**跨域解决方法：**

1. Jsonp （常用）（**只支持get请求，其他请求都无法实现，所以真实项目中只把：从服务器获取信息的需求，可能采用jsonp跨域的方式来处理，对于post等其他方式需要另起它法**）
2. Cors （常用）
3. Window.name
4. Document.domain （特定场景）
5. PostMessage(H5)
6. Webpack proxy (webScoket) （常用）
7. Ngix反向代理

**什么是跨域？**

拿当前html地址和在页面中ajax请求的api地址做比较：

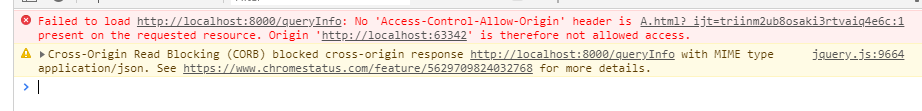
如果两个地址的协议域名端口号都相同，相当于html页面从同一个源下根据某个地址获取数据，属于“同源策略请求”，基于ajax是可以直接请求到数据的

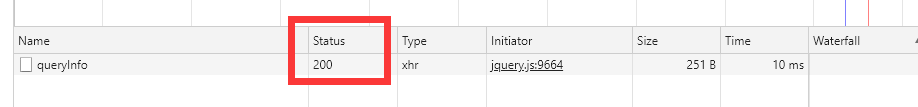
如果三者（协议域名端口号）只要有一个不一样，那么就是“非同源策略请求（跨域请求）”，使用ajax不能直接获取数据了。

当前HTML页面的地址：HTTP：//localhost:8000/A.html

在页面中ajax请求的地址：<http://localhost:8000/queryInfo>

不同源出现的下列问题：





**跨域的普遍性？**

现在的项目一般都是前后端分离的，大部分公司都会把后台的程序用一个新的服务管理，把客户端程序也用一个新的服务管理，两个服务不是同一个源：这样就导致客户端是向其他源发送ajax请求，跨域称为请求的阻碍问题：

同源：把客户端程序和服务器程序在一个服务中发布！

方案一：JSONP

在客户端ajax不允许直接的跨域请求，但是很多东西都可以直接的跨域，例如：script，link，img，iframe等（这些标签的src或者href设置任何一个资源请求地址，哪怕是其他源下的，也都没有跨域的限制，直接可以把内容获取到【除非服务器做特殊处理了=>针对这个特点，真实项目中某些js文件加载的都是cdn地址】）

Jsonp可以实现跨域，利用script标签下不存在跨域限制来实现的，需要客户端和服务器端双方面支持配合才可以完成

客户端;:

1. 准备一个全局函数（不能是私有函数）
2. 创建一个script标签，把需要请求的地址放到src属性上，通过问号传递参数的方式，把全局函数传递给服务器

<jsonp>

Function fn(result){

Console.log(result)

}

</jsonp>

<script src=” [http://localhost:9000/queryInfo?callback=fn”></script](http://localhost:9000/queryInfo?callback=fn”%3e%3c/script)>

一般传递函数的属性名都被起做callback，但是这个可以和服务器上商量着改（协商好即可）

基于src地址立即向服务器发送一个http请求，

服务器上

接收客户端的请求信息（script src的请求都是get方式）

1. 获取问号传递的参数的内容，也就是callback后面传递的函数名，（当然callback可以和客户端协商好）
2. 把函数名和需要返回给客户端的数据拼成字符串”函数名（数据）”的格式，
3. 客户端开始渲染返回的额内容=》就是把之前传递进去的函数执行（也就是把fn执行），把括号中的数据（服务器返回的数据）当做参数传递给函数fn了

传递给服务器的函数必须是全局的，否者后期服务器返回的数据，浏览器执行这个函数（在全局下执行）找不到对应的函数

**方案二：cors跨域资源共享**

主要是服务器设置：配置允许跨域的相关头部信息

**res.header("Access-Control-Allow-Origin","http://localhost:8000" ); //允许哪些源可以向这个服务发送ajax请求（通配符是允许所有的源，也可以单独设置某个源<http://localhost:8000这样就是只允许8000>服务过来请求）=>不适用通配符是为了保证接口和数据的安全，不能让所有的源都能访问：而且一旦设置了允许携带凭证过来，则设置\*会被报错，此时只能设置具体的源。只能设置一个单一的源！  
    res.header("Access-Control-Allow-Credentials", true);//是否允许跨域的时候携带凭证（例如：cookie就是凭证的一种，设置为false，客户端和服务器之间不会出传递cookie，这样session存储就失效了）**

**（**

**Xhr.withCredentials=true;//服务器和客户的这个属性都要设置为true，否则客户端传递给服务器端请求中的cookie无法基于请求头设置传递。**

**）  
    res.header("Access-Control-Allow-Headers",**Content-Type,Content-Length,Authorization, Accept,X-Requested-With

**);//允许的请求头部  
    res.header("Access-Control-Allow-Methods", “PUT,POST,GET,DELETE,HEAD,OPTIONS”);//允许的请求方式（options一定要有）**

**//设置options请求的目的：我们把这个请求当做一个试探性请求，当客户端需要向服务器发送请求的时候，首先发送一个options请求，服务器接收到是options请求后，看一下是否允许跨域，允许返回成功，如果服务器不允许跨域，则客户端会出现跨域请求不允许的错误：如果客户端检测到不允许跨域，则后续的请求都不再进行=》客户端axios框架就是这样处理的**

**{**

**If(req.method===’OPTIONS’){**

**Res.send(‘’);**

**Return;**

**}**

**}  
    req.method === 'OPTIONS' ? res.send('CURRENT SERVICES SUPPORT CROSS DOMAIN REQUESTS!') : next();**

**Session中间件**

app.use(session({  
    secret: 'ZFPX',  
    saveUninitialized: false,  
    resave: false,  
    cookie: {maxAge: 1000 \* 60 \* 60 \* 24 \* 30}  
}));

Axios在某些特定场景下，在发送真实请求之前都会发送一个预请求（options）格式的，来验证是否允许跨域

**弊端：只能指定一个允许源（不能用通配符合指定多个源），所以目前真实项目中基于cors实现跨域资源是主流方案**

**方案三：webpack代理（webpack proxy）**

1. **安装webpack-dev-server**
2. **配置代理**

**Proxy:{**

**‘/api’:{**

**Target:’https:/other-server.example.com’,**

**changetOrigin:true,**

**secure:false,**

**pathRewrite:{‘^api’:’’}**

**},**

**‘/getInfo’:{**

**Target:’https:/other-server.example.com’,**

**changetOrigin:true,**

**secure:false,**

**pathRewrite:{‘^api’:’’}**

**}**

**}**

**Package.json文件中**

"proxy": "http://localhost:8000"  
"proxy": {  
    "/course/banner": {  
      "target": "http://localhost:8000",  
      "changeOrigin": true  
    }  
  }

在create-react-app脚手架中，我们只需要在package.json中设置porxy代理属性，属性值是目标服务器的地址：在客户端发送请求的时候，请求的接口先保证是同源，如果当前源下没有这个接口，没关系，基于dev-server使用webscoket已经把目标代理服务器上的对应接口数据获取到了，所以直接请求同源下没有的接口，也可以获取数据

**Package.json**

"proxy": "http://localhost:8000"

方案四：node作为中间件代理

方案五：基于iframe实现跨域

Iframe可以实现父页面嵌入子页面（父页面中可以基于js获取子页面中的内容）

1. Window.name
   1. Name是window天生自带的属性，而且有一个特点，同源下，早x页面中设置name的值，页面关掉或者刷新，上次设置的值不消失，能够一直存储最后一次修改的值信息
2. Document,domain
   1. 只能处理主域相同，但是子域不同的情况