【Spring Cloud OpenFeign】

1.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**

2.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(注解)**Web 服务**客户端**。它使编写 Web 服务客户端更加容易。**要使用 Feign，**
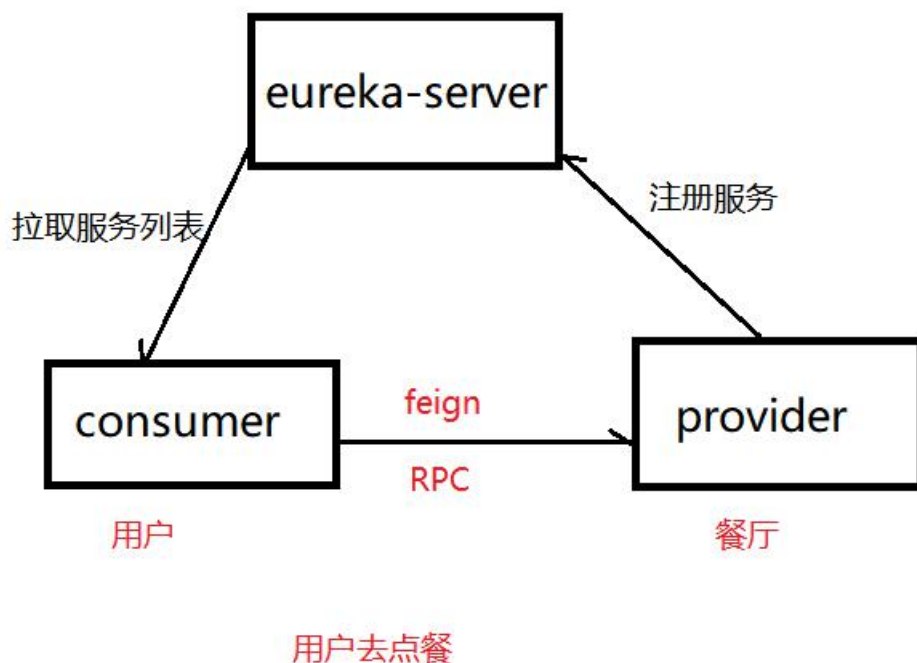
请创建一个接口并对其进行注解。它具有可插入注解支持，包括 Feign 注解和 JAX-RS 注解。

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。

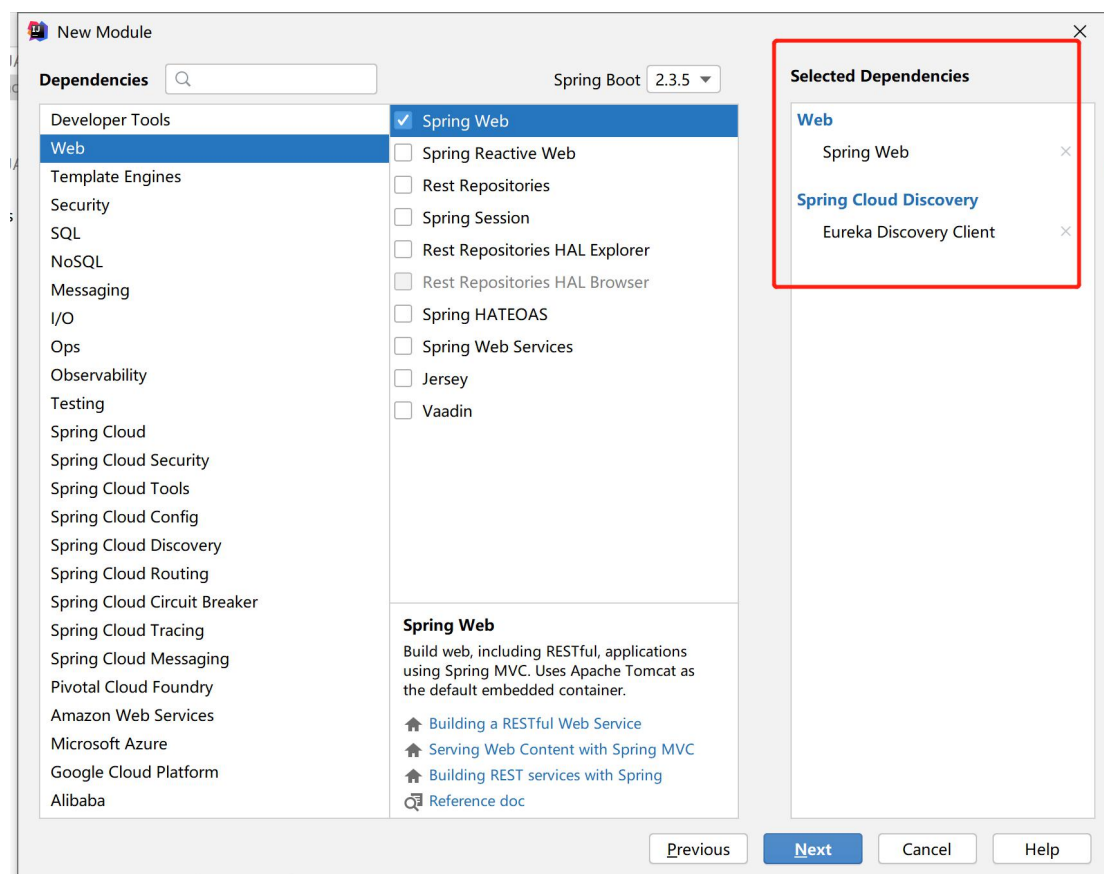Feign 是一个**远程调用**的组件（接口，注解）http 调用的

Feign 集成了 ribbon   ribbon 里面集成了 eureka

3.OpenFeign 快速入门

3.1 本次调用的设计图

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

3.3 先创建 provider-order-service，选择依赖

3.4 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
```

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

```java
package com.bjpowernode.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @Author: 动力节点
 */
@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";

    }
}
```
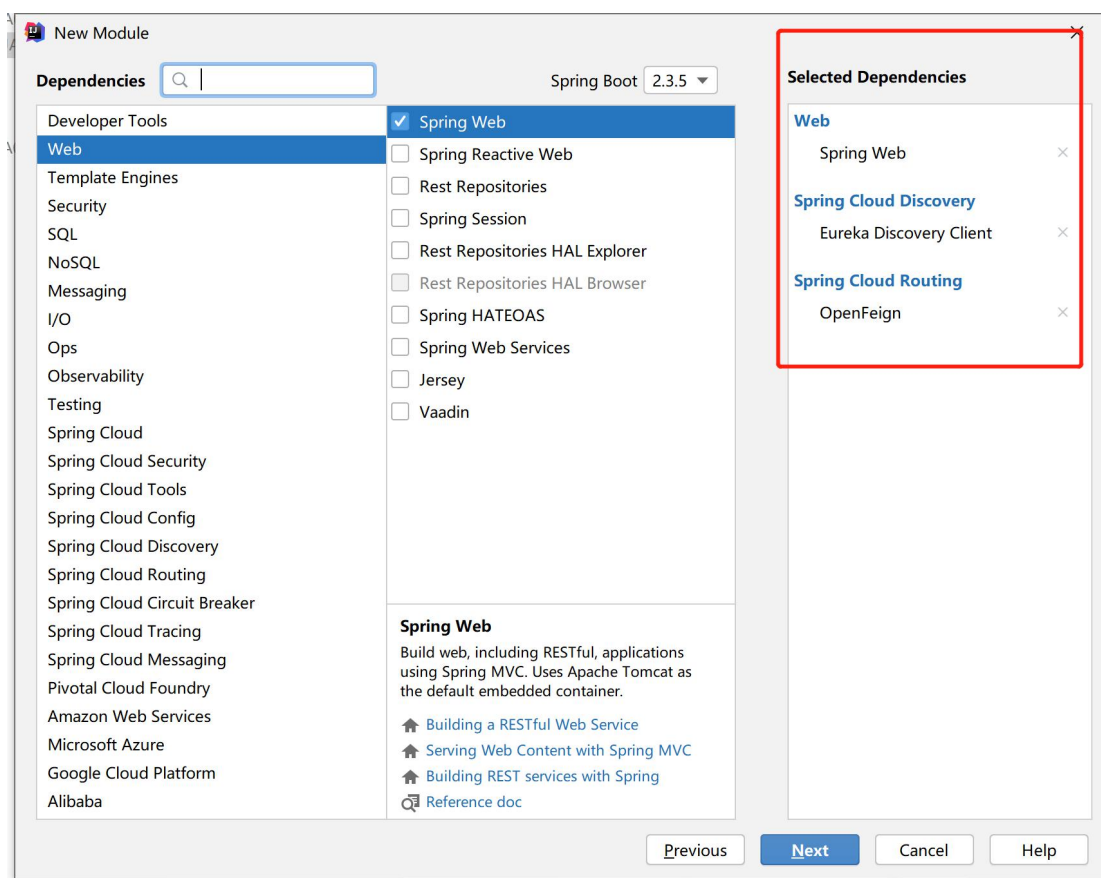
3.6 provider-order-service 启动测试访问

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PROVIDER-ORDER-SERVICE	n/a (1)	(1)	UP (1) - provider-order-service:8082

localhost:8082/doOrder

下单成功

3.7 再创建 consumer-user-service，选择依赖

3.8 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
```

3.9 consumer-user-service 创建一个接口（重点）

```java
package com.bjpowernode.feign;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * @Author: 动力节点
 *
 * @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();

}
```

3.10 consumer-user-service 创建 controller

```java
package com.bjpowernode.controller;

import com.bjpowernode.feign.UserOrderFeign;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @Author: 动力节点
 */
@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;
    }
}
```

3.11 consumer-user-service 修改启动类

```java
package com.bjpowernode;

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);
    }
}
```

3.12 启动调用测试

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CONSUMER-USER-SERVICE	n/a (1)	(1)	UP (1) - consumer-user-service:8081
PROVIDER-ORDER-SERVICE	n/a (1)	(1)	UP (1) - provider-order-service:8082

访问：http://localhost:8081/userDoOrder

← → C ① localhost:8081/userDoOrder

⠿ 应用 百度一下，你就知道 beway 开发工具

下单成功

3.13 本次调用总结

consumer-user-service---》 /userDoOrder ---》通过 feign 调用 /doOrder ---》

provider-order-service **下单成功**

3.14 测试 feign 调用的负载均衡

启动多台 provider-order-service：

Application	AMIs	Availability Zones	Status
CONSUMER-USER-SERVICE	n/a (1)	(1)	UP (1) - consumer-user-service:8081
PROVIDER-ORDER-SERVICE	n/a (2)	(2)	UP (2) - provider-order-service:8083 , provider-order-service:8082

测试访问：

下单成功
下单成功222
下单成功
下单成功222
下单成功
下单成功222
下单成功
下单成功222
下单成功
下单成功222
下单成功
下单成功222

3.15 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
```

4.OpenFeign 调用参数处理（开发重点）

4.1 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 方法签名要一致

1．通过 URL 传参数，GET 请求，参数列表使用@PathVariable（""）

2．如果是 GET 请求，每个基本参数必须加@RequestParam（""）

3．如果是 POST 请求，而且是对象集合等参数，必须加@Requestbody 或者@RequestParam

4.2 修改 provider-order-service

4.2.1 创建 BaseResult 类

```java
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;
    }
}
```

4.2.2 创建 Order 类

```java
public class Order implements Serializable {

    private String orderSn;
    private String orderName;
    private String orderDetail;
```

```
    private Date orderTime;
    private String userId;
}
```

4.2.3 创建 TestParamController 类

```java
package com.bjpowernode.controller;

import com.bjpowernode.domain.Order;
import com.bjpowernode.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,
```

```java
@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);
    }
}
```

4.3 修改 consumer-user-service

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

4.3.2 修改 UserOrderFeign 接口

```
package com.bjpowernode.feign;

import com.bjpowernode.domain.Order;
import com.bjpowernode.model.BaseResult;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.*;

/**
 * @Author: 动力节点
 *
 * @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
     *
```

```java
 * @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);
```

```java
    /**
     * 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);
}
```

4.3.3 创建 TestController 类

```java
package com.bjpowernode.controller;

import com.bjpowernode.domain.Order;
import com.bjpowernode.feign.UserOrderFeign;
import com.bjpowernode.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 {
```

```java
    @Autowired
    private UserOrderFeign userOrderFeign;

    @RequestMapping("testFeignParam")
    public String testFeignParam() {
        //测试一个参数
        BaseResult result1 = userOrderFeign.oneParam("bjpowernode");
        System.out.println(result1);
        System.out.println("--------------------------------------------------");
        //测试多个参数
        BaseResult result2 = userOrderFeign.twoParam("bjpowernode", 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,
"bjpowernodebjpowernode");
        System.out.println(result5);
        System.out.println("--------------------------------------------------");
        return "ok";
    }
}
```

4.3.4 测试调用

访问: http://localhost:8081/testFeignParam

4.3.5 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）不要单独传递时间

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

传参总结：

get 请求只用来传递基本参数 而且加注解@RequestParam

post 请求用来传递对象参数 并且加注解@RequestBody

5.OpenFeign 源码分析

（学习别人的思想，可以找 bug，优化你的代码，提高代码的健壮性）

看源码之前要先大致猜想一下 他是怎么实现的？ （先使用在分析）

5.1 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。

动态代理 1jdk （invoke） 2cglib 子类继承的

1．给接口创建代理对象（启动扫描）

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：

A．扫描注解得到要调用的服务名称和 url

```
    */
@FeignClient(value = "provider-order-service")    要调用的服务名称
public interface UserOrderFeign {

    /**
     * 远程调用下单的方法
     *
     * @return
     */
    @RequestMapping("order")    url拼接
    String doOrder();

    /**
```

B．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

C．发起请求，远程调用

5.2 看看 OpenFeign 的内部是如何实现这些的

5.2.1 如何扫描注解@FeignClient

查看启动类的@EnableFeignClients

```
       */
      @Retention(RetentionPolicy.RUNTIME)
      @Target(ElementType.TYPE)
      @Documented
      @Import(FeignClientsRegistrar.class)
      public @interface EnableFeignClients {
```

feignClient的注册

进入 FeignClientsRegistrar 这个类 去查看里面的东西

```
       * @author Gang Li
       * @author Michal Domagala
       * @author Marcin Grzejszczak
       * @author Olga Maciaszek-Sh
       */
      class FeignClientsRegistrar
          implements ImportBeanDefinitionRegistrar, ResourceLoaderAware, EnvironmentAware {

          // patterned after Spring Integration IntegrationComponentScanRegistrar
          // and RibbonClientsConfigurationRegistrar
```

这个接口是spring的
只要实现了这个接口 重写里面的注册方法
那么就会了被ioc管理

资源加载拓展   环境变量的拓展

```
FeignClientsRegistrar.java
      @Override
142   public void registerBeanDefinitions(AnnotationMetadata metadata,
143           BeanDefinitionRegistry registry) {
144       registerDefaultConfiguration(metadata, registry);
145       registerFeignClients(metadata, registry);
146   }
147
```

扫描注解，注册

真正的扫描拿到注解和服务名称

5.2.2 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

```
44          * to cache the result.
45          */
46         /unchecked/
47         @Override
48         public <T> T newInstance(Target<T> target) {  target: "HardCodedTarget(type=UserOrderFeign, name=provider-order-servic
49             Map<String, MethodHandler> nameToHandler = targetToHandlersByName.apply(target);  nameToHandler:  size = 6  targetTo
50             Map<Method, MethodHandler> methodToHandler = new LinkedHashMap<Method, MethodHandler>();  methodToHandler:  size = 6
51             List<DefaultMethodHandler> defaultMethodHandlers = new LinkedList<DefaultMethodHandler>();  defaultMethodHandlers:
52
53             for (Method method : target.type().getMethods()) {
54                 if (method.getDeclaringClass() == Object.class) {
55                     continue;
56                 } else if (Util.isDefault(method)) {
57                     DefaultMethodHandler handler = new DefaultMethodHandler(method)
58                     defaultMethodHandlers.add(handler);
59                     methodToHandler.put(method, handler);
60                 } else {
61                     methodToHandler.put(method, nameToHandler.get(Feign.configKey(t
62                 }
63             }
64             InvocationHandler handler = factory.create(target, methodToHandler)
65             T proxy = (T) Proxy.newProxyInstance(target.type().getClassLoader()
66                 new Class<?>[] {target.type()}, handler);  target: "HardCodedTa
67
68             for (DefaultMethodHandler defaultMethodHandler : defaultMethodHandl
69                 defaultMethodHandler.bindTo(proxy);
70             }
```

创建代理对象

当我们执行调用的时候，打个断点去查看

```
19          /**
20           * 用户远程调用下单的接口
21           *
22           * @return
23           */
24          @GetMapping("userDoOrder")
25          public String userDoOrder() {
26              String result = userOrderFeign.doOrder();  userOrderFeign: "HardCodedTarget(type=UserOrderFeign, name=provider-
27              System.out.println(result);
28              return result;
29          }
30      }
31
```
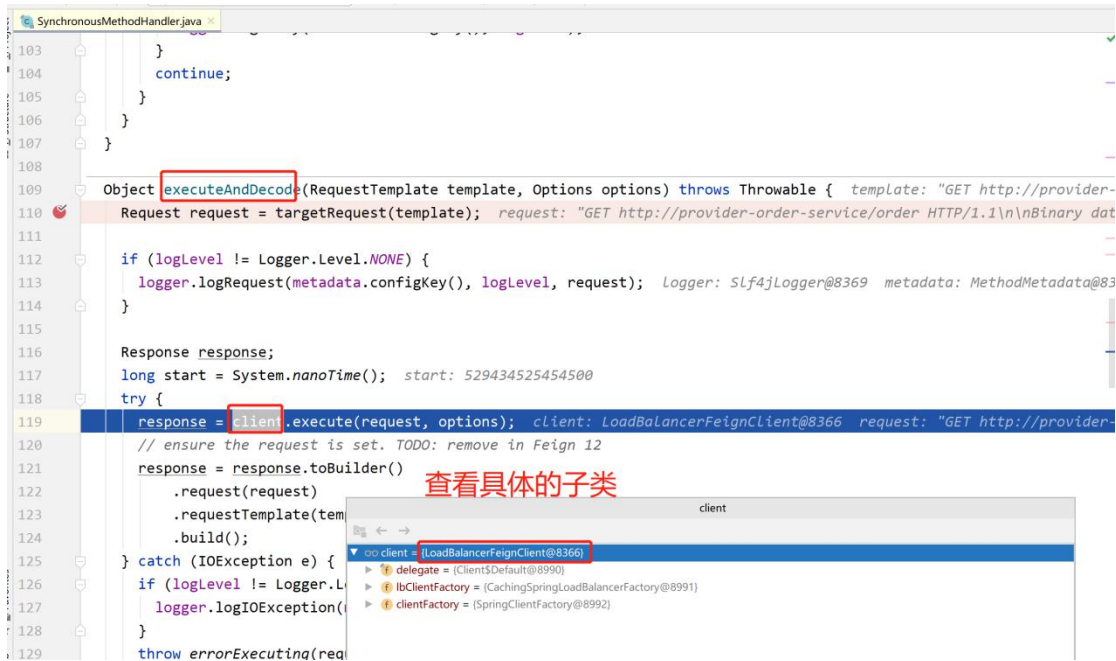
代理对象帮我们完成的调用

ReflectiveFeign 类中的 invoke 方法帮我们完成调用
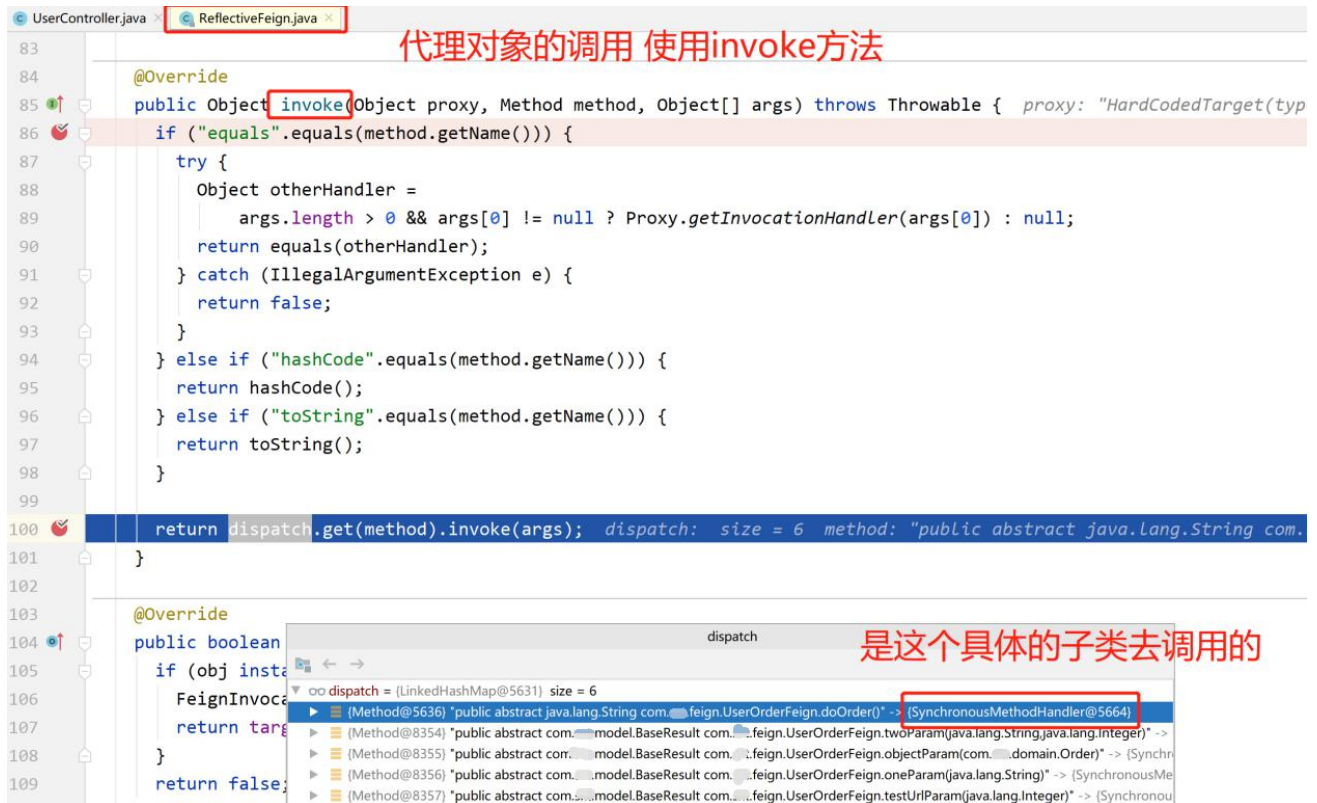
SynchronousMethodHandler 的 invoke 中给每一个请求创建了一个 requestTemplate 对象，去执行请求

executeAndDecode

我们去看 LoadBalancerFeignClient 的 execute 方法

executeWithLoadBalancer 继续往下看

最终是Client对象完成了调用

是HttpURLConnection发起的调用

拿到返回状态码

只要是 feign 调用出了问题

看 feign 包下面的 Client 接口下面的 108 行

200 成功

400 请求参数错误

401 没有权限

403 权限不够

404 路径不匹配

405 方法不允许

500 提供者报错了

302 资源重定向

6.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

2.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 JDK 代理

3.OpenFeign 到底是用什么做的远程调用？

使用的是 HttpURLConnection （java.net)

4.OpenFeign 怎么和 ribbon 整合的？

在代理对象执行调用的时候

```
  70        }
  71
  72        @Override
  73        public Response execute(Request request, Request.Options options) throws IOException {
  74            try {
  75                URI asUri = URI.create(request.url());
  76                String clientName = asUri.getHost();
  77                URI uriWithoutHost = cleanUrl(request.url(), clientName);
  78                FeignLoadBalancer.RibbonRequest ribbonRequest = new FeignLoadBalancer.RibbonRequest(
  79                        this.delegate, request, uriWithoutHost);
  80
  81                IClientConfig requestConfig = getClientConfig(options, clientName);
  82                return lbClient(clientName)
  83                        .executeWithLoadBalancer(ribbonRequest, requestConfig).toResponse();
  84            }
  85            catch (ClientException e) {
  86                IOException io = findIOException(e);
  87                if (io != null) {
```
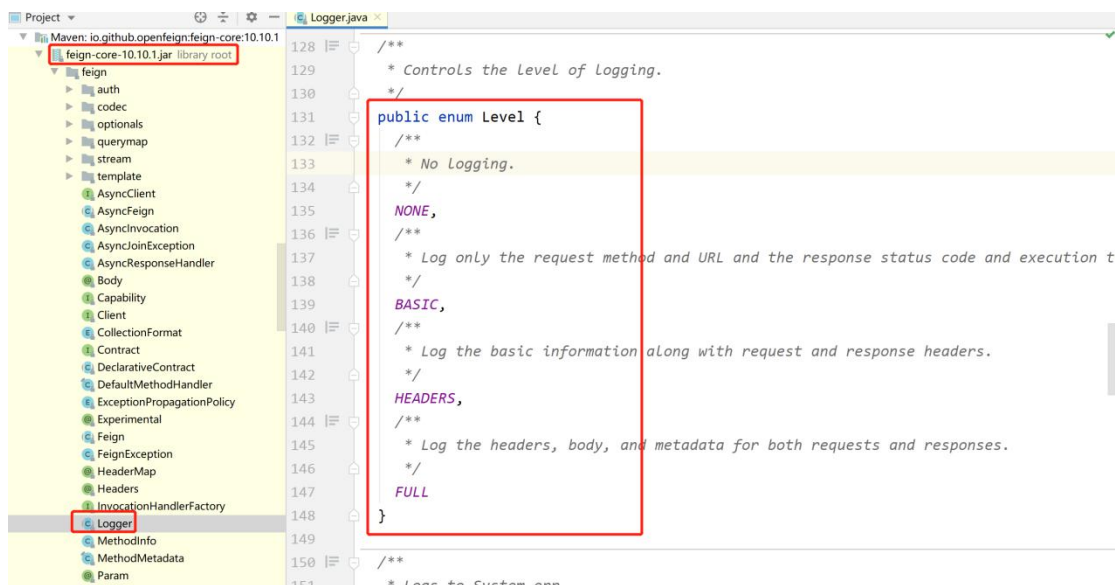
7.OpenFeign 其他

7.1 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

7.1.1 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

7.1.2 创建配置类

```java
package com.bjpowernode.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;
```

```
    }

}
```

7.1.3 修改配置文件

```
logging:
    level:
        com.bjpowernode.feign.UserOrderFeign: debug
```

7.1.4 调用测试

```
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] <--- HTTP/1.1 200 (184ms)
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] connection: keep-alive
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] content-length: 12
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] content-type: text/plain;charset=UTF-8
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] date: Mon, 09 Nov 2020 05:32:32 GMT
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] keep-alive: timeout=60
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder]
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] 下单成功
              DEBUG 7960 --- [nio-8081-exec-1] com.sxt.feign.UserOrderFeign            :
 [UserOrderFeign#doOrder] <--- END HTTP (12-byte body)
```