

主讲老师：Fox老师

本期Nacos版本：v2.1.0

### 课前须知：

1. 没有接触过微服务的同学，可以先快速的学习一下spring cloud alibaba的基础课程：

[https://vip.tulingxueyuan.cn/detail/p\\_60decb8be4b0151fc94c41a4/8?product\\_id=p\\_60decb8be4b0151fc94c41a4](https://vip.tulingxueyuan.cn/detail/p_60decb8be4b0151fc94c41a4/8?product_id=p_60decb8be4b0151fc94c41a4)

2. 微服务专题怎么学

- 第一步：基础比较薄弱的同学，先掌握微服务组件是做什么的，有什么用，怎么用，先系统的把spring cloud&spring cloud alibaba大部分组件学一遍  
<https://www.processon.com/view/link/60519545f346fb348a97c9d5>
- 第二步：深入学习每个组件的设计理念，设计思路，底层原理，扩展应用
- 第三步：掌握spring，spring boot源码（主线流程）前提下，再去看微服务组件源码
- 第四步：综合应用实战

1 有道云笔记地址：

2 文档：01 Alibaba微服务组件Nacos注册中心实...

3 链接：[http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=353a5a9ca9136507102122049b78dc56&sub=F957B151B29742B3B00CEAB39AF8EFED)

[id=353a5a9ca9136507102122049b78dc56&sub=F957B151B29742B3B00CEAB39AF8EFED](http://note.youdao.com/noteshare?id=353a5a9ca9136507102122049b78dc56&sub=F957B151B29742B3B00CEAB39AF8EFED)

## 1.注册中心介绍

### 1.1 注册中心的作用

### 1.2 注册中心设计思路分析

### 1.3 注册中心对比

## 2. 什么是 Nacos

### 2.1 Nacos 注册中心架构和基本概念

### 2.2 Nacos注册中心核心功能

### 2.3 Nacos注册中心（Nacos Server）环境搭建

单机模式

集群模式

## 2.4 Spring Cloud Alibaba Nacos快速开始

Spring Cloud Alibaba版本选型

微服务(Nacos Client)整合Nacos注册中心(Nacos Server)

## 2.5 Nacos注册中心常见配置

服务分级存储模型

服务逻辑隔离

临时实例和持久化实例

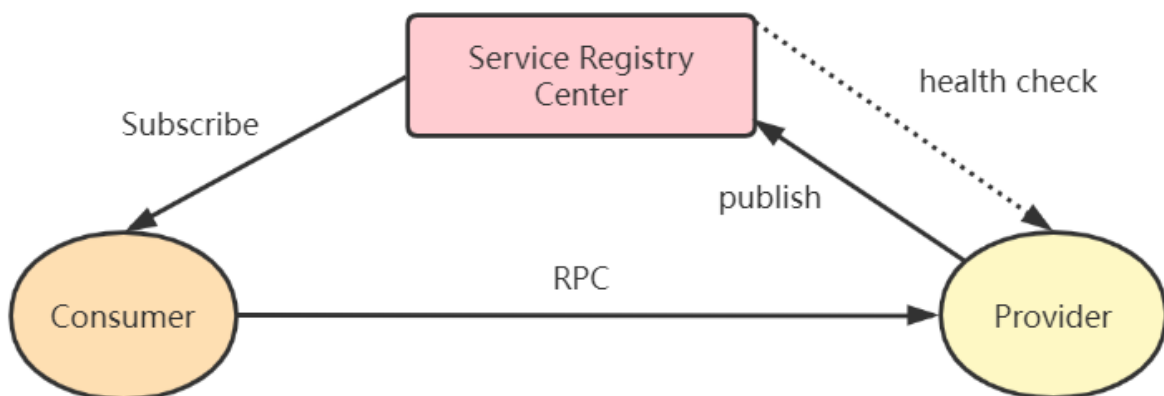
# 1.注册中心介绍

## 1.1 注册中心的作用

思考：不同的微服务如何维护复杂的调用关系？

```
1 //服务之间通过RestTemplate调用，url写死
2 String url = "http://localhost:8020/order/findOrderByUserId/"+id;
3 R result = restTemplate.getForObject(url,R.class);
```

实现服务发现的设计思路：

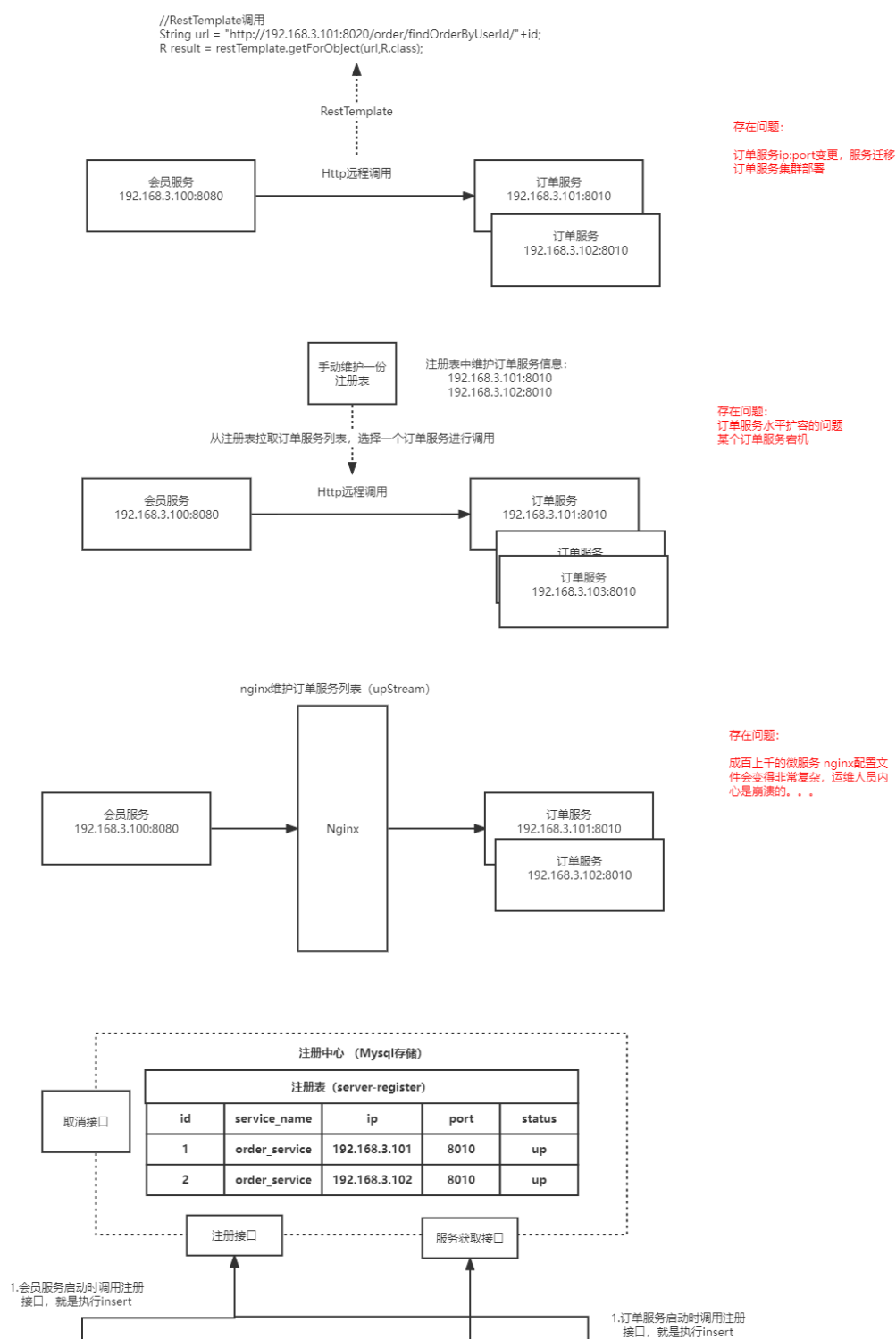


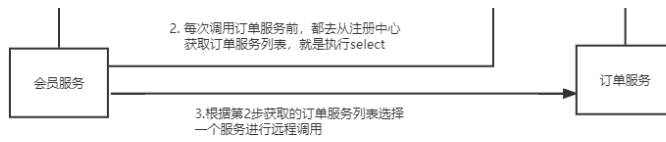
服务注册中心的作用就是服务注册与发现

- 服务注册，就是将提供某个服务的模块信息(通常是这个服务的ip和端口)注册到1个公共的组件上去。
- 服务发现，就是新注册的这个服务模块能够及时的被其他调用者发现。不管是服务新增和服务删减都能实现自动发现。

## 1.2 注册中心设计思路分析

<https://www.processon.com/view/link/5e71cc85e4b011fcce9d604d>

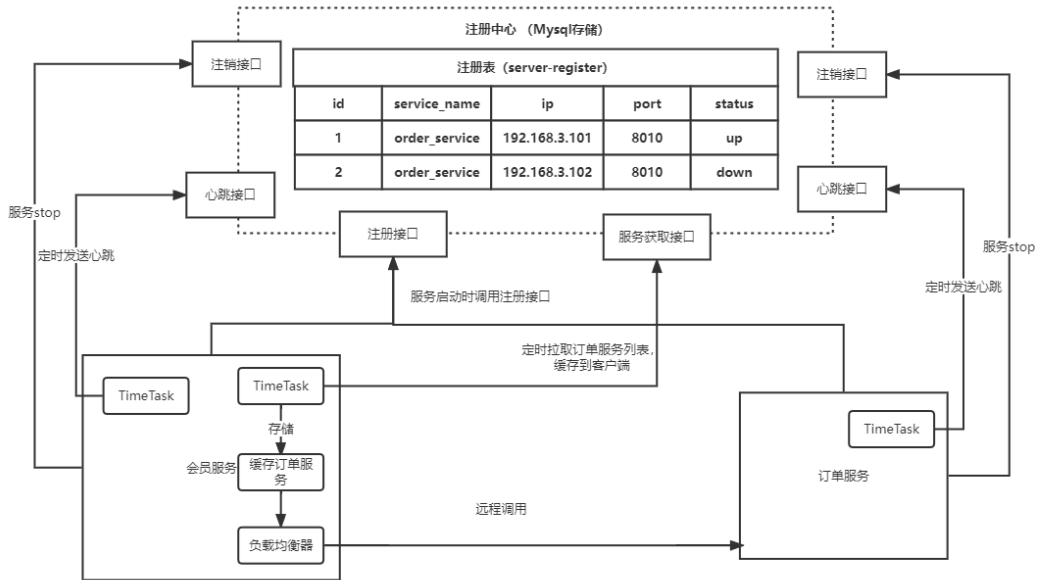




思考：

1. 每次都要去注册中心拉取订单服务列表吗？注册中心宕机了怎么办？

2. 如果拉取的订单服务对应的机器宕机了怎么办？



## 1.3 注册中心对比

	Nacos	Eureka	Consul	CoreDNS	Zookeeper
一致性协议	CP+AP	AP	CP	—	CP
健康检查	TCP/HTTP/MYSQL/Client Beat	Client Beat	TCP/HTTP/gRPC/Cmd	—	Keep Alive
负载均衡策略	权重/metadata/Selector	Ribbon	Fabio	RoundRobin	—
雪崩保护	有	有	无	无	无
自动注销实例	支持	支持	支持	不支持	支持
访问协议	HTTP/DNS	HTTP	HTTP/DNS	DNS	TCP
监听支持	支持	支持	支持	不支持	支持
多数据中心	支持	支持	支持	不支持	不支持
跨注册中心同步	支持	不支持	支持	不支持	不支持
SpringCloud 集成	支持	支持	支持	不支持	支持
Dubbo集成	支持	不支持	支持	不支持	支持
K8S集成	支持	不支持	支持	支持	不支持

## 2. 什么是 Nacos

Nacos 是 Dynamic Naming and Configuration Service 的首字母简称；一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。

**Nacos 致力于帮助您发现、配置和管理微服务。** Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。

Nacos 的关键特性包括：

- 服务发现和服务健康监测
- 动态配置服务
- 动态 DNS 服务
- 服务及其元数据管理

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

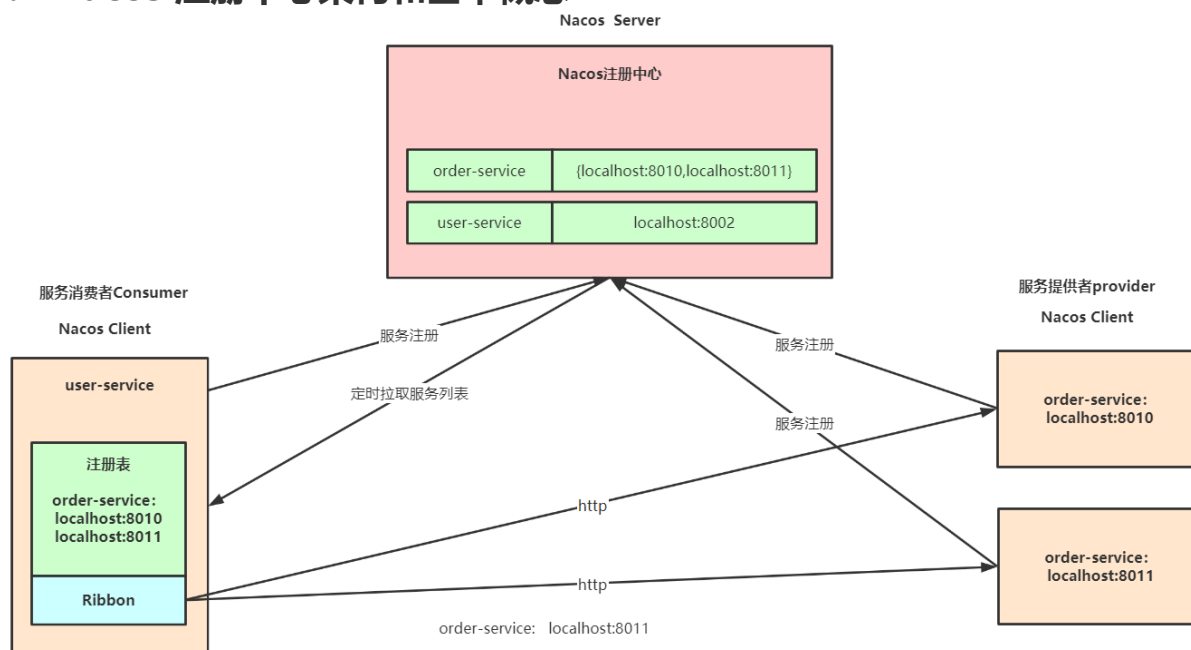
OpenAPI文档：<https://nacos.io/zh-cn/docs/open-api.html>

**nacos版本：** v2.1.0

### Nacos 优势

- **易用**：简单的数据模型，标准的 restfulAPI，易用的控制台，丰富的使用文档。
- **稳定**：99.9% 高可用，脱胎于历经阿里巴巴 10 年生产验证的内部产品，支持具有数百万服务的大规模场景，具备企业级 SLA 的开源产品。
- **实时**：数据变更毫秒级推送生效；1w 级，SLA 承诺 1w 实例上下线 1s，99.9% 推送完成；10w 级，SLA 承诺 1w 实例上下线 3s，99.9% 推送完成；100w 级别，SLA 承诺 1w 实例上下线 9s 99.9% 推送完成。
- **规模**：十万级服务/配置，百万级连接，具备强大扩展性。

## 2.1 Nacos 注册中心架构和基本概念



### 服务 (Service)

服务是指一个或一组软件功能（例如特定信息的检索或一组操作的执行），其目的是不同的客户端可以为不同的目的重用（例如通过跨进程的网络调用）。Nacos 支持主流的服务生态，如 Kubernetes Service、gRPC|Dubbo RPC Service 或者 Spring Cloud RESTful Service。

### 服务注册中心 (Service Registry)

**服务注册中心**，它是服务及其实例和元数据的数据库。服务实例在启动时注册到服务注册表，并在关闭时注销。服务和路由器的客户端查询服务注册表以查找服务的可用实例。服务注册中心可能会调用服务实例的健康检查 API 来验证它是否能够处理请求。

### 服务元数据 (Service Metadata)

服务元数据是指包括服务端点(endpoints)、服务标签、服务版本号、服务实例权重、路由规则、安全策略等描述服务的数据。

### 服务提供方 (Service Provider)

是指提供可复用和可调用服务的应用方。

### 服务消费方 (Service Consumer)

是指会发起对某个服务调用的应用方。

## 2.2 Nacos注册中心核心功能

**服务注册：**Nacos Client会通过发送REST请求的方式向Nacos Server注册自己的服务，提供自身的元数据，比如ip地址、端口等信息。Nacos Server接收到注册请求后，就会把这些元数据信息存储在一个双层的内存Map中。

**服务心跳：**在服务注册后，Nacos Client会维护一个定时心跳来持续通知Nacos Server，说明服务一直处于可用状态，防止被剔除。**默认5s发送一次心跳。**

**服务同步：**Nacos Server集群之间会互相同步服务实例，用来保证服务信息的一致性。

**服务发现：**服务消费者（Nacos Client）在调用服务提供者的服务时，会发送一个REST请求给Nacos Server，获取上面注册的服务清单，并且缓存在Nacos Client本地，同时会在Nacos Client本地开启一个定时任务定时拉取服务端最新的注册表信息更新到本地缓存

**服务健康检查：**Nacos Server会开启一个定时任务用来检查注册服务实例的健康情况，对于**超过15s没有收到客户端心跳的实例**会将它的**healthy属性置为false**(客户端服务发现时不会发现)，如果某个实例**超过30秒没有收到心跳，直接剔除该实例**(被剔除的实例如果恢复发送心跳则会重新注册)

## 2.3 Nacos注册中心（Nacos Server）环境搭建

### 单机模式

官方文档：<https://nacos.io/zh-cn/docs/deployment.html>

### 下载安装包

下载地址：<https://github.com/alibaba/nacos/releases/download/2.1.0/nacos-server-2.1.0.tar.gz>

解压，进入nacos目录，单机模式启动nacos

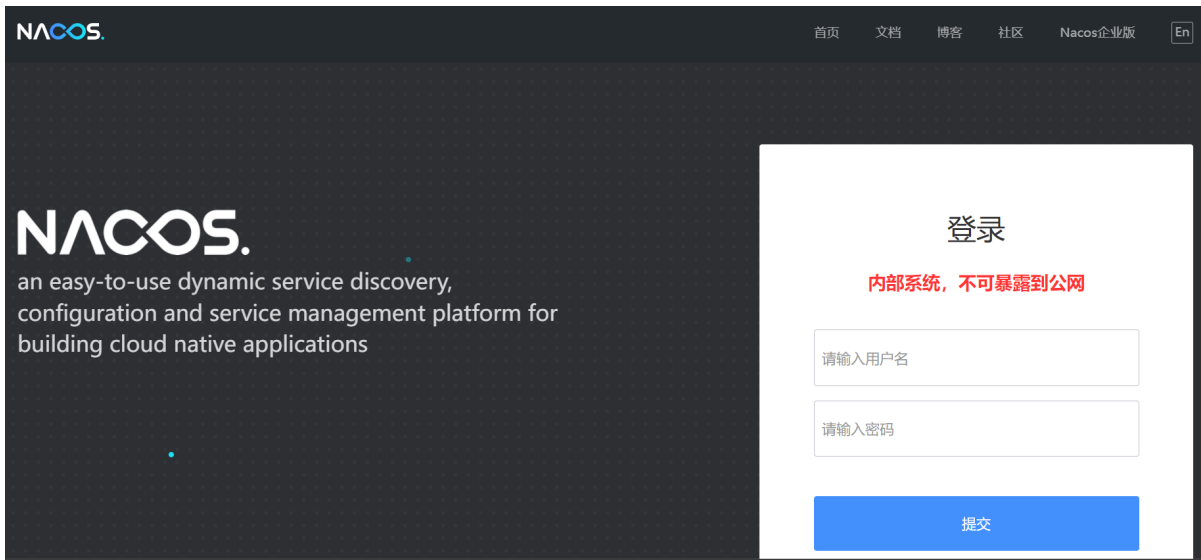
```
1 #单机模式启动nacos
2 bin/startup.sh -m standalone
```

也可以修改默认启动方式

```
export SERVER="nacos-server"
export MODE="cluster"
export FUNCTION_MODE="all"
export MEMBER_LIST=""
export EMBEDDED_STORAGE=""
case $opt in
f)
```

可以改为standalone，单机启动

访问nacos的管理端：<http://192.168.65.103:8848/nacos>，默认的用户名密码是 nacos/nacos



### 集群模式

官网文档：<https://nacos.io/zh-cn/docs/cluster-mode-quick-start.html>

#### 1) 环境准备

- 安装好 JDK，需要 1.8 及其以上版本
- 建议: 2核 CPU / 4G 内存 及其以上
- 建议: 生产环境 3 个节点 及其以上

```
1 # 准备三台centos7服务器
2 192.168.65.174
3 192.168.65.192
4 192.168.65.204
```

注意：在单台服务器上搭建伪集群不能使用连续端口号（比如8848,8849,8850），因为使用8848（grpc会占用9848,9849），8849（grpc会占用9849,9850），会导致端口冲突

原因：Nacos2.x版本相比1.X新增了gRPC的通信方式，因此需要增加2个端口。新增端口是在配置的主端口(server.port)基础上，进行一定偏移量自动生成。

端口	与主端口的偏移量	描述
9848	1000	客户端gRPC请求服务端端口，用于客户端向服务端发起连接和请求





## 2) 以192.168.65.204为例，进入nacos目录

### 2.1) 修改conf/application.properties的配置，使用外置数据源

```

1 #使用外置mysql数据源
2 spring.datasource.platform=mysql
3
4 ### Count of DB:
5 db.num=1
6
7 ### Connect URL of DB:
8 db.url.0=jdbc:mysql://192.168.65.204:3306/nacos?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=UTC
9 db.user.0=root
10 db.password.0=root

```

```

### If use MySQL as datasource:
spring.datasource.platform=mysql

### Count of DB:
db.num=1

### Connect URL of DB:
db.url.0=jdbc:mysql://192.168.65.206:3306/nacos?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=UTC
db.user.0=root
db.password.0=123456

```

### 2.2) 将conf/cluster.conf.example改为cluster.conf,添加节点配置

```

1 mv conf/cluster.conf.example conf/cluster.conf
2 vim conf/cluster.conf
3
4 # ip:port
5 192.168.65.174:8848
6 192.168.65.192:8848
7 192.168.65.204:8848

```

**注意：**不要使用localhost或127.0.0.1，针对多网卡环境，nacos可以指定网卡或ip

```

1 #多网卡选择
2 #ip-address参数可以直接设置nacos的ip
3 #该参数设置后，将会使用这个IP去cluster.conf里进行匹配，请确保这个IP的值在cluster.conf里是存在的
4 nacos.inetutils.ip-address=192.168.65.206

```

```

5
6 #use-only-site-local-interfaces参数可以让nacos使用局域网ip，这个在nacos部署
  的机器有多网卡时很有用，可以让nacos选择局域网网卡
7 nacos.inetutils.use-only-site-local-interfaces=true
8
9 #ignored-interfaces支持网卡数组，可以让nacos忽略多个网卡
10 nacos.inetutils.ignored-interfaces[0]=eth0
11 nacos.inetutils.ignored-interfaces[1]=eth1
12
13 #preferred-networks参数可以让nacos优先选择匹配的ip，支持正则匹配和前缀匹配
14 nacos.inetutils.preferred-networks[0]=30.5.124.
15

```

192.168.65.174, 192.168.65.192 按同样的方式配置。

### 3) mysql中创建nacos数据库

sql脚本：

<https://github.com/alibaba/nacos/blob/2.1.0/distribution/conf/nacos-mysql.sql>

### 4) 如果内存不够，可以调整jvm参数

```

1 #修改启动脚本
2 vim bin\startup.sh

```

```

# JVM Configuration
#=====
=====
if [[ "${MODE}" == "standalone" ]]; then
    JAVA_OPT="${JAVA_OPT} -Xms512m -Xmx512m -Xmn256m"
    JAVA_OPT="${JAVA_OPT} -Dnacos.standalone=true"
else

```

### 5) 分别启动三个节点上的nacos

以192.168.65.204为例，进入nacos目录，启动nacos

```

1 #启动nacos
2 bin/startup.sh

```

```

[es@hadoop02 nacos]$ bin/startup.sh
/home/es/jdk-11.0.12/bin/java -server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=
or -XX:HeapDumpPath=/home/es/nacos/logs/java_heapdump.hprof -XX:-UseLargePages -Dnacos.member.list= -Xl
, filesize=102400 -Dloader.path=/home/es/nacos/plugins/health,/home/es/nacos/plugins/cmd,/home/es/nacos
target/nacos-server.jar --spring.config.additional-location=file:/home/es/nacos/conf/ --logging.config=
=524288
nacos is starting with cluster
nacos is starting, you can check the /home/es/nacos/logs/start.out

```

### 6) 访问nacos管理界面

登录<http://192.168.65.204:8848/nacos>，用户名和密码都是nacos

配置管理

服务管理

权限控制

命名空间

集群管理

节点列表

节点列表 | public

节点Ip

请输入节点Ip

查询

节点Ip	节点状态	节点元数据	操作
192.168.65.174:8848	UP	> 节点元数据	下线
192.168.65.192:8848	UP	> 节点元数据	下线
192.168.65.204:8848	UP	> 节点元数据	下线

## 2.4 Spring Cloud Alibaba Nacos快速开始

## Spring Cloud Alibaba版本选型

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

Spring Cloud Alibaba Version	Spring Cloud Version	Spring Boot Version	Nacos Version
2.2.8.RELEASE	Spring Cloud Hoxton.SR12	2.3.12.RELEASE	2.1.0

## 父pom配置

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apac
he.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.3.12.RELEASE</version>
9     <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.tuling.mall</groupId>
12    <artifactId>vip-spring-cloud-alibaba</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>vip-spring-cloud-alibaba</name>
15    <packaging>pom</packaging>
16    <description>Demo project for vip-spring-cloud-alibaba</description>
17
18    <properties>
```

```

19 <java.version>1.8</java.version>
20 <spring-cloud.version>Hoxton.SR12</spring-cloud.version>
21 <spring-cloud-alibaba.version>2.2.8.RELEASE</spring-cloud-alibaba.version>
22 </properties>
23
24
25 <dependencyManagement>
26 <dependencies>
27
28 <dependency>
29 <groupId>org.springframework.cloud</groupId>
30 <artifactId>spring-cloud-dependencies</artifactId>
31 <version>${spring-cloud.version}</version>
32 <type>pom</type>
33 <scope>import</scope>
34 </dependency>
35 <dependency>
36 <groupId>com.alibaba.cloud</groupId>
37 <artifactId>spring-cloud-alibaba-dependencies</artifactId>
38 <version>${spring-cloud-alibaba.version}</version>
39 <type>pom</type>
40 <scope>import</scope>
41 </dependency>
42
43 </dependencies>
44 </dependencyManagement>
45
46
47 </project>

```

## 微服务(Nacos Client)整合Nacos注册中心(Nacos Server)

### 配置服务提供者mall-order

服务提供者可以通过 Nacos 的服务注册发现功能将其服务注册到 Nacos server 上。

### 1) 引入依赖

当前项目pom中引入依赖

```

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

```

```

4 </dependency>
5
6 <dependency>
7   <groupId>com.alibaba.nacos</groupId>
8   <artifactId>nacos-common</artifactId>
9   <version>2.1.0</version>
10 </dependency>

```

注意：只引入spring-cloud-starter-alibaba-nacos-discovery包会出现下面的错误，还需要再引入nacos-common包

```

java.lang.NoClassDefFoundError: Create breakpoint : com/alibaba/nacos/common/utils/StringUtils
    at com.alibaba.nacos.client.logging.AbstractNacosLogging.<clinit>(AbstractNacosLogging.java:41) ~[nacos-client-2.1.0.jar:na
    at com.alibaba.nacos.client.logging.NacosLogging.<init>(NacosLogging.java:40) ~[nacos-client-2.1.0.jar:na]
    at com.alibaba.nacos.client.logging.NacosLogging.<init>(NacosLogging.java:29) ~[nacos-client-2.1.0.jar:na]
    at com.alibaba.nacos.client.logging.NacosLogging$NacosLoggingInstance.<clinit>(NacosLogging.java:49) ~[nacos-client-2.1.0.
    at com.alibaba.nacos.client.logging.NacosLogging.getInstance(NacosLogging.java:53) ~[nacos-client-2.1.0.jar:na]
    at com.alibaba.cloud.nacos.discovery.logging.NacosLoggingListener.onApplicationEvent(NacosLoggingListener.java:44) ~[spring

```

此bug已经修复了，不需要再引入nacos-common了，只需要引入spring-cloud-starter-alibaba-nacos-discovery

## 2)配置nacos注册中心

```

1 server:
2   port: 8020
3
4 spring:
5   application:
6     name: mall-order #微服务名称
7
8   #配置nacos注册中心地址
9   cloud:
10    nacos:
11     discovery:
12     server-addr: 127.0.0.1:8848

```

更多配置：<https://github.com/alibaba/spring-cloud-alibaba/wiki/Nacos-discovery>

配置项	key	默认值	说明
服务端地址	spring.cloud.nacos.discovery.server-addr	无	Nacos Server 启动监听的ip地址和端口
服务名	spring.cloud.nacos.discovery.service	\${spring.application.name}	给当前的服务命名
服务分组	spring.cloud.nacos.discovery.group	DEFAULT_GROUP	设置服务所处的分组
权重	spring.cloud.nacos.discovery.weight	1	取值范围 1 到 100，数值越大，权重越大

网卡名	<code>spring.cloud.nacos.discovery.network-interface</code>	无	当IP未配置时，注册的IP为此网卡所对应的IP地址，如果此项也未配置，则默认取第一块网卡的地址
注册的IP地址	<code>spring.cloud.nacos.discovery.ip</code>	无	优先级最高
注册的端口	<code>spring.cloud.nacos.discovery.port</code>	-1	默认情况下不用配置，会自动探测
命名空间	<code>spring.cloud.nacos.discovery.namespace</code>	无	常用场景之一是不同的环境的注册的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。
AccessKey	<code>spring.cloud.nacos.discovery.access-key</code>	无	当要上阿里云时，阿里云上面的一个云账号名
SecretKey	<code>spring.cloud.nacos.discovery.secret-key</code>	无	当要上阿里云时，阿里云上面的一个云账号密码
Metadata	<code>spring.cloud.nacos.discovery.metadata</code>	无	使用Map格式配置，用户可以根据自己的需要自定义一些和服务相关的元数据信息
日志文件名	<code>spring.cloud.nacos.discovery.log-name</code>	无	
集群	<code>spring.cloud.nacos.discovery.cluster-name</code>	DEFAULT	配置成Nacos集群名称
接入点	<code>spring.cloud.nacos.discovery.endpoint</code>	UTF-8	地域的某个服务的入口域名，通过此域名可以动态地拿到服务端地址
是否集成Ribbon	<code>ribbon.nacos.enabled</code>	true	一般都设置成true即可
是否开启Nacos Watch	<code>spring.cloud.nacos.discovery.watch.enabled</code>	true	可以设置成false来关闭watch



### 3) 启动mall-order，nacos管理端界面查看是否成功注册

历史版本							
监听查询							
▼ 服务管理	服务名	分组名称	集群数	实例数	健康实例数	触发保护阈值	操作
	service-order	DEFAULT_GROUP	1	1	1	false	<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">删除</a>
服务列表							
订阅者列表							

测试，通过[Open API](#)查询实例列表

<http://localhost:8848/nacos/v1/ns/instance/list?serviceName=mall-order>

```
{
  name: "DEFAULT_GROUP@@mall-order",
  groupName: "DEFAULT_GROUP",
  clusters: "",
  cacheMillis: 10000,
- hosts: [
  - {
    ip: "192.168.65.103",
    port: 8021,
    weight: 1,
    healthy: true,
    enabled: true,
    ephemeral: true,
    clusterName: "SH",
    serviceName: "DEFAULT_GROUP@@mall-order",
- metadata: {
      preserved.register.source: "SPRING_CLOUD"
    },
    instanceHeartBeatTimeOut: 15000,
    ipDeleteTimeout: 30000,
    instanceHeartBeatInterval: 5000
  },
  - {
    ip: "192.168.65.103",
    port: 8020,
    weight: 1,
    healthy: true,
    enabled: true,
    ephemeral: true,
    clusterName: "SH",
    serviceName: "DEFAULT_GROUP@@mall-order",
- metadata: {
      preserved.register.source: "SPRING_CLOUD"
    },
  }
]
```

配置服务消费者mall-user

服务消费者可以通过 Nacos 的服务注册发现功能从 Nacos server 上获取到它要调用的服务。

### 1) 引入依赖

当前项目pom中引入依赖

```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
4 </dependency>
5
6 <dependency>
7   <groupId>com.alibaba.nacos</groupId>
8   <artifactId>nacos-common</artifactId>
9   <version>2.1.0</version>
10 </dependency>
```

### 2)配置nacos注册中心

```
1 server:
2   port: 8040
3
4 spring:
5   application:
6     name: mall-user #微服务名称
7
8   #配置nacos注册中心地址
9   cloud:
10     nacos:
11       discovery:
12         server-addr: 127.0.0.1:8848
13
```

### 3) 启动mall-user，nacos管理端界面查看是否成功注册

NACOS 2.1.0

配置管理

服务管理

服务列表

订阅者列表

权限控制

命名空间

public

服务列表 | public

服务名称  分组名称  隐藏空服务: ☒

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
mall-order	DEFAULT_GROUP	1	2	2	false	<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">订阅者</a>   <a href="#">删除</a>
mall-user	DEFAULT_GROUP	1	1	1	false	<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">订阅者</a>   <a href="#">删除</a>

### 4) 使用RestTemplate进行服务调用



给 RestTemplate 实例添加 @LoadBalanced 注解，开启 @LoadBalanced 与 Ribbon 的集成

```
1 @Configuration
2 public class RestConfig {
3     @Bean
4     @LoadBalanced
5     public RestTemplate restTemplate() {
6         return new RestTemplate();
7     }
8 }
```

调用逻辑

```
1 #使用微服务名发起调用
2 String url = "http://mall-order/order/findOrderByUserId/"+id;
3 List<Order> orderList = restTemplate.getForObject(url, List.class);
```

测试：<http://localhost:8040/user/findOrderByUserId/1>，返回数据：

```

{
  msg: "success",
  code: 0,
  - orders: [
    - {
      id: 1,
      userId: 1,
      commodityCode: "000001",
      count: 10,
      amount: 1000
    },
    - {
      id: 2,
      userId: 1,
      commodityCode: "000002",
      count: 4,
      amount: 800
    }
  ]
}

```

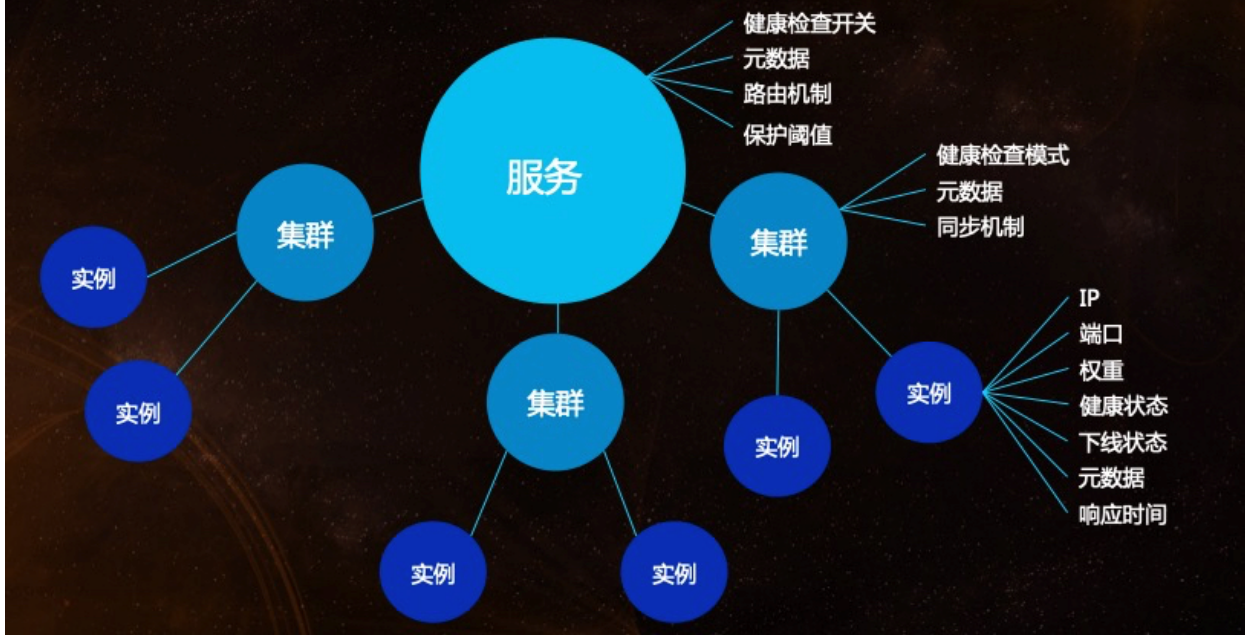
## 2.5 Nacos注册中心常见配置

### 服务分级存储模型

注册中心的核心数据是服务的名字和它对应的网络地址，当服务注册了多个实例时，我们需要对不健康的实例进行过滤或者针对实例的一些特征进行流量的分配，那么就需要在实例上存储一些例如健康状态、权重等属性。随着服务规模的扩大，渐渐的又需要在整个服务级别设定一些权限规则、以及对所有实例都生效的一些开关，于是在服务级别又会设立一些属性。再往后，我们又发现单个服务的实例又会有划分为多个子集的需求，例如一个服务是多机房部署的，那么可能需要对每个机房的实例做不同的配置，这样又需要在服务和实例之间再设定一个数据级别。

Nacos 在经过内部多年生产经验后提炼出的数据模型，则是一种服务-集群-实例的三层模型。这样基本可以满足服务在所有场景下的数据存储和管理。

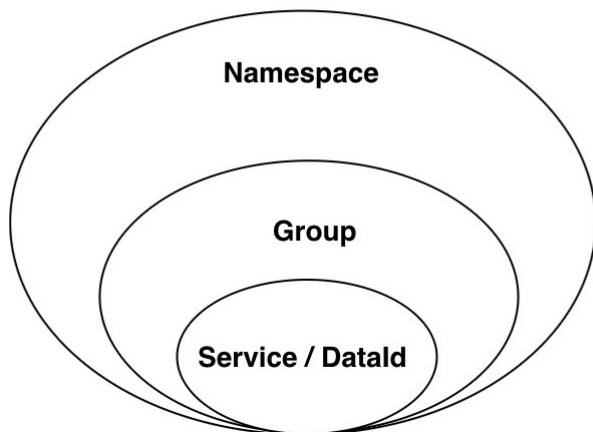
## Nacos服务的分级存储模型



### 服务逻辑隔离

Nacos 数据模型 Key 由三元组唯一确定, Namespace默认是空串, 公共命名空间 (public) , 分组默认是 DEFAULT\_GROUP。

#### Nacos data model



### Namespace 隔离设计

命名空间(Namespace)用于进行租户（用户）粒度的隔离，Namespace 的常用场景之一是不同环境的隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。

命名空间

新建命名空间

刷新

命名空间名称	命名空间ID	描述	配置数	操作
public(保留空间)			0	<a href="#">详情</a> <a href="#">删除</a> <a href="#">编辑</a>
dev	bc50d386-8870-4a26-8803-0187486c57be	开发环境	0	<a href="#">详情</a> <a href="#">删除</a> <a href="#">编辑</a>

## 修改yaml配置

```
1 spring:
2   application:
3     name: mall-user #微服务名称
4
5   cloud:
6     nacos:
7       discovery:
8         server-addr: 127.0.0.1:8848 #配置nacos注册中心地址
9         namespace: bc50d386-8870-4a26-8803-0187486c57be # dev 开发环境
```

启动mall-user，进入nacos控制台可以看到mall-user注册成功，所属namespace是dev



测试：http://localhost:8040/user/findOrderByUserId/1，报错

```
2022-06-29 20:43:15.639 ERROR 8532 --- [nio-8040-exec-1] o.a.c.c.C.[.][.].[dispatcherServlet]
java.lang.IllegalStateException Create breakpoint : No instances available for mall-order
    at org.springframework.cloud.netflix.ribbon.RibbonLoadBalancerClient.execute(RibbonLoadBalancerClient.execute)
    at org.springframework.cloud.netflix.ribbon.RibbonLoadBalancerClient.execute(RibbonLoadBalancerClient.execute)
    at org.springframework.cloud.client.loadbalancer.LoadBalancerInterceptor.intercept(LoadBalancerInterceptor.intercept)
```

原因：mall-order和mall-user使用了不同的namespace，导致服务隔离。

## group服务分组

不同的服务可以归类到同一分组，group也可以起到服务隔离的作用。yaml中可以通过spring.cloud.nacos.discovery.group参数配置

## 临时实例和持久化实例

在定义上区分临时实例和持久化实例的关键是健康检查的方式。临时实例使用客户端上报模式，而持久化实例使用服务端反向探测模式。临时实例需要能够自动摘除不健康实例，而且无需持久化存储实例。持久化实例使用服务端探测的健康检查方式，因为客户端不会上报心跳，所以不能自动摘除下线的实例。

在大中型的公司里，这两种类型的服务往往都有。一些基础的组件例如数据库、缓存等，这些往往不能上报心跳，这种类型的服务在注册时，就需要作为持久化实例注册。而上层的业务服务，例如微服务或者Dubbo服务，服务的Provider端支持添加汇报心跳的逻辑，此时就可以使用动态服务的注册方式。

Nacos 1.x 中持久化及非持久化的属性是作为实例的一个元数据进行存储和识别。

Nacos 2.x 中继续沿用了持久化及非持久化的设定，但是有了一些调整。在 Nacos2.0 中是否持久化的数据抽象至服务级别，且不再允许一个服务同时存在持久化实例和非持久化实例，实例的持久化属性继承自服务的持久化属性。

```
1 # 持久化实例
2 spring.cloud.nacos.discovery.ephemeral: false
```