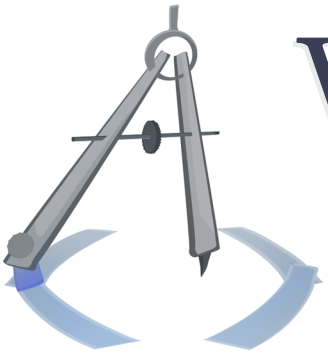


# Website Architecture

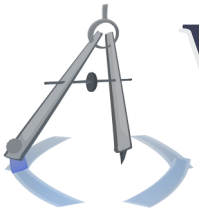
## Lezione 6b

# jQuery

An overview and getting-started guide for the ubiquitous JavaScript library.

**Michael Serritella**

Summer 2012





# Intro to jQuery

jQuery is a library which simplifies many common tasks in JavaScript, such as getting your hands on DOM nodes, registering event handlers, and changing CSS; it also runs the same (or as closely as possible) on virtually any browser, including IE6, which is kind of incredible. jQuery's feature list has crept to include almost all foundational aspects of JavaScript development, and it has become ubiquitous.



# First things first: $

In JavaScript, the dollar sign - $ - can be part of a symbol name, such as a variable name. Indeed, you can have a symbol name which is only a dollar sign. And indeed, jQuery has done this. jQuery is accessible by one main function (which acts as a class, as per JavaScript's prototype model), which is named **$**. So, somewhere in jQuery code, they define this:

JavaScript

```
function $() {
  // a ton of code
}
```

# Quick sales pitch

Why use jQuery? How easy is it to perform some simple and complex tasks? Let's see a code example.



JavaScript

```
// get all .active descendants of span which are descendants of div
var someNodes = $("div span .active");

// change their css "color" property values to red
someNodes.css('color', 'red');

// hide them using display: none (inline)
someNodes.hide();

// show them by removing the "display" property from inline
someNodes.show();

// give a handler for the "click" event
someNodes.click(function() { alert("You clicked me!"); });

// get all paragraph children of div.header which contain text "today"
// (CSS3 magic)
var otherNodes = $("div.header > p:contains('today')");
```

# Getting started

OK, so you want to use jQuery. Hard. How do you do it? It's a simple as including a JavaScript file. You can download the file **from jQuery** and serve it from your site like any .js file. Or you can link to their .js file via an absolute link from your site. The ".min.js" versions are "minified" JavaScript, in which whitespace and comments have been removed, etc.



HTML

```
<script type="text/javascript" src="myfiles/jquery-1.7.2.min.js">
</script>

<!-- or -->

<script type="text/javascript"
        src="http://code.jquery.com/jquery-1.7.2.js">
</script>

<!-- ..or -->

<script type="text/javascript"
        src="//code.jquery.com/jquery-1.7.2.js">
</script>
```

The last one is a "protocol-relative URI". If you are using HTTPS, it will also fetch the .js file using HTTPS. Neat. Note that if you are looking at the file from your hard drive, the browser is probably using the "file://" protocol, so this will break.

jQuery's **documentation** is fairly thorough, so you should take a look around.



# Querying the DOM

Now we have jQuery. How do we get some nodes?

The `$()` function actually can accept several types of arguments. If you pass a string, jQuery tries to interpret it as a CSS selector, and it searches the entire document for the matching nodes.

JavaScript

```
var $nodes = $("p.IntroParagraph");

var $moreNodes = $("html, body");
```

jQuery actually returns an object which holds a collection of nodes, which we'll see. By convention, variable names which hold those objects usually begin with `$` in application code.

jQuery even implements most/all of CSS3, which is more than can be said for the browsers in which jQuery executes. So you can use some advanced selectors. For a nice list, view jQuery's **documentation**.



If you have a raw DOM node, you can wrap it up nicely in a jQuery object:

JavaScript

```
// a regular DOM node
var footer = document.getElementById("theFooterID");

// a jQuery object containing the node
var $footer  = $(footer);
```

Lastly, you can make some nodes up on the spot. These would not yet be attached to the DOM and thus would not appear on the page until you attach them.

JavaScript

```
var aDiv = $("<div></div>");
```



# Collections & jQuery objects

Each jQuery object, such as the one you get back from `$()`, is a collection of zero or more DOM nodes. You can use these objects in a few different ways, such as to execute changes to multiple nodes, retrieve information about one node, prune the set of nodes, and more. We'll see some typical usage patterns.

## Calling methods of the collection

jQuery objects expose very many methods. You can call them like this:



JavaScript

```
//get a collection going
var $nodes  = $([aNode, anotherNode]);

// call one method
$nodes.css('font-size', '120%');

// call several independent methods using a compact syntax
// ("fluent interface", found in common style of other languages,
// such as Java)
$nodes.css('font-size', '120%').css('position', 'relative').hide();

// often formatted like this
$nodes.css('font-size', '120%')
      .css('position', 'relative')
      .hide();
```

This fluent interface works because each call to these methods returns the whole collection again. Some methods won't do this, of course, such as **`width()`**, which just returns a number. But you can usually do this and should probably format it nicely.



# Properties & metrics

This section covers some of the most basic jQuery-object methods, such as those that return scalar information. The following is a set of some of the most common methods; it is not comprehensive.

### `size()`

Get the size of the collection. Good way to tell if there are any such nodes in the document:

```JavaScript
if ($(".PromotionalFooter").size() == 0) { ... }
```

### `is()`

Determine whether **at least one element of** the collection satisfies the given selector:

```JavaScript
if ($myCollection.is(".active")) { ... }
```



Since the semantics are blurry for multiple nodes, you should probably only do this with a collection of zero or one nodes.

### `width()` and friends

Get the dimensions of a node, perhaps including margin and whatnot. These functions apply to **the first element in the collection**.

JavaScript

```
// width excluding padding, border, margin
var theWidth_IntegerInPixels = $myNode.width();

// width including padding only
var theWidthWithPadding = $myNode.innerWidth();

// width including border but not margin
var theWidthWithPadding = $myNode.outerWidth();

// width including margin
var theWidthWithPadding = $myNode.outerWidth(true);
```

There is also **`height()`** and it also has friends.

## `offset()`

Get the (x,y) coordinate of an element on the page. See the **category** of related functions. Again, these apply to **the first element** of a collection.

JavaScript

```
// returns a quick little object
var theOffset = $myNode.offset();

var theX = theOffset.left;
var theY = theOffset.top;
```

# CSS & Attribute manipulation

You can easily retrieve or change the CSS or element attributes of an entire collection. A quick example says everything.

JavaScript

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

var oldColor = $myCollection.css('color');
$myCollection.css('color', 'red');
```

There is an **`attr()`** function which works similarly for changing (markup) attributes, though the accessor operates upon **the first element in the collection** and the mutator operates upon **all elements in the collection**.

The function **`prop()`** is just like **`attr()`**, except it's for DOM-node properties.

The functions **`hasClass()`**, **`addClass()`**, **`removeClass()`**, and **`toggleClass()`** exist and are cool.



# Event handlers

jQuery provides an easy and cross-browser way to register event handlers. There are a few ways to do it. Fortunately, at least these methods operate upon the entire collection of nodes.

First, let's see an example:

JavaScript

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

// add a click event for each element in the collection
$myCollection.click(function() { alert("click!"); });

// in this handler, work with the divs in the collection
$myCollection.hover(function(event) {
  // the "this" keyword refers to each div's DOM node inside here;
  // you may want to make a jQuery object from it right away.
  var $thisDiv = $(this);

  // you can also be passed the event object which has information
  // about the event itself (e.g. keyboard key pressed)
  if (event.something) { }

  // do something special per each div
  if ($thisDiv.is(".blue"))
  {
    $thisDiv.css('color', '#00007D');
  }
 });
```



You may recall that there are names for events, such as "click", and then the HTML attributes that correspond to them, such as `onclick`. Keep in mind that the actual name is "click".

There are many types of events, but basically all of them follow this syntax. Read the **documentation** for the complete documentation.

Sideline note: If you want to cancel an event, so to speak, you can return false from the event handler or do `event.stopPropagation()` or `event.preventDefault()`, whichever is semantically correct. See **this** angry blog post for more details.

If you want to invoke an event, you can sometimes call the same jQuery method (the `focus()` method, for example), with no parameters.

There are a couple notable exceptions to this pattern. You can call the function `on()`, which binds event handlers to nodes which may not exist yet. And you can do this:

JavaScript

```
$(document).ready(function() {
  // execute this when **the DOM is ready**;
  // do NOT wait until the page and all its resources have loaded!
  cool();
 });
```

Yes, you can do some jQuery stunts off of **`document`** and **`window`** and friends. This specific document/ready event is so important that there is an alternate syntax:

JavaScript

```
$(function() {
  // you just passed a function to the jQuery $() function
  cool();
 });
```



# DOM and collection manipulation

Perhaps the most powerful operations are the ones which manipulate the DOM or the collection of jQuery elements. First, let's look at collection manipulation. This uses a "functional" programming style, which is alien to most people; it has just a few quirks.

## Collection manipulation

JavaScript

```
$myCollection = $("div");

$myCollection.size(); // is 3

$myCollection.add($("span"));
$myCollection.size(); // is 3

$myCollection.add($("span"));
$myCollection.size(); // is 3

// what?
$myCollection = $myCollection.add($("span"));

// oh.
$myCollection.size(); // is 9
```



You can manipulate the collection, but the manipulation generally only occurs in **a copy of the collection**, which is returned. Thus you can do stunts like this:

JavaScript

```
$myCollection = $("div");

$myCollection.filter(".active").css('text-decoration', 'underline');

// collection remains unchanged
```

So that is the style; those are the semantics. What operations can you perform?

## Filtering by CSS selectors

You can prune the collection to only those elements which match a CSS selector. Use the function **`filter()`** as shown above.

## Finding descendants by CSS selectors

Simiarly, but differently, you can look to see which **descendant nodes of the nodes in the collection** match a



CSS selector. These found nodes may not be in the collection; maybe only their parents are in your collection. Use the function **`find()`** like this:

JavaScript

```
$myCollection = $("div");

$myCollection.find("span").hide();
```

## Iteration over each element

This is probably the first weird thing. How do you do a **`for`** loop over the nodes in a collection? Well, you can't do that, exactly. You do this:

JavaScript

```
$myCollection.each(function(index) {
  // like an event handler, "this" is the current DOM node
  var $this = $(this);

  // index is the zero-indexed index in the collection

  // if you want to stop iterating, return false.
  if (index > 5)
  {
     return false;
  }
 });
```



## DOM manipulation

Here is the big draw - maybe. You can add, remove, clone, and reorder DOM elements, changing the document on the fly. The **documentation** is clear, and there is a pattern which all the related methods follow, so we'll see the pattern by looking at some common functions.

### `append()`

Adds a node - or a whole collection of nodes - as the children of another node.

JavaScript

```
$theContainer = $("div#theContainer");
$theChildrenToBe = $("span");
$theContainer.append($theChildrenToBe);
```

There are only a few caveats. For one, the elements are *moved* to the new location, not copied. For another, they are inserted in the order in which they occur in the DOM.

### `appendTo()`

Same as **`append()`**, but the roles are reversed:



JavaScript

```
$theContainer = $("div#theContainer");
$theChildrenToBe = $("span");
$theChildrenToBe.appendTo($theContainer);
```

## And friends..

See also: **`detach()`**, **`remove()`**, **`insertAfter()`**, **`replaceWith()`**. So it is fairly simple to move these nodes around. In addition to moving nodes, what might you want to do?

## Capturing & replacing content

The methods text() and html() are accessors and mutators. They pretty much do what they sound like.

Using text(), you can capture the plaintext inside the collection. This is extremely useful for text parsing and searching. If you pass an argument, such as text("some text"), you replace the content of the collection with plain text - you clear out the nodes.

Using html(), you can capture the HTML or set new HTML. If you pass an argument like html("this & that"), the ampersand wil be escaped for HTML syntax, which is quite convenient.

## Creating nodes

As we mentioned earlier, you can create nodes out of thin air. Of course, we could do this before jQuery, but this is a little more natural:

JavaScript
```
var $collection = $("<div></div>");
```

You may want to fill in a larger segment of HTML, such as " ... ", or you may want to build it more programmatically with nodes, such as this:

JavaScript
```
var $collection=  $("<div></div>").append("<span></span>").text(theInput);
```

Yes, the combination of this node creation and the text() method can make for nice and safe handling of user input.

## Copying nodes

Using the clone() method, you can create a copy of the collection. This has some obvious benefits and a hidden performance benefit. First, an example:

JavaScript
```
$myCollection.clone().appendTo("#container");
```

Now for the performance benefit. Well, basically ,parsing is hard. At least we can say that it has a substantial cost. If you repetitively set the same HTML for new content, such as

JavaScript
```
$theContainer.append("<div><p>asdf<a href...>asdfsf</a>sfa</p></div>");
```

you should probably make a node/jQuery version of this HTML string and then just clone that every time. It is significantly cheaper. Keep a hidden copy of your template HTML nodes and then just clone them, populate them, and show them.

You can optionally clone the event bindings and any associated data.

# Utility functions

jQuery provides some functions which are simply nice functions, some of them isolated from the concept of collections.

See the **documentation** for their category of utility functions.

### `$collection.data()`

You can add and retrieve abitrary data associated with a collection. You could maybe do that with a node, like by adding a property to the object, but this is less dirty. As you can imagine, you can do this:

JavaScript

```
// set it
$collection.data('fieldName', 567);

// get it back
var it = $collection.data('fieldName');
```



### $.trim()

Trimming is removing whitespace from the beginning and end of a string. There is no native JavaScript function for this. Call it like so:

JavaScript
```
var trimmed = $.trim("  my String ");
```

### $.extend()

This function may be used to clone an object, optionally with a deep copy, and optionally add new fields to the object. Here is its usage:

JavaScript
```
var aShallowCopy = $.extend({}, originalObject);

var aDeepCopy = $.extend(true, {}, originalObject);

var aShallowCopyWithExtra = $.extend({fieldOne: 2}, originalObject);

var aDeepCopyWithExtra = $.extend(true, {fieldTwo: 3}, originalObject);
```


# jQuery plugins

jQuery is extensible, and there is a large market of widgets out there which are built on top of jQuery. These are usually made for purposes of slick UI effects (nice photo galleries), animations, or typical presentational formats, such as data tables with the ability to sort by column values. The important thing is that they usually follow a convention for initialization and further manipulation of the widget.

First, see the most popular and officially-sanctioned extension: **jQuery UI**. We can use one of its widgets as an example.

Let's say we want to turn an element into a "draggable" so a user can drag it around with the mouse. The most simple use case is normally like this:

JavaScript

```
$myCollection.draggable();
```

If you want to give options to the constructor, you typically do it like this:



```javascript
$myCollection.draggable({
    axis: 'y',
    delay: 300,
    revert: true,
    stop: function() { alert("just finished dragging.."); }
 });
```

And finally, the widget, such as draggable, may expose some methods. Maybe you even want to retrieve some information, like where it was dragged (not directly available in this case). You can usually call a method like this:

```javascript
$myDraggable.draggable('methodName', 'maybeAParamater');
```

And that's about it! Most widgets have a bit of variation, and all have their own quirky naming schemes and semantics for their parameters. But this is the common set of knowledge you would be expected to have when you get started.