# Spring Cloud网关

1. Zuul动态路由
2. 创建独立的网关微服务模块： myshop-gateway
3. 导入zuul和eureka的依赖（网关服务本身也需要注册到Eureka）

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-zuul</artifactId>  
 </dependency>

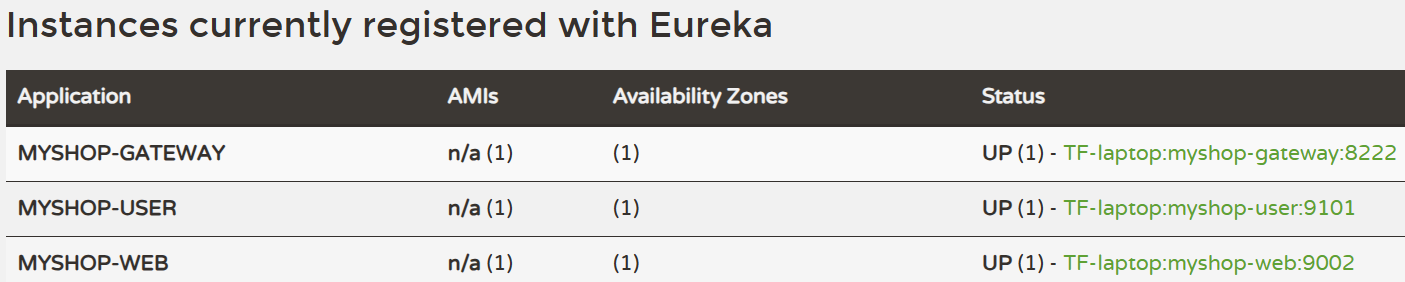
</dependencies>

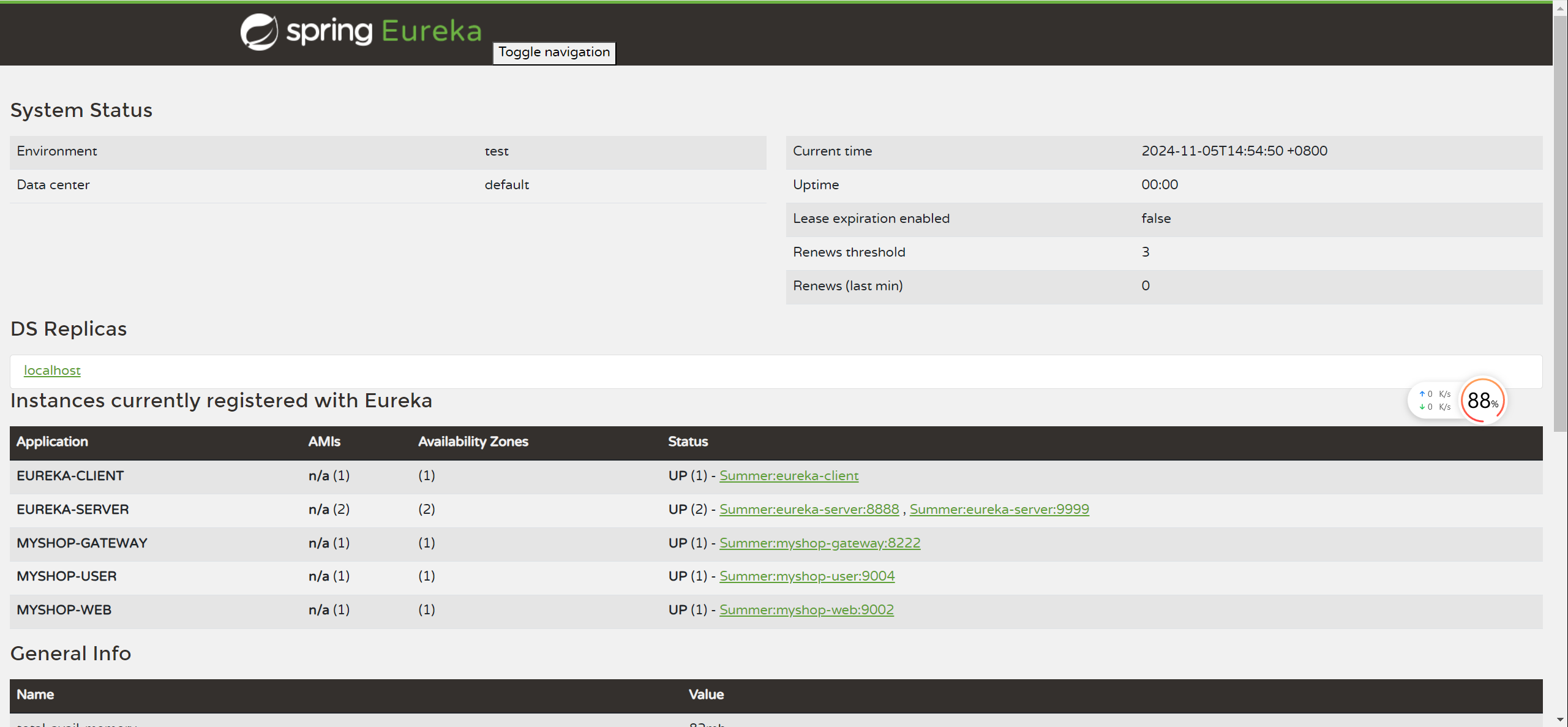
1. 启动类添加@EnableZuulProxy注解
2. 配置application.yml路由规则

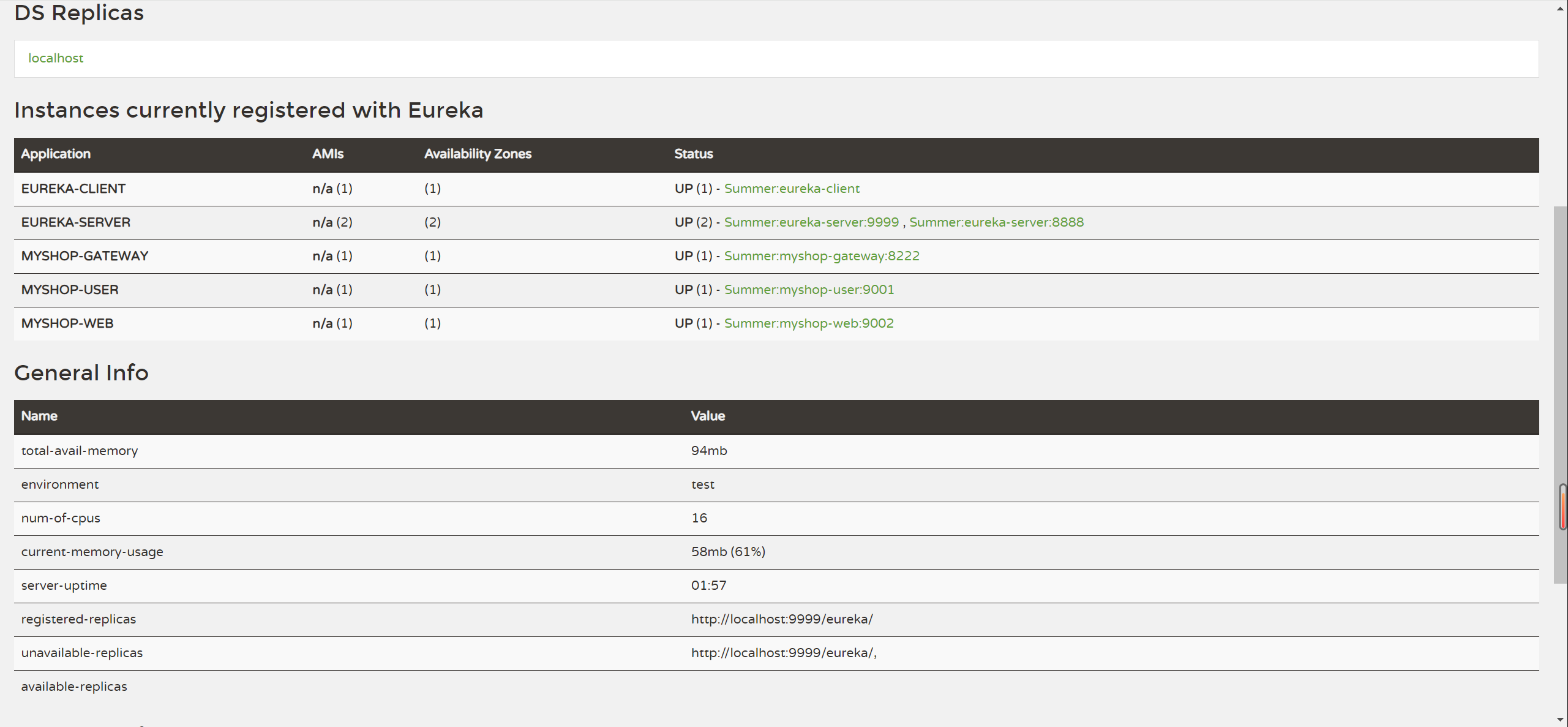
server:  
 port: 8222  
spring:  
 application:  
 name: myshop-gateway  
eureka:  
 client:  
 register-with-eureka: true  
 fetch-registry: true  
 service-url:  
 defaultZone: http://localhost:8888/eureka  
 instance:  
 prefer-ip-address: true  
#zuul的动态路由配置   
zuul:  
 routes:  
 myshop-user:  
 path: /myshop-user # 需要转发的路径  
 serviceId: myshop-user # 最终转发的微服务（名称）  
 myshop-web:  
 path: /myshop-web # 需要转发的路径  
 serviceId: myshop-web # 最终转发的微服务（名称）

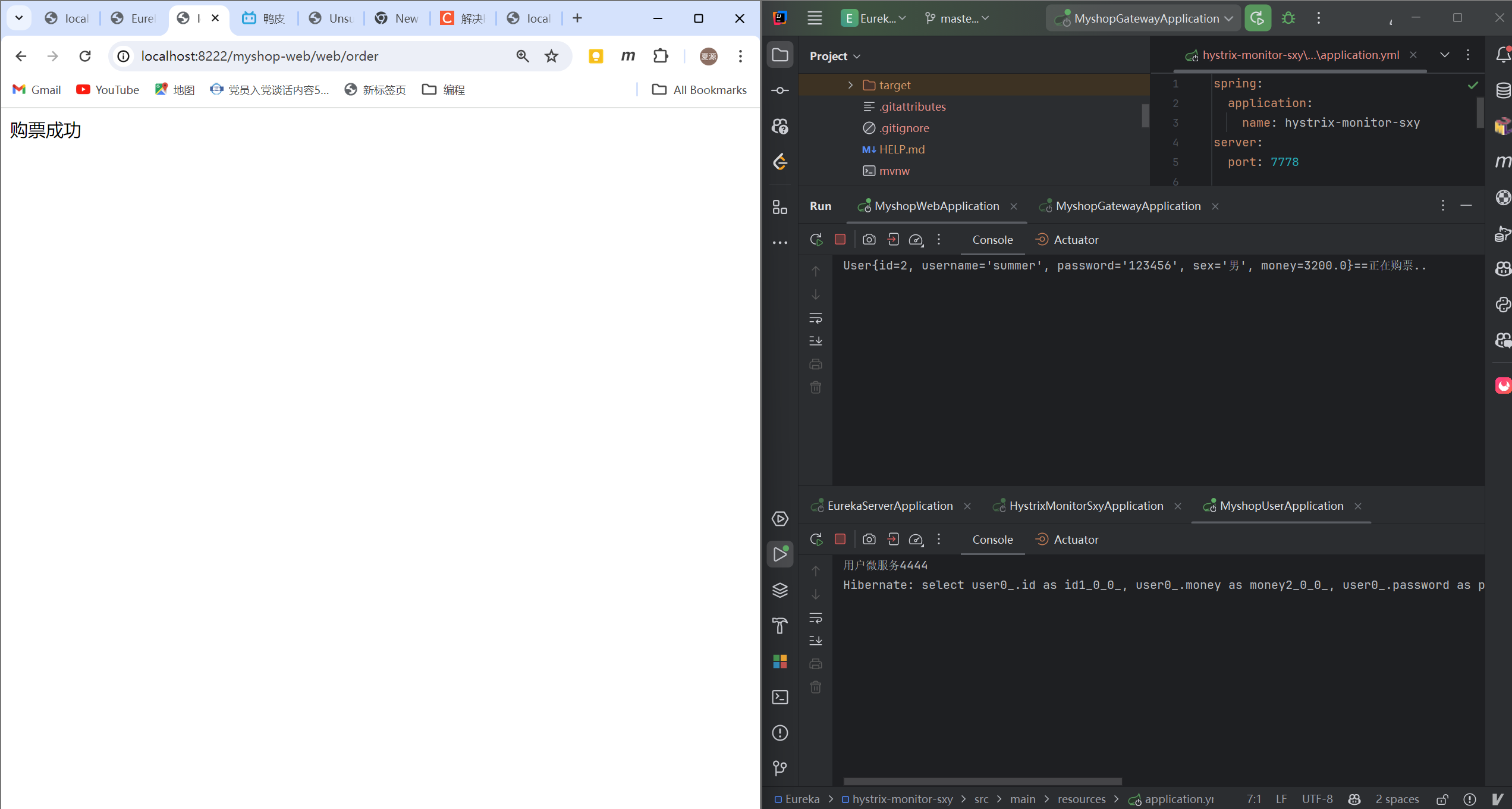
注：当转发的路径（path）和 转发的微服务的名称（serviceId）是一致的话，那么可以省略zuul路由配置

1. 启动各服务并测试网关路由转发是否成功（截图）





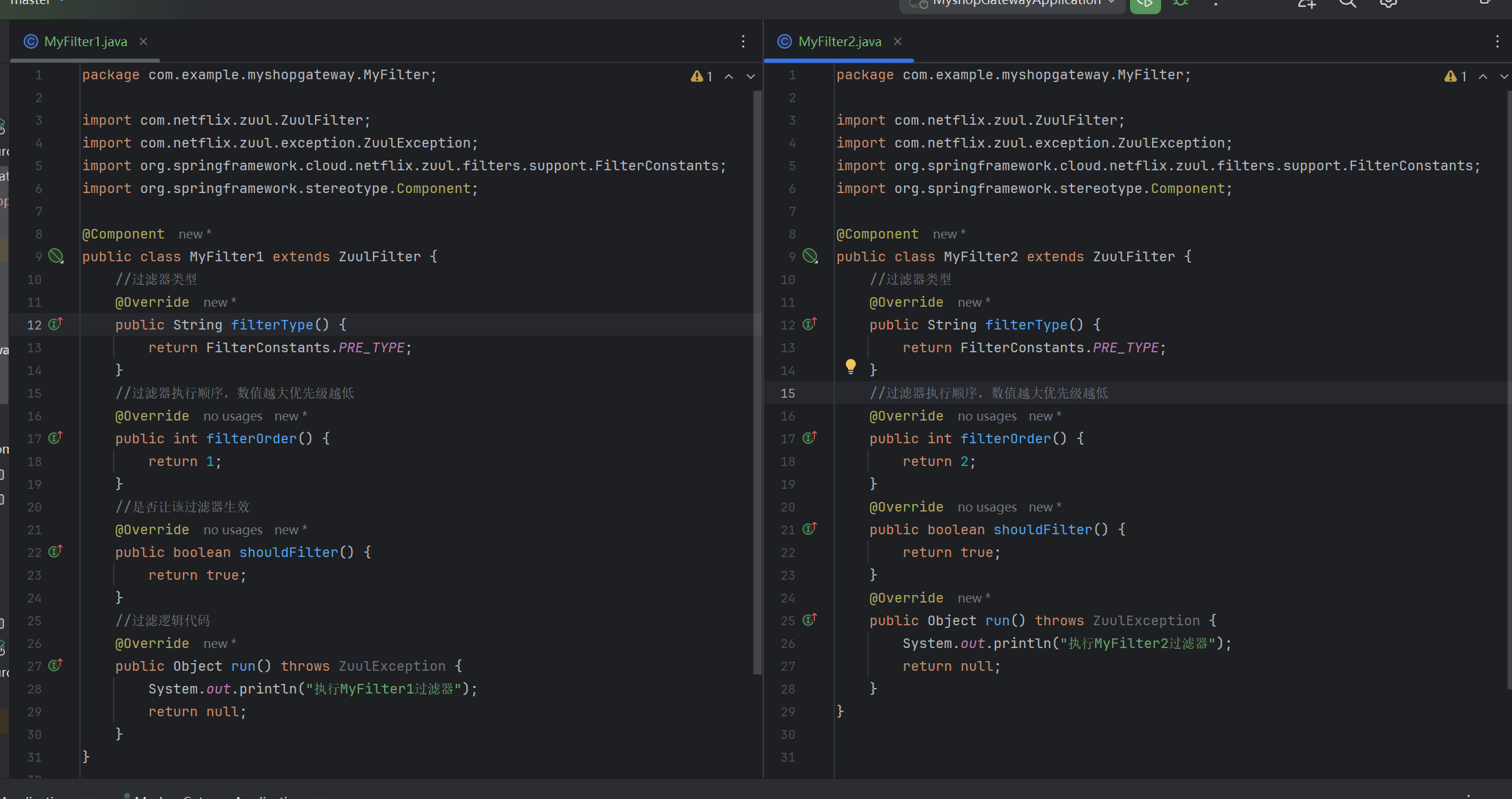
前面实验我们均采用localhost:9002/web/order直接请求微服务，今后我们将通过网关请求服务**localhost:8222/myshop-web/web/order**



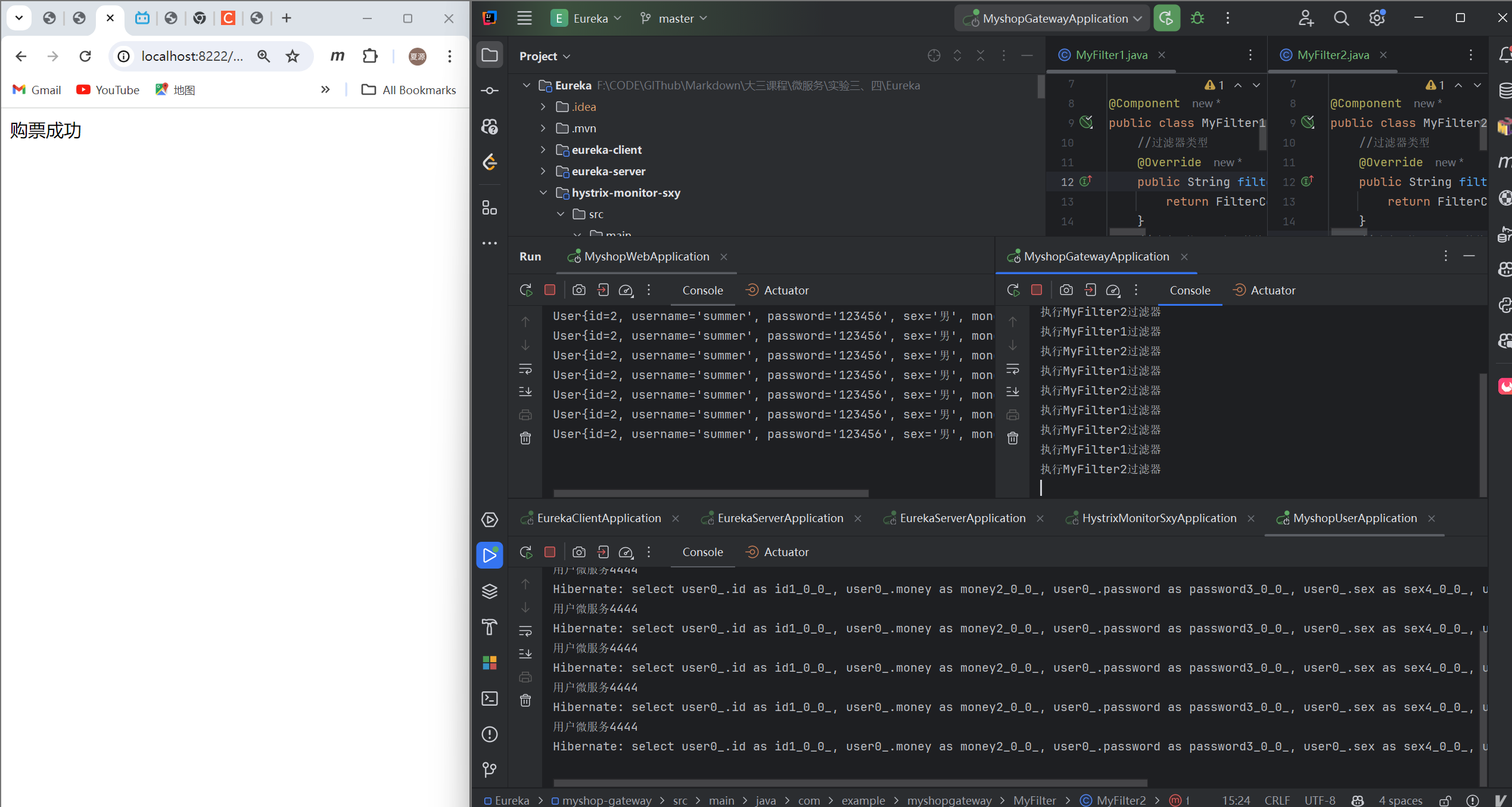
1. Zuul自定义过滤器的编写
2. 编写两个自定义过滤器MyFilter1和MyFilter2

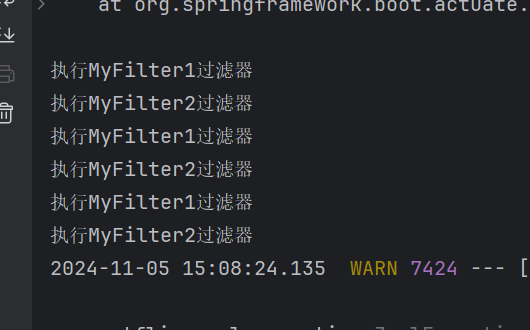
/\*\*  
 \* 第一个Zuul过滤器MyFilter1  
 \*/  
@Component  
public class MyFilter1 extends ZuulFilter {  
 //过滤器类型  
 @Override  
 public String filterType() {  
 return FilterConstants.PRE\_TYPE;  
 }  
 //过滤器执行顺序，数值越大优先级越低  
 @Override  
 public int filterOrder() {  
 return 1;  
 }  
 //是否让该过滤器生效  
 @Override  
 public boolean shouldFilter() {  
 return true;  
 }  
 //过滤逻辑代码  
 @Override  
 public Object run() throws ZuulException {  
 System.out.println("执行MyFilter1过滤器");  
 return null;  
 }  
}

/\*\*  
 \* 第二个Zuul过滤器MyFilter2  
 \*/  
@Component  
public class MyFilter2 extends ZuulFilter {  
 //过滤器类型  
 @Override  
 public String filterType() {  
 return FilterConstants.PRE\_TYPE;  
 }  
 //过滤器执行顺序，数值越大优先级越低  
 @Override  
 public int filterOrder() {  
 return 2;  
 }  
 @Override  
 public boolean shouldFilter() {  
 return true;  
 }  
 @Override  
 public Object run() throws ZuulException {  
 System.out.println("执行MyFilter2过滤器");  
 return null;  
 }  
}



1. 重启网关微服务，浏览器多次请求order方法（localhost:8222/myshop-web/web/order），记录IDEA控制台信息，观察自定义的过滤器是否起作用，执行顺序是怎样的？（截图+结合图内容说明执行顺序）
2. 交替执行





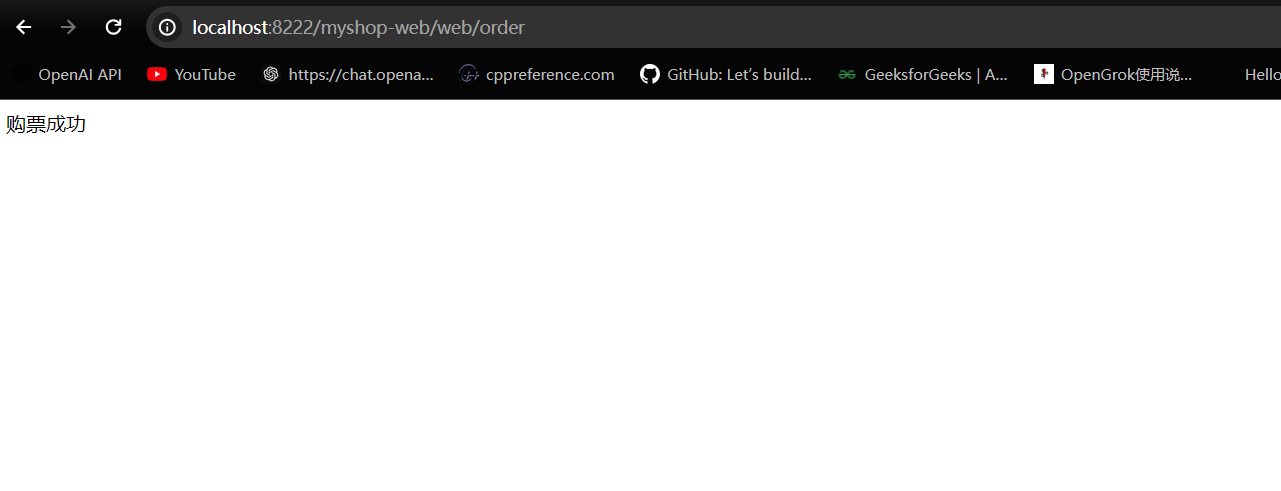
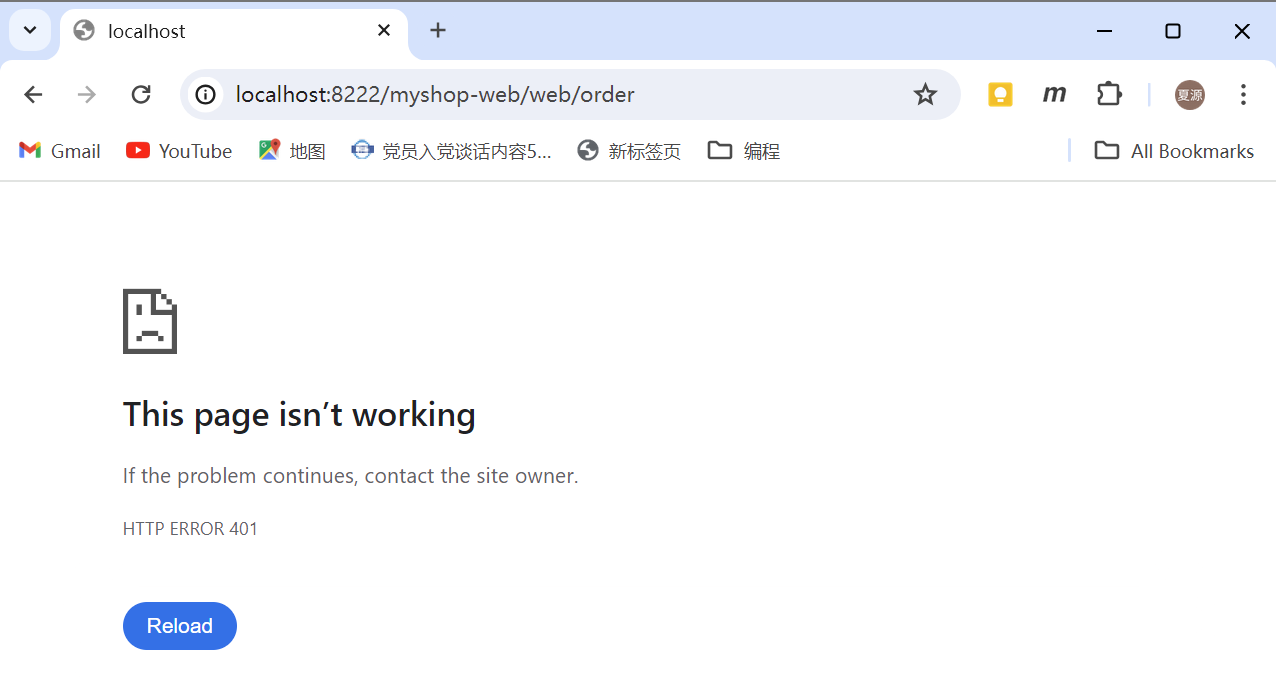
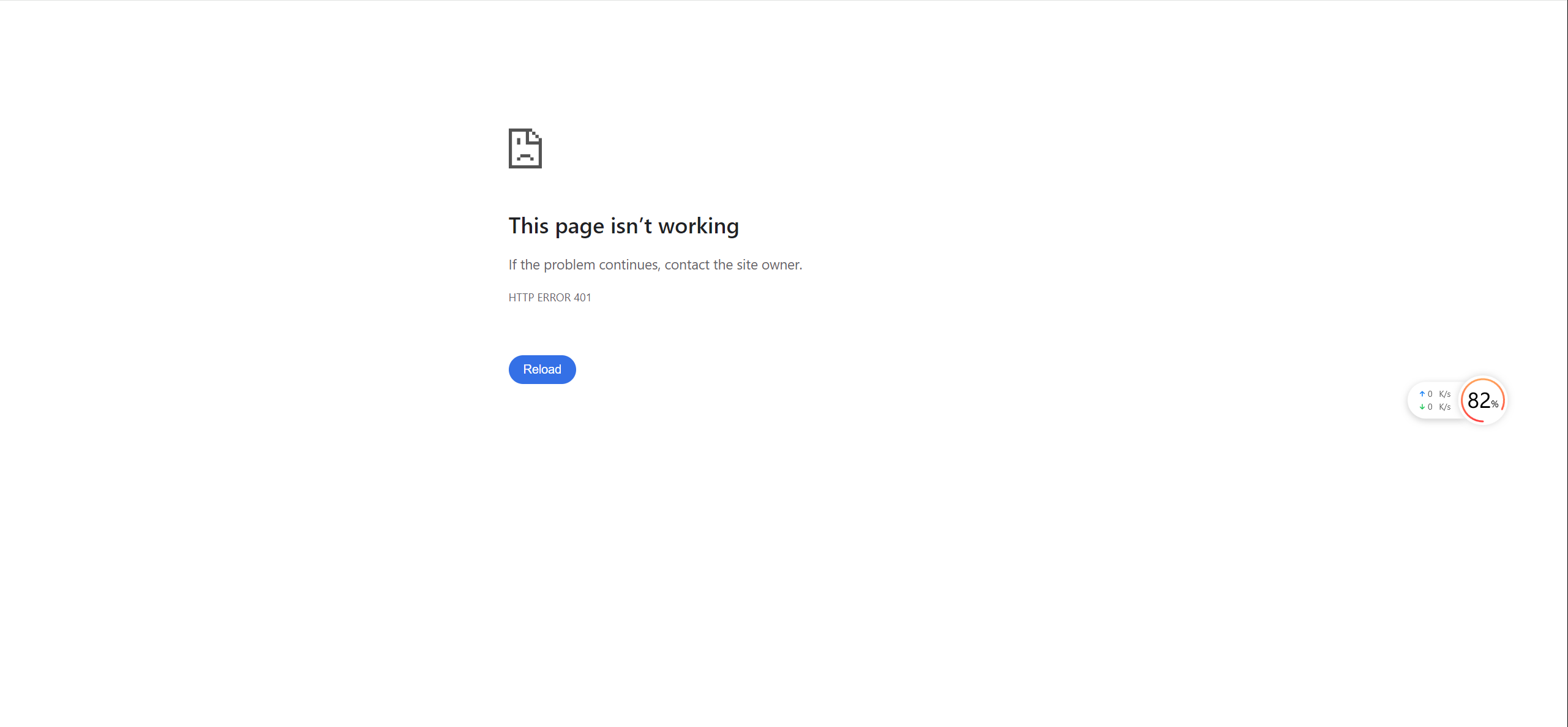
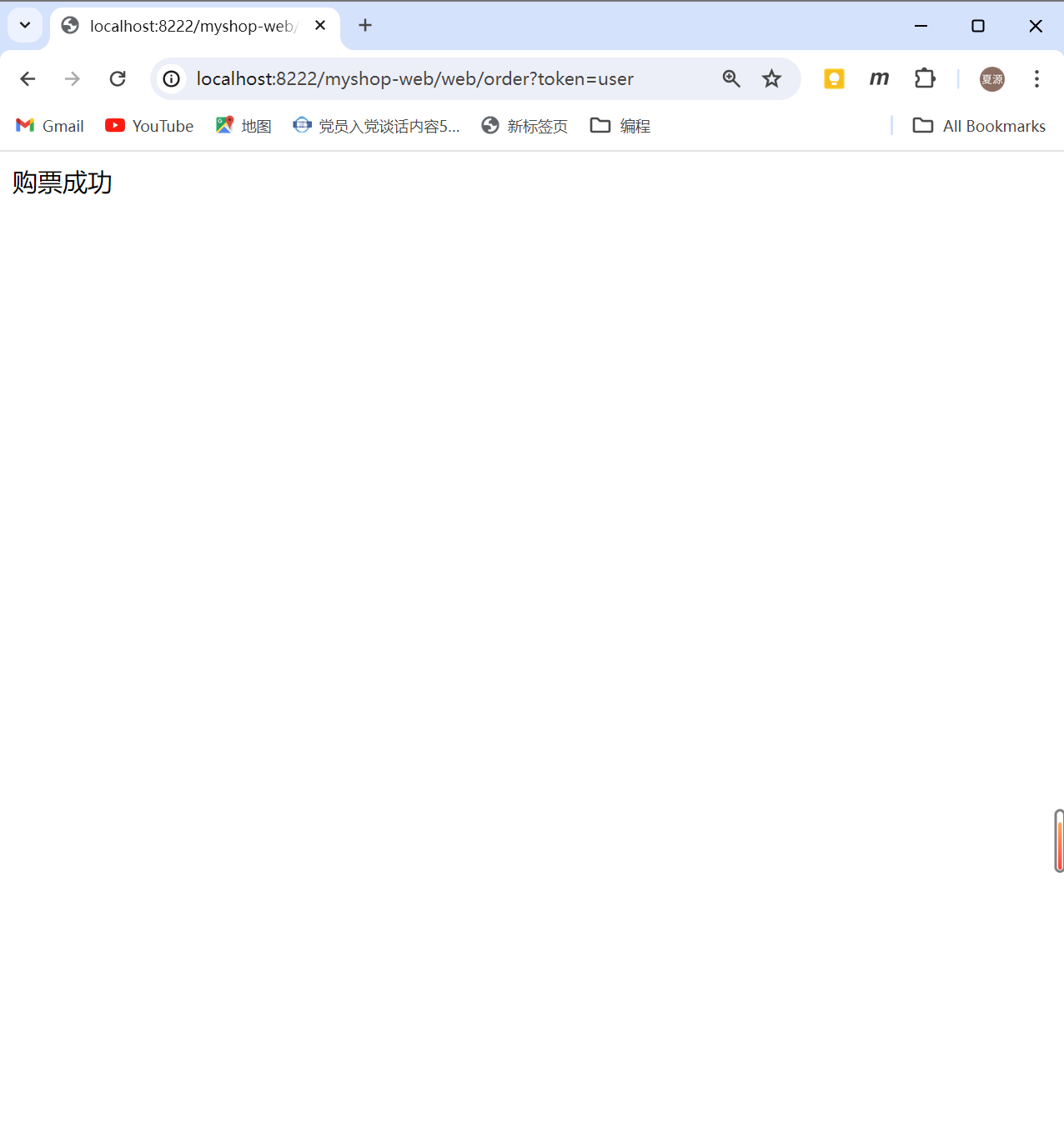
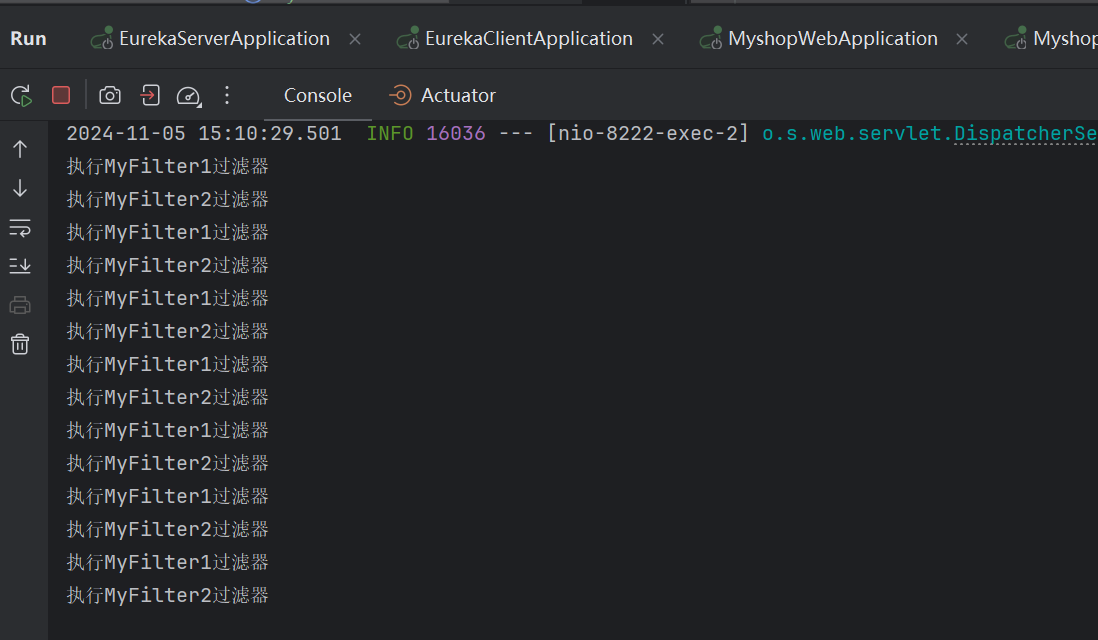
1. 自定义权限控制过滤器
2. 编写自定义权限控制过滤器AuthFilter

/\*\*\* 权限控制过滤器 \*/  
@Component  
public class AuthFilter extends ZuulFilter {  
 @Override  
 public String filterType() {  
 return FilterConstants.PRE\_TYPE;  
 }  
 @Override  
 public int filterOrder() {  
 return 0;  
 }  
 @Override  
 public boolean shouldFilter() {  
 return true;  
 }  
 @Override  
 public Object run() throws ZuulException {

//模拟执行异常  
 //int i = 10/0;

//1.获取当前请求的参数：token=user  
 RequestContext currentContext = RequestContext.getCurrentContext();  
 HttpServletRequest request = currentContext.getRequest();  
 HttpServletResponse response = currentContext.getResponse();  
 String token = request.getParameter("token");  
 if(!"user".equals(token)){  
 //不转发微服务，给用户响应  
 currentContext.setSendZuulResponse(false);  
 //设置401状态码  
 response.setStatus(401);  
 return null;  
 }  
 //继续转发  
 return null;  
 }  
}

1. 测试。

* 多次localhost:8222/myshop-web/web/order请求order方法，会出现什么返回结果？（截图）
* 
* 
* 
* 怎样请求order方法才能返回“购票成功”？（截图+结合图内容说明如何请求）
* 
* 

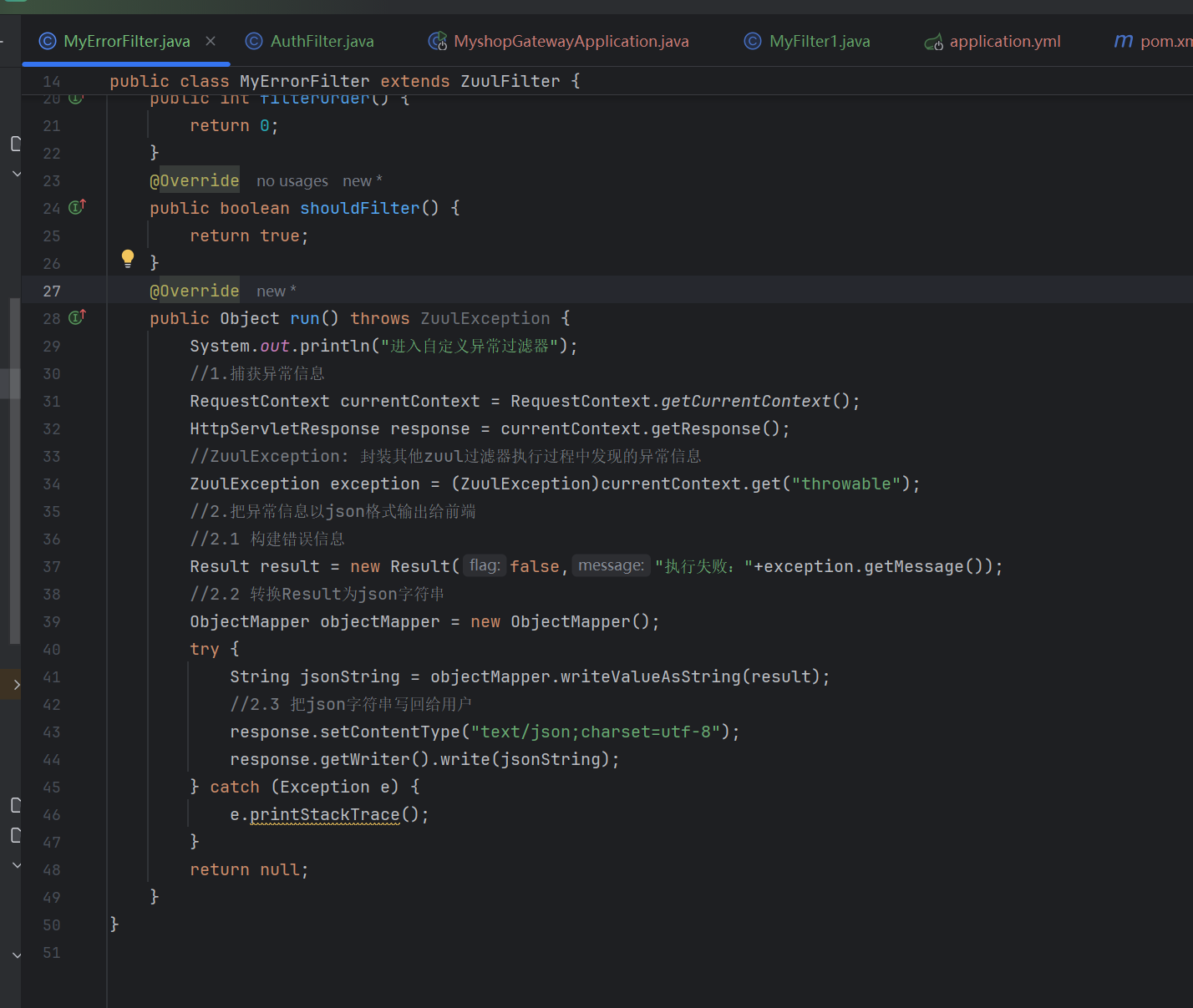
1. 自定义异常处理过滤器

需求：AuthFilter过滤器代码int i = 10/0模拟异常，需自定义一个异常处理过滤器，捕捉到此异常，并将异常信息以json格式返回给客户端。

1. 编写异常处理过滤器MyErrorFilter

/\*\*\* 自定义错误类型的Zuul过滤器 \*/  
@Component  
public class MyErrorFilter extends ZuulFilter {  
 @Override  
 public String filterType() {  
 return FilterConstants.ERROR\_TYPE;  
 }  
 @Override  
 public int filterOrder() {  
 return 0;  
 }  
 @Override  
 public boolean shouldFilter() {  
 return true;  
 }  
 @Override  
 public Object run() throws ZuulException {  
 System.out.println("进入自定义异常过滤器");  
 //1.捕获异常信息  
 RequestContext currentContext = RequestContext.getCurrentContext();  
 HttpServletResponse response = currentContext.getResponse();  
 //ZuulException: 封装其他zuul过滤器执行过程中发现的异常信息  
 ZuulException exception = (ZuulException)currentContext.get("throwable");  
 //2.把异常信息以json格式输出给前端  
 //2.1 构建错误信息  
 Result result = new Result(false,"执行失败："+exception.getMessage());  
 //2.2 转换Result为json字符串  
 ObjectMapper objectMapper = new ObjectMapper();  
 try {  
 String jsonString = objectMapper.writeValueAsString(result);  
 //2.3 把json字符串写回给用户  
 response.setContentType("text/json;charset=utf-8");  
 response.getWriter().write(jsonString);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
}

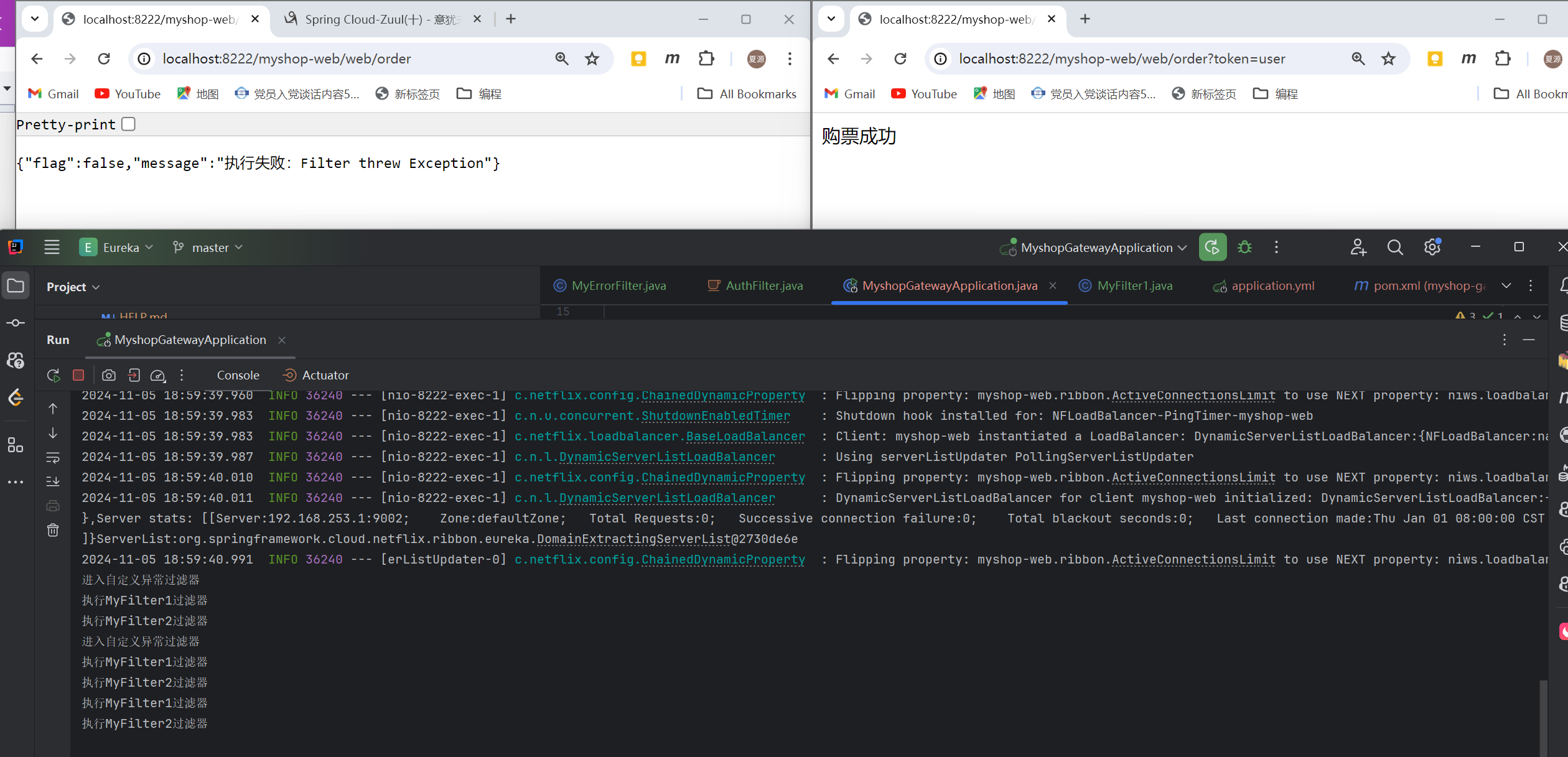
/\*\*\* 封装响应数据 \*/  
public class Result {  
 private Boolean flag;  
 private String message;  
 public Result() {  
 }  
 public Result(Boolean flag, String message) {  
 this.flag = flag;  
 this.message = message;  
 }  
 public Boolean getFlag() {  
 return flag;  
 }  
 public void setFlag(Boolean flag) {  
 this.flag = flag;  
 }  
 public String getMessage() {  
 return message;  
 }  
 public void setMessage(String message) {  
 this.message = message;  
 }  
}

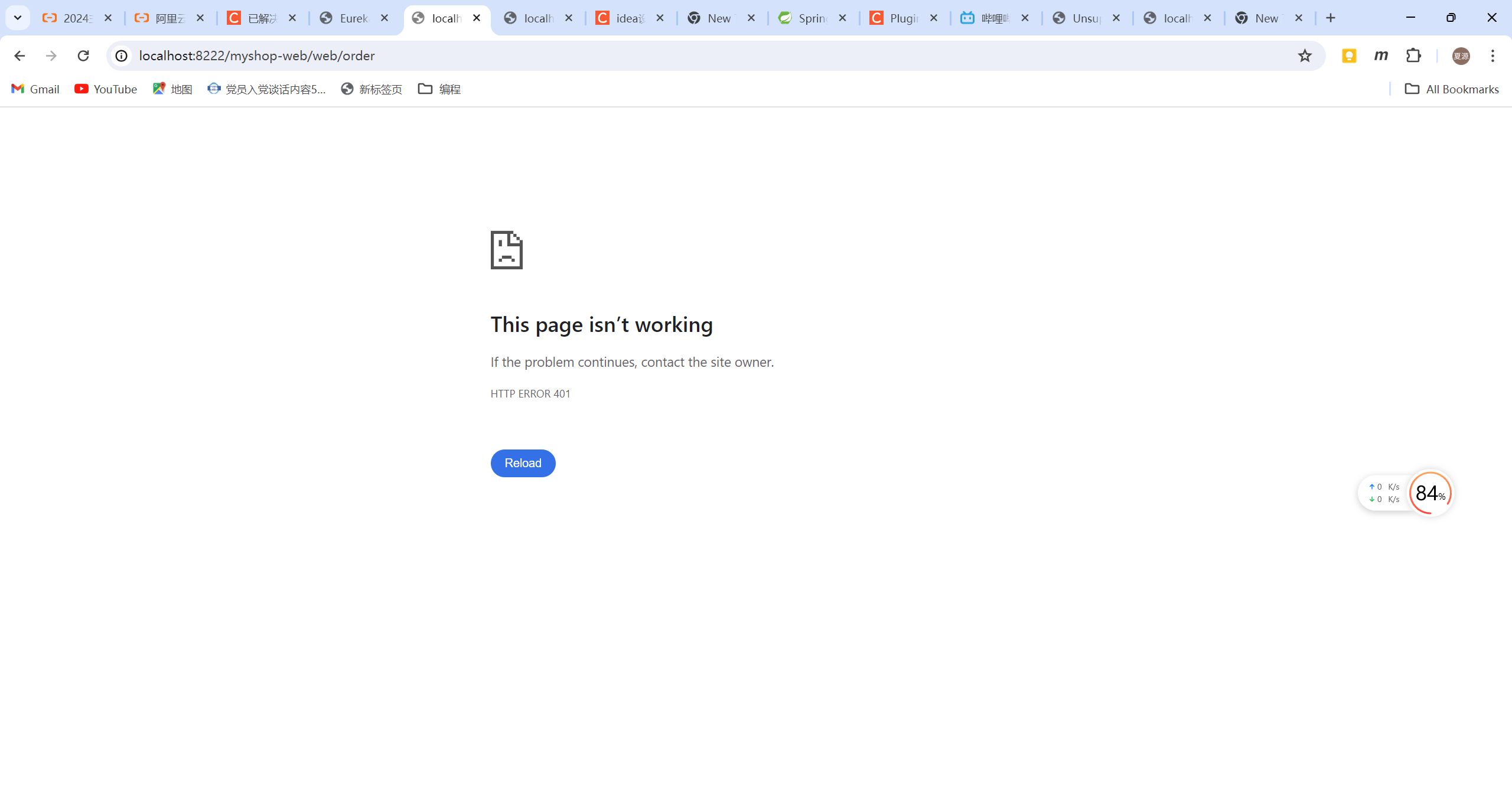


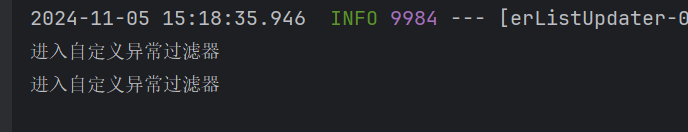
1. 屏蔽系统预定义的异常过滤器

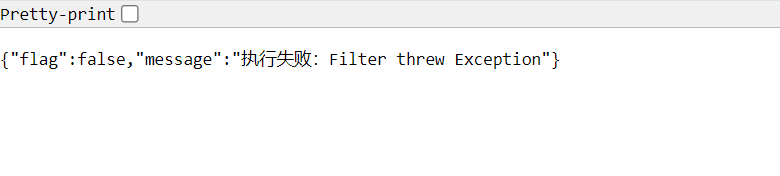
# 让zuul预定义的异常过滤器失效  
zuul:  
 SendErrorFilter:  
 error:  
 disable: true

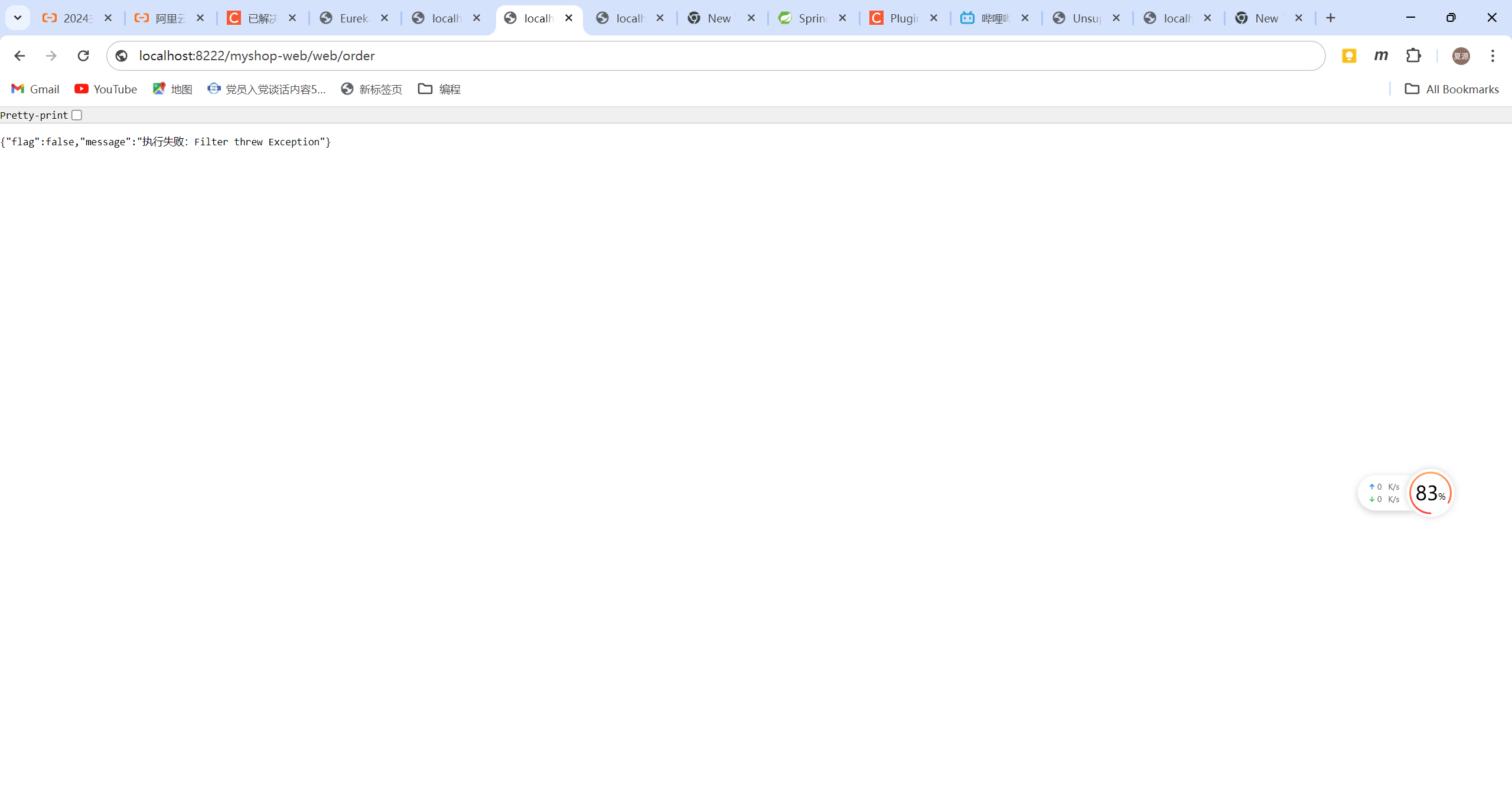
1. 测试。localhost:8222/myshop-web/web/order请求，自定义的MyErrorFilter过滤器是否生效？记录返回界面及IDEA控制台消息。（截图+说明）

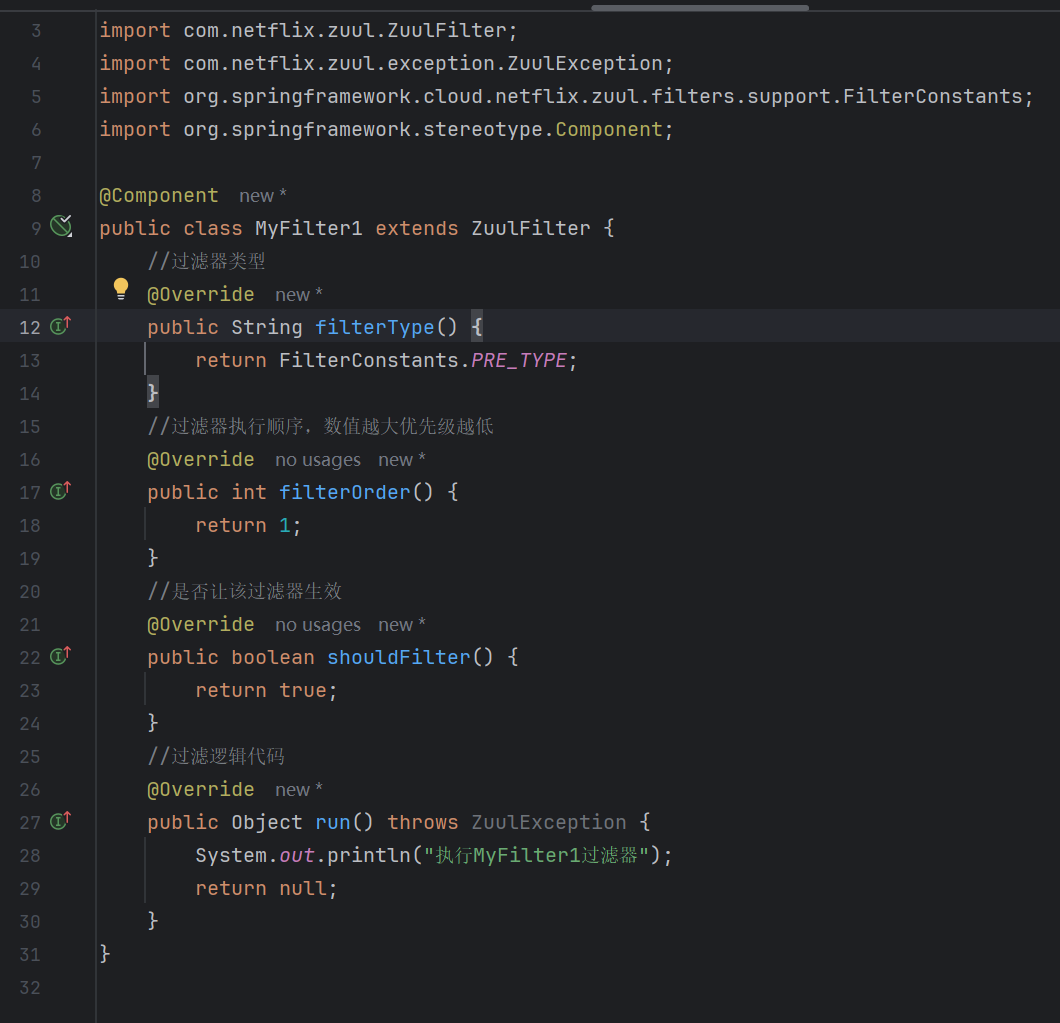












1. Swagger
2. 网关融合Swagger

1）导入swagger的依赖

<!-- swagger -->  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.8.0</version>  
</dependency>  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.8.0</version>  
</dependency>

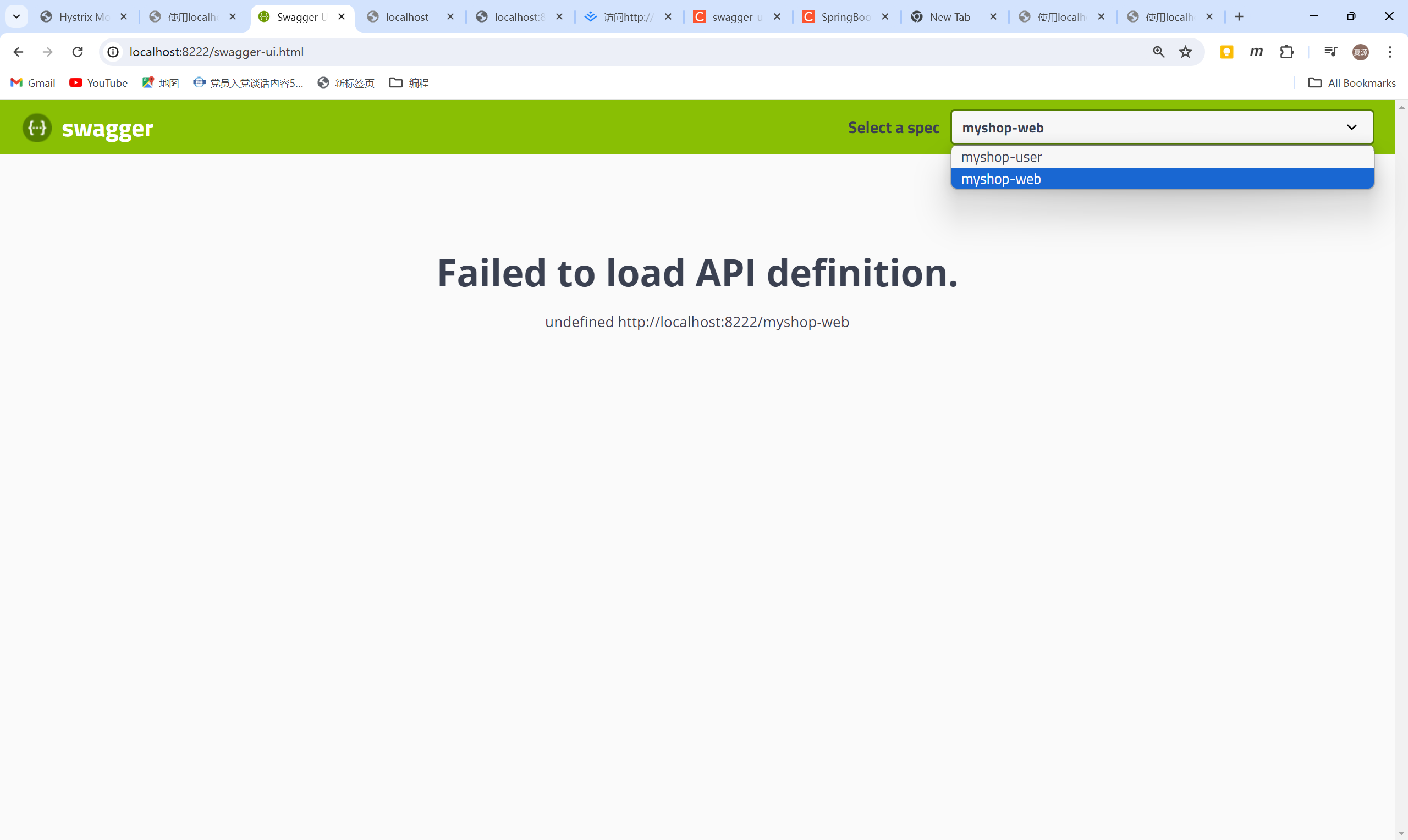
2）编写DocumentationConfig配置类

注：DocumentationConfig配置类需与引导类同一级目录

@Component  
@Primary  
public class DocumentationConfig implements SwaggerResourcesProvider {  
 private final RouteLocator routeLocator;  
 public DocumentationConfig(RouteLocator routeLocator) {  
 this.routeLocator = routeLocator;  
 }  
 @Override  
 public List<SwaggerResource> get() {  
 List<SwaggerResource> resources = new ArrayList<>();  
 List<Route> routes = routeLocator.getRoutes();  
 routes.forEach(route -> {  
 resources.add(swaggerResource(route.getId(), route.getFullPath().replace(  
 "\*\*", "v2/api-docs"), "1.0"));  
 });  
 return resources;  
 }  
 private SwaggerResource swaggerResource(String name, String location, String version) {  
 SwaggerResource swaggerResource = new SwaggerResource();  
 swaggerResource.setName(name);  
 swaggerResource.setLocation(location);  
 swaggerResource.setSwaggerVersion(version);  
 return swaggerResource;  
 }  
}

3）引导类开启swagger功能，添加@EnableSwagger2注解

4）访问网关的swagger主页localhost:8222/swagger-ui.html，记录主页内容，是否能查看微服务的API？（截图+说明）

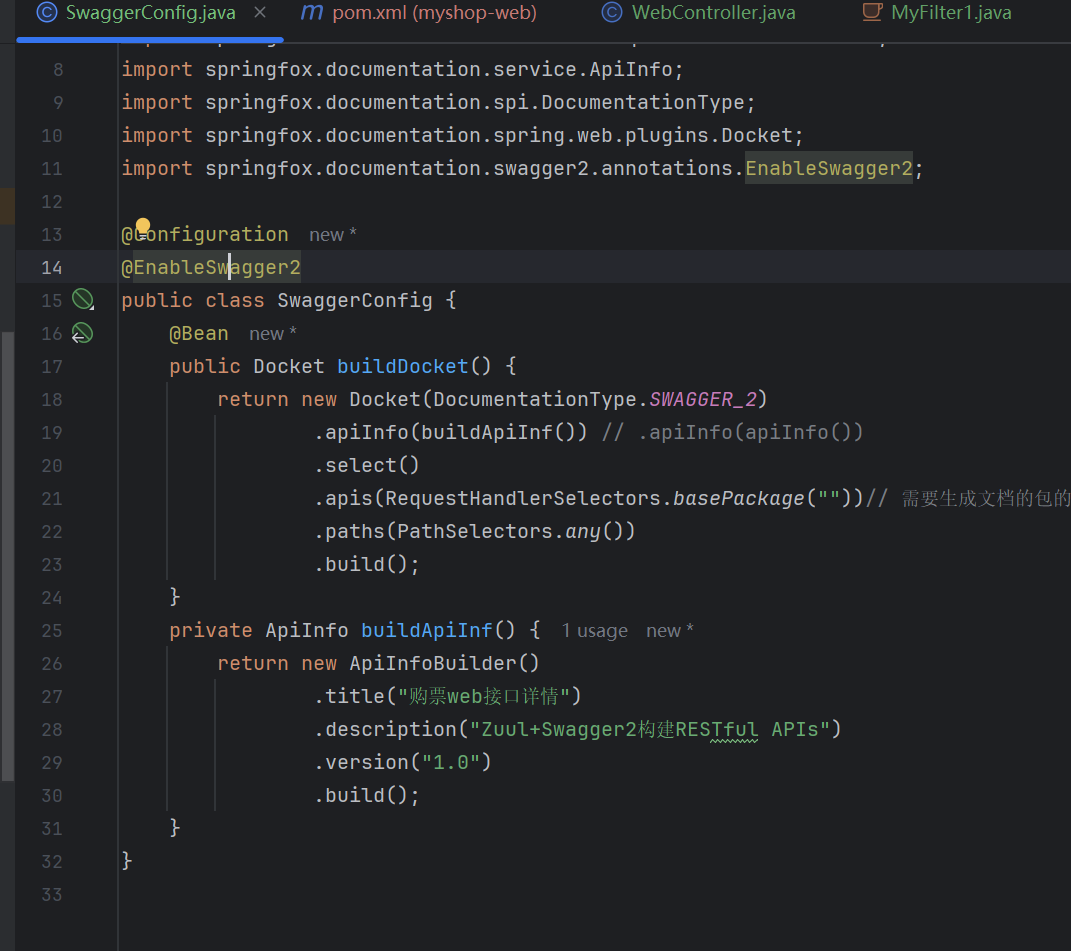


1. 微服务(myshop-web及myshop-user)暴露API（前面定义的过滤器都取消）
2. 导入swagger的依赖

<!-- swagger -->  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger2</artifactId>  
 <version>2.8.0</version>  
</dependency>  
<dependency>  
 <groupId>io.springfox</groupId>  
 <artifactId>springfox-swagger-ui</artifactId>  
 <version>2.8.0</version>  
</dependency>

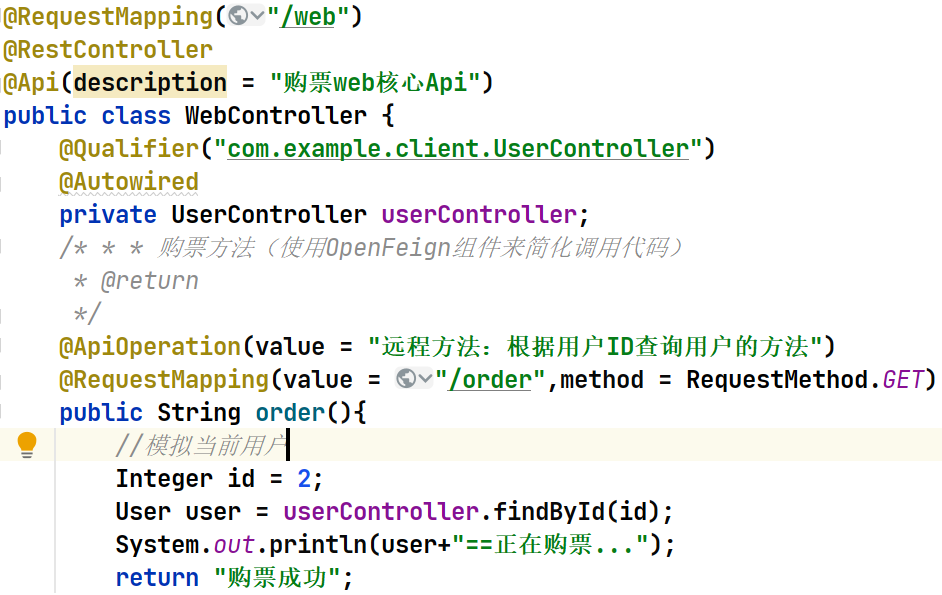
1. 编写购票微服务端的SwaggerConfig配置类

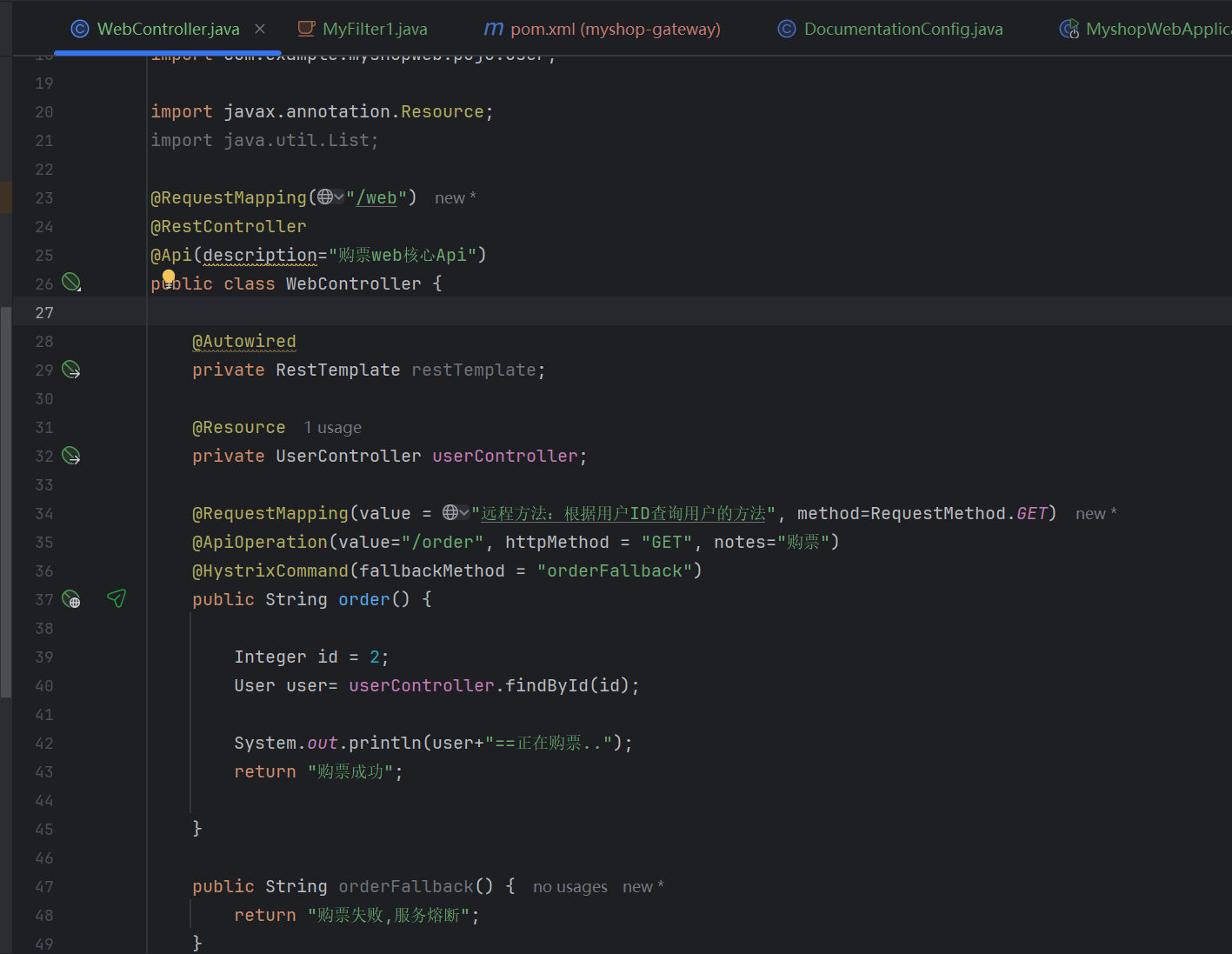
@Configuration  
@EnableSwagger2  
public class SwaggerConfig {  
 @Bean  
 public Docket buildDocket() {  
 return new Docket(DocumentationType.SWAGGER\_2)  
 .apiInfo(buildApiInf()) // .apiInfo(apiInfo())  
 .select()  
 .apis(RequestHandlerSelectors.basePackage(""))// 需要生成文档的包的位置  
 .paths(PathSelectors.any())  
 .build();  
 }  
 private ApiInfo buildApiInf() {  
 return new ApiInfoBuilder()  
 .title("购票web接口详情")  
 .description("Zuul+Swagger2构建RESTful APIs")  
 .version("1.0")  
 .build();  
 }  
}



1. 指定购票web微服务中API的内容

添加@Api注解和@ApiOperation注解

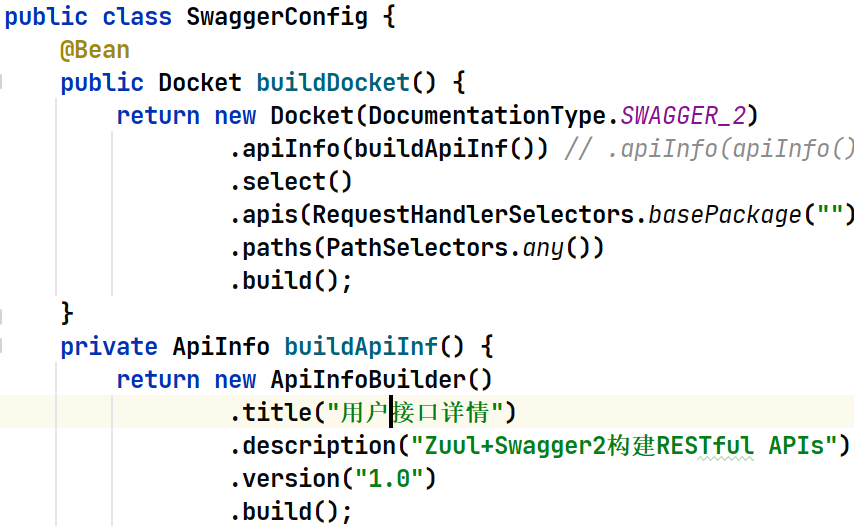


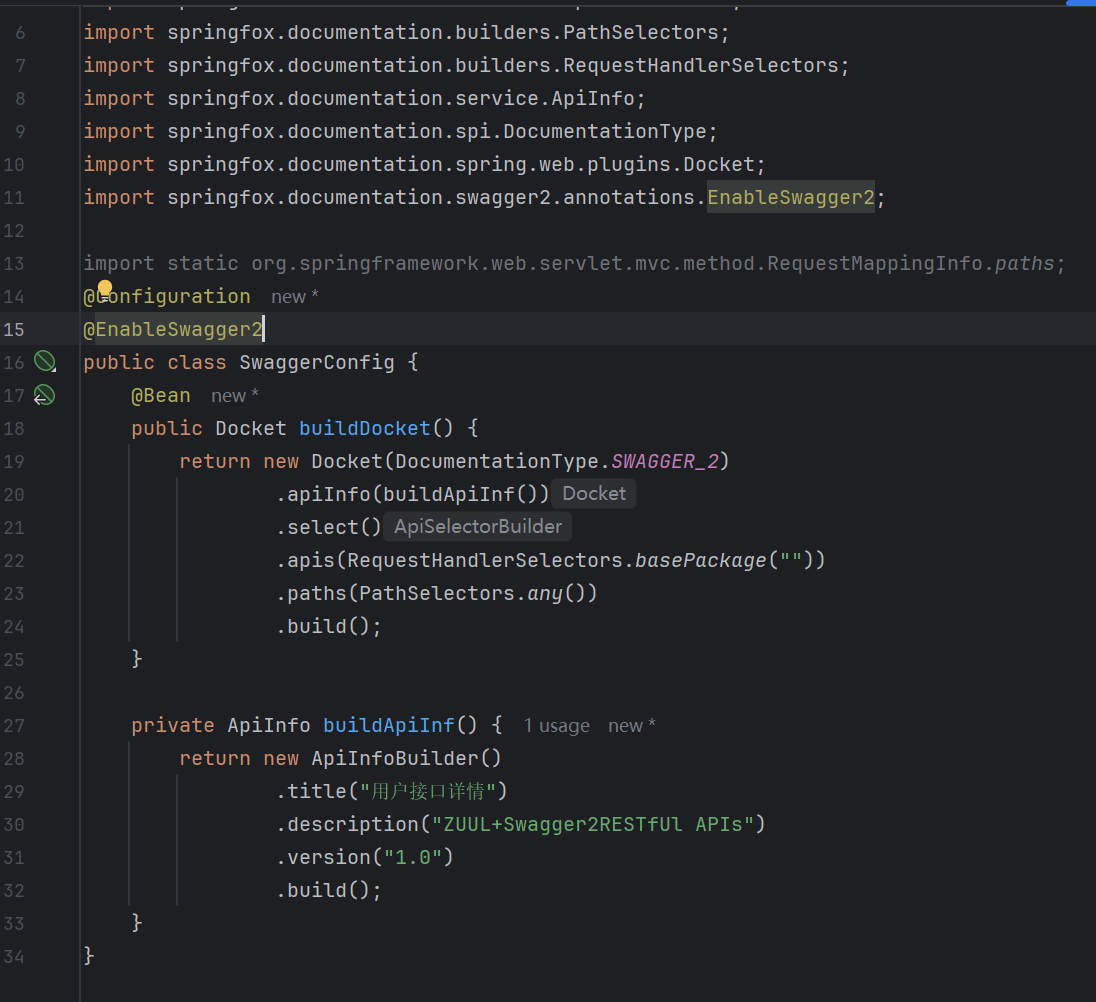


1. 编写用户微服务SwaggerConfig配置类

主要修改buildApiInf方法中内容

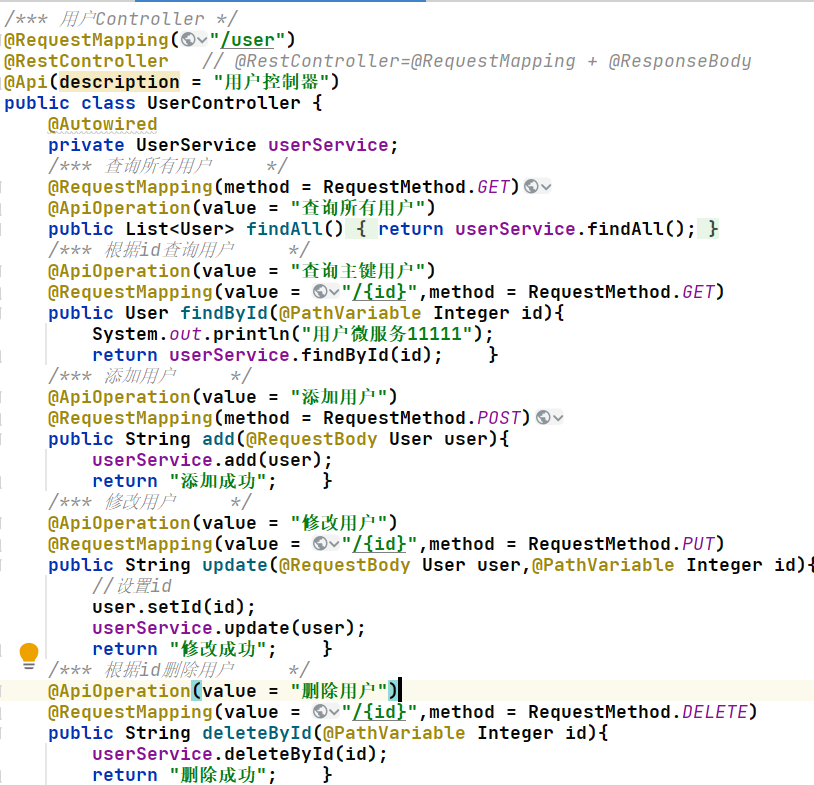
@Configuration  
@EnableSwagger2

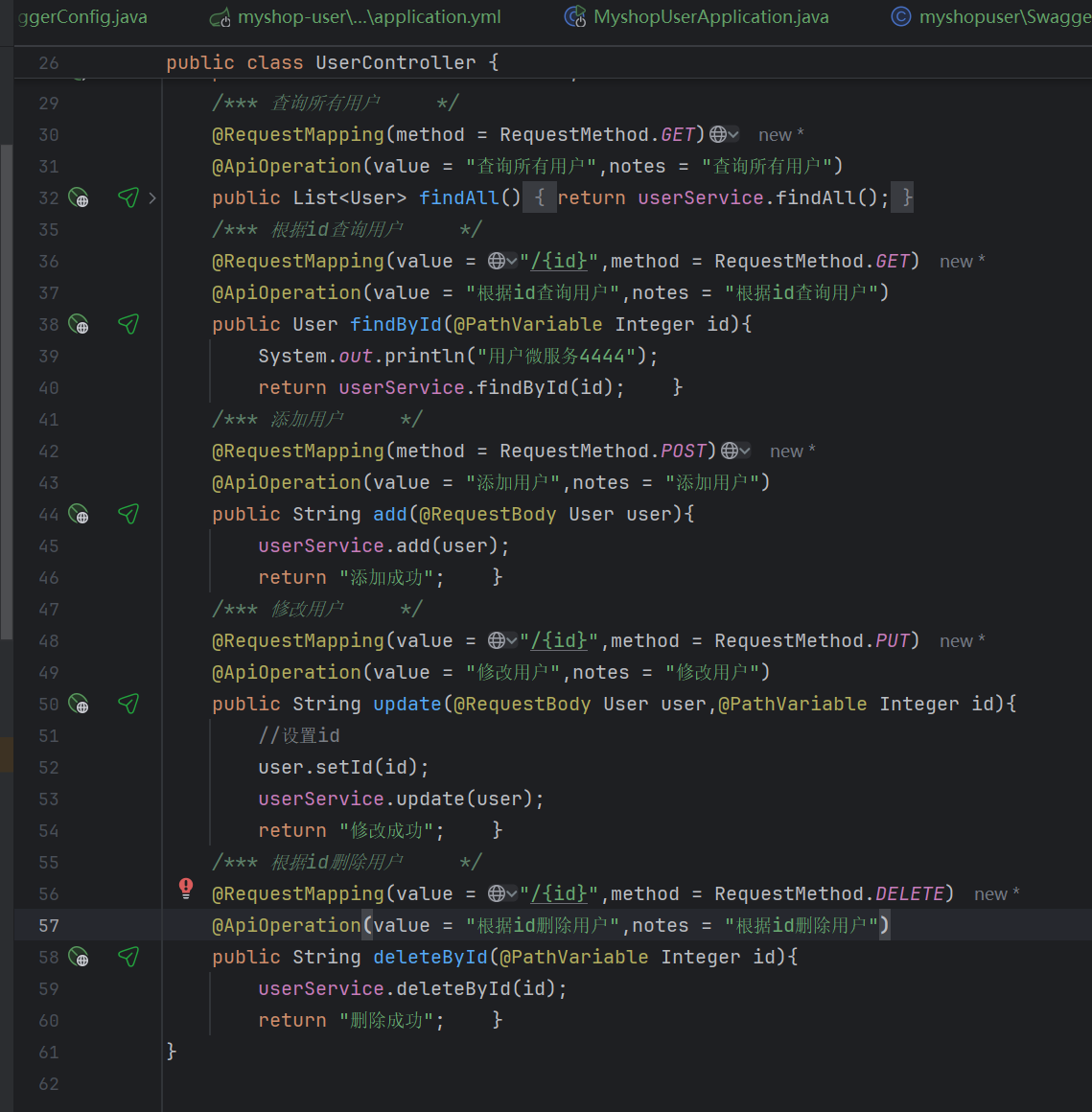




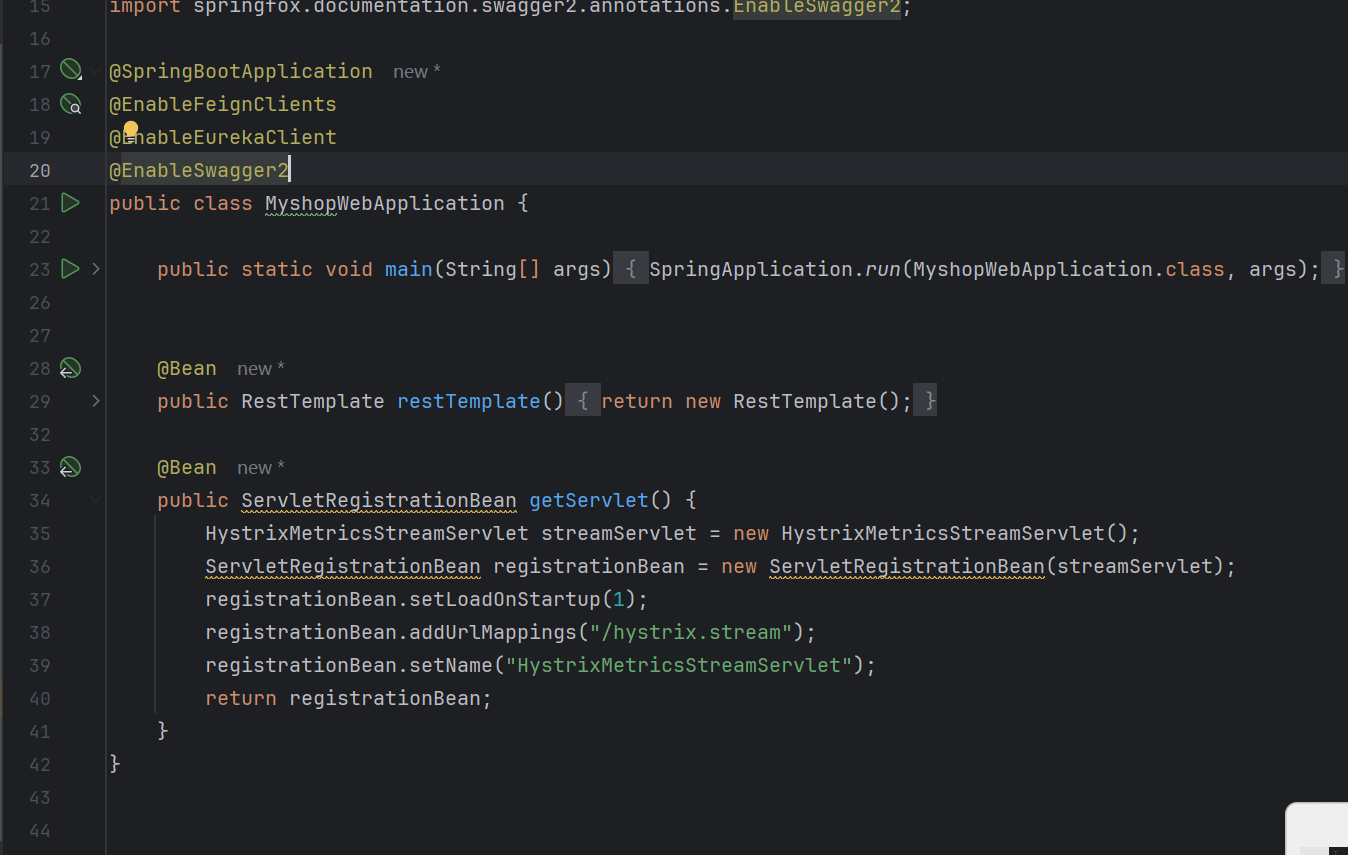
1. 指定用户微服务中API的内容

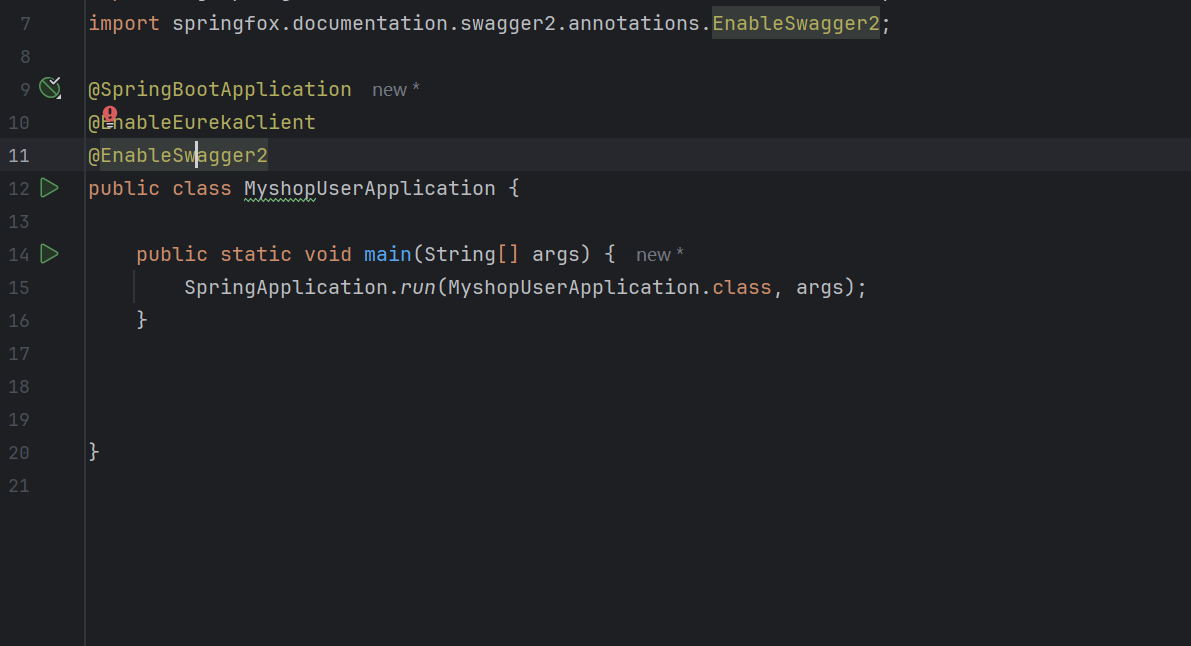
添加@Api注解和@ApiOperation注解





1. 购票微服务与用户微服务引导类开启swagger功能，添加@EnableSwagger2注解





1. 重启各微服务

* 访问网关的swagger主页，此时是否能查看微服务的API？（截图）
* 记录主页内容，并使用swagger进行API测试（截图+测试说明）

