TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

AJAX

The usual way we operate in the Web

- 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.

The usual way we operate in the Web

- 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).

Things change…

- Ajax is a means of using JavaScript to communicate with a web server without submitting a form or loading a new page.
- Ajax makes use of a built-in object, XMLHttpRequest, to perform this function.
- All modern browsers (Chrome, Firefox, Edge (and IE7+), Safari, Opera) have a built-in XMLHttpRequest object.

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:
  - A JavaScript script 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.
  - The server responds by sending the contents of a file or the output of a server side program (written, for example, in PHP).
  - When the response arrives from the server, a JavaScript function is triggered to act on the data supplied by the server.
  - This JavaScript response function typically refreshes the display using the DOM, avoiding the requirement to reload or refresh the entire page.

The Back End

- 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.
- 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 that performs an operation and sends results back to the client browser.

The XMLHttpRequest object

- XMLHttpRequest was first introduced by Microsoft in Internet Explorer 5.0 as an ActiveX control. However, in IE7 and other browsers XMLHttpRequest is a native JavaScript object.

```
let request;
if (window.XMLHttpRequest) {
    //Firefox, Opera, IE7, and other browsers will use
the native object
    request = new XMLHttpRequest();
} else {
    //IE 5 and 6 will use the ActiveX control
    request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

The XMLHttpRequest object (cont.)

```
function getXMLHttpRequest()
/*    This function attempts to get an Ajax request object by trying
      a few different methods for different browsers.  */
{
     let request;
     if (window.XMLHttpRequest) {
       //Firefox, Opera, IE7, and other browsers will use the native object
        request = new XMLHttpRequest();
     } else {
        //IE 5 and 6 will use the ActiveX control
        request = new ActiveXObject("Microsoft.XMLHTTP");
     }
     return request;
}
```

The XMLHttpRequest object (cont.)

- As with any object in JavaScript (and other programming languages), the XMLHttpRequest object contains various properties and methods.

- 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.

XMLHttpRequest object methods

Method	Description
• open('method', 'URL', asyn)	Specifies the HTTP method to be used (GET, POST, PUT, etc. 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 method is used to set up the request, and the send method starts the request by sending it to the server

A general skeleton for an Ajax application

```
<scrip>
      let  ajaxRequest = getXMLHttpRequest();
      if (ajaxRequest) {
            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).
- 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');
?>
```

```
<!DOCTYPE html>
<head>
    <script>
    let ajaxRequest;

    function getXMLHttpRequest(){
        let request;
        if (window.XMLHttpRequest) {
          //Firefox, Opera, IE7, and other browsers will use the native
        object
           request = new XMLHttpRequest();
        } else {
           //IE 5 and 6 will use the ActiveX control
           request = new ActiveXObject("Microsoft.XMLHTTP");
        }
        return request;
   }
```

```
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;      }
   let myURL = "telltime.php";
   let 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 page</h2>
    <form>
       <input type="button" value="Get Server Time”
             onclick="getServerTime();" />
    </form>
    <div id="showtime" class="displaybox"></div>
</body>
</html>
```

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.

Sending text back the server

- The response stored in XMLHttpRequest.responseText from the server can be any text that JavaScript is capable of processing as a string.
- Thus, you can send back a simple text string as the first example did, or you could send a string with HTML tags embedded in it.
- You could even send back a string that has JavaScript code it in and execute it using the JavaScript eval() method.
- Recall, however, that the responseText property is a read-only variable, so if you're going to alter it you must first copy it to another variable.

The other PHP scripts for the time examples

Here's the script with a simple HTML tag in it.

```
<?php
  echo '<span style="color: red;">' . date('H:i:s') . "</span>";
?>
```

The output with a table.

```
<?php
    $tr = '<tr style="border: 2px solid;">';
    $td = '<td style="border: 2px solid">';
    $table = '<table style="border: 2px solid; margin: auto;">';
    $table .= $tr . $td . date('j M Y') . '</td></tr>';
    $table .= $tr . $td . date('H:i:s') . '</td></tr>';
    $table .= '</table>';
    echo $table;
?>
```

XML: a (very) brief intro

- XML, the eXtensible Markup Language, is used in many ways, the most relevant to us being the transfer of structured information.

- XML and HTML look similar in many ways and this is because both are based on SGML, the Standard Generalized Markup Language established by the International Organization for Standards (ISO).

- Like HTML, XML uses tags to denote information but is not limited to the types of tags that occur in HTML. Tags can be essentially anything a user likes and are used to define the type of data present in the document.

XML: a (very) brief intro (cont.)

Here's an example:

```
<library>
    <book>
        <title>Programming PHP</title>
        <author>Rasmus Lerdorf</author>
        <author>Kevin Tatroe</author>
        <author>Peter MacIntyre</author>
        <chapter number="1">Introduction to PHP</chapter>
        <chapter number="2">Language Basics</chapter>
        . . .
        <pages>521</pages>
    </book>
  . . .
</library>
```

Accessing an XML document in JavaScript

```
<script>
   var text, parser, xmlDoc;
   text = "<bookstore><book>" +
          "<title>Everyday Italian</title>" +
          "<author>Giada De Laurentiis</author>" +
          "<year>2005</year>" +
          "</book></bookstore>";
   parser = new DOMParser();
   xmlDoc = parser.parseFromString(text,"text/xml");

   document.getElementById("demo").innerHTML =
   xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;

</script>
```

The “time” example using XML

- The first change is to make a new PHP script that returns an XML document to the browser.

```
<?php
   header('Content-Type: text/xml');
   echo "<?xml version=\"1.0\" ?>\n";
   echo "<clock><timenow>" .
     date('H:i:s') .
       "</timenow></clock>";
 ?>
```

- After that change (and inserting the new script name into the HTML code), we need to alter the ajaxResponse function to parse the XML document.
- Note that we need not explicitly create an object to hold the XML document, but that responseXML (as a property of XMLHttpRequest) is already such an object.

The new Ajax response function

```
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
          {   let timeValue = ajaxRequest.responseXML.
                    getElementsByTagName("timenow")[0];
               document.getElementById("showtime").innerHTML =
                    timeValue.childNodes[0].nodeValue; }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
```

view the output page

A second example (live search)

- We'll build a "live search" function. When you typically use a form, you must submit the data to the server and wait for it to return the results. Here we want to consider a case where you get (partial) results as you enter data into the input field, and that these results are updated (almost) instantly.
- We use PHP again for the backend. First consider the case where the possible results are a list of names, stored in a PHP array. As you type data into the input field, it's matched against this list of names, and the (partial) matches are returned and displayed on screen.
- Later, we will see the same type of application, but using PHP to search through the names stored in a database.

The PHP backend

```
<?php
header("Content-Type: text/xml");
$people = array( "Abraham Lincoln", "Martin Luther King", "Jimi Hendrix", "John Wayne",
  "John Carpenter", "Elizabeth Shue", "Benny Hill", "Lou Costello", "Bud  Abbott",
  "Albert Einstein", "Rich Hall", "Anthony Soprano", "Michelle Rodriguez",      "Carmen
  Miranda", "Sandra Bullock", "Moe Howard", "Ulysses S. Grant", "Stephen  Fry", "Kurt
  Vonnegut", "Yosemite Sam", "Ed Norton", "George Armstrong Custer",      "Alice
  Kramden", "Evangeline Lilly", "Harlan Ellison");
$query = $_GET['query'];
echo "<?xml version=\"1.0\"?>\n";
echo "<names>\n";
foreach($people as $v)
   {
      if (stristr ($v, $query))
         echo "<name>$v</name>\n";
   }
echo "</names>';
?>
```

The HTML layout (no JavaScript yet)

```
<html>
<body>
  <h1>Ajax Demonstration of Live Search</h1>
  <p> Search for: <input type="text" id="searchstring"/></p>
  <div id="results">
    <ul id="list">
        <li>Results will be displayed here.</li>
    </ul>
  </div>

  <script>
     let obj=document.getElementById("searchstring");
     obj.onkeydown = startSearch;
  </script>
</body>
</html>
```

view the output page

The JavaScript functions

- We obviously need the function for creating a new XMLHttpRequest object, which we will store in a global variable called “ajaxRequest”.
- The search will be handled by setting up a Timeout event, based on entering text in the input field (using the “onkeydown” attribute).

```
let t;  //  public variable for the timeout
function startSearch()
{
    if (t) window.clearTimeout(t);
    t = window.setTimeout("liveSearch()",200);
}
```

The JavaScript functions (cont.)

```
let ajaxRequest;
function liveSearch()
{
   ajaxRequest = getXMLHttpRequest();
   if (!ajaxRequest)  alert("Request error!");
   let myURL = "search.php";
   let query = document.getElementById("searchstring").value;
   myURL = myURL + "?query=" + query;
   ajaxRequest.onreadystatechange = ajaxResponse;
   ajaxRequest.open("GET", myURL);
   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
          {   displaySearchResults();   }
     else {
          alert("Request failed: " + ajaxRequest.statusText);
          }
     }
}
```

The JavaScript functions (cont.)

```
function displaySearchResults()
{  let i, n, li, t;
   let ul = document.getElementById("list");
   let div = document.getElementById("results");

   div.removeChild(ul);  //  delete the old search results
   ul = document.createElement("UL");  //  create a new list container
   ul.id="list";

   let names=ajaxRequest.responseXML.getElementsByTagName("name");
   for (i = 0; i < names.length; i++)
       {
         li = document.createElement("LI");  //  create a new list element
         n = names[i].firstChild.nodeValue;
         t = document.createTextNode(n);
         li.appendChild(t);
         ul.appendChild(li);
       }
   if (names.length == 0) { // if no results are found, say so
         li = document.createElement("LI");
         li.appendChild(document.createTextNode("No results."));
         ul.appendChild(li);
      }
   div.appendChild(ul);  // display the new list
}
```

The finished product

- We add all of the functions (and the two global variables) to the header script section

view the output page

Using a database for the live search

- Instead of storing the names in an array, we could alter the PHP script to search through a mySQL database for matches.
- The JavaScript need not be changed (except for the name of the script to call).
- As before, the PHP script will return the names as an XML document, using methods for a case-insensitive search on the query string.

Some cautions

- As with any JavaScript element, you can't (or shouldn't) rely upon a user's browser being able to execute JavaScript (some people turn it off on their browsers).

- Ajax can possibly introduce strange behavior, like you use some "hidden" elements in your page (generated by Ajax), then they will likely not show up in a form that search engines will recognize.