jQuery Basics

By NKS

$(document).ready()

You cannot safely manipulate a page until the document is “ready.” jQuery detects this state of readiness for you; code included inside `$(document).ready()` will only run once the page is ready for JavaScript code to execute.

Example1: A $(document).ready() block

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

There is a shorthand for `$(document).ready()` that you will sometimes see; however, I recommend against using it if you are writing code that people who aren't experienced with jQuery may see.

Example.2: Shorthand for $(document).ready()

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

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

Example.3: Passing a named function instead of an anonymous function

```
function readyFn() {
    // code to run when the document is ready
}


$(document).ready(readyFn);
```

Selecting Elements

The most basic concept of jQuery is to “select some elements and do something with them.” jQuery supports most CSS3 selectors, as well as some non-standard selectors. For a complete selector reference, visit http://api.jquery.com/category/selectors/.

Following are a few examples of common selection techniques.

Example.4: Selecting elements by ID

```
$('#myId'); // note IDs must be unique per page
```

Example.5: Selecting elements by class name

```
$('div.myClass'); // performance improves if you specify element type
```

Example.6: Selecting elements by attribute

```
$('input[name=first_name]'); // beware, this can be very slow
```

Example.7: Selecting elements by compound CSS selector

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

Example.8: Pseudo-selectors

```
$('a.external:first');
$('tr:odd');
$('#myForm :input');   // select all input-like elements in a form
$('div:visible');
$('div:gt(2)');        // all except the first three divs
$('div:animated');     // all currently animated divs
```

Note

When you use the `:visible` and `:hidden` pseudo-selectors, jQuery tests the actual visibility of the element, not its CSS visibility or display — that is, it looks to see if the element's *physical height and width on the page* are both greater than zero. However, this test doesn't work with `<tr>` elements; in this case, jQuery does check the CSS `display` property, and considers an element hidden if its `display` property is set to `none`. Elements that have not been added to the DOM will always be considered hidden, even if the CSS that would affect them would render them visible. (See the Manipulation section later in this chapter to learn how to create and add elements to the DOM.)

For reference, here is the code jQuery uses to determine whether an element is visible or hidden, with comments added for clarity:

```
jQuery.expr.filters.hidden = function( elem ) {
    var width = elem.offsetWidth, height = elem.offsetHeight,
        skip = elem.nodeName.toLowerCase() === "tr";

    // does the element have 0 height, 0 width,
    // and it's not a <tr>?
    return width === 0 && height === 0 && !skip ?

        // then it must be hidden
        true :

        // but if it has width and height
        // and it's not a <tr>
        width > 0 && height > 0 && !skip ?

            // then it must be visible
            false :

            // if we get here, the element has width
            // and height, but it's also a <tr>,
            // so check its display property to
            // decide whether it's hidden
            jQuery.curCSS(elem, "display") === "none";
};

jQuery.expr.filters.visible = function( elem ) {
    return !jQuery.expr.filters.hidden( elem );
};
```

Choosing Selectors

Choosing good selectors is one way to improve the performance of your JavaScript. A little specificity — for example, including an element type such as `div` when selecting elements by class name — can go a long way. Generally, any time you can give jQuery a hint about where it might expect to find what you're looking for, you should. On the other hand, too much specificity can be a bad thing. A selector such as `#myTable thead tr th.special` is overkill if a selector such as `#myTable th.special` will get you what you want.

jQuery offers many attribute-based selectors, allowing you to make selections based on the content of arbitrary attributes using simplified regular expressions.

```
// find all <a>s whose rel attribute
// ends with "thinger"
$("a[rel$='thinger']");
```

While these can be useful in a pinch, they can also be extremely slow — I once wrote an attribute-based selector that locked up my page for multiple seconds. Wherever possible, make your selections using IDs, class names, and tag names.

Want to know more? Paul Irish has a great presentation about improving performance in JavaScript, with several slides focused specifically on selector performance.

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. You may be inclined to try something like:

```
if ($('div.foo')) { ... }
```

This won't work. When you make a selection using `$()`, an object is always returned, and objects always evaluate to `true`. Even if your selection doesn't contain any elements, the code inside the `if` statement will still run.

Instead, you need 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.

Example.9: Testing whether a selection contains elements

```
if ($('div.foo').length) { ... }
```

Saving Selections

Every time you make a selection, a lot of code runs, and jQuery doesn't do caching of selections 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.

Example.10: Storing selections in a variable

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

Note

In "Storing selections in a variable", the variable name begins with a dollar sign. Unlike in other languages, there's nothing special about the dollar sign in JavaScript — it's just another character. We use it here to indicate that the variable contains a jQuery object. This practice — a sort of Hungarian notation — is merely convention, and is not mandatory.

Once you've stored your selection, you can call jQuery methods on the variable you stored it in just like you would have called them on the original selection.

Note

A selection only fetches the elements that are on the page when you make the selection. If you add elements 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.

Refining & Filtering Selections

Sometimes you have a selection that contains more than what you're after; in this case, you may want to refine your selection. jQuery offers several methods for zeroing in on exactly what you're after.

Example.11: Refining selections

```
$('div.foo').has('p');          // div.foo elements that contain <p>'s
$('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
```

Selecting Form Elements

jQuery offers several pseudo-selectors that help you find elements in your 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

Selects `<button>` elements and elements with `type="button"`

:checkbox

Selects inputs with `type="checkbox"`

:checked

Selects checked inputs

:disabled

Selects disabled form elements

:enabled

Selects enabled form elements

:file

Selects inputs with `type="file"`

:image

Selects inputs with `type="image"`

:input

Selects `<input>`, `<textarea>`, and `<select>` elements

:password

Selects inputs with `type="password"`

:radio

Selects inputs with `type="radio"`

:reset

Selects inputs with `type="reset"`

:selected

Selects options that are selected

:submit

Selects inputs with `type="submit"`

:text

Selects inputs with `type="text"`

Example.12: Using form-related pseduo-selectors

```
$('#myForm :input'); // get all elements
that accept input
```

Working with Selections

Once you have a selection, you can call methods on the selection. Methods generally come in two different flavors: getters and setters. Getters return a property of the first selected element; setters set a property on all selected elements.

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.

Example.13: Chaining

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

If you are writing a chain that includes several steps, you (and the person who comes after you) may find your code more readable if you break the chain over several lines.

Example.14: Formatting chained code

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

If you change your selection in the midst of a chain, jQuery provides the `$.fn.end` method to get you back to your original selection.

Example.15: Restoring your original selection using $.fn.end

```
$('#content')
    .find('h3')
    .eq(2)
        .html('new text for the third h3!')
    .end() // restores the selection to all h3's in #content
    .eq(0)
        .html('new text for the first h3!');
```

Note

Chaining is extraordinarily powerful, and it's 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 is easy to get carried away.

Getters & Setters

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 is called a setter. When a method is used to get (or read) a value, it is called a getter. Setters affect all elements in a selection; getters get the requested value only for the first element in the selection.

Example.16: The $.fn.html method used as a setter

```
$('h1').html('hello world');
```

Example.17: The html method used as a getter

```
$('h1').html();
```

Setters return a jQuery object, allowing you to continue to call jQuery methods on your selection; getters return whatever they were asked to get, meaning you cannot continue to call jQuery methods on the value returned by the getter.

CSS, Styling, & Dimensions

jQuery includes a handy way to get and set CSS properties of elements.

Note

CSS properties that normally include a hyphen need to be *camel cased* in JavaScript. For example, the CSS property `font-size` is expressed as `fontSize` when used as a property name in JavaScript. This does not apply, however, when passing the name of a CSS property to the `$.fn.css` method as a string — in that case, either the camel cased or hyphenated form will work.

Example.18: Getting CSS properties

```
$('h1').css('fontSize'); // returns a string such as "19px"
$('h1').css('font-size'); // also works
```

Example.19: Setting CSS properties

```
$('h1').css('fontSize', '100px'); // setting an individual property
$('h1').css({ 'fontSize' : '100px', 'color' : 'red' }); // setting multiple properties
```

Note the style of the argument we use on the second line — it is an object that contains multiple properties. This is a common way to pass multiple arguments to a function, and many jQuery setter methods accept objects to set mulitple values at once.

Using CSS Classes for Styling

As a getter, the `$.fn.css` method is valuable; however, it should generally be avoided as a setter in production-ready code, because you don't want presentational information in your JavaScript. Instead, write CSS rules for classes that describe the various visual states, and then simply change the class on the element you want to affect.

Example.20: Working with classes

```
var $h1 = $('h1');

$h1.addClass('big');
$h1.removeClass('big');
$h1.toggleClass('big');

if ($h1.hasClass('big')) { ... }
```

Classes can also be useful for storing state information about an element, such as indicating that an element is selected.

Dimensions

jQuery offers a variety of methods for obtaining and modifying dimension and position information about an element.

The code in “Basic dimensions methods”, is just a very brief overview of the dimensions functionality in jQuery; for complete details about jQuery dimension methods, visit http://api.jquery.com/category/dimensions/.

Example.21: Basic dimensions methods

```
$('h1').width('50px');   // sets the width of all H1 elements
$('h1').width();         // gets the width of the first H1

$('h1').height('50px');  // sets the height of all H1 elements
$('h1').height();        // gets the height of the first H1

$('h1').position();      // returns an object containing position
                         // information for the first H1 relative to
                         // its "offset (positioned) parent"
```

Attributes

An element's attributes can contain useful information for your application, so it's important to be able to get and set them.

The `$.fn.attr` method acts as both a getter and a setter. As with the `$.fn.css` method, `$.fn.attr` as a setter can accept either a key and a value, or an object containing one or more key/value pairs.

Example.22: Setting attributes

```
$('a').attr('href', 'allMyHrefsAreTheSameNow.html');
$('a').attr({
    'title' : 'all titles are the same too!',
    'href' : 'somethingNew.html'
});
```

This time, we broke the object up into multiple lines. Remember, whitespace doesn't matter in JavaScript, so you should feel free to use it liberally to make your code more legible! You can use a minification tool later to strip out unnecessary whitespace for production.

Example.23: Getting attributes

```
$('a').attr('href');  // returns the href for the first a element in the document
```

Traversing

Once you have a jQuery selection, you can find other elements using your selection as a starting point.

For complete documentation of jQuery traversal methods, visit http://api.jquery.com/category/traversing/.

Note

Be cautious with traversing long distances in your documents — complex traversal makes it imperative that your document's structure remain the same, something that's difficult to guarantee even if you're the one creating the whole application from server to client. One- or two-step traversal is fine, but you generally want to avoid traversals that take you from one container to another.

Example.24: Moving around the DOM using traversal methods

```
$('h1').next('p');
```

```
$('div:visible').parent();
$('input[name=first_name]').closest('form');
$('#myList').children();
$('li.selected').siblings();
```

You can also iterate over a selection using `$.fn.each`. This method iterates over all of the elements in a selection, and runs a function for each one. The function receives the index of the current element and the DOM element itself as arguments. Inside the function, the DOM element is also available as `this` by default.

Example.25: Iterating over a selection

```
$('#myList li').each(function(idx, el) {
    console.log(
        'Element ' + idx +
        'has the following html: ' +
        $(el).html()
    );
});
```

Manipulating Elements

Once you've made a selection, the fun begins. You can change, move, remove, and clone elements. You can also create new elements via a simple syntax.

For complete documentation of jQuery manipulation methods, visit http://api.jquery.com/category/manipulation/.

Getting and Setting Information about Elements

There are any number of ways you can change an existing element. Among the most common tasks you'll perform 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. We'll see examples of these throughout this section, but specifically, here are a few methods you can use to get and set information about elements.

Note

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

Note

When methods act as getters, they generally only work on the first element in the selection, and they do not return a jQuery object, so you can't chain additional methods to them. One notable exception is `$.fn.text`; as mentioned below, it gets the text for all elements in the selection.

$.fn.html

Get or set the html contents.

$.fn.text

Get or set the text contents; HTML will be stripped.

$.fn.attr

Get or set the value of the provided attribute.

$.fn.width

Get or set the width in pixels of the first element in the selection as an integer.

$.fn.height

Get or set the height in pixels of the first element in the selection as an integer.

$.fn.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.*

$.fn.val

Get or set the value of form elements.

Example.26: Changing the HTML of an element

```
$('#myDiv p:first')
    .html('New <strong>first</strong> paragraph!');
```

Moving, Copying, and Removing Elements

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

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

For example, jQuery provides `$.fn.insertAfter` and `$.fn.after`. The `$.fn.insertAfter` method places the selected element(s) after the element that you provide as an argument; the `$.fn.after` method places the element provided as an argument after the selected element. Several other methods follow this pattern: `$.fn.insertBefore` and `$.fn.before`; `$.fn.appendTo` and `$.fn.append`; and `$.fn.prependTo` and `$.fn.prepend`.

The method that makes the most sense for you will depend on what elements you already have selected, and whether you will 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, `$.fn.insertAfter`, `$.fn.insertBefore`, `$.fn.appendTo`, and `$.fn.prependTo` will be your tools of choice.

Example.27: 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
```

Cloning Elements

When you use methods such as $.fn.appendTo, you are moving the element; sometimes you want to make a copy of the element instead. In this case, you'll need to use $.fn.clone first.

Example.28: Making a copy of an element

```
// copy the first list item to the end of the list
$('#myList li:first').clone().appendTo('#myList');
```

Note

If you need to copy related data and events, be sure to pass `true` as an argument to `$.fn.clone`.

Removing Elements

There are two ways to remove elements from the page: `$.fn.remove` and `$.fn.detach`. You'll use `$.fn.remove` when you want to permanently remove the selection from the page; while the method 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.

If you need the data and events to persist, you'll want to use `$.fn.detach` instead. Like `$.fn.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.

Note

The `$.fn.detach` method is extremely valuable if you are doing heavy manipulation to an element. In that case, it's beneficial to `$.fn.detach` the element from the page, work on it in your code, and then restore it to the page when you're done. This saves you from expensive "DOM touches" while maintaining the element's data and events.

If you want to leave the element on the page but simply want to remove its contents, you can use `$.fn.empty` to dispose of the element's inner HTML.

Creating New Elements

jQuery offers a trivial and elegant way to create new elements using the same `$()` method you use to make selections.

Example.29: Creating new elements

```
$('<p>This is a new paragraph</p>');
$('<li class="new">new list item</li>');
```

Example.30: Creating a new element with an attribute object

```
$('<a/>', {
    html : 'This is a <strong>new</strong> link',
    'class' : 'new',
    href : 'foo.html'
});
```

Note that in the attributes object we included as the second argument, the property name class is quoted, while the property names text 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.

Example.31: Getting a new element on to the page

```
var $myNewElement = $('<p>New element</p>');
$myNewElement.appendTo('#content');

$myNewElement.insertAfter('ul:last'); // this will remove the p from #content!
$('ul').last().after($myNewElement.clone()); // clone the p so now we have 2
```

Strictly speaking, you don't have to store the created element in a variable — you could just call the method to add the element to the page directly after the $(). However, most of the time you will want a reference to the element you added, so you don't need to select it later.

You can even 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.

Example.32: Creating and adding an element to the page at the same time

```
$('ul').append('<li>list item</li>');
```

Note

The syntax for adding new elements to the page is so easy, it's tempting to forget that there's a huge performance cost for adding to the DOM repeatedly. If you are 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. You can use an array to gather all the pieces together, then `join` them into a single string for appending.

```
var myItems = [], $myList =
$('#myList');

for (var i=0; i<100; i++) {
    myItems.push('<li>item ' + i +
'</li>');
}

$myList.append(myItems.join(''));
```

Manipulating Attributes

jQuery's attribute manipulation capabilities are extensive. Basic changes are simple, but the $.fn.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.

Example.33: Manipulating a single attribute

```
$('#myDiv a:first').attr('href',
'newDestination.html');
```

Example.34: Manipulating multiple attributes

```
$('#myDiv a:first').attr({
    href : 'newDestination.html',
    rel : 'super-special'
});
```

Example.35: Using a function to determine an attribute's new value

```
$('#myDiv a:first').attr({
    rel : 'super-special',
    href : function(idx, href) {
            return '/new/' + href;
        }
});

$('#myDiv a:first').attr('href',
function(idx, href) {
    return '/new/' + href;
});
```

Exercises

Selecting

Open the file `/exercises/index.html` in your browser. work to accomplish the following:

1. Select all of the div elements that have a class of "module".
2. Come up with three selectors that you could use to get the third item in the #myList unordered list. Which is the best to use? Why?
3. Select the label for the search input using an attribute selector.
4. Figure out how many elements on the page are hidden (hint: .length).
5. Figure out how many image elements on the page have an alt attribute.
6. Select all of the odd table rows in the table body.

Traversing

Open the file `/exercises/index.html` in your browser. work to accomplish the following:

1. Select all of the image elements on the page; log each image's alt attribute.
2. Select the search input text box, then traverse up to the form and add a class to the form.
3. Select the list item inside #myList that has a class of "current" and remove that class from it; add a class of "current" to the next list item.
4. Select the select element inside #specials; traverse your way to the submit button.

5. Select the first list item in the #slideshow element; add the class "current" to it, and then add a class of "disabled" to its sibling elements.

Manipulating

Open the file `/exercises/index.html` in your browser.

1. Add five new list items to the end of the unordered list #myList. Hint:

   ```
   for (var i = 0; i<5; i++) { ... }
   ```

2. Remove the odd list items

3. Add another h2 and another paragraph to the last div.module

4. Add another option to the select element; give the option the value "Wednesday"

5. Add a new div.module to the page after the last one; put a copy of one of the existing images inside of it.