An Introduction to Ajax

« article index

1. Introduction

Web applications have traditionally performed in a request/response paradigm where any change to a page's content requires the page to be completely reloaded. While this is an effective model for basic applications, is it unquestionably limiting in comparison to the asynchronous model common in desktop applications.

Techniques and technologies designed to achieve the asynchronous model have existed for years with much of the focus on replacing HTML entirely (e.g. Adobe Flash, Java applets). Other, less radical, approaches have exploited `<iframe>` [1], an HTML element designed for inline frames.

Asynchronous JavaScript and XML, or Ajax, is another technique that allows the asynchronous model to be used within the context of a web browser and one that is becoming increasingly prevalent as an enabler in 'Web 2.0' applications.

Ajax is unique in that it leverages existing cross-browser client technologies rather than seeking to replace them and is also a far more powerful approach than the limited `<iframe>` one, providing a complete API, full control of HTTP request types and headers, and callback hooks.

1.1. Ajax Defined

The Ajax term was first coined by Jesse James Garrett in his seminal 2005 article *Ajax: A New Approach to Web Applications.* [2]

As defined by Garrett, Ajax is not a distinct technology in itself, but is an engineering technique based on the use of:

- standards-based presentation using XHTML and CSS
- dynamic display and interaction using the DOM
- data interchange and manipulating using XML and XSL
- asynchronous data retrieval using the `XMLHttpRequest` object
- Java/ECMAScript as the binding technology

While memorable, the term Ajax is rather non-representative of a variety of reasons.

Firstly, there is no such thing as asynchronous JavaScript; rather it is the underlying HTTP-based communication that is asynchronous in nature. Secondly, Ajax is not limited to JavaScript - other client-side scripting languages such as VBScript can be used. Nor is Ajax intrinsically tied to XML, as data may be exchanged in any format from raw character sequences, to XML, through to JSON (1.3 JSON).

Given this, Ajax can be more accurately posited as a technique for bi-directional client-server transactions from within client-side scripts, conducted through instances of the `XMLHttpRequest` class. [4]

1.2. XMLHttpRequest

1.2.1. History

`XMLHttpRequest` [5] was originally developed by Microsoft as part of its MSXML library for use in their Outlook Web Access 2000 product in 1999. Microsoft's implementation, called `XMLHTTP`, provided an interface for programmatically sending parameterised GET and POST queries to HTTP servers, retrieving data back for processing or display [6]. Support for XMLHTTP was included in Internet Explorer version 5.0 shortly after as an ActiveX component.

In 2001, the Mozilla developers incorporated the first compatible native implementation of `XMLHttpRequest` with Apple (Safari version 1.2), Opera (Opera version 8.0), iCab (version 3.0b352), and Open Source projects such as Konqueror following suit.

In April 2006 the W3C published a working draft specification for the `XMLHttpRequest` API with a goal to "document a minimum set of interoperable features based on existing implementations, allowing web developers to use these features without platform-specific code". [7]

1.2.2. Browsers, and Instantiating XMLHttpRequest

While most browser vendors have chosen to implement `XMLHttpRequest` as a native JavaScript object, Microsoft implemented `XMLHttpRequest` as an ActiveX control in their Internet Explorer browser versions 5.0 – 6.0. (Internet Explorer version 7.0 has native JavaScript support.)

Because of this, any JavaScript code used to create an instance of `XMLHttpRequest` must contain logic that can use either implementation.

To do this is relatively straightforward and involves checking a browser's support for ActiveX objects. For browsers supporting ActiveX objects the `XMLHttpRequest` instance is created using the ActiveX technique, for other browsers the alternative technique is used.

Listing 1.1 provides the necessary code. Note that there are two versions of the ActiveX control and both must be catered for.

Listing 1.1. Instantiating XMLHttpRequest

```
var xmlHttp;
if (window.XMLHttpRequest) {
 xmlHttp = new XMLHttpRequest();
} else if (window.ActiveXObject) {
 try {
  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
 } catch(e) {
  try {
   xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  } catch {
   throw(e);
  }
 }
}
```

While all major browsers now provide native JavaScript support for `XMLHttpRequest`, for backwards compatibility reasons the method described above will still be necessary for some time.

1.2.2 Using XMLHttpRequest

The properties and methods exposed by the `XMLHttpRequest` object are detailed in tables 1.1 and 1.2.

Table 1.1. Standard XMLHttpRequest Properties

Property	Description

`onreadystatechange`	Event handler that fires at every state change.
`readyState`	The state of the request as follows: 0 = uninitialized 1 = loading 2 = loaded 3 = interactive 4 = complete
`responseText`	The response from the server as a string.
`responseXML`	The response from the server as an XML document object.
`status`	The HTTP status code from the server.
`statusText`	The textual version of the HTTP status code.

Table 1.2. Standard XMLHttpRequest Methods

Method	Description
`abort()`	Aborts the current request.
`getAllResponseHeaders()`	Returns all the response headers for the HTTP request as a string.
`getResponseHeader(headerName:string)`	Returns the value of the specified header as a string.
`open(method:string, URL:string, [, asyncFlag:Boolean [, userName:string [, password:string]]])`	Opens the connection with the server. The method parameter can be either "GET", "POST", or "PUT". The URL can be relative or absolute. The async parameter indicates whether the request should be handler asynchronously.
`send(content:string)`	Dispatches the request to the server, optionally with postable string or DOM object data.
`setRequestHeader(label:string, value:string)`	Adds a label/value pair to the HTTP header to be sent.

A typical Ajax interaction using `XMLHttpRequest` consists of the following steps:

1. A client-side event such as an `onclick()` occurs, automatically calling a specified client-side JavaScript function.
2. An instance of `XMLHttpRequest` is created and a function reference assigned to its `onreadystatechange` event handler.
3. The `XMLHttpRequest` instance makes a remote call based on arguments passed to its `open()` function, followed by a call to `send()`.
4. The function assigned to `onreadystatechange` tests the `readyState` and `status` properties, and assuming success, processes the response that has been stored in either the `responseText` property (in the case of JSON, or another custom format) or the `responseXML` property (in the case of XML).
5. The DOM is updated reflecting any changes automatically without page refresh.

One interesting point to note is the automatic handling of XML. If XML is returned, the `XMLHttpRequest` instance parses it automatically upon receipt, populating the `responseXML` property and resulting in an XML DOM object-hierarchy.

1.3. JSON

As discussed previously (1.2.2 Using XMLHttpRequest), the use of XML with `XMLHttpRequest` is not mandatory and alternative data formats may be used.

JSON [8], or JavaScript Object Notation, is a lightweight data-interchange format based on a subset of the JavaScript programming language - specifically, Standard ECMA-262 3rd Edition - December 1999 [9]. In July 2006, the creator of JSON, Douglas Crockford, submitted the JSON specification as a draft to the IETF.

JSON is a text format that is language independent but uses conventions similar to the C family of languages such as C, C++, C#, Java, JavaScript, Perl, Python, Ruby, and others. JavaScript is not required for its use.

JSON is built on two data structures:

- A collection of name/value pairs. In modern programming languages, name/value pairs are typically realised as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In modern languages, ordered lists are typically realised as an array, vector, list, or sequence.

A JSON object is an unordered set of name/value pairs beginning with a { and ending with a }. A colon separates the name/value pairs.

A JSON array is an ordered collection of values beginning with a [and ending with a]. A comma separates the values of the array. A value can be a string enclosed in double quotes, or a number, or a true or false or null, or an object, or an array. These structures can be nested.

Strings are a collection of zero or more Unicode characters, enclosed in double quotes, using backslashes to escape control characters. Numbers are comparative to those in languages such as C or Java, however hexadecimal or octal formats are not used.

Listing 1.2. An XML and JSON Comparative Example

```
<Invoice Number = "43508" Date = "10/01/2008">
 <Item Description = "Bolts" Quantity = "2500" Price = "0.10" />
 <Item Description = "Washers" Quantity = "5000" Price = "0.05" />
 <Item Description = "Screws" Quantity = "3000" Price = "0.10" />
</Invoice>

// An invoice element in JavaScript object-literal syntax.
var invoice = {
 Number : "01001",
 Date : "10/01/2008",
 Items : [
  { Description : "Bolts", Quantity : "2500", Price : "0.10" },
  { Description : "Washers", Quantity : "5000", Price : "0.05" },
  { Description : "Screws", Quantity : "3000", Price : "0.10" }
 ]
}
```

Listing 1.2 shows a sample invoice first in XML and then in JSON for comparative purposes:

JSON is of interest for Ajax development primarily because as a subset of JavaScript's object-literal syntax, it relates implicitly to run-time JavaScript objects, removing the impedance between the communication format and the object representation in client-side code.

The JavaScript `eval` function (part of the underlying ECMAScript standard) allows for direct invocation of the interpreter, and when passed a JSON string will return a first class object that can be manipulated using standard JavaScript notation (listing 1.3).

Listing 1.3. Instantiating a JavaScript object via JSON

```
var invoice = eval("("
+ "{ Number : '01001', Date : '10/01/2008' }"
+ ")");
alert(invoice.Number + " - " + invoice.Date);
```

Further, as JavaScript object-literal definitions can contain functions, Ajax applications leveraging JSON could be written to dynamically retrieve additional functionality from the server as-and-when needed.

1.4. Ajax Frameworks

A wide range of frameworks and libraries designed to simplify Ajax development have sprung up over the past few years ranging from simple cross-browser `XMLHttpRequest` wrappers to feature-rich toolkits [11, appendix B]. In addition to Open Source projects, companies such as Google, Yahoo!, and Microsoft are contributing to the space with their own frameworks and libraries.

The growing number of Ajax frameworks and libraries available can be generally split into two groups: client-side and server-side.

Client-Side

Client-side Ajax frameworks and libraries are written solely in a client-side language such as JavaScript, contain no server-side dependencies, and are agnostic to underlying server technologies. Examples in this space include the popular jQuery and Prototype frameworks and the Yahoo! UI and Scriptaculous libraries.

Server-Side

Server-side Ajax frameworks enable developers to build Ajax web applications by targeting APIs native to their preferred server-side programming language rather than working directly with client-side technologies.

With server-side Ajax frameworks HTML and associated client-side script (i.e. JavaScript) are typically generated automatically on client request by either a servlet or server-side script. Examples in this space include the Google Web Toolkit (for Java), and Ajax.NET (for Microsoft.NET).

1.5. Security

When the `XMLHttpRequest` object is used in a web browser, it adopts the same-domain security policies that exist for typical JavaScript activity (the so-called 'sandbox'). That is, an `XMLHttpRequest` instance can only access resources from the same domain as the web page that the request is made from. Any attempt to request an outside resource will either silently fail or warn the user (depending on the browser in question).

While there for a good reason, the same-domain security policy is an obvious limitation for those attempting to use `XMLHttpRequest` in conjunction with an external resource such as a web service.

Workarounds do exist but their use should to be carefully considered in respect to potential security implications. In particular, all data coming from an external resource should be thoroughly sanitised prior to use. This is especially critical for Ajax applications using external JSON (1.3 JSON) data, as JSON data is essentially arbitrary third-party code that will be executed in the scope of the requesting web page.

The growing use of Ajax will inevitably lead to more sophisticated and complex web applications and with that complexity, the number of potential security issues will increase. However Ajax itself is not inherently insecure – indeed, the security issues related to input validation and verification on the Web existed prior to Ajax and, as was the case then, can be addressed by applying long-established secure-coding techniques.

1.6. Usability

While Ajax allows for more flexible, responsive, and powerful web applications than previously possible (at least, without resorting to technologies such as Adobe Flash or Java applets), it also brings new challenges in terms of usability and accessibility. As a relatively new approach, there are few established best practices or heuristics that can be applied.

Web users are accustomed to web applications performing in a request/response paradigm – the asynchronous nature of Ajax means this is no longer necessarily the case. As web applications are moving from a fixed-page model, issues such as URL linearity, site indexing, and 'broken back buttons' need to be addressed.

A number of frameworks have attempted to address the 'broken back button' issue by providing hooks that developers can use to manipulate the browser's history (for example, the Dojo framework supports this via its `dojo.io.bind()` method [11]).

Some commentators argue [4] that as web applications move closer towards being constantly running programs with ever-changing states, it is unreasonable to expect search engines to be able to catalogue a particular state. Nevertheless, indexing is an issue of high importance and will no doubt have a direct effect on how Ajax is introduced into many web applications.

Interaction designers should consider these issues when designing Ajax web applications, particularly where such applications are targeted at the general public and not a controlled group of users. Ajax-enabled features should either degrade gracefully or alternative non-Ajax pages should be provided where possible.

1.7. Summary

In this article it has been shown how Ajax represents a shift from the request/response paradigm of traditional web applications to the asynchronous model common in desktop applications.

A definition for Ajax has been provided, and the core enabler for Ajax web applications, the `XMLHttpRequest` object, has been discussed in detail. JSON, or JavaScript Object Notation, has been demonstrated as an alternative to XML for data interchange.

Both client-side and server-side Ajax frameworks have been defined and the difference between both groups explained.

The potential security implications of Ajax-enabled websites have been discussed and it has been shown that while Ajax itself it not inherently insecure, due diligence must be taken.

Finally, the usability challenges of Ajax web applications have been highlighted, specifically in respect to URL linearity, site indexing, and the 'broken back button'.

References

1. Frames in HTML Documents
2. Garrett J. J., Ajax: A New Approach to Web Applications
3. Thomas D., Hansson D., et al, Agile Web Development with Rails, The Pragmatic Bookshelf, 2005
4. Vaughan R., An Analysis of Ajax, Objective View, Issue 9
5. XMLHttpRequest
6. Forbes D., How I Came To Despise Ajax
7. The XMLHttpRequest Object
8. JavaScript Object Notation
9. ECMAScript 3rd Edition (PDF document)
10. Asleson R., Schutta N. T., Foundations of Ajax, Apress, 2006
11. dojo.io.*

Colophon

Last updated February 12, 2008. Creative Commons licensed.