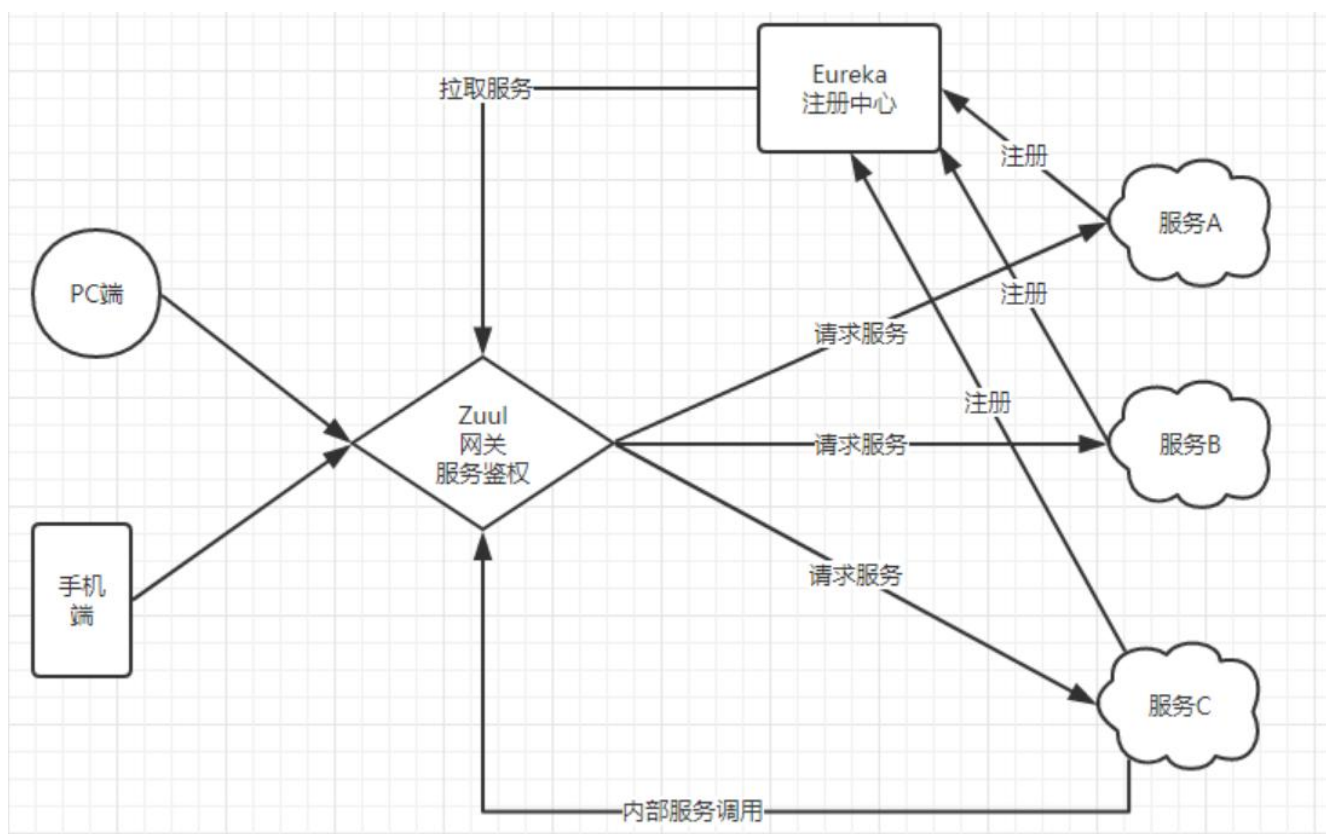


第十一章 Spring Cloud网关

为什么需要网关 (API Gateway)

- 现在的情况是，后端有各种各样的微服务，外部应用如何访问呢？
- 在微服务架构中，后端服务往往不直接开放给调用端，而是通过一个API网关根据请求的url，路由到相应的服务。当添加API网关后，在第三方调用端和服务提供方之间就创建了一面墙，这面墙直接与调用方通信进行权限控制，后将请求均衡分发给后台服务端。



认识Spring Cloud Zuul

Spring Cloud Zuul路由是微服务架构的不可或缺的一部分，提供**动态路由，监控，弹性，安全等**的边缘服务。Zuul是Netflix出品的一个基于JVM路由和服务端的负载均衡器。

Zuul是Netflix开源的微服务网关，它可以和Eureka、Ribbon、Hystrix等组件配合使用。Zuul的核心是一系列的过滤器，这些过滤器可以完成以下功能。

- 身份认证与安全:识别每个资源的验证要求，并拒绝那些与要求不符的请求。
- 审查与监控:在边缘位置追踪有意义的数据和统计结果，从而带来精确的生产视图。
- 动态路由:动态地将请求路由到不同的后端集群。
- 压力测试:逐渐增加指向集群的流量，以了解性能。
- 负载分配:为每一种负载类型分配对应容量，并弃用超出限定值的请求。
- 静态响应处理:在边缘位置直接建立部分响应，从而避免其转发到内部集群。
- 多区域弹性:跨越AWSRegion进行请求路由，旨在实现ELB(Elastic Load Balancing)使用的多样化，以及让系统的边缘更贴近系统的使用者。

Spring Cloud对Zuul进行了整合与增强。目前，Zuul使用的默认HTTP客户端是ApacheHTTP Client,也可以使用RestClient或者okhttp3.OkHttpClient。如果想要使用RestClient,可以设置`ribbon.restclient.enabled=true`;想要使用okhttp3.OkHttpClient,可以设置`ribbon.okhttp.enabled=true`。

Zuul动态路由-实现步骤

- 创建独立的网关微服务模块： myshop-gateway
- 导入zuul和eureka的依赖（网关服务本身也需要注册到Eureka）
- 启动类添加@EnableZuulProxy注解
- 配置application.yml路由规则

1. 创建网关微服务模块



Zuul动态路由-实现步骤

- 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>
```

• 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 前提: 转发的路径(path)和 转发的微服
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)
```


• 引导类

```
@SpringBootApplication
@EnableZuulProxy    // 开启网关代理功能
public class GateWayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GateWayApplication.class, args);
    }
}
```

• 启动各服务

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MYSHOP-GATEWAY	n/a (1)	(1)	UP (1) - TF-laptop:myshop-gateway:8222
MYSHOP-USER	n/a (1)	(1)	UP (1) - TF-laptop:myshop-user:9101
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

• 访问用户微服务



```
1 {  
2   "id": 2,  
3   "username": "txc",  
4   "password": "123456",  
5   "sex": "男",  
6   "money": 6000  
7 }
```

• 通过网关访问服务



```
1 {  
2   "id": 2,  
3   "username": "txc",  
4   "password": "123456",  
5   "sex": "男",  
6   "money": 6000  
7 }
```


- 路由配置中：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

#zuul的动态路由配置 前提：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

```
#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）
```

Zuul动态路由-负载均衡

- 通过上一步的Zuul动态路由配置，已经负载均衡，Zuul是通过Ribbon实现负载均衡，所以默认策略是：轮询。

演示Zuul对用户微服务的负载均衡

Application	AMIs	Availability Zones	Status
MYSHOP-GATEWAY	n/a (1)	(1)	UP (1) - TF-laptop:myshop-gateway:8222
MYSHOP-USER	n/a (2)	(2)	UP (2) - TF-laptop:myshop-user:9001 , TF-laptop:myshop-user:9101
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

← ↻ 🏠 ⓘ localhost:8222/myshop-web/web/order

购票成功

用户微服务22222

Hibernate: select user0_.id

用户微服务22222

Hibernate: select user0_.id

用户微服务11111

Hibernate: select user0_.id

用户微服务11111

Hibernate: select user0_.id

Zuul过滤器

Zuul动态路由基本原理回顾

- 外部请求经过API网关的时候，它是如何被解析并转发到服务具体实例的呢？
- 在Spring Cloud Netflix中，Zuul巧妙的整合了Eureka来实现面向服务的路由。实际上，我们可以直接将API网关也看做是Eureka服务治理下的一个普通微服务应用。它除了会将自己注册到Eureka服务注册中心上之外，也会从注册中心获取所有服务以及它们的实例清单。所以，在Eureka的帮助下，API网关服务本身就已经维护了系统中所有serviceld与实例地址的映射关系。当有外部请求到达API网关的时候，根据请求的URL路径找到最佳匹配的path规则，API网关就可以知道要将该请求路由到哪个具体的serviceld上去。由于在API网关中已经知道serviceld对应服务实例的地址清单，那么只需要通过Ribbon的负载均衡策略，直接在这些清单中选择一个具体的实例进行转发就能完成路由工作了。

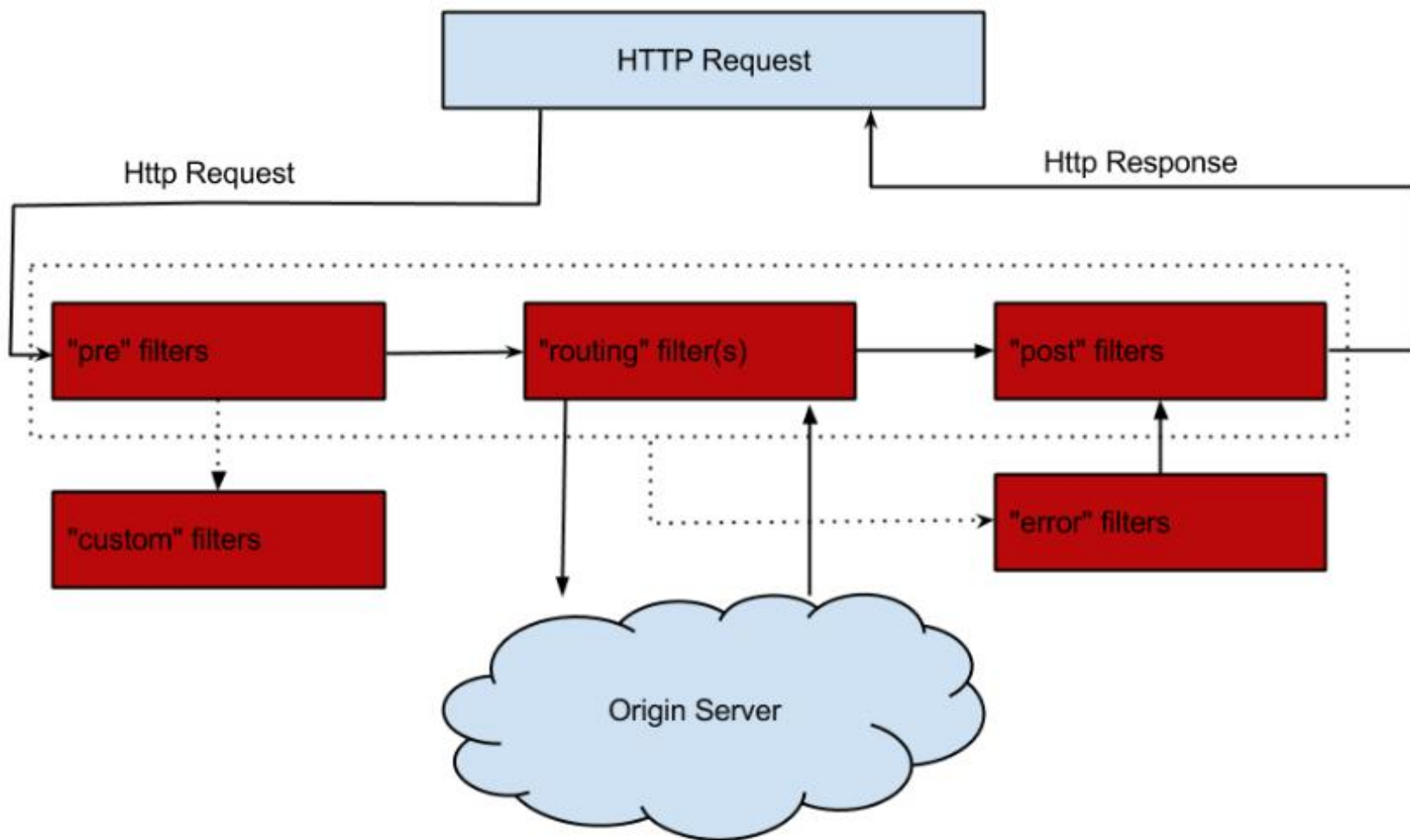
认识Zuul过滤器

- 实际上，动态路由功能在真正运行时，它的路由映射和请求转发都是由几个不同的Zuul过滤器完成的。其中，路由映射主要通过pre类型的过滤器完成，它将请求路径与配置的路由规则进行匹配，以找到需要转发的目标地址；而请求转发的部分则是由route类型的过滤器来完成，对pre类型过滤器获得的路由地址进行转发。所以，Zuul过滤器可以说是Zuul实现API网关功能最为核心的部件。
- 接下来，我们学习Zuul过滤器的编写和使用。

Zuul过滤器-方法说明

- 上面Zuul过滤器中方法含义与功能总结如下：
- **filterType**：该函数需要返回一个字符串来代表过滤器的类型，而这个类型就是在HTTP请求过程中定义各个阶段。在Zuul中默认定义了四种不同生命周期的过滤器类型，具体如下：
 - pre：可以在请求被路由之前调用。
 - route：在路由请求时候被调用。
 - post：在routing和error过滤器之后被调用。
 - error：处理请求时发生错误时被调用。
- **filterOrder**：通过int值来定义过滤器的执行顺序，数值越小优先级越高。
- **shouldFilter**：返回一个boolean类型来判断该过滤器是否要执行。我们可以通过此方法来指定过滤器的有效范围。
- **run**：过滤器的具体逻辑。在该函数中，我们可以实现自定义的过滤逻辑，来确定是否要拦截当前的请求，不对其进行后续的路由，或是在请求路由返回结果之后，对处理结果做一些加工等。

Zuul过滤器-生命周期



Zuul过滤器-生命周期

- 正常流程:

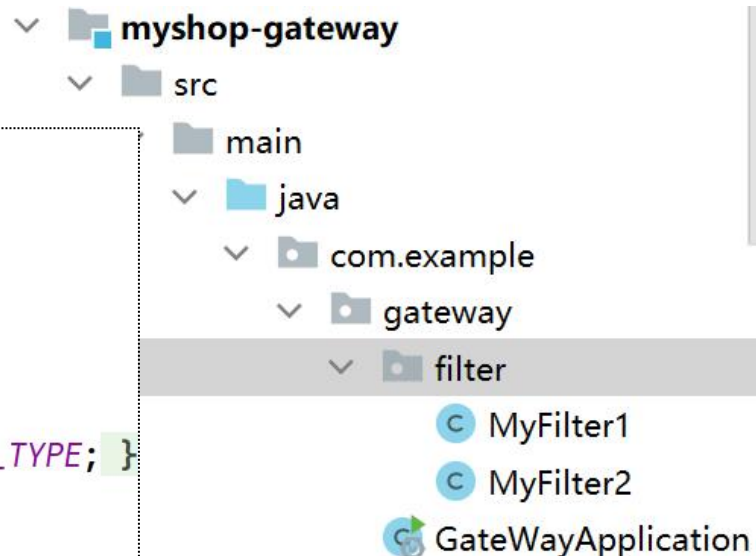
- 请求到达首先会经过pre类型过滤器，而后到达routing类型，进行路由，请求就到达真正的服务提供者，执行请求，返回结果后，会到达post过滤器。而后返回响应。

- 异常流程:

- 整个过程中，pre或者routing过滤器出现异常，都会直接进入error过滤器，再error处理完毕后，会将请求交给POST过滤器，最后返回给用户。
- 如果是error过滤器自己出现异常，最终也会进入POST过滤器，而后返回。
- 如果是POST过滤器出现异常，会跳转到error过滤器，但是与pre和routing不同的时，请求不会再到达POST过滤器了。

自定义过滤器示例

```
/**
 * 第一个Zuul过滤器
 */
@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
 */
@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;
    }
}
```

Application	AMIs	Availability Zones	Status
MYSHOP-GATEWAY	n/a (1)	(1)	UP (1) - TF-laptop:myshop-gateway:8222
MYSHOP-USER	n/a (2)	(2)	UP (2) - TF-laptop:myshop-user:9001 , TF-laptop:myshop-user:9101
MYSHOP-WEB	n/a (1)	(1)	UP (1) - TF-laptop:myshop-web:9002

← ↻ 🏠 ⓘ localhost:8222/myshop-web/web/order

购票成功

Spring Boot

正在运行

GateWayApplication (1)

EurekaApplication

UserApplication

UserApplication

WebApplication

已失败

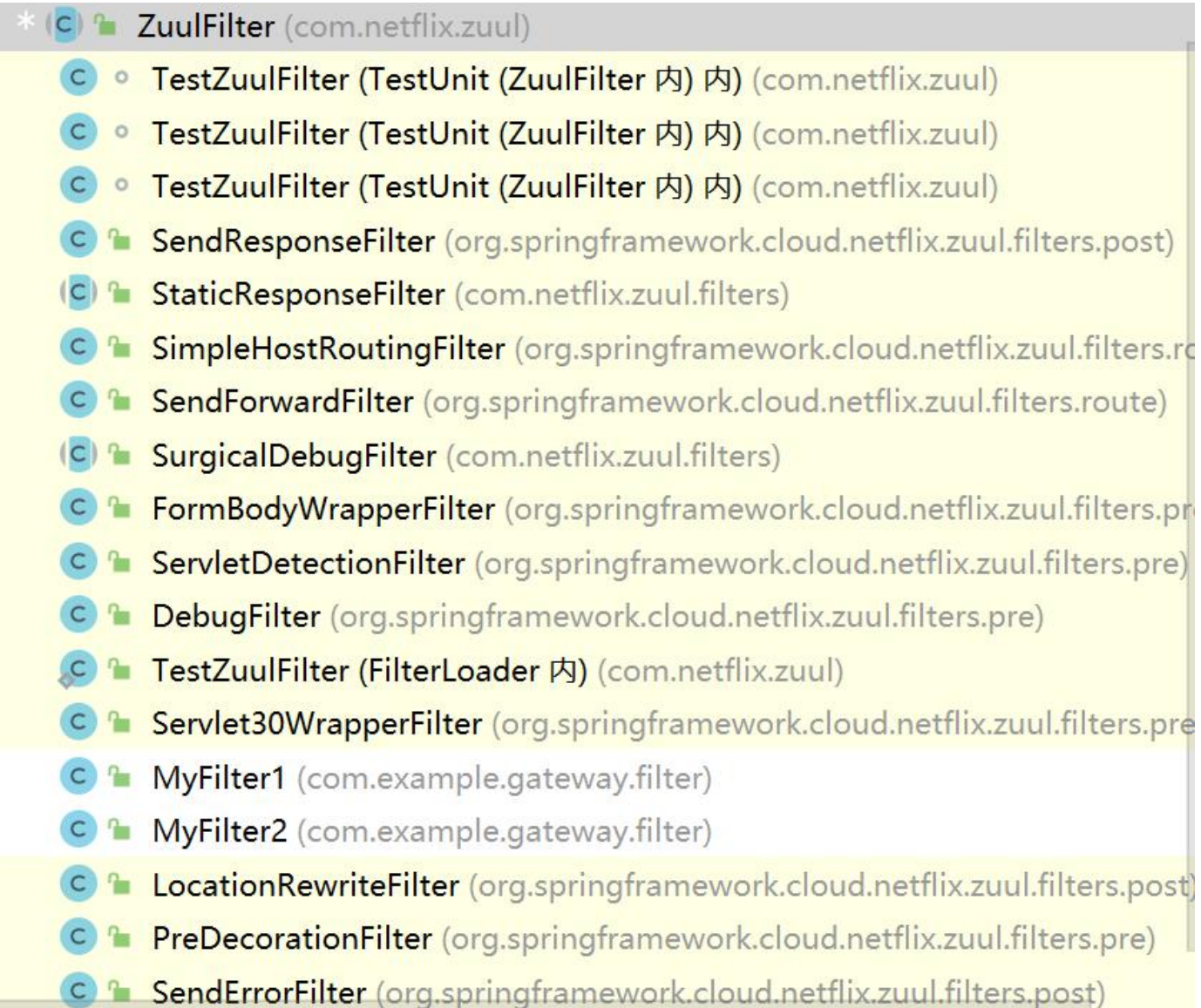
控制台

Actuator

2023-10-22 14:47:51.117 INFO 18700 ---
},Server stats: [[Server:TF-laptop:9002
]]ServerList:org.springframework.cloud.
执行MyFilter1过滤器
执行MyFilter2过滤器
2023-10-22 14:49:52.110 INFO 18760 ---
执行MyFilter1过滤器
执行MyFilter2过滤器
执行MyFilter1过滤器
执行MyFilter2过滤器

Zuul过滤器

public class
MyFilter1 extends
ZuulFilter



自定义权限控制过滤器示例

@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 {  
        //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( name: "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;  
    }  
}
```

localhost:8222/myshop-web/web/order



当前无法使用此页面

如果问题仍然存在，请联系网站所有者。

HTTP ERROR 401

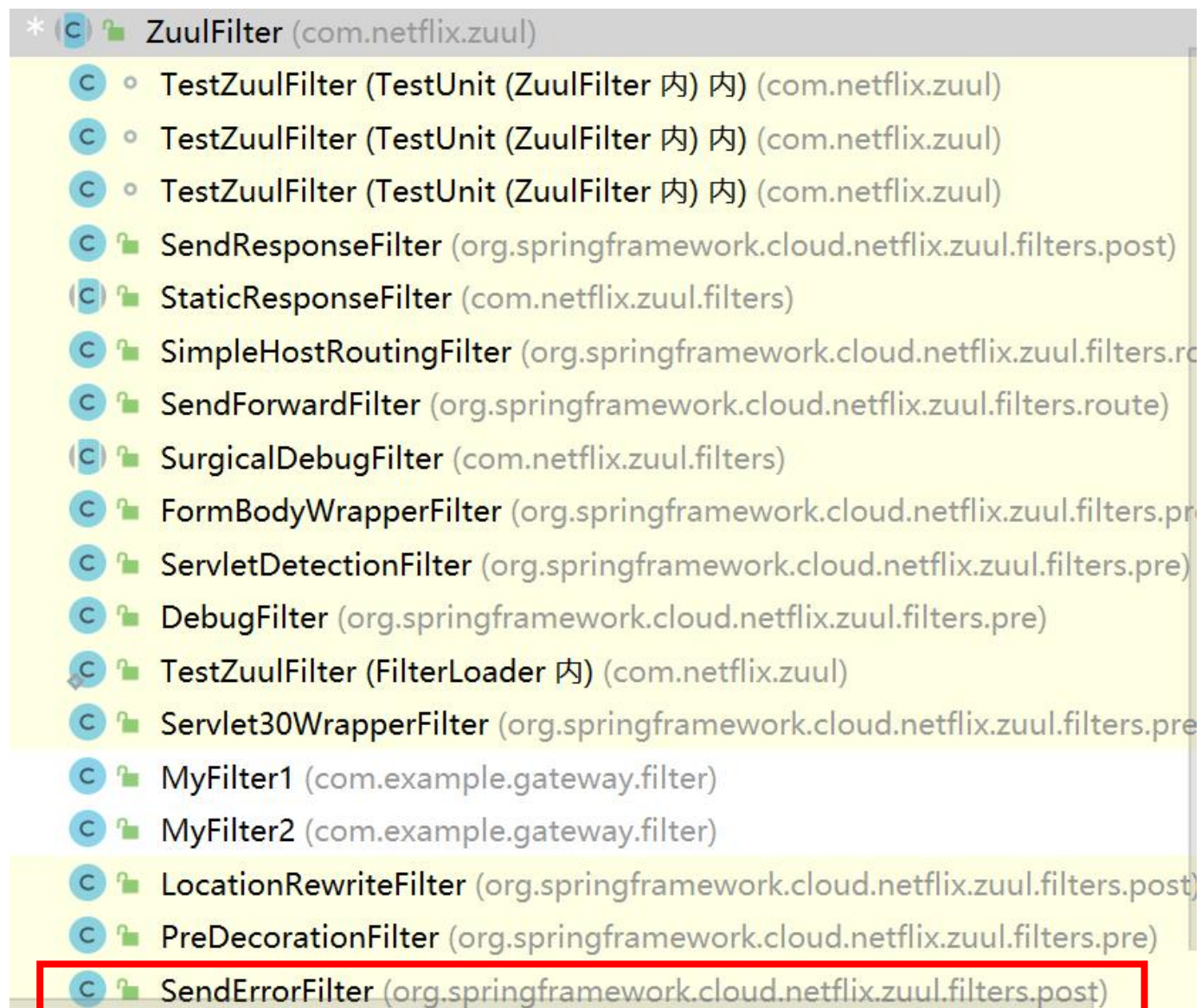


localhost:8222/myshop-web/web/order?token=user

购票成功

Zuul过滤器-网关异常处理 (error)

public class
MyFilter1 extends
ZuulFilter



Zuul过滤器-网关异常处理 (error)

```
public class SendErrorFilter extends ZuulFilter {
    private static final Log log = LoggerFactory.getLogger(this.getClass());
    protected static final String SEND_ERROR_FILTER_PATH = "/error";
    @Value("${error.path:/error}")
    private String errorPath;

    public SendErrorFilter() {}

    public String filterType() { return "error"; }

    public int filterOrder() { return 0; }

    public Object run() {
        try {
            RequestContext ctx = RequestContext.getCurrentContext();
            ZuulException exception = this.findZuulException(ctx.getThrowable());
            HttpServletRequest request = ctx.getRequest();
            request.setAttribute("javax.servlet.error.status_code", exception.getStatusCode());
            log.warn("Error during filtering", exception);
            request.setAttribute("javax.servlet.error.exception", exception);
            if (StringUtils.hasText(exception.getMessage())) {
                request.setAttribute("javax.servlet.error.message", exception.getMessage());
            }
        } catch (Exception e) {
            log.error("Error during filtering", e);
        }
    }
}
```



```
@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;
    }
}
```

← ↻ 🏠 ⓘ localhost:8222/myshop-web/web/order?token=user A

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Oct 22 17:56:50 CST 2023

There was an unexpected error (type=Internal Server Error, status=500).
pre:AuthFilter

自定义异常处理过滤器

*/** 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 {

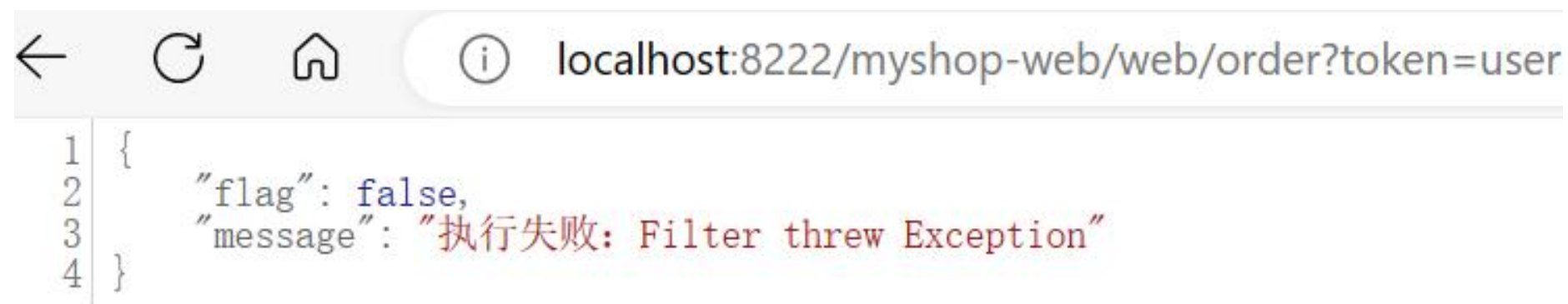
```
public Object run() throws ZuulException {
    System.out.println("进入自定义异常过滤器");
    //1. 捕获异常信息
    RequestContext currentContext = RequestContext.getCurrentContext();
    HttpServletResponse response = currentContext.getResponse();
    //ZuulException: 封装其他zuul过滤器执行过程中发现的异常信息
    ZuulException 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( flag: false, message: "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;
}
```

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

/** 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; }  
}
```


自定义异常过滤器生效，将 Json格式的信息返回给前端用户



```
1 {  
2   "flag": false,  
3   "message": "执行失败: Filter threw Exception"  
4 }
```

后台显示进入自定义异常过滤器



什么是Swagger

- 手写Api文档的几个痛点：
 - 文档需要更新的时候，需要再次发送一份给前端，也就是文档更新交流不及时。
 - 接口返回结果不明确
 - 不能直接在线测试接口，通常需要使用工具，比如postman
 - 接口文档太多，不好管理
- Swagger也就是为了解决这个问题，当然也不能说Swagger就一定是完美的，当然也有缺点，最明显的就是代码移入性比较强。

Zuul网关整合Swagger

- 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>
```


• 编写DocumentationConfig配置类

通过配置资源文档，在首页下拉框选择订单系统时，会请求 <http://localhost:8081/order/v2/api-docs> 获取文档详

```
/**
 * @Component
 * @Primary
 */
public class DocumentationConfig implements SwaggerResourcesProvider {
    private final RouteLocator routeLocator;

    public DocumentationConfig(RouteLocator routeLocator) { this.routeLocator = routeLocator; }

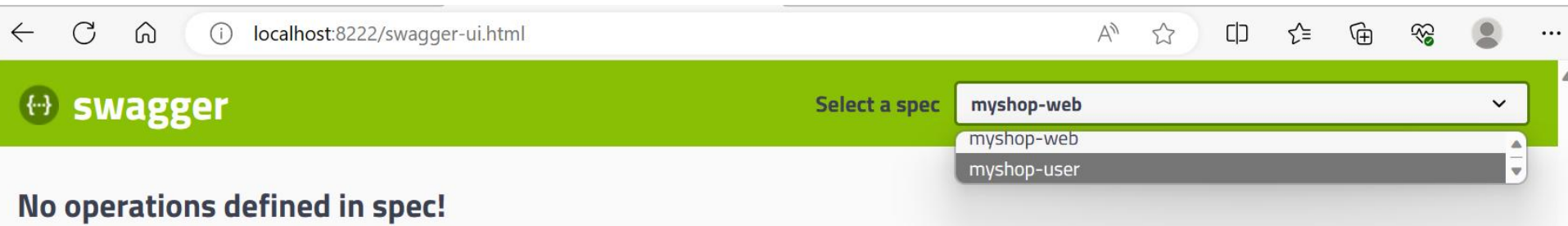
    @Override
    public List<SwaggerResource> get() {
        List<SwaggerResource> resources = new ArrayList<>();
        List<Route> routes = routeLocator.getRoutes();
        routes.forEach(route -> {
            resources.add(swaggerResource(route.getId(), route.getFullPath().replace(
                target: "**", replacement: "v2/api-docs"), version: "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;
    }
}
```

- 引导类开启swagger功能

```
@SpringBootApplication
@EnableZuulProxy    //开启网关代理功能
@EnableSwagger2    //开启Swagger功能
public class GateWayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GateWayApplication.class, args);
    }
}
```

- 访问网关的swagger主页



- 购票微服务(myshop-web)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>
```

- 编写购票微服务的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();
    }
}
```



- 引导类开启swagger功能

```
@EnableSwagger2 //开启Swagger功能
public class WebApplication {
    public static void main(String[] args)
```


- 指定购票web微服务中API的内容

```
@RequestMapping(🌐"/web")
@RestController
@Api(description = "购票web核心Api")
public class WebController {
    @Qualifier("com.example.client.UserController")
    @Autowired
    private UserController userController;
    /* * * 购票方法（使用OpenFeign组件来简化调用代码）
     * @return
     */
    @ApiOperation(value = "远程方法：根据用户ID查询用户的方法")
    @RequestMapping(value = 🌐"/order",method = RequestMethod.GET)
    public String order(){
        // 模拟当前用户
        Integer id = 2;
        User user = userController.findById(id);
        System.out.println(user+"==正在购票...");
        return "购票成功";
    }
}
```

Swagger常用注解

- @Api: 修饰整个类，描述Controller的作用
- @ApiOperation: 描述一个类的一个方法，或者说一个接口
- @ApiParam: 单个参数描述
- @ApiModel: 用对象来接收参数
- @ApiModelProperty: 用对象接收参数时，描述对象的一个字段
- @ApiResponse: HTTP响应其中1个描述
- @ApiResponses: HTTP响应整体描述
- @ApiIgnore: 使用该注解忽略这个API
- @ApiError : 发生错误返回的信息
- @ApiImplicitParam: 一个请求参数
- @ApiImplicitParams: 多个请求参数

- 用户微服务(myshop-user)导入swagger的依赖
- 编写用户微服务SwaggerConfig配置类

```
public class SwaggerConfig {  
    @Bean  
    public Docket buildDocket() {  
        return new Docket(DocumentationType.SWAGGER_2)  
            .apiInfo(buildApiInf()) // .apiInfo(apiInfo()  
            .select()  
            .apis(RequestHandlerSelectors.basePackage(""))  
            .paths(PathSelectors.any())  
            .build();  
    }  
    private ApiInfo buildApiInf() {  
        return new ApiInfoBuilder()  
            .title("用户接口详情")  
            .description("Zuul+Swagger2构建RESTful APIs")  
            .version("1.0")  
            .build();  
    }  
}
```

- 引导类开启swagger功能

```
@EnableSwagger2 //开启Swagger功能  
public class UserApplication {  
    public static void main(String[] args) {
```


指定用
户微服
务中
API的
内容

```
/** 用户Controller */
@RequestMapping(value = "/user")
@RestController // @RestController=@RequestMapping + @ResponseBody
@Api(description = "用户控制器")
public class UserController {
    @Autowired
    private UserService userService;

    /** 查询所有用户 */
    @RequestMapping(method = RequestMethod.GET)
    @ApiOperation(value = "查询所有用户")
    public List<User> findAll() { return userService.findAll(); }

    /** 根据id查询用户 */
    @ApiOperation(value = "查询主键用户")
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public User findById(@PathVariable Integer id){
        System.out.println("用户微服务11111");
        return userService.findById(id);
    }

    /** 添加用户 */
    @ApiOperation(value = "添加用户")
    @RequestMapping(method = RequestMethod.POST)
    public String add(@RequestBody User user){
        userService.add(user);
        return "添加成功";
    }

    /** 修改用户 */
    @ApiOperation(value = "修改用户")
    @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
    public String update(@RequestBody User user, @PathVariable Integer id){
        // 设置id
        user.setId(id);
        userService.update(user);
        return "修改成功";
    }

    /** 根据id删除用户 */
    @ApiOperation(value = "删除用户")
    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    public String deleteById(@PathVariable Integer id){
        userService.deleteById(id);
        return "删除成功";
    }
}
```

用户接口详情 ^{1.0}

[Base URL: localhost:8222/myshop-user/]

<http://localhost:8222/myshop-user/v2/api-docs>

Zuul+Swagger2构建RESTful APIs

basic-error-controller Basic Error Controller

user-controller 用户控制器

GET**/user** 查询所有用户**POST****/user** 添加用户**GET****/user/{id}** 查询主键用户**PUT****/user/{id}** 修改用户

GET

/user 查询所有用户

Parameters

Try it out

No parameters

Responses

Response content type

/



Code

Description

200

OK

Example Value | Model

```
[
  {
    "id": 0,
    "money": 0,
    "password": "string",
    "sex": "string",
    "username": "string"
  }
]
```

401

Unauthorized

GET

/user 查询所有用户

Parameters

Cancel

No parameters

Execute

Request URL

`http://localhost:8222/myshop-user/user`

Server response

Code

Details

200

Response body

```
[
  {
    "id": 1,
    "username": "tf",
    "password": "123456",
    "sex": "女",
    "money": 5000
  },
  {
    "id": 2,
    "username": "txc",
    "password": "123456",
    "sex": "男",
    "money": 6000
  }
]
```

Response headers

```
content-type: application/json;charset=UTF-8
date: Mon, 23 Oct 2023 06:01:39 GMT
transfer-encoding: chunked
```

Responses

购票web接口详情 1.0

[Base URL: localhost:8222/myshop-web/]

<http://localhost:8222/myshop-web/v2/api-docs>

Zuul+Swagger2构建RESTful APIs

basic-error-controller Basic Error Controller

web-controller 购票web核心Api

GET

/web/order 远程方法: 根据用户ID查询用户的方法

Parameters

No parameters

Execute

Clear

Responses

Response content type

/