

# 一:微服务 & 微服务架构

## 1: 单体架构 VS 微服务架构

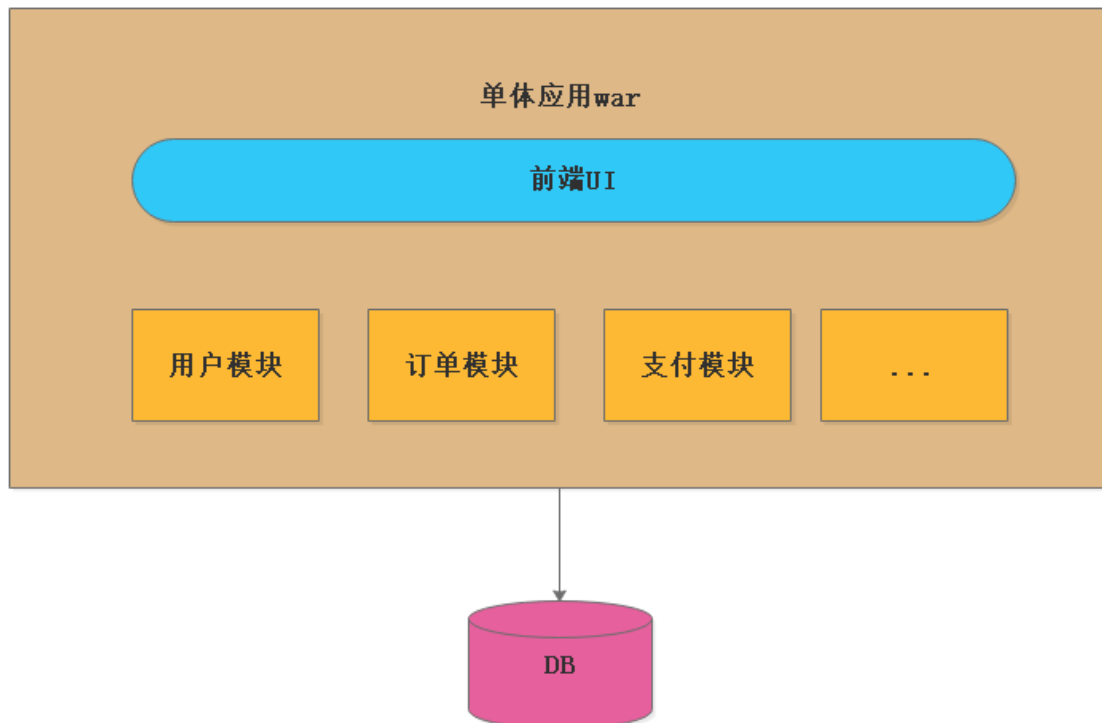
### 1.1)从单体架构说起

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**

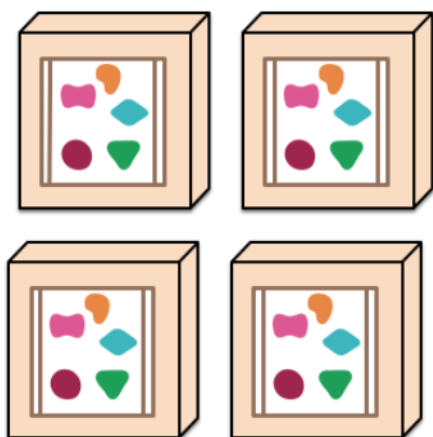
一个单体应用程序把它所有的功能放在一个单一进程中...



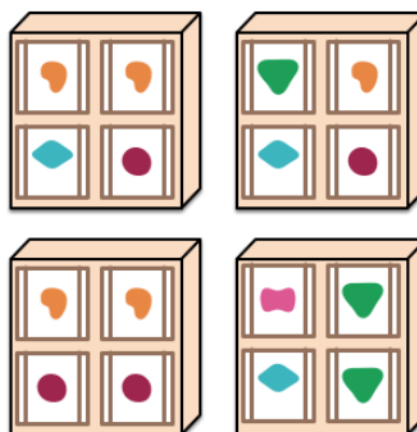
一个微服务架构把每个功能元素放进一个独立的服务中...



...并且通过在多个服务器上复制这个单体进行扩展

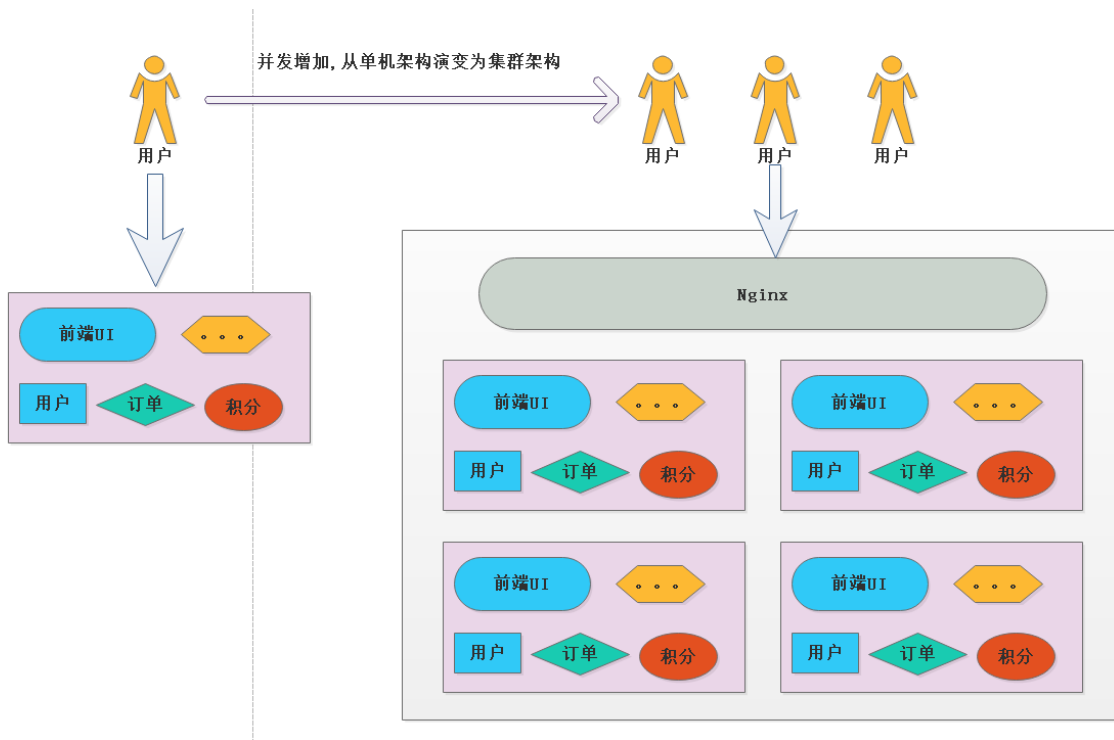


...并且通过跨服务器分发这些服务进行扩展，只在需要时才复制。

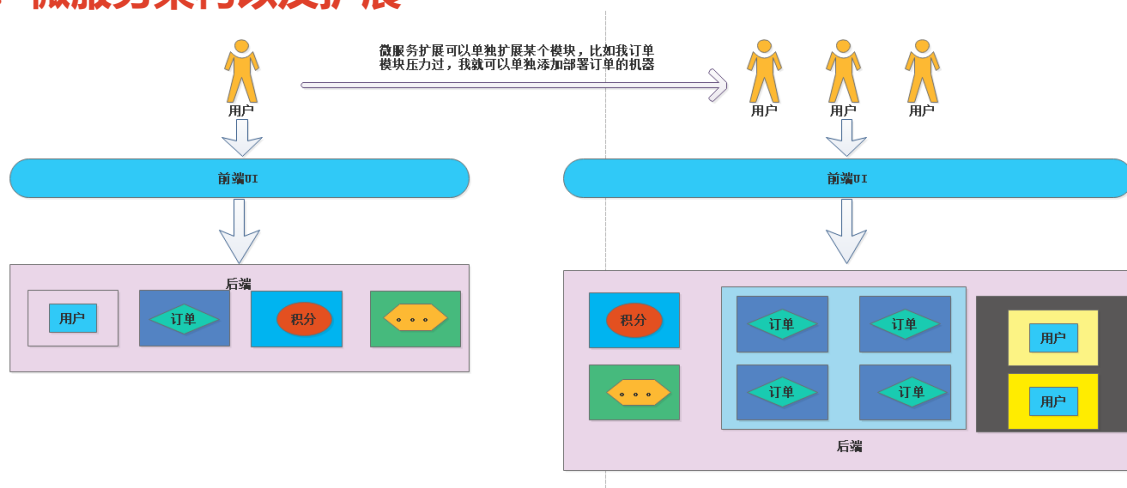


### 1.4.3)单机架构扩展与微服务扩展

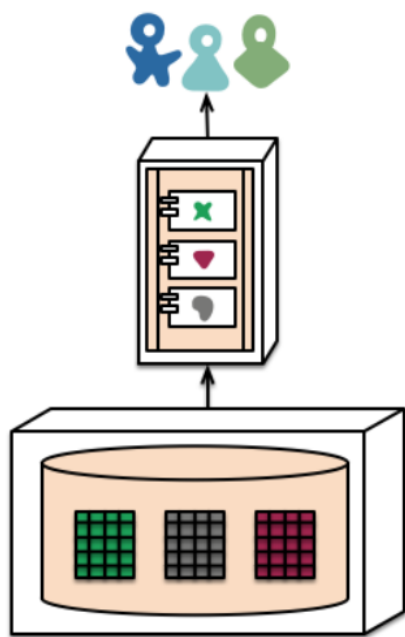
#### ①：单机架构扩展



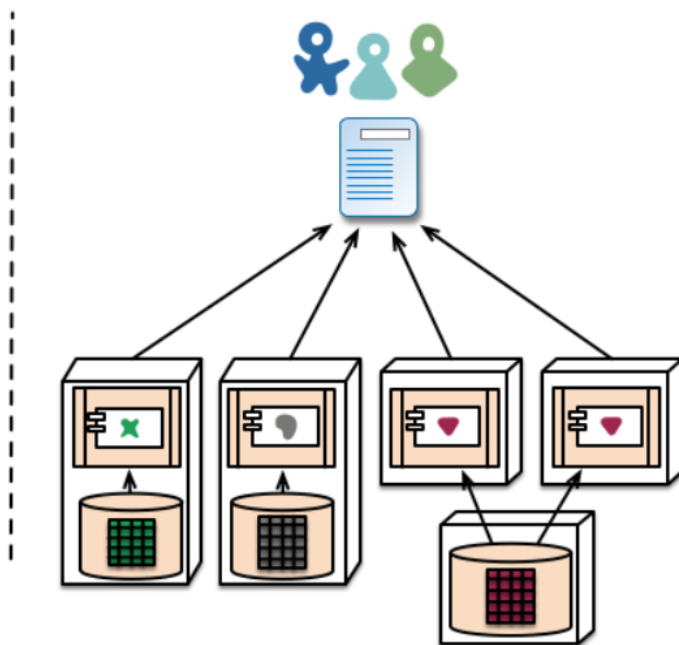
## ②：微服务架构以及扩展



## 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

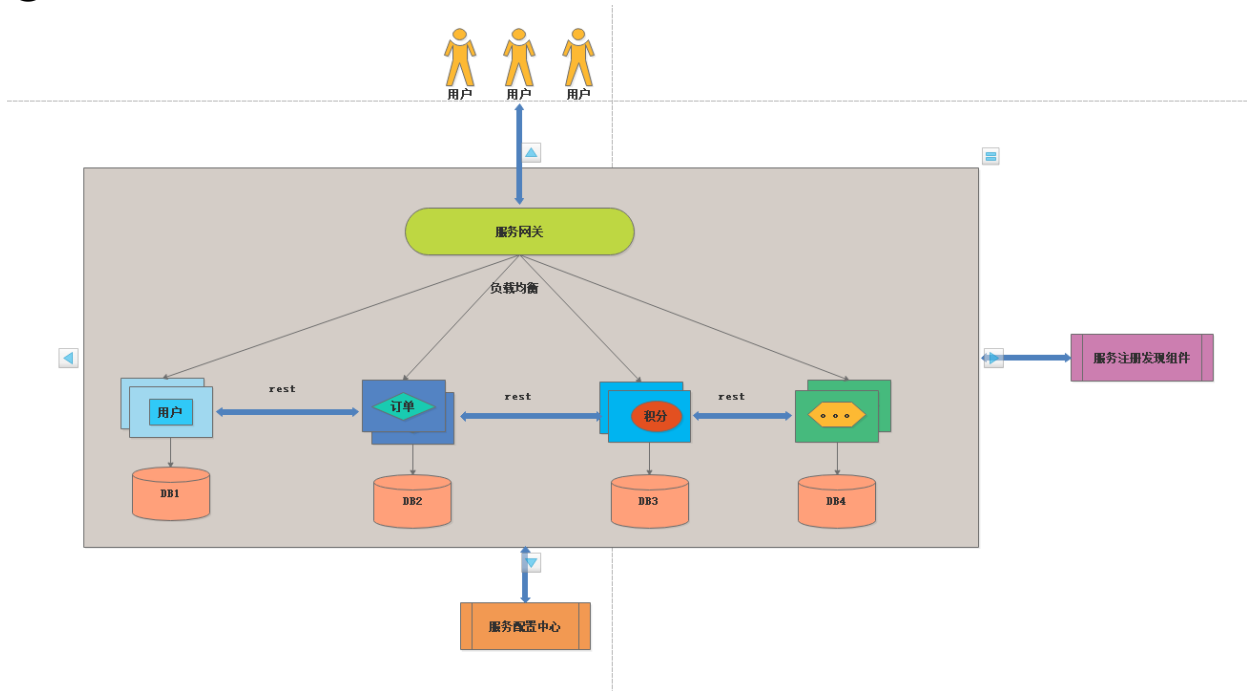
**A:优点:**

- 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:系统能监控等,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

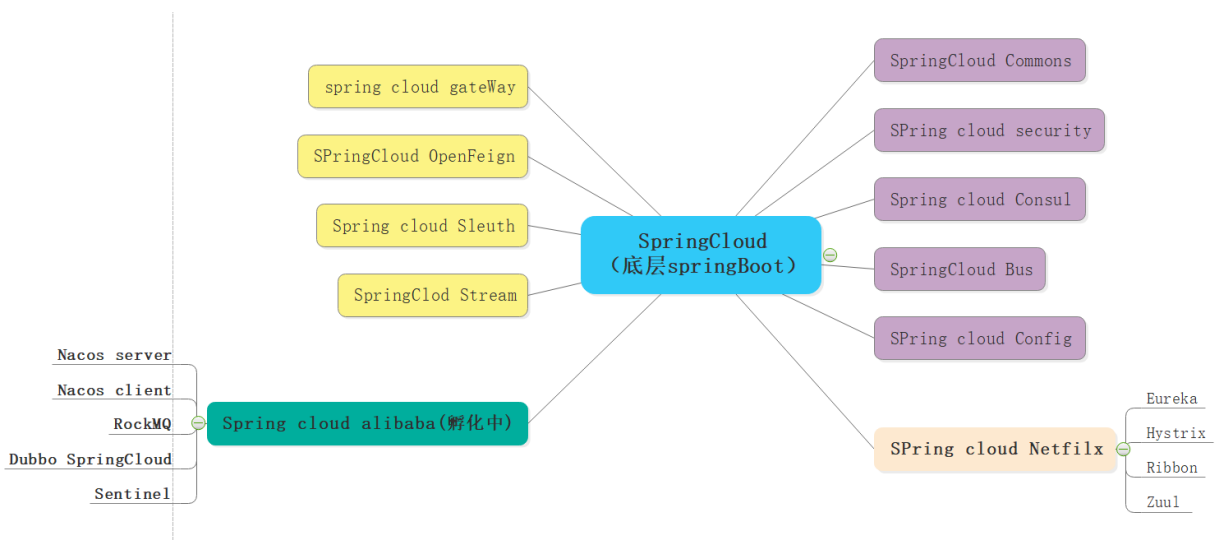
2.1)什么是SpringCloud?<https://spring.io/projects/spring-cloud>

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 confiuration	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	Zuu1, <b>SpringCloud gateWay</b>
Service Call	服务调用	<b>RestTemplate</b> , <b>Ribbon</b> , <b>Feign</b>
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/ <b>Rabbitmq</b> / <b>RockMq</b>



## 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

### 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:

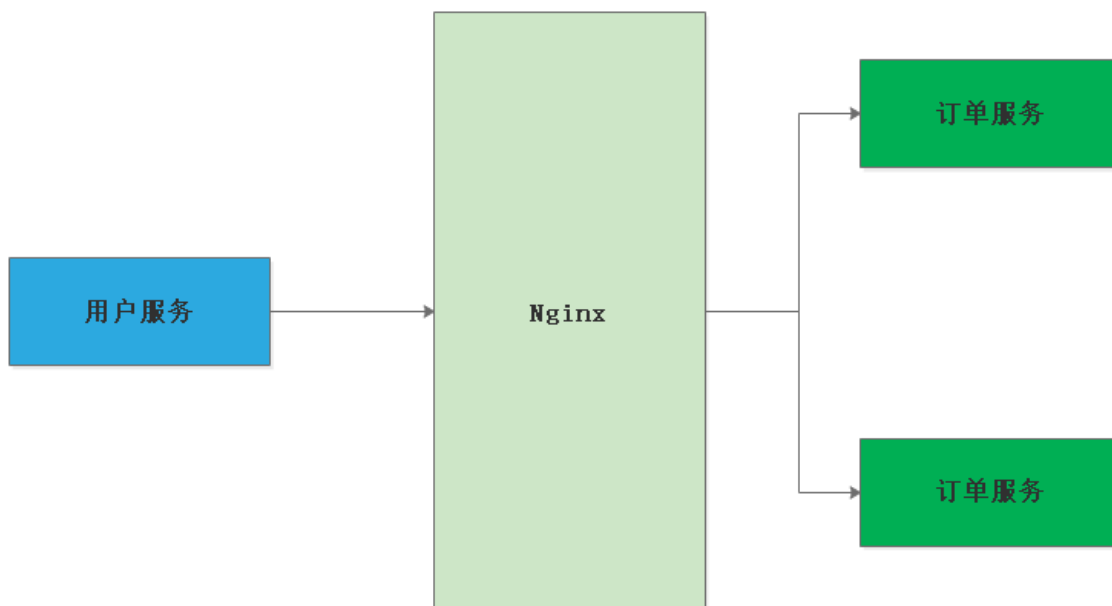
1)从上面看出的缺点就是，我们的在调用的时候，请求的Ip地址和端口是硬编码的.

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)

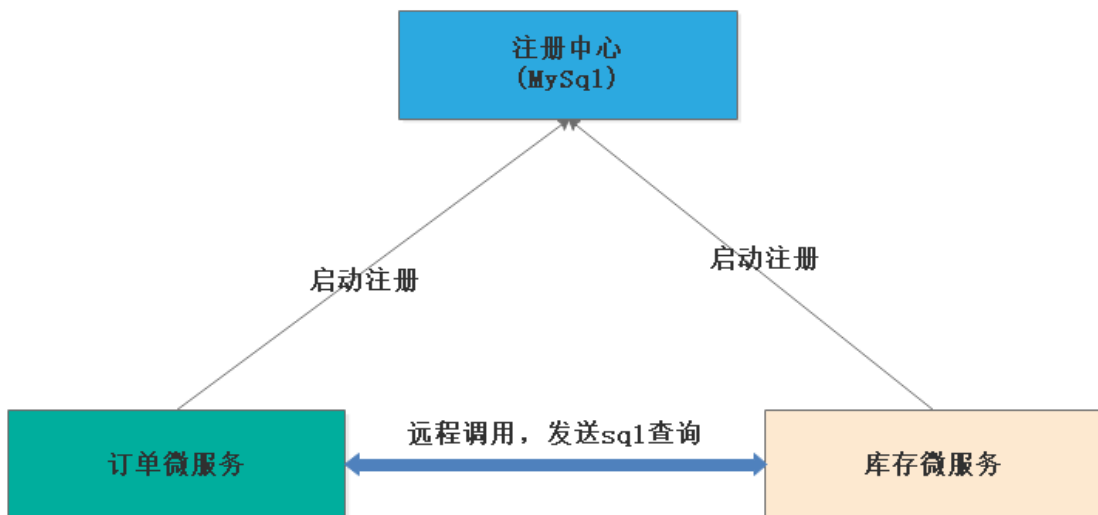
3)有人马上回驳我说，我可以通过ng来做负载均衡,对，我首先认为这是可行的，当时微服务成百上千的服务，难道我们要那成百上千ng么？或者使用一个Ng 那么我们能想一下哪个ng的配置文件有多么复杂。



### 3.2) 大话 服务注册发现原理

V1架构图:

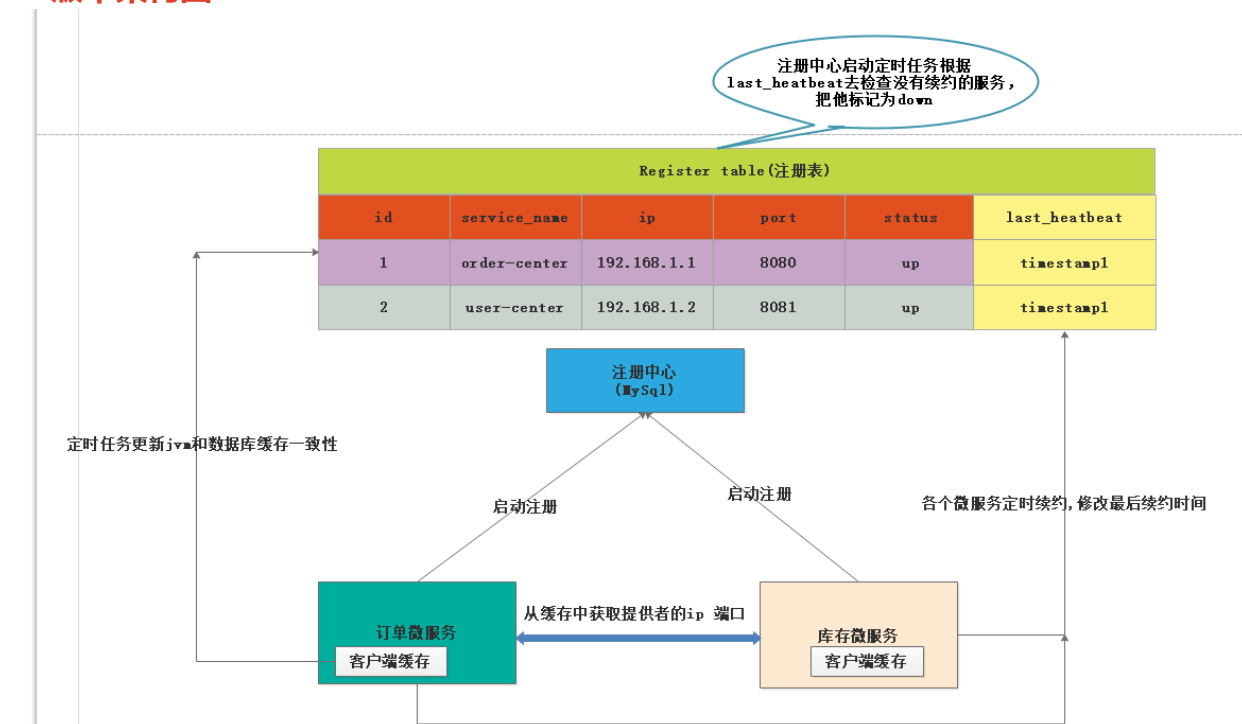
Register table(注册表)				
id	service_name	ip	port	status
1	order-center	192.168.1.1	8080	up
2	user-center	192.168.1.2	8081	up



### 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		
 <a href="#">nacos-server-1.1.4.tar.gz</a>	linux版本	49.7 MB
 <a href="#">nacos-server-1.1.4.zip</a>	windows版本	49.7 MB
 <a href="#">Source code (zip)</a>		
 <a href="#">Source code (tar.gz)</a>		

#### 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**

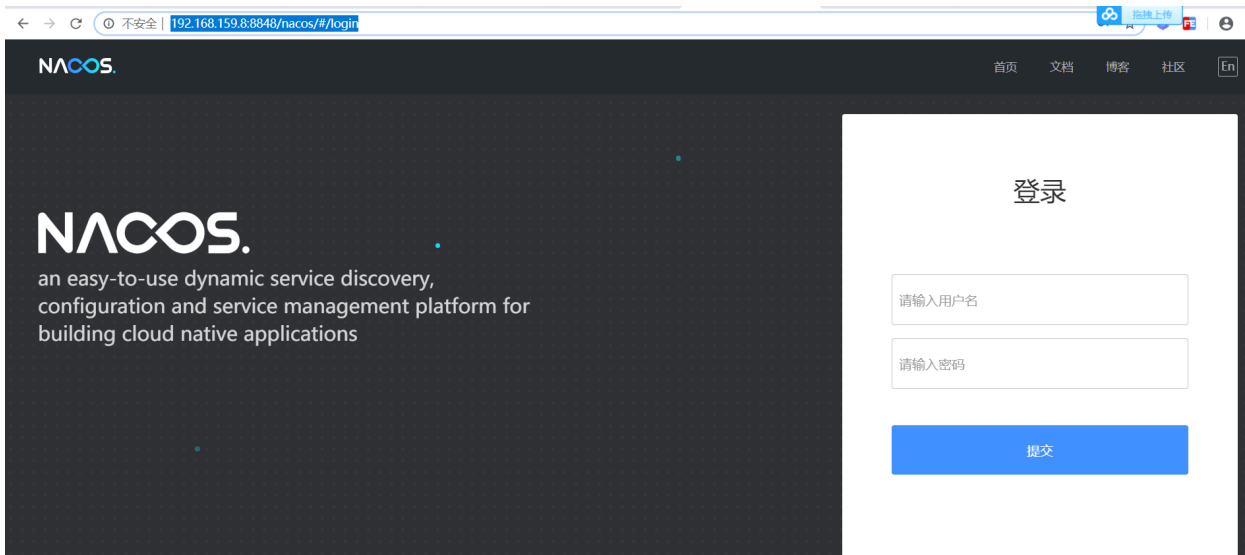
```
[root@smlz bin]# sh startup.sh -m standalone
/usr/local/jdk/jdk1.8.0_221/bin/java -Xms512m -Xmx512m -Xmn256m -Dnacos.standalone=true -Djava.ext.dirs=/usr/local/jdk/jdk1.8.0_221/j
re/lib/ext:/usr/local/jdk/jdk1.8.0_221/lib/ext:/usr/local/spring-cloud-alibaba/nacos/nacos/plugins/cmdb:/usr/local/spring-cloud-alibab
a/nacos/nacos/plugins/mysql -Xloggc:/usr/local/spring-cloud-alibaba/nacos/nacos/logs/nacos_gc.log -verbose:gc -XX:+PrintGCDetails -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/sprin
g-cloud-alibaba/nacos/nacos/target/nacos-server.jar --spring.config.location=classpath:/,classpath:/config/,file:./,file:./config/,fi
le:/usr/local/spring-cloud-alibaba/nacos/nacos/conf/ --logging.config=/usr/local/spring-cloud-alibaba/nacos/nacos/conf/nacos-logback.x
ml --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nacos的服务端 <http://192.168.159.8:8848/nacos/index.html>

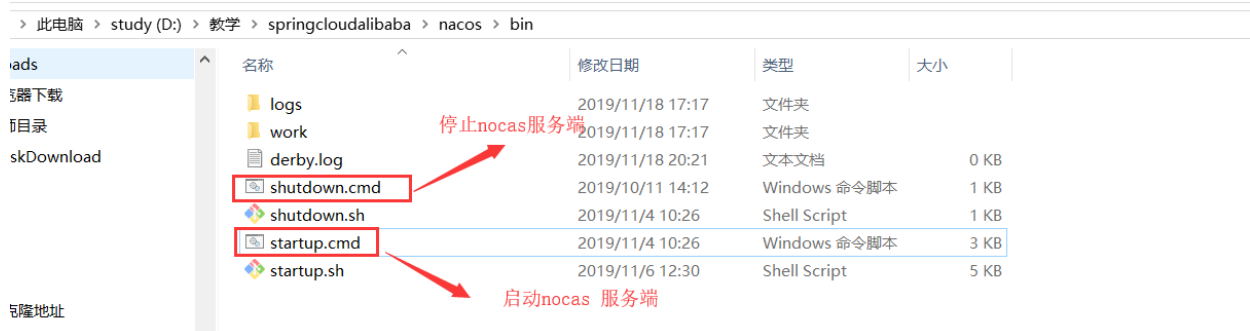
默认的用户名密码是 nocas/nocas



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.cl
ass, args);
7  }
8  }

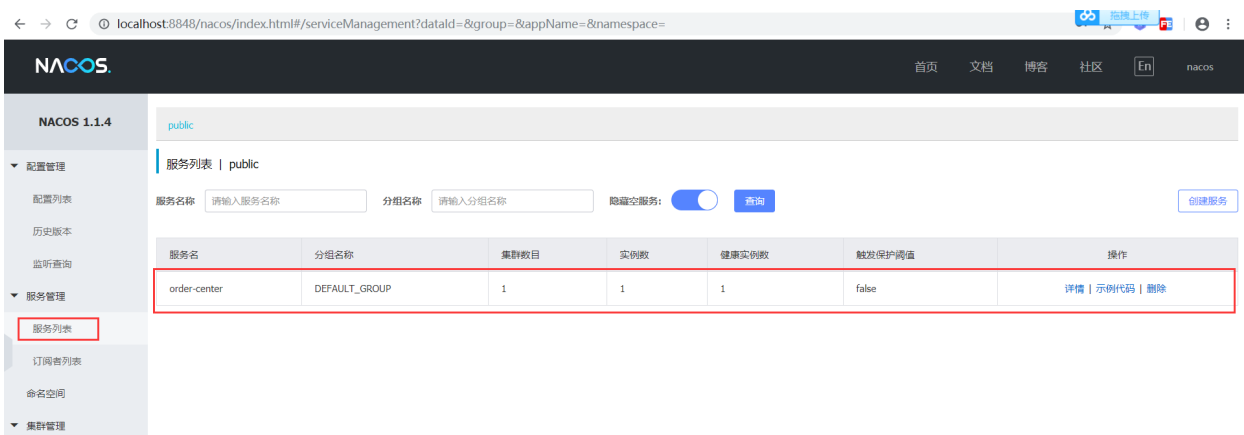
```

### 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

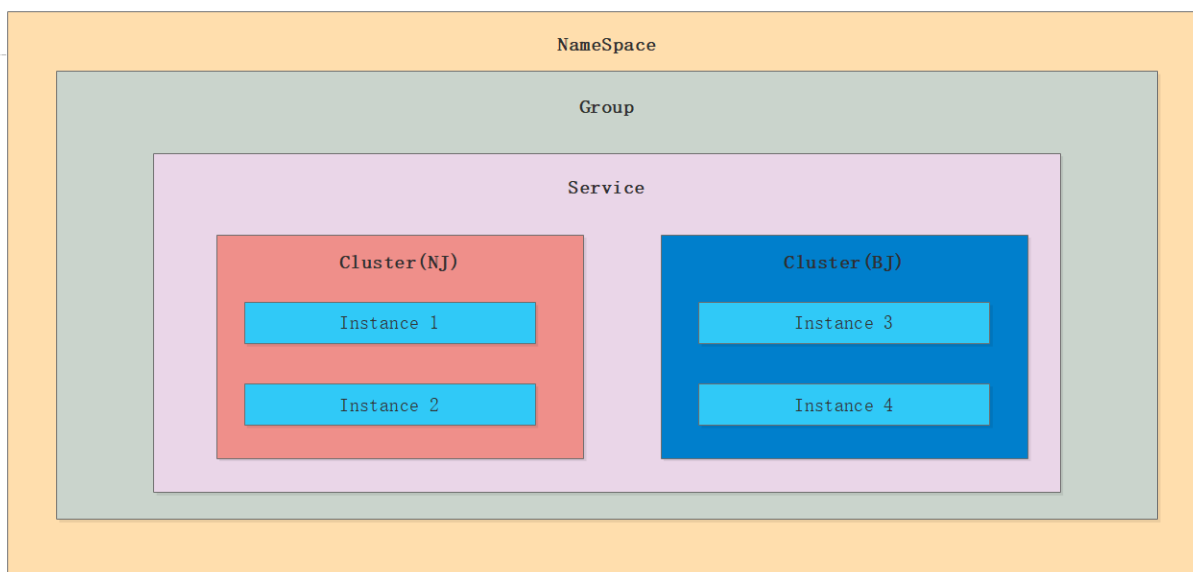
```

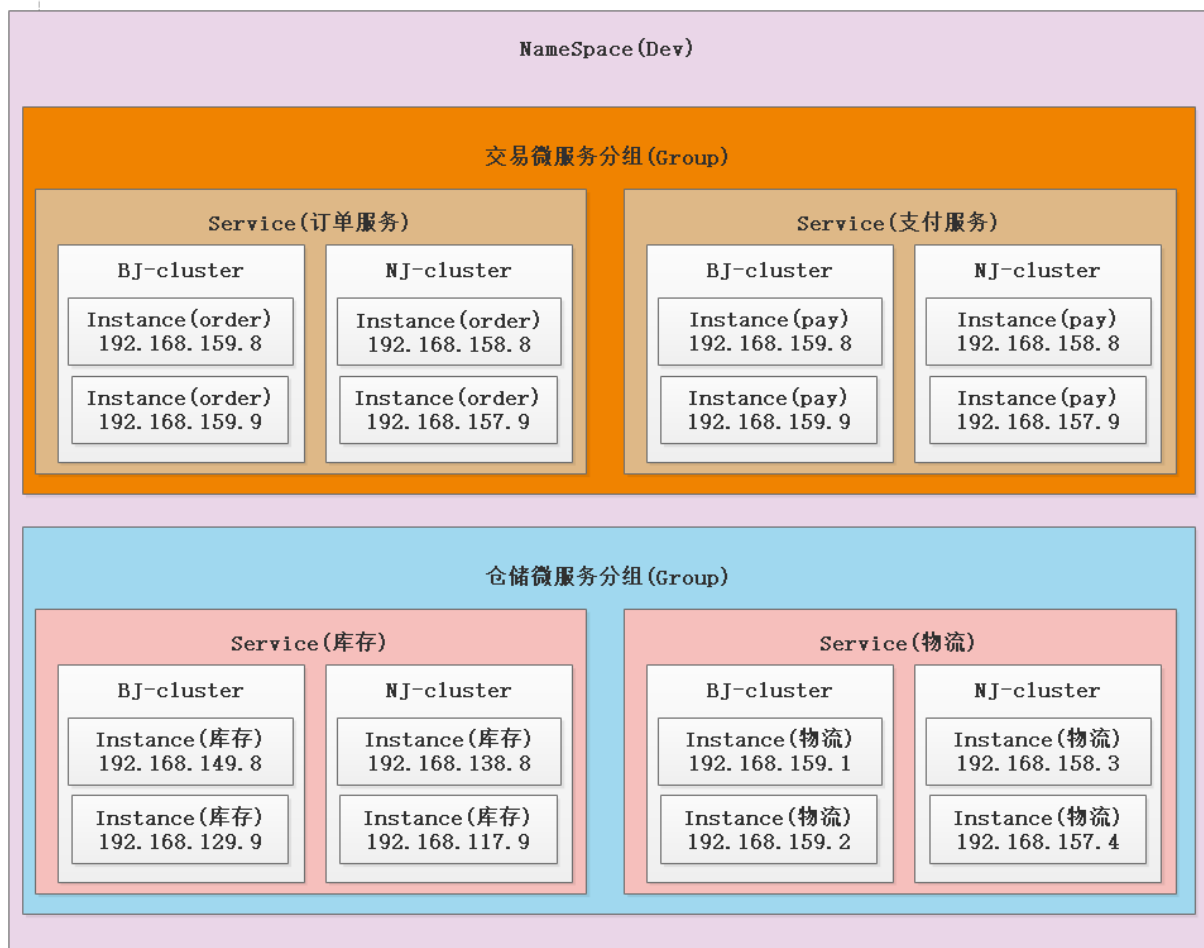
1  @Autowired
2  private DiscoveryClient discoveryClient;
3
4  @GetMapping("/getServiceList")
5  public List<ServiceInstance> getServiceList() {
6    List<ServiceInstance> serviceInstanceList =
discoveryClient.getInstances("order-center");
7    return serviceInstanceList;
8  }

```

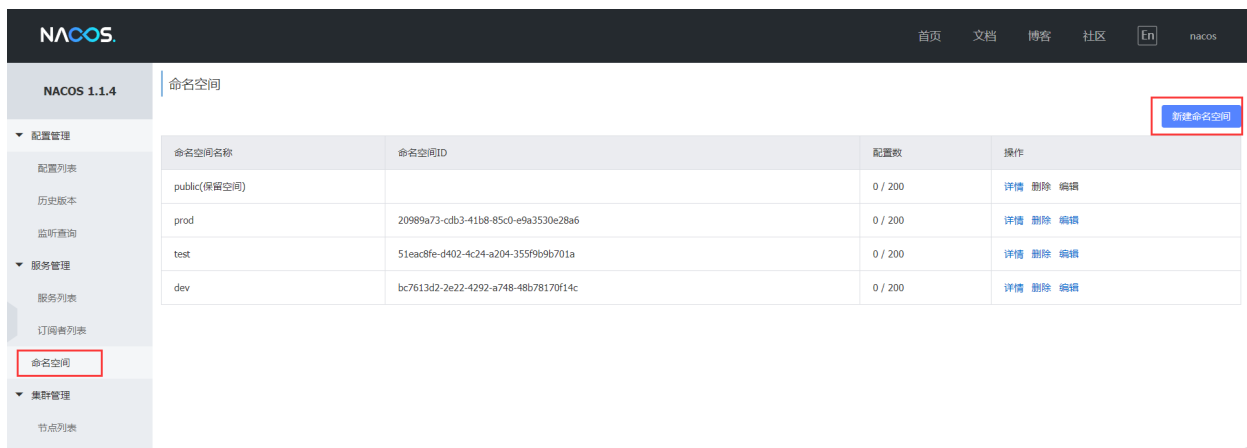
```
[
  {
    "serviceId": "order-center",
    "host": "192.168.0.224",
    "port": 8080,
    "secure": false,
    "metadata": {
      "nacos.instanceId": "192.168.0.224#8080#DEFAULT#DEFAULT_GROUP@@order-center",
      "nacos.weight": "1.0",
      "nacos.cluster": "DEFAULT",
      "nacos.healthy": "true",
      "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)**



**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>

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh) on the left, followed by a circular icon with an 'i' (likely for developer tools or a specific extension). The address bar itself is highlighted in blue and contains the text 'localhost:8080/selectOrderInfoById/1'.

用户微服务没有对应的实例可用