Lecture 11
Asynchronous JavaScript and XML (AJAX)

Outline

- *Rich Internet Applications (RIAs)*
- Ajax
- XMLHttpRequest
- Limits of Ajax
- Tips

From Web Sites to Web Applications

- Client/Server
  - static content
- Basic Three-Tier
  - user interface, business or process logic, and a data access module
- Model-View-Controller
- Rich Internet Applications

Web Applications Revisited

- How are web applications different from traditional desktop applications?

How "Traditional" Web Applications Fair?

- Interface – HTML + CSS + JavaScript + Images
  - Not rich enough
    - No video playing capabilities (need plug-in)
    - No support for vector graphics (no pixel-level manipulation)
  - Not "natural" for creating dynamic GUI components like collapsible tree or for creating behavior like drag-and-drop
  - Less responsiveness (JavaScript is interpreted)

How "Traditional" Web Applications Fair?

- Thin Client Architecture
  - Servers do most of the work and keep most data → higher traffic volumes
- Synchronous communication
  - For each request sent as a result of some actions, the client cannot interact with the application until the server completes the response.
  - i.e., less interactive user experience

Rich Internet Applications

An RIA implementation on top of MVC

RIA: Pros (In General)

- Installation is not required
- Easy to upgrade
- Easily made available over internet/intranet
- Richer UI
- More responsive UI
- Client/Server Balance
- Asynchronous communication
- Network efficiency

RIA: Cons (In General)

- Loss of visibility to search engines
- Proprietary (as opposed to Open Standard)
- Loss of integrity (RIAs typically don't mix well with HTML)
- Software development complications (What to cache or not to cache at client's computer?)
- RIA architecture breaks the Web page paradigm

RIA Approaches

- Browser plug-in
  - Flash/Flex, Java Swing, Silverlight
  - Potentially greater interactivity, higher barrier to adoption
  - Concerns about openness/control
- In browser, no plug-ins
  - AJAX
  - Lower barrier to adoption
  - Cross-browser mayhem?

Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Limits of Ajax
- Tips

Synchronous web communication

- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)
- almost all changes with new data lead to page refresh

Asynchronous web communication

- asynchronous: user can keep interacting with page while data loads
  - communication pattern made possible by Ajax
- Changing with new data but without page refresh

What is AJAX?

browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend processing, legacy systems
server-side systems
classic web application model
Jesse James Garrett / adaptivepath.com
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend processing, legacy systems
server-side systems
Ajax web application model

Web applications and Ajax

- web application: a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9
- Ajax: Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page
  - avoids the "click-wait-refresh" pattern
  - examples: Google Suggest

AJAX event handling

Advantages and Disadvantages of AJAX

Advantages

- Better interactivity and responsiveness.
- Your page will be more pleasant to use.
- Reduced connections to the Web server because of partial rendering.
- Because you only load the data you need to update the page, instead of refreshing the entire page, you save bandwidth.
- It helps in reducing the network traffic.

Disadvantages

- The back and refresh buttons become useless.
- Bookmarking this page will become useless.
- Requires JavaScript to be enabled on the Web browser.
- Network latency may break usability.
- Data loaded through AJAX won't be indexed by any of the major search engines. Hence, making it SEO unfriendly.

Real-Life Scenarios for Using AJAX

- The following list depicts some real-life scenarios where AJAX can be helpful:
  - Autocomplete search text boxes
  - Cascading dropdown list box
  - Real-time communication, such as instant messaging
  - Real-time data updates, such as score updates
  - Immediate forms validation feedback
  - Auto save user information

Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Limits of Ajax
- Tips

Components of AJAX

- The AJAX cannot work independently.
- It is used in combination with other technologies to create interactive Web pages that are described in the following list:
  - JavaScript:
    - Loosely typed scripting language.
    - JavaScript function is called when an event occurs in a page.
    - Glue for the whole AJAX operation.
  - DOM:
    - API for accessing and manipulating structured documents.
    - Represents the structure of XML and HTML documents.
  - CSS:
    - Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.
  - XMLHttpRequest:
    - JavaScript object that performs asynchronous interaction with the server.

XMLHttpRequest Object

- The XMLHttpRequest object:
  - Is the most important component of AJAX.
  - Has been available ever since Internet Explorer 5.5 was released in July 2000.
  - Is an API that can be used by JavaScript, JScript, VBScript, and other Web browser scripting languages.
  - Is used to transfer and manipulate XML data to and from a Web server using HTTP.
  - Establishes an independent connection channel between a Web page's client-side and server-side.
  - Besides XML, XMLHttpRequest can be used to fetch data in other formats, such as JSON or even plain text.
  - Performs following operations:
    - Sends data from the client in the background.
    - Receives the data from the server.
    - Updates the webpage without reloading it.

XMLHttpRequest Object (Contd.)

- The following figure depicts the process cycle of AJAX Web application model:

XMLHttpRequest Object (Contd.)

- The following table describes the common properties of the `XMLHttpRequest` object.

Property	*Description*
`onReadyStateChange`	*Is called whenever readyState attribute changes. It must not be used with synchronous requests.*
`readyState`	*Represents the state of the request. It ranges from 0 to 4, as described in the following list:* • ***0**: UNOPENED, open() is not called.* • ***1**: OPENED, open is called but send() is not called.* • ***2**: HEADERS_RECEIVED, send() is called, and headers and status are available.* • ***3**: LOADING, downloading data; responseText holds the data.* • ***4**: DONE, the operation is completed fully.*
`reponseText`	*Returns response as text.*
`responseXML`	*Returns response as XML.*

XMLHttpRequest Object (Contd.)

- The following table describes the essential methods of the `XMLHttpRequest` object.

Method	*Description*
`void open(method, URL)`	*Opens the request specifying get or post method and URL of the requested Web page.*
`void open(method, URL, async)`	*same as above but specifies asynchronous or not.*
`void open(method, URL, async, username, password)`	*same as above but specifies username and password.*
`void send()`	*sends get request.*
`void send(string)`	*send post request.*
`setRequestHeader(header,value)`	*it adds request headers.*

XMLHttpRequest Object (Contd.)

- The following figure depicts the syntax of creating an `XMLHttpRequest` object for modern Web browsers:

```
Variable = new XMLHttpRequest();
```

- The following figure depicts the syntax of creating an `XMLHttpRequest` object for old versions of Microsoft IE, that is, IE5 and IE6:

```
Variable = new ActiveXObject("Microsoft.XMLHTTP");
```

XMLHttpRequest Object (Contd.)

- To handle all modern Web browsers, including Microsoft IE5 and IE6, you need to check whether the Web browser supports the `XMLHttpRequest` object.
- The following code depicts how to create an `XMLHttpRequest` object for all the modern Web browsers, including Microsoft IE5 and IE6:

```
var xmlhttp;
if (window.XMLHttpRequest)
  {
 //code for IE7+,Firefox,Chrome,Opera and Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {
  // code for IE6, IE5
  xmlhttp=new  ActiveXObject("Microsoft.XMLHTTP");
  }
```

XMLHttpRequest Object (Contd.)

- After creating an `XMLHttpRequest` object, you need to decide what you want to do after you receive the server response to your request.
- At this step, you need to define the JavaScript function, which will handle the server response.
- This can be done using the `onreadystatechange` property of the `XMLHttpRequest` object, as shown in the following code snippet:

```
xmlhttp.onreadystatechange = function(){
        //process the server response
};
```

XMLHttpRequest Object (Contd.)

- After setting the response function, you need to make the request.
- To make the request, you need to call the open()and send()methods of the XMLHttpRequest object, as shown in the following code snippet:

```
xmlhttp.open('GET','serverpage.php',true);
xmlhttp.send(null);
```

- The following list describes the parameter passed to open()method in the preceding code snippet:
  - The first parameter is the HTTP request method, such as GET, POST, and HEAD.
  - The second parameter is the URL of the Web page that you are requesting.
  - The third parameter, optional, sets whether the request is asynchronous.

XMLHttpRequest Object (Contd.)

- The following figure depicts some samples of using HTTP GET and POST request methods with AJAX:

```
//HTTP GET request without querystring
xmlhttp.open('GET','serverpage.php',true);
xmlhttp.send(null);

//HTTP GET request with querystring
xmlhttp.open('GET','serverpage.php?
username='+Math.random(),true);
xmlhttp.send(null);

//HTTP GET request without querystring
xmlhttp.open('GET','serverpage.php?
username=user1&pass=password",true);
xmlhttp.send(null);

//HTTP POST request
xmlhttp.open('POST','serverpage.php',true);
xmlhttp.send('username=user1&pass=password');
```

XMLHttpRequest Object (Contd.)

- To handle the server's response:
  - First, the response function needs to check the ready state of the request.
    - If the ready state has the value of 4, then you can proceed further.
  - Next, you need to check the response code of the HTTP server response.
- The following code snippet depicts how to handle the server's response:

```
xmlhttp.onreadystatechange = function(){
if (xmlhttp.readyState === 4) {
  // everything is good, the response is received
  if (xmlhttp.status === 200) { // process the response }
  else { // request encountered some problem,
         // for example, the response may contain a HTTP
404 (Not Found) response code
       }
} else { // still not ready } };
```

Fetching Data using XMLHttpRequest Object (Contd.)

- The following code snippet depicts how to display response from Web server on a Web page:

AJAXExample.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script>
function loadXMLDoc()
{
  var xmlhttp=false;
  if (window.XMLHttpRequest)
  {
    xmlhttp=new XMLHttpRequest();
  }
  else
  {
    xmlhttp=new ActiveXObject   ( “Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function()
  {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
       document.getElementById(“myDiv").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","/try/ajax/ajax_info.txt",true);
  xmlhttp.send();
}
</script>
</head>
<body>

<div id="myDiv"><h2>使用 AJAX 修改该文本内容</h2></div>
<button type="button" onclick="loadXMLDoc()">修改内容</button>

</body>
</html>
```

Fetching Data using XMLHttpRequest Object (Contd.)

- The following figures depict the output of the preceding code.

Fetching Data using XMLHttpRequest Object (Contd.)

- To display XML data from the server response, you need to make the following changes:

AJAXExample.htm

```
. . . . . .
xmlhttp.onreadystatechange = function()
{
if (xmlhttp.readyState == 4 &&
xmlhttp.status == 200) {
var xmldoc = xmlhttp.responseXML; var
root_node =
xmldoc.getElementsByTagName('root').item
(0);
div.innerHTML =
root_node.firstChild.data;
}
else {
 alert ("Something went wrong!!");
}}
xmlhttp.open("GET",“TestFile.xml",
true);
xmlhttp.send(null);
}}
. . . . . .
```

TestFile.xml

```
<?xml version="1.0" ?>
<root>
    This is a test XML file.
</root>
```

Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- *Limits of Ajax*
- Tips

Ajax Risk References

- Bookmarking Issues
- Back and Forward Button Problems
- Security Risks
- Search Engines

XMLHttpRequest security restrictions

- cannot be run from a web page stored on your hard drive
- can only be run on a web page stored on a web server

Two-request limit

- The HTTP 1.1 (RFC 2616) recommends that a single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.
- Most browsers (including IE) abide by this rule

prototypejs.org	prototypejs.org	2 KB	17ms
home.css	prototypejs.org	2 KB	159ms
prototype.js	prototypejs.org	18 KB	112ms
code_highlighter.j	prototypejs.org	3 KB	52ms
javascript.js	prototypejs.org	422 b	10ms
html.js	prototypejs.org	279 b	9ms
css.js	prototypejs.org	267 b	6ms
ruby.js	prototypejs.org	316 b	7ms
ebnf.js	prototypejs.org	146 b	4ms
urchin.js	google-analytics.com	6 KB	45ms
__utm.gif	google-analytics.com	35 b	140ms
hdrtile-home.gif	prototypejs.org	3 KB	138ms
codesample1.gif	prototypejs.org	6 KB	348ms
tagline-home.gif	prototypejs.org	8 KB	346ms
14 requests		45 KB	743ms

Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Limits of Ajax
- *Tips*

Ajax Optimization

- The way to optimize any Ajax application is to find the best method to optimize every element that may go into it. It is important that the application run as quickly and efficiently as possible.

Communication

- An Ajax application is fast when there is no huge delay in receiving new data from the server. Optimizing two areas will help your application succeed here.
  - The first is to compress all data sent to the client from the server. This is important in terms of quick data transport.
  - The second concerns the data itself. The data that is sent back and forth, both from the client and from the server, should be optimized as much as possible as well.

Data

- When sending data to the server, send what would be considered the minimum amount of information. If you are sending key/value pairs, keep both the key and the value small. Enumerate choices whenever possible. Instead of something like this:
  - user_choice=add_data_to_database&data1=value1&data2=value2
- consider letting each choice be set to a single value, and send that value instead:
  - c=3&d1=value1&d2=value2
- Smaller data makes it harder for intercepted information to be interpreted (a good security benefit) and keeps the size of the data that needs to be sent and parsed smaller.

Code Optimization

- The other important part of Ajax optimization is to optimize the JavaScript code that is executed on the client, and to create the fastest data retrieval that you can.
  - inline SQL queries
  - stored procedures
- On the client, a number of JavaScript techniques will help to increase the execution time of your code. This is especially important when it comes to the DOM manipulation that your code may need to do when the server receives an Ajax response.
- Nothing says that your Ajax application will not run smoothly and efficiently without any optimization. In most cases, the speed of Ethernet connections, the processing power of computers, and the better implementation of JavaScript in browsers will ensure that your applications run well. Optimization will give you an application that runs that much faster. For the user, faster is always better.

Best Practices for AJAX Implementations

- Understand What it All Means
  - WikiPedia
- Check for Appropriate Usage Scenarios
- Learn to Implement it With Raw Code

Use a Library

- Libraries not only provide an exhaustive feature set you can make use of but also makes sure your code is compatible with all browsers without you having to do anything extra.
  - axios
  - AngularJS
- Master the Library

Know the Limitations of Your Ajax Library

- All JavaScript libraries give you access to an Ajax object, which normalizes the differences between browsers and gives you a consistent interface. However, in giving you a unified interface, these libraries must also simplify the interface, because not every browser implements each feature. This prevents you from accessing the full power of XMLHttpRequest.
- Interacting directly with the XHR object also reduces the amount of function overhead, further improving performance. Just beware that by forgoing the use of an Ajax library, you may encounter some problems with older and more obscure browsers.

Utilize Proper Events and Callback Functions

- Make proper use of these events and their respective callbacks to manipulate the UI for a better user experience.

```
$.ajax({
    //Other code
      success: function(msg)
    {
      // Update the UI here to reflect that the request was successful.
      doSomethingClever();
    },
    error: function(msg)
    {
      // Update the UI here to reflect that the request was unsuccessful
      doSomethingMoreClever();
    },
    complete: function(msg)
    {
      // Update the UI here to reflect completion
      doSomethingEvenMoreClever();
    }
});
```

Choose the Right Format for the Job

- When considering data transmission techniques, you must take into account several factors: feature set, compatibility, performance, and direction (to or from the server).
- When considering data formats, the only scale you need for comparison is speed.

- There isn't one data format that will always be better than the others. Depending on what data is being transferred and its intended use on the page, one might be faster to download, while another might be faster to parse.

Data Formats - XML

- XML
  - ✓ extreme interoperability (with excellent support on both the server side and the client side)
  - ✓ strict formatting
  - ✓ easy validation
  - - extremely verbose. Each discrete piece of data requires a lot of structure, and the ratio of data to structure is extremely low.
  - - XML also has a slightly ambiguous syntax.
  - - Parsing this syntax is equally ambiguous,
- XML has no place in high-performance Ajax

JSON

- Formalized and popularized by Douglas Crockford, JSON is a lightweight and easy-to-parse data format written using JavaScript object and array literal syntax.

Verbose JSON :

```
[
{ "id": 1, "username": "alice", "realname": "Alice Smith",
"email": "alice@alicesmith.com" },...]
```

Simple JSON

```
[
{ "i": 1, "u": "alice", "r": "Alice Smith", "e":
"alice@alicesmith.com" }, ......]
```

Array JSON

```
[
[ 1, "alice", "Alice Smith", "alice@alicesmith.com" ], ......]
```

JSON-P

- JSON with padding
  - when dynamic script tag insertion is used, JSON data is treated as just another JavaScript file and executed as native code. In order to accomplish this, the data must be wrapped in a callback function.
- Since the data is treated as native JavaScript, it is parsed at native JavaScript speeds.
- There is one reason to avoid using JSON-P that has nothing to do with performance: since JSON-P must be executable JavaScript, it can be called by anyone and included in any website using dynamic script tag insertion.
- Do not encode any sensitive data in JSON-P, because you cannot ensure that it will remain private, even with random URLs or cookies.

HTML

- HTML, as a data format, is slow and bloated.
-

Custom Formatting

- This type of format is extremely terse and offers a very high data-to-structure ratio (significantly higher than any other format, excluding plain text)
- one of the most important decisions is what to use as the separators.
-

```
1:alice:Alice Smith:alice@alicesmith.com;
2:bob:Bob Jones:bob@bobjones.com;
```

Data Format Conclusions

- Favor lightweight formats in general; the best are JSON and a character-delimited custom format. If the data set is large and parse time becomes an issue, use one of these two techniques:
  - JSON-P data, fetched using dynamic script tag insertion. This treats the data as executable JavaScript, not a string, and allows for extremely fast parsing. This can be used across domains, but shouldn't be used with sensitive data.
  - A character-delimited custom format, fetched using either XHR or dynamic script tag insertion and parsed using split(). This technique parses extremely large datasets slightly faster than the JSON-P technique, and generally has a smaller file size.

Cache Data

- The fastest Ajax request is one that you don't have to make. There are two main ways of preventing an unnecessary request:
  - On the server side, set HTTP headers that ensure your response will be cached in the browser.
  - On the client side, store fetched data locally so that it doesn't have be requested again.

References

- Ajax: The Definitive Guide.Anthony T. Holdener III. O'Reilly Media.
- Wiki – Rich Internet Application
  - http://en.wikipedia.org/wiki/Rich_Internet_application

Thanks!!!