One-hour AJAX

Em Tonkin

December 15, 2006

Contents

1 JavaScript

I'm assuming we're all comfortable with JavaScript in general, but I recommend the cheat sheet from `http://www.ilovejackdaniels.com/cheat-sheets/javascript-cheat-sheet/` anyway.

2 Document Object Model (DOM)

The W3C Document Object Model (theoretically) supersedes most of the non-standard nastiness of the ancient past, in terms of the way in which documents were accessed via scripts. Back in the early days of Netscape, it all looked a little messier. The W3C DOM activity is documented at `http://www.w3.org/DOM/`.

In theory, the DOM permits you to directly address and alter pretty much anything about your document, adding, deleting and altering things. DOM is not JavaScript specific (Java, VBScript, etc, can also make use of

it). Actual implementations of the DOM are generally in a state of flux, although the situation is a great improvement on that of a few years ago.

2.1 Browser detection

Back in the old days, you would have put in place a browser detection function to determine what you could expect the client to support. This is deprecated in favour of :

```
if (document.getElementById) {
// do DOM stuff
}
```

The great advantage of getElementById is that it saves you most of the worry about DOM incompatibilities. If you can find an element by ID, you can address it directly without having to explicitly provide its full path in the DOM.

3 AJAX = DHTML + XMLHttpRequest

DHTML was, several years ago, the funky new name for spending months of your life sweating over inconsistent and annoying browser implementations in the hope of producing funky and annoying animated web pages. DHTML, Dynamic HTML, was effectively the use of JavaScript to manipulate (a) DOM elements, and (b) CSS properties.

The killer bug with DHTML was the DOM; at the time (around 2000) Microsoft and Netscape were doing completely inconsistent things. DHTML effectively depended on code like the following :

```
if(document.getElementById){
      // IE 5+, Firefox, Mozilla
  document.getElementById(layerID)
}else if(document.all){
      // IE 4 code goes here
  document.all(somelayerID)...
}else if(document.layers){
      //Netscape 4
  document.layers[somelayerID]...
}
```

These days, nobody's particularly bothered about Netscape or IE 4, and most of the incompatibilities are related to the DOM. Check out the QuirksMode site for DOM incompatibility tables :

`http://www.quirksmode.org/dom/compatibility.html`

The Wikipedia also have a (massive) compatibility table :

`http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(DOM)`

One thing that DHTML was missing was an easy way of handing data back to the server, and retrieving information back. People did find solutions (some of them quite evil; you could for example hand information back to the server by loading an image with a rich query string appended back to it, because JavaScript is able to preload images, but this was clearly in the spirit of hackery rather than design). The solution : XMLHttpRequest.

3.1 XMLHttpRequest

XMLHttpRequest was one of those rare birds ; a really good idea pioneered by Microsoft (but they wrote it as an ActiveX object, so you may sneer at it anyway). First appearing in IE 5, it was quickly replicated in Mozilla 1.0, Firefox and so forth. The W3c came up with DOM 3 Load and Save :

`http://www.w3.org/TR/DOM-Level-3-LS/`

which lets you serialise DOM into XML or the other way around, but XMLHttpRequest is the most common choice anyway.

Creating an XMLHttpRequest object (Firefox)	`var req = new XMLHttpRequest();`
Creating an XMLHttpRequest object (IE)	`var req = new ActiveXObject("Microsoft.XMLHTTP");`
Abort a request	`XMLHttpRequest.abort();`
Get response headers	`getAllResponseHeaders();`
Open a URL	`open( method, URL, [async], [userName], [password]);`
Send a request	`send(content);`
Add a key/value to the HTTP header	`setRequestHeader(label, value)`

An XMLHttpRequest object also has a set of properties :

State change event handler	`onreadystatechange`
Current state of object	`readyState` (from 0 to 4, with 0 =uninitialized, 1 = open, 2 = sent, 3 = receiving, 4 = loaded)
Get response as a string	`responseText`
Get response as XML	`responseXML`
HTTP status code	`status`
Get representative string for status	`statusText`

In theory, then, XMLHttpRequest is the glue that allows you to asynchronously request (and send) information from a server, receiving it in XML, and do whatever you wanted with the result using DOM and JavaScript.

3.2 AJAX limitations

In fact there are an awful lot of these. For example :

- Incompatibilities
- Security concerns
- Increased application complexity
- Usability/accessibility
- Just plain bandwidth usage

The first of these has largely been covered. The second is touched upon in the AJAX QA-Focus security briefing paper, and the third ought to be obvious : anything with as many caveats as AJAX is going to be a bit of a drain on developer time and money.

The fourth is a particularly irritating problem (touched on in another QA-focus briefing paper, by the way). AJAX breaks a lot of useful things, like bookmarks and the back button. There are accessibility concerns, starting from the fact that not everyone has JavaScript, and some who do have it switched off, and – painful as it sounds – best practice is probably to ensure that AJAX pages gracefully degrade to pre-AJAX equivalents.

Regarding bandwidth usage ; one common (mis)use of AJAX is to preload everything in sight. Sometimes this is a good idea, but sometimes it's used almost like one of those 'speed up web surfing' apps that preload pages in the hope that you will click on that link next. This may work well as a strategy,

but overall it can be a problem when you consider large volumes of people all preloading a lot of stuff at the same time...

3.3 Proxies and XMLHttpRequest

XMLHttpRequest isn't enough to make a mashup, unless you're doing it from a privileged place (like a browser plugin, something which is automatically granted permission to do more or less what it likes — in other words, when you visit a web page you do not permit scripts on the web page to do as much as scripts in your browser plugins may do). In general JavaScript is permitted to 'phone home' only to the site from which it began. So you will find yourself spending a lot of time producing proxies for XMLHttpRequest. How to do this depends on the type of service you're calling.

A HTTP Post in PHP :

```
function HTTP_Post($URL,$data, $referrer="") {
        // expects data to be an array !!!
        // parsing the given URL
        $URL_Info=parse_url($URL);

        // Building referrer
        if($referrer=="") // if not given use this script as referrer
          $referrer=$_SERVER["PHP_SELF"];

        // making string from $data
        foreach ($data as $key=>$value)
          $values[]="$key=".urlencode($value);
        $data_string=implode("&",$values);

        // Find out which port is needed - if not given use standard (=80)
        if(!isset($URL_Info["port"]))
          $URL_Info["port"]=80;

        $request="";
        // building POST-request:
        $request.="POST ".$URL_Info["path"]." HTTP/1.1\n";
        $request.="Host: ".$URL_Info["host"]."\n";
        $request.="Referer: $referrer\n";
        $request.="Content-type: application/x-www-form-urlencoded\n";
        $request.="Content-length: ".strlen($data_string)."\n";
        $request.="Connection: close\n";
```

```
        $request.="\n";
        $request.=$data_string."\n";

        $result="";
        $fp = fsockopen($URL_Info["host"],$URL_Info["port"]);
        fputs($fp, $request);
        while(!feof($fp)) {
            $result .= fgets($fp, 128);
        }
        fclose($fp);

// $result is a HTTP response
        return $result;
      }
```

Reading a REST response :

```
class GenericHTTPResponse{
var $HTTPResponse;
    var $HTTPDate;
var $P3P;
var $Connection;
var $Encoding;
var $TransferEncoding;
var $ContentType;
var $HTTPBody;

    function setHTTPResponse($HTTPResponse){
    $this->HTTPResponse=$HTTPResponse;
    }

    function getHTTPResponse(){
    return $this->HTTPResponse;
    }

// ... lots more access functions ...

    function chewRESTResponse($response){
      /* This function assumes that a REST response has been returned...
      and tries to parse it as such, returning the body of the REST response */

       $pieces= explode("\n",$response);
       $body="";
```

```
        $prebody="false";
        for($i=0;$i<sizeof($pieces);$i++){
            if(preg_match("/^HTTP/",$pieces[$i])){
                $this->setHTTPResponse(trim($pieces[$i]));
            } else if(preg_match("/^Date\:/",$pieces[$i])){
                $this->setHTTPDate(trim($pieces[$i]));
            } else if(preg_match("/^P3P\:/",$pieces[$i])){
                $this->setP3P(trim($pieces[$i]));
            }       else if(preg_match("/^Connection\:/", $pieces[$i])){
                $this->setConnection(trim($pieces[$i]));
            } else if(preg_match("/^Encoding\:/", $pieces[$i])){
                $this->setEncoding(trim($pieces[$i]));
            } else if(preg_match("/^Transfer-Encoding\:/",$pieces[$i])){
                $this->setTransferEncoding(trim($pieces[$i]));
            } else if(preg_match("/^Content-Type/",$pieces[$i])){
                $this->setContentType(trim($pieces[$i]));
            } else if(preg_match("/^<\?xml/",$pieces[$i])){
                $prebody="true";
            }
            if($prebody=="true"){
                $body=$body.$pieces[$i];
            }

            }
        $this->setHTTPBody($body);
    }
}
```

3.4 AJAX example

A simple form for retrieving Yahoo! term extraction information :

`http://software.typodemon.com/Yahoo_tools/Term_extractor/`

The AJAX added in the example-ajax version is in the file ajax.js.

```
// Redirect the event from that submit to our newsubmit function
function addEvent(eventobj, eventtype, eventfunct){
// event handling isn't quite standardised yet
    if(eventobj.addEventListener){
        // W3c (Moz)
        eventobj.addEventListener(eventtype, eventfunct, true);
        return true;
    }  else if (eventobj.attachEvent){
        // it's easiest just to do this in a separate function
```

```
        addHandlerForIE();
    } else {
        return false;
    }
}

function addHandlerForIE(){
    // get the element 'myform'
    var handleme = document.getElementById('myForm');
    // attach the onsubmit event to that newsubmit function we created
    handleme.attachEvent('onsubmit',newsubmit);
    return false;
}

function findParentForm(object){
    // check to see if the current object's tag name is "form"
    if(object.tagName.toUpperCase() != 'FORM'){
     // if not, check the parent element
     return findParentForm(object.parentNode);

    } else {
        // if so, return the object
        return object;

    }
}

function emptyDisplay(){
// a bit of basic DOM manipulation
    if (confirm("Are you sure you want to delete the current text?")){
        window.document.myForm.inputcontent.value='';
        document.getElementById('myresponses').innerHTML="";return false;
    }
    return false;
}

function newsubmit(event){
    var target= event ? event.target:this;
    event = (window.event) ? window.event : event;
    // Stop it doing what it was going to do -
// otherwise there's not much point in AJAX
    // This stops it in W3C speak
    if(event.preventDefault){
        event.preventDefault();
    } else {
        // and this stops it in IE
        event.returnValue=false;
        //return false;
    }
    // Get the content of inputcontent
    var mycontent=document.getElementById('inputcontent').value;
    // urlencode it!
    var myencodedcontent=encodeURI(mycontent);
// eew, handcoded url string
```

```
    makeRequest('./AJAXproxy.php?_submit_check=true&inputcontent='+myencodedcontent+'&Iama.xml');

    return false;
    // Now obviously at the end of this function,
// nothing whatsoever happens.
    // That's because you didn't WANT your submit
// to actually go through - you want your AJAXY
// stuff to replace the manual method
    // the point of having the manual method on the
// page is that it works that way if js is switched off.
}


function makeRequest(url) {
// taken from the moz tutorial
    var http_request = false;

    if (window.XMLHttpRequest) { // Mozilla, Safari, ...
        http_request = new XMLHttpRequest();
        if (http_request.overrideMimeType) {
         http_request.overrideMimeType('text/xml');
            // See note below about this line
        }
    } else if (window.ActiveXObject) { // IE
        try {
            http_request = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
         http_request = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) {}
            }
        }

        if (!http_request) {
            alert('Giving up :( Cannot create an XMLHTTP instance');
            return false;
        }
        http_request.onreadystatechange = function() { useContents(http_request); };
        http_request.open('GET', url, true);
        http_request.send(null);

    }


function useContents(http_request) {

    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
            var xmldoc=http_request.responseXML;

            var root_node=xmldoc.getElementsByTagName('ResultSet').item(0);
            // you can be sure to have a resultset,
            // but may not have any results
            // <ResultSet><Result></Result></ResultSet>
```

```
            var result_nodes=xmldoc.getElementsByTagName('Result');
            var resultoutput="";
            if(result_nodes.length>0){
                for (var iCount=0; iCount<result_nodes.length; iCount++){
                    //alert("Result!"+result_nodes.item(iCount).firstChild.nodeValue);
                    if(iCount>0){
                        resultoutput=resultoutput+", "+
result_nodes.item(iCount).firstChild.nodeValue;
                    } else {
                        resultoutput=result_nodes.item(iCount).firstChild.nodeValue;


                    }

                }
              document.getElementById('myresponses').innerHTML=resultoutput;
          } else {
              document.getElementById('myresponses').innerHTML="No keywords were given";
          }
      } else {
          alert('There was a problem with the request.');
      }
    }
}
```

3.5 AJAX frameworks

On the principle that nobody sane wants to waste their time producing and debugging miles of code of this variety, there have been several AJAX frameworks designed.

AJAXPatterns wiki list	`http://ajaxpatterns.org/wiki/index.php?title=AJAXFrameworks`
Prototype	`http://prototype.conio.net/`
Yahoo! User Interface Library	`http://developer.yahoo.net/yui/`
Zimbra AJAX Toolkit	`http://www.zimbra.com/community/ajaxtk_download.html`

And so forth. I've tried essentially none of these, however.