

Ajax (programming)

From Wikipedia, the free encyclopedia

Ajax, shorthand for *Asynchronous JavaScript and XML*, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is meant to increase the web page's interactivity, speed, and usability.

The Ajax technique uses a combination of:

- XHTML (or HTML) and CSS, for marking up and styling information.
- The DOM accessed with a client-side scripting language, especially ECMAScript implementations such as JavaScript and JScript, to dynamically display and interact with the information presented.
- The XMLHttpRequest object is used to exchange data asynchronously with the web server. In some Ajax frameworks and in certain situations, an IFrame object is used instead of the XMLHttpRequest object to exchange data with the web server, and in other implementations, dynamically added <script> tags may be used.
- XML is sometimes used as the format for transferring data between the server and client, although any format will work, including preformatted HTML, plain text, JSON and even EBML. These files may be created dynamically by some form of server-side scripting.

Like DHTML, LAMP and SPA, Ajax is not a technology in itself, but a term that refers to the use of a group of technologies.

Contents

- 1 History
- 2 Justification
- 3 Pros and cons
 - 3.1 Pros
 - 3.1.1 Bandwidth utilization
 - 3.1.2 User interface
 - 3.1.3 Separation of Data, Format, Style, and Function
 - 3.2 Cons
 - 3.2.1 Browser integration
 - 3.2.2 Response-time concerns
 - 3.2.3 Search Engine Optimization
 - 3.2.4 Javascript reliability
- 4 Accessibility
- 5 See also
- 6 References
- 7 External links

History

The first use of the term in public was by Jesse James Garrett in February 2005^[1]. Garrett thought of the term when he realized the need for a shorthand term to represent the suite of technologies he was proposing to a client^[2].

Although the term "Ajax" was coined in 2005, most histories of the technologies that enable Ajax start a decade earlier with Microsoft's initiatives in developing Remote Scripting. Techniques for the asynchronous loading of content on an existing Web page without requiring a full reload date back as far

as the `IFRAME` element type (introduced in Internet Explorer 3 in 1996) and the `LAYER` element type (introduced in Netscape 4 in 1997, abandoned during early development of Mozilla). Both element types had a `src` attribute that could take any external URL, and by loading a page containing JavaScript that manipulated the parent page, Ajax-like effects could be attained. This set of client-side technologies was usually grouped together under the generic term of DHTML. Macromedia's Flash could also, from version 4, load XML and CSV files from a remote server without requiring a browser refresh.

Microsoft's Remote Scripting (or MSRS, introduced in 1998) acted as a more elegant replacement for these techniques, with data being pulled in by a Java applet with which the client side could communicate using JavaScript. This technique worked on both Internet Explorer version 4 and Netscape Navigator version 4 onwards. Microsoft then created the `XMLHttpRequest` object in Internet Explorer version 5 and first took advantage of these techniques using `XMLHttpRequest` in Outlook Web Access supplied with the Microsoft Exchange Server 2000 release.

The Web development community, first collaborating via the *microsoft.public.scripting.remote* newsgroup and later through blog aggregation, subsequently developed a range of techniques for remote scripting in order to enable consistent results across different browsers. In 2002, a user-community modification^[3] to Microsoft Remote Scripting was made to replace the Java applet with `XMLHttpRequest`.

Remote Scripting Frameworks such as ARSCIF^[4] surfaced in 2003 not long before Microsoft introduced Callbacks in ASP.NET^[5].

In addition, the World Wide Web Consortium has several Recommendations that also allow for dynamic communication between a server and user agent, though few of them are well supported. These would include:

- The object element defined in HTML 4 for embedding arbitrary content types into documents, (replaces inline frames under XHTML 1.1)
- The Document Object Model (DOM) Level 3 Load and Save Specification [1] (<http://www.w3.org/TR/DOM-Level-3-LS/>)

Justification

The core justification for AJAX style programming is to overcome the page loading requirements of HTML/HTTP-mediated web pages. AJAX creates the necessary initial conditions for the evolution of complex, intuitive, dynamic, data-centric user interfaces in web pages - the realization of that goal is still a work in progress.

Web pages, unlike native applications, are loosely coupled, meaning that the data they display are not tightly bound to data sources and must be first marshalled into an HTML page format before they can be presented to a user agent on the client machine. For this reason, web pages have to be re-loaded each time a user needs to view different datasets. By using the `XmlHttpRequest` object to request and return data without a re-load, a programmer by-passes this requirement and makes the loosely coupled web page behave much like a tightly coupled application, but with a more variable lag time for the data to pass through a longer "wire" to the remote web browser.

For example, in a classic desktop application, a programmer has the choice of populating a tree view control with all the data needed when the form initially loads, or with just the top-most level of data - which would load quicker, especially when the dataset is very large. In the second case, the application would fetch additional data into the tree control depending on which item the user selects. This functionality is difficult to achieve in a web page without AJAX. To update the tree based on a user's selection would require the entire page to re-load, leading to a very jerky, non-intuitive feel for the web user who is browsing the data in the tree.

Pros and cons

Pros

Bandwidth utilization

By generating the HTML locally within the browser, and only bringing down JavaScript calls and the actual data, Ajax web pages can appear to load relatively quickly since the payload coming down is much smaller in size. An example of this technique is a large result set where multiple pages of data exist. With Ajax, the HTML of the page, e.g., a table control and related TD and TR tags can be produced locally in the browser and not brought down with the first page of the document.

In addition to "load on demand" of contents, some web based applications load stubs of event handlers and then load the functions on the fly. This technique significantly cuts down the bandwidth consumption for web applications that have complex logic and functionality.

User interface

The most obvious reason for using Ajax is an improvement to the user experience. Pages using Ajax behave more like a standalone application than a typical web page. Clicking on links that cause the entire page to refresh feels like a "heavy" operation. With Ajax, the page often can be updated dynamically, allowing a faster response to the user's interaction. While the full potential of Ajax has yet to be determined, some believe it will prove to be an important technology, helping making the web even more interactive and popular than it currently is.

Separation of Data, Format, Style, and Function

A less specific benefit of the AJAX approach is that it tends to encourage programmers to clearly separate the methods and formats used for the different aspects of information delivery via the web. Although AJAX can appear to be a jumble of languages and techniques, and programmers are free to adopt and adapt whatever works for them, they are generally propelled by the development motif itself to adopt separation between: (1) the raw data or content to be delivered - which is normally embedded in XML and sometimes derived from a server-side database; (2) the format or structure of the webpage - which is almost always built in HTML (or better, XHTML) and is then reflected and made available to dynamic manipulation in the DOM; (3) the style elements of the webpage - everything from fonts to picture placement - are derived by reference to embedded or referenced CSS; and (4) the functionality of the web page is provided by a combination of (A) Javascript on the client browser (also called DHTML), (B) Standard HTTP and XMLHttpRequest for client-to-server communication, and (C) server-side scripting and/or programs utilizing any suitable language preferred by the programmer to receive the client's specific requests and respond appropriately.

Cons

Browser integration

The dynamically created page does not register itself with the browser history engine, so triggering the "Back" function of the users' browser might not bring the desired result.

Developers have implemented various solutions to this problem. These solutions can involve using invisible IFRAMEs to invoke changes that populate the history used by a browser's back button. Google Maps, for example, performs searches in an invisible IFRAME and then pulls results back into an element on the visible web page. The World Wide Web Consortium (W3C) did not include an *iframe* element in its XHTML 1.1 Recommendation; the Consortium recommends the *object* element instead.

Another issue is that dynamic web page updates make it difficult for a user to bookmark a particular state of the application. Solutions to this problem exist, many of which use the URL *fragment identifier* (the portion of a URL after the '#' ^[6] ^[7]) to keep track of, and allow users to return to, the application in a given state. This is possible because many browsers allow JavaScript to update the fragment identifier of the URL dynamically, so that Ajax applications can maintain it as the user changes the application's state. This solution also improves back-button support. It is, however, not a complete solution.

Response-time concerns

Network latency — or the interval between user request and server response — needs to be considered carefully during Ajax development. Without clear feedback to the user ^[8], smart preloading of data and proper handling of the XMLHttpRequest object, users might experience delay in the interface of the web application, something which users might not expect or understand. Additionally, when an entire page is rendered there is a brief moment of re-adjustment for the eye when the content changes. The lack of this re-adjustment with smaller portions of the screen changing makes the latency more apparent. The use of visual feedback (such as throbbers) to alert the user of background activity and/or preloading of content and data are often suggested solutions to these latency issues.

In general the potential impact of latency has not been "solved" by any of the open source Ajax toolkits and frameworks available today, such as the effect of latency variance over time.

Search Engine Optimization

Websites that use Ajax to load data which should be indexed by search engines must be careful to provide equivalent data at a public, linked URL and in a format that the search engine can read, as search engines do not generally execute the JavaScript code required for Ajax functionality. This problem is not specific to Ajax, as the same issue occurs with sites that provide dynamic data as a full-page refresh in response to, eg, a form submit (the general problem is sometimes called the hidden web).

Javascript reliability

Ajax relies on Javascript, which may be implemented differently by different browsers or versions of a particular browser. Because of this, sites that use Javascript may need to be tested in multiple browsers to check for compatibility issues. It's not uncommon to see a Javascript code written twice, a part for IE, a part for Mozilla compatibles.

Accessibility

Using Ajax technologies in web applications provides many challenges for developers interested in adhering to WAI accessibility guidelines. In addition there are numerous development groups working on USA government projects which require strict adherence to Section 508 Compliance standards. Failure to comply with these standards can often lead to cancellation of contracts or lawsuits intended to ensure compliance.

Because of this, developers need to provide fallback options for users on other platforms or browsers, as most methods of Ajax implementation rely on features only present in desktop graphical browsers.

Web developers use Ajax in some instances to provide content only to specific portions of a web page, allowing data manipulation without incurring the cost of re-rendering the entire page in the web browser. Non-Ajax users would ideally continue to load and manipulate the whole page as a fallback, allowing the developers to preserve the experience of users in non-Ajax environments (including all relevant accessibility concerns) while giving those with capable browsers a much more responsive experience.

See also

- Ajax framework
- Comet (programming)
- HTTP streaming
- Progressive enhancement
- Reverse Ajax
- Rich Internet application
- Single page application
- Web 2.0
- XMLHttpRequest

References

- ↑ Ajax: A New Approach to Web Applications (<http://www.adaptivepath.com/publications/essays/archives/000385.php>) . Adaptive Path (2005-02-18). Retrieved on 2006-08-01.
- ↑ At subsequent talks and seminars Garrett has made the point that Ajax is not an acronym.
- ↑ HTTPRequest-enabled RS (http://groups.google.ca/group/microsoft.public.scripting.remote/browse_thread/thread/99b7e6152t . microsoft.public.scripting.remote newsgroup (2002-06-18). Retrieved on 2006-08-01.
- ↑ ARSCIF: A Framework for Asynchronous Remote–Script Callback Invocation (<http://arscif.dsi.unimi.it/>) . Sebastiano Vigna. Retrieved on 2006-08-01.
- ↑ Cutting Edge: Script Callbacks in ASP.NET (<http://msdn.microsoft.com/msdnmag/issues/04/08/CuttingEdge/>) . MSDN Magazine (2004-08-08). Retrieved on 2006-08-01.
- ↑ Uniform Resource Identifiers (URI): Generic Syntax (<http://www.ietf.org/rfc/rfc2396.txt>) . The Internet Society (August 1998). Retrieved on 2006-07-21.
- ↑ Uniform Resource Identifier (URI): Generic Syntax (<http://www.ietf.org/rfc/rfc3986.txt>) . The Internet Society (January 2005). Retrieved on 2006-07-21.
- ↑ Remote Scripting with AJAX, Part 2 (<http://www.xml.com/pub/a/2005/08/22/ajax.html>) . O'Reilly XML.com (2005-08-22). Retrieved on 2006-07-21.

External links

- AJAX category (<http://dmoz.org/Computers/Programming/Languages/JavaScript/AJAX>) on the Open Directory Project.

Articles

- Jesse James Garrett. “Ajax: A New Approach to Web Applications”, Adaptive Path (<http://www.adaptivepath.com/publications/essays/archives/000385.php>)

Tutorials

- Ajax:Getting Started (http://developer.mozilla.org/en/docs/AJAX:Getting_Started) by Mozilla Developer Center.
- Ajax Tutorial (<http://www.xul.fr/en-xml-ajax.html>) with get, post, text and XML examples.
- Presentation (http://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf) on Ajax Security issues given at the Black Hat security conference.

Retrieved from "http://en.wikipedia.org/wiki/Ajax_%28programming%29"

Categories: Ajax (programming) | JavaScript programming language | Programming language topics | Software architecture | Web 2.0 | XML

-
- This page was last modified 20:39, 9 January 2007.
 - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a US-registered 501(c)(3) tax-deductible

nonprofit charity.