# 我的研究

1. window.location.href=<https://...>跳转（指定https）。此为get方式。
2. 用jquery的$.getJSON进行ajax跨域请求（如$.getJSON(“https://...”)）。该方式只能用get方式。
3. 通过使用iframe中的form提交来实现ajax的post请求。此为post方式。
4. 使用html中的表单form提交（action=”https://...”）。此为get或者post方式。
5. 使用CORS跨域资源共享实现跨域

我在“考试系统维护”项目中，使用的是第4种（表单提交），来实现https的post请求。详见xampp/htdocs/ExamSystem/Tpl/default/Index/index.html中。

在后台（xampp/htdocs/ExamSystem/Lib/Action/IndexAction.class.php index）中，如果用户名和密码验证正确，则使用http跳转。参见：

//强制使用http传输。

//如果直接”用$this->redirect("Main/main");“跳转的话，则会用https传输。

$url = $this->\_getUrl();

header('Location: '.$url);

1. GetJSON

使用jquery的getJSON方法可以实现跨域请求。

示例1

客户端

1. $.getJSON(
2. "http://192.168.0.66/services/WebService1.asmx/ws?callback=?",
3. { name: "ff", time: "2pm" },
4. **function**(data) { alert(decodeURI(data.msg)) }
5. );

服务器端

1. [WebMethod]
2. **public** **void** ws(string name,string time) {
3. HttpRequest Request = HttpContext.Current.Request;
4. string callback = Request["callback"];
5. HttpResponse Response = HttpContext.Current.Response;
6. Response.Write(callback + "({msg:'this is"+name+"jsonp'})");
7. Response.End();
8. }

客户端获得的json数据data为json对象

原理

Jquery.GetJson首先在客户端注册一个callback函数，然后把callback的名字(如:jsonp1236827957501)传给服务器。注意：服务端得到callback的数值后，要用jsonp1236827957501(......)把将要输出的json内容包括起来，用Response.Write（jsonp1236827957501(......)）写入时，服务器生成 json 数据才能被客户端正确接收。执行完毕后就会移除callback函数。

缺点

第一也是最重要的：JSONP不提供错误处理。如果动态插入的代码正常运行，你可以得到返回，但是如果失败了，那么什么都不会发生。你无法获得一个404的错误，也不能取消这个请求。

另外一个重要的缺点是如果使用了不信任的服务会造成很大的安全隐患。

## 示例2（动态加载）

JSONP example: Tic-tac-toe

Let’s put the JSONP to work with an example—a game of tic-tac-toe, where the players

are the client (the browser) and the server. Both will generate random numbers between

1 and 9, and we’ll use JSONP to get the value of the server’s turn (see Figure 8-2).

You can play the game live at http://jspatterns.com/book/8/ttt.html.

Figure 8-2. Tic-tac-toe JSONP challenge

There are two buttons: to start a new game and to get the server’s turn (the client’s turn

will be automatic after a timeout):

<button id="new">New game</button>

<button id="server">Server play</button>

Remote Scripting | 193

The board will contain nine table cells with corresponding id attributes. For example:

<td id="cell-1">&nbsp;</td>

<td id="cell-2">&nbsp;</td>

<td id="cell-3">&nbsp;</td>

...

The whole game is implemented in a ttt global object:

var ttt = {

// cells played so far

played: [],

// shorthand

get: function (id) {

return document.getElementById(id);

},

// handle clicks

setup: function () {

this.get('new').onclick = this.newGame;

this.get('server').onclick = this.remoteRequest;

},

// clean the board

newGame: function () {

var tds = document.getElementsByTagName("td"),

max = tds.length,

i;

for (i = 0; i < max; i += 1) {

tds[i].innerHTML = "&nbsp;";

}

ttt.played = [];

},

// make a request

remoteRequest: function () {

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

script.src = "server.php?callback=ttt.serverPlay&played=" + ttt.played.join(',');

document.body.appendChild(script);

},

// callback, server's turn to play

serverPlay: function (data) {

if (data.error) {

alert(data.error);

return;

}

data = parseInt(data, 10);

this.played.push(data);

this.get('cell-' + data).innerHTML = '<span class="server">X<\/span>';

setTimeout(function () {

ttt.clientPlay();

}, 300); // as if thinking hard

194 | Chapter 8: DOM and Browser Patterns

},

// client's turn to play

clientPlay: function () {

var data = 5;

if (this.played.length === 9) {

alert("Game over");

return;

}

// keep coming up with random numbers 1-9

// until one not taken cell is found

while (this.get('cell-' + data).innerHTML !== "&nbsp;") {

data = Math.ceil(Math.random() \* 9);

}

this.get('cell-' + data).innerHTML = 'O';

this.played.push(data);

}

};

The object ttt maintains a list of cells played so far in ttt.played and sends them to

the server, so the server can return a new number excluding the ones that were already

played. If an error occurs, the server will respond with output like:

ttt.serverPlay({"error": "Error description here"});

As you can see, the callback in JSONP has to be a publicly and globally available func-

tion, not necessarily a global, but it can be a method of a global object. If there are no

errors, the server will respond with a method call like:

ttt.serverPlay(3);

Here 3 means that cell number 3 is the server’s random choice. In this case the data is

so simple that we don’t even need the JSON format; a single value is all it takes

# CORS跨域

下面文章出自<http://blog.csdn.net/hfahe/article/details/7730944>

前言：像CORS对于现代前端这么重要的技术在国内基本上居然很少有人使用和提及，在百度或者Google上搜索CORS，搜到的中文文章基本都是另外一种卫星定位技术CORS的介绍，让我等前端同学情何以堪（对比起来，用Google搜到的国外文章，基本都是跨域资源共享的介绍，说明了前端技术在国内外环境和发展的巨大差距）。

        我之前《[用HTML5实现人脸识别](http://blog.csdn.net/hfahe/article/details/7485452)》这篇文章中提到了“Face.com实现了[**CORS**](http://dvcs.w3.org/hg/cors/raw-file/tip/Overview.html)（跨域资源共享）。CORS系统基本上可以让服务器暴露给其它域上文件的Ajax调用。这是一个伟大的功能，我希望更多的服务能够使用它。”在这篇文章介绍的实现方式里，我们可以自由的使用自己本域的JS代码通过Ajax来调用Face.com的API，这是一种很美妙的方式，而在以前我们很难做到这一点。

        由此我将引入和介绍CORS，希望对大家有所帮助。

**定义**

        CORS其实出现时间不短了，它在[维基百科](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing)上的定义是：**跨域资源共享（CORS** ）是一种网络浏览器的技术规范，它为Web服务器定义了一种方式，允许网页从不同的域访问其资源。而这种访问是被[同源策略](http://en.wikipedia.org/wiki/Same_origin_policy)所禁止的。CORS系统定义了一种浏览器和服务器交互的方式来确定是否允许跨域请求。 它是一个妥协，有更大的灵活性，但比起简单地允许所有这些的要求来说更加安全。

        而W3C的[官方文档](http://www.w3.org/TR/cors/)目前还是工作草案，但是正在朝着W3C推荐的方向前进。

        简言之，CORS就是为了让AJAX可以实现可控的跨域访问而生的。

**以往的解决方案**

        以前要实现跨域访问，可以通过JSONP、Flash或者服务器中转的方式来实现，但是现在我们有了CORS。

        CORS与JSONP相比，无疑更为先进、方便和可靠。

        1、 JSONP只能实现GET请求，而CORS支持所有类型的HTTP请求。

        2、 使用CORS，开发者可以使用普通的XMLHttpRequest发起请求和获得数据，比起JSONP有更好的错误处理。

        3、 JSONP主要被老的浏览器支持，它们往往不支持CORS，而绝大多数现代浏览器都已经支持了CORS（这部分会在后文浏览器支持部分介绍）。

**详细内容**

        要使用CORS，我们需要了解前端和服务器端的使用方法。

        1、  前端

        以前我们使用Ajax，代码类似于如下的方式：

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. var xhr = new XMLHttpRequest();
2. xhr.open("GET", "/hfahe", true);
3. xhr.send();

        这里的“/hfahe”是本域的相对路径。

        如果我们要使用CORS，相关Ajax代码可能如下所示：

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. var xhr = new XMLHttpRequest();
2. xhr.open("GET", "http://blog.csdn.net/hfahe", true);
3. xhr.send();

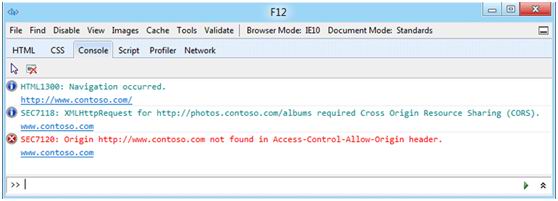
        请注意，代码与之前的区别就在于相对路径换成了其他域的绝对路径，也就是你要跨域访问的接口地址。

        我们还必须提供浏览器回退功能检测和支持，避免浏览器不支持的情况。

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. function createCORSRequest(method, url) {
2. var xhr = new XMLHttpRequest();
3. if ("withCredentials" in xhr) {
4. // 此时即支持CORS的情况
5. // 检查XMLHttpRequest对象是否有“withCredentials”属性
6. // “withCredentials”仅存在于XMLHTTPRequest2对象里
7. xhr.open(method, url, true);
9. } else if (typeof!= "undefined") {
11. // 否则检查是否支持XDomainRequest，IE8和IE9支持
12. // XDomainRequest仅存在于IE中，是IE用于支持CORS请求的方式
13. xhr = new XDomainRequest();
14. xhr.open(method, url);
16. } else {
18. // 否则，浏览器不支持CORS
19. xhr = null;
21. }
22. return xhr;
23. }
25. var xhr = createCORSRequest('GET', url);
26. if (!xhr) {
27. throw new Error('CORS not supported');
28. }

        现在如果直接使用上面的脚本进行请求，会看到浏览器里控制台的报错如下：



        错误显示的很明显，这是因为我们还未设置Access-Control-Allow-Origin头。

        2、  服务器

        服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

        HTTP 头的设置方法有很多，<http://enable-cors.org/>这篇文章里对各种服务器和语言的设置都有详细的介绍，下面我们主要介绍Apache和PHP里的设置方法。

        Apache：Apache需要使用mod\_headers模块来激活HTTP头的设置，它默认是激活的。你只需要在Apache配置文件的<Directory>, <Location>, <Files>或<VirtualHost>的配置里加入以下内容即可：

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. Header set Access-Control-Allow-Origin \*

        PHP：只需要使用如下的代码设置即可。

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. **<?php**
2. header("Access-Control-Allow-Origin:\*");

        以上的配置的含义是允许任何域发起的请求都可以获取当前服务器的数据。当然，这样有很大的危险性，恶意站点可能通过XSS攻击我们的服务器。所以我们应该尽量有针对性的对限制安全的来源，例如下面的设置使得只有http://blog.csdn.net这个域才能跨域访问服务器的API。

**[html]** [view plaincopy](http://blog.csdn.net/hfahe/article/details/7730944)

1. Access-Control-Allow-Origin: http://blog.csdn.net

**浏览器支持情况**



        上图为各浏览器对于CORS的支持情况（绿色为支持，数据来源：<http://caniuse.com/cors>），看起来相当乐观。主流浏览器都已基本提供对跨域资源共享的支持，所以，CORS才会在国外使用的如此普遍。

        上文曾经提到，IE8和IE9在某种程度上可以通过XDomainRequest来提供同样功能的支持。

**使用案例**

        目前国外支持CORS的平台有很多，例如：



[Google APIClient Library for JS](http://code.google.com/p/google-api-javascript-client/wiki/CORS)

[Google CloudStorage](https://developers.google.com/storage/docs/cross-origin)



[Face.com API](http://developers.face.com/docs/api/)

**未来**

        从所有的浏览器都支持来看，CORS将成为未来跨域访问的标准解决方案。无论是自己服务器间的跨域访问，还是开放平台为第三方提供API，都将采用这种统一的解决方案，因为它简单、高效，受到所有主流浏览器的支持。它非常重要，也会让我们的网络变得更加开放。

**参考文章**

[IE10中的CORS forXHR](http://www.iefans.net/ie10-cors-for-xhr/)

[USING CORS](http://www.html5rocks.com/en/tutorials/cors/)

        原创文章，转载请注明：来自蒋宇捷的博客（http://blog.csdn.net/hfahe）

# 网上资料

[iframe和异步的跨域请求](http://www.limboy.com/2008/12/31/iframe-cross-site-request/)

[**JavaScript跨域总结与解决办法**](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html)

[什么是跨域](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m0)

## [1、document.domain+iframe的设置](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m1)

## [2、动态创建script](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m2)

## [3、利用iframe和location.hash](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m3)

## [4、window.name实现的跨域数据传输](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m4)

## [5、使用HTML5 postMessage](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m5)

## [6、利用flash](http://www.cnblogs.com/rainman/archive/2011/02/20/1959325.html#m6)

本文来自网络（http://f2e.me/200904/cross-scripting/，该网址已不能访问），仅作个人读书笔记之用，并稍作修改和补充。

**什么是跨域**

JavaScript出于安全方面的考虑，不允许跨域调用其他页面的对象。但在安全限制的同时也给注入iframe或是ajax应用上带来了不少麻烦。这里把涉及到跨域的一些问题简单地整理一下：

首先什么是跨域，简单地理解就是因为JavaScript同源策略的限制，a.com 域名下的js无法操作b.com或是c.a.com域名下的对象。更详细的说明可以看下表：

|  |  |  |
| --- | --- | --- |
| **URL** | **说明** | **是否允许通信** |
| http://www.a.com/a.js http://www.a.com/b.js | 同一域名下 | 允许 |
| http://www.a.com/lab/a.js http://www.a.com/script/b.js | 同一域名下不同文件夹 | 允许 |
| http://www.a.com:8000/a.js http://www.a.com/b.js | 同一域名，不同端口 | 不允许 |
| http://www.a.com/a.js https://www.a.com/b.js | 同一域名，不同协议 | 不允许 |
| http://www.a.com/a.js http://70.32.92.74/b.js | 域名和域名对应ip | 不允许 |
| http://www.a.com/a.js http://script.a.com/b.js | 主域相同，子域不同 | 不允许 |
| http://www.a.com/a.js http://a.com/b.js | 同一域名，不同二级域名（同上） | 不允许（cookie这种情况下也不允许访问） |
| http://www.cnblogs.com/a.js http://www.a.com/b.js | 不同域名 | 不允许 |

**特别注意两点：**

第一，如果是协议和端口造成的跨域问题“前台”是无能为力的，

第二：在跨域问题上，域仅仅是通过“URL的首部”来识别而不会去尝试判断相同的ip地址对应着两个域或两个域是否在同一个ip上。  
“URL的首部”指window.location.protocol +window.location.host，也可以理解为“Domains, protocols and ports must match”。

接下来简单地总结一下在“前台”一般处理跨域的办法，后台proxy这种方案牵涉到后台配置，这里就不阐述了，有兴趣的可以看看yahoo的这篇文章：《[JavaScript: Use a Web Proxy for Cross-Domain XMLHttpRequest Calls](http://developer.yahoo.com/javascript/howto-proxy.html)》

**1、document.domain+iframe的设置**

**对于主域相同而子域不同的例子，可以通过设置document.domain的办法来解决**。具体的做法是可以在http://www.a.com/a.html和http://script.a.com/b.html两个文件中分别加上document.domain = ‘a.com’；然后通过a.html文件中创建一个iframe，去控制iframe的contentDocument，这样两个js文件之间就可以“交互”了。当然这种办法只能解决主域相同而二级域名不同的情况，如果你异想天开的把script.a.com的domian设为alibaba.com那显然是会报错地！代码如下：

www.a.com上的a.html

document.domain = 'a.com';

var ifr = document.createElement('iframe');

ifr.src = 'http://script.a.com/b.html';

ifr.style.display = 'none';

document.body.appendChild(ifr);

ifr.onload = function(){

var doc = ifr.contentDocument || ifr.contentWindow.document;

// 在这里操纵b.html

alert(doc.getElementsByTagName("h1")[0].childNodes[0].nodeValue);

};

script.a.com上的b.html

document.domain = 'a.com';

这种方式适用于{www.kuqin.com, kuqin.com, script.kuqin.com, css.kuqin.com}中的任何页面相互通信。

备注：某一页面的domain默认等于window.location.hostname。主域名是不带www的域名，例如a.com，主域名前面带前缀的通常都为二级域名或多级域名，例如www.a.com其实是二级域名。 domain只能设置为主域名，不可以在b.a.com中将domain设置为c.a.com。

**问题：**

1、安全性，当一个站点（b.a.com）被攻击后，另一个站点（c.a.com）会引起安全漏洞。

2、如果一个页面中引入多个iframe，要想能够操作所有iframe，必须都得设置相同domain。

**2、动态创建script**

**虽然浏览器默认禁止了跨域访问，但并不禁止在页面中引用其他域的JS文件，并可以自由执行引入的JS文件中的function（包括操作cookie、Dom等等）**。根据这一点，可以方便地通过创建script节点的方法来实现完全跨域的通信。具体的做法可以参考YUI的[Get Utility](http://developer.yahoo.com/yui/get/)

这里判断script节点加载完毕还是蛮有意思的：ie只能通过script的readystatechange属性，其它浏览器是script的load事件。以下是部分判断script加载完毕的方法。

js.onload = js.onreadystatechange = function() {

if (!this.readyState || this.readyState === 'loaded' || this.readyState === 'complete') {

// callback在此处执行

js.onload = js.onreadystatechange = null;

}

};

**3、利用iframe和location.hash**

这个办法比较绕，但是可以解决完全跨域情况下的脚步置换问题。原理是利用location.hash来进行传值。在url： http://a.com#helloword中的‘#helloworld’就是location.hash，改变hash并不会导致页面刷新，所以可以利用hash值来进行数据传递，当然数据容量是有限的。假设域名a.com下的文件cs1.html要和cnblogs.com域名下的cs2.html传递信息，cs1.html首先创建自动创建一个隐藏的iframe，iframe的src指向cnblogs.com域名下的cs2.html页面，这时的hash值可以做参数传递用。cs2.html响应请求后再将通过修改cs1.html的hash值来传递数据（由于两个页面不在同一个域下IE、Chrome不允许修改parent.location.hash的值，所以要借助于a.com域名下的一个代理iframe；Firefox可以修改）。同时在cs1.html上加一个定时器，隔一段时间来判断location.hash的值有没有变化，一点有变化则获取获取hash值。代码如下：

先是a.com下的文件cs1.html文件：

function startRequest(){

var ifr = document.createElement('iframe');

ifr.style.display = 'none';

ifr.src = 'http://www.cnblogs.com/lab/cscript/cs2.html#paramdo';

document.body.appendChild(ifr);

}

function checkHash() {

try {

var data = location.hash ? location.hash.substring(1) : '';

if (console.log) {

console.log('Now the data is '+data);

}

} catch(e) {};

}

setInterval(checkHash, 2000);

cnblogs.com域名下的cs2.html:

//模拟一个简单的参数处理操作

switch(location.hash){

case '#paramdo':

callBack();

break;

case '#paramset':

//do something……

break;

}

function callBack(){

try {

parent.location.hash = 'somedata';

} catch (e) {

// ie、chrome的安全机制无法修改parent.location.hash，

// 所以要利用一个中间的cnblogs域下的代理iframe

var ifrproxy = document.createElement('iframe');

ifrproxy.style.display = 'none';

ifrproxy.src = 'http://a.com/test/cscript/cs3.html#somedata'; // 注意该文件在"a.com"域下

document.body.appendChild(ifrproxy);

}

}

a.com下的域名cs3.html

//因为parent.parent和自身属于同一个域，所以可以改变其location.hash的值

parent.parent.location.hash = self.location.hash.substring(1);

当然这样做也存在很多缺点，诸如数据直接暴露在了url中，数据容量和类型都有限等……

**4、window.name实现的跨域数据传输**

文章较长列在此处不便于阅读，详细请看 [window.name实现的跨域数据传输](http://www.cnblogs.com/rainman/archive/2011/02/21/1960044.html)。

**5、使用HTML5 postMessage**

[HTML5](http://dev.w3.org/html5/spec/)中最酷的新功能之一就是 [跨文档消息传输Cross Document Messaging](http://www.whatwg.org/specs/web-apps/current-work/#crossDocumentMessages)。下一代浏览器都将支持这个功能：Chrome 2.0+、Internet Explorer 8.0+, Firefox 3.0+, Opera 9.6+, 和 Safari 4.0+ 。 Facebook已经使用了这个功能，用postMessage支持基于web的实时消息传递。

**otherWindow.postMessage(message, targetOrigin);**

otherWindow: 对接收信息页面的window的引用。可以是页面中iframe的contentWindow属性；[window.open](https://developer.mozilla.org/en/DOM/window.open)的返回值；通过name或下标从[window.frames](https://developer.mozilla.org/en/DOM/window.frames)取到的值。  
message: 所要发送的数据，string类型。  
targetOrigin: 用于限制otherWindow，“\*”表示不作限制

a.com/index.html中的代码：

<iframe id="ifr" src="b.com/index.html"></iframe>

<script type="text/javascript">

window.onload = function() {

var ifr = document.getElementById('ifr');

var targetOrigin = 'http://b.com'; // 若写成'http://b.com/c/proxy.html'效果一样

// 若写成'http://c.com'就不会执行postMessage了

ifr.contentWindow.postMessage('I was there!', targetOrigin);

};

</script>

b.com/index.html中的代码：

<script type="text/javascript">

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

// 通过origin属性判断消息来源地址

if (event.origin == 'http://a.com') {

alert(event.data); // 弹出"I was there!"

alert(event.source); // 对a.com、index.html中window对象的引用

// 但由于同源策略，这里event.source不可以访问window对象

}

}, false);

</script>

参考文章：[《精通HTML5编程》第五章——跨文档消息机制](http://www.grati.org/?p=429)、<https://developer.mozilla.org/en/dom/window.postmessage>

**6、利用flash**

这是从YUI3的IO组件中看到的办法，具体可见<http://developer.yahoo.com/yui/3/io/>。  
可以看在Adobe Developer Connection看到更多的跨域代理文件规范：[ross-Domain Policy File Specifications](http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html)、[HTTP Headers Blacklist](http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html)。