Chapter 5.1 - AJAX Introduction

INTRODUCTION

- AJAX is about updating parts of a web page, without reloading the whole page.
- AJAX = Asynchronous JavaScript and XML.
- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.
- This means that it is possible to update parts of a web page, without reloading the whole page.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

INTRODUCTION

- The processing of web page formerly was only in the server-side, using web services or PHP scripts, before the whole page was sent within the network.
- Ajax allows to perform processing in the client computer (in JavaScript) with data taken from the server.
- "Asynchronous", means that the response of the server will be processed only when available; not having to wait and to freeze the display of the page.
- Examples of applications using AJAX:
  - **Google Maps (since 2005), Gmail, YouTube, and Facebook tabs**.

AJAX Components

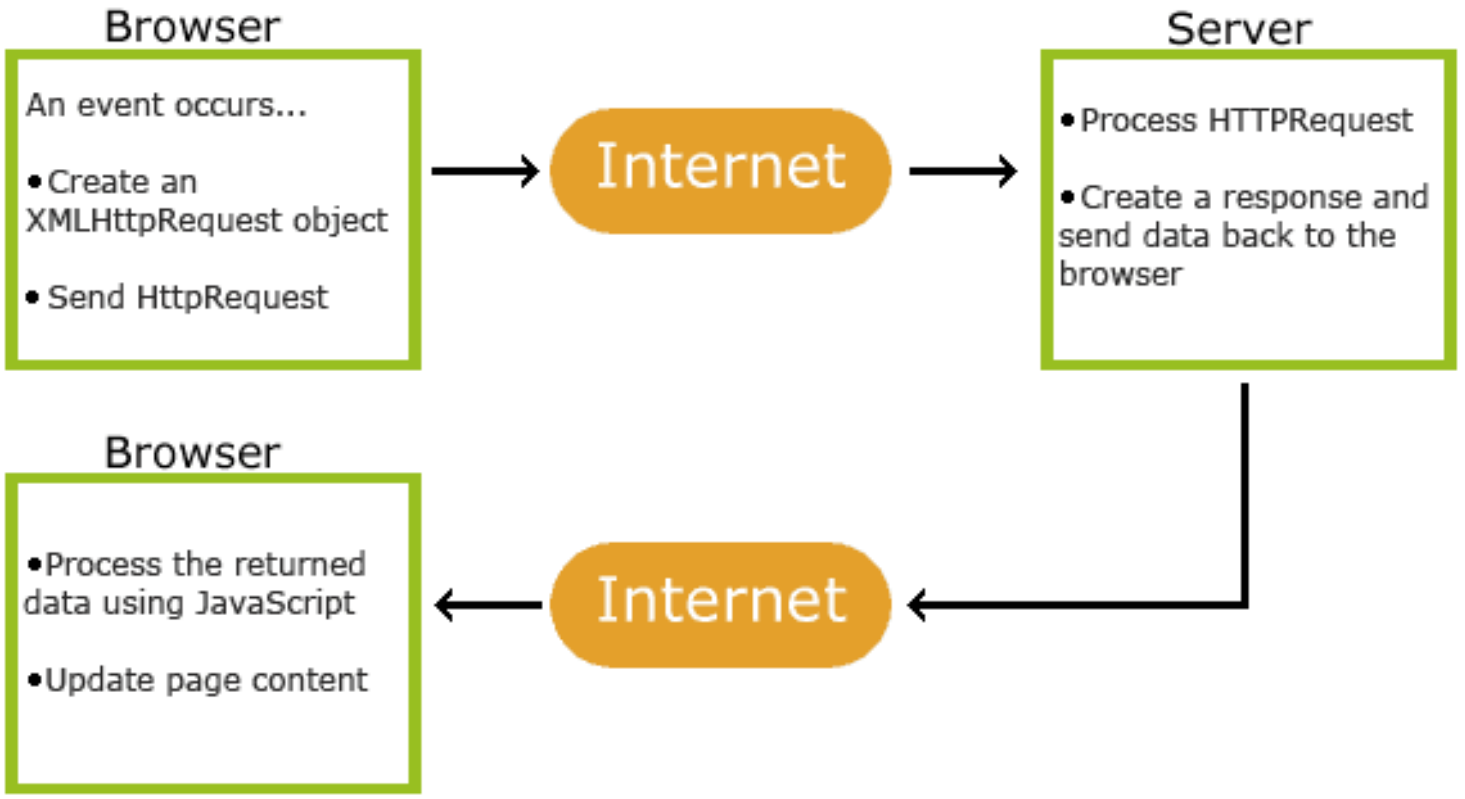
- **XHTML and CSS**
- Ajax applies these familiar Web standards for styling the look and feel of a page and to markup those areas on a page that will be targeted for data updates.
- **DOM (document object model)**
- Ajax uses the DOM to manipulate dynamic page views for data and to walkthrough documents to “cherrypick” data.
- The DOM enables certain pieces of an Ajax page to be transformed and updated with data.

AJAX Components

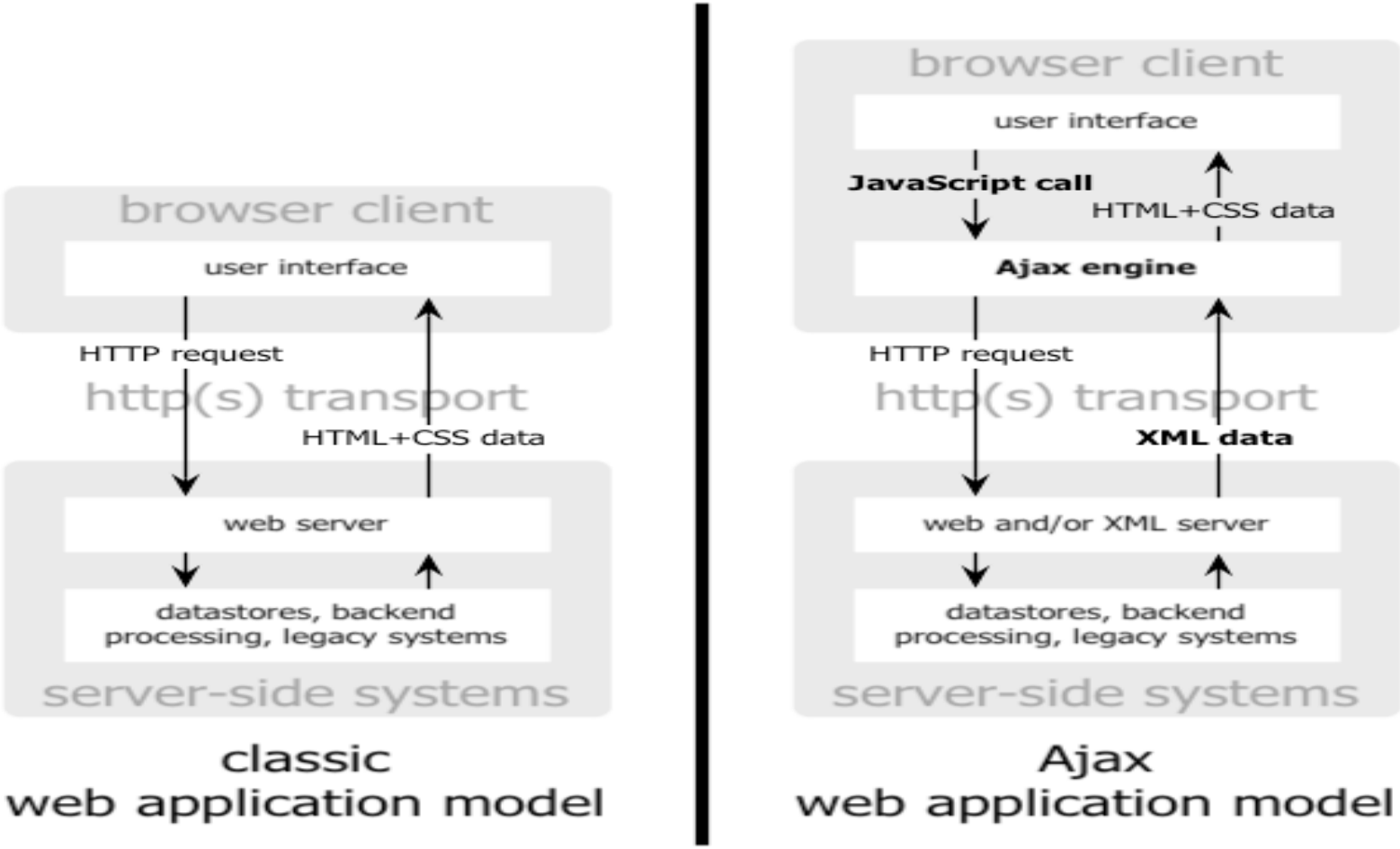
- **XML, JSON (JavaScript Object Notation), HTML, or plain text**
- Ajax can use any of these standards to provide structure to the data it passes to and from a page.
- **XMLHttpRequest object**
- The heavy lifter for Ajax: It's a JavaScript object embedded in most modern browsers that sets up data request/response pipelines between client and server.
- **JavaScript**
- Lightweight programming language that Ajax uses for instructions to bind all of the components together.

How AJAX Works?

- AJAX is based on internet standards, and uses a combination of:
  - XMLHttpRequest object (to exchange data asynchronously with a server)
  - JavaScript/DOM (to display/interact with the information)
  - CSS (to style the data)
  - XML (often used as the format for transferring data)



How AJAX Works?



Google Suggest

- AJAX was made popular in 2005 by Google, with Google Suggest.
- Google Suggest is using AJAX to create a very dynamic web interface:
  - When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

- N.B. **Google Suggest** is the name of **Google's** auto-complete function. If a user enters a letter or a word in **Google's** search field, they are automatically shown associated terms in a dropdown menu.
  - These **suggestions** are generated based on the most frequently searched terms

What's so special?

- Initially If you wanted to get any information from a database on the server, or send user information to a server-side script like PHP, you had to make an HTML form to GET or POST data to the server.
- The user would then have to click "Submit", wait for the server to respond, then a new page would load with the results.
- I'm sure we have all gotten slightly annoyed when having to wait for especially slow websites!

What's so special?

- Ajax attempts to remedy this problem by letting your JavaScript communicate directly with the server, using a special JavaScript object **XMLHttpRequest**.
- All modern browsers support the XMLHttpRequest object (IE5 and IE6 use an ActiveXObject).
- With this object, your JavaScript can get information from the server without having to load a new page!

AJAX - CREATING AN HTML FORM

- To keep this Ajax easy to understand, we are going to create an HTML form that has two text fields: name and time.
- The name field will be filled in by the user, while the **time field** will be filled in using **Ajax**.
- order1.html, order2.html
- That's the great thing about Ajax, you do not need a form submit button to send the user's data to the server.
- Order3.htm, ServerTime.php

Ajax - Browser Support

- It would be nice if all the web browsers required the same JavaScript code to use Ajax, but life isn't fair and you've got your work cut out for you!
- Not only will you know how to make XMLHttpRequest - important Ajax object, also how to make it compatible with all the popular browsers:
  - Internet Explorer, Opera, Firefox, and Safari.
- We try three times to make our XMLHttpRequest object. Our first attempt:

  ajaxRequest = new XMLHttpRequest();
  - is for the Opera 8.0+, Firefox and Safari browsers.

AJAX - TRY/CATCH BLOCKS

- We are going to "try" three different ways to make a new XMLHttpRequest object.
- Every time we fail and get an error, we will catch the error and try the next a different command.

Ajax - Try/Catch Blocks

- If that fails we try two more times to make the correct object for an Internet Explorer browser with:

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

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

- If that doesn't work, then they are using a very outdated browser that doesn't support *XMLHttpRequest*, which also means it doesn't support Ajax.

AJAX - TRY/CATCH BLOCKS

- **Conclusion:**
  - To handle all modern browsers, including IE5 and IE6, check if the browser supports the XMLHttpRequest object.
  - If it does, create an XMLHttpRequest object, if not, create an ActiveXObject.
  - The code in this lesson was quite complex, but the good thing is that you can just copy and paste this code and don't really have to understand it.

AJAX - Send a Request To a Server

- **Send a Request To a Server**
- To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. *method*: the type of request: GET or POST *url*: the location of the file on the server *async*: true (asynchronous) or false (synchronous)
send(*string*)	Sends the request off to the server. *string*: Only used for POST requests

GET OR POST?

- GET is simpler and faster than POST, and can be used in most cases.
- However, always use POST requests when:
  - A cached file is not an option (update a file or database on the server)
  - Sending a large amount of data to the server (POST has no size limitations)
  - Sending user input (which can contain unknown characters), POST is more robust and secure than GET

The url - A File On a Server

- The url parameter of the open() method, is an address to a file on a server:

  xmlhttp.open("GET",“serverTime.php",true);

- The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php

Asynchronous - True or False?

- AJAX stands for Asynchronous JavaScript and XML, and for the XMLHttpRequest object to behave as AJAX, the async parameter of the open() method has to be set to true:

  xmlhttp.open("GET"," serverTime.php", true);
- Many of the tasks performed on the server are very time consuming.
- Before AJAX, this operation could cause the application to hang or stop.
- With AJAX, the JavaScript does not have to wait for the server response, but can instead:
  - execute other scripts while waiting for server response
  - deal with the response when the response is ready

Asynchronous - True or False?

- **Async=true**
- When using async =true, specify a function to execute when the response is ready in the onreadystatechange event.
- **Async=false**

  xmlhttp.open("GET", "ajax_info.txt", false);
- Using async =false is not recommended, but for a few small requests this can be ok.
- Remember that the JavaScript will NOT continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.
- **Note:** When you use async = false, do NOT write an onreadystatechange function - just put the code after the send() statement

AJAX - XMLHTTPREQUEST OBJECT

- **Ajax - onreadystatechange Property**
- The **XMLHttpRequest** object is useful if you want to send and retrieve data from a server without reloading the current page used for retrieving XML files or pure text content from a server.
- The *XMLHttpRequest* object has a special property called *onreadystatechange*.
- *onreadystatechange* stores the function that will process the response from the server.
- **Syntax:**
- // Create a function that will receive data sent from the server
  ajaxRequest.onreadystatechange = function(){

  // We still need to write some code here

  }

AJAX - XMLHTTPREQUEST OBJECT

- To send a request, create an instance of the **XMLHttpRequest** object first, then initialize a request with the open method, specify the necessary request headers with the setRequestHeader method and finally send the request with the send method.
- The data transfer can be handled synchronously or asynchronously (see the open method).
  - In case of **synchronous** data transfers, the send method does not return until the response arrives.
  - Do not use this type of data transfer if you do not want to keep the user waiting.
  - If you use **asynchronous** data transfer, register the onreadystatechange event before you send the request.
  - The onreadystatechange event fires every time when the state of the request changes.

AJAX - READYSTATE PROPERTY

- This property, *onreadystatechange*, stores a function.
- As the name sort of implies, every time the "ready state" changes this function will be executed.
- What is this "ready state" and is it any use to us?
- The *XMLHttpRequest* object has another property called *readyState*.
- This is where the status of our server's *response* is stored.

AJAX - READYSTATE PROPERTY

- The *response* can be processing, downloading or completed. Each time the *readyState* changes then our *onreadystatechange* function executes.
- The only *readyState* that we are going to care about is when our response is *complete* and we can get our hands on that information.
- So let's add an If Statement to our function to check if the response is *complete*.
- **Note:** When the property *readyState* is 4 that means the response is complete and we can get our data.

AJAX - READYSTATE PROPERTY

- **JavaScript Code:**
- // Create a function that will receive data sent from the server

```
ajaxRequest.onreadystatechange = function(){
        if(ajaxRequest.readyState == 4){
        // Get the data from the server's response
        }
}
```

- Now that we know how to check if the response is complete, we can access the property that stores the server's response, *responseText*.
- For simple Ajax applications you can retrieve the server's response by using the *responseText* property.

AJAX - RESPONSETEXT PROPERTY

- Using a little bit of JavaScript and HTML forms we can change our text box to equal *responseText*.
- The HTML input we want to change is the "time" text box, as in our first example.
- Here's a little refresher on how to access form inputs with JavaScript:

  document.FormName.InputName.value
- Our form's name is "myForm" and the text box is "time".
- Below is the code that will set our "time" text box to the server's time.
- //Create a function that will receive data sent from the server

```
ajaxRequest.onreadystatechange = function(){
        if(ajaxRequest.readyState == 4){
        document.myForm.time.value = ajaxRequest.responseText;
        }
}
```

The onreadystatechange event

- The onreadystatechange event is triggered every time the readyState changes.
- The readyState property holds the status of the XMLHttpRequest.
- Three important properties of the XMLHttpRequest object:

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

The onreadystatechange event

- In the onreadystatechange event, we specify what will happen when the server response is ready to be processed.
- When readyState is 4 and status is 200, the response is ready.
- The status of the response, xhr.status, is (generally) used to determine whether the request was successful or not.
- xhr.readyState is simply used to determine the state of the request.
- Status indicates if server response is ok.
  In general words, when you got an status
  - **500 - 599:** the server had an error
  - **400 - 499:** this is a client error (Ex: 404 page not found)
  - **300 - 399:** then exists a redirect
  - **200 - 299:** then it is correct and
  - **100 - 199:** means information message

Server Response

- To get the response from a server, use the responseText or responseXML property of the XMLHttpRequest object.

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

- **The responseText Property**
- If the response from the server is not XML, use the responseText property.
- The responseText property returns the response as a string.
- **The responseXML Property**
- If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property.

Ajax - Sending a Request for Information

- Now that our *onreadystatechange* property has a proper response-handling function, we can send our request.
- Sending a request is comprised of two steps:
  1. Specify the URL of server-side script that will be used in our Ajax application.
  2. Use the send function to send off the request.
- Our simple PHP script, that we have yet to write, will be called "serverTime.php", so we can already do step 1.
- The URL is set using the *open* method, which takes three arguments.

AJAX - SENDING A REQUEST FOR INFORMATION

- Assuming that the HTML and PHP files are in the same directory, the code would be:
- **JavaScript Code:**
- // Create a function that will receive data sent from the server

```
ajaxRequest.onreadystatechange = function(){
        if(ajaxRequest.readyState == 4){
         document.myForm.time.value = ajaxRequest.responseText;
        }
}
ajaxRequest.open("GET", "serverTime.php", true);
```

Ajax - Sending a Request for Information

- With all of our JavaScript setup work complete, we can then use the *send* method to send our request to the server.
- **JavaScript Code:**
- // Create a function that will receive data sent from the server

```
ajaxRequest.onreadystatechange = function(){
if(ajaxRequest.readyState == 4){
        document.myForm.time.value = ajaxRequest.responseText;
        }
}
ajaxRequest.open("GET", "serverTime.php", true);
ajaxRequest.send(null);
```

Ajax - Finishing up "order.html"

- Before we plug in our freshly written JavaScript code into the "order.html" file, we need some way for the visitor to run our Ajax function.
- Using the *onChange* attribute, we can make it so our function is called whenever the user makes a change to the "username" text box.
- **JavaScript Code:**

```
<input type='text' onChange="ajaxFunction();“ name='username' />
<br />
```

AJAX - MYSQL DATABASE

- Our "order.html" file and PHP script will have to be updated and we also need to make a new database.
- **Create the MySQL Table**
- It is easy to access information from a database using Ajax,
- Create the table *ajax_example* and insert sample data rows. The table has four columns:
  - **ae_name** - The name of the person
  - **ae_age** - Person's age
  - **ae_sex** - The gender of the person
  - **ae_wpm** - The words per minute that person can type

Update order.html

- We want to be able to build queries from our HTML file, so there are a few form elements that will need to be added.
- The three inputs we are going to implement are:
  - Maximum Age (Text Input) - Let the user select the maximum age to be returned.
  - Maximum WPM (Text Input) - Let the user select the maximum wpm to be returned.
  - Gender (Select Input) - Let the user select the gender of a valid person.