CSCC09F
Programming on the Web

jQuery

motivation, binding, event handling,
reading and writing DOM elements

What is jQuery?

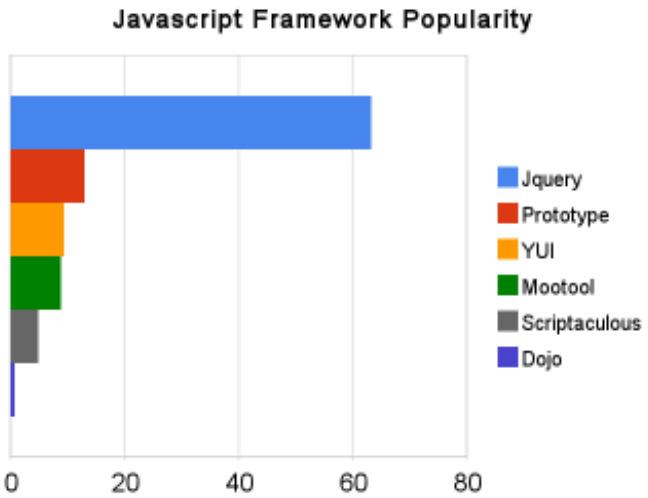
- A JavaScript library created by John Resig that simplifies client-side programming tasks and solves messy issues such as cross-browser code compatibility (write once, deploy across many browsers)
  - somewhat analogous to a server-side framework, but for the client side
  - peer libraries include Scriptaculous and Prototype
- Implemented as a JavaScript file; to use jQuery include a reference to its definition file in your html document head element using a **`<script>`** element
- Delivers huge boost in productivity and sophistication over coding in native JavaScript

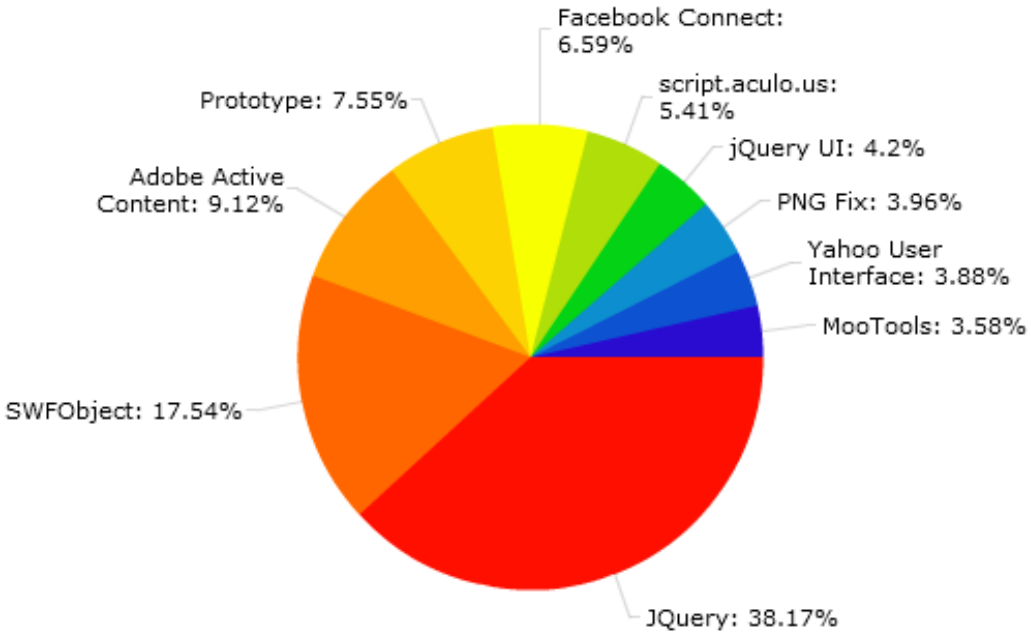
Why jQuery?

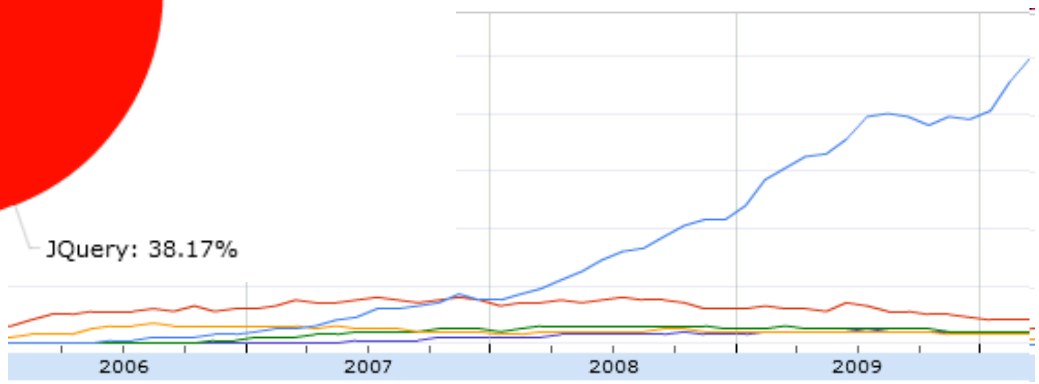
- Isn't this course about learning the fundamentals, not frills? Can't we do everything necessary in plain JS + DOM?
- Yes, but … writing JavaScript code is a tedious, time-consuming, and error-prone process, e.g.:
  - significant effort required for DOM navigation
  - need to accommodate browser differences
  - unexpected effects due to browser loading order
  - lack of mature tools to support development
- Learning the fundamentals doesn't mean we have to do everything the hard way (think back to Python in 1st year … you used sort() before learning how to code it for yourself)

Why jQuery?

- Why jQuery, not Scriptaculous, Prototype, or … ? jQuery most widely adopted, lots of momentum







Why jQuery?

- A few reasons why jQuery seems more powerful and easier to use than other libraries:
  - queries can be chained together to perform complex tasks
    - since each jQuery operation returns a jQuery value
  - result sequences can be referenced as a single unit e.g.: **`$('.tab').hide()`** to hide all elements of class tab
  - API, including its use of CSS selector notation, is intuitive, consistent, and common-sense, so even with only a little knowledge you can usually achieve your desired result
  - jQuery has an extensible plugin architecture that has spawned a large community of plugin features useful for building more sophisticated RIA/Web 2.0 type Web apps with much less time and effort
- Some of these are matters of opinion, but there's no disputing jQuery is now the most popular JS library

Why jQuery?

- Simplifies coding, making it easier to get right
- With a single consistent interface you can control:
  - DOM interaction
  - Event handlers
  - Style properties
- Consistency across browsers; a major headache when using plain JS + DOM, one of the Achilles' heels of Web-app development
- Takes care of error-prone hard-to-debug race conditions between object creation and object use (JS referring to a page element that isn't yet loaded – its value is undefined)
- Good for your resume

Why a JavaScript Library?

- Cross-browser compatibility in native JavaScript requires extra code/checks, adds to maintenance effort
  - Although there is a standard DOM API, it is inconsistently implemented across browsers
  - to support older browsers, you have to also make your code backward-compatible
  - JavaScript libraries abstract cross-browser and backward-compatibility issues
- The DOM API is not very expressive
  - common operations require more code than they should. Code written using a JavaScript library is typically more concise and expressive than code operating on the raw DOM

jQuery: What's in a Name?

- The name is catchy, but is there more to it?
- jQuery is structured around "queries"
- jQuery takes CSS's selector design and applies it to bind document elements to actions rather than style – very cool idea
- A query can use either CSS or XPath selectors to identify a collection of elements in the document
- JavaScript/jQuery operations are then performed against those selected elements

jQuery: it's all about the $

- jQuery exports a single function object, "jQuery", into the global namespace of an importing document
- For convenience, jQuery is aliased to variable name "$", which being much easier to type, is almost always used in lieu of "jQuery"
- $() has a number of properties (mostly methods) that perform useful tasks
- $() takes a single argument, and always returns a jQuery collection
- The type of the argument determines how $() behaves

jQuery: it's all about the $

- $() returns one of 4 different results, depending on the type of its argument:

1. CSS selector: returns a jQuery collection of DOM elements that match that selector
2. HTML string: creates a new DOM element from the HTML string, and returns it as a jQuery collection.
3. DOM element: returns the DOM element as a jQuery collection
4. Absent: returns an empty jQuery collection

DOM Selection Uses CSS Selectors

- Element Type `E`
- Grouping `E, F, G`
- Universal `*`
- Class `[E].classvalue`
- Id `[E]#myID`

(element name E optional – meta brackets)

- Contextual
  - Descendent `E F` (prior jQuery example)
  - Child `E > F`
  - Adjacent `E + F`
- Pseudo-element `E:pseudo-element`
- Attribute `E[foo="hi"]` (literal brackets)

JavaScript DOM-Element Selection

- Getting Elements using JavaScript DOM:
  - **`document.getElementById("idval")`**
    - returns unique DOM object with an id attribute of idval (id-values must be unique within documents)
  - **`document.getElementsByTagName("tagname")`**
    - returns an array of DOM objects, all of type tagname.
    - powerful, but now must process the array using a loop

jQuery DOM-Element Selection

- Getting Elements with jQuery:
  - **`$("#idval")`**
    - select unique element with “id=idval”
  - **`$(".classname")`**
    - selects all elements with matching classname
  - **`$("tagname")`**
    - selects all elements with matching tagname
  - returns collection of jQuery objects corresponding to the DOM elements that match the selection criteria

jQuery: Differences from POJS DOM

- One of the claims for jQuery is that it reduces the verbosity of Plain Old JavaScript (POJS) DOM references
- For example:

```
var domTable = document.getElementById("mytable");
var jqTable = $("#mytable");
```

- The differences go more than skin deep, however:
  - domTable is a raw DOM-element object, whereas jqTable is a jQuery object
- A jQuery object behaves like a wrapper around a DOM object, that confers the ability to apply jQuery methods, rather than DOM methods

jQuery vs POJS

- Let's consider a few simple examples to compare how things are done in jQuery vs plain old JavaScript (POJS) with the Document Object Model (DOM)
- Suppose we want to associate a click event with each link within a particular area of a page:
- using JavaScript and DOM:

```
// confirm user hypertext links upon click
var external_links =
      document.getElementById('external_links');
var links = external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
  var link = links.item(i);
  link.onclick = function() {
   return confirm('You are going to visit: ' + this.href);
  };
}
```

jQuery vs POJS

- using JavaScript and DOM:

```
var external_links =
    document.getElementById('external_links');
var links = external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
  var link = links.item(i);
  link.onclick = function() {
    return confirm('You are going to visit: ' +
       this.href);
  };
}
```

- using jQuery:

```
$('#external_links a').click(function() {
  return confirm('You are going to visit: ' +
         this.href);
});
```

jQuery vs POJS

- Suppose we want to create a new paragraph <p> and add it at the end of the existing page:
- using JavaScript DOM:

```
var new_p = document.createElement("p");
var new_text = document.createTextNode("Hello World");
new_p.appendChild(new_text);
var bodyRef =
   document.getElementsByTagName("body").item(0);
bodyRef.appendChild(new_p);
```

- Typical POJS pattern: incrementally build elements and glue (append) them together; only when glued to current document do the new elements become visible

jQuery vs POJS

- using JavaScript DOM:

```
var newPP = document.createElement("p");
var newTxt = document.createTextNode("Hello World");
newPP.appendChild(newTxt);
var bodyRef =
    document.getElementsByTagName("body").item(0);
bodyRef.appendChild(newPP);
```

- using jQuery:

```
$('<p></p>').html('Hello World!').appendTo('body');
```

  - Create an element by quoting the HTML (' ')
  - Set it's content by calling html() with parameter value content
  - Add it to the end of the document (appendTo body element)
  - jQuery excels at "chaining" together actions in a fluid way

jQuery vs POJS

- Suppose we want to bind a handler-function to an event associated with a particular element.
- using JavaScript DOM we use an on-event attribute to bind builtin function alert to a click event:

```
<a href="" onclick="alert('Hello world')">hey</a>
```

- using jQuery:

```
$(document).ready(function() {
    $("a").click(
        function() { alert("Hello world!"); }
    );
});
```

- wait a minute, how is that an improvement?!

jQuery vs POJS

```
<a href="" onclick="alert('Hello world')">hey</a>
```

- versus

```
$(document).ready(function() {
    $("a").click(
        function() { alert("Hello world!"); }
    );
});
```

- The difference is that the jQuery version is handling every single **<a>** element, POJS just a single element
- Just as CSS separates presentation from structure, jQuery separates behavior from structure.
- This is one of the key insights that has made jQuery such a powerful & successful JavaScript library.