, and then append that string to the container **instead of appending the elements one at a time**. Use an array to gather all the pieces together, then join them into a single string for appending:

```
var myItems = [];
```

```
var myList = $( "#myList" );
for ( var i = 0; i < 100; i++ ) {
myItems.push( "<li>item " + i + "</li>" );
}
myList.append( myItems.join( "" ) );
```

1.7.6 Manipulating Attributes

jQuery's attribute manipulation capabilities are extensive. Basic changes are simple, but the *.attr()* method also allows for more complex manipulations. It can either **set an explicit value**, or **set a value using the return value of a function**. When the function syntax is used, the function receives two arguments: the zero-based index of the element whose attribute is being changed, and the current value of the attribute being changed.

```
// Manipulating a single attribute.
$( "#myDiv a:first" ).attr( "href", "newDestination.html" );

// Manipulating multiple attributes.
$( "#myDiv a:first" ).attr({
href: "newDestination.html",
rel: "nofollow"
});

// Using a function to determine an attribute's new value.
$( "#myDiv a:first" ).attr({
rel: "nofollow",
href: function( idx, href ) {
return "/new/" + href;
}
});
$( "#myDiv a:first" ).attr( "href", function( idx, href ) {
return "/new/" + href;
});
```