**【Spring Cloud OpenFeign】**

**【主要内容】**

1. OpenFeign简介
2. OpenFeign快速入门
3. OpenFeign参数处理
4. OpenFeign源码分析

**【学习目标】**

|  |  |
| --- | --- |
| 知识点 | 要求 |
| OpenFeign简介 | 了解 |
| OpenFeign快速入门 | 掌握 |
| OpenFeign参数处理 | 掌握 |
| OpenFeign源码分析 | 了解 |

# 说在前面

上一节我们讲到Ribbon做了负载均衡，用Eureka-Client来做服务发现，通过RestTemplate来完成服务调用，但是这都不是我们的终极方案，终极方案是使用OpenFeign

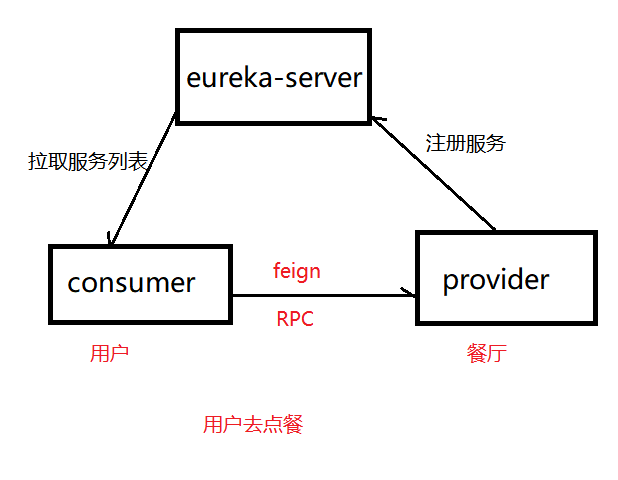
# OpenFeign简介

<https://docs.spring.io/spring-cloud-openfeign/docs/2.2.4.RELEASE/reference/html/#spring-cloud-feign>

Feign是**声明性**Web服务**客户端**。它使编写Web服务客户端更加容易。**要使用Feign，请创建一个接口并对其进行注解**。它具有可插入注解支持，包括Feign注解和JAX-RS注解。Feign还支持可插拔编码器和解码器。Spring Cloud添加了对Spring MVC注解的支持，并支持使用HttpMessageConverters，Spring Web中默认使用的注解。Spring Cloud集成了Ribbon和Eureka以及Spring Cloud LoadBalancer，以**在使用Feign时提供负载平衡的http客户端**。

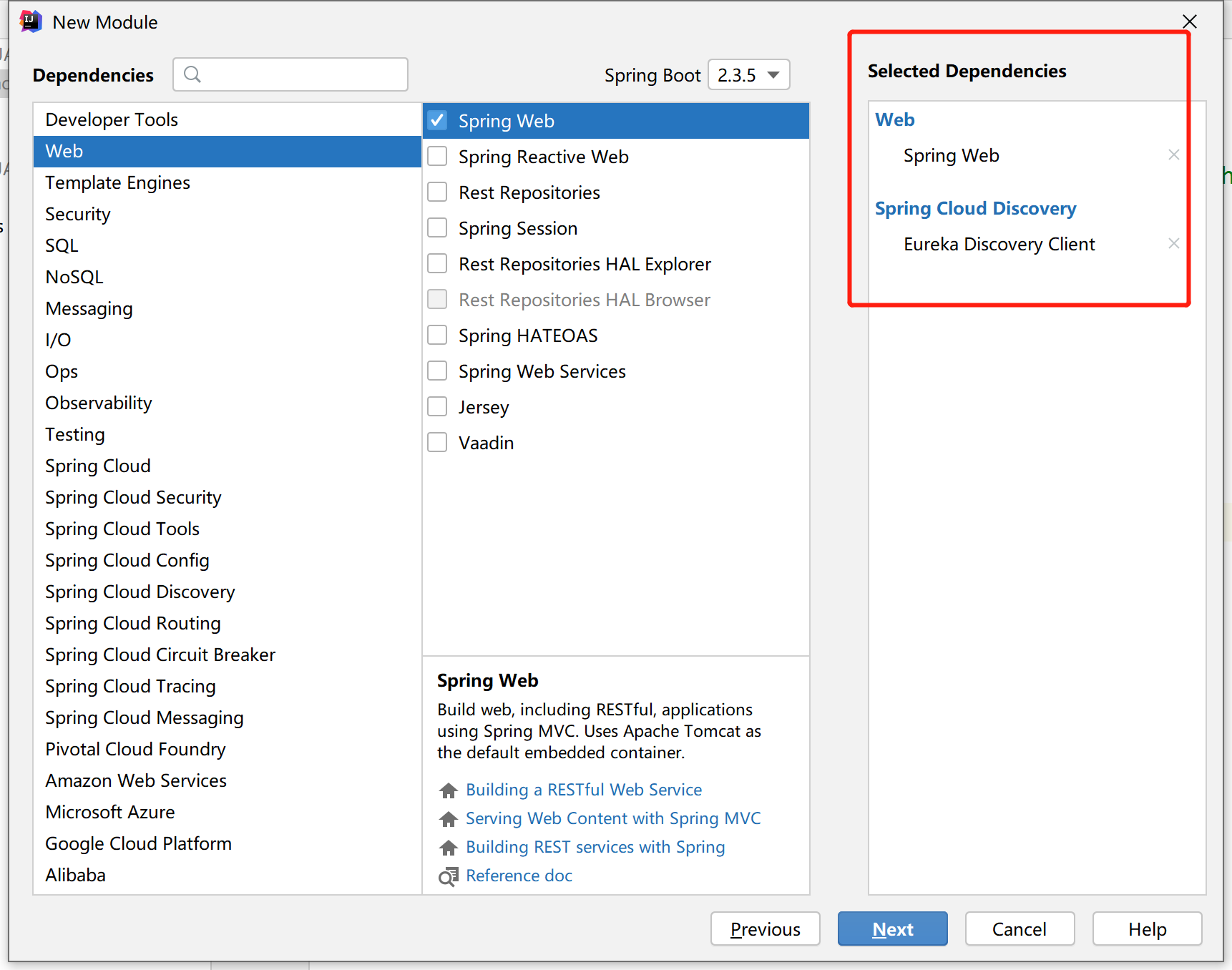
# OpenFeign快速入门

## 本次调用的设计图



## 启动一个eureka-server服务，这里不重复演示，参考eureka文档

## 先创建provider-order-service，选择依赖



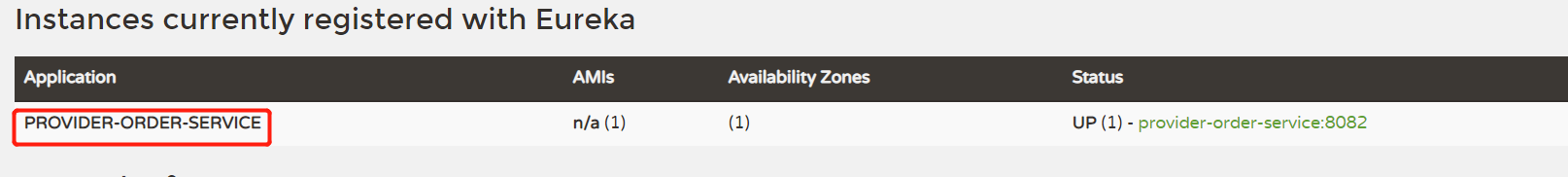
## provider-order-service修改配置文件

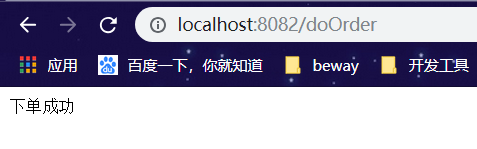
|  |
| --- |
| server:  port: 8081 spring:  application:  name: consumer-user-service eureka:  client:  service-url:  defaultZone: http://localhost:8761/eureka  instance:  instance-id: ${spring.application.name}:${server.port}  prefer-ip-address: true |

## provider-order-service修改启动类增加一个访问接口

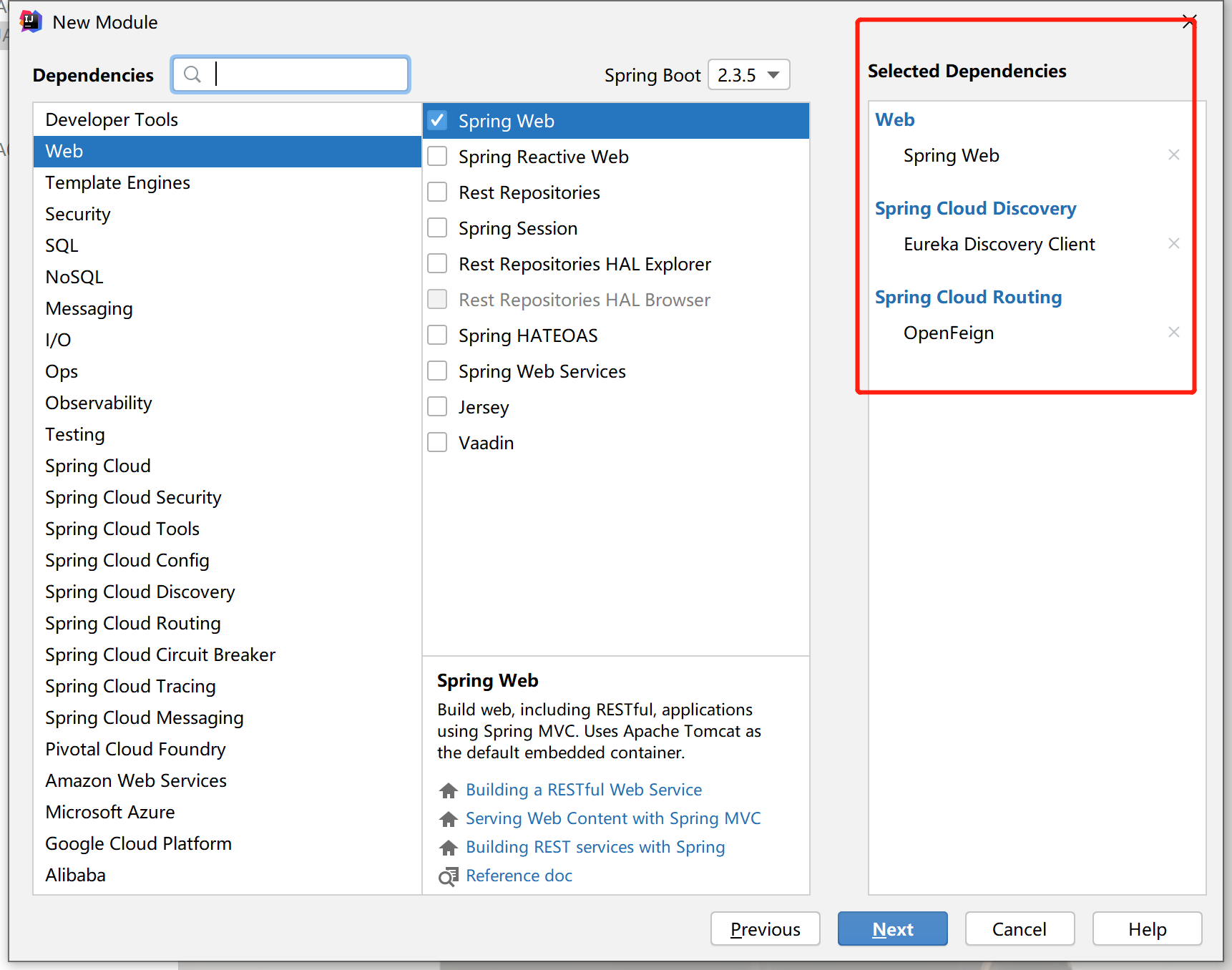
|  |
| --- |
| package com.sxt.controller;  import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;  */\*\*  \* @Author: 武汉尚学堂  \*/* @RestController public class OrderController {   */\*\*  \* 订单服务下单接口  \*  \* @return  \*/* @GetMapping("doOrder")  public String doOrder() {  System.*out*.println("有用户来下单了");  return "下单成功";  } } |

## provider-order-service启动测试访问





## 再创建consumer-user-service，选择依赖



## consumer-user-service修改配置文件

|  |
| --- |
| server:  port: 8081 spring:  application:  name: consumer-user-service eureka:  client:  service-url:  defaultZone: http://localhost:8761/eureka  instance:  instance-id: ${spring.application.name}:${server.port}  prefer-ip-address: true |

## consumer-user-service创建一个接口（重点）

|  |
| --- |
| package com.sxt.feign;  import org.springframework.cloud.openfeign.FeignClient; import org.springframework.web.bind.annotation.GetMapping;  */\*\*  \* @Author: 武汉尚学堂  \*   \* @FeignClient声明是feign的调用   \* value = "provider-order-service" value后面的值必须和提供者的服务名一致  \*/* @FeignClient(value = "provider-order-service") public interface UserOrderFeign {   */\*\*  \* 描述: 下单的方法 这里的路径必须和提供者的路径一致  \*  \* @param :  \* @return java.lang.String  \*/* @GetMapping("doOrder")  String doOrder();  } |

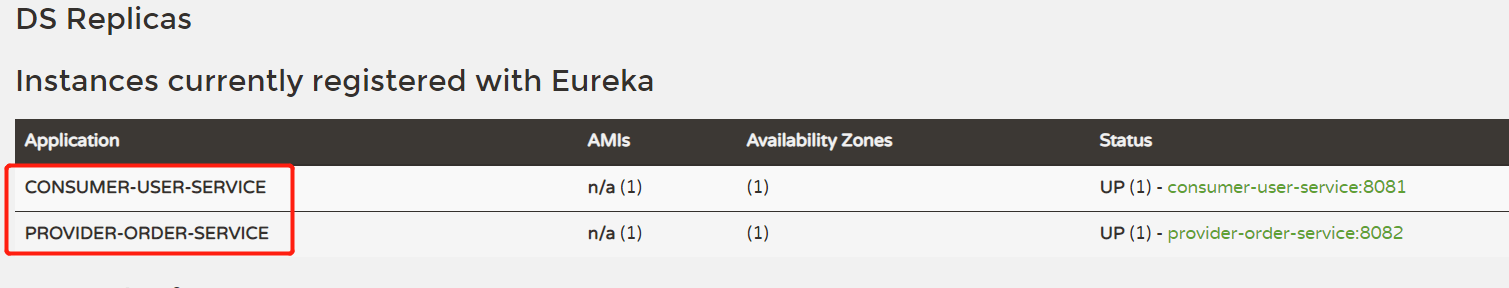
## consumer-user-service创建controller

|  |
| --- |
| package com.sxt.controller;  import com.sxt.feign.UserOrderFeign; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;  */\*\*  \* @Author: 武汉尚学堂  \*/* @RestController public class UserController {   @Autowired  private UserOrderFeign userOrderFeign;   */\*\*  \* 用户远程调用下单的接口  \*  \* @return  \*/* @GetMapping("userDoOrder")  public String userDoOrder() {  String result = userOrderFeign.doOrder();  System.*out*.println(result);  return result;  } } |

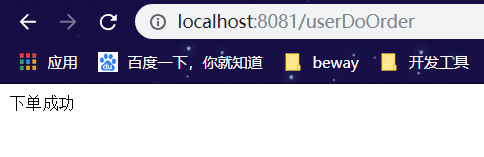
## consumer-user-service修改启动类

|  |
| --- |
| package com.sxt;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.netflix.eureka.EnableEurekaClient; import org.springframework.cloud.openfeign.EnableFeignClients;  @SpringBootApplication @EnableEurekaClient @EnableFeignClients *//标记feign的客户端* public class ConsumerUserServiceApplication {   public static void main(String[] args) {  SpringApplication.*run*(ConsumerUserServiceApplication.class, args);  } } |

## 启动调用测试



访问：<http://localhost:8081/userDoOrder>

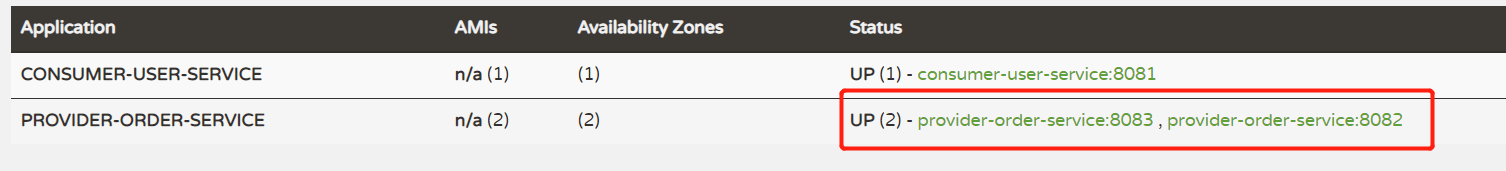


## 本次调用总结

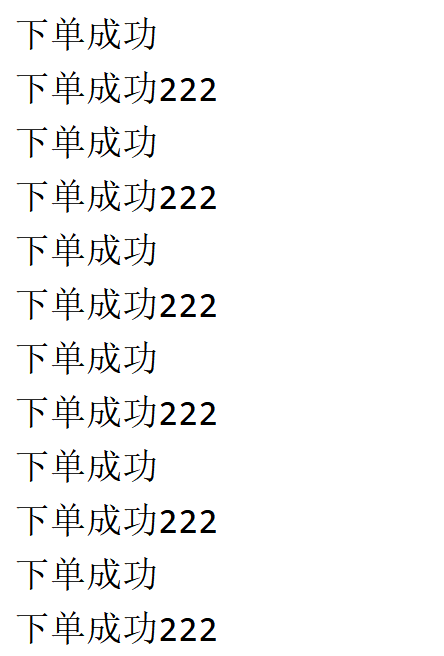
**consumer-user-service---》 /userDoOrder ---》通过feign调用 /doOrder ---》provider-order-service下单成功**

## 测试feign调用的负载均衡

启动多台provider-order-service:



测试访问:



## 调用超时设置

**因为ribbon默认调用超时时长为1s，可以修改，超时调整 可以查看DefaultClientConfigImpl**

|  |
| --- |
| ribbon: *#feign默认调用1s超时* ReadTimeout: 5000 *#修改调用时长为5s* ConnectTimeout: 5000 *#修改连接时长为5s* |

# OpenFeign调用参数处理

## 说在前面

**Feign传参确保消费者和提供者的参数列表一致 包括返回值**

1. **通过URL传参数，GET请求，参数列表使用@PathVariable（“”）**
2. **如果是GET请求，每个参数必须加@RequestParam（“”）**
3. **如果是POST请求，而且是对象集合等参数，必须加@Requestbody**

## 修改provider-order-service

### 创建BaseResult类

|  |
| --- |
| public class BaseResult implements Serializable {   private Integer code;  private String msg;  private Object data;   public static BaseResult success(Integer code, String msg, Object data) {  BaseResult baseResult = new BaseResult();  baseResult.setCode(code);  baseResult.setData(data);  baseResult.setMsg(msg);  return baseResult;  }  } |

### 创建Order类

|  |
| --- |
| public class Order implements Serializable {   private String orderSn;  private String orderName;  private String orderDetail;  private Date orderTime;  private String userId;  } |

### 创建TestParamController类

|  |
| --- |
| package com.sxt.controller;  import com.sxt.domain.Order; import com.sxt.model.BaseResult; import org.springframework.web.bind.annotation.\*;  @RestController public class TestParamController {   */\*\*  \* 测试单个参数  \*  \* @param name  \* @return  \*/* @GetMapping("testOneParam")  public BaseResult oneParam(@RequestParam("name") String name) {  System.*out*.println(name);  return BaseResult.*success*(200, "成功", "ok");  }   */\*\*  \* 测试两个参数  \*  \* @param name  \* @param age  \* @return  \*/* @PostMapping("testTwoParam")  public BaseResult twoParam(@RequestParam("name") String name, @RequestParam("age") Integer age) {  System.*out*.println(name + ":" + age);  return BaseResult.*success*(200, "ok", "ok");  }   */\*\*  \* 测试一个对象的传参  \*  \* @param order  \* @return  \*/* @PostMapping("testObjectParam")  public BaseResult objectParam(@RequestBody Order order) {  System.*out*.println(order);  return BaseResult.*success*(200, "ok", order);  }   */\*\*  \* 测试一个对象 一个参数  \*  \* @param order  \* @param name  \* @return  \*/* @PostMapping("testOneObjectOneParam")  public BaseResult oneObjectOneParam(@RequestBody Order order, @RequestParam String name) {  System.*out*.println(order);  System.*out*.println(name);  return BaseResult.*success*(200, "ok", order);  }   */\*\*  \* 测试url传参  \*  \* @param id  \* @return  \*/* @GetMapping("testUrlParam/{id}")  public BaseResult testUrlParam(@PathVariable("id") Integer id) {  System.*out*.println(id);  return BaseResult.*success*(200, "ok", id);  } } |

## 修改consumer-user-service

### 将Order类和BaseResult类拷贝过来，后面会抽到公共模块里

### 修改UserOrderFeign接口

|  |
| --- |
| package com.sxt.feign;  import com.sxt.domain.Order; import com.sxt.model.BaseResult; import org.springframework.cloud.openfeign.FeignClient; import org.springframework.web.bind.annotation.\*;  */\*\*  \* @Author: 武汉尚学堂  \*  \* @FeignClient声明是feign的调用  \* value = "provider-order-service" value后面的值必须和提供者的服务名一致  \*/* @FeignClient(value = "provider-order-service") public interface UserOrderFeign {   */\*\*  \* 远程调用下单的方法  \*  \* @return  \*/* @RequestMapping("doOrder")  String doOrder();   */\*\*  \* 测试单个参数  \*  \* @param name  \* @return  \*/* @GetMapping("testOneParam")  public BaseResult oneParam(@RequestParam("name") String name);    */\*\*  \* 测试两个参数  \*  \* @param name  \* @param age  \* @return  \*/* @PostMapping("testTwoParam")  public BaseResult twoParam(@RequestParam("name") String name, @RequestParam("age") Integer age);   */\*\*  \* 测试一个对象的传参  \*  \* @param order  \* @return  \*/* @PostMapping("testObjectParam")  public BaseResult objectParam(@RequestBody Order order);   */\*\*  \* 测试一个对象 一个参数  \*  \* @param order  \* @param name  \* @return  \*/* @PostMapping("testOneObjectOneParam")  public BaseResult oneObjectOneParam(@RequestBody Order order, @RequestParam String name);   */\*\*  \* 测试url传参  \*  \* @param id  \* @return  \*/* @GetMapping("testUrlParam/{id}")  public BaseResult testUrlParam(@PathVariable("id") Integer id); } |

### 创建TestController类

|  |
| --- |
| package com.sxt.controller;  import com.sxt.domain.Order; import com.sxt.feign.UserOrderFeign; import com.sxt.model.BaseResult; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;  import java.util.Date;  @RestController public class TestController {   @Autowired  private UserOrderFeign userOrderFeign;   @RequestMapping("testFeignParam")  public String testFeignParam() {  *//测试一个参数* BaseResult result1 = userOrderFeign.oneParam("sxt");  System.*out*.println(result1);  System.*out*.println("--------------------------------------------------");  *//测试多个参数* BaseResult result2 = userOrderFeign.twoParam("sxt", 666);  System.*out*.println(result2);  System.*out*.println("--------------------------------------------------");  *//测试一个对象* Order order = new Order("111", "牛排", "一份牛排256g", new Date(), "159357");  BaseResult result3 = userOrderFeign.objectParam(order);  System.*out*.println(result3);  System.*out*.println("--------------------------------------------------");  *//测试url传参* BaseResult result4 = userOrderFeign.testUrlParam(999);  System.*out*.println(result4);  System.*out*.println("--------------------------------------------------");  *//测试一个对象 一个参数* BaseResult result5 = userOrderFeign.oneObjectOneParam(order, "sxtsxt");  System.*out*.println(result5);  System.*out*.println("--------------------------------------------------");  return "ok";  } } |

### 测试调用

访问： [http://localhost:8081/testFeignParam](http://localhost:8081/user/testFeignParam)

### 时间日期参数问题

使用feign远程调用时，传递Date类型，接收方的时间会相差14个小时，是因为时区造成的

处理方案：

1. 使用字符串传递参数，接收方转换成时间类型（推荐使用）
2. 使用JDK8的LocalDate(日期) 或 LocalDateTime(日期和时间，接收方只有秒，没有毫秒)
3. 自定义转换方法

# OpenFeign源码分析

## OpenFeign的原理是什么？

**根据上面的案例，我们知道feign是接口调用，接口如果想做事，必须要有实现类**

**可是我们并没有写实现类，只是加了一个@FeignClient(value=”xxx-service”)的注解**

**所以我们猜测feign帮我们创建了代理对象，然后完成真实的调用。**

1. **给接口创建代理对象**
2. **代理对象执行进入invoke方法**
3. **在invoke方法里面做远程调用**

**具体我们这次的流程：**

1. **扫描注解得到要调用的服务名称和url**

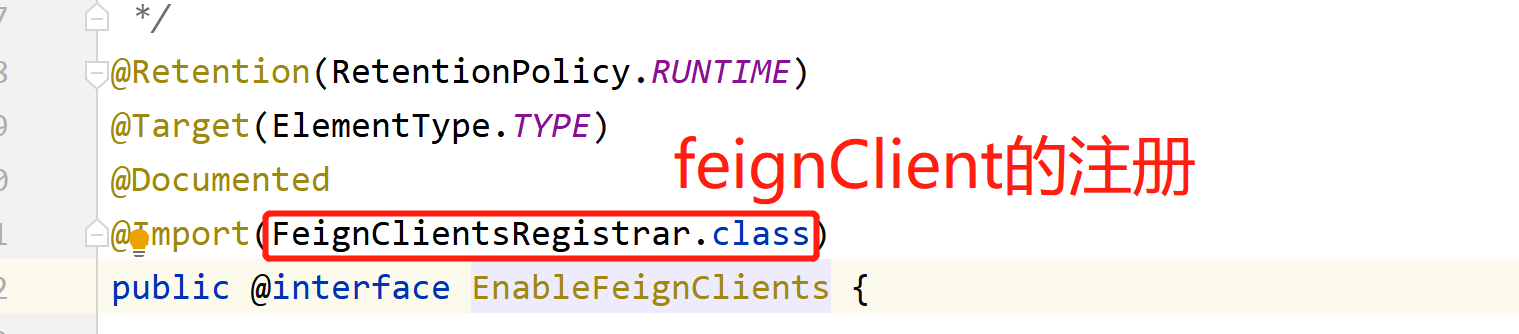


1. **拿到provider-order-service/doOrder，通过ribbon的负载均衡拿到一个服务，provider-order-service/doOrder---》http://ip:port/doOrder**
2. **发起请求，远程调用**

## 看看OpenFeign的内部是如何实现这些的

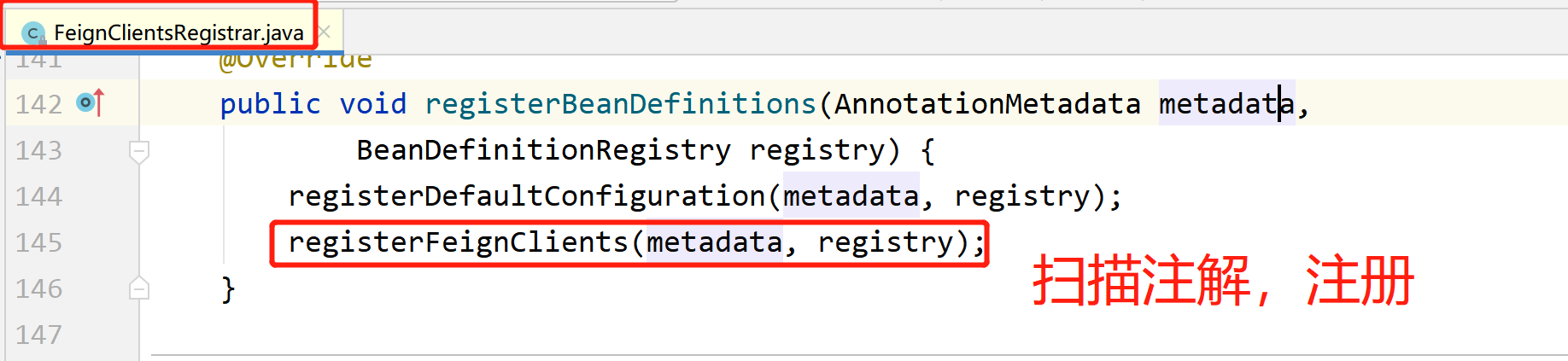
### 如何扫描注解@FeignClient

**查看启动类的@EnableFeignClients**



**进入FeignClientsRegistrar这个类 去查看里面的东西**



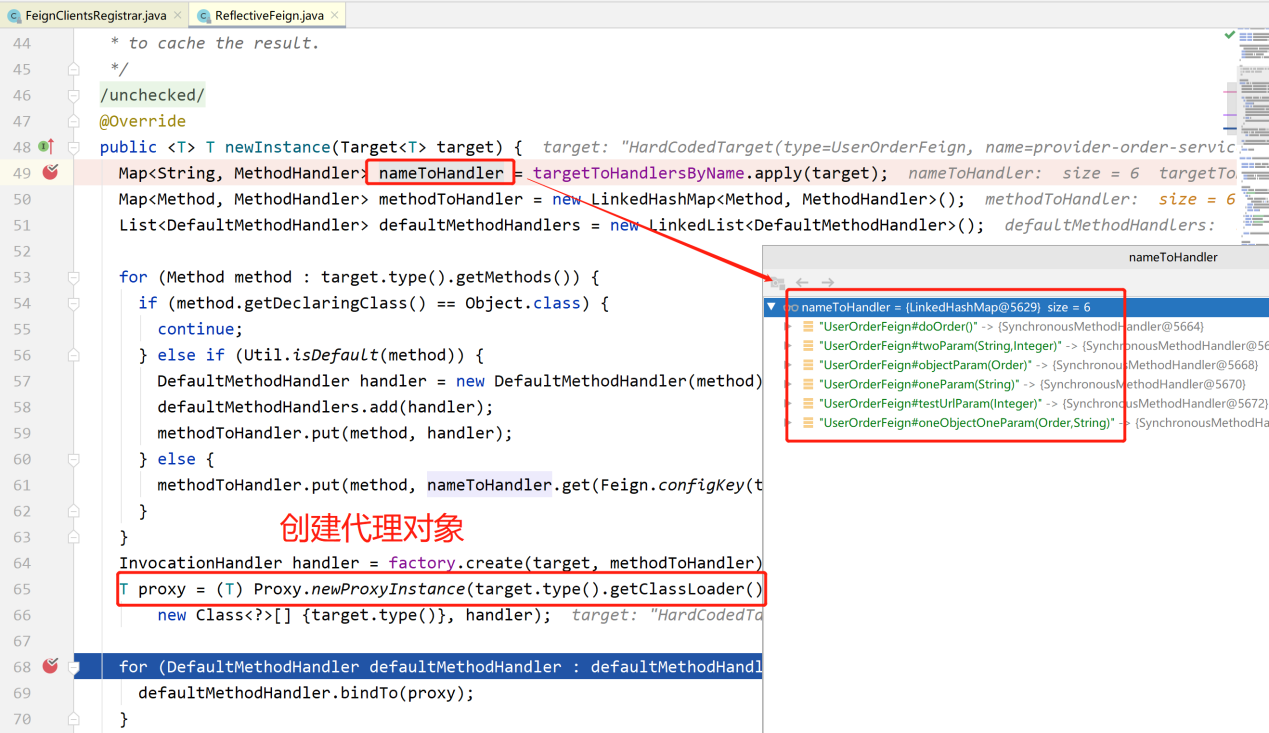


**真正的扫描拿到注解和服务名称**

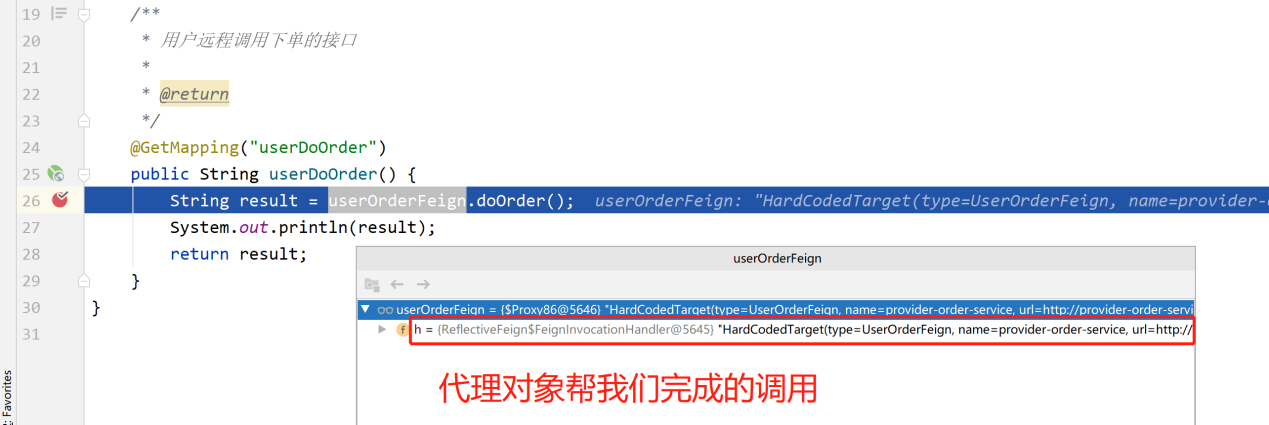


### 如何创建代理对象去执行调用？

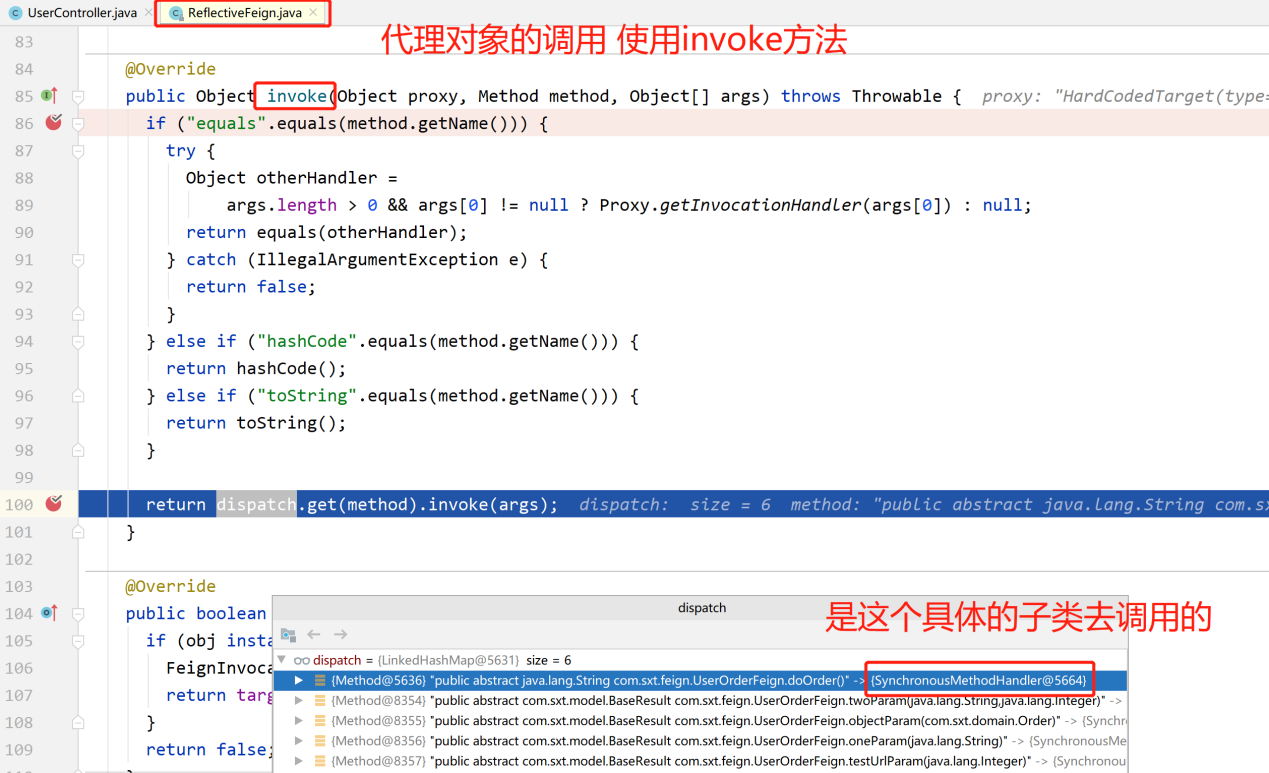
**当我们启动时，在ReflectiveFeign类的newInstance方法，给接口创建了代理对象**



**当我们执行调用的时候，打个断点去查看**



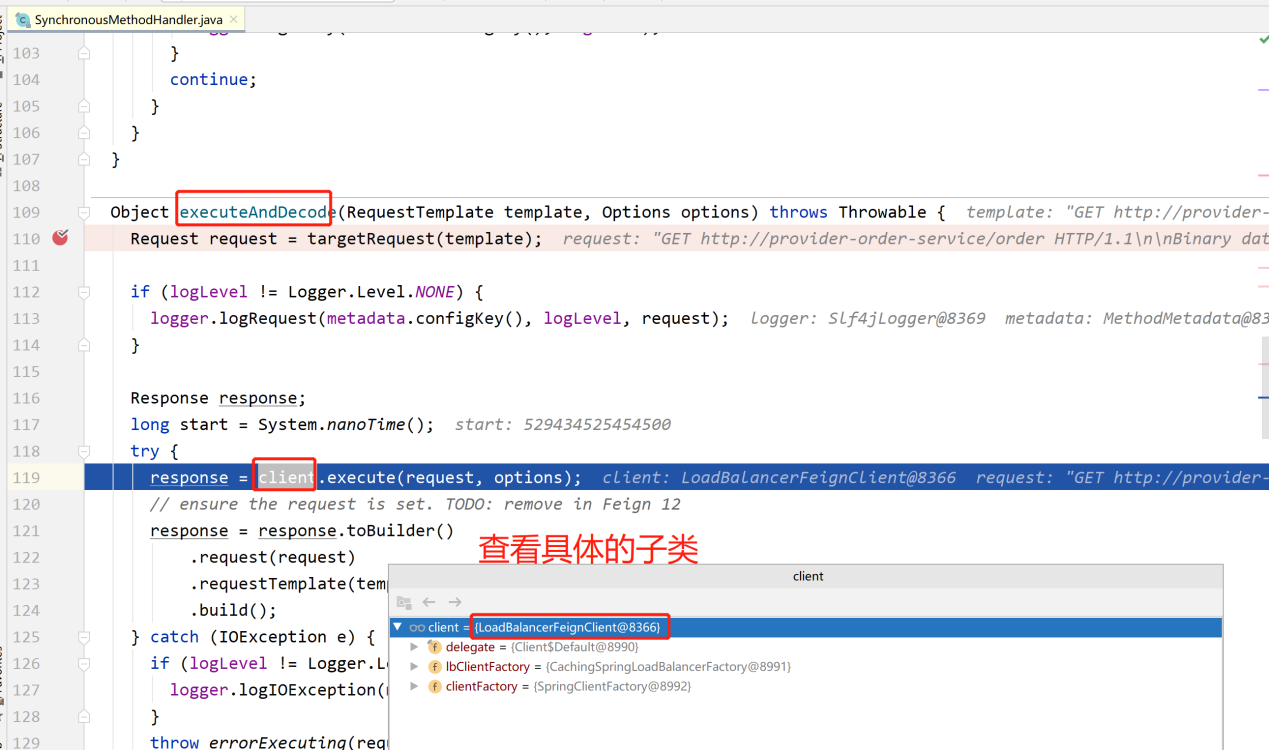
**ReflectiveFeign类中的invoke方法帮我们完成调用**



**SynchronousMethodHandler的invoke中给每一个请求创建了一个requestTemplate对象，去执行请求**



**executeAndDecode**

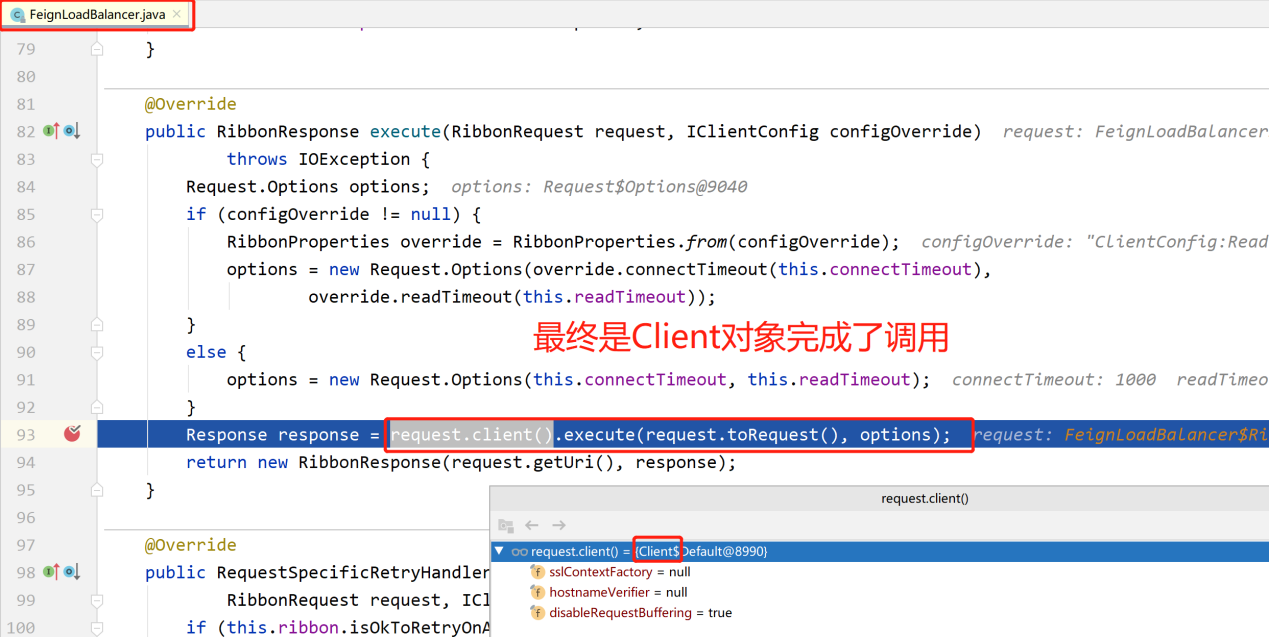


**我们去看LoadBalancerFeignClient的execute方法**



**executeWithLoadBalancer继续往下看**







# OpenFeign总结

**OpenFeign主要基于接口和注解实现了远程调用**

**源码总结：面试**

1. **OpenFeign用过吗？它是如何运作的？**

**在主启动类上加上@EnableFeignClients注解后，启动会进行包扫描，把所有加了@FeignClient(value=”xxx-service”)注解的接口进行创建代理对象通过代理对象，使用ribbon做了负载均衡和远程调用**

1. **如何创建的代理对象？**

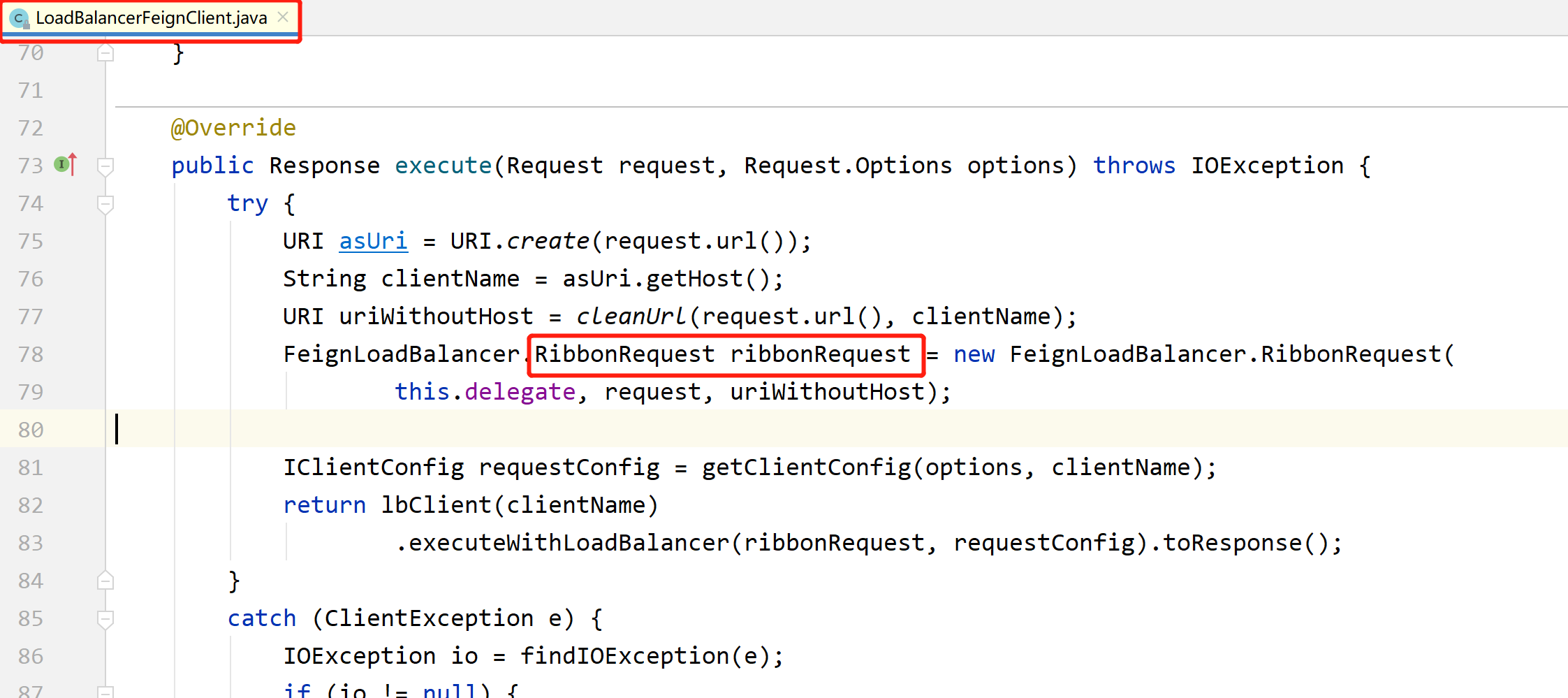
**当项目在启动时，先扫描，然后拿到标记了@FeignClient注解的接口信息，由ReflectiveFeign类的newInstance方法创建了代理对象**

1. **OpenFeign到底是用什么做的远程调用？**

**使用的是HttpURLConnection （java.net）**

1. **OpenFeign怎么和ribbon整合的？**

**在代理对象执行调用的时候**



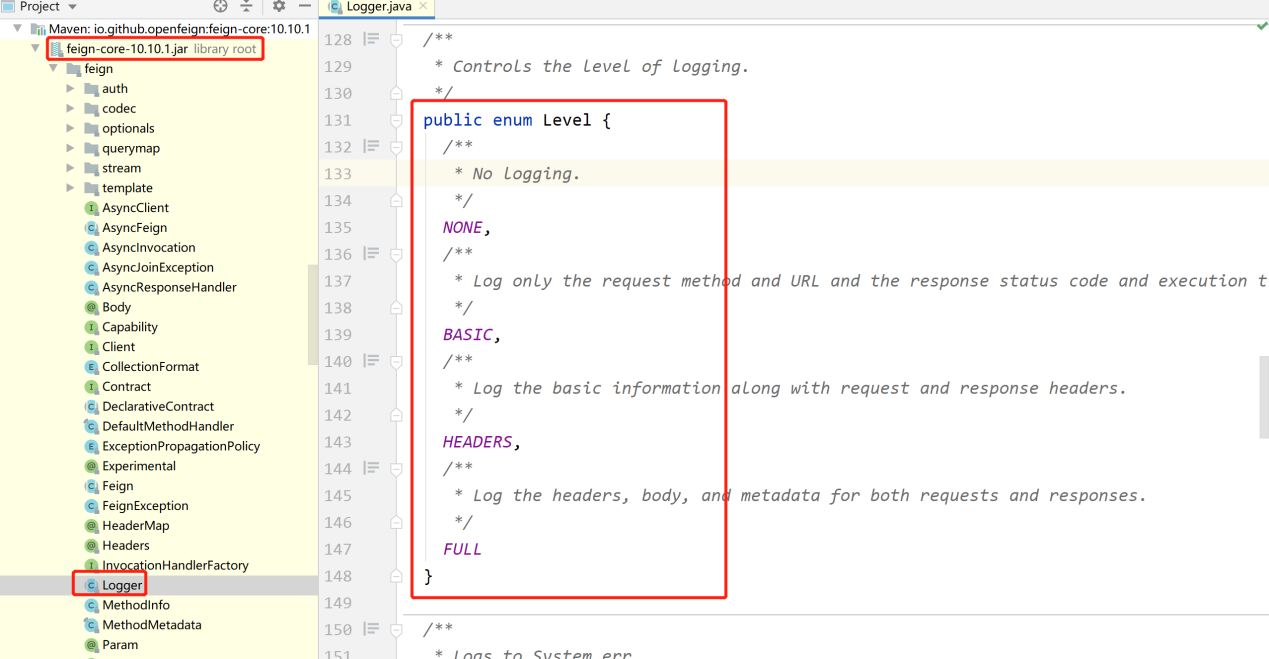
# OpenFeign其他

## OpenFeign的日志功能

从前面的测试中我们可以看出，没有任何关于远程调用的日志输出，如请头，参数

Feign提供了日志打印功能，我们可以通过配置来调整日志级别，从而揭开Feign中Http请求的所有细节

### OpenFeign的日志级别



NONE 默认的，不显示日志

BASE 仅记录请求方法，URL ，响应状态码及执行时间

HEADERS 在BASE之上增加了请求和响应头的信息

FULL 在HEADERS之上增加了请求和响应的正文及无数据

### 创建配置类

|  |
| --- |
| package com.sxt.config;  import feign.Logger; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  @Configuration public class FeignConfig {   @Bean  Logger.Level feignLogger() {  return Logger.Level.*FULL*;  }  } |

### 修改配置文件

|  |
| --- |
| logging:  level:  com.sxt.feign.UserOrderFeign: debug |

### 调用测试

