# 浏览器同源政策及其规避方法

一、概述

1.1 含义

1995年，同源政策由 Netscape 公司引入浏览器。目前，所有浏览器都实行这个政策。

最初，它的含义是指，A网页设置的 Cookie，B网页不能打开，除非这两个网页"同源"。所谓"同源"指的是"三个相同"。

* 协议相同
* 域名相同
* 端口相同

举例来说，http://www.example.com/dir/page.html这个网址，协议是http://，域名是www.example.com，端口是80（默认端口可以省略）。它的同源情况如下。

* http://www.example.com/dir2/other.html：同源
* http://example.com/dir/other.html：不同源（域名不同）
* http://v2.www.example.com/dir/other.html：不同源（域名不同）
* http://www.example.com:81/dir/other.html：不同源（端口不同）

1.2 目的

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。

1.3 限制范围

随着互联网的发展，"同源政策"越来越严格。目前，如果非同源，共有三种行为受到限制。

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

（2） DOM 无法获得。

（3） AJAX 请求不能发送。

虽然这些限制是必要的，但是有时很不方便，合理的用途也受到影响。下面，我将详细介绍，如何规避上面三种限制。

二、Cookie

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。

举例来说，A网页是http://w1.example.com/a.html，B网页是http://w2.example.com/b.html，那么只要设置相同的document.domain，两个网页就可以共享Cookie。

document.domain = 'example.com';

现在，A网页通过脚本设置一个 Cookie。

document.cookie = "test1=hello";

B网页就可以读到这个 Cookie。

var allCookie = document.cookie;

注意，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。

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，比如.example.com。

Set-Cookie: key=value; domain=.example.com; path=/

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。

三、iframe

如果两个网页不同源，就无法拿到对方的DOM。典型的例子是iframe窗口和window.open方法打开的窗口，它们与父窗口无法通信。

比如，父窗口运行下面的命令，如果iframe窗口不是同源，就会报错。

document.getElementById("myIFrame").contentWindow.document

// Uncaught DOMException: Blocked a frame from accessing a cross-origin frame.

上面命令中，父窗口想获取子窗口的DOM，因为跨源导致报错。

反之亦然，子窗口获取主窗口的DOM也会报错。

window.parent.document.body

// 报错

如果两个窗口一级域名相同，只是二级域名不同，那么设置上一节介绍的document.domain属性，就可以规避同源政策，拿到DOM。

对于完全不同源的网站，目前有三种方法，可以解决跨域窗口的通信问题。

* 片段识别符（fragment identifier）
* window.name
* 跨文档通信API（Cross-document messaging）

3.1 片段识别符

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。

var src = originURL + '#' + data;

document.getElementById('myIFrame').src = src;

子窗口通过监听hashchange事件得到通知。

window.onhashchange = checkMessage;

function checkMessage() {

var message = window.location.hash;

// ...

}

同样的，子窗口也可以改变父窗口的片段标识符。

parent.location.href= target + "#" + hash;

3.2 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。

window.name = data;

接着，子窗口跳回一个与主窗口同域的网址。

location = '<http://parent.url.com/xxx.html>';

然后，主窗口就可以读取子窗口的window.name了。

var data = document.getElementById('myFrame').contentWindow.name;

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。

3.3 window.postMessage

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。

举例来说，父窗口http://aaa.com向子窗口http://bbb.com发消息，调用postMessage方法就可以了。

var popup = window.open('[http://bbb.com](http://bbb.com/)', 'title');

popup.postMessage('Hello World!', '[http://bbb.com](http://bbb.com/)');

postMessage方法的第一个参数是具体的信息内容，第二个参数是接收消息的窗口的源（origin），即"协议 + 域名 + 端口"。也可以设为\*，表示不限制域名，向所有窗口发送。

子窗口向父窗口发送消息的写法类似。

window.opener.postMessage('Nice to see you', '[http://aaa.com](http://aaa.com/)');

父窗口和子窗口都可以通过message事件，监听对方的消息。

window.addEventListener('message', function(e) {

console.log(e.data);

},false);

message事件的事件对象event，提供以下三个属性。

* event.source：发送消息的窗口
* event.origin: 消息发向的网址
* event.data: 消息内容

下面的例子是，子窗口通过event.source属性引用父窗口，然后发送消息。

window.addEventListener('message', receiveMessage);

function receiveMessage(event) {

event.source.postMessage('Nice to see you!', '\*');

}

event.origin属性可以过滤不是发给本窗口的消息。

window.addEventListener('message', receiveMessage);

function receiveMessage(event) {

if (event.origin !== '[http://aaa.com](http://aaa.com/)') return;

if (event.data === 'Hello World') {

event.source.postMessage('Hello', event.origin);

} else {

console.log(event.data);

}

}

3.4 LocalStorage

通过window.postMessage，读写其他窗口的 LocalStorage 也成为了可能。

下面是一个例子，主窗口写入iframe子窗口的localStorage。

window.onmessage = function(e) {

if (e.origin !== '[http://bbb.com](http://bbb.com/)') {

return;

}

var payload = JSON.parse(e.data);

localStorage.setItem(payload.key, JSON.stringify(payload.data));

};

上面代码中，子窗口将父窗口发来的消息，写入自己的LocalStorage。

父窗口发送消息的代码如下。

var win = document.getElementsByTagName('iframe')[0].contentWindow;

var obj = { name: 'Jack' };

win.postMessage(JSON.stringify({key: 'storage', data: obj}), '[http://bbb.com](http://bbb.com/)');

加强版的子窗口接收消息的代码如下。

window.onmessage = function(e) {

if (e.origin !== '[http://bbb.com](http://bbb.com/)') return;

var payload = JSON.parse(e.data);

switch (payload.method) {

case 'set':

localStorage.setItem(payload.key, JSON.stringify(payload.data));

break;

case 'get':

var parent = window.parent;

var data = localStorage.getItem(payload.key);

parent.postMessage(data, '[http://aaa.com](http://aaa.com/)');

break;

case 'remove':

localStorage.removeItem(payload.key);

break;

}

};

加强版的父窗口发送消息代码如下。

var win = document.getElementsByTagName('iframe')[0].contentWindow;

var obj = { name: 'Jack' };

// 存入对象

win.postMessage(JSON.stringify({key: 'storage', method: 'set', data: obj}), '[http://bbb.com](http://bbb.com/)');

// 读取对象

win.postMessage(JSON.stringify({key: 'storage', method: "get"}), "\*");

window.onmessage = function(e) {

if (e.origin != '[http://aaa.com](http://aaa.com/)') return;

// "Jack"

console.log(JSON.parse(e.data).name);

};

四、AJAX

同源政策规定，AJAX请求只能发给同源的网址，否则就报错。

除了架设服务器代理（浏览器请求同源服务器，再由后者请求外部服务），有三种方法规避这个限制。

* JSONP
* WebSocket
* CORS

4.1 JSONP

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。

首先，网页动态插入<script>元素，由它向跨源网址发出请求。

function addScriptTag(src) {

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

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

script.src = src;

document.body.appendChild(script);

}

window.onload = function () {

addScriptTag('<http://example.com/ip>?callback=foo');

}

function foo(data) {

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

};

上面代码通过动态添加<script>元素，向服务器example.com发出请求。注意，该请求的查询字符串有一个callback参数，用来指定回调函数的名字，这对于JSONP是必需的。

服务器收到这个请求以后，会将数据放在回调函数的参数位置返回。

foo({

"ip": "8.8.8.8"

});

由于<script>元素请求的脚本，直接作为代码运行。这时，只要浏览器定义了foo函数，该函数就会立即调用。作为参数的JSON数据被视为JavaScript对象，而不是字符串，因此避免了使用JSON.parse的步骤。

4.2 WebSocket

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。

下面是一个例子，浏览器发出的WebSocket请求的头信息（摘自[维基百科](https://en.wikipedia.org/wiki/WebSocket" \t "_blank)）。

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: [http://example.com](http://example.com/)

上面代码中，有一个字段是Origin，表示该请求的请求源（origin），即发自哪个域名。

正是因为有了Origin这个字段，所以WebSocket才没有实行同源政策。因为服务器可以根据这个字段，判断是否许可本次通信。如果该域名在白名单内，服务器就会做出如下回应。

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

4.3 CORS

CORS是跨源资源分享（Cross-Origin Resource Sharing）的缩写。它是W3C标准，是跨源AJAX请求的根本解决方法。相比JSONP只能发GET请求，CORS允许任何类型的请求。

下一篇文章，我会详细介绍，如何通过[CORS](http://www.ruanyifeng.com/blog/2016/04/cors.html)完成跨源AJAX请求。

（完）