Contents

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