

Chapter 16: Ajax

CS 80: Internet Programming

Instructor: Mark Edmonds

What is Ajax?

- **Asynchronous Javascript And XML**
 - Misleading name! Originally developed for XML, but you can transfer plain text or JSON with it as well.

What is Ajax?

- The idea: we load data as the user is viewing and interacting with the page; Javascript communicates with the server in the background to update the page.
- The effect: web applications that behave much more similarly to desktop applications
- The benefit: web applications don't have to reload a page to get new data. This can be incredibly simple data or complex data to enable drastic changes to the page

What is Ajax?

- A side note about practicality:
 - We will learn about Ajax, but running Ajax requires a webserver to respond to requests. We will eventually cover web servers which will enable us to run our own basic Ajax examples

Live Examples

- http://test.deitel.com/iw3http5/ch16/fig16_05/SwitchContent.html
- http://test.deitel.com/iw3http5/ch16/fig16_08/PullImagesOntoPage.html
- http://test.deitel.com/iw3http5/ch16/fig16_09-10/AddressBook.html
- <http://kengeddes.com/cs80/examples/ajax.html>
 - <http://kengeddes.com/cs80/examples/ajax.js>
- <http://kengeddes.com/cs80/examples/ajax-b.html>
 - <http://kengeddes.com/cs80/examples/ajax-b.js>
- <http://javascript.cs.lmu.edu/playground/ajax/>

Ajax Basics

- "Raw" Ajax uses Javascript directly to send asynchronous requests to the server, and updates the webpage using DOM
- There are a lot of cross-browser, cross-operating system considerations you have to handle when using raw Ajax
 - Instead, jQuery, ASP.NET Ajax, etc can provide easy-to-use cross-platform support

Ajax basics

- `XMLHttpRequest` - object that manages the interaction between the server and the webpage (without reloading)
 - Abbreviated `XHR`

Traditional Webpage

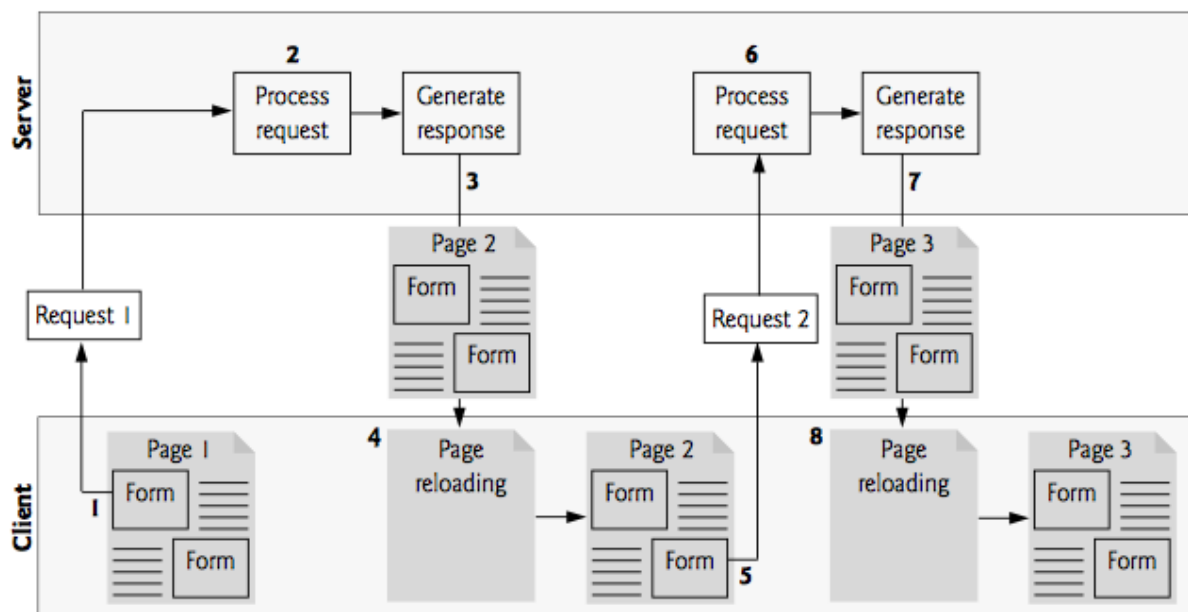


Figure 1: Traditional webpage

Ajax Webpage

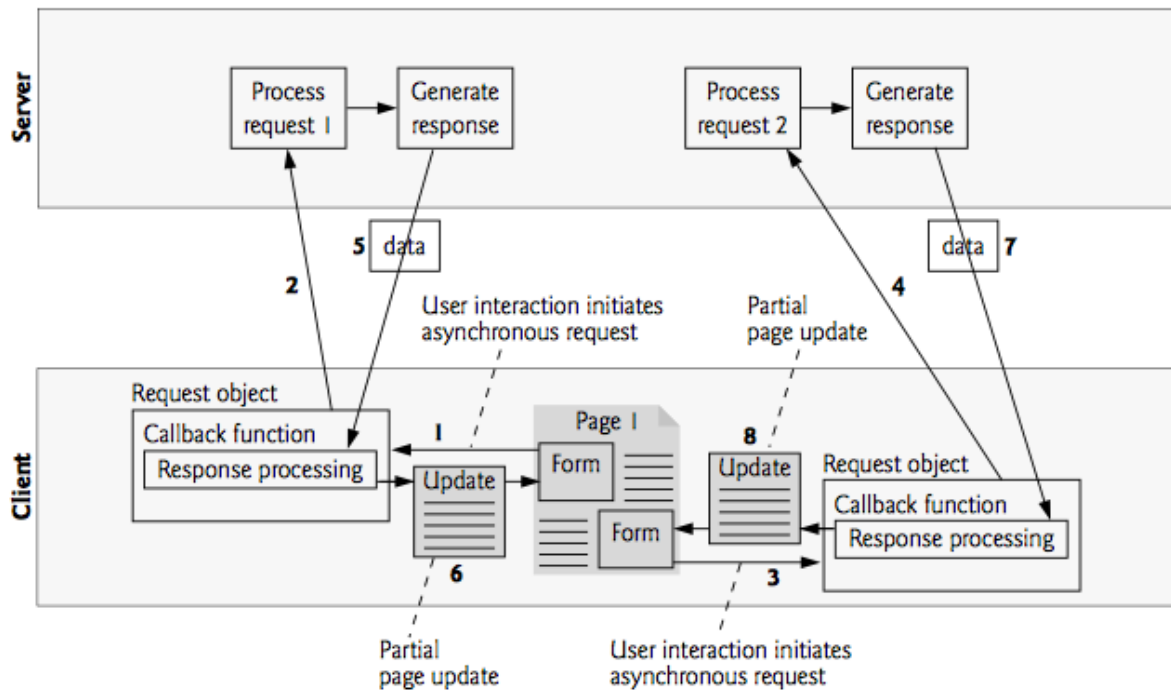


Figure 2: Ajax webpage

Ajax Steps

1. Client creates XHR object
2. XHR sends a request to the server and waits for a response
 - These requests are made **asynchronously**, which means the user can keep interacting with the web page while the request finishes
3. Many things can happen here, client could interact more with the webpage, create new XHR requests, etc
4. Server replies to the request in step 2
5. Client executes a callback function, which processes the data recieved in step 4 (could modify the DOM, etc). Commonly a partial page update

Ajax Basics

- This process is asynchronous, so tracking the exact execution can be difficult. Lots of things could happen during step 3 above

Conceptual Example - Validating a form

- We can accomplish a lot of this using the new HTML5 forms, but provides more generic form support
- We can validate any type of form data (e.g. zip code, etc), asynchronously, as the user fills in the form.
- Enables a more powerful form model, you see this all the time online!

Example - SwitchContent

- The basic concept: when the user puts their mouse over a textbook cover, we load that textbook's corresponding information

Example: SwitchContent.html

```
1 <!DOCTYPE html>
2 <!-- Fig. 16.5: SwitchContent.html -->
3 <!-- Asynchronously display content without reloading the page. -->
4 <html>
5
6 <head>
7   <meta charset="utf-8">
8   <style type="text/css">
9     .box {
10       border: 1px solid black;
11       padding: 10px
12     }
13   </style>
14   <title>Switch Content Asynchronously</title>
15   <script>
16     var asyncRequest; // variable to hold XMLHttpRequest object
17
18     // set up event handlers
19     function registerListeners() {
20       var img;
21       img = document.getElementById("cpphttp");
22       img.addEventListener("mouseover",
23         function() {
24           getContent("cpphttp8.html");
25         }
26       );
27     }
28   </script>
29 </head>
30 <body>
31   <img alt="C++ HTTP logo" id="cpphttp" />
32 </body>
33 </html>
```

```
26     img.addEventListener("mouseout", clearContent);
27     img = document.getElementById("iw3http");
28     img.addEventListener("mouseover",
29         function() {
30             getContent("iw3http.html");
31         });
32     img.addEventListener("mouseout", clearContent);
33     img = document.getElementById("jhhttp");
34     img.addEventListener("mouseover",
35         function() {
36             getContent("jhhttp.html");
37         });
38     img.addEventListener("mouseout", clearContent);
39     img = document.getElementById("vbhttp");
40     img.addEventListener("mouseover",
41         function() {
42             getContent("vbhttp.html");
43         });
44     img.addEventListener("mouseout", clearContent);
45     img = document.getElementById("vcshttp");
46     img.addEventListener("mouseover",
47         function() {
48             getContent("vcshttp.html");
49         });
50     img.addEventListener("mouseout", clearContent);
51     img = document.getElementById("javaftp");
52     img.addEventListener("mouseover",
53         function() {
54             getContent("javaftp.html");
55         });
56     img.addEventListener("mouseout", clearContent);
57 } // end function registerListeners
58
59 // set up and send the asynchronous request.
60 function getContent(url) {
61     // attempt to create XMLHttpRequest object and make the request
62     try {
63         asyncRequest = new XMLHttpRequest(); // create request object
64         // register event handler
65         asyncRequest.addEventListener(
66             "readystatechange", stateChange);
67         asyncRequest.open("GET", url, true); // prepare the request
68         asyncRequest.send(null); // send the request
```

```
69     } // end try
70     catch (exception) {
71         alert("Request failed.");
72     } // end catch
73 } // end function getContent
74
75 // displays the response data on the page
76 function stateChange() {
77     if (asyncRequest.readyState == 4 && asyncRequest.status == 200) {
78         document.getElementById("contentArea").innerHTML =
79             asyncRequest.responseText; // places text in contentArea
80     } // end if
81 } // end function stateChange
82
83 // clear the content of the box
84 function clearContent() {
85     document.getElementById("contentArea").innerHTML = "";
86 } // end function clearContent
87 window.addEventListener("load", registerListeners);
88 </script>
89 </head>
90
91 <body>
92     <h1>Mouse over a book for more information.</h1>
93     
95     
97     
98     <img id="vbhttp" alt="Visual Basic 2010 How to Program book cover" src
99         ="vb2010http.jpg">
100     
102     
104     <div class="box" id="contentArea"></div>
105 </body>
106 </html>
```

Example - SwitchContent

- What's doing all the Ajax heavy lifting?
 - `getContent` and `stateChange`

Pelimaries: Exceptions

- Exceptions indicate an error happened during data processes, but allow the program to continue running **if** the error is "handled"
- We refer to "handling" an error as **catching** an exception
- We refer to indicating an error occurred as **throwing** an exception

Pelimaries: Exceptions

- When we want to catch an exception, we acknowledge an error by happen by wrapping the relevant portion of code in a **try...catch** block
 - We put code that might cause the exception in the **try** portion
 - We put error-recovery code in the **catch** block
 - The try block will always execute (that's the code we are trying to run)
 - The catch block will only run if an exception is thrown

Pelimaries: Exceptions

- Syntax

```
1 // syntax for trying a block of code and catching an exception
2 try {
3     // code that might throw an exception
4 } catch (exception) {
5     // error recovery code
6 }
```

getContent

1. Creates a raw Ajax object
2. Registers the function `stateChange` as the callback functino for the `readystatechange` event
 - The `readystatechange` event is triggered when the value of of the XHR's `readyState` property is changed

- `readyState` can be 5 values: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState>
- A related property, `status` contains the HTTP status code of the HTTP request (200 = success)

getContent

3. Opens the url and specifies the HTTP request with the GET method, and `true` says to do this operation asynchronously
 - Basically creates the HTTP request
4. Send the HTTP request

stateChange

- The conditional statement makes sure that the async request is completed.
 - Question: when will the `stateChange` function get called? How many times will it get called?
- Body of the state change processes the data from the request.

Running SwitchContent

- If you want to run this example, download the files from the ch16 examples
- But this isn't enough, we need an actual webserver to respond to the Ajax request
- We can start a simple webserver (using any python console) with `python -m SimpleHTTPServer` from the folder with our examples downloaded
- Then navigate to `http://localhost:8000/SwitchContent.html` in your web browser

Ajax Events and Objects

Event or Property	Description
readystatechange	Register a listener for this event to specify the <i>callback</i> function—the event handler that gets called when the server responds.
readyState	Keeps track of the request's progress. It's usually used in the callback function to determine when the code that processes the response should be launched. The <code>readyState</code> value 0 signifies that the request is uninitialized; 1 that the request is loading; 2 that the request has been loaded; 3 that data is actively being sent from the server; and 4 that the request has been completed.
responseText	Text that's returned to the client by the server.
responseXML	If the server's response is in XML format, this property contains the XML document; otherwise, it's empty. It can be used like a document object in JavaScript, which makes it useful for receiving complex data (e.g., populating a table).
status	HTTP status code of the request. A status of 200 means that request was <i>successful</i> . A status of 404 means that the requested resource was <i>not found</i> . A status of 500 denotes that there was an <i>error</i> while the server was processing the request. For a complete status reference, visit www.w3.org/Protocols/rfc2616/rfc2616-sec10.html .
statusText	Additional information on the request's status. It's often used to display the error to the user when the request fails.

Fig. 16.6 | XMLHttpRequest object event and properties.

Figure 3: Ajax events and objects

Ajax Methods

Method	Description
open	Initializes the request and has two <i>mandatory</i> parameters—method and URL. The method parameter specifies the purpose of the request—typically GET or POST. The URL parameter specifies the address of the file on the server that will generate the response. A third optional Boolean parameter specifies whether the request is <i>asynchronous</i> —it's set to true by default.
send	Sends the request to the server. It has one optional parameter, data, which specifies the <i>data to be POSTed to the server</i> —it's set to null by default.
setRequestHeader	Alters the request header. The two parameters specify the header and its new value. It's often used to set the content-type field.
getResponseHeader	Returns the header data that precedes the response body. It takes one parameter, the name of the header to retrieve. This call is often used to <i>determine the response's type</i> , to parse the response correctly.
getAllResponseHeaders	Returns an array that contains all the headers that precede the response body.
abort	Cancels the current request.

Fig. 16.7 | XMLHttpRequest object methods.

Figure 4: Ajax methods

Ajax, XML, and DOM

- When XHR receives XML data, it is stored as an XML DOM object (tree)
- This is best explained with the following example

Example: PullImagesOntoPage.html

```
1 <!DOCTYPE html>
2 <!-- Fig. 16.8: PullImagesOntoPage.html -->
3 <!-- Image catalog that uses Ajax to request XML data asynchronously.
   -->
4 <html>
5
```

```
6 <head>
7   <meta charset="utf-8">
8   <title> Pulling Images onto the Page </title>
9   <style type="text/css">
10    li {
11      display: inline-block;
12      padding: 4px;
13      width: 120px;
14    }
15
16    img {
17      border: 1px solid black
18    }
19  </style>
20  <script>
21    var asyncRequest; // variable to hold XMLHttpRequest object
22
23    // set up and send the asynchronous request to get the XML file
24    function getImages(url) {
25      // attempt to create XMLHttpRequest object and make the request
26      try {
27        asyncRequest = new XMLHttpRequest(); // create request object
28        // register event handler
29        asyncRequest.addEventListener(
30          "readystatechange", processResponse, false);
31        asyncRequest.open("GET", url, true); // prepare the request
32        asyncRequest.send(null); // send the request
33      } // end try
34      catch (exception) {
35        alert('Request Failed');
36      } // end catch
37    } // end function getImages
38
39    // parses the XML response; dynamically creates an unordered list
40    // and
41    // populates it with the response data; displays the list on the
42    // page
43    function processResponse() {
44      // if request completed successfully and responseXML is non-null
45      if (asyncRequest.readyState == 4 && asyncRequest.status == 200 &&
46        asyncRequest.responseXML) {
47
48        clearImages(); // prepare to display a new set of images
```

```
47
48     // get the covers from the responseXML
49     var covers = asyncRequest.responseXML.getElementsByTagName(
50         "cover")
51
52     // get base URL for the images
53     var baseUrl = asyncRequest.responseXML.getElementsByTagName(
54         "baseUrl").item(0).firstChild.nodeValue;
55     // get the placeholder div element named covers
56     var html_covers = document.getElementById("covers");
57     // create an unordered list to display the images
58     var imagesUL = document.createElement("ul");
59     // place images in unordered list
60     for (var i = 0; i < covers.length; ++i) {
61         var cover = covers.item(i); // get a cover from covers array
62         // get the image filename
63         var image = cover.getElementsByTagName("image").
64             item(0).firstChild.nodeValue;
65         var title = cover.getElementsByTagName("title").
66             item(0).firstChild.nodeValue;
67         // create li and img element to display the image
68         var imageLI = document.createElement("li");
69         var imageTag = document.createElement("img");
70         // set img element's src attribute
71         imageTag.setAttribute("src", baseUrl + encodeURIComponent(image));
72         imageTag.setAttribute("alt", title);
73         imageLI.appendChild(imageTag); // place img in li
74         imagesUL.appendChild(imageLI); // place li in ul
75     } // end for statement
76     html_covers.appendChild(imagesUL); // append ul to covers div
77 } // end if
78 } // end function processResponse
79
80 // clears the covers div
81 function clearImages() {
82     document.getElementById("covers").innerHTML = "";
83 } // end function clearImages
84
85 var global_name = "all.xml";
86 // register event listeners
87 function registerListeners() {
88     document.getElementById("all").addEventListener(
89         "click",
```

```
90     function() {
91         getImages("all.xml");
92     }, false);
93     document.getElementById("simply").addEventListener(
94         "click",
95         function() {
96             getImages(global_name);
97         }, false);
98     document.getElementById("howto").addEventListener(
99         "click",
100     function() {
101         getImages("howto.xml");
102     }, false);
103     document.getElementById("dotnet").addEventListener(
104         "click",
105     function() {
106         getImages("dotnet.xml");
107     }, false);
108     document.getElementById("javaccpp").addEventListener(
109         "click",
110     function() {
111         getImages("javaccpp.xml");
112     }, false);
113     document.getElementById("none").addEventListener(
114         "click", clearImages, false);
115 } // end function registerListeners
116
117 window.addEventListener("load", registerListeners, false);
118 </script>
119 </head>
120
121 <body>
122     <input type="radio" name="Books" value="all" id="all"> All Books
123     <input type="radio" name="Books" value="simply" id="simply"> Simply
        Books
124     <input type="radio" name="Books" value="howto" id="howto"> How to
        Program Books
125     <input type="radio" name="Books" value="dotnet" id="dotnet"> .NET
        Books
126     <input type="radio" name="Books" value="javaccpp" id="javaccpp"> Java
        /C/C++ Books
127     <input type="radio" checked name="Books" value="none" id="none"> None
128     <div id="covers"></div>
```

```
129 </body>
130
131 </html>
```