

# 一:微服务 & 微服务架构

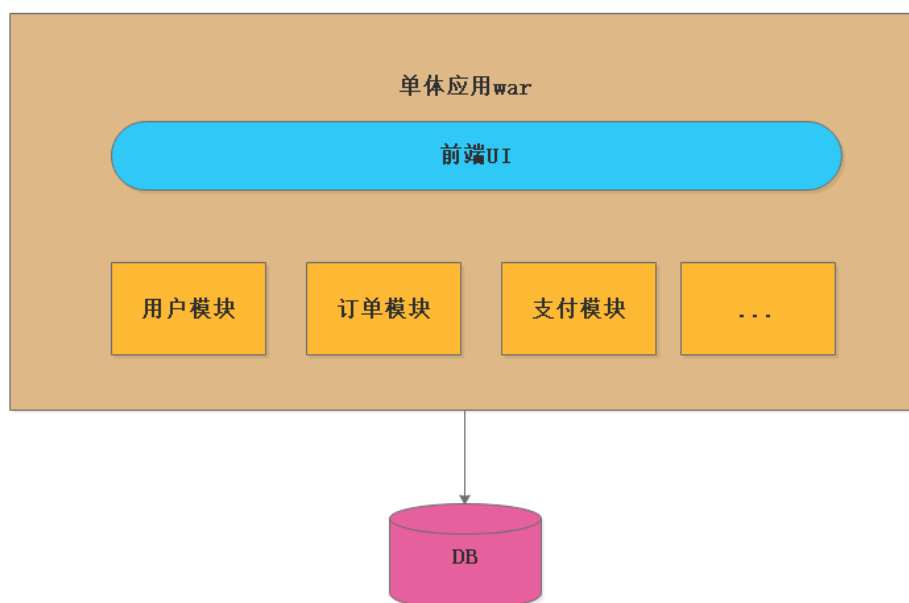
## 1: 单体架构 VS 微服务架构

### 1.1)从单体架构说起

一个工程对应一个归档包(war)，这个war包 包含了该工程的所有功能。我们成为这种应用为单体应用，也就是我们常说的单体架构(一个war包打天下)。

具体描述: 就是在我们的一个war包种，聚集了各种功能以及资源，比如JSP JS,CSS等。而业务种包含了我们的用户模块，订单模块，支付模块等等.

### 1.2)单体架构图



### 1.3)单体架构优缺点总结

优点:

- ①: 架构简单明了，没有“花里胡哨”的问题需要解决。
- ②: 开发，测试，部署简单 (尤其是运维人员 睡着都会笑醒)

缺点:

- ①: 随着业务扩展，代码越来越复杂，代码质量参差不齐(开发人员的水平不一),会让你每次提交代码，修改每一个小bug都是心惊胆战的。
- ②: 部署慢(由于单体架构，功能复杂) 能想像下一个来自200W+代码部署的速度(15分钟)
- ③: 扩展成本高，根据单体架构图 假设用户模块是一个CPU密集型的模块(涉及到大量的运算)那么我们需要替换更加牛逼的CPU，而我们的订单模块是一个IO密集模块 (涉及大量的读写磁盘)，那我们需要替换更加牛逼的内存以及高效的磁盘。但是我们的单

体架构上 无法针对单个功能模块进行扩展，那么就需要替换更牛逼的CPU 更牛逼的内存 更牛逼的磁盘 价格蹭蹭的往上涨。

④:阻碍了新技术的发展。。。。。。比如我们的web架构模块 从struts2迁移到springboot, 那么就会成为灾难性

#### 1.4) 微服务以及微服务架构



##### 1.4.1)微服务的定义

①: 英文:<https://martinfowler.com/articles/microservices.html>

②: 中文:<http://blog.cuicc.com/blog/2015/07/22/microservices>

1.4.2) 微服务核心就是把传统的单机应用, 根据业务将单机应用拆分为一个一个的服务, 彻底的解耦, 每一个服务都是提供特定的功能, 一个服务只做一件事,类似进程, 每个服务都能够单独部署, 甚至可以拥有自己的数据库。这样的一个一个的小服务就是 微服务.

①: 比如传统的单机电商应用, tulingshop 里面有 订单/支付/库存/物流/积分等模块(理解为service)

②:我们根据 业务模型来拆分,可以拆分为 订单服务, 支付服务, 库存服务, 物流服务, 积分服务

\*③\*若不拆分的时候, 我的非核心业务积分模块 出现了重大bug 导致系统内存溢出, 导致整个服务宕机.

,若拆分之后, 只是说我的积分微服务不可用, 我的整个系统核心功能还是能使用

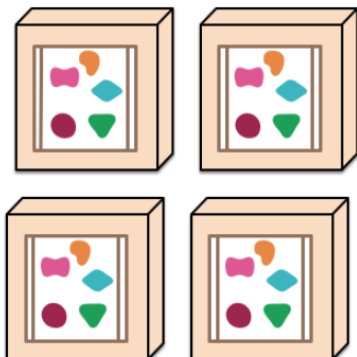
一个单体应用程序把它所有的功能放在一个单一进程中...



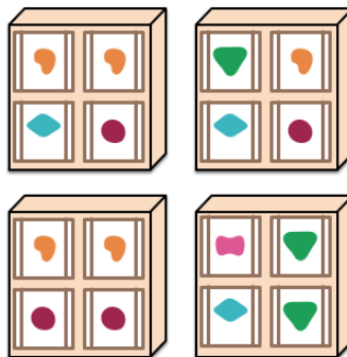
一个微服务架构把每个功能元素放进一个独立的服务中...



...并且通过在多个服务器上复制这个单体进行扩展

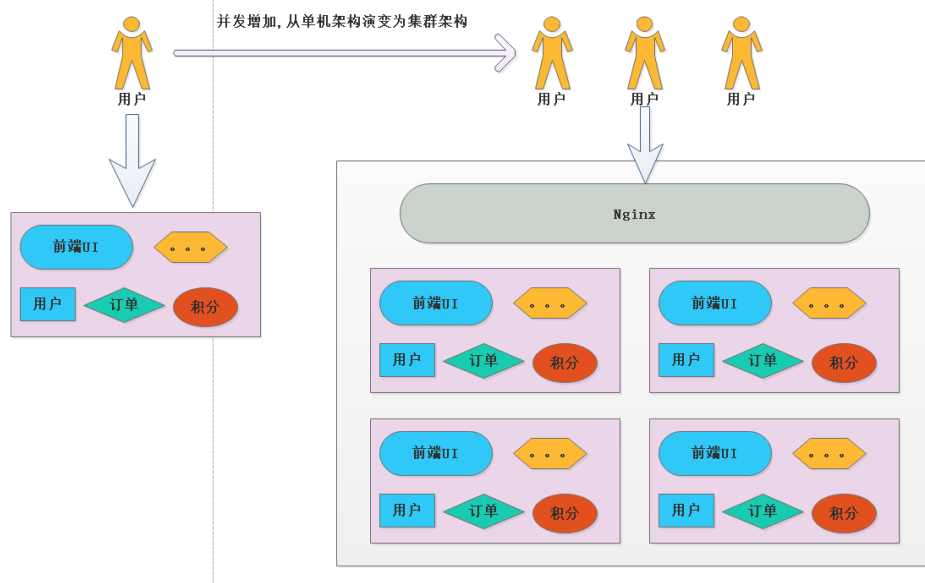


...并且通过跨服务器分发这些服务进行扩展，只在需要时才复制。

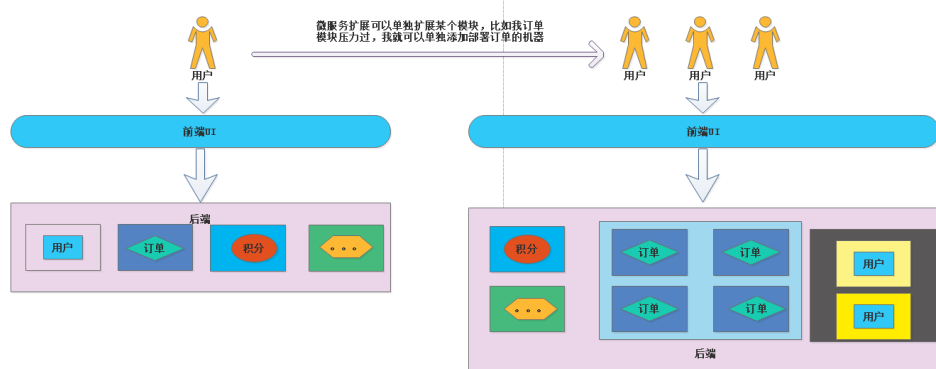


### 1.4.3) 单机架构扩展与微服务扩展

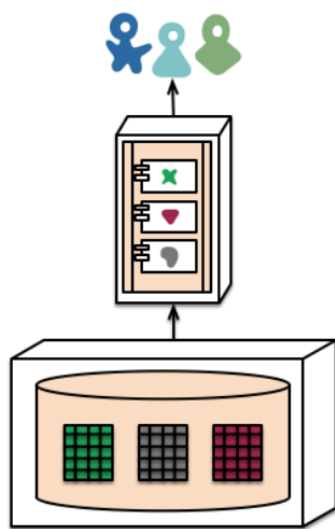
#### ①：单机架构扩展



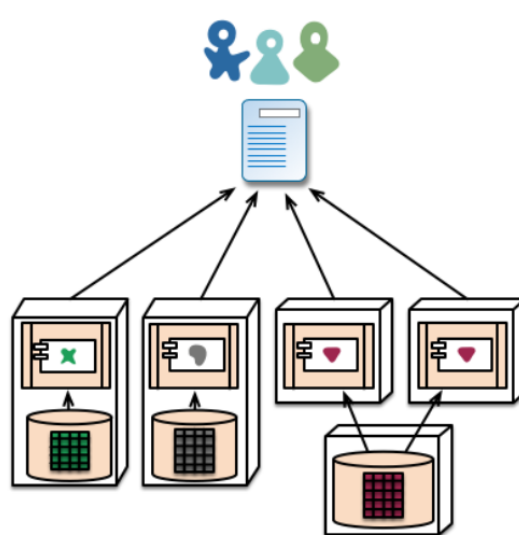
#### ②：微服务架构以及扩展



#### ③：微服务数据存储



单体 - 单一数据库



微服务 - 应用程序数据库

#### 1.4.4)微服务架构是什么？

**微服务架构是一个架构风格, 提倡**

- ①:将一个单一应用程序开发为一组小型服务.
- ②:每个服务运行在自己的进程中
- ③:服务之间通过轻量级的通信机制(http rest api)
- ④:每个服务都能够独立的部署
- ⑤:每个服务甚至可以拥有自己的数据库

#### 1.4.5) 微服务以及微服务架构的是二个完全不同的概念。

**微服务**强调的是服务的大小和对外提供的单一功能，而**微服务架构**是指把 一个一个的微服务组合管理起来，对外提供一套完整的服务。

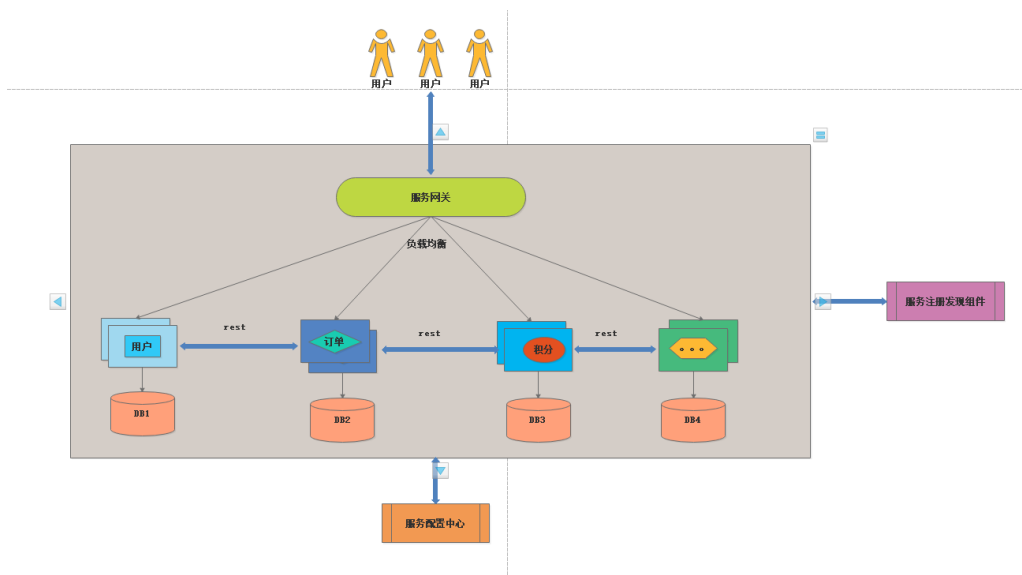
#### 1.4.6)微服务的优缺点

**A:优点:**

- ①：每个服务足够小,足够内聚，代码更加容易理解,专注一个业务功能点(对比传统应用，可能改几行代码 需要了解整个系统)
- ②：开发简单，一个服务只干一个事情。（加入你做支付服务，你只要了解支付相关代码就可以了）
- ③：微服务能够被2-5个人的小团队开发，提高效率
- ④：按需伸缩
- ⑤：前后段分离, 作为java开发人员，我们只要关系后端接口的安全性以及性能，不要去关注页面的人机交互(H5工程师)根据前后端接口协议，根据入参，返回json的回参
- ⑥：一个服务可用拥有自己的数据库。也可以多个服务连接同一个数据库.

**缺点:**

- ①:增加了**运维人员的工作量**，以前只要部署一个war包，现在可能需要部署成百上千个war包 (k8s+docker+jenkins )
- ②: 服务之间相互调用，增加**通信成本**
- ③:**数据一致性问题(分布式事物问题)**
- ④:系统**监控等,问题定位**.....



### 1.4.6) 微服务的适用场景

#### A: 合适

- ①:大型复杂的项目.....(来自单体架构200W行代码的恐惧)
- ②:快速迭代的项目.....(来自一天一版的恐惧)
- ③:并发高的项目.....(考虑弹性伸缩扩容的恐惧)

#### B: 不合适

- ①: 业务稳定, 就是修修bug, 改改数据
- ②: 迭代周期长 发版频率 一二月一次.

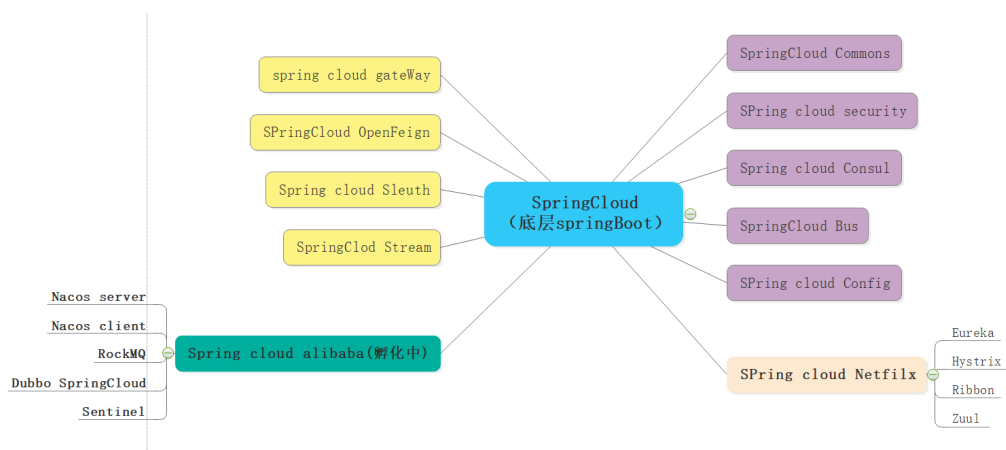
## 二:Spring Cloud Alibaba

### 2.1)什么是SpringCloud?<https://spring.io/projects/spring-cloud>

spring cloud子项目孵化器地址:<https://github.com/spring-cloud-incubator> 孵化成功就变为springcloud的子项目了。

SpringCloud是程序员用来开发我们微服务的一整套技术方案.包含如下  
**服务注册发现**, **服务容错降级**, **服务网关**, 服务调用, 服务调用**负载均衡**,消息等.

功能	作用	选择方案
Distributed configuration	分布式配置中心	Springcloud config, zk, <b>Nacos</b>
Service register and discovery	服务注册发现	Eureka, Consul, <b>Nacos</b> , zk
Routing	服务网关路由	Zuul, <b>SpringCloud gateWay</b>
Service Call	服务调用	<b>RestTemplate</b> , Ribbon, Feign
Loading blance	客户端负载均衡	<b>Ribbon</b>
Circuit Breakers	断路器	Hystrix, <b>Sentinel</b>
Distributed Messageing	分布式消息	<b>SpringCloud Stream</b> +kafka/Rabbitmq/RockMq



## 2.2)什么是Spring cloud Alibaba

Spring cloud Alibaba是我们SpringCloud的一个子项目,是提供微服务开发的一站式解决方案,包含微服务开发的必要组件。

2.2.1)基于SpringCloud 符合SpringCloud标准,是阿里的微服务的解决方案。

文档:<https://github.com/alibaba/spring-cloud-alibaba/blob/master/README-zh.md>

主要功能描述:

功能	产品	DESC
服务限流降级	Sentinel	开源
服务注册发现	Nocas	开源
分布式配置中心	Nocas	开源
消息驱动	Springcloud Stream+rocketMQ	开源
分布式事务	Seata	1.0.0后能用于生产
OSS	云存储	收费
SMS SchedulerX	短信 分布式调度中心	收费

## 2.2.2)SpringCloud SpringCloudalibaba SpringBoot的生产版本选择

### ①:我们的SpringBoot版本 说明选择

```

1  /**
2  其中2: 表示的主版本号, 表示是我们的SpringBoot第二代产品
3  其中1:表示的是次版本号, 增加了一些新的功能但是主体的架构是没有变化的, 是兼容的
4  其中6:表示的是bug修复版
5  所以2.1.6合起来就是springboot的第二代版本的第一个小版本的 第6次bug修复版本
6  RELEASE: 存在哪些取值了 ①:snapshot(开发版本) ②:M1...M2(里程碑版本, 在
7  正式版发布之前 会出几个里程碑的版本) ③:release(正式版本)
8  **/
9  <parent>
10 <groupId>org.springframework.boot</groupId>
11 <artifactId>spring-boot-starter-parent</artifactId>
12 <version>2.1.6.RELEASE</version>
13 </parent>

```

## ②:Spring cloud的版本说明

第一代版本:Angle

第二代版本:Brixton

第三代版本:Camden

第四代版本:Edgware

第五代版本:Finchley

第六代版本:GreenWich

第七代版本:Hoxton(还在酝酿中, 没正式版本)

**这种发布的版本是 以伦敦地铁站发行地铁的站。**

**为什么我们的SpringCloud会以这种方式来发布版本,因为假如我们传统的**

**5.1.5release这种发布的而 SpringCloud会包含很多子项目的版本就会给人造成混淆。**

Table 2. Release train contents			
Component	Edgware.SR6	Greenwich.SR2	Greenwich.BUILD-SNAPSHOT
spring-cloud-aws	1.2.4.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-bus	1.3.4.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-cli	1.4.1.RELEASE	2.0.0.RELEASE	2.0.1.BUILD-SNAPSHOT
spring-cloud-commons	1.3.6.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
			2.1.3.BUILD-

**SNAPSHOT:** 快照版本, 随时可能修改

Hoxton **SNAPSHOT**

Greenwich **SNAPSHOT**

**M:** MileStone, M1表示第1个里程碑版本, 一般同时标注PRE, 表示预览版版。

**RC** 版本英文版名字叫Release Candidate (候选版本) 一般标注PRE表示预览版

Hoxton RC2 **PRE**

**SR:** Service Release, SR1表示第1个正式版本, 一般同时标注GA: (GenerallyAvailable),表示稳定版本。

Greenwich SR3 **CURRENT** **GA**

**比如还有一种RELEASE版本 (正式版本) 比如 Greenwich版本顺序**

**Greenwich.release----->发现bug----->Greenwich.SR1----->发现bug---->Greenwich.SR2。**

**SpringCloud的发布计划**

<https://github.com/spring-cloud/spring-cloud-release/milestones>

**SpringCloud曾经发布的版本:**

<https://github.com/spring-cloud/spring-cloud-release/releases>

### ③:Springboot SpringCloud SpringCloudalibaba 的版本对应关系

<https://github.com/alibaba/spring-cloud-alibaba/wiki/%E7%89%88%E6%9C%AC%E8%AF%B4%E6%98%8E>

#### 毕业版本依赖关系(推荐使用)

Spring Cloud Version	Spring Cloud Alibaba Version	Spring Boot Version
Spring Cloud Greenwich	2.1.0.RELEASE	2.1.X.RELEASE
Spring Cloud Finchley	2.0.0.RELEASE	2.0.X.RELEASE
Spring Cloud Edgware	1.5.0.RELEASE	1.5.X.RELEASE

#### ④：生产版本选择

a:打死不用 非稳定版本/ end-of-life (不维护) 版本

b:release版本先等等(等别人去探雷)

c:推荐 SR2以后的可以放心使用.

### 三:微服务注册中心Nacos入门

<https://nacos.io/zh-cn/docs/what-is-nacos.html>

名词	定义
服务提供者	服务的被调用方 (即：为其他服务提供服务的服务)
服务消费者	服务的调用方 (即：依赖其他服务的服务)

服务的提供者 &服务的消费者是相对的概念

比如**用户服务**是**订单服务**的消费者，**订单服务**是**用户服务**的提供者。

但是对于 **订单服务**---->**库存服务**，那么订单服务就成为服务消费者。



#### 3.1) 无注册中心的调用的缺点。

比如现在我的用户服务是占用(User服务)8081端口的服务, 此时我的服务提供方 (order服务端口是8080)端口

我们可以通过RestTemplate 调用方式来进行调用

```
1 ResponseEntity<ProductInfo> responseEntity=  
2 restTemplate.getForEntity("http://localhost:8081/selectProductInfoById/"+  
3 orderInfo.getProductNo(), ProductInfo.class);
```

#### 缺点:

1)从上面看出的缺点就是，我们的在调用的时候，请求的Ip地址和端口是硬编码的。



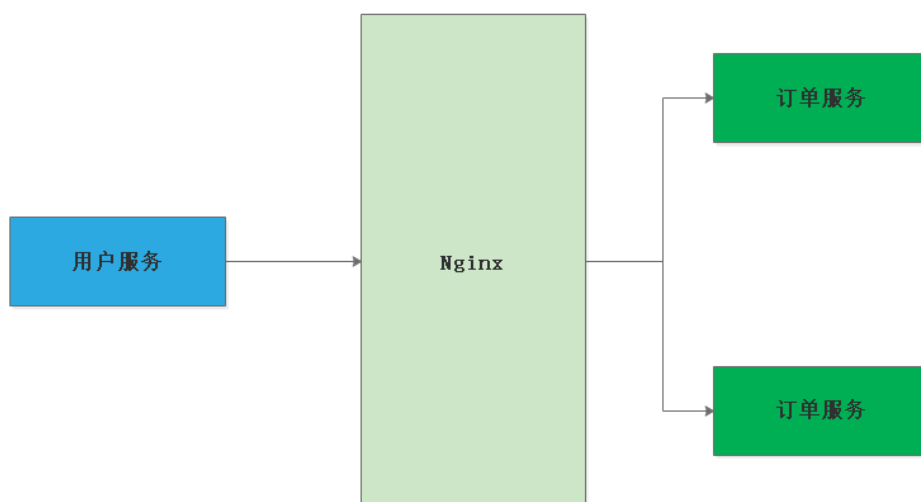
若此时，服务提供方(order)服务部署的机器换了端口或者是更换了部署机器的Ip,那么我们需要修改代码重新发布部署.

2) 假设我们的order服务压力过大，我们需要把order服务作为集群，那么意味着order是多节点部署

比如原来的，我们只有一台服务器，现在有多台服务器，那么作为运维人员 需要在服务消费方进行手工维护一份注册表(容易出错)

3)有人马上回驳我说，我可以通过ng来做负载均衡,对，我首先认为这是可行的，当时微服务成百上千的服务，难道我们要那成百上千

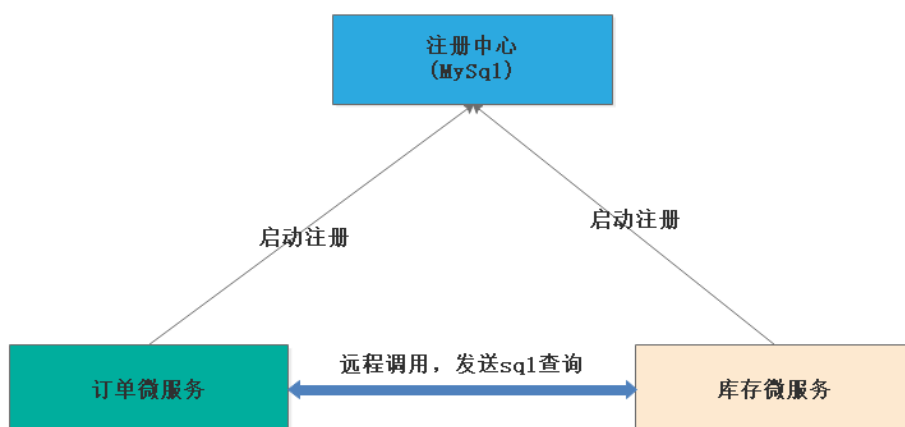
ng么？或者使用一个Ng 那么我们能想一下哪个ng的配置文件有多么复杂。



### 3.2) 大话 服务注册发现原理

V1架构图：

Register table(注册表)				
id	service_name	ip	port	status
1	order-center	192. 168. 1. 1	8080	up
2	user-center	192. 168. 1. 2	8081	up



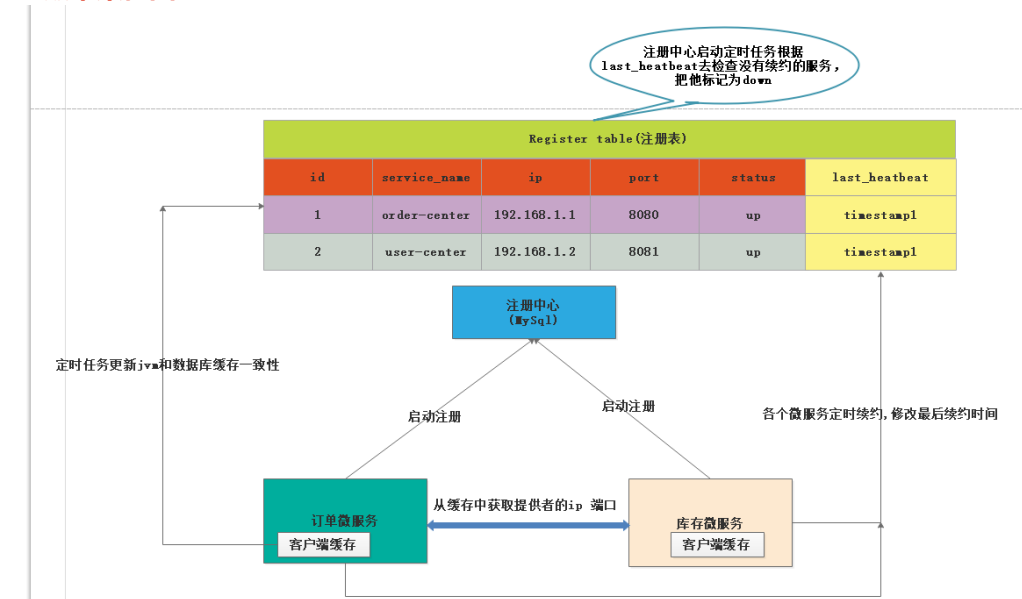
#### 3.2.1) V1版本的架构，存在以下几个问题

①:我们的微服务每次调用，都会去进行对数据库的查询，并发一高，数据库性能就是一个瓶颈问题.

②:若我们的mysql挂了，那么我们所有的微服务调用都不能正常进行。

③:若mysql是正常的,库存微服务挂了, 那么也不能正常的调用

## V2版本架构图



## 3.3)Nacos服务端搭建

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

Assets 4		
nacos-server-1.1.4.tar.gz	linux版本	49.7 MB
nacos-server-1.1.4.zip	windows版本	49.7 MB
Source code (zip)		
Source code (tar.gz)		

### 3.3.1)linux环境启停:

①:把我们的Nacos包解压 `tar -zxvf nacos-server-1.1.4.tar.gz`

```
drwxr-xr-x. 7 root root      89 Nov 18 01:06 nacos
-rw-r--r--. 1 root root 52115827 Nov 18 01:00 nacos-server-1.1.4.tar.gz
[root@smlz nacos]#
```

②: `cd` 到我们的解压目录nacos `cd nacos`

```
drwxr-xr-x. 4 root root  4096 Nov 18 01:06 bin
drwxr-xr-x. 2 502 games 4096 Nov  3 18:26 conf
drwxr-xr-x. 4 root root   36 Nov 18 01:06 data
-rw-r--r--. 1 502 games 17336 Oct 10 23:09 LICENSE
drwxr-xr-x. 2 root root  4096 Nov 18 01:06 logs
-rw-r--r--. 1 502 games  1305 Oct 10 23:09 NOTICE
drwxr-xr-x. 2 root root   29 Nov 18 01:01 target
```

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

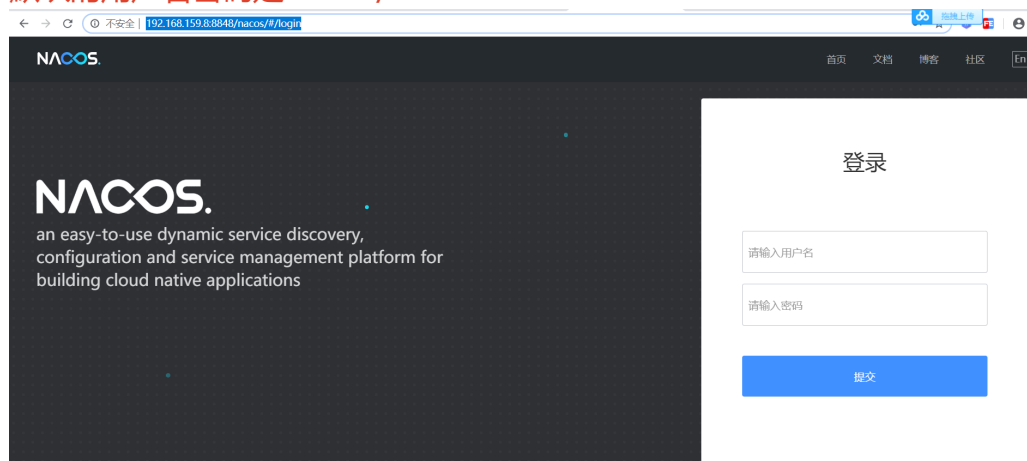
```
[root@smlz bin]# sh startup.sh -m standalone
/usr/local/jdk/jdk1.8.0_221/bin/java -Xms512m -Xmn256m -Dnacos.standalone=true -Djava.ext.dirs=/usr/local/jdk/jdk1.8.0_221/jre/lib/ext:/usr/local/jdk/jdk1.8.0_221/lib/ext:/usr/local/spring-cloud-alibaba/nacos/nacos/plugins/cmds:/usr/local/spring-cloud-alibaba/nacos/nacos/plugins/mysql -Xloggc:/usr/local/spring-cloud-alibaba/nacos/nacos/logs/nacos_gc.log -verbose:gc -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M -Dnacos.home=/usr/local/spring-cloud-alibaba/nacos/nacos -Dloader.path=/usr/local/spring-cloud-alibaba/nacos/nacos/plugins/health -jar /usr/local/spring-cloud-alibaba/nacos/nacos/target/nacos-server.jar --spring.config.location=classpath:/,classpath:/config/,file:./,file:/config/,file:/usr/local/spring-cloud-alibaba/nacos/nacos/conf/ --logging.config=/usr/local/spring-cloud-alibaba/nacos/nacos/conf/nacos-logback.xml --server.max-http-header-size=524288
nacos is starting with standalone
nacos is starting, you can check the /usr/local/spring-cloud-alibaba/nacos/nacos/logs/start.out
```

④:检查nacos启动的端口 `lsof -i:8848`

```
[root@smlz bin]# lsof -i:8848
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
java    69048 root  104u IPv6 2101510 0t0 TCP *:8848 (LISTEN)
java    69048 root  112u IPv6 2101515 0t0 TCP smlz:8848->smlz:49181 (ESTABLISHED)
java    69048 root  113u IPv6 2101520 0t0 TCP smlz:49181->smlz:8848 (ESTABLISHED)
```

⑤:访问nocas的服务端 <http://192.168.159.8:8848/nacos/index.html>

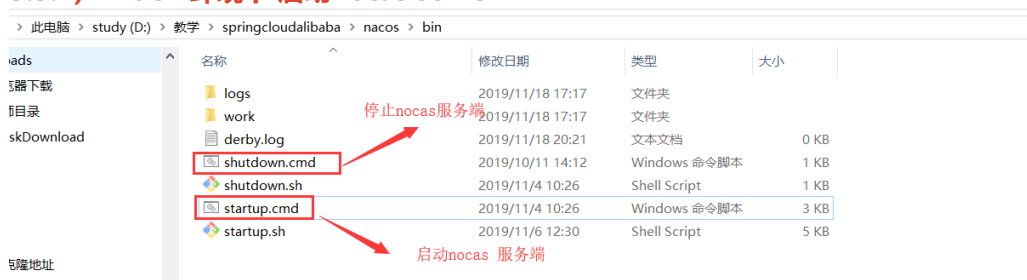
默认的用户名密码是 nocas/nocas



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

```
[root@smlz bin]# sh shutdown.sh
The nacosServer(69048) is running...
Send shutdown request to nacosServer(69048)
[root@smlz bin]#
```

### 3.3.2)window环境下 启动nocas server



## 4: Nacos client服务端的搭建

①:三板斧之:第一板斧 加入依赖

```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
4 </dependency>
```

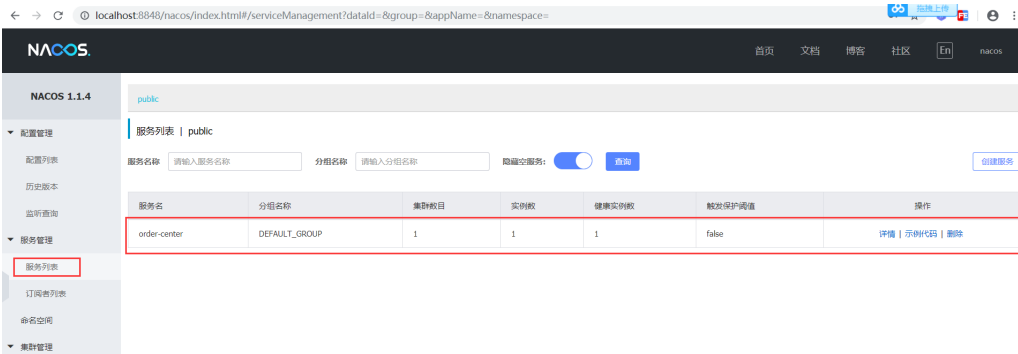
②:三板斧之:第二板斧写注解(也可以不写) **@EnableDiscoveryClient**

```
1 @SpringBootApplication
2 @EnableDiscoveryClient
3 public class Tulingvip01MsAlibabaNacosClientOrderApplication {
4
5     public static void main(String[] args) {
6         SpringApplication.run(Tulingvip01MsAlibabaNacosClientOrderApplication.class, args);
7     }
8 }
```

③:第三板斧之:写配置文件 **\*\*注意\*\*server-addr: 不需要写协议**

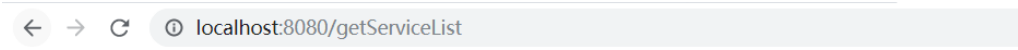
```
1 spring:
```

```
2 cloud:
3 nacos:
4 discovery:
5 server-addr: localhost:8848
6 application:
7 name: order-center
```



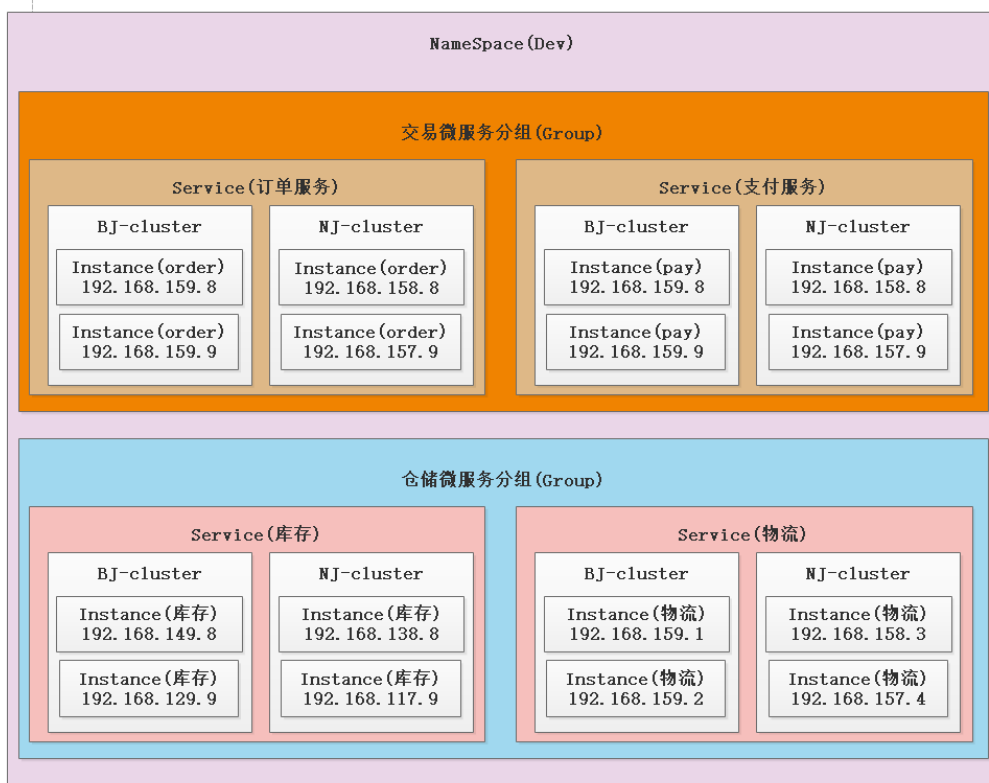
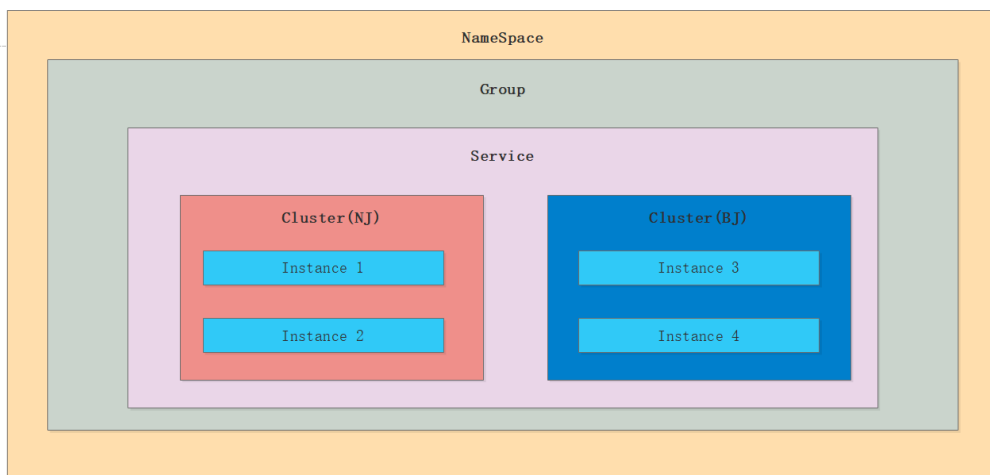
#### ④:验证我们的order-center注册到我们的nacos上

```
1 @Autowired
2 private DiscoveryClient discoveryClient;
3
4 @GetMapping("/getServiceList")
5 public List<ServiceInstance> getServiceList() {
6     List<ServiceInstance> serviceInstanceList = discoveryClient.getInstances("order-center");
7     return serviceInstanceList;
8 }
```



```
[
  {
    "serviceId": "order-center",
    "host": "192.168.0.224",
    "port": 8080,
    "secure": false,
    "metadata": {
      "nacos.instanceId": "192.168.0.224#8080#DEFAULT#DEFAULT_GROUP@@order-center",
      "nacos.weight": "1.0",
      "nacos.cluster": "DEFAULT",
      "nacos.healthy": "true",
      "preserved.register.source": "SPRING_CLOUD"
    },
    "uri": "http://192.168.0.224:8080",
    "scheme": null
  }
]
```

## 5: Nacos 领域模型划分以及概念详解



5.1)NameSpace(默认的名称空间是” 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的名称空间下的微服务)

NACOS				
NACOS 1.1.4				
命名空间				
<div> <div>配置管理</div> <div>配置列表</div> <div>历史版本</div> <div>当前查询</div> <div>服务管理</div> <div>服务列表</div> <div>订阅列表</div> <div>命名空间</div> <div>集群管理</div> <div>节点列表</div> </div>				
命名空间名称	命名空间ID	配置数	操作	
public(保留空间)		0 / 200	详情 删除 编辑	
prod	20989a73-cfb3-41b8-85c0-e9a3530e28a6	0 / 200	详情 删除 编辑	
test	51ead8fe-d402-4c24-a204-355f9b6b701a	0 / 200	详情 删除 编辑	
dev	bc7613d2-2e22-4292-a748-48b78170f14c	0 / 200	详情 删除 编辑	

证明1)我们dev环境下的order-center 调用 prod环境下的product-center

### ①:order-center所在的namespace为dev

```
1 spring:
2   cloud:
3     nacos:
4       discovery:
5         server-addr: localhost:8848
6       #dev环境的
7       namespace: bc7613d2-2e22-4292-a748-48b78170f14c #指定namespace的id
8       application:
9         name: order-center
```

### ②:product-center所在的namespace 为prod

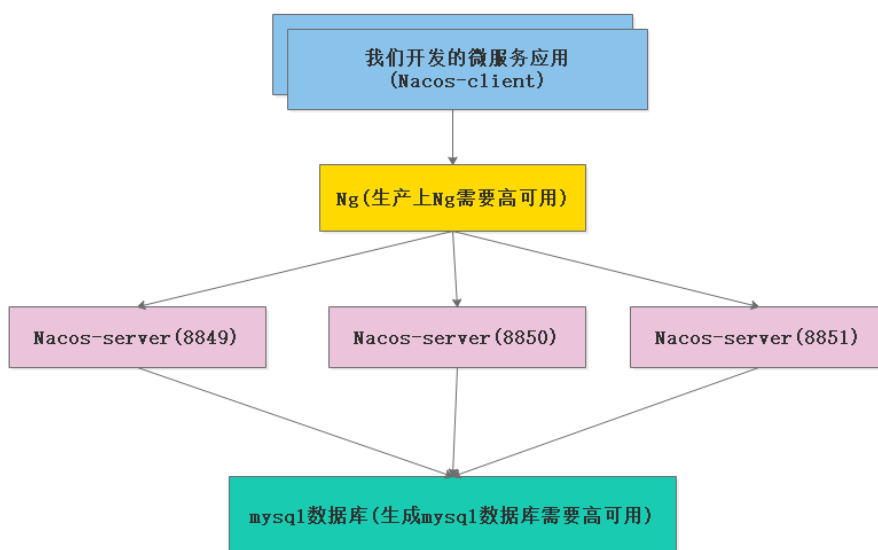
```
1 spring:
2   application:
3     name: product-center
4   cloud:
5     nacos:
6       discovery:
7         server-addr: localhost:8848
8       #prod环境的
9       namespace: 20989a73-cdb3-41b8-85c0-e9a3530e28a6
```

### ③: 测试调用: <http://localhost:8080/selectOrderInfoById/1>

← → ↻ ⓘ localhost:8080/selectOrderInfoById/1

用户微服务没有对应的实例可用

## 5.2) Nacos的集群模式



### 5.2.1)首先 我们需要安装我们的ng

<http://nginx.org/download/nginx-1.14.2.tar.gz>

第一步:下载ng (老师下载的是:路径是/usr/local/software)

wget <http://nginx.org/download/nginx-1.14.2.tar.gz>

```
total 286636
-rw-r--r--. 1 root root 18345424 Apr  5 2016 erlang-18.3-1.el7.centos.x86_64.rpm
-rw-r--r--. 1 root root 195094741 Aug 27 00:35 jdk-8u221-linux-x64.tar.gz
-rw-r--r--. 1 root root 73214518 Dec 19 2018 mongodb-linux-x86_64-4.0.5.tgz
-rw-r--r--. 1 root root 25680 Apr 27 2017 mysql57-community-release-el7-11.noarch.rpm
drwxr-xr-x. 9 1001 1001 4096 Jan 31 23:53 nginx-1.14.2
-rw-r--r--. 1 root root 1015384 Dec  4 2018 nginx-1.14.2.tar.gz
-rw-r--r--. 1 root root 5520417 Aug  5 2016 rabbitmq-server-3.6.5-1.noarch.rpm
-rw-r--r--. 1 root root 284676 Jun 23 2017 socat-1.7.3.2-5.el7.linux.x86_64.rpm
```

第二步:解压ng

tar -xzf nginx-1.14.2.tar.gz 得到解压目录

```
total 286636
-rw-r--r--. 1 root root 18345424 Apr  5 2016 erlang-18.3-1.el7.centos.x86_64.rpm
-rw-r--r--. 1 root root 195094741 Aug 27 00:35 jdk-8u221-linux-x64.tar.gz
-rw-r--r--. 1 root root 73214518 Dec 19 2018 mongodb-linux-x86_64-4.0.5.tgz
-rw-r--r--. 1 root root 25680 Apr 27 2017 mysql57-community-release-el7-11.noarch.rpm
drwxr-xr-x. 9 1001 1001 4096 Jan 31 23:53 nginx-1.14.2
-rw-r--r--. 1 root root 1015384 Dec  4 2018 nginx-1.14.2.tar.gz
-rw-r--r--. 1 root root 5520417 Aug  5 2016 rabbitmq-server-3.6.5-1.noarch.rpm
-rw-r--r--. 1 root root 284676 Jun 23 2017 socat-1.7.3.2-5.el7.linux.x86_64.rpm
```

第三步: 进入解压目录

cd nginx-1.14.2

```
[root@smlz software]# cd nginx-1.14.2/
[root@smlz nginx-1.14.2]# ll
total 744
drwxr-xr-x. 6 1001 1001 4096 Jan 31 23:52 auto
-rw-r--r--. 1 1001 1001 288742 Dec  4 2018 CHANGES
-rw-r--r--. 1 1001 1001 440121 Dec  4 2018 CHANGES.ru
drwxr-xr-x. 2 1001 1001 4096 Jan 31 23:52 conf
-rwxr-xr-x. 1 1001 1001 2502 Dec  4 2018 configure
drwxr-xr-x. 4 1001 1001 68 Jan 31 23:52 contrib
drwxr-xr-x. 2 1001 1001 38 Jan 31 23:52 html
-rw-r--r--. 1 1001 1001 1397 Dec  4 2018 LICENSE
-rw-r--r--. 1 root root 404 Jan 31 23:53 Makefile
drwxr-xr-x. 2 1001 1001 20 Jan 31 23:52 man
drwxr-xr-x. 3 root root 4096 Jan 31 23:54 objs
-rw-r--r--. 1 1001 1001 49 Dec  4 2018 README
drwxr-xr-x. 9 1001 1001 84 Jan 31 23:52 src
```

第四步: 执行命令指定安装目录

./configure --prefix=/usr/local/nginx-1.14.2 意思是告诉等会安装的文件要放在哪里

第五步: 接下来通过命令 make 编译

执行 make 命令

第六步:执行 make install 命令

```
[root@smlz nginx-1.14.2]# make install
```

第七步:还记得第四步我们安装ng的目录么? /usr/local/nginx-1.14.2

进入该目录下的配置目录conf

```
drwx-----. 2 nobody root      6 Feb  1 00:02 client_body_temp
drwxr-xr-x.  2 root   root    4096 Jan 31 23:55 conf
drwx-----. 2 nobody root      6 Feb  1 00:02 fastcgi_temp
drwxr-xr-x.  2 root   root     38 Jan 31 23:55 html
drwxr-xr-x.  2 root   root     55 Feb  1 00:02 logs
drwx-----. 10 nobody root     70 Feb  1 00:19 proxy_temp
drwxr-xr-x.  2 root   root     18 Jan 31 23:55 sbin
drwx-----. 2 nobody root      6 Feb  1 00:02 scgi_temp
drwx-----. 2 nobody root      6 Feb  1 00:02 uwsgi_temp
[root@smlz nginx-1.14.2]#
```

## 第八步:修改ng的conf文件

```
total 68
-rw-r--r--. 1 root root 1077 Jan 31 23:55 fastcgi.conf
-rw-r--r--. 1 root root 1077 Jan 31 23:55 fastcgi.conf.default
-rw-r--r--. 1 root root 1007 Jan 31 23:55 fastcgi_params
-rw-r--r--. 1 root root 1007 Jan 31 23:55 fastcgi_params.default
-rw-r--r--. 1 root root 2837 Jan 31 23:55 koi-utf
-rw-r--r--. 1 root root 2223 Jan 31 23:55 koi-win
-rw-r--r--. 1 root root 5170 Jan 31 23:55 mime.types
-rw-r--r--. 1 root root 5170 Jan 31 23:55 mime.types.default
-rw-r--r--. 1 root root 2732 Feb  1 00:18 nginx.conf
-rw-r--r--. 1 root root 2656 Jan 31 23:55 nginx.conf.default
-rw-r--r--. 1 root root 636 Jan 31 23:55 scgi_params
-rw-r--r--. 1 root root 636 Jan 31 23:55 scgi_params.default
-rw-r--r--. 1 root root 664 Jan 31 23:55 uwsgi_params
-rw-r--r--. 1 root root 664 Jan 31 23:55 uwsgi_params.default
-rw-r--r--. 1 root root 3610 Jan 31 23:55 win-utf
```

```
1 修改的内容为:
2  upstream nacos_cluster {
3     server 192.168.159.8:8849;
4     server 192.168.159.8:8850;
5     server 192.168.159.8:8851;
6  }
7  server {
8     listen 8847;
9     server_name localhost;
10
11    #charset koi8-r;
12
13    #access_log logs/host.access.log main;
14
15    location /nacos/ {
16        proxy_pass http://nacos_cluster/nacos/;
17    }
18  }
19  ng配置到此完成。
```

## 5.2.2)安装 我们的nacos-server(搭建三个集群端口分别为8849,8850,8851)

```
[root@smlz nacos]# ll
total 50896
drwxr-xr-x. 7 root root      89 Nov 18 01:06 nacos
drwxr-xr-x. 8 root root    102 Jan 31 22:31 nacos8849
drwxr-xr-x. 8 root root    102 Jan 31 22:31 nacos8850
drwxr-xr-x. 8 root root    102 Jan 31 22:31 nacos8851
-rw-r--r--. 1 root root 52115827 Nov 18 01:00 nacos-server-1.1.4.tar.gz
[root@smlz nacos]#
```

## 我们以配置一台为例 (8849) 为例

### 第一步:修改nacos8849/conf文件 application.properties

```
1 spring.datasource.platform=mysql
2
3 # 数据库实例数量
```



```

4 db.num=1
5 //自己数据库的连接信息
6 db.url.0=jdbc:mysql://192.168.159.8:3306/nacos_test?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true
7 db.user=root
8 db.password=Zw726515@

```

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

cluster.conf文件

```

# nacos.naming.expireInstance=true

nacos.istio.mcp.server.enabled=false
[root@smlz conf]# ll
total 56
-rw-r--r--. 1 502 games 1799 Jan 31 22:41 application.properties
-rw-r--r--. 1 502 games 408 Oct 10 23:12 application.properties.example
-rw-r--r--. 1 root root 119 Jan 31 22:26 cluster.conf
-rw-r--r--. 1 502 games 58 Oct 10 23:09 cluster.conf.example
-rw-r--r--. 1 502 games 20210 Nov 3 18:26 nacos-logback.xml
-rw-r--r--. 1 502 games 9788 Oct 10 23:12 nacos-mysql.sql
-rw-r--r--. 1 502 games 7196 Oct 10 23:12 schema.sql
[root@smlz conf]#

```

文件内容为如下

```

192.168.159.8:8849
192.168.159.8:8850
192.168.159.8:8851
[root@smlz conf]#

```

到此为止 nacos8849安装完成了 nacos8850 nacos8851同样安装.

5.2.3)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/nacos-mysql.sql

5.2.4)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)

```

[root@smlz bin]# pwd
/usr/local/spring-cloud-alibaba/nacos/nacos8849/bin
[root@smlz bin]# ll
total 108
-rw-r--r--. 1 root root 792 Jan 21 17:52 derby.log
-rw-r--r--. 1 root root 16489 Jan 31 22:09 hs_err_pid27804.log
-rw-r--r--. 1 root root 16489 Jan 31 22:11 hs_err_pid27887.log
-rw-r--r--. 1 root root 16489 Jan 31 22:26 hs_err_pid28295.log
-rw-r--r--. 1 root root 16539 Jan 31 22:41 hs_err_pid29055.log
drwxr-xr-x. 2 root root 6 Jan 31 22:41 logPath_IS_UNDEFINED
drwxr-xr-x. 2 root root 4096 Feb 1 00:00 logs
-rwxr-xr-x. 1 502 games 954 Oct 10 23:12 shutdown.cmd
-rwxr-xr-x. 1 502 games 949 Nov 3 18:26 shutdown.sh
-rwxr-xr-x. 1 502 games 2854 Nov 3 18:26 startup.cmd
-rwxr-xr-x. 1 502 games 4807 Jan 31 23:03 startup.sh
drwxr-xr-x. 3 root root 19 Nov 18 01:06 work
[root@smlz bin]#

```

```

=====
# JVM Configuration
=====
if [[ "${MODE}" == "standalone" ]]; then
    JAVA_OPT="{JAVA_OPT} -Xms512m -Xmx512m -Xmn256m"
    JAVA_OPT="{JAVA_OPT} -Dnacos.standalone=true"
else
    JAVA_OPT="{JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
    JAVA_OPT="{JAVA_OPT} -XX:-OmitStackTraceInFastThrow -XX:-HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=${BASE_DIR}/logs/ja
mp.hprof"
    JAVA_OPT="{JAVA_OPT} -XX:-UseLargePages"
fi

if [[ "${FUNCTION_MODE}" == "config" ]]; then

```

5.2.4)分别启动 8849 8850 8851

进入到 nacos的目录下的bin目录

```
[root@smlz bin]# ll
total 88
-rw-r--r--. 1 root root    792 Jan 21 17:52 derby.log
-rw-r--r--. 1 root root  16489 Jan 31 22:09 hs_err_pid27804.log
-rw-r--r--. 1 root root  16489 Jan 31 22:11 hs_err_pid27887.log
-rw-r--r--. 1 root root  16489 Jan 31 22:26 hs_err_pid28295.log
drwxr-xr-x. 2 root root   4096 Feb  1 01:11 logs
-rwxr-xr-x. 1 502 games   954 Oct 10 23:12 shutdown.cmd
-rwxr-xr-x. 1 502 games   949 Nov  3 18:26 shutdown.sh
-rwxr-xr-x. 1 502 games  2854 Nov  3 18:26 startup.cmd
-rwxr-xr-x. 1 502 games  4807 Jan 31 23:01 startup.sh
drwxr-xr-x. 3 root root    19 Nov 18 01:06 work
[root@smlz bin]#
```

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

```
2020-02-01 01:11:48,792 INFO Nacos is starting...
2020-02-01 01:11:49,803 INFO Nacos is starting...
2020-02-01 01:11:50,814 INFO Nacos is starting...
2020-02-01 01:11:51,823 INFO Nacos is starting...
2020-02-01 01:11:52,830 INFO Nacos is starting...
2020-02-01 01:11:53,858 INFO Nacos is starting...
2020-02-01 01:11:54,869 INFO Nacos is starting...
2020-02-01 01:11:55,622 INFO Nacos Log files: /usr/local/spring-cloud-alibaba/nacos/nacos8850/logs/
2020-02-01 01:11:55,622 INFO Nacos Conf files: /usr/local/spring-cloud-alibaba/nacos/nacos8850/conf/
2020-02-01 01:11:55,622 INFO Nacos Data files: /usr/local/spring-cloud-alibaba/nacos/nacos8850/data/
2020-02-01 01:11:55,622 INFO Nacos started successfully in cluster mode.
```

测试:

分别登陆地址:

<http://192.168.159.8:8849/nacos>

<http://192.168.159.8:8850/nacos>

<http://192.168.159.8:8851/nacos>

NG测试

<http://192.168.159.8:8847/nacos/>

← → 不安全 | 192.168.159.8:8849/nacos/#/clusterManagement?dataId=&group=&appName=&namespace=

NACOS 1.1.4 public 首页 文档 博客 社区 [En]

配置管理 节点列表 | public

配置列表 节点Ip 请输入节点Ip 查询

节点Ip	节点状态	集群任期	Leader任期(ms)	心跳止时(ms)
192.168.159.8:8849	LEADER	12	15867	3500
192.168.159.8:8851	FOLLOWER	12	13238	3000
192.168.159.8:8850	FOLLOWER	0	16020	3000

服务列表 | public

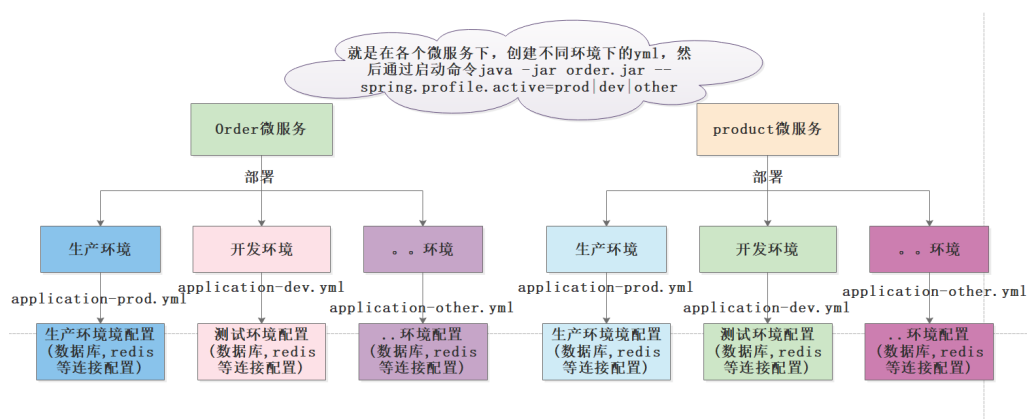
服务名称 请输入服务名称 分组名称 请输入分组名称 隐藏空服务: ☐ 查询 创建服务

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值	操作
order-center	DEFAULT_GROUP	1	1	1	false	详情   示例代码   删除

客户端地址: 直接填写8847ng的地址就可以了

```
connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500
cloud:
  nacos:
    discovery:
      server-addr: 192.168.159.8:8847
      #namespace: bc7613d2-2e22-4292-a748-48b78170f14c #指定namespace的id
  application:
```

## 6.什么是配置管理??



### 上图缺点:

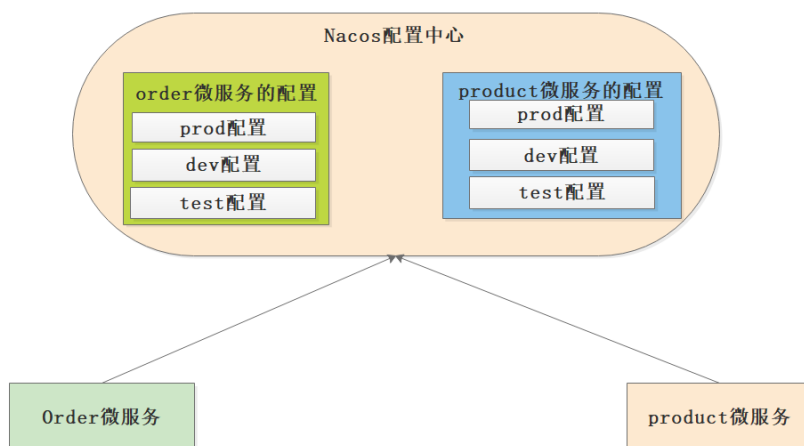
所有的环境的配置都是明文的 被太多开发人员知道了。

业务场景: 张三开发了一个新功能, 业务需要, 保留原来老逻辑的代码, 所有他抽取了一个开关变量 `isNewBusi` 来控制, 突然新功能上了生产后, 发现有bug, 怎么做到修改 `isNewBusi` 的值不需要重启。

根据上图我们知道配置管理的作用可以主要总结如下

- 1) 不同环境不管配置
- 2) 配置属性动态刷新

## 引入配置中心



根据这幅图，我们微服务需要解决的问题

- 1)我微服务怎么知道配置中心的地址
- 2) 我微服务到底需要连接哪个环境
- 3)怎么找到nacos config上的对于的配置文件

## 微服务接入配置中心的步骤

### ①:添加依赖包spring-cloud-alibaba-nacos-config

```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-alibaba-nacos-config</artifactId>
4 </dependency>
```

### ②:编写配置文件,需要写一个bootstrap.yml配置文件

#### 配置解释:

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**

```
1 spring:
2   cloud:
3     nacos:
4       config:
5         server-addr: localhost:8848
6         file-extension: yaml
```

```
7 application:
8 name: order-center
9 profiles:
10 active: prod
```

#### 编辑配置

\* Data ID:

\* Group:

[更多高级选项](#)

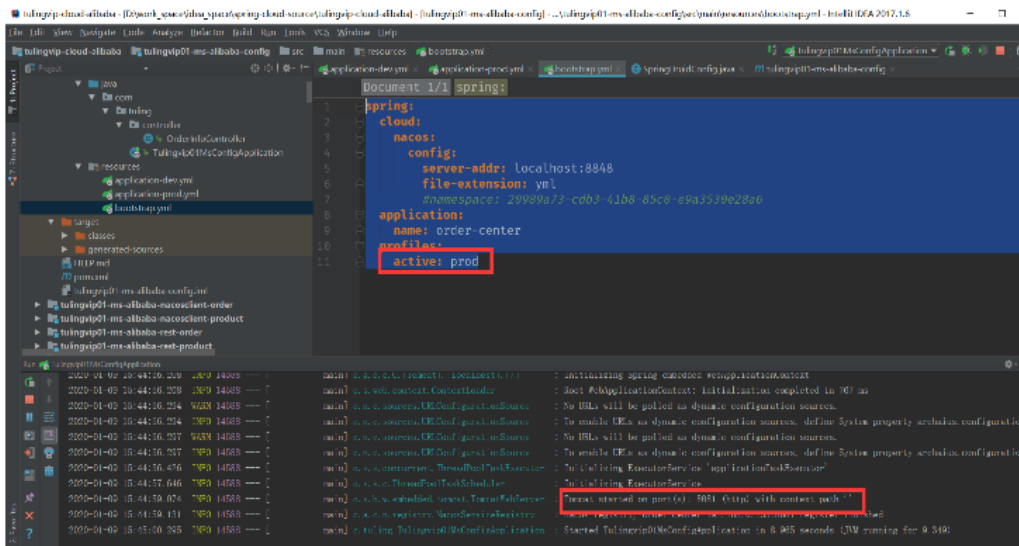
描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容: 

```
isNewBusi: true
server:
  port: 8081
```



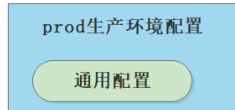
← → ↺ ⓘ localhost:8081/selectOrderInfoById/1

查询订单执行新逻辑->execute new busi : 1

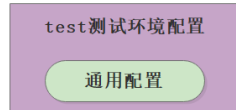
现在我们需要不停机改变我们的生产环境isNewBusi的值来控制我们的业务逻辑。我们需要在对应的Controller上添加一@RefreshScope 进行动态刷新

## 6.2) 怎么解决 生产环境，测试环境，开发环境相同的配置。（配置通用）

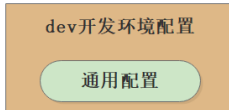
order-center-prod.yml



order-center-test.yml



order-center-dev.yml



比如我们的servlet-context 为order-center

查看工程启动日志

```
org.springframework.boot.SpringApplication
Loading nacos data, dataId: 'order-center-prod.yml', group: 'DEFAULT_GROUP'
Located property source: CompositePropertySource {name='NACOS', propertySources=[NacosPropertySource {name='order-center-prod.yml'}, NacosPropertySource {name='order-center.yml'}]}
The following profiles are active: prod
BeanFactory id:20849224-031f-379f-9013-993ce7b3847
Bean 'org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration' of type [org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration]
Tomcat initialized with port(s): 8081 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/9.0.16]
The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: [C:\Program Files\Java\jdk1.8.0_131\bin.C:\WINDOWS\system32\bin.C:\WINDOWS\system32\bin]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 200 ms
```

所以我们需要创建一个通用配置文件:order-center.yml配置

那么order-center.yml就是一个通用配置了，不管是启动prod,还是dev

都会有该段配置order-server的 context-path 配置

## 新建配置

\* Data ID: order-center.yml

\* Group: DEFAULT\_GROUP

[更多高级选项](#)

描述: order各个环境的通用配置

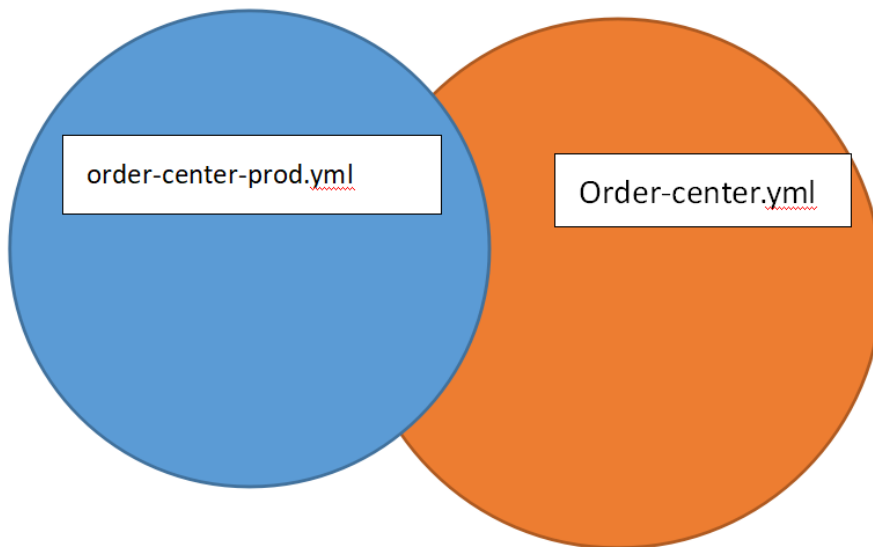
配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

\* 配置内容:

?:

```
1 server:
2   servlet:
3     context-path: /order-server
```

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。



### 6.3)不同微服务的通用配置。



#### 6.3.1)通过 shared-dataids 方式

比如 每一个微服务都要写服务注册地址等

```
1 spring:
2   cloud:
3     nacos:
4       discovery:
5         server-addr: localhost:8848
```

我们就可以把该配置提取为一个公共配置yml,提供给所有的工程使用

```
1 spring:
2   cloud:
3     nacos:
4       config:
5         server-addr: localhost:8848
6         file-extension: yaml
7         #各个微服务共享的配置,注意越拍到后面的公共配置yml优先级越高
8         shared-dataids: common.yml,common2.yml
9         #支持动态刷新的配置文件
10        refreshable-dataids: common.yml,common2.yml
11       application:
12         name: order-center
```

```
13 profiles:
14 active: dev
```

配置优先级

2020-01-09 21:21:19.115 INFO 15316 --- [ main] b.c.PropertySourceBootstrapConfiguration :

Located property source:

```
CompositePropertySource {name='NACOS',
propertySources=[
NacosPropertySource {name='order-center-dev.yml'},
NacosPropertySource {name='order-center.yml'},
NacosPropertySource {name='common2.yml'},
NacosPropertySource {name='common.yml'}
]
}
```

### 6.3.1)通过 ext-config方式

同样配置到越后面的配置 优先级越高

```
1 spring:
2   cloud:
3     nacos:
4     config:
5       server-addr: localhost:8848
6       file-extension: yaml
7       ext-config:
8         - data-id: common3.yml
9           group: DEFAULT_GROUP
10          refresh: true
11         - data-id: common4.yml
12           group: DEFAULT_GROUP
13          refresh: true
```

### 6.3.3)各个配置的优先级

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)

```
1 spring:
2   cloud:
3     nacos:
4     config:
5       server-addr: localhost:8848
6       file-extension: yaml
7       shared-dataids: common.yml,common2.yml
8       refreshable-dataids: common.yml,common2.yml
9       ext-config:
10         - data-id: common3.yml
11           group: DEFAULT_GROUP
12          refresh: true
13         - data-id: common4.yml
```



```
14 group: DEFAULT_GROUP
15 refresh: true
16
17 application:
18   name: order-center
19   profiles:
20     active: dev
```

### 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

```
http://localhost:8888/oauth/authorize?
response_type=code&client_id=portal_app&redirect_uri=http://www.portal.com:8855/callback&state=123'
```

```
http://localhost:8888/oauth/token?
grant_type=authorization_code
&client_id=portal_app
&client_secret=portal_app
&redirect_uri=http://www.baidu.com
&code=5ViNua
&scope=read
```

```
http://localhost:9060/oauth/authorize?
response_type=code&client_id=webApp&redirect_uri=http://www.baidu.com&state=123'
```