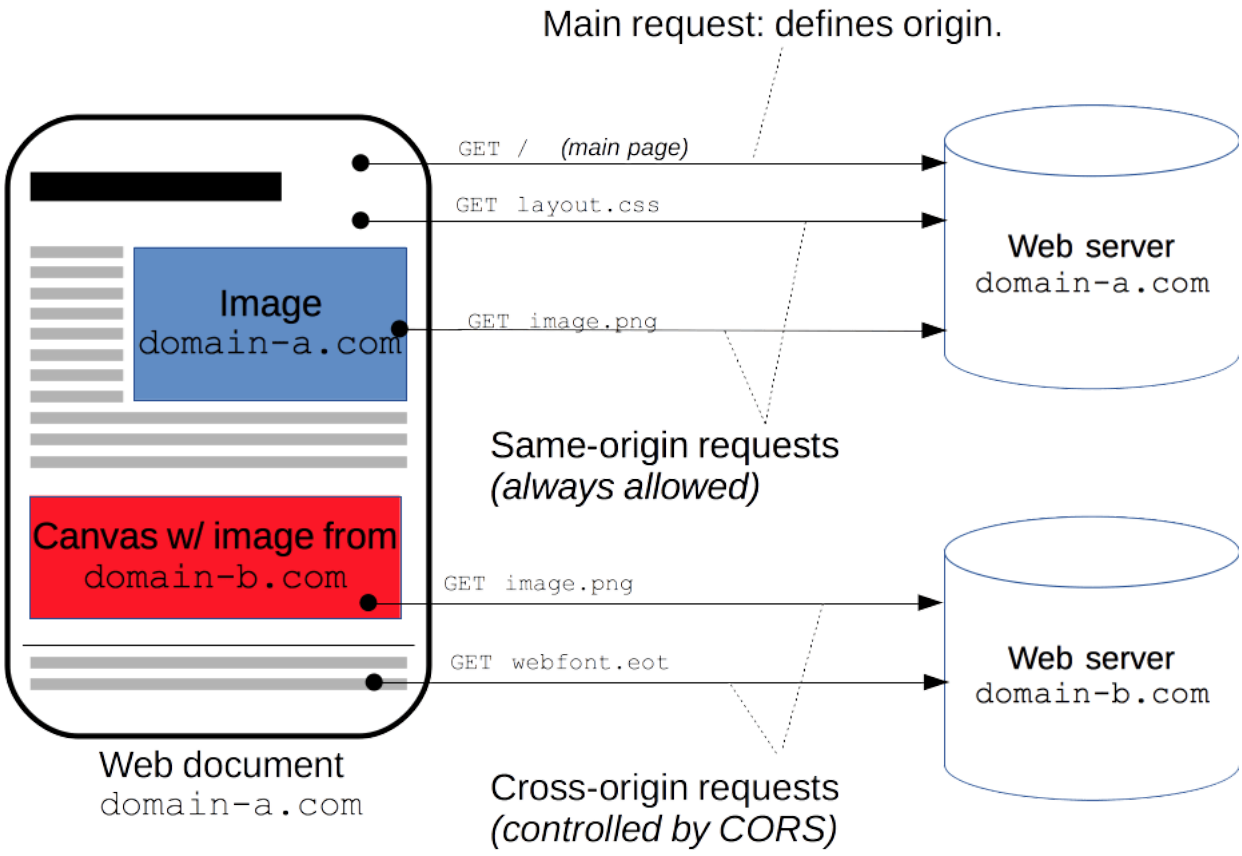
此页面由社区从英文翻译而来。了解更多并加入 MDN Web Docs 社区。

跨源资源共享（CORS）

跨源资源共享 (CORS)（或通俗地译为跨域资源共享）是一种基于 HTTP 头的机制，该机制通过允许服务器标示除了它自己以外的其它 origin（域，协议和端口），使得浏览器允许这些 origin 访问加载自己的资源。跨源资源共享还通过一种机制来检查服务器是否会允许要发送的真实请求，该机制通过浏览器发起一个到服务器托管的跨源资源的"预检"请求。在预检中，浏览器发送的头中标示有HTTP方法和真实请求中会用到的头。

跨源HTTP请求的一个例子：运行在 `https://domain-a.com` 的 JavaScript 代码使用 `XMLHttpRequest` 来发起一个到 `https://domain-b.com/data.json` 的请求。

出于安全性，浏览器限制脚本内发起的跨源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，除非响应报文包含了正确 CORS 响应头。



跨源域资源共享（CORS）机制允许 Web 应用服务器进行跨源访问控制，从而使跨源数据传输得以安全进行。现代浏览器支持在 API 容器中（例如 `XMLHttpRequest` 或 Fetch）使用 CORS，以降低跨源 HTTP 请求所带来的风险。

谁应该读这篇文章？

说实话，每个人。

更具体地来讲，这篇文章适用于 **网站管理员**、**后端和前端开发者**。现代浏览器处理跨源资源共享的客户端部分，包括HTTP头和相关策略的执行。但是这一新标准意味着服务器需要处理新的请求头和响应头。

什么情况下需要 CORS ？

这份 cross-origin sharing standard 允许在下列场景中使用跨站点 HTTP 请求：

- 前文提到的由 `XMLHttpRequest` 或 Fetch APIs 发起的跨源 HTTP 请求。
- Web 字体 (CSS 中通过 `@font-face` 使用跨源字体资源)，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 。
- WebGL 贴图
- 使用 `drawImage` 将 Images/video 画面绘制到 canvas。
- 来自图像的 CSS 图形

本文概述了跨源资源共享机制及其所涉及的 HTTP 头。

功能概述

跨源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通过浏览器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` 以外的 HTTP 请求，或者搭配某些 MIME类型 的 `POST` 请求），浏览器必须首先使用 `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源请求。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 和 HTTP认证 相关数据）。

CORS 请求失败会产生错误，但是为了安全，在 JavaScript 代码层面是无法获知到底具体是哪里出了问题。你只能查看浏览器的控制台以得知具体是哪里出现了错误。

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

这里，我们使用三个场景来解释跨源资源共享机制的工作原理。这些例子都使用 `XMLHttpRequest` 对象。

简单请求

某些请求不会触发 CORS 预检请求。本文称这样的请求为“简单请求”，请注意，该术语并不属于 Fetch （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`
  - `HEAD`
  - `POST`
- 除了被用户代理自动设置的首部字段（例如 `Connection` ， `User-Agent` ）和在 Fetch 规范中定义为 禁用首部名称 的其他首部，允许人为设置的字段为 Fetch 规范定义的 对 CORS 安全的首部字段集合 。该集合为：
  - `Accept`
  - `Accept-Language`
  - `Content-Language`
  - `Content-Type` （需要注意额外的限制）
- `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 `XMLHttpRequest` 对象均没有注册任何事件监听器； `XMLHttpRequest` 对象可以使用 `XMLHttpRequest.upload` 属性访问。
- 请求中没有使用 `ReadableStream` 对象。

> **备注：** WebKit Nightly 和 Safari Technology Preview 为 `Accept` , `Accept-Language` , 和 `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
> - Allow commas in Accept, Accept-Language, and Content-Language request headers for simple CORS
> - Switch to a blacklist model for restricted Accept headers in simple CORS requests

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 `https://foo.example` 的网页应用想要访问 `https://bar.other` 的资源。 `foo.example` 的网页中可能包含类似于下面的 JavaScript 代码：

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';

xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```

客户端和服务器之间使用 CORS 首部字段来处理权限：

以下是浏览器发送给服务器的请求报文：

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
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
Connection: keep-alive
Origin: https://foo.example
```

请求首部字段 `Origin` 表明该请求来源于 `http://foo.example` 。

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[XML Data]
```

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。

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

使用 `Origin` 和 `Access-Control-Allow-Origin` 就能完成最简单的访问控制。如果服务端仅允许来自 `https://foo.example` 的访问，该首部字段的内容如下：

```
Access-Control-Allow-Origin: https://foo.example
```

> **备注：** 当响应的是附带身份凭证的请求时，服务端 **必须** 明确 `Access-Control-Allow-Origin` 的值，而不能使用通配符“ `*` ”。

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` 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。

如下是一个需要执行预检请求的 HTTP 请求：

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
```

```
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```

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”。

备注： 如下所述，实际的 `POST` 请求不会携带 `Access-Control-Request-*` 首部，它们仅用于 `OPTIONS` 请求。

下面是服务端和客户端完整的信息交互。首次交互是 *预检请求/响应*：

```
OPTIONS /doc HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
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
Connection: keep-alive
Origin: https://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
```

```
HTTP/1.1 204 No Content
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
```

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 1~10 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 13~22 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 16~19 行：

```
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
```

服务器的响应携带了 `Access-Control-Allow-Origin: https://foo.example` ，从而限制请求的源域。同时，携带的 `Access-Control-Allow-Methods` 表明服务器允许客户端使用 `POST` 和 `GET` 方法发起请求（与 `Allow` 响应首部类似，但其具有严格的访问控制）。

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。

预检请求完成之后，发送实际请求：

```
POST /doc HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
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
Connection: keep-alive
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Referer: https://foo.example/examples/preflightInvocation.html
Content-Length: 55
Origin: https://foo.example
Pragma: no-cache
Cache-Control: no-cache

<person><name>Arun</name></person>

HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:40 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://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 XML payload]
```

预检请求与重定向

并不是所有浏览器都支持预检请求的重定向。如果一个预检请求发生了重定向，一部分浏览器将报告错误：

> 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 redirects.

CORS 最初要求浏览器具有该行为，不过在后续的 修订　中废弃了这一要求。但并非所有浏览器都实现了这一变更，而仍然表现出最初要求的行为。

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：

1. 发出一个简单请求（使用 Response.url 或 XHR.responseURL）以判断真正的预检请求会返回什么地址。
2. 发出另一个请求（真正的请求），使用在上一步通过 Response.url 或 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

> **备注：** 当发出跨源请求时，第三方 cookie 策略仍将适用。无论如何改变本章节中描述的服务器和客户端的设置，该策略都会强制执行。

`XMLHttpRequest` 或 Fetch 与 CORS 的一个有趣的特性是，可以基于 HTTP cookies 和 HTTP 认证信息发送身份凭证。一般而言，对于跨源 `XMLHttpRequest` 或 Fetch 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` 的某个特殊标志位。

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

```
const invocation = new XMLHttpRequest();
const url = 'https://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` 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/credentialed-content/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
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
Connection: keep-alive
Referer: https://foo.example/examples/credential.html
Origin: https://foo.example
Cookie: pageAccess=2

HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:34:52 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://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 10 行指定了 Cookie 的相关信息，但是，如果 `https://bar.other` 的响应中缺失 `Access-Control-Allow-Credentials : true` （第 17 行），则响应内容不会返回给请求的发起者。

预检请求和凭据

CORS 预检请求不能包含凭据。预检请求的 *响应* 必须指定 `Access-Control-Allow-Credentials: true` 来表明可以携带凭据进行实际的请求。

> **备注：** 一些企业认证服务要求在预检请求时发送 TLS 客户端证书，这违反了 Fetch 的规范。
>
> Firefox 87 允许通过在设置中设定 `network.cors_preflight.allow_client_cert` 为 `true` （bug 1511151 ）来允许这种不规范的行为。基于 chromium 的浏览器目前总是在 CORS 预检请求中发送 TLS 客户端证书（Chrome bug 775438 ）。

附带身份凭证的请求与通配符

在响应附带身份凭证的请求时：

- 服务器不能将 `Access-Control-Allow-Origin` 的值设为通配符“`*`”，而应将其设置为特定的域，如： `Access-Control-Allow-Origin: https://example.com` 。
- 服务器不能将 `Access-Control-Allow-Headers` 的值设为通配符“`*`”，而应将其设置为首部名称的列表，如： `Access-Control-Allow-Headers: X-PINGOTHER, Content-Type`
- 服务器不能将 `Access-Control-Allow-Methods` 的值设为通配符“`*`”，而应将其设置为特定请求方法名称的列表，如： `Access-Control-Allow-Methods: POST, GET`

对于附带身份凭证的请求（通常是 `Cookie` ），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 `https://example.com` ，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。

第三方 cookies

注意在 CORS 响应中设置的 cookies 适用一般性第三方 cookie 策略。在上面的例子中，页面是在 `foo.example` 加载，但是第 20 行的 cookie 是被 `bar.other` 发送的，如果用户设置其浏览器拒绝所有第三方 cookies，那么将不会被保存。

请求中的 cookie（第 10 行）也可能在正常的第三方 cookie 策略下被阻止。因此，强制执行的 cookie 策略可能会使本节描述的内容无效（阻止你发出任何携带凭据的请求）。

Cookie 策略受 SameSite 属性控制。

HTTP 响应首部字段

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` 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 `https://mozilla.org` 的请求：

```
Access-Control-Allow-Origin: https://mozilla.org
Vary: Origin
```

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：在跨源访问时， `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` 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` 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 预检请求的结果在多少秒内有效。

Access-Control-Allow-Credentials

`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。

Access-Control-Allow-Methods

`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>]*
```

有关 preflight request 的示例已在上方给出。

Access-Control-Allow-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

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

Origin

`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。

> **备注：** 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，在所有访问控制请求（Access control request）中，`Origin` 首部字段 **总是** 被发送。

Access-Control-Request-Method

`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。

Access-Control-Request-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。

规范

Specification	Status	Comment
Fetch CORS	Living Standard	New definition; supplants W3C CORS specification.

浏览器兼容性

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS
`Access-Control-Allow-Origin`	Chrome4	Edge12	Firefox3.5	Internet Explorer 10	Opera12	Safari4	WebView2 Android	ChromeYes Android	Firefox 4 for Android	Opera 12 Android	Safari3 on iOS

Full support

参见

- CORS errors
- Enable CORS: I want to add CORS support to my server
- `XMLHttpRequest`
- Fetch API
- Will it CORS? - an interactive CORS explainer & generator
- How to run Chrome browser without CORS
- Using CORS with All (Modern) Browsers
- Stack Overflow answer with “how to” info for dealing with common problems :
  - How to avoid the CORS preflight
  - How to use a CORS proxy to get around *“No Access-Control-Allow-Origin header”*
  - How to fix *“Access-Control-Allow-Origin header must not be the wildcard”*

Last modified: 2022年4月7日, by MDN contributors