JS跨域：<https://www.cnblogs.com/yongshaoye/p/7423881.html>

自我尝试前两种：

|  |
| --- |
| $.get("http://www.afd.com/lesson\_plan/test", function (*$data*) {  alert(*$data*); },'json')  php端：  构造函数里加上：  *header*('Access-Control-Allow-Origin:\*'); *header*('Access-Control-Allow-Methods:GET,POST'); *header*('Access-Control-Allow-Headers:x-requested-with,content-type');  然后正常输出即可：  $data = array(1,2,4);  echo *json\_encode*($data);  2.jsonp格式  $.getJSON("http://www.afd.com/lesson\_plan/test?callback=?", function (*$data*) {  alert(*$data*); })  $callback = $this->input->get\_post('callback'); $data = array(1,2,4); echo $callback.'('.*json\_encode*($data).')';  //PHP返回时，需要拼成callback({‘’:’’})格式 |

**一、什么是跨域？**

1.定义：跨域是指从一个域名的网页去请求另一个域名的资源。比如从www.baidu.com 页面去请求 www.google.com 的资源。但是一般情况下不能这么做，它是由浏览器的同源策略造成的，是浏览器对[JavaScript](http://lib.csdn.net/base/javascript)施加的安全限制。跨域的严格一点的定义是：只要 协议，域名，端口有任何一个的不同，就被当作是跨域

**所谓同源是指，域名，协议，端口均相同。**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。

概念：只要协议、域名、端口有任何一个不同，都被当作是不同的域。

http://www.123.com/index.html 调用 http://www.123.com/server.[PHP](http://lib.csdn.net/base/php) （非跨域）

http://www.123.com/index.html 调用 http://www.456.com/server.[php](http://lib.csdn.net/base/php) （主域名不同:123/456，跨域）

http://abc.123.com/index.html 调用 http://def.123.com/server.php （子域名不同:abc/def，跨域）

http://www.123.com:8080/index.html 调用 http://www.123.com:8081/server.php （端口不同:8080/8081，跨域）

http://www.123.com/index.html 调用 https://www.123.com/server.php （协议不同:http/https，跨域）

请注意：localhost和127.0.0.1虽然都指向本机，但也属于跨域。

浏览器执行[javascript](http://lib.csdn.net/base/javascript)脚本时，会检查这个脚本属于哪个页面，如果不是同源页面，就不会被执行。

对于端口和协议的不同，只能通过后台来解决。

**二、为什么浏览器要限制跨域访问呢？**

原因就是安全问题：如果一个网页可以随意地访问另外一个网站的资源，那么就有可能在客户完全不知情的情况下出现安全问题。比如下面的操作就有安全问题：

1. 用户访问www.mybank.com ，登陆并进行网银操作，这时cookie啥的都生成并存放在浏览器
2. 用户突然想起件事，并迷迷糊糊地访问了一个邪恶的网站 www.xiee.com
3. 这时该网站就可以在它的页面中，拿到银行的cookie，比如用户名，登陆token等，然后发起对www.mybank.com 的操作。
4. 如果这时浏览器不予限制，并且银行也没有做响应的安全处理的话，那么用户的信息有可能就这么泄露了。

**三、为什么要跨域？**

既然有安全问题，那为什么又要跨域呢？ 有时公司内部有多个不同的子域，比如一个是location.company.com ,而应用是放在app.company.com , 这时想从 app.company.com去访问 location.company.com 的资源就属于跨域。

**四、解决跨域问题的方法**

**（1）跨域资源共享（CORS）**

CORS（Cross-Origin Resource Sharing）跨域资源共享，定义了必须在访问跨域资源时，浏览器与服务器应该如何沟通。CORS背后的基本思想就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功还是失败。

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。

只需要在后台中加上响应头来允许域请求！在被请求的Response header中加入以下设置，就可以实现跨域访问了！

[制代码](javascript:void(0);)

//指定允许其他域名访问

'Access-Control-Allow-Origin:\*'//或指定域

//响应类型

'Access-Control-Allow-Methods:GET,POST'

//响应头设置

'Access-Control-Allow-Headers:x-requested-with,content-type'

[制代码](javascript:void(0);)

**（2）通过jsonp跨域**

JSONP是JSON with Padding（填充式json）的简写，是应用JSON的一种新方法，只不过是被包含在函数调用中的JSON，例如：

callback({"name","trigkit4"});

JSONP由两部分组成：**回调函数**和**数据**。回调函数是当响应到来时应该在页面中调用的函数，而数据就是传入回调函数中的JSON数据。

**JSONP的原理**：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。（即用[javascript](http://lib.csdn.net/base/javascript)动态加载一个script文件，同时定义一个callback函数给script执行而已。）

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。 例如：

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：

[制代码](javascript:void(0);)

<script type="text/javascript">

function dosomething(jsondata){

//处理获得的json数据

}

</script>

<script src="http://example.com/data.php?callback=dosomething"></script>

[制代码](javascript:void(0);)

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。

<?php

$callback = $\_GET['callback'];//得到回调函数名

$data = array('a','b','c');//要返回的数据

echo $callback.'('.json\_encode($data).')';//输出

?>

最终，输出结果为：dosomething(['a','b','c']);



如果你的页面使用jquery，那么通过它封装的方法就能很方便的来进行jsonp操作了。

<script type="text/javascript">

$.getJSON('http://example.com/data.php?callback=?,function(jsondata)'){

//处理获得的json数据

});

</script>

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。

**JSONP的优缺点**

JSONP的优点是：它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行，不需要XMLHttpRequest或ActiveX的支持；并且在请求完毕后可以通过调用callback的方式回传结果。

JSONP的缺点则是：它只支持GET请求而不支持POST等其它类型的HTTP请求；它只支持跨域HTTP请求这种情况，不能解决不同域的两个页面之间如何进行JavaScript调用的问题。

**CORS和JSONP对比**

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）。

**（3）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。  
不同的框架之间是可以获取window对象的，但却无法获取相应的属性和方法。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：

[制代码](javascript:void(0);)

<script type="text/javascript">

function test(){

var iframe = document.getElementById('￼ifame');

var win = document.contentWindow;//可以获取到iframe里的window对象，但该window对象的属性和方法几乎是不可用的

var doc = win.document;//这里获取不到iframe里的document对象

var name = win.name;//这里同样获取不到window对象的name属性

}

</script>

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

[制代码](javascript:void(0);)

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。

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

[制代码](javascript:void(0);)

<iframe id = "iframe" src="http://example.com/b.html" onload = "test()"></iframe>

<script type="text/javascript">

document.domain = 'example.com';//设置成主域

function test(){

alert(document.getElementById('￼iframe').contentWindow);//contentWindow 可取得子窗口的 window 对象

}

</script>

[制代码](javascript:void(0);)

2.在页面 http://example.com/b.html 中也设置document.domain:

<script type="text/javascript">

document.domain = 'example.com';//在iframe载入这个页面也设置document.domain，使之与主页面的document.domain相同

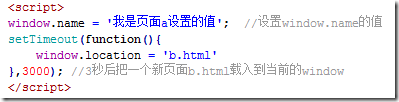
</script>

修改document.domain的方法**只适用于不同子域的框架间的交互**。

**（4）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。

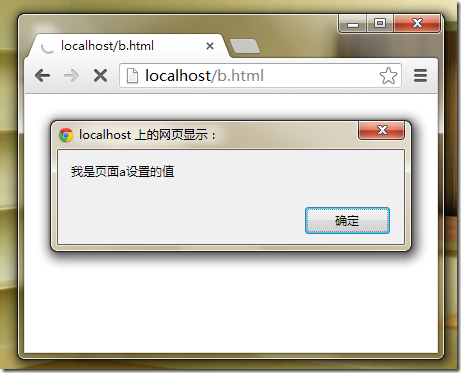
比如：有一个页面a.html,它里面有这样的代码：

****

再看看b.html页面的代码：

https://images2017.cnblogs.com/blog/1191190/201708/1191190-20170824162633589-2143316311.png

a.html页面载入后3秒，跳转到了b.html页面，结果为：



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：

https://images2017.cnblogs.com/blog/1191190/201708/1191190-20170824162805886-1803632899.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。

**（5）使用HTML5的window.postMessage方法跨域**

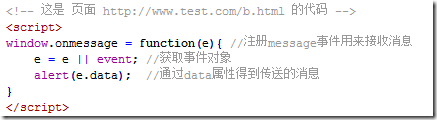
**window.postMessage**(message,targetOrigin)  方法是html5新引进的特性，可以使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源，目前IE8+、FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。

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 \*  。

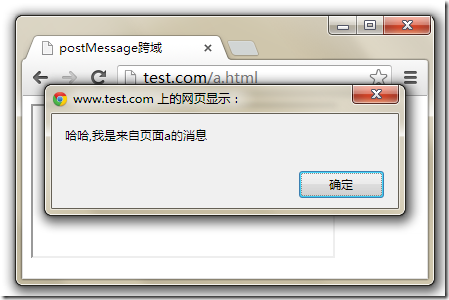
需要接收消息的window对象，可是通过监听自身的message事件来获取传过来的消息，消息内容储存在该事件对象的data属性中。

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





我们运行a页面后得到的结果:



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。

**（6）web sockets**

web sockets是一种浏览器的API，它的目标是在一个单独的持久连接上提供全双工、双向通信。(同源策略对web sockets不适用)

web sockets原理：在[js](http://lib.csdn.net/base/javascript)创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。

只有在支持web socket协议的服务器上才能正常工作。

var socket = new WebSockt('ws://www.baidu.com');//http->ws; https->wss

socket.send('hello WebSockt');

socket.onmessage = function(event){

var data = event.data;

}

**（7）图像ping（单向）**

    1、什么是图像ping：

        图像ping是与服务器进行简单、单向的跨域通信的一种方式，请求的数据是通过查询字符串的形式发送的，而相应可以是任意内容，但通常是像素图或204相应（No Content）。 图像ping有两个主要缺点：首先就是只能发送get请求，其次就是无法访问服务器的响应文本。

   2、使用方法：

[制代码](javascript:void(0);)

var img = new Image();

img.onload = img.onerror = function(){

alert("done!");

};

img.src = "https://raw.githubusercontent.com/zhangmengxue/Todo-List/master/me.jpg";

document.body.insertBefore(img,document.body.firstChild);

[制代码](javascript:void(0);)

然后页面上就可以显示我放在我的github上某个地方的照片啦。

与<img>类似的可以跨域内嵌资源的还有:

(1)<script src=""></script>标签嵌入跨域脚本。语法错误信息只能在同源脚本中捕捉到。上面jsonp也用到了呢。

(2) <link src="">标签嵌入[CSS](http://www.2cto.com/kf/qianduan/css/)。由于CSS的松散的语法规则，CSS的跨域需要一个设置正确的Content-Type消息头。不同浏览器有不同的限制： IE, Firefox, Chrome, Safari (跳至CVE-2010-0051)部分 和 Opera。

(3)<video> 和 <audio>嵌入多媒体资源。

(4)<object>, <embed> 和 <applet>的插件。

(5)@font-face引入的字体。一些浏览器允许跨域字体（ cross-origin fonts），一些需要同源字体（same-origin fonts）。

(6) <frame> 和 <iframe>载入的任何资源。站点可以使用X-Frame-Options消息头来阻止这种形式的跨域交互。