链滴

SPRING CLOUD GATEWAY 集成 SWAGGER 方案总结 (请求路径中带有 ",")

作者：jockming112

原文链接：https://ld246.com/article/1627097790858

来源网站：链滴

转载：https://www.freesion.com/article/50961265470/

前言

在微服务大行其道到今天，服务到碎片化也带来了管理和监控的困难（统一集成网关系统在前面的sia-ateway的文章中有做分享，感兴趣的可以前往阅读）。

swagger为我们开发带来了极大到便利，但是在庞杂的系统中，即便对于开发的接入和调试，对各个散对在线文档，也显得有些杂乱无章。即便是有如同yapi等在线文档系统，也同样需要精力去维护，且对于文档与代码分离，同样会带来版本不一致和调试困难等问题。

在网关层集成swagger显得非常易用和优雅。但是面对各个公司不同网关配置方案，给swagger的集带来和诸多不便和技术难题。下面我分享我在实现spring cloud gateway 集成swagger的实现方案。
系列文章目录

提示：这里可以添加系列文章的所有文章的目录，目录需要自己手动添加

一、服务的SWAGGER配置

1.引入依赖包

在pom文件中添加如下依赖，推荐使用2.9.2版本

```
<!-- swagger 包-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
```

2.添加SWAGGER配置类

```
@Configuration
@EnableSwagger2
@ConditionalOnProperty(name = "swagger.enable",  havingValue = "true")
public class SwaggerConfig {

    @Bean
    public Docket openApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .groupName("open")
                .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.example.api.controller"))
                .paths(PathSelectors.any())
                .build();
```

```
    }

    @Bean
    public Docket adminApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .groupName("admin")
                .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.example.api.controller"))
                .paths(PathSelectors.any())
                .build();
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("微服务swagger")
                .contact(new Contact("LiuChao", "", "liuchao332@163.com"))
                .version("1.0")
                .description("API 描述")
                .build();
    }

}
```

说明： 1、此配置类为基本的swagger集成的基本配置，有些区别的是，此处配置类2个Docket元素分别设置了不同的groupName。此操作是为了满足，将业务端和管理端分成不同的swagger页面进展示。这个地方会为网关集成带来一些改变。不是必须要这样设置，不想将接口分开，放到同一个页也没什么不可。 2、在配置项中需要新增swagger.enable配置，来控制是否启用swagger。# 二、SPING CLOUD GATEWAY 集成SWAGGER

1.SPRING CLOUD GATEWAY搭建

spring cloud gateway 的集成非常简单，基本上实现来了开箱即用，进行简单的配置，集成eureka就能满足我们基本的业务场景，这里就不在赘述了。

2.在网关中引入SWAGGER包

和微服务所应用的包相同

3.添加配置类

网关的swagger集成方案与服务单元有较大不同，也更为复杂。

```
@Configuration
public class SwaggerConfig {

    @Bean
    public SecurityConfiguration securityConfiguration() {
        return SecurityConfigurationBuilder.builder().build();
    }

    @Bean
```

```
    public UiConfiguration uiConfiguration() {
        return UiConfigurationBuilder.builder().showExtensions(true).build();
    }

}
```

此配置类中定义SecurityConfiguration，UiConfiguration 将在下面的配置中用到这2个bean

```
@RestController
@RequestMapping("/swagger-resources")
public class SwaggerHandler {

    private final SecurityConfiguration securityConfiguration;
    private final UiConfiguration uiConfiguration;
    private final SwaggerResourcesProvider swaggerResources;

    @Autowired
    public SwaggerHandler(SwaggerResourcesProvider swaggerResources, SecurityConfigurati
n securityConfiguration, UiConfiguration uiConfiguration) {
        this.swaggerResources = swaggerResources;
        this.securityConfiguration = securityConfiguration;
        this.uiConfiguration = uiConfiguration;
    }

    @GetMapping("/configuration/security")
    public Mono<ResponseEntity<SecurityConfiguration>> securityConfiguration() {
        return Mono.just(new ResponseEntity<>(
                Optional.ofNullable(securityConfiguration).orElse(SecurityConfigurationBuilder.buil
er().build()),
                HttpStatus.OK));
    }

    @GetMapping("/configuration/ui")
    public Mono<ResponseEntity<UiConfiguration>> uiConfiguration() {
        return Mono.just(new ResponseEntity<>(
                Optional.ofNullable(uiConfiguration).orElse(UiConfigurationBuilder.builder().build()),
HttpStatus.OK));
    }

    @GetMapping
    public Mono<ResponseEntity<List<SwaggerResource>>> swaggerResources() {
        return Mono.just((new ResponseEntity<>(swaggerResources.get(), HttpStatus.OK)));
    }

}
```

此类定义了swagger网关层的开放接口，在访问swagger-ui中会拉去此接口的数据。

```
@Primary
@Component
@AllArgsConstructor
public class SwaggerProvider implements SwaggerResourcesProvider {

    public static final String SOURCE_URI = "http://%s/swagger-resources";
```

```
    private final RouteLocator routeLocator;
    private final RouteDefinitionLocator routeDefinitionLocator;
    private final RestTemplate restTemplate;

    @Override
    public List<SwaggerResource> get() {
        List<SwaggerResource> resources = new ArrayList<>();
        List<String> routes = new ArrayList<>();
        routeLocator.getRoutes().subscribe(route -> routes.add(route.getId()));
        routeDefinitionLocator.getRouteDefinitions()
                .filter(routeDefinition -> routes.contains(routeDefinition.getId()))
                .subscribe(routeDefinition -> routeDefinition.getPredicates().stream()
                        .filter(predicateDefinition -> ("Path").equalsIgnoreCase(predicateDefinition.ge
tName()))
                        .forEach(predicateDefinition -> resources
                                .addAll(swaggerResource(routeDefinition))));

        return resources;
    }

    private List<SwaggerResource> swaggerResource(RouteDefinition route) {

        try {
            String sourceUrl = String.format(SOURCE_URI, route.getUri().getHost());
            ResponseEntity<String> content = restTemplate.getForEntity(sourceUrl, String.class);
            List<SwaggerResource> swaggerResources = JsonUtil.toList(content.getBody(), Swag
erResource.class);

            swaggerResources.stream().forEach(swaggerResource -> {
                swaggerResource.setName(route.getUri().getHost().toLowerCase() + "-" + swagger
esource.getName());
                swaggerResource.setUrl("/" + route.getUri().getHost().toLowerCase() + swaggerRes
urce.getUrl());
            });

            return swaggerResources;
        } catch (Exception e) {
            return new ArrayList<>();
        }
    }
}
```

这个类是核心，这个类封装的是SwaggerResource，即在swagger-ui.html页面中顶部的选择框，选服务的swagger页面内容。

RouteLocator：获取spring cloud gateway中注册的路由

RouteDefinitionLocator：获取spring cloud gateway路由的详细信息

RestTemplate：获取各个配置有swagger的服务的swagger-resources

三、私货

在一般的情况下，网关一般使用一个服务，一个路由的配置方式，我也建议大家这么使用，这样管理制都非常灵活。但在项目中，也遇到过一种特别的配置方式。如下：

```
routes:
  - id: remove_api_prefix
    uri: http://127.0.0.1:80
    order: 1
    predicates:
      - Path=/api/**
    filters:
      - StripPrefix=1
```

按照这种配置方式，在请求url前加上/api前缀，网关在收到此类请求后，将api截取掉，在转发给自；网关再次收到没有api的请求后，对请求第二级目录按服务名进行路由。

这种方式可以将在注册中心注册对服务都暴露到网关到路由中，就不需要对每个服务配置路由规则。

对于新加的服务，也不行要配置新的路由。

但是这样也引起了新的问题，在swagger的baseUrl，出现了,符号

www.example-gateway.com/api,/example-service/

且通过swagger发送的请求，在url上都出现了,。导致不能够请求到正确到地址上，然后就开始分析个,的来源。

```
curl -X POST  “http://www.example-gateway.com/api,/example-service/user/info”  -H  “accept: / ”
```

1.分析一

初步可以判断，这个符号肯定和这种特殊的路由方式有关系，因为在使用一服务一路由的配置方式的候，并没有这个问题。从哪里入手呢，还是从服务的swagger页面入手，因为这个符号就来自这里。

如上截图，在api-docs接口中所返回的数据里，basePath就已经带上里这个符号。有了一个初步的位思路，从swagger的api-docs接口入手。

2.分析二

在找到swagger接口源头，往下debug，终于在XForwardPrefixPathAdjuster类中发现类这个符号现的地方。源码如下图，

```
public class XForwardPrefixPathAdjuster implements PathAdjuster {
    static final String X_FORWARDED_PREFIX = "X-Forwarded-Prefix";
    private final HttpServletRequest request;

    public XForwardPrefixPathAdjuster(HttpServletRequest request) {
        this.request = request;
    }

    public String adjustedPath(String path) {
        String prefix = this.request.getHeader("X-Forwarded-Prefix");
        if (prefix != null) {
            return !SpringVersionCapability.supportsXForwardPrefixHeader(SpringVersion.getVersion()) ? prefix + path : prefix;
        } else {
            return path;
```

```
        }
    }
}
```

本以为发现源头，但是在debug的过程中，到这里接收到http头的X-Forwarded-Prefix参数，已经有这个符号。

由此可以判断，在swagger接收到到http请求就已经带有这个符号。所以只能往上找原因。那就只有个地方了，问题出在网关。

3.分析三

然后我就开始spring-cloud-gateway-core-2.2.2.RELEASE.jar包里，找什么地方在封装header的X-Frwarded-Prefix参数。有赖于spring合理的代码和包结构，没用多少时间就找到XForwardedHeadersilter类，如下所示：

```
private void updateRequest(HttpHeaders updated, URI originalUri, String originalUriPath, Strig requestUriPath) {
        if (requestUriPath != null && originalUriPath.endsWith(requestUriPath)) {
            String prefix = substringBeforeLast(originalUriPath, requestUriPath);
            if (prefix != null && prefix.length() > 0 && prefix.length() <= originalUri.getPath().lengh()) {
                this.write(updated, "X-Forwarded-Prefix", prefix, this.isPrefixAppend());
            }
        }

    }
```

在此方法，完成的header头的X-Forwarded-Prefix的封装。再往下看write方法

```
private void write(HttpHeaders headers, String name, String value, boolean append) {
        if (append) {
            headers.add(name, value);
            List<String> values = headers.get(name);
            String delimitedValue = StringUtils.collectionToCommaDelimitedString(values);
            headers.set(name, delimitedValue);
        } else {
            headers.set(name, value);
        }

    }
```

在这里，当网关在截取掉api，再发送给自己，这个地方就会在此请求中，List会将这个请求头中封装次的X-Forwarded-Prefix，/api和/example-service。StringUtils.collectionToCommaDelimitedStrng(values)方法将它们拼接到一起，就产生了,符号。如今要想办法消除掉这个符号。灵机一动，在写filer，将,干掉就行了。

4.解决

添加一个global filter，删除X-Forwarded-Prefix参数到,符号。完美解决。

```
@Component
public class XForwardedPrefixFilter implements HttpHeadersFilter, Ordered {
```

```
    @Override
    public HttpHeaders filter(HttpHeaders input, ServerWebExchange exchange) {
        List<String> xForwareds = input.get("X-Forwarded-Prefix");
        String forward = xForwareds.get(0);
        if (forward.contains(",")) {
            forward = forward.replaceAll(",", "/");
        }

        input.set("X-Forwarded-Prefix", forward);
        return input;
    }

    @Override
    public int getOrder() {
        return 1;
    }
}
```