Cross-origin resource sharing

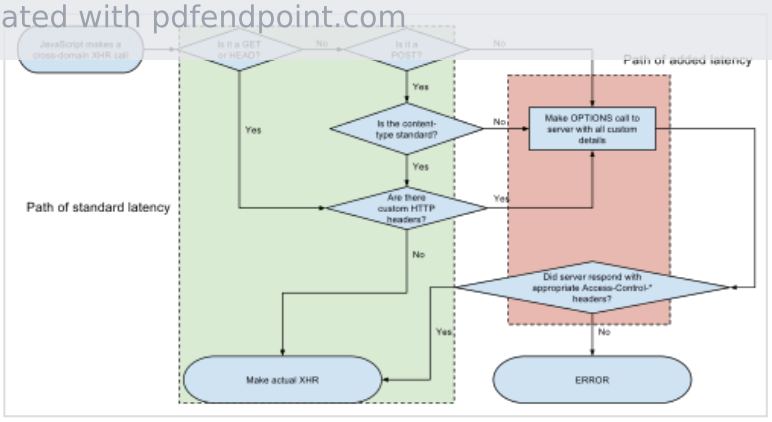
In computing, **cross-origin resource sharing** (**CORS**) is a mechanism to safely bypass the same-origin policy; that is, it allows a web page to access restricted resources from a web server on a domain name different from the domain that served the web page.

A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos. Certain "cross-domain" HTTP requests, notably Ajax requests, are forbidden by default by the same-origin security policy. CORS defines a way in which a web browser and server can interact to determine whether it is safe to allow the cross-origin request.[1] It allows for more freedom and functionality than purely same-origin requests, but is more secure than simply allowing all cross-origin requests.

The specification for CORS is included as part of the WHATWG's Fetch Living Standard.[2] This specification describes how CORS is currently implemented in browsers.[3] An earlier specification was published as a W3C Recommendation.[4]

Technical overview

For HTTP requests made from JavaScript that can't be made by using a <form> tag pointing to another domain or containing non-safelisted headers, the specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP OPTIONS request method, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method. Servers can also notify clients whether "credentials" (including Cookies and HTTP Authentication data) should be sent with requests.[5]



Path of an XMLHttpRequest (XHR) through CORS

Simple request example

Suppose a user visits http://www.example.com and the page attempts a cross-origin request to fetch data from http://service.example.com. A CORS-compatible browser will attempt to make a cross-origin request to service.example.com as follows.

1. The browser sends the GET request with an extra `Origin` HTTP header to service.example.com containing the domain that served the parent page:

   ```
   Origin: http://www.example.com
   ```

2. The server at service.example.com sends one of these three responses:

   - The requested data along with an `Access-Control-Allow-Origin` (ACAO) header in its response indicating the requests from the origin are allowed. For example in this case it should be:

     ```
     Access-Control-Allow-Origin: http://www.example.com
     ```

   - The requested data along with an `Access-Control-Allow-Origin` (ACAO) header with a wildcard indicating that the requests from all domains are allowed:

     ```
     Access-Control-Allow-Origin: *
     ```

   - An error page if the server does not allow a cross-origin request[6]

A wildcard same-origin policy is appropriate when a page or API response is intended to be accessible to any code on any site. A freely available web font on a public hosting service like Google Fonts is an example.

The value of "*" is special in that it does not allow requests to supply credentials, meaning that it does not allow HTTP authentication, client-side SSL certificates, or cookies to be sent in the cross-domain request.[7]

Note that in the CORS architecture, the Access-Control-Allow-Origin header is being set by the external web service (*service.example.com*), not the original web application server (*www.example.com*). Here, *service.example.com* uses CORS to permit the browser to authorize *www.example.com* to make requests to *service.example.com*.

If a site specifies the header "Access-Control-Allow-Credentials:true", third-party sites may be able to carry out privileged actions and retrieve sensitive information.

Preflight example

When performing certain types of cross-domain Ajax requests, modern browsers that support CORS will initiate an extra "preflight" request to determine whether they have permission to perform the action. Cross-origin requests are preflighted this way because they may have implications to user data.

```
OPTIONS /
Host: service.example.com
Origin: http://www.example.com
Access-Control-Request-Method: PUT
```

If service.example.com is willing to accept the action, it may respond with the following headers:

```
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Methods: PUT
```

The browser will then make the actual request. If service.example.com does not accept cross-site requests from this origin then it will respond with error to the OPTIONS request and the browser will not make the actual request.

Headers

The HTTP headers that relate to CORS are:

Request headers

- `Origin`
- `Host`
- `Access-Control-Request-Method`
- `Access-Control-Request-Headers`

Response headers

- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Credentials`
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age`
- `Access-Control-Allow-Methods`
- `Access-Control-Allow-Headers`

Browser support

CORS is supported by all browsers based on the following layout engines:

- Blink- and Chromium-based browsers (Chrome 28+,[8][9] Opera 15+,[8] Amazon Silk, Android's 4.4+ WebView and Qt's WebEngine)
- Gecko 1.9.1 (Firefox 3.5,[10] SeaMonkey 2.0[11]) and above.
- MSHTML/Trident 6.0 (Internet Explorer 10) has native support.[12] MSHTML/Trident 4.0 & 5.0 (Internet Explorer 8 & 9) provide partial support via the XDomainRequest object.[13]
- Presto-based browsers (Opera) implement CORS as of Opera 12.00[14] and Opera Mobile 12, but not Opera Mini.[15]
- WebKit (Initial revision uncertain, Safari 4 and above,[16] Google Chrome 3 and above, possibly earlier).[17]
- Microsoft Edge All versions.[18]

History

Cross-origin support was originally proposed by Matt Oshry, Brad Porter, and Michael Bodell of Tellme Networks in March 2004 for inclusion in VoiceXML 2.1[19] to allow safe cross-origin data requests by VoiceXML browsers. The mechanism was deemed general in nature and not specific to VoiceXML and was subsequently separated into an implementation NOTE.[20] The WebApps Working Group of the W3C with participation from the major browser vendors began to formalize the NOTE into a W3C Working Draft on track toward formal W3C Recommendation status.

In May 2006 the first W3C Working Draft was submitted.[21] In March 2009 the draft was renamed to "Cross-Origin Resource Sharing"[22] and in January 2014 it was accepted as a W3C Recommendation.[23]

CORS vs JSONP

The main advantage of JSONP was its ability to work on legacy browsers which predate CORS support (Opera Mini and Internet Explorer 9 and earlier). CORS is now supported by most modern web browsers,[24] and can be used as a modern alternative to the JSONP pattern. The benefits of CORS are:

- While JSONP supports only the GET request method, CORS also supports other types of HTTP requests.
- CORS enables a web programmer to use regular XMLHttpRequest, which supports better error handling than JSONP.
- While JSONP can cause cross-site scripting (XSS) issues when the external site is compromised, CORS allows websites to manually parse responses to increase security.[1]

See also

- Content Security Policy
- Cross-document messaging
- Cross site leaks

References

1. "Cross-domain Ajax with Cross-Origin Resource Sharing" (http://www.nczonline.net/blog/2010/05/25/cross-domain-ajax-with-cross-origin-resource-sharing/). NCZOnline. 25 May 2010. Retrieved 2012-07-05.
2. "Fetch Living Standard" (https://fetch.spec.whatwg.org/).
3. "WebAppSec Working Group Minutes" (https://www.w3.org/2017/08/16-webappsec-minutes.html#item03).
4. "Cross-Origin Resource Sharing" (http://www.w3.org/TR/cors/).
5. "Cross-Origin Resource Sharing (CORS) - HTTP | MDN" (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS). *developer.mozilla.org*. 10 May 2023. Retrieved 7 June 2023.
6. "CORS errors - HTTP | MDN" (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS/Errors). *developer.mozilla.org*. 2023-05-10. Retrieved 2023-07-04.
7. [1] (https://fetch.spec.whatwg.org/#cors-protocol-and-credentials). W3.org. Retrieved on 2021-31-07.
8. "Blink" (http://www.quirksmode.org/blog/archives/2013/04/blink.html). QuirksBlog. April 2013. Retrieved 4 April 2013.
9. "Google going its own way, forking WebKit rendering engine" (https://arstechnica.com/information-technology/2013/04/google-going-its-own-way-forking-webkit-rendering-engine/). Ars Technica. April 2013. Retrieved 4 April 2013.
10. "HTTP access control (CORS) - MDN" (https://web.archive.org/web/20100527153021/https://developer.mozilla.org/En/HTTP_access_control). Developer.mozilla.org. Archived from the original (https://developer.mozilla.org/En/HTTP_access_control) on 2010-05-27. Retrieved 2012-07-05.
11. "Gecko - MDN" (https://web.archive.org/web/20120803092112/https://developer.mozilla.org/en/Gecko). Developer.mozilla.org. 2012-06-08. Archived from the original (https://developer.mozilla.org/en/Gecko) on 2012-08-03. Retrieved 2012-07-05.
12. Tony Ross; Program Manager; Internet Explorer (2012-02-09). "CORS for XHR in IE10" (http://blogs.msdn.com/b/ie/archive/2012/02/09/cors-for-xhr-in-ie10.aspx). MSDN. Retrieved 2012-12-14.
13. "cross-site xmlhttprequest with CORS" (https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/). MOZILLA. Retrieved 2012-09-05.
14. David Honneffer, Documentation Specialist (2012-06-14). "12.00 for UNIX Changelog" (https://web.archive.org/web/20120618170555/http://www.opera.com/docs/changelogs/unix/1200/). Opera. Archived from the original (http://www.opera.com/docs/changelogs/unix/1200/) on 2012-06-18. Retrieved 2012-07-05.
15. David Honneffer, Documentation Specialist (2012-04-23). "Opera Software: Web specifications support in Opera Presto 2.10" (http://www.opera.com/docs/specs/presto2.10/#m210-236). Opera.com. Retrieved 2012-07-05.

16. on July 6, 2009 by Arun Ranganathan (2009-07-06). "cross-site xmlhttprequest with CORS ☆ Mozilla Hacks – the Web developer blog" (https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/). Hacks.mozilla.org. Retrieved 2012-07-05.
17. "59940: Apple Safari WebKit Cross-Origin Resource Sharing Bypass" (https://archive.today/20120719142244/http://osvdb.org/59940). Osvdb.org. Archived from the original (http://osvdb.org/59940) on 2012-07-19. Retrieved 2012-07-05.
18. "Microsoft Edge deverloper's guide" (https://docs.microsoft.com/en-us/microsoft-edge/dev-guide/performance/xmlhttprequest/). 21 December 2023.
19. "Voice Extensible Markup Language (VoiceXML) 2.1" (https://www.w3.org/TR/2004/WD-voicexml21-20040323/#sec-data-security). W3.org. 2004-03-23. Retrieved 2012-07-05.
20. "Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction 1.0" (http://www.w3.org/TR/2005/NOTE-access-control-20050613/). W3.org. Retrieved 2012-07-05.
21. "Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction 1.0 W3C - Working Draft 17 May 2006" (http://www.w3.org/TR/2006/WD-access-control-20060517/). W3.org. Retrieved 17 August 2015.
22. "Cross-Origin Resource Sharing - W3C Working Draft 17 March 2009" (http://www.w3.org/TR/2009/WD-cors-20090317/). W3.org. Retrieved 17 August 2015.
23. "Cross-Origin Resource Sharing - W3C Recommendation 16 January 2014" (http://www.w3.org/TR/2014/REC-cors-20140116/). W3.org. Retrieved 17 August 2015.
24. "When can I use... Cross Origin Resource Sharing" (http://caniuse.com/#feat=cors). caniuse.com. Retrieved 2012-07-12.

External links

- Fetch Living Standard (https://fetch.spec.whatwg.org/) (the current specification for CORS)
- Setting CORS on Apache with correct response headers allowing everything through (https://benjaminhorn.io/code/setting-cors-cross-origin-resource-sharing-on-apache-with-correct-response-headers-allowing-everything-through/)
- Detailed how-to information for enabling CORS support in various (web) servers (https://enable-cors.org/server.html)
- *HTML5 Rocks* explains how CORS works in detail (http://www.html5rocks.com/en/tutorials/cors/)
- Online CORS misconfiguration scanner (https://helpertools.app/web-security/view/cors-scanner) Archived (https://web.archive.org/web/20200810161738/https://helpertools.app/web-security/view/cors-scanner) 2020-08-10 at the Wayback Machine

---