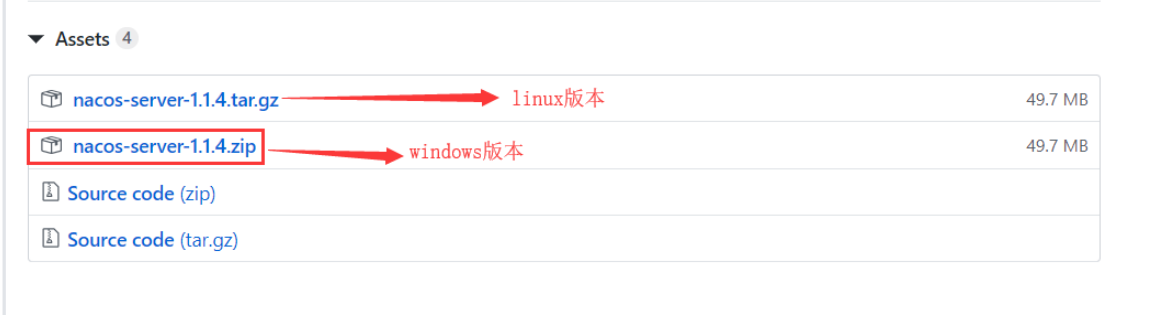
# 1、nacos

## 1.1nacos的安装部署

下载地址:https://github.com/alibaba/Nacos/releases



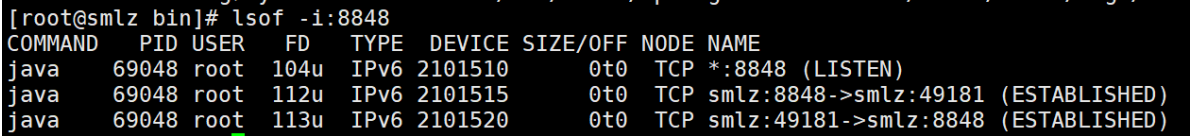
### 1.1.1 nacos单机安装

1、把我们的Nacos包解压 tar -zxvf nacos-server-1 .1 .4.tar.gz；

2、进入冬解压目录nacos；

3、进入到bin目录下 执行命令(启动单机) sh startup.sh -m standalone；

4、检查nacos启动的端口 lsof -i:8848



5、访问nocas的服务端 http://1 92.1 68.1 59.8:8848/nacos/index.html

6、停止nocas 在nocas/bin目录下 执行 sh shutdown.sh

### 1.1.2 nacos集群搭建

Nacos集群使用nginx做负载

#### 1、安装nginx

1、wget [http://nginx.org/download/nginx-1 .1 4.2.tar.gz](http://nginx.org/download/nginx-1%20.1%204.2.tar.gz)

2、tar -xzvf nginx-1 .1 4.2.tar.gz 得到解压目录

3、进入解压目录，./configure ­­prefix=/usr/local/nginx­1 .1 4.2 意思是告诉等会安装的文件要放在哪里

4、执行make命令进行编译

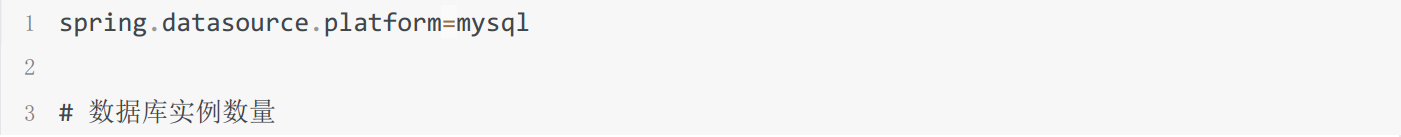
5、执行make install

6、进入安装目录修改conf文件



#### 2、搭建nacos集群

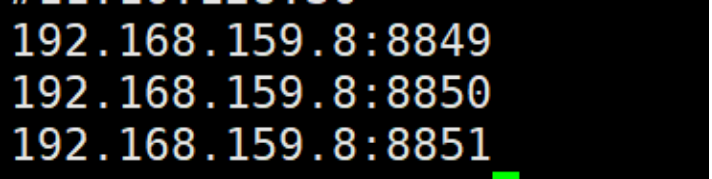
我们以配置一台为例（8849） 为例  
第一步:修改nacos8849/conf文件 application.properties





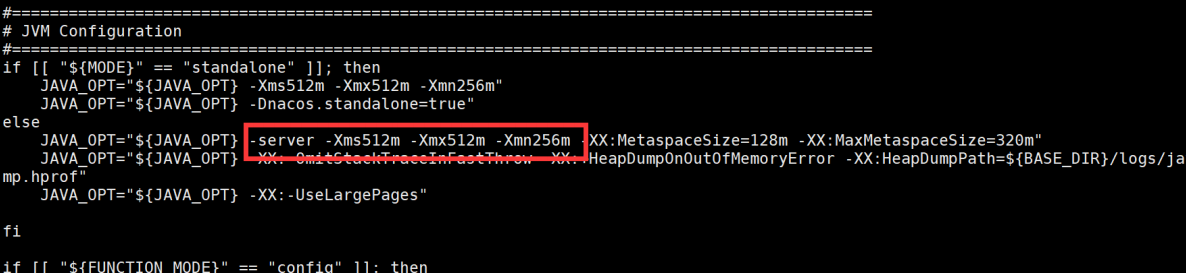
第二步:修改nacos8859/conf文件 把原来的cluster.conf.example改为cluster.conf文件

在文件内配置集群各个节点的端口号

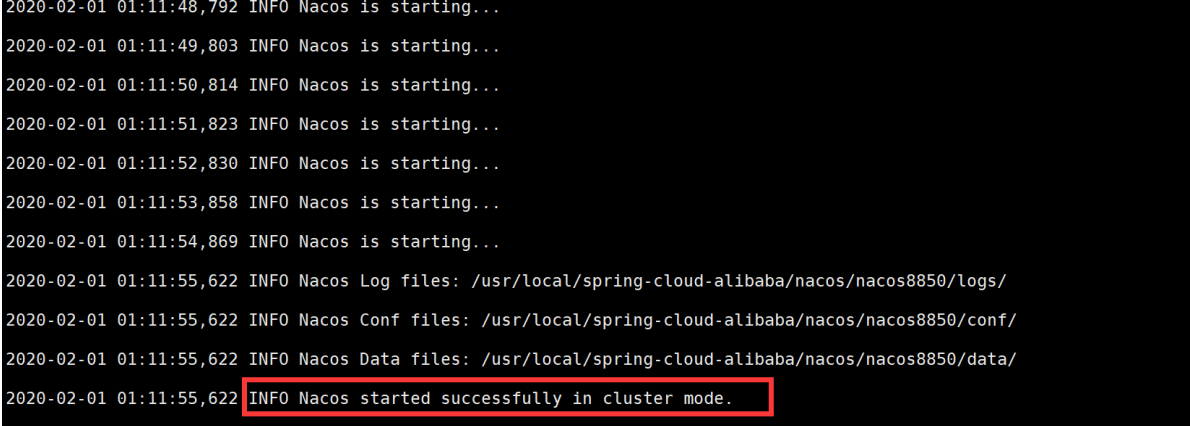


3、最后一步： 创建一个数据库（需要自己创建一个数据库） 脚本的位子在 nacos/conf/nacosmysql.sql（nacos使用内置nerby存储服务以及配置信息，在集群模式下各个节点需要通讯保持数据一致性，此时将数据存储于MySQL中用于数据同步）

4、需要修改nacos-server的 启动脚本jvm参数 (怕你们内存参数设置的过小启动不了这么多服务)



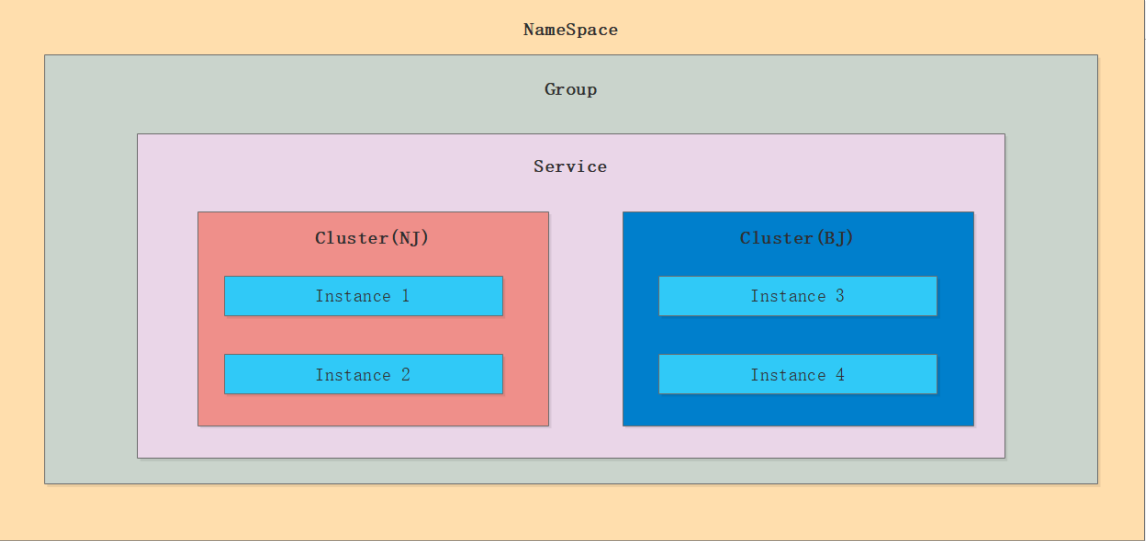
5、分别进入各个节点的bin目录下启动节点，执行./start.sh启动成功的依据 查看日志/usr/local/spring-cloud-alibaba/nacos/nacos8850/logs/start.out



NG测试  
http://1 92.1 68.1 59.8:8847/nacos/

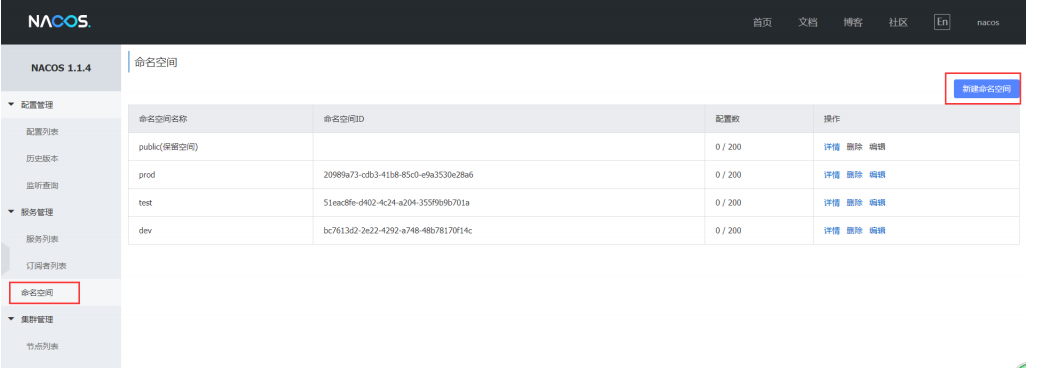


### 1.1.3 Nacos 领域模型划分以及概念详解





Namespace：代表开发环境，例如开发、测试、生产，(默认的NameSpace是” public“ NameSpace可以进行资源隔离， 比如我们dev环境下的NameSpace下的服 务是调用不到prod的NameSpace下的微服务)

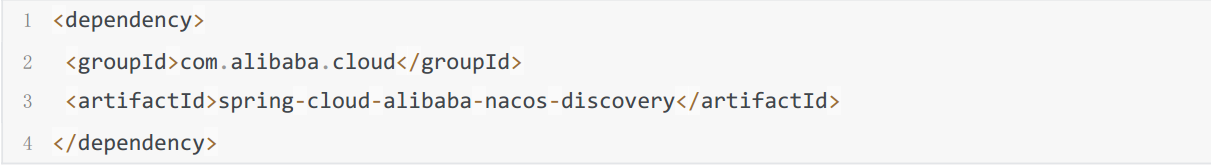


Group：分组，某个大功能，或者功能衔接的几个服务，默认为default

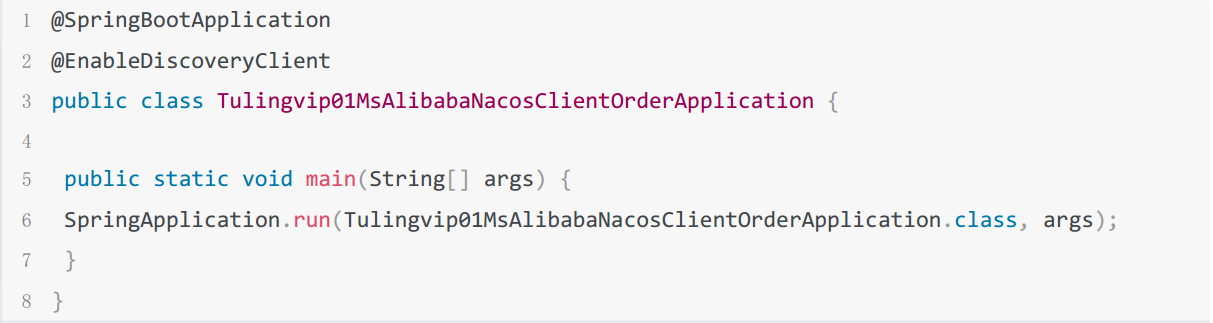
Cluster：单个服务的集群，可以根据部署地点将单一服务分为多个集群

## 1.2 NACOS的注册发现

1、加入依赖

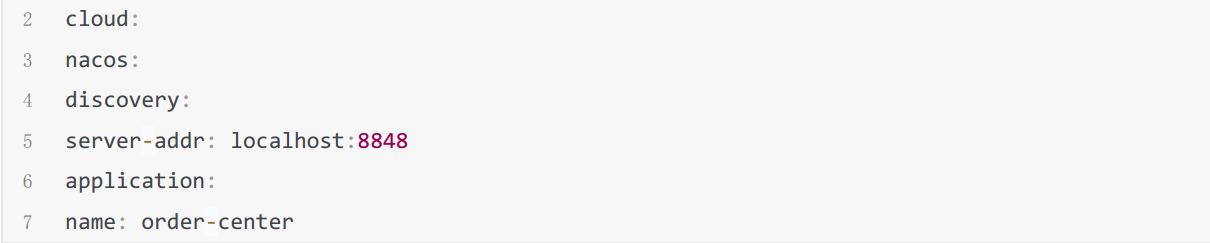


2、写注解，也可以不写



3、配置文件





## 1.3 配置中心

不同的环境使用不同配置，配置属性的动态刷新

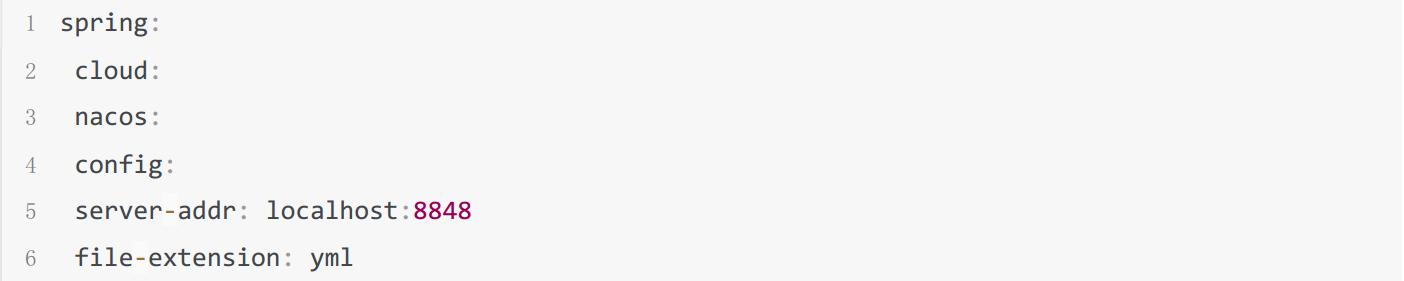
### 3.1 引入配置在中心

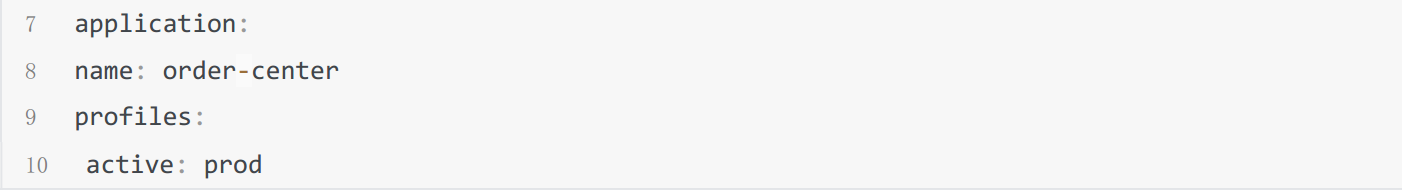
1、加入依赖

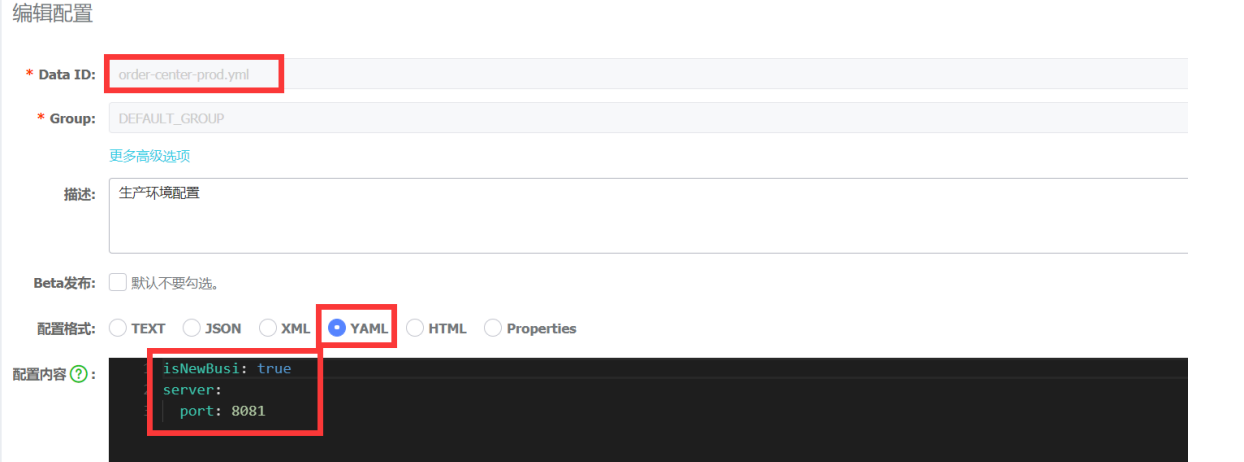


2、编写配置文件,需要写一个bootstrap.yml配置文件

server-addr: localhost:8848 表示我微服务怎么去找我的配置中心  
spring.application.name=order-center 表示当前微服务需要向配置中心索要order-center  
的配置  
spring.profiles.active=prod 表示我需要向配置中心索要order-center的生产环境的配置  
索要文件的格式为  
${application.name}- ${spring.profiles.active}.${file-extension}  
真正在nacos配置中心上 就是 order-center-prod.yml

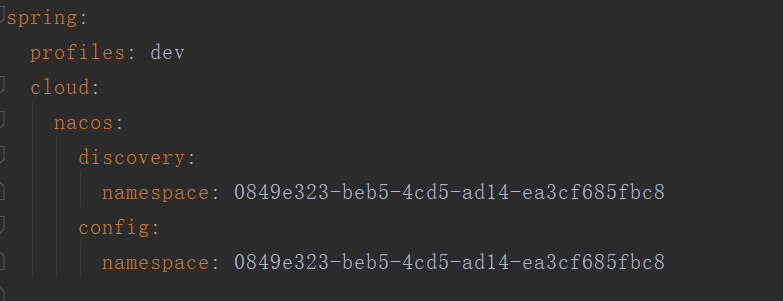






如果有需要动态的变量，则需要在controller类上添加@RefreshScope注解

**如果每个微服务均独自引入配置中心，则服务发现与配置中心均需配置namespace**



### 3.2 配置分级

精准配置>不同环境的通用配置>不同工程的(ext-config)>不同工程(shared- dataids)

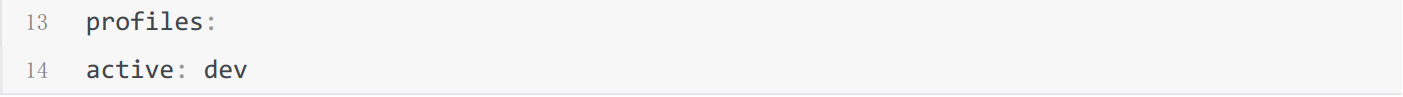
配置的优先级 精准配置 会覆盖 与通用配置 相同的配置， 然后再和通用配置互补

1、同一微服务不同环境的通用配置，创建的的配置id直接为微服务名.yml

2、不同微服务通过 shared-dataids 方式做通用配置

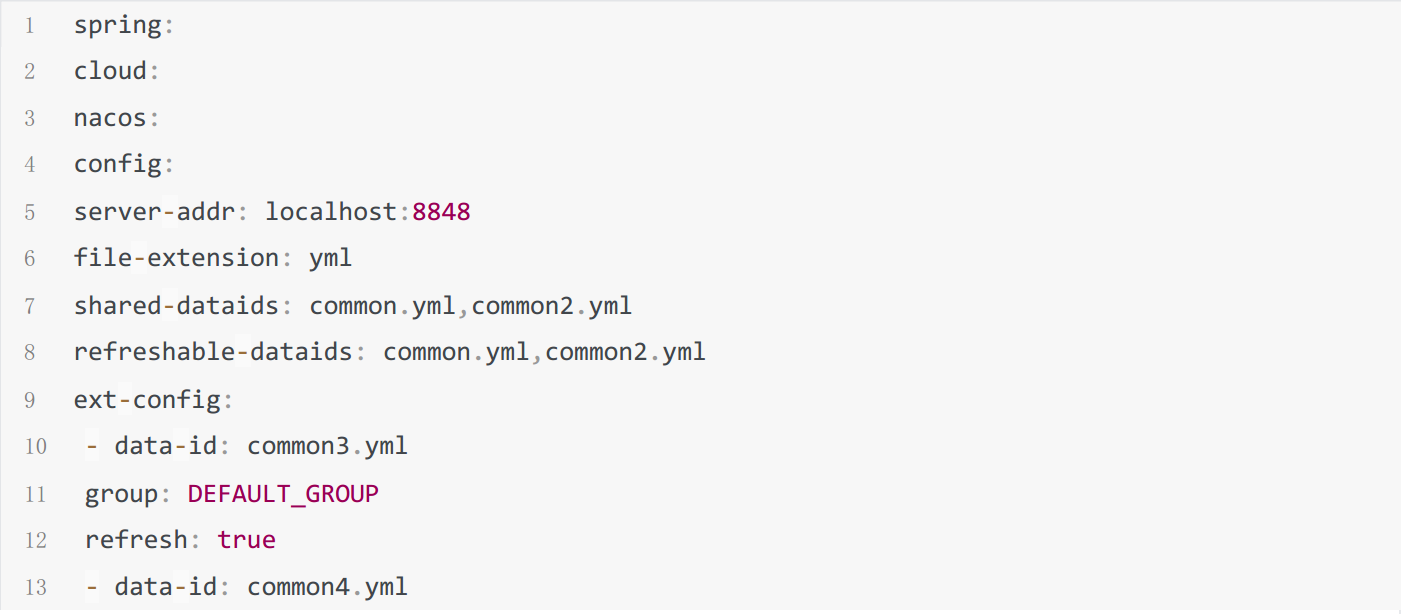
我们可以把每个微服务均配置的相同配置抽取出来作为通用配置之

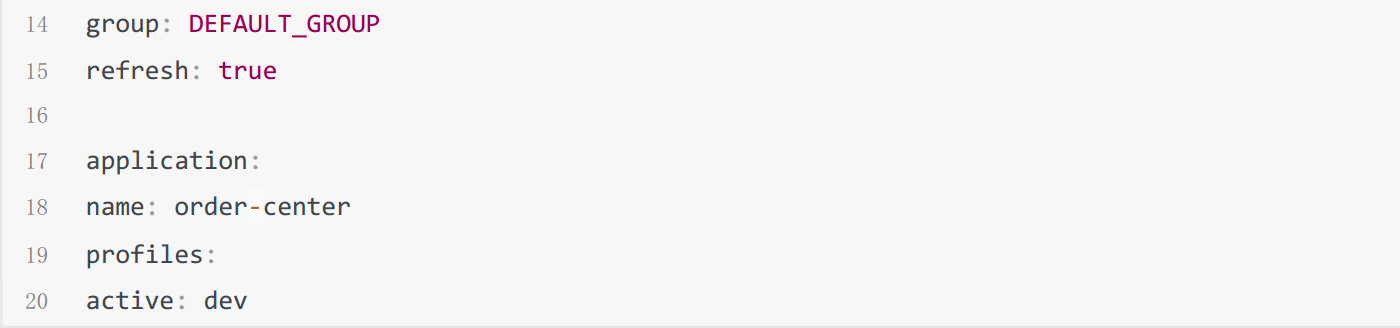




3、不通微服务通过ext-config方式，同样配置到越后面的配置 优先级越高







上述配置 加载的优先级  
1 )order-center-dev.yml 精准配置  
2)order-center.yml 同工程不同环境的通用配置  
3)ext-config: 不同工程 通用配置  
3.1 )： common4.yml  
3.2): common3.yml  
4) shared-dataids 不同工程通用配置  
4.1 )common2.yml  
4.2)common1 .yml

# 2、ribbon

spring cloud ribbon是 基于NetFilix ribbon 实现的一套客户端的负均衡工具,Ribbon客户端组件提供一系列的完善的配置， 如超时， 重试等。通过Load Balancer（LB） 获取到服务提供的所有机器实例， Ribbon会自动基于某种规则(轮询， 随机)去调用这些服务。 Ribbon也可以实现我们自己的负载均衡算法。其默认算法为轮询加zone，但国内没有时钟zone，所以默认为轮询

## 2.1 Ribbon的使用

### 2.1.1 引入ribbon

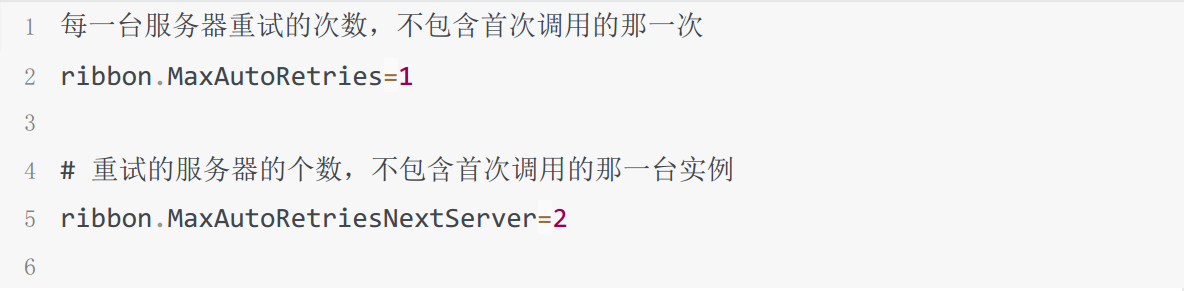
1、引入依赖加入nocas-client和ribbon的依赖



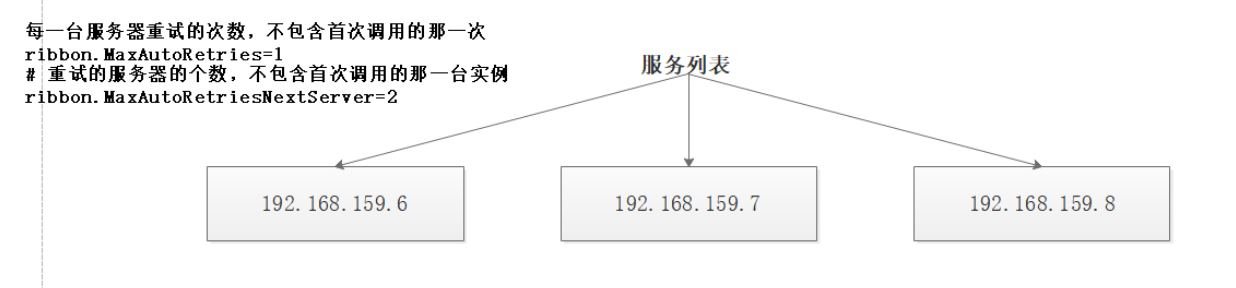
2、写注解: 在RestTemplate上加入@LoadBalanced注解



3、配置信息

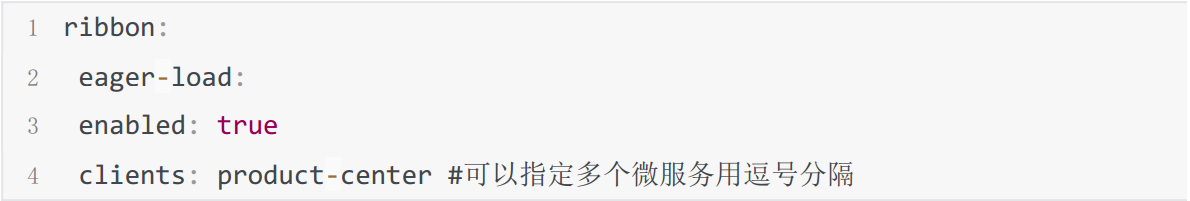




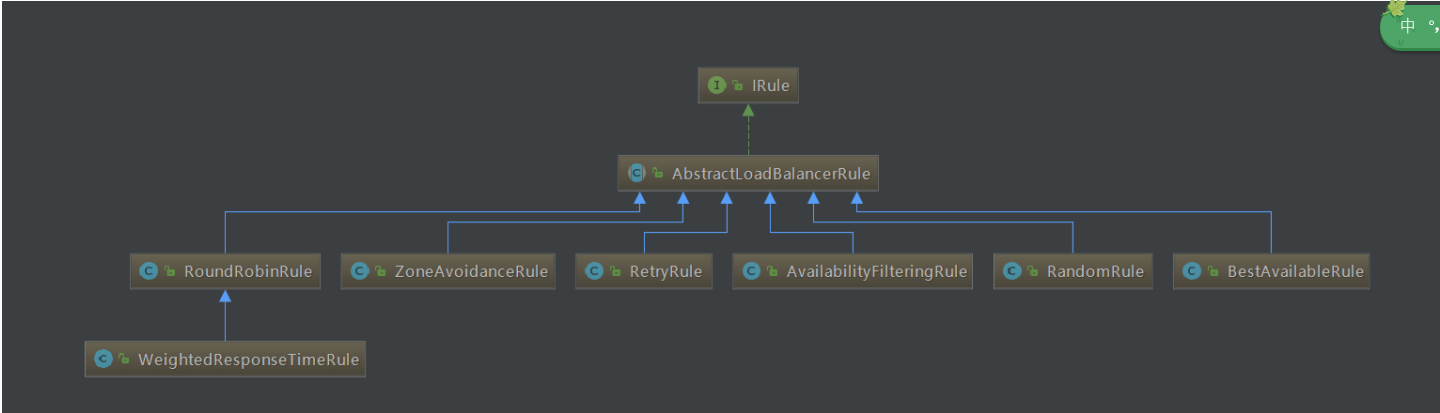


Ribbon详细配置:http://c.biancheng.net/view/5356.html

解决Ribbon 第一次调用耗时高的配置  
开启饥饿加载



### 2.1.2 ribbon 的负载均衡策略



①:RandomRule(随机选择一个Server)  
②:RetryRule 对选定的负载均衡策略机上重试机制， 在一个配置时间段内当选择Server不

成功，则一直尝试使用subRule的方式选择一个可用的server.  
③:RoundRobinRule 轮询选择，轮询index，选择index对应位置的Server

④:AvailabilityFilteringRule 过滤掉一直连接失败的被标记为circuit tripped的后端Server， 并过滤掉那些高并发的后端 Server或者使用一个AvailabilityPredicate来包含过滤server的逻辑， 其实就就是检查 status里记录的各个Server的运行状态

⑤:BestAvailableRule 选择一个最小的并发请求的Server， 逐个考察Server， 如果Server被tripped了，则跳过。

⑥:WeightedResponseTimeRule 根据响应时间加权， 响应时间越长， 权重越小， 被选中的可能性越低；

⑦:ZoneAvoidanceRule（默认是这个） 复合判断Server所在Zone的性能和 Server的可用性选择Server， 在没有Zone的情况下类似轮询

### 2.2.3 robbin的细粒度配置

1、使用yml的方式为每个微服务设置各自的负载均衡策略（推荐使用）

serviceName:  
ribbon: NFLoadBalancerRuleClassName: 负载均衡的对应class的全类名 配置案例: 我们的order-center调用我们的product-center



2、基于代码配置

细粒度的配置类不能放在主启动类相同的包或者其子包内，不然起不到细粒度配置的作用





推荐:推荐同学们使用yml的配置方法配置(没有坑)

极力不推荐同学们 使用二种配置混合使用

### 2.2.4 ribbon自定义负载均衡策略

1、自定义一个权重负载均衡策略

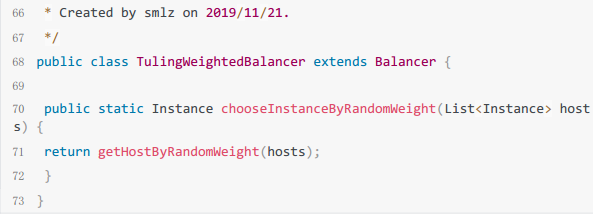




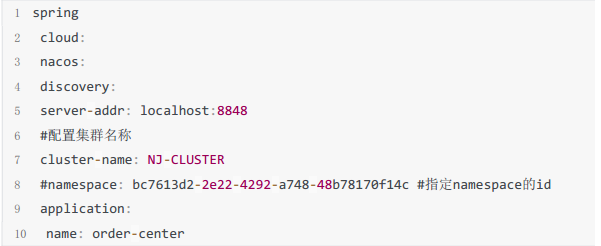
2、基于分组集群实现负载







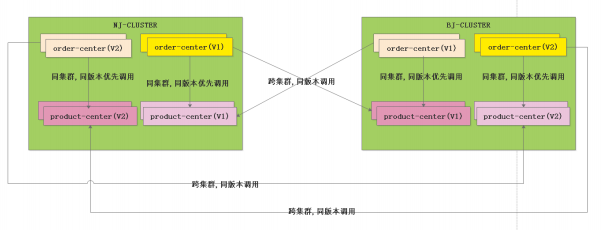
配置文件



3、基于金丝雀发布（灰度）发布实现负载均衡策略

此时需要引入nacos的版本配置，微服务之间只调用相同版本的实例

比如 order-center 存在二个版本 V1 (老版本) V2（新版本） ,product-center也存在二个版本V1 (老版本) V2新版本 现在需要做到的是order-center(V1 )----> product-center（v1 ） ， order-center(V2)----> product-center（v2）。记住v2版本是小面积部署的，用来测试用户对新版本功能的。 若用户完全接受了v2。我们就可以把V1版本卸载完全部署V2版本





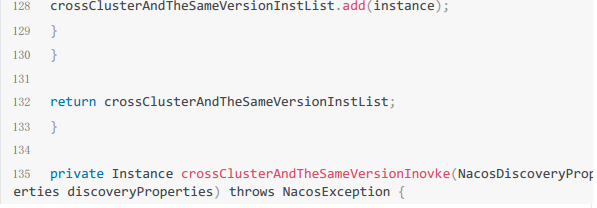






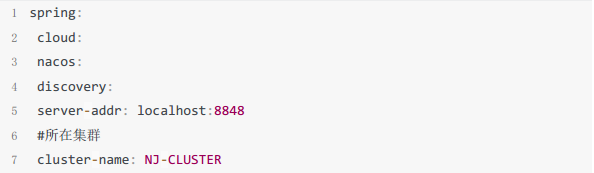








Yml配置文件





# 3、feign

Feign是Netflix开发的声明式、 模板化的HTTP客户端， 其灵感来自Retrofit、 JAXRS­2.0以及WebSocket。 Feign可帮助我们更加便捷、优雅地调用HTTP API。  
在Spring Cloud中，使用Feign非常简单——只需创建接口， 并在接口上添加注解即可。  
Feign支持多种注解，例如Feign自带的注解或者JAX­RS注解等。 Spring Cloud对Feign进行了增强，其支持SpringMVC注解，另外还整合了Ribbon和Eureka， 从而使得Feign的使用更加方便

## 3.1 feign的使用

### 3.1.1 feign 与ribbon的对比

Feign与ribbon的对比

Ribbon+RestTemplate进行微服务调用 模式  
ResponseEntity<List> responseEntity =

restTemplate.getForEntity("http://orderservice/order/queryOrdersByUserId/"+userId,List.class;

缺点:  
ResponseEntity<List> responseEntity =

restTemplate.getForEntity("http://orderservice/order/queryOrdersByUserId/"+userId,List.class);  
①： 我们不难发现， 我们构建上诉的URL 是比较简单的， 假如我们业务系统十分复杂， 类似如下节点https://www.baidu.com/s?  
wd=asf&rsv\_spt=1 &rsv\_iqid=0xa25bbeba000047fd&issp=1 &f=8&rsv\_bp=0&rsv\_idx=2&ie=utf-  
8&tn=baiduhome\_pg&rsv\_enter=1 &rsv\_sug3=3&rsv\_sug1 =2&rsv\_sug7=1

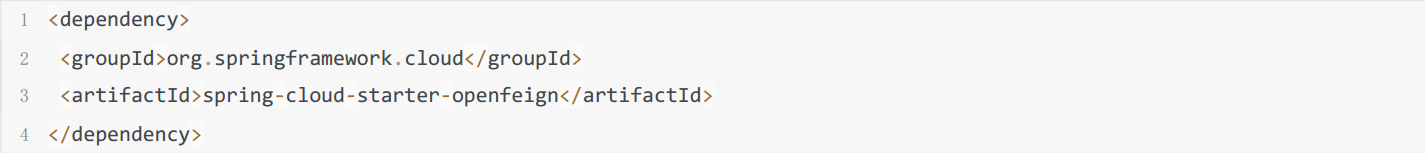
00&rsv\_sug2=0&inputT=328&rsv\_sug4=328  
那么我们构建这个请求的URL是不是很复杂， 若我们请求的参数有可能变动， 那么是否这个URL是不是很复杂

②：  
如果系统业务非常复杂， 而你是一个新人， 当你看到这行代码， 恐怕很难一眼看出其用途是什么！ 此时， 你很可能需要寻求老同事的帮助（往往是这行代码的作者， 哈哈哈， 可万一离职了呢？ ） ， 或者查阅该目 标地址对应的文档（文档常常还和代码不匹配） ， 才能清晰了解这行代码背后的含义！ 否则， 你只能陷入蛋疼的

### **3.1.2 引入feign**

此种情况是单独新建一个工程，存放所有微服务的暴露接口，然后各个微服务引入此工程的jar包，也可以每个微服务均引入feign，用于调用所需的接口

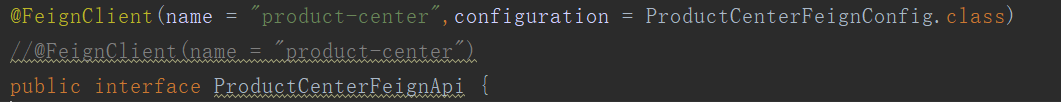
1、引入依赖

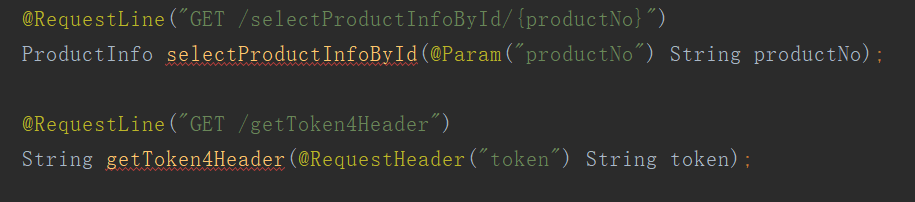


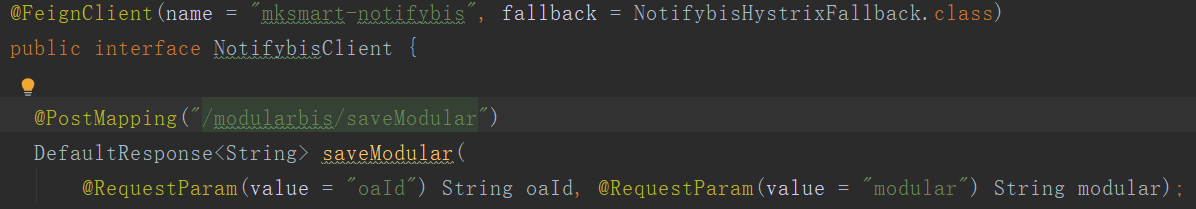
2、开启注解加入 @EnableFeignClients

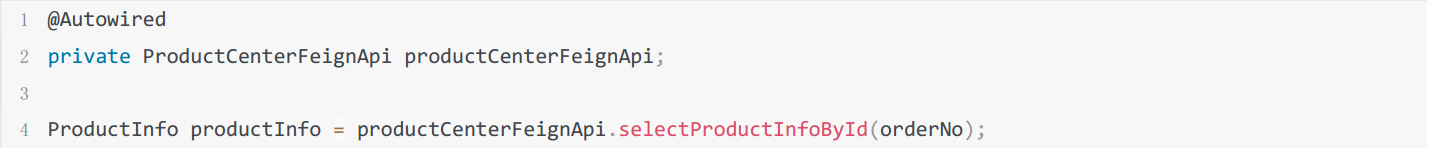


3、调用方式(像调用本地方式一样调用远程服务)









我们服务提供者tulingvip03-ms-alibaba-feign-product 的controller 需要实现我们的productCenterFeignApi接口， 防止修改(比如我们的productCenterFeignApi修改了， 若没有实现该接口， 服务提供者感知不到)  
第一步:引入依赖 tulingvip03-ms-alibaba-feign-api  
第二步:我们的ProductInfoController实现productCenterFeignApi接口



### 3.1.3 feign的日志

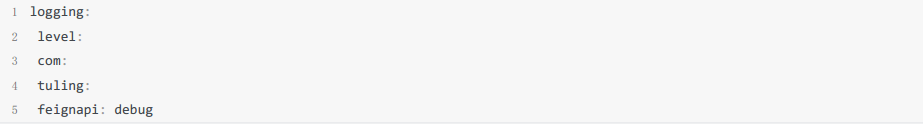
默认情况下， Feign的调用式不打印日志， 我们需要通过自定义来打印我们的Feign的日志 （basic适用于生产环境）



1、我们在tulingvip03-ms-alibaba-feign-api工程中添加Feign的自定义配置



针对调用端工程tulingvip03-ms-alibaba-customcfg-feign-order针对日志com.tuling.feignapi 包下的日志级别必须调整为DEBUG级别的 不然是不会打印日志的，即只打印com.tuling.feignapi包内的日志

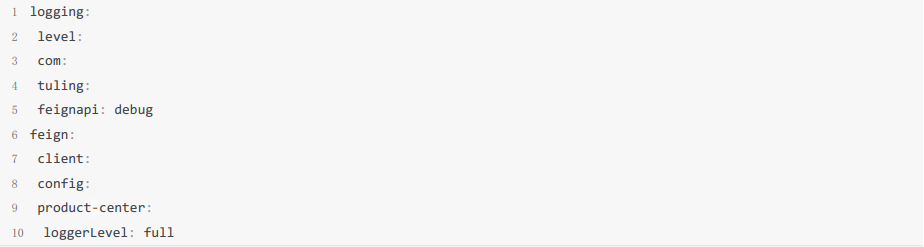


2、日志在微服务内的细粒度配置

ProductCenterFeignApi 不用指定configuration的选项



在调用方: tulingvip03-ms-alibaba-customcfg-feign-order  
通过feign:client:config:微服务名称:loggerLevel: 日志级别来指定



### 3.1.4 feign的原生注解配置

根据自动装配我们FeignClients的配置中的默认锲约是springmvc（也就是说支持SpingMvc注解）

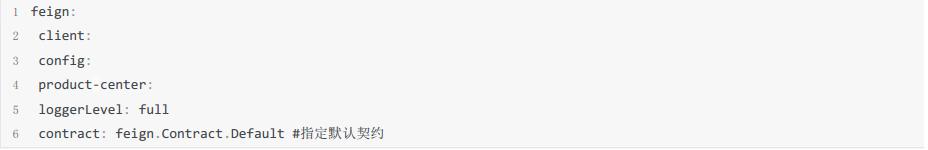
现在我们需要想使用Feign的原生注解来标识方法需要修改锲约



FeignClient类ProductCenterFeignApi使用Feign原生的注解

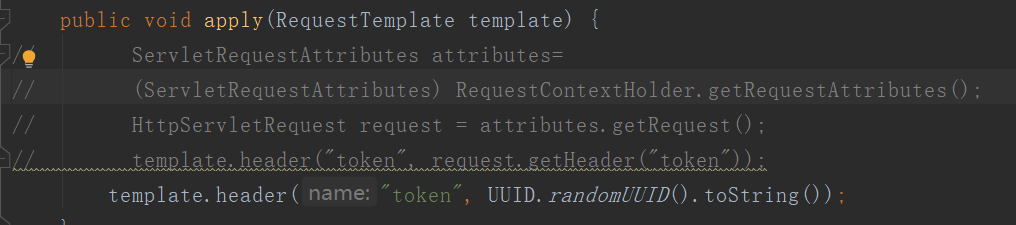


也可以通过配置文件的形式来指定我们的契约



### 3.1.5 拦截器

可以用于将上游微服务的token传递给被调用的服务

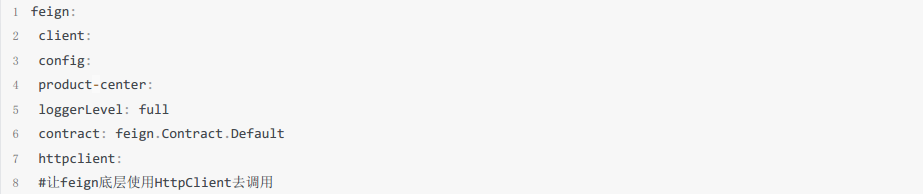






### 3.1.6 feign调用优化方案

1、开启连接池配置





2、调整Feign的日志级别（强烈推荐使用Basic级别的)，如果使用其他工具收集日志，则可以不打印日志

3、超时时间设置

Feign的底层使用ribbon调用，负载均衡策略为轮询，超时时间以feign的配置为准



