

AJAX Overview

Scott Martin

`http://www.ling.osu.edu/~scott/`

January 19, 2007

What is AJAX and what is it for?

- Promising new technique in client-server web development
- AJAX stands for *Asynchronous Javascript And XML*
- Allows web clients to programmatically fetch data *between* full requests
- Uses the built-in scripting and DOM (*Document Object Model*) XML parsing capabilities of modern web browsers
- Eliminates the need to use hidden IFRAMEs, Flash, Java applets and other hacks for making micro requests

How does it change the web?

Before AJAX Inefficient, start/stop interaction

- HTTP is designed as a stateless protocol
- Web applications were limited to fetching and rendering entire HTML documents

Since AJAX More like a desktop application

- Things can happen in the foreground and in the background
- Can deal with user input while the user is interacting with the application, instead of requiring the user to stop each time
- Web applications can interact more quickly and fluidly with large data sets, as small pieces of data can be retrieved as needed
- Allows data to be retrieved in different formats than just standard HTML

The *XMLHttpRequest* class

- Simple, flexible Javascript API class
- Name is slightly misleading, since data need not be formatted as XML
- Allows construction and configuration of HTTP requests
- Returns HTTP status codes, response meta-data, and payload data in several formats
 - XML-based data (including XHTML, XSLT, etc.)
 - Plain-text data
 - Empty response with data in response headers (meta-data)
- Does not rely on any particular server configuration

Using *XMLHttpRequest* : Preliminary requirements

- A browser that supports *XMLHttpRequest* (or equivalent)
 - Internet Explorer** Since version 5, in ActiveX as `Microsoft.XMLHTTP`
 - Mozilla** All variants
 - Opera** Since version 9
 - Safari** Since version 1.2
- A server-side process that can produce appropriate responses

Using *XMLHttpRequest* : Creating and sending the request

1. Create an *XMLHttpRequest* object
2. Specify the type of request and configure the request with attribute-value pairs in the standard HTTP format (name=value&othername=othervalue& ...)
GET **requests** Parameters in query string
POST **requests** Parameters in request body
3. Send the request

Using *XMLHttpRequest* : Handling the response

1. Wait for response to be completely received
 - Ready state 0** Uninitialized
 - Ready state 1** Loading
 - Ready state 2** Loaded
 - Ready state 3** Interactive
 - Ready state 4** Complete
2. Handle any errors properly
3. Access the response data
 - Using the DOM, if the response is encoded as XML
 - By parsing the text, if the response is plain text
 - By accessing the value of response header fields, if the data is sent as HTTP meta-data
4. Update the interface with the new data

Using *XMLHttpRequest* : Things to watch out for

Submitted data must be urlencoded using the Javascript `escape()` function

POST **requests** should provide meta-data that describe the request content

`Content-Type: application/x-www-form-urlencoded`

`Content-Length: (number of bytes of data being sent)`

Response content type is crucial as *XMLHttpRequest* only provides a DOM object representing the XML response data if `Content-Type:text/xml` is specified in the response

Microsoft's implementation of *XMLHttpRequest* is slightly different than the mainstream, as in the `XMLHttpRequest.send()` method

Empty response body causes an error to be incorrectly triggered in early versions of Safari. Explicitly closing the response fixes this.

References and further reading

Example implementation

<http://www.coffeeblack.org/work/ajax/example.html>

Early article describing this technique

<http://www.adaptivepath.com/publications/essays/archives/000385.php>

W3C introduction to *XMLHttpRequest*

http://www.w3schools.com/xml/xml_http.asp

Mozilla documentation of the *XMLHttpRequest* class

<http://developer.mozilla.org/en/docs/XMLHttpRequest>

Apple documentation of *XMLHttpRequest* class

<http://developer.apple.com/internet/webcontent/xmlhttpreq.html>