一 服务拆分的方法论

1 每个微服务都需要有自己的数据库,不同微服务间的数据调用通过调用对方所暴露的api来得到需要的数据,而不能访问对方的数据库

2 通过服务的类型选择不同的数据库的类型

如对事务的要求不高的展示型的可以使用mogoDB

对于查询要求比较高的可以用es

对于事务要求比较高的可以用常见的关系型数据库

3 难点在于确定边界

不同的服务对于数据的关注点不同,如订单服务关心用户是否有效,而积分服务关心用户注册时间,所以对于用户的数据,不同的服务有不同的关注点

二 springCloud 采用轻量级http进行调用,有两种调用方式,

1 RestTemplate

缺点 地址写死,如果有多个生产者无法负载均衡,且很多时候其实并不知道生产者的地址

2 feign

三 Ribbon

1 通过服务的名字发现在这个名字下的所有的服务的地址

2 通过一些负载均衡策略进行服务器的挑选

3 高效监听各个服务器的情况,挑选正常的服务器

4 Ribbon的主要组件

4.1 serviceList 这个组件主要用于找到某个服务名下所有的服务的地址列表

4.2 serviceListFilter 用于过滤掉一部分服务

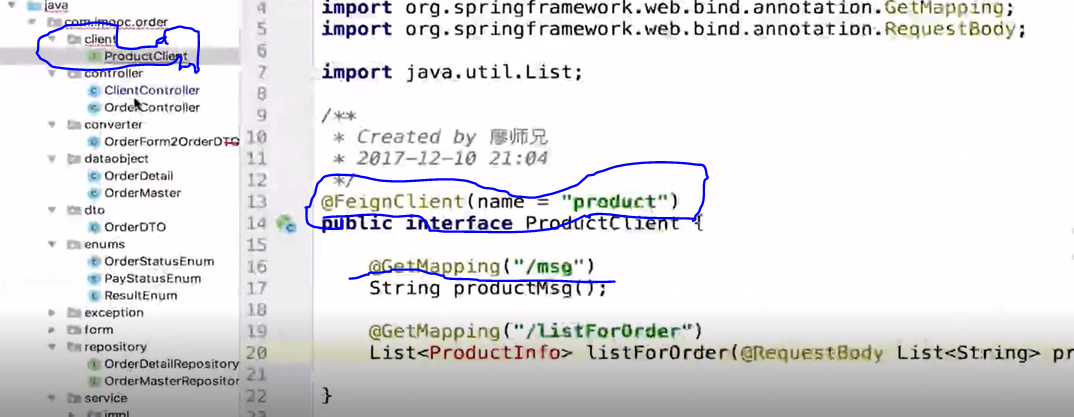
4.3 IRule 这个组件用于定义负载均衡策略,可自定义

4.4 过程,------->serviceList发现所有服务的地址--->serviceListFilter过滤掉一些服务----->IRule通过策略算法找到具体的微服务进行调用(默认是轮询)

5 feign

Feign内部集成了Ribbon,它采用声明试接口的做法在一个微服务端声明需要调用的另一个微服务的相同的接口然后feign它自己就能调用到这个微服务端的接口,

然后引入这个类,可直接调用微服务的接口的方法,



其中的内部原理在于feign会根据声明的接口上的注解(标注蓝色)来找到该微服务然后去调用,其中@feignClient:表示需要调用的微服务的名字,@Getmapping为需要调用的接口,除此之外,feign还使用了Ribbon作为负载均衡

6 Feign的实际的用法,

假设我们有两个模块商品和订单,订单模块需要调用商品模块的接口,

代码应该这样写

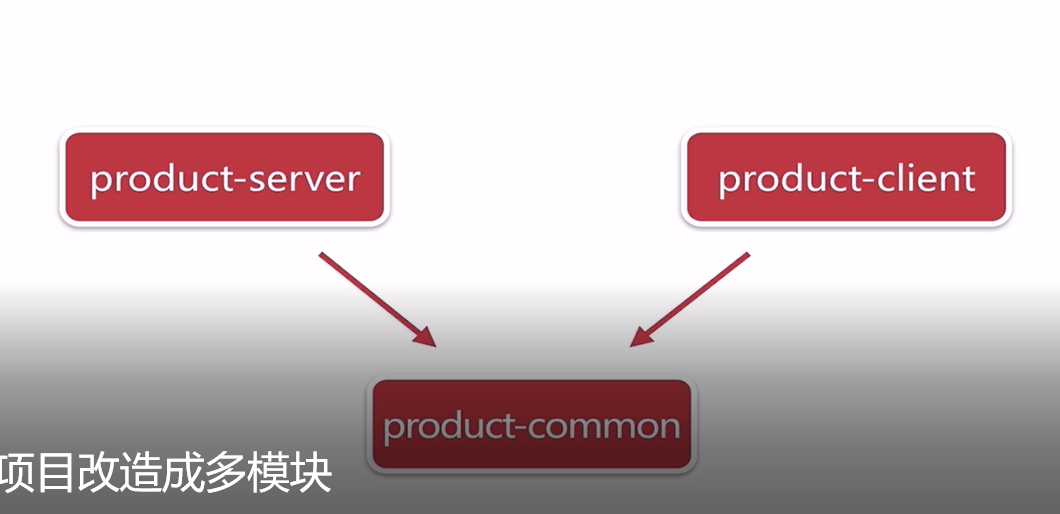
1分模块,所有的微服务应该分为Client模块,sever模块,common模块

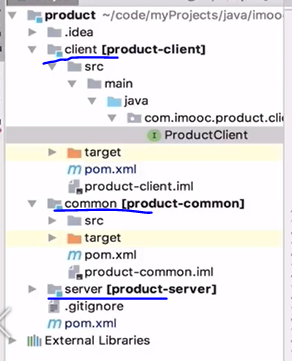
A Client模块:用于feign的客户端,暴露服务的接口需要引入common

B sever模块:和普通的单体服务差不多,用于数据访问和业务处理,一直到controller层,需要引入common

C common模块:用于存放这个微服务公工的东西

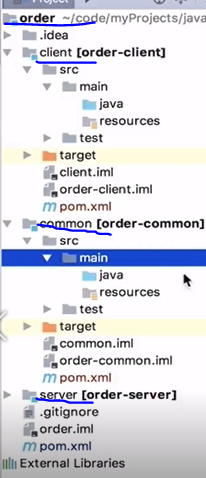
他们的依赖关系如下

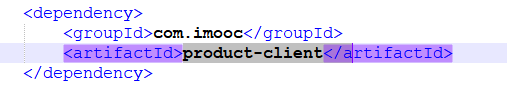


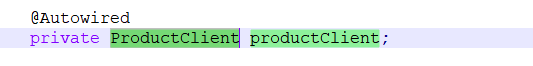


2 将被调用模块(商品模块)build,然后安装进入自己的仓库

3 消费者微服务(订单微服务)同样需要分模块,client无内容,因为他不会被调用,而消费者用pom文件引入生产者(商品微服务),然后使用生产者的client来进行调用生产者微服务.







7 异步调用

商品和订单之间需要加一个消息中间件,如rabbitmq,当有人下单需要扣库存的时候,商品服务只需发布消息到rabbitmq,然后商品服务就能够拿到消息,然后进行下一步操作,能大大加快订单下单的速度,

两个微服务的异步化能够大大加快响应的速度,如订单与积分,订单与邮件,订单与短信等,这样的化下单只需将相关信息写入队列,然后马上响应完成,而后续任务其他微服务会在消息队列里面取出消息,然后完成任务,

除此之外,商品抢购的流量肖锋也很有用

Docker 的好处

1解决了环境部署问题,解决了不同服务器环境不同导致的微服务出现问题

2 轻量级,环境隔离

3 可通过镜像文件来进行环境的传播,在部署的时候直接拷贝镜像