[**Springboot2.0解决跨域问题**](https://segmentfault.com/a/1190000019550329)

**问题背景**

Same Origin Policy，译为“同源策略”。它是对于客户端脚本（尤其是JavaScript）的重要安全度量标准，其目的在于防止某个文档或者脚本从多个不同“origin”（源）装载。  
它认为自任何站点装载的信赖内容是不安全的。当被浏览器半信半疑的脚本运行在沙箱时，它们应该只被允许访问来自同一站点的资源，而不是那些来自其它站点可能怀有恶意的资源。  
**注：具有相同的Origin，也即是拥有相同的协议、主机地址以及端口。一旦这三项数据中有一项不同，那么该资源就将被认为是从不同的Origin得来的，进而不被允许访问。**

**URL** 说明 是否允许通信

http:*//www.domain.com/a.js*

http:*//www.domain.com/b.js 同一域名，不同文件或路径 允许*

http:*//www.domain.com/lab/c.js*

http:*//www.domain.com:8000/a.js*

http:*//www.domain.com/b.js 同一域名，不同端口 不允许*

http:*//www.domain.com/a.js*

https:*//www.domain.com/b.js 同一域名，不同协议 不允许*

http:*//www.domain.com/a.js*

http:*//192.168.4.12/b.js 域名和域名对应相同ip 不允许*

http:*//www.domain.com/a.js*

http:*//x.domain.com/b.js 主域相同，子域不同 不允许*

http:*//domain.com/c.js*

http:*//www.domain1.com/a.js*

http:*//www.domain2.com/b.js 不同域名 不允许*

### **跨域解决方案**

1、 通过jsonp跨域(jsonp缺点：只能实现get一种请求)  
2、 document.domain + iframe跨域  
3、 location.hash + iframe  
4、 window.name + iframe跨域  
5、 postMessage跨域  
6、 跨域资源共享（CORS）  
7、 nginx代理跨域  
8、 nodejs中间件代理跨域  
9、 WebSocket协议跨域

**CORS简介:**

CORS是一个W3C标准，全称是"跨域资源共享"（Cross-origin resource sharing）。它允许浏览器向跨源(协议 + 域名 + 端口)服务器，发出XMLHttpRequest请求，从而克服了AJAX只能同源使用的限制。  
CORS需要浏览器和服务器同时支持。它的通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS通信与同源的AJAX/Fetch通信没有差别，代码完全一样。浏览器一旦发现请求跨源，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。  
因此，实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信。

**浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）。**

浏览器发出CORS简单请求，只需要在头信息之中增加一个Origin字段。

浏览器发出CORS非简单请求，会在正式通信之前，增加一次OPTIONS查询请求，称为"预检"请求（preflight）。浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。

简单请求就是<font color=#9400D3>HEAD、GET、POST</font>请求，并且HTTP的头信息不超出以下几种字段  
<font color=#9400D3>Accept、Accept-Language、Content-Language、Last-Event-ID、Content-Type</font>  
**注：Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain**

反之，就是非简单请求。

其实实现CORS很简单，就是在服务端加一些响应头，并且这样做对前端来说是无感知的，很方便。

**详解响应头**

Access-Control-Allow-Origin  
该字段必填。它的值要么是请求时Origin字段的具体值，要么是一个\*，表示接受任意域名的请求。

Access-Control-Allow-Methods  
该字段必填。它的值是逗号分隔的一个具体的字符串或者\*，表明服务器支持的所有跨域请求的方法。注意，返回的是所有支持的方法，而不单是浏览器请求的那个方法。这是为了避免多次"预检"请求。

Access-Control-Expose-Headers  
该字段可选。CORS请求时，XMLHttpRequest对象的getResponseHeader()方法只能拿到6个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想拿到其他字段，就必须在Access-Control-Expose-Headers里面指定。

Access-Control-Allow-Credentials  
该字段可选。它的值是一个布尔值，表示是否允许发送Cookie.默认情况下，不发生Cookie，即：false。对服务器有特殊要求的请求，比如请求方法是PUT或DELETE，或者Content-Type字段的类型是application/json，这个值只能设为true。如果服务器不要浏览器发送Cookie，删除该字段即可。

Access-Control-Max-Age  
该字段可选，用来指定本次预检请求的有效期，单位为秒。在有效期间，不用发出另一条预检请求。

**顺便提一下，如果在开发中，发现每次发起请求都是两条，一次OPTIONS，一次正常请求，注意是每次，那么就需要配置Access-Control-Max-Age，避免每次都发出预检请求。**

**解决办法**

**第一种办法:**

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.web.servlet.config.annotation.CorsRegistry;

**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

**@Configuration**

**public** **class** **CorsConfig** **implements** **WebMvcConfigurer** {

**@Override**

**public** **void** **addCorsMappings**(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS")

.allowCredentials(**true**)

.maxAge(3600)

.allowedHeaders("\*");

}

}

这种方式是全局配置的，网上也大都是这种解决办法，但是很多都是基于旧的spring版本，比如：springboot中**WebMvcConfigurerAdapter,但是**在spring5.0已经被标记为Deprecated，点开源码可以看到：

*/\*\**

*\* An implementation of {@link WebMvcConfigurer} with empty methods allowing*

*\* subclasses to override only the methods they're interested in.*

*\**

*\* @author Rossen Stoyanchev*

*\* @since 3.1*

*\* @deprecated as of 5.0 {@link WebMvcConfigurer} has default methods (made*

*\* possible by a Java 8 baseline) and can be implemented directly without the*

*\* need for this adapter*

*\*/*

**@Deprecated**

**public** **abstract** **class** **WebMvcConfigurerAdapter** **implements** **WebMvcConfigurer** {}

像这种过时的类或者方法，spring的作者们一定会在注解上面说明原因，并告诉你新的该用哪个，这是非常优秀的编码习惯，点赞！

spring5最低支持到jdk1.8，所以注释中明确表明，你可以直接实现WebMvcConfigurer接口，无需再用这个适配器，因为jdk1.8支持接口中存在default-method。

**第二种办法:**

**import** org.springframework.context.annotation.Configuration;

**import** javax.servlet.\*;

**import** javax.servlet.annotation.WebFilter;

**import** javax.servlet.http.HttpServletResponse;

**import** java.io.IOException;

**@WebFilter**(filterName = "CorsFilter ")

**@Configuration**

**public** **class** **CorsFilter** **implements** **Filter** {

**@Override**

**public** **void** **doFilter**(ServletRequest req, ServletResponse res, FilterChain chain) **throws** IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) res;

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

response.setHeader("Access-Control-Allow-Credentials", "true");

response.setHeader("Access-Control-Allow-Methods", "POST, GET, PATCH, DELETE, PUT");

response.setHeader("Access-Control-Max-Age", "3600");

response.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

chain.doFilter(req, res);

}

}

这种办法，是基于过滤器的方式，方式简单明了，就是在response中写入这些响应头，好多文章都是第一种和第二种方式都叫你配置，其实这是没有必要的，只需要一种即可。

这里也吐槽一下，大家不求甚解的精神。

**第三种办法：**

**public** **class** **GoodsController** {

**@CrossOrigin**(origins = "http://localhost:4000")

**@GetMapping**("goods-url")

**public** Response **queryGoodsWithGoodsUrl**(@RequestParam String goodsUrl) **throws** Exception {}

}

没错就是**@CrossOrigin**注解，点开注解

**@Target**({ ElementType.METHOD, ElementType.TYPE })

**@Retention**(RetentionPolicy.RUNTIME)

**@Documented**

**public** **@interface** CrossOrigin {}

从元注解@Target可以看出，注解可以放在method、class等上面，类似RequestMapping，也就是说，整个controller下面的方法可以都受控制，也可以单个方法受控制。

也可以得知，这个是最小粒度的cors控制办法了，精确到单个请求级别。

以上三种方法都可以解决问题，最常用的应该是第一种、第二种，控制在自家几个域名范围下足以，一般没必要搞得太细。

这三种配置方式都用了的话，谁生效呢，类似css中样式，就近原则，懂了吧。

所以在开发新项目时，不需要等联调时候，让前端来找你了，我早就解决了跨域问题。