# 跨域问题

在我们前面的接口开发当中，我们比较顺利的完成了后台的接口开发。但是，随着我们开发前台接口的过程中，利用postMan去测试接口。发现可以通过，可以成功。但是，一旦我们利用我们正在开发的客户端去调试接口。发现不能成功通过。这是为什么呢？

这就是我们这一次课程所需要掌握的。

那么，什么是跨域呢？

## 跨域

**官方解释：**

跨域，指的是浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是**浏览器**施加的安全限制。

**通俗解释：**

跨域，既跨域名访问。其中，又分为二级域名跨域、多级域名跨域、以及协议跨域、端口号跨域。

那么，我们在开发过程中，开发后台的地址为http://localhost:3000

前台Vue脚手架开发地址为http://localhost:8080

我们跨域了。该怎么解决呢？

典型的解决跨域问题的方案主要为以下两种：

跨域的几种情况： A网站向各访问

A 网站的网址是 http://[www.baidu.com](http://www.baidu.com)

B 网站的网址是 <http://baidu.com> 二级域名跨域

C 网站的网址是 <http://www.baidu.cn> 主域名跨域

D 网站的网址是 <http://www.baidu.com/index> 不跨域

E 网站的网址是 <https://www.baidu.com> 协议跨域

## jsonp跨域

什么是jsonp呢？实际上来说。jsonp允许跨域，是绕过了ajax。在浏览器当中。img标签的src、以及script的src、以及link的href是不受同源政策限制的。那么，我们就可以利用动态创建script标签的方式，去跨域请求数据了。

那么，我们利用express来建立一个jsonp数据。

用法如下：

app.get("/jsonp",(req, res, next) => {  
 res.jsonp({data: '我是jsonp的数据'})  
})

那么。我们该怎么去请求这个接口呢？

ajax？不是。实际上jsonp并不是利用ajax去请求的，而是通过一个script标签去请求的数据。那么，显然，我们需要动态的创建这个script标签。

代码如下：

**第一步：**

定义一个处理函数。处理接收回来的数据

function getData (res) {  
 console.log(res);  
}

**第二步：**

创建一个script标签去请求数据。

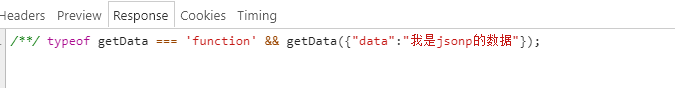
var jsonp = document.createElement("script")  
jsonp.src = "http://localhost:3000/jsonp?callback=getData"

**第三步：**

讲这个script标签添加到页面当中去。

document.body.appendChild(jsonp)

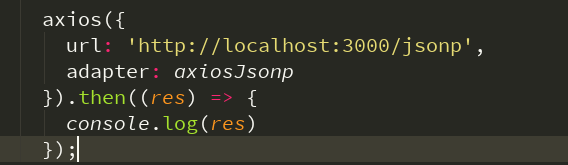
然后打开我们的页面。利用network去查看请求的数据。





当然，实际开发中。我们不可能这么繁琐的请求数据。而是使用插件进行。用法如下：

import axios from 'axios'  
let axiosJsonp = require('axios-jsonp');



### jsonp的优势：

使用起来比较简单。因为几乎所有的浏览器都支持script标签，所以。兼容性极好。

### jsonp的劣势：

只支持get请求。

## cors跨域

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

那么应该怎么使用CORS进行跨域请求呢？

整个CORS跨域的过程，都是浏览器自动完成的。对于前端开发工程师而言。跨域的ajax和不跨域的ajax几乎没有差别。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。

那么cors将我们的请求分成两种。

一种是简单请求（simple request），一种是非简单请求（simple request）。

### 简单请求

对于简单请求来说。

同时满足以下两个条件，既是简单请求

（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

axios的content-type是post下是： application/json

简单请求的流程：

第一步：浏览器直接发出cors请求。

发送cors中，会自动带上一个请求头

Origin：客户端域名

第二步：服务器根据请求返回内容。

第三步：浏览器对返回的请求头进行分析。

如果存在请求头为

Access-Control-Allow-Oringin：”客户端的域名”;

那么，浏览器就会将请求到的数据返回给ajax对象。否则，浏览器将会触发ajax对象的onerror函数抛出一个错误。

这种错误没有办法通过http状态码捕获，因为服务端是正确响应了的。也就是正常的200.

### 非简单请求

对于非简单请求来说。

请求过程会分为以下几步。

第一步：

先发出一个options的请求。该请求会带上一些头信息

Origin： 客户端域名

Access-Control-Request-Headers: content-type (表明客户端发送的请求头数据)

Access-Control-Request-Method: POST （表明客户端发送的请求类型）

第二步：

服务器返回options请求

Access-Control-Allow-Origin：允许跨域的域名（可以是指定域名，也可以是全域名“\*”）

Access-Control-Allow-Headers: “与客户端对应”

Access-Control-Allow-Method： 与客户端的对应或者大于客户端

第三步：

客户端检查options请求的响应头。如果

Access-Control-Allow-Origin：允许跨域的域名

Access-Control-Allow-Headers: “与客户端对应”

Access-Control-Allow-Method： 与客户端的对应或者大于客户端

如果对应上了。那么，浏览器正式发起一个请求。

第四步：

服务器对请求做出回应。同时，带上之前发送的请求头。完成跨域过程。

那么，回到我们的问题当中。我们之前，做客户端登录的时候。明明登陆了。但是用另一个接口请求，却丢失了session状态。为什么？

因为纵使cors允许跨域了。但是，默认情况下Access-Control-Allow-Credentials = false;

也就是说，指明了客户端不会携带cookies到服务端。

那么，我们的session将无法根据cookie获取到用户的登录信息。

解决方案：

1在响应头里边，指定唯一的允许跨域的域名。

2服务端在响应头里边。指定Access-Control-Allow-Credentials：true

3客户端的ajax对象，必须指定withCredentials: true；

相关知识点：

http://www.ruanyifeng.com/blog/2016/04/cors.html

## 代理跨域

前面根据跨域的概念，显而易见的指出了。跨域是因为浏览器的同源政策引起的。那么换句话说。是否只有浏览器存在跨域问题？

答案是显然的。

那么，既然只有浏览器存在跨域，我们就可以使用服务器将跨域的接口转发过来就好了。

实现代码如下

app.get('/getCors', (req, res) => {  
 request("http://localhost:3000/getCors",(err, response, body) => {  
 res.json(response)  
 })  
})

那么，我们是否针对每个接口都这么转发是否是太累了？代码太冗余了。怎么办？

### 方案一：（程序员方案）使用http-proxy-middleware

第一步：安装中间件

npm install --save-dev http-proxy-middleware

第二步：使用中间件



### 方案二：（懒人方案）使用json-bird

看文档

https://github.com/xCss/JsonBird/wiki

用法示例：



## 四、Vue脚手架当中，跨域的方案

经过前面的学习。我们很容易的选择一个适合我们的方案。即三种跨域方式里边。唯一适合我们的就是cors了。那么我们真的需要使用么？

事实上，vue脚手架就是一个服务器开启的服务。那么，我们是否可以使用代理跨域解决跨域问题呢？

答案是可以的。Webpack给我们提供了一个proxyTable的方法。

相关资料：

https://blog.csdn.net/qq\_33559304/article/details/72966028