跨域

什么是跨域

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，是浏览器对javascript施加的安全限制。

错误表示：

> 从原产地“http://127.0.0.1:5500”访问“http://127.0.0.1:3001/user”的XMLHttpRequest已被CORS策略阻止:被请求的资源上没有“Access- control - allow - origin”报头。

同源策略

同源策略（Same origin policy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

同源策略，它是由Netscape提出的一个著名的安全策略。。现在所有支持JavaScript 的浏览器都会使用这个策略。

判断以下内容是否能够正常访问：

URL	结果	原因
http://store.company.com/dir2/other.html	成功	
http://store.company.com/dir/inner/another.html	成功	
https://store.company.com/secure.html	失败	协议不同
http://store.company.com:81/dir/etc.html	失败	端口不同
http://news.company.com/dir/other.html	失败	主机名不同

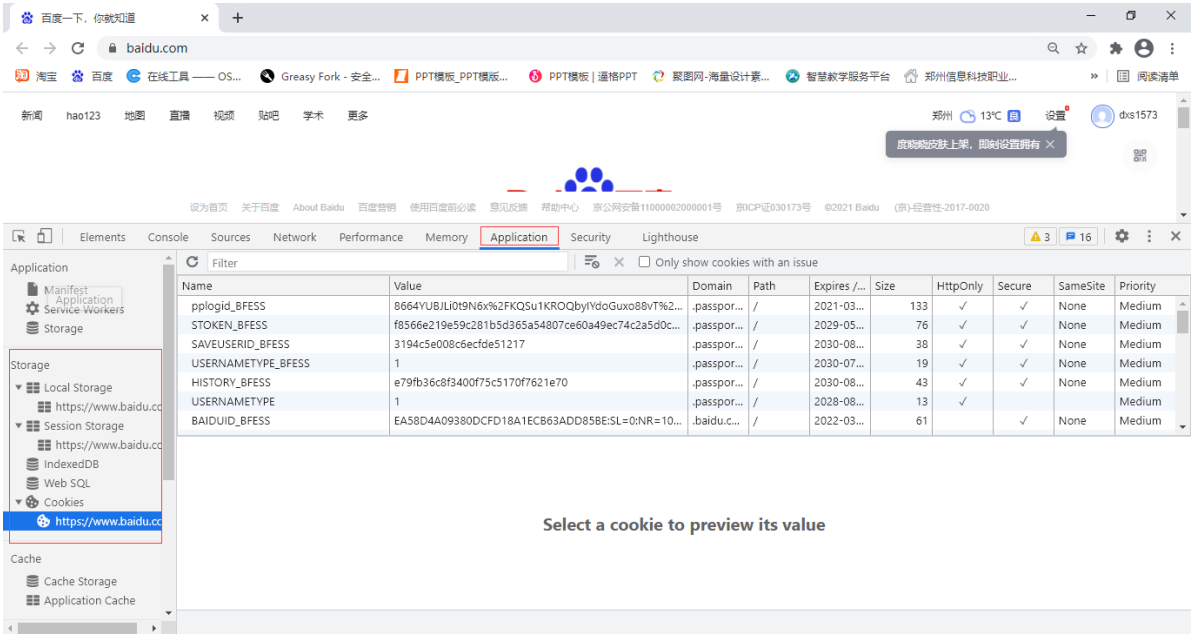
跨域的限制

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。

- 无法读取非同源网页的 Cookie、LocalStorage 和 IndexedDB。【缓存数据】
- 无法接触非同源网页的 DOM。
- 无法向非同源地址发送 AJAX 请求（可以发送，但浏览器会拒绝接受响应）

查看LocalStorage:

右键，检查，application，localStroage



跨域解决方案

跨域方案：
a。添加响应头实现跨域（还行）
b。使用cors模块包实现跨域（推荐）
c。JSONP方式实现跨域（不推荐）
c。使用代理服务器
注： a和b本质都是使用响应头实现跨域的，称为跨源资源共享，即cors

1、设置CORS跨域资源共享

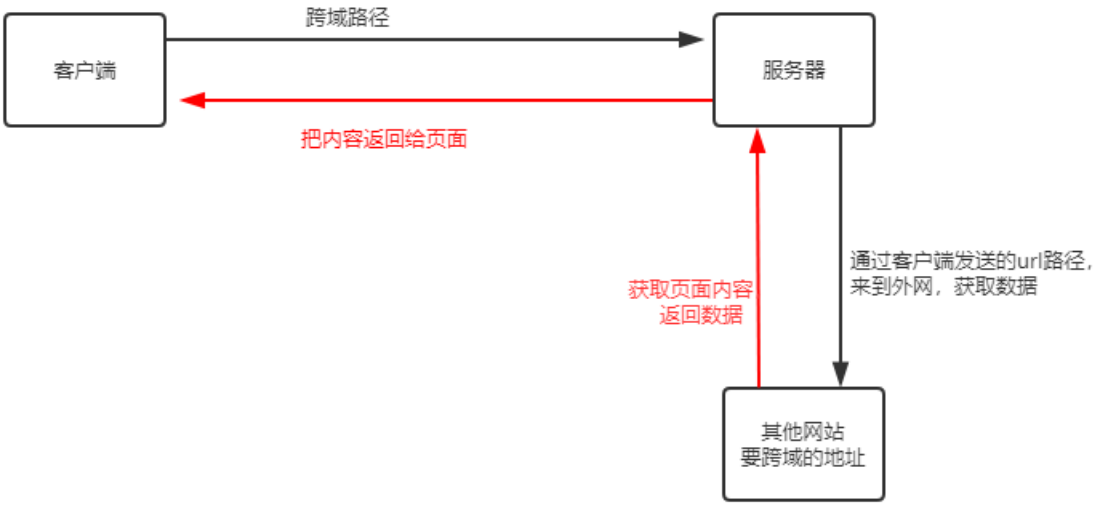
后台在请求头信息里面添加：服务端：、

```
    response.setHeader("Access-Control-Allow-Origin", "*");
```

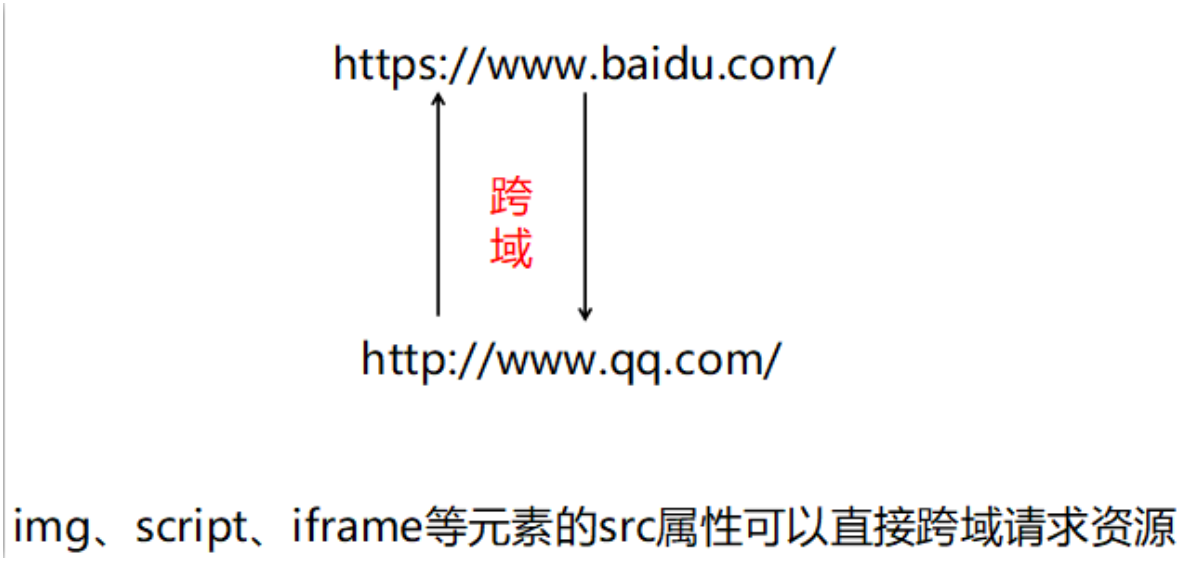
“*”表示所有的域都可以接受
CORS 需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能。

2、Ajax 跨域请求:代理

由于在工作中需要使用AJAX请求其他域名下的资源，但是会出现拒绝访问的情况，这是因为基于安全的考虑，AJAX只能访问本地,同域的资源，而不能跨域访问
可以让服务器去别的网站获取内容然后返回页面



3、Jsonp



例：

通过img的src属性，访问百度的图片：

```
 <img
src="https://ss0.bdstatic.com/70cFuHSh_Q1YnxGkpoWK1HF6hhy/it/u=1910846161,216642
357&fm=26&gp=0.jpg" alt="">
```

> jsonp就是利用script标签的跨域能力请求资源

Jsonp 实现思路:

- 全称是 JSON with Padding，请求时通过动态创建一个 Script，在 Script 中发出请求，

- 通过这种变通的方式让请求资源可以跨域。
- 它不是一个官方协议，是一个约定，约定请求的参数里面如果包含指定的参数（默认是 callback），就说明是一个 JSONP 请求，服务器发现是 JSONP 请求，就会把原来的返回对象变成 JS 代码。
- JS 代码是函数调用的形式，它的函数名是 callback 的值，它的函数的参数就是原来需要返回的结果。
- 后台会把函数调用，重新返回给前端

json_01.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>模拟jsonp</title>
</head>
<body>
   <script>
    //    调用后台函数
       function fn(value){
           console.log(value);
       }
   </script>
   <!--
       通过script的src属性进行跨域
       传递的参数默认是callback
       后台看到callback就能够判断出来是一个jsonp请求
       会根据callback的值，自动生成函数，函数名就是callback的值
       后台把数据封装成对象，作为参数传递给函数作为值
    -->
   <script src="./testJs.js?callback=fn"></script>
</body>
</html>
```

testjs.js

```
// testjs.js就是我们模拟的后台
// 后台能判断出来是否是jsonp请求
// 如果是，就会生成一个函数的调用，而函数名，就是前端提交上来的callback的值在假如我们要给前端返回一个数据



var data = {
    "name":"张三",
    age : 20,
    address:"郑州"
}

// 在后台生成了函数的调用，把数据以参数的形式，进行传递
// 后台会把这个函数的调用，重新返回给前端
fn(data);
```

JSONP优缺点：

优点：兼容性强&不受同源策略的限制
缺点：只能用get方法，不能使用post方法

因为get请求方式把请求参数放在了url地址中，后台解析jsonp是通过url的callback参数进行判断的，所以之支持get请求

优化jsonp.html,动态创建script的src属性：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>

    <button onclick="getJsonp()">获取jsonp</button>


    <script>
        function test(v){
            console.log(v);
        }

        function getJsonp(){
            var script_ = document.createElement('script');
            script_.src = 'http://localhost:3003/jsonp?callback=test';
            document.body.appendChild(script_);

            //清除页面上的script标签
            if(script_){
                document.body.removeChild(script_);
            }
        }
    </script>



</body>

</html>
```