JavaScript Frameworks

Problems with JavaScript

JavaScript is a powerful language, but it has many flaws:

- the DOM can be clunky to use
- the same code doesn't always work the same way in every browser
  - code that works great in Firefox, Safari, ... will fail in IE and vice versa
- many developers work around these problems with hacks (checking if browser is IE, etc.)

Frameworks

- Web application frameworks, or simply "web frameworks", are the de facto way to build web-enabled applications
  - Supports the development of web applications including web services, web resources and web APIs
  - Alleviate the overhead associated with common activities performed in web development
- A framework promotes code reuse

Frameworks vs. Libraries

- A framework differs from a library in one very important way:
  - library code is always called by code that you write
  - A framework always calls code that you write.
- When one uses a framework:
  - Required to cede a greater portion of control to code that resides in the framework itself.
- There is no need to use a framework at all to create a web application
  - A framework is more practical if the framework provides a set of facilities that fits your application requirements.

Prototype framework

```
<script src="
https://ajax.googleapis.com/ajax/libs/prototype/1.7.0.0/prototype.js "
type="text/javascript"></script>                          JS
```

- A JavaScript framework created by Sam Stephenson in 2005
  - Although many have been downgraded `Prototype` to a library!

Prototype framework

- the Prototype JavaScript library adds many useful features to JavaScript:
  - many useful extensions to the DOM
  - added methods to `String`, `Array`, `Date`, `Number`, `Object`
  - improves event-driven programming
  - many cross-browser compatibility fixes
  - makes Ajax programming easier (seen later)

Prototype framework

- Ajax utilities
  - `Ajax.Request`, `Ajax.Updater`, `Ajax.PeriodicalUpdater`
  - Wraps response in `Ajax.Response`
    - Several new properties, but especially `responseJSON`
- General DOM utilities
  - `$(id…)` to find element
    - The $ function is the cornerstone of Prototype
    - Takes in an arbitrary number of arguments.
    - Returns one Element if given one argument; otherwise returns an Array of Elements
  - `$F(element)` to get form value
    - A convenient alias for `Form.Element.getValue`
  - `$$(cssRule…)`
    - Takes an arbitrary number of CSS selectors (strings) and returns a document-order array of extended DOM elements that match any of them.
  - `element.update()` to put into `innerHTML`
  - Many helpers in Element class
- General utilities
  - Extensions for `Class`, `Function`, `Array`, `String`

The $ function

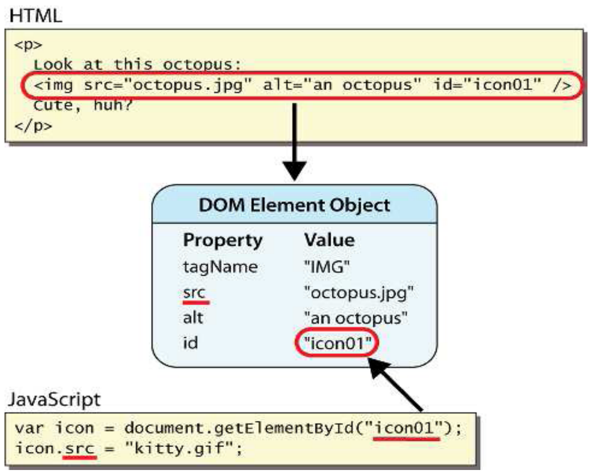
```
$("id")                                                    JS
```

- returns the DOM object representing the element with the given id
- short for `document.getElementById("id")`
- often used to write more concise DOM code:

```
$("footer").innerHTML = $("username").value.toUpperCase();
                                                           JS
```

Warning: $() function in prototype is not the same as the $() function in jQuery. The jQuery's $() function can be described as a more powerful version of Prototype's $$()

DOM element objects



HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

DOM object properties

```
<div id="main" class="foo bar">
<p>Hello, I am <em>very</em> happy to see you!</p>
<img id="icon" src="images/potter.jpg" alt=“Potter" />
</div>
```

HTML

Property	Description	Example
tagName	element's HTML tag	$("main").tagName is "DIV"
className	CSS classes of element	$("main").className is "foo bar"
innerHTML	content inside element	$("main").innerHTML is "\n <p>Hello, <em>ve...
src	URL target of an image	$("icon").src is "images/potter.jpg"

DOM properties for form controls

```
<input id="sid" type="text" size="7" maxlength="7" />
<input id="frosh" type="checkbox" checked="checked" />
Freshman?
```

HTML

Property	Description	Example
value	the text in an input control	$("sid").value could be "1234567"
checked	whether a box is checked	$("frosh").checked is true
disabled	whether a control is disabled (boolean)	$("frosh").disabled is false
readOnly	whether a text box is read-only	$("sid").readOnly is false

Abuse of innerHTML

```
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a
href="page.html">link</a>";
```

JS

- innerHTML can inject arbitrary HTML content into the page
- however, this is prone to bugs and errors and is considered poor style

Adjusting styles with the DOM

```
<button id="clickme">Color Me</button>                 HTML
```

```
window.onload = function() {
        document.getElementById("clickme").onclick = changeColor;
};

function changeColor() {
        var clickMe = document.getElementById("clickme");
        clickMe.style.color = "red";
}                                                              JS
```

- contains same properties as in CSS, but with camelCasedNames
  - examples: `backgroundColor, borderLeftWidth, fontFamily`

Common DOM styling errors

- forgetting to write .style when setting styles:

```
var clickMe = document.getElementById("clickme");
~~clickMe.color = "red";~~
clickMe.style.color = "red";                             JS
```

- style properties are capitalized likeThis, not like-this:

```
~~clickMe.style.font-size = "14pt";~~
clickMe.style.fontSize = "14pt";                         JS
```

- style properties must be set as strings, often with units at the end:

```
~~clickMe.style.width = 200;~~
clickMe.style.width = "200px";
clickMe.style.padding = "0.5em";                         JS
```

Unobtrusive styling

```
function okayClick() {
        ~~this.style.color = "red";~~
        this.className = "highlighted";
}                                                        JS
```

```
.highlighted { color: red; }                             CSS
```

- well-written JavaScript code should contain as little CSS as possible
- use JS to set CSS classes/IDs on elements
- define the styles of those classes/IDs in your CSS file

Timer events

method	description
setTimeout(function, delayMS);	arranges to call given function after given delay in ms
setInterval(function, delayMS);	arranges to call function repeatedly every delayMS ms

clearTimeout(timerID);
clearInterval(timerID);
	stops the given timer so it will not call its function

- both `setTimeout` and `setInterval` return an ID representing the timer
  - this ID can be passed `to clearTimeout/Interval` later to stop the timer

`setTimeout` example

```
<button onclick="delayMsg();">Click me!</button>
<span id="output"></span>                                   HTML
```

```
function delayMsg() {
        setTimeout(booyah, 5000);    // 1000 ms = 1 second
        $("output").innerHTML = "Wait for it...";
}
function booyah() { // called when the timer goes off
        $("output").innerHTML = "BOOYAH!";
}                                                            JS
```

`setInterval` example

```
<button onclick="delayMsg2();">Click me!</button>
<span id="output"></span>                                   HTML
```

```
var timer = null; // stores ID of interval timer
function delayMsg2() {
        if (timer == null) {
                timer = setInterval(rudy, 1000);
        } else {
                clearInterval(timer);
                timer = null;
        }
}
function rudy() { // called each time the timer goes off
        $("output").innerHTML += " Rudy!";
}                                                            JS
```

Passing parameters to timers

```
function delayedMultiply() {
// 6 and 7 are passed to multiply when timer goes off
        setTimeout(multiply, 2000, 6, 7);
}
function multiply(a, b) {
        alert(a * b);
}                                                            JS
```

- any parameters after the delay are eventually passed to the timer function
- why not just write this?

```
setTimeout(multiply(6 * 7), 2000);
                                                             JS
```

Common timer errors

```
~~setTimeout(booyah(), 2000);~~
setTimeout(booyah, 2000);
~~setTimeout(multiply(num1 * num2), 2000);~~
setTimeout(multiply, 2000, num1, num2);
                                                  JS
```

- what does it actually do if you have the () ?
  - it calls the function immediately, rather than waiting the 2000ms!

Unobtrusive JavaScript

The six global DOM objects

name	description
document	current HTML page and its content
history	list of pages the user has visited
location	URL of the current HTML page
navigator	info about the web browser you are using
screen	info about the screen area occupied by the browser
window	the browser window

The window object

- *the entire browser window; the top-level object in DOM hierarchy*
- technically, all global code and variables become part of the window object properties:
  - document, history, location, name
- methods:
  - alert, confirm, prompt (popup boxes)
  - `setInterval, setTimeout clearInterval, clearTimeout` (timers)
  - `open, close` (popping up new browser windows)
  - `blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo`

The document object

- *the current web page and the elements inside it*
- properties:
  - `anchors, body, cookie, domain, forms, images, links, referrer, title, URL`
- methods:
  - `getElementById`
  - `getElementsByName`
  - `getElementsByTagName`
  - `close, open, write, writeln`

The location object

- *the URL of the current web page*
- properties:
  - `host, hostname, href, pathname, port, protocol, search`
- methods:
  - `assign, reload, replace`

The navigator object

- *information about the web browser application*
- properties:
  - `appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent`
- Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and hacks:

```
if (navigator.appName === "Microsoft Internet Explorer") {
...
                                                          JS
```

The screen object

- *information about the client's display screen*
- properties:
  - `availHeight, availWidth, colorDepth, height, pixelDepth, width`

The history object

- the list of sites the browser has visited in this window
- properties:
  - `length`
- methods:
  - `back, forward, go`
- sometimes the browser won't let scripts view `history` properties, for security

Unobtrusive JavaScript

- JavaScript event code seen previously was *obtrusive*, in the HTML; this is bad style
- now we'll see how to write unobtrusive JavaScript code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions

Unobtrusive JavaScript

- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?

Obtrusive event handlers (bad)

```
<button id="ok" onclick="okayClick();">OK</button>
                                                    HTML
```

```
// called when OK button is clicked
function okayClick() {
      alert("booyah");
}                                                   JS
```

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

Attaching an event handler in JavaScript code

```
// where element is a DOM element object
element.event = function;
                                                    JS
```

```
$("ok").onclick = okayClick;
                                                    JS
```

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do **not** put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

When does my code run?

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>                                    HTML
```

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);                                               JS
```

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them

When does my code run?

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>                                    HTML
```

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);                                               JS
```

- at this point in time, the browser has not yet read your page's body
  - none of the DOM objects for tags on the page have been created

A failed attempt at being unobtrusive

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body>
<div><button id="ok">OK</button></div>               HTML
```

```
// global code
$("ok").onclick = okayClick; // error: $("ok") is null
                                                        JS
```

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM

The `window.onload` event

```
// this will run once the page has finished loading
function functionName() {
      element.event = functionName;
      element.event = functionName;
...
}
window.onload = functionName; // global code
                                                     JS
```

- we want to attach our event handlers right after the page is done loading
  - there is a global event called `window.onload` event that occurs at that moment
- in `window.onload` handler we attach all the other handlers to run when events occur

An unobtrusive event handler

```
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
                                                   HTML
```

```
// called when page loads; sets up event handlers
function pageLoad() {
      $("ok").onclick = okayClick;
}
function okayClick() {
      alert("booyah");
}
window.onload = pageLoad; // global code
                                                   JS
```

Common unobtrusive JS errors

```
~~window.onload = pageLoad();~~
window.onload = pageLoad;
~~okButton.onclick = okayClick();~~
okButton.onclick = okayClick;
                                                   JS
```

- □ event names are all lowercase, not capitalized like most variables

```
~~window.onLoad = pageLoad;~~
window.onload = pageLoad;
                                              JS
```

Anonymous functions

```
function(parameters) {
      statements;
}
                                                   JS
```

- □ JavaScript allows you to declare anonymous functions
- □ quickly creates a function without giving it a name
- □ can be stored as a variable, attached as an event handler, etc.

Anonymous function example

```
window.onload = function() {
      var okButton = document.getElementById("ok");
      okButton.onclick = okayClick;
};
function okayClick() {
      alert("booyah");
}
                                                   JS
```

The keyword `this`

```
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
                                                        JS
```

- all JavaScript code actually runs inside of an object
- by default, code runs inside the global window object
  - all global variables and functions you declare become part of window
- the `this` keyword refers to the current object

The keyword `this`

```
function pageLoad() {
        $("ok").onclick = okayClick; // bound to okButton
here
}
function okayClick() { // okayClick knows what DOM object
        this.innerHTML = "booyah"; // it was called on
}
window.onload = pageLoad;
                                                        JS
```

- event handlers attached unobtrusively are **bound** to the element
- inside the handler, that element becomes this (rather than the window)

Fixing redundant code with `this`

```
<fieldset>
        <label><input type="radio" name="ducks" value="Huey" />
Huey</label>
        <label><input type="radio" name="ducks" value="Dewey" />
Dewey</label>
        <label><input type="radio" name="ducks" value="Louie" />
Louie</label>
</fieldset>                                             HTML
```

```
function processDucks() {
        ~~if ($("huey").checked) {~~
~~                alert("Huey is checked!");~~
~~        } else if ($("dewey").checked) {~~
~~                alert("Dewey is checked!");~~
~~        } else {~~
~~                alert("Louie is checked!");~~
~~        }~~
        alert(this.value + " is checked!");
}                                                       JS
```

Example: Tip Calculator

```
<h1>Tip Calculator</h1>
<div>
        $<input id="subtotal" type="text" size= "5" /> subtotal <br />
        <button id="tenpercent">10%</button>
        <button id="fifteenpercent"> 15%</button>
        <button id="eighteenpercent"> 18%</button>

        <span id="total"></span>
</div>                                                  HTML
```

```
window.onload = function() {
        $("tenpercent").onclick = computeTip;
        }
function computeTip{
        var subtotal = parseFloat($("subtotal").value);
        var tipAmount = subtotal*0.1;//Add this code
        $("total").innerHTML = "Tip: $" + tipAmount;
}                                                       JS
```