Nacos简介及使用

2021-08-25

Nacos简介及使用

目录
CONTENTS

01

浅谈注册中心

1、如何构建一个注册中心?
2、注册中心中的分布式问题
3、主流的注册中心

MPOOM

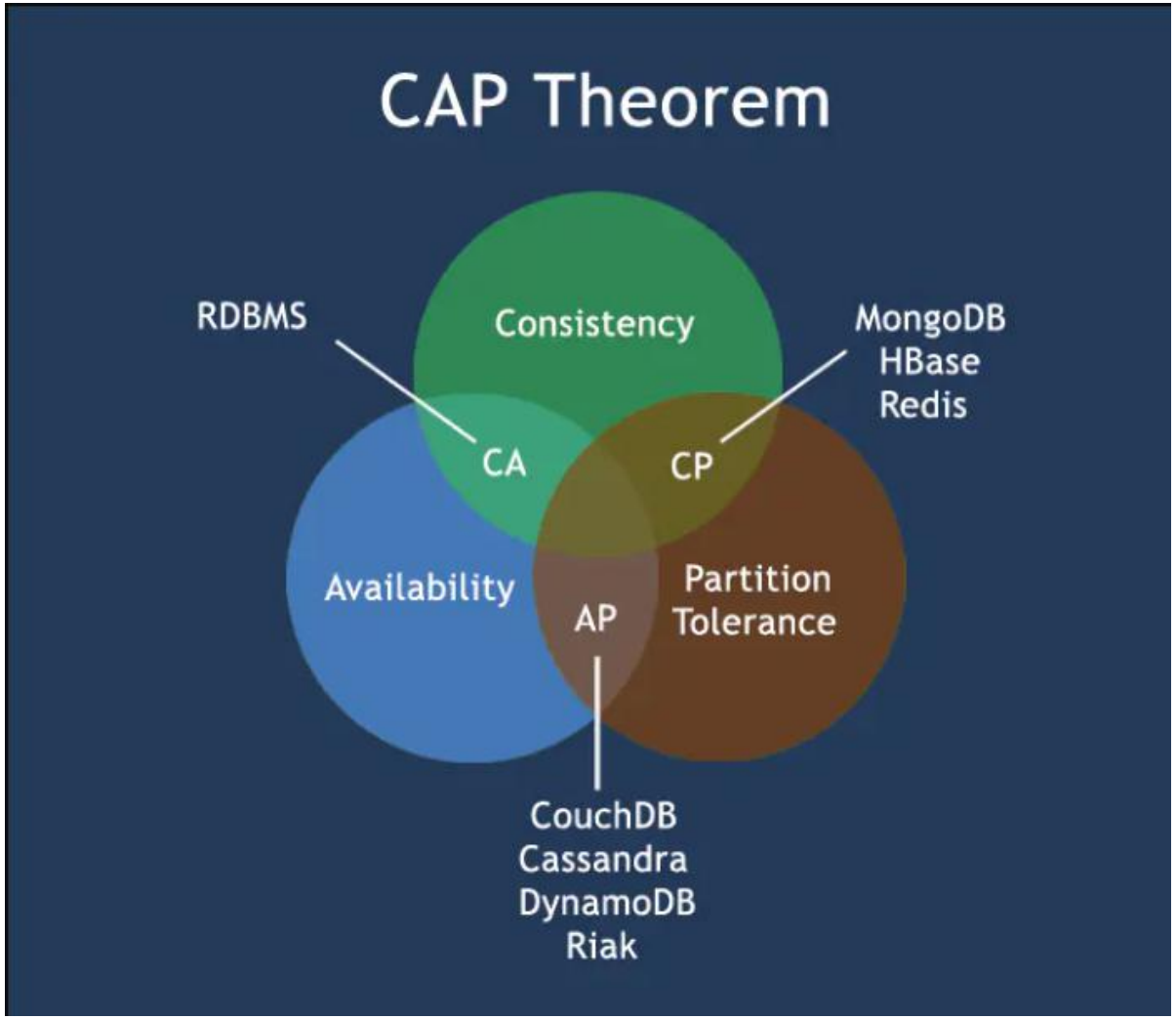
1.1 构建一个注册中心需要考虑哪些问题?

- 数据模型
- 数据一致性
- 负载均衡
- 健康检查
- 性能与容量
- 易用性
- 集群扩展性
- 用户扩展性
- 其它……

MPOOM

1.2 注册中心中的分布式问题

在分布式系统中，我们无法无法避开的一个问题就是CAP理论。



•C（Consistency）：这里指的是强一致性。保证在一定时间内，集群中的各个节点会达到较强的一致性，同时，为了达到这一点，一般会牺牲一点响应时间。而放弃C也不意味着放弃一致性，而是放弃强一致性。允许系统内有一定的数据不一致情况的存在。

•A (Avalibility)：可用性。意味着系统一直处于可用状态。个别节点的故障不会影响整个服务的运作。

•P（Partition Tolerance)：分区容忍性。当系统出现网络分区等情况时，依然能对外提供服务。想到达到这一点，一般来说会把数据复制到多个分区里，来提高分区容忍性。这个一般是不会被抛弃的。

MPOOM

CAP理论分析

一个分布式系统最多只能满足CAP中的两个条件，不可能同时满足三个条件。

据CAP定理可以分成满足CA原则、满足CP原则和满足AP原则的三大类:

- CA-单点集群，满足一致性可用性的系统，扩展能力不强
- CP-满足一致性和容错性系统，性能不高
- AP-满足可用性、容错性的系统，对一致性要求低一些。

MPOOM

1.3 主流的注册中心

市面上比较流行注册中心有：Zookeeper，Eureka，Consul，Nacos，CoreDNS

•Eureka

Spring Cloud Netflix 在设计 Eureka 时就紧遵AP原则,Eureka Server 也可以运行多个实例来构建集群，解决单点问题，但不同于 ZooKeeper 的选举 leader 的过程，Eureka Server 采用的是Peer to Peer 对等通信。这是一种去中心化的架构，无 master/slave 之分，每一个 Peer 都是对等的。在这种架构风格中，节点通过彼此互相注册来提高可用性，每个节点需要添加一个或多个有效的 serviceUrl 指向其他节点。每个节点都可被视为其他节点的副本。

在集群环境中如果某台 Eureka Server 宕机，Eureka Client 的请求会自动切换到新的 Eureka Server 节点上，当宕机的服务器重新恢复后，Eureka 会再次将其纳入到服务器集群管理之中。当节点开始接受客户端请求时，所有的操作都会在节点间进行复制（replicate To Peer）操作，将请求复制到该 Eureka Server 当前所知的其它所有节点中。

•Zookeeper

Apache Zookeeper 在设计时就紧遵CP原则，即任何时候对 Zookeeper 的访问请求能得到一致的数据结果，同时系统对网络分割具备容错性，但是 Zookeeper 不能保证每次服务请求都是可达的。从 Zookeeper 的实际应用情况来看，在使用 Zookeeper 获取服务列表时，如果此时的 Zookeeper 集群中的 Leader 宕机了，该集群就要进行 Leader 的选举，又或者 Zookeeper 集群中半数以上服务器节点不可用（例如有三个节点，如果节点一检测到节点三挂了，节点二也检测到节点三挂了，那这个节点才算是真的挂了），那么将无法处理该请求。所以说，Zookeeper 不能保证服务可用性。

MPOOM

主流的注册中心

•Consul

Consul 是 HashiCorp 公司推出的开源工具，用于实现分布式系统的服务发现与配置。Consul 使用 Go 语言编写，因此具有天然可移植性（支持Linux、windows和Mac OS X）。Consul 内置了服务注册与发现框架、分布一致性协议实现、健康检查、Key/Value 存储、多数据中心方案，不再需要依赖其他工具（比如 ZooKeeper 等），使用起来也较为简单。

Consul 遵循CAP原理中的CP原则，保证了强一致性和分区容错性，且使用的是Raft算法，比zookeeper使用的Paxos算法更加简单。虽然保证了强一致性，但是可用性就相应下降了，例如服务注册的时间会稍长一些，因为 Consul 的 raft 协议要求必须过半数的节点都写入成功才认为注册成功 ；在leader挂掉了之后，重新选举出leader之前会导致Consul 服务不可用。

•Nacos

Nacos是阿里开源的，Nacos 支持基于 DNS 和基于 RPC 的服务发现。Nacos除了服务的注册发现之外，还支持动态配置服务。动态配置服务可以让您以中心化、外部化和动态化的方式管理所有环境的应用配置和服务配置。动态配置消除了配置变更时重新部署应用和服务的需要，让配置管理变得更加高效和敏捷。配置中心化管理让实现无状态服务变得更简单，让服务按需弹性扩展变得更容易。 一句话概括就是Nacos（Naming Configuration Service） = 注册中心 + 配置中心。

•CoreDNS

CoreDNS is a DNS server. It is written in Go. It can be used in a multitude of environments because of its flexibility. CoreDNS is licensed under the Apache License Version 2, and completely open source.

MPOOM

注册中心对比

特性\注册中心	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

02

Nacos简介和使用

1、什么是Nacos?
2、Nacos的特性
3、Nacos的基本概念

MPOOM

2.1 什么是Nacos?

Nacos （Naming Configuration Service）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。

MPOOM

2.2 Nacos的特性

- 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

MPOOM

2.3 Nacos的基本概念

命名空间

用于进行租户粒度的配置隔离。不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置。Namespace 的常用场景之一是不同环境的配置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。

MPOOM

Nacos的基本概念

配置管理

系统配置的编辑、存储、分发、变更管理、历史版本管理、变更审计等所有与配置相关的活动。

配置集

一组相关或者不相关的配置项的集合称为配置集。在系统中，一个配置文件通常就是一个配置集，包含了系统各个方面的配置。例如，一个配置集可能包含了数据源、线程池、日志级别等配置项。

配置集 ID

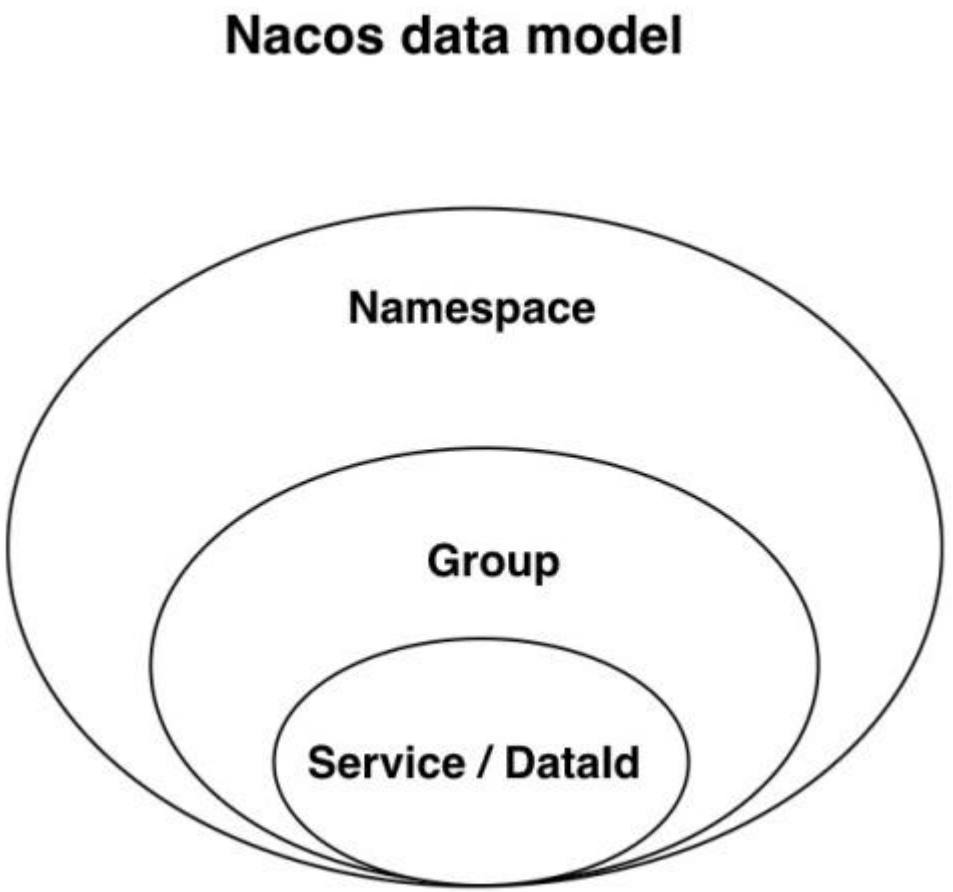
Nacos 中的某个配置集的 ID。配置集 ID 是组织划分配置的维度之一。Data ID 通常用于组织划分系统的配置集。一个系统或者应用可以包含多个配置集，每个配置集都可以被一个有意义的名称标识。Data ID 通常采用类 Java 包（如 com.taobao.tc.refund.log.level）的命名规则保证全局唯一性。此命名规则非强制。

配置分组

Nacos 中的一组配置集，是组织配置的维度之一。通过一个有意义的字符串（如 Buy 或 Trade ）对配置集进行分组，从而区分 Data ID 相同的配置集。当您在 Nacos 上创建一个配置时，如果未填写配置分组的名称，则配置分组的名称默认采用 DEFAULT_GROUP 。配置分组的常见场景：不同的应用或组件使用了相同的配置类型，如 database_url 配置和 MQ_topic 配置

MPOOM

Nacos的数据模型



MPOOM

Nacos的命名空间、配置集、配置分组示例

下图为在“develop”命名空间下创建了两个属于不同配置分组但配置集ID相同的RocketMQ的Topic配置集。

03

Nacos的部署使用

1、Nacos部署
2、Nacos特性

MPOOM

3.1 Nacos的部署

部署环境

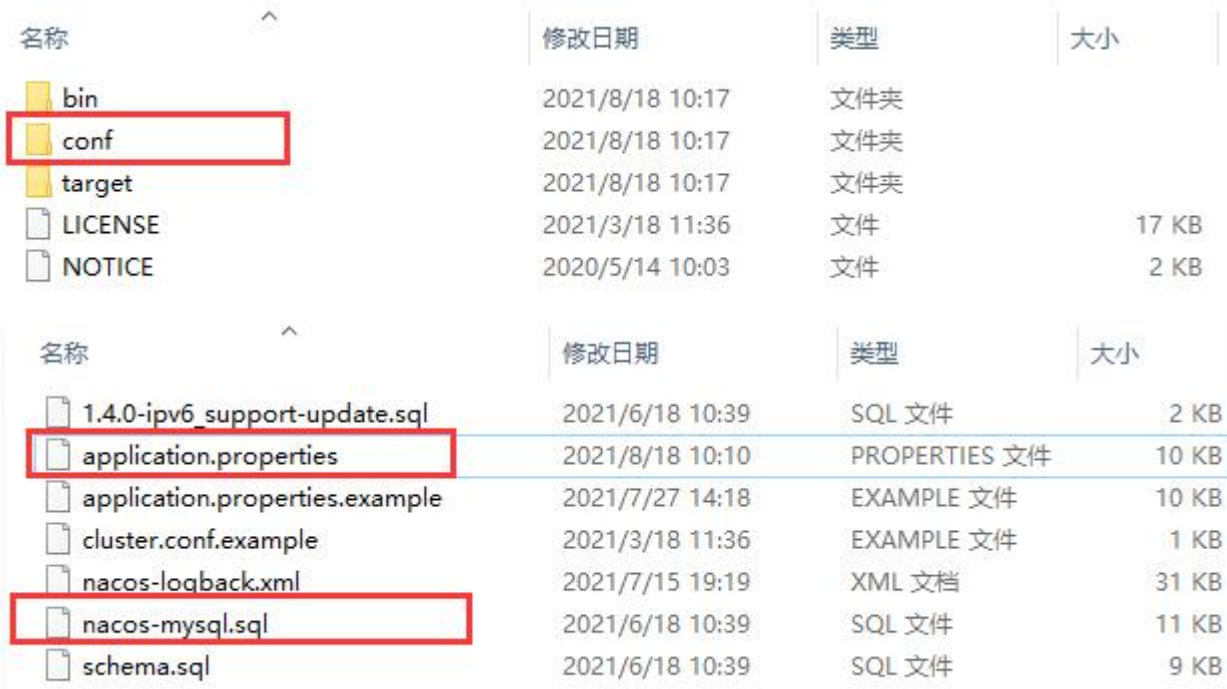
Nacos定义为一个IDC内部应用组件，并非面向公网环境的产品，建议在内部隔离网络环境中部署，强烈不建议部署在公共网络环境。

部署模式

- 单机模式 - 用于测试和单机试用。
- 集群模式 - 用于生产环境，确保高可用。
- 多集群模式 - 用于多数据中心场景。

MPOOM

3.1.1 基于MySQL部署启动



名称	修改日期	类型	大小
bin	2021/8/18 10:17	文件夹	
conf	2021/8/18 10:17	文件夹	
target	2021/8/18 10:17	文件夹	
LICENSE	2021/3/18 11:36	文件	17 KB
NOTICE	2020/5/14 10:03	文件	2 KB

名称	修改日期	类型	大小
1.4.0-ipv6_support-update.sql	2021/6/18 10:39	SQL 文件	2 KB
application.properties	2021/8/18 10:10	PROPERTIES 文件	10 KB
application.properties.example	2021/7/27 14:18	EXAMPLE 文件	10 KB
cluster.conf.example	2021/3/18 11:36	EXAMPLE 文件	1 KB
nacos-logback.xml	2021/7/15 19:19	XML 文档	31 KB
nacos-mysql.sql	2021/6/18 10:39	SQL 文件	11 KB
schema.sql	2021/6/18 10:39	SQL 文件	9 KB

```
30
31  #*************** Config Module Related Configurations ***************#
32  ### If use MySQL as datasource:
33  spring.datasource.platform=mysql
34
35  ### Count of DB:
36  db.num=1
37
38  ### Connect URL of DB:
39  db.url.0=jdbc:mysql://127.0.0.1:3306/nacos?characterEncoding=utf8&connectTimeout=1000&s
40  db.user.0=root
41  db.password.0=123456
42
```

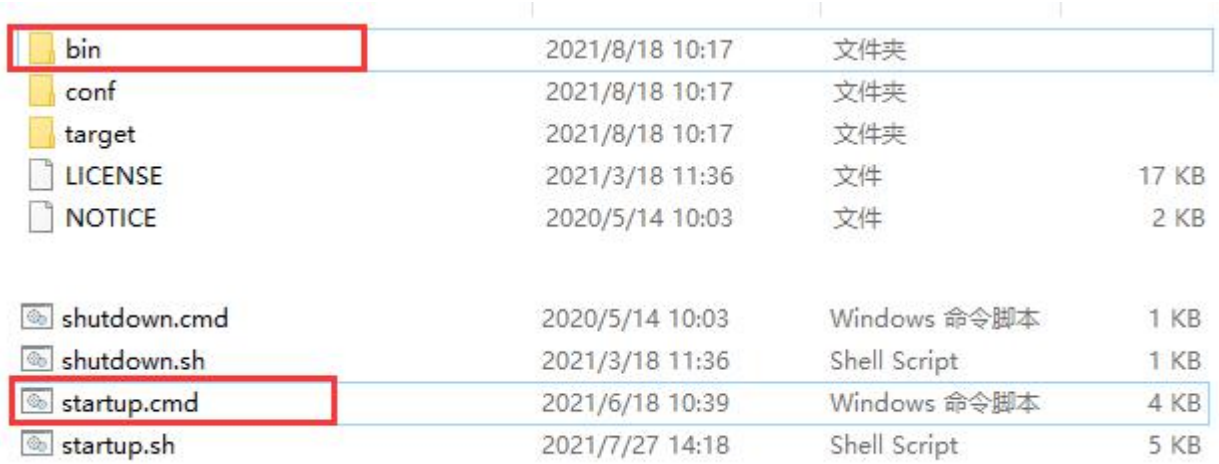
（1）创建Nacos的MySQL数据库实例，导入nacos-msyql.sql数据库脚本。

（2）修改配置文件，将数据源配置为MySQL数据库

（3）在0.7版本之前，在单机模式时nacos使用嵌入式数据库Derby实现数据的存储，“schema.sql”为Deryb数据库脚本。

MPOOM

3.1.2 单机模式启动



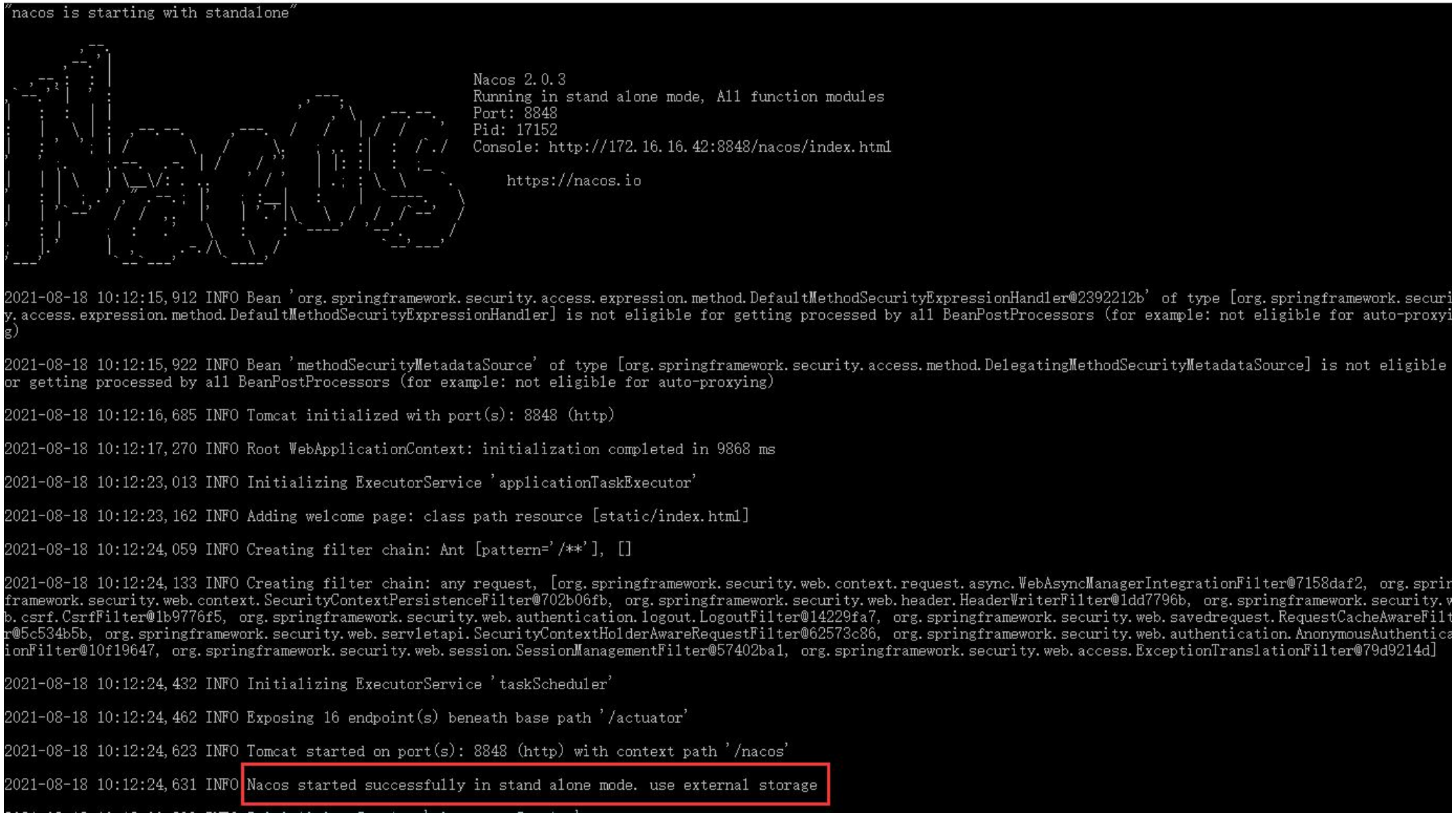
（1）修改启动脚本，使用单机模式“standalone”启动

（2）或者使用命令行参数 –m standalone以单机模式启动。

```
14  if not exist "%JAVA_HOME%\bin\java.exe" echo Please set the JAVA_HOME variable in your environment,
15  set "JAVA=%JAVA_HOME%\bin\java.exe"
16
17  setlocal enabledelayedexpansion
18
19  set BASE_DIR=%~dp0
20  rem added double quotation marks to avoid the issue caused by the folder names containing spaces.
21  rem removed the last 5 chars(which means \bin\) to get the base DIR.
22  set BASE_DIR="%BASE_DIR:~0,-5%"
23
24  set CUSTOM_SEARCH_LOCATIONS=file:%BASE_DIR%/conf/
25
26  set MODE="cluster"
27  set FUNCTION_MODE="all"
28  set SERVER=nacos-server
29  set MODE_INDEX=-1
30  set FUNCTION_MODE_INDEX=-1
31  set SERVER_INDEX=-1
32  set EMBEDDED_STORAGE_INDEX=-1
33  set EMBEDDED_STORAGE=""
```

MPOOM

在Windows下单机模式启动



MPOOM

3.1.3 集群模式启动

添加集群配置文件，增加集群IP列表

```
total 92
-rw-r--r-- 1 deploy deploy  1224 Jun 18 10:39 1.4.0-ipv6_support-update.sql
-rw-r--r-- 1 deploy deploy  9500 Aug 18 14:35 application.properties
-rw-r--r-- 1 deploy deploy  9506 Jul 27 14:18 application.properties.example
-rw-r--r-- 1 deploy deploy   665 Aug 18 14:35 cluster.conf
-rw-r--r-- 1 deploy deploy   670 Mar 18 11:36 cluster.conf.example
-rw-r--r-- 1 deploy deploy 31156 Jul 15 19:19 nacos-logback.xml
-rw-r--r-- 1 deploy deploy 10660 Jun 18 10:39 nacos-mysql.sql
-rw-r--r-- 1 deploy deploy  8795 Jun 18 10:39 schema.sql
```

```
#it is ip
#example
127.0.0.1:8845
127.0.0.1:8846
127.0.0.1:8847
```

```
Nacos 2.0.3
Running in cluster mode, All function modules
Port: 8845
Pid: 717426
Console: http://10.13.149.183:8845/nacos/index.html

https://nacos.io

2021-08-18 14:40:59,492 INFO The server IP list of Nacos is [127.0.0.1:8845, 127.0.0.1:8846, 127.0.0.1:8847]

2021-08-18 14:41:00,496 INFO Nacos is starting...

2021-08-18 14:41:01,507 INFO Nacos is starting...
```

```
2021-08-18 14:41:25,653 INFO Nacos is starting...

2021-08-18 14:41:26,657 INFO Nacos is starting...

2021-08-18 14:41:26,945 INFO Nacos started successfully in cluster mode. use external storage
```

MPOOM

Fgmp-server中Nacos的配置

```yaml
# 使用nacos配置中心，配置文件是：boostrap.properties或者bootstrap.yml
spring:
  application:
    name: fgmp-server-frontkit
  cloud:
    nacos:
      config:
        server-addr: 172.16.1.15:8848
        file-extension: yml #配置文件拓展名，默认是properties
  profiles:
    active: dev # 开发环境，nacos支持多环境配置：dev/test/prod
config:
  appName: server-frontkit
```

MPOOM

spring cloud配置文件加载接口

```java
package org.springframework.cloud.bootstrap.config;

import org.springframework.core.env.Environment;
import org.springframework.core.env.PropertySource;

/**
 * Strategy for locating (possibly remote) property sources for the Environment.
 * Implementations should not fail unless they intend to prevent the application from
 * starting.
 *
 * @author Dave Syer
 *
 */
public interface PropertySourceLocator {

   /**
    * @param environment The current Environment.
    * @return A PropertySource, or null if there is none.
    * @throws IllegalStateException if there is a fail-fast condition.
    */
   PropertySource<?> locate(Environment environment);

}
```

MPOOM

Nacos配置文件实现原理

```
@Override
public PropertySource<?> locate(Environment env) {
   nacosConfigProperties.setEnvironment(env);
   ConfigService configService = nacosConfigManager.getConfigService();

   nacosPropertySourceBuilder = new NacosPropertySourceBuilder(configService,
        nacosConfigProperties.getTimeout());

   String dataIdPrefix = nacosConfigProperties.getPrefix();
   if (StringUtils.isEmpty(dataIdPrefix)) {
      dataIdPrefix = nacosConfigProperties.getName();
   }
   if (StringUtils.isEmpty(dataIdPrefix)) {
      dataIdPrefix = env.getProperty("spring.application.name");
   }

   CompositePropertySource composite = new CompositePropertySource(NACOS_PROPERTY_SOURCE_NAME);

   loadSharedConfiguration(composite);
   loadExtConfiguration(composite);
   loadApplicationConfiguration(composite, dataIdPrefix, nacosConfigProperties, env);
   return composite;
}
```

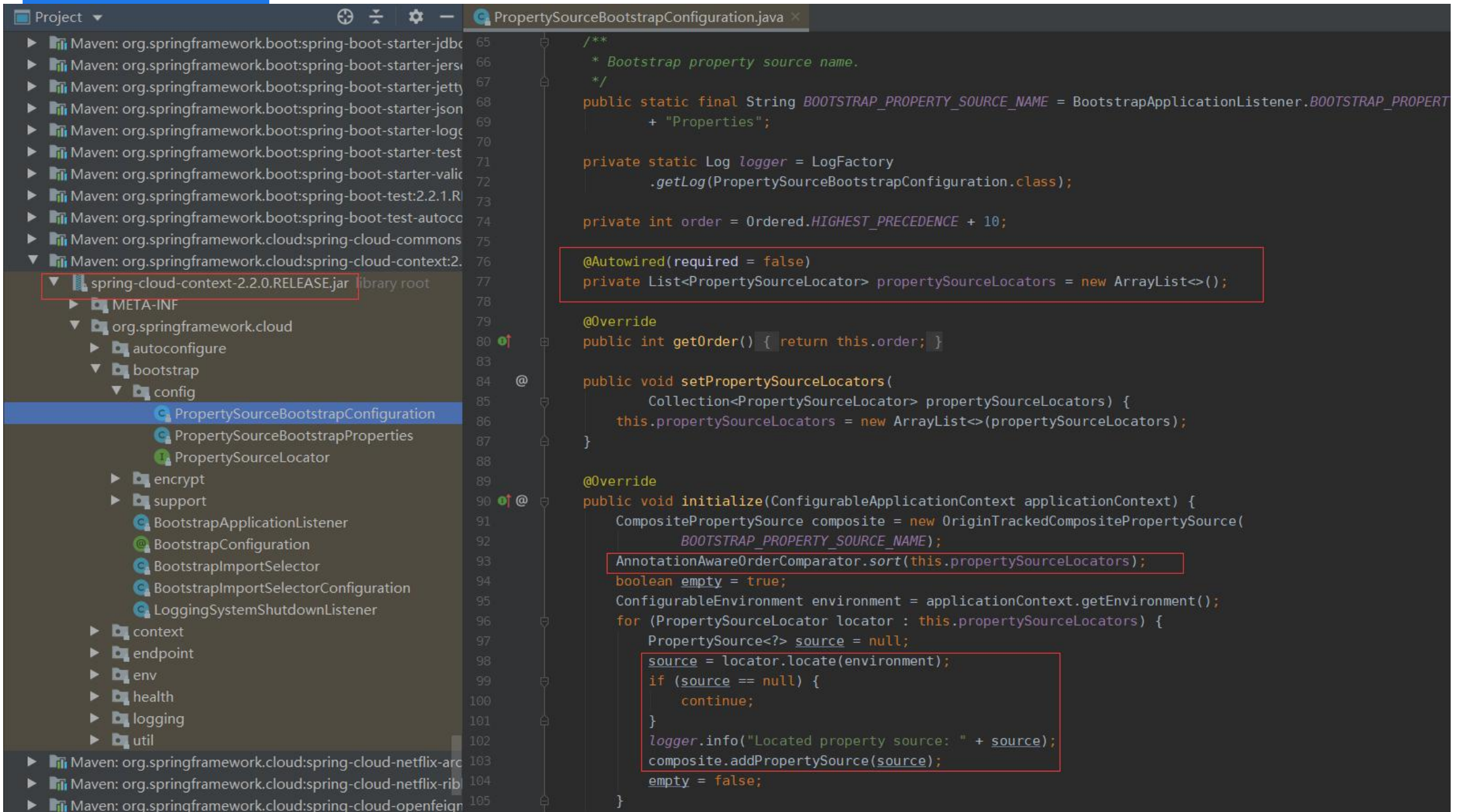
MPOOM

应用程序配置文件加载顺序

```
private void loadApplicationConfiguration(
      CompositePropertySource compositePropertySource, String dataIdPrefix,
      NacosConfigProperties properties, Environment environment) {

   String fileExtension = properties.getFileExtension();
   String nacosGroup = properties.getGroup();

   // load directly once by default
   loadNacosDataIfPresent(compositePropertySource, dataIdPrefix, nacosGroup,
         fileExtension, true);

   // load with suffix, which have a higher priority than the default
   loadNacosDataIfPresent(compositePropertySource,
         dataIdPrefix + DOT + fileExtension, nacosGroup, fileExtension, true);

   // Loaded with profile, which have a higher priority than the suffix
   for (String profile : environment.getActiveProfiles()) {
      String dataId = dataIdPrefix + SEP1 + profile + DOT + fileExtension;
      loadNacosDataIfPresent(compositePropertySource, dataId, nacosGroup,
            fileExtension, true);
   }

}
```

MPOOM

spring cloud locate执行的时机



MPOOM

spring cloud配置文件格式

在 Nacos Spring Cloud 中，dataId 的完整格式如下：

${prefix}-${spring.profiles.active}.${file-extension}

04

Nacos实现原理

1、Nacos的数据一致性
2、Nacos的负载均衡
3、Nacos的高可用实现

MPOOM

4.1 Nacos的数据一致性

数据一致性是分布式系统永恒的话题，Paxos 协议的艰深更让数据一致性成为程序员大牛们吹水的常见话题。

目前来看分布式协议基本可以分为两类：

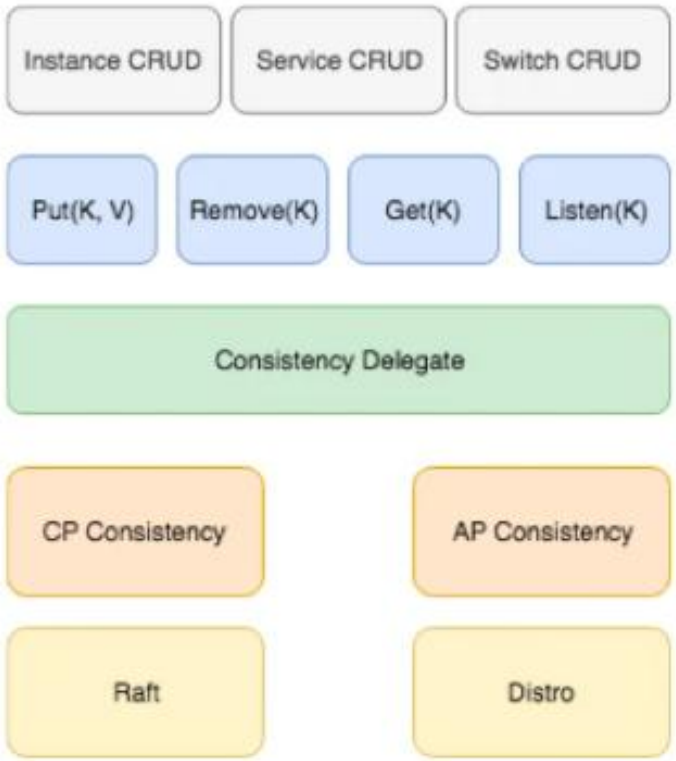
- 一种是基于 Leader 的非对等部署的单点写一致性
- 一种是对等部署的多写一致性

Nacos目前的一致性协议实现，一个是基于简化的 Raft 的 CP 一致性，一个是基于自研协议 Distro 的 AP 一致性。

Raft 协议不必多言，基于 Leader 进行写入，其 CP 也并不是严格的，只是能保证一半所见一致，以及数据的丢失概率较小。

Distro 协议则是参考了内部 ConfigServer 和开源 Eureka，在不借助第三方存储的情况下，实现基本大同小异。Distro 重点是做了一些逻辑的优化和性能的调优。

MPOOM



- Bussiness CRUD does not feel consistency type
- Abstracted datum CRUD
- Use a delegate to determine which way to go
- Two paths to choose: CP and AP
- Raft acting as CP, Distro acting as AP

Nacos的架构图

MPOOM

4.2 Nacos的负载均衡

服务消费者往往并不关心所访问的服务提供者的负载均衡，它们只关心以最高效和正确的访问服务提供者的服务。

而**服务提供者**，则非常关注自身被访问的流量的调配，这其中的第一个原因是，阿里巴巴集团内部服务访问流量巨大，稍有不慎就会导致流量异常压垮服务提供者的服务。因此服务提供者需要能够完全掌控服务的流量调配，并可以动态调整。

MPOOM

服务端负载均衡

服务端的负载均衡，给服务提供者更强的流量控制权，但是无法满足不同的消费者希望使用不同负载均衡策略的需求。而不同负载均衡策略的场景，确实是存在的。

而客户端的负载均衡则提供了这种灵活性，并对用户扩展提供更加友好的支持。但是客户端负载均衡策略如果配置不当，可能会导致服务提供者出现热点，或者压根就拿不到任何服务提供者。

MPOOM

客户端负载均衡

抛开负载均衡到底是在服务提供者实现还是在服务消费者实现，我们看到目前的负载均衡有基于权重、服务提供者负载、响应时间、标签等策略。

其中 Ribbon 设计的客户端负载均衡机制，主要是选择合适现有的 IRule、ServerListFilter 等接口实现，或者自己继承这些接口，实现自己的过滤逻辑。

这里 Ribbon 采用的是**两步负载均衡**

- 第一步是先过滤掉不会采用的服务提供者实例
- 第二步是在过滤后的服务提供者实例里，实施负载均衡策略。

Ribbon 内置的几种负载均衡策略功能还是比较强大的，同时又因为允许用户去扩展，这可以说是一种比较好的设计。

MPOOM

Nacos的负载均衡设计

负载均衡是一个很大的话题，当我们在关注注册中心提供的负载均衡策略时，需要注意该注册中心是否有我需要的**负载均衡方式，使用方式是否复杂**。如果没有，那么是否允许我**方便的扩展来**实现我需求的负载均衡策略。

Nacos 试图做的是将**服务端负载均衡**与**客户端负载均衡**通过某种机制结合起来，提供用户扩展性，并给予用户充分的自主选择权和轻便的使用方式。

MPOOM

4.3 Nacos高可用原理

Nacos 的高可用不仅仅存在于服务端，同时它也存在于客户端，以及一些与可用性相关的功能特性中。这些点组装起来，共同构成了 Nacos 的高可用。

客户端重试

当其中一台机器宕机时，为了不影响整体运行，客户端会存在重试机制。

逻辑非常简单，拿到地址列表，在请求成功之前逐个尝试，直到成功为止。

该可用性保证存在于 nacos-client 端。

MPOOM

一致性协议 distro

Nacos中的临时服务和持久化服务

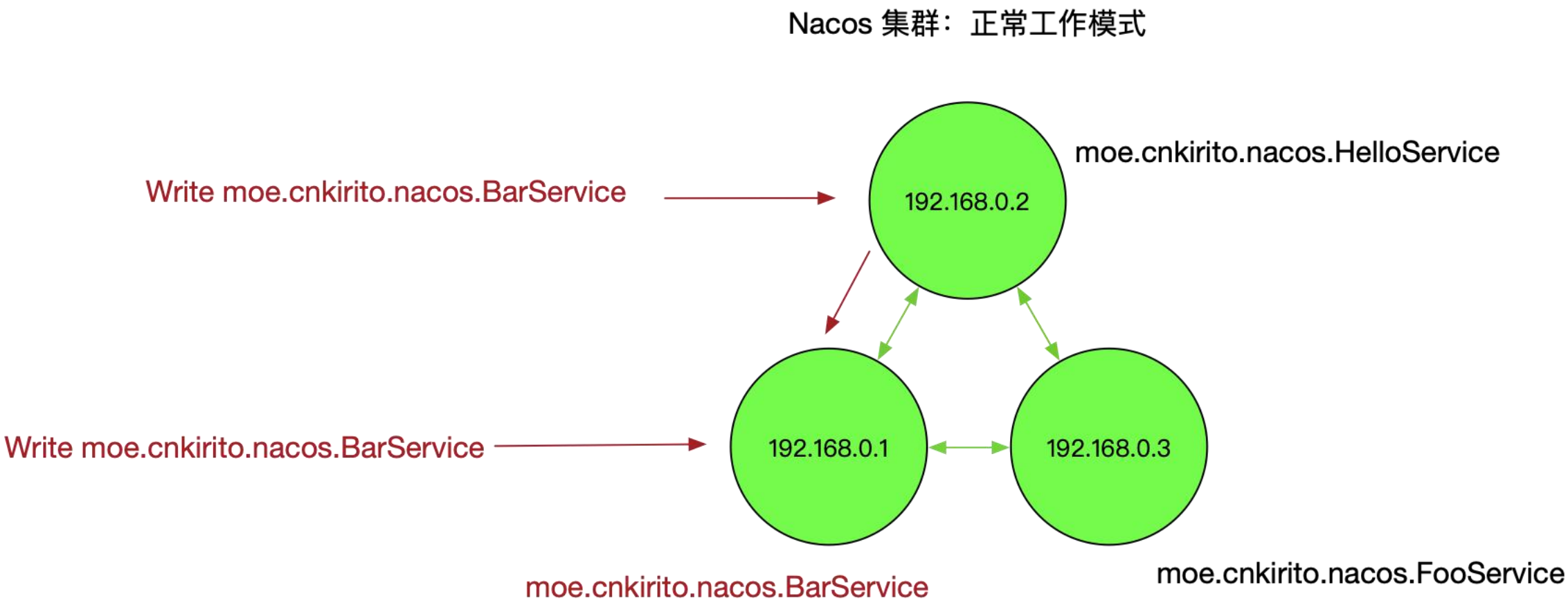
- 临时服务（Ephemeral）：临时服务健康检查失败后会从列表中删除，常用于服务注册发现场景。
- 持久化服务（Persistent）：持久化服务健康检查失败后会被标记成不健康，常用于 DNS 场景。

临时服务使用的是 Nacos 为服务注册发现场景定制化的私有**协议 distro**，其一致性模型是 **AP**；
而**持久化服务**使用的是 **raft 协议**，其一致性模型是 **CP**。
所以以后不要再说 Nacos 是 AP + CP 了，更建议加上服务节点状态或者使用场景的约束。

MPOOM

distro 协议的工作流程

- Nacos 启动时首先从其他远程节点同步全部数据
- Nacos 每个节点是平等的都可以处理写入请求，同时把新数据同步到其他节点
- 每个节点只负责部分数据，定时发送自己负责数据校验值到其他节点来保持数据一致性



MPOOM

Nacos集群正常工作模式

如上图所示，每个节点服务一部分服务的写入，但每个节点都可以接收到写入请求，这时就存在两种写情况：

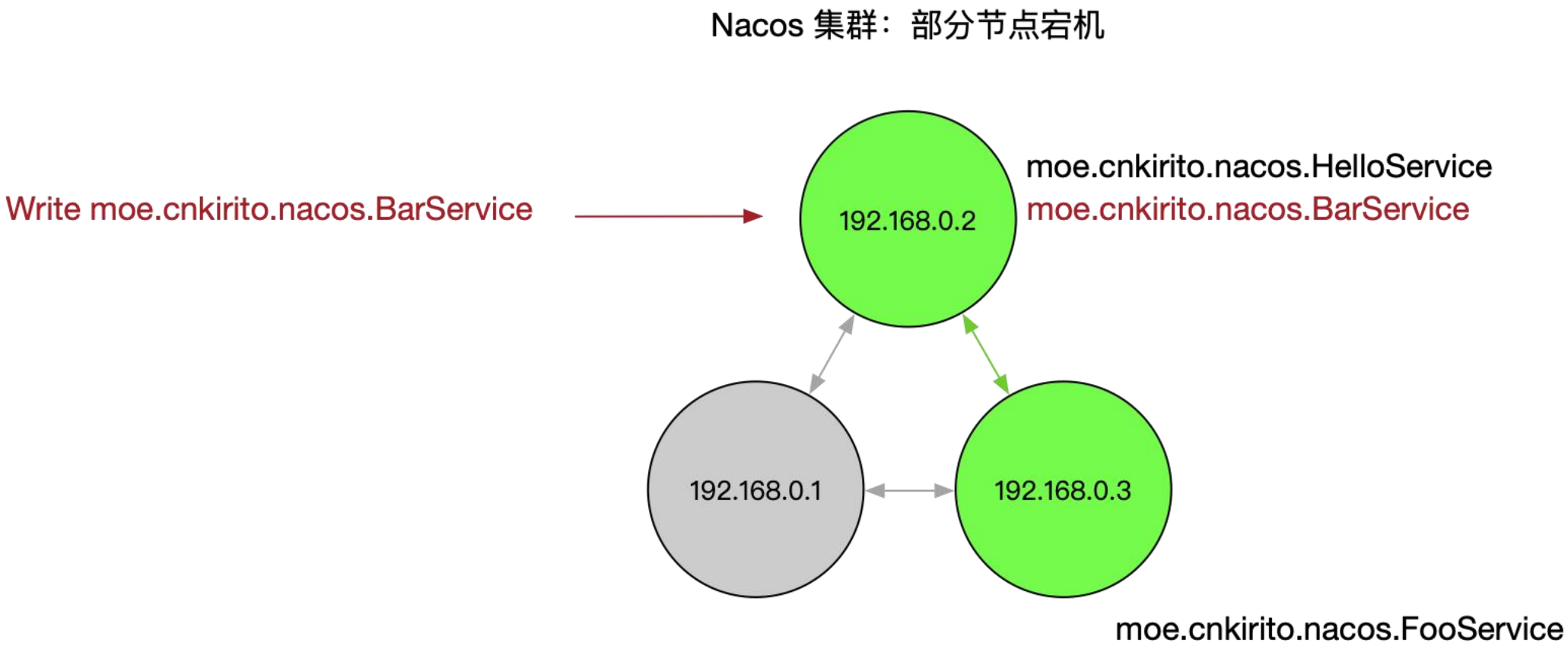
1. 当该节点接收到属于该节点负责的服务时，直接写入。

2. 当该节点接收到不属于该节点负责的服务时，将在集群内部路由，转发给对应的节点，从而完成写入。

读取操作则不需要路由，因为集群中的各个节点会同步服务状态，每个节点都会有一份最新的服务数据。

MPOOM

Nacos集群部分节点宕机



而当节点发生宕机后，原本该节点负责的一部分服务的写入任务会转移到其他节点，从而保证 Nacos 集群整体的可用性。

MPOOM

本地缓存文件 Failover 机制

注册中心发生故障最坏的一个情况是整个 Server 端宕机，这时候 Nacos 依旧有高可用机制做兜底。

Nacos 存在本地文件缓存机制，nacos-client 在接收到 nacos-server 的服务推送之后，会在内存中保存一份，随后会落盘存储一份快照。snapshot 默认的存储路径为：{USER_HOME}/nacos/naming/

这份文件有两种价值，一是用来排查服务端是否正常推送了服务；二是当客户端加载服务时，如果无法从服务端拉取到数据，会默认从本地文件中加载。

MPOOM

心跳同步服务

心跳机制一般广泛存在于分布式通信领域，用于确认存活状态。一般心跳请求和普通请求的设计是有差异的，心跳请求一般被设计的足够精简，这样在定时探测时可以尽可能避免性能下降。而在 Nacos 中，处于可用性的考虑，一个心跳报文包含了全部的服务信息，这样相比仅仅发送探测信息降低了吞吐量，而提升了可用性，怎么理解呢？

考虑以下的两种场景：

- nacos-server 节点全部宕机，服务数据全部丢失。nacos-server 即使恢复运作，也无法恢复出服务，而心跳包含全部内容可以在心跳期间就恢复出服务，保证可用性。
- nacos-server 出现网络分区。由于心跳可以创建服务，从而在极端网络故障下，依旧保证基础的可用性

MPOOM

集群部署模式高可用

来自于Nacos部署架构的高可用特性

节点数量

在生产集群中肯定不能以单机模式运行 Nacos，那么第一个问题便是：我应该部署几台机器？前面我们提到 Nacos 有两个一致性协议：distro 和 raft，distro 协议不会有脑裂问题，所以理论来说，节点数大于等于 2 即可；raft 协议的投票选举机制则建议是 2n+1 个节点。

综合来看，选择 3 个节点是起码的，其次处于吞吐量和更高可用性的考量，可以选择 5 个，7 个，甚至 9 个节点的集群。

多可用区部署

组成集群的 Nacos 节点，应该尽可能考虑两个因素：

1.各个节点之间的网络时延不能很高，否则会影响数据同步

2.各个节点所处机房、可用区应当尽可能分散，以避免单点故障

谢谢观看!