Remaining Weeks

- This week (July 23)
  - AJAX

- Next week (July 30)
  - guest lecture
    - Nilton Bila on "PageTailor"
    - Related to some of your final exam questions
  - 2 office hours (both running 5pm-6pm)
    - Ali (BA3289): A2 re-marking
    - Torsten (BA5287): A4-related questions

- Last week (Aug 6)
  - A4 due (no extension; zero tolerance)
  - Security & course review

There are still spaces available in Prof. Greg Wilson's

CSC491: Capstone Project
Fall 2008
Real projects!
Real clients!
Class held in a pub!
Mail gwilson@cs.toronto.edu for details
or see
www.cs.toronto.edu/~gwilson/capstone.pdf
for last term's projects
(this term's will be even cooler)

AJAX (Asynchronous JavaScript And XML)

Nan Niu (nn@cs.toronto.edu)
CSC309 -- Summer 2008

RIA (Rich Internet Application)

- Bridge gap between native application and normal Internet ones
- More interactive Web applications
- Perform similar to native application
- Enabling technologies: JavaScript, DHTML (DOM), AJAX
- Shift from page-based interaction to event-driven programming where event result in updates of portions of pages

Traditional Web vs. AJAX

- Traditional Web
  - User initiated HTTP requests
    - Typing on navigation window, or clicking on form
    - Response from server overrides existing page
    - Low request rate, and random amount of time between requests
- AJAX application
  - New type of request that does not trigger page reload
  - Not initiated by user
  - Requests are typically small, but more frequent

AJAX

- 2nd communication path b/w browser and server
  - Remove communication bottleneck b/w user and Web application
  - Talk to server from JavaScript
  - Skip page reload

- Changes the typical page flow
  - More frequent requests
  - Smaller responses of non-HTML data

- Defined by Jesse James Garret in Feb. of 2005
  - Underlying technology in place since 2001
  - Google releases Gmail in March of 2004
  - One of first mainstream apps to use AJAX
  - Examples: read/tag/spell check messages without a page reload, auto save drafts
**Request Flow**

**Web Application**

**AJAX Application**

---

**Classic vs. AJAX**

**Taxonomy**

- **Core AJAX**
  - Asynchronous communication (XMLHttpRequest)
  - Data encoding (XML)
- Closely related to AJAX
  - Interactive part of UI
  - Visual effects libraries (fades, swipes)
  - Drag and drop
  - Dynamic positioning
- Part of the rest of RIA
  - Scalable Vector Graphics (SVG)

---

**Synchronous vs. Asynchronous**

**XMLHttpRequest**

- JavaScript HTTP client
  - Can load content other than XML
- Issue GET and POST requests
- Security limitations apply
  - Same point of origin: Can only connect to same domain as currently loaded page
  - Only load HTTP requests
  - File URLs don't work

---

**XMLHttpRequest Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send()</td>
<td>Sends the current request.</td>
</tr>
<tr>
<td>open(method, url, async)</td>
<td>Specifies the method, URL, and other optional attributes of a request.</td>
</tr>
<tr>
<td></td>
<td>The method parameter can have a value of &quot;GET&quot;, &quot;POST&quot;, &quot;HEAD&quot;, &quot;PUT&quot;, &quot;DELETE&quot; or a variety of other HTTP methods based on the XMLHttpRequest specification.</td>
</tr>
<tr>
<td></td>
<td>The URL parameter may be either a relative or absolute URL.</td>
</tr>
<tr>
<td></td>
<td>The &quot;async&quot; parameter specifies whether the request should be handled asynchronously or not. &quot;true&quot; means that script processing can continue after the request has been sent. &quot;false&quot; means that the script waits for a response before continuing with script processing.</td>
</tr>
<tr>
<td></td>
<td>Sends the request.</td>
</tr>
<tr>
<td>setRequestHeader(label, value)</td>
<td>Adds a header to the HTTP request to be sent.</td>
</tr>
</tbody>
</table>
XMLHttpRequest Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onreadystatechange</td>
<td>Defines a function to retrieve data returned by the server after a request is sent</td>
</tr>
<tr>
<td>readyState</td>
<td>Returns the state of the request as a number: 0 = not initialized, 1 = open, 2 = sent, 3 = in process, 4 = complete</td>
</tr>
<tr>
<td>open</td>
<td>Indicates the request has been set up</td>
</tr>
<tr>
<td>sent</td>
<td>Indicates the request has been sent</td>
</tr>
<tr>
<td>inprocess</td>
<td>Indicates the request is in process</td>
</tr>
<tr>
<td>complete</td>
<td>Indicates the request is complete</td>
</tr>
</tbody>
</table>

readyState Property

- This property holds the status of the server's response
- Each time the readyState changes, the onreadystatechange function will be executed
- State description
  - 0: The request is not initialized
  - 1: The request has been set up
  - 2: The request has been sent
  - 3: The request is in process
  - 4: The request is complete

onreadystatechange Property

- Defines a function to retrieve data returned by the server after a request is sent
- Must be set before sending request
- The following code defines a function for this purpose (with an empty body for now):

```javascript
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function()
{
    // code for receiving response data
}
```

responseText Property

- This property retrieves the response body returned by the server as a string
- Type: DOMString (read-only)

```javascript
xmlhttp.onreadystatechange = function()
{
    if(xmlhttp.readyState==4)
    {
        document.getElementById("formentry").value = xmlhttp.responseText;
    }
}
```

responseXML Property

- Retrieve document data returned by the server as an XML DOM object
- Type: Document (read-only)
- You can access it as a DOM document

```javascript
var xmlIdoc = xmlhttp.responseXML.documentElement;
```
XMLHttpRequest Methods

- Asking for data (send request)
  - open()
    - Two required arguments
      - method (GET, POST, PUT, DELETE, HEAD, OPTION)
      - server-side URI
    - send()
      - One argument
        - data to be sent (DOMString or Document)
        - null for GET
        - can be omitted

Methods (Verbs)

- GET
  - Retrieve a representation of a resource
- POST
  - Create a new resource
- HEAD
  - Request a resource without body
- PUT
  - Update a resource
- DELETE
  - Delete a resource

Execute the AJAX Function

- Want it to run "behind the scenes"
  `<script type="text/javascript">
    function myajax()
    {
      /* all of the code from before */
    }
  </script>
  <form>
    <input type="text" onkeyup="myajax();" name="userdata" />
    <input type="text" id="formentry" />
  </form>`

The Basic AJAX Process

- JavaScript
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate a request handler function
      - Supply as onreadystatechange attribute of request
    - Initiate a GET or POST request
    - Send data
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
    - Do something with the result

The Basic AJAX Process (Cont'd)

- HTML
  - Loads JavaScript
  - Designates control that initiates request
  - Gives ids to input elements that will be read by script

XMLHttpRequest Events

- Firefox resets handlers after request completes
send(payload)

- Makes connection to the URL specified in open
- Blocks for synchronous requests, and returns immediately otherwise.
- Sends cookies and other default headers

- POST request:
  - payload is send as body of HTTP message
  - Need to set Content-type header

- Example: Mimic sending a form

```javascript
var req = new XMLHttpRequest();
req.open('POST', 'index.php', false);
req.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
req.send('one=firstArg&two=secondArg');
if (req.status==200) {
  alert(req.responseText);
}
```

Asynchronous Request

- Happens in the background
  - Does not block browser
  - User can continue interaction with page
  - Can mask long latency/low bandwidth connections

```javascript
var req=XMLHttpRequest();
req.onreadystatechange = function () {
  if (req.readyState==4) {
    if (req.status==200) {
      alert(req.responseText);
    } else {
      alert('Loading Error: ['+req.status+'] '+req.statusText);
    }
  }
};
req.open('GET', '/csc309/servlet/ajaxserver_true');
req.send(null);
```

Add HTML to a Page

```html
<!-- Add dynamically generated HTML to the page -->
```

IE AJAX

- XMLHttpRequest is an ActiveX object

```javascript
var xmlhttp = new ActiveXObject('MSXML2.XMLHTTP.5.0');
```

Multiple versions
- Microsoft.XMLHTTP
  - MSXML2.XMLHTTP
    - MSXML2.XMLHTTP 3.0
    - MSXML2.XMLHTTP 4.0
    - MSXML2.XMLHTTP 5.0

Cross-Browser XMLHttpRequest

```javascript
function XMLHttpRequest() {
  try {
    // Mozilla / Safari / Firefox
    xmlhttp = new XMLHttpRequest();
  } catch (e) {
    //IE
    var XMLHttpRequest = new ActiveXObject('MSXML2.XMLHTTP.5.0');
  }

  var success = false;
  for (var i=0; i < XMLHttpRequest.length && !success; i++) {
    try {
      xmlhttp = new ActiveXObject(XMLHTTP_IDS[i]);
      success = true;
    } catch (e) {}
  }
  if (!success) {
    throw new Error('Unable to create XMLHttpRequest.');
  }
}
return xmlhttp;
```