[js跨越问题解决方法](http://blog.csdn.net/tomcat_baby/article/details/51799622)

标签： [jsonp](http://www.csdn.net/tag/jsonp)

2016-07-01 11:24 833人阅读 [评论](http://blog.csdn.net/tomcat_baby/article/details/51799622#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/tomcat_baby/article/details/51799622#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

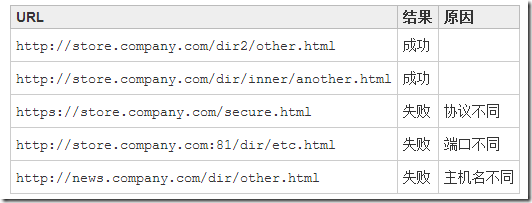
javascript

目录[(?)[+]](http://blog.csdn.net/tomcat_baby/article/details/51799622)

**[js中几种实用的跨域方法原理详解](http://www.cnblogs.com/2050/p/3191744.html" \t "_blank)**

这里说的js跨域是指通过js在不同的域之间进行数据传输或通信，比如用ajax向一个不同的域请求数据，或者通过js获取页面中不同域的框架中(iframe)的数据。只要协议、域名、端口有任何一个不同，都被当作是不同的域。

下表给出了相对http://store.company.com/dir/page.html同源检测的结果:

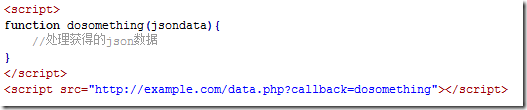
[](http://images.cnitblog.com/blog/130623/201307/15184525-747de461b3b14f938b443e72ea25b25a.png)

要解决跨域的问题，我们可以使用以下几种方法：

**一、通过jsonp跨域**

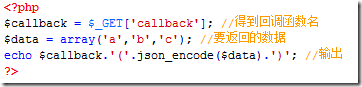
在js中，我们直接用XMLHttpRequest请求不同域上的数据时，是不可以的。但是，在页面上引入不同域上的js脚本文件却是可以的，jsonp正是利用这个特性来实现的。

比如，有个a.html页面，它里面的代码需要利用ajax获取一个不同域上的json数据，假设这个json数据地址是<http://example.com/data.php>,那么a.html中的代码就可以这样：

[](http://images.cnitblog.com/blog/130623/201307/15184526-bcc1971f27094439b746cb1f52a715be.png)

我们看到获取数据的地址后面还有一个callback参数，按惯例是用这个参数名，但是你用其他的也一样。当然如果获取数据的jsonp地址页面不是你自己能控制的，就得按照提供数据的那一方的规定格式来操作了。

因为是当做一个js文件来引入的，所以<http://example.com/data.php>返回的必须是一个能执行的js文件，所以这个页面的[**PHP**](http://lib.csdn.net/base/php)代码可能是这样的:

[](http://images.cnitblog.com/blog/130623/201307/15184527-28a9558c3caa424683980a50b8371134.png)

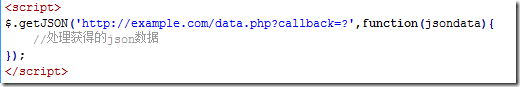
最终那个页面输出的结果是:

[QQ截图20130613230631](http://images.cnitblog.com/blog/130623/201307/15184528-32bbea18197b4e35bb5f67480da1f367.png)

所以通过<http://example.com/data.php?callback=dosomething>得到的js文件，就是我们之前定义的dosomething函数,并且它的参数就是我们需要的json数据，这样我们就跨域获得了我们需要的数据。

这样jsonp的原理就很清楚了，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。所以jsonp是需要服务器端的页面进行相应的配合的。

知道jsonp跨域的原理后我们就可以用js动态生成script标签来进行跨域操作了，而不用特意的手动的书写那些script标签。如果你的页面使用[**jQuery**](http://lib.csdn.net/base/jquery)，那么通过它封装的方法就能很方便的来进行jsonp操作了。

[](http://images.cnitblog.com/blog/130623/201307/15184529-f4a5efe2881a40eebbdca2b463e264bf.png)

原理是一样的，只不过我们不需要手动的插入script标签以及定义回掉函数。jquery会自动生成一个全局函数来替换callback=?中的问号，之后获取到数据后又会自动销毁，实际上就是起一个临时代理函数的作用。$.getJSON方法会自动判断是否跨域，不跨域的话，就调用普通的ajax方法；跨域的话，则会以异步加载js文件的形式来调用jsonp的回调函数。

**2、通过修改document.domain来跨子域**

浏览器都有一个同源策略，其限制之一就是第一种方法中我们说的不能通过ajax的方法去请求不同源中的文档。 它的第二个限制是浏览器中不同域的框架之间是不能进行js的交互操作的。有一点需要说明，不同的框架之间（父子或同辈），是能够获取到彼此的window对象的，但蛋疼的是你却不能使用获取到的window对象的属性和方法([**HTML5**](http://lib.csdn.net/base/html5)中的postMessage方法是一个例外，还有些浏览器比如ie6也可以使用top、parent等少数几个属性)，总之，你可以当做是只能获取到一个几乎无用的window对象。比如，有一个页面，它的地址是<http://www.example.com/a.html>  ， 在这个页面里面有一个iframe，它的src是<http://example.com/b.html>, 很显然，这个页面与它里面的iframe框架是不同域的，所以我们是无法通过在页面中书写js代码来获取iframe中的东西的：

[](http://images.cnitblog.com/blog/130623/201307/15184530-8b5ff5011fc24c7a9949eac174a4dc73.png)

这个时候，document.domain就可以派上用场了，我们只要把<http://www.example.com/a.html> 和<http://example.com/b.html>这两个页面的document.domain都设成相同的域名就可以了。但要注意的是，document.domain的设置是有限制的，我们只能把document.domain设置成自身或更高一级的父域，且主域必须相同。例如：a.b.example.com 中某个文档的document.domain 可以设成a.b.example.com、b.example.com 、example.com中的任意一个，但是不可以设成 c.a.b.example.com,因为这是当前域的子域，也不可以设成baidu.com,因为主域已经不相同了。

在页面 <http://www.example.com/a.html> 中设置document.domain:

[](http://images.cnitblog.com/blog/130623/201307/15184531-49f512a730204cab803ce03833e4b554.png)

在页面 <http://example.com/b.html> 中也设置document.domain，而且这也是必须的，虽然这个文档的domain就是example.com,但是还是必须显示的设置document.domain的值：

[QQ截图20130613230631](http://images.cnitblog.com/blog/130623/201307/15184533-f10eef95c6994d09bc2d595325f45759.png)

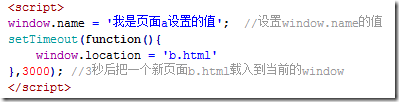
这样我们就可以通过js访问到iframe中的各种属性和对象了。

不过如果你想在<http://www.example.com/a.html> 页面中通过ajax直接请求<http://example.com/b.html> 页面，即使你设置了相同的document.domain也还是不行的，所以修改document.domain的方法只适用于不同子域的框架间的交互。如果你想通过ajax的方法去与不同子域的页面交互，除了使用jsonp的方法外，还可以用一个隐藏的iframe来做一个代理。原理就是让这个iframe载入一个与你想要通过ajax获取数据的目标页面处在相同的域的页面，所以这个iframe中的页面是可以正常使用ajax去获取你要的数据的，然后就是通过我们刚刚讲得修改document.domain的方法，让我们能通过js完全控制这个iframe，这样我们就可以让iframe去发送ajax请求，然后收到的数据我们也可以获得了。

**3、使用window.name来进行跨域**

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置。

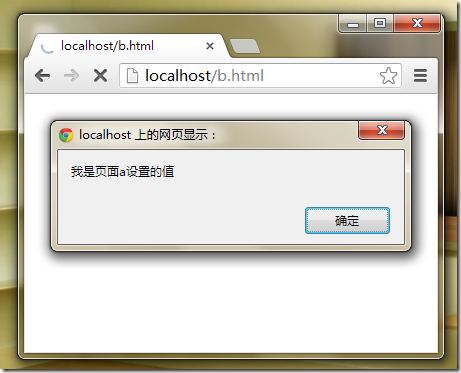
比如：有一个页面a.html,它里面有这样的代码：

[](http://images.cnitblog.com/blog/130623/201307/15184534-769ff1574c304d11accd26cbbbd1f9da.png)

再看看b.html页面的代码：

[QQ截图20130613230631](http://images.cnitblog.com/blog/130623/201307/15184535-fd6a9a0604754241a8f8947ced97b8e9.png)

a.html页面载入后3秒，跳转到了b.html页面，结果为：

[](http://images.cnitblog.com/blog/130623/201307/15184536-5eadbc78e1fc4c8bb573aefa486ca52c.png)

我们看到在b.html页面上成功获取到了它的上一个页面a.html给window.name设置的值。如果在之后所有载入的页面都没对window.name进行修改的话，那么所有这些页面获取到的window.name的值都是a.html页面设置的那个值。当然，如果有需要，其中的任何一个页面都可以对window.name的值进行修改。注意，window.name的值只能是字符串的形式，这个字符串的大小最大能允许2M左右甚至更大的一个容量，具体取决于不同的浏览器，但一般是够用了。

上面的例子中，我们用到的页面a.html和b.html是处于同一个域的，但是即使a.html与b.html处于不同的域中，上述结论同样是适用的，这也正是利用window.name进行跨域的原理。

下面就来看一看具体是怎么样通过window.name来跨域获取数据的。还是举例说明。

比如有一个[www.example.com/a.html](http://www.example.com/a.html)页面,需要通过a.html页面里的js来获取另一个位于不同域上的页面[www.cnblogs.com/data.html](http://www.cnblogs.com/data.html)里的数据。

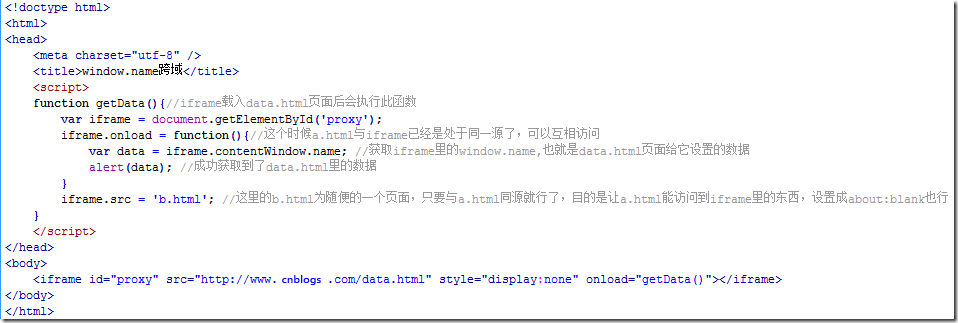
data.html页面里的代码很简单，就是给当前的window.name设置一个a.html页面想要得到的数据值。data.html里的代码：

[QQ截图20130613230631](http://images.cnitblog.com/blog/130623/201307/15184538-f329d65db57d49c3b89bef6d413e34ad.png)

那么在a.html页面中，我们怎么把data.html页面载入进来呢？显然我们不能直接在a.html页面中通过改变window.location来载入data.html页面，因为我们想要即使a.html页面不跳转也能得到data.html里的数据。答案就是在a.html页面中使用一个隐藏的iframe来充当一个中间人角色，由iframe去获取data.html的数据，然后a.html再去得到iframe获取到的数据。

充当中间人的iframe想要获取到data.html的通过window.name设置的数据，只需要把这个iframe的src设为[www.cnblogs.com/data.html](http://www.cnblogs.com/data.html)就行了。然后a.html想要得到iframe所获取到的数据，也就是想要得到iframe的window.name的值，还必须把这个iframe的src设成跟a.html页面同一个域才行，不然根据前面讲的同源策略，a.html是不能访问到iframe里的window.name属性的。这就是整个跨域过程。

看下a.html页面的代码：



上面的代码只是最简单的原理演示代码，你可以对使用js封装上面的过程，比如动态的创建iframe,动态的注册各种事件等等，当然为了安全，获取完数据后，还可以销毁作为代理的iframe。网上也有很多类似的现成代码，有兴趣的可以去找一下。

通过window.name来进行跨域，就是这样子的。

**4、使用**[**Html5**](http://lib.csdn.net/base/html5)**中新引进的window.postMessage方法来跨域传送数据**

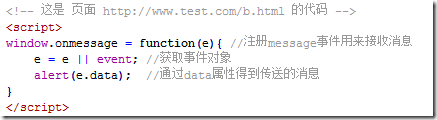
window.postMessage(message,targetOrigin)  方法是html5新引进的特性，可以使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源，目前IE8+、FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。

调用postMessage方法的window对象是指要接收消息的那一个window对象，该方法的第一个参数message为要发送的消息，类型只能为字符串；第二个参数targetOrigin用来限定接收消息的那个window对象所在的域，如果不想限定域，可以使用通配符 \*  。

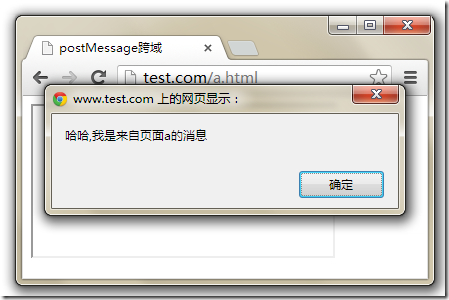
需要接收消息的window对象，可是通过监听自身的message事件来获取传过来的消息，消息内容储存在该事件对象的data属性中。

上面所说的向其他window对象发送消息，其实就是指一个页面有几个框架的那种情况，因为每一个框架都有一个window对象。在讨论第二种方法的时候，我们说过，不同域的框架间是可以获取到对方的window对象的，而且也可以使用window.postMessage这个方法。下面看一个简单的示例，有两个页面

[](http://images.cnitblog.com/blog/130623/201307/15184544-594f30a8bacd4da0a4a2293f63f53534.png)

[](http://images.cnitblog.com/blog/130623/201307/15184545-4285b632b74c4abea9aba5ecb2587f5a.png)

我们运行a页面后得到的结果:

[](http://images.cnitblog.com/blog/130623/201307/15184547-b7a7e7b771054807b112248f39201e53.png)

我们看到b页面成功的收到了消息。

使用postMessage来跨域传送数据还是比较直观和方便的，但是缺点是IE6、IE7不支持，所以用不用还得根据实际需要来决定。

**[服务器端解决跨越问题](http://www.cnblogs.com/2050/p/3191744.html" \t "_blank)**

1、 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。

 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。

**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。

Java:使用下面代码片段来实现服务器端对客户端的跨越请求

**[java]** [view plain](http://blog.csdn.net/tomcat_baby/article/details/51799622) [copy](http://blog.csdn.net/tomcat_baby/article/details/51799622)

[在CODE上查看代码片](https://code.csdn.net/snippets/1738985)

1. **public** **static** String gettrustURLStr(String configName){
2. ResourceBundle ms = **null**;
3. String str="";
4. **try**{
5. **if** (ms == **null**)
6. ms = PropertyResourceBundle.getBundle("xxxx");//读取配置文件
7. str = ms.getString(configName).trim();//获取配置文件中配置的信任域（ps:配置可以正常跨域访问的客户端白名单）
8. }**catch**(Exception e){
9. e.printStackTrace();
10. }
11. **return** str;
12. }
13. **private** **static** **void** checkAndSetTrustURL(HttpServletRequest request,HttpServletResponse response){
14. **boolean** ret=**false**;
15. String requestURI= (String)request.getHeader("referer");//获取客户端主机域
16. String trustURLStr=gettrustURLStr("ebank.system.trustURL");     //获取信任域
17. String[] trustURLStrArray=trustURLStr.split(",");
18. **for**(**int** i=0;i<trustURLStrArray.length;i++)
19. {
20. **if**(StringUtil.isNullOrBlank(requestURI)){
21. **break**;
22. }
23. **if**(requestURI.indexOf(trustURLStrArray[i])!=-1){
24. response.addHeader("Access-Control-Allow-Origin",trustURLStrArray[i]);//通过设置响应头里的Header，来指定可以跨域访问的客户端
25. ret=**true**;**break**;
26. }
27. }
28. **if**(ret==**false**)
29. response.addHeader("Access-Control-Allow-Origin",gettrustURLStr("ebank.system.trustURL"));
30. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)