

# AJAX

Asynchronous

*event handling*

JavaScript

*client side*

XML

*Document exchange with server*

# ajax and XMLHttpRequest

- The javascript XMLHttpRequest object is used to send a query to the server.
  - must not be a third party server!
- The query can be anything (GET/POST/HEAD).
- The content comes back to the client (javascript) as an XML document.
- The client parses the XML and decides what to do (update part of the current HTML document, etc).

# XMLHttpRequest

- First introduced by MS as an ActiveX object
  - Prior to this, MS had been promoting the use of Java for client-side networking.
- Netscape (and others) realized the potential and followed MS.
- There are (of course) differences between how MS and everyone else supports this object.

[W3C is working on a standard](#)

# Smarter Clients

- With javascript networking capability, clients (web pages) can do more.
- It is no longer necessary to force a complete document refresh every time the user does anything.
- There are some important ajax applications:
  - google maps, gmail and google suggest
  - [Many others](#)

# Programming with Ajax

- Client side:
  - building an HTTP request and sending it.
  - receiving HTTP reply
    - checking for errors including HTTP errors.
  - Parsing HTTP reply (typically an XML document).
  - Updating HTML elements in the current page.
- Server side:
  - Generating XML documents
  - The original HTML comes from the server...

# Creating an XMLHttpRequest Object

- For IE:

```
http_request = new ActiveXObject("Msxml2.XMLHTTP");
```

-or possibly-

```
http_request = new ActiveXObject("Microsoft.XMLHTTP");
```

- Netscape/Mozilla/Firefox/Safari:

```
http_request = new XMLHttpRequest();
```

See [AJAX: Getting Started](#) for more details.

# Some XMLHttpRequest Properties

## **readyState:**

indicates that state of the transaction

0: uninitialized, 1:loading, ... 4 means "done".

## **onreadystatechange :**

the *callback* function (async only).

## **responseText**

content of the reply as text.

## **responseXML**

top of the XML DOM document

## **status:** HTTP status code (200, 404, ...)

# Some XMLHttpRequest Methods

## **open:**

specify the HTTP method, URL and async/sync

## **send:**

initiates the request. Can also specify POST data.

A complete XMLHttpRequest reference is here:

[XMLHttpRequest Reference](#)

# Sample Code (NS)

```
// create the object
x = new XMLHttpRequest();

// set the function that should be called when
// the ready state changes (the callback)
x.onreadystatechange = handleDoc;

// establish the method and URL
x.open('GET', 'http://foo.com/blah', true);

// initiate the HTTP transaction
x.send(null);
```

↑  
true means "asynchronous"

# Why Asynchronous?

- You can tell `open` to wait until the transaction is complete (set the last parameter to *false*).
  - If the server is slow, or something goes wrong, your javascript is hung at that point.
    - It's hard to recover from this!
- Most people use a callback (asynchronous operation).
  - the Javascript continues on.
  - once something happens, your callback function is called.

# Example Callback

x must be a global variable that is an XMLHttpRequest object

```
function handleChange() {  
    if (x.readyState==4) {  
        // transaction is complete  
        if (x.status==200) {  
            // Server says "OK"  
            doc = x.responseXML; // grab the XML document.  
            ... parse and process the XML document ...  
        } else {  
            ... error (could be 404, etc.) ...  
        } // ignore anything other than readState==4  
    }  
}
```

# HTML vs. XML

- The XMLHttpRequest object can retrieve anything the server can send.
- You (typically) write the server
- You can have the server send HTML to your JavaScript code.
  - replace the innerHTML of some tag with the new HTML from the server.

# HTML example

x must be a global variable that is an XMLHttpRequest object that has already completed a transaction.

```
<div id=shoppingCart></div>
```

```
<script>
```

```
...set up HTTP request, set callback to  
handleCartUpdate ...
```

```
function handleCartUpdate() {
```

```
... check x.readyState and x.status...
```

```
cart = document.getElementById('shoppingcart');
```

```
cart.innerHTML = x.responseText;
```

```
}
```

```
</script>
```

# Using XML

- In general, we want the JavaScript to be able to process the response
  - understand individual elements of the response.
- This requires XML parsing.
- Browsers support an XML DOM
  - this has been standardized, but of course there are some minor differences between MS and everyone else...

# responseXML

- The parsing is done for you, the result is a DOM XML document.
- You can use methods like:
  - `getElementsByTagName()`
    - returns a list of elements with a specific tag name
  - `getElementById()`
    - allows you to select an element by ID attribute.

# DOM XML Parsing

- Very similar to the PHP DOM parser we looked at:
  - elements: represent a tag and everything it contains.
  - node: a node in the parse tree.
  - attribute lists
  - character data (data in JavaScript XML DOM).

# Example: Stock Ticker

- Server has access to current stock prices.
- Server has a web page in which we want a constantly changing stock quote.
  - *constantly meaning every few seconds*
- Need to write a server that generates quotes
  - use random number generator to simulate.
- Need Javascript to periodically send an HTTP request to the server, get current quote and update the page.

# Quote Server

- Returns an XML document with a single stock quote:

```
<quote>
```

```
  <price>current_price</price>
```

```
</quote>
```

# Server in PHP

complete code available: [feed.php](#)

```
<?php
header('Content-Type: text/xml');
$lastprice = $_SESSION['lastprice'];

if (! $lastprice) {
    // first time - initialize to the midpoint
    $lastprice = 20;
}
// generate a new price
$newprice = $lastprice + rand(-1,1);
$_SESSION['lastprice']=$newprice;
echo "<quote>
    <price>$newprice</price>
</quote>\n";
```

# Client HTML Document

```
<h3>Stock Quote: <span id=quote>?</span></h3>
```

```
<script>
```

```
q = document.getElementById('quote');
```

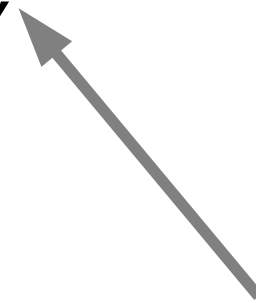
```
var req; // will be the XMLHttpRequest object.
```

```
... code that schedules updates to the span every  
few seconds ...
```

```
</script>
```

# Client HTML Document (cont.)

```
<script>
// called when we want to initiate a new request.
function fetch() {
    x = new XMLHttpRequest();
    x.onreadystatechange = handleDoc;
    x.open('GET', 'feed.php', true);
    x.send(null);
}
```



handleDoc will be called when state of the request changes.

# Client HTML Document (cont.)

```
function handleDoc() {
    ... check for errors, etc. ...
    // fetch was successful - we have a document.
    doc = x.responseXML;
    // find all price tags
    list = doc.getElementsByTagName('price');
    // we are only looking for one
    elem = list.item(0);
    // grab the first child (the content of the tag)
    content = elem.firstChild;
    // finally, grab the value (CDATA)
    price = content.data;
    // update the span in this document
    q.innerHTML=price;
    window.setTimeout('fetch()',1000);
}
```

# Exercise

- AJAX based web-chat
- Start with php code for the server:
  - chatserv.php
    - supports adding new chat message if form field 'msg' is found.
    - returns all messages with ids greater than or equal to form field 'first' (incremental updates).
    - needs a database: [chat database \(sql\)](#)
- Create an HTML document that provides a chat system (using the chatserv server).

# Links

[XMLHttpRequest Reference](#)

[AJAX Getting Started \(Mozilla\)](#)

[XML DOM from w3schools.com](#)

[XMLHttpRequest info from wikipedia \(lots of links\)](#)