**1**、浏览器的同源安全策略

没错，就是这家伙干的，浏览器只允许请求当前域的资源，而对其他域的资源表示不信任。那怎么才算跨域呢？

### 2、CORS出来搞事了

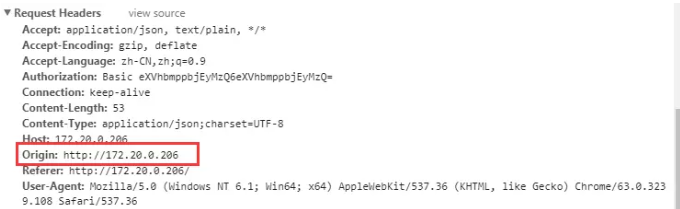
这是W3C的大佬们搞出来的标准，全称是"跨域资源共享"（Cross-origin resource sharing）。其实呢，这个大部分还是后端人员的工作。我们先来看看整个流程下，都发生了什么

1.简单请求：  
1): 请求方式只能是：head，get，post  
2): 请求头允许的字段：Accept，Accept-Language，Content-Language，Last-Event-ID  
Content-Type：application/x-www-form-urlencoded、multipart/form-data、text/plain 三选一

2.复杂请求：没错，不满足上面的，都是我啦！

**简单请求:**

浏览器：诶，你小子要跨域是吧，我得问问服务器大哥肯不肯！往请求头添加origin亮一下牌面;有个奇怪现象，谷歌游览器在非跨域情况下，也会发送origin字段



请求头origin字段为当前域

服务器：诶，你是谁，我来看看你的origin，嗯嗯，可以，符合我的要求，放行！顺便告诉你，老夫的规矩

  
其中，最重要的就是Access-Control-Allow-Origin，标识允许哪个域的请求。当然，如果服务器不通过，根本没有这个字段，接着触发XHR的onerror，再接着你就看到浏览器的提示xxx的服务器没有响应Access-Control-Allow-Origin字段

//指定允许其他域名访问

'Access-Control-Allow-Origin:http://172.20.0.206'//一般用法（\*，指定域，动态设置），3是因为\*不允许携带认证头和cookies

//是否允许后续请求携带认证信息（cookies）,该值只能是true,否则不返回

'Access-Control-Allow-Credentials:true'

上面第一行说到的Access-Control-Allow-Origin有多种设置方法：

1. 设置\*是最简单粗暴的，但是服务器出于安全考虑，肯定不会这么干，而且，如果是\*的话，游览器将不会发送cookies，即使你的XHR设置了withCredentials
2. 指定域，如上图中的http://172.20.0.206，一般的系统中间都有一个nginx，所以推荐这种
3. 动态设置为请求域，多人协作时，多个前端对接一个后台，这样很方便

withCredentials：表示XHR是否接收cookies和发送cookies，也就是说如果该值是false，响应头的Set-Cookie，浏览器也不会理，并且即使有目标站点的cookies，浏览器也不会发送。

**复杂请求:**

最常见的情况，当我们使用put和delete请求时，浏览器会先发送option（预检）请求，不过有时候，你会发现并没有，这是后面我们会讲到缓存。

**预检请求**

与简单请求不同的是，option请求多了2个字段：  
Access-Control-Request-Method：该次请求的请求方式  
Access-Control-Request-Headers：该次请求的自定义请求头字段

服务器检查通过后，做出响应：

//指定允许其他域名访问

'Access-Control-Allow-Origin:http://172.20.0.206'//一般用法（\*，指定域，动态设置），3是因为\*不允许携带认证头和cookies

//是否允许后续请求携带认证信息（cookies）,该值只能是true,否则不返回

'Access-Control-Allow-Credentials:true'

//预检结果缓存时间,也就是上面说到的缓存啦

'Access-Control-Max-Age: 1800'

//允许的请求类型

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

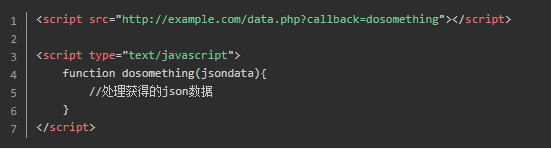
//允许的请求头字段

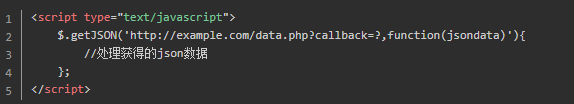
'Access-Control-Allow-Headers:x-requested-with,content-type'

这里有个注意点：Access-Control-Request-Method，Access-Control-Request-Headers返回的是满足服务器要求的所有请求方式，请求头，不限于该次请求，我一次性告诉你了，别TM问我了

**3、大家好，我是渣渣辉，是兄dei就来...呸呸呸，我是JSONP**

好啦，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，有种回调的味道！例子：





JSONP的优缺点  
优点：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。

缺点：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。

