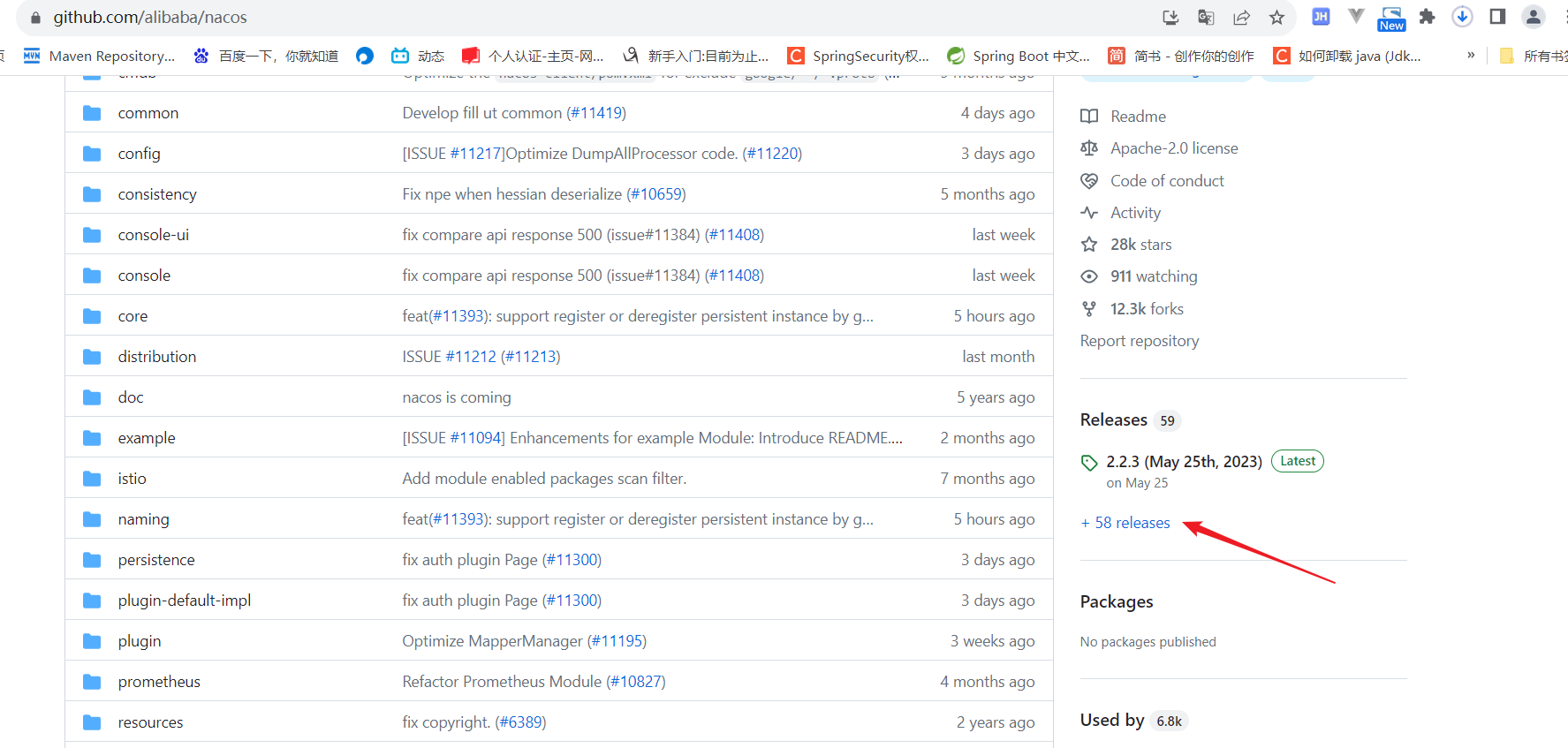
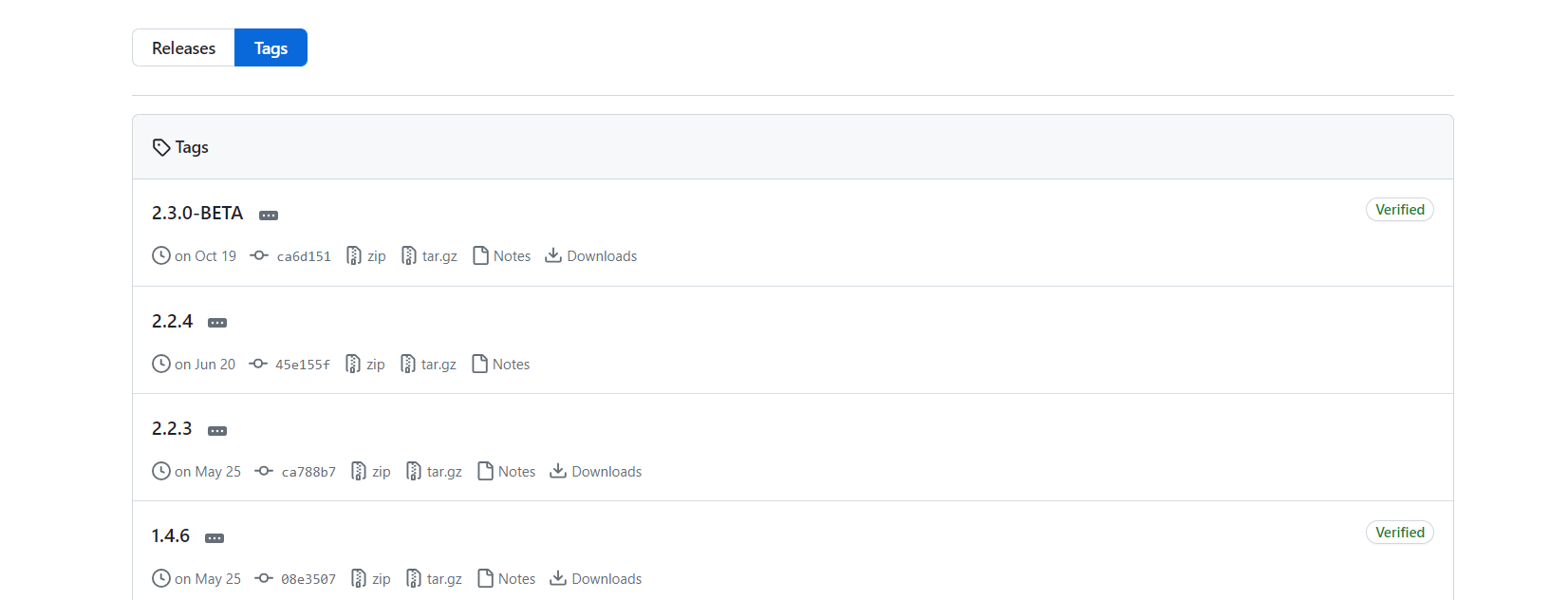
# SpringCloudAlibaba注册中心Nacos

## 下载安装：

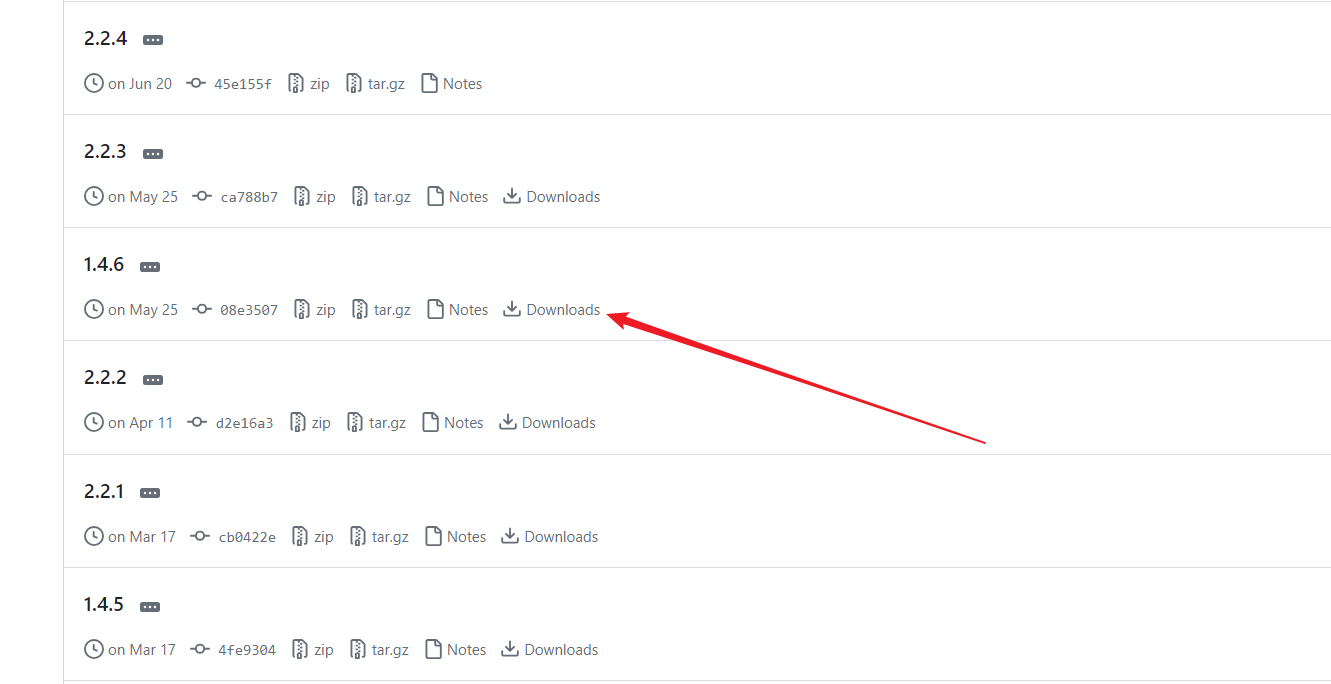
1：进入nacos的github页面



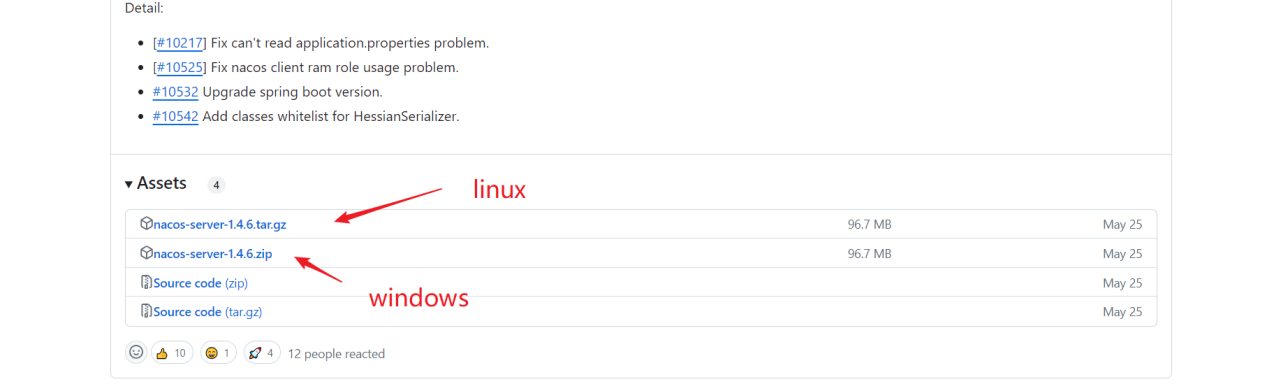
2：选择tags



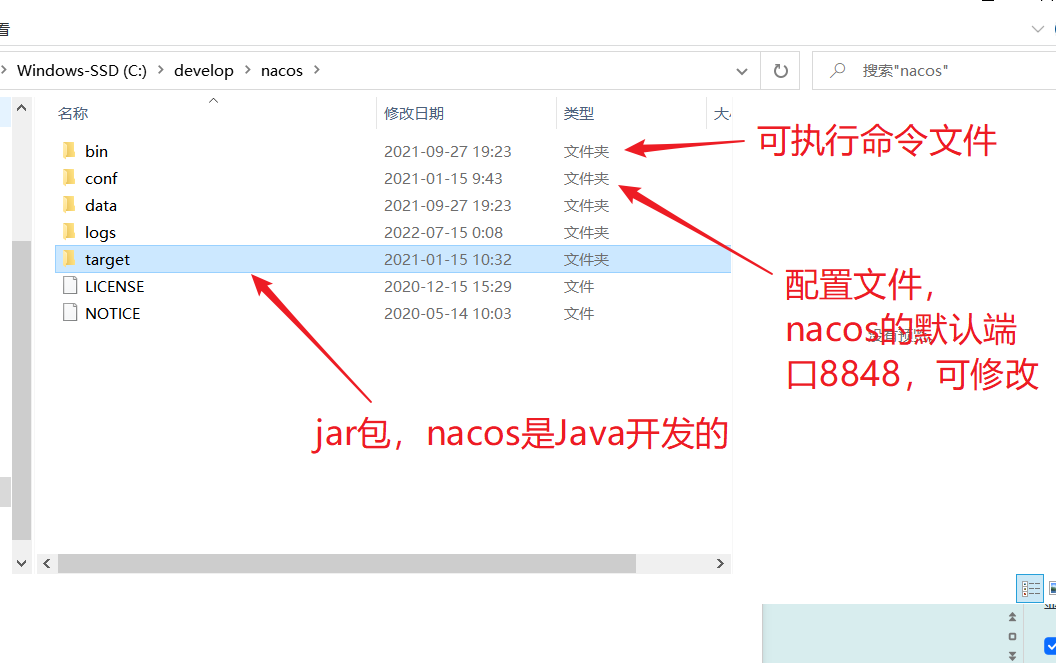
3：选择想安装的版本下载：



4：下载：

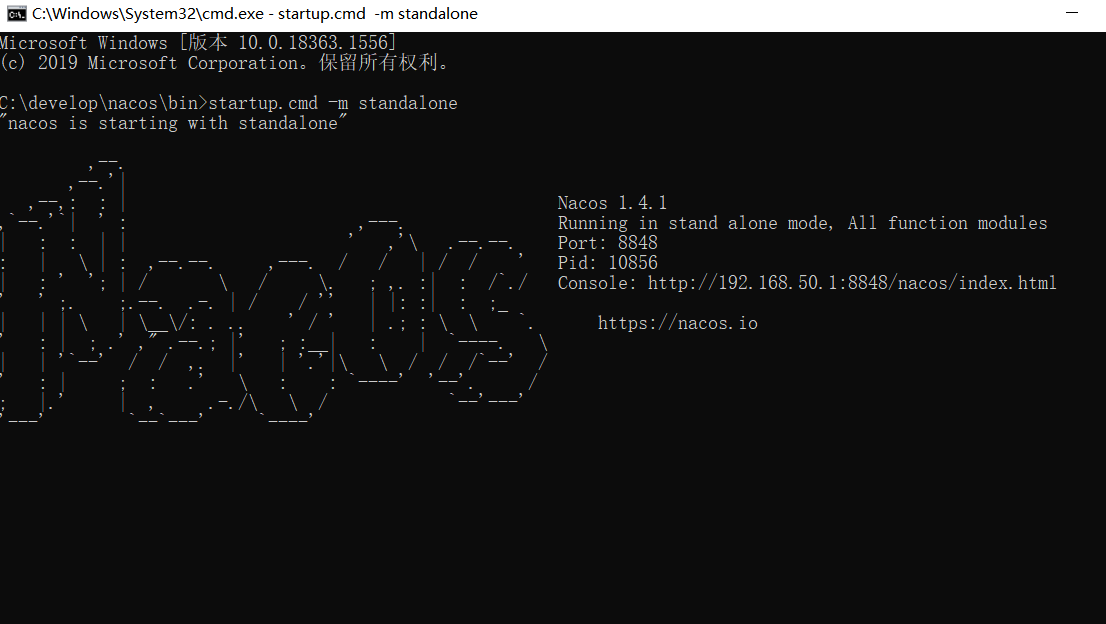


5：window安装Nacos：把下载的nacos在非中文路径下解压即可



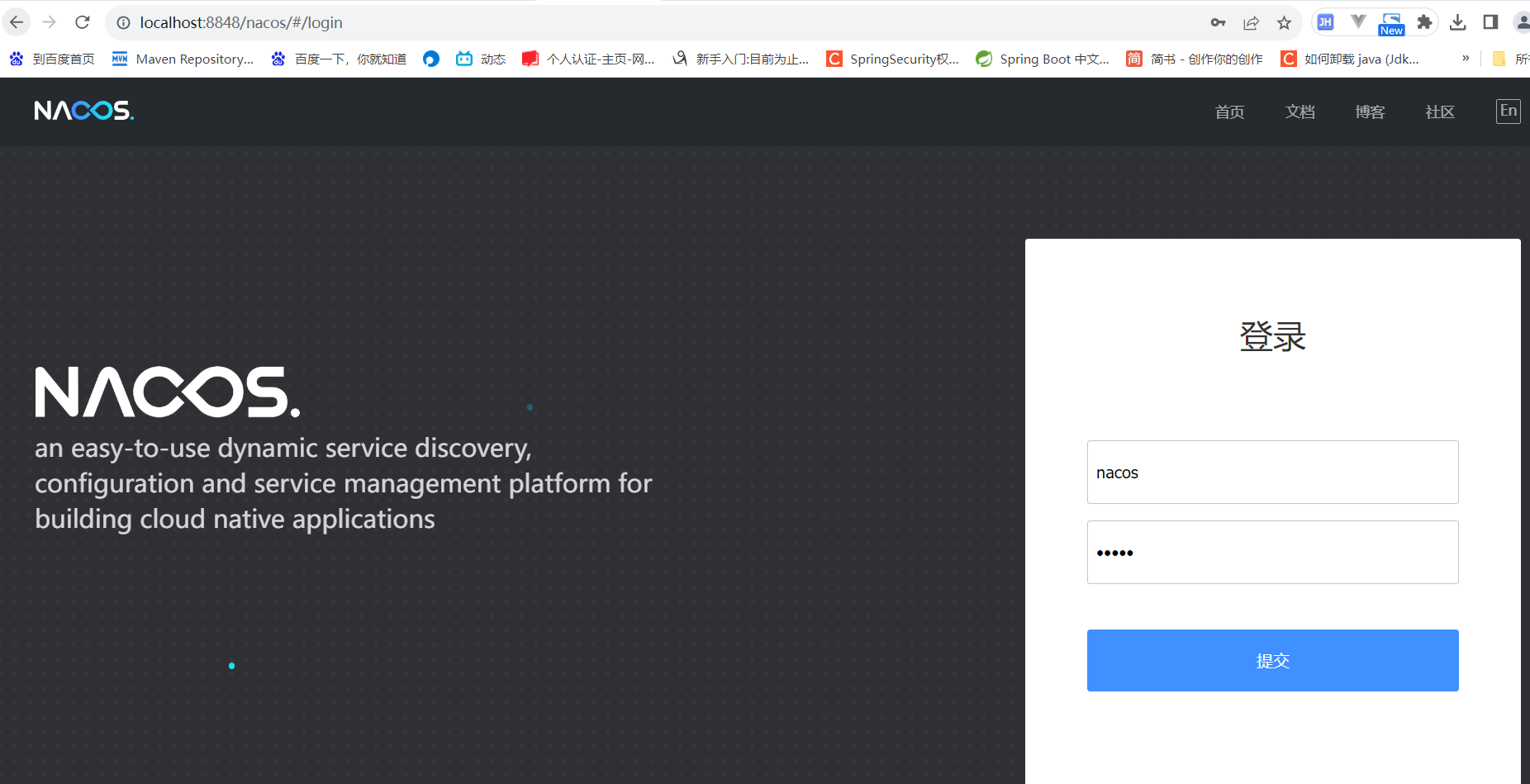
6：windows单机启动nacos：进入可执行文件bin，打开bin目录下的cmd控制台，执行命令：startup.cmd -m standalone

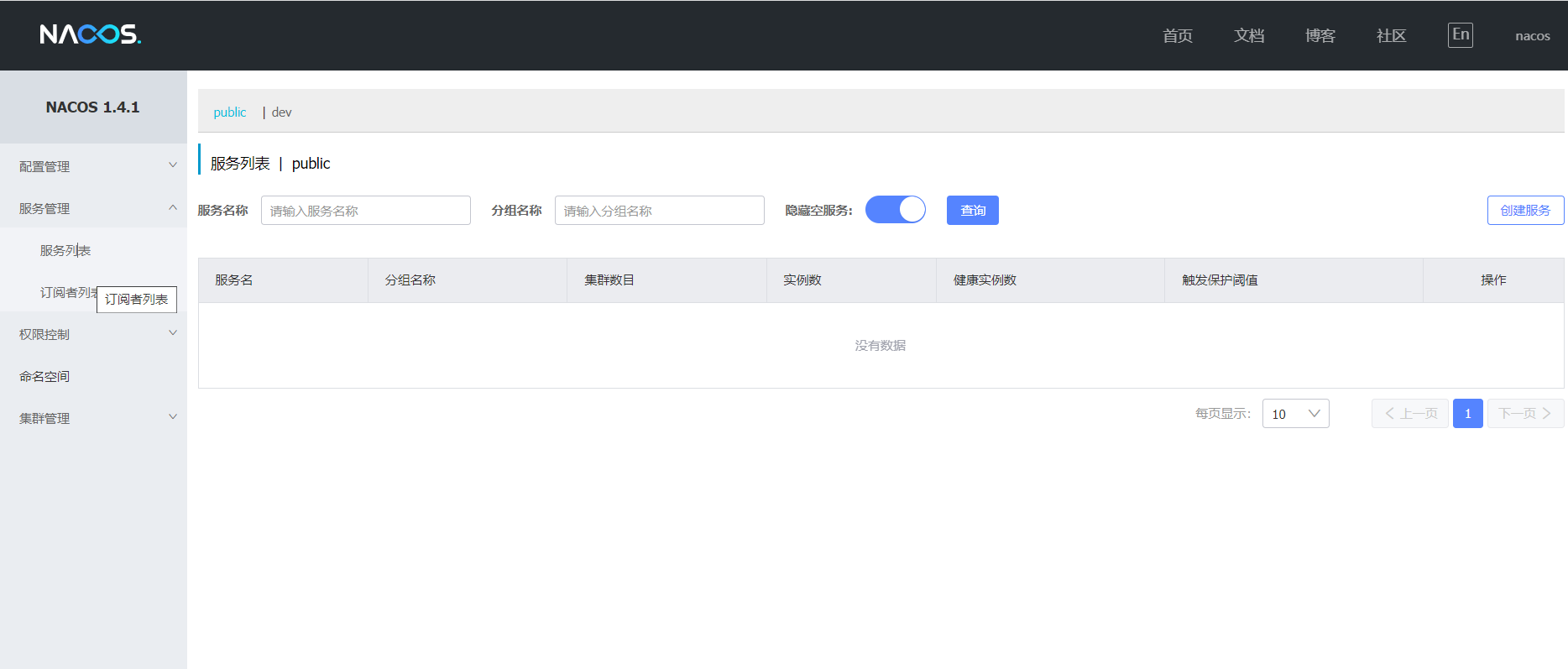
-m表示启动模式参数，standalone表示单机启动



7：登录nacos：<http://localhost:8848/nacos>

账号密码都是：nacos



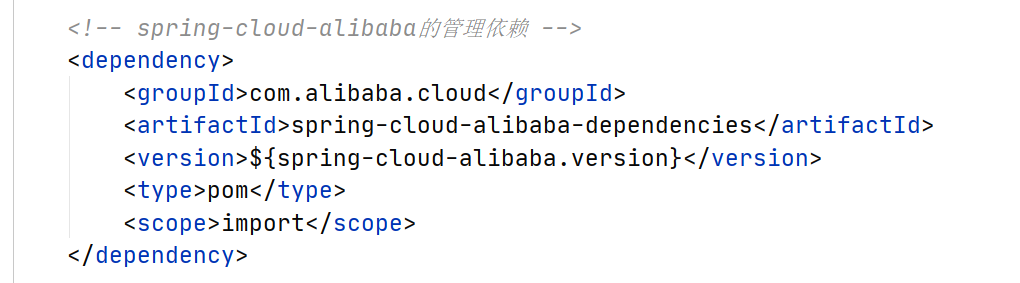


## 服务注册到Nacos：

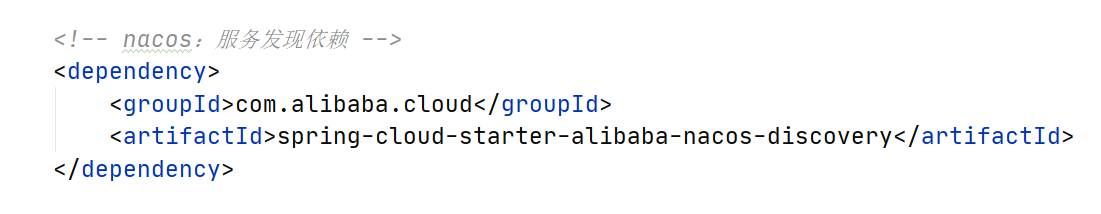
需要提前声明的是，SpringCloudAlibaba是基于SpringCloud的，所以SpringCloudAlibaba中的微服务技术组件是遵循SpringCloud定下的微服务技术规范的：

在SpringCloud中，有个Spring Cloud Commons组件，即通用，其实在这里是定义SpringCloud技术栈接口规范的，比仿说：DiscoveryClient interface和ServiceRegistry interface，服务发现和服务注册的接口规范，因此，不管是eureka还是nacos，只要是做服务注册、服务发现，都要遵循这些接口规范，所以我们在使用nacos或者eureka的时候，服务提供者和服务消费者的代码是几乎不用做任何改动的。变化的只有引用的依赖不同，服务中心的地址以前是配eureka的地址，现在是配nacos的地址！

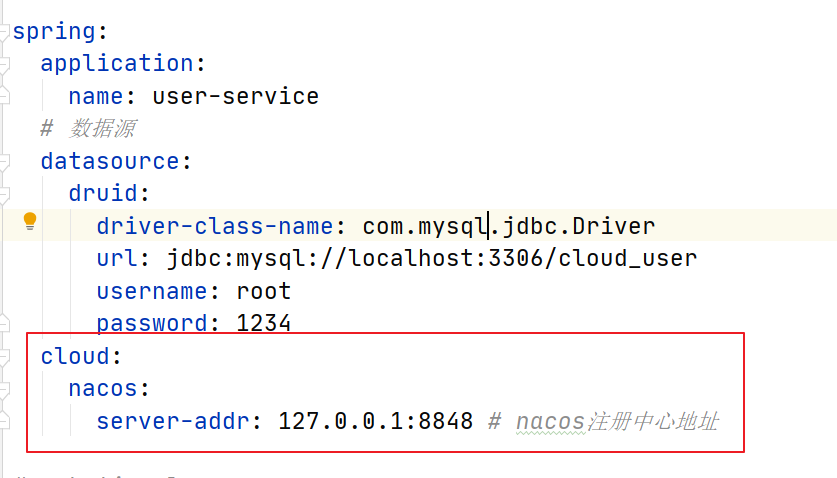
1：在父工程中添加SpringCloudAlibaba的管理依赖：



2：添加nacos客户端依赖：



3：配置nacos注册中心地址：



4：查看服务实例：



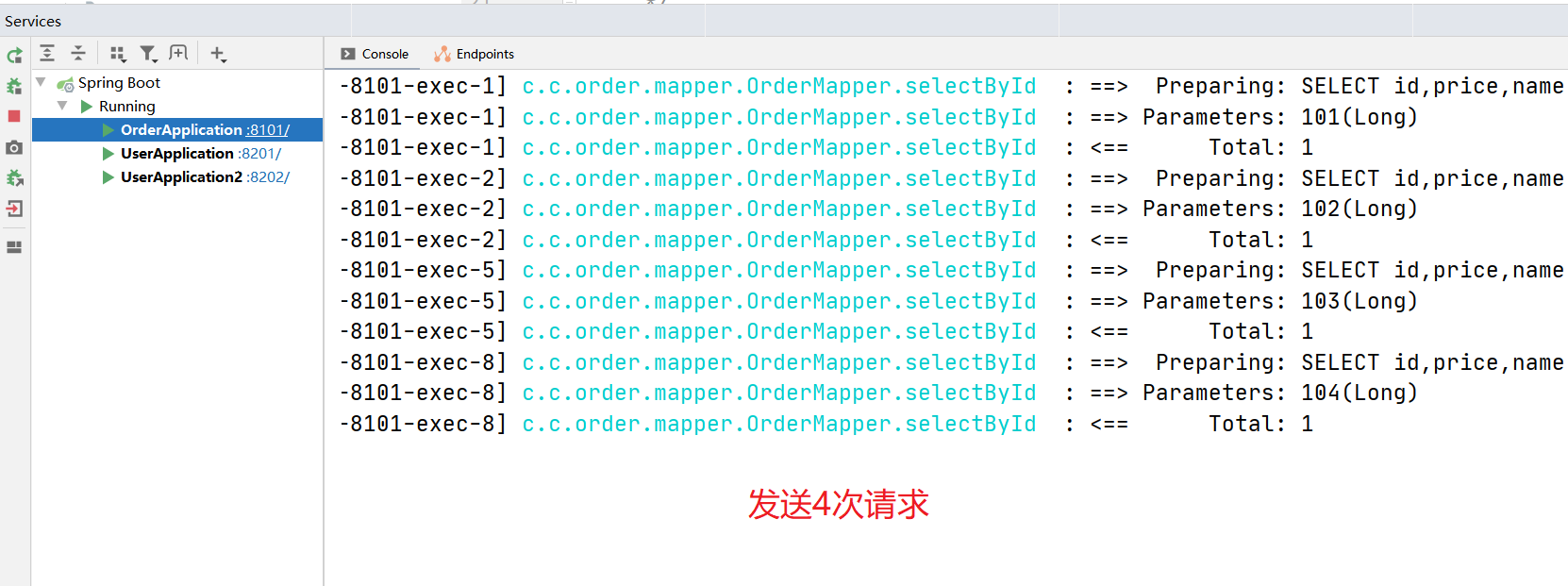
详情请看cloud-hyh项目：

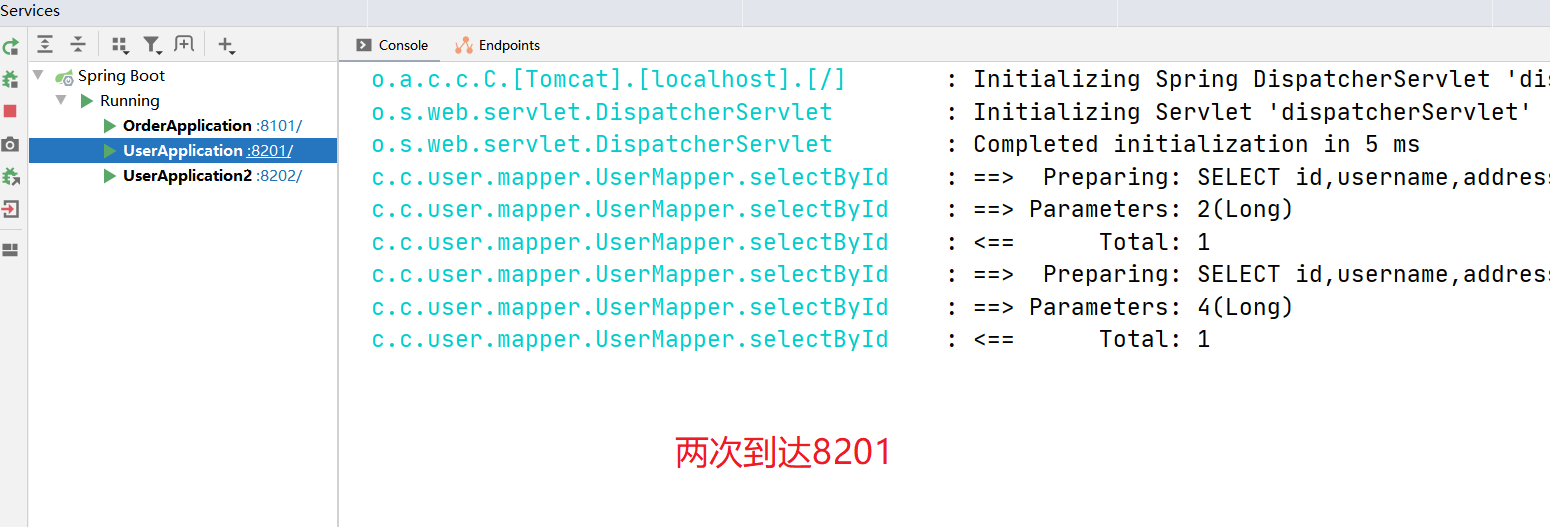
<https://github.com/huangyuanhui/spring-cloud-alibaba>

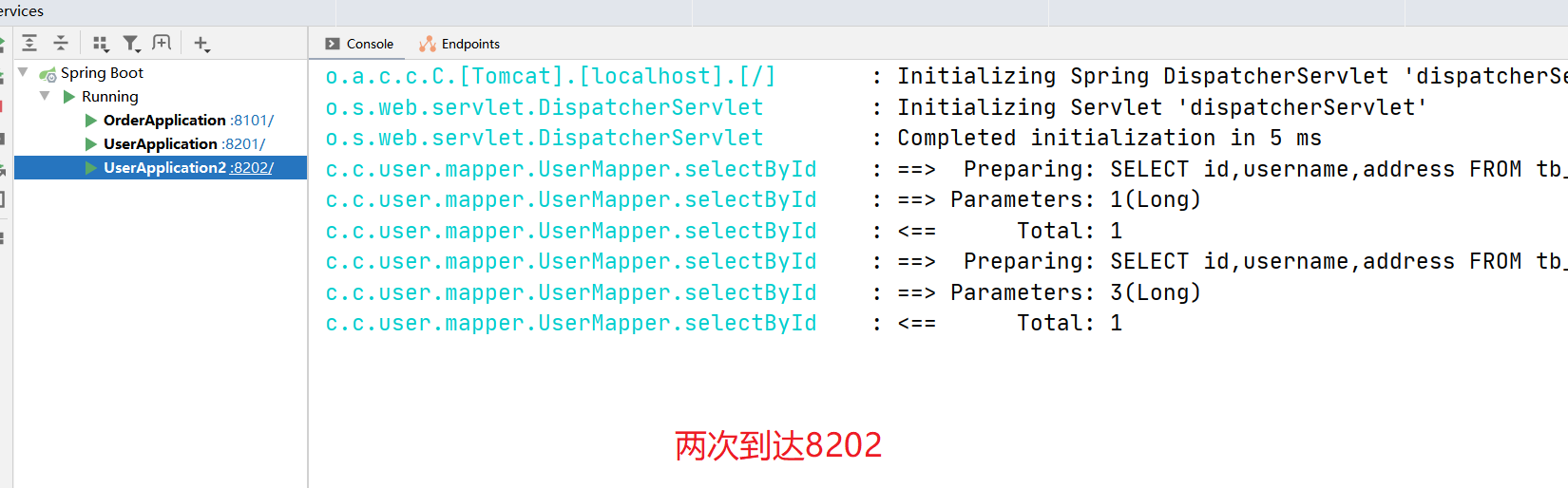
可以看到，仍然使用RestTemplate实现服务的远程调用！



为什么还可以使用RestTemplate实现服务的远程调用？因为nacos知识实现了SpringCloud定义的注册中心的接口，服务调用的实现完全可以用回RestTemplate！之前说的Ribbon啥的什么都不用动，都是通用的！







## Nacos服务分级存储模型

在Nacos中，可以分成服务、地区、实例这三个概念。

我们已经接触过这样的分级概念了，比如在demo中的。

我们有服务的概念：比如user-service、order-service，提供用户查询和订单查询，都是服务；然后我们的user-service还提供了多个实例：8201和8202，。。。这两个都是user-service的实例，所以我们至少是具有两层概念了，一层是服务，一层是实例，一个服务可以包含多个实例，不过随着业务规模越来越扩大，我们就要考虑更多的问题，比如我们把所有的实例都部署在一个机房，就像是把鸡蛋都放在一个篮子里，那一天不小心篮子翻了，那所有蛋就都打了，所以机房有一天万一出了问题，那整个服务也就玩完了。所以为了解决这个问题，我们会将一个服务的多个实例部署到多个机房，特别是像阿里巴巴、京东、腾讯这些大公司，会在全国重要城市都整一个机房：北京、杭州、深圳、成都、、、、，就是把鸡蛋分散开了，这就能做到容灾！

而Nacos的服务分级存储模型就是引入了机房或者是地域的概念，把同在一个机房的多个实例称为一个集群，比方说杭州机房的来给两个user-service实例就称为杭州user-service集群，北京机房的来给两个user-service实例就称为北京user-service集群，所以在nacos服务分级存储模型中，第一级是服务，第二级时集群，第三级是实例。

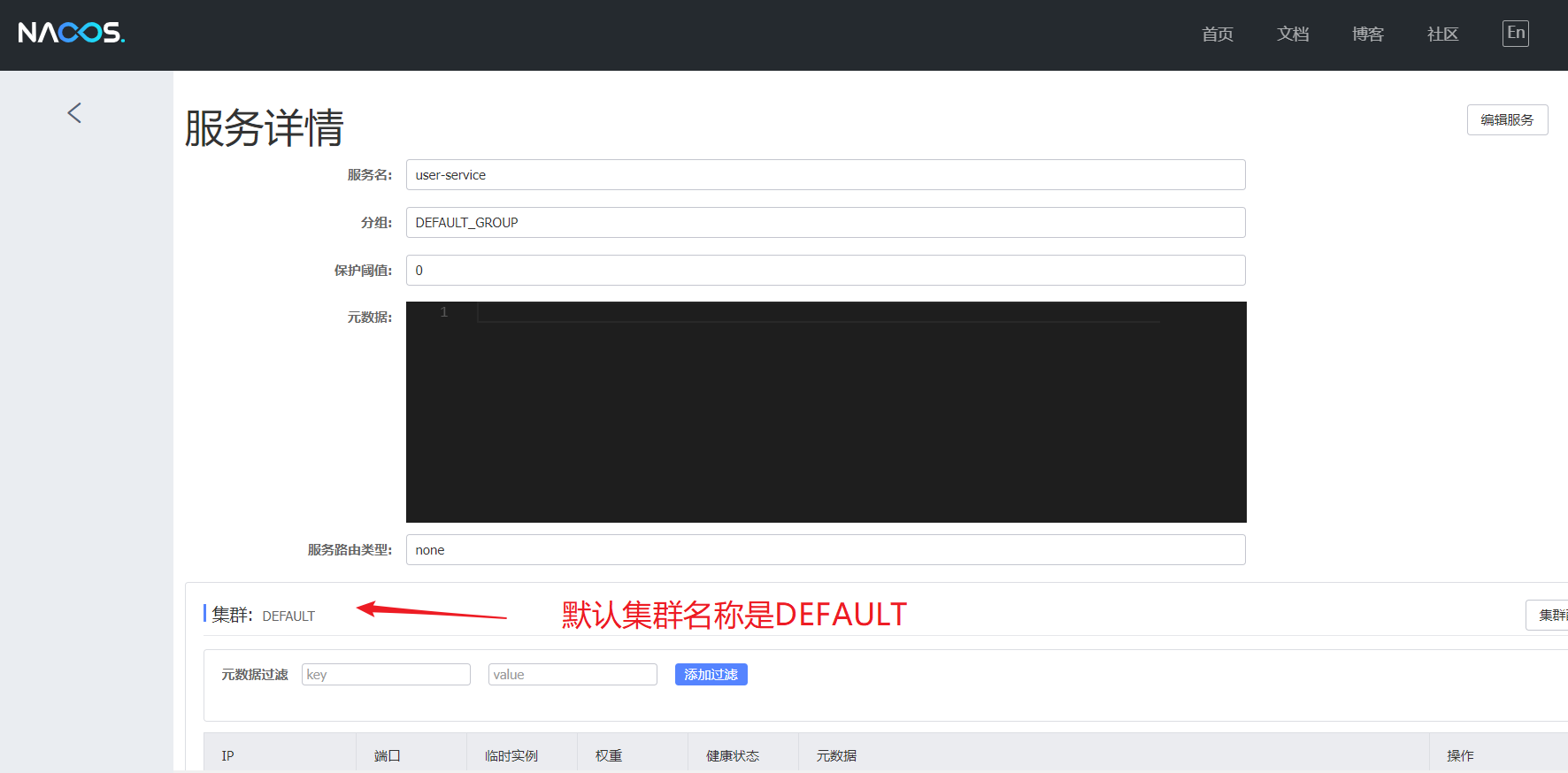
为什么nacos要引入这样的服务分级存储模型？

我原来直接用服务找实例不好吗，为什么要在中间加多一层地域划分，集群的概念？

我们设想一下这样的情况，比方说我们现在由杭州的机房，里面有order-service集群和user-service集群，然后还有一个深圳集群，里面同样有order-service集群和user-service集群，将来可能还有北京机房、成都机房等等，现在order-service需要访问user-service，那么现在有两种选择，一是在自己本机房/集群里的user-service访问，二是到另一个机房/集群的user-service访问，毫无疑问，正常情况下肯定是现在本机房/集群的user-service，因为局域网里的访问距离比较短，速度比较快，延时比较低，而你跨越网络/机房的访问，达到数百上千公距离，这个时候延迟是非常高的，所以在服务调用的时候，应该尽可能地区访问本地集群，只有在本地集群不可用的情况下才会去访问其他集群，Nacos引入集群概念就是为了尽可能防止出现跨集群调用！

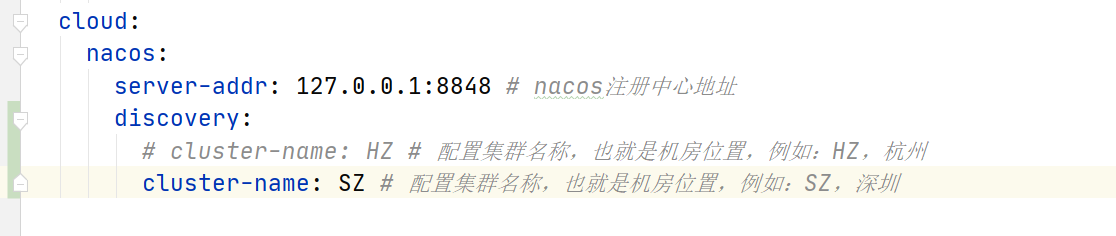
### 如何在nacos配置集群属性:

可以发现，默认集群的名称是Default，也就是没有集群：

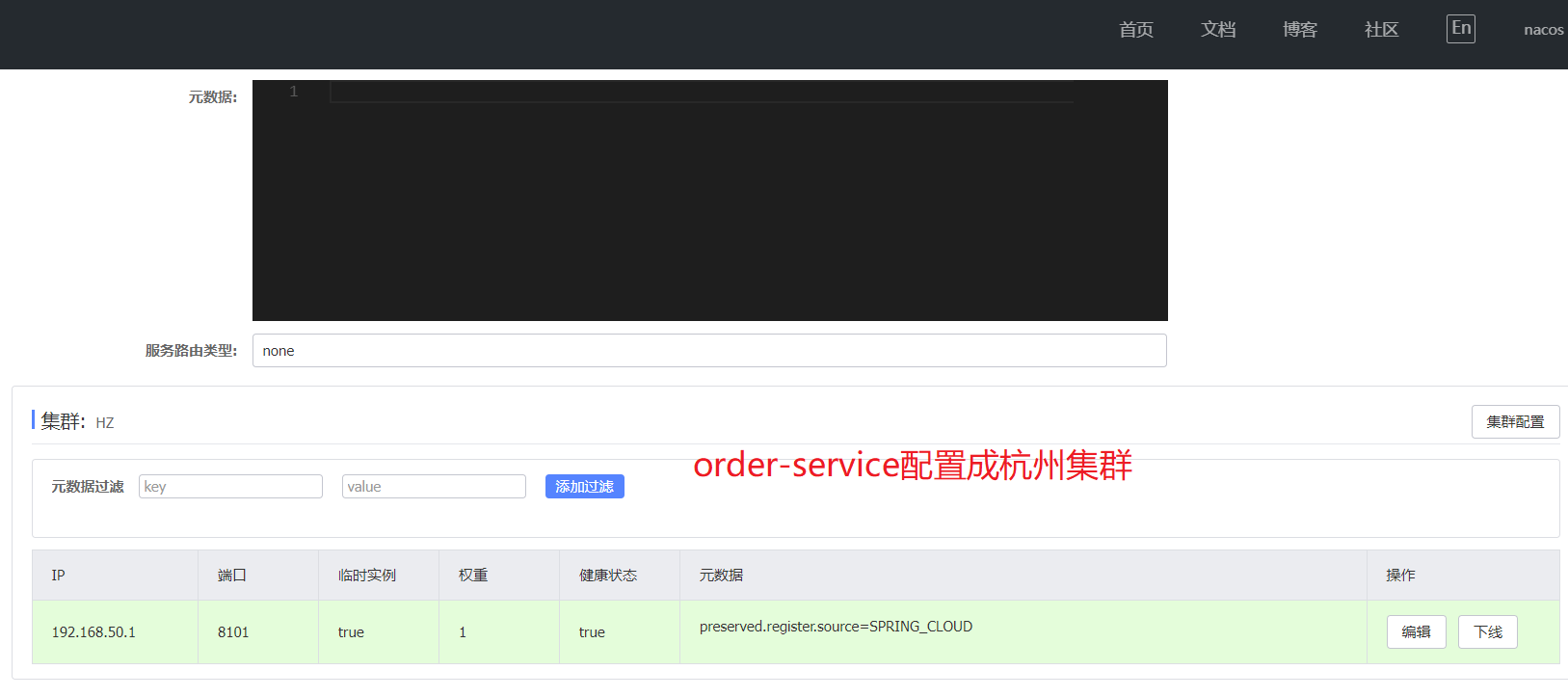


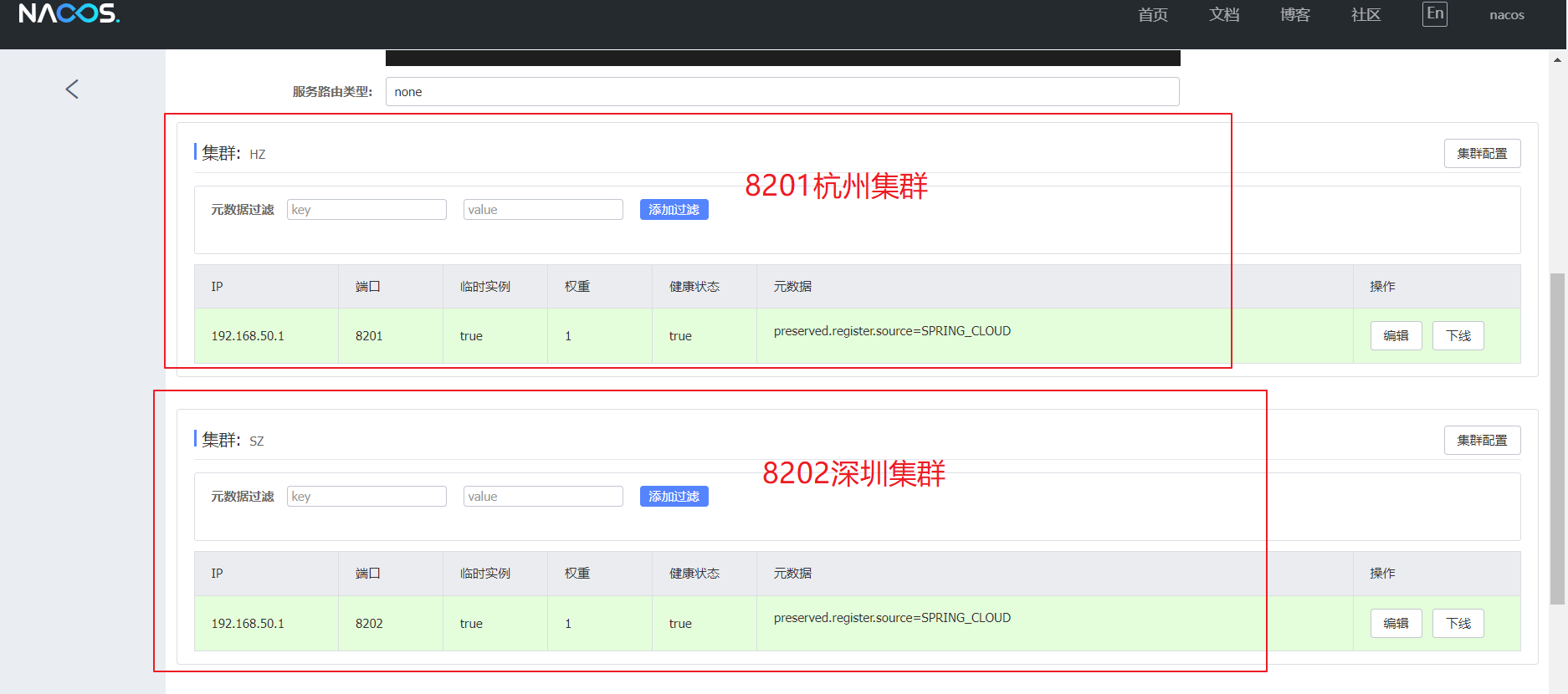
下面看一下，如何配置一个实例的集群属性：

1：修改application.yml配置文件，添加如下配置：

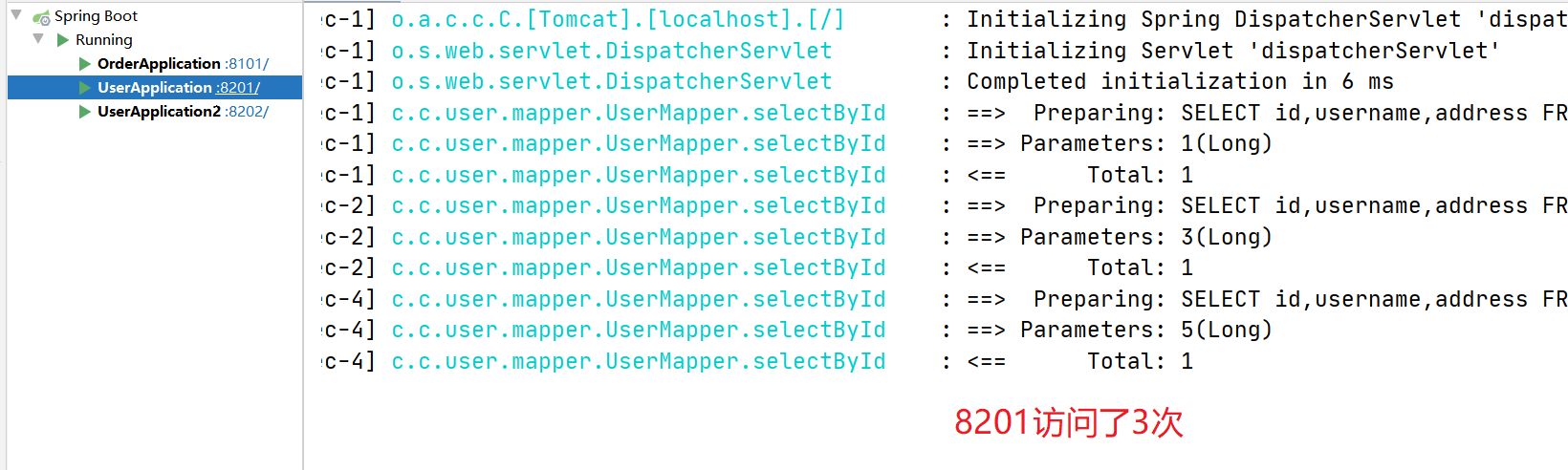


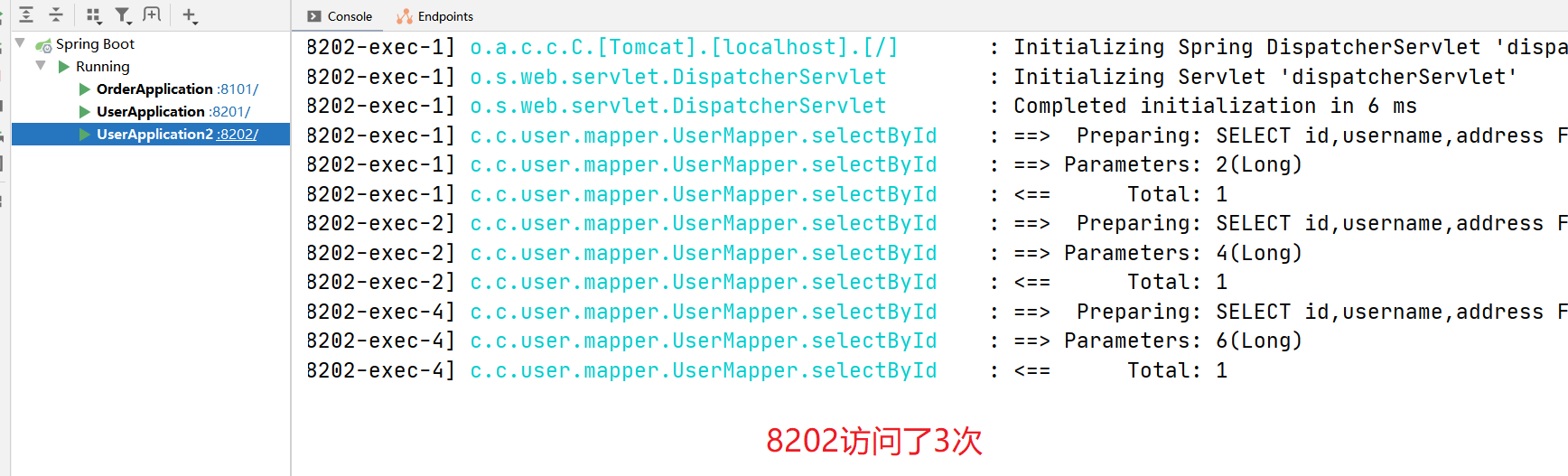
2：在Nacos控制台查看服务实例的集群变化：





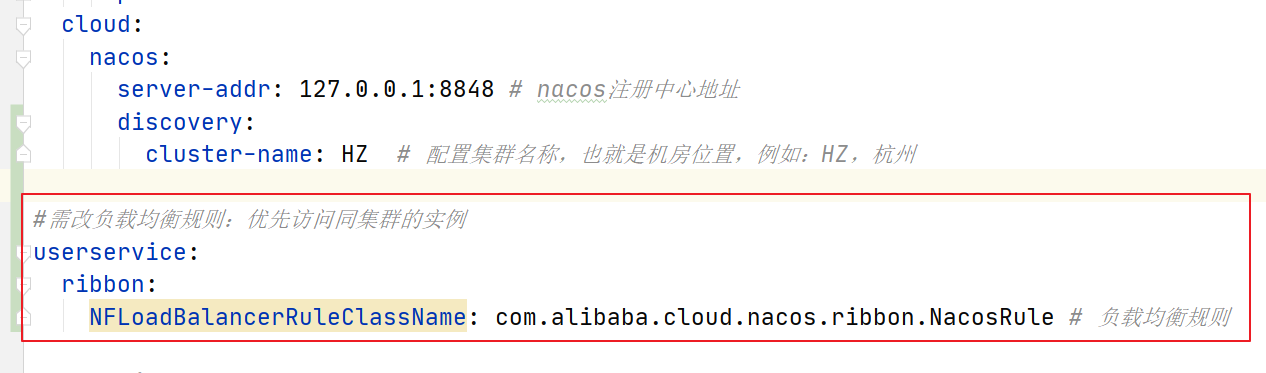
完成以上配置，我们想要实现的想过是，order-service远程调用user-servcie时，总是优先选择同属于一个本地集群的HZ，即杭州集群，但是我们发现结果还是轮询各三次：





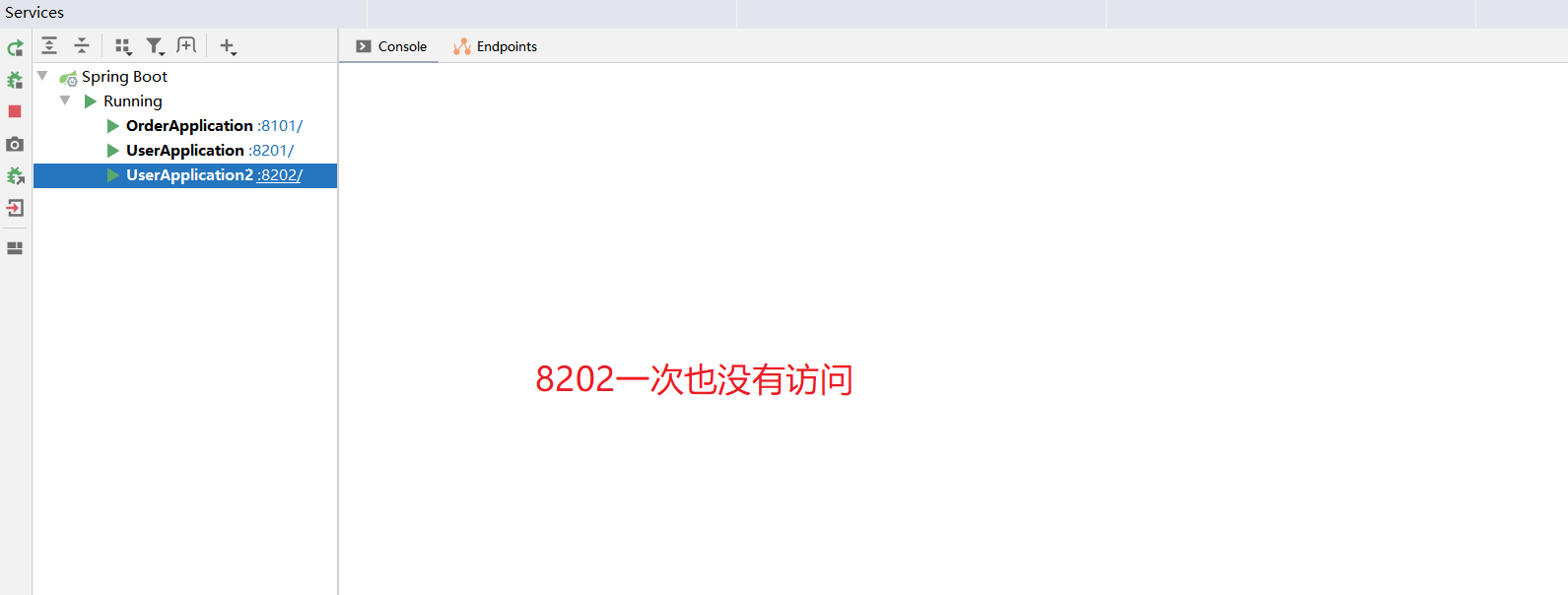
这是什么原因：我们知道服务调用在选择实例时，规则是由负载均衡的规则来决定的，也就是IRle，默认的IRule就是轮询的策略。所以要想实现优先同集群访问的负载均衡规则，必须要去修改负载均衡！

如何修改：



结果如下如，说明修改了配置，负载均衡策略改成了NacosRule，确实实现了优先选择本地集群访问：





NacosRule的特点：优先选择本地集群，在本地集群内的多个服务中再采取随机的方式进行负载均衡！优先本地，本地没有，再去跨集群访问！

### 总结：

NacosRule负载均衡策略：

1：优先选择同集群服务实例列表；

2：本地集群找不到提供者，采取其他集群寻找，并且会报警告；

3：确定了可用实例列表后，再采用随机负载均衡挑选实例。

### Nacos注册中心根据权重负载均衡：

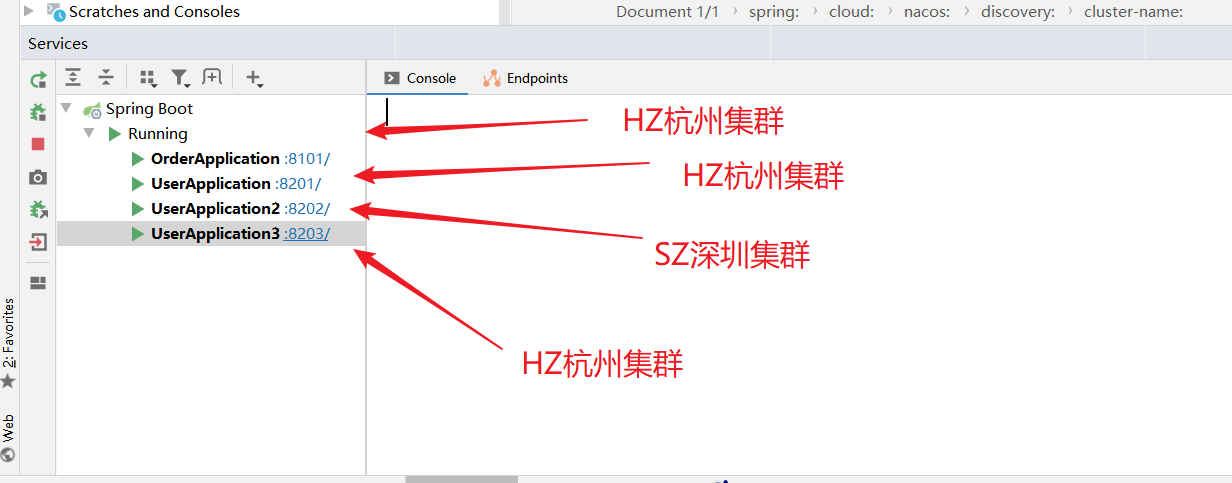
以上学习了Nacos负载均衡规则：NacosRule，可以做到集群优先的负载均衡，不过我们部署服务的时候，可能存在这样一种情况：：企业的的服务器设备会更新迭代，有一些设备性能好一些，有一些设备性能性能差一些，这个时候，我们肯定希望性能好的机器承担更多的用户请求，性能差的机器承担少一点的请求，即能者多劳，但是，目前看来，NacosRule实现的负载均衡是集群优先，而后再集群内做随机，当用户请求来的时候，NacosRule的负载均衡规则可不管你是性能好的还是性能差的，这个时候性能差的肯定就会出问题！

一句话话就是实际部署会出现这样得情景：服务器设备得性能有差异，部分实力所在机器性能较好，另一些交叉，我们希望性能好得机器承担更多得用户请求！

那么我们应该怎样去控制不同服务实例的请求量呢？

答：Nacoss提供了权重配置来控制风闻频率，权重越大访问频率越高！即Nacos给我们提供了权重的配置！通过修改服务实例的权重，可以控制服务实例的访问频率！权重越大，访问到的频率越高，那么我们就可以把在性能好的机器的服务的权重设置得大些，性能差的机器权重设的小些！

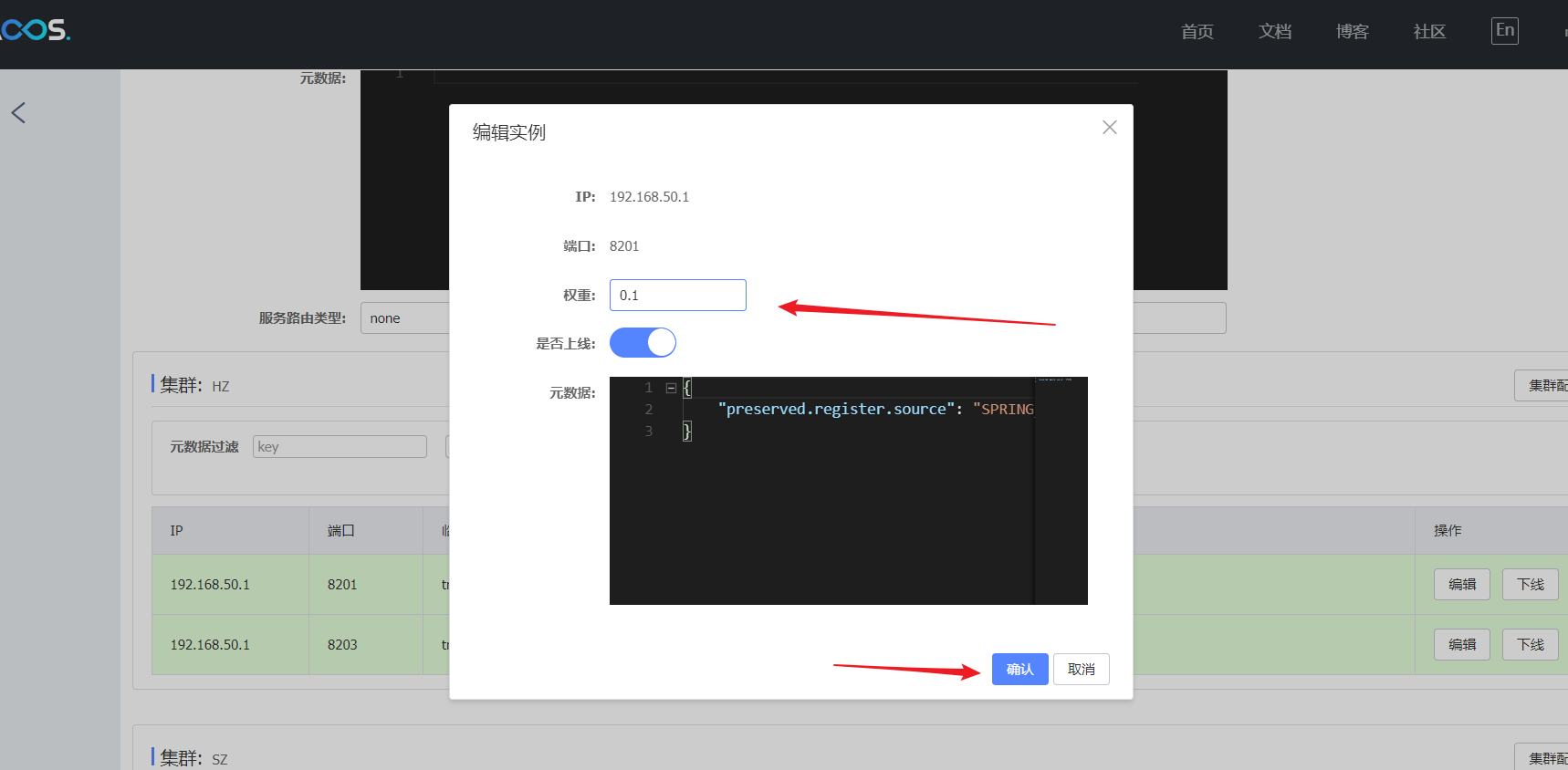
#### 根据权重负载均衡

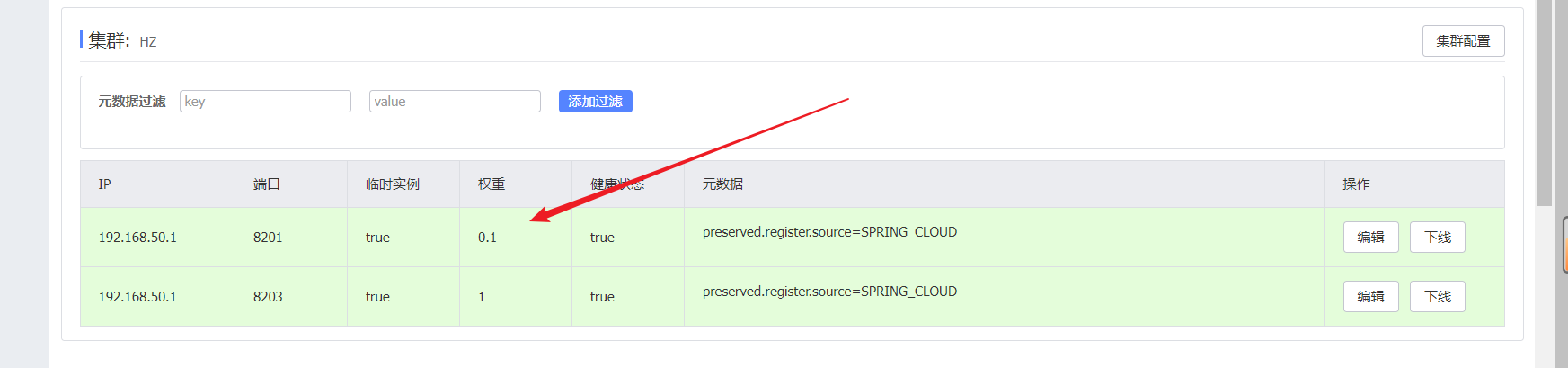


1：在Nacos控制台可以设置实例得权重值，首先选中共实例后面得编辑按钮：

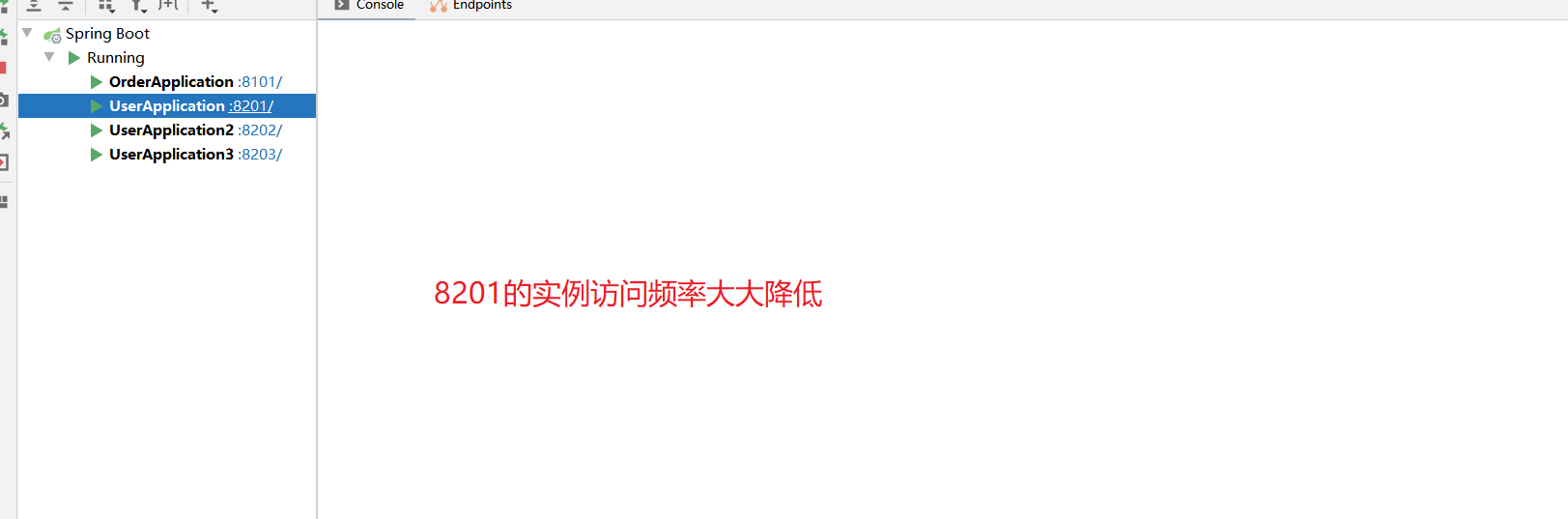


2：将权重设置为0.1，测试可以发现8081被访问得频率大大降低（权重值一般配0到1之间）：

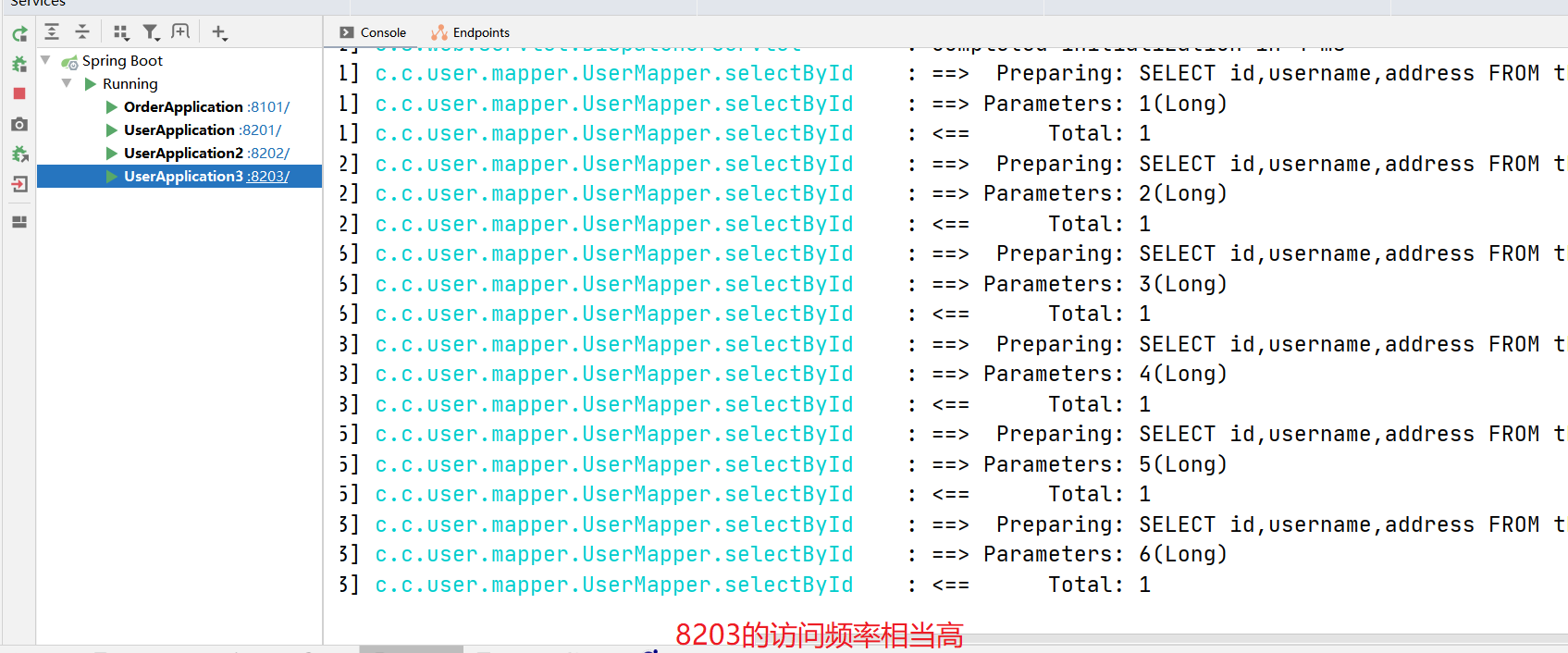




3：权重为0.1得8201实例访问频率大大降低：



4：权重为1得8203实例访问频率相当高：



### 配置实例权重和NacosRule集群的区别：

可以简单视为：NacosRule负载均衡规则是优先同一集群，再在同集群里对服务实例做随机选择；

配置实例权重可以看作是对Nacos的补充，就是权重确定服务实例的访问频率，权重和NacosRule一起作用！

设置成0实例不会被访问）

### 将权重设成0可以实现灰度发布

将服务实例的权重设成0，该服务实例就不会被访问，将权重设成0可以实现灰度发布，可以让上线更友好：

以前一个服务，我们想要做版本升级，为避免用户正在访问着，重启服务的时候，用户访问不了了，会小心翼翼，所以，版本升级往往会选在夜黑风高无人的夜晚，用户都下线了，偷偷的把服务停机，去做版本升级，但是现在有了Nacos的权重配置，我们可以**先把多个服务器里的其中一个服务调成0，大白天也没事，调成0后，该服务渐渐地不承担用户请求，这个时候对他做停机（就是杀死服务的意思），用户是没有感知的**，然后对这个服务做完版本升级后重启，权重先别调太大，调小一点，放进少数用户请求进来做测试看有没有问题，如果没什么问题，我们就逐渐扩大比例，依次升级。这个时候用户是无感知的，就可以做到平滑升级！非常优雅！

实例的权重控制总结：

Nacos控制台可以设置实例的权重值，0-1之间；

同集群内的多个实例，权重越高被访问的频率越高；

权重设置为0则完全不会被访问。

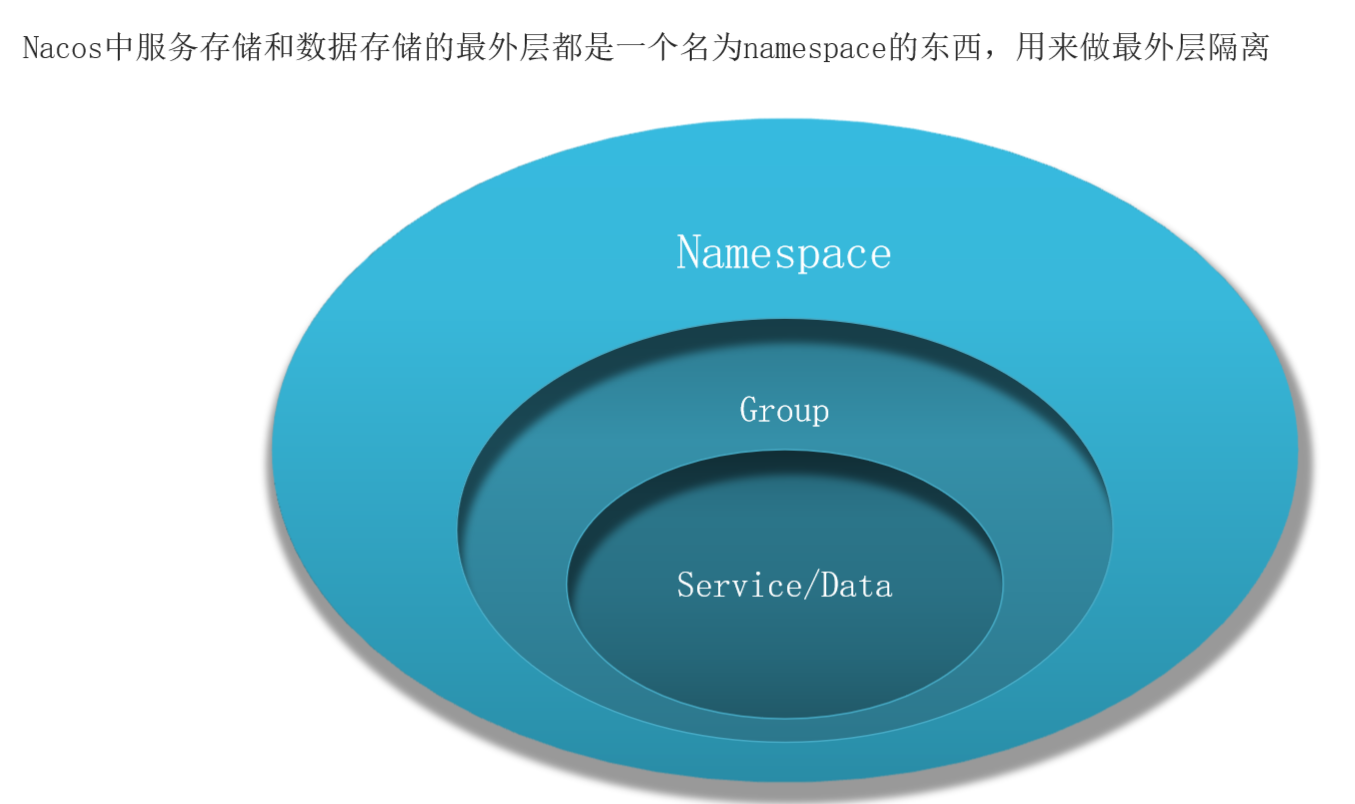
### Nacos注册中心：环境隔离-namespace

本次学习Nacos中的一个概念：环境隔离，我们知道Nacos首先是一个注册中心，但Nacos还是一个数据中心！所以在Nacos里面为了做数据和服务的管理，Nacos会有一个隔离的概念，环境隔离的概念！

环境隔离：Nacos中服务存储和数据存储的最外层都是一个名为namespace的东西，用来做最外层隔离

Namespace：命名空间，可以把命名空间看成一个隔离的空间，多个隔离的空间就是有多个namespace，多个namespace是相互隔离开的！

在namespace内部会有group的属性，也就是说同一个命名空间内部的多个东西还可以分组，组内就是具体的东西：服务/数据



环境隔离就是对服务做隔离，可想而知，不同环境、不同命名空间的服务是不能相互访问的！

简而言之：namespace就是对服务/数据做隔离的！

问：我们既然已经对服务划分成了不同集群，为什么又要在整个隔离呢？

答：服务划分、实例划分是基于业务或者是地域做的划分，事实上我们还会有开发环境、测试环境、生产环境的变化。所以我们会基于环境变化去做隔离。Namespace就是来做这件事的，group是分组的意思，我们可以把一些业务相关度比较高的服务放在一个组，比如订单服务和支付服务业务相关度比较高，那么就可以把她们放在一个group里面。

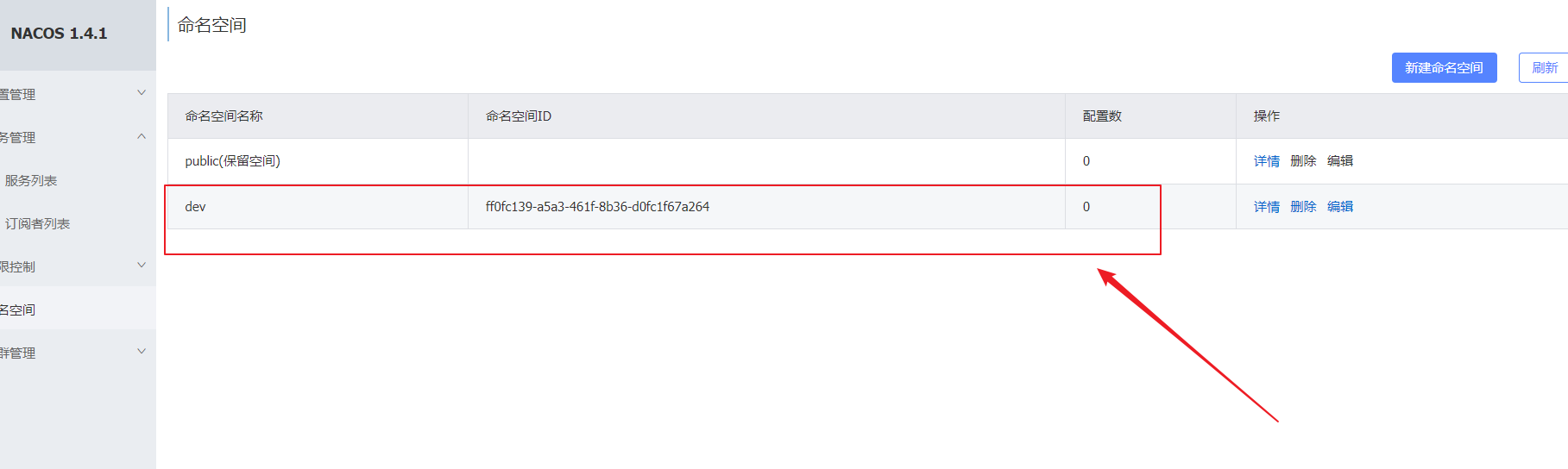
以上都是一些概念上的划分，并不是要求你一定要用namespace、group，不是强制的！

#### 新建命名空间namespace：

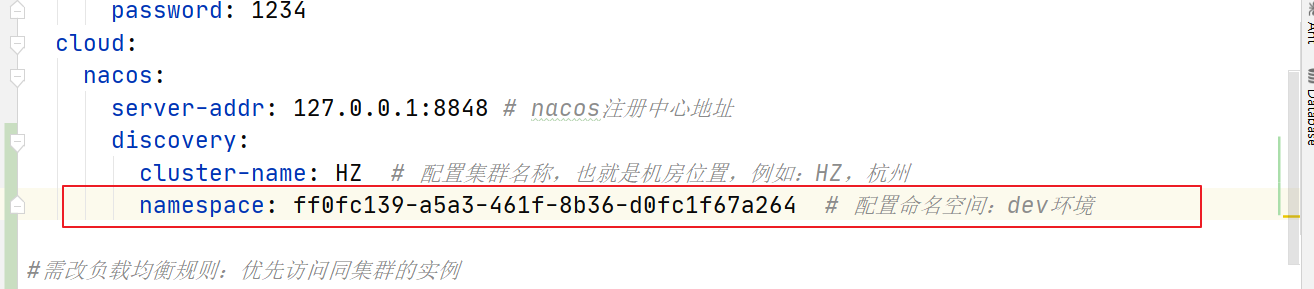
1：默认的命名空间是public，默认服务都在public这个命名空间下：



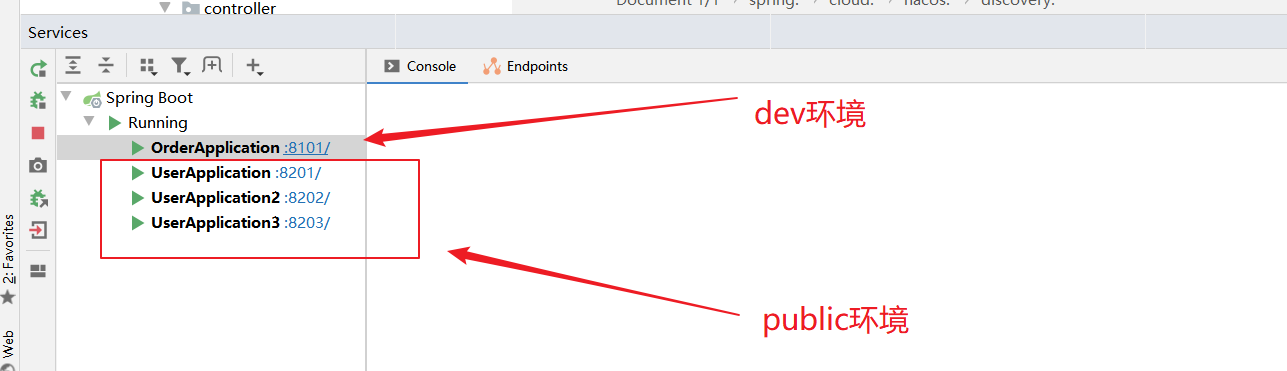
2：新建的dev命名空间：



3；修改服务所属的命名空间：



目前修改了order-service的命名空间，修改成了dev环境，其他的没改，都是在默认的public环境下

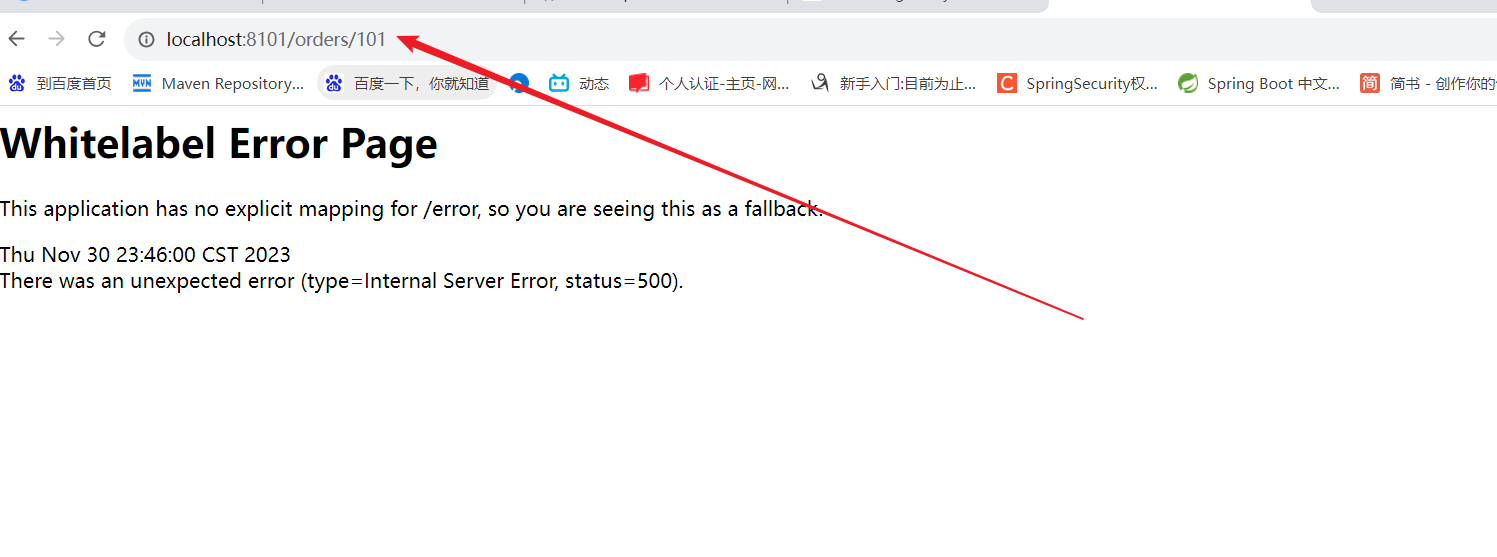


4：在nacos控制台的服务列表中，可以看到多了个dev菜单，即order-service实例和user-service实例已经不再同一个环境中了：





5：order-service实例和user-service实例不再同一个环境，已经是两个世界的人了，不能相互访问：



Nacos环境隔离小总结：

1：namespace用来做环境隔离；

2：每个namespace都有唯一id；

3：不同namespace下的服务不可见

可以利用不同namespace下的服务不可见的特性做环境隔离，比如生产环境、测试环境、开发环境

## Nacos和Eureka的对比：

### Nacos注册中心原理 ：

服务提供者在启动的时候会把自己的信息注册到注册中心，注册中心就会保存下服务提供者的信息，当服务消费者需要去消费时，服务消费者就会定时向注册中心拉取服务列表，拉取到服务列表之后发起远程调用。注意，这个拉取的动作不是每一次消费都要去做的，如果每次发请求时都要去做拉取，对注册中心来说涯里就太大了！所以消费者做服务拉取的时候，会将拉拉取到的服务信息缓存到服务列表缓存！这样拉取完一次，接下来的一段时间就可以用再去注册中心拉取，而是直接从缓存中获取服务提供者信息。当然了，服务列表缓存一直更新也不行，万一服务提供者变化了怎么办？所以会每隔30秒重新拉取一次服务列表进行更新！

Nacos对比Eureka的优点是：当服务列表发生变更时，Nacos会主动推送服务列表，即PULL与PUSH结合，服务列表更新及时、效率更高。

### Nacos和Eureka的区别在于服务提供者的健康检测！

差别一：nacos中，服务分为临时实例 与 非临时实例，默认情况下，实例是非临时实例！

临时实例和非临时实例在nacos做健康检测是不一样的：

     1：临时实例的健康检测是：临时实例会主动定期向注册中心发送心跳，实现服务的健康检测，当一段时间临时实例没有心跳上来，Nacos会把临时实例从服务列表中剔除：舔狗！

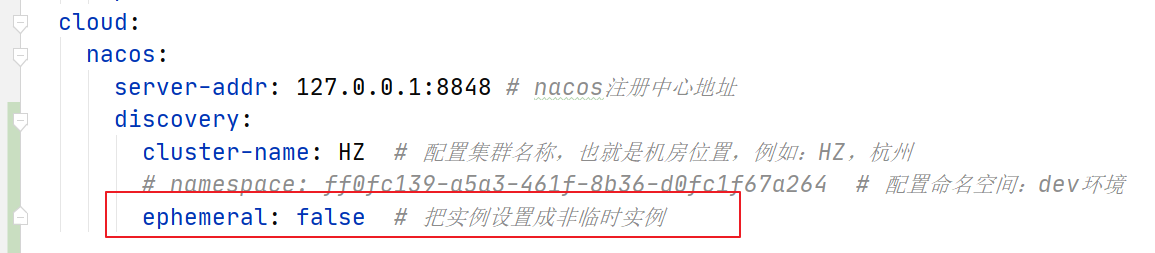
     2：非临时实例的健康检测则是：由Nacos主动去发请求询问非临时实例的健康状态，即使非临时实例挂掉了，Nacos也不会把非临时实例在服务列表中剔除：非舔狗！

差别二：eureka消费者只采取每隔30秒定时拉取一次服务列表，所以如果在30秒内，有服务提供者挂了，消费者是不知道的，如果此时直接去消费，是会出问题的；而nacos除了每隔30秒定时拉取一次服务列表外，当有服务消费者挂了，Nacos还能立即主动推送服务变更消息给消费者。

即Eureka采用的时pull，而nacos采用的时pull和push结合，做到服务更新效率高！

### Nacos临时实例和非临时实例设置

服务注册到nacos时，可以选择注册为临时和为临时实例，通过下面的配置来设置：



效果：停止order-service后，不管过多长时间，Nacos都不会把order-service在服务列表剔除掉：



总结：

Nacos与Eureka的共同点：

1：都支持服务注册和服务拉取；

2：都支持服务提供者心跳方式做健康检测。

Nacos与Eureka的区别：

1：Nacos支持服务端主动检测提供者状态：临时实例采用心跳模式，非临时实例采用主动检测模式;

2：临时实例心跳不正常会被剔除，非临时实例则不会被剔除;

3：Nacos支持服务列表变更的消息推送模式，服务列表更新更及时；

4：Nacos集群默认采用AP方式，当集群存在非临时实例时，采用CP模式；Eureka采用AP方式。

5：默认非临时实例就好。

## Nacos配置中心

之前的学习已经知道如何使用eureka和nacos作为服务的注册中心，以及使用ribbon作为负载均衡器。

今天学习Nacos作为配置管理中心的使用；

以及基于Feign的声明式远程调用，用Feign来代替之前的额RestTemplate的方式的远程调用，可以发现基于Feign实现的远程调用会非常非常优雅！代码逼格上升n个档次。

最后学习Gateway网关，这是微服务中非常重要的一个组件，就是起到微服务群看门大爷的作用，任何请求进来微服务都先要经过gateway。

## Nacos配置管理

1：什么是统一配置管理，以及为什么需要统一配置管理，和怎么使用统一配置！

2：在统一配置管理中怎么去实现配置的热更新；

3：不同微服务之间、微服务不同环境之间的配置如何共享；

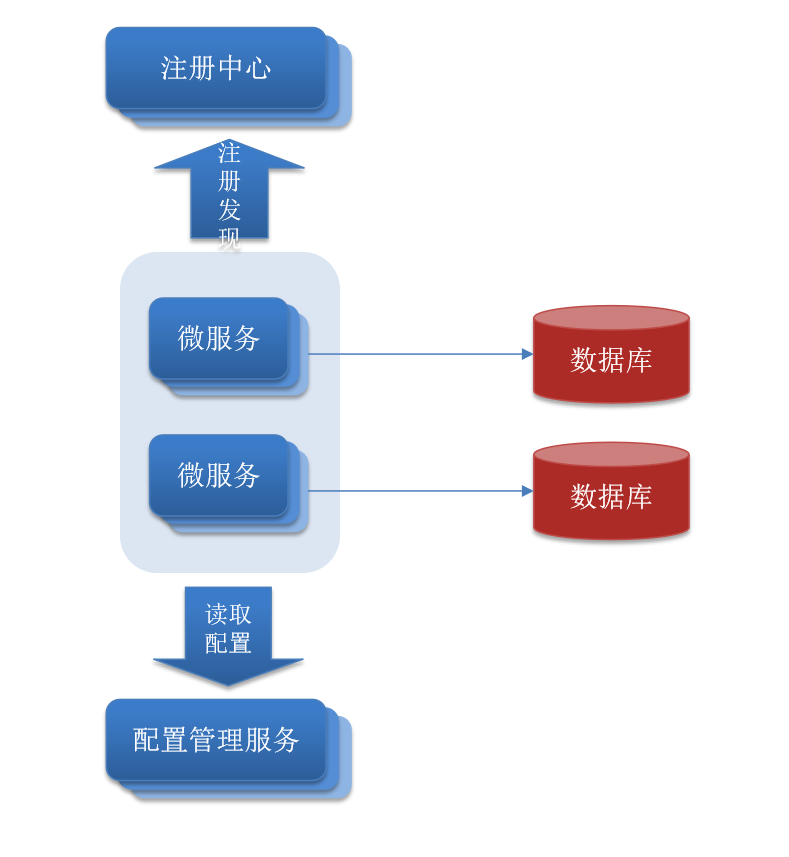
4：搭建生产环境可用的Nacos集群

### 统一配置管理：

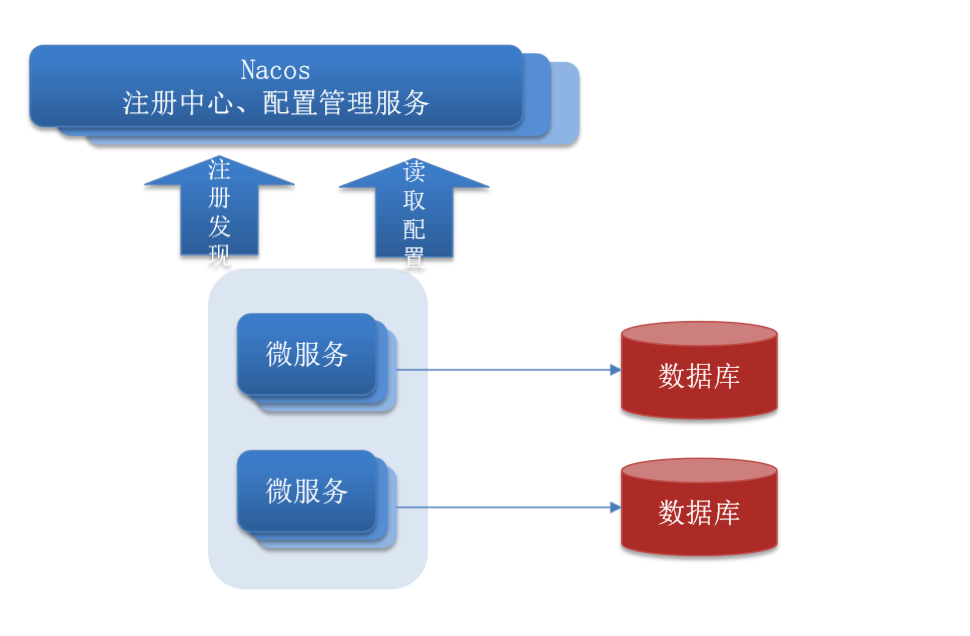
在生产环境中，随着微服务越来越多，甚至可能达到成百上千的情况，此时，如果服务的配置文件需要变化、修改，并且这个配置文件可能跟数十个微服务都有关系，这个时候要逐个微服务去调整配置？这样会很麻烦，第二，就算逐个调整完配置以后，这些关联的微服务都得重启，在生产环境下，一个服务的重启带来的影响是很大的，何况数十个重启。

所以，我们的需求就是希望这些微服务的配置文件实现统一的管理，比如说，现在数十个微服务的配置需要修改，现在不需要逐个去修改，而是在一个地方完成改动就行！并且改动完了之后，影响的这些服务不用去做重启，这些配置就立马能够生效，这就叫配置的热更新!

以上要如何做到？答：就需要配置管理服务！及配置中心，这个服务的作用是回去记录微服务中核心的配置，那么微服务启动的时候，就回去读取配置中心里的配置，然后和自己本地的配置做结合，作为完整配置去使用。将来这些核心配置如果要做修改，我们不用逐个服务去改动，而是找到配置中心服务里去修改，在里面把需要变化的配置去做修改！配置中心发现了配置有改动，会立即通知对应的微服务，微服务就会去配置中心完成配置的读取，并且完成配置的热更新，即不用重启，新配置自动生效！



因为Nacos同时具备注册中心和配置中心的功能，所以：



当然，你公司财大气粗，两套Nacos集群，一套做配置中心，一套做注册中心，也没问题！

### Nacos配置管理实现：

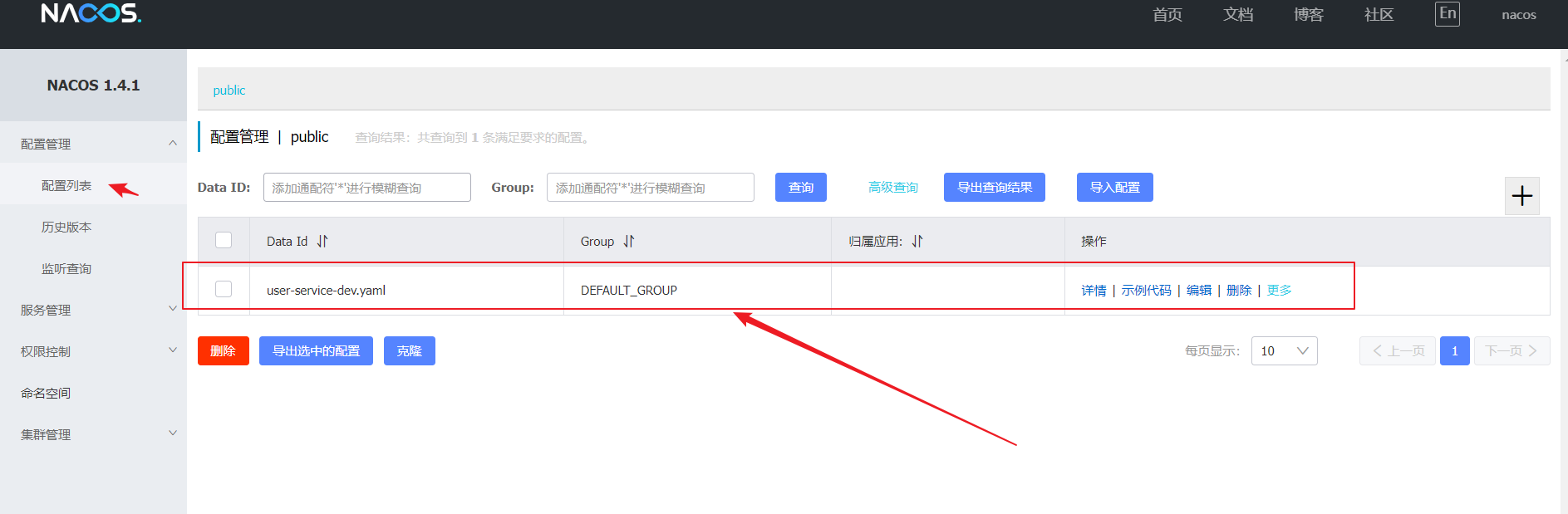
第一步：添加配置



第二步：



3：查看新建的配置文件：



以上就已经完成了配置的统一管理了，就是把配置交给Nacos去做管理，至于微服务如何去读取这个配置，请看下面内容！

  总结：

一：在生产环境中，随着微服务越来越多，甚至可能达到成百上千的情况，此时，如果服务的配置文件需要变化、修改，从而会影响不少的服务，此时会有以下情况：

     1：需要对服务的文件逐个调整，

     2：调整完服务的配置之后，与配置关联的众多服务都需要重启

     在生产环境下，服务重启的影响时挺大的，所以，就有以下的需求：

     1：对服务的配置进行统一的管理，比如只修改一个地方，就可以完成众多服务的修改；

     2：配置改动完之后，受影响的服务不需要重启，配置就能立马生效，实现配置的热更新。

二：需要热更新的配置才放到统一配置管理里去：比如一些开关配置

在Nacos管理界面中配置：配置命名规则：服务命-[profile].yaml

### 微服务获取统一配置

我们已经把部分配置放到nacos配置中心，现在就是要想办法得到这些配置，怎么得到呢？

我们先看一下在没有nacos配置中心时，服务是如何获取配置的！

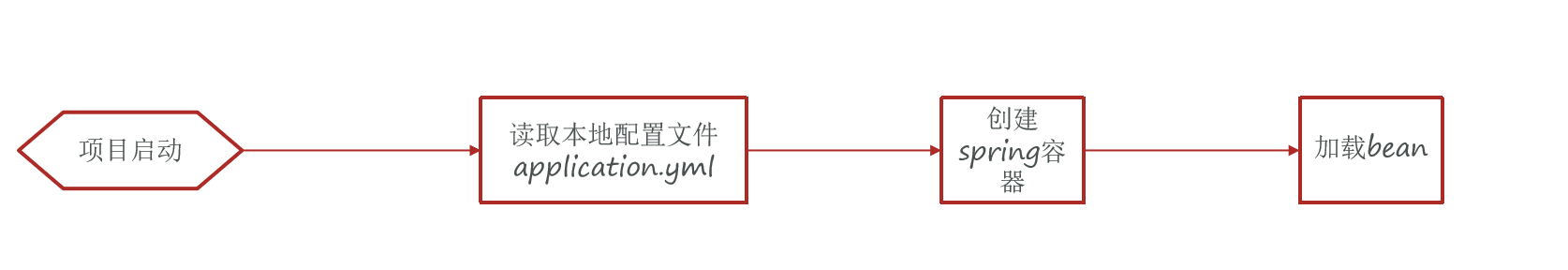
流程如下：

1：项目启动；

2：项目启动以后，会读取本地配置文件application.yml；

3：读取完本地配置文件，创建Spring容器；

4：创建完spring容器，就会去加载各种各样的bean



以上是在没有nacos配置中心时，服务获取配置的流程！现在我们多了一个配置文件，就是在nacos配置中心的配置文件，将来项目会把Nacos中的配置和本地配置做一个合并，而后再去完成后续的容器创建、Bean的创建等动作。所以Nacos配置要加入到以上的流程当中，于是就变成：

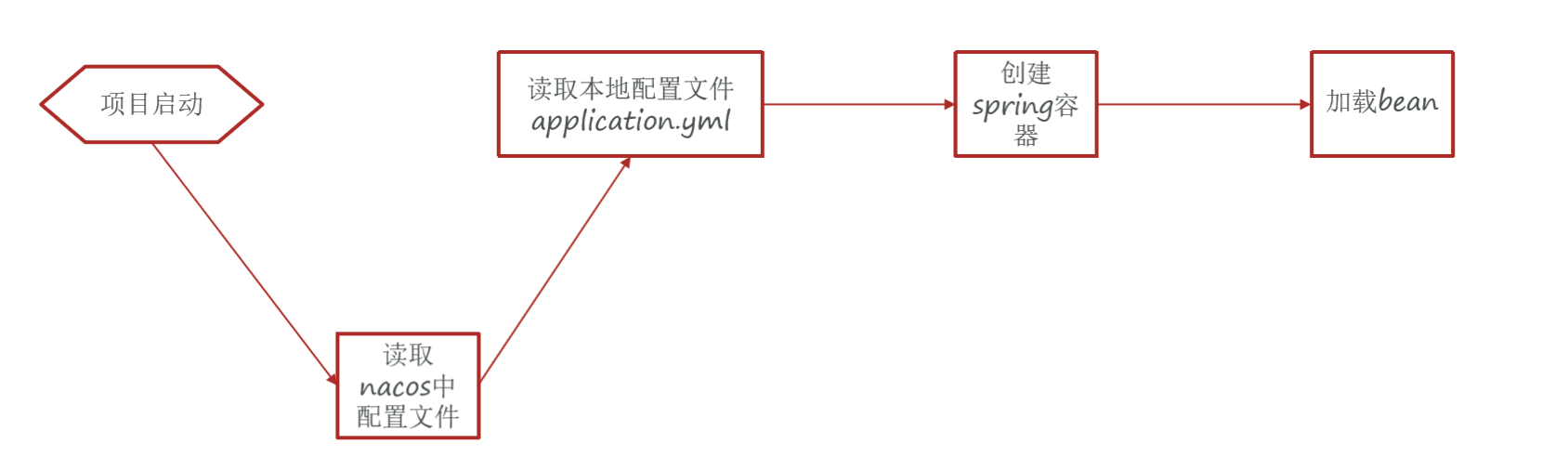
1：项目启动；

2：先去读取Nacos配置中心的配置文件；

2：在来读取本地配置文件application.yml，两者配置合并；

3：读取完本地配置文件，创建Spring容器；

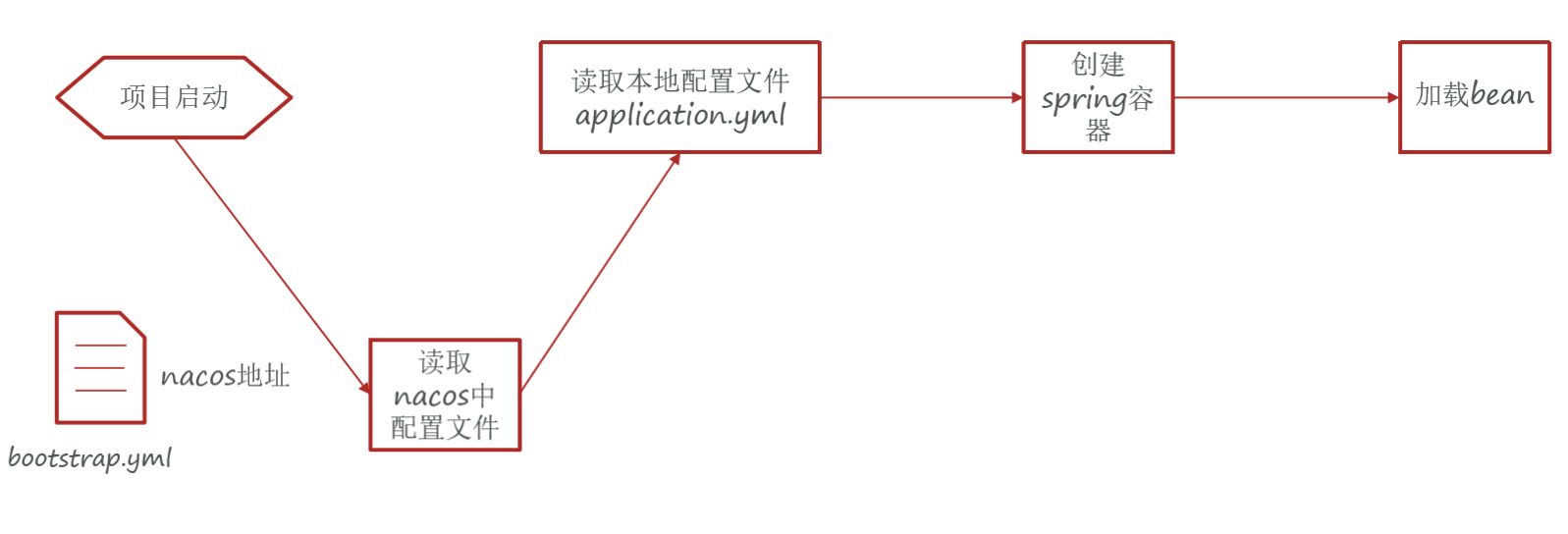
4：创建完spring容器，就会去加载各种各样的bean



以上流程看起来很简单，但是需要注意服务读取Nacos配置中心的配置的时候，需要知道一些信息，1：去哪读取？2：读取谁？

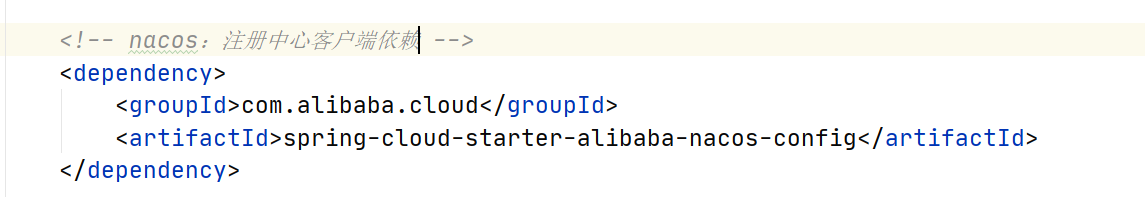
所以服务在读取nacos配置中心配置文件的时候，最起码要知道nacos的地址，但是现在的nacos地址写在application.Yml中，但以上流程，我们要在2步骤就要读取Nacos配置中心的配置，所以我们要提前知道Nacos的地址，所以在项目启动之后，先得得到Nacos的地址，所以现在的问题是，SpringBoot项目中，有什么是比application.yml还要提前的？

回答：bootstrap.yml，这个文件的有优先级回避application.yml的优先级高很多，所以这样一来，项目启动的时候会优先读取bootstrap.yml，那我们只需要把nacos地址啊、Nacos配置中心的配置文件的相关信息啊都配到bootstrap.Yml，那么就可以完成Nacos配置的读取了，而后再去跟本地配置结合完成后续动作！

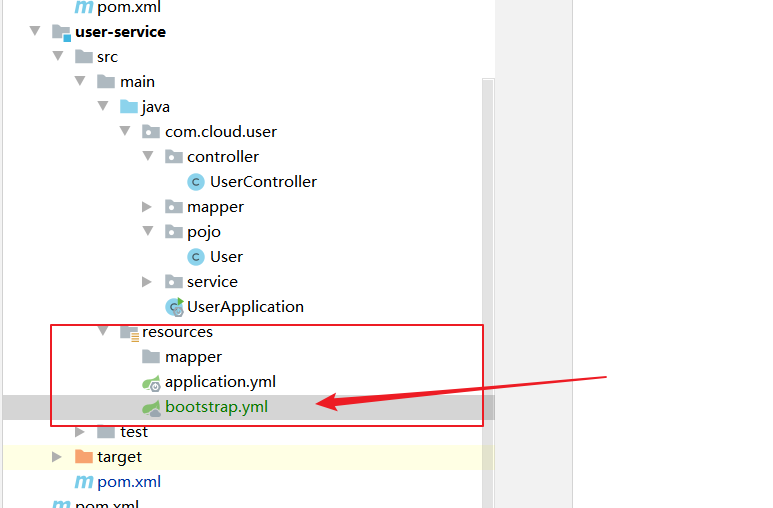


即与Nacos地址和配置文件有关的所有信息都应该放在bootstrap.yml当中！

步骤1：引入Nacos的配置管理客户管依赖！



步骤二：在resource目录中添加一个bootstrap.yml文件，这个文件是引导文件，优先级高于application.yml



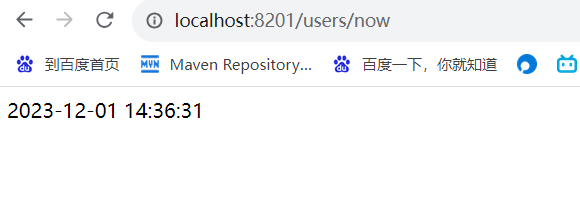
步骤三：将Nacos地址和配置文件有关的所有信息放在bootstrap.yml当中：

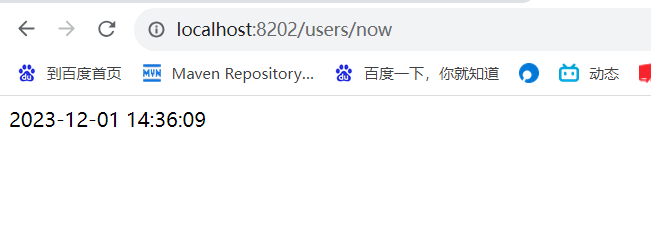


步骤四：读取配置：



步骤五：观察结果：







说明格式化成功了，说明微服务获取配置中心的配置成功了！

总结：将配置交给Nacos管理的步骤：

1：在Nacos控制台中添加配置文件

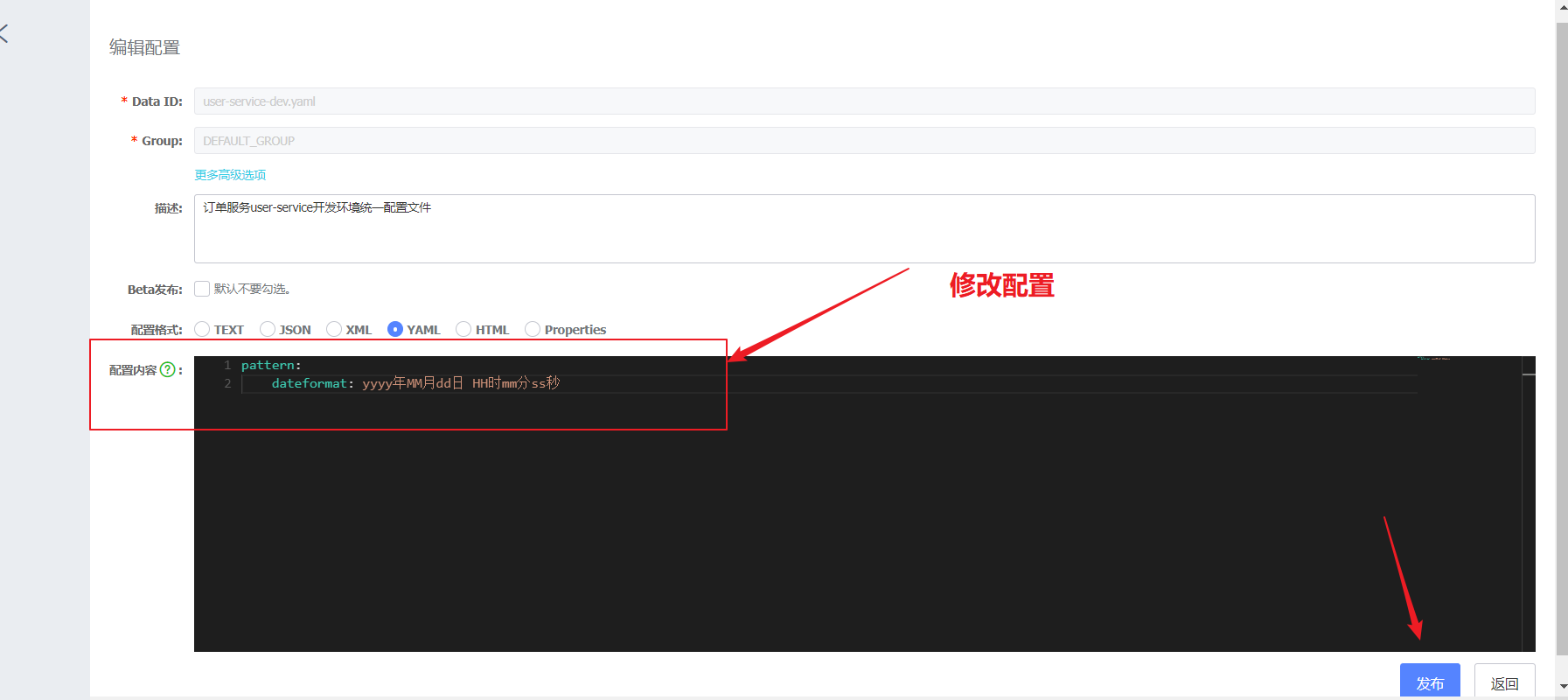
2：在微服务中引入Nacos的config依赖；

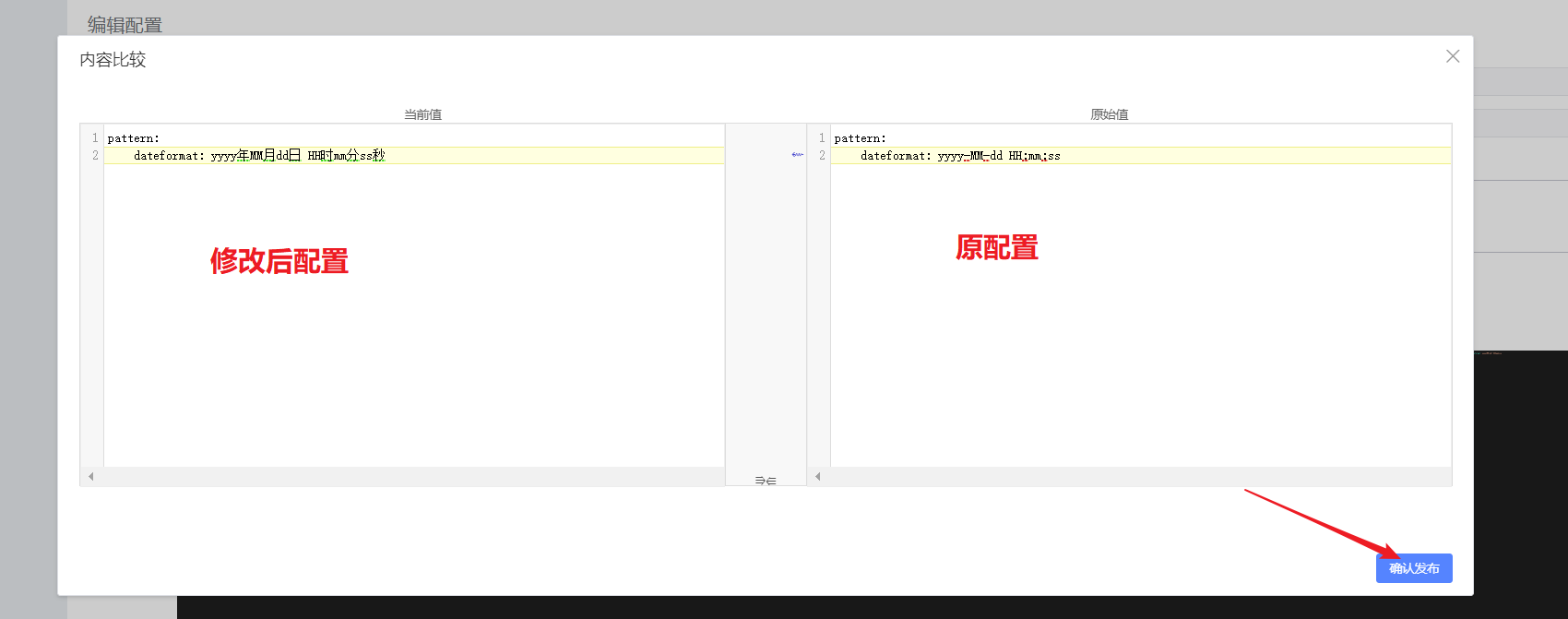
3：在微服务中添加bootstrap.yml，配置nacos地址、当前环境、服务名称、文件后缀名。这些决定了程序启动时去nacos读取哪个文件。

### 如何实现配置的热更新

以上已经实现了配置的统一管理，即已经把配置文件交给了Nacos去做管理，并且微服务也已经能够成功地从nacos拉取到配置了，但是这不是最终目标，最终目标是实现配置的热更新。

1：编辑修改Nacos配置文件：





我们希望改完配置之后，新配置立马就能生效，但是：



不是说好配置热更新吗？怎么没有实现呢？

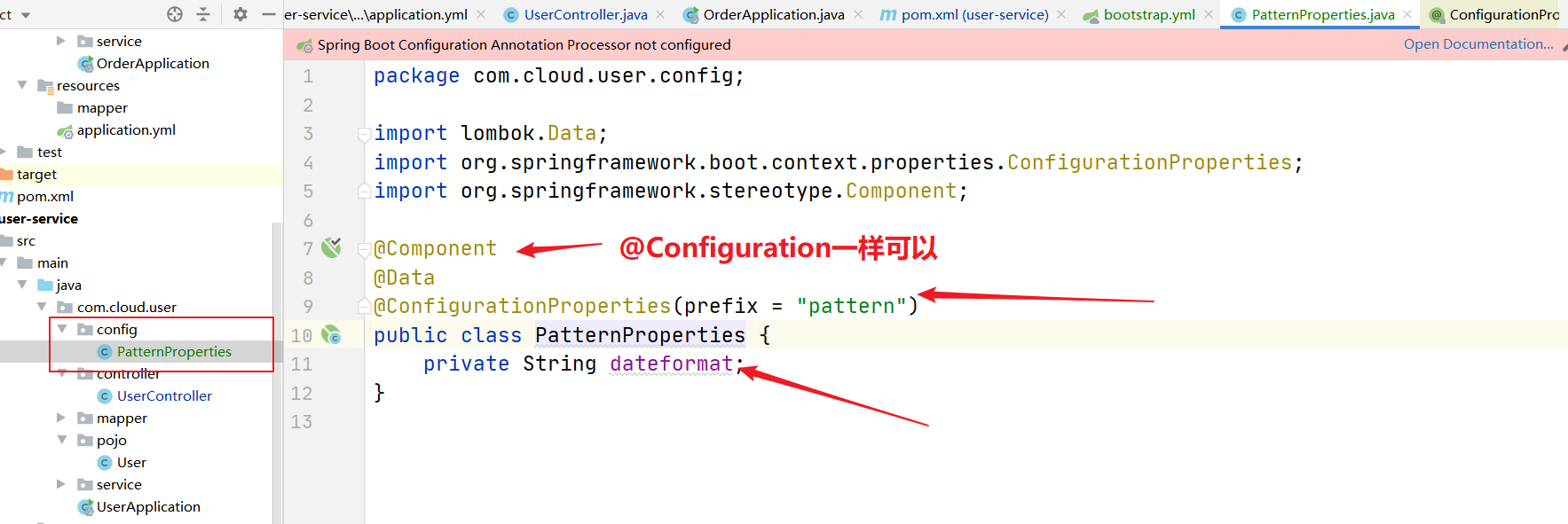
答：这是因为我们还缺少了一些东西：要想实现配置的自动更新，还需要一下步骤：

Nacos中的配置文件变更后，微服务无需重启就可以感知，不过需要通过下面两种配置实现：

方式一：在@Value注解注入的变量的类上添加注解@RefreshScope



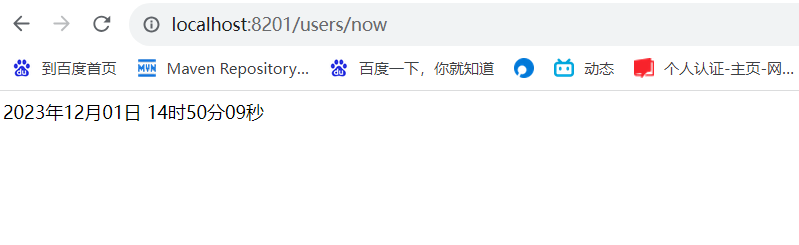
方式二：使用@ConfigurationProperties注解，更常用



使用时修改成：



不管是方式一还是方式二，都可以看到服务不需要重启，配置自动更新：



#### 小总结

Nacos配置变更之后，微服务可以实现热更新：方式：

1：通过@Value注解注入，结果@RefreshScope来刷新；

2：通过@ConfigurationProperties注解注入，自动刷新；

注意事项：

1：不是所有的配置都适合放到配置中心，维护起来比较麻烦；

2：建议将一些关键参数，需要运行时调整的参数放到Nacos配置中心，一般都是自定义配置。

### Nacos配置管理：多环境配置共享

前面的学子中，我们已经学会了Nacos配置管理的基本用法，实现了配置的热更新，现在来了解一下微服务之间的配置共享问题！

有什么情况下会碰到微服务配置共享呢？

比如以下场景：

有一个配置属性，在开发、测试、生产不同环境下的值是一样的，像这样的配置，不需要在每一个环境的配置文件都写一份，而且将来如果要改动，如果每隔配置文件都写一份，那么需要每个配置文件都要去改！这样显示时不合适的，我们想要找到一个地方，只配一次。不管环境如何变，这个配置都可以被加载，这就是多环境配置共享的场景需求！

微服务启动时会从Nacos读取多个配置文件：

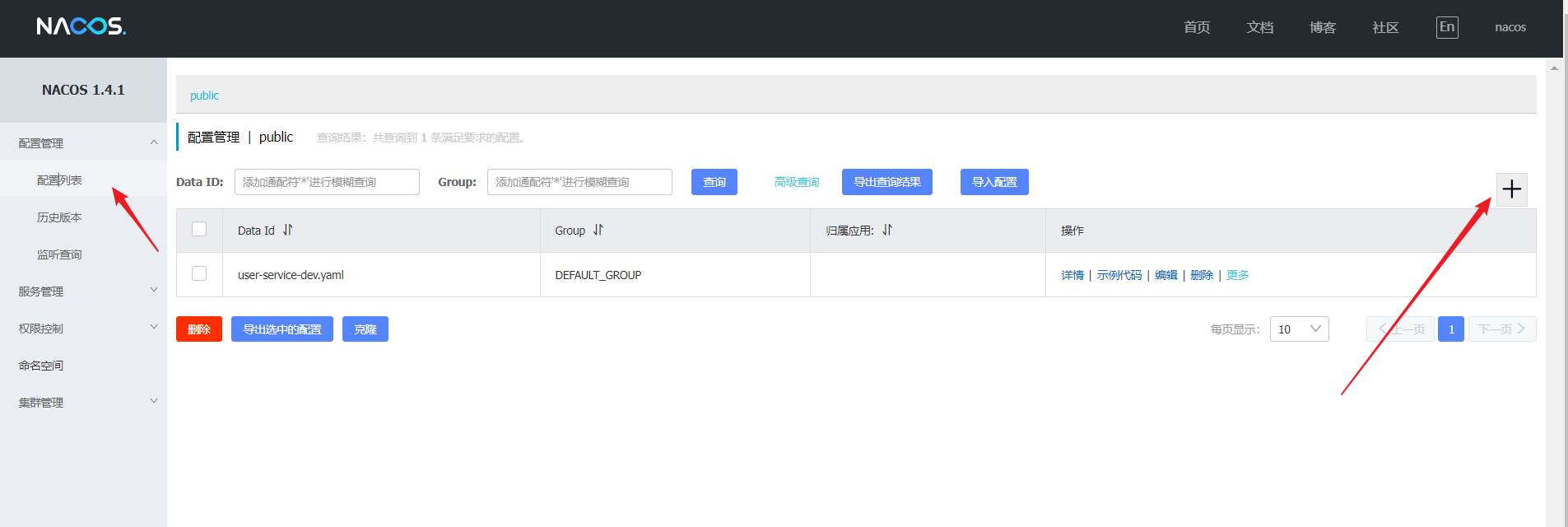
第一个：[spring.application.name]-[spring.profiles.active].yaml，例如：user-service-dev.yaml，即特定环境配置；

第二个：[spring.application.name].yaml，例如：user-service.yaml，第二个显然跟环境没有关系，环境发生改变时，第一个文件会改变，但第二个一定不会变化，可以认为，不管环境怎么变，微服务启动时一定会读取第二个文件，即默认配置，多环境共享。

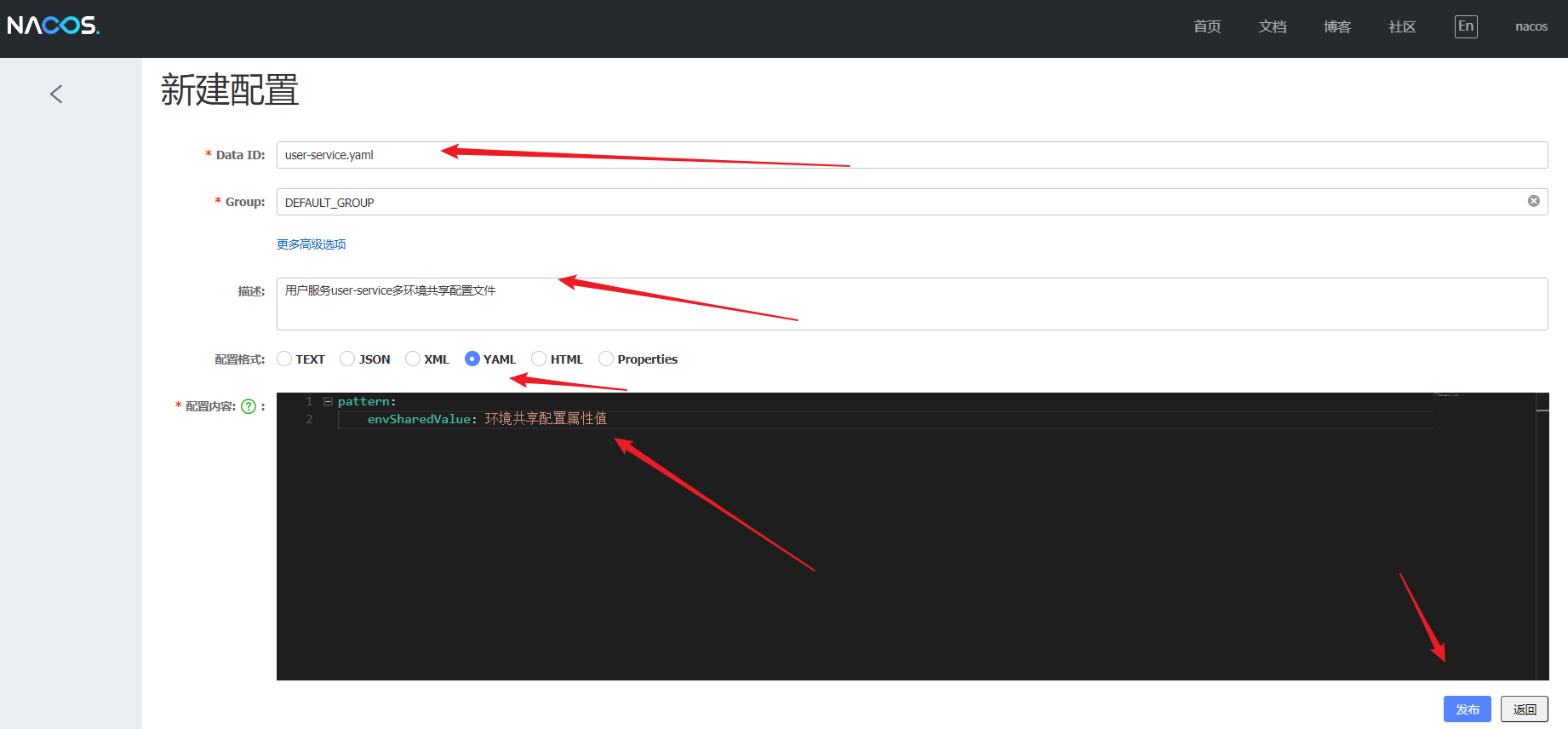
无论profile如何变化，[spring.application.name].yaml这个文件是一定会加载的，因此多环境共享配置可以写入这个文件！

步骤：

步骤一：在nacos新增一个多环境共享的配置文件：



user-service.yaml多环境共享配置文件：

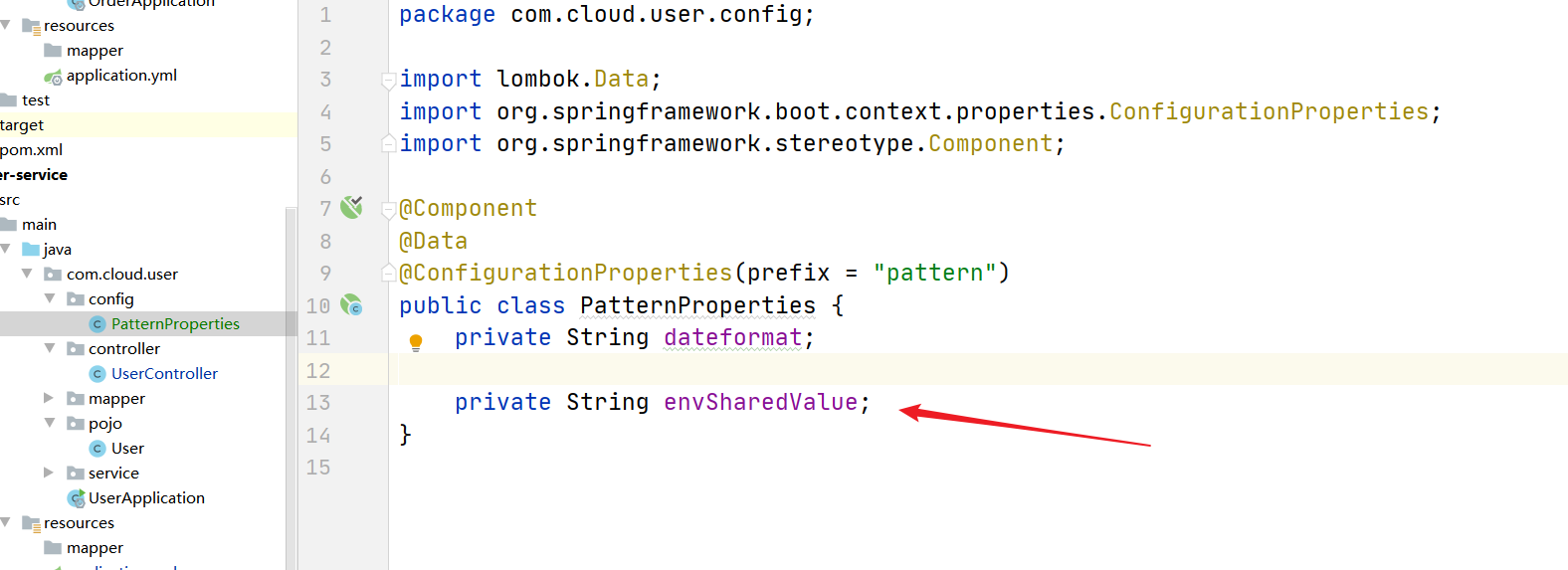


配置列表：开发环境配置文件和多环境共享配置文件：

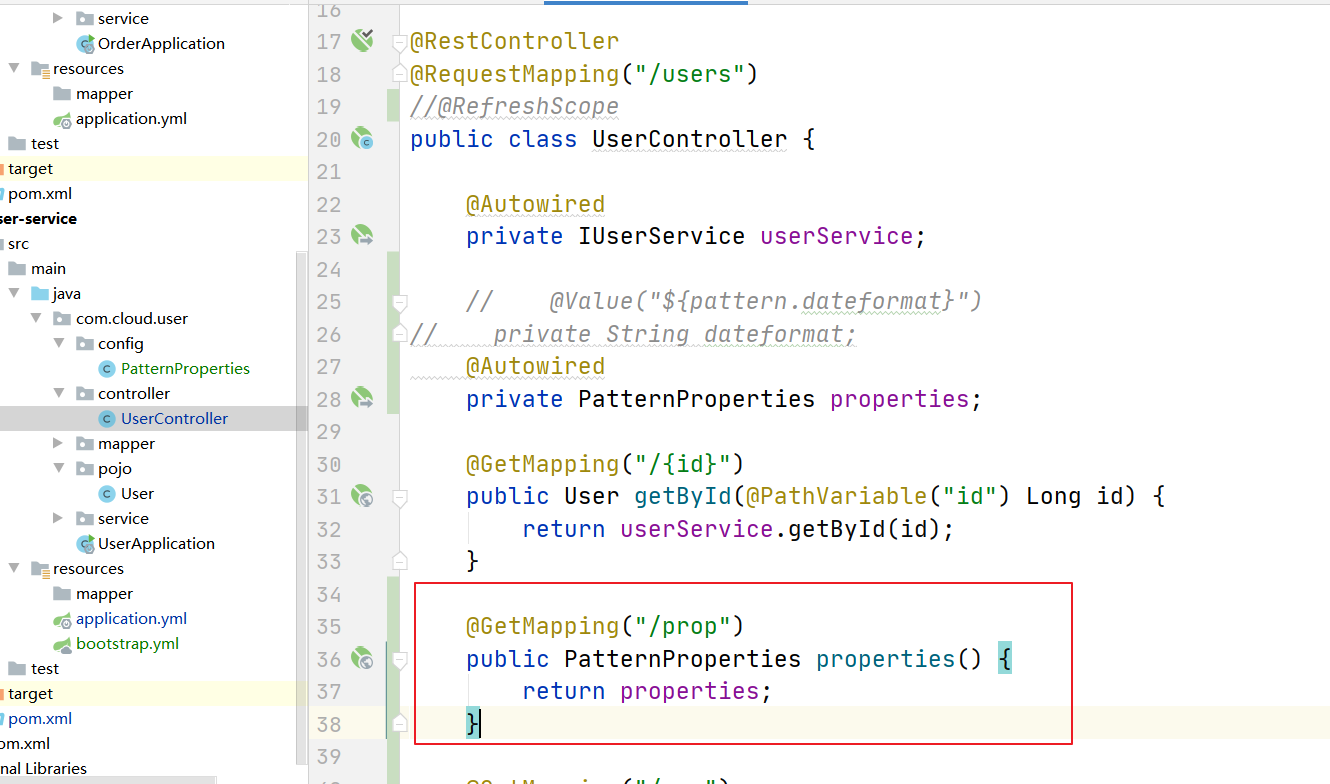


第二步：读取多环境共享配置文件：

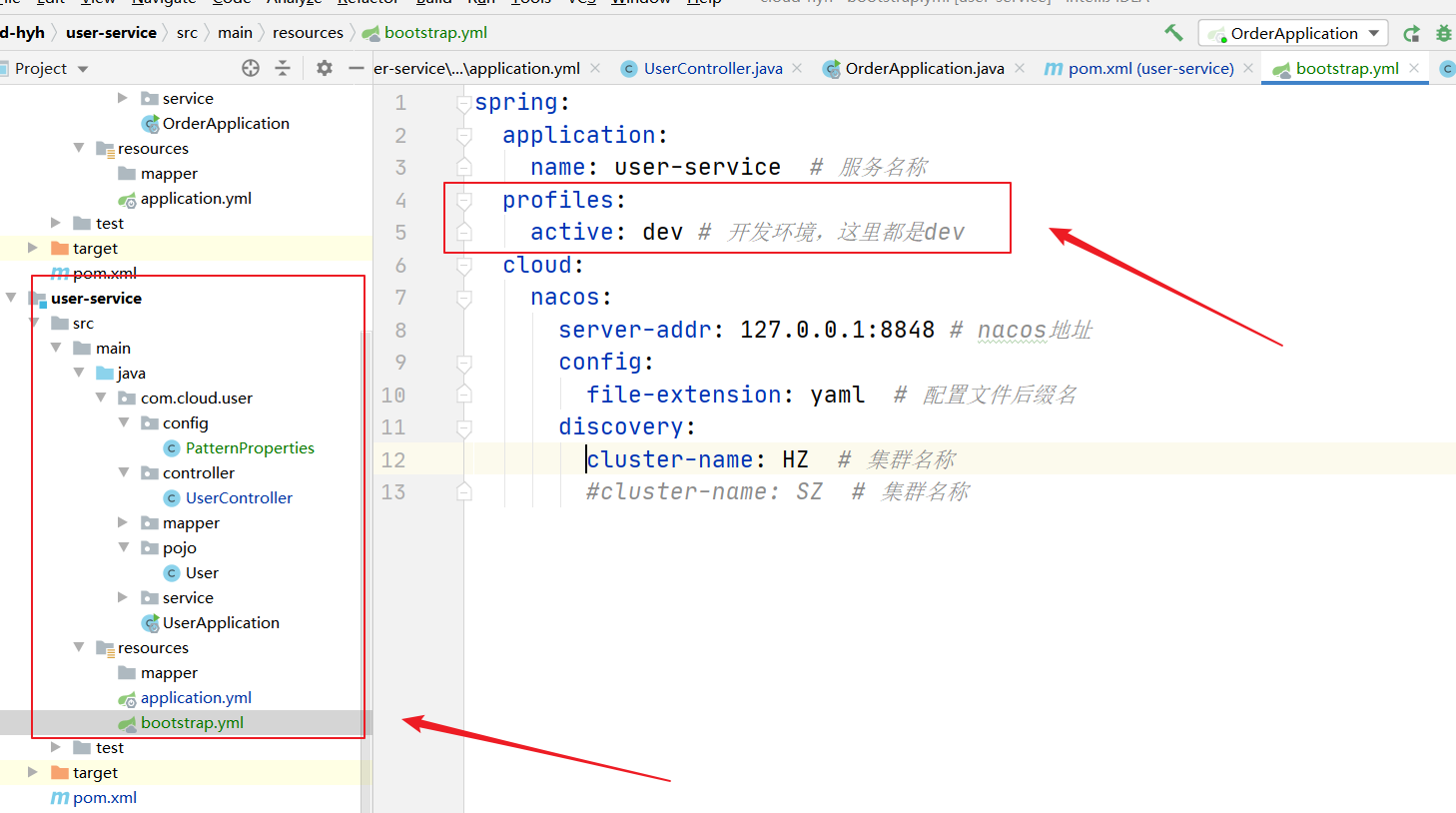
2.1：修改配置类



2.2：修改controller



注意了，此时的user-service实例实在dev环境下的，所以可以同时读取user-service-dev.yaml和user-service.yml这两个配置文件：

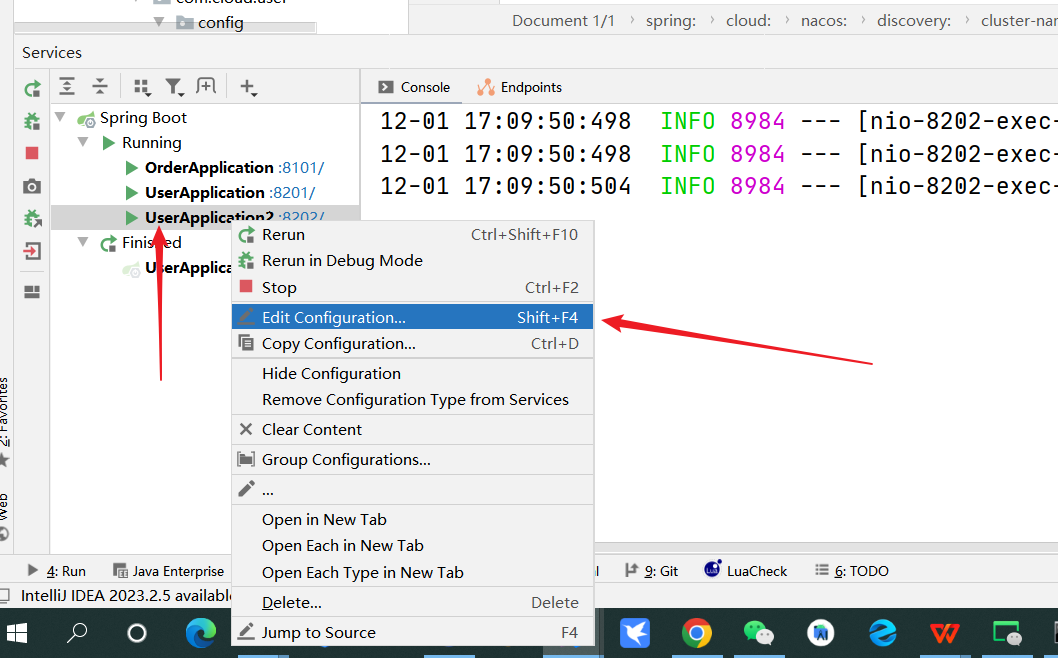


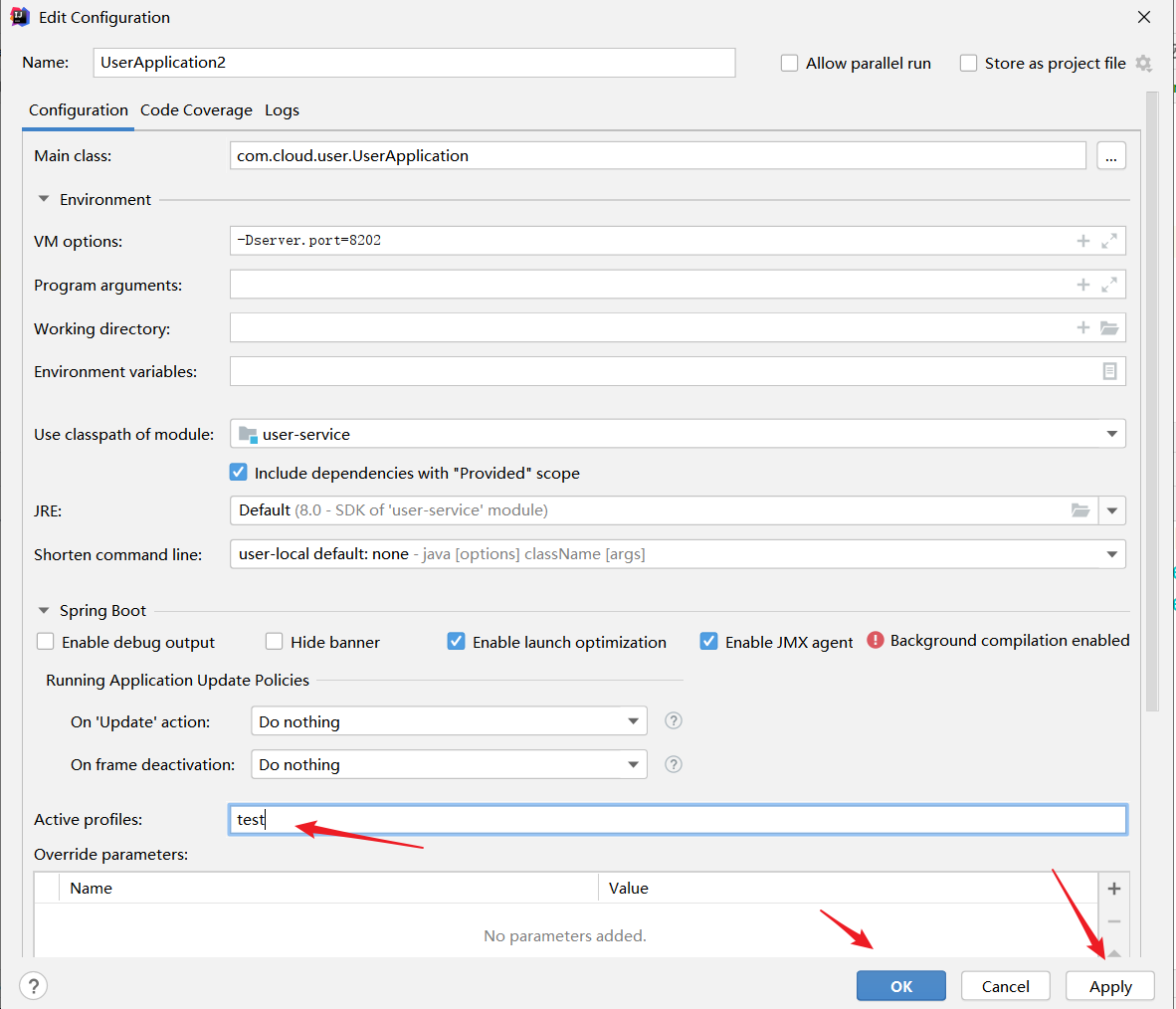
第三步：查看效果，可以看到共享配置可以读取



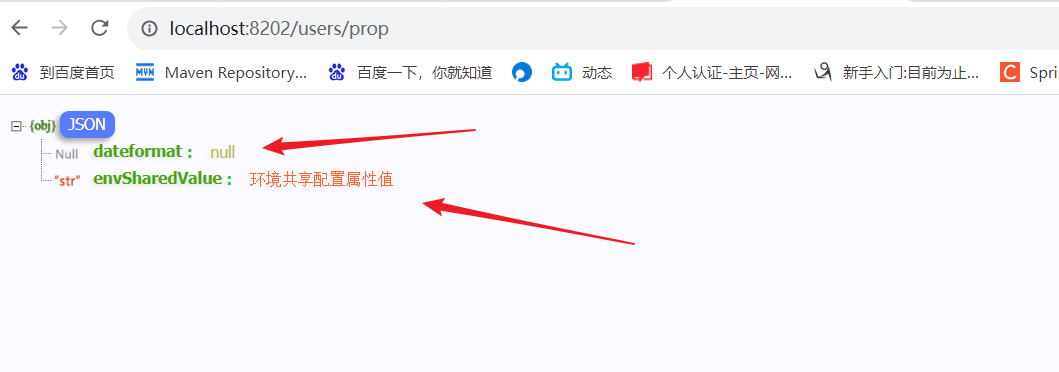


第四步：修改8202实例的配置：把dev开发环境改成test测试环境：





第五步：重启8202实例，查看看到共享配置属性还是可以读取，但是dev环境配置读取不了了，但是dev环境的8081还是可以读到全部配置属性：

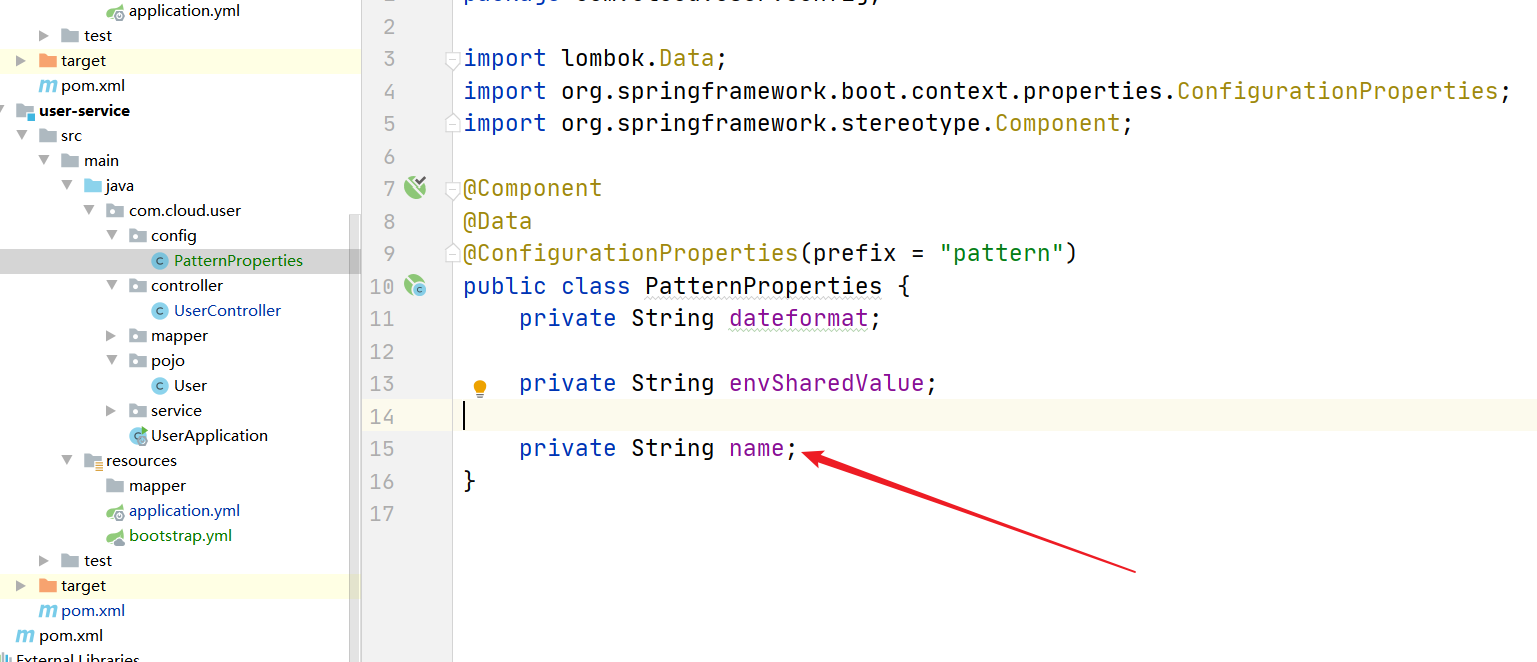




问一个问题，此时Nacos有两个配置文件：user-service-dev.yml 和 user-service.yml，并能做到多环境配置共享，但是假设这两个配置文件中都存在相同的配置，那么读取的时候是读取谁的呢，会以谁的准呢？即配置文件的优先级！而且远端的nacos有配置，本地的application也有配置，那有又以谁的优先级为准呢？

检验方法：

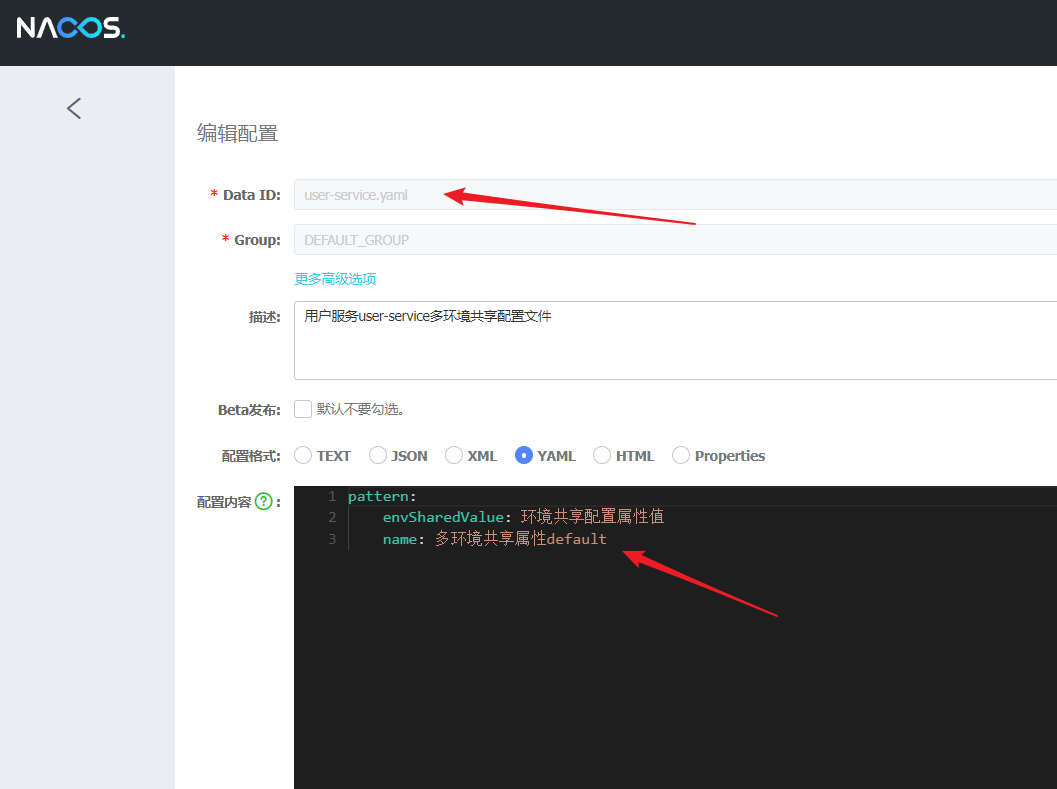
在user-service-dev.yml 和 user-service.yml 和 application.yml都配置一个相同的配置：pattern.name



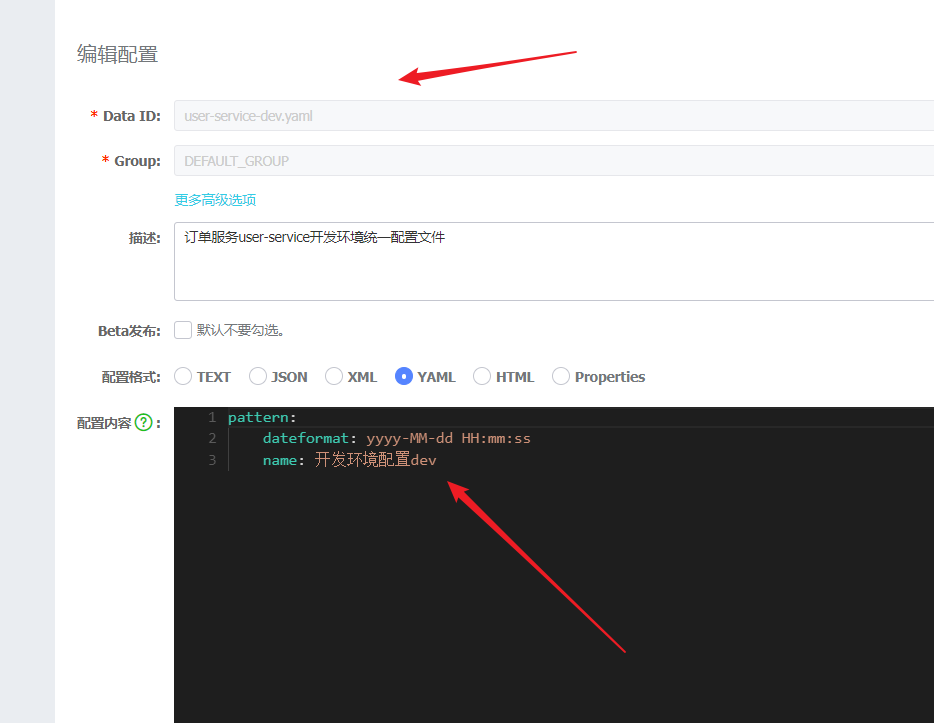
**application.yml：**



**user-service.yaml：**



**user-service-dev.yaml：**



**多种配置优先级的测试结果是：**

**user-service-dev.yaml > user-service.yaml > application.yml**

**即：**

**（nacos中当前环境配置）服务名-profile.yaml  > （nacos中共享配置）服务名.yaml > 本地配置**

