自：<https://segmentfault.com/a/1190000012158485>

2017-11-25 发布

# [跨域问题汇总](https://segmentfault.com/a/1190000012158485)

* [javascript](https://segmentfault.com/t/javascript/blogs)

* [[https://avatar-static.segmentfault.com/102/500/1025005915-54cb538538eea_small](https://segmentfault.com/t/html5/blogs) html5](https://segmentfault.com/t/html5/blogs)

 3.9k 次阅读  ·  读完需要 24 分钟

11

跨域问题的场景和解决方案多种多样，只要是做前端开发，总会遇到。而且面试时也是必问的问题。所以自己学习总结记录一下。

因为浏览器的同源策略，前端开发会遇到各种跨域问题。本篇文章总结了遇到跨域问题的不同的场景以及对应的解决方案。

## 前言

在总结各种跨域问题之前，我们先来了解一下浏览器的同源策略。  
协议、域名、端口都相同才叫同源。具体的这里就不赘述了。

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。  
设想这样一种情况：A网站是一家银行，用户登录以后，又去浏览其他网站。如果其他网站可以读取A网站的 Cookie，会发生什么？  
很显然，如果 Cookie 包含隐私（比如存款总额），这些信息就会泄漏。更可怕的是，Cookie 往往用来保存用户的登录状态，如果用户没有退出登录，其他网站就可以冒充用户，为所欲为。因为浏览器同时还规定，提交表单不受同源政策的限制。  
由此可见，"同源政策"是必需的，否则 Cookie 可以共享，互联网就毫无安全可言了。

**受到同源限制：**

1）无法读取不同源的 Cookie、LocalStorage 和 IndexDB 。  
2）无法获得不同源的DOM 。  
3）不能向不同源的服务器发送ajax请求。

**不受同源限制：**  
在浏览器中，<script>、<img>、<iframe>、<link>等标签都可以跨域加载资源，而不受同源策略的限制。

**浏览器对跨域访问的判定：**  
CORS机制把跨域请求分为两类：简单请求和非简单请求。

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。浏览器对这两种请求的处理，是不一样的。

**简单请求：**浏览器会带上Origin的请求头发送到服务器，服务器根据Origin判断是否许可。如果许可就会带上CORS相关想要头，如果不在许可范围内就不会带上CORS相关的响应头。浏览器再根据响应头中是否有相关的CORS响应头，来判断拦截响应body和抛出错误。  
**非简单请求：**非简单请求会在发真正的请求之前发送一个OPTIONS的带着Origin、Access-Control-Request-Method、Access-Control-Request-Headers等CORS相关的请求头的预检请求到服务器，服务器确认可以这样请求，就会返回带着Access-Control-Allow-Origin、Access-Control-Allow-Methods、Access-Control-Allow-Headers等CORS相关的响应头的响应，浏览器检查到相关的CORS响应头，说明通过预检可以继续发送真正的请求；服务器确认不可以，则不会返回这些相关响应头，浏览器没检查到CORS的响应头就会抛出错误。

## 一、代理跨域

**场景1：**你的项目myweb，myweb的前端有一个接口是去访问一个非myweb的服务器。非myweb服务器是第三方服务器，你不能去对第三方服务器做改动。  
**场景2：**你的项目是个微服务架构的。那你的前端页面可能就需要去很多个服务器上访问数据。

**原理解析：**  
跨域请求报错归根结底是浏览器禁止使用XHR对象向不同源的服务器地址发起HTTP请求。如果是服务器跨域向多个不同的服务器发送请求就不会有跨域问题存在。因此，我们可以让浏览器只向一个服务器方式请求，让这个服务器代替浏览器去不同的服务器上请求资源再返回给浏览器。这个服务器就是代理服务器了。

下面推荐一个常用代理服务器nginx。

**什么是nginx?**  
Nginx (engine x) 是一款轻量级的Web 服务器 、反向代理服务器及电子邮件（IMAP/POP3）代理服务器。

把ui所在的服务器和跨域服务器都用nginx代理转发，浏览器访问nginx，nginx到ui服务获取ui，再把ui下载到浏览器，浏览器发起ui中的URL，该URL为Nginx封装后的跨域服务器的URL或ui服务器的URL，该URL到达Nginx之后，会被转发到跨域服务器或ui服务器，请求处理完毕后，会通过Nginx中转返回给浏览器。暴露出来的或者浏览器所发起的url都是nginx的url，nginx去跨域服务器和ui服务器获取响应，返给浏览器，这样就没有跨域问题了。

## 二、CORS

**场景：**  
前后端分离的开发模式下，在本地进行接口联调时：也许在你的项目里，你想尝试前后端分离的开发模式。你在本地开发时，mock了一些假数据来帮助自己本地开发。而有一天，你希望在本地和后端同学进行联调。此时，后端rd的接口地址和你发生了跨域问题。这阻止了你们的联调，你只能继续使用你mock的假数据。

**解决方案：**  
CORS需要浏览器和服务器同时支持。如何支持？请看**浏览器对跨域访问的判定**小节。  
整个CORS通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS通信与同源的AJAX通信没有差别，代码完全一样。浏览器一旦发现AJAX请求跨源，就会自动添加一些附加的头信息，有时还会多出一次附加的请求，但用户不会有感觉。  
因此，实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信。  
服务器要给接口的响应头设置：Access-Control-Allow-Origin：＊

## 三、jsonp

**场景：跨域发送get请求**

jsonp解决跨域问题的本质： <script> 标签可以请求不同域名下的资源，即 <script> 请求不受浏览器同源策略影响。  
首先给body动态添加一个 <script>

**var** script = document.createElement('script');

script.setAttribute("type","text/javascript");

script.src = 'http://example.com/ip?callback=foo';

document.body.appendChild(script);

**function** **foo**(data) {

console.log('Your public IP address is: ' + data.ip);

};

上面的script会向 http://example.com/ 服务器发送请求，这个请求的url后面带了个callback参数，是用来告诉服务器回调方法的方法名的。因为服务器收到请求后，会把相应数据写进foo的参数位置，也就是说服务器会返回的脚本如下

foo({

"ip": "8.8.8.8"

});

这样浏览器通过<script>下载的资源就是上面的脚本了，<script>下载完成就会立即执行，也就是说http://example.com/ip?callback=foo这个请求返回后就会立即执行上面的脚本代码，而这个脚本代码就是调用回调方法和拿到json数据了。

## 四、document.domain跨域

**场景1：**你的http://www.damonare.cn/a.html页面里使用<iframe>调用另一个http://damonare.cn/b.html页面。这时候你想在a页面里获取b页面里的dom，然后进行操作。然后你会发现你不能获得b的dom。document.getElementById("myIFrame").contentWindow.document或window.parent.document.body因为两个窗口不同源而报错。

**解决方案：**这时候你只需要在a页面里和b页面里把document.domain设置成相同的值就可以在两个页面里操作Dom了。

**场景2：**你在http://www.damonare.cn/a.html页面里写入了document.cookie = "test1=hello";你在http://damonare.cn/b.html页面是拿不到这个cookie的。

**解决方案：**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。另外，服务器也可以在设置Cookie的时候，指定Cookie的所属域名为一级域名。这样的话，二级域名和三级域名不用做任何设置，都可以读取这个Cookie。

**注意：**  
document.domain限制：虽然可读写，但只能设置成自身或者是高一级的父域且主域必须相同。所以只能解决一级域名相同二级域名不同的跨域问题。  
document.domain只适用于 Cookie 和 iframe 窗口，LocalStorage 和 IndexDB 无法通过这种方法跨域。

## 五、window.name跨域

**场景1：**现在浏览器的一个标签页里打开http://www.damonare.cn/a.html页面，你通过location.href=http://baidu.com/b.html，在同一个浏览器标签页里打开了不同域名下的页面。这时候这两个页面你可以使用window.name来传递参数。因为window.name指的是浏览器窗口的名字，只要浏览器窗口相同，那么无论在哪个网页里访问值都是一样的。

**场景2：**你的http://www.damonare.cn/a.html页面里使用<iframe>调用另一个http://baidu.com/b.html页面。这时候你想在a页面里获取b页面里的dom，然后进行操作。然后你会发现你不能获得b的dom。同样会因为不同源而报错，和上面提到的不同之处就是两个页面的一级域名也不相同。这时候document.domain就解决不了了。

**解决方案：**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。。比如你在b页面里设定window.name="hello"，你再返回到a页面，在a页面里访问window.name，可以得到hello。

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。

## 六、postMessage方法跨域

**场景1:**在a页面里打开了另一个不同源的页面b，你想要让a和b两个页面互相通信。比如，a要访问b的LocalStorage。

**场景2:**你的a页面里的iframe的src是不同源的b页面，你想要让a和b两个页面互相通信。比如，a要访问b的LocalStorage。

**解决方案：**HTML5y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。a就可以把它的LocalStorage，发送给b，b也可以把自己的LocalStorage发给a。

window.postMessage(message, targetOrigin, [transfer])，有三个参数：

**message**是向目标窗口发送的数据；  
**targetOrigin**属性来指定哪些窗口能接收到消息事件，其值可以是字符串"\*"（表示无限制）或者一个URI（或者说是发送消息的目标域名）；  
**transfer**可选参数，是一串和message 同时传递的 Transferable 对象. 这些对象的所有权将被转移给消息的接收方，而发送一方将不再保有所有权。

另外消息的**接收方必须有监听事件**，否则发送消息时就会报错。The target origin provided ('http://localhost:8080') does not match the recipient window's origin ('http://localhost:63343').

window.addEventListener("message",onmessage);onmessage接收到的message事件包含三个属性：

**data：**从其他 window 中传递过来的数据。  
**origin：**调用 postMessage 时消息发送方窗口的 origin 。请注意，这个origin不能保证是该窗口的当前或未来origin，因为postMessage被调用后可能被导航到不同的位置。  
**source：**对发送消息的窗口对象的引用; 您可以使用此来在具有不同origin的两个窗口之间建立双向通信。

例子：我在a页面执行

**var** popup = window.open('http://localhost:3000', 'title');

popup.postMessage('Hello World!', 'http://localhost:3000');

同时在http://localhost:3000的页面里监听message事件：

window.onload=**function** () {

window.addEventListener("message",onmessage);

}

**function** **onmessage**(event) {

**if**(event.origin=="http://localhost:63343"){//http://localhost:63343是发送方a的域名

console.log(event.data);//'Hello World!'

}

console.log(event.data);//'Hello World!'

}

注意: 在 Gecko 6.0 (Firefox 6.0 / Thunderbird 6.0 / SeaMonkey 2.3)之前， 参数 message 必须是一个字符串。 从 Gecko 6.0 (Firefox 6.0 / Thunderbird 6.0 / SeaMonkey 2.3)开始，参数 message被使用结构化克隆算法进行序列化。这意味着您可以将各种各样的数据对象安全地传递到目标窗口，而不必自己序列化它们。

## 七、location.hash跨域

location.hash就是指URL的#号后面的部分。  
**场景：**  
父窗口和iframe的子窗口之间通讯或者是window.open打开的子窗口之间的通讯。  
**解决方案：**  
父窗口改变子窗口的url的#号后面的部分，后者把要传递的参数写在#后面，子窗口监听window.onhashchange事件，得到通知，读取window.location.hash解析出有用的数据。同样子窗口也可以向父窗口传递数据。

**参考资料：**

[浏览器同源策略与ajax跨域方法汇总](http://www.jianshu.com/p/438183ddcea8)  
[前端跨域整理](http://blog.damonare.cn/2016/12/01/%E5%89%8D%E7%AB%AF%E8%B7%A8%E5%9F%9F%E6%95%B4%E7%90%86/)  
[浏览器同源政策及其规避方法](http://www.ruanyifeng.com/blog/2016/04/same-origin-policy.html)  
[跨域资源共享 CORS 详解](http://www.ruanyifeng.com/blog/2016/04/cors.html)