# 1发布ego-order订单服务

## 1.1创建ego-order项目

## 1.2添加依赖

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.7.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.bjsxt.app</groupId>  <artifactId>ego-order</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>ego-order</name>  <description>http://blog.jetbrains.org.cn/</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>Greenwich.SR2</spring-cloud.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <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>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

## 1.3创建Orders订单实体类

|  |
| --- |
| **package** com.bjsxt.app.pojo;  **import** java.io.Serializable;  /\*\*\*  \* 订单实体类  \* **@author** EDZ  \*  \*/  **public** **class** Orders **implements** Serializable{  **private** Integer id;  **private** String remark;  **private** Integer total;  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getRemark() {  **return** remark;  }  **public** **void** setRemark(String remark) {  **this**.remark = remark;  }  **public** Integer getTotal() {  **return** total;  }  **public** **void** setTotal(Integer total) {  **this**.total = total;  }      } |

## 1.4创建OrdersService业务接口

|  |
| --- |
| **package** com.bjsxt.app.service;  **import** java.util.List;  **import** com.bjsxt.app.pojo.Orders;  **public** **interface** OrdersService {  /\*\*\*  \* 查询某个订单信息  \*/  **public** Orders loadOrdersService(Integer id);  /\*\*\*  \* 查询订单集合  \*/  **public** List<Orders> loadOrdersListService();  } |

## 1.5创建OrdersServiceImpl实现类

|  |
| --- |
| **package** com.bjsxt.app.service.impl;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.stereotype.Service;  **import** com.bjsxt.app.pojo.Orders;  **import** com.bjsxt.app.service.OrdersService;  @Service  **public** **class** OrdersServiceImpl **implements** OrdersService {    @Override  **public** Orders loadOrdersService(Integer id) {  // **TODO** Auto-generated method stub  Orders o=**new** Orders();  o.setId(id);  o.setRemark("周末配送....");  o.setTotal(123);  **return** o;  }  @Override  **public** List<Orders> loadOrdersListService() {  // **TODO** Auto-generated method stub  List<Orders> results=**new** ArrayList<Orders>();    Orders o=**new** Orders();  o.setId(6666);  o.setRemark("周末配送....");  o.setTotal(123);    Orders o2=**new** Orders();  o2.setId(8888);  o2.setRemark("周末配送....");  o2.setTotal(123);    results.add(o);  results.add(o2);    **return** results;  }  } |

## 1.6创建OrdersController

|  |
| --- |
| **package** com.bjsxt.app.controller;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.bjsxt.app.pojo.Orders;  **import** com.bjsxt.app.service.OrdersService;  @RestController  **public** **class** OrdersController {  @Autowired  **private** OrdersService ordersService;  /\*\*  \* 处理查询某个订单详细信息的请求  \*/  @GetMapping("/orders/{id}")  **public** Orders loadOrders(@PathVariable("id") Integer id) {  **return** ordersService.loadOrdersService(id);  }  /\*\*\*  \* 处理加载订单集合信息的请求  \*/  @GetMapping("/orders")  **public** List<Orders> loadOrdersList(){  **return** ordersService.loadOrdersListService();  }  } |

## 1.7启动发布服务

使用PostMan测试服务端。

# 2Ribbon远程服务消费介绍

## 2.1 Ribbon介绍

Ribbon 是一个基于 Http 和 TCP 的客服端负载均衡工具，它是基于 Netflix Ribbon 实现的。

它不像 spring cloud 服务注册中心、配置中心、API 网关那样独立部署，但是它几乎存在于每个 spring cloud 微服务中。包括 feign 提供的声明式服务调用也是基于该 Ribbon 实现的。

ribbon 默认提供很多种负载均衡算法，例如 轮询、随机 等等。甚至包含自定义的负载均衡算法。

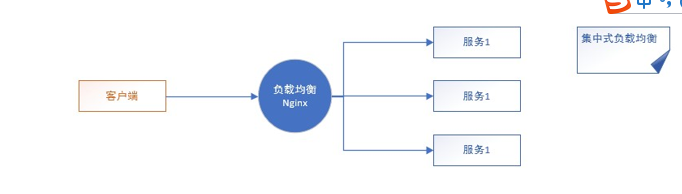
## 2.2Ribbon作用

他解决并提供了微服务的访问和负载均衡的问题。

## 2.3负载均衡解决方案

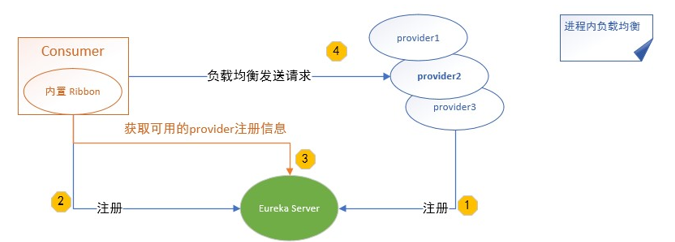
### 2.3.1集中式负载均衡

即在 consumer 和 provider 之间使用独立的负载均衡设施(可以是硬件，如 F5, 也可以是软件，如 nginx), 由该设施负责把 访问请求 通过某种策略转发至 provider。



### 2.3.2进程内负载均衡

将负载均衡逻辑集成到 consumer，consumer 从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的 provider。Ribbon 就属于后者，它只是一个类库，集成于 consumer 进程，consumer 通过它来获取到 provider 的地址。



# 3使用Ribbon消费远程服务

## 3.1创建ego-vip项目

## 3.2添加依赖

## 3.3拷贝Orders实体类

拷贝服务提供者，提供的实体类。

实际中需要将实体类单独打包

## 3.4创建EgoVipConfigs配置类

|  |
| --- |
| **package** com.bjsxt.app.config;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.web.client.RestTemplate;  @Configuration  **public** **class** EgoVipConfigs {  /\*\*\*  \* Spring容器中实例化RestTemplate对象  \* **@return**  \*/  @Bean  **public** RestTemplate restTemplate() {  **return** **new** RestTemplate();  }  } |

## 3.5创建VipSevice接口

|  |
| --- |
| **package** com.bjsxt.app.service;  **import** java.util.List;  **import** com.bjsxt.app.pojo.Orders;  **public** **interface** VipService {    /\*\*\*  \* 查询会员某个订单  \*/  **public** Orders loadVipOrdersService(Integer id);  /\*\*\*  \* 查询会员所有订单  \*/  **public** List<Orders> loadVipOrdersListService();  } |

## 3.5创建VipServiceImpl实现类

将LoadBalancerClient对象注入到VipServiceImpl实现类中。

|  |
| --- |
| **package** com.bjsxt.app.service.impl;  **import** java.util.Arrays;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.cloud.client.ServiceInstance;  **import** org.springframework.cloud.client.loadbalancer.LoadBalancerClient;  **import** org.springframework.stereotype.Service;  **import** org.springframework.web.client.RestTemplate;  **import** com.bjsxt.app.pojo.Orders;  **import** com.bjsxt.app.service.VipService;  @Service  **public** **class** VipServiceImpl **implements** VipService {  //注入Ribbon进行远程调用的负载均衡器对象  @Autowired  **private** LoadBalancerClient loadBalancerClient;  //注入RestTemplate对象  @Autowired  **private** RestTemplate restTemplate;  @Override  **public** Orders loadVipOrdersService(Integer id) {  // **TODO** Auto-generated method stub  /\*\*\*  \* 通过远程服务的名称，获得远程服务的实例对象  \* ServiceInstance对象中封装了服务的 IP，端口信息  \*/  ServiceInstance si = loadBalancerClient.choose("ego-provider");  //拼接远程访问的URL  StringBuffer url=**new** StringBuffer();  url.append("http://");  url.append(si.getHost());  url.append(":").append(si.getPort());  url.append("/orders/"+id);  //发远进行远程访问的url  System.***out***.println("url======="+url.toString());  Orders orders = restTemplate.getForObject(url.toString(), Orders.**class**);  **return** orders;  }  @Override  **public** List<Orders> loadVipOrdersListService() {  // **TODO** Auto-generated method stub  ServiceInstance si = loadBalancerClient.choose("ego-provider");  //拼接远程访问的URL  StringBuffer url=**new** StringBuffer();  url.append("http://");  url.append(si.getHost());  url.append(":").append(si.getPort());  url.append("/orders");  //发远进行远程访问的url  System.***out***.println("url======="+url.toString());  Orders[] results = restTemplate.getForObject(url.toString(), Orders[].**class**);  **return** Arrays.*asList*(results);  }  } |

## 3.6创建VipController

|  |
| --- |
| **package** com.bjsxt.app.controller;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.bjsxt.app.pojo.Orders;  **import** com.bjsxt.app.service.VipService;  @RestController  **public** **class** VipController {    //注入本地service对象  @Autowired  **private** VipService vipService;    /\*\*\*  \* 处理加载某个订单信息的请求  \*/  @GetMapping("/vorders/{id}")  **public** Orders loadOrders(@PathVariable Integer id) {  **return** vipService.loadVipOrdersService(id);  }  /\*\*\*  \* 处理加载订单集合信息的请求  \*/  @GetMapping("/vorders")  **public** List<Orders> loadOrdersList(){  **return** vipService.loadVipOrdersListService();  }  } |

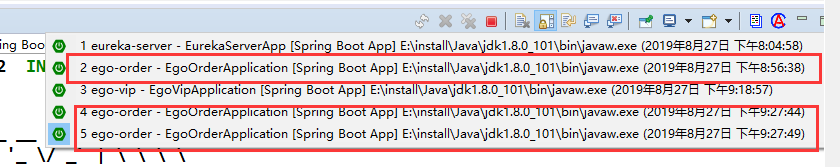
## 3.7启动测试(PostMan)

测试服务消费端。

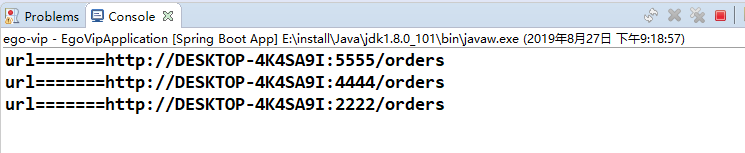
# 4测试Ribbon的负载均衡

将ego-provider服务以集群的方式启动，注意每次启动需要修改端口号。

## 4.1启动器群服务



## 4.2测试Ribbon负载均衡



# 5Ribbon常见负载均很策略

|  |  |  |  |
| --- | --- | --- | --- |
| id | 策略名称 | 策略对应的类名 | 实现原理 |
| 1 | 轮询策略（默认） | RoundRobinRule | 轮询策略表示每次都顺序取  下一个 provider，比如一共有 5 个 provider，第 1 次取第 1 个，第 2 次取第 2 个，第 3 次取第 3 个，以此类推 |
| 2 | 权重轮询策略 | WeightedResponseTimeR  ule | 1.根据每个provider的响应时间分配一个权重，响应时间越  长，权重越小，被选中的可能性越低。  2.原理：一开始为轮询策略，并开启一个计时器，每30秒收集一次每个provider的平均响应时间，当信息足够时，给每个provider 附上一个权重，并按权重随机选择 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | provider，高权越重的provider  会被高概率选中。 |
| 3 | 随机策略 | RandomRule | 从provider列表中随机选择一个  provider |
| 4 | 最少并发数策略 | BestAvailableRule | 选择正在请求中的并发数最小的provider，除非这个 provider在熔断中。 |
| 5 | 在“选定的负载均衡策略”基础上进行重试机制 | RetryRule | 1.“选定的负载均衡策略”这  个策略是轮询策略  RoundRobinRule  2.该重试策略先设定一个阈值时  间段，如果在这个阈值时间段内当选择provider不成功，则一直尝试采用“选定的负载均衡策略：轮  询策略”最后选择一个可用的 provider |
| 6 | 可用性敏感策略 | AvailabilityFilteringRule： | 过滤性能差的provider,有2  种：  第一种：过滤掉在eureka中处于一直连接失败provider 第二种：过滤掉高并发的provider |
| 7 | 区域敏感性策略 | ZoneAvoidanceRule： | 1.以一个区域为单位考察可用性，对于不可用的区域整个丢  弃，从剩下区域中选可用的 provider  2.如果这个 ip 区域内有一个或多个实例不可达或响应变慢，都会降低该 ip 区域内其他 ip 被选中的权重。 |

# 6Ribbon配置负载均衡策略

默认的负载均衡策略为轮询方式

## 6.1编码方式指定负载均衡

配置类中指定负载均衡策略

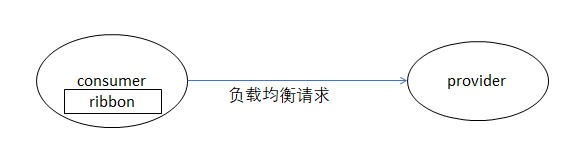
|  |
| --- |
| /\*\*  \* 配置消费端的负载均衡策略  \*/  @Bean  **public** RandomRule randomRule() {  **return** **new** RandomRule();  } |

## 6.2配置方式指定负载均衡

|  |
| --- |
| #配置消费端负载均衡策略  ego-provider:  ribbon:  NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule |

# 7Ribbon点对点直连

在特殊情况下(比如开发测试阶段)，不需要经过Eureka注册服务来访问Provider，可以利用Ribbon手动来指定一个或者多个Provider服务来实现负载均衡访问。结构如下图：



## 7.1创建ego-vip-p2p项目

## 7.2配置application.yml

|  |
| --- |
| server:  port: 3333    spring:  application:  name: ego-consumer  eureka:  client:  register-with-eureka: **false** #是否将当前应用注册到注册中心  fetch-registry: **false** #当前项目是否到注册中心中拉去服务列表  #service-url: #给Map集合属性中赋值  #defaultZone: http://localhost:1111/eureka  #禁用eureka注册中心  ribbon:  eureka:  enabled: **false**  #配置具体需要访问的服务列表  ego-provider:  ribbon:  listOfServers: http://localhost:4444,http://localhost:5555 |

## 7.3启动测试(PostMan)

|  |
| --- |
| **package** com.bjsxt.app;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  //@EnableEurekaClient  **public** **class** EgoVipP2pApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(EgoVipP2pApplication.**class**, args);  }  } |

# 8使用RestTemplate访问微服务