
AJAX

Asynchronous **J**avaScript **A**nd **X**ML

AJAX

AJAX Keywords

- JavaScript and XML
- Web browser technology
- Open web standards
- Browser and platform independent
- Better and faster Internet applications
- XML and HTTP Requests

How we interacted with the web till now…

- Typical browsing behaviour consists of loading a web page, then selecting some action that we want to do, filling out a form, submitting the information, etc.

- We work in this sequential manner, requesting one page at a time, and have to wait for the server to respond, loading a whole new web page before we continue.

- This is also one of the limitations of web pages, where transmitting information between a client and server generally requires a new page to be loaded.

- JavaScript is one way to cut down on (some of) the client-server response time, by using it to verify form (or other) information *before* it's submitted to a server.
- One of the limitations of JavaScript is (or used to be) that there was no way to communicate directly with a web server.
- Another drawback to this usual sequential access method is that there are many situations where you load a new page that shares lots of the same parts as the old (consider the case where you have a "menu bar" on the top or side of the page that doesn't change from page to page).

Why AJAX ?

- Ajax (sometimes written AJAX) is a means of using JavaScript to communicate with a web server without submitting a form or loading a new page.

- The term “Ajax” was coined in 2005, but the `XMLHttpRequest` object was first supported by Internet Explorer several years before this.

What is AJAX?

- AJAX is an acronym for **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX is not a programming language. but simply a development technique for creating interactive web applications.
- The technology uses JavaScript to send and receive data between a web browser and a web server.
- The AJAX technique makes web pages more responsive by exchanging data with a server behind the scenes, instead of reloading an entire web page each time a user makes a change.
- With AJAX, web applications can be faster, more interactive, and more user friendly.
- AJAX uses an XMLHttpRequest object to send data to a web server, and XML is commonly used as the format for receiving server data, although any format including and plain text can be used.

AJAX Application Example

- Online test
  - Many multiple choice questions
  - All questions are displayed on one page
  - After the user answers one question, the correct answer and explanation about why the user answer is wrong is shown on the page
  - For all already-answered questions, their correct answers and explanations are always shown on the page
- Pure sever-side solution using conventional web application
  - For each question answer submission, the whole page with most of repeated data sent to the browser
- Pure client-side solution using conventional JavaScript
  - The user can read JavaScript source code to view what is correct answer
  - Large amount of explanation data will be carried by the JavaScript code
- AJAX solution
  - After the user answers a question, use XmlHttpRequest to ask the server to send the correct answer and explanation.
  - Display the correct answer and explanation received from the server

Ajax

- Ajax stands for “Asynchronous JavaScript and XML”.
- The word “asynchronous” means that the user isn’t left waiting for the server the respond to a request, but can continue using the web page.

- The typical method for using Ajax is the following:

1) A JavaScript creates an `XMLHttpRequest` object, initializes it with relevant information as necessary, and sends it to the server. The script (or web page) can continue after sending it to the server.
2) The server responds by sending the contents of a file or the output of a server side program (written, for example, in PHP).
3) When the response arrives from the server, a JavaScript function is triggered to act on the data supplied by the server.
4) This JavaScript response function typically refreshes the display using the DOM, avoiding the requirement to reload or refresh the entire page.

The Back End

- The part of the Ajax application that resides on the web server is referred to as the "back end".
- This back end could be simply a file that the server passes back to the client, which is then displayed for the user.
- Alternatively, the back end could be a program, written in PHP, Perl, Ruby, Python, C, or some other language that performs an operation and sends results back to the client browser.
- An `XMLHttpRequest` object can send information using the GET and POST methods to the server in the same way that an HTML form sends information.
- Recall from our previous discussions that the GET request encodes the information inside of the URL, while a POST request sends its data separately (and can contain more information than a GET request can).

Writing an Ajax application

- We have to write the “front end” of the application in JavaScript to initiate the request.
- The back end, as mentioned, processes the request and sends it’s response back to the client. The back end is typically a short program we write for performing some dedicated task. This could be scripted in any language that is capable of sending back communication to the browser, like PHP or Perl.
- We also need to write the JavaScript response function for processing the response and displaying any results (or alterations to the web page).

- The “x” in Ajax stands for XML, the extensible markup language. XML looks like HTML, which is no mistake as the latest versions of HTML are built upon XML. The back end could send data back in XML format and the JavaScript response function can process it using built-in functions for working with XML. The back end could also send plain text, HTML, or even data in the JavaScript format.

- We will discuss some of these methods for sending data back to the requesting client and how it can be processed.

The XMLHttpRequest object

- The `XMLHttpRequest` object is the backbone of every Ajax method. Each application requires the creation of one of these objects. So how do we do it?

- As with most things in web programming, this depends upon the web browser that the client is using because of the different ways in which the object has been implemented in the browsers.

- Firefox, Safari, Opera, and some other browsers can create one of these objects simply using the “new” keyword.

```
<script type="text/javascript">
          ajaxRequest = new XMLHttpRequest();

</script>
```

The XMLHttpRequest object (cont.)

- Microsoft Internet Explorer implements this object using its proprietary ActiveX technology. This requires a different syntax for creating the object (and can also depend upon the particular version of Internet Explorer being used).

- To handle different types of browsers, we use the

```
try { . . . } catch (error) { . . . }
```

  format. The "try" section attempts to execute some JavaScipt code. If an error occurs, the "catch" section is used to intervene before the error crashes the JavaScript (either to indicate an error has happened, or to attempt something else).

- To create one of these objects we can use a sequence of try. . . catch blocks, attempting different ways to create an `XMLHttpRequest` object.

The XMLHttpRequest object (cont.)

```
function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
   var request, err;
   try {
          request = new XMLHttpRequest();   // Firefox, Safari, Opera, etc.
       }
   catch(err) {
       try {              //  first attempt for Internet Explorer
          request = new ActiveXObject("MSXML2.XMLHttp.6.0");
           }
       catch (err) {
                      try {    //  second attempt for Internet Explorer
                      request = new ActiveXObject("MSXML2.XMLHttp.3.0");
                          }
                      catch (err) {
                           request = false;  // oops, can't create one!
                                  }
                  }
              }
   return request;
}
```

If this function doesn't return "false" then we were successful in creating an `XMLHttpRequest` object.

The XMLHttpRequest object (cont.)

- As with any object in JavaScript (and other programming languages), the XMLHttpRequest object contains various properties and methods.
- We list the most important of these properties and methods on the next pages.
- The main idea is that the properties are set after the object is created to specify information to be sent to the server, as well as how to handle the response received from the server. Some properties will be updated to hold status information about whether the request finished successfully.
- The methods are used to send the request to the server, and to monitor the progress of the request as it is executed (and to determine if it was completed successfully).

XMLHttpRequest object properties

Property	Description
• readyState	An integer from 0. . .4. (0 means the call is uninitialized, 4 means that the call is complete.)
• onreadystatechange	Determines the function called when the objects readyState changes.
• responseText	Data returned from the server as a text string (read-only).
• responseXML	Data returned from the server as an XML document object (read-only).
• status	HTTP status code returned by the server
• statusText	HTTP status phrase returned by the server

We use the `readyState` to determine when the request has been completed, and then check the `status` to see if it executed without an error. (We'll see how to do this shortly.)

XMLHttpRequest object methods

Method	Description
• `open('method', 'URL', asyn)`	Specifies the HTTP method to be used (GET or POST as a string, the target URL, and whether or not the request should be handled asynchronously (asyn should be true or false, if omitted, true is assumed).
• `send(content)`	Sends the data for a POST request and starts the request, if GET is used you should call send(null).
• `setRequestHeader('x','y')`	Sets a parameter and value pair x=y and assigns it to the header to be sent with the request.
• `getAllResponseHeaders()`	Returns all headers as a string.
• `getResponseHeader(x)`	Returns header x as a string.
• `abort()`	Stops the current operation.

The `open` object method is used to set up the request, and the `send` method starts the request by sending it to the server (with data for the server if the POST method is used).

A general skeleton for an Ajax application

```
<script type="text/javascript">
       // *****  include the getXMLHttpRequest function defined before
var ajaxRequest = getXMLHttpRequest();

if (ajaxRequest) {   //  if the object was created successfully

      ajaxRequest.onreadystatechange = ajaxResponse;
      ajaxRequest.open("GET", "search.php?query=Bob");
      ajaxRequest.send(null);
   }


function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {   //  process server data here. . . }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
</script>
```

A first example

- Here's an example to illustrate the ideas we've mentioned (inspired by an example in the book Ajax in 10 Minutes by Phil Ballard).
- The main idea is that we're going to get the time on the server and display it to the screen (and provide a button for a user to update this time). The point I want to demonstrate here is how to use Ajax to do this update without updating/refreshing the entire webpage.

- We use a (very) small PHP script to get the date from the server, and return it as a string as a response to the request. Here is the script:

```
<?php
echo date('H:i:s');
?>
```

- I saved this as the file "telltime.php".
- The HTML file and JavaScript code follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Ajax Demonstration</title>
<style>
body {
   background-color: #CCCCCC;
   text-align: center;
     }
.displaybox {
   margin: auto;
   width: 150px;
   background-color: #FFFFFF;
   border: 2px solid #000000;
   padding: 10px;
   font: 1.5em normal verdana, helvetica, arial, sans-serif;
            }
</style>

<script type="text/javascript">
var ajaxRequest;

function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
      //  same code as before. . .
}
```

```
function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
          {
                   document.getElementById("showtime").innerHTML =
                                    ajaxRequest.responseText; }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}


function getServerTime()   //  The main JavaScript for calling the update.
{
   ajaxRequest = getXMLHttpRequest();
   if (!ajaxRequest)  {
         document.getElementById("showtime").innerHTML = "Request error!";
         return;      }
   var myURL = "telltime.php";
   var myRand = parseInt(Math.random()*999999999999999);
   myURL = myURL + "?rand=" + myRand;
   ajaxRequest.onreadystatechange = ajaxResponse;
   ajaxRequest.open("GET", myURL);
   ajaxRequest.send(null);
}
</script>
</head>
```

```
<body onload="getServerTime();">
<h1>Ajax Demonstration</h1>

<h2>Getting the server time without refreshing the page</h2>

<form>
   <input type="button" value="Get Server Time" onclick="getServerTime();" />
</form>
<div id="showtime" class="displaybox"></div>

</body>
</html>
```

The main functionality is handled by the `getServerTime()` function in setting up and sending the `XMLHttpRequest` object, and the `ajaxResponse()` function to display the time.

view the output page

What's this business with the random numbers?

- Web browsers use caches to store copies of the web page. Depending upon how they are set up, a browser could use data from its cache instead of making a request to the web server.

- The whole point of Ajax is to make server requests and not to read data from the cache. To avoid this potential problem, we can add a parameter with a random string to the URL so that the browser won't be reading data from its cache to satisfy the request (as then it looks like a different request than previous ones).

- This is only necessary if the request method is GET, as POST requests don't use the cache. (This also seems to be more of an issue with Microsoft Internet Explorer than with other browsers.)