

HTTP access control (CORS)



所謂跨站HTTP請求(Cross-site HTTP request)是指發出請求所在網域不同於請求所指向之網域的[HTTP](#)請求，例如網域A(<http://domaina.example>)的網頁載入一個元素向網域B(<http://domainb.foo>)請求影像資源(<http://domainb.foo/image.jpg>)。這種作法在現今各網頁相當常見，網頁常常會載入其他網站資源，像是CSS樣式表、影像、程式碼等等資源。

基於安全性考量，程式碼所發出的跨站HTTP請求受到相當限制，好比說用[XMLHttpRequest](#)物件所發出的請求受限於[同源政策\(same-origin policy\)](#)，只能發送HTTP請求到和其所來自的相同的網域，不過開發者也希望發展出安全的跨站請求方法來開發更好、更安全、搭配多種資源的網頁應用。

[W3C](#)下轄的[網頁應用工作小組\(Web Applications Working Group\)](#)建議採用[跨來源資源共享\(Cross-Origin Resource Sharing, CORS\)](#)來達成安全的跨站資料傳輸，CORS提供網頁伺服器支援跨站存取控制方法，另外，這份規範是用於API容器，例如以XMLHttpRequest作為調解機制，好跳脫同網域限制；也因為客戶端瀏覽器會多出新的跨來源共享元件，包括標頭和政策施定，所以說，伺服器也必須相應地處理這些新增機制，也就是處理新標頭以及用新標頭發送資源。本文內容主要和網站管理員、伺服器端開發者和網站開發者有關，然後關於伺服器部分請參閱[同跨來源共享:從伺服器觀點出發\(以PHP為範例\)](#)補充文章。

[跨來源資源共享標準](#)可用來開啟以下跨站HTTP請求:

- 用XMLHttpRequest API進行跨站請求，如前所述。
- 網頁字體(跨網域CSS的@font-face的字體用途)，[所以伺服器可以部屬TrueType字體並只允許授權網站進行跨站載入使用](#)。
- WebGL紋理
- 用drawImage畫到畫布上的影像

本文主要在總覽跨來源資源共享和FireFox 3.5所實作的HTTP標頭。

總覽

跨來源資源共享標準加入了新HTTP標頭，使得伺服器能夠描述那些來源可以用瀏覽器讀取資料。另外，針對會造成副作用的HTTP請求方法(特別是GET以外的HTTP方法或搭配某些MIME種類的POST方法)，標準規範了瀏覽器必須要發送「先導」請求，以HTTP的OPTIONS方法從伺服器取得支援方法，然後當伺服器許可後再用真實的HTTP請求方法送出真實的請求。伺服器也可以通知客戶端是否要連安全性資料(包括cookie和HTTP驗證資料)一併隨請求送出。

後面我們將討論相關情境和相關HTTP標頭。

存取控制情境範例


這邊我們會展示三種情境說明跨來源資源共享如何運作，所有的跨站請求範例都使用[XMLHttpRequest](#)物件。

本處Javascript程式碼範本要運行在支援跨站XMLHttpRequest 請求的瀏覽器上，如果想看Javascript程式(以及處理跨站請求的伺服器端程式運作實體)的[實際運行請到這裡](#)，如果想了解[伺服器端的跨來源資源共享探討\(包含PHP程式碼範例\)](#)請到[這裡](#)。

Simple requests

所謂簡單的跨站請求意味著:

- 只用GET, HEAD, POST方法
- 標頭必須為下列類型：Accept, Accept-Language, Content-Language(不區分大小寫),或者是Content-Type必須為下列application/x-www-form-urlencoded, multipart/form-data, 或text/plain其中一種。
- 沒有自訂義的標頭，例如X-Modified等等。

 **Note:** 雖然這些都是網頁目前已經可以送出的跨站請求，但除非伺服器回傳適當標頭，否則不會有資料回傳，因此不允許跨站請求的網站無須擔心會受到新的HTTP存取控制影響。

假設<http://foo.example>網域上的網頁內容想要呼叫<http://bar.other>網域內的內容，以下程式碼可能會在foo.example上執行:

```
1  var invocation = new XMLHttpRequest();
2  var url = 'http://bar.other/resources/public-data/';
3
4  function callOtherDomain() {
5      if(invocation) {
6          invocation.open('GET', url, true);
7          invocation.onreadystatechange = handler;
8          invocation.send();
9      }
10 }
```

我們來看看瀏覽器傳送了甚麼到伺服器，伺服器又回傳了甚麼:

```
1  GET /resources/public-data/ 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  Referer: http://foo.example/examples/access-control/simpleXSInvocation.html
10 Origin: http://foo.example
11
12
13 HTTP/1.1 200 OK
14 Date: Mon, 01 Dec 2008 00:23:53 GMT
15 Server: Apache/2.0.61
16 Access-Control-Allow-Origin: *
17 Keep-Alive: timeout=2, max=100
18 Connection: Keep-Alive
19 Transfer-Encoding: chunked
20 Content-Type: application/xml
21
22 [XML Data]
```

第1 ~ 10行是Firefox 3.5送出的標頭，請注意第10行的origin標頭，它標示出請求來自<http://foo.example>網域上的內容。

第13 ~ 22行是http://bar.other網域伺服器回傳的HTTP回應。第16行伺服器回傳了一個叫Access-Control-Allow-Origin的標頭，從origin標頭與Access-Control-Allow-Origin標頭中可以看到存取控制協定最簡單的用途。這個例子中，伺服器回傳" Access-Control-Allow-Origin: *"代表允許任一網域跨站存取資源，倘若http://bar.other資源擁有者只准許http://foo.example存取資源，那麼回傳會改成:

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

如此一來，origin標頭不為http://foo.example網域將無法跨站存取資源。Access-Control-Allow-Origin標頭需要包含請求origin標頭的值。

先導請求

不同於前述簡單請求例子，"先導"請求會先以HTTP的OPTIONS方法送出請求到另一個網域，確認後續真實請求是否可安全送出，由於跨站請求可能會攜帶使用者資料，所以要先進行先導請求。先導請求會被送出當符合以下其中一種條件時:

- 使用非[GET 簡單方法](#)(GET, HEAD, POST)。
- 使用以下類型除外的標頭：Accept, Accept-Language, Content-Language,Content-Type非為下列application/x-www-form-urlencoded, multipart/form-data, 或text/plain其中一種(上述類型為[GET 簡單標頭](#)，且皆不區分大小寫)。例如POST請求連帶的資料是application/xml或text/xml的XML類型資料，那麼先導請求就會先送出。
- 請求中有自訂義標頭，例如自訂義一個X-PINGOTHER標頭。

Gecko 2.0 備註
(Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1)

自Gecko 2.0(Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1)起，text/plain, application/x-www-form-urlencoded與multipart/form-data編碼資料都不需要先導請求，在之前只有text/plain不需要先導。

下面是一段會引起先導請求的範例:

```
1  var invocation = new XMLHttpRequest();
2  var url = 'http://bar.other/resources/post-here/';
3  var body = '<?xml version="1.0"?><person><name>Arun</name></person>';
4
5  function callOtherDomain(){
6      if(invocation)
7      {
8          invocation.open('POST', url, true);
9          invocation.setRequestHeader('X-PINGOTHER', 'pingpong');
10         invocation.setRequestHeader('Content-Type', 'application/xml');
11         invocation.onreadystatechange = handler;
12         invocation.send(body);
13     }
14
15     .....
```

上方第3行產生一段XML類型資料，然後第8行POST請求送出，第9行自訂義一個不屬於HTTP/1.1協定的X-PINGOTHER: pingpong標頭，因為POST送出Content-type為application/xml的資料而且又含有自訂義標頭，所以需要做先導請求。

我們來看看客戶端和伺服器端間完整的來回資訊:

```
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
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
20 Access-Control-Max-Age: 1728000
21 Vary: Accept-Encoding
22 Content-Encoding: gzip
23 Content-Length: 0
24 Keep-Alive: timeout=2, max=100
25 Connection: Keep-Alive
26 Content-Type: text/plain
27
28 POST /resources/post-here/ HTTP/1.1
29 Host: bar.other
30 User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre
31 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
32 Accept-Language: en-us,en;q=0.5
33 Accept-Encoding: gzip,deflate
34 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
35 Connection: keep-alive
36 X-PINGOTHER: pingpong
37 Content-Type: text/xml; charset=UTF-8
38 Referer: http://foo.example/examples/preflightInvocation.html
39 Content-Length: 55
40 Origin: http://foo.example
41 Pragma: no-cache
42 Cache-Control: no-cache
43
44 <?xml version="1.0"?><person><name>Arun</name></person>
45
46
47 HTTP/1.1 200 OK
48 Date: Mon, 01 Dec 2008 01:15:40 GMT
49 Server: Apache/2.0.61 (Unix)
50 Access-Control-Allow-Origin: http://foo.example
51 Vary: Accept-Encoding
```



```
52 Content-Encoding: gzip
53 Content-Length: 235
54 Keep-Alive: timeout=2, max=99
55 Connection: Keep-Alive
56 Content-Type: text/plain
57
58 [Some GZIP'd payload]
```

第1~12行屬於OPTIONS方法的先導請求，Firefox 3.1根據前面的Javascript程式碼決定送出先導請求，好讓伺服器回應是否允許後續送出真實請求。OPTIONS是一個HTTP/1.1方法，這個方法用來確認來自伺服器進一步的資訊，重複執行不會造成任何影響，不會造成資源更動。除了OPTIONS方法，有另外兩個請求標頭送出如下：

```
1 Access-Control-Request-Method: POST
2 Access-Control-Request-Headers: X-PINGOTHER
```

Access-Control-Request-Method告訴伺服器送出真實請求的方法會是POST，Access-Control-Request-Headers告訴伺服器真實請求會攜帶一個自訂義的X-PINGOTHER標頭。在這些資訊下，接著伺服器將會確定是否接受請求。

第15~27行屬於伺服器的回應，它說明了伺服器接受POST請求方法和X-PINGOTHER標頭。另外讓我們特別來看看18~21行：

```
1 Access-Control-Allow-Origin: http://foo.example
2 Access-Control-Allow-Methods: POST, GET, OPTIONS
3 Access-Control-Allow-Headers: X-PINGOTHER
4 Access-Control-Max-Age: 1728000
```

Access-Control-Allow-Methods顯示了POST, GET和OPTIONS皆為可接受方法，請注意這個標頭和[HTTP/1.1 Allow: 回應標頭](#)十分相似，但它只在存取控制範圍下才有意義。Access-Control-Allow-Headers顯示了X-PINGOTHER為可接受標頭，和Access-Control-Allow-Methods一樣，Access-Control-Allow-Headers也是用逗號分隔可接受標頭。Access-Control-Max-Age顯示了本次先導請求回應所以快取的秒數，其中1728000秒等於20天。

送出附帶安全性資料的請求

送出附帶安全性資料(HTTP Cookies和驗證資訊)的請求是XMLHttpRequest和存取控制最有趣的功能。一般來說，跨站XMLHttpRequest請求，瀏覽器不會送出這些安全性資料，不過在XMLHttpRequest物件上設定一個特定的旗標後就可以送出附帶安全性資料的請求。

下面來自http://foo.example的內容送出一個GET請求到http://bar.other索取資源，而http://bar.other會設定cookie：

```
1 var invocation = new XMLHttpRequest();
2 var url = 'http://bar.other/resources/credentialed-content/';
3
4 function callOtherDomain(){
5     if(invocation) {
6         invocation.open('GET', url, true);
7         invocation.withCredentials = true;
8         invocation.onreadystatechange = handler;
9         invocation.send();
10    }
```

預設不會附帶安全性資料，為了附帶，第7行上XMLHttpRequest有一個withCredentials旗標必須要設定為true，而且由於這只是一個簡單的GET請求，所以沒有先導請求。如果沒有Access-Control-Allow-Credentials: true的標頭回傳，瀏覽器會拒絕任何回應，而我們也無法取得遭拒絕之回應。

下面是客戶端和伺服器端交換的資料：

```
1 GET /resources/access-control-with-credentials/ 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 Referer: http://foo.example/examples/credential.html
10 Origin: http://foo.example
11 Cookie: pageAccess=2
12
13
14 HTTP/1.1 200 OK
15 Date: Mon, 01 Dec 2008 01:34:52 GMT
16 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
17 X-Powered-By: PHP/5.2.6
18 Access-Control-Allow-Origin: http://foo.example
19 Access-Control-Allow-Credentials: true
20 Cache-Control: no-cache
21 Pragma: no-cache
22 Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT
23 Vary: Accept-Encoding
24 Content-Encoding: gzip
25 Content-Length: 106
26 Keep-Alive: timeout=2, max=100
27 Connection: Keep-Alive
28 Content-Type: text/plain
29
30
31 [text/plain payload]
```

雖然第12行顯示了cookie連帶被傳到http://bar.other上的資源，然而假如bar.other沒有回傳Access-Control-Allow-Credentials: true標頭(第20行)，那麼bar.other的回應將被忽略且無法取得。**重要事項：**當回應附帶安全性資料的請求時，伺服器必須要明白指出網域，不可以只有“*”萬用字元，像如果上面Access-Control-Allow-Origin標投如果為“*”而非http://foo.example，便不允許。第23行顯示了cookie又被設置了。

[↗ 範例運作請到這裡](#)，下面我們將看看HTTP各標頭。

HTTP回應標頭

這邊我們將列出跨來源資源共享規範所定義伺服器端會回傳的存取控制回應標頭。

Access-Control-Allow-Origin

回應可能會附帶以下標頭格式：

```
1 | Access-Control-Allow-Origin: <origin> | *
```

origin參數代表允許存取資源的URI。如果請求中沒有安全性資料，伺服器可以設定Access-Control-Allow-Origin為"*"，允許任何網域存取資源。

如果要允許http://mozilla.com存取伺服器，我們可以這麼做:

```
1 | Access-Control-Allow-Origin: http://mozilla.com
```

Access-Control-Expose-Headers

(Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1)

這個標頭指示瀏覽器允許存取的標頭白名單:

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

上面允許了向瀏覽器顯示X-My-Custom-Header與X-Another-Custom-Header標頭。

Access-Control-Max-Age

這個標頭指示了先導請求的結果快取多長時間(關於先導請求請參照上方說明)。

```
1 | Access-Control-Max-Age: <delta-seconds>
```

delta-seconds代表結果可以被快取的秒數。

Access-Control-Allow-Credentials

這個標頭指示了當請求的credentials設定true時，是否要回應請求。當用在先導請求的回應中，那就是指示後續的真實請求可否附帶安全性資料。由於簡單的GET請求沒有先導請求，所以如果附帶安全性資料的GET請求沒有受到這個標頭的回應，那回應將被忽略而且無法取得。

```
1 | Access-Control-Allow-Credentials: true | false
```

[送出附帶安全性資料的請求請參考上方說明](#)。

Access-Control-Allow-Methods

指示存取資源所允許的方法，用來回應先導請求。

```
1 | Access-Control-Allow-Methods: <method>[, <method>]*
```

[關於先導請求請參考上方說明](#)。

Access-Control-Allow-Headers

指示那些HTTP header可以出現在真實請求，用於回應先導請求。

```
1 | Access-Control-Allow-Headers: <field-name>[, <field-name>]*
```

HTTP 請求標頭

這裡列出當進行跨來源資源共享請求客戶端需要送出的標頭。請注意使用跨站XMLHttpRequest不需要在程式碼中設定這些標頭，這些標頭會由瀏覽器來設定。

Origin

指示跨站存取請求或先導請求的來源。

```
1 | Origin: <origin>
```

其值是一個URI告訴伺服器請求來源，不含有路徑資訊，僅有伺服器名稱。

 **Note:** origin可為空字串，相當有用，例如來源為data URL。

這個標頭在任何存取控制請求都**一定需要**送出。

Access-Control-Request-Method

用在先導請求上，告訴伺服器端後續真實請求所用的HTTP方法。

```
1 | Access-Control-Request-Method: <method>
```

[相關說明請參照本文上方](#)。

Access-Control-Request-Headers

用在先導請求上，告訴伺服器端後續真實請求所帶的HTTP標頭

```
1 | Access-Control-Request-Headers: <field-name>[, <field-name>]*
```

[範例說明請參考上方](#)。

瀏覽器相容性

	Desktop	Mobile			
Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	4	3.5	8 (via XDomainRequest) 10	12	4

Note

IE8和IE9支援CORS透過XDomainRequest物件，IE10開始則完全正常支援。Firefox 3.5引進支援跨站XMLHttpRequests與Web Fonts，較舊版本上某些請求會受到限制。Firefox 7 引進支援跨站WebGL的跨站HTTP請求

延伸閱讀

- [↗ Code Samples Showing XMLHttpRequest and Cross-Origin Resource Sharing](#)
- [Cross-Origing Resource Sharing From a Server-Side Perspective \(PHP, etc.\)](#)
- [↗ Cross-Origin Resource Sharing specification](#)
- [XMLHttpRequest](#)
- [↗ Further Discussion of the Origin Header](#)
- [↗ Using CORS with All \(Modern\) Browsers](#)
- [↗ Using CORS - HTML5 Rocks](#)