# AJAX Basics

Using an AJAX “callback”

Sending data to the server

Manipulating the DOM

# Why Ajax?

HTML and HTTP are weak

- Non-interactive
- Coarse-grained updates

Everyone wants to use a browser, even on your device

- Not a custom application

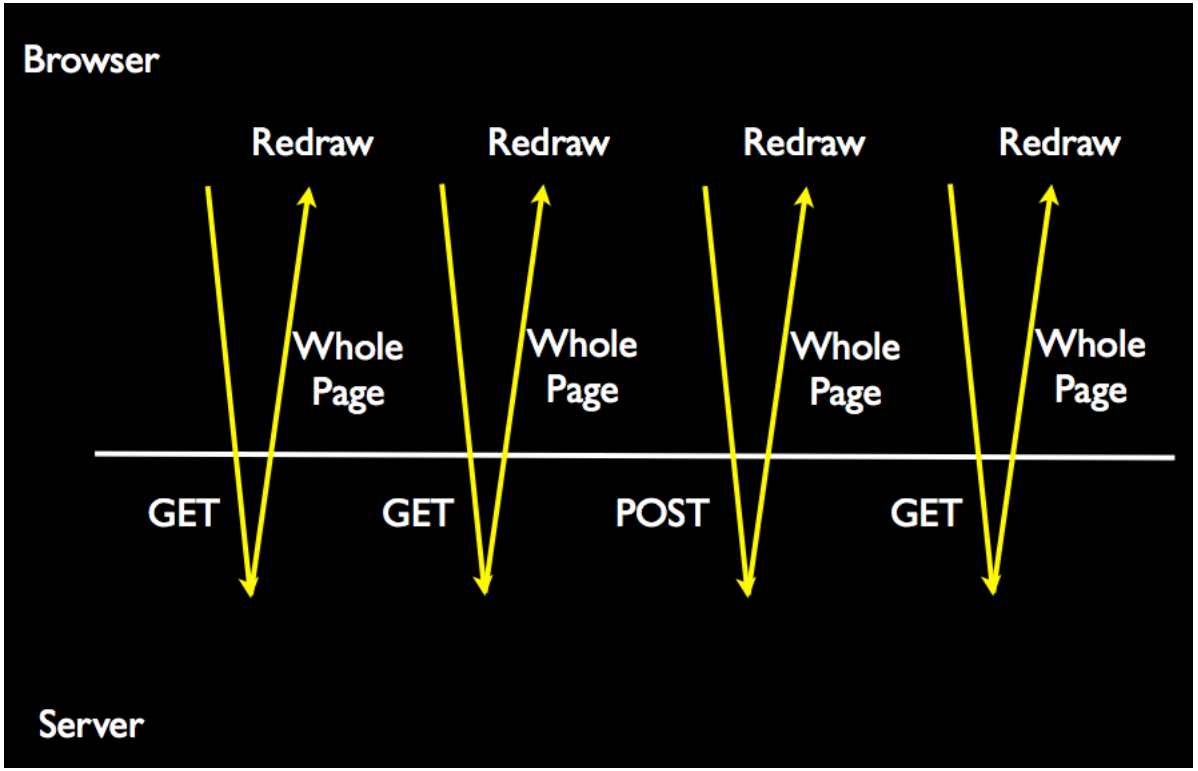
"Real" browser-based active content

- Failed: Java Applets
  - Not universally supported; can't interact with the HTML
- Serious alternative: Flash (and Flex)
  - This worked for a long time – rich content and page interaction
  - Various hurdles – Apple, and now HTML5
- Plenty of others tried with varying degrees of success
  - Microsoft Silverlight
  - JavaFX
  - Adobe Apollo

# AJAX Motivation

## In The Good Old Days

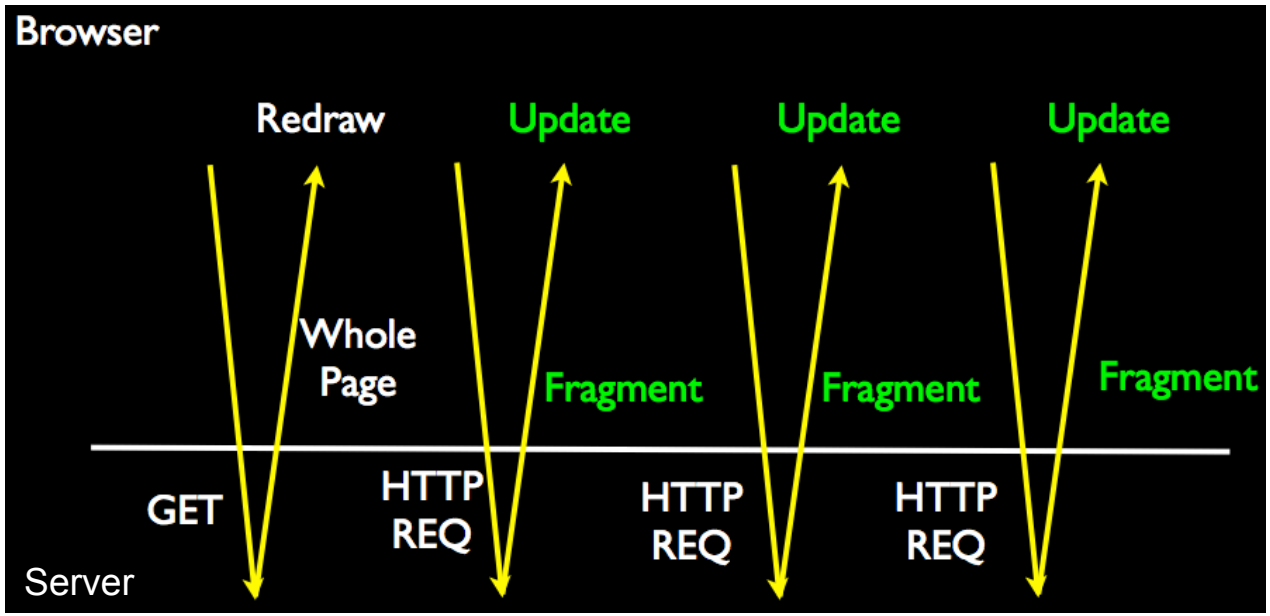
- A user would take some action like a click on a link or button
- The Browser would make a TCP/IP connection to the web server
- The browser would send a POST or GET request
- The Server would send back a page to display to the user
- Repeat the Request-Response Cycle...



# XMLHttpRequest

By 1999, Microsoft wanted to move some of the processing of web pages from the web server to the web browser

- The idea was instead of sending whole pages of HTML to the browser, send out the data to be displayed as XML and then produce presentation in JavaScript in the browser
- Originally a Microsoft innovation - other browsers soon adopted the idea and it became a defacto standard with a little variation between browsers
- It soon became clear that this could send *anything* - not just XML back and forth between a browser and client



# AJAX Motivation Summary

Standard Request/Response

- Each click presents a whole new screen

AJAX

- Each action sends data and receives results in the background.
- The browser typically gets back a fragment of HTML, XML, or JSON which is used to update a portion of the screen using the browser document model

What we will look at is the ability to consume already existing web services using AJAX, and use the returned content to manipulate our applications.

Then we will map this capability into the MVC and MVVM design patterns in the context of a framework (*stay tuned…*)

# The Basic Ajax Process

JavaScript

- Define an object for sending HTTP requests
- Initiate request
  - Get request object
  - Designate an anonymous response handler function
    - Supply as onreadystatechange attribute of request
  - Initiate a GET [and later, a POST or PUT] request
- Handle response
  - Wait for readyState of 4 – the response has been received
  - And an HTTP status of 200 – what does this mean again?
  - Extract return text with responseText or responseXML
  - Do something with result (like, say, manipulate a DOM?)

HTML

- Load JavaScript
- Designate control that initiates request
- Give ids to input elements and to output placeholder region

# Define/Initiate a Request Object

```
function getRequestObject() {
   if (window.XMLHttpRequest) {
     return(new XMLHttpRequest());
   } else {
     return(null);
   }
}

function sendRequest() {
   var request = getRequestObject();
   request.onreadystatechange =
     function() { handleResponse(request) };
   request.open("GET", "message-data.html", true);
   request.send(null);
}

function handleResponse(request) {
   if (request.readyState == 4) {
     alert(request.responseText);
   }
}
```

**Show-message.js**

Code to call when server responds

Don't wait for response
(Send request asynchronously)

Response from server is complete
(handler gets invoked multiple times)

# HTML Code (show-message.html)

```
<html>
<head><title>Ajax: Simple Message</title>
<script src="show-message.js"
        type="text/javascript"></script>
</head>
<body>
<center>
<table border="1" bgcolor="gray">
  <tr><th><big>Ajax: Simple Message</big></th></tr>
</table>
<p/>
<form action="#">
 <input type="button" value="Show Message"
        onclick="sendRequest()"/>
</form>
</center>
</body>
</html>
```

# AJAX-based solutions and DOM manipulation

Conceptually, we have now extended our DOM-as-UI-Model to include our server

- AJAX can return any object we want in any form – it is just an embedded HTTP request!
- This implies the world model is at our fingertips.
- The most common thing to do is grab a projection of the world model and inject it into our DOM model, thereby updating the page

Options:

- Solution 1: use HTML to update page with result
  - Have server return an HTML fragment to the page
  - Use DOM manipulation to update the page
- Solution 2: return non-HTML content to the page
  - Yep, XML or JSON
  - Parse it, and do your DOM manipulation

Previous example did an alert, so it didn't consider this issue!

# Dynamically Inserting HTML

HTML

- 

See example in ajax-basics-1.html

JavaScript

- resultRegion = document.getElementById("results-placeholder");
- resultRegion.innerHTML = "<h2>Wow!</h2>";  // returned
  - For the innerHTML text, you usually use request.responseText or some string based on request.responseText

Result after running code

- <h2>Wow!</h2>
  - "View source" won't show this, but dev tools should.

Deficiences

- Page author has no control over format
- Cannot use the same data for different tasks – tight coupling – what happens if the outer page constructs change?
- Having server-side resource generate HTML is often easier and better. But not always.

# HTML Example: Design Deficiencies

Solution #2 server returns XML or JSON content

- JavaScript parses XML/JSON and decides what to do with it
- Steps from before stay the same, just return data to parse

Getting the main XML/JSON document

- If XML, use responseXML instead of responseText

```
var xmlDocument = request.responseXML;
```

- If JSON, stay with responseText but then parse the result

```
Var jsonObj = JSON.parse(request.responseText);
```

XML originally the way to go (hence XMLHttpRequest) but now JSON is considered the best practice

- The use of parse and the lightweight format (we don't need elaborate structure and validation here) make it a viable alternative to XML.
- Besides it is JavaScript Object Notation, and you are in JS!

# Steps for Solution #2

See example in ajax-basics-1c.html

## JavaScript

- Define an object for sending HTTP requests
- Initiate request
  - Get request object
  - Designate an anonymous response handler function
  - Initiate a GET [or later POST] request to the server
- Handle response
  - Wait for readyState of 4 and HTTP status of 200
  - Extract return text with responseText or responseXML
    - *Pass string to "parse" to get a real JavaScript object*
    - *Access fields, array elements, etc., with normal JavaScript syntax*
  - Use innerHTML to insert result into designated element

## HTML

- Load JavaScript from centralized directory
- Designate control that initiates request
- Give ids to input elements
- Define a blank placeholder element with a known id

# Request lifecycle, Error-handling, & Summary

## What is magical about onreadystate == 4?

- The request has a lifecycle indicated by state codes:
  - 0 UNSENT – request constructed but not yet sent
  - 1 OPENED – open() has been called so request is constructed, but one could still set headers and call send()
  - 2 HEADERS RECEIVED – redirects processed & all response headers received
  - 3 LOADING – response body is being received
  - 4 DONE – response processing completed (successfully or not)

## What if I don't get a 200?

- You will still get DONE (4)
- You have to decide what to do – if 4xx fix your request or wait until the state clears, if 5xx perhaps do at a later point?

## Summary: MVC architecture on server moves toward client

- Server retains the World Model
- But a projection of the model goes to the client to back the UI
- Presentation now mostly on client