

Table of Contents

Introduction	1.1
第一章 基础	1.2
1.1 什么是nacos	1.2.1
1.2 nacos的功能和生态	1.2.2
1.3 nacos专业名词	1.2.3
1.4 nacos架构	1.2.4
1.5 nacos的安装启动	1.2.5
1.6 nacos的配置解析	1.2.6
第二章 实践	2.1
2.1 NacosClient	2.1.1
2.2 Nacos集成Spring	2.1.2
2.3 Nacos集成SpringBoot	2.1.3
2.4 Nacos集成SpringCloud	2.1.4
2.5 Nacos集成Docker	2.1.5
2.6 Nacos集成dubbo	2.1.6
2.7 Nacos集成K8s	2.1.7
2.8 Nacos集成Sync	2.1.8
2.9 技术探讨OR学习总结	2.1.9
第三章 源码解析	3.1
第四章 总结期许	3.2
第五章 技术畅想	3.3

- 写作背景
- 本书简介

写作背景

这本书是本人写的第一本gitbook书，大概耗费一个下午 将gitbook给安装。因为最近在研究nacos这个配置分发和注册中心，所以就想以此为题写一本书。希望之后的自己看到后，能够专心的把这本书给写出。

希望本书的写作内容，能给后来者一些启发，本书不仅仅是包含内容干活，更是面向个人的学习习惯触发。因为观念不同，可能我们看框架的视角不同，可做参考，但不可作为依赖。

本书基于Nacos2.2.3进行编写，如有错误的地方欢迎指正。

本书简介

首先，我们学习一门技术最好的方案就是从官方网站开始，下面是nacos的官方网站。



其中包含了其基本的介绍和与其它项目的集成,接下来我们将会从基本的概念和架构开始,先进行了解其基本的作用和实现,再与其他项目集成,最后我们再深挖源码,下面就开始我们的学习之旅把!

- 1.1什么是nacos?

1.1什么是nacos?

一门技术的兴起一定会有它自己独特的功能，正如缓存、数据库、消息队列等一系列的中间件一样，nacos也有它自己独特的起源。

nacos主要是阿里的开源产品，伴随的是阿里的生产实践以及借鉴其他的注册中心而有的孵化品，在官网上我们可以看到有篇关于阿里巴巴服务注册中心产品的发展回顾，[《阿里巴巴服务注册中心产品ConfigServer 10年技术发展回顾》](#)，这篇文档的大概意思是在阿里的业务拓展下，最初的服务注册发现产品Eureka不再符合阿里的业务，于是在2018年左右，阿里开始了自研服务注册中心的道路，最开始也是借鉴Eureka的设计理念，往后推进时也添加上了一些自己的思考和阿里线上的具体实践，一步步的迭代，从最初的SDK，到单机版，再到集群一步步的解决了服务注册发现方面的一些问题，然后就形成了我们今天所看到的从ConfigServer进化而来的nacos。

	ConfigServer	Eureka
2008年	V1.0: 单机版，定义了服务发现的领域模型	
2009年初	V1.5: 应用和ConfigServer集群发布解耦	
2009年7月	V2.0: 基于客户端模式同步数据，支持集群部署	
2010年底	V2.5: 优化集群间数据同步模式，申请国家专利。	
2012年9月1号		Eureka1.0正式开源
2012年底	V3.0: 支持session和data分层部署	
2014年	V3.5: 支持异地多活等细分场景	
2015年		Eureka2.0架构升级方案公布
2017年	V4.0: 支持data分片能力	
2018年7月		Eureka2.0架构升级宣布停止

在分布式系统中，有三个特性一致性、可用性、分区容错性，而注册中心必然处于分布式系统中，那么它必然要满足[CAP原则](#)。而Eureka和ConfigServer则是同属于AP类型的注册中心，他们两个在之后的业务拓展中

拥有着相似的阻碍，而阿里巴巴则将ConfigServer的技术架构和生产环境的发现融合到了开源产品nacos中，继往开来在云原生、微服务的时代继续着发光发热。

- 1.2nacos功能和生态
 - 1.2.1特性
 - 1.2.2生态

上一节我们讨论了nacos的发展及一些基本的注册中心知识，这一节我们将对于它的功能和生态进行进一步的探讨。

1.2nacos功能和生态

对于我们学习技术来说，最重要的就是这门新的技术有什么样的功能，和他的生态是否强大。功能决定了它的业务适用性，任何的技术都是为业务而生。而生态则是技术的后备支持，比如说漏洞维护，功能新增等一系列的技术支持。很显然Nacos是阿里巴巴的产品，而且可以与多种技术进行集成，由此决定了它在微服务中的适用性。

1.2.1特性

nacos的官网是这么说的，它是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。

1、服务发现和服务健康检测

服务发现和服务健康检测基本上是每一个注册中心都需要进行保证的功能。Nacos采用的是基于DNS和RPC的服务发现。支持传输层层面的健康检测，也就是我们常用的ping命令或者tcp的一系列指令，同时也迟迟应用层自定义的健康检查。

对于复杂的云环境还提供了agent 上报模式和服务端主动检测2种健康检查模式。

2、动态配置服务

动态配置可以让我们的配置文件中心化、外部化和动态化的管理所有环境的应用配置和服务配置，可以更加方便的管理服务，让服务进行弹性的拓展更加的简单。换句话说就是我们的配置文件不再由应用管理，而是交由一个中心化的应用进行管理，方便了配置文件的更改和版本追踪，最终还是服务于应用。

3、动态DNS服务（DDNS）

通过支持权重路由，动态DNS服务能让您轻松实现中间层负载均衡、更灵活的路由策略、流量控制以及简单数据中心内网的简单DNS解析服务。动态DNS服务还能让您更容易地实现以DNS协议为基础的服务发现，以消除耦合到厂商私有服务发现API上的风险。

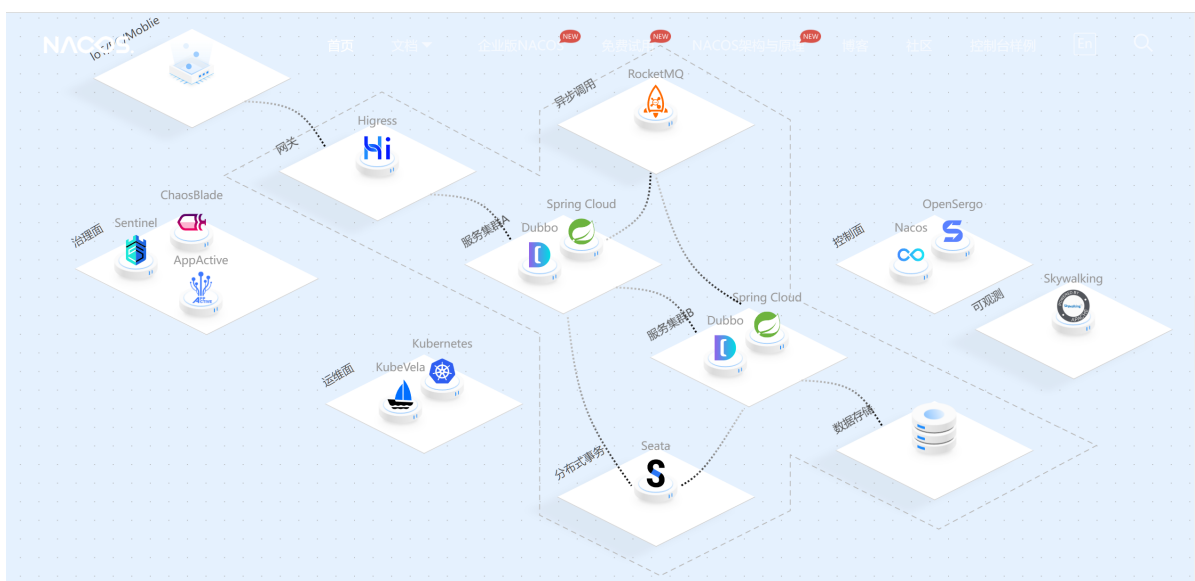
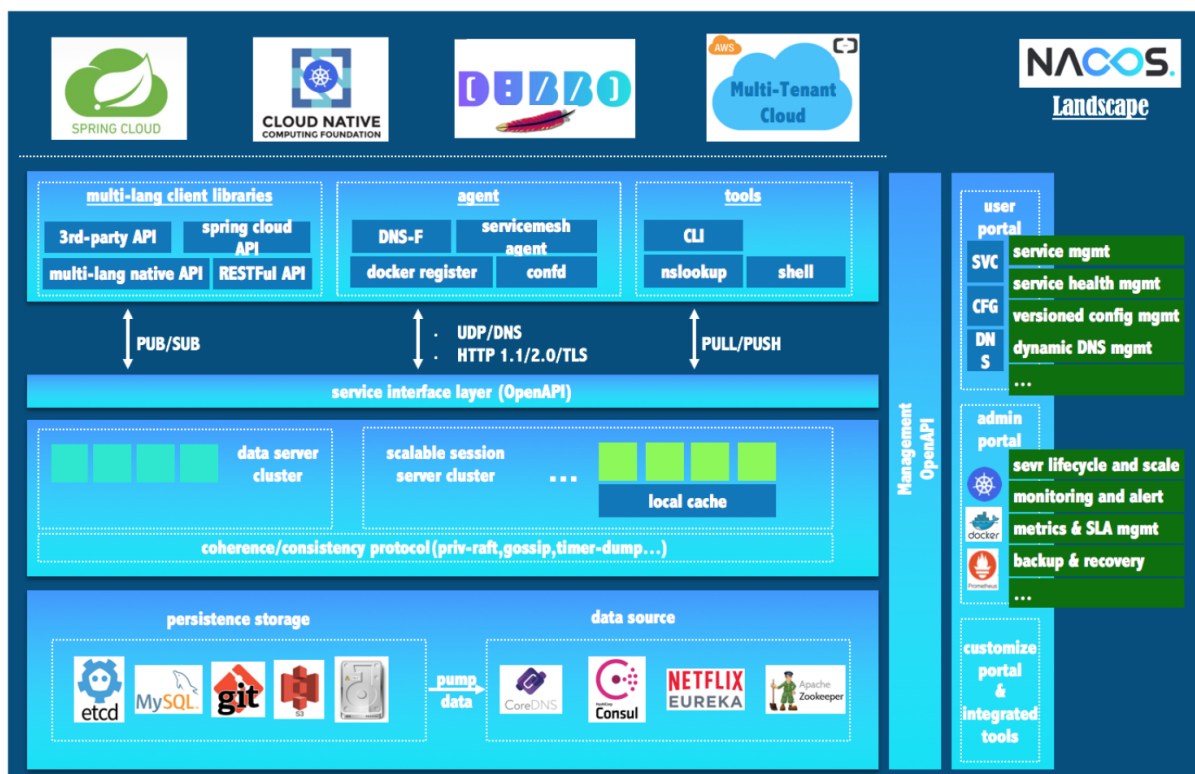
4、服务及其元数据管理

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。

1.2.2生态

nacos的生态十分强大，它可以很方便的与一些第三方框架进行集成，具体的在此不再阐述，之后会在项目中进行一一的展现。而在此我想说的是，对于一个服务的生态，我们需要关心的是这个技术的使用方向以及使用的广度。

对于本书的nacos来说，他有非常多的使用方案，包括配置分发，服务发现等，因为背靠阿里巴巴这个巨大的生态圈，也使得它的更新迭代也是非常快的，我们在使用的初期仅需要去掌握使用即可，等到我们使用的次数多了，再去探讨核心功能的实现才是最好的‘食用方式’。下图为Nacos的生态图：



- 1.3Nacos的专业名词
 - 1.3.1地域
 - 1.3.2可用区
 - 1.3.3接入点
 - 1.3.4命名空间
 - 1.3.5配置
 - 1.3.6配置管理
 - 1.3.7配置项
 - 1.3.8配置集
 - 1.3.9配置集 ID
 - 1.3.10配置分组
 - 1.3.11配置快照
 - 1.3.12服务
 - 1.3.13服务名
 - 1.3.14服务注册中心
 - 1.3.15服务发现
 - 1.3.16元信息
 - 1.3.17应用
 - 1.3.18服务分组
 - 1.3.19虚拟集群
 - 1.3.20实例
 - 1.3.21权重
 - 1.3.22健康检查
 - 1.3.23健康保护阈值

上一节我们说到了Nacos的一些特性和它的生态圈，如果不出意外的话，或者作者未来的几年依然在这个行业，一定会慢慢的把生态圈上的所有给更新完毕。接下来我们将要来学习下Nacos的一些专业术语

1.3Nacos的专业名词

这一部分主要取自Nacos的官方文档，因为对于专业名词来说，官网给的才是最准确的，其中也添加上了一点自己的见解。

1.3.1地域

物理的数据中心，资源创建成功之后就不能再次更换。

1.3.2可用区

同一地域内，电力和网络互相独立的物理区域。同一可用区内，实例的网络延迟较低。

1.3.3接入点

地域的某个服务的入口域名。

1.3.4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的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。这个我们之后的使用会比较多，主要用于隔绝环境，方便我们在不同环境下的开发工作。

1.3.5配置

在系统开发过程中，开发者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。配置变更是调整系统运行时的行为的有效手段。这里的配置文件有很多格式，text,yml,properties等，已经可以符合我们的日常配置文件的格式。

1.3.6配置管理

主要是对上面的配置进行一系列的操作，后面我们会进行探讨其源码的具体实现。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。

1.3.7配置项

一个具体的可配置的参数与其值域，通常以 `param-key=param-value` 的形式存在。例如我们常配置系统的日志输出级别
(`logLevel=INFO|WARN|ERROR`) 就是一个配置项。

1.3.8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。

1.3.9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。

1.3.10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。

1.3.11配置快照

Nacos 的客户端 SDK 会在本地生成配置的快照。当客户端无法连接到 Nacos Server 时，可以使用配置快照显示系统的整体容灾能力。配置快照类似于 Git 中的本地 commit，也类似于缓存，会在适当的时机更新，但是并没有缓存过期（expiration）的概念。可以说是本地的缓存，用于缓解 Nacos出现问题无法访问时可以生效，保证服务的正常运行。

1.3.12服务

通过预定义接口网络访问的提供给客户端的软件功能。例如在大型网购系统中的订单系统、消费券系统、会员系统等。

1.3.13服务名

服务提供的标识，通过该标识可以唯一确定其指代的服务。

1.3.14服务注册中心

存储服务实例和服务负载均衡策略的数据库，可以将自己自身的服务信息提交到注册中心，其他的服务模块想要调用时则可以进行注册中心寻找调用的信息，注册中心也可以在此时做一层负载均衡，将请求分发到算力充足或者

业务不是很繁忙的服务上面去。

1.3.15 服务发现

在计算机网络上，（通常使用服务名）对服务下的实例的地址和元数据进行探测，并以预先定义的接口提供给客户端进行查询。

1.3.16 元信息

Nacos数据（如配置和服务）描述信息，如服务版本、权重、容灾策略、负载均衡策略、鉴权配置、各种自定义标签（label），从作用范围来看，分为服务级别的元信息、集群的元信息及实例的元信息。

1.3.17 应用

用于标识服务提供方的服务的属性。

1.3.18 服务分组

不同的服务可以归类到同一分组。

1.3.19 虚拟集群

同一个服务下的所有服务实例组成一个默认集群，集群可以被进一步按需求划分，划分的单位可以是虚拟集群。

1.3.20 实例

提供一个或多个服务的具有可访问网络地址（IP:Port）的进程。

1.3.21 权重

实例级别的配置。权重为浮点数。权重越大，分配给该实例的流量越大。

1.3.22 健康检查

以指定方式检查服务下挂载的实例（Instance）的健康度，从而确认该实例（Instance）是否能提供服务。根据检查结果，实例（Instance）会被判断为健康或不健康。对服务发起解析请求时，不健康的实例（Instance）不会返回给客户端。

1.3.23 健康保护阈值

为了防止因过多实例（Instance）不健康导致流量全部流向健康实例（Instance），继而造成流量压力把健康实例（Instance）压垮并形成雪崩效应，应将健康保护阈值定义为一个 0 到 1 之间的浮点数。当域名健康实例数（Instance）占总服务实例数（Instance）的比例小于该值时，无论实例（Instance）是否健康，都会将这个实例（Instance）返回给客户端。这样做虽然损失了一部分流量，但是保证了集群中剩余健康实例（Instance）能正常工作。

- 1. 4Nacos架构

上一节我们着重了解了下Nacos的一些官方名词，接下来我们将继续深入Nacos的架构继续探讨。这时可能有人会疑惑了，为什么架构这么重要的部分只介绍了一节，因为官网已经给了一本300多页的书来讲Nacos的架构，产品的开发人员肯定比我更加的了解技术架构，所以在此就不再过多的阐述，就大概的讲述一下我对于软件架构的学习方法。

1. 4Nacos架构

对于一门技术的架构的话，我们如果学习，还是需要从官方入手，如果是比较火的技术，官方一般都会出一个文档关于研发这个软件的过程，以及致力于解决什么问题。



如上图所示，这本书是由阿里出的，所以内容也不会太差，我们在学习的初期，一般是不会太多的去关注架构。大概是在使用之后，我们需要对于开源软件有我们自己的DIY时才会去关注软件的架构，然后对于模块进行魔改，然后进行内测上线。

《Nacos架构&原理》

- 1.5Nacos的安装和启动
 - 1.5.1单机部署启动

这一节我们来探讨一下软件的安装和启动，在我们之前的学习中，我们可能倾向于去将手动下载，然后解压安装，但是随着技术的迭代，越来越多的虚拟技术的出现使得软件安装部署也有了新的花样。

假如我们来假设这样一个场景，我们需要部署一个集群，这样的话如果是在我们自己测试的环境还好说，通过暴力进行解决，或者说也可以通过写一个shell脚本进行，但是如果说是在不同的主机上，这样就会变得很麻烦，于是乎就有了Docker, k8s等一系列的实现方案，我们可以在虚拟的容器内进行部署，然后也可以动态的进行上线下线的控制。对于Docker的学习还是有一定的必要的，之后可以了解一下这里只是简单的介绍一下之后的软件安装模式。

1.5Nacos的安装和启动

我们这部分还是根据Nacos的官网来进行，有什么需要注意的细节我会一一的点明。

因为Nacos是基于JAVA实现的，所以最基础的要JAVA的环境，这一点是必不可少的，一下的安装都基于Linux完成，原因的话可以下去自己去了解。

1.5.1单机部署启动

我们就从最基础的开始安装，只演示一次，之后我们会只使用docker安装。因为学到这里就已经默认已经具备独立安装JDK的条件了，我们主要来进行安装Nacos。

首先我们要先下载适合我们的软件包，下载地址我们一般是在github上找稳定的版本，我选择的是最新的


Nacos，选择第一个进行下载。



The vulnerability only affects port 7848 (by default), which is typically used as the communication port for Nacos cluster inter-raft protocol and does not handle client requests. Therefore, the risk can be controlled by disabling requests from outside of Nacos clusters (e.g. by limiting or not exposing the port) in older versions.

Detail:

- [#10318](#) Fix import problem when disable auth.
- [#10542](#) Add classes whitelist for HessianSerializer.

▼Assets 4

 nacos-server-2.2.3.tar.gz	142 MB	May 25
 nacos-server-2.2.3.zip	142 MB	May 25
 Source code (zip)		May 25
 Source code (tar.gz)		May 25

  40  10  8 50 people reacted

下载成功后，我们将其上传至Linux中，也可以通过linux的curl命令下载，但是一般很慢不建议。

#解压

```
tar -zxvf nacos-server-2.2.3.tar.gz nacos
```

修改配置，nacos的配置是在application.properties中

```

***** Spring Boot Related Configurations *****
#这些一般用默认的就可以
### Default web context path:
server.servlet.contextPath=/nacos
### Include message field
server.error.include-message=ALWAYS
### Default web server port:
server.port=8848

***** Network Related Configurations *****
### If prefer hostname over ip for Nacos server addresses in cluster
# nacos.inetutils.prefer-hostname-over-ip=false

### Specify local server's IP:
# nacos.inetutils.ip-address=

***** Config Module Related Configurations *****
### If use MySQL as datasource:
### Deprecated configuration property, it is recommended to use `spring.datasource.platform`
# spring.datasource.platform=mysql
# spring.sql.init.platform=mysql

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

#数据库的配置，我就用我本机的mysql，也不再进行安装了
db.url.0=jdbc:mysql://10.102.46.60:3306/nacos?characterEncoding=utf8
db.user.0=root
db.password.0=123456

```

```
### Connection pool configuration: hikariCP
db.pool.config.connectionTimeout=30000
db.pool.config.validationTimeout=10000
db.pool.config.maximumPoolSize=20
db.pool.config.minimumIdle=2

***** Naming Module Related Configurations *****

### If enable data warmup. If set to false, the server would accept
# nacos.naming.data.warmup=true

### If enable the instance auto expiration, kind like of health check
# nacos.naming.expireInstance=true

### Add in 2.0.0
### The interval to clean empty service, unit: milliseconds.
# nacos.naming.clean.empty-service.interval=60000

### The expired time to clean empty service, unit: milliseconds.
# nacos.naming.clean.empty-service.expired-time=60000

### The interval to clean expired metadata, unit: milliseconds.
# nacos.naming.clean.expired-metadata.interval=5000

### The expired time to clean metadata, unit: milliseconds.
# nacos.naming.clean.expired-metadata.expired-time=60000

### The delay time before push task to execute from service changed
# nacos.naming.push.pushTaskDelay=500

### The timeout for push task execute, unit: milliseconds.
# nacos.naming.push.pushTaskTimeout=5000
```

```

### The delay time for retrying failed push task, unit: milliseconds
# nacos.naming.push.pushTaskRetryDelay=1000

### Since 2.0.3
### The expired time for inactive client, unit: milliseconds.
# nacos.naming.client.expired.time=180000

##### Cmdb Module Related Configurations #####
### The interval to dump external CMDB in seconds:
# nacos.cmdb.dumpTaskInterval=3600

### The interval of polling data change event in seconds:
# nacos.cmdb.eventTaskInterval=10

### The interval of loading labels in seconds:
# nacos.cmdb.labelTaskInterval=300

### If turn on data loading task:
# nacos.cmdb.loadDataAtStart=false

##### Metrics Related Configurations #####
### Metrics for prometheus
#management.endpoints.web.exposure.include=*

### Metrics for elastic search
management.metrics.export.elastic.enabled=false
#management.metrics.export.elastic.host=http://localhost:9200

### Metrics for influx
management.metrics.export.influx.enabled=false

```

```

#management.metrics.export.influx.db=springboot
#management.metrics.export.influx.uri=http://localhost:8086
#management.metrics.export.influx.auto-create-db=true
#management.metrics.export.influx.consistency=one
#management.metrics.export.influx.compressed=true

##### Access Log Related Configurations #####
### If turn on the access log:
server.tomcat.accesslog.enabled=true

### The access log pattern:
server.tomcat.accesslog.pattern=%h %l %u %t "%r" %s %b %D %{User-Agent}

### The directory of access log:
server.tomcat.basedir=file:..

##### Access Control Related Configurations #####
### If enable spring security, this option is deprecated in 1.2.0:
#spring.security.enabled=false

### The ignore urls of auth
nacos.security.ignore.urls=/,/error,/**/*.css,/**/*.js,/**/*.html,/

### The auth system to use, currently only 'nacos' and 'ldap' is supported
nacos.core.auth.system.type=nacos

### If turn on auth system:
nacos.core.auth.enabled=false

### Turn on/off caching of auth information. By turning on this switch, the auth information will be cached in the local memory.
nacos.core.auth.caching.enabled=true

```

```

### Since 1.4.1, Turn on/off white auth for user-agent: nacos-server
nacos.core.auth.enable.userAgentAuthWhite=false

### Since 1.4.1, worked when nacos.core.auth.enabled=true and nacos
### The two properties is the white list for auth and used by identity
nacos.core.auth.server.identity.key=
nacos.core.auth.server.identity.value=

### worked when nacos.core.auth.system.type=nacos
### The token expiration in seconds:
nacos.core.auth.plugin.nacos.token.cache.enable=false
nacos.core.auth.plugin.nacos.token.expire.seconds=18000
### The default token (Base64 String):
nacos.core.auth.plugin.nacos.token.secret.key=

### worked when nacos.core.auth.system.type=ldap, {0} is Placeholder
#nacos.core.auth.ldap.url=ldap://localhost:389
#nacos.core.auth.ldap.basedc=dc=example,dc=org
#nacos.core.auth.ldap.userDn=cn=admin,{nacos.core.auth.ldap.basedc}
#nacos.core.auth.ldap.password=admin
#nacos.core.auth.ldap.userdn=cn={0},dc=example,dc=org
#nacos.core.auth.ldap.filter.prefix=uid
#nacos.core.auth.ldap.case.sensitive=true

***** Istio Related Configurations *****#
### If turn on the MCP server:
nacos.istio.mcp.server.enabled=false

***** Core Related Configurations *****#

### set the WorkerID manually

```

```

# nacos.core.snowflake.worker-id=

### Member-MetaData
# nacos.core.member.meta.site=
# nacos.core.member.meta.adweight=
# nacos.core.member.meta.weight=

### MemberLookup
### Addressing pattern category, If set, the priority is highest
# nacos.core.member.lookup.type=[file,address-server]
## Set the cluster list with a configuration file or command-line arguments
# nacos.member.list=192.168.16.101:8847?raft_port=8807,192.168.16.102:8847
## for AddressServerMemberLookup
# Maximum number of retries to query the address server upon initialization
# nacos.core.address-server.retry=5
## Server domain name address of [address-server] mode
# address.server.domain=jmenv.tbsite.net
## Server port of [address-server] mode
# address.server.port=8080
## Request address of [address-server] mode
# address.server.url=/nacos/serverlist

***** JRaft Related Configurations *****#

### Sets the Raft cluster election timeout, default value is 5 seconds
# nacos.core.protocol.raft.data.election_timeout_ms=5000
### Sets the amount of time the Raft snapshot will execute periodically
# nacos.core.protocol.raft.data.snapshot_interval_secs=30
### raft internal worker threads
# nacos.core.protocol.raft.data.core_thread_num=8
### Number of threads required for raft business request processing
# nacos.core.protocol.raft.data.cli_service_thread_num=4

```

```

### raft linear read strategy. Safe linear reads are used by default.
# nacos.core.protocol.raft.data.read_index_type=ReadOnlySafe
### rpc request timeout, default 5 seconds
# nacos.core.protocol.raft.data.rpc_request_timeout_ms=5000

##### Distro Related Configurations #####

### Distro data sync delay time, when sync task delayed, task will
# nacos.core.protocol.distro.data.sync.delayMs=1000

### Distro data sync timeout for one sync data, default 3 seconds.
# nacos.core.protocol.distro.data.sync.timeoutMs=3000

### Distro data sync retry delay time when sync data failed or timeout
# nacos.core.protocol.distro.data.sync.retryDelayMs=3000

### Distro data verify interval time, verify synced data whether exist
# nacos.core.protocol.distro.data.verify.intervalMs=5000

### Distro data verify timeout for one verify, default 3 seconds.
# nacos.core.protocol.distro.data.verify.timeoutMs=3000

### 加载快照数据失败时，分区数据加载重试的延迟时间，默认为 30 秒。
# nacos.core.protocol.distro.data.load.retryDelayMs=30000

### 启用以支持 prometheus 服务发现
#nacos.prometheus.metrics.enabled=true

### Since 2.3
##### Grpc Configurations #####

## sdk grpc(between nacos server and client) configuration

```



```
## Sets the maximum message size allowed to be received on the server
#nacos.remote.server.grpc.sdk.max-inbound-message-size=10485760

## 设置发送 keepalive ping 之前无读取活动的时间（毫秒）。典型的默认值是
#nacos.remote.server.grpc.sdk.keep-alive-time=7200000

## 设置发送 keepalive ping 后等待读取活动的时间（毫秒）。默认为 20 秒。
#nacos.remote.server.grpc.sdk.keep-alive-timeout=20000

## 设置时间（毫秒），指定允许客户端配置的最长保持连接时间。典型的默认值是
#nacos.remote.server.grpc.sdk.permit-keep-alive-time=300000

## cluster grpc(inside the nacos server) configuration
#nacos.remote.server.grpc.cluster.max-inbound-message-size=10485760

## 设置发送 keepalive ping 之前无读取活动的时间（毫秒）。典型的默认值是
#nacos.remote.server.grpc.cluster.keep-alive-time=7200000

## 设置发送 keepalive ping 后等待读取活动的时间（毫秒）。默认为 20 秒。
#nacos.remote.server.grpc.cluster.keep-alive-timeout=20000

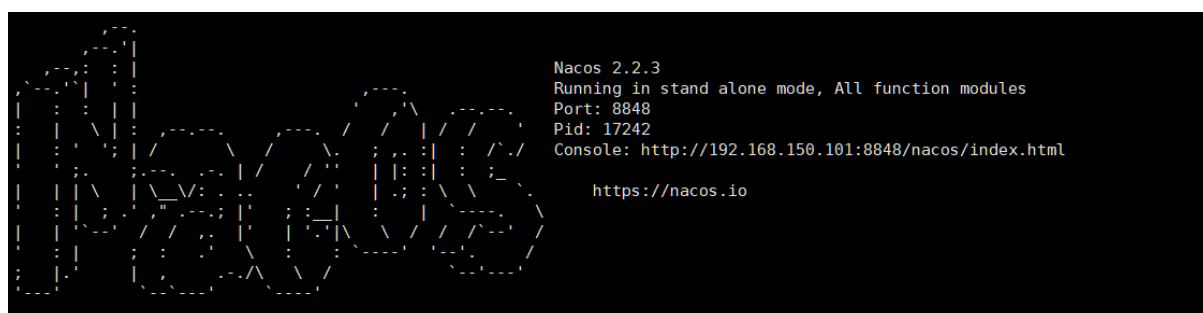
## 设置时间（毫秒），指定允许客户端配置的最长保持连接时间。典型的默认值是
#nacos.remote.server.grpc.cluster.permit-keep-alive-time=300000
```

我们一开始仅需关注权限认证和数据库配置的配置文件即可，后续的参数配置在使用时再进行探讨。接下来我们来进行启动nacos。我们先不开启认证，先进行体验，后续再进行开启认证。

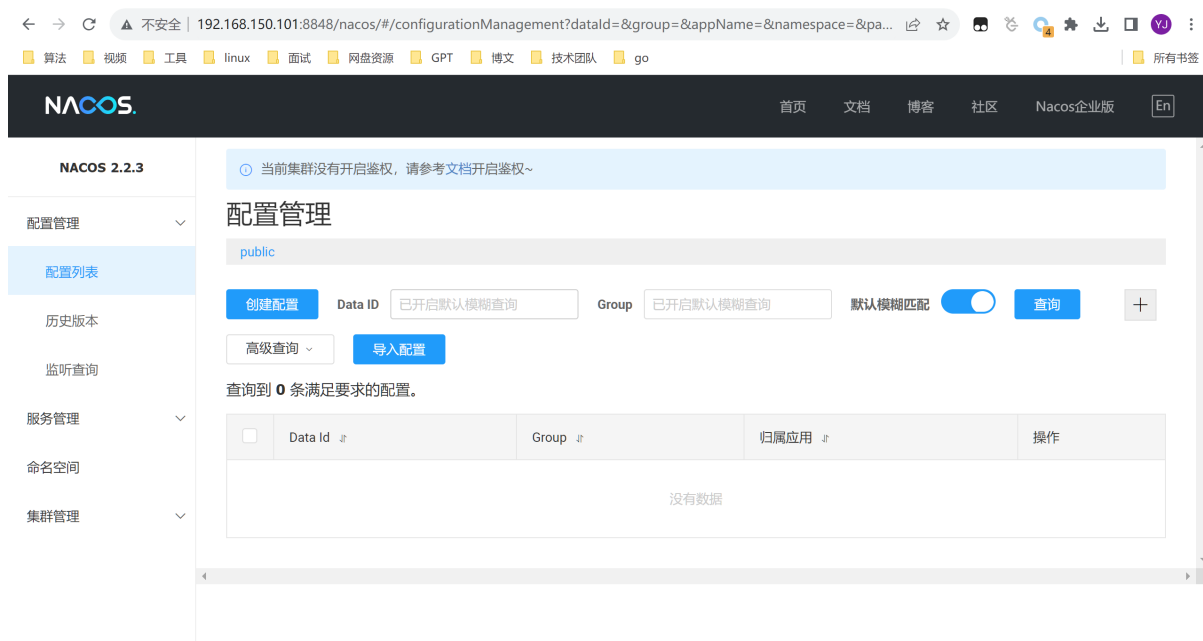
如果启动过程中出现libstdc++.so.6: cannot open shared object file: No such file or directory, 大概是共享库有所缺失, 我们可以通过一位网友的做法来进行解决。 [解决方案](#)

```
sh startup.sh -m standalone
```

```
[root@hadoop-father bin]# sh startup.sh -m standalone
/opt/jdk1.8.0_201/bin/java -Djava.ext.dirs=/opt/jdk1.8.0_201/jre/lib/ext:/opt/jdk1.8.0_201/lib/ext -Xms512m -Xmx512m -Xmn256m -Dnacos.standalone=true -Dnacos.member.list= -Xloggc:/soft/nacos/nacos/logs/nacos-gc-log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M -Dloader.path=/soft/nacos/nacos/plugins -/soft/nacos/nacos/plugins/health,/soft/nacos/nacos/plugins/cmd,/soft/nacos/nacos/plugins/selector -Dnacos.home=/soft/nacos/nacos -jar /soft/nacos/nacos/target/nacos-server.jar --spring.config.additional-location=file:/soft/nacos/nacos/conf/ --logging.config=/soft/nacos/nacos/conf/nacos-logback.xml --server.max-http-header-size=524288
nacos is starting with standalone
nacos is starting, you can check the /soft/nacos/nacos/logs/start.out
[root@hadoop-father bin]#
```



启动成功, 然后我们进行访问。



此时我们可以先自己进行稍微体验一下, 再然后我们将开始docker部署。