**Java 跨域问题**

原文：<https://www.jianshu.com/p/3be176f9e3a0>

**前言**

在日常的前后端项目分离的项目开发中，通常会遇到资源请求跨域的问题。本文将介绍跨域出现的原因，以及在Springboot中的解决方式。

**什么是跨域**

跨域是指一个页面想获取另一个页面中的资源，如果这两个页面的协议、域名、子域名、端口不同，或者两个页面一个为IP地址另一个为域名地址，这种情况下所进行的访问行动都是跨域的。而出于安全性的考虑，浏览器通常会限制跨域访问，不允许跨域请求资源。

**CORS**

跨域资源共享（Cross Origin Resource Share）是一种允许一个网页上的JavaScript向另一个域中发出AJAX请求机制。 默认情况下浏览器中禁止此类Web请求，并且这些请求会触发相同的同源安全策略错误。而使用java的CORS过滤器就可以实现网页向其他域发送请求。  
CORS机制的工作原理通过添加一些特定的HTTP头部标识，使得浏览器被告知下载的资源应当被允许给定的域（或者所有的域）进行Web请求。除此之外还可以通过添加信息来通知浏览器仅允许通过这些域的链接上的某些HTTP方法，例如GET / PUT / POST / DELETE等。  
CORS预检请求是一个用来检查另一个域是否理解CORS协议的CORS请求。它是一个使用两个可选的HTTP请求头部标识的请求：分别是Access-Control-Request-Method请求头部标识和Access-Control-Request-Headers请求头部标识。

**Response Headers参数**

Access-Control-Allow-Origin ：指定授权域来进行跨域请求，如果此项没有限制，使用\*作为参数值。  
Access-Control-Allow-Credentials ：指定跨域请求是否需要验证。  
Access-Control-Expose-Headers ：指示哪一种头部标识可以被安全的暴露（公开）。  
Access-Control-Max-Age ：指定预检请求的最大缓存时间。  
Access-Control-Allow-Methods ：指定进行跨域请求时使用的方法。  
Access-Control-Allow-Headers ：指定实际的请求过程中哪些头部标识可以使用。

**Request Headers**

Origin ：指定跨域请求或者预检请求的域的来源。  
Access-Control-Request-Method ：在发出预检请求时使用，目的使服务器知道在实际请求中将使用何种HTTP方法。  
Access-Control-Request-Headers ：在发出预检请求时使用，目的使服务器知道在实际请求中将使用何种头部标识。

**Springboot 通过 Filter 实现跨域处理**

@SpringBootApplication

public class DemoApplication {

/\*跨域处理\*/

@Bean

public CorsFilter corsFilter() {

final UrlBasedCorsConfigurationSource urlBasedCorsConfigurationSource = new UrlBasedCorsConfigurationSource();

final CorsConfiguration corsConfiguration = new CorsConfiguration();

corsConfiguration.setAllowCredentials(true);

corsConfiguration.addAllowedOrigin("\*");

corsConfiguration.addAllowedHeader("\*");

corsConfiguration.addAllowedMethod("\*");

urlBasedCorsConfigurationSource.registerCorsConfiguration("/\*\*", corsConfiguration);

return new CorsFilter(urlBasedCorsConfigurationSource);

}

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

在Springboot中的启动类中，添加以上代码，解决跨域处理的问题。

**Springboot 通过 WebMvcConfigurer 实现跨域处理**

先上代码

@Configuration

public class CorsConfig {

@Bean

public WebMvcConfigurer corsConfigurer(){

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

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

}

};

}

}

首先，进入WebMvcConfigurer的源码，部分如下：

public interface WebMvcConfigurer {

...

default void addCorsMappings(CorsRegistry registry) {

}

...

}

可以看到WebMvcConfigurer类是一个接口类，并且在接口类中定义了一个名为addCorsMappings()的方法。而这个方法的最前面有一个"default"关键字进行修饰，''default"关键字是Java8中加入的新特性，相当于提供了一个默认的实现，接口类中被该关键字修饰的方法若没有被其实现类实现，则采用接口类中的默认实现；若实现类中对该方法进行重写，则默认的实现将会被覆盖。这一设计也符合了Java的多态性。  
继续进入CorsRegistry 源码中：

public class CorsRegistry {

private final List<CorsRegistration> registrations = new ArrayList();

public CorsRegistry() {

}

public CorsRegistration addMapping(String pathPattern) {

CorsRegistration registration = new CorsRegistration(pathPattern);

this.registrations.add(registration);

return registration;

}

protected Map<String, CorsConfiguration> getCorsConfigurations() {

Map<String, CorsConfiguration> configs = new LinkedHashMap(this.registrations.size());

Iterator var2 = this.registrations.iterator();

while(var2.hasNext()) {

CorsRegistration registration = (CorsRegistration)var2.next();

configs.put(registration.getPathPattern(), registration.getCorsConfiguration());

}

return configs;

}

}

CorsRegistry中声明了两个方法 ，CorsRegistration 类型的addMapping()方法和Map<String, CorsConfiguration> 类型的getCorsConfigurations()方法。addMapping()方法用来对接收一条Url,getCorsConfigurations()方法来获取配置的相关参数。  
继续进入CorsRegistration 类中，源码如下：

public class CorsRegistration {

private final String pathPattern;

private final CorsConfiguration config;

public CorsRegistration(String pathPattern) {

this.pathPattern = pathPattern;

this.config = (new CorsConfiguration()).applyPermitDefaultValues();

}

public CorsRegistration allowedOrigins(String... origins) {

this.config.setAllowedOrigins(Arrays.asList(origins));

return this;

}

public CorsRegistration allowedMethods(String... methods) {

this.config.setAllowedMethods(Arrays.asList(methods));

return this;

}

public CorsRegistration allowedHeaders(String... headers) {

this.config.setAllowedHeaders(Arrays.asList(headers));

return this;

}

public CorsRegistration exposedHeaders(String... headers) {

this.config.setExposedHeaders(Arrays.asList(headers));

return this;

}

public CorsRegistration allowCredentials(boolean allowCredentials) {

this.config.setAllowCredentials(allowCredentials);

return this;

}

public CorsRegistration maxAge(long maxAge) {

this.config.setMaxAge(maxAge);

return this;

}

protected String getPathPattern() {

return this.pathPattern;

}

protected CorsConfiguration getCorsConfiguration() {

return this.config;

}

}

CorsRegistration 类中定义了众多的方法，根据方法名可以直接的推断出方法的具体作用，每个方法的具体作用在上文中已有提到。类中定义了一个CorsConfiguration 类型的成员变量config，继续深究CorsConfiguration 的源码，可以发现在CorsConfiguration 类中进行了具体的跨域处理，包括设置默认允许的HTTP请求方法等操作，代码量较大，不在此贴出。