# 实现一个Eureka Server

## 导入依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency> |

## 设置application.yml

|  |
| --- |
| server:  port: 8761  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ |

## 添加注解

|  |
| --- |
| **package** cn.yanliang;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication  @EnableEurekaServer  **public** **class** EurekaServer {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(EurekaServer.**class**, args);  }  } |

# 将微服务注册到Eureka

## 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency> |

## 修改application.yml

|  |
| --- |
| eureka:  client:  serviceUrl:  defaultZone: http://localhost:8761/eureka/ |

## 添加注解

@EnableEurekaClient 或者 @EnableDiscoveryClient

## 怎么去设置应用的名称

|  |
| --- |
| spring:  application:  name: microservice-provider-user |

## 怎么去设置成IP地址

|  |
| --- |
| eureka.instance.preferIpAddress = true |

## 修改eureka实例的ID

|  |
| --- |
| eureka:  instance:  instanceId: ${spring.cloud.client.hostname}:${spring.application.name}:${spring.application.instance\_id:${server.port}}}  instanceId: ${spring.application.name}:${spring.cloud.client.ipAddress}:${spring.application.instance\_id:${server.port}} |

## Status Page and Health Indicator

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency> |

## Eureka’s Health Checks

|  |
| --- |
| eureka:  client:  healthcheck:  enabled: true |

## 第一次超时延长的处理

|  |
| --- |
| # 解决第一次请求报超时异常的方案：  # hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 5000  # 或者：  # hystrix.command.default.execution.timeout.enabled: false  # 或者：  feign.hystrix.enabled: false ## 索性禁用feign的hystrix支持  # 超时的issue：https://github.com/spring-cloud/spring-cloud-netflix/issues/768  # 超时的解决方案： http://stackoverflow.com/questions/27375557/hystrix-command-fails-with-timed-out-and-no-fallback-available  # hystrix配置： https://github.com/Netflix/Hystrix/wiki/Configuration#execution.isolation.thread.timeoutInMilliseconds |

# Authenticating with the Eureka Server

## 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency> |

## 修改application.yml

|  |
| --- |
| security:  basic:  enabled: **true**  user:  name: user  password: password123  spring:  application:  name: eureka\_server  server:  port: 8761  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://user:password123@${eureka.instance.hostname}:${server.port}/eureka |

## 使用eureka客户端--简单测试

|  |
| --- |
| @Autowired  private EurekaClient discoveryClient;  public String serviceUrl() {  InstanceInfo instance = discoveryClient.getNextServerFromEureka("STORES", false);  return instance.getHomePageUrl();  } |

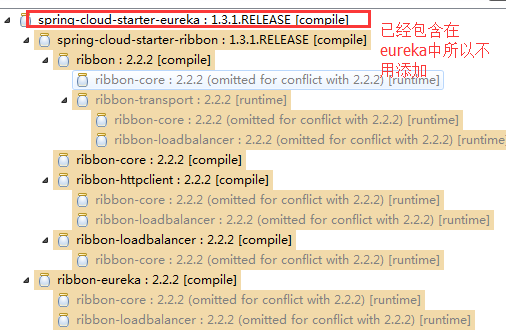
## Client使用

|  |
| --- |
| eureka:  client:  healthcheck:  enabled: **true**  serviceUrl:  defaultZone: http://user:password123@localhost:8761/eureka/  instance:  preferIpAddress : **true**  instanceId: ${spring.application.name}:${spring.cloud.client.ipAddress}:${spring.application.instance\_id:${server.port}} |

# 负载均衡ribbon

## 添加依赖

|  |
| --- |
| group  org.springframework.cloud  artifact id  spring-cloud-starter-ribbon |



## 添加注解

|  |
| --- |
| package cn.yanliang;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.client.loadbalancer.LoadBalanced;  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  import org.springframework.context.annotation.Bean;  import org.springframework.web.client.RestTemplate;  @SpringBootApplication  @EnableEurekaClient  public class MovieApp {  @Bean  @LoadBalanced //默认采用的是轮询算法  public RestTemplate restTemplate() {  return new RestTemplate();  }  public static void main(String[] args) {  SpringApplication.run(MovieApp.class, args);  }  } |

## 调用接口的修改 （VIP虚拟IP）

|  |
| --- |
| @GetMapping("/movie/{id}")  **public** User findByUser(@PathVariable Long id){  **return** restTemplate.getForObject("http://microservice-provider-user/simple/" + id, User.**class**);  } |

# 通过代码自定配置ribbon

## 创建自己定义的Ribbon客户端

|  |
| --- |
| **package** cn.config;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** com.netflix.client.config.IClientConfig;  **import** com.netflix.loadbalancer.IRule;  **import** com.netflix.loadbalancer.RandomRule;  /\*\*  \* 代码定义ribbon  \* 返回的路由选择采用的是随机调度  \* **@see** 代码定义ribbon时候必须在注解SpringBootApplication的上一层不然等价于@RibbonClients  \*/  @Configuration  //name -- 微服务的application.name | configuration  **public** **class** TestConfiguration {  @Autowired  IClientConfig config;    /\*\*  \* 使用随机路由  \* **@param** config  \* **@return**  \*/  @Bean  **public** IRule ribbonRule(IClientConfig config) {  **return** **new** RandomRule();  }  } |

## 添加注解

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  //在调用microservice-provider-user的时候，我们使用自己定制的ribbon-随机调度  @RibbonClient(name = "microservice-provider-user", configuration = TestConfiguration.**class**) |

## 如何让RibbonClient的configuration在ComponentScan 注解内

### 创建不扫描的注解

|  |
| --- |
| **package** cn.yanliang;  **public** **@interface** ExcludeFromComponentScan {  } |

### 添加@ComponentScan注解过滤

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  //在调用microservice-provider-user的时候，我们使用自己定制的ribbon-随机调度  @RibbonClient(name = "microservice-provider-user", configuration = TestConfiguration.**class**)  @ComponentScan(excludeFilters={@ComponentScan.Filter(type=FilterType.***ANNOTATION***, value = ExcludeFromComponentScan.**class**)}) |

### 添加自定义注解

|  |
| --- |
| **package** cn.yanliang;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** com.netflix.loadbalancer.IRule;  **import** com.netflix.loadbalancer.RandomRule;  /\*\*  \* 代码定义ribbon  \* 返回的路由选择采用的是随机调度  \* **@see** 代码定义ribbon时候必须在注解SpringBootApplication的上一层  \*/  @Configuration  @ExcludeFromComponentScan  //name -- 微服务的application.name | configuration  **public** **class** TestConfiguration {    /\*\*  \* 使用随机路由  \* **@param** config  \* **@return**  \*/  @Bean  **public** IRule ribbonRule() {  **return** **new** RandomRule();  }  } |

## 通过properties配置ribbon

|  |
| --- |
| # 微服务的spring.applicaton.name  microservice-provider-user:  ribbon:  NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule |

## How to Use Ribbon Without Eureka

|  |
| --- |
| microservice-provider-user:  ribbon:  listOfServers: localhost:7900  #配置这个后请求microservice-provider-user将统一使用localhost:7900 |

# 使用Feign

## 添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-feign</artifactId>  </dependency> |

## 创建接口

|  |
| --- |
| **package** cn.yanliang.feign;  **import** org.springframework.cloud.netflix.feign.FeignClient;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.PostMapping;  **import** org.springframework.web.bind.annotation.RequestBody;  **import** cn.yanliang.entity.User;  @FeignClient("microservice-provider-user")  **public** **interface** UserFeignClient { // @RequestMapping(value = "/simple/{id}", method = RequestMethod.GET) @GetMapping("/simple/{id}")  **public** User findById(@PathVariable("id") Long id); // 1.@PathVariable得设置value  // @PathVariable不能写  // java.lang.IllegalStateException: PathVariable annotation was empty on  // param 0.    @PostMapping("/testUser")  **public** User getUser(@RequestBody User user);  } |

## 注入使用

|  |
| --- |
| @Autowired  **private** UserFeignClient userFeignClient;  @GetMapping("/movie/{id}")  **public** User findById(@PathVariable Long id) {  **return** **this**.userFeignClient.findById(id);  }    @PostMapping("/testUser")  **public** User getUser(@RequestBody User user){  **return** **this**.userFeignClient.getUser(user);  } |

## 添加注解

|  |
| --- |
| @EnableFeignClients |

# 配置Feign

## 添加配置类

|  |
| --- |
| **package** cn.config;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** feign.Contract;  **import** feign.Logger;  /\*\*  \* 创建一个默认的Feign,这样在使用我们就不能使用SpringMVC的注解必须使用Feign提供  \* **@See** https://github.com/OpenFeign/feign --> Base Apis  \*  \*/  @Configuration  **public** **class** Configuration1 {  @Bean  **public** Contract feignContract() {  **return** **new** feign.Contract.Default();  }  }  ---------------------------------------------------------------------  package cn.config;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import feign.Contract;  import feign.auth.BasicAuthRequestInterceptor;  /\*\*  \* Eureka需要用户名和密码才能请求接口的配置  \*  \*/  @Configuration  public class Configuration2 {  @Bean  public BasicAuthRequestInterceptor basicAuthRequestInterceptor() {  return new BasicAuthRequestInterceptor("user", "password123");  }  } |

## @FeignClient中添加自定义config

|  |
| --- |
| package cn.yanliang.feign;  import org.springframework.cloud.netflix.feign.FeignClient;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import cn.config.Configuration1;  import cn.yanliang.entity.User;  import feign.Param;  import feign.RequestLine;  /\*\*  \* 使用自定义的Configuration  \* -- 这个配置中不能使用  \* -- 报错 org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'movieController': Unsatisfied dependency expressed through field 'userFeignClient'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'cn.yanliang.feign.UserFeignClient': FactoryBean threw exception on object creation; nested exception is java.lang.IllegalStateException: Method findById not annotated with HTTP method type (ex. GET, POST)  \*  \*/  @FeignClient(name = "microservice-provider-user", configuration = Configuration1.class)  public interface UserFeignClient {  @RequestLine("GET /simple/{id}")  public User findById(@Param("id") Long id);  }  package cn.yanliang.feign;  import org.springframework.cloud.netflix.feign.FeignClient;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import cn.config.Configuration1;  import cn.config.Configuration2;  import cn.yanliang.entity.User;  import feign.Param;  import feign.RequestLine;  /\*\*  \* 使用自定义的Configuration  \* -- 这个配置中不能使用  \* -- 报错 org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'movieController': Unsatisfied dependency expressed through field 'userFeignClient'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'cn.yanliang.feign.UserFeignClient': FactoryBean threw exception on object creation; nested exception is java.lang.IllegalStateException: Method findById not annotated with HTTP method type (ex. GET, POST)  \*  \*/  @FeignClient(name = "eureka", url="http://localhost:8761/", configuration = Configuration2.class)  public interface EurekaFeignClient {  @GetMapping("/eureka/apps/{serverName}")  public String getServerName(@PathVariable("serverName") String serverName);  } |

# Feign日志开启

## 在application.yml中配置

|  |
| --- |
| logging:  level:  cn.yanliang.feign.UserFeignClient: DEBUG  #需要连接的对象 |

## 在自定义的config类中添加日志的实体

|  |
| --- |
| **package** cn.config;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** feign.Contract;  **import** feign.Logger;  /\*\*  \* 创建一个默认的Feign,这样在使用我们就不能使用SpringMVC的注解必须使用Feign提供  \* **@See** https://github.com/OpenFeign/feign --> Base Apis  \*  \*/  @Configuration  **public** **class** Configuration1 {    @Bean  Logger.Level feignLoggerLevel() {  **return** Logger.Level.***FULL***;  }    @Bean  **public** Contract feignContract() {  **return** **new** feign.Contract.Default();  }  } |

# Hystrix的引入

## 引入pom

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-hystrix</artifactId>  </dependency> |

## 添加注解

|  |
| --- |
| //开启断路器  @EnableCircuitBreaker |

## 编写回调

|  |
| --- |
| @RestController  **public** **class** MovieController {    @Autowired  **private** RestTemplate restTemplate;    @GetMapping("/movie/{id}")  //如果断路器打开自动调用defauleValue方法  @HystrixCommand(fallbackMethod = "defauleValue")  **public** User findByUser(@PathVariable Long id){  **return** restTemplate.getForObject("http://microservice-provider-user/simple/" + id, User.**class**);  }    **public** User defauleValue(Long id){  **return** **new** User();  }  } |

## Propagating the Security Context or using Spring Scopes

|  |
| --- |
| import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;  @HystrixCommand(fallbackMethod = "defauleValue", commandProperties = {@HystrixProperty(name = "execution.isolation.strategy", value = "SEMAPHORE")})  不设置的话回调中的方法是在另外一个线程中执行的，如果是有报错，我们采用配置，配置后是在同一个线程下执行 |

# feign中支持hystrix

## 在application中开启hystrix

|  |
| --- |
| feign.hystrix.enabled: true |

## 创建HystrixClientFallback类

|  |
| --- |
| **package** cn.yanliang.feign;  **import** org.springframework.stereotype.Component;  **import** cn.yanliang.entity.User;  /\*\*  \* HystrixClientFallback类  \* 1.这个类必须实现UserFeignClient也就是FeignClient的类  \* 2.重写请求失败后的返回  \*/  @Component  **public** **class** HystrixClientFallback **implements** UserFeignClient {  @Override  **public** User findById(Long id) {  **return** **new** User();  }  } |

## 在@FeignClient注解中添加fallback

|  |
| --- |
| package cn.yanliang.feign;  import org.springframework.cloud.netflix.feign.FeignClient;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import cn.yanliang.entity.User;  //fallback 就是这个FeignClient中接口请求失败后要处理的类  @FeignClient(name = "microservice-provider-user", fallback = HystrixClientFallback.class)  public interface UserFeignClient {  @GetMapping("/simple/{id}")  public User findById(@PathVariable("id") Long id);  } |

# feign中某一个客户端禁用hystrix

|  |
| --- |
| @Bean  @Scope("prototype")  public Feign.Builder feignBuilder() {  return Feign.builder();  } |

## 全局在feign禁用hystrix

feign.hystrix.enabled=false

# 引入网关zuul

## 添加依赖

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

## 添加注解

|  |
| --- |
| @SpringBootApplication  @EnableZuulProxy  **public** **class** ZuulServer {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(ZuulServer.**class**, args);  }  } |

在main方法上添加

这里面我们不需要添加@EnableDiscoveryClient主要原因是因为，@EnableZuulProxy是一个复合注解。

|  |
| --- |
| @EnableCircuitBreaker  @EnableDiscoveryClient  @Target(ElementType.***TYPE***)  @Retention(RetentionPolicy.***RUNTIME***)  @Import(ZuulProxyConfiguration.**class**)  **public** **@interface** EnableZuulProxy {  } |

## 使用

<http://192.168.20.180:7800/movie/1> server-id是microservice-consumer-movie-ribbon-with-hystrix

我们使用zuul后我们可以使用：<http://192.168.20.180:8040/microservice-consumer-movie-ribbon-with-hystrix/movie/1>来访问

## 自定义方向代理的路由

|  |
| --- |
| zuul:  routes:  # server-id: 自定义的路由  microservice-provider-user: /user/\*\* |

使用：<http://192.168.20.180:8040/user/simple/1>

## 过滤方向代理

|  |
| --- |
| ignoredServices: '\*'  #过滤所有  ignoredServices: microservice-provider-user, 应用2  #过滤部分应用 |

## path+serviceId 的方式来配置

|  |
| --- |
| zuul:  routes:  # abc 只要是唯一的，可以随意书写  abc:  path: /user-path/\*\*  serviceId: microservice-provider-user |

## path+url 的方式来配置 不能负载均衡

|  |
| --- |
| zuul:  routes:  # abc 只要是唯一的，可以随意书写  abc:  path: /user-path/\*\*  url: http://192.168.20.180:7900/ |

## Zuul来实现负载均衡的配置

|  |
| --- |
| zuul:  routes:  abc:  path: /user/\*\*  serviceId: microservice-provider-user  ribbon:  eureka:  enabled: **false**  microservice-provider-user: #这边是ribbon要请求的微服务的ServiceId  ribbon:  listOfServers: http://localhost:7900,http://localhost:7901 |

## 通过正则方式自动生成path

|  |
| --- |
| /\*\*  \* zuul通过正则生成path的实体类  \*/  @Bean  **public** PatternServiceRouteMapper serviceRouteMapper() {  **return** **new** PatternServiceRouteMapper(  "(?<name>^.+)-(?<version>v.+$)",  "${version}/${name}");  } |

简单介绍：现在有一个微服务的名称是[microservice-provider-user-v1](http://192.168.20.180:7900/info),我们通过Zuul放访问的路径是<http://192.168.20.180:8040/v1/microservice-provider-user/simple/1>

# Zuul实现文件上传

## 创建Controller

|  |
| --- |
| **package** cn.yanliang.controller;  **import** java.io.File;  **import** java.io.IOException;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.util.FileCopyUtils;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.RequestParam;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** org.springframework.web.multipart.MultipartFile;  @Controller  **public** **class** FileUploadController {    @RequestMapping(value = "/upload", method = RequestMethod.***POST***)  @ResponseBody  **public** String handleFileUpload(@RequestParam(value = "file", required = **true**) MultipartFile file) **throws** IOException{  **byte**[] bytes = file.getBytes();  File fileToSave = **new** File("uploadFiles/" + file.getOriginalFilename());  FileCopyUtils.*copy*(bytes, fileToSave);  **return** fileToSave.getAbsolutePath();  }  } |

## 添加文件大小的限制

|  |
| --- |
| spring:  application:  name: microservice-file-upload  http:  multipart:  max-file-size: 2000Mb # Max file size，默认1M  max-request-size: 2500Mb # Max request size，默认10M |

## 在Zuul文件大小的限制处理

|  |
| --- |
| hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 60000  ribbon:  ConnectTimeout: 3000  ReadTimeout: 60000 |

如果还是出现文件大小问题，可以在路径/zuul/\*\*实现。

如http://192.168.20.180:8040 /microservice-file-upload/upload 🡪

http://192.168.20.180:8040 /zuul/microservice-file-upload/upload 就可以跳过zuul对上传文件大小的限制

# 为路由添加Hystrix断路器

|  |
| --- |
| **package** cn.yanliang.fallback;  **import** java.io.ByteArrayInputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** org.springframework.cloud.netflix.zuul.filters.route.ZuulFallbackProvider;  **import** org.springframework.http.HttpHeaders;  **import** org.springframework.http.HttpStatus;  **import** org.springframework.http.MediaType;  **import** org.springframework.http.client.ClientHttpResponse;  **import** org.springframework.stereotype.Component;  /\*\*  \* 所有路由的hystrix 断路器打开时的回调方法  \*/  @Component  **class** MyFallbackProvider **implements** ZuulFallbackProvider {  @Override  **public** String getRoute() {  **return** "\*";  //如果只是想连接某一个项目，那就添加项目名就可以，如return “microservice-provider-user”  }  @Override  **public** ClientHttpResponse fallbackResponse() {  **return** **new** ClientHttpResponse() {  @Override  **public** HttpStatus getStatusCode() **throws** IOException {  **return** HttpStatus.***BAD\_REQUEST***;  }  @Override  **public** **int** getRawStatusCode() **throws** IOException {  **return** HttpStatus.***BAD\_REQUEST***.value();  }  @Override  **public** String getStatusText() **throws** IOException {  **return** HttpStatus.***BAD\_REQUEST***.getReasonPhrase();  }  @Override  **public** **void** close() {  }  @Override  **public** InputStream getBody() **throws** IOException {  **return** **new** ByteArrayInputStream("{'state':'error'}".getBytes());  }  @Override  **public** HttpHeaders getHeaders() {  HttpHeaders headers = **new** HttpHeaders();  headers.setContentType(MediaType.***APPLICATION\_JSON***);  **return** headers;  }  };  }  } |

# Zuul过滤器

## 添加自定义过滤器

|  |
| --- |
| package com.itmuch.cloud;  import javax.servlet.http.HttpServletRequest;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import com.netflix.zuul.ZuulFilter;  import com.netflix.zuul.context.RequestContext;  public class PreZuulFilter extends ZuulFilter {  private static final Logger LOGGER = LoggerFactory.getLogger(PreZuulFilter.class);  @Override  public boolean shouldFilter() {  return true;  }  @Override  public Object run() {  HttpServletRequest request = RequestContext.getCurrentContext().getRequest();  String host = request.getRemoteHost();  PreZuulFilter.LOGGER.info("请求的host:{}", host);  return null;  }  @Override  public String filterType() {  return "pre";  }  @Override  public int filterOrder() {  return 1;  }  } |

## 创建实体

|  |
| --- |
| package com.itmuch.cloud;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.zuul.EnableZuulProxy;  import org.springframework.context.annotation.Bean;  @SpringBootApplication  @EnableZuulProxy  public class ZuulApplication {  public static void main(String[] args) {  SpringApplication.run(ZuulApplication.class, args);  }  @Bean  public PreZuulFilter preZuulFilter() {  return new PreZuulFilter();  }  } |