Comp215: Web client development

Dan S. Wallach (Rice University)

Copyright © 2015, Dan S. Wallach. All rights reserved.
First, a reminder of last week

The web is:

Browsers (running JavaScript, loading HTML, images, etc.)
  Also mobile clients (iOS/Android/etc.)

Servers (written in all sorts of different programming languages)

Protocols (HTTP, possibly encrypted, etc.)
  Also higher-level things on top of HTTP, like sending JSON-formatted messages

Networks (the Internet)
JavaScript is:

A dynamically typed scripting language, not unlike Python

\texttt{var \ x = \ldots;} \ The \ type \ of \ \ x \ isn't \ known \ until \ runtime.

With \textbf{lambdas everywhere and an unusual object system}

Based on “prototypes” rather than “classes” like Java

With a “standard” set of \textbf{APIs} for dealing with the browser

Today’s lecture: JavaScript and the browser
Digression: Traditional graphics programming

In the old days, graphics APIs exposed low-level things
Draw lines & rectangles, render text
High-performance 3D graphics was typically an unrelated API

UI widgets exposed higher-level things
Buttons, sliders, text boxes

Wildly varying APIs for dealing with system events (mouse, keyboard)
Event queues, callbacks, and more

Many current systems (e.g., Android, iOS) follow a similar pattern
The web is much nicer

Sophisticated fonts, layouts, style sheets, etc.
And non-programmer tools (Adobe DreamWeaver, etc.) to help out

Web 1.0: lay out a static web document, decorate it with JavaScript behaviors
onClick, onMouseOver, etc. placed discretely into the HTML

But that’s not really how it’s done today
What, exactly, is “web 2.0”? 

Much debate on what this term even means. Here’s my definition.

The web browser is hosting a dynamic application, not just a static page
  If you disable JavaScript, you don’t just get a limited page. You get nothing.

The web server responds to requests from client applications
  These might be JSON, XML, protobufs, whatever.

“Content” might come from users, so there are interesting security issues
  More on that next week.

Web pages aggregate stuff from multiple sources
  Advertising, “tracking”, “mashups”, etc.
  And again, lots more interesting security issues.
The two biggest Web 2.0 features

“Ajax” (asynchronous JavaScript): JS code can make its own network connections, fetch data, without forcing the page to reload.

DOM (document object model): JS code can reach deep into a web page and change its structure and contents, on the fly.

Notable: XMLHttpRequest invented in 1999 by Microsoft for Outlook Web Access
Rapidly adopted by other browsers.
Formed the basis for Google’s Gmail and other web apps to come.

Also of note, “HTML5”: attempts to standardize all of this
But older browsers, especially IE, don’t follow HTML5
Marc Andreessen’s (?) predictions (1995?)

Anderson: A quote of yours that I’ve always loved is that Netscape would render Windows “a poorly debugged set of device drivers.”

Andreessen: In fairness, you have to give credit for that quote to Bob Metcalfe, the 3Com founder.

Anderson: Oh, it wasn’t you? It’s always attributed to you.

Andreessen: I used to say it, but it was a retweet on my part. [Laughs.] But yes, the idea we had then, which seems obvious today, was to lift the computing off of each user’s device and perform it in the network instead.

http://www.wired.com/2012/04/ff_andreessen/
Putting it together

Example app: our RPN calculator, adventure game shell, etc.

HTML: set up the layout of the screen, load the JavaScript
CSS (cascading style sheets): fonts, etc.
JavaScript: add behaviors
Comp215 RPN Calculator!
RPNCalc’s HTML (generated by Java code)
<html>
<head>
	<title>Comp215 RPN Calculator!</title>
	<meta charset="utf-8">
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<meta name="viewport" content="width=device-width, initial-scale=1">
	<link href="/mui-0.1.18/css/mui.css" rel="stylesheet" type="text/css">
	<link href="/comp215.css" rel="stylesheet" type="text/css">
	<script type="text/javascript" src="/mui-0.1.18/js/mui.min.js"></script>
	<script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/react/0.13.2/react.min.js"></script>
	<script type="text/javascript" src="//cdnjs.cloudflare.com/ajax/libs/react/0.13.2/JSXTransformer.js"></script>
	<script type="text/javascript" src="/snack-min.js"></script>
	<script type="text/javascript" src="/rpncalc.js"></script>
</head>
<body>
<header id="header">
	<nav id="appbar" class="mui-container-fluid">
		<h1>Comp215 RPN Calculator!</h1>
	</nav>
</header>
<div id="textOutput"></div>
<div id="goButton">
	<button class="mui-btn mui-btn-floating mui-z3">&mdash;</button>
</div>
<div id="footer">
<form>
	<div class="mui-form-group">
		<input id="commandLine" type="text" class="mui-form-control" placeholder="type your commands here">
	</div>
</form>
</div>
</body>
</html>
RPNCalc’s HTML (generated by Java code)

Every element has a “type”, an optional “id”, and optional “classes”
What's a “class” and an “id”?

**Class**: handles into the style sheets for fonts, spacing, etc.

```
.mui-form-control {
  -webkit-animation-duration: 0.0001s;
  animation-duration: 0.0001s;
  -webkit-animation-name: mui-node-inserted;
  animation-name: mui-node-inserted;
  display: block;
  background-color: transparent;
  color: rgba(0, 0, 0, 0.87);
  border: none;
  border-bottom: 1px solid rgba(0, 0, 0, 0.26);
  outline: none;
  height: 32px;
  width: 100%;
  font-size: 16px;
  padding: 0;
  box-shadow: none;
  border-radius: 0px;
  background-image: none;
}
```

**Id**: handles that JavaScript can grab onto for adding behaviors

```javascript
document.getElementById('commandLine')
```
Do you need to understand CSS?

No, not really
Suffice to say that multiple rules can attach to any given element in a document. CSS provides rules (and exceptions to the rules) for how to resolve all that.

There really isn’t a whole lot of magic behind CSS.
But there are a lot of people complaining about how awful it is.

When in doubt, beat on it until you get what you want.
Lots of web pages with examples, reference guides, and of course you can read the “source code” to web sites that you like.
Do you need to understand JavaScript?

We introduced the JavaScript language last week

This week: JavaScript’s view of the browser environment: the Document Object Model (DOM)

Several top-level objects and everything else is below that
Learning your way around the DOM

See the real-time status of the DOM (not just the HTML that made it)
And see what CSS styles are attached to each object
Using the JavaScript Console

Type commands and see what they return

Helpful to do this while staring at the reference docs in another window
Where do you get started?

Lots of general documentation online, tons of StackOverflow posts, etc.
Mozilla’s intro is as good as anything:


Key concepts: document, element, window

document: the top-level view of the browser page

element: any individual widget in the page, as found above

function printRaw(text) {
    var textBox = document.getElementById('textOutput') // locate the textOutput widget on the screen
    textBox.innerHTML += text; // append the supplied text to the end of the widget
    textBox.scrollTop = textBox.scrollHeight; // scroll the page so we always see the newest stuff we just added
}

window: the container where the document lives

Common trick: setting window.location to a new URL causes the whole page to be reloaded
Everything wants a callback

Example: when the user clicks the button, we want to send the request to our remote web server

```javascript
snack.wrap(document.getElementById('goButton'))
  .each(function (element, index) {
    var params = {node: element, event: 'click'}
    snack.listener(params, function (event) {
      fetchQuery()
    })
  })
```

Q: What's with `snack.wrap`? Why not do it directly?
A: Portability (libraries like “snack” hide minor differences between browsers)

Here’s the modern way (but won’t work on IE6):

```javascript
var goButton = document.getElementById('goButton')
goButton.addEventListener('click', fetchQuery)
```
Notable: the window doesn’t load right away

No problem, just register for a callback

```javascript
window.onload = function() {
    printParagraph("<i><u>RPN calculator initialized, ready to rock</u></i>"
    // other objects are in the DOM, ready for more callbacks to be registered
    ...
}
```

Sometimes you just stick a lambda into an object (as above)
Other times, you “add” a listener (still a lambda)
Asynchronous requests
Of course, they’re not all compatible

Again, this is where the snack library tries to paper over the difference

```javascript
var xhrObject = (function(){
  // adapted from MooTools
  var XMLHttpRequest = function(){
    return new XMLHttpRequest();
  }

  var MSXML2 = function(){
    return new ActiveXObject('MSXML2.XMLHTTP');
  }

  var MSXML = function(){
    return new ActiveXObject('Microsoft.XMLHTTP');
  }

  try {
    XMLHttpRequest()
    return XMLHttpRequest
  } catch (e){
    try {
      MSXML2()
      return MSXML2
    } catch (e){
      MSXML()
      return MSXML
    }
  }

  return MSXML()
})();

Three different commands: let’s see if any of them work!
```
Generate a request, send it, wait for a response

Set up (another) lambda for the response callback

```javascript
function dispatchQuery(input) {
    var options = {
        method: 'get',
        url: '/rpnserver/',
        data: {'input': input}
    }

    snack.request(options, function (err, res){
        // check for an error
        if (err) {
            printParagraph('<b>Bah! ' + err + ' error!</b>')
            return
        }
        // no error
        handleResponse(res)
    })
}

function handleResponse(data) {
    var result = JSON.parse(data).response
    printParagraph(result)
}
```
Generate a request, send it, wait for a response

Set up (another) lambda for the response callback

```javascript
function dispatchQuery(input) {
    var options = {
        method: 'get',
        url: '/rpnserver/',
        data: {'input': input}
    }

    snack.request(options, function (err, res) {
        // check for an error
        if (err) {
            printParagraph('<b>Bah! ' + err + ' error!</b>')
            return
        }
        // no error
        handleResponse(res)
    })
}

function handleResponse(data) {
    var result = JSON.parse(data).response
    printParagraph(result)
}
```
Generate a request, send it, wait for a response

Set up (another) lambda for the response callback

```javascript
function dispatchQuery(input) {
  var options = {
    method: 'get',
    url: '/rpnserver/',
    data: {'input': input}
  }

  snack.request(options, function (err, res) {
    // check for an error
    if (err) {
      printParagraph('<b>Bah! </b>' + err + ' error!')
      return
    }
    // no error
    handleResponse(res)
  })
}

function handleResponse(data) {
  var result = JSON.parse(data).response
  printParagraph(result)
}
```

Lambda to handle the response when it comes back
Set up (another) lambda for the response callback

```javascript
function dispatchQuery(input) {
    var options = {
        method: 'get',
        url: '/rpnserver/',
        data: {'input': input}
    }

    snack.request(options, function (err, res){
        // check for an error
        if (err) {
            printParagraph('<b>Bah! </b>' + err + ' error!)</b>'
            return
        }
        // no error
        handleResponse(res)
    })
}

function handleResponse(data) {
    var result = JSON.parse(data).response
    printParagraph(result)
}
```

built-in safe JSON parsing (in recent JS, not in older browsers)
Where did you find “snack”, anyway?

Sometimes you need simple things
So go get a simple library!
I tried several “micro frameworks” until I found one that did what I needed.

Many “big” sites use “big” frameworks
jQuery, AngularJS, etc.
Or they just write in some other language.

Any advice I give you today will be completely obsolete in two years!
Special guest: Mark Maxham (Wiess ’90)