XMLHttpRequest

XMLHttpRequest (**XHR**) is an API in the form of a JavaScript object whose methods transmit HTTP requests from a web browser to a web server.[1] The methods allow a browser-based application to send requests to the server after page loading is complete, and receive information back.[2] XMLHttpRequest is a component of Ajax programming. Prior to Ajax, hyperlinks and form submissions were the primary mechanisms for interacting with the server, often replacing the current page with another one.[2]

History

The concept behind XMLHttpRequest was conceived in 2000 by the developers of Microsoft Outlook.[3] The concept was then implemented within the Internet Explorer 5 browser (2001). However, the original syntax did not use the `XMLHttpRequest` identifier. Instead, the developers used the identifiers `ActiveXObject("Msxml2.XMLHTTP")` and `ActiveXObject("Microsoft.XMLHTTP")`.[4] As of Internet Explorer 7 (2006), all browsers support the `XMLHttpRequest` identifier.[4]

The `XMLHttpRequest` identifier is now the *de facto* standard in all the major browsers, including Mozilla's Gecko layout engine (2002),[5] Safari 1.2 (2004) and Opera 8.0 (2005).[6]

Standards

The World Wide Web Consortium (W3C) published a *Working Draft* specification for the *XMLHttpRequest* object on April 5, 2006.[7] [a] On February 25, 2008, the W3C published the *Working Draft Level 2* specification.[8] Level 2 added methods to monitor event progress, allow cross-site requests, and handle byte streams. At the end of 2011, the Level 2 specification was absorbed into the original specification.[9]

At the end of 2012, the WHATWG took over development and maintains a living document using Web IDL.[10]

Usage

Generally, sending a request with XMLHttpRequest has several programming steps.[11]

1. Create an XMLHttpRequest object by calling a constructor:

```
var request = new XMLHttpRequest();
```

2. Call the "open" method to specify the request type, identify the relevant resource, and select synchronous or asynchronous operation:

```
request.open('GET', '/api/message', true /* asynchronous */);
```

3. For an asynchronous request, set a listener that will be notified when the request's state changes:

```
request.onreadystatechange = listener;
```

4. Initiate the request by calling the "send" method:

```
request.send();
```

5. Respond to state changes in the event listener. If the server sends response data, by default it is captured in the "responseText" property. When the object stops processing the response, it changes to state 4, the "done" state.

```
function listener() {
  // Check whether the request is done and successful.
  if (request.readyState == 4 && request.status == 200)
    console.log(request.responseText); // Display the text.
}
```

Aside from these general steps, XMLHttpRequest has many options to control how the request is sent and how the response is processed. Custom header fields can be added to the request to indicate how the server should fulfill it,[12] and data can be uploaded to the server by providing it in the "send" call.[13] The response can be parsed from the JSON format into a readily usable JavaScript object, or processed gradually as it arrives rather than waiting for the entire text.[14] The request can be aborted prematurely[15] or set to fail if not completed in a specified amount of time.[16]

Cross-domain requests

In the early development of the World Wide Web, it was found possible to breach users' security by the use of JavaScript to exchange information from one web site with that from another less reputable one. All modern browsers therefore implement a same origin policy that prevents many such attacks, such as cross-site scripting. XMLHttpRequest data is subject to this security policy, but sometimes web developers want to intentionally circumvent its restrictions. This is

sometimes due to the legitimate use of subdomains as, for example, making an XMLHttpRequest from a page created by `foo.example.com` for information from `bar.example.com` will normally fail.

Various alternatives exist to circumvent this security feature, including using JSONP, Cross-Origin Resource Sharing (CORS) or alternatives with plugins such as Flash or Silverlight (both now deprecated). Cross-origin XMLHttpRequest is specified in W3C's XMLHttpRequest Level 2 specification.[17] Internet Explorer did not implement CORS until version 10. The two previous versions (8 and 9) offered similar functionality through the XDomainRequest (XDR) API. CORS is now supported by all modern browsers (desktop and mobile).[18]

The CORS protocol has several restrictions, with two models of support. The *simple* model does not allow setting custom request headers and omits cookies. Further, only the HEAD, GET and POST request methods are supported, and POST only allows the following MIME types: "text/plain", "application/x-www-urlencoded" and "multipart/form-data". Only "text/plain" was initially supported.[19] The other model detects when one of the *non-simple* features are requested and sends a *pre-flight request*[20] to the server to negotiate the feature.

Fetch alternative

Program flow using asynchronous XHR callbacks can present difficulty with readability and maintenance. ECMAScript 2015 (ES6) added the promise construct to simplify asynchronous logic. Browsers have since implemented the alternative `fetch()` interface to achieve the same functionality as XHR using promises instead of callbacks.

Fetch is also standardized by WHATWG.[21]

Example

```
fetch('/api/message')
  .then(response => {
    if (response.status != 200) throw new Error('Request failed');
    return response.text();
  })
  .then(text => {
    console.log(text);
  });
```

See also

- WebSocket
- Representational state transfer (REST)

References

1. Mahemoff, Michael (2006). *Ajax Design Patterns*. O'Reilly. p. 92. ISBN 978-0-596-10180-0. "Javascript lacks a portable mechanism for general network communication[.] ... But thanks to the XMLHttpRequest object, ... Javascript code can make HTTP calls back to its originating server[.]"
2. Mahemoff, Michael (2006). *Ajax Design Patterns*. O'Reilly. p. 92. ISBN 978-0-596-10180-0.
3. "Article on the history of XMLHTTP by an original developer" (https://web.archive.org/web/20090130092236/http://www.alexhopmann.com/xmlhttp.htm) . Alexhopmann.com. 2007-01-31. Archived from the original (http://www.alexhopmann.com/xmlhttp.htm) on 2009-01-30. Retrieved 2009-07-14. "The reality is that the client architecture of GMail appears to follow the rough design of the Exchange 2000 implementation of Outlook Web Access for IE5 and later which shipped way back in 2000."
4. Mahemoff, Michael (2006). *Ajax Design Patterns*. O'Reilly. p. 93. ISBN 978-0-596-10180-0.
5. "Archived news from Mozillazine stating the release date of Safari 1.2" (https://web.archive.org/web/20090602041255/http://weblogs.mozillazine.org/hyatt/archives/2004_02.html) . Weblogs.mozillazine.org. Archived from the original (http://weblogs.mozillazine.org/hyatt/archives/2004_02.html) on 2009-06-02. Retrieved 2009-07-14.
6. "Press release stating the release date of Opera 8.0 from the Opera website" (http://www.opera.com/press/releases/2005/06/16/) . Opera.com. 2005-04-19. Retrieved 2009-07-14.
7. "Specification of the XMLHttpRequest object from the Level 1 W3C Working Draft released on April 5th, 2006" (http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/) . W3.org. Retrieved 2009-07-14.
8. "Specification of the XMLHttpRequest object from the Level 2 W3C Working Draft released on February 25th, 2008" (http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080225/) . W3.org. Retrieved 2009-07-14.
9. "XMLHttpRequest Editor's Draft 5 December 2011" (http://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html) . w3.org. Retrieved 5 December 2011.

10. van Kesteren, Anne (February 19, 2024). XMLHttpRequest Living Standard (https://xhr.spec.whatwg.org/commit-snapshots/fdd619d0a13fe4d8af1c9ffdb6de24cb88c53cc0/) (Report). Retrieved 2024-04-09.
11. Holdener, Anthony T. III (2008). *Ajax: The Definitive Guide*. pp. 70–71, 76.
12. van Kesteren 2024, 3.5.2.
13. van Kesteren 2024, 3.5.6.
14. van Kesteren 2024, 3.6.9.
15. van Kesteren 2024, 3.5.7.
16. van Kesteren 2024, 3.5.3.
17. "XMLHttpRequest Level 2" (http://www.w3.org/TR/XMLHttpRequest2/) . Retrieved 2013-11-14.
18. "Can I use Cross-Origin Resource Sharing?" (http://caniuse.com/cors) . Retrieved 2013-11-14.
19. "XDomainRequest - Restrictions, Limitations and Workarounds" (http://blogs.msdn.com/b/ieinternals/archive/2010/05/13/xdomainrequest-restrictions-limitations-and-workarounds.aspx) . Retrieved 2013-11-14.
20. "7.1.5 Cross-Origin Request with Preflight" (http://www.w3.org/TR/cors/#cross-origin-request-with-preflight-0) . Retrieved 2014-04-25.
21. "Fetch Standard" (https://fetch.spec.whatwg.org/) .

Notes

a. The standard was edited by Anne van Kesteren of Opera Software and Dean Jackson of W3C.

External links

- XMLHttpRequest Living Standard (https://xhr.spec.whatwg.org/) by the WHATWG
- XMLHttpRequest Level 1 (https://www.w3.org/TR/XMLHttpRequest/) draft by the W3C