Cross-origin resource sharing

From Wikipedia, the free encyclopedia

Cross-origin resource sharing (**CORS**) is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was served.[1] A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos.[2] Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy.

CORS defines a way in which a browser and server can interact to determine whether or not it is safe to allow the cross-origin request.[3] 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 The specification for CORS was originally published as a W3C Recommendation[4] but that document is obsolete.[5] The current actively-maintained specification that defines CORS is the Fetch Living Standard.[6]

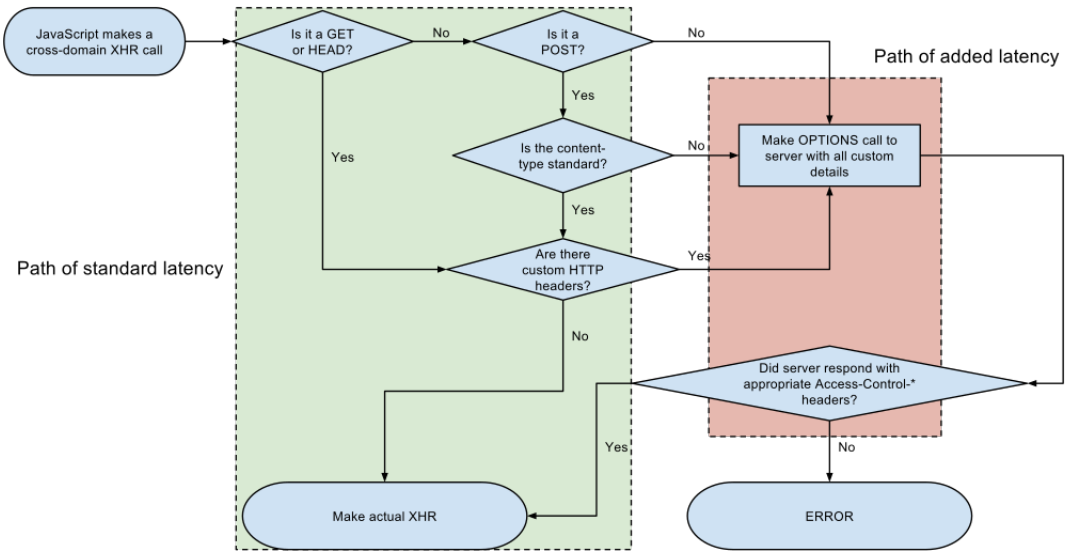
Contents

How CORS works

The CORS standard describes new HTTP headers which provide browsers and servers a way to request remote URLs only when they have permission. Although some validation and authorization can be performed by the server, it is generally the browser's responsibility to support these headers and honor the restrictions they impose.

For Ajax and HTTP request methods that can modify data (usually HTTP methods other than GET, or for POST usage with certain MIME types), 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.[7]



Simple example

This is generally not appropriate when using the same-origin security policy. When a CORS-compatible browser attempts to make a cross-origin request:

1. The browser sends the OPTIONS request with an `Origin` HTTP header. The value of this header is the domain that served the parent page. When a page from http://www.example.com attempts to access a user's data in service.example.com, the following request header would be sent to service.example.com:

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

2. The server at service.example.com may respond with:
   - An `Access-Control-Allow-Origin` (ACAO) header in its response indicating which origin sites are allowed. For example:

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

   - An error page if the server does not allow the cross-origin request
   - An `Access-Control-Allow-Origin` (ACAO) header with a wildcard that allows all domains:

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

A wildcard same-origin policy is appropriate when a page or API response is considered completely public content and it is intended to be accessible to everyone, including any code on any site. For example, a freely-available web font on a public hosting service like Google Fonts.

A wildcard same-origin policy is also widely and appropriately used in the object-capability model, where pages have unguessable URLs and are meant to be accessible to anyone who knows the secret.

The value of "*" is special in that it does not allow requests to supply credentials, meaning HTTP authentication, client-side SSL certificates, nor does it allow cookies to be sent.[8]

Note that in the CORS architecture, the ACAO header is being set by the external web service (*service.example.com*), not the original web application server (*www.example.com*). CORS allows the external web service to authorise the web application to use its services and does not control external services accessed by the web application. For the latter, Content Security Policy should be used (`connect-src` directive)...

Preflight example

When performing certain types of cross-domain Ajax requests, modern browsers that support CORS will insert an extra "preflight" request to determine whether they have permission to perform the action.

```
OPTIONS /
Host: service.example.com
Origin: http://www.example.com
```

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, DELETE
```

Headers

The HTTP headers that relate to CORS are

Request headers

- Origin
- 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+,[9][10] Opera 15+,[9] Amazon Silk, Android's 4.4+ WebView and Qt's WebEngine)
- Gecko 1.9.1 (Firefox 3.5,[11] SeaMonkey 2.0[12]) and above.
- MSHTML/Trident 6.0 (Internet Explorer 10) has native support.[13] MSHTML/Trident 4.0 & 5.0 (Internet Explorer 8 & 9) provide partial support via the XDomainRequest object.[1]
- 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,[1] Google Chrome 3 and above, possibly earlier).[16]
- Microsoft Edge All versions. [17]

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[18] 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.[19] 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.[20] In March 2009 the draft was renamed to "Cross-Origin Resource Sharing"[21] and in January 2014 it was accepted as a W3C Recommendation.[22]

CORS vs JSONP

CORS can be used as a modern alternative to the JSONP pattern. While JSONP supports only the GET request method, CORS also supports other types of HTTP requests. Using CORS enables a web programmer to use regular XMLHttpRequest, which supports better error handling than JSONP. On the other hand, JSONP works on legacy browsers which predate CORS support. CORS is supported by most modern web browsers. Also, while JSONP can cause cross-site scripting (XSS) issues when the external site is compromised, CORS allows websites to manually parse responses to ensure security.[3][23]

See also

- Content Security Policy
- Cross-document messaging
- JSONP

References

1. on July 6, 2009 by Arun Ranganathan (2009-07-06). "cross-site xmlhttprequest with CORS ☆ Mozilla Hacks – the Web developer blog" (http://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/). Hacks.mozilla.org. Retrieved 2012-07-05.
2. "Same-origin policy / Cross-origin network access" (https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#Cross-origin_network_access). MDN.
3. "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. Retrieved 2012-07-05.
4. "Cross-Origin Resource Sharing" (http://www.w3.org/TR/cors/).
5. "WebAppSec Working Group Minutes" (https://www.w3.org/2017/08/16-webappsec-minutes.html#item03).
6. "Fetch Living Standard" (https://fetch.spec.whatwg.org/).
7. "cross-site xmlhttprequest with CORS" (http://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/). MOZILLA. Retrieved 2012-09-05.
8. Cross-Origin Resource Sharing (http://www.w3.org/TR/cors/#resource-requests). W3.org. Retrieved on 2014-04-12.
9. "Blink" (http://www.quirksmode.org/blog/archives/2013/04/blink.html). QuirksBlog. April 2013. Retrieved 4 April 2013.
10. "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.
11. "HTTP access control (CORS) - MDN" (https://developer.mozilla.org/En/HTTP_access_control). Developer.mozilla.org. Retrieved 2012-07-05.
12. "Gecko - MDN" (https://developer.mozilla.org/en/Gecko). Developer.mozilla.org. 2012-06-08. Retrieved 2012-07-05.
13. 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.
14. David Honneffer, Documentation Specialist (2012-06-14). "12.00 for UNIX Changelog" (http://www.opera.com/docs/changelogs/unix/1200/). Opera. 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. "59940: Apple Safari WebKit Cross-Origin Resource Sharing Bypass" (http://osvdb.org/59940). Osvdb.org. Retrieved 2012-07-05.
17. "Microsoft Edge deverloper's guide" (https://docs.microsoft.com/en-us/microsoft-edge/dev-guide/performance/xmlhttprequest/).
18. "Voice Extensible Markup Language (VoiceXML) 2.1" (http://www.w3.org/TR/2004/WD-voicexml21-20040323/). W3.org. 2004-03-23. Retrieved 2012-07-05.
19. "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.
20. "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.
21. "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.

22. "Cross-Origin Resource Sharing - W3C Recommendation 16 January 2014" (http://www.w3.org/TR/2014/REC-cors-20140116/). W3.org. Retrieved 17 August 2015.
23. "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)
- MDN *HTTP access control (CORS)* article (https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_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/)
- W3C *CORS for Developers* guide (https://w3c.github.io/webappsec-cors-for-developers/)
- Test your browser for CORS support (https://test-cors.appspot.com/#technical)
- Detailed how-to information for dealing with some common CORS-related problems (https://stackoverflow.com/questions/43871637/no-access-control-allow-origin-header-is-present-on-the-requested-resource-whe/43881141#43881141), including:
  - How to avoid the CORS preflight
  - How to fix *"Access-Control-Allow-Origin header must not be the wildcard"* problems
  - How to use a CORS proxy to get around *"No Access-Control-Allow-Origin header is present on the requested resource"* problems

Retrieved from "https://en.wikipedia.org/w/index.php?title=Cross-origin_resource_sharing&oldid=811592953"
