一、为何要研究跨域问题

浏览器的同源策略规定某域下的客户端在没明确授权的情况下，不能读写另一个域的资源。

在实际开发中，前后端常常是相互分离的，并且前后端的项目部署也常常不在一个服务器内或者在一个服务器的不同端口下。前端想要获取后端的数据，就必须发起请求，如果不做一些处理，就会受到浏览器同源策略的约束。后端可以收到请求并返回数据，但是前端无法收到数据。

二、为何浏览器会指定同源策略

之所以有同源策略，其中一个重要原因就是对cookie的保护。cookie中存着sessionID。黑客一旦获取了sessionID，并且在有效期内，就可以登录。当我们访问了一个恶意网站，如果没有同源策略，那么这个网站就能通过js访问document.cookie得到用户关于的各个网站的sessionID其中可能有银行网站 等等。通过已经建立好的session连接进行攻击，比如CSRF攻击。

这里需要服务端配合再举个例子，现在我扮演坏人，我通过一个iframe加载某宝的登录页面，等傻傻的用户登录我的网站的时候，我就把这个页面弹出，用户一看，阿里唉大公司，肯定安全，就屁颠屁颠的输入了密码，注意：如果没有同源策略，我这个恶意网站就能通过dom操作获取到用户输入的值，从而控制该账户所以同源策略是绝对必要的。

需要注意的是同源策略无法完全防御CSRF。

三、多种跨域方法

跨域可以大概分为两种目的：

* 前后端分离时，前端为了获取后端数据而跨域。
* 为不同域下的前端页面通信而跨域。

#### 1、为前后端分离而跨域

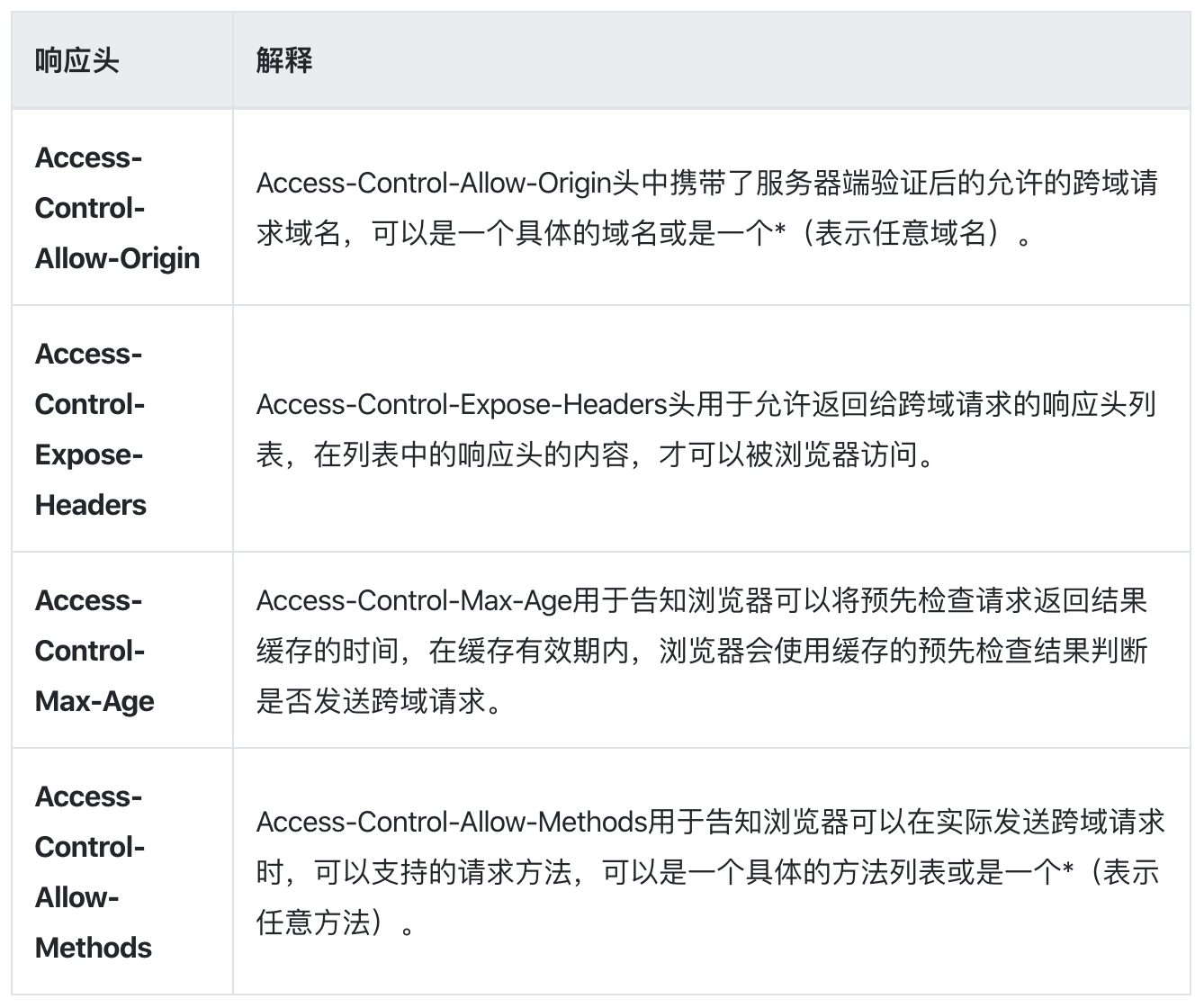
Cross Origin Resource Share（CORS）——跨域资源共享方案

为了解决跨域问题，通过增加一系列请求头和响应头，规范安全地进行跨站数据传输。

（1）请求头主要包括：

|  |  |
| --- | --- |
| 请求头 | 解释 |
| Origin | Origin头在跨域请求或预先请求中，标明发起跨域请求的源域名。 |
| Access-Control-Request-Method | Access-Control-Request-Method头用于表明跨域请求使用的实际HTTP方法 |
| Access-Control-Request-Headers | Access-Control-Request-Headers用于在预先请求时，告知服务器要发起的跨域请求中会携带请求头信息 |
| with-credentials | 跨域请求携带cookie |

（2）响应头主要包括：



（3）如何使用

* 客户端只需按规范设置请求头。
* 服务端按规范识别并返回对应响应头，或者安装相应的插件，修改相应框架配置文件等。具体视服务端所用的语言和框架而定。

SpringBoot设置CORS例子

@Configuration

public class CorsConfig {  
  
 private CorsConfiguration buildConfig() {  
 CorsConfiguration corsConfiguration = new CorsConfiguration();  
 // 你需要跨域的地址 注意这里的 127.0.0.1 != localhost  
 // \* 表示对所有的地址都可以访问  
 corsConfiguration.addAllowedOrigin("\*");  
 // 跨域的请求头  
 corsConfiguration.addAllowedHeader("\*"); // 2  
 // 跨域的请求方法  
 corsConfiguration.addAllowedMethod("\*"); // 3  
 //加上了这一句，大致意思是可以携带 cookie  
 //最终的结果是可以 在跨域请求的时候获取同一个 session  
 corsConfiguration.setAllowCredentials(true);  
 return corsConfiguration;  
 }  
  
 @Bean  
 public CorsFilter corsFilter() {  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 //配置可以访问的地址  
 source.registerCorsConfiguration("/\*\*", buildConfig()); // 4  
 return new CorsFilter(source);  
 }  
}

JSONP跨域

jsonp的原理就是借助HTML中的<script>标签可以跨域引入资源。所以动态创建一个<srcipt>标签，src为目的接口 + get数据包 + 处理数据的函数名。后台收到GET请求后解析并返回函数名(数据)给前端，前端<script>标签动态执行处理函数

注意：JSONP既是利用了<script>，那么就只能支持GET请求。其他请求都无法实现。

nginx反向代理实现跨域

思路：

既然浏览器有同源策略限制，那我们把前端项目和前端要请求的api接口地址放在同源下不就可以了？再结合web服务器提供的反向代理，便可以在前端和后端都不做配置的情况下解决跨域问题。

以nginx为例

* 后端真实后台地址：http://xxx.xxx.xxx.xxx:8001后台地址使用tomcat部署的spring boot项目名为mall。
* nginx服务器地址：http://xxx.xxx.xxx.xxx:8010
* tomcat和nginx都是用docker架设的，做了端口转发。
* 使用条件：开发环境为linux系统
* nginx/etc/nginx/conf.d/defaultconf配置代码如下：

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log /var/log/nginx/host.access.log main;

location / {

# root /usr/share/nginx/html/dist; # 前端项目路径

# index index.html index.htm;

proxy\_pass http://localhost:8001/; # 前端本机地址，实现自动更新

autoindex on;

autoindex\_exact\_size on;

autoindex\_localtime on;

}

location /gsms\_test/ {

proxy\_pass 后端真实地址;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80

#

#location ~ \.php$ {

# proxy\_pass http://127.0.0.1;

#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

#

#location ~ \.php$ {

# root html;

# fastcgi\_pass 127.0.0.1:9000;

# fastcgi\_index index.php;

# fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

# include fastcgi\_params;

#}

# deny access to .htaccess files, if Apache's document root

# concurs with nginx's one

#

#location ~ /\.ht {

# deny all;

#}

}

#### 2、不同域下页面通信而跨域

window.name + iframe 跨域

window.name是浏览器中一个窗口所共享的数据，在不同的页面（甚至不同域名）加载后依旧存在（如果没修改则值不会变化），并且可以支持非常长的 name 值（2MB）。比如 a域的某页面想获取b域某页面的数据，可以在b域中修改window.name值，a域切换到b域再切回来即可得到b域的window.name值。可是我们在开发中肯定不想页面切来切去，所以就要结合iframe来实现。

注意：由于受同源策略限制，父页面获取跨域的iframe页面的信息不全，所以要在iframe的windows.name被B域修改后，转为A域下的任一页面（该一面不得修改windows.name），在进行获取。

代理页面+iframe实现跨域访问

由于iframe与父页面相互访问也受同源策略限制，所以要借助一代理页面实现跨域。



四、总结

除以上几种跨域方法，还有postMessage，WebSocket等跨域方法由于从未接触不做说明。在项目中具体使用那些方法还需具体考虑各种问题。

|  |  |
| --- | --- |
| 情况 | 方法 |
| 只有GET请求 | JSONP |
| 对兼容性及浏览器版本无要求 | CORS |
| 对兼容性及浏览器版本有要求 | iframe或服务器反向代理（Linux环境下开发） |