当一个资源从与该资源本身所在的服务器不同的域或端口请求一个资源时，资源会发起一个**跨域 HTTP 请求**。

比如，站点 http://domain-a.com 的某 HTML 页面通过 [<img> 的 src](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Element/Img#Attributes)请求 http://domain-b.com/image.jpg。网络上的许多页面都会加载来自不同域的CSS样式表，图像和脚本等资源。

出于安全原因，浏览器限制从脚本内发起的跨源HTTP请求。 例如，XMLHttpRequest和Fetch API遵循同源策略。 这意味着使用这些API的Web应用程序只能从加载应用程序的同一个域请求HTTP资源，除非使用CORS头文件。

  （译者注：~~这段描述跨域不准确，~~跨域并~~非~~不一定是浏览器限制了发起跨站请求，~~而~~也可能是跨站请求可以正常发起，但是返回结果被浏览器拦截了。最好的例子是 CSRF 跨站攻击原理，请求是发送到了后端服务器无论是否跨域！注意：有些浏览器不允许从 HTTPS 的域跨域访问 HTTP，比如  Chrome 和 Firefox，这些浏览器在请求还未发出的时候就会拦截请求，这是一个特例。）

跨域资源共享（ [CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS) ）机制允许 Web 应用服务器进行跨域访问控制，从而使跨域数据传输得以安全进行。浏览器支持在 API 容器中（例如 [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 或 [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) ）使用 CORS，以降低跨域 HTTP 请求所带来的风险。

这篇文章适用于网站管理员、后端和前端开发者。CORS 需要客户端和服务器同时支持。目前，所有浏览器都支持该机制。 对于服务端的支持，开发者可以阅读补充材料 [cross-origin sharing from a server perspective (with PHP code snippets)](https://developer.mozilla.org/En/Server-Side_Access_Control) 。

跨域资源共享标准（ [cross-origin sharing standard](http://www.w3.org/TR/cors/) ）允许在下列场景中使用跨域 HTTP 请求：

* 前文提到的由 [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 或 [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) 发起的跨域 HTTP 请求。
* Web 字体 (CSS 中通过 @font-face 使用跨域字体资源), [因此，网站就可以发布 TrueType 字体资源，并只允许已授权网站进行跨站调用](http://www.webfonts.info/wiki/index.php?title=%40font-face_support_in_Firefox)。
* [WebGL 贴图](https://developer.mozilla.org/zh-CN/docs/Web/API/WebGL_API/Tutorial/Using_textures_in_WebGL)
* 使用 [drawImage](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage) 将 Images/video 画面绘制到 canvas
* 样式表（使用 [CSSOM](https://developer.mozilla.org/en-US/docs/Web/CSS/CSSOM_View)）
* Scripts (未处理的异常)

本文概述了跨域资源共享机制及其所涉及的 HTTP 首部字段。

**概述**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#%E6%A6%82%E8%BF%B0)

跨域资源共享标准新增了一组 HTTP 首部字段，允许服务器声明哪些源站有权限访问哪些资源。另外，规范要求，对那些可能对服务器数据产生副作用的 HTTP 请求方法（特别是 [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET) 以外的 HTTP 请求，或者搭配某些 MIME 类型的 [POST](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/POST) 请求），浏览器必须首先使用 [OPTIONS](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/OPTIONS) 方法发起一个预检请求（preflight request），从而获知服务端是否允许该跨域请求。服务器确认允许之后，才发起实际的 HTTP 请求。在预检请求的返回中，服务器端也可以通知客户端，是否需要携带身份凭证（包括 [Cookies](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Cookies)和 HTTP 认证相关数据）。

接下来的内容将讨论相关场景，并剖析该机制所涉及的 HTTP 首部字段。

**若干访问控制场景**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#%E8%8B%A5%E5%B9%B2%E8%AE%BF%E9%97%AE%E6%8E%A7%E5%88%B6%E5%9C%BA%E6%99%AF)

这里，我们使用三个场景来解释跨域资源共享机制的工作原理。这些例子都使用 [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 对象。

本文中的 JavaScript 代码片段都可以从 <http://arunranga.com/examples/access-control/> 获得。另外，使用支持跨域  [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 的浏览器访问该地址，可以看到代码的实际运行结果。

关于服务端对跨域资源共享的支持的讨论，请参见这篇文章： [Server-Side\_Access\_Control (CORS)](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Server-Side_Access_Control)。

简单请求

某些请求不会触发 [CORS 预检请求](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS#Preflighted_requests)。本文称这样的请求为“简单请求”，请注意，该术语并不属于 [Fetch](https://fetch.spec.whatwg.org/) （其中定义了 CORS）规范。若请求满足所有下述条件，则该请求可视为“简单请求”：

* 使用下列方法之一：
  + [GET](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/GET)
  + [HEAD](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/HEAD)
  + [POST](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/POST)
* Fetch 规范定义了[对 CORS 安全的首部字段集合](https://fetch.spec.whatwg.org/#cors-safelisted-request-header)，不得人为设置该集合之外的其他首部字段。该集合为：
  + [Accept](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept)
  + [Accept-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept-Language)
  + [Content-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Language)
  + [Content-Type](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Type) （需要注意额外的限制）
  + [DPR](http://httpwg.org/http-extensions/client-hints.html#dpr)
  + [Downlink](http://httpwg.org/http-extensions/client-hints.html#downlink)
  + [Save-Data](http://httpwg.org/http-extensions/client-hints.html#save-data)
  + [Viewport-Width](http://httpwg.org/http-extensions/client-hints.html#viewport-width)
  + [Width](http://httpwg.org/http-extensions/client-hints.html#width)
* [Content-Type](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Type) 的值仅限于下列三者之一：
  + text/plain
  + multipart/form-data
  + application/x-www-form-urlencoded
* 请求中的任意[XMLHttpRequestUpload](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequestUpload) 对象均没有注册任何事件监听器；[XMLHttpRequestUpload](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequestUpload) 对象可以使用 [XMLHttpRequest.upload](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest/upload) 属性访问。
* 请求中没有使用 [ReadableStream](https://developer.mozilla.org/zh-CN/docs/Web/API/ReadableStream) 对象。

**注意:** 这些跨域请求与浏览器发出的其他跨域请求并无二致。如果服务器未返回正确的响应首部，则请求方不会收到任何数据。因此，那些不允许跨域请求的网站无需为这一新的 HTTP 访问控制特性担心。

**注意:** WebKit Nightly 和 Safari Technology Preview 为[Accept](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept), [Accept-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept-Language), 和 [Content-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Language) 首部字段的值添加了额外的限制。如果这些首部字段的值是“非标准”的，WebKit/Safari 就不会将这些请求视为“简单请求”。WebKit/Safari 并没有在文档中列出哪些值是“非标准”的，不过我们可以在这里找到相关讨论：[Require preflight for non-standard CORS-safelisted request headers Accept, Accept-Language, and Content-Language](https://bugs.webkit.org/show_bug.cgi?id=165178), [Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS](https://bugs.webkit.org/show_bug.cgi?id=165566), and [Switch to a blacklist model for restricted Accept headers in simple CORS requests](https://bugs.webkit.org/show_bug.cgi?id=166363)。其它浏览器并不支持这些额外的限制，因为它们不属于规范的一部分。

比如说，假如站点 http://foo.example 的网页应用想要访问 http://bar.other 的资源。http://foo.example 的网页中可能包含类似于下面的 JavaScript 代码：

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/public-data/';

function callOtherDomain() {

if(invocation) {

invocation.open('GET', url, true);

invocation.onreadystatechange = handler;

invocation.send();

}

}

客户端和服务器之间使用 CORS 首部字段来处理跨域权限：

分别检视请求报文和响应报文：

GET /resources/public-data/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Referer: http://foo.example/examples/access-control/simpleXSInvocation.html

Origin: http://foo.example

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 00:23:53 GMT

Server: Apache/2.0.61

Access-Control-Allow-Origin: \*

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Transfer-Encoding: chunked

Content-Type: application/xml

[XML Data]

第 1~10 行是请求首部。第10行 的请求首部字段 [Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Origin) 表明该请求来源于 http://foo.exmaple。

第 13~22 行是来自于 http://bar.other 的服务端响应。响应中携带了响应首部字段 [Access-Control-Allow-Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Origin)（第 16 行）。使用 [Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Origin) 和 [Access-Control-Allow-Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) 就能完成最简单的访问控制。本例中，服务端返回的 Access-Control-Allow-Origin: \* 表明，该资源可以被**任意**外域访问。如果服务端仅允许来自 http://foo.example 的访问，该首部字段的内容如下：

Access-Control-Allow-Origin: http://foo.example

现在，除了 http://foo.example，其它外域均不能访问该资源（该策略由请求首部中的 ORIGIN 字段定义，见第10行）。Access-Control-Allow-Origin 应当为 \* 或者包含由 Origin 首部字段所指明的域名。

预检请求

与前述简单请求不同，“需预检的请求”要求必须首先使用 [OPTIONS](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/OPTIONS)   方法发起一个预检请求到服务器，以获知服务器是否允许该实际请求。"预检请求“的使用，可以避免跨域请求对服务器的用户数据产生未预期的影响。

当请求满足下述任一条件时，即应首先发送预检请求：

* 使用了下面任一 HTTP 方法：
  + [PUT](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/PUT)
  + [DELETE](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/DELETE)
  + [CONNECT](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/CONNECT)
  + [OPTIONS](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/OPTIONS)
  + [TRACE](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/TRACE)
  + [PATCH](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Methods/PATCH)
* 人为设置了[对 CORS 安全的首部字段集合](https://fetch.spec.whatwg.org/#cors-safelisted-request-header)之外的其他首部字段。该集合为：
  + [Accept](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept)
  + [Accept-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept-Language)
  + [Content-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Language)
  + [Content-Type](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Type) (but note the additional requirements below)
  + [DPR](http://httpwg.org/http-extensions/client-hints.html#dpr)
  + [Downlink](http://httpwg.org/http-extensions/client-hints.html#downlink)
  + [Save-Data](http://httpwg.org/http-extensions/client-hints.html#save-data)
  + [Viewport-Width](http://httpwg.org/http-extensions/client-hints.html#viewport-width)
  + [Width](http://httpwg.org/http-extensions/client-hints.html#width)
* [Content-Type](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Type) 的值不属于下列之一:
  + application/x-www-form-urlencoded
  + multipart/form-data
  + text/plain
* 请求中的[XMLHttpRequestUpload](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequestUpload) 对象注册了任意多个事件监听器。
* 请求中使用了[ReadableStream](https://developer.mozilla.org/zh-CN/docs/Web/API/ReadableStream)对象。

**注意:** WebKit Nightly 和 Safari Technology Preview 为[Accept](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept), [Accept-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Accept-Language), 和 [Content-Language](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Content-Language) 首部字段的值添加了额外的限制。如果这些首部字段的值是“非标准”的，WebKit/Safari 就不会将这些请求视为“简单请求”。WebKit/Safari 并没有在文档中列出哪些值是“非标准”的，不过我们可以在这里找到相关讨论：[Require preflight for non-standard CORS-safelisted request headers Accept, Accept-Language, and Content-Language](https://bugs.webkit.org/show_bug.cgi?id=165178), [Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS](https://bugs.webkit.org/show_bug.cgi?id=165566), and [Switch to a blacklist model for restricted Accept headers in simple CORS requests](https://bugs.webkit.org/show_bug.cgi?id=166363)。其它浏览器并不支持这些额外的限制，因为它们不属于规范的一部分。

如下是一个需要执行预检请求的 HTTP 请求：

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/post-here/';

var body = '<?xml version="1.0"?><person><name>Arun</name></person>';

function callOtherDomain(){

if(invocation)

{

invocation.open('POST', url, true);

invocation.setRequestHeader('X-PINGOTHER', 'pingpong');

invocation.setRequestHeader('Content-Type', 'application/xml');

invocation.onreadystatechange = handler;

invocation.send(body);

}

}

......

上面的代码使用 POST 请求发送一个 XML 文档，该请求包含了一个自定义的请求首部字段（X-PINGOTHER: pingpong）。另外，该请求的 Content-Type 为 application/xml。因此，该请求需要首先发起“预检请求”。

1.OPTIONS /resources/post-here/ HTTP/1.1

2.Host: bar.other

3.User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

4.Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

5.Accept-Language: en-us,en;q=0.5

6.Accept-Encoding: gzip,deflate

7.Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

8.Connection: keep-alive

9.Origin: http://foo.example

10.Access-Control-Request-Method: POST

11.Access-Control-Request-Headers: X-PINGOTHER, Content-Type

12.

13.

14.HTTP/1.1 200 OK

15.Date: Mon, 01 Dec 2008 01:15:39 GMT

16.Server: Apache/2.0.61 (Unix)

17.Access-Control-Allow-Origin: http://foo.example

18.Access-Control-Allow-Methods: POST, GET, OPTIONS

19.Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

20.Access-Control-Max-Age: 86400

21.Vary: Accept-Encoding, Origin

22.Content-Encoding: gzip

23.Content-Length: 0

24.Keep-Alive: timeout=2, max=100

25.Connection: Keep-Alive

26.Content-Type: text/plain

预检请求完成之后，发送实际请求：

POST /resources/post-here/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

X-PINGOTHER: pingpong

Content-Type: text/xml; charset=UTF-8

Referer: http://foo.example/examples/preflightInvocation.html

Content-Length: 55

Origin: http://foo.example

Pragma: no-cache

Cache-Control: no-cache

<?xml version="1.0"?><person><name>Arun</name></person>

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:15:40 GMT

Server: Apache/2.0.61 (Unix)

Access-Control-Allow-Origin: http://foo.example

Vary: Accept-Encoding, Origin

Content-Encoding: gzip

Content-Length: 235

Keep-Alive: timeout=2, max=99

Connection: Keep-Alive

Content-Type: text/plain

[Some GZIP'd payload]

浏览器检测到，从 JavaScript 中发起的请求需要被预检。从上面的报文中，我们看到，第 1~12 行发送了一个使用 OPTIONS 方法的“预检请求”。 OPTIONS 是 HTTP/1.1 协议中定义的方法，用以从服务器获取更多信息。该方法不会对服务器资源产生影响。 预检请求中同时携带了下面两个首部字段：

Access-Control-Request-Method: POST

Access-Control-Request-Headers: X-PINGOTHER, Content-Type

首部字段 Access-Control-Request-Method 告知服务器，实际请求将使用 POST 方法。首部字段 Access-Control-Request-Headers 告知服务器，实际请求将携带两个自定义请求首部字段：X-PINGOTHER 与 Content-Type。服务器据此决定，该实际请求是否被允许。

第14~26 行为预检请求的响应，表明服务器将接受后续的实际请求。重点看第 17~20 行：

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

Access-Control-Max-Age: 86400

首部字段 Access-Control-Allow-Methods 表明服务器允许客户端使用 POST, GET 和 OPTIONS 方法发起请求。该字段与 [HTTP/1.1 Allow: response header](http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.7) 类似，但仅限于在需要访问控制的场景中使用。

首部字段 Access-Control-Allow-Headers 表明服务器允许请求中携带字段 X-PINGOTHER与 Content-Type。与 Access-Control-Allow-Methods 一样，Access-Control-Allow-Headers 的值为逗号分割的列表。

最后，首部字段 Access-Control-Max-Age 表明该响应的有效时间为 86400 秒，也就是 24 小时。在有效时间内，浏览器无须为同一请求再次发起预检请求。请注意，浏览器自身维护了一个最大有效时间，如果该首部字段的值超过了最大有效时间，将不会生效。

**预检请求与重定向**

大多数浏览器不支持针对于预检请求的重定向。如果一个预检请求发生了重定向，浏览器将报告错误：

The request was redirected to 'https://example.com/foo', which is disallowed for cross-origin requests that require preflight

Request requires preflight, which is disallowed to follow cross-origin redirect

CORS 最初要求该行为，不过[在后续的修订中废弃了这一要求](https://github.com/whatwg/fetch/commit/0d9a4db8bc02251cc9e391543bb3c1322fb882f2)。

在浏览器的实现跟上规范之前，有两种方式规避上述报错行为：

* 在服务端去掉对预检请求的重定向；
* 将实际请求变成一个简单请求。

如果上面两种方式难以做到，我们仍有其他办法：

* 发出一个简单请求（使用  [Response.url](https://developer.mozilla.org/en-US/docs/Web/API/Response/url) 或 [XHR.responseURL](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseURL)）以判断真正的预检请求会返回什么地址。
* 发出另一个请求（真正的请求），使用在上一步通过[Response.url](https://developer.mozilla.org/en-US/docs/Web/API/Response/url) 或 [XMLHttpRequest.responseURL](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseURL)获得的URL。

不过，如果请求是由于存在 Authorization 字段而引发了预检请求，则这一方法将无法使用。这种情况只能由服务端进行更改。

附带身份凭证的请求

[Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) 与 CORS 的一个有趣的特性是，可以基于  [HTTP cookies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies) 和 HTTP 认证信息发送身份凭证。一般而言，对于跨域 [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest) 或 [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) 请求，浏览器**不会**发送身份凭证信息。如果要发送凭证信息，需要设置 [XMLHttpRequest](https://developer.mozilla.org/en/DOM/XMLHttpRequest) 的某个特殊标志位。

本例中，http://foo.example 的某脚本向 http://bar.other 发起一个GET 请求，并设置 Cookies：

var invocation = new XMLHttpRequest();

var url = 'http://bar.other/resources/credentialed-content/';

function callOtherDomain(){

if(invocation) {

invocation.open('GET', url, true);

invocation.withCredentials = true;

invocation.onreadystatechange = handler;

invocation.send();

}

}

第 7 行将 [XMLHttpRequest](https://developer.mozilla.org/en/DOM/XMLHttpRequest) 的 withCredentials 标志设置为 true，从而向服务器发送 Cookies。因为这是一个简单 GET 请求，所以浏览器不会对其发起“预检请求”。但是，如果服务器端的响应中未携带 Access-Control-Allow-Credentials: true ，浏览器将不会把响应内容返回给请求的发送者。

客户端与服务器端交互示例如下：

GET /resources/access-control-with-credentials/ HTTP/1.1

Host: bar.other

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Connection: keep-alive

Referer: http://foo.example/examples/credential.html

Origin: http://foo.example

Cookie: pageAccess=2

HTTP/1.1 200 OK

Date: Mon, 01 Dec 2008 01:34:52 GMT

Server: Apache/2.0.61 (Unix) PHP/4.4.7 mod\_ssl/2.0.61 OpenSSL/0.9.7e mod\_fastcgi/2.4.2 DAV/2 SVN/1.4.2

X-Powered-By: PHP/5.2.6

Access-Control-Allow-Origin: http://foo.example

Access-Control-Allow-Credentials: true

Cache-Control: no-cache

Pragma: no-cache

Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT

Vary: Accept-Encoding, Origin

Content-Encoding: gzip

Content-Length: 106

Keep-Alive: timeout=2, max=100

Connection: Keep-Alive

Content-Type: text/plain

[text/plain payload]

即使第 11 行指定了 Cookie 的相关信息，但是，如果 bar.other 的响应中缺失 [Access-Control-Allow-Credentials](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials): true（第 19 行），则响应内容不会返回给请求的发起者。

**附带身份凭证的请求与通配符**

对于附带身份凭证的请求，服务器不得设置 Access-Control-Allow-Origin 的值为“\*”。

这是因为请求的首部中携带了 Cookie 信息，如果 Access-Control-Allow-Origin 的值为“\*”，请求将会失败。而将 Access-Control-Allow-Origin 的值设置为 http://foo.example，则请求将成功执行。

另外，响应首部中也携带了 Set-Cookie 字段，尝试对 Cookie 进行修改。如果操作失败，将会抛出异常。

**HTTP 响应首部字段**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#HTTP_%E5%93%8D%E5%BA%94%E9%A6%96%E9%83%A8%E5%AD%97%E6%AE%B5)

本节列出了规范所定义的响应首部字段。上一小节中，我们已经看到了这些首部字段在实际场景中是如何工作的。

Access-Control-Allow-Origin

响应首部中可以携带一个 [Access-Control-Allow-Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) 字段，其语法如下:

Access-Control-Allow-Origin: <origin> | \*

其中，origin 参数的值指定了允许访问该资源的外域 URI。对于不需要携带身份凭证的请求，服务器可以指定该字段的值为通配符，表示允许来自所有域的请求。

例如，下面的字段值将允许来自 http://mozilla.com 的请求：

Access-Control-Allow-Origin: http://mozilla.com

如果服务端指定了具体的域名而非“\*”，那么响应首部中的 Vary 字段的值必须包含 Origin。这将告诉客户端：服务器对不同的源站返回不同的内容。

Access-Control-Expose-Headers

译者注：在跨域访问时，XMLHttpRequest对象的getResponseHeader()方法只能拿到一些最基本的响应头，Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma，如果要访问其他头，则需要服务器设置本响应头。

[Access-Control-Expose-Headers](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Expose-Headers) 头让服务器把允许浏览器访问的头放入白名单，例如：

Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header

这样浏览器就能够通过getResponseHeader访问X-My-Custom-Header和 X-Another-Custom-Header 响应头了。

Access-Control-Max-Age

[Access-Control-Max-Age](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Max-Age) 头指定了preflight请求的结果能够被缓存多久，请参考本文在前面提到的preflight例子。

Access-Control-Max-Age: <delta-seconds>

delta-seconds 参数表示preflight请求的结果在多少秒内有效。

Access-Control-Allow-Credentials

[Access-Control-Allow-Credentials](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Credentials) 头指定了当浏览器的credentials设置为true时是否允许浏览器读取response的内容。当用在对preflight预检测请求的响应中时，它指定了实际的请求是否可以使用credentials。请注意：简单 GET 请求不会被预检；如果对此类请求的响应中不包含该字段，这个响应将被忽略掉，并且浏览器也不会将相应内容返回给网页。

Access-Control-Allow-Credentials: true

上文已经讨论了[附带身份凭证的请求](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS#Requests_with_credentials)。

Access-Control-Allow-Methods

[Access-Control-Allow-Methods](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Methods) 首部字段用于预检请求的响应。其指明了实际请求所允许使用的 HTTP 方法。

Access-Control-Allow-Methods: <method>[, <method>]\*

相关示例见[这里](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#Preflighted_requests)。

Access-Control-Allow-Headers

[Access-Control-Allow-Headers](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Allow-Headers) 首部字段用于预检请求的响应。其指明了实际请求中允许携带的首部字段。

Access-Control-Allow-Headers: <field-name>[, <field-name>]\*

**HTTP 请求首部字段**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#HTTP_%E8%AF%B7%E6%B1%82%E9%A6%96%E9%83%A8%E5%AD%97%E6%AE%B5)

本节列出了可用于发起跨域请求的首部字段。请注意，这些首部字段无须手动设置。 当开发者使用 XMLHttpRequest 对象发起跨域请求时，它们已经被设置就绪。

Origin

[Origin](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Origin) 首部字段表明预检请求或实际请求的源站。

Origin: <origin>

origin 参数的值为源站 URI。它不包含任何路径信息，只是服务器名称。

**Note:** 有时候将该字段的值设置为空字符串是有用的，例如，当源站是一个 data URL 时。

注意，不管是否为跨域请求，ORIGIN 字段总是被发送。

Access-Control-Request-Method

[Access-Control-Request-Method](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Request-Method) 首部字段用于预检请求。其作用是，将实际请求所使用的 HTTP 方法告诉服务器。

Access-Control-Request-Method: <method>

相关示例见[这里](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS#Preflighted_requests)。

Access-Control-Request-Headers

[Access-Control-Request-Headers](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Access-Control-Request-Headers) 首部字段用于预检请求。其作用是，将实际请求所携带的首部字段告诉服务器。

Access-Control-Request-Headers: <field-name>[, <field-name>]\*

相关示例见[这里](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS)。

**规范**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#%E8%A7%84%E8%8C%83)

|  |  |  |
| --- | --- | --- |
| **Specification** | **Status** | **Comment** |
| [Fetch CORS](https://fetch.spec.whatwg.org/#cors-protocol) | Living Standard | New definition; supplants CORS specification. |
| Unknown | Unknown | Initial definition. |

**浏览器兼容性**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#%E6%B5%8F%E8%A7%88%E5%99%A8%E5%85%BC%E5%AE%B9%E6%80%A7)

[**We're converting our compatibility data into a machine-readable JSON format**](https://github.com/mdn/browser-compat-data). This compatibility table still uses the old format, because we haven't yet converted the data it contains. [**Find out how you can help!**](https://developer.mozilla.org/zh-CN/docs/MDN/Contribute/Structures/Compatibility_tables)

* **Desktop**

* Mobile

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Feature** | **Chrome** | **Firefox (Gecko)** | **Internet Explorer** | **Opera** | **Safari** |
| Basic support | 4 | 3.5 | 8 (via XDomainRequest) 10 | 12 | 4 |

**注：**

* IE 10 提供了对规范的完整支持，但在较早版本（8 和 9）中，CORS 机制是借由 XDomainRequest 对象完成的。
* Firefox 3.5 引入了对 XMLHttpRequests 和 Web 字体的跨域支持（但最初的实现并不完整，这在后续版本中得到完善）；Firefox 7 引入了对 WebGL 贴图的跨域支持；Firefox 9 引入了对 drawImage 的跨域支持。

**参见**[**Edit**](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS$edit#%E5%8F%82%E8%A7%81)

* [Code Samples Showing XMLHttpRequest and Cross-Origin Resource Sharing](http://arunranga.com/examples/access-control/)
* [Cross-Origin Resource Sharing From a Server-Side Perspective (PHP, etc.)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Server-Side_Access_Control)
* [Cross-Origin Resource Sharing specification](http://www.w3.org/TR/cors/)
* [XMLHttpRequest](https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest)
* [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
* [Using CORS with All (Modern) Browsers](http://www.kendoui.com/blogs/teamblog/posts/11-10-03/using_cors_with_all_modern_browsers.aspx)
* [Using CORS - HTML5 Rocks](http://www.html5rocks.com/en/tutorials/cors/)