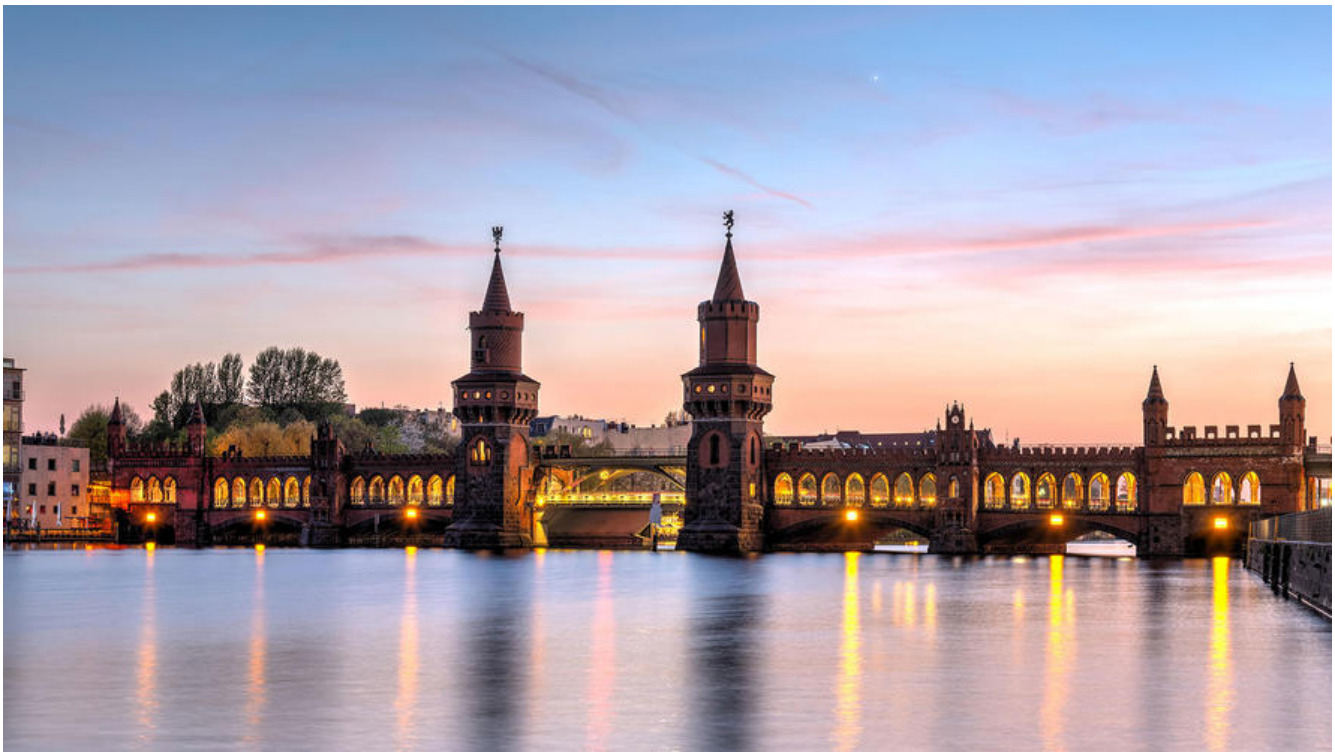
链滴

SpringCloud Alibaba 微服务实战三 - 服务调用

作者：jianzh5

原文链接：https://ld246.com/article/1575599039590

来源网站：链滴

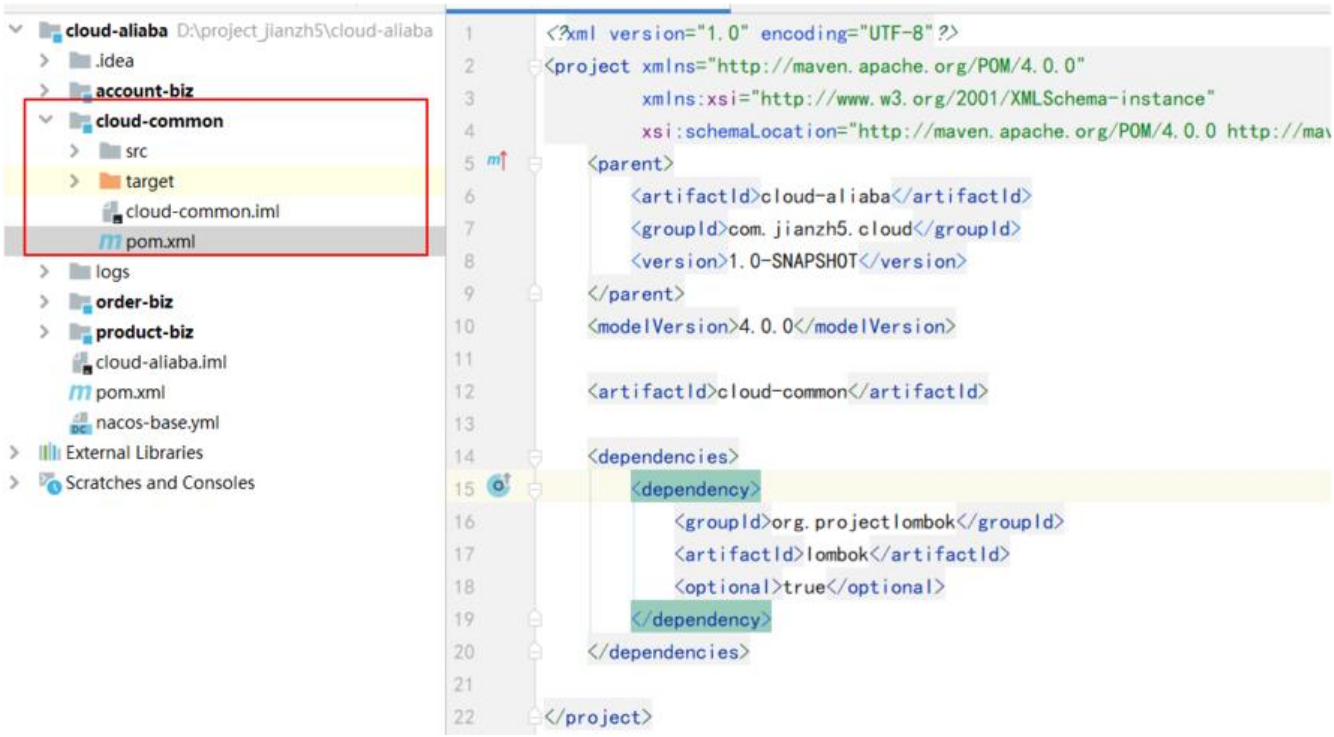


导读：通过前面两篇文章我们准备好了微服务的基础环境并让accout-service 和 product-service对提供了增删改查的能力，本篇我们的内容是让order-service作为消费者远程调用accout-service和product-service的服务接口。

统一接口返回结构

在开始今天的正餐之前我们先把上篇文章中那个丑陋的接口返回给优化掉，让所有的接口都有统一的回结构。

- 建立公共模块cloud-common



- 其他模块都引入cloud-common,修改pom文件，加入依赖

```
<dependency>
    <groupId>com.jianzh5.cloud</groupId>
    <artifactId>cloud-common</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

- 建立接口返回的数据结构，这个数据结构大家可以根据自身项目情况统一即可

```
@Data
public class ResultData<T> {
    /** 结果状态 ,正常响应200，其他状态码都为失败*/
    private int status;
    private String message;
    private T data;
    private boolean success;
    private long timestamp ;

    ... 提供一些静态方法 ...
}
```

- 改造 accout-service 和 product-service 模块中controller层的返回结构，改造完的代码如下：

```
@RestController
@Log4j2
public class AccountController {
    @Autowired
    private AccountService accountService;
    @PostMapping("/account/insert")
    public ResultData<String> insert(@RequestBody AccountDTO accountDTO){
        log.info("insert account:{}",accountDTO);
        accountService.insertAccount(accountDTO);
        return ResultData.success("insert account succeed");
    }
    @PostMapping("/account/delete")
    public ResultData<String> delete(@RequestParam String accountCode){
        log.info("delete account,accountCode is {}",accountCode);
        accountService.deleteAccount(accountCode);
        return ResultData.success("delete account succeed");
    }
    @PostMapping("/account/update")
    public  ResultData<String> update(@RequestBody AccountDTO accountDTO){
        log.info("update account:{}",accountDTO);
        accountService.updateAccount(accountDTO);
        return ResultData.success("update account succeed");
    }
    @GetMapping("/account/getByCode/{accountCode}")
    public ResultData<AccountDTO> getByCode(@PathVariable(value = "accountCode") String
accountCode){
        log.info("get account detail,accountCode is :{}",accountCode);
        AccountDTO accountDTO = accountService.selectByCode(accountCode);
        return ResultData.success(accountDTO);
    }
```

```
}
```

服务调用

在SpringCloud体系中，所有微服务间的通信都是通过Feign进行调用，Feign是一个http请求调用的量级框架，可以以Java接口注解的方式调用Http请求，而不用像使用HttpClient、OKHttp3等组件通封装HTTP请求报文的方式调用。Feign通过处理注解，将请求模板化，当实际调用的时候，传入参数根据参数再应用到请求上，进而转化成真正的请求，这种请求相对而言比较直观。而且Feign默认集了负载均衡器Ribbon，不需要自己实现负载均衡逻辑。

Feign是SpringCloud的组件，在引入Feign之前我们先看看Spring Boot， Spring Cloud，Spring C oud Alibaba 三者之间的关系，防止在业务中引入了错误的版本。

Spring Boot	Spring Cloud	Spr ng Cloud Alibaba
2.1.x	Greenwich	0.9.x
2.0.x	Finchley	0.2.x
1.5.x	Edgware	0.1.x
1.5.x	Dalston	0.1.x

很显然，我们引用的是SpringCloud Alibab 0.9.0,所以我们需要引入SpringCloud Greenwich 。

- 引入SpringCloud版本依赖

在项目主pom <dependencyManagement>中引入SpringCloud依赖

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-dependencies</artifactId>
   <version>Greenwich.SR2</version>
   <type>pom</type>
   <scope>import</scope>
</dependency>
```

- 在所有需要用到 Feign的模块中引入openfeign依赖

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

- 抽取公共的接口层，建立 Feign接口，接口的定义和返回值需要跟Controller层保持一致

```
@FeignClient(name = "account-service")
public interface AccountFeign {
   @PostMapping("/account/insert")
   ResultData<String> insert(@RequestBody AccountDTO accountDTO);

   @PostMapping("/account/delete")
   ResultData<String> delete(@RequestParam("accountCode") String accountCode);

   @PostMapping("/account/update")
```

```
    ResultData<String> update(@RequestBody AccountDTO accountDTO);

    @GetMapping("/account/getByCode/{accountCode}")
    ResultData<AccountDTO> getByCode(@PathVariable(value = "accountCode") String acco
ntCode);
}
```

在接口上添加注解@FeignClient(name = "account-service")，表明这是一个Feign客户端，name性的配置表示我这个接口最终会转发到accout-service上。

正如回字有多种写法，这里Feign也有多种使用方式。

第一种就是我们这里介绍的，**Feign和生产者的RequestMapping保持一致**，大家可以看看上面改后的AccountController 和 AccountFeign一模一样有米有。

第二种方式就是让我们的Controller**直接实现Feign接口，不再需要写RequestMapping**，如：

```
@RestController
@Log4j2
public class ProductController implements ProductFeign {
    @Autowired
    private ProductService productService;

    @Override
    public ResultData<String> insert(@RequestBody ProductDTO productDTO){
        log.info("insert product:{}",productDTO);
        productService.insertProduct(productDTO);
        return ResultData.success("insert product succeed");
    }
}
```

- 消费者模块引入 Feign接口层的依赖

```
<dependency>
    <groupId>com.jianzh5.cloud</groupId>
    <artifactId>account-feign</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

- 在消费者 product-service启动类上添加@EnableFeignClients注解

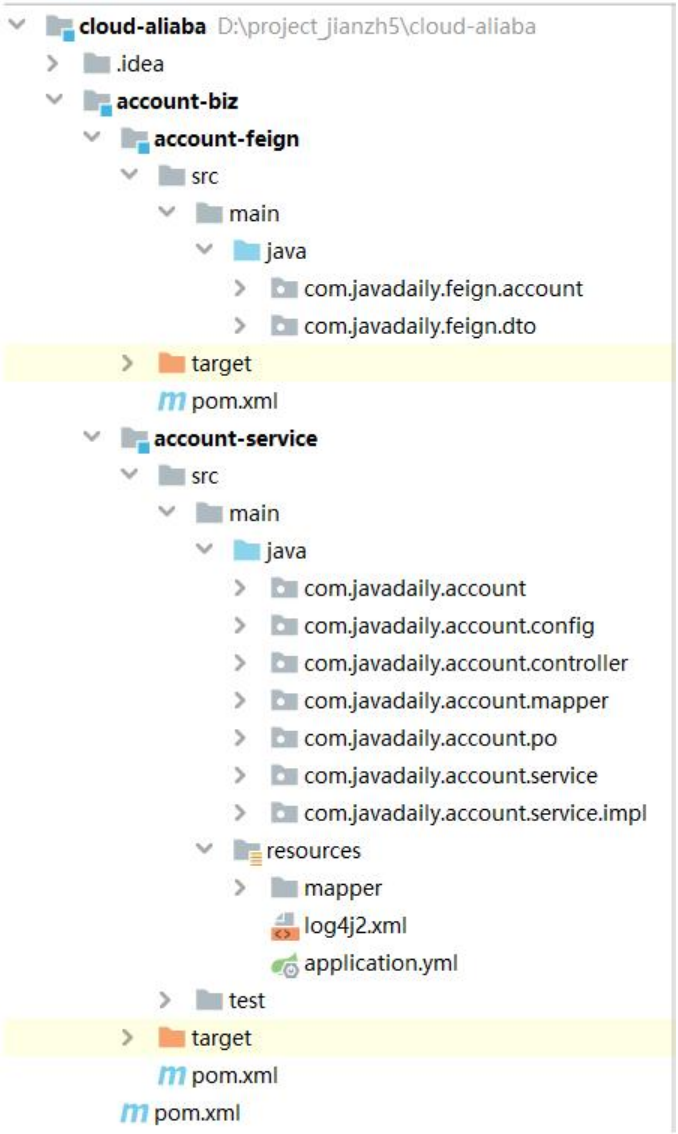
```
@SpringBootApplication
@RestController
@EnableDiscoveryClient
@EnableFeignClients(basePackages = "com.javadaily.feign.*")
public class OrderServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(OrderServiceApplication.class, args);
    }
}
```

- 消费者端跟使用本地service一样使用Feign

```
@RestController
public class OrderController {
```

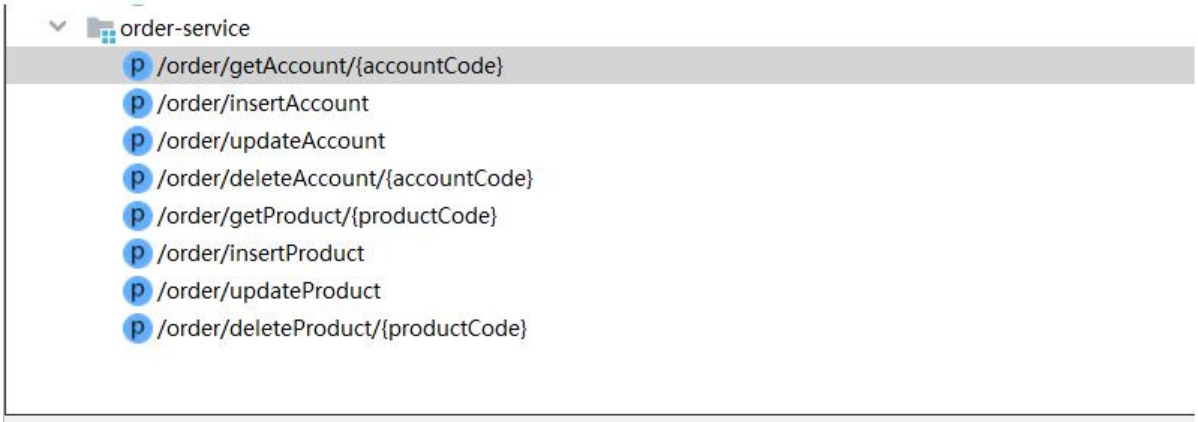
```
    @Autowired
    private AccountFeign accountFeign;

    @Autowired
    private ProductFeign productFeign;

    @PostMapping("/order/getAccount/{accountCode}")
    public ResultData<AccountDTO> getAccount(@PathVariable String accountCode){
        return accountFeign.getByCode(accountCode);
    }

    @PostMapping("/order/insertAccount")
    public ResultData<String> insertAccount(AccountDTO accountDTO){
        return accountFeign.insert(accountDTO);
    }

    @PostMapping("/order/updateAccount")
    public ResultData<String> updateAccount(AccountDTO accountDTO){
        return accountFeign.update(accountDTO);
    }

    @PostMapping("/order/deleteAccount/{accountCode}")
    public ResultData<String> deleteAccount(@PathVariable String accountCode){
        return accountFeign.delete(accountCode);
    }
}
```

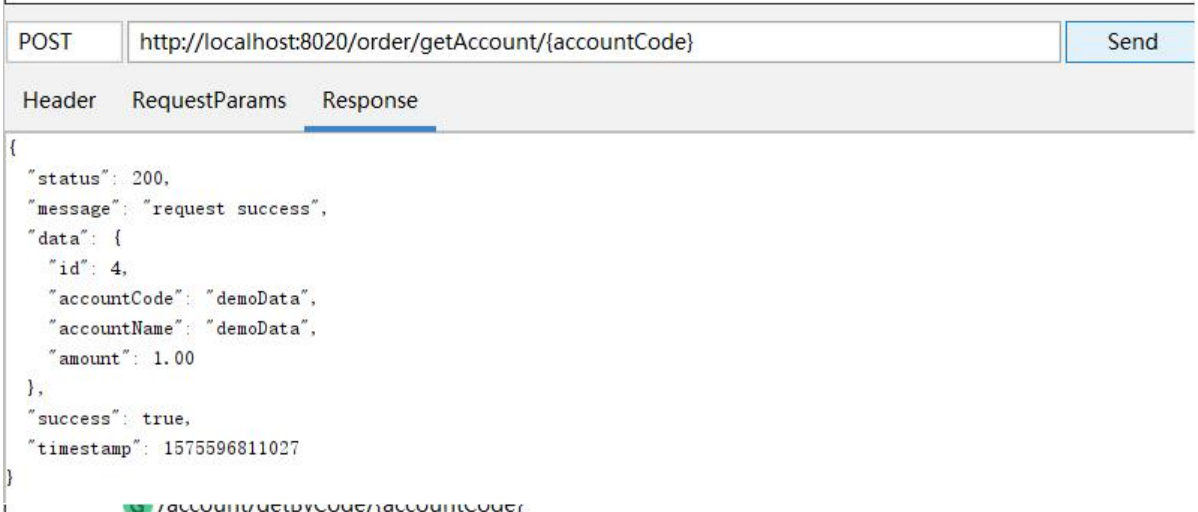
- 项目模块截图

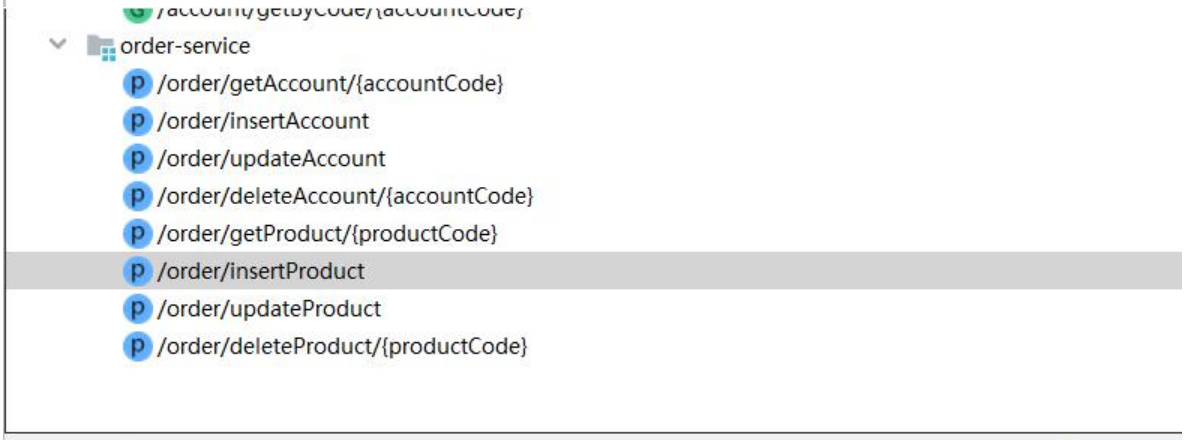


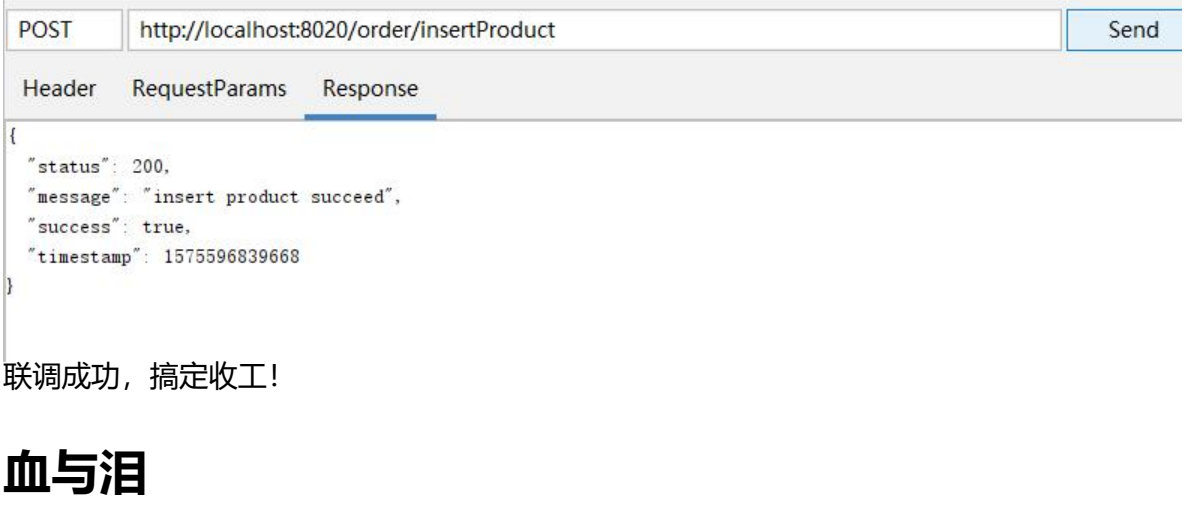
- 联调测试

我们请求OrderController中的接口，验证下接口请求结果：









联调成功，搞定收工！

血与泪

使用feign过程中有以下几点需要注意，否则一不小心你就会掉进坑里。（我不会告诉你我当时在坑踩了多久才爬上来）

- **Feign不支持直接使用对象作为参数请求**

接口中如果有多参数需要用实体接收，要么把参数一个一个摆开，要么在对象参数上加上@RequestBody注解，让其以json方式接收，如：

```
@PostMapping("/account/insert")ResultData<String> insert(@RequestBody AccountDTO accountDTO);
```

- **消费者模块启动类上使用@EnableFeignClients注解后一定要指明Feign接口所在的包路径**

如：@EnableFeignClients(basePackages = "com.javadaily.feign.*")

否则你的消费者启动时会报如下的错误：

```
The injection point has the following annotations:
    - @org.springframework.beans.factory.annotation.Autowired(required=true)


Action:

Consider defining a bean of type 'com.javadaily.feign.account.AccountFeign' in your configuration.
```

所以这里推荐你们在开发中所有feign模块最好能统一包名前缀com.javadaily.feign

- **@RequestParam的坑**

在Feign接口层使用@RequestParam注解要注意，一定要加上value属性，如：

```
ResultData<String> delete(@RequestParam("accountCode") String  accountCode);
```

否则你会看到类似如下的错误：

```
Caused by: java.lang.IllegalStateException: RequestParam.value() was empty on parameter 0
```

个异常

- **@PathVariable的坑**

在Feign接口层使用@PathVariable注解要注意，一定要跟上面一样加上value属性,如：

```
ResultData<AccountDTO> getByCode(@PathVariable(value =  "accountCode") String accoun
Code);
```

否则你也会看到类似如下的错误@PathVariable(value = "accountCode") String accountCode

- **在消费者配置文件中添加Feign超时时间配置**

在我们的order-service配置文件中增加feign超时时间配置

```
feign:
  client:
    config:
      default:
        connectTimeout: 5000
        readTimeout: 5000
```

否则你会经常看到如下所示的错误：

```
java.net.SocketTimeoutException: Read timed out
    at java.net.SocketInputStream.socketRead0(Native Method) ~[?:1.8.0_112]
```

至此我们已经完成了项目公共返回接口的统一并且成功使用Feign调用远程生产者的服务，那么本期“SpringCloud Alibaba微服务实战三 - 服务调用”篇也就该结束啦，咱们下期有缘再见！