目录

**[Nacos简介](#_Toc18718_WPSOffice_Level1)** **[1](#_Toc18718_WPSOffice_Level1)**

[服务 (Service)](#_Toc15755_WPSOffice_Level2) [4](#_Toc15755_WPSOffice_Level2)

[服务注册中心 (Service Registry)](#_Toc24687_WPSOffice_Level2) [4](#_Toc24687_WPSOffice_Level2)

[服务元数据 (Service Metadata)](#_Toc14493_WPSOffice_Level2) [4](#_Toc14493_WPSOffice_Level2)

[服务提供方 (Service Provider)](#_Toc32625_WPSOffice_Level2) [4](#_Toc32625_WPSOffice_Level2)

[服务消费方 (Service Consumer)](#_Toc18191_WPSOffice_Level2) [4](#_Toc18191_WPSOffice_Level2)

[配置 (Configuration)](#_Toc15702_WPSOffice_Level2) [4](#_Toc15702_WPSOffice_Level2)

[配置管理 (Configuration Management)](#_Toc20168_WPSOffice_Level2) [5](#_Toc20168_WPSOffice_Level2)

[名字服务 (Naming Service)](#_Toc27280_WPSOffice_Level2) [5](#_Toc27280_WPSOffice_Level2)

[配置服务 (Configuration Service)](#_Toc24593_WPSOffice_Level2) [5](#_Toc24593_WPSOffice_Level2)

**[Nacos控制台](#_Toc15755_WPSOffice_Level1)** **[6](#_Toc15755_WPSOffice_Level1)**

**[Nacos Open API](#_Toc24687_WPSOffice_Level1)** **[7](#_Toc24687_WPSOffice_Level1)**

**[Nacos配置中心](#_Toc14493_WPSOffice_Level1)** **[7](#_Toc14493_WPSOffice_Level1)**

[长轮询源码](#_Toc14702_WPSOffice_Level2) [9](#_Toc14702_WPSOffice_Level2)

**[Nacos服务发现](#_Toc32625_WPSOffice_Level1)** **[14](#_Toc32625_WPSOffice_Level1)**

[服务注册客户端](#_Toc613_WPSOffice_Level2) [14](#_Toc613_WPSOffice_Level2)

[服务注册服务端](#_Toc24650_WPSOffice_Level2) [17](#_Toc24650_WPSOffice_Level2)

[一致性算法](#_Toc13424_WPSOffice_Level2) [19](#_Toc13424_WPSOffice_Level2)

[1.选取主节点](#_Toc26091_WPSOffice_Level2) [20](#_Toc26091_WPSOffice_Level2)

[2.同步数据](#_Toc8554_WPSOffice_Level2) [20](#_Toc8554_WPSOffice_Level2)

[Leader选举](#_Toc12812_WPSOffice_Level2) [20](#_Toc12812_WPSOffice_Level2)

[同步数据](#_Toc24054_WPSOffice_Level2) [21](#_Toc24054_WPSOffice_Level2)

[Raft心跳保持和续约](#_Toc2873_WPSOffice_Level2) [22](#_Toc2873_WPSOffice_Level2)

[健康检查](#_Toc20765_WPSOffice_Level2) [22](#_Toc20765_WPSOffice_Level2)

**[Nacos优缺点](#_Toc18191_WPSOffice_Level1)** **[23](#_Toc18191_WPSOffice_Level1)**

**[Nacos发展及规划](#_Toc15702_WPSOffice_Level1)** **[24](#_Toc15702_WPSOffice_Level1)**

**[1.0.0版本：主要目标有两个：](#_Toc20168_WPSOffice_Level1)** **[24](#_Toc20168_WPSOffice_Level1)**

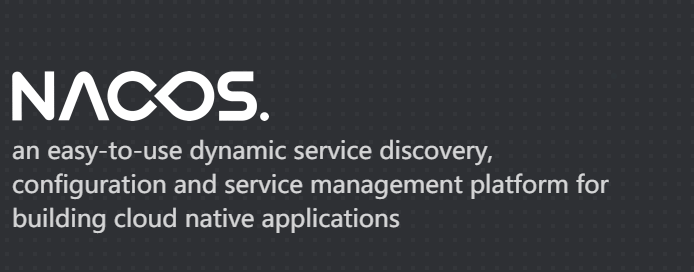
**[2.0版本：](#_Toc27280_WPSOffice_Level1)** **[24](#_Toc27280_WPSOffice_Level1)**

**[Nacos部署和运维](#_Toc24593_WPSOffice_Level1)** **[25](#_Toc24593_WPSOffice_Level1)**

# Nacos简介

Nacos 提供了一组简单易用的特性集，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 帮助您更敏捷和容易地构建、交付和管理微服务平台。 Nacos 是构建以“服务”为中心的现代应用架构 (例如微服务范式、云原生范式) 的服务基础设施。



Nacos关键特性：

* 服务发现和服务健康监测

Nacos 支持基于RPC 的服务发现。服务提供者使用 原生SDK、OpenAPI、或一个独立的Agent TODO注册 Service 后，服务消费者可以使用HTTP&API查找和发现服务。

Nacos 提供对服务的实时的健康检查，阻止向不健康的主机或服务实例发送请求。Nacos 支持传输层 (PING 或 TCP)和应用层 (如 HTTP、MySQL、用户自定义）的健康检查。 对于复杂的云环境和网络拓扑环境中（如 VPC、边缘网络等）服务的健康检查，Nacos 提供了 agent 上报模式和服务端主动检测2种健康检查模式。Nacos 还提供了统一的健康检查仪表盘，帮助您根据健康状态管理服务的可用性及流量。

* 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。

Nacos 提供了一个简洁易用的UI (控制台样例 Demo) 帮助您管理所有的服务和应用的配置。Nacos 还提供包括配置版本跟踪、金丝雀发布、一键回滚配置以及客户端配置更新状态跟踪在内的一系列开箱即用的配置管理特性，帮助您更安全地在生产环境中管理配置变更和降低配置变更带来的风险。

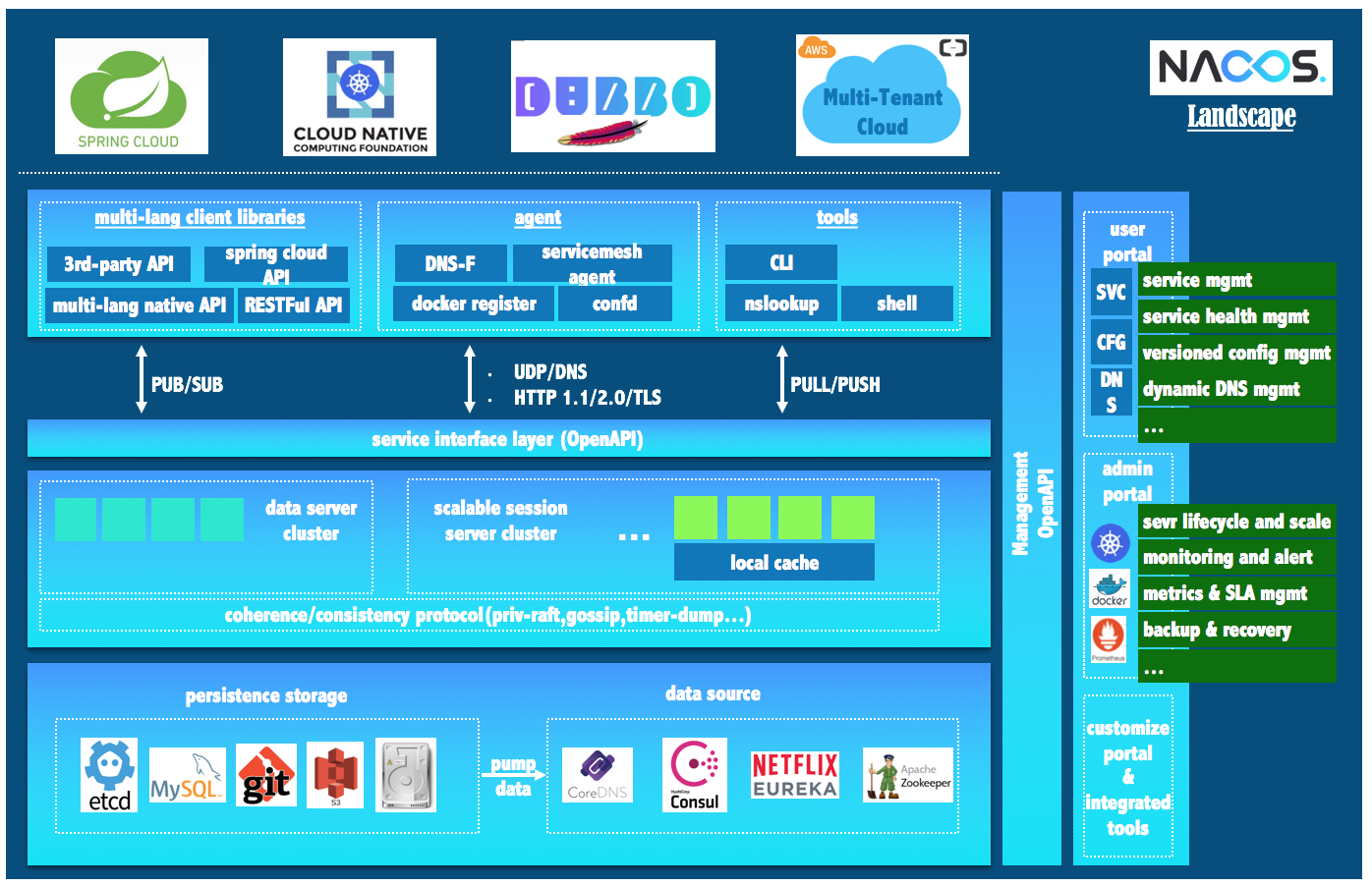
* 服务及其元数据管理

Nacos 能让您从微服务平台建设的视角管理数据中心的所有服务及元数据，包括管理服务的描述、生命周期、服务的静态依赖分析、服务的健康状态、服务的流量管理、路由及安全策略、服务的 SLA 以及最首要的 metrics 统计数据。

nacos地图



nacos生态图



[what-is-nacos](https://nacos.io/zh-cn/docs/what-is-nacos.html)

[concepts](https://nacos.io/zh-cn/docs/concepts.html)

****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，其实例及元数据的数据库。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

****配置 (Configuration)****

在系统开发过程中通常会将一些需要变更的参数、变量等从代码中分离出来独立管理，以独立的配置文件的形式存在。目的是让静态的系统工件或者交付物（如 WAR，JAR 包等）更好地和实际的物理运行环境进行适配。配置管理一般包含在系统部署的过程中，由系统管理员或者运维人员完成这个步骤。配置变更是调整系统运行时的行为的有效手段之一。

****配置管理 (Configuration Management)****

在数据中心中，系统中所有配置的编辑、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统称为配置管理。

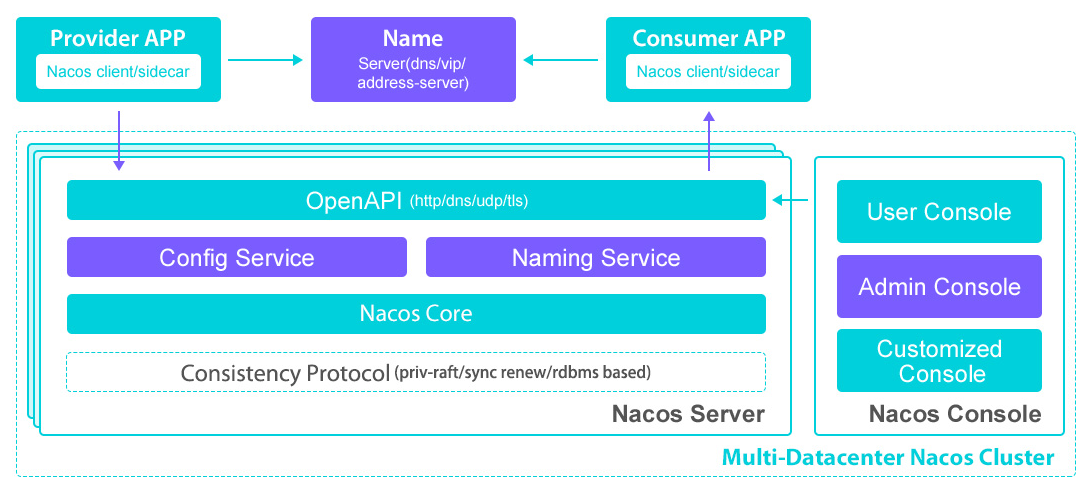
****名字服务 (Naming Service)****

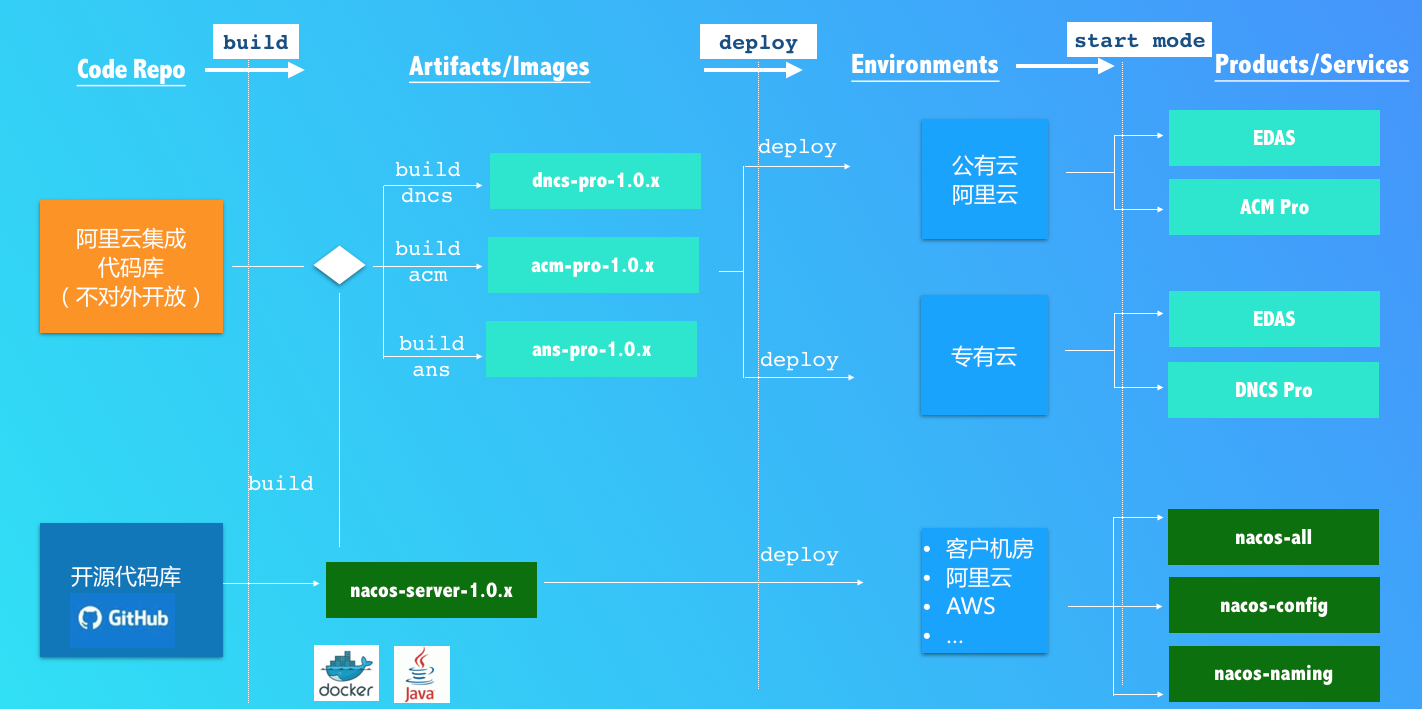
提供分布式系统中所有对象(Object)、实体(Entity)的“名字”到关联的元数据之间的映射管理服务，例如 ServiceName -> Endpoints Info, Distributed Lock Name -> Lock Owner/Status Info, DNS Domain Name -> IP List, 服务发现和 DNS 就是名字服务的2大场景。

****配置服务 (Configuration Service)****

在服务或者应用运行过程中，提供动态配置或者元数据以及配置管理的服务提供者。

[architecture](https://nacos.io/zh-cn/docs/architecture.html)





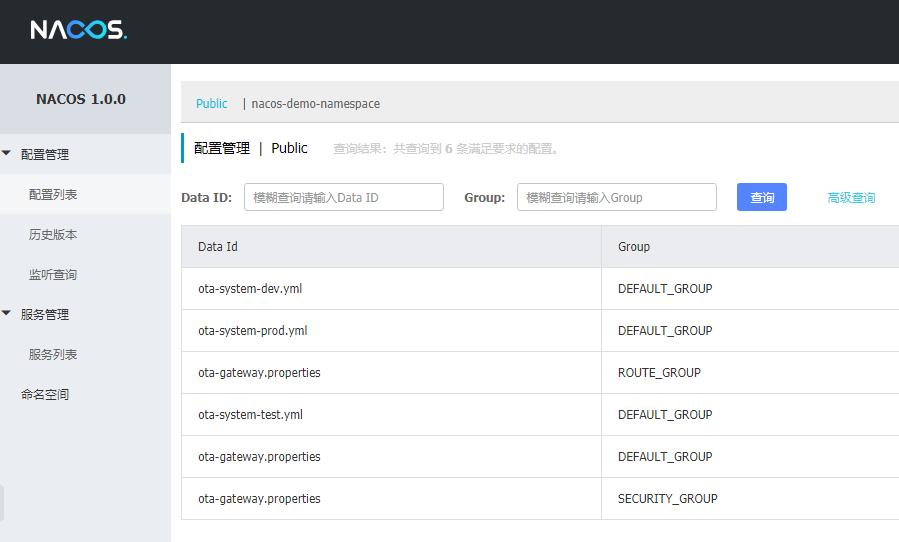
[faq](https://nacos.io/zh-cn/docs/faq.html)

# Nacos控制台

<https://nacos.io/zh-cn/docs/console-guide.html>

<http://10.104.112.222:8848/nacos/index.html>

nacos/nacos



# Nacos Open API

[open-api](https://nacos.io/zh-cn/docs/open-api.html)

|  |
| --- |
| 注册服务实例：  curl -X POST 'http://127.0.0.1:8848/nacos/v1/ns/instance?port=8848&healthy=true&ip=11.11.11.11&weight=1.0&serviceName=nacos.test.1&ephemeral=false'  查看当前集群Server列表：  curl -X GET '127.0.0.1:8848/nacos/v1/ns/operator/servers'  查看当前集群leader：  curl -X GET '127.0.0.1:8848/nacos/v1/ns/raft/leader' |

# Nacos配置中心

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。

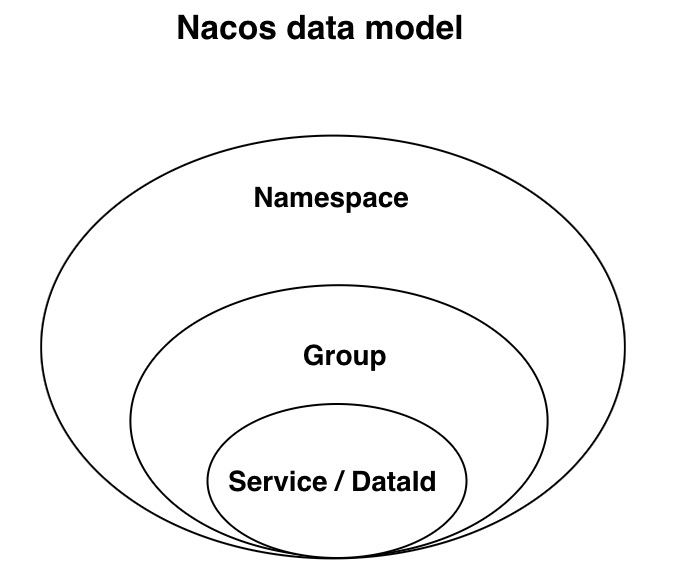
配置中心化管理让实现无状态服务变得更简单，让服务按需弹性扩展变得更容易。

Nacos 提供了一个简洁易用的UI (控制台样例 Demo) 帮助您管理所有的服务和应用的配置。Nacos 还提供包括配置版本跟踪、金丝雀发布、一键回滚配置以及客户端配置更新状态跟踪在内的一系列开箱即用的配置管理特性，帮助您更安全地在生产环境中管理配置变更和降低配置变更带来的风险。

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。

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

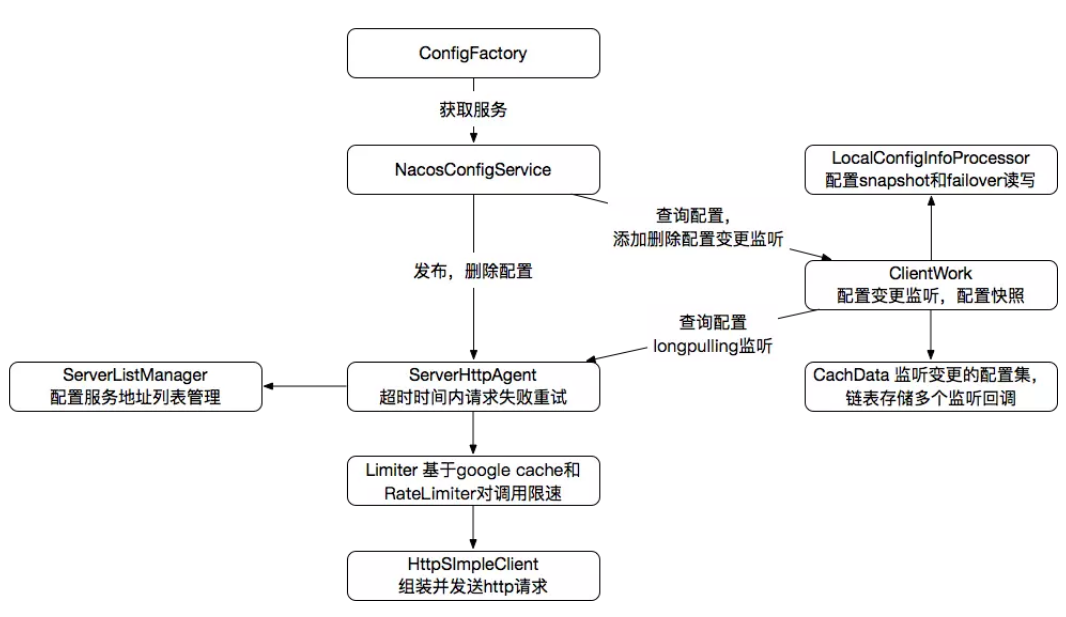
${prefix}-${spring.profile.active}.${file-extension}，prefix 默认为 spring.application.name 的值



Nacos 服务端保存了配置信息，客户端连接到服务端之后，根据 dataID，group可以获取到具体的配置信息，当服务端的配置发生变更时，客户端会收到通知。

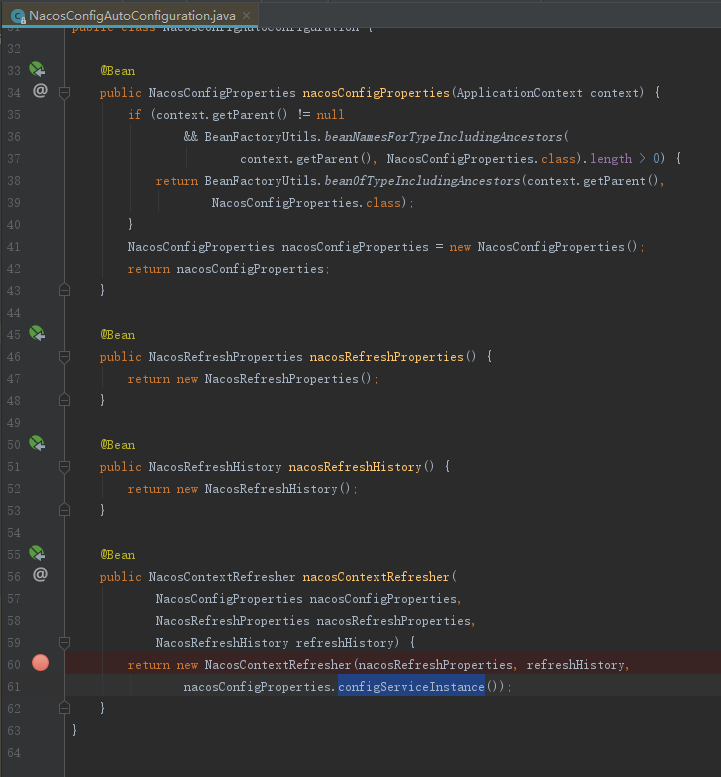
Nacos动态配置原理

类视图

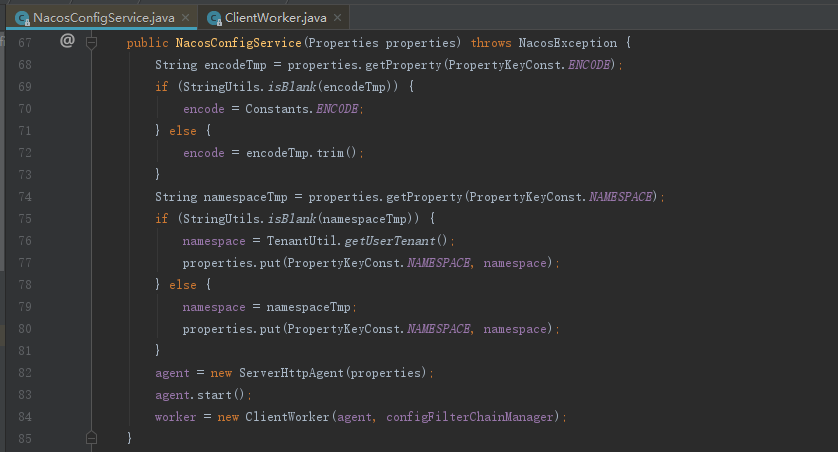


## 长轮询源码

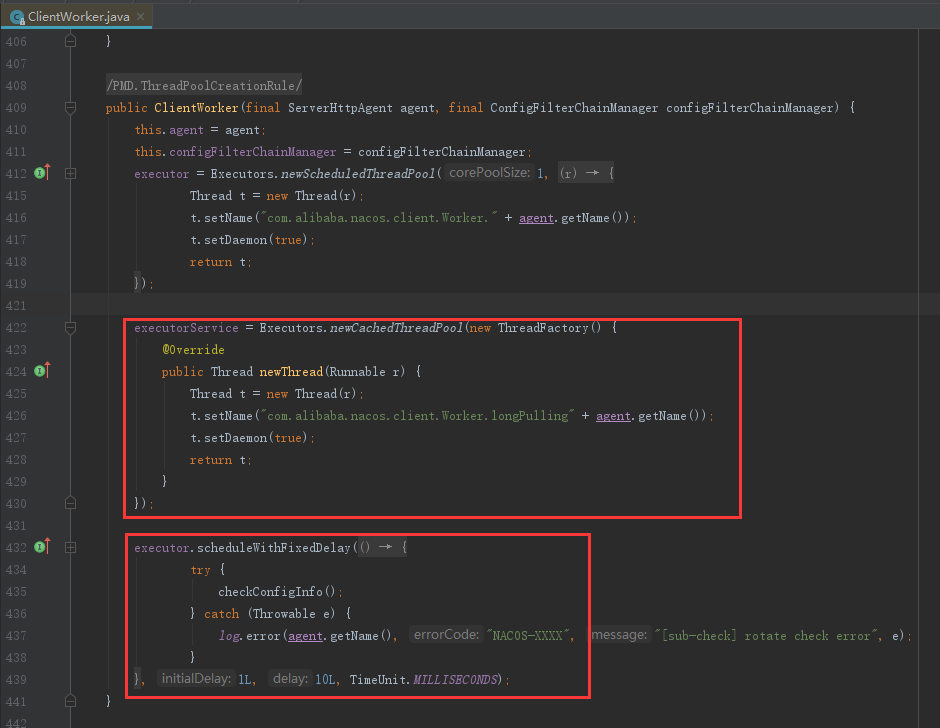
NacosConfigAutoConfiguration nacosConfigProperties.configServiceInstance()



NacosFactory.createConfigService() -> 实例化NacosConfigService

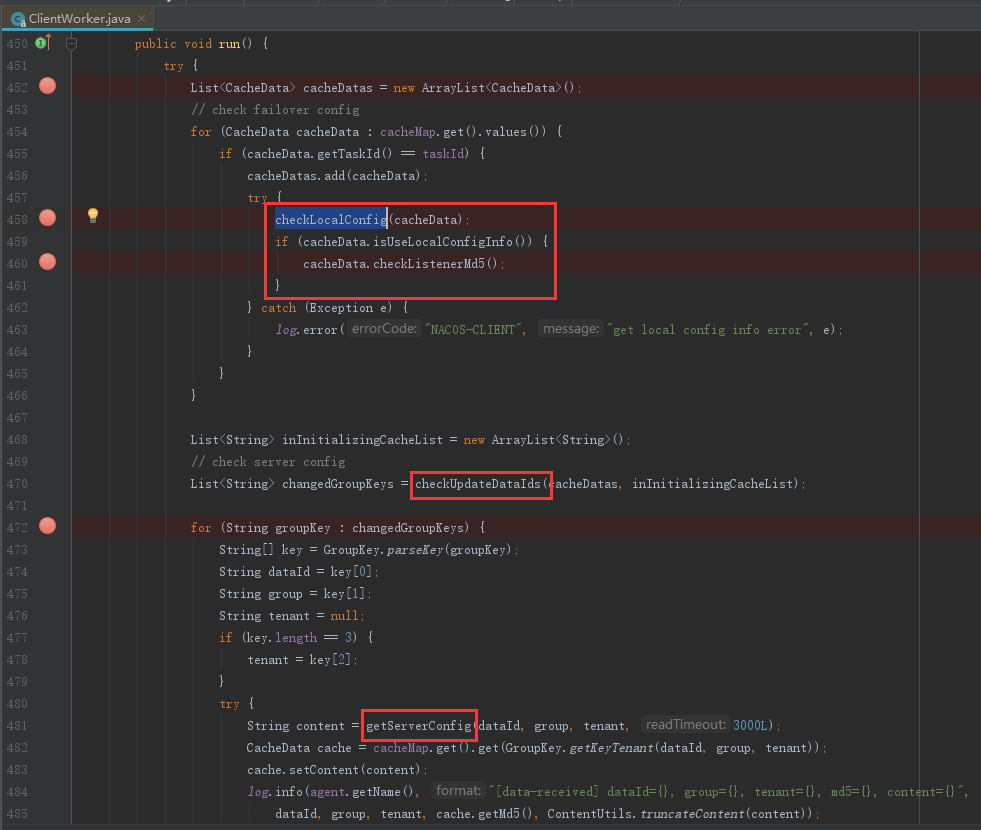


new ClientWorker()



初始化时，创建10ms定时任务，启动LongPullingRunnable任务

根据总监听数量计算需要启动的LongPullingRunnable任务数



检查本地配置：

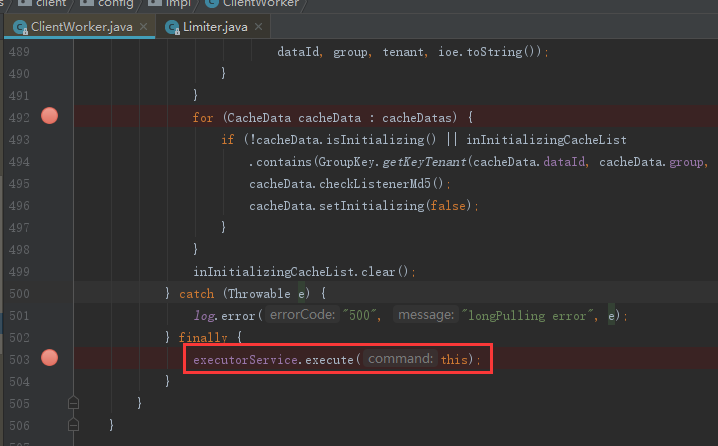
没有->有，有->有且存在变更，更新cachData配置信息；

有->没有cacheData.setUseLocalConfigInfo(false); 从服务端获取配置。

checkUpdateDataIds：长轮询获取更新的dataId列表

getServerConfig：获取远程配置，更新到本地

finally中再次提交本任务



值得关注的地方：

* http 长轮询

服务器收到请求后如果有数据, 立刻响应请求; 如果没有数据就会 hold 一段时间, 这段时间内如果有数据立刻响应请求; 如果时间到了还没有数据, 则响应 http 请求；

优点：客户端只需要通过一个无状态的 http 请求即可获取到服务端的数据；

缺点：服务器端没有数据 hold 住连接时会造成浪费, 容易产生服务器瓶颈；

* Limiter调用限速

Limiter.isLimit

* 容错机制

线程安全（AtomicReference、synchronized），本地snapshot文件、MD5、这样保证即使配置服务挂掉也不受影响。

总结：

Nacos 服务端创建了相关的配置项后，客户端就可以进行监听了。

客户端是通过一个定时任务来检查自己监听的配置项的数据的，一旦服务端的数据发生变化时，客户端将会获取到最新的数据，并将最新的数据保存在一个 CacheData 对象中，然后会重新计算 CacheData 的 md5 属性的值，此时就会对该 CacheData 所绑定的 Listener 触发 receiveConfigInfo 回调。

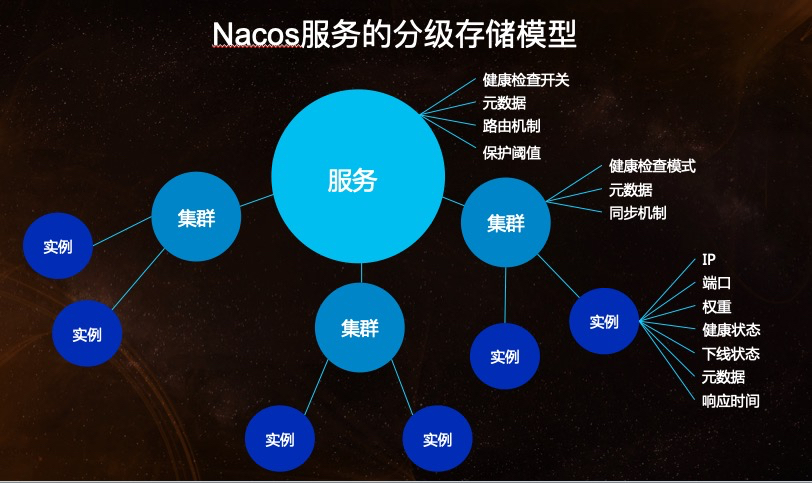
考虑到服务端故障的问题，客户端将最新数据获取后会保存在本地的 snapshot 文件中，以后会优先从文件中获取配置信息的值。

# Nacos服务发现

为什么需要服务发现：

由于服务的跨度很大（公司很大的情况下）、数量很多（数以百计甚至更多），为保障系统的正常运行，必然需要有一个中心化的组件完成对各个服务的整合，即将分散于各处的服务进行汇总，汇总的信息可以是提供服务的组件名称、地址、数量等，每个组件拥有一个监听设备，当本组件内的某个服务的状态变化时报告至中心化的组件进行状态的更新。服务的调用方在请求某项服务时首先到中心化组件获取可提供该项服务的组件信息（IP、端口等），通过默认或自定义的策略选择该服务的某一提供者进行访问，实现服务的调用。

服务领域模型

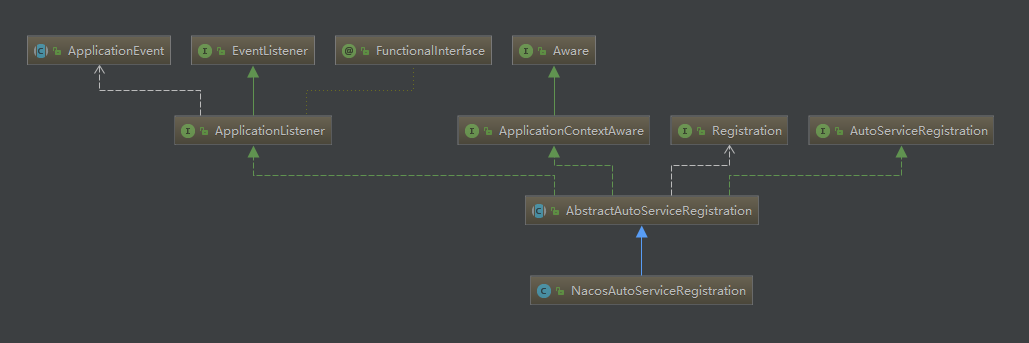


nacos服务发现原理

## 服务注册客户端

NacosDiscoveryAutoConfiguration自动配置，实例化NacosAutoServiceRegistration

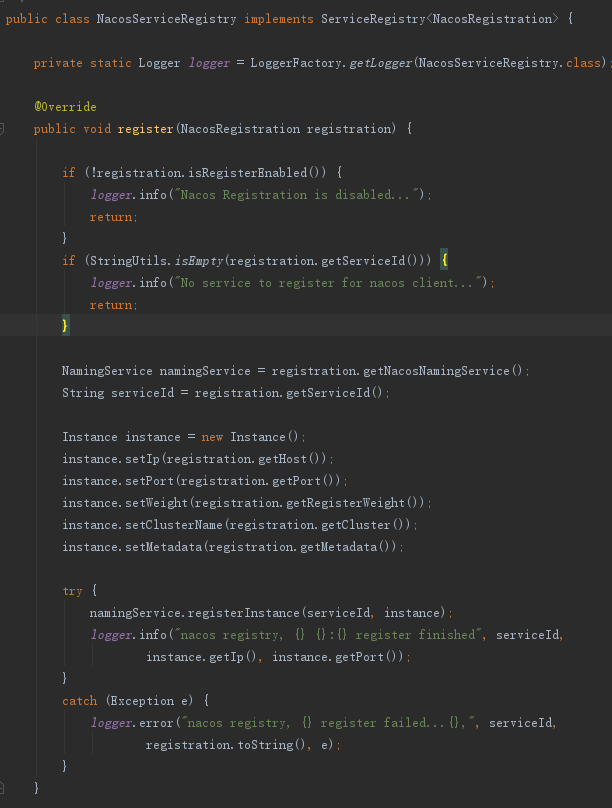




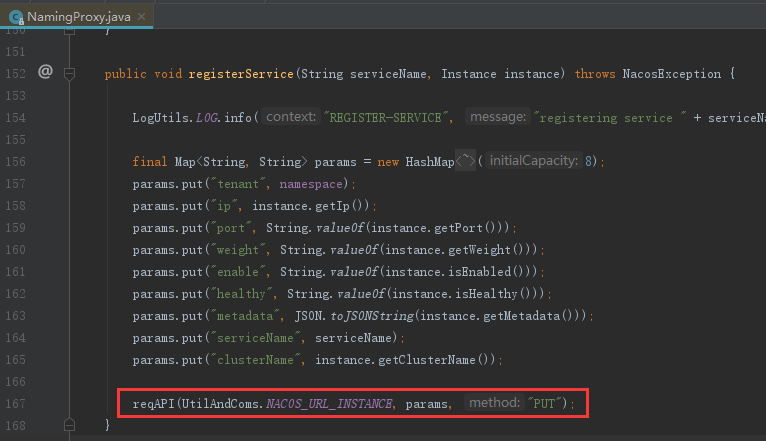
项目启动产生WebServerInitializedEvent时，向注册中心进行注册

代码为AbstractAutoServiceRegistration bind() -> start()

走到NacosServiceRegistry register()

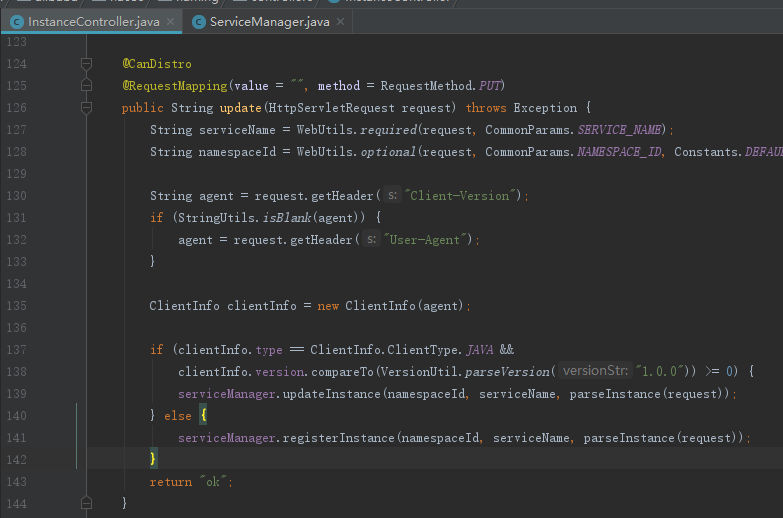


最终走到NamingProxy registerService()，向nacos服务器发起注册服务请求



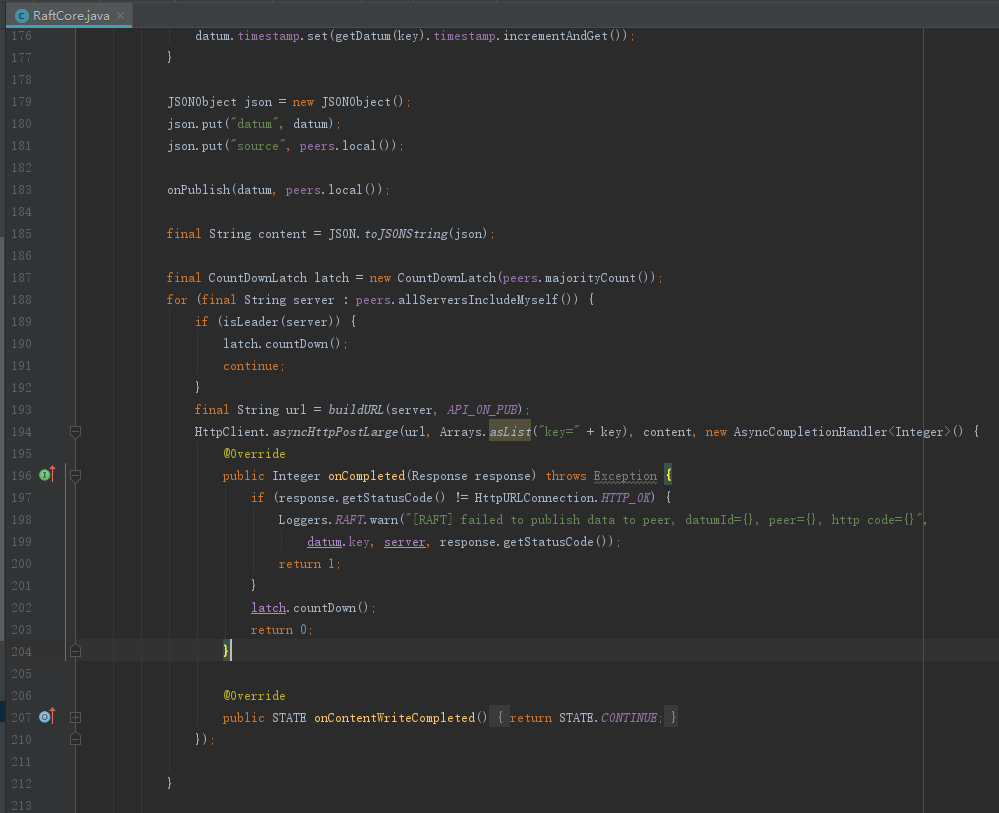
## 服务注册服务端

REST接口为InstanceController. update()

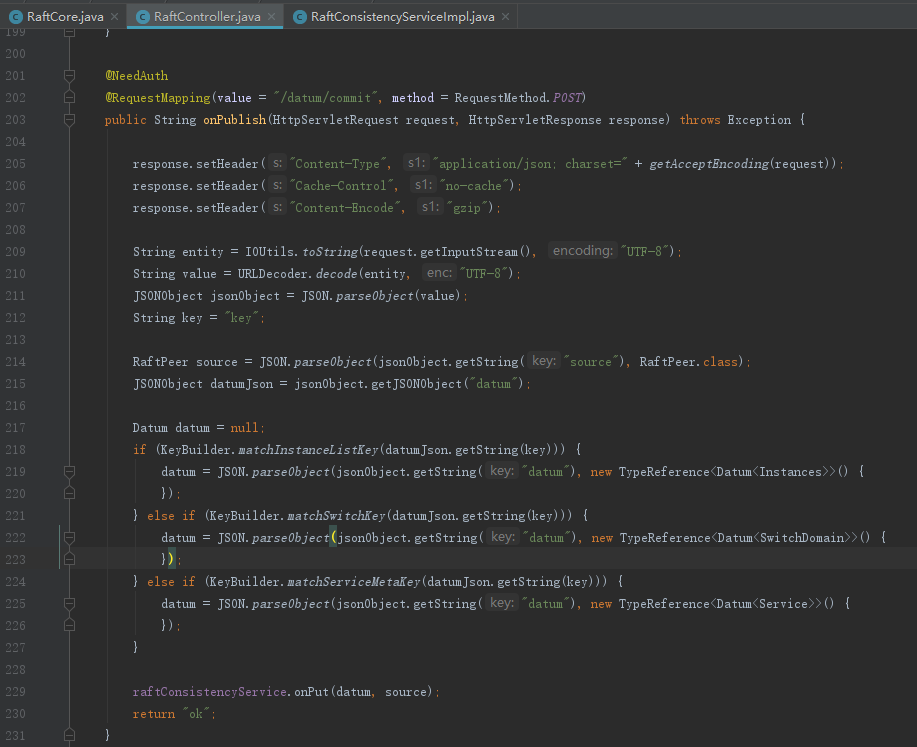


走到ServiceManager.addInstance() -> RaftConsistencyServiceImpl.put()

-> RaftCore.signalPublish() 提交数据



req RaftController.onPublish()，进入raft同步数据流程



## 一致性算法

分布式系统除了提升性能另外一个重要特性就是高可用，即系统中少部分机器故障不会使系统不可用。保证高可用的关键就是多副本，一旦有多副本，就面临多副本之间的一致性问题。

raft是一个共识算法（consensus algorithm），所谓共识，就是多个节点对某个事情达成一致的看法，即使是在部分节点故障、网络故障的情况下。在分布式系统中，主要用来管理日志复制的一致性。可以称为一种强一致性、去中心化、高可用的分布式协议。

Raft 通过选取领导者实现共识。在 Raft 集群中，一个服务器可以是领导者（leader），也可以是追随者（follower），也可以作为一些特定选举情况下（例如缺少领导者）的候选者。领导者负责向追随者发送日志副本。领导者通过发送心跳消息，定期通知追随者自身的存活情况。每位追随者维护一个超时（通常在 150 到 300 毫秒之间），正常情况下应在此时间范围内收到领导者的心跳。一旦收到心跳，超时就会重置。如果没有收到心跳，那么追随者就将自身状态更改为候选者，并开始领导者选举。

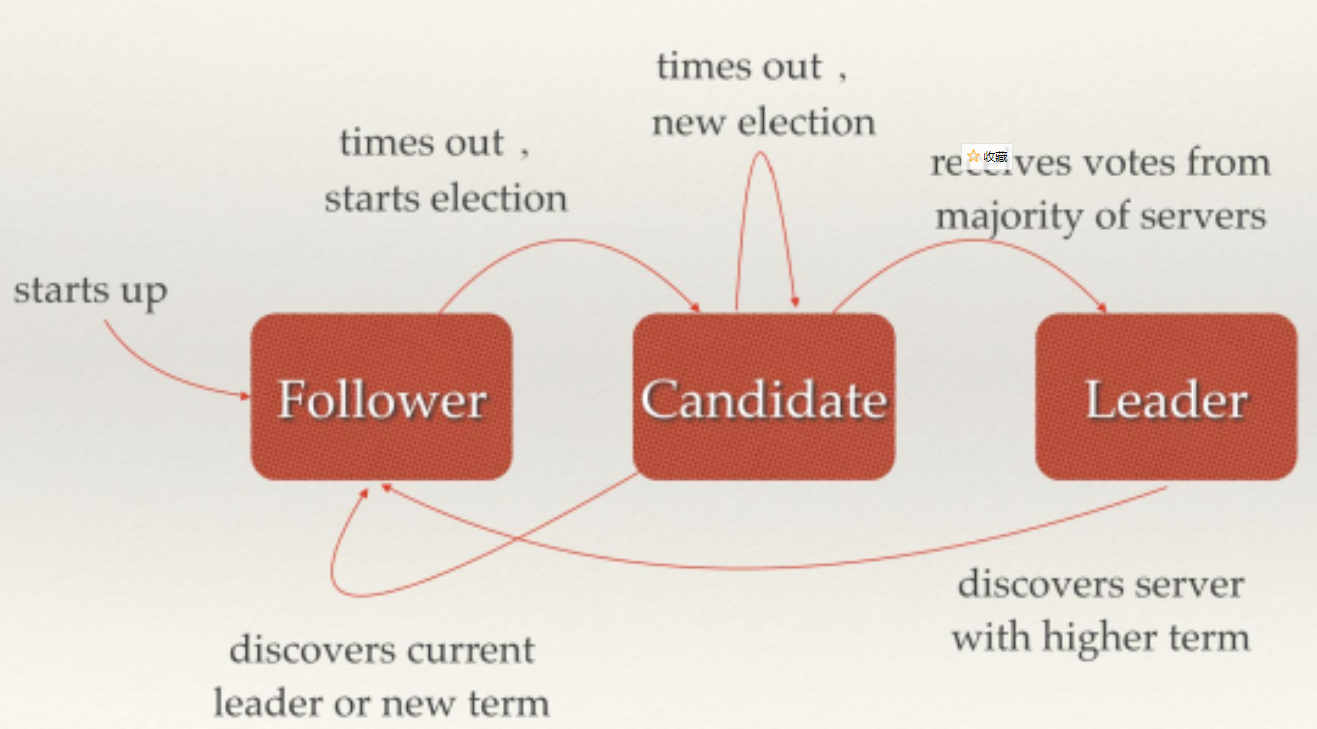
Raft 的解决方案大概可以理解成 先在所有将军中选出一个大将军，所有的决定由大将军来做。选举环节：比如说现在一共有3个将军 A, B, C，每个将军都有一个随机时间的倒计时器，倒计时一结束，这个将军就会把自己当成大将军候选人，然后派信使去问其他几个将军，能不能选我为总将军？假设现在将军A倒计时结束了，他派信使传递选举投票的信息给将军B和C，如果将军B和C还没把自己当成候选人（倒计时还没有结束），并且没有把选举票投给其他，他们把票投给将军A，信使在回到将军A时，将军A知道自己收到了足够的票数，成为了大将军。在这之后，是否要进攻就由大将军决定，然后派信使去通知另外两个将军，如果在一段时间后还没有收到回复（可能信使被暗杀），那就再重派一个信使，直到收到回复。

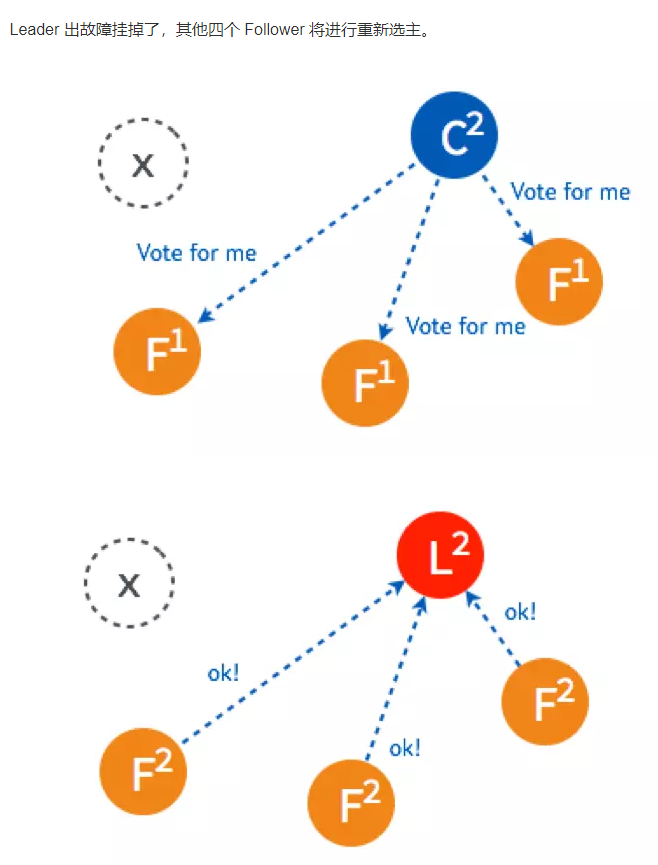
Raft的两个核心要点：

1.选取主节点

2.同步数据

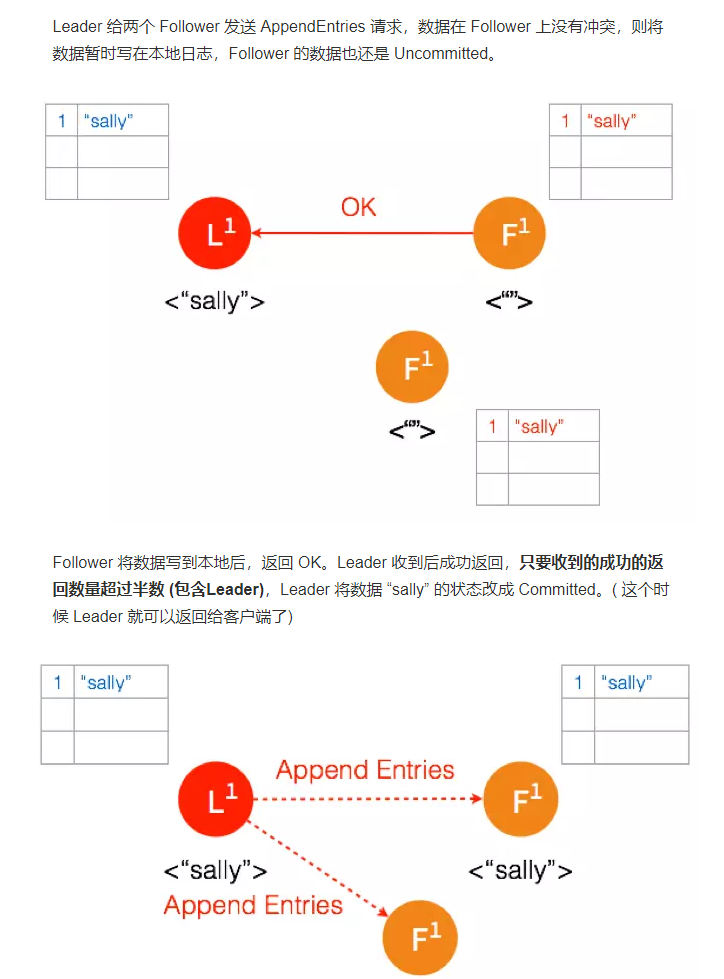
## Leader选举





## 同步数据

二阶段提交



## 心跳保持和续约

## 健康检查

# Nacos优缺点

Nacos = Spring Cloud Eureka + Spring Cloud Config

Nacos与Eureka对比

|  |
| --- |
| Spring Cloud Nacos  优点：  1）开箱即用，适用于dubbo，spring cloud等  2）AP模型，数据最终一致性  3）注册中心，配置中心二合一(二合一也不一定是优点)，提供控制台管理  4）纯国产，各种有中文文档，久经双十一考验  缺点：  1）刚刚开源不久，社区热度不够，依然存在bug  Spring Cloud Eureka：  优点：  1）Spring Cloud 官方推荐  2）AP模型，数据最终一致性  3）开箱即用，具有控制台管理  缺点：  1）客户端注册服务上报所有信息，节点多的情况下，网络，服务端压力过大，且浪费内存  2）客户端更新服务信息通过简单的轮询机制，当服务数量巨大时，服务器压力过大。  3）集群伸缩性不强，服务端集群通过广播式的复制，增加服务器压力  4）Eureka2.0 闭源（Spring Cloud最新版本还是使用的1.X版本的Eureka） |

Nacos与ZooKeeper对比

|  |
| --- |
| nacos直接面向微服务，AP  zookeeper是个CP系统，强一致性。  当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。但是zk会出现这样一种情况，当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s, 且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。 |

# Nacos发展及规划

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

2019.01.22

0.8.0版本：达到生产可用状态。在这个版本之前，仅用于开发和测试

2019.04.10

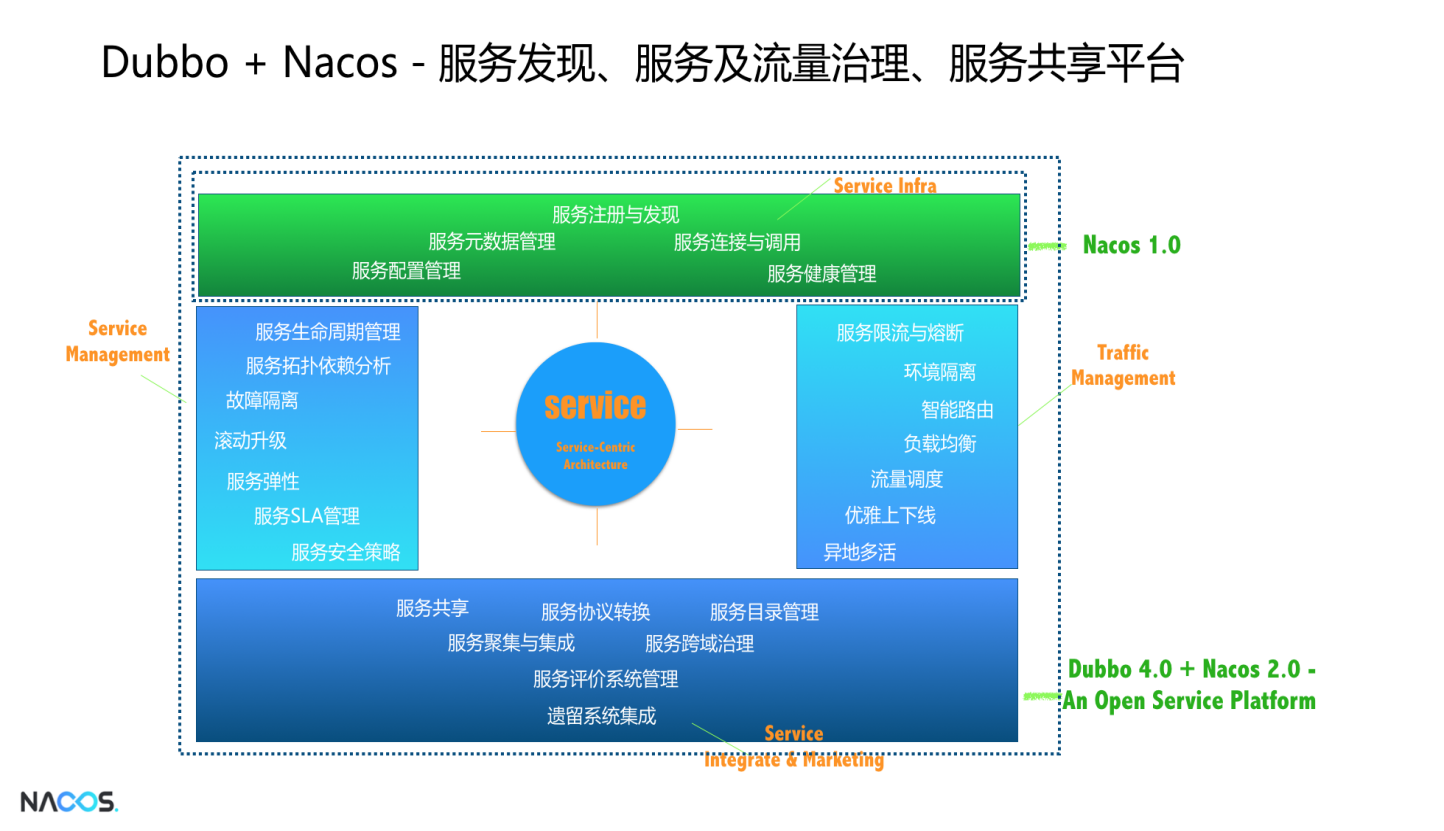
1.0.0版本：主要目标有两个：

构建简单易用的，服务相关的工具集，包括服务发现、配置管理、服务元数据存储、推送、一致性及元数据管理等；

与包括Spring Cloud、Kubernetes、Dubbo等开源生态做无缝的融合与支持，同时给这些生态带来很多面向生产时需要的优秀特性。

1年内

2.0版本：



# Nacos部署和运维

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