一：Spring Cloud 的路由 Zuul:

在Cloud中，项目的提供往往提供分布式操作。提供者应该是多个，而且提供者往往还要采取集群的方式。导致 提供者，被管理者是众多的。

消费者要进行消费，必须得知道有那些提供者，知道对方提供者的实例名称。还得知道提供者的API列表。

软件公司都有开发文档：

A程序员商品系统：

<Http://GoodsProvider:8000>

<URL:FindAll>

Parama: null

DESC:

A程序员订单系统：

<Http://OrderProvider:8001>

<URL:FindAll>

Parama: null

DESC:

消费者调用的时候是很痛苦的。

路由的功能：

就是请求转发。过滤器。



1：实现一个简单的路由：

两个个提供者 一个路由 一个消费者

提供者1：

在<http://localhost:9000/findAll>方法

提供者2：

在<http://localhost:9001/findPerson>方法

如果没有路由，则消费者必须完全清楚，我要访问谁，ip是什么，端口是什么，方法是什么？

路由：

1：pom.xml

<dependency>

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

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

</dependency>

2: 启动类 @EnableZuulProxy

3：yml

server:

port: 10000

zuul:

routes:

goods: #第一个键 ，键是可以自己定义的，叫什么都可以

path: /goods/\*\* #今后对当前路由的地址是 /goods的所有操作，都把请求转交给下面的url

url: http://localhost:9000/

# 意味着你原来的提供者的访问路径

#http://localhost:9000/findAll

#现在路由的访问规则是

#http://localhost:10000/goods/findAll

#路由的IP地址和端口 + 你在路由中描述的键 + 提供者的API名称

orders: #第一个键 ，键是可以自己定义的，叫什么都可以

path: /orders/\*\* #今后对当前路由的地址是 /goods的所有操作，都把请求转交给下面的url

url: http://localhost:9001/

4: 访问者现在访问

提供者1：

在<http://localhost:9000/findAll>方法

提供者2：

在<http://localhost:9001/findPerson>方法

现在：在浏览器

http://localhost:10000/ goods/findAll 相当于访问的提供者1的findAll方法

http://localhost:10000/ orders/findPerson 相当于访问的提供者2的findPerson方法

final结果：

消费者只需要知道路由就ok了，饭店吃饭只跟服务员(陆云)打交道就已经够了。

2：路由跟Eureka进行整合：

核心点只有一个，就是把你的2个提供者，1个消费者，一个路由，注册到Eureka中去。

eureka:

client:

register-with-eureka: true

fetch-registry: true

service-url:

defaultZone:

http://localhost:8888/eureka/

instance:

prefer-ip-address: true

注册后的结果就是你的提供者现在可以是集群中。因为消费者访问的时候，访问的是实例名称，而不是ip地址了。

zuul:

routes:

goods:

path: /goods/\*\*

service-id: PROVIDER

orders:

path: /orders/\*\*

service-id: PROVIDER1

3：今后还有提供者怎么办：

路由匹配规则：

zuul:

routes:

goods:

path: /goods/\*\*

service-id: PROVIDER

orders:

path: /orders/\*\*

service-id: PROVIDER1

我要再加一个提供者，yml还得再来描述这个提供者。如果我有80个提供者，还得描述80个。有可能新项目要上线，加了新模块。

其实，只要你把自己注册到Eureka中。默认Eureka就存在一种规则，跟路由绑定的规则。

哪怕你的yml没有写任何的路由匹配规则。

<http://localhost:10000/shop/find>

路由的地址+提供者的实例名称(yml中对应)+提供者的方法

路由匹配规则，其实我可以不写，不配置，只要注册到Eureka中就ok拉。

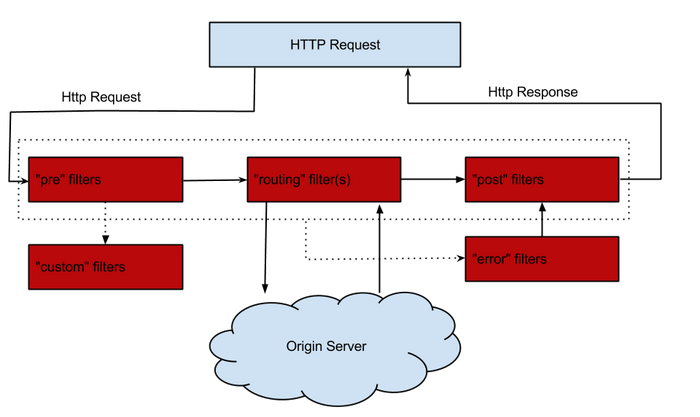
四：路由的过滤器：

他跟我们过去的HttpFilter是不一样的。

路由总会经过过滤器。所有请求都会先经过过滤器。

能干什么事：

权限判断，IP过滤，要求程序必须先进路由，再进提供者，你以前过滤器能干的很多事，它都能干。



@Component

public class MyZuulFilter extends ZuulFilter{

private List<String> ips = new ArrayList<String>();//应该来自于数据库

public MyZuulFilter() {

ips.add("192.168.9.60");

ips.add("192.168.9.8");

ips.add("192.168.9.22");

}

@Override

public boolean shouldFilter() {

// TODO Auto-generated method stub

//是否执行这个过滤器

return true;

}

@Override

public Object run() throws ZuulException {

// TODO Auto-generated method stub

System.out.println("进入过滤器了");

return null;

}

@Override

public String filterType() {

// TODO Auto-generated method stub

//当前过滤器的类型 类型总共只有四个 pre routing post error

//pre代表当前过滤器在你的行为执行之前执行这个过滤器

//ip黑名单

RequestContext rc = RequestContext.getCurrentContext();

HttpServletRequest request = rc.getRequest();

String ip = request.getRemoteHost();

System.out.println("当前ip为 "+ip);

if(ips.contains(ip)) {//非法请求，不会把请求发送给提供者，而是选择代码从当前节点直接中断

rc.setSendZuulResponse(false);//不会把请求发送给提供者，而是选择代码从当前节点直接中断

rc.setResponseBody("您的ip已经被锁定了，警察正在来的路上");

rc.getResponse().setContentType("text/html;charset=UTF-8");

}

return "pre";

}

@Override

public int filterOrder() {

// TODO Auto-generated method stub

//当前过滤器的执行顺序 这个号码越大 越后执行 在项目 中可以有n个过滤器

return 0;

}

}

五：路由的传参：

RequestContext rc = RequestContext.getCurrentContext();

实现原理是***threadLocal.只要在当前线程中，这个变量值一直是有的。***

rc.set(key);

rc.get(key);