**AJAX (Asynchronous Javascript and Xml)**

1. AJAX is not a new technology. It is simply based on existing technology like html, css, javascript and xml.
2. In the case of old request-response model browser is responsible to send the request and to receive the response.
3. When browse is sending the request to server. It will do the following tasks:
4. Browser opens the “http” connection.
5. Browser collects the client submitted data.
6. Browser collects the browser supported information.
7. Browser collects cookies available in the current machine.
8. Browser is responsible to send all the collected data to the server along with HttpRequest.
9. When browser receives the response do the following things:
10. Collects response stream.
11. Collects the html tags available in the response stream.
12. Constructs the html DOM tree.
13. Renders the html DOM of the document in the browser.
14. Closes the Http connection.
15. Because of doing many tasks set request-response time, which are not required in all the cases. Performance of the application will be come down i.e. response will be rendered to the client very slowly every time and also because of reloading the same page again and again, User of the application don’t feel comfortable to use the application.
16. Ajax solves these problems by sending the request asynchronously and by avoiding the page reloads every time.
17. To develop ajax based applications your browsers must support XMLHttpRequest or XMLHttp depending on the browser.
18. XMLHttp object support only by the internet explorer and all other browsers support XMLHttpRquest.
19. AJAX object has the following functionality.
20. **State**

* UNSENT = 0
* OPENED = 1
* HEADERS\_RECIEVED = 2
* LOADING = 3
* DONE = 4

1. **Request**

* open(method, url
* open(method, url, asyn)
* abort()
* send()

1. **Response**

* DOMString getAllResponseHeaders()
* DOMString getResponseHeader(header)
* DOMString responseText()
* DOMString responseXML()
* short status
* DOMString statusText

1. **EventHandler**

* onReadyStateChange

--------------------

**How ajax works?**

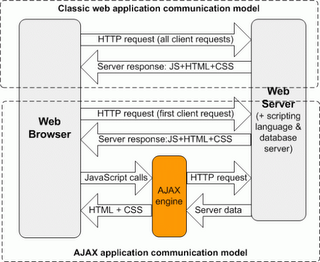
It is important to understand that Ajax is not a single technology, but a group of technologies e.g. HTML, CSS, DOM and JavaScript etc. [**HTML**](https://en.wikipedia.org/wiki/HTML)and [**CSS**](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) can be used in combination to mark up and style information. The [**DOM**](http://en.wikipedia.org/wiki/Document_Object_Model)is accessed with[**JavaScript**](https://en.wikipedia.org/wiki/JavaScript) to dynamically display, and allow the user to interact with, the information presented. JavaScript and the[**XMLHttpRequest**](http://en.wikipedia.org/wiki/XMLHttpRequest) object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

In recent years, essence of XML has been reduced. [**JSON**](http://en.wikipedia.org/wiki/JSON) (JavaScript Object Notation) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used for data purpose.

Normally, an ajax call to server and getting back response (**life cycle events**) from server involve following steps:

* You type the URL of a webpage in browser’s address bar and hit enter. Page is loaded in browser window.
* Some action triggers an event, like the user clicking a button.
* Event fires the ajax call, and sends a request to a server using xml or json.
* The server service takes the input from ajax/http request, and processes the request. It also prepare the response data if required.
* Server sends the data back to the original webpage that made the request.
* Another JavaScript function, called a callback function, receives the data, and updates the web page.

Easy enough, right? Lets see all the action in below given picture.



***How AJAX works?***

**Ajax request and response objects**

The**core of AJAX is the XMLHttpRequest object** (available in client side scripting languages like javascript). The XMLHttpRequest object is used to exchange data with a server behind the scenes. All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.**If you are using IE 5 or IE6** (I wonder if someone still uses it), then **ActiveXObject will be used** for server communication to send ajax requests.

A new object of XMLHttpRequest is created like this :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | //Creating a new XMLHttpRequest object  var xmlhttp;  if (window.XMLHttpRequest)  {      xmlhttp = new XMLHttpRequest(); //for IE7+, Firefox, Chrome, Opera, Safari  }  else  {      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); //for IE6, IE5  } |

This xmlhttp variable can be re-used to send multiple ajax requests, without creating new objects. **XMLHttpRequest is subject to the browser’s same origin policy for security reasons**. It means that requests will only succeed if they are made to the same server that served the original web page.

**Useful methods to work with XMLHttpRequest**

To send request and set request attributes, XMLHttpRequest object has some methods. Lets have a look onto them:

**a) open(method, url, isAsync, userName, password)**

The HTTP and HTTPS requests of the **XMLHttpRequest object must be initialized through the open method**.  This method specifies the type of request (GET, POST etc.), the URL, and if the request should be handled asynchronously or not. I will cover this third parameter in next section.

The fourth and fifth parameters are the username and password, respectively. These parameters, or just the username, may be provided for authentication and authorization if required by the server for this request.

Example:

|  |  |
| --- | --- |
| 1  2  3 | xmlhttp.open("GET","report\_data.xml",true);  xmlhttp.open("GET","sensitive\_data.xml",false);  xmlhttp.open("POST","saveData",true,"myUserName","somePassord"); |

**b) setRequestHeader(name, value)**

Upon successful initialization of a request, the setRequestHeader method of the XMLHttpRequest object can be invoked **to send HTTP headers with the request**.

Example:

|  |  |
| --- | --- |
| 1  2 | //Tells server that this call is made for ajax purposes.  xmlhttp.setRequestHeader('X-Requested-With', 'XMLHttpRequest'); |

**c) send(payload)**

**To send an HTTP request**, the send method of the XMLHttpRequest must be invoked. This method accepts a single parameter containing the content to be sent with the request.  
The content is necessary in POST requests. For GET methods, imply pass null as parameter.

Example:

|  |  |
| --- | --- |
| 1  2 | xmlhttp.send(null); //Request with no data in request body; Mostly used in GET requests.  xmlhttp.send( {"id":"23423"} ); //Request with data in request body; Mostly used in POST/ PUT requests. |

**4) abort()**

This method **aborts the request if the readyState of the XMLHttpRequest object has not yet become 4** (request complete). The abort method ensures that the callback method does not get invoked in an asynchronous request.

Syntax:

|  |  |
| --- | --- |
| 1  2 | //Abort the processing  xmlhttp.abort(); |

Apart from above method, onreadystatechange event listener is very important which we will discuss in next section.

**Synchronous and Asynchronous requests**

XMLHttpRequest object is capable of sending synchronous and asynchronous requests, as required within webpage. The behavior is controlled by **third parameter of open method**. This parameter is set to **true for an asynchronous requests, and false for synchronous requests**.

|  |  |
| --- | --- |
| 1  2 | xmlhttp.open("GET", "report\_data.xml", true); //Asynchrnonos request as third parameter is true  xmlhttp.open("GET", "report\_data.xml", false); Synchrnonos request as third parameter is false |

The **default value of this parameter is “true”** if it is not provided.

Asynchronous Ajax Requests do not block the webpage and user can continue to interact with other elements on the page, while the request is processed on server. You should always use asynchronous Ajax Requests because a synchronous Ajax Request makes the UI (browser) unresponsive. It means user will not be able to interact with the webpage, until the request is complete.

Synchronous requests should be used in rare cases with utmost care. For example, synchronous Ajax Request should be used if you’re embedding a new JavaScript file on the client using ajax and then referencing types and/or objects from that JavaScript file. Then the fetching of this new JS file should be included through using a synchronous Ajax Request.

**Example synchronous request**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var request = new XMLHttpRequest();  request.open('GET', '/bar/foo.txt', false);  //"false" makes the request synchronous  request.send(null);    if(request.status === 200)  {      //request successful; handle the response  }  else  {      //Request failed; Show error message  } |

**Example asynchronous request**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | var request = new XMLHttpRequest();  request.open('GET', '/bar/foo.txt', true);  //"true" makes the request asynchronous    request.onreadystatechange = function() {      if (request.readyState == 4) {          if (request.status == 200)          {              //request succeed          }          else          {              //request failed          }      }  };    request.send(null) |

In above example, onreadystatechange is a event listener registered with XMLHttpRequest request. onreadystatechange stores a function that will process the response returned from the server. It will be called for all important events in request’s life cycle. Every time an step is completed in request processing, the value of readyState will be changed and set to some other value. Lets have a look at possible values:

0 : request not initialized  
1 : server connection established  
2 : request received  
3 : processing request  
4 : request finished and response is ready to be handled

**Handling returned response from server**

To get the response from a server, responseText or responseXML property of the XMLHttpRequest object is used. If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property. If the response from the server is not XML, use the responseText property.

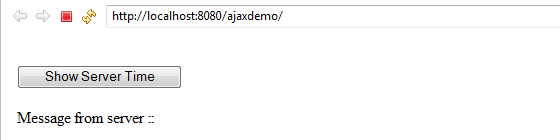
**responseText** : Get the response from server as a string  
**responseXML** : Get the response from server as XML

**Example code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | if (xmlhttp.readyState == 4) {      if (xmlhttp.status == 200)      {          document.getElementById("message").innerHTML = xmlhttp.responseText;      }      else {          alert('Something is wrong !!');      }  } |

**Demo application code**

For demonstration purpose, I am creating a very simple hello world application. In this application, webpage sends a ajax GET request to query the current server’s system time. In response, server sends the current time. Easy enough.



**Asynchronous request example**

To enable the webpage to send such request, I have written following javascript code in JSP page:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | function ajaxAsyncRequest(reqURL)  {      //Creating a new XMLHttpRequest object      var xmlhttp;      if (window.XMLHttpRequest){          xmlhttp = new XMLHttpRequest(); //for IE7+, Firefox, Chrome, Opera, Safari      } else {          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); //for IE6, IE5      }      //Create a asynchronous GET request      xmlhttp.open("GET", reqURL, true);        //When readyState is 4 then get the server output      xmlhttp.onreadystatechange = function() {          if (xmlhttp.readyState == 4) {              if (xmlhttp.status == 200)              {                  document.getElementById("message").innerHTML = xmlhttp.responseText;                  //alert(xmlhttp.responseText);              }              else              {                  alert('Something is wrong !!');              }          }      };        xmlhttp.send(null);  } |

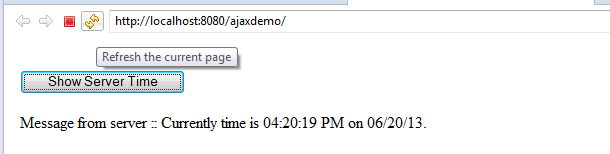
and to fire the ajax request, a button should be clicked which is written as:

|  |  |
| --- | --- |
| 1 | <input type="button" value="Show Server Time" onclick='ajaxAsyncRequest("get-current-time")'/> |

To handle the request on server side, I have written a servlet like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public class GetTimeServlet extends HttpServlet  {      private static final long serialVersionUID = 1L;        public void doGet (HttpServletRequest request,HttpServletResponse response)              throws ServletException, IOException      {          response.setHeader("Cache-Control", "no-cache");          response.setHeader("Pragma", "no-cache");          PrintWriter out = response.getWriter();          Date currentTime= new Date();          String message = String.format("Currently time is %tr on %tD.",currentTime, currentTime);          out.print(message);      }  } |

Above code will return the current server time in response in text form, which client code receives and prints on webpage.



**Synchronous request example**

To send synchronous ajax request, change the javascript code in index.jsp file with this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | function ajaxSyncRequest(reqURL)  {      //Creating a new XMLHttpRequest object      var xmlhttp;      if (window.XMLHttpRequest){          xmlhttp = new XMLHttpRequest(); //for IE7+, Firefox, Chrome, Opera, Safari      } else {          xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); //for IE6, IE5      }      //Create a asynchronous GET request      xmlhttp.open("GET", reqURL, false);      xmlhttp.send(null);        //Execution blocked till server send the response      if (xmlhttp.readyState == 4) {          if (xmlhttp.status == 200)          {              document.getElementById("message").innerHTML = xmlhttp.responseText;          }          else          {              alert('Something is wrong !!');          }      }  } |

You do not need to check the ready state in synchronous requests, because response will be available only when request is completed. Till the time, page will be blocked.

**Some popular ajax capable libraries**

As it is evident that ajax is very popular technology nowadays for making webpages highly interactive and user friendly. To ease the development of ajax related components, various frameworks are available for developers in market today. The good thing is that they all are free to use.

**1) JQuery**

[**JQuery**](http://jquery.com/) is probably the post popular among its alternatives. It has got its own developer community which is highly active also. A sample code for sending ajax request using jquery will be like is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | //Current request reference; can be used else where  var request;    /\* attach a submit handler to the form \*/  $("#buttonId").submit(function(event) {        // abort any pending request      if (request) {          request.abort();      }      /\* stop form from submitting normally \*/    event.preventDefault();      /\*clear result div\*/     $("#result").html('');      /\* get some values from elements on the page: \*/     var values = $(this).serialize();      /\* Send the data using post and put the results in a div \*/    request =$.ajax({        url: "ajaxRequest",        type: "post",        data: values,        success: function(){             $("#result").html('submitted successfully');        },        error:function(){            $("#result").html('there is error while submit');        }    });  }); |

**2) Prototype**

[**Prototype**](http://prototypejs.org/) is another popular framework for the same purpose. But, please beware that prototype is known to be incompatible with some other frameworks. A example code for sending ajax request using prototype will look like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | new Ajax.Request('/some\_url',   {      method:'get',      onSuccess: function(transport)      {          var response = transport.responseText || "no response text";      },      onFailure: function()      {          alert('Something went wrong...');      }  }); |

That’s all for now. I will write more posts on ajax in future. You may like to register your email id for update notifications.