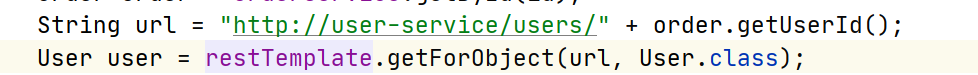
# SpringCloudAlibaba-Http客户端Feign

RestTemplate存在得问题：

先来看我们以前利用RestTemplate发起远程调用得代码：



通过url地址指明要访问的服务名称、请求路径、请求的参数信息、请求方式、请求返回值类型，而后ResetTemplate向指定地址发起远程调用，再把结果转成对应类型，这段代码已经实在ribbon的基础上做了很大的优化了，但是依然存在一些问题：

1：代码的可读性差，编程体验不统一，这段代码是http请求的代码，又有url路径，又有复杂的请求方式、。。。对于从来没有接触过微服务远程调用的人来说，这段代码可读性太差了！而且我们正常写代码都是在写方法，这里突然冒出来url，编程体验也不够统一；

2：参数复杂url难以维护，有的访问的请求参数就是很复杂的，请求参数甚至又十几二十个；

总而言之，用RestTemplate发起远程调用不优雅！

## 优雅的Feign

Feign是一个声明式的http客户端，什么是声明式：声明式我们并不陌生，在学习事务的时候就已经接触过这个概念，早期事务需要手动自己去开启事务，提交事务，简直就是灾难！后来有了Spring，Spring的声明式事务只需要你简单地在配置文件里告诉Spring，要对谁加事务，把规则定义好，就行了，以后所有的事务都由Spring帮你做！

那么声明式Http客户端也是如此，你不是要发http请求吗？使用restTemplate发起来不好发啊，现在你只要把发请求需要的信息声明出来，剩下的事你不用管，由Feign帮你做！这就是Feign的作用：

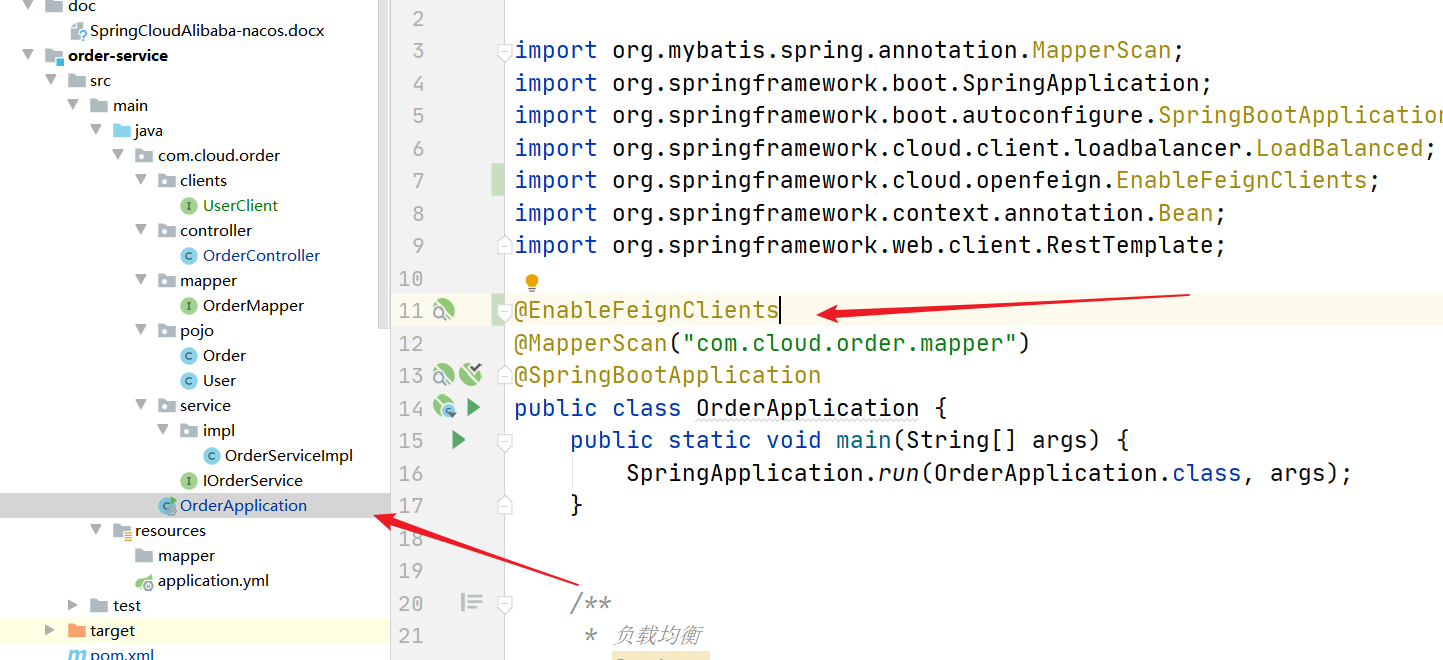
帮助我们优雅地实现http请求的发送，解决上面提到的问题！

## 使用Feign

1：引入Feign依赖



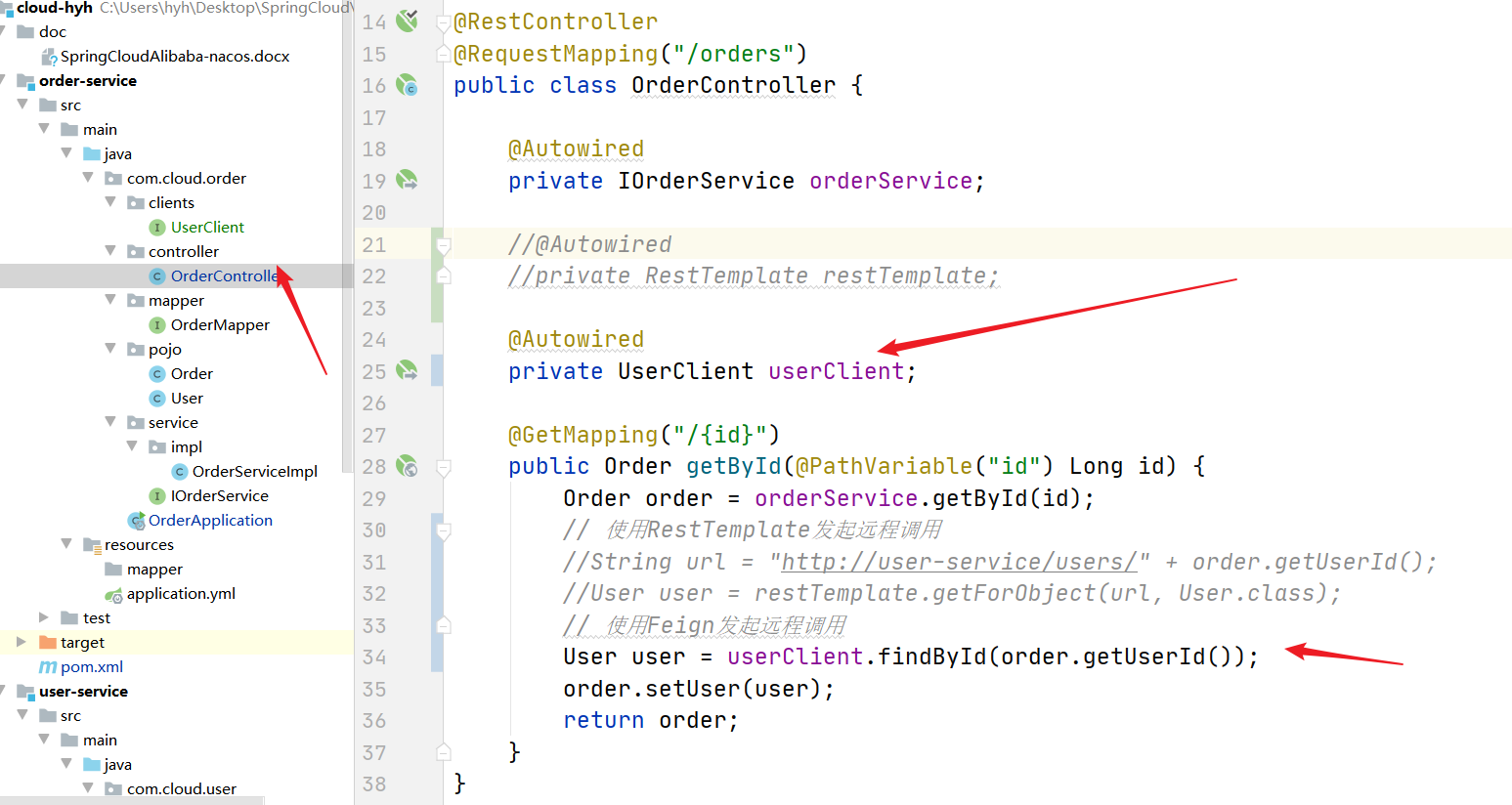
2：在order-service启动类加注解开启Feign的功能



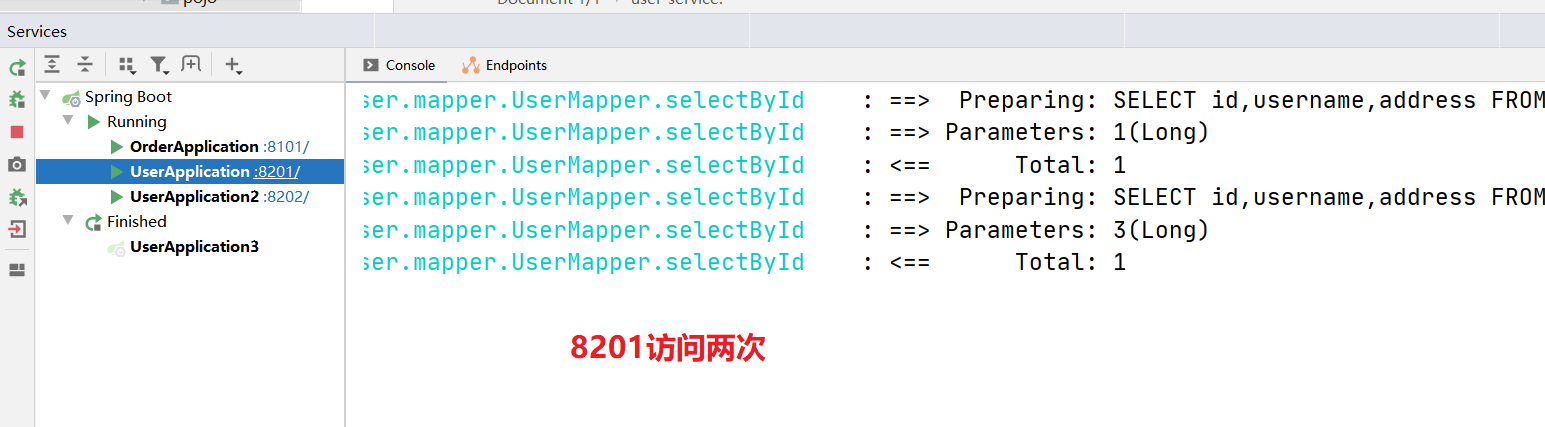
3：做声明，声明一个远程调用，编写Feign客户端：

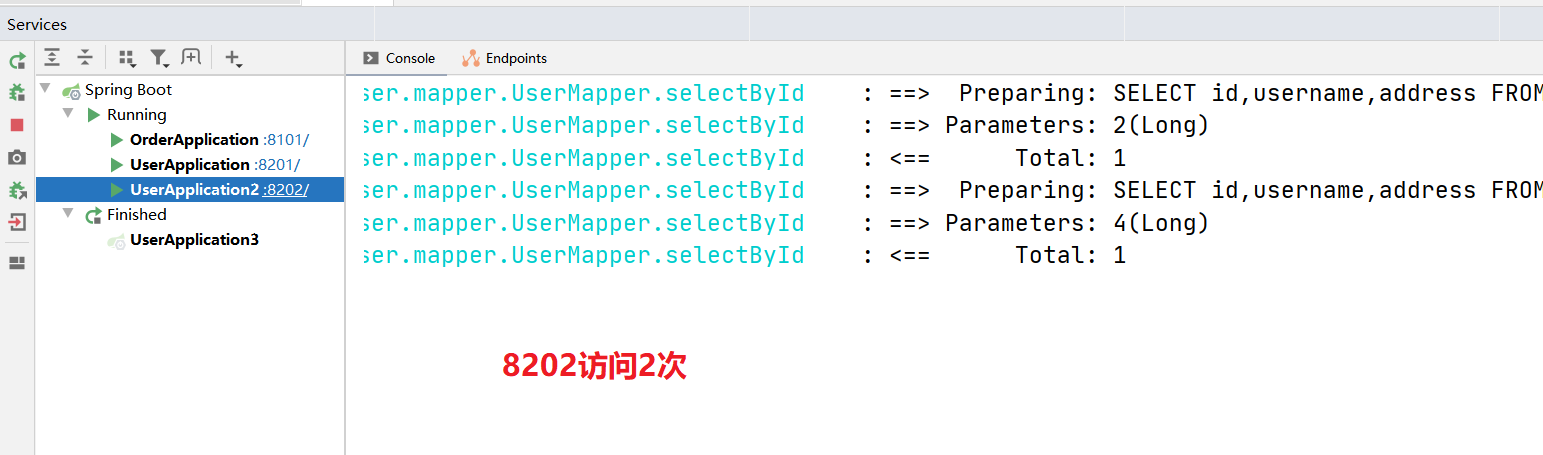


4：使用Feign客户端：



5：查看结果，说明不仅仅实现了远程调用，还是先了负载均衡！





## Feign的自定义配置



上面已经学习了Feign的基本使用，现在讲一下Feign的自定义配置，我们知道SpringBoot虽然帮我们实现了自动装配，但是是允许我们覆盖默认配置的。

像Feign，上图中5个常用的配置是可以被覆盖的，当然这不是全部，第一个是feign日志的级别，修改日志的级别：NONE，BASIC、HEADERS、FULL。NONE就是没有任何日志，我们的例子就是NONE：没有任何日志，默认就是NONE；BASIC代表当发起一次HTTP请求时，会帮你记录你请求什么时候发的、什么时候结束的、耗时多久等基本信息;HEADERS顾名思义除了记录请求基本信息之外，还会带上请求头、响应头信息，都给你记录下来；FULL就更恐怖了，除了会记录请求基本信息、请求头、响应头之外，还会把请求体、响应体信息都记录下来，是最完整的信息。

第二个是Decoder，响应结果解析器，当Feign发起一次远程调用时，比如说查到了一个用户，其实默认拿到的是一个JSON，但是最终我们想把JSON转成一个USER对象，谁来转呢？就是这个Decoder响应结果解析器。负责把查询到的JSON数据转变成对象。

第三个是Encoder，发请求时我们传递的参数可以是各种格式数据，那么真正发请求时需要把这些格式转变成Request请求体，这个动作就是由这个Encoder完成的。作为请求参数编码器。

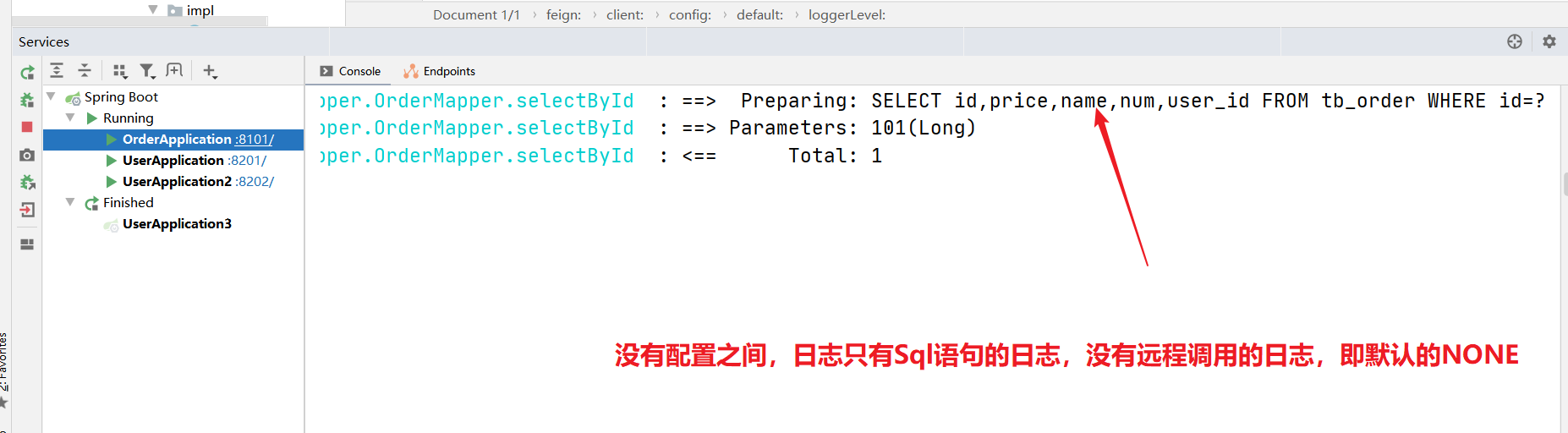
第四个是Contract，注解契约，是来去规定Feign支持的注解格式，默认情况下支持SpringMVC的注解，一般不用修改！

第五个是Retryer，即失败重试，在默认情况下，Feign是没有去做失败重试的，是不重试机制，不过尽管Feign自己不会做充实，但是Feign底层是依赖了Ribbon的，Ribbon底层是由重试机制的，所以Feign等于就也有了失败重试！什么叫失败重试呢？简单来说就是发送了一次请求，比如请求查用户，第一次查了8201实例，但是不知道什么原因，网络波动或者什么，导致这侧查询一直查不到，可能卡住了，卡了超过一定时间之后，Ribbon就不会等了，就回去做一次重试，重新试一下8202实例，看一下8202能不能查到。

以上是我们在Feign中常见的5种可以覆盖的配置，但是一般情况下我们需要配置的就是日志级别！

### 自定义Feign的的日志级别配置：

修改Feign的日志级别有两种方式，第一种是基于配置文件的方式，第二种是基于Java代码的方式！



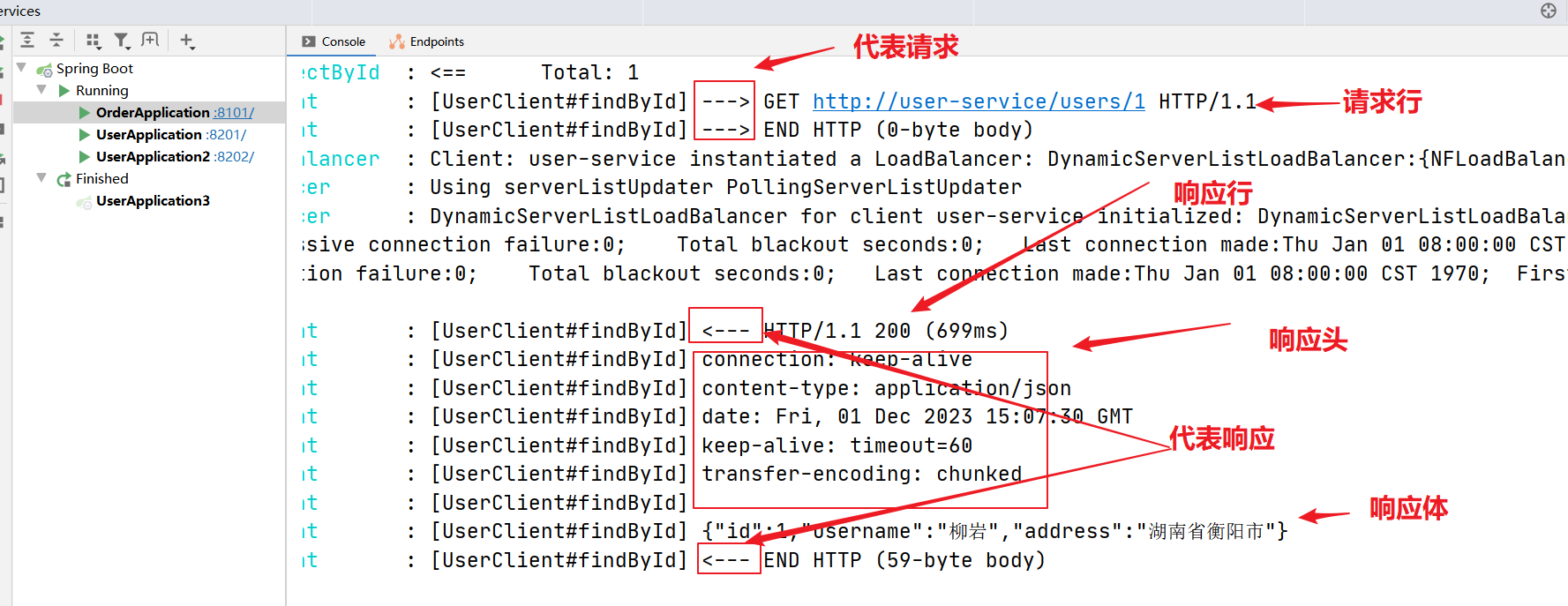
#### 基于配置文件

方式一是配置文件，feign.client.config.xxx.loggerLevel

如果xxx是default，则代表全局配置，就是针对于一切访问的微服务都生效，都按照配置的来；

如果xxx是服务名称，例如user-service，则代表这个配置只针对访问某服务的请求生效，其他服务不生效。





#### 基于Java代码

方式二是Java代码配置Logger.Level这个Bean

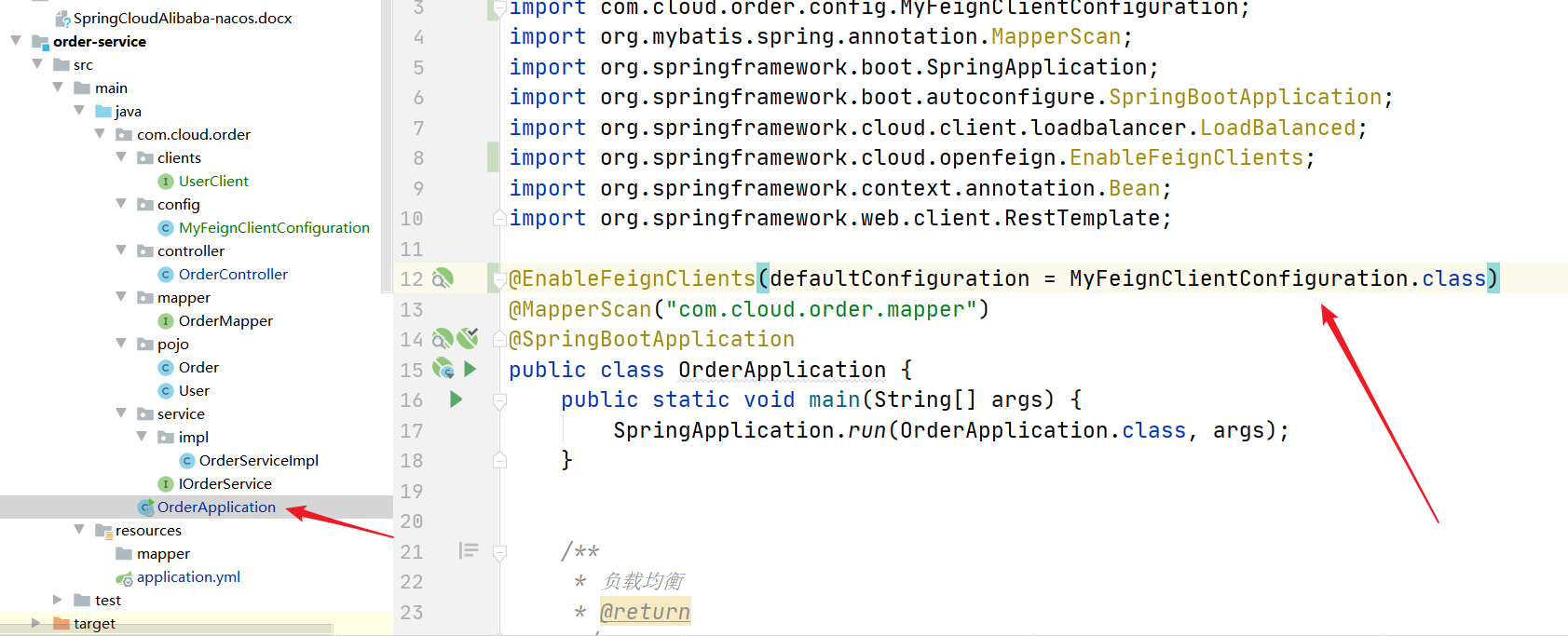
如果在@EnableFeignClients注解声明则代表全局；

如果在@FeignClient注解声明则代表某服务。

步骤一：需要先声明一个Bean



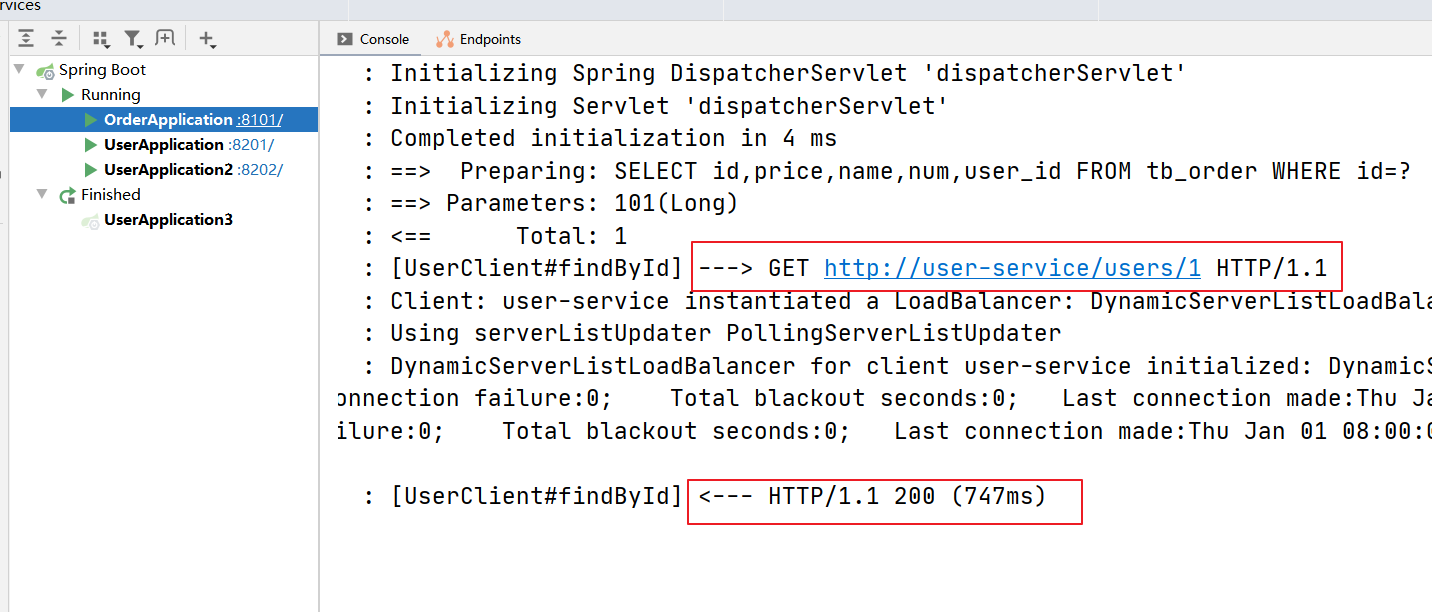
步骤二：如果是全局配置，则把它放到@EnableFeignClients这个注解中



步骤二：如果是针对某个微服务配置，则把它放到@FeignClient这个注解中



效果：看到Feign日志只有请求行和响应行



#### 小总结：

Feign日志级别修改有两种方式（注意线上不要配置，NONE就可以）：

方式1：配置文件方式

feign.client.config.default.loggerLevel=FULL  # 这里用default就是全局配置，如果写服务名称，则是针对某个微服务的配置；FULL表示日志级别（生产环境日志级别设成NONE或者BASIC）。

feign.client.config.user-service.loggerLevel=FULL  # 这里就是针对user-service这个微服务的配置。

方式2：java代码方式，需要生命一个Bean：Logger.Level

如果是全局配置，则把它放到@EnableFeignClients这个注解中：

@EnableFeignClients(defaultConfiguration = 配置.class)

如果是局部配置，则把它放到@FeignClient这个注解中：

@FeignClient(value=“服务名”, configuration=配置.class)

## Feign的性能优化

Feign的性能已经很好了，但还可以能好！为什么？

### 先了解一下Feign底层的客户端实现：

Feign底层的客户端实现的实现有三种模式，

1：URLConnection，默认实现，不支持连接池

Feign是一种声明式客户端，Feign只是帮助我们，把我们的声明变成Http请求，最终发送Http请求时，还是会用到别的Http客户端。Feign默认用的是URLConnection，这是JDK里自带的HTTP客户端，URLConnection性能是不太好的，而且URLConnection也不支持连接池，我们知道建立连接池以后，可以减少连接创建和销毁的性能损耗，因为我们知道连接每次创建都要做三次握手，断开连接要四次挥手，其实是比较浪费性能、浪费资源的，所以我们肯定是希望尽可能地较少创建这种操作！怎么办？使用连接池，而URLConnection是不支持连接池的，因此推荐使用另外两种：

2：Apache HttpClient，支持连接池

3：OKHttp，支持连接池

所以对Feign做性能优化，最主要的一点是对底层的客户端实现做改变，即用带连接池的实现来替代URLConnection。

因此优化Feign的性能主要包括：

1：使用连接池代替默认的URLConnection;

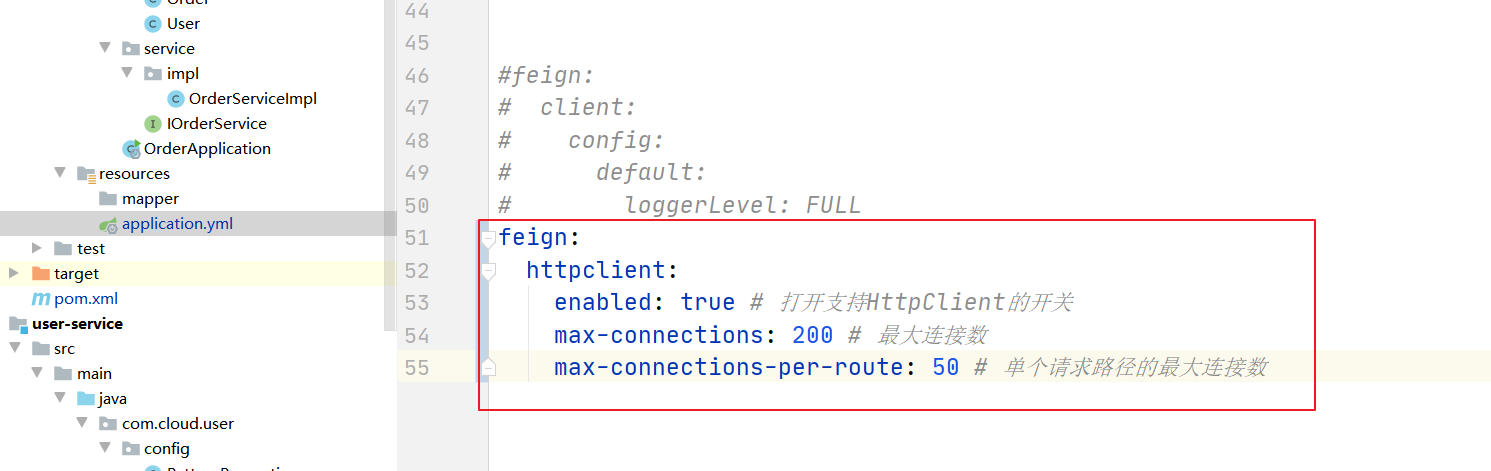
2：日志级别，最好用basic或none

### 使用Apache HttpClient取代URLConnection

步骤一：引入Apache HttpClient



步骤二：配置连接池，当然了，真实情况下要去做压测，试一下要设置成多少才合理



Feign的优化：

1：日志级别尽量用NONE 或 BASIC

2：使用HttpClient或者OKHttp代替URLConnection

2.1：引入feign-httpclient依赖

2.2：配置文件开启httpClient功能，设置连接池参数

## Feign的最佳实践

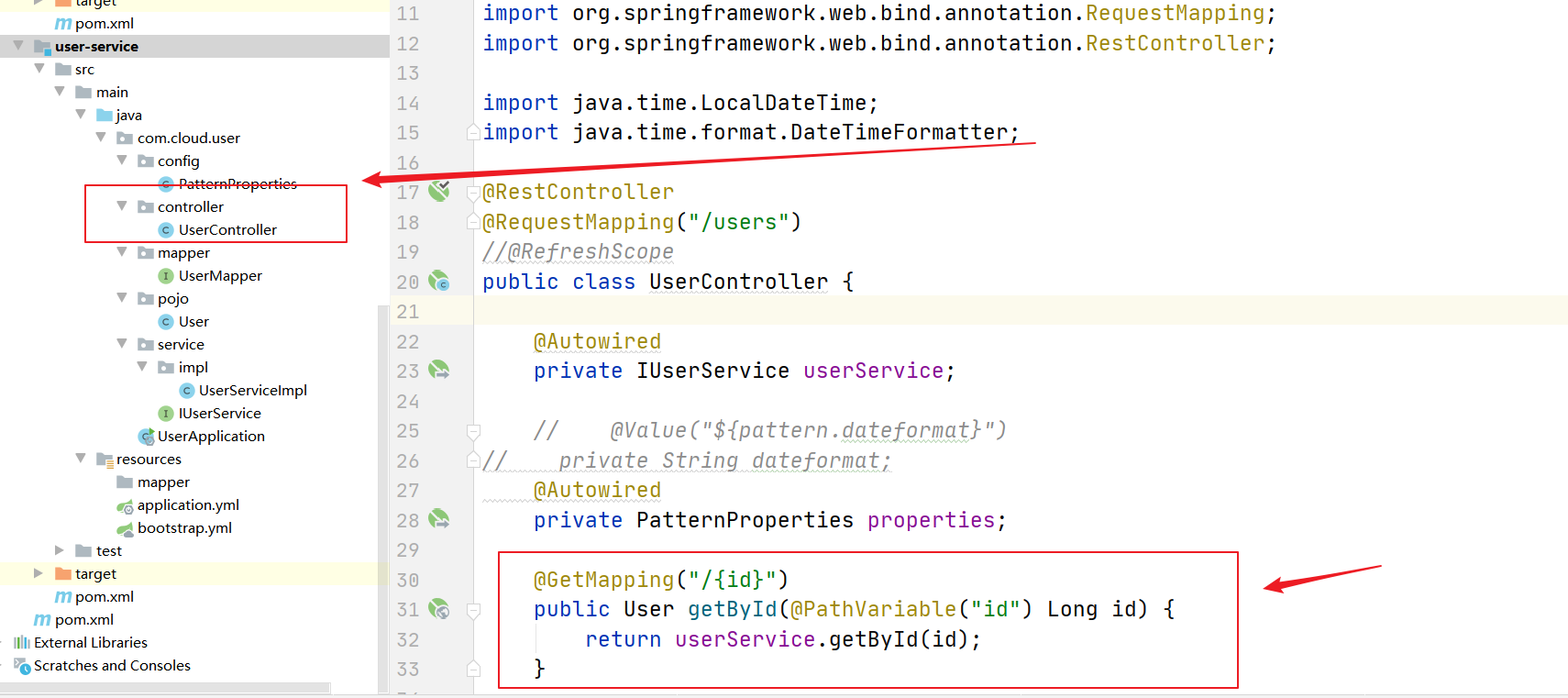
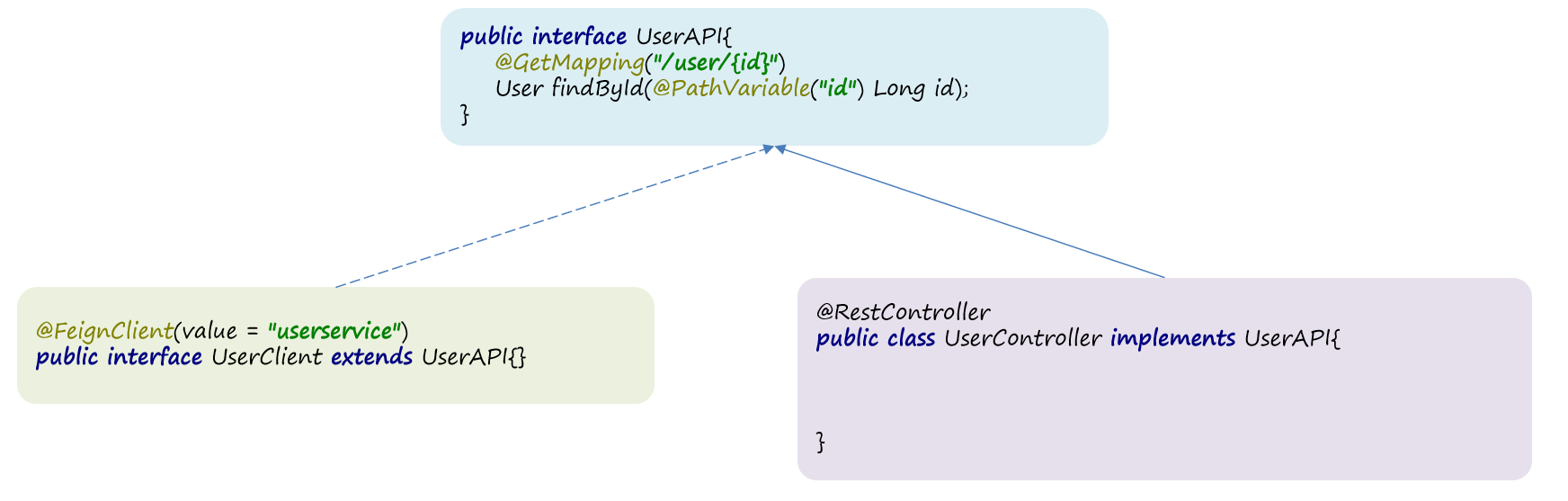
什么是最佳实践，就是企业在使用Feign的时候，经过各种踩坑，最后总结出来的一个相对比较好的使用方式

方式一（继承）：给服务消费者的FeignClient和服务提供者的controller定义统一的父接口作为标准.

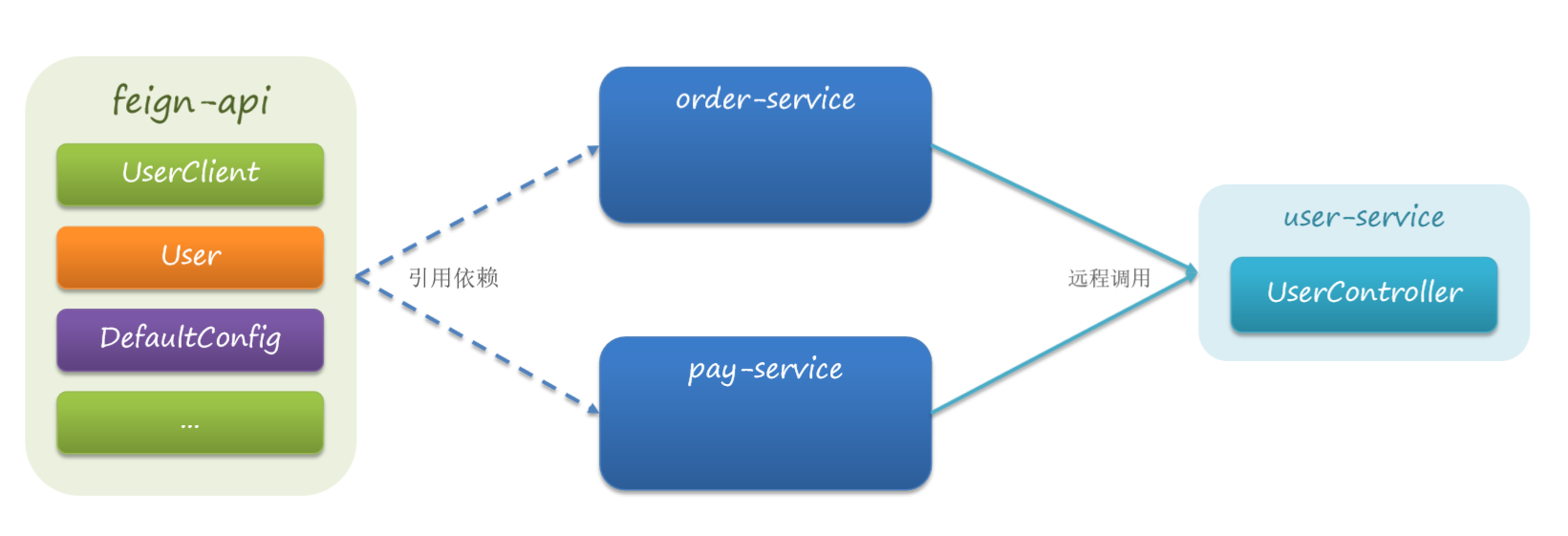
服务消费者的FeignClient：



服务提供者的controller：

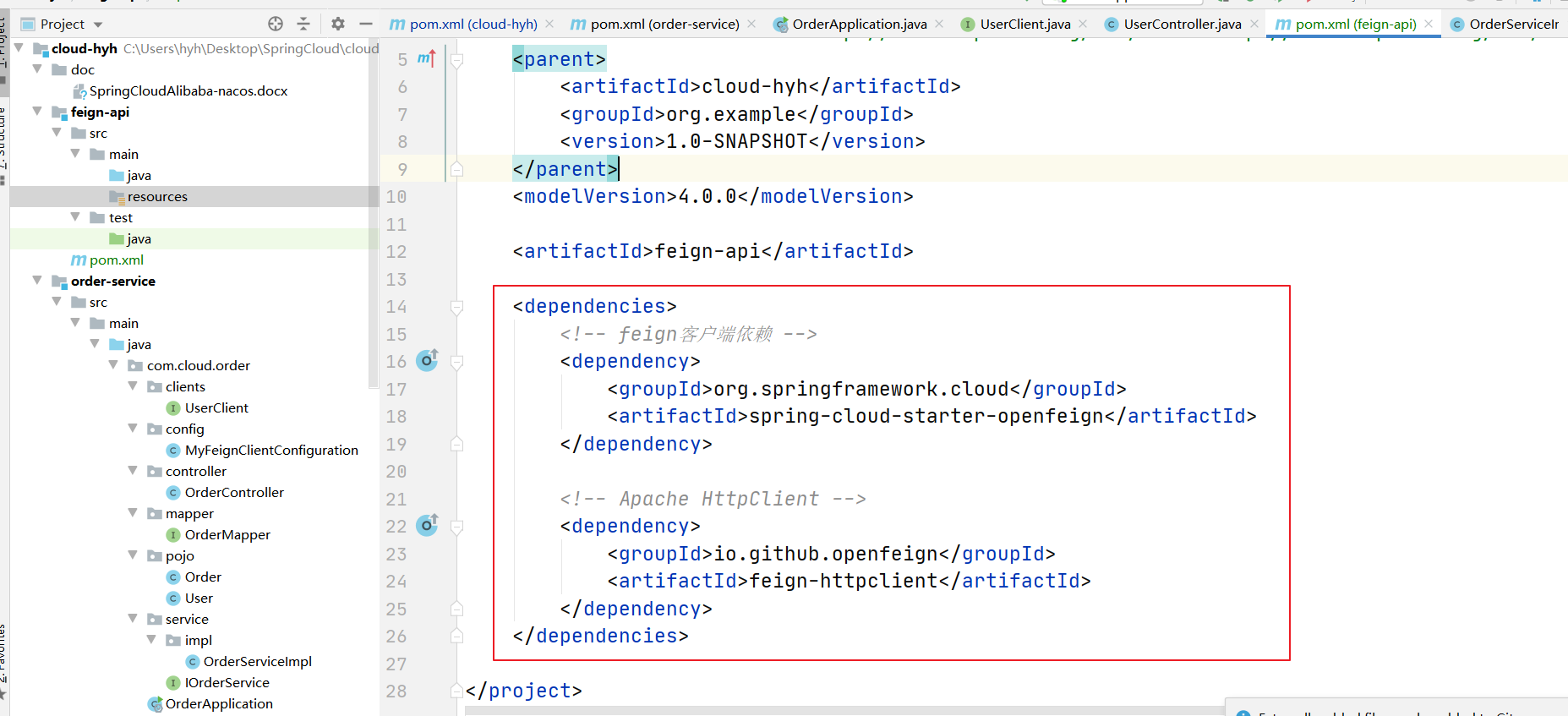
‘ 

方式二（抽取）：将FeignClient抽取为独立模块，并把接口相关的pojo、默认的Feign配置都放到这个模块中，提供给所有服务消费者使用

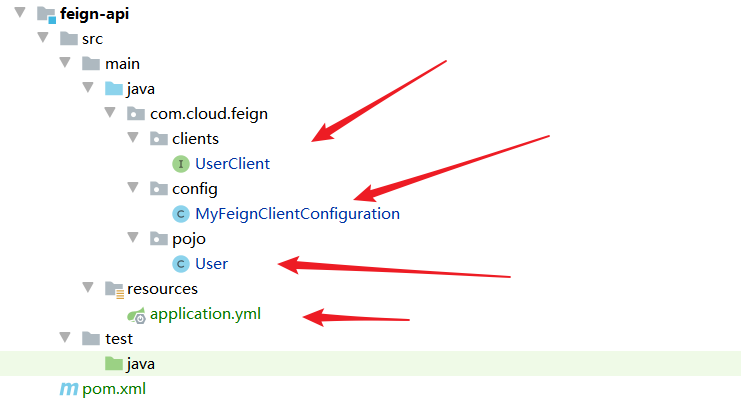


方式二实现步骤：

1：首先创建一个module，然后引入feign相关的的starter依赖：



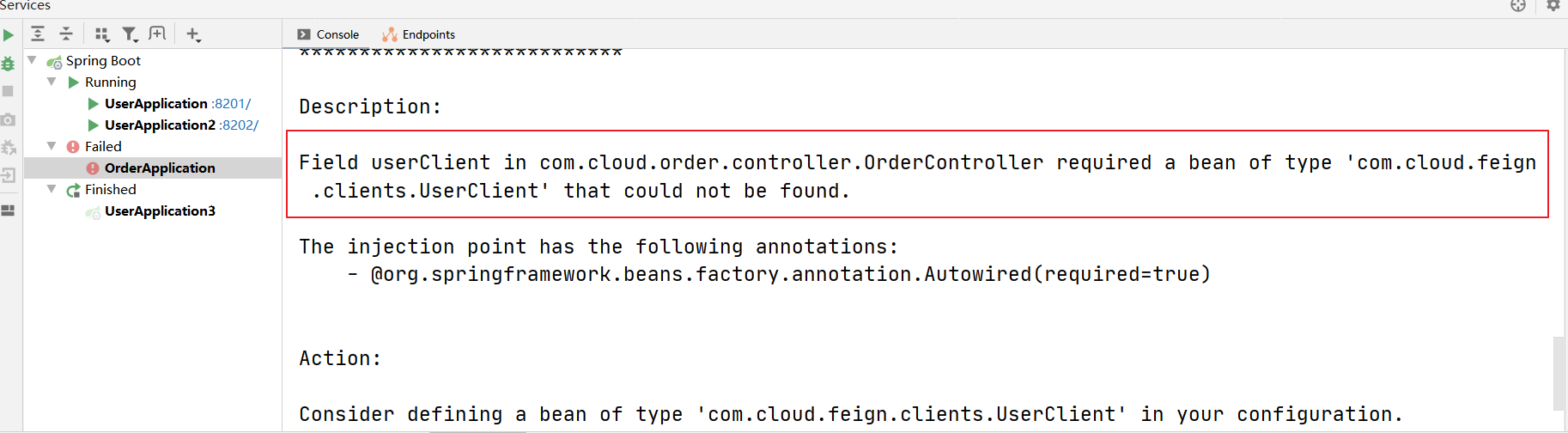
2：将order-service中的UserClient、User、MyFeignClientConfiguration都复制到新模块中：



3：在order-service模块中引入feign-api模块：



4：然后重启order-service，会报错：



提示UserClient没有创建，编译没报错，运行报错了！

这是因为如果你熟悉Spring，你应该知道编译没报错，证明这个类UserClient确实有，但是运行报错了，无法注入成功，证明这个类没有创建对象，即在Spring的容器里找不到。所以注入失败了！

那为什么UserClient接口没有对应的对象呢？要搞庆祝，这个接口之前之所以有对象，是因为这个接口有@FeignClient(“服务名称”)这个注解，Spring扫描到这个注解，就会给这个接口创建对象，现在呢，说明Spring没有扫描到这个注解！为什么呢？因为Order-Service默认的扫描包是启动类所在的包，启动类所在的包是：com.cloud.order，而此时UserClient所在的包是com.cloud.feign，这两个包都不一样，当然扫描不到，扫描不到自然不会创建对象UserClient，既然创建不了，Spring的容器里自然没有UserClient这个Bean，自然也就不能注入！这就是原因。

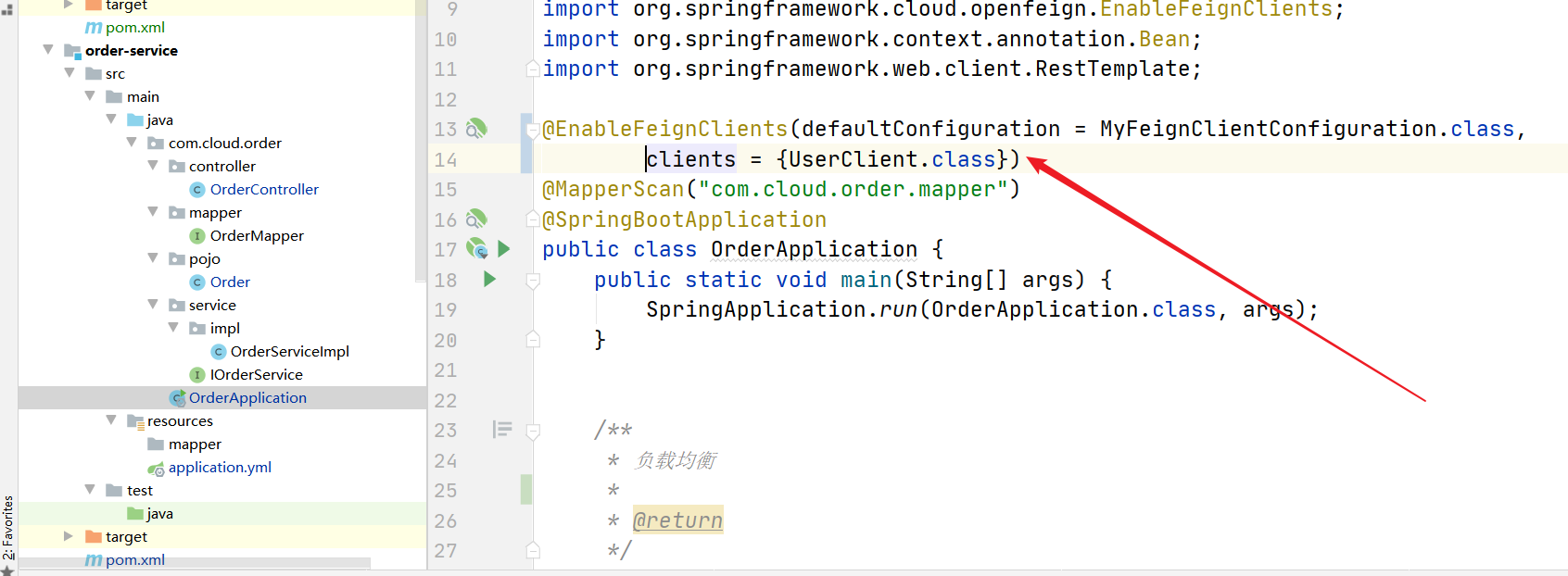
可能你会有一个想法：那就该feign-api的包名，将com.cloud.feign改成com.cloud.order，这是不合适的，你仅仅为了想要拿到UserClient，而扩大了扫描范围，是不合适的。

最佳方法：当定义的FeignClient不在SpringBootApplication的扫描包范围时，这些FeignClient无法使用，有两种解决方法：

方法一：在@EnableFeignClients注解中指定FeignClient所在包

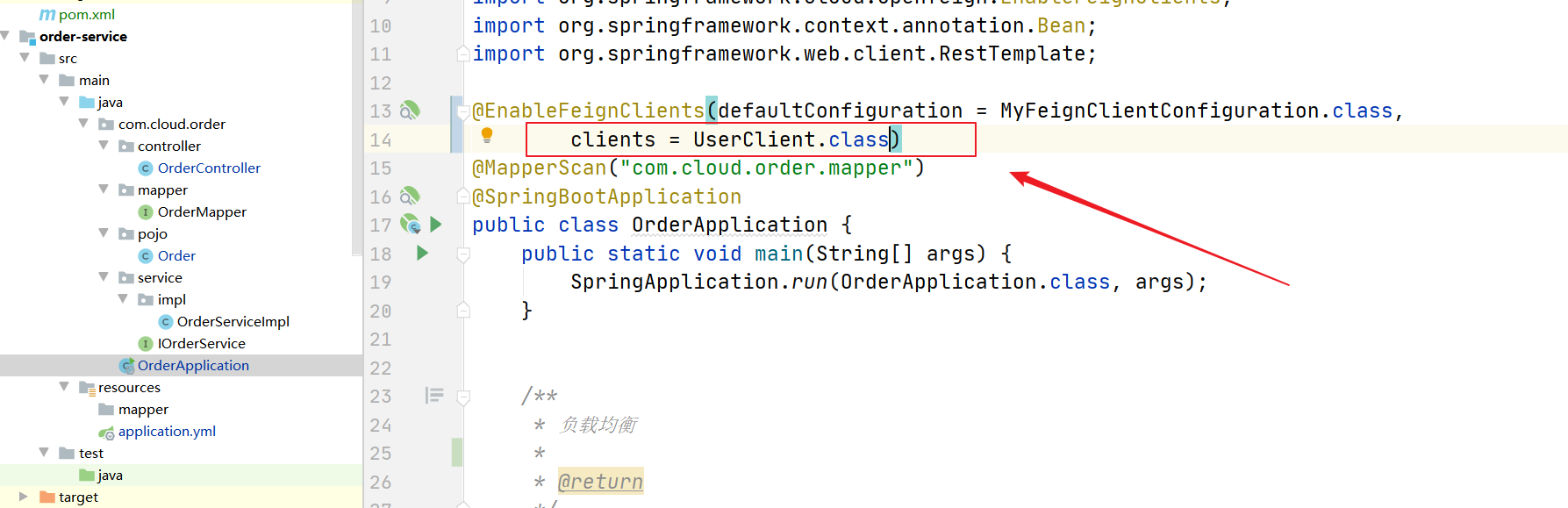


方式二：在@EnableFeignClients注解中指定FeignClient的字节码



推荐使用第二种方案：

第一种方法属于全拿来，批量地搞，存在不需要的FeignClient也被扫描进来；第二种方案属于精准定位，只加载指定的FeignClient，第二宗方案只有一个FeignClient时，大括号可以省略：



成功启动，没有问题：

