jQuery

Lecture 13
Chapter 11 in HTML Text
Chapter 19 in the JavaScript Text

Problems with JavaScript on client

- The core JavaScript API (what we learned with Node) is pretty nice
- The client-side API has some problems
    - Its verbose
    - Browser support is inconsistent (especially < IE9)
    - Its not particularly powerful (makes things tedious)

- To avoid many of these problems, most developers use a JavaScript library to abstract away some of the nastiness

JavaScript libraries

- JavaScript libraries are simply included via the <script> element on your HTML pages
- They typically provide enhanced support for
  - Retrieving (querying) elements in the DOM
  - Manipulating the DOM
    - Setting attributes, HTML content, inserting/removing nodes
  - More powerful event handling

Another important goal: Clean HTML

CSS and JavaScript frameworks help us rid our HTML of presentation and interactivity distractions… achieving better separation in our code

jQuery

There are many popular JavaScript libraries

    There is another being written right now…

jQuery is the most popular

- Initially released in 2006
- Written by John Resig
- Used by over 65% of the 10,000 most visited websites (Wikipedia)
- Open Source (MIT License)
- Bundled in many web frameworks (even Microsoft)
- Spin-offs include jQuery UI and jQuery Mobile

Versions and CDNs

```
<script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
```

- Currently there are two “current” versions
  - 1x supports IE6+ (and all other browsers)
  - 2x supports IE9+ (and all other browsers)
- Current version is 1.10.x and 2.0x
- 1.10x is recommended, as IE6, IE7, and IE8 are still too common to ignore
- You can download and link locally or use a **content delivery network.**

Content Delivery Networks

- There are a few advantages to CDN's
  - If you use a popular one, chances are your users won't actually need to download it
  - They tend to be stable, and can reduce your bandwidth usage

- Disadvantages
  - What if the CDN goes down?
  - What if your version is pulled?
  - Offline development work?
  - Loss of control, privacy?

The disadvantages aren't significant for the "run of the mill" web application - but shouldn't be ignored.

jQuery

- At the core of jQuery is a **global** function called "the jQuery function".
    - You can call it using the identifier **jQuery** or just **$**.
    - The primary purpose of the jQuery function is to **create jQuery objects**.
    - jQuery objects are collections of *selected* HTML elements, which can later be interrogated and manipulated

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

Selection

- The string you pass to the $ function forms a selection (or query).
- You may use any valid CSS selector to get elements

```
var a = $("#unique");
var b = $(".special");
var c = $("p.special span");
```

We'll see other forms of queries shortly as well

jQuery object

- Its critical to understand that the result of the $ function is not (directly) a set of HTML elements
- The $ function returns a jQuery object
- The object contains:
  - A list of all the HTML elements selected
  - Properties, such as length, selector used
  - Many methods used to manipulate the selected elements
- Note - the object may contain no elements at all, a single HTML element, or many elements

Getters and Setters - combined

- The **`css`** is a <u>getter/setter</u> function (there are others). It is used to either retrieve properties or set properties.
- If you call **`css`** with a single string, it returns the value of the css property you specify of the **first element** in the $ object
- Call **`css`** with two strings sets the value of specified css property on **ALL ELEMENTS** in the $ object

This is critical - jQuery functions that manipulate elements do so on ALL selected objects. This is incredibly powerful.

CSS properties

```
var divs = $(“div”);
divs.css(“color”, “blue”);

var uniqueDiv = $(“#unique”);
console.log(uniqueDiv.css(“background-color”);
console.log($(“div”).css(“margin-left”));

$(“div”).css( {
        margin-left: “5em”,
        margin-right: “1em”,
        margin-top: “1em”,
        margin-bottom: “0em”
});
```

You can pass an **object** to the css function to apply several properties all at once

Multiple assignment

```
$("div").css( {
        margin-left: "5em",
        margin-right: "1em",
        margin-top: "1em",
        margin-bottom: "0em"
});


 $("div").css("margin-left", "5em")
         .css("margin-right": "1em")
         .css("margin-top", "1em")
         .css("margin-bottom",  "0em");
```

All setter methods return the modified $ object, allowing you to chain together many calls in succession.

This has performance benefits

CSS classes

- The css function is only one method - there are many more

- You can add, remove, or toggle CSS class assignments

```
$("div").addClass("special");
$("div").removeClass("special");
$("div").toggleClass("special");
```

HTML Attributes

Any attribute on an HTML element can be accessed with the `attr` function

```
$("form").attr("method", "get");
$("img.special").attr("src", "special.png");
console.log($("a").attr("href"));
console.log($("div p span").attr("id"));
```

Note - **don't** use the **style attribute** - that will blow away other styling!

Form elements

There is a specialized function to retrieve and set the value of form elements

```
$("#firstName").val();
$("#lastName").val("a new last name");
$("input:text").val("default");
$("input[name=firstName]").val("a new first name");
```

Note also the use of type selectors

Method Chaining

Now we start to see the value of the method chaining functionality

```
$(“text:input”).attr(“name”, “firstName”)
    .addClass(“required”)
    .css(“font-weight”, “bold”);
```

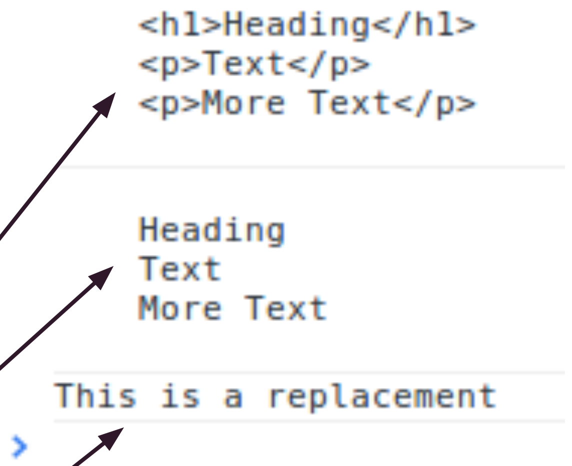
This is also called a *fluent interface*.

Element content

The text() and html() functions let you retrieve the content of elements themselves

```
<div>
    <h1>Heading</h1>
    <p>Text</p>
    <p>More Text</p>
</div>
```

```
console.log($(“div”).html());
console.log($(“div”).text());
$(“div”).html ( “<p>This is a replacement</p>” );
console.log($(“div”).text());
```



```
<h1>Heading</h1>
<p>Text</p>
<p>More Text</p>

Heading
Text
More Text

This is a replacement
```

Where do you put this?

- Recall - all JavaScript code at global scope (not in functions) runs as soon as the browser sees it.
  - **Note** - if the script element with the code is in head, its quite unlikely the entire page has loaded when your code executed!
- We got around this using window.load
  - For a variety of reasons (Internet Explorer), this is a bit dangerous.
- jQuery provides a more reliable method

```
<script>
    $( function () {
        console.log("Executes as soon as the page is completely loaded.");
    } );
</script>
```

Manipulating the DOM

- You can create new element using the $ function
    - `var a = $(“<p> new text </p>“)`

- Whenever the “selector” string has raw html, it is interpreted as the creation of a new element
- The new paragraph is **not** inserted yet
    - `a.appendTo(“#parent“);`
    - or `$(“#parent“).append(a)`
    - or `$(“#parent“).append(“<p> new text</p>“);`

Example - build a TOC

- Locate a div with id = “toc”
- Insert a link to all h1 elements within the toc div

Other ways to change DOM

append and **appendTo** have counterparts - **prepend** and **prependTo**

You can replace elements with **replaceWith** or **replaceAll**

```
$("div").replaceWith("<span>");
$("span").replaceAll("<div>");
```

More element operations

- Element removal
  - `$(".toRemove").remove();`
    - Remember - all these functions are performed on ALL the elements selected!
- Element copying
  - `$(".special").clone().appendTo("#here");`
- Element wrapping
  - `$("i").wrap("b")`
    - turns all `<i>..</i>` to `<b><i>..</i></b>`

Iterating over sets

Often you'll want to perform some custom operation on a bunch of elements

```
$("div").each ( function ()  {
    console.log(this);
    $(this).css("color", "blue");
});
```

this is the raw node
$(this) forms a proper jQuery object around the node

```
$("div").each ( function (index)  {
    var n = $("<span>" + index + "</span>");
    n.addClass("number");
    $(this).prepend(c)
});
```

Associating data with elements

- We start to think about the DOM as our program’s “data”.
- It's often helpful to store data *on* particular elements for later retrieval
  - You can do this with data-* attributes
  - Or you can use the **data** function in jQuery

```
$(“#myelement”).data(“x”, “5”);
console.log($(“#myelement”).data());
```

Why associate data?

Lets say we have a list of addresses

```
<ul>
    <li id="a1">...address 1 text...</li>
    <li id="a1">...address 2 text...</li>
    ....
</ul>
```

If we want to put them on a map, we'll need to geocode them and know their latitude and longitude

Why associate data?

So we could geocode each address and store the lat/long data with the element

```
$(“li”).each(function () ) {
    var a = $(this).text()
    var geo = magic_geocoding_oracle(a);
    $(this).data(“geo“, geo“);
}
```

Then, we could get that data later (perhaps to draw it on a map) when its clicked…

```
$(“#al”).data(“geo”).latitude
$(“#al”).data(“geo”).longitude
```

Events

One of the reasons jQuery absolutely took off what its powerful event mechanisms

Old days…

```
<button onclick="dostuff('a1')" id="a1">A1</button>
<button onclick="dostuff('a2')" id="a2">A2</button>
<button onclick="dostuff('a3')" id="a3">A3</button>
```

With jQuery

```
<button id="a1">A1</button>
<button id="a2">A2</button>
<button id="a3">A3</button>
```

```
… elsewhere…
      $("button").on("click", function() {
            doStuff($(this).attr("id"));
      });
```

Why is that any good?

- Attaching event handlers in pure JavaScript keeps your HTML clean
  - HTML is error prone
  - Embedding lots of JavaScript in HTML is even worse.
- It also allows you to decide what to do with events in a central location, rather than sprinkled throughout the entire HTML site

Often designers work on the HTML, and hand it over to a programmer. Programmer adds the programming…

This keeps things nice and separate

Event Registration

- There are actually two ways of registering events with jQuery
  - Older way: `$("div").click(function() { ... } );`
  - Newer way: `$("div").on("click", function() {...});`
- Why the change?
  - The older style was a bit more complicated, especially in the way it handled changes to the document - so called "live" events.
  - In some ways, on is easier - however the most important point is that on is the one that is going to be improved over time - not the older ones

Direct vs. Delegated Handlers

Lets say you execute

```
$(“div”).on(“click”, function() { alert(“clicked”); });
```

Then you do this:

```
$(“body”).append(“<div>New Div</div”);
```

- You’ll notice that unlike the other events, your new div element doesn’t respond!
- This is because the new div wasn’t around when you registered the event handler. Your registration was considered **directly bound**.

Direct vs. Delegated Handlers

The **on** function has an alternative representation:

Direct: `$("div").on("click", function() { alert("clicked"); });`

Delegated: `$("body").on("click", "div", function() { alert("clicked"); });`

- The delegated version says - "for every div under body, apply this handler"
  - Note - if you click on body (not a div in body), you will not see the alert
  - But now, if you add a div later, it will still get the event

In the delegated version, your selector should select parents of the elements you wish to attach your handler to.

Removing handlers

- The function **off** works in the reverse, removing the provided handler

- `$(“p”).off(“click”)`
  - removes all handlers attached to click event

- `$(“p”).off(“click”, myfunction)`
  - removes the specified handler (won’t work with anonymous handlers)

One

If you want an action to take place only on the first time something is clicked - you could do this.

```
function f() {
    alert( "This will be displayed only once." );
    $(this).off("click", f)
}
$("#foo").on("click", f) ;
```

Or this

```
$( "#foo" ).one( "click", function() {
 alert( "This will be displayed only once." );
});
```

Other events

jQuery doesn't just support "click" - it supports all the intrinsic events defined by the HTML standard

blur	focusin	mousedown	mouseup
change	focusout	mouseenter	resize
click	keydown	mouseleave	scroll
dblclick	keypress	mousemove	select
error	keyup	mouseout	submit
focus	load	mouseover	unload

Event objects

All event handlers are passed an **Event object**

```
$("div").on("mousemove", function(e) {
    console.log(e.offsetX + " / " + e.offsetY);
});
```

- offsetX and offsetY are pixel locations of mouse (relative to parent element)
- pageX and pageY are relative to the page
- See http://api.jquery.com/category/events/event-object/ for additional properties associated with the event

Triggering events

- Often you go through a lot of work to write a really nice event handler, and then you want to have the *same* behavior happen based on your code.
- You can programmatically "fake" and event using the **trigger** method
- `$("#myForm").trigger("submit");`
- This is especially helpful when you've blocked the default event from being processed

Note: If you return false from an event handler, it has the effect of calling "preventDefault" and "stopPropagation"

Show, Hide, and Animations

- jQuery provides some basic animation / effect methods that can hide/show elements
    - `$("div").hide()`
    - `$("div").show()`
    - `$("div").toggle()`

- These are immediate - if you prefer to fade
    - `$("div").fadeIn()`
    - `$("div").fadeOut()`
    - `$("div").fadeOut("fast", function() { alert("gone"); });`
    - Functions to call when fade is done, along with speeds (or absolute times) can also be provided.

Animation Queue

Each animation you kick off enters a queue, so it is possible to chain them together.

```
$(".blink").fadeOut(300)
          .fadeIn(300)
          .fadeOut(300)
          .fadeIn(300);
```

You could achieve similar with callbacks for when the fades complete… but this is much cleaner.

Other animations

slideDown, slideUp, slideToggle

Also, you can create custom animations

```
$("img").animate( { height: 100px, width: 100px})
```

All of these are considered **effects**

http://api.jquery.com/category/effects/

Guessing Game... again...

Revise the client-side Guessing Game to use jQuery.

- Set event handlers
- Apply classes to new elements
- Toggle visibility
- Animate