### 2019.12.17 服务注册中心

1.创建eurekaserver工程，启动服务注册中心需要在Springboot工程的application类加上@EnableEurekaServer注解；

2.注册中心的配置文件：

server:

port: 端口号

eureka:

instance:

hostname: 主机名，不配置的时候根据操作系统的主机名来获取，可配置localhost

client:

registerWithEureka: flase

fetchRegisty: flase

(以上两项设置为flase来表名自己是一个注册中心)

serviceUrl:

defaultZone: <http://${eureka.instance.hostname}:${server.port}/eureka/>

### 2019.12.18 服务提供者

1.创建eurekaclient工程通过注解@EnableEurekaClient 表明自己是一个eurekaclient.

2.配置文件：

eureka:

client:

service-url:

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

server:

port: 8762

spring:

application:

name: service-hi

(spring.application.name,这个很重要，这在以后的服务与服务之间相互调用一般都 是根据这个name)

### 2019.12.22 服务消费者rest+ribbon

1.创建eurekribbon工程，引入依赖

：spring-cloud-starter-netflix-ribbon、spring-cloud-starter-openfeign

2.配置文件：

eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://localhost:8761/eureka/  
server:  
 port: 8764  
spring:  
 application:  
 name: service-ribbons

1. 在工程启动类上添加@EnableDiscoveryClient注解，并向启动类注入一个RestTemplate通过@LoadBalanced来启动负载均衡。
2. 创建service和controller来消费service-hi的服务，启动ribbon后访问：

<http://localhost:8764/controller中自定义的地址+参数名称>

1. 本人启动后有两处问题：
2. 浏览器访问报404，这个是由于我没有将启动类放在所有包的最外层；
3. 浏览器访问报500，这个是Spring Cloud注解的问题，@EnableDiscoveryClient注解不能使ribbon向注册中心注册成功，所以我们换成了@EnableEurekaClient注解，并在pom文件中引入相关依赖spring-cloud-starter-netflix-eureka-client，启动后浏览器访问成功

### 2019.12.23 服务消费者 Feign

1.服务消费者rest+ribbon这种方式有点复杂，使用Feign这种更为简化，Feign采用的是基于接口的注解，并整合了Ribbon。

2.创建eurekafeign工程，并在启动类上添加注解@EnableEurekaClient向注册中心注册，添加@EnableFeignClients注解开启Feign功能。

3.配置文件类似于rest+ribbon的方式，端口号需要改。

4.在启动类的同一层级新建API包并新建接口用来消费服务，在接口上添加注解@ FeignClient来指定调用哪个服务

例：@ FeignClient(value = “service-hi”)说明调用的是service-hi这个服务。

5.在web的controller层对外暴露一个接口通过第4步定义的Feign客户端接口来消费服务。

总结：在service-hi中可定义多个方法，通过controller定义多个URL来访问Feign客户端接口来消费服务。并且此方式不用再繁琐的配置负载均衡RestTemplate。

### 2019.12.24 断路器Hystrix

1.在ribbon中使用断路器：

a.需要在ribbon中添加依赖：

<dependency>

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

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

</dependency>

b.在启动类上添加注解@EnableHystrix开启Hystrix；

c.并在ribbon的service包下消费对应服务的方法上添加

@HystrixCommand(fallbackMethod = "熔断方法名称")注解；

d.当消费的服务启动时与之前访问无异，但当被消费的服务未开启时就会自动调用熔断方法，返回方法内的内容。

2.在feign中使用断路器：

a.Feign是自带断路器的，需要在配置文件中添加feign.hystrix.enabled=true；

b.将调用服务的API的@FeignClient注解修改为

@FeignClient(value = “service-hi”,fallback = 熔断类.class)

c.熔断类需要实现消费服务的接口，并重写方法，将对应的方法重新为调用失败是返回的内容，效果与ribbon中的相同。

注意：在熔断类上需要添加注解@Component

### 2019.12.26 路由网关zuul

1.创建eurekazuul工程，引入相关依赖。

2.在启动类上添加@EnableZuulProxy注解，开启zuul功能；

3.配置文件同之前的功能，在此之上添加：

zuul:

routes:

api-a:

path: /api-a/\*\*

serviceId: service-ribbon

api-b:

path: /api-b/\*\*

serviceId: service-feign

注：以/api-a/ 开头的请求都转发给service-ribbon服务；以/api-b/开头的请求都转发给service-feign服务；

例：<http://localhost:8769/api-a/hi?name=forezp>

1. zuul不仅能起到路由的作用，还可以作为过滤器，做一些安全验证。
2. 创建过滤类继承ZuulFilter，重写相关方法。
3. filterType：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下：
4. pre：路由之前
5. routing：路由之时
6. post： 路由之后
7. error：发送错误调用
8. filterOrder：过滤的顺序
9. shouldFilter：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。
10. run：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问。