**AJAX: Asynchronous JavaScript and XML**

A new technique for creating better, faster, and more **interactive web applications** with the help of **XML, HTML, CSS, and Java Script**.

* Uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
* Conventional web applications transmit information to and from the sever using synchronous requests.
* On form submits, JavaScript requests to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server. A user can continue to use the application while the client program requests information from the server in the background.
* XML is used as the format for receiving server data in any format.
* A web browser technology independent of web server software.
* Intuitive and natural user interaction. Clicking is not required, mouse movement is a enough event trigger.
* Data-driven as opposed to page-driven.
* Rich Internet Application Technology
* AJAX is the most viable Rich Internet Application (RIA) technology but AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.
* With AJAX, there is no discontinuity and get the response very quickly, but with the standard GCI, would have to wait for the response and your page also gets refreshed.

**AJAX is based on the following open standards** −

* Browser-based presentation using HTML and Cascading Style Sheets (CSS).
* Data is stored in XML format and fetched from the server.
* Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
* JavaScript to make everything happen.

**AJAX cannot work independently.**

* **JavaScript**: Glue for the whole AJAX operation.
* **DOM**: API for accessing and manipulating structured documents. Represents the structure of XML and HTML documents.
* **CSS**
* **XMLHttpRequest:** JavaScript object that performs asynchronous interaction with the server.

**A list of major browsers that support AJAX:**

* Mozilla Firefox 1.0 and above
* Netscape version 7.1 and above
* Apple Safari 1.2 and above
* Microsoft Internet Explorer 5 and above
* Konqueror
* Opera 7.6 and above

NOTE − A browser does not support AJAX simply means the browser does not support the creation of Javascript object – XMLHttpRequest object.

**Writing Browser Specific Code**

<html>

<body>

<script language = "javascript" type = "text/javascript">

<!--

//Browser Support Code

function ajaxFunction() {

var ajaxRequest; // The variable that makes Ajax possible!

try {

// Opera 8.0+, Firefox, Safari

ajaxRequest = new XMLHttpRequest();

} catch (e) {

// Internet Explorer Browsers

try {

ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");

} catch (e) {

try {

ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");

} catch (e) {

// Something went wrong

alert("Your browser broke!");

return false;

}

}

}

}

//-->

</script>

<form name = 'myForm'>

Name: <input type = 'text' name = 'username' /> <br />

Time: <input type = 'text' name = 'time' />

</form>

</body>

</html>

**Steps of AJAX Operation**

* A client event occurs.
* An XMLHttpRequest object is created.
* The XMLHttpRequest object is configured.
* The XMLHttpRequest object makes an asynchronous request to the Webserver.
* The Webserver returns the result containing XML document.
* The XMLHttpRequest object calls the callback() function and processes the result.
* The HTML DOM is updated.

Let us take these steps one by one.

**A Client Event Occurs**

* A JavaScript function is called as the result of an event.
* Example − *validateUserId()* JavaScript function is mapped as an event handler to an *onkeyup* event on input form field whose id is set to *"userid"*
* <input type = "text" size = "20" id = "userid" name = "id" onkeyup = "validateUserId();">.

**The XMLHttpRequest Object is Created**

var ajaxRequest; // The variable that makes Ajax possible!

function ajaxFunction() {

try {

// Opera 8.0+, Firefox, Safari

ajaxRequest = new XMLHttpRequest();

} catch (e) {

// Internet Explorer Browsers

try {

ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");

} catch (e) {

try {

ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");

} catch (e) {

// Something went wrong

alert("Your browser broke!");

return false;

}

}

}

}

**The XMLHttpRequest Object is Configured**

In this step, we will write a function that will be triggered by the client event and a callback function processRequest() will be registered.

function validateUserId() {

ajaxFunction();

// Here processRequest() is the callback function.

ajaxRequest.onreadystatechange = processRequest;

if (!target) target = document.getElementById("userid");

var url = "validate?id=" + escape(target.value);

ajaxRequest.open("GET", url, true);

ajaxRequest.send(null);

}

**Making Asynchronous Request to the Webserver**

Source code is available in the above piece of code. Code written in bold typeface is responsible to make a request to the webserver. This is all being done using the XMLHttpRequest object *ajaxRequest*.

function validateUserId() {

ajaxFunction();

// Here processRequest() is the callback function.

ajaxRequest.onreadystatechange = processRequest;

**if (!target) target = document.getElementById("userid");**

**var url = "validate?id = " + escape(target.value);**

**ajaxRequest.open("GET", url, true);**

**ajaxRequest.send(null);**

}

Assume you enter Zara in the userid box, then in the above request, the URL is set to "validate?id = Zara".

Webserver Returns the Result Containing XML Document

You can implement your server-side script in any language, however its logic should be as follows.

* Get a request from the client.
* Parse the input from the client.
* Do required processing.
* Send the output to the client.

If we assume that you are going to write a servlet, then here is the piece of code.

public void doGet(HttpServletRequest request,

HttpServletResponse response) throws IOException, ServletException {

String targetId = request.getParameter("id");

if ((targetId != null) && !accounts.containsKey(targetId.trim())) {

response.setContentType("text/xml");

response.setHeader("Cache-Control", "no-cache");

response.getWriter().write("<valid>true</valid>");

} else {

response.setContentType("text/xml");

response.setHeader("Cache-Control", "no-cache");

response.getWriter().write("<valid>false</valid>");

}

}

**Callback Function processRequest() is Called**

The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the *readyState* of the *XMLHttpRequest* object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

function processRequest() {

if (req.readyState == 4) {

if (req.status == 200) {

var message = ...;

...

}

**The HTML DOM is Updated**

This is the final step and in this step, your HTML page will be updated. It happens in the following way −

* JavaScript gets a reference to any element in a page using DOM API.
* The recommended way to gain a reference to an element is to call.

document.getElementById("userIdMessage"),

// where "userIdMessage" is the ID attribute

// of an element appearing in the HTML document

* JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements. Here is an example –

<script type = "text/javascript">

<!--

function setMessageUsingDOM(message) {

var userMessageElement = document.getElementById("userIdMessage");

var messageText;

if (message == "false") {

userMessageElement.style.color = "red";

messageText = "Invalid User Id";

} else {

userMessageElement.style.color = "green";

messageText = "Valid User Id";

}

var messageBody = document.createTextNode(messageText);

// if the messageBody element has been created simple

// replace it otherwise append the new element

if (userMessageElement.childNodes[0]) {

userMessageElement.replaceChild(messageBody, userMessageElement.childNodes[0]);

} else {

userMessageElement.appendChild(messageBody);

}

}

-->

</script>

<body>

<div id = "userIdMessage"><div>

</body>

XMLHttpRequest Methods

* **abort()**

Cancels the current request.

* **getAllResponseHeaders()**

Returns the complete set of HTTP headers as a string.

* **getResponseHeader( headerName )**

Returns the value of the specified HTTP header.

* **open( method, URL )**
* **open( method, URL, async )**
* **open( method, URL, async, userName )**
* **open( method, URL, async, userName, password )**

Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

* **send( content )**

Sends the request.

* **setRequestHeader( label, value )**

Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties

* **onreadystatechange**

An event handler for an event that fires at every state change.

* **readyState**

The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property −

|  |  |
| --- | --- |
| **State** | **Description** |
| 0 | The request is not initialized. |
| 1 | The request has been set up. |
| 2 | The request has been sent. |
| 3 | The request is in process. |
| 4 | The request is completed. |

**readyState = 0** After you have created the XMLHttpRequest object, but before you have called the open() method.

**readyState = 1** After you have called the open() method, but before you have called send().

**readyState = 2** After you have called send().

**readyState = 3** After the browser has established a communication with the server, but before the server has completed the response.

**readyState = 4** After the request has been completed, and the response data has been completely received from the server.

* **responseText**

Returns the response as a string.

* **responseXML**

Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.

* **status**

Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").

* **statusText**

Returns the status as a string (e.g., "Not Found" or "OK").

Create a table using the following command.

**NOTE** − We are assuming you have sufficient privilege to perform the following MySQL operations.

CREATE TABLE 'ajax\_example' (

'name' varchar(50) NOT NULL,

'age' int(11) NOT NULL,

'sex' varchar(1) NOT NULL,

'wpm' int(11) NOT NULL,

PRIMARY KEY ('name')

)

Now dump the following data into this table using the following SQL statements −

INSERT INTO 'ajax\_example' VALUES ('Jerry', 120, 'm', 20);

INSERT INTO 'ajax\_example' VALUES ('Regis', 75, 'm', 44);

INSERT INTO 'ajax\_example' VALUES ('Frank', 45, 'm', 87);

INSERT INTO 'ajax\_example' VALUES ('Jill', 22, 'f', 72);

INSERT INTO 'ajax\_example' VALUES ('Tracy', 27, 'f', 0);

INSERT INTO 'ajax\_example' VALUES ('Julie', 35, 'f', 90);

Client Side HTML File

Now let us have our client side HTML file, which is ajax.html, and it will have the following code −

<html>

<body>

<script language = "javascript" type = "text/javascript">

<!--

//Browser Support Code

function ajaxFunction() {

var ajaxRequest; // The variable that makes Ajax possible!

try {

// Opera 8.0+, Firefox, Safari

ajaxRequest = new XMLHttpRequest();

} catch (e) {

// Internet Explorer Browsers

try {

ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");

} catch (e) {

try {

ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");

} catch (e) {

// Something went wrong

alert("Your browser broke!");

return false;

}

}

}

// Create a function that will receive data

// sent from the server and will update

// div section in the same page.

ajaxRequest.onreadystatechange = function() {

if(ajaxRequest.readyState == 4) {

var ajaxDisplay = document.getElementById('ajaxDiv');

ajaxDisplay.innerHTML = ajaxRequest.responseText;

}

}

// Now get the value from user and pass it to

// server script.

var age = document.getElementById('age').value;

var wpm = document.getElementById('wpm').value;

var sex = document.getElementById('sex').value;

var queryString = "?age = " + age ;

queryString += "&wpm = " + wpm + "&sex = " + sex;

ajaxRequest.open("GET", "ajax-example.php" + queryString, true);

ajaxRequest.send(null);

}

//-->

</script>

<form name = 'myForm'>

Max Age: <input type = 'text' id = 'age' /> <br />

Max WPM: <input type = 'text' id = 'wpm' /> <br />

Sex:

<select id = 'sex'>

<option value = "m">m</option>

<option value = "f">f</option>

</select>

<input type = 'button' onclick = 'ajaxFunction()' value = 'Query MySQL'/>

</form>

<div id = 'ajaxDiv'>Your result will display here</div>

</body>

</html>

**NOTE** − The way of passing variables in the Query is according to HTTP standard and have formA.

URL?variable1 = value1;&variable2 = value2;

The above code will give you a screen as given below −

Top of Form

Max Age:  

Max WPM: 

Sex:   

Bottom of Form

Your result will display here in this section after you have made your entry.

**NOTE** − This is a dummy screen.

Server Side PHP File

Your client-side script is ready. Now, we have to write our server-side script, which will fetch age, wpm, and sex from the database and will send it back to the client. Put the following code into the file "ajax-example.php".

<?php

$dbhost = "localhost";

$dbuser = "dbusername";

$dbpass = "dbpassword";

$dbname = "dbname";

//Connect to MySQL Server

mysql\_connect($dbhost, $dbuser, $dbpass);

//Select Database

mysql\_select\_db($dbname) or die(mysql\_error());

// Retrieve data from Query String

$age = $\_GET['age'];

$sex = $\_GET['sex'];

$wpm = $\_GET['wpm'];

// Escape User Input to help prevent SQL Injection

$age = mysql\_real\_escape\_string($age);

$sex = mysql\_real\_escape\_string($sex);

$wpm = mysql\_real\_escape\_string($wpm);

//build query

$query = "SELECT \* FROM ajax\_example WHERE sex = '$sex'";

if(is\_numeric($age))

$query .= " AND age <= $age";

if(is\_numeric($wpm))

$query .= " AND wpm <= $wpm";

//Execute query

$qry\_result = mysql\_query($query) or die(mysql\_error());

//Build Result String

$display\_string = "<table>";

$display\_string .= "<tr>";

$display\_string .= "<th>Name</th>";

$display\_string .= "<th>Age</th>";

$display\_string .= "<th>Sex</th>";

$display\_string .= "<th>WPM</th>";

$display\_string .= "</tr>";

// Insert a new row in the table for each person returned

while($row = mysql\_fetch\_array($qry\_result)) {

$display\_string .= "<tr>";

$display\_string .= "<td>$row[name]</td>";

$display\_string .= "<td>$row[age]</td>";

$display\_string .= "<td>$row[sex]</td>";

$display\_string .= "<td>$row[wpm]</td>";

$display\_string .= "</tr>";

}

echo "Query: " . $query . "<br />";

$display\_string .= "</table>";

echo $display\_string;

?>

Now try by entering a valid value (e.g., 120) in *Max Age* or any other box and then click Query MySQL button.

Top of Form

Max Age:  

Max WPM: 

Sex:   

Bottom of Form

Your result will display here in this section after you have made your entry.

If you have successfully completed this lesson, then you know how to use MySQL, PHP, HTML, and Javascript in tandem to write AJAX applications.