Asynchronous Processing and AJAX

Introduction to Asynchronous Processing

There are two ways that a user web interface (webpage) can interact with the server. These are synchronous (where the webpage stops and waits for the server to send back a reply before allowing the user to continue) and asynchronous (where the webpage allows the user to continue using the page and will handle the reply when it arrives).

Asynchronous communication decouples user interaction from server interaction. As a result, when the user is using a web application, a client-side script running on that web application can request information from the server in the background. When new information arrives, the client-side script will only update the related user interface portion of the browser. An asynchronous processing is independent of the sessions on which requests are sent and replies are received.

The term "Ajax" was first mentioned in 2005, originally stands for "*Asynchronous JavaScript and XML*". It has since gone beyond XML, and currently serves as a client-side technology for JavaScript to transfer data between browser and server in any format, being JSON the most preferred format because JavaScript can easily turn JSON responses into a JavaScript objects. In fact Ajax provides a way to make synchronous calls but there is a 99.9% chance that you will never need to use Ajax to execute synchronous processing.

Advantages of Asynchronous Processing:

- ✓ Asynchronous processing does not use up threads waiting for the server to complete on blocking calls. This can increase the scalability of your system by reducing the number of full webpage downloads.
- ✓ Download of parts of the webpage can be broken up into different asynchronous calls that can be processed in different threads and concurrently.
- ✓ Increase the usability and interactivity of web pages because users can access information on the server without having to reload the web page in each user-server interaction. A Web page can update just a part of the page without disrupting what the user is doing.
- ✓ Nowadays implementing asynchronous processing is less complex with Ajax and JavaScript libraries.

Using the XMLHttpRequest Object

Ajax uses a JavaScript object called XMLHttpRequest, which contains a set of properties (or variables) and methods.

XMLHttpRequest was designed by Microsoft and adopted by Mozilla, Apple, and Google. It's now being standardized in the W3C. It provides an easy way to retrieve data from a URL without having to do a full page refresh. XMLHttpRequest is used heavily in AJAX programming.

Despite its name, XMLHttpRequest can be used to retrieve any type of data, not just XML, and it supports protocols other than HTTP (including file and ftp).

To create an instance of XMLHttpRequest use the following JavaScript code:

var myRequest = new XMLHttpRequest();

Example:

```
<html>
<head>
   <meta charset="UTF-8">
   <title>Example</title>
   <script>
     function showData(str) {
        var xmlhttp;
        if (str=="") {
           document.getElementById("txtHint").innerHTML="";
           return;
        }
        xmlhttp=new XMLHttpRequest();
        xmlhttp.onreadystatechange=function() {
           if (xmlhttp.readyState==4 && xmlhttp.status==200){
              document.getElementById("txtHint").innerHTML=
              xmlhttp.responseText;
           }
        }
        xmlhttp.open("GET","getdata.php?id="+str,true);
        xmlhttp.send();
     }
   </script>
   <style>
      td {align:left;background: #eee;padding:15px;}
    </style>
</head>
<body>
   <table border="1" width="100%" colspan="4" cellspacing="4">
     <tr>
        <td width="40%">
        <h3>Nation names</h3>
       <ul>
           <li> <a href="javascript: showData('1')">England</a></li>
           <li> <a href="javascript: showData('2')">Northern Ireland</li>
           <li> <a href="javascript: showData('3')">Scotland</li>
           <li> <a href="javascript: showData('4')">Wales</li>
        </ul>
        </td>
       <td valign="top">
           <h3>Description</h3>
           <div id="txtHint">Click on the name of a nation to start</div>
        </td>
     </tr>
   </table>
</body>
</html>
```

XMLHttpRequest is subject to the browser's same-origin policy: for security reasons, requests will only succeed if they are made to the same server that served the original web page.

To create an XMLHttpRequest object, however, is browser dependent:

- All browsers except old IE: `new XMLHttpRequest()`

The properties are:

- readyState: 0: uninitialized; 1: loading; 2: Loaded; 3: Interactive; and 4: completed.
- onreadystatechange: for specifying an event handler for the change in `readyState`.
- responseText and responseXML: The response returned by the server in text and XML format.
- status and statusText: The status code (e.g., 200, 404) and status message (e.g., OK, Page not found).

The methods are:

- open(*method*, *url*, *isAsyn*): Open an HTTP connection with the specified *`method`* (GET or POST), *`url`*, and whether the request should be handled asynchronously (`true` or `false`). Asynchronous request run in the background and does not freeze the browser. Using async request is always recommended, except short and small request.
- setRequestHeader(*param*, *value*): Set the request header for the given param=value pair.
- send(*params*): send the GET/POST key=value parameters to the server.
- abort(): abort the current request.
- getAllResponseHeader(): returns all headers as a string.
- getResponseHeader(*param*): returns the value of given *`param`* as a string.

Examples

Example 1: First Ajax

In this example, we use Ajax to send a POST request asynchronously to request for a text file.

HelloAjax.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hello, Ajax</title>
</head>

<body>
  <h2>Hello, Ajax</h2>
  <!-- holder for Ajax response text -->
  <div id="ajaxText"></div>
  <a href="test" onclick="return loadAjaxText()">SEND</a>
</body>

<script type="text/javascript">

// Load Ajax responseText into element with id="ajaxText"
function loadAjaxText() {
   // Allocate an XMLHttpRequest object
   if (window.XMLHttpRequest) {
      // IE7+, Firefox, Chrome, Opera, Safari
      var xmlhttp=new XMLHttpRequest();
   } else {
      // IE6, IE5
      var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
   }
   // Set up the readyState change event handler
   xmlhttp.onreadystatechange = function() {
      if ((this.readyState === 4) && (this.status === 200)) {
         if (this.responseText !== null) {
            // Load the responseText into element with id="ajaxText"
            var outElm = document.getElementById('ajaxText');
            outElm.innerHTML = this.responseText;
         }
      }
   }
   // Open an asynchronous POST connection and send request
   xmlhttp.open("POST", "HelloAjax.txt", true);
   xmlhttp.send();
   return false;  // Do not follow hyperlink
}
</script>
</html>
```

HelloAjax.txt

```
This is the Response Text of Ajax Request!!!
```

You need to run this example under a HTTP server (e.g., Apache) as it sends a HTTP POST request.

The client-side HTML contains a `<div>`, to be used as placeholder for the response text. It also contains a hyperlink to trigger the Ajax request (the JavaScript function `loadAjaxText()`).

The steps in using Ajax, in JavaScript function loadAjaxText() are:

1. Allocate an `XMLHttpRequest` object, via `new XMLHttpRequest()` (for all browsers except IE) or `new ActiveXObject("Microsoft.XMLHTTP")` (for IE).
2. Open a GET or POST connection to the server, via `open("GET|POST", url, isAync)` method. In this case, we request for a text file called "`HelloAjax.txt`".
3. Invoke `send()` to send the request to the server.
4. Provide the event handler `onreadystatechange` to handle the `readyState` change event. The `readyState` of 4 indicates Ajax transfer completed; `status` of 200 (`statusCode` of OK) indicates successful request; the response message can be retrieved from the `responseText`. In this example, we load the `responseText` into the element with `id="ajaxText"`.

You can place the allocation of ajax object in a separate function for ease of maintenance:

```
/*
 * Request for an AJAX object.
 * Usage: var request = ajaxRequest();
 */
function ajaxRequest() {
   var activexModes = ["Msxml2.XMLHTTP", "Microsoft.XMLHTTP"];
                               // activeX versions to check for in IE
   if (window.ActiveXObject) { // for IE first (as XMLHttpRequest in IE7 is
broken)
      for (var i = 0; i < activexModes.length; ++i) {
         try {
            return new ActiveXObject(activexModes[i]);
         } catch(e) { }        // Suppress error
      }
   } else if (window.XMLHttpRequest) { // for Mozilla, Safari, etc
      return new XMLHttpRequest();
   } else {
      return false;
   }
}
```

Why Ajax (Instead of an Ordinary HTTP Form POST request)?

Asynchronous Request for Better Responsiveness: Without Ajax with asynchronous request, the browser will freeze (and hang) while processing the request. On the other hand, with the asynchronous request in the background, the browser (JavaScript) do not have to wait for the response and can process other tasks. The `onreadystatechange()` handler will be called back when the `readyState` changes (e.g., transfer completed).

No Page Reloading: the Ajax request is run in the background, without the need to reload the page.

Example 2: Ajax with PHP

In this example, we trigger an Ajax POST request to a PHP page to obtain dynamic response.

HelloAjaxPHP.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hello, Ajax</title>
</head>

<body>
  <h2>Hello, Ajax</h2>
  <table>
    <tr>
      <td><textarea id="inText" name="inText" cols="40" rows="5">Enter
your text here...</textarea></td>
      <td><textarea id="ajaxText" name="ajaxText" cols="40"
rows="5"></textarea></td>
    </tr>
   </table>
   <a href="test" onclick="return loadAjaxText()">SEND</a>
</body>

<script type="text/javascript">
// Append Ajax responseText into element with id="ajaxText"
function loadAjaxText() {
   // Allocate an XMLHttpRequest object
   if (window.XMLHttpRequest) {
      // IE7+, Firefox, Chrome, Opera, Safari
      var xmlhttp=new XMLHttpRequest();
   } else {
      // IE6, IE5
      var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
   }
   // Set up the readyState change event handler
   xmlhttp.onreadystatechange = function() {
      if ((this.readyState === 4) && (this.status === 200)) {
         if (this.responseText !== null) {
            var ajaxElm = document.getElementById('ajaxText');
            ajaxElm.innerHTML = this.responseText + ajaxElm.innerHTML; // append in front
         }
      }
   }
   // Open an asynchronous POST connection and send request
   xmlhttp.open("POST", "HelloAjax.php", true);
   xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
   var inTextElm = document.getElementById('inText');
   xmlhttp.send("inText=" + inTextElm.value);
   return false;  // Do not follow hyperlink
}
</script>
</html>
```

The client-side HTML contains two `<textarea>`s, for the input and output respectively. It also contains a hyperlink to trigger the Ajax request (the JavaScript function `loadText()`).

The steps in using Ajax, in JavaScript function `loadText()` are:

1. Allocate an `XMLHttpRequest` object, via `new XMLHttpRequest()` (for all browsers except IE) or `new ActiveXObject("Microsoft.XMLHTTP")` (for IE).
2. Open a GET or POST connection to the server, via `open("GET|POST", url, isAync)` method.
3. Set the HTTP request headers, if needed. For example, you need to set the HTTP request header "`Content-type`" to "`application/x-www-form-urlencoded`" if you are sending POST data.
4. Invoke `send(data)` to send the POST request with query parameters to the server.
5. Provide the event handler `onreadystatechange()` for the `readyState` change event. The `readyState` of 4 indicates Ajax transfer completed; `status` of 200 (`statusCode` of OK) indicates successful request; the response message can be retrieved from the `responseText`.

Server-side Program: HelloAjax.php

```
<?php
// Echo back the POST parameter inText=xxx
if (isset($_POST['inText'])) {
  echo ">> {$_POST['inText']}\n";
}
?>
```

The server-side program `HelloAjax.php` handles a POST request with parameter `inText=`*`message`*, and echos back the input message.

Example 3: Processing XML Document

HelloAjaxXML.html

```



<!DOCTYPE html>
<html>
<head>
<title>Hello, Ajax</title>
</head>

<body>
  <h2>Hello, Ajax with XML</h2>
  <!-- holder for Ajax response text -->
  <div id="ajaxText"></div>
  <a href="test" onclick="return loadText()">SEND</a>
</body>

<script type="text/javascript">
function loadText() {
   var xmlhttp=new XMLHttpRequest();
   xmlhttp.onreadystatechange = function() {
      if ((this.readyState === 4) && (this.status === 200)) {
         if (this.responseXML !== null) {
            var xmlDoc = this.responseXML;
            var txt = "";
            // Retrieve all the "title" elements from the XML document
            var titleElms = xmlDoc.getElementsByTagName("title");
            for (i = 0; i < titleElms.length; i++) {
               txt += titleElms[i].childNodes[0].nodeValue + "<br />";
            }
            document.getElementById("ajaxText").innerHTML = txt;
         }
      }
   }
   xmlhttp.open("POST", "HelloAjax.xml", true);
   xmlhttp.send();
   return false;
}
</script>
</html>



```

The XML File: HelloAjax.xml

```
<bookstore>
  <book>
    <title>Java For Dummies</title>
    <author>Tan Ah Teck</author>
  </book>
  <book>
    <title>Java For More Dummies</title>
    <author>Tan Ah Teck</author>
  </book>
</bookstore>
```

Using JSON with Ajax

I recommend JSON over XML when doing Ajax. Why? Because a JavaScript engine can easily turn that JSON response into a JavaScript object... allowing you to access/manipulate that data very easily. You just have to use eval() or JSON.parse() or something similar (depending on the browser/javascript library).

JSON is valid JavaScript; so on the whole it meshes much better with Ajax/Javascript/Web than XML does.

JSON also tends to be a bit less verbose, especially in regards to arrays and key/value pairs... something you are likely to be encountering a lot with web services.

With XML people tend to create their own specialized XML vocabulary. So if anyone wanted to use your services, they'd also have to learn your XML vocab. JSON is much more universal in this regard.

The jQuery ajax() Function is the Lowest-Level Abstraction

The jQuery `ajax()` function is the lowest level of abstraction available for XMLHttpRequests (aka AJAX). All the other jQuery AJAX functions, such as `load`(), leverage the `ajax()` function. Using the `ajax()` function provides the greatest functionality available for `XMLHttpRequests`. jQuery also provides other higher-level abstractions for doing very specific types of `XMLHttpRequests`. These functions are essentially shortcuts for the `ajax()` method.

These shortcut methods are:

- `load()`
- `get()`
- `getJSON()`
- `getScript()`
- `post()`

The point to take away is that while the shortcuts are nice at times, they all use `ajax()` behind the scenes. And, when you want all the features and customizations that jQuery offers when it comes to AJAX, then you should just use the `ajax()` method.

Notes: By default, the `ajax()` and `load()` AJAX functions both use the GET HTTP protocol.

---

jQuery Supports Cross-Domain JSONP

JSON with Padding (JSONP) is a technique that allows the sender of an HTTP request, where JSON is returned, to provide a name for a function that is invoked with the JSON object as a parameter of the function. This technique does not use

XHR. It uses the script element so data can be pulled into any site, from any site. The purpose is to circumvent the same-source policy limitations of XMLHttpRequest.

Using the getJSON() jQuery function, you can load JSON data from another domain when a JSONP callback function is added to the URL. As an example, here is what a URL request would look like using the Flickr API.

http://api.flickr.com/services/feeds/photos_public.gne?tags=resig&tagmode=all&format=json&jsoncallback=?

The ? value is used as a shortcut that tells jQuery to call the function that is passed as a parameter of the getJSON() function. You could replace the ? with the name of another function if you do not want to use this shortcut.

Below, I am pulling into a Web page, using the Flickr JSONP API, the most recent photos tagged with “resig.” Notice that I am using the ? shortcut so jQuery will simply call the callback function I provided the getJSON() function. The parameter passed to the callback function is the JSON data requested.

```
<!DOCTYPE html>
<html lang="en">
<body>
    <script
src="ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
    <script>  (function($){
      $.getJSON("http://api.flickr.com/...",
      // Using ? just means call the callback function provided
      function (data) { // Data is the JSON object from Flickr
           $.each(data.items, function (i, item) { $('<img
/>').attr("src", item.media.m).appendTo('body');
           if (i == 30) return false; });
          });
  })(jQuery); </script>
</body>
</html>
```

Notes: Make sure you check the API of the service you are using for the correct usage of the callback. As an example, Flickr uses the name `jsoncallback=?` whereas Yahoo! and Digg use the name `callback=?`.

Stop a Browser From Caching XHR Requests

When doing an XHR request, Internet Explorer will cache the response. If the response contains static content with a long shelf life, caching may be a good thing. However, if the content being requested is dynamic and could change by the second, you will want to make sure that the request is not cached by the browser. One possible solution is to append a unique query string value to the end of the URL. This will ensure that for each request the browser is requesting a unique URL.

```
// Add unique query string at end of the URL
url: 'some?nocache='+(newDate()).getTime()
```

Another solution, which is more of a global solution, is to set up all AJAX requests by default to contain the no-cache logic we just discussed. To do this, use the ajaxSetup function to globally switch off caching.

```
$.ajaxSetup({
            cache: false // True by default. False means caching is off.
});
```

Now, in order to overwrite this global setting with individual XHR requests, you simply change the cache option when using the ajax() function. Below is a coded example of doing an XHR request using the ajax() function, which will overwrite the global setting and cache the request.

```
$.ajaxSetup ({
    cache: false // True by default. False means caching is off.
});
jQuery.ajax({ cache: true, url: 'some', type: 'POST' } );
```

Further Reading

Head Rush Ajax by Brett McLaughlin

http://www.icbl.hw.ac.uk/santiago/teaching/F28WP/References/Ajax_SampleChapter.pdf
https://codehandbook.org/python-flask-jquery-ajax-post/