在微服务架构中，后端服务往往不会直接开放给调用端，而是通过一个网关根据请求的url，路由到相应的服务。当添加网关后，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。在Spring Cloud体系中，Spring Cloud Zuul就起到了网关的作用，它可以提供动态路由，监控，弹性，安全等的边缘服务。

Zuul的简单使用。

#### 1、创建项目

新建一个Springboot项目zuul-server1。

#### 2、引入依赖

<!--web-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--eureka client-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<!--zuul-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

springcloud依赖，在标签外面：

<!--spring-cloud依赖-->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

properties标签里设置spring-cloud版本：

<!--spring-cloud版本-->

<spring-cloud.version>Finchley.SR2</spring-cloud.version>

#### 3、修改配置文件

zuul需要注册到eureka上，然后我们配置路由，配合指定类型的请求路由到指定的服务商；当请求path是以/api-a/开头的，就转发到feign-server1服务上，如果有其他路由，在routes下继续添加即可：

**server**:

**port**: 8018

**spring**:

**application**:

**name**: zuul-server

**eureka**:

**client**:

**service-url**:

**defaultZone**: http://localhost:8761/eureka

**zuul**:

**routes**:

**apia**:

**path**: /api-a/\*\*

**serviceId**: feign-server1

**apib**:

**path**: /api-b/\*\*

**serviceId**: service-ribbon

#### 4、添加启动注解

*/\*\**

*\* 开启zuul的功能*

*\*/*

@EnableZuulProxy

@EnableDiscoveryClient

@EnableEurekaClient

@SpringBootApplication

public class ZuulServer1Application {

public static void main(String[] args) {

SpringApplication.*run*(ZuulServer1Application.class, args);

}

}

#### 5、自定义Filter

zuul提供的功能，不仅仅是路由，我们还可以把所有的请求，在网关这一层，做个安全校验，过滤，或者，也可以做一些其他的的处理。这里做个简单的token是否为空的校验，我们需要用到Filter。

Filter是Zuul的核心，用来实现对外服务的控制。Filter的生命周期有4个，分别是“PRE”、“ROUTING”、“POST”、“ERROR”。Zuul的大部分功能都是通过过滤器来实现的，这些过滤器类型对应于请求的典型生命周期。

* PRE：这种过滤器在请求被路由之前调用。我们可以利用这种过滤器实现身份验证、在集群中选择请求的微服务、记录调试信息等。
* ROUTING：这种过滤器将请求路由到微服务。这种过滤器用于构建发送给微服务的请求，并使用Apache HttpClient或Netflix Ribbon请求微服务。
* POST：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTP Header、手机统计信息和指标、将响应从微服务发送给客户端等。
* ERROR：在其他阶段发生错误时执行该过滤器。除了默认的过滤器类型，Zuul还允许我们创建自定义的过滤器类型。例如，我们可以定制一种STATIC类型的过滤器，直接在Zuul中生成响应，而不将请求转发到后端的微服务。

具体内容可参考[服务网关Zuul高级篇](http://www.ityouknow.com/springcloud/2018/01/20/spring-cloud-zuul.html)

自定义一个Filter，用来校验token是否为空。自定义Filter需要继承ZuulFilter，并实现其中的一些方法。在run中，多请求做个校验，当token为空时，就直接返回，不再向后端服务转发。

@Component

public class RequestFilter extends ZuulFilter {

private Logger logger = LoggerFactory.*getLogger*(RequestFilter.class);

*/\*\**

*\* pre 路由之前*

*\* routing 路由之时*

*\* post 路由之后*

*\* error 发送错误调用*

*\**

*\** ***@return***

*\*/*

@Override

public String filterType() {

return "pre";

}

*/\*\**

*\* 过滤的顺序，数字越小越先执行*

*\**

*\** ***@return***

*\*/*

@Override

public int filterOrder() {

return 0;

}

*/\*\**

*\* 这里可以写逻辑判断，是否要过滤，true表示过滤，false表示不过滤*

*\**

*\** ***@return***

*\*/*

@Override

public boolean shouldFilter() {

return true;

}

*/\*\**

*\* filter需要执行的具体操作*

*\**

*\** ***@return***

*\** ***@throws*** *ZuulException*

*\*/*

@Override

public Object run() throws ZuulException {

RequestContext currentContext = RequestContext.*getCurrentContext*();

HttpServletRequest request = currentContext.getRequest();

String token = request.getParameter("token");

if (StringUtils.*isEmpty*(token)) {

logger.warn("=========>token is empty");

//不对其进行路由

currentContext.setSendZuulResponse(false);

currentContext.setResponseStatusCode(401);

try {

currentContext.getResponse().getWriter().write("token is empty");

} catch (Exception ex) {

ex.printStackTrace();

}

} else {

//进行路由

currentContext.setSendZuulResponse(true);

currentContext.setResponseStatusCode(200);

}

return null;

}

}

#### 6、启动项目，测试

启动项目，访问接口<http://localhost:8018/api-a/getCompany?id=234>。这个请求中没有带token参数，访问结果：



会发现这个请求并没有到后台，直接在网关这里就被拦截返回了。

再访问接口<http://localhost:8018/api-b/getCompany?id=234&token=qweqew>，

这里带token参数，请求顺利到后台，请求到了指定服务。