08 – Javascript

What is JavaScript?

- A script language for web browsers
  - Runs directly in the browser
  - can respond to user interactions quickly, without wide-area latencies
  - Can interact with web pages through the DOM
- Syntax very similar to Java
  - Similar operators, constructs, naming conventions...
  - This is intentional
- But very different from Java 'under the hood'
  - Dynamic typing, no inheritance, runtime evaluation...
  - Details can be a bit messy

A brief history of JavaScript

- Developed at Netscape (1995)
  - Author: Brendan Eich
  - Originally called 'Mocha', then 'LiveScript'
  - Became 'JavaScript' when Netscape and Sun got together
  - NO direct connection to Java!
- Microsoft's dialect: JScript (1996)
  - To avoid licensing issues
- Standardized as ECMAscript (1997)
- History of incompatibilities across browsers
  - Example: Different ways of getting XMLHttpRequest object in different browsers (see later)

A simple client-side example

- Example: Form validation
  - Warns the user if no search term is specified

```
<html>
  <head><title>Test</title></head>
  <script type="text/javascript">
  <!--
    function check(myform) {
      if (myform.term.value=="") {
        alert("Please enter a search term!");
        return false;
      } else {
        return true;
      }
   } // -->
  </script>
  <body>
    <h1>Input a search term</h1>
    <form method="post" action="process.php" onsubmit="return check(this)">
      <input type="text" name="term" size="20">
      <input type="submit" value="Search">
    </form>
  </body>
</html>
```

Prevents problems if browser does not support scripting

Script is embedded in the web page

Calls function when form is submitted; aborts submission when function returns false

Including JavaScript in HTML

- Option #1: Embed entirely in HTML document
  - As in previous example: Use <script>...</script>
  - To be safe, enclose script in HTML comments (to avoid confusing browsers that don't recognize JavaScript)
- Option #2: Attach as a separate file
  - <script type="text/javascript" src="myscript.js"> </script>
  - Some browsers need the space between <script>...</script> and don't load the script if this is omitted
- Remember that script is visible to the client!
  - Do NOT hardcode any passwords or include any secrets

Event handlers

- Client-side JS can react to various events
  - Examples: User clicks on an element or presses a key, user submits a form, user changes a selection in a form, page finishes loading, mouse moves over a certain element...
  - Web page can request that the browser call a certain JavaScript function when the event occurs
- Events can be requested from the web page:
  - <a href="foo.html" onClick="alert('You\'ve clicked!')">
- ... or directly from JavaScript:
  - theElement.onclick = functionName (DOM 0)
  - theElement.addEventListener(type, function, opt)

Document Object Model

- Document Object Model
  - An interface for manipulating HTML and XML documents
  - Document is represented as a tree of objects
  - Program can traverse the tree, add/delete/read/write nodes
    - We're not even scratching the surface in this course: DOM exists at several levels (0,1,2,3); Level 3 core spec has 216 pages!

<div> and <span>

- Primary use is to hold an identifier
  - We can use this identifier to find the element in the DOM
  - Otherwise, the elements do very little; <div> is rendered similar to <p>, and <span> does not affect text flow at all
  - Other elements can hold 'id' too, but have other functions

Accessing the DOM from JavaScript

- JavaScript programs can interact with the DOM
  - Document root provided as the 'document' object
  - Read from, and write to, elements in the DOM tree
  - How do we find the right element in the tree?

```
<html>
  <head><title>Test</title></head>
  <script type="text/javascript">
  <!--
    function replace() {
      var t = document.getElementById('abc').value;
      document.getElementById('xyz').innerHTML = t;
   } // -->
  </script>
  <body>
    <h1>Test</h1>
    <form action="xyz" method="get" onSubmit="replace(); return false">
      <input type="text" name="thetext" size="20" id="abc">
      <input type="submit" value="Replace">
    </form>
    <div id="xyz">(this is where the text will go)</div>
  </body>
</html>
```

Reads current contents of input field

Replaces the text in the 'div' element below

<div> and <span>

```
<html>
  <head><title>Test</title></head>
  <body>
    <h1>Test</h1>
    This is some normal text with
    a <span id="foo">span</span> in it.
    <div id="bar">And this is a div.</div>
    Here is some more normal text.
</html>
```

```
...
document.getElementById('foo').innerHTML = "<p>Some text</p>";
...
```

Functions for accessing the DOM

- The HTML page itself is called 'document'
- To get information from the document:
  - var price = document.getElementById('price').value;
  - var allimages = document.getElementsByName('img');
  - var firstimg = document.getElementsByName('img')[0];
- To put information into the document:
  - Create new elements; replace, or append to, existing nodes

- Finding HTML Elements
  - document.getElementById()
  - document.getElementsByTagName()
  - document.getElementsByClassName()
- Changing HTML Elements
  - element.innerHTML=
  - element.attribute=
  - element.setAttribute(attribute,value)
  - element.style.property=

- Adding and Deleting Elements
  - document.createElement()
  - document.removeChild()
  - document.appendChild()
  - document.replaceChild()
- Adding Events Handlers
  - document.getElementById(id).onclick=function(){code}
  -

Functions for accessing the DOM

- Find thing to be replaced
  - `var mainDiv = document.getElementById("main-page");`
  - `var orderForm = document.getElementById("target");`
- Create replacement
  - `var paragraph = document.createElement("p");`
  - `var text = document.createTextNode("Here is the new text.");`
  - `paragraph.appendChild(text);`
- Do the replacement
  - `mainDiv.replaceChild(paragraph, target);`

innerHTML

- Building new DOM subtree is a lot of work
  - Isn't there an easier way?
- Solution: innerHTML
  - A non-W3C DOM property that gets or sets the text between the start and end tags
  - Argument completely replaces elements' existing content
  - If the new value contains HTML tags, it is parsed and formatted before being placed into the document
- Example:
  - `document.getElementById('foo').innerHTML="<p>XYZ</p>"`

JavaScript

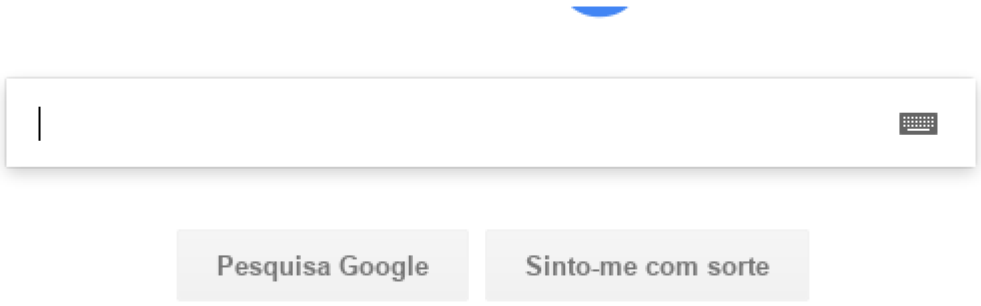
- A scripting language for browsers
  - Can make pages interactive w/o sending requests to server
  - Syntax is very similar to Java, but internals are very different
  - Lots of small differences between browser implementations
- JavaScript can interact with the DOM
  - Examples: Read data from forms, replace parts of the page
  - Can register event handlers with DOM elements to respond to clicks, keypresses, mouse movements, selections...

Resources

- http://www.w3schools.com/js/
  - Covers the basics (use the navbar on the left!)\
- http://net.tutsplus.com/tutorials/javascript-ajax/introduction-to-express/
- https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
- JQuery tutorial
  - http://www.w3schools.com/jquery/jquery_intro.asp
  - http://jqfundamentals.com/

Implementing search suggestions

- How would you do this with pure JavaScript?



XMLHttpRequest

- A JavaScript object that enables web pages to dynamically load more content
  - Example: Ask the server for search suggestions while the user is typing the search term
- Request can be asynchronous
  - Browser performs the HTTP request in the background while the user continues to interact with the web page
  - Script defines a callback function that should be invoked when the requested content has arrived
- Content does not have to be XML
  - But often is or JSON

XMLHttpRequest workflow

- Instantiate a new XMLHttpRequest object
- Prepare the object
  - Call open() to set the URL and the method (GET, POST, ...)
  - Can add headers, HTTP authentication, ...
  - Need to send a callback function that will be called by the browser when the results are available
- Send the request
  - Invoke send(), optionally with data to submit (for POST)
- Handle invocations of the callback function
  - Do something with the response if request was successful
  - Optionally, handle errors

XMLHttpRequest properties

Property	Description
readyState	Current state of the object: 0 = UNSENT (before open() is called) 1 = OPENED (before send() is called) 2 = HEADERS_RECEIVED (header+status available) 3 = LOADING (partial data available) 4 = DONE (operation complete)
onreadystatechange	Can be assigned a function that is called whenever readyState changes (e.g., to process the response)
responseText	Response as text
responseXML	Response as a DOM document object (parsed as text/xml)
status	Status of the response (a HTTP result code, e.g. 200)
statusText	Response string returned by the server (e.g., '200 OK')

XMLHttpRequest methods

Method	Description
open("method","url" [,asyncFlag[,user[,pwd]]])	Initializes a new request. Default is asynchronous (asyncFlag=true). Optional HTTP user+password.
setRequestHeader("L","V")	Used to add headers to the request
send([content])	Sends the request. Content is optional (omitted, e.g., for GET)
abort()	Aborts a request that has already been sent
getResponseHeader("L")	Returns a specific header from the response
getAllResponseHeaders()	Returns all the headers from the response

Security restrictions

- Requests are subject to same-origin policy
  - Can only connect to the domain that also sent the currently loaded page
    - Example: Page from foo.com can't request content from bar.com
  - Similar to the restriction that applies to cookies
- Some workarounds exist
  - JSONP: Encode data in a JavaScript
  - CORS: Additional HTTP header allows access from different domains; part of XMLHttpRequest Level 2
  - Plugins (Flash, Silverlight, ...)

Instantiating XMLHttpRequest

- Implementation in browsers varies:
  - var request = new XMLHttpRequest(); (most browsers)
  - var request = new ActiveXObject("Microsoft.XMLHTTP");
  - var request = new ActiveXObject("Msxml2.XMLHTTP");
  - Using incorrect method causes an exception!

- When doing a POST, set the content type
  - `request.setRequestHeader('Content-Type',`
  - `      'application/x-www-form-urlencoded');`
  - `request.send('param1='+param1+'&param2='+param2);`
- When doing a GET, encode parameters
  - Not all characters are legal in a URL (example: space)
  - Need to escape these characters (' ' '%20' etc.)
  - Can use the escape() method to do this:
    - request.send('param1='+escape(param1));

XMLHttpRequest

- XMLHttpRequest is a JavaScript object
  - Enables web pages to dynamically request more data
  - For security reasons, same-origin policy applies
  - Despite the name, data does not have to be in XML; other formats (text, HTML) work as well
- Requests can be asynchronous
  - Programmer supplies a callback function that is invoked by the browser when a response arrives
  - Can load data in the background (example: Google Maps)

What is AJAX?

- Asynchronous JavaScript and XML
  - Firsty mentioned by Jesse James Garrett in 2005

- Not a single technology - a mix of technologies for building faster web apps
  - HTML and CSS for presentation
  - DOM for dynamic display
  - XML for data interchange
  - XMLHttpRequest for asynchronous requests
  - JavaScript for binding everything together

Where is AJAX used?

- Widely used by major web pages today
  - Examples: Google Maps, Gmail, Search Suggestions, ...

Building web applications with AJAX

- Several puzzle pieces are needed
  - Host page: A web page that we'd like to make interactive
  - Client-side script: A JavaScript program that
    - registers handlers for relevant events, such as inputs or mouse clicks
    - requests additional data from the server using XMLHttpRequest objects
    - integrates the responses with the web page using the DOM
  - Server side: Another JavaScript program that supplies the data

AJAX with XML

- Despite its name, XMLHttpRequest can handle content other than XML
  - Server must set Content-Type header appropriately: text/xml for XML; text/plain or text/html otherwise
  - XML content can be accessed through responseXML field
  - DOM has the same methods as the HTML DOM

- Upload data to the server as XML?
  - Possible, but a lot of work (server may not have a DOM)

Example: Server side

Example output:

```
<?xml version="1.0" ?>
<root>
  <element>Anaconda</element>
  <element>Ant</element>
  <element>Anteater</element>
  <element>Antelope</element>
</root>
```

```
var express = require('express');
var app = express();

app.use(express.bodyParser());
app.use('/', express.static(__dirname + "/public",{maxAge:1}));

var terms = ["Aardvark", "Adelie penguin", "Alligator", "Alpaca", "Anaconda",
  "Ant", "Anteater", "Antelope", "Ape", "Arctic seal", "Armadillo",
  "Ass", "Axolotl" ];

app.get('/suggest/:term', function(req, res) {
  console.log('Requested term: ' + req.params.term);
  res.type('text/xml');
  var response = "<?xml version=\"1.0\"?>\n<root>\n";
  terms.forEach(function(t) {
    if (t.substring(0, req.params.term.length) == req.params.term)
      response = response + "  <element>"+t+"</element>\n";
  });
  response = response + "</root>";
  res.send(response);
});

app.listen(8080);
console.log('Server running on port 8080');
```

Example: Client side

```
<html>
  <head><title>Test</title></head>
  <script type="text/javascript">
  <!--
    function updateSuggestions() {
      var term = document.getElementById('abc').value;
      request = new XMLHttpRequest();
      request.open("GET", "http://localhost:8080/suggest/"+escape(term));
      request.onreadystatechange = function() {
        if ((request.readyState == 4) && (request.status == 200)) {
          var xmldoc = request.responseXML;
          var root = xmldoc.getElementsByTagName('root').item(0);
          var elements = root.getElementsByTagName('element');
          var htmlOut = (elements.length)+ " suggestion(s):<br><br>";
          for (var i=0; i<elements.length; i++)
            htmlOut += "#"+(1+i)+": "+elements.item(i).textContent+"<br>";
          document.getElementById('xyz').innerHTML = htmlOut;
        }
      }
      request.send();
    } // -->
  </script>
  <body>
    <h1>Input a search term</h1>
    <form action="" method="" onSubmit="return false">
      <input type="text" name="thetext" size="20" id="abc" onKeyUp="updateSuggestions()">
      <input type="submit" value="Replace">
    </form>
    <div id="xyz">(this is where the text will go)</div>
  </body>
</html>
```

URL of server-side component (servlet)

Callback function

Get data from XML

Put data into page via DOM

Registers event handler

Some common problems

- Problem: Request is never sent
  - Are you following the same-origin policy?
  - To debug, use try-catch block (as in java) and call alert() to display the exception in a dialog box, or open the browser's 'error console' window
- Problem: Response is never received
  - Did you use the XMLHttpRequest object to issue another request in the meantime? (causes response to be lost)
  - Solution: Create multiple request objects
- Problem: Request is only sent once

Pros and cons of AJAX

- Much more responsive than plain HTML
  - Can avoid wide-area latency in many cases (why not all?)
  - Faster - can transfer just the required information after each interaction, rather than the entire page (+less bandwidth)
- Difficult to integrate navigation elements
  - 'Back' button, bookmarks, external links from other pages etc. require special care (window.location.hash)
- Difficult to accommodate search engines

Recap: AJAX

- A mix of several technologies
  - "Asynchronous JavaScript and XML"
- Can be used to build interactive web pages
  - HTML and CSS for rendering the host page
  - JavaScript event handlers for responding to inputs
  - XMLHttpRequest object for getting more data from server
  - XML for encoding the responses
  - DOM for integrating data with the host page

Goals for today

NEXT

- Brief introduction to JavaScript
  - Event handlers
  - Accessing the DOM
  - The XMLHttpRequest object
- AJAX
  - Putting everything together
  - Example: Search suggestions
- Simplifying things
  - jQuery support for AJAX
  - JSON

AJAX troubles

- "I don't want to write all that code for packing and unpacking XML"
- "I don't want to deal with XMLHttpRequest objects directly"
- "There are still browser incompatibilities?!?"

jQuery support for AJAX

http://www.w3schools.com/jquery/jquery_ajax_load.asp

- jQuery makes AJAX much more convenient
  - Transparently handles browser incompatibilities
  - Comes with convenience methods like
    - $("#someid").load(...)
  - No need to deal with XMLHttpRequest directly

```
<html><head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script><script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").load("demo_test.txt",function(responseTxt,statusTxt,xhr){
      if (statusTxt=="success")
        alert("External content loaded successfully!");
      if (statusTxt=="error")
        alert("Error: "+xhr.status+": "+xhr.statusText);
    });
  });
});
</script></head><body>
<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>
<button>Get External Content</button>
</body></html>
```

Problem: Sending structured data

- What if we want the server to return an object, or an array, or ...?
- Use JSON!

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumber": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ]
}
```

Working with JSON

- On the client side:
  - Invoke $.getJSON('/url', function(data)) instead of $.get
  - When the server returns JSON-encoded data, it is parsed (via eval()) and returned as an object
  - Warning: Security implications!

- On the server side:
  - Step 1: In the route callback, build the object to return
  - Step 2: Send with

AJAX with jQuery/JSON: Server side

```
var express = require('express');
var app = express();

app.use(express.bodyParser());
app.use('/', express.static(__dirname + "/public",{maxAge:1}));

var terms = ["Aardvark", "Adelie penguin", "Alligator", "Alpaca", "Anaconda",
  "Ant", "Anteater", "Antelope", "Ape", "Arctic seal", "Armadillo",
  "Ass", "Axolotl" ];

app.get('/suggest/:term', function(req, res) {
  console.log('requested term: ' + req.params.term);
  var response = [];
  terms.forEach(function(t) {
    if (t.substring(0, req.params.term.length) == req.params.term)
      response.push(t);
  });
  res.send(JSON.stringify(response));
});

app.listen(8080);
console.log('Server running on port 8080');
```

AJAX with jQuery/JSON: Client side

```
<html>
  <head><title>Test</title></head>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
  <script type="text/javascript">
  <!--
    function updateSugg() {
      var term = document.getElementById('abc').value;
      $.getJSON('http://localhost:8080/suggest/'+escape(term), function(elements) {
        var htmlOut = (elements.length)+ " suggestion(s):<p><table border=\"1\">\n";
        for (var i=0; i<elements.length; i++)
          htmlOut += "<tr><td>#"+(1+i)+"</td><td>"+elements[i]+"</td></tr>\n";
        htmlOut += "</table>\n";
        $("#xyz").html(htmlOut);
      });
    } // -->
  </script>
  <body>
    <h1>Input a search term</h1>
    <form action="" method="" onSubmit="return false">
      <input type="text" name="thetext" size="20" id="abc" onKeyUp="updateSugg()">
      <input type="submit" value="Replace">
    </form>
    <div id="xyz">(this is where the text will go)</div>
  </body>
</html>
```

Refreshing data

- What if something on your page should refresh periodically?
  - Example: Chat server; local window should display new chat messages soon after remote users have posted them
- Solution: Timeouts
  -

```
<html>
  <head><title>Timeout</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js">
  </script><script type="text/javascript">
  var refreshTime = function() {
    $("#clock").html((new Date()).toString());
    setTimeout(refreshTime, 1000); /* 1000 ms */
  };
  $(document).ready(function() {
    setTimeout(refreshTime, 1000); /* 1000 ms */
  });
  </script>
</head><body><div id="clock">(Time goes here)</div></body></html>
```