

1. RestTemplate存在问题

观察咱们远程调用的代码

```
1 public OrderInfo selectOrderById(Integer orderId) {
2     OrderInfo orderInfo = orderMapper.selectOrderById(orderId);
3     String url = "http://product-service/product/" + orderInfo.getProductId();
4     ProductInfo productInfo = restTemplate.getForObject(url,
        ProductInfo.class);
5     orderInfo.setProductInfo(productInfo);
6     return orderInfo;
7 }
```

虽说RestTemplate 对HTTP封装后, 已经比直接使用HttpClient简单方便很多, 但是还存在一些问题.

1. 需要拼接URL, 灵活性高, 但是封装臃肿, URL复杂时, 容易出错.
2. 代码可读性差, 风格不统一.

微服务之间的通信方式, 通常有两种: RPC 和 HTTP.

在SpringCloud中, 默认是使用HTTP来进行微服务的通信, 最常用的实现形式有两种:

- RestTemplate
- OpenFeign

RPC (Remote Procedure Call) 远程过程调用, 是一种通过网络从远程计算机上请求服务, 而不需要了解底层网络通信细节. RPC可以使用多种网络协议进行通信, 如HTTP、TCP、UDP等, 并且在TCP/IP网络四层模型中跨越了传输层和应用层. 简言之RPC就是像调用本地方法一样调用远程方法.

常见的RPC框架有:

1. Dubbo: [Apache Dubbo 中文](#)
2. Thrift: [Apache Thrift - Home](#)
3. gRPC: [gRPC](#)

2. OpenFeign介绍

OpenFeign 是一个声明式的 Web Service 客户端. 它让微服务之间的调用变得更简单, 类似controller调用service, 只需要创建一个接口, 然后添加注解即可使用OpenFeign.

OpenFeign 的前身

Feign 是 `Netflix` 公司开源的一个组件。

- 2013年6月, Netflix发布 Feign的第一个版本 `1.0.0`
- 2016年7月, Netflix发布Feign的最后一个版本 `8.18.0`

2016年, Netflix 将 Feign 捐献给社区

- 2016年7月 OpenFeign 的首个版本 `9.0.0` 发布, 之后一直持续发布到现在。

可以简单理解为 Netflix Feign 是OpenFeign的祖先, 或者说OpenFeign 是Netflix Feign的升级版。
OpenFeign 是Feign的一个更强大更灵活的实现。



我们现在网络上看到的文章, 或者公司使用的Feign, 大多都是OpenFeign。

本课程后续讲的Feign, 指的是OpenFeign

Spring Cloud Feign

Spring Cloud Feign 是 Spring 对 Feign 的封装, 将 Feign 项目集成到 Spring Cloud 生态系统中。

受 **Feign** 更名影响, **Spring Cloud Feign** 也有两个 `starter`

- `spring-cloud-starter-feign`
- `spring-cloud-starter-openfeign`

由于Feign的停更维护, 对应的, 我们使用的依赖是 `spring-cloud-starter-openfeign`

OpenFeign 官方文档: [GitHub - OpenFeign/feign: Feign makes writing java http clients easier](#)

Spring Cloud Feign文档: [Spring Cloud OpenFeign](#)

3. 快速上手

3.1 引入依赖

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-openfeign</artifactId>
4 </dependency>
```

3.2 添加注解

在order-service的启动类添加注解 `@EnableFeignClients` , 开启OpenFeign的功能.

```
1 @EnableFeignClients
2 @SpringBootApplication
3 public class OrderServiceApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(OrderServiceApplication.class, args);
6     }
7 }
```

3.3 编写OpenFeign的客户端

基于SpringMVC的注解来声明远程调用的信息

```
1 import com.bite.order.model.ProductInfo;
2 import org.springframework.cloud.openfeign.FeignClient;
3 import org.springframework.web.bind.annotation.PathVariable;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 @FeignClient(value = "product-service", path = "/product")
7 public interface ProductApi {
8     @RequestMapping("/{productId}")
9     ProductInfo getProductById(@PathVariable("productId") Integer productId);
10 }
```

@FeignClient 注解作用在接口上, 参数说明:

- name/value: 指定FeignClient的名称, 也就是微服务的名称, 用于服务发现, Feign底层会使用Spring Cloud LoadBalance进行负载均衡. 也可以使用 `url` 属性指定一个具体的url.
- path: 定义当前FeignClient的统一前缀.

3.4 远程调用

修改远程调用的方法

```
1 @Autowired
2 private ProductApi productApi;
3 /**
4  * Feign实现远程调用
```

```

5  * @param orderId
6  */
7  public OrderInfo selectOrderById(Integer orderId) {
8      OrderInfo orderInfo = orderMapper.selectOrderById(orderId);
9      ProductInfo productInfo =
    productApi.getProductById(orderInfo.getProductId());
10     orderInfo.setProductInfo(productInfo);
11     return orderInfo;
12 }

```

代码对比:

```

1 public OrderInfo selectOrderById(Integer orderId) {
2     OrderInfo orderInfo = orderMapper.selectOrderById(orderId);
3     String url = "http://product-service/product/" + orderInfo.getId();
4     ProductInfo productInfo = restTemplate.getForObject(url, ProductInfo.class);
5     orderInfo.setProductInfo(productInfo);
6     return orderInfo;
7 }

```

```

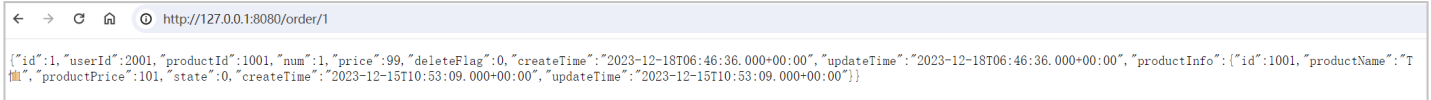
1 public OrderInfo selectOrderById(Integer orderId) {
2     OrderInfo orderInfo = orderMapper.selectOrderById(orderId);
3     ProductInfo productInfo = productApi.getProductById(orderInfo.getId());
4     orderInfo.setProductInfo(productInfo);
5     return orderInfo;
6 }

```

3.5 测试

启动服务, 访问接口, 测试远程调用:

<http://127.0.0.1:8080/order/1>



```

{"id":1,"userId":2001,"productId":1001,"num":1,"price":99,"deleteFlag":0,"createTime":"2023-12-18T06:46:36.000+00:00","updateTime":"2023-12-18T06:46:36.000+00:00","productInfo":{"id":1001,"productName":"T恤","productPrice":101,"state":0,"createTime":"2023-12-15T10:53:09.000+00:00","updateTime":"2023-12-15T10:53:09.000+00:00"}}

```

可以看出来, 使用Feign也可以实现远程调用.

Feign 简化了与HTTP服务交互的过程, 把REST客户端的定义转换为Java接口, 并通过注解的方式来声明请求参数,请求方式等信息, 使远程调用更加方便和间接.

4. OpenFeign参数传递

通过观察, 我们也可以发现, Feign的客户端和服务提供者的接口声明非常相似

上面例子中, 演示了Feign 从URL中获取参数, 接下来演示下Feign参数传递的其他方式

只做代码演示, 不做功能

4.1 传递单个参数

服务提供方 product-service

```

1
2 @RequestMapping("/product")
3 @RestController

```

```
4 public class ProductController {
5     @RequestMapping("/p1")
6     public String p1(Integer id){
7         return "p1接收到参数:"+id;
8     }
9 }
```

Feign客户端

```
1 @FeignClient(value = "product-service", path = "/product")
2 public interface ProductApi {
3
4     @RequestMapping("/p1")
5     String p1(@RequestParam("id") Integer id);
6 }
```



注意: @RequestParam 做参数绑定, 不能省略

服务消费方order-service

```
1
2 @RequestMapping("/feign")
3 @RestController
4 public class TestFeignController {
5     @Autowired
6     private ProductApi productApi;
7
8     @RequestMapping("/o1")
9     public String o1(Integer id){
10         return productApi.p1(id);
11     }
12 }
```

测试远程调用

<http://127.0.0.1:8080/feign/o1?id=5>



4.2 传递多个参数

使用多个@RequestParam 进行参数绑定即可

服务提供方 product-service

```
1 @RequestMapping("/p2")
2 public String p2(Integer id,String name){
3     return "p2接收到参数,id:"+id+",name:"+name;
4 }
5
```

Feign客户端

```
1 @RequestMapping("/p2")
2 String p2(@RequestParam("id")Integer id,@RequestParam("name")String name);
```

服务消费方order-service

```
1 @RequestMapping("/o2")
2 public String o2(@RequestParam("id")Integer id,@RequestParam("name")String
   name){
3     return productApi.p2(id,name);
4 }
5
```

测试远程调用

<http://127.0.0.1:8080/feign/o2?id=5&name=zhangsan>

← → ↻ 🏠 ⓘ http://127.0.0.1:8080/feign/o2?id=5&name=zhangsan

p2接收到参数,id:5,name:zhangsan

4.3 传递对象

服务提供方 product-service

```
1 @RequestMapping("/p3")
2 public String p3(ProductInfo productInfo){
3     return "接收到对象, productInfo:"+productInfo;
4 }
5
```

Feign客户端

```
1 @RequestMapping("/p3")
2 String p3(@SpringQueryMap ProductInfo productInfo);
```

服务消费方 order-service

```
1 @RequestMapping("/o3")
2 public String o3(ProductInfo productInfo){
3     return productApi.p3(productInfo);
4 }
5
```

测试远程调用

<http://127.0.0.1:8080/feign/o3?id=5&productName=zhangsan>

← → ↻ 🏠 ⓘ http://127.0.0.1:8080/feign/o3?id=5&productName=zhangsan

接收到对象, productInfo:ProductInfo(id=5, productName=zhangsan, productPrice=null, state=null, createTime=null, updateTime=null)

4.4 传递JSON

服务提供方 product-service

```

1 @RequestMapping("/p4")
2 public String p4(@RequestBody ProductInfo productInfo){
3     return "接收到对象, productInfo:"+productInfo;
4 }
5

```

Feign客户端

```

1 @RequestMapping("/p4")
2 String p4(@RequestBody ProductInfo productInfo);

```

服务消费方order-service

```

1 @RequestMapping("/o4")
2 public String o4(@RequestBody ProductInfo productInfo){
3     System.out.println(productInfo.toString());
4     return productApi.p4(productInfo);
5 }

```

测试远程调用

<http://127.0.0.1:8080/feign/o4>

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://127.0.0.1:8080/feign/o4
- Body Type:** JSON
- Request Body:**

```

1 {
2   "id":66,
3   "productName":"T恤"
4 }

```
- Status:** 200 OK
- Time:** 291 ms
- Response Body:**

```

1 接收到对象, productInfo:ProductInfo(id=66, productName=T恤, productPrice=null, state=null, createTime=null, updateTime=null)

```

5. 最佳实践

最佳实践, 其实也就是经过历史的迭代, 在项目中的实践过程中, 总结出来的最好的使用方式.

通过观察, 我们也能看出来, Feign的客户端与服务提供者的controller代码非常相似

Feign 客户端

```
1 @FeignClient(value = "product-service", path = "/product")
2 public interface ProductApi {
3     @RequestMapping("/{productId}")
4     ProductInfo getProductById(@PathVariable("productId") Integer productId);
5 }
```

服务提供方Controller

```
1 @RequestMapping("/product")
2 @RestController
3 public class ProductController {
4
5     @RequestMapping("/{productId}")
6     public ProductInfo getProductById(@PathVariable("productId") Integer
7     productId){
8         //...
9     }
```

有没有一种方法可以简化这种写法呢?

5.1 Feign 继承方式

Feign 支持继承的方式, 我们可以把一些常见的操作封装到接口里.

我们可以定义好一个接口, 服务提供方实现这个接口, 服务消费方编写Feign 接口的时候, 直接继承这个接口

具体参考: [Spring Cloud OpenFeign Features :: Spring Cloud Openfeign](#)

5.1.1 创建一个Module

接口可以放在一个公共的Jar包里, 供服务提供方和服务消费方使用

New Module

New Module

Generators

m

 Maven Archetype

JavaFX

Compose Multiplatform

IDE Plugin

Android

Spring Initializr

Name:

product-api

Location:

D:\Git\spring-cloud\spring-cloud-feign2

Module will be created in: D:\Git\spring-cloud

Language:

Java

Groovy

HTML

+

Build system:

IntelliJ

Maven

Gradle

JDK:

Project SDK 17

Parent:

m

 spring-cloud-feign2

☐ Add sample code

> Advanced Settings

?

Create

Cancel

5.1.2 引入依赖

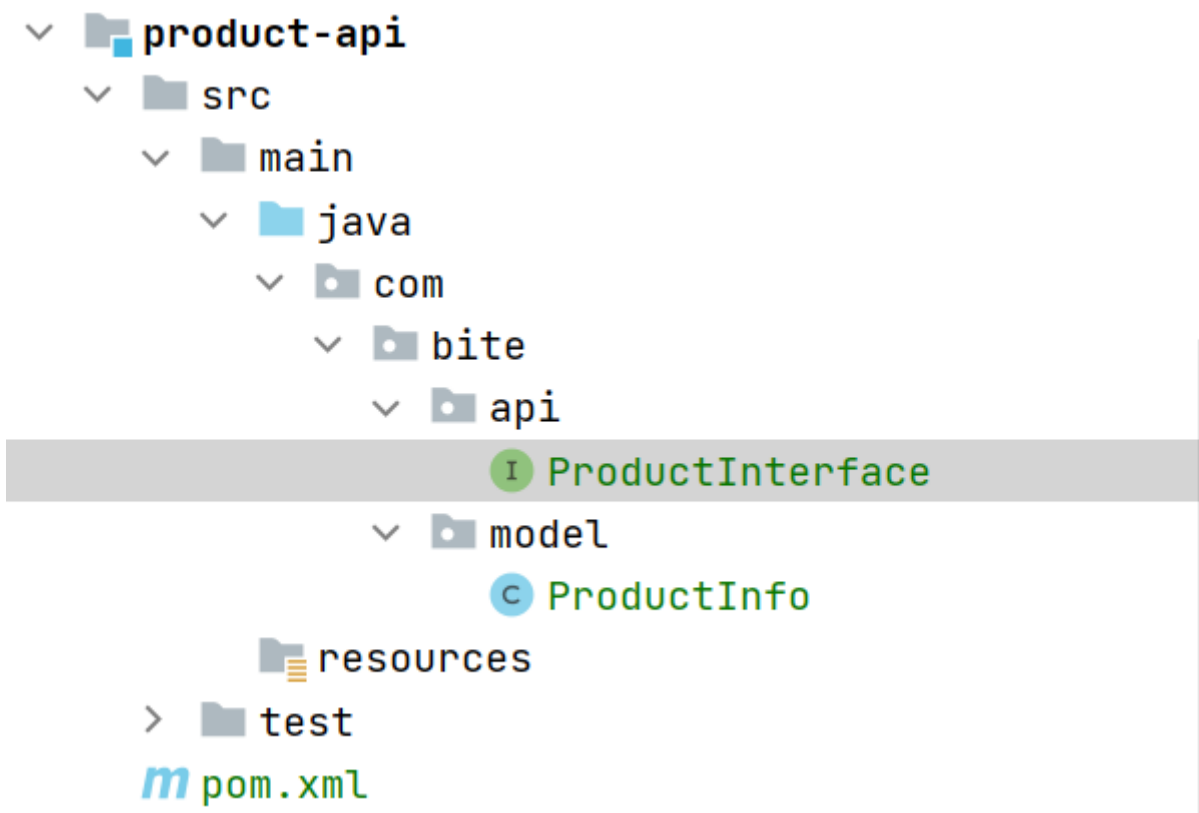
```
1 <dependencies>
2     <dependency>
3         <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-starter-web</artifactId>
5     </dependency>
6     <dependency>
7         <groupId>org.springframework.cloud</groupId>
8         <artifactId>spring-cloud-starter-openfeign</artifactId>
9     </dependency>
10 </dependencies>
```

5.1.3 编写接口

复制 ProductApi, ProductInfo 到product-api模块中

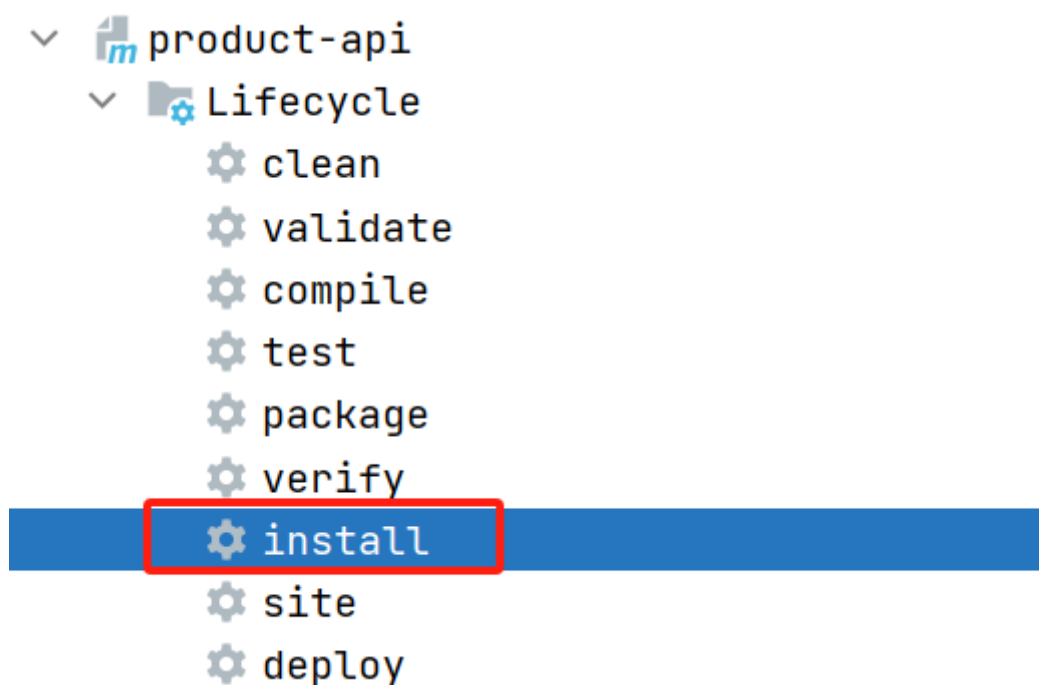
```
1 import com.bite.model.ProductInfo;
2 import org.springframework.cloud.openfeign.SpringQueryMap;
3 import org.springframework.web.bind.annotation.PathVariable;
4 import org.springframework.web.bind.annotation.RequestBody;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7
8 public interface ProductInterface {
9     @RequestMapping("/{productId}")
10     ProductInfo getProductById(@PathVariable("productId") Integer productId);
11
12     @RequestMapping("/p1")
13     String p1(@RequestParam("id") Integer id);
14
15     @RequestMapping("/p2")
16     String p2(@RequestParam("id") Integer id, @RequestParam("name") String name);
17
18     @RequestMapping("/p3")
19     String p3(@SpringQueryMap ProductInfo productInfo);
20
21     @RequestMapping("/p4")
22     String p4(@RequestBody ProductInfo productInfo);
23 }
```

目录结构如下:



5.1.4 打Jar包

通过Maven打包



观察Maven本地仓库, Jar包是否打成功

- 1 [INFO] Installing D:\Git\spring-cloud\spring-cloud-feign2\product-api\target\product-api-1.0-SNAPSHOT.jar to D:\Maven\.m2\repository\org\example\product-api\1.0-SNAPSHOT\product-api-1.0-SNAPSHOT.jar
- 2 [INFO] -----

```
3 [INFO] BUILD SUCCESS
4 [INFO] -----
5 [INFO] Total time: 2.796 s
6 [INFO] Finished at: 2024-01-03T19:31:35+08:00
7 [INFO] -----
8
```

此电脑 > Data (D:) > Maven > .m2 > repository > org > example > product-api

名称	修改日期	类型	大小
1.0-SNAPSHOT	2024/1/3 17:55	文件夹	
 maven-metadata-local.xml	2024/1/3 19:31	XML 文件	1 KB

5.1.5 服务提供方

服务提供方实现接口 ProductInterface

```
1 @RequestMapping("/product")
2 @RestController
3 public class ProductController implements ProductInterface {
4     @Autowired
5     private ProductService productService;
6
7     @RequestMapping("/{productId}")
8     public ProductInfo getProductById(@PathVariable("productId") Integer
9 productId){
10         System.out.println("收到请求,Id:"+productId);
11         return productService.selectProductById(productId);
12     }
13
14     @RequestMapping("/p1")
15     public String p1(Integer id){
16         return "p1接收到参数:"+id;
17     }
18
19     @RequestMapping("/p2")
20     public String p2(Integer id,String name){
21         return "p2接收到参数,id:"+id +" ,name:"+name;
22     }
23
24     @RequestMapping("/p3")
25     public String p3(ProductInfo productInfo){
26         return "接收到对象, productInfo:"+productInfo;
```

```

27     @RequestMapping("/p4")
28     public String p4(@RequestBody ProductInfo productInfo){
29         return "接收到对象, productInfo:"+productInfo;
30     }
31
32 }

```

5.1.6 服务消费方

服务消费方继承ProductInterface

```

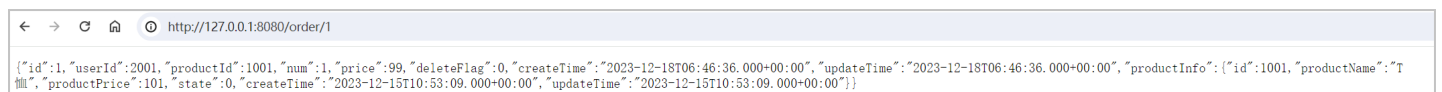
1 import com.bite.api.ProductInterface;
2 import org.springframework.cloud.openfeign.FeignClient;
3
4 @FeignClient(value = "product-service", path = "/product")
5 public interface ProductApi extends ProductInterface {
6
7 }

```

5.1.7 测试

试远程调用

<http://127.0.0.1:8080/order/1>



```

{
  "id": 1,
  "userId": 2001,
  "productId": 1001,
  "num": 1,
  "price": 99,
  "deleteFlag": 0,
  "createTime": "2023-12-18T06:46:36.000+00:00",
  "updateTime": "2023-12-18T06:46:36.000+00:00",
  "productInfo": {
    "id": 1001,
    "productName": "T恤",
    "productPrice": 101,
    "state": 0,
    "createTime": "2023-12-15T10:53:09.000+00:00",
    "updateTime": "2023-12-15T10:53:09.000+00:00"
  }
}

```

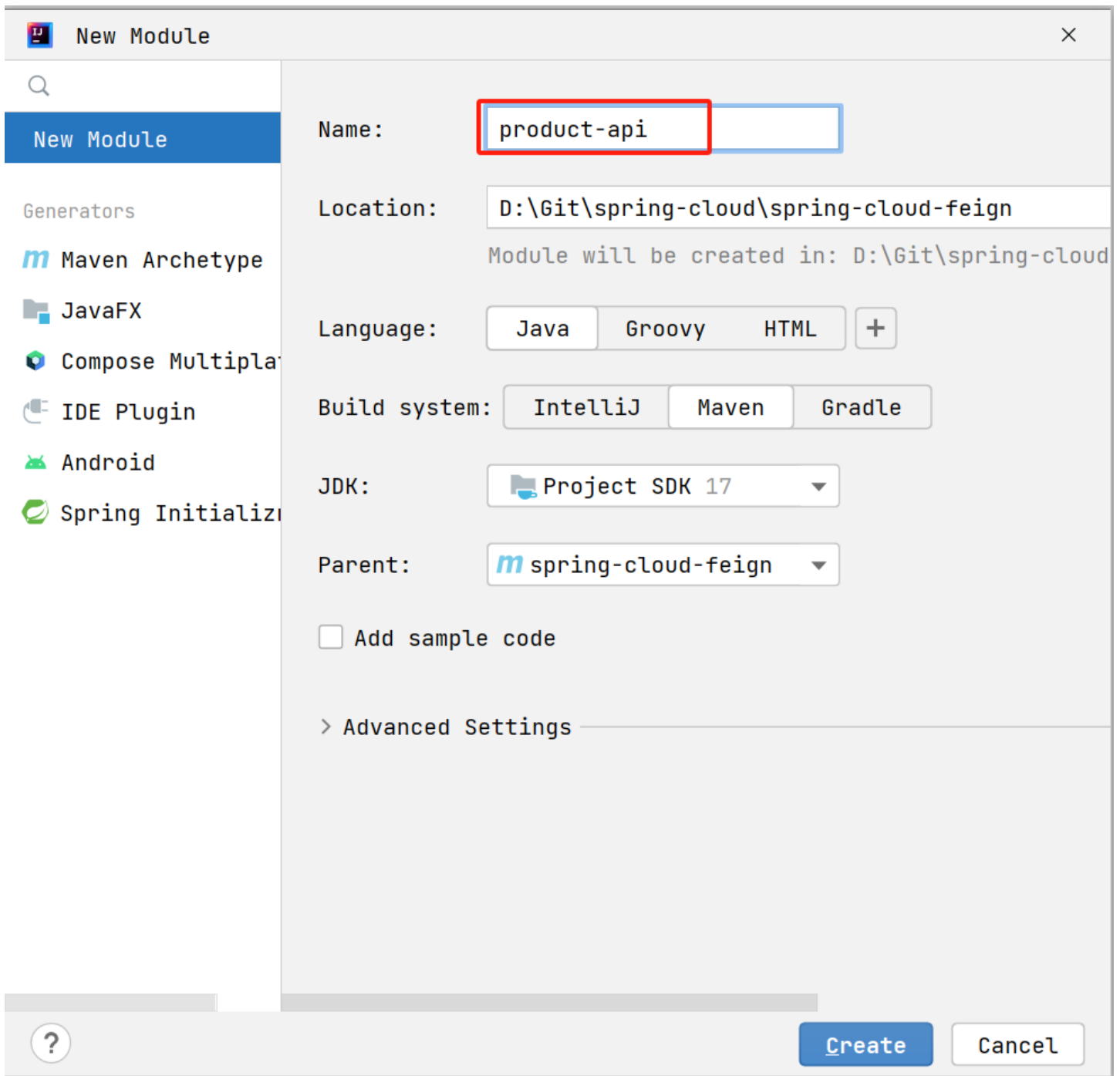
5.2 Feign 抽取方式

官方推荐Feign的使用方式为继承的方式, 但是企业开发中, 更多是把Feign接口抽取为一个独立的模块 (做法和继承相似, 但理念不同).

操作方法:

将Feign的Client抽取为一个独立的模块, 并把涉及到的实体类等都放在这个模块中, 打成一个Jar. 服务消费方只需要依赖该Jar包即可. 这种方式在企业中比较常见, Jar包通常由服务提供方来实现.

5.2.1 创建一个module

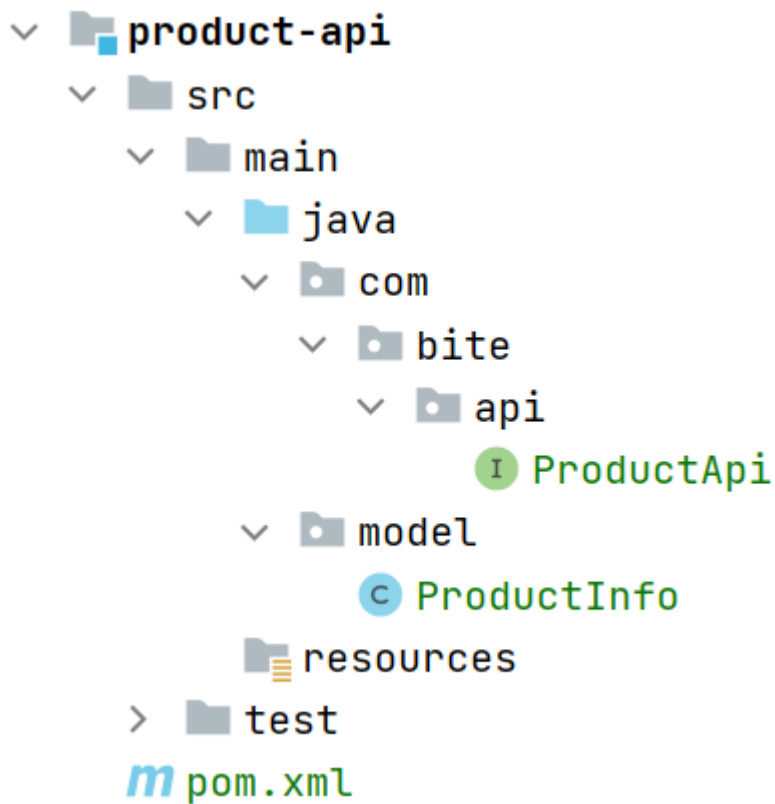


5.2.2 引入依赖

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-starter-openfeign</artifactId>
4 </dependency>
```

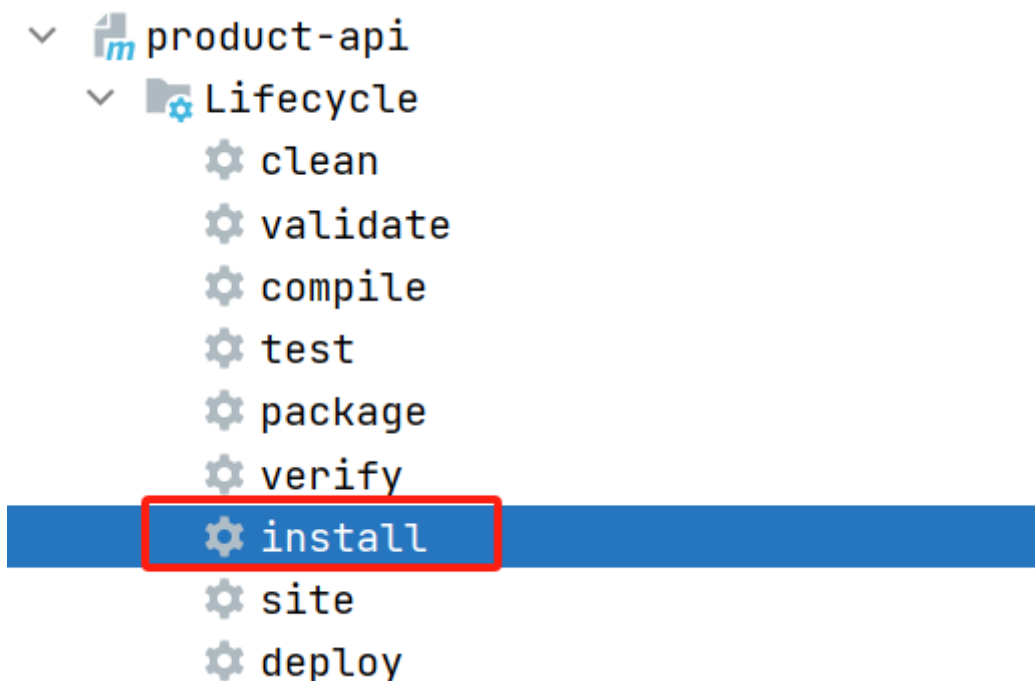
5.2.3 编写API

复制 ProductApi, ProductInfo 到product-api模块中



5.2.4 打Jar包

通过Maven打包



观察Maven本地仓库, Jar包是否打成功

- 1 [INFO] Installing D:\Git\spring-cloud\spring-cloud-feign\product-api\target\product-api-1.0-SNAPSHOT.jar to D:\Maven\.m2\repository\org\example\product-api\1.0-SNAPSHOT\product-api-1.0-SNAPSHOT.jar
- 2 [INFO] -----


```
3 [INFO] BUILD SUCCESS
4 [INFO] -----
5 [INFO] Total time: 3.441 s
6 [INFO] Finished at: 2024-01-03T17:55:14+08:00
7 [INFO] -----
```

此电脑 > Data (D:) > Maven > .m2 > repository > org > example > product-api				
名称	修改日期	类型	大小	
1.0-SNAPSHOT	2024/1/3 17:55	文件夹		
maven-metadata-local.xml	2024/1/3 17:55	XML 文件	1 KB	

5.2.5 服务消费方使用product-api

1. 删除 ProductApi, ProductInfo
2. 引入依赖

```
1 <dependency>
2     <groupId>org.example</groupId>
3     <artifactId>product-api</artifactId>
4     <version>1.0-SNAPSHOT</version>
5 </dependency>
```

修改项目中ProductApi, ProductInfo的路径为product-api中的路径

3. 指定扫描类: ProductApi

在启动类添加扫描路径

```
1 @EnableFeignClients(basePackages = {"com.bite.api"})
```

完整代码如下:

```
1 @EnableFeignClients(basePackages = {"com.bite.api"})
2 @SpringBootApplication
3 public class OrderServiceApplication {
4     public static void main(String[] args) {
5         SpringApplication.run(OrderServiceApplication.class, args);
6     }
7 }
```

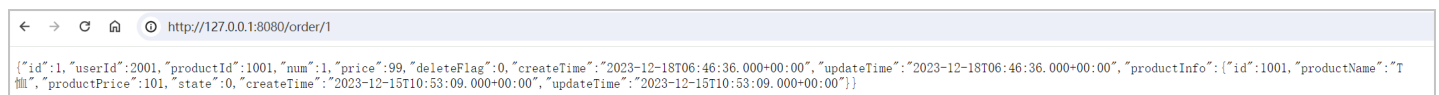
也可以指定需要加载的Feign客户端

```
1 @EnableFeignClients(clients = {ProductApi.class})
```

5.2.6 测试

测试远程调用

<http://127.0.0.1:8080/order/1>



6. 服务部署

1. 修改数据库, Nacos等相关配置

2. 对两个服务进行打包

Maven打包默认是从远程仓库下载的, product-api 这个包在本地, 有以下解决方案:

- 上传到Maven中央仓库(参考: [如何发布Jar包到Maven中央仓库](#), 比较麻烦)[**不推荐**]
- 搭建Maven私服, 上传Jar包到私服[**企业推荐**]
- 从本地读取Jar包[**个人学习阶段推荐**]

前两种方法比较复杂, 咱们使用第三种方式

修改pom文件

```
1 <dependency>
2     <groupId>org.example</groupId>
3     <artifactId>product-api</artifactId>
4     <version>1.0-SNAPSHOT</version>
5     <!-- scope 为system. 此时必须提供systemPath即本地依赖路径. 表示maven不会去中央
      仓库查找依赖 不推荐使用-->
6     <scope>system</scope>
7     <systemPath>D:/Maven/.m2/repository/org/example/product-api/1.0-
      SNAPSHOT/product-api-1.0-SNAPSHOT.jar</systemPath>
8 </dependency>
9
10
11 //....
12 <build>
13     <plugins>
```

```

14         <plugin>
15             <groupId>org.springframework.boot</groupId>
16             <artifactId>spring-boot-maven-plugin</artifactId>
17             <configuration>
18                 <includeSystemScope>true</includeSystemScope>
19             </configuration>
20         </plugin>
21     </plugins>
22 </build>

```

把D:/Maven/.m2/repository 改为本地仓库的路径

3. 上传jar到Linux服务器

4. 启动Nacos

启动前最好把data数据删除掉.

5. 启动服务

```

1 #后台启动order-service, 并设置输出日志到logs/order.log
2 nohup java -jar order-service.jar >logs/order.log &
3
4 #后台启动product-service, 并设置输出日志到logs/order.log
5 nohup java -jar product-service.jar >logs/product-9090.log &
6
7 #启动实例, 指定端口号为9091
8 nohup java -jar product-service.jar --server.port=9091 >logs/product-9091.log &

```

观察Nacos控制台

The screenshot shows the Nacos 2.2.3 console interface. The top navigation bar includes links for 首页 (Home), 文档 (Documentation), 博客 (Blog), 社区 (Community), and Nacos企业版 (Nacos Enterprise Edition). The left sidebar contains a menu with 配置管理 (Configuration Management), 服务管理 (Service Management), 服务列表 (Service List), 订阅者列表 (Subscriber List), 命名空间 (Namespace), and 集群管理 (Cluster Management). The main content area is titled '服务列表' (Service List) and shows a table of services. The table has columns for 服务名 (Service Name), 分组名称 (Group Name), 集群数目 (Cluster Count), 实例数 (Instance Count), 健康实例数 (Healthy Instance Count), 触发保护阈值 (Trigger Protection Threshold), and 操作 (Operations). The table lists two services: 'product-service' and 'order-service', both in the 'DEFAULT_GROUP' with 1 cluster and 1 instance each. The '操作' column provides links for '详情' (Details), '示例代码' (Sample Code), '订阅者' (Subscriber), and '删除' (Delete). The bottom right of the interface shows pagination controls: '每页显示: 10' (Show 10 per page) and '1' (Page 1 of 1).

6. 测试

访问接口: <http://110.41.51.65:8080/order/1>

观察远程调用的结果:

```
{
  "id": 1,
  "userId": 2001,
  "productId": 1001,
  "num": 1,
  "price": 99,
  "deleteFlag": 0,
  "createTime": "2023-12-29T10:13:53.000+00:00",
  "updateTime": "2023-12-29T10:13:53.000+00:00",
  "productInfo": {
    "id": 1001,
    "productName": "T恤",
    "productPrice": 101,
    "state": 0,
    "createTime": "2023-12-29T10:14:24.000+00:00",
    "updateTime": "2023-12-29T10:14:24.000+00:00"
  }
}
```