Rich Internet Applications

4. Ajax

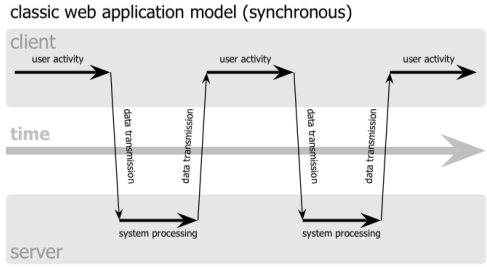
What is Ajax?

- **A**synchronous **Ja**vaScript and **X**ML
- Term coined in 2005 by Jesse James Garrett
  - http://www.adaptivepath.com/ideas/essays/archives/000385.php
- Ajax isn't really new, and isn't a single technology
- *"It's really several technologies, each flourishing in its own right, coming together in powerful new ways."*

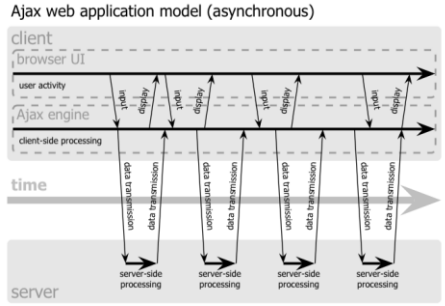
Defining AJAX

- Ajax incorporates:
  - standards-based presentation using **XHTML** and **CSS**
  - dynamic display and interaction using the **DOM**
  - data interchange and manipulation using **XML** and **XSLT**
  - asynchronous data retrieval using **XMLHttpRequest**
  - and **JavaScript** binding everything together.

Classic web application model



Ajax web application model



"Raw" Ajax

- Even if you are using an Ajax toolkit it is important to understand the underlying architecture
- Raw Ajax involves programming in JavaScript to:
  - use the XMLHttpRequest
  - write callback functions to handle response
  - write code to update the DOM
  - use only the native browser DOM and JavaScript support

Standard HTTP request

- Sent by browser when:
  - URL entered in address bar
  - Bookmark/favourite selected
  - Hyperlink in page clicked
  - Loading page refers to linked resources
- Uses HTTP protocol
- Response replaces currently displayed page unless it is a linked resource

XMLHttpRequest

- XMLHttpRequest is a scriptable DOM object
  - sent by **script** running in browser
  - typically sent by JavaScript event handler function
- Uses HTTP protocol
- Response returned to script
  - Can be XML, but doesn't have to be
  - does **not** cause browser to refresh page
  - script controls what is done with response data
  - typically used to modify DOM of currently displayed page

XMLHttpRequest history

- Originally developed by Microsoft in IE5 for Outlook Web Access 2000
- The Microsoft implementation is an ActiveX object called XMLHTTP
- Native DOM implementation in Mozilla 1.0 in 2002
- Most browsers, including IE7+, now natively support XMLHttpRequest
- W3C draft standard, April 2008

Creating an XMLHttpRequest

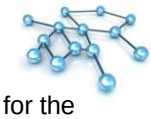
- Create a new object within a script

```
req = new XMLHttpRequest();
```

- For the benefit of older versions of IE, need to create an ActiveX object if native object is not supported

```
if (window.XMLHttpRequest)
{
    req = new XMLHttpRequest();
}
else if (window.ActiveXObject)
{
    req = new ActiveXObject("Microsoft.XMLHTTP");
}
```


Setting up an XMLHttpRequest



- Need to specify URL and method for the HTTP request which will be sent
- URL is the resource which returns the information required
  - Can be a file or a server script
  - Not usually a complete XHTML document

```
var url = "textdestinations.txt";
req = new XMLHttpRequest();
req.open("GET", url, true);
```

true specifies that request is asynchronous
false would cause script to wait for response

Ready states and callback functions

- Script needs to know when a response has been received
- XMLHttpRequest object generates an event to report on its progress
  - **onreadystatechange** event
  - **readyState** property
- **readyState** can have values from 0 to 4
- Event is fired when **readyState** changes
- Define **callback function** to handle event

XMLHttpRequest ready states

0: The request is uninitialized (before you've called open())

1: The request is set up, but not sent (before you've called send())

2: The request was sent and is in process

3: The request is in process; often some partial data is available from the response, but the server isn't finished with its response

4: The response is complete; you can get the server's response and use it

Defining a callback function

- Register event handler (using traditional model):

```
req.onreadystatechange = getDestinationsCallBack;
```

- Write callback function:

```
function getDestinationsCallBack()
{
   // event handling code
   ...
}
```


Ajax example

```
var req;
function getDestinations()
{
    var url = "/StaticData/destinations.txt";
    req = new XMLHttpRequest();
    req.open("GET", url, true);
    req.onreadystatechange = getDestinationsCallBack;
    req.send(null);
}
function getDestinationsCallBack()
{
    if (req.readyState == 4) {
        if (req.status == 200) {
            document.getElementById( 'destinationlist' ).innerHTML =  req.responseText;
        }
    }
}
window.onload = getDestinations;
...
<body>
<h3>Dynamically Populating a List</h3>
<div id="destinationlist"> </div>
</body>
```

getDestinations handles the **onload** event of the window

don't do anything unless **readyState** has reached **4** as data may not be available

don't do anything unless **status** is **200** which indicates a successful response

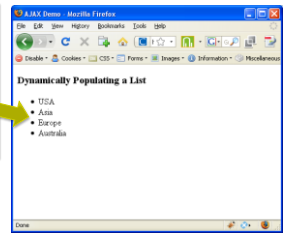
this example replaces the innerHTML property of the specified XHTML element with the content of the response

Ajax example in action

- Response contains an HTML snippet representing a list

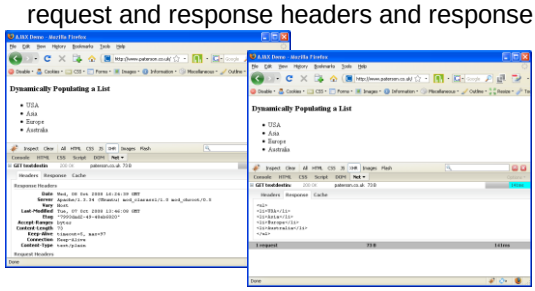
destinations.txt – contents

```
<ul>
<li>USA</li>
<li>Asia</li>
<li>Europe</li>
<li>Australia</li>
</ul>
```



Examining an XMLHttpRequest

- Firefox with **Firebug** lets you examine request and response headers and response



Ajax patterns

- This example used the following techniques:
  - XmlHttpRequest to get data
  - HTML formatting for the data
- These techniques are examples of **Ajax patterns**
- A **pattern** is a general reusable solution to a commonly occurring problem
- Description or template for how to solve a problem that can be used in many different situations

About Ajax patterns

- Ajaxpatterns.org
- Book: Ajax Design Patterns by Michael Mahemoff
- Types of patterns:
  - Foundational Technology patterns
  - Programming patterns
  - Functionality and Usability patterns
  - Development Practices

Pattern contents

- Developer/Goal story
- Problem
- Forces
- Solution
- Decisions
- Examples
- Alternatives
- Related patterns

Patterns used so far

- Foundational Technology Patterns
  - *Web Remoting*
    - **XmlHttpRequest Call**
- Programming Patterns
  - *Web Services*
    - **HTML Message**

HTML Message pattern

- **Developer Story**
  - Dave's identified the need for a credit history service, providing a list of transactions. JavaScript resources are limited, so the entire HTML is created server side, and the browser application has only to morph a DOM element with the entire HTML response
- **Problem**
  - What format should be used for server responses?

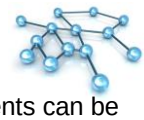
HTML Message pattern

- **Forces**
  - The browser display needs to be dynamic
  - The display changes by altering the DOM, and HTML is often used to specify the new value.
  - The nature of the change is often complex and needs to be determined server side.
  - Not all developers know JavaScript, and many feel its usage is best minimized.
- **Solution**
  - Have the server generate HTML snippets to be displayed in the browser....

Related patterns

- HTML Message is one of several possible ways of formatting data to be returned by a request:
- **Programming Patterns**
  - *Web Services*
    - **XML Message**
    - **JSON Message**
    - **Plain-Text Message**
- Similarly, XmlHttpRequestCall is one of several possible ways of getting the data - we will look at alternatives later on

Related patterns



- We have seen that the page contents can be modified by manipulating the DOM
- This relates to the following patterns:
- Foundational Technology Patterns
  - *Display Manipulation*
    - Display Morphing
    - Page Rearrangement

XML

- The **Extensible Markup Language** (**XML**) is a general-purpose *specification* for creating custom markup languages
- Allows its users to define their own elements
- Its primary purpose is to help information systems share structured data
- XML document is a text document which marks elements with tags enclosed in <> brackets

Simple XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<destinations company="gcutours">
        <destination>USA</destination>
        <destination>Asia</destination>
        <destination>Europe</destination>
        <destination>Australia</destination>
</destinations>
```

optional XML declaration

destinations element has an **attribute**

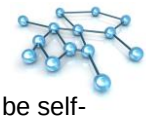
destination elements have **content**

destination elements nested inside **destinations** element

element **destinations** defined by opening and closing tags – tags **must** be properly closed

elements with no content or nested elements can be self-closing, e.g. <customer name="Jim"/>

XML rules



- All tags must have closing tags or be self-closing
- Tags are **case sensitive**, and must be **properly nested**
- A document which conforms to the XML syntax rules is called **well-formed**
- Can optionally specify document type definition or schema to **validate** structure of XML document

XML Message pattern

- **Forces**
  - Ajax Apps require messages to be transmitted back and forth
  - Both browser and the server must be able to access the message. That usually means the format must be easily accessed in JavaScript as well as whichever server-side language is used
  - The server application should not be tied to the client-side display as the message may be used by other applications
- **Problem**
  - How can you transfer data between server and browser?
- **Solution**
  - Pass messages between server and browser in XML format...

XML Message example

```
var req;
function getDestinations()
{
    var url = "textdestinations.xml";
    req = new XMLHttpRequest();
    req.open("GET", url, true);
    req.onreadystatechange = getDestinationsCallBack;
    req.send(null);
}

function getDestinationsCallBack()
{
...
}
...
<body onLoad="getDestinations()">
  <h3>Dynamically Populating a List</h3>
    <form name="destinationform">
      <p>Select a destination :</p>
      <p><select id="destinationlist"></select></p>
    </form>
</body>
```

getDestinations is much the same as before except for the URL

this time we will use the data to add options to a drop-down **select** box – the select box has no options to start with

XML Message example

```
function getDestinationsCallBack()
{
  if (req.readyState == 4) {
    if (req.status == 200) {
      var xmlDoc = req.responseXML;
      var destinations = xmlDoc.getElementsByTagName("destination");
      destSelect = document.getElementById("destinationlist");
      for (i=0;i<destinations.length;i++){
        value = destinations[i].childNodes[0].nodeValue;
        text = value;
        if (destSelect != null && destSelect.options != null)
        {
          destSelect.options[destSelect.options.length] =
              new Option(text, value, false, true);
        }
      }
      destSelect.options[0].selected = true;
    }
  }
}
```

parse response and construct an **XML document** in memory

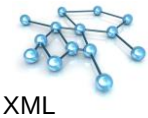
destinations is an *array* of the **destination** elements in the XML document

destSelect is the select box in the XHTML page

this syntax is needed to get the value of each **destination** element in the XML

this adds a new **option** to the **destSelect** box – displayed text and value are the same

JavaScript and XML



- XML data is represented as a live XML document
- XML document has its own DOM similar to browser DOM
- Need to use JavaScript DOM functions and traverse nodes to extract information

```
var destinations = xmlDoc.getElementsByTagName("destination");
value = destinations[i].childNodes[0].nodeValue;
```

- Can become tricky with complex data

JSON

- **JSON** (**JavaScript Object Notation**) is a lightweight data interchange format
- Text-based, human-readable format for representing simple data structures and associative arrays (called objects)
- Based on **JavaScript literal syntax** but can be used by other programming languages
- JSON data can be understood natively by JavaScript without having to access a DOM

JavaScript literals

- JavaScript allows variables to be created by specifying literal values, with no type declaration
- Simplest case is a single numeric or string **value**
- Can create **arrays** and arbitrarily complex **structured objects**

JavaScript literals

- Examples:

```
var myValue = "value";
var myArray = ["value1", "value2", "value3"];
var myObject = {"member1": "value" ,"member2": "value"}
```

- Combine these to get complex objects:

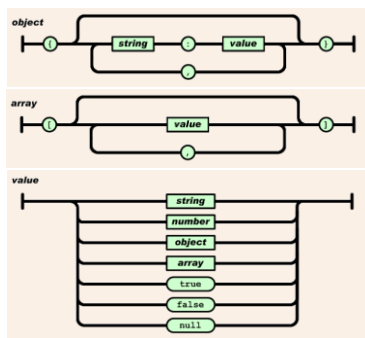
array of objects

```
var myComplexObject1 = [
                         {"member1": "value" ,"member2": "value"},
                         {"member1": "value" ,"member2": "value"},
                         {"member1": "value" ,"member2": "value"}
                       ];
```

object with an array as a member

```
var myComplexObject2 = {
                         {"member1": "value"},
                         {"member2": ["value1", "value2", "value3"] }
                       };
```


JSON structures



JSON and XML compared

XML

```
<destinations>
        <destination>USA</destination>
        <destination>Asia</destination>
        <destination>Europe</destination>
        <destination>Australia</destination>
</destinations>
```

JSON

```
["Asia","Australia","Europe","South America","USA"]
```


JSON and XML compared

XML

```
<packages>
  <package>
    <name>Western Adventure</name>
    <description>A typical tour is our Western Adventure ...</description>
    <tourpicurl>images/western.jpg</tourpicurl>
  </package>
  <package>
    <name>Roof of the World Explorer</name>
    <description>New this year is our Roof of the World tour... </description>
    <tourpicurl>images/roofoftheworld.jpg</tourpicurl>
  </package>
<packages>
```


JSON and XML compared

JSON

```
[
    {
    "name": "Western Adventure",
    "description": "A typical tour is our Western Adventure tour..",
    "tourpicurl": "images/western.jpg"
    },
    {
    "name": "Roof of the World Explorer",
    "description": "New this year is our Roof of the World tour...",
    "tourpicurl": "images/roofoftheworld.jpg"
    }
]
```

See http://www.json.org/example.html for more examples

JSON Message pattern

- **Forces**
  - Ajax Apps require messages to be transmitted back and forth
  - Both browser and the server must be able to access the message. That usually means the format must be easily accessed in JavaScript as well as whichever server-side language is used
  - The server application should not be tied to the client-side display as the message may be used by other applications
- **Problem**
  - How can you transfer data between server and browser?
- **Solution**
  - Pass messages between server and browser in JSON format...

JSON Message example

```
var req;
function getDestinations()
{
    var url = "/Datasources/Locations";
    req = new XMLHttpRequest();
    req.open("GET", url, true);
    req.setRequestHeader("Accept","application/json; charset=utf-8");
    req.onreadystatechange = getDestinationsCallBack;
    req.send(null);
}

function getDestinationsCallBack()
{
...
}
...
<body onLoad="getDestinations()">
  <h3>Dynamically Populating a List</h3>
    <form name="destinationform">
      <p>Select a destination :</p>
      <p><select id="destinationlist"></select></p>
    </form>
</body>
```

getDestinations is much the same as before except for the URL

the XHTML is exactly the same as the XML message example

JSON Message example

```
function getDestinationsCallBack()
{
   if (req.readyState == 4) {
      if (req.status == 200) {
         var response = JSON.Parse(req.responseText);
         destSelect = document.getElementById("destinationlist");
         for (i=0;i<response.length;i++){
            value = response [i];
            text = value;
            if (destSelect != null && destSelect.options != null)
            {
               destSelect.options[destSelect.options.length] =
                     new Option(text, value, false, true);
            }
         }
         destSelect.options[0].selected = true;
      }
   }
}
```

JavaScript **eval** function can evaluates JSON data as a JavaScript expression – essentially as a JavaScript object literal

response is an array

JSON data is:
[
"Asia",
"Australia",
"Europe",
"South America",
"USA"]

Plain-Text Message pattern

- **Forces**
  - Ajax Apps require messages to be transmitted back and forth
  - Both browser and the server must be able to access the message. That usually means the format must be easily accessed in JavaScript as well as whichever server-side language is used
- **Problem**
  - How can you transfer data between server and browser?
- **Solution**
  - Pass simple messages between server and browser in plain-text format. The emphasis here is on keeping things simple—XML or JSON are often more appropriate for complex data structures, but can also add unnecessary complexity....

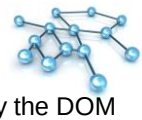
Plain-Text format examples

- Response Code
  - Sometimes, all that's required is a simple response code, such as a numeric ID or a text message such as "OK" or "Failed"
- A simple text string
  - For example, a user's name
- A list
  - For example, a comma-separated list of search results

Plain-Text format examples

- A custom data format
  - for example, a list of strings, where each string is a comma-separated list of object attributes.
- A JavaScript function call
  - text is a call to a function which exists in the page
  - can include parameter values created by server
  - use eval() to execute
  - security vulnerabilities

Using plain-text



- Simply use response text to modify the DOM

```
var response = req.responseText;
usernameTextbox= document.getElementById("username");
usernameTextbox.value = response;
...
<input type="text" id="username">
```

- Need to do some text parsing for more complex text formats, such as lists

Message format choices

- Plain-Text message
  - Good for simple messages
  - Skill-base: XML skills not required
  - More complex messages require parsing, and there is ready-made support available for XML and JSON
  - Non-standard message format
  - Not suitable for public web services which make their data available to many web sites

Message format choices

- HTML message
  - Good for performance and browser compatibility, as little client-side parsing is required
  - Not suitable for public web services as message must match a particular page format
  - Suitable when all HTML code is generated by server templates
  - Need to identify how much of the page content to load: form, div, paragraph, text box, etc.

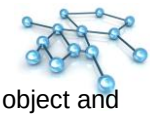
Message format choices

- XML message
  - Definitive standard for message passing
  - Wide client and server support for creating and parsing messages
  - Suitable for public web services
  - Choice of ways to transform XML to web page content
  - Manual JavaScript conversion (as in the example) or client-side XSLT (Extensible Stylesheet Transformations)

Message format choices

- JSON message
  - Increasingly widely used standard for message passing
  - Suitable for public web services
  - Cleaner and lighter than XML
  - Fairly good client and server support for creating and using messages
  - Passes JavaScript objects, no need to parse manually, no XML skills needed

Using jQuery



- jQuery is based on global **jQuery** object and the **jQuery()** function
- Shorthand syntax for **jQuery** is **$**
- Queries for DOM elements, e.g. $("#mydiv")
- Returns a **jQuery-wrapped set** that contains results and has methods which can operate on these results, e.g. $("#mydiv").hide()
- Each method also returns a jQuery-wrapped set, so can chain methods, , e.g. $("#mydiv").html(" Hello ").show()

jQuery selectors

- **$('*')**
  - all elements in document
- **$('.myclass')**
  - All elements with class="myclass"
- **$('td')**
  - All elements of type <td>
- **$('#mydiv')**
  - Element with id="mydiv"

jQuery selectors

- **$('td, th')**
  - All <td> and <th> elements
- **$('td input')**
  - All <input> elements inside <td> elements
- **$('[value="Submit"]')**
  - All elements with attribute value="Submit"
- **$(document)**
  - Whole document

jQuery inline functions

- **$(document).ready(handler)**
- Parameter for **ready** is a handler function which is called when DOM is loaded
- Parameter can be name of function, or inline function definition

```
$(document).ready(function() {
  ...
});
```

jQuery event handlers

- Handlers can be attached to other DOM elements in a similar way
- In this example, the click event handler for an element is attached once the document has loaded

```
$(document).ready(function () {
    $("#submit").click(function () {
        // code to handle click event...
    });
});
```


jQuery Ajax events

- **ajaxSend** event is fired when an Ajax request is about to be sent
- All handlers which have been registered with **.ajaxSend()** are invoked

```
$("#log").ajaxSend(function () {
        $(this).slideDown("fast").text("LOADING...");
});
```

this refers to element which handler is attached to - id="log"

- Other events include **ajaxError**, **ajaxComplete**

jQuery load() function

- Loads data from the server and place the returned HTML into the matched element

```
$('#result').load('ajax/test ');
```

- Can specify a callback function

```
$('#result').load('ajax/test', function() {
    alert('Load was performed.');
});
```

jQuery ajax() function

- Performs an asynchronous HTTP request
- Allows a wide range of options to be configured, including url, data type, callbacks,

```
$.ajax({
        url: '/home/person',
        type: 'POST',
        dataType: 'json',
        data: json,
        contentType: 'application/json; charset=utf-8',
        success: function (data) {
            // callback actions
            }
    });
```


Shorthand versions of .ajax()

- Functions which are essentially wrappers for .ajax() with some configurations preconfigured
- **.post()**
- **.get()**
- **.getJSON()**

JSON Message example using jQuery

```
$(document).ready(function () {
    $.ajax({
        type: "GET",
        url: "/Datasources/Locations",
        dataType: "json",
        success: function (json) {
            var select = $('#destinationlist');
            $.each(json, function (i, item) {
                select.append("<option>" + item + "</option>");
            });
            $('#destinationlist').attr('selectedIndex', '0');
        }
    });
});
```

use jQuery **ready** event

the XHTML is exactly the same as the JSON message example

Mocking Ajax

- Useful during development to be able to test client code in isolation
- Without making “real” calls to server for data
- Useful to be able to define “mock” data in client code for testing
- Example – Mockjax jQuery plugin
  - https://github.com/appendto/jquery-mockjax

Mockjax example

```
$(document).ready(function () {
   $.mockjax({                          // mock response for locations
          url: "/Locations",
          responseTime: 1000,
          responseText: ["Asia", "Australia", "Europe", "South America", "USA"]
       });

   $.ajax({
       type: "GET",
       url: "/Datasources/Locations",
       dataType: "json",
       success: function (json) {
          var select = $('#destinationlist');
          $.each(json, function (i, item) {
             select.append("<option>" + item + "</option>");
          });
          $('#destinationlist').attr('selectedIndex', '0');
       }
   });
});
```

Any calls to specified URL will be intercepted and mocked, will receive specified response

What’s next

- Using Ajax with remote services
- Dealing with JavaScript’s single-origin policy