# 跨域与安全

## 一、同源策略：

跨域只存在于浏览器端，不存在于安卓/ios/Node.js/python/ java等其它环境

跨域请求能发出去，服务端能收到请求并正常返回结果，只是结果被浏览器拦截了

之所以会跨域，是因为受到了同源策略的限制，同源策略要求源相同才能正常进行通信，即**协议、域名、端口号**都完全一致。

同源政策的**目的**，是为了**保证用户信息的安全，防止恶意的网站窃取数据。**

同源策略具体限制些什么呢？

1. 不能向工作在不同源的的服务请求数据（client to server）
2. 无法获取不同源的document/cookie等BOM和DOM，可以说任何有关另外一个源的信息都无法得到 （client to client）

大致分为以下三类：

（1） Cookie、LocalStorage 和 IndexDB 无法读取。

（2） DOM 无法获得。

（3） AJAX 请求不能发送。

## 二、跨域解决方案

script标签能够加载非同源的资源，不受同源策略的影响。

添加这个http头可以限制别人把你的网站套成它的iframe：

X-Frame-Options: SAMEORIGIN

Cookie 是服务器写入浏览器的一小段信息，只有同源的网页才能共享。但是，两个网页一级域名相同，只是二级域名不同，浏览器允许通过设置document.domain共享 Cookie。但是，这种方法只适用于 Cookie 和 iframe 窗口，LocalStorage 和 IndexDB 无法通过这种方法，规避同源政策，而要使用下文介绍的PostMessage API。

### client to server：

**跨域方法：**

1. **CORS**只要浏览器检测到响应头带上了CORS，并且允许的源包括了本网站，那么就不会拦截请求响应。

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。

本文详细介绍CORS的内部机制。

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

浏览器将CORS**请求分成两类**：**简单请求（simple request）**和**非简单请求（not-so-simple request）**。

只要同时满足以下两大条件，就属于简单请求。

1) 请求方法是以下三种方法之一：

HEAD

GET

POST

2）HTTP的头信息不超出以下几种字段：

Accept

Accept-Language

Content-Language

Last-Event-ID

Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

凡是不同时满足上面两个条件，就属于非简单请求。

**简单请求：**

对于简单请求，浏览器直接发出CORS请求。具体来说，就是在头信息之中，增加一个Origin字段。Origin字段用来说明，本次请求来自哪个源（协议 + 域名 + 端口）。服务器根据这个值，决定是否同意这次请求。

如果Origin指定的源，不在许可范围内，服务器会返回一个正常的HTTP回应。如果Origin指定的域名在许可范围内，服务器返回的响应，会多出几个头信息字段。

**Access-Control-Allow-Origin：**该字段是**必须的**。它的值要么是请求时Origin字段的值，要么是一个\*，表示接受任意域名的请求。

**Access-Control- Expose -Headers:** 该字段可选。CORS请求时，XMLHttpRequest对象的getResponseHeader()方法只能拿到6个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想拿到其他字段，就必须在Access-Control-Expose-Headers里面指定。

**Access-Control-Allow-Credentials:** 该字段可选。它的值是一个布尔值，表示是否允许发送Cookie。默认情况下，Cookie不包括在CORS请求之中。设为true，即表示服务器明确许可，Cookie可以包含在请求中，一起发给服务器。这个值也只能设为true，如果服务器不要浏览器发送Cookie，删除该字段即可。

**非简单请求：**

非简单请求是那种对服务器有特殊要求的请求，比如请求方法是PUT或DELETE，或者Content-Type字段的类型是application/json。非简单请求的CORS请求，会在正式通信之前，增加一次HTTP查询请求，称为"预检"请求（preflight）。浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。

1. 预检请求

实例：

OPTIONS /cors HTTP/1.1

Origin: [http://api.bob.com](http://api.bob.com/)

Access-Control-Request-Method: PUT

Access-Control-Request-Headers: X-Custom-Header

Host: api.alice.com

Accept-Language: en-US

Connection: keep-alive

User-Agent: Mozilla/5.0...

"预检"请求用的请求方法是**OPTIONS**，表示这个请求是用来询问的。头信息里面，关键字段是**Origin**，表示请求来自哪个源。

除了Origin字段，"预检"请求的头信息包括两个特殊字段。

（1）**Access-Control-Request-Method**

该字段是必须的，用来列出浏览器的CORS请求会用到哪些HTTP方法，上例是PUT。

（2）**Access-Control-Request-Headers**

该字段是一个逗号分隔的字符串，指定浏览器CORS请求会额外发送的头信息字段，上例是X-Custom-Header。

预检请求返回字段：

（1）**Access-Control-Allow-Methods**

该字段必需，它的值是逗号分隔的一个字符串，表明服务器支持的所有跨域请求的方法。注意，返回的是所有支持的方法，而不单是浏览器请求的那个方法。这是为了避免多次"预检"请求。

（2）**Access-Control-Allow-Headers**

如果浏览器请求包括Access-Control-Request-Headers字段，则Access-Control-Allow-Headers字段是必需的。它也是一个逗号分隔的字符串，表明服务器支持的所有头信息字段，不限于浏览器在"预检"中请求的字段。

（3）**Access-Control-Allow-Credentials**

该字段与简单请求时的含义相同。

（4）**Access-Control-Max-Age**

该字段可选，用来指定本次预检请求的有效期，单位为秒。上面结果中，有效期是20天（1728000秒），即允许缓存该条回应1728000秒（即20天），在此期间，不用发出另一条预检请求。

1. 正常请求

一旦服务器通过了"预检"请求，以后每次浏览器正常的CORS请求，就都跟简单请求一样，会有一个Origin头信息字段。服务器的回应，也都会有一个Access-Control-Allow-Origin头信息字段。

请求中的Origin字段是浏览器自动添加的。响应中的Access-Control-Allow-Origin字段是每次回应都必定包含的。

1. **JSONP**是利用了script标签能够跨域

JSONP是服务器与客户端跨源通信的常用方法。最大特点就是**简单适用，老式浏览器全部支持，服务器改造非常小**。

它的基本思想是，网页通过添加**一个<script>元素**，向服务器请求JSON数据，这种做法不受同源政策限制；服务器收到请求后，将数据放在一个指定名字的回调函数里传回来。

由于script/iframe/img等标签的请求默认是能带上cookie（cookie里面带上了登陆验证的票token），用这些标签发请求是能够绕过同源策略的，因此就可以利用这些标签做**跨站请求伪造（CSRF）。而动态ajax请求默认是不带cookie的，如果你要带cookie，可以设置ajax的一个属性withCredentials.**

1. **WebSocket**是一种通信协议，使用ws://（非加密）和wss://（加密）作为协议前缀。该协议不实行同源政策，只要服务器支持，就可以通过它进行跨源通信。

CORS与JSONP的使用目的相同，但是比JSONP更强大。

JSONP只支持GET请求，CORS支持所有类型的HTTP请求。JSONP的优势在于支持老式浏览器，以及可以向不支持CORS的网站请求数据。

### Client to client

**如何和一个frame(LocalStorage)通信**，包括iframe或者使用window.open打开的页面

方法一：片段识别符（fragment identifier）

片段标识符（fragment identifier）指的是，URL的#号后面的部分，比如http://example.com/x.html#fragment的#fragment。如果只是改变片段标识符，页面不会重新刷新。

父窗口可以把信息，写入子窗口的片段标识符。子窗口通过监听hashchange事件得到通知。

方法二：window.name

浏览器窗口有window.name属性。这个属性的最大特点是，无论是否同源，只要在同一个窗口里，前一个网页设置了这个属性，后一个网页可以读取它。

父窗口先打开一个子窗口，载入一个不同源的网页，该网页将信息写入window.name属性。接着，子窗口跳回一个与主窗口同域的网址。然后，主窗口就可以读取子窗口的window.name了。

这种方法的优点是，window.name容量很大，可以放置非常长的字符串；缺点是必须监听子窗口window.name属性的变化，影响网页性能。

方法三：跨文档通信API（Cross-document messaging）

上面两种方法都属于破解，HTML5为了解决这个问题，引入了一个全新的API：跨文档通信 API（Cross-document messaging）。

这个API为window对象新增了一个window.postMessage方法，允许跨窗口通信，不论这两个窗口是否同源。

1. iframe通知父页面是通过window.parent.postMessage
2. 父页面使用 window.contentWindow.postMessage 通知 iframe

补充：

如果需要和同源的不同标签页进行通信可以使用localStorage，即一个页面设置localStorage，其它页面就会触发storage事件。

websocket是不受同源策略限制的，没有跨域的问题。CSS的字体文件是会有跨域问题，指定CORS就能加载其它源的字体文件（通常是放在cdn上的）。而canvas动态加载的外部image，也是需要指定CORS头才能进行图片处理，否则只能画不能读取。

最后，跨域分为两种，一种是跨域请求，另一种访问跨域的页面，跨域请求可以通过CORS/JSONP等方法进行访问，跨域的页面主要通过postMesssage的方式。由于跨域请求不但能发出去还能带上cookie，所以要规避跨站请求伪造攻击的风险，特别是涉及到钱的那种请求。

参考文献：

<https://juejin.im/post/5a6320d56fb9a01cb64ee191>

<http://www.ruanyifeng.com/blog/2016/04/cors.html>

<http://www.ruanyifeng.com/blog/2016/04/same-origin-policy.html>