Nacos 中文文档

书栈(BookStack.CN)

目　录

致谢

当前文档 《Nacos 中文文档》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-11-22。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/Nacos

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

什么是 Nacos

概览

欢迎来到 Nacos 的世界！

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 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。

什么是 Nacos？

服务（Service）是 Nacos 世界的一等公民。Nacos 支持几乎所有主流类型的“服务”的发现、配置和管理：

Kubernetes Service

gRPC & Dubbo RPC Service

Spring Cloud RESTful Service

Nacos 的关键特性包括：

- 服务发现和服务健康监测

Nacos 支持基于 DNS 和基于 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DNS TODO 或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。

- 动态 **DNS** 服务

动态 DNS 服务支持权重路由，让您更容易地实现中间层负载均衡、更灵活的路由策略、流量控制以及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，以帮助您消除耦合到厂商私有服务发现 API 上的风险。

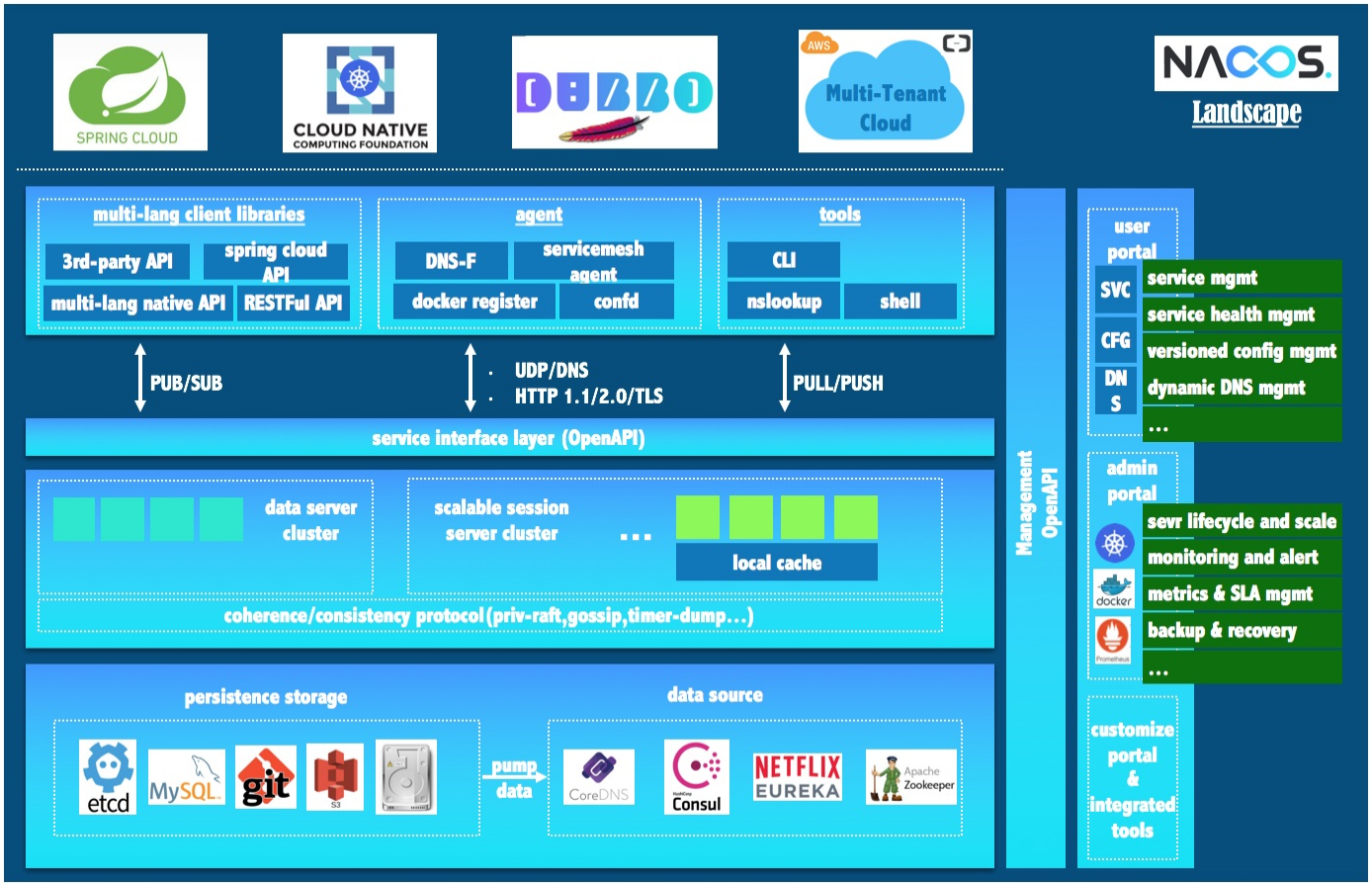
Nacos 提供了一些简单的 DNS APIs TODO 帮助您管理服务的关联域名和可用的 IP:PORT 列表.

- 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。

- 更多的特性列表 ...

Nacos 全景图



如 Nacos 全景图所示，Nacos 无缝支持一些主流的开源生态，例如

- Spring Cloud
- Apache Dubbo and Dubbo Mesh TODO
- Kubernetes and CNCF TODO。
  使用 Nacos 简化服务发现、配置管理、服务治理及管理的解决方案，让微服务的发现、管理、共享、组合更加容易。

关于如何在这些生态中使用 Nacos，请参考以下文档：

Nacos与Spring Cloud一起使用

Nacos与Kubernetes一起使用

Nacos与Dubbo一起使用

Nacos与gRPC一起使用

Nacos与Istio一起使用

下一步

继续阅读 快速开始 以快速上手 Nacos。

> 原文: *https://nacos.io/zh-cn/docs/what-is-nacos.html*

Nacos 概念 (Concepts)

> *NOTE: Nacos* 引入了一些基本的概念，系统性的了解一下这些概念可以帮助您更好的理解和正确的使用*Nacos* 产品。

地域 （Region）

物理的数据中心，资源创建成功后不能更换。

可用区（Available Zone）

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。

接入点（Endpoint）

地域的某个服务的入口域名。

命名空间（Namespace）

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。

配置 (Configuration)

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。

配置管理 (Configuration Management)

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。

配置项 (Configuration Item)

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。

配置集 (Configuration Set)

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（Data 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（Group）

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。

配置快照（Configuration Snapshot）

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。

服务（Service）

通过预定义接口网络访问的提供给客户端的软件功能。

服务名（Service Name）

服务提供的标识，通过该标识可以唯一确定其指代的服务。

服务注册中心（Service Registry）

存储服务实例和服务负载均衡策略的数据库。

服务发现（Service Discovery）

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。

元信息（Metadata）

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。

应用（Application）

用于标识服务提供方的服务的属性。

服务分组（Service Group）

不同的服务可以归类到同一分组。

虚拟集群（Virtual Cluster）

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。

实例（Instance）

提供一个或多个服务的具有可访问网络地址（IP:Port）的进程。

权重（Weight）

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。

健康检查（Health Check）

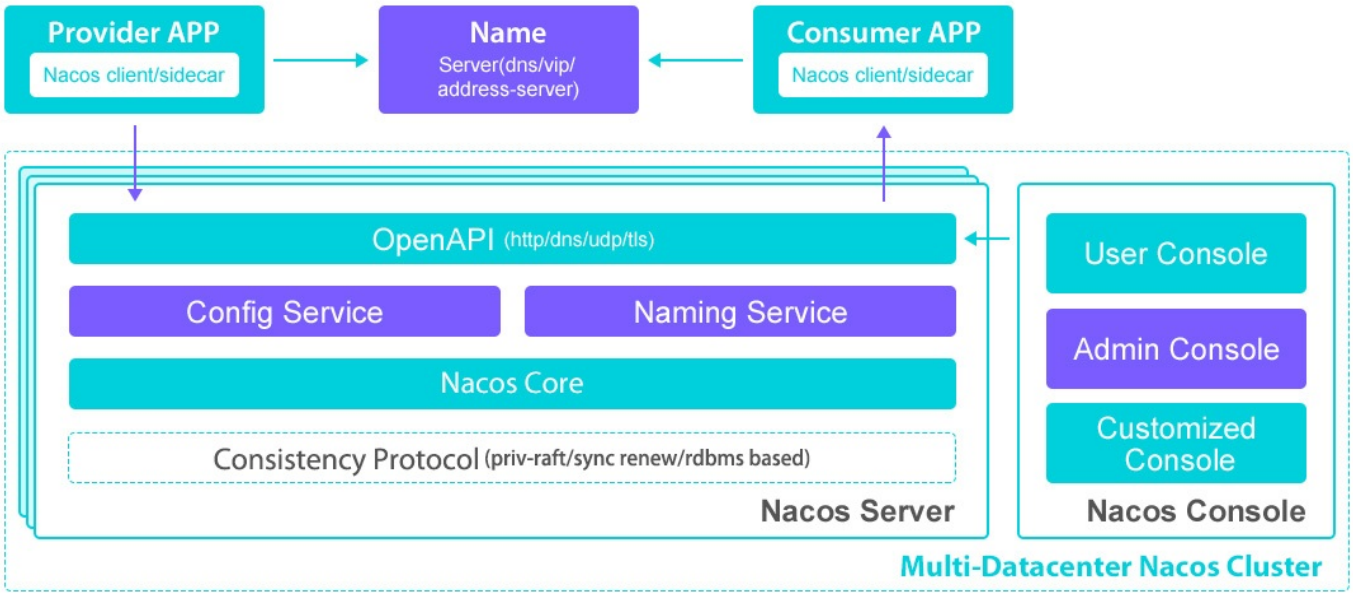
以指定方式检查服务下挂载的实例 (Instance) 的健康度，从而确认该实例 (Instance) 是否能提供服务。根据检查结果，实例 (Instance) 会被判断为健康或不健康。对服务发起解析请求时，不健康的实例 (Instance) 不会返回给客户端。

健康保护阈值（Protect Threshold）

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 (Instance) 占总服务实例 (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，但是保证了集群的剩余健康实例 (Instance) 能正常工作。

> 原文: *https://nacos.io/zh-cn/docs/concepts.html*

I. 基本架构及概念



- 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，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)**

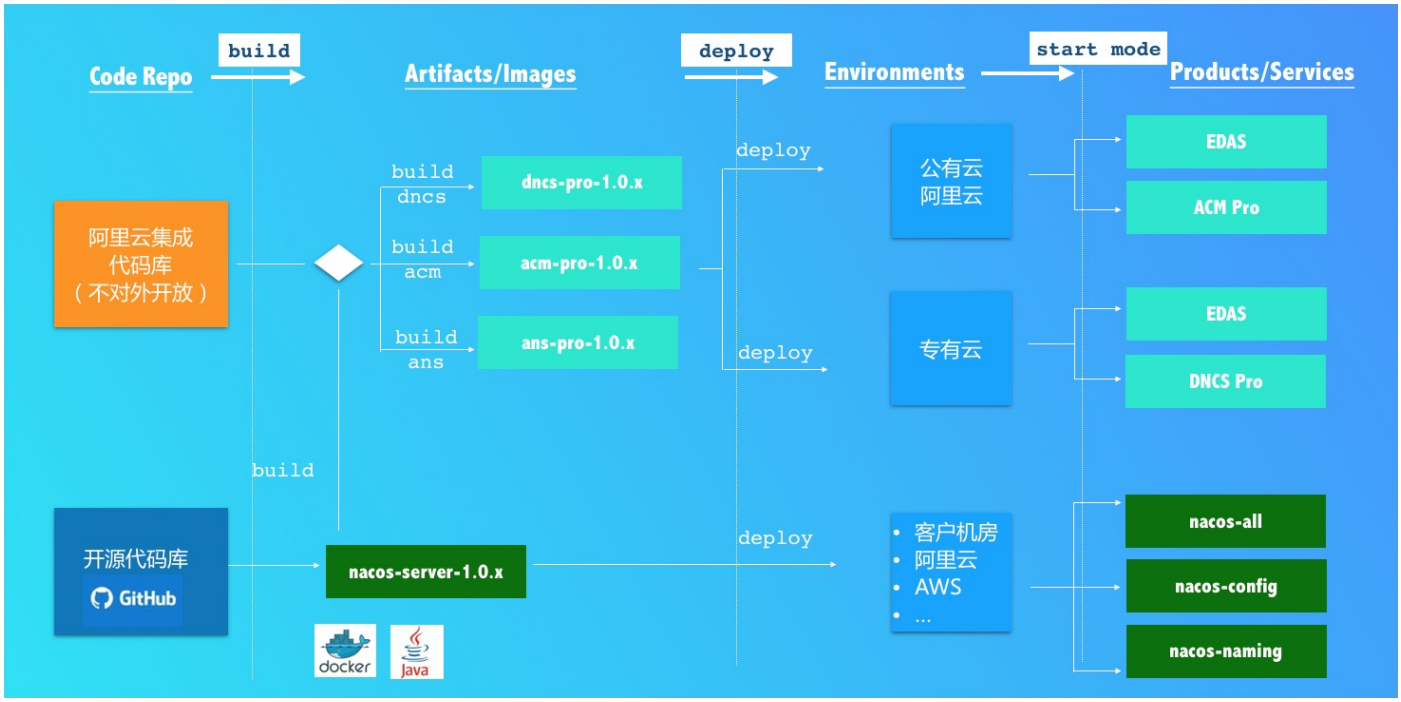
提供分布式系统中所有对象(Object)、实体(Entity)的“名字”到关联的元数据之间的映射管理服务，例如 ServiceName -> Endpoints Info, Distributed Lock Name -> Lock Owner/Status Info, DNS Domain Name -> IP List, 服务发现和 DNS 就是名字服务的2大场景。

- 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。

- 更多概念...

II. 构建物、部署及启动模式



- **2**种交付工件

Nacos 支持标准 Docker 镜像(TODO: 0.2版本开始支持）及 zip(tar.gz)压缩包的构建物。

- 启动模式

Nacos 支持将注册中心(Service Registry）与配置中心(Config Center) 在一个进程合并部署或者将2者分离部署的两种模式。

- 免费的公有云服务模式

除了您自己部署和启动 Nacos 服务之外，在云计算时代，Nacos 也支持公有云模式，在阿里云公有云的商业产品（如ACM, EDAS）中会提供 Nacos 的免费的公有云服务。我们也欢迎和支持其他的公有云提供商提供 Nacos 的公有云服务。

> 原文: *https://nacos.io/zh-cn/docs/architecture.html*

这个快速开始手册是帮忙您快速在您的电脑上，下载、安装并使用 Nacos。

1.预备环境准备

Nacos 依赖 Java 环境来运行。如果您是从代码开始构建并运行Nacos，还需要为此配置 Maven环境，请确保是在以下版本环境中安装使用：

- 64 bit OS，支持 Linux/Unix/Mac/Windows，推荐选用 Linux/Unix/Mac。
- 64 bit JDK 1.8+；下载 & 配置。
- Maven 3.2.x+；下载 & 配置。

2.下载源码或者安装包

你可以通过源码和发行包两种方式来获取 Nacos。

从 Github 上下载源码方式

```
git clone https://github.com/alibaba/nacos.git
cd nacos/
mvn -Prelease-nacos clean install -U
ls -al distribution/target/

// change the $version to your actual path
cd distribution/target/nacos-server-$version/nacos/bin

```

下载编译后压缩包方式

您可以从 最新稳定版本 下载 `nacos-server-$version.zip` 包。

```
  unzip nacos-server-$version.zip 或者 tar -xvf nacos-server-$version.tar.gz
  cd nacos/bin
```

3.启动服务器

Linux/Unix/Mac

启动命令(standalone代表着单机模式运行，非集群模式)：

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

Windows

启动命令：

```
cmd startup.cmd
```

或者双击startup.cmd运行文件。

4.服务注册&发现和配置管理

服务注册

```
curl -X PUT 'http://127.0.0.1:8848/nacos/v1/ns/instance?serviceName=nacos.naming.serviceName&ip=20.18.7.10&port=8080&#39;
```

服务发现

```
curl -X GET 'http://127.0.0.1:8848/nacos/v1/ns/instances?serviceName=nacos.naming.serviceName&#39;
```

发布配置

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test&content=HelloWorld&#34;
```

获取配置

```
curl -X GET "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test&#34;
```

5.关闭服务器

Linux/Unix/Mac

```
sh shutdown.sh
```

Windows

```
cmd shutdown.cmd
```

或者双击shutdown.cmd运行文件。

> 原文: *https://nacos.io/zh-cn/docs/quick-start.html*

本文主要面向 Spring 的使用者，通过两个示例来介绍如何使用 Nacos 来实现分布式环境下的配置管理和服务发现。

- 通过 Nacos server 和 Nacos Spring 配置管理模块，实现配置的动态变更；
- 通过 Nacos server 和 Nacos Spring 服务发现模块，实现服务的注册与发现。

前提条件

您需要先下载 Nacos 并启动 Nacos server。操作步骤参见 Nacos 快速入门。

启动配置管理

启动了 Nacos server 后，您就可以参考以下示例代码，为您的 Spring 应用启动 Nacos 配置管理服务了。完整示例代码请参考：nacos-spring-config-example

- 添加依赖。

```
<dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-spring-context</artifactId>
    <version>${latest.version}</version>
</dependency>
```

- 添加 `@EnableNacosConfig` 注解启用 Nacos Spring 的配置管理服务。以下示例中，我们使用 `@NacosPropertySource` 加载了 `dataId` 为 `example` 的配置源，并开启自动更新：

```
@Configuration
@EnableNacosConfig(globalProperties = @NacosProperties(serverAddr = "127.0.0.1:8848"))
@NacosPropertySource(dataId = "example", autoRefreshed = true)
public class NacosConfiguration {

}
```

- 通过 Spring 的 `@Value` 注解设置属性值。
  注意：需要同时有 `Setter` 方法才能在配置变更的时候自动更新。

```
@Controller
@RequestMapping("config")
public class ConfigController {

    @Value("${useLocalCache:false}")
    private boolean useLocalCache;

    public void setUseLocalCache(boolean useLocalCache) {
        this.useLocalCache = useLocalCache;
    }

    @RequestMapping(value = "/get", method = GET)
```

```
    @ResponseBody
    public boolean get() {
        return useLocalCache;
    }
}
```

- 启动 Tomcat，调用 `curl http://localhost:8080/config/get` 尝试获取配置信息。由于此时还未发布过配置，所以返回内容是 `false` 。
- 通过调用 Nacos Open API 向 Nacos Server 发布配置：dataId 为 `example` ，内容为 `useLocalCache=true`

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?
dataId=example&group=DEFAULT_GROUP&content=useLocalCache=true"
```

- 再次访问 `http://localhost:8080/config/get` ，此时返回内容为 `true` ，说明程序中的 `useLocalCache` 值已经被动态更新了。

启动服务发现

本节演示如何在您的 Spring 项目中启动 Nacos 的服务发现功能。完整示例代码请参考：nacos-spring-discovery-example

- 添加依赖。

```
<dependency>
    <groupId>com.alibaba.nacos</groupId>
    <artifactId>nacos-spring-context</artifactId>
    <version>${latest.version}</version>
</dependency>
```

- 通过添加 `@EnableNacosDiscovery` 注解开启 Nacos Spring 的服务发现功能：

```
@Configuration
@EnableNacosDiscovery(globalProperties = @NacosProperties(serverAddr = "127.0.0.1:8848"))
public class NacosConfiguration {

}
```

- 使用 `@NacosInjected` 注入 Nacos 的 `NamingService` 实例：

```
@Controller
@RequestMapping("discovery")
public class DiscoveryController {

    @NacosInjected
    private NamingService namingService;

```

```
    @RequestMapping(value = "/get", method = GET)
    @ResponseBody
    public List<Instance> get(@RequestParam String serviceName) throws NacosException {
        return namingService.getAllInstances(serviceName);
    }
}
```

- 启动 Tomcat，调用 `curl http://localhost:8080/discovery/get?serviceName=example`，此时返回为空 JSON 数组 `[]`。
- 通过调用 Nacos Open API 向 Nacos server 注册一个名称为 `example` 服务。

```
curl -X PUT 'http://127.0.0.1:8848/nacos/v1/ns/instance?serviceName=example&ip=127.0.0.1&port=8080'
```

- 再次访问 `curl http://localhost:8080/discovery/get?serviceName=example`，此时返回内容为：

```
[
  {
    "instanceId": "127.0.0.1#8080#DEFAULT#example",
    "ip": "127.0.0.1",
    "port": 8080,
    "weight": 1.0,
    "healthy": true,
    "cluster": {
      "serviceName": null,
      "name": "",
      "healthChecker": {
        "type": "TCP"
      },
      "defaultPort": 80,
      "defaultCheckPort": 80,
      "useIPPort4Check": true,
      "metadata": {}
    },
    "service": null,
    "metadata": {}
  }
]
```

相关项目

- Nacos
- Nacos Spring
- Nacos Spring Boot
- Spring Cloud Alibaba

> 原文: *https://nacos.io/zh-cn/docs/quick-start-spring.html*

本文主要面向 Spring Boot 的使用者，通过两个示例来介绍如何使用 Nacos 来实现分布式环境下的配置管理和服务发现。

- 通过 Nacos Server 和 nacos-config-spring-boot-starter 实现配置的动态变更；
- 通过 Nacos Server 和 nacos-discovery-spring-boot-starter 实现服务的注册与发现。

前提条件

您需要先下载 Nacos 并启动 Nacos server。操作步骤参见 Nacos 快速入门。

启动配置管理

启动了 Nacos server 后，您就可以参考以下示例代码，为您的 Spring Boot 应用启动 Nacos 配置管理服务了。完整示例代码请参考：nacos-spring-boot-config-example

- 添加依赖。

```
<dependency>
    <groupId>com.alibaba.boot</groupId>
    <artifactId>nacos-config-spring-boot-starter</artifactId>
    <version>${latest.version}</version>
</dependency>
```

注意：版本 0.2.x.RELEASE 对应的是 Spring Boot 2.x 版本，版本 0.1.x.RELEASE 对应的是 Spring Boot 1.x 版本。

- 在 `application.properties` 中配置 Nacos server 的地址：

```
nacos.config.server-addr=127.0.0.1:8848
```

- 使用 `@NacosPropertySource` 加载 `dataId` 为 `example` 的配置源，并开启自动更新：

```
@SpringBootApplication
@NacosPropertySource(dataId = "example", autoRefreshed = true)
public class NacosConfigApplication {

    public static void main(String[] args) {
        SpringApplication.run(NacosConfigApplication.class, args);
    }
}
```

- 通过 Spring 的 `@Value` 注解设置属性值。
  注意：需要同时有 `Setter` 方法才能在配置变更的时候自动更新。

```
@Controller
@RequestMapping("config")
public class ConfigController {
```

```
4.
5.      @Value("${useLocalCache:false}")
6.      private boolean useLocalCache;
7.
8.      public void setUseLocalCache(boolean useLocalCache) {
9.          this.useLocalCache = useLocalCache;
10.     }
11.
12.     @RequestMapping(value = "/get", method = GET)
13.     @ResponseBody
14.     public boolean get() {
15.         return useLocalCache;
16.     }
17. }
```

- 启动 `NacosConfigApplication`，调用 `curl http://localhost:8080/config/get`，返回内容是 `false`。
- 通过调用 Nacos Open API 向 Nacos server 发布配置：dataId 为 `example`，内容为 `useLocalCache=true`

```
1. curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?
   dataId=example&group=DEFAULT_GROUP&content=useLocalCache=true"
```

- 再次访问 `http://localhost:8080/config/get`，此时返回内容为`true`，说明程序中的`useLocalCache`值已经被动态更新了。

启动服务发现

本节演示如何在您的 Spring Boot 项目中启动 Nacos 的服务发现功能。完整示例代码请参考：nacos-spring-boot-discovery-example

- 添加依赖。

```
1. <dependency>
2.     <groupId>com.alibaba.boot</groupId>
3.     <artifactId>nacos-discovery-spring-boot-starter</artifactId>
4.     <version>${latest.version}</version>
5. </dependency>
```

注意：版本 0.2.x.RELEASE 对应的是 Spring Boot 2.x 版本，版本 0.1.x.RELEASE 对应的是 Spring Boot 1.x 版本。

- 在 `application.properties` 中配置 Nacos server 的地址：

```
1. nacos.discovery.server-addr=127.0.0.1:8848
```

- 使用 `@NacosInjected` 注入 Nacos 的 `NamingService` 实例：

```java
@Controller
@RequestMapping("discovery")
public class DiscoveryController {

    @NacosInjected
    private NamingService namingService;

    @RequestMapping(value = "/get", method = GET)
    @ResponseBody
    public List<Instance> get(@RequestParam String serviceName) throws NacosException {
        return namingService.getAllInstances(serviceName);
    }
}

@SpringBootApplication
public class NacosDiscoveryApplication {

    public static void main(String[] args) {
        SpringApplication.run(NacosDiscoveryApplication.class, args);
    }
}
```

- 启动 `NacosDiscoveryApplication`，调用 `curl http://localhost:8080/discovery/get?serviceName=example`，此时返回为空 JSON 数组 `[]`。
- 通过调用 Nacos Open API 向 Nacos server 注册一个名称为 `example` 服务

```
curl -X PUT 'http://127.0.0.1:8848/nacos/v1/ns/instance?serviceName=example&ip=127.0.0.1&port=8080'
```

- 再次访问 `curl http://localhost:8080/discovery/get?serviceName=example`，此时返回内容为：

```
[
  {
    "instanceId": "127.0.0.1-8080-DEFAULT-example",
    "ip": "127.0.0.1",
    "port": 8080,
    "weight": 1.0,
    "healthy": true,
    "cluster": {
      "serviceName": null,
      "name": "",
      "healthChecker": {
        "type": "TCP"
      },
      "defaultPort": 80,
      "defaultCheckPort": 80,
      "useIPPort4Check": true,
      "metadata": {}
    },
    "service": null,
    "metadata": {}
```

```
  }
]
```

相关项目

- Nacos
- Nacos Spring
- Nacos Spring Boot
- Spring Cloud Alibaba

> 原文: *https://nacos.io/zh-cn/docs/quick-start-spring-boot.html*

本文主要面向 Spring Cloud 的使用者，通过两个示例来介绍如何使用 Nacos 来实现分布式环境下的配置管理和服务注册发现。

更详细的文档请参看：Nacos Config Example 和 Nacos Discovery Example。

- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-config 实现配置的动态变更。
- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-discovery 实现服务的注册与发现。

前提条件

您需要先下载 Nacos 并启动 Nacos server。操作步骤参见 Nacos 快速入门

启动配置管理

启动了 Nacos server 后，您就可以参考以下示例代码，为您的 Spring Cloud 应用启动 Nacos 配置管理服务了。完整示例代码请参考：nacos-spring-cloud-config-example

- 添加依赖：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
    <version>${latest.version}</version>
</dependency>
```

注意：版本 0.2.x.RELEASE 对应的是 Spring Boot 2.x 版本，版本 0.1.x.RELEASE 对应的是 Spring Boot 1.x 版本。

- 在 `bootstrap.properties` 中配置 Nacos server 的地址和应用名

```
spring.cloud.nacos.config.server-addr=127.0.0.1:8848

spring.application.name=example
```

说明：之所以需要配置 `spring.application.name` ，是因为它是构成 Nacos 配置管理 `dataId` 字段的一部分。

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，也可以通过配置项 `spring.cloud.nacos.config.prefix`来配置。
- `spring.profile.active` 即为当前环境对应的 profile，详情可以参考 Spring Boot文档。注意：当 `spring.profile.active` 为空时，对应的连接符 `-` 也将不存在，**dataId** 的拼接格式变成 **`${prefix}.${file-extension}`**
- `file-exetension` 为配置内容的数据格式，可以通过配置项 `spring.cloud.nacos.config.file-extension` 来配

置。目前只支持 `properties` 和 `yaml` 类型。

- 通过 Spring Cloud 原生注解 `@RefreshScope` 实现配置自动更新：

```
@RestController
@RequestMapping("/config")
@RefreshScope
public class ConfigController {

    @Value("${useLocalCache:false}")
    private boolean useLocalCache;

    @RequestMapping("/get")
    public boolean get() {
        return useLocalCache;
    }
}
```

- 首先通过调用 Nacos Open API 向 Nacos Server 发布配置：dataId 为`example.properties`，内容为`useLocalCache=true`

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=example.properties&group=DEFAULT_GROUP&content=useLocalCache=true"
```

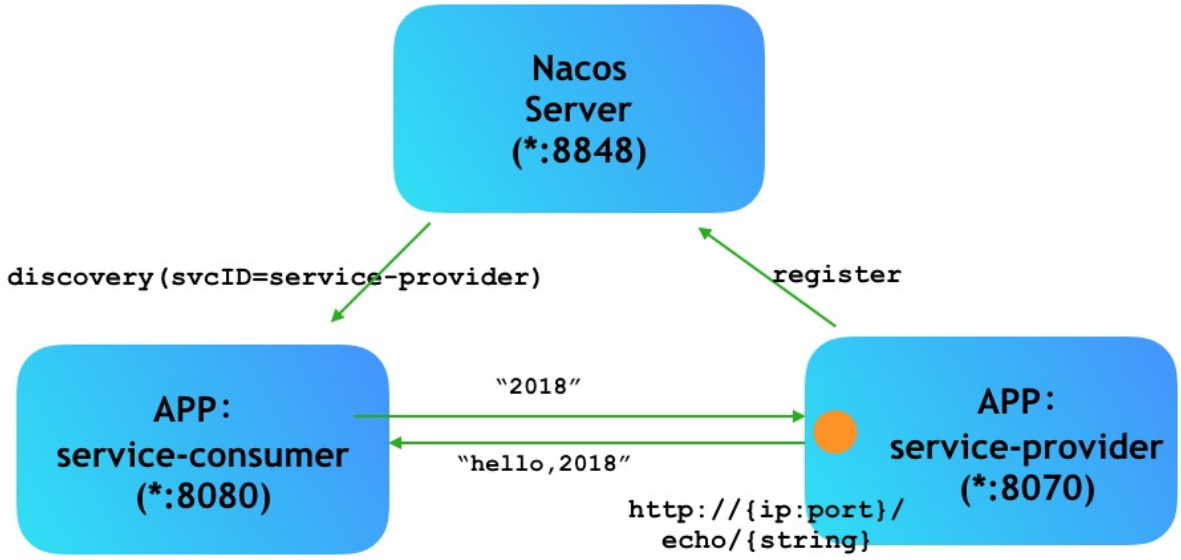
- 运行 `NacosConfigApplication`，调用 `curl http://localhost:8080/config/get`，返回内容是 `true`。
- 再次调用 Nacos Open API 向 Nacos server 发布配置：dataId 为 `example.properties`，内容为 `useLocalCache=false`

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=example.properties&group=DEFAULT_GROUP&content=useLocalCache=false"
```

- 再次访问 `http://localhost:8080/config/get`，此时返回内容为`false`，说明程序中的`useLocalCache`值已经被动态更新了。

启动服务发现

本节通过实现一个简单的 `echo service` 演示如何在您的 Spring Cloud 项目中启用 Nacos 的服务发现功能，如下图示：



完整示例代码请参考：nacos-spring-cloud-discovery-example

- 添加依赖：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    <version>${latest.version}</version>
</dependency>
```

注意：版本 0.2.x.RELEASE 对应的是 Spring Boot 2.x 版本，版本 0.1.x.RELEASE 对应的是 Spring Boot 1.x 版本。

- 配置服务提供者，从而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。
  i. 在 `application.properties` 中配置 Nacos server 的地址：

```
server.port=8070
spring.application.name=service-provider

spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

ii. 通过 Spring Cloud 原生注解 `@EnableDiscoveryClient` 开启服务注册发现功能：

```
@SpringBootApplication
@EnableDiscoveryClient
public class NacosProviderApplication {

    public static void main(String[] args) {
```

```
        SpringApplication.run(NacosProviderApplication.class, args);
    }

    @RestController
    class EchoController {
        @RequestMapping(value = "/echo/{string}", method = RequestMethod.GET)
        public String echo(@PathVariable String string) {
            return "Hello Nacos Discovery " + string;
        }
    }
}
```

- 配置服务消费者，从而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。
  i. 在 `application.properties` 中配置 Nacos server 的地址：

```
server.port=8080
spring.application.name=service-consumer

spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

ii. 通过 Spring Cloud 原生注解 `@EnableDiscoveryClient` 开启服务注册发现功能。给 RestTemplate 实例添加 `@LoadBalanced` 注解，开启 `@LoadBalanced` 与 Ribbon 的集成：

```
@SpringBootApplication
@EnableDiscoveryClient
public class NacosConsumerApplication {

    @LoadBalanced
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    public static void main(String[] args) {
        SpringApplication.run(NacosConsumerApplication.class, args);
    }

    @RestController
    public class TestController {

        private final RestTemplate restTemplate;

        @Autowired
        public TestController(RestTemplate restTemplate) {this.restTemplate = restTemplate;}

        @RequestMapping(value = "/echo/{str}", method = RequestMethod.GET)
        public String echo(@PathVariable String str) {
            return restTemplate.getForObject("http://service-provider/echo/" + str, String.class);
        }
    }
```

```
28. }
```

- 启动 `ProviderApplication` 和 `ConsumerApplication` ，调用 `http://localhost:8080/echo/2018`，返回内容为 `Hello Nacos Discovery 2018`。

相关项目

- Nacos
- Nacos Spring
- Nacos Spring Boot
- Spring Cloud Alibaba

> 原文: *https://nacos.io/zh-cn/docs/quick-start-spring-cloud.html*

- Java的SDK
- 其他语言的SDK
- Open-API指南

概述部分

maven坐标

```
<dependency>
<groupId>com.alibaba.nacos</groupId>
<artifactId>nacos-client</artifactId>
<version>${version}</version>
</dependency>
```

配置管理

获取配置

描述

用于服务启动的时候从 Nacos 获取配置。

```
public String getConfig(String dataId, String group, long timeoutMs) throws NacosException
```

请求参数

参数名	参数类型	描述
dataId	string	配置 ID，采用类似 package.class（如com.taobao.tc.refund.log.level）的命名规则保证全局唯一性，class 部分建议是配置的业务含义。全部字符小写。只允许英文字符和 4 种特殊字符（"."、":"、"-"、"_"），不超过 256 字节。
group	string	配置分组，建议填写产品名:模块名（Nacos:Test）保证唯一性，只允许英文字符和4种特殊字符（"."、":"、"-"、"_"），不超过128字节。
timeout	long	读取配置超时时间，单位 ms，推荐值 3000。

返回值

参数类型	描述
string	配置值

请求示例

```
try {
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
```

```
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);
    ConfigService configService = NacosFactory.createConfigService(properties);
    String content = configService.getConfig(dataId, group, 5000);
    System.out.println(content);
} catch (NacosException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 NacosException 异常。

监听配置

描述

如果希望 Nacos 推送配置变更，可以使用 Nacos 动态监听配置接口来实现。

```
public void addListener(String dataId, String group, Listener listener)
```

请求参数

参数名	参数类型	描述
dataId	string	配置 ID，采用类似 package.class（如com.taobao.tc.refund.log.level）的命名规则保证全局唯一性，class 部分建议是配置的业务含义。 全部字符小写。只允许英文字符和 4 种特殊字符（"."、":"、"-"、"_"）。不超过 256 字节。
group	string	

配置分组，建议填写产品名：模块名（如 Nacos:Test）保证唯一性。 只允许英文字符和4种特殊字符（"."、":"、"-"、"_"），不超过128字节。

|
listener
|
Listener
|
监听器，配置变更进入监听器的回调函数。

返回值

参数类型	描述
string	配置值，初始化或者配置变更的时候通过回调函数返回该值。

请求示例

```
String serverAddr = "{serverAddr}";
String dataId = "{dataId}";
String group = "{group}";
Properties properties = new Properties();
properties.put("serverAddr", serverAddr);
ConfigService configService = NacosFactory.createConfigService(properties);
String content = configService.getConfig(dataId, group, 5000);
System.out.println(content);
configService.addListener(dataId, group, new Listener() {
    @Override
    public void receiveConfigInfo(String configInfo) {
        System.out.println("recieve1:" + configInfo);
    }
    @Override
    public Executor getExecutor() {
        return null;
    }
});

// 测试让主线程不退出，因为订阅配置是守护线程，主线程退出守护线程就会退出。 正式代码中无需下面代码
while (true) {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

删除监听

描述

取消监听配置，取消监听后配置不会再推送。

```
1. public void removeListener(String dataId, String group, Listener listener)
```

请求参数

参数名	参数类型	描述
dataId	string	配置 ID，采用类似 package.class（如 com.taobao.tc.refund.log.level）的命名规则保证全局唯一性，class 部分建议是配置的业务含义。全部字符小写。只允许英文字符和 4 种特殊字符（"."、":"、"-"、"_"），不超过 256 字节。
listener	ConfigChangeListenerAdapter	监听器，配置变更进入监听器的回调函数。

使用示例

```
1. String serverAddr = "{serverAddr}";
2. String dataId = "{dataId}";
3. String group = "{group}";
4. Properties properties = new Properties();
5. properties.put("serverAddr", serverAddr);
6. ConfigService configService = NacosFactory.createConfigService(properties);
7. configService.removeListener(dataId, group, yourListener);
```

发布配置

描述

用于通过程序自动发布 Nacos 配置，以便通过自动化手段降低运维成本。

注意：创建和修改配置时使用的同一个发布接口，当配置不存在时会创建配置，当配置已存在时会更新配置。

```
1. public boolean publishConfig(String dataId, String group, String content) throws NacosException
2.
```

请求参数

参数名	参数类型	描述
dataId	string	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 class 部分。全部字符均为小写。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。

group	string	配置分组，建议填写 `产品名:模块名` （如 Nacos `:Test` ）来保证唯一性。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
content	string	配置内容，不超过 100K 字节。

返回参数

参数类型	描述
boolean	是否发布成功

请求示例

```
try {
    // 初始化配置服务，控制台通过示例代码自动获取下面参数
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);
    ConfigService configService = NacosFactory.createConfigService(properties);
    boolean isPublishOk = configService.publishConfig(dataId, group, "content");
    System.out.println(isPublishOk);
} catch (NacosException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 NacosException 异常。

删除配置

描述

用于通过程序自动删除 Nacos 配置，以便通过自动化手段降低运维成本。

注意： 当配置已存在时会删除该配置，当配置不存在时会直接返回成功消息。

```
public boolean removeConfig(String dataId, String group) throws NacosException

```

请求参数

参数名	参数类型	描述
dataId	string	配置 ID

group	string	配置分组

返回参数

参数类型	描述
boolean	是否删除成功

请求示例

```
try {
    // 初始化配置服务，控制台通过示例代码自动获取下面参数
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);

    ConfigService configService = NacosFactory.createConfigService(properties);
    boolean isRemoveOk = configService.removeConfig(dataId, group);
    System.out.println(isRemoveOk);
} catch (NacosException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 NacosException 异常。

服务发现SDK

注册实例

描述注册一个实例到服务。

```
void registerInstance(String serviceName, String ip, int port) throws NacosException;

void registerInstance(String serviceName, String ip, int port, String clusterName) throws NacosException;

void registerInstance(String serviceName, Instance instance) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
ip	字符串	服务实例IP
port	int	服务实例port
clusterName	字符串	集群名
instance	参见代码注释	实例属性

返回参数

无

请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.registerInstance("nacos.test.3", "11.11.11.11", 8888, "TEST1");

Instance instance = new Instance();
instance.setIp("55.55.55.55");
instance.setPort(9999);
instance.setHealthy(false);
instance.setWeight(2.0);
Map<String, String> instanceMeta = new HashMap<>();
instanceMeta.put("site", "et2");
instance.setMetadata(instanceMeta);

Service service = new Service("nacos.test.4");
service.setApp("nacos-naming");
service.sethealthCheckMode("server");
service.setEnableHealthCheck(true);
service.setProtectThreshold(0.8F);
service.setGroup("CNCF");
Map<String, String> serviceMeta = new HashMap<>();
serviceMeta.put("symmetricCall", "true");
service.setMetadata(serviceMeta);
instance.setService(service);

Cluster cluster = new Cluster();
cluster.setName("TEST5");
AbstractHealthChecker.Http healthChecker = new AbstractHealthChecker.Http();
healthChecker.setExpectedResponseCode(400);
healthChecker.setCurlHost("USer-Agent|Nacos");
healthChecker.setCurlPath("/xxx.html");
cluster.setHealthChecker(healthChecker);
Map<String, String> clusterMeta = new HashMap<>();
clusterMeta.put("xxx", "yyyy");
cluster.setMetadata(clusterMeta);

instance.setCluster(cluster);

```

```
37. naming.registerInstance("nacos.test.4", instance);
```

注销实例

描述

删除服务下的一个实例。

```
1. void deregisterInstance(String serviceName, String ip, int port) throws NacosException;
2.
3. void deregisterInstance(String serviceName, String ip, int port, String clusterName) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
ip	字符串	服务实例IP
port	int	服务实例port
clusterName	字符串	集群名

返回参数

无

请求示例

```
1. NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
2. naming.deregisterInstance("nacos.test.3", "11.11.11.11", 8888, "DEFAULT");
```

获取全部实例

描述

获取服务下的所有实例。

```
1. List<Instance> getAllInstances(String serviceName) throws NacosException;
2.
3. List<Instance> getAllInstances(String serviceName, List<String> clusters) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表

返回参数

List 实例列表。

请求示例

```
1. NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
2. System.out.println(naming.getAllInstances("nacos.test.3"));
```

获取健康或不健康实例列表

描述

根据条件获取过滤后的实例列表。

```
1. List<Instance> selectInstances(String serviceName, boolean healthy) throws NacosException;
2.
3. List<Instance> selectInstances(String serviceName, List<String> clusters, boolean healthy) throws
   NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表
healthy	boolean	是否健康

返回参数

List 实例列表。

请求示例

```
1. NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
2. System.out.println(naming.selectInstances("nacos.test.3", true));
```

获取一个健康实例

描述

根据负载均衡算法随机获取一个健康实例。

```
1. Instance selectOneHealthyInstance(String serviceName) throws NacosException;
2.
3. Instance selectOneHealthyInstance(String serviceName, List<String> clusters) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表

返回参数

Instance 实例。

请求示例

```
1. NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
2. System.out.println(naming.selectOneHealthyInstance("nacos.test.3"));
```

监听服务

描述

监听服务下的实例列表变化。

```
1. void subscribe(String serviceName, EventListener listener) throws NacosException;
2.
3. void subscribe(String serviceName, List<String> clusters, EventListener listener) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表
listener	EventListener	回调listener

返回参数

无

请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.subscribe("nacos.test.3", event -> {
    if (event instanceof NamingEvent) {
        System.out.println(((NamingEvent) event).getServceName());
        System.out.println(((NamingEvent) event).getInstances());
    }
});
```

取消监听服务

描述

取消监听服务下的实例列表变化。

```
void unsubscribe(String serviceName, EventListener listener) throws NacosException;

void unsubscribe(String serviceName, List<String> clusters, EventListener listener) throws NacosException;
```

请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表
listener	EventListener	回调listener

返回参数

无

请求示例

```

NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.unsubscribe("nacos.test.3", event -> {});

```

> 原文: *https://nacos.io/zh-cn/docs/sdk.html*

Nacos社区当前仅提供了Java版本的客户端，我们将主要依靠社区的贡献来发展多语言客户端。在未来，我们将向Nacos社区用户推荐那些最被广泛使用的以及支持最好的客户端作为Nacos相应语言的官方版本。

- go - IN PLAN
- cpp - IN PLAN
- python - IN PLAN
- nodejs - IN PLAN
- more …

> 原文: *https://nacos.io/zh-cn/docs/other-language.html*

配置管理

获取配置

描述

获取Nacos上的配置。

请求类型

GET

请求URL

/nacos/v1/cs/configs

请求参数

名称	类型	是否必须	描述
tenant	string	否	租户信息，对应 Nacos 的命名空间字段。
dataId	string	是	配置 ID。
group	string	是	配置分组。

返回参数

参数类型	描述
string	配置值

错误编码

错误代码	描述	语义
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	无法找到资源
500	Internal Server Error	服务器内部错误
200	OK	正常

示例

- 请求示例

```
http:serverIp:8848/nacos/v1/cs/configs?dataId=dataIdparam&group=groupParam&tenant=tenantParam

```

- 返回示例

```
contentTest
```

监听配置

描述

监听 Nacos 上的配置，以便实时感知配置变更。如果配置变更，则用获取配置接口获取配置的最新值，动态刷新本地缓存。

注册监听采用的是异步 Servlet 技术。注册监听本质就是带着配置和配置值的 MD5 值和后台对比。如果 MD5 值不一致，就立即返回不一致的配置。如果值一致，就等待住 30 秒。返回值为空。

请求类型

POST

请求URL

/nacos/v1/cs/configs/listener

请求参数

|
名称
|
类型
|
是否必须
|
描述

|
Listening-Configs
|
string
|
是
|

监听数据报文。格式为 dataId^2Group^2contentMD5^2tenant^1或者dataId^2Group^2contentMD5^1。

- dataId: 配置 ID
- group: 配置分组
- contentMD5: 配置内容 MD5 值
- tenant: 租户信息，对应 Nacos 的命名空间字段(非必填)

Header 参数

名称	类型	是否必须	描述
Long-Pulling-Timeout	string	是	长轮训等待 30s，此处填写 30000。

参数说明

- 配置多个字段间分隔符: ^2 = Character.toString((char) 2
- 配置间分隔符: ^1 = Character.toString((char) 1)
- contentMD5: MD5(content)，第一次本地缓存为空，所以这块为空串

返回参数

参数类型	描述
string	配置值

错误编码

错误代码	描述	语义
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	无法找到资源
500	Internal Server Error	服务器内部错误
200	OK	正常

示例

- 请求示例

```
http://serverIp:8848/nacos/v1/cs/configs/listener

POST 请求体数据内容：

Listening-Configs=dataId^2group^2contentMD5^2tenant^1
```

- 返回示例

```
1. 如果配置变化
2.
3. dataId^2group^2tenant^1
4.
5. 如果配置无变化：会返回空串
```

发布配置

描述

发布 Nacos 上的配置。

请求类型

POST

请求 URL

/nacos/v1/cs/configs

请求参数

名称	类型	是否必须	描述
tenant	string	否	租户信息，对应 Nacos 的命名空间字段
dataId	string	是	配置 ID
group	string	是	配置分组
content	string	是	配置内容

返回参数

参数类型	描述
boolean	是否发布成功

错误编码

错误代码	描述	语义
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	无法找到资源
500	Internal Server Error	服务器内部错误

200	OK	正常

示例

- 请求示例

```
http:serverIp:8848/nacos/v1/cs/configs

http body:
dataId=dataIdparam&group=groupParam&tenant=tenantParam&content=contentParam

```

- 返回示例

```
true
```

删除配置

描述

删除 Nacos 上的配置。

请求类型

DELETE

请求 URL

/nacos/v1/cs/configs

请求参数

名称	类型	是否必须	描述
tenant	string	否	租户信息，对应 Naocs 的命名空间字段
dataId	string	是	配置 ID
group	string	是	配置分组

返回参数

参数类型	描述
boolean	是否删除成功

错误编码

错误代码	描述	语义
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	无法找到资源
500	Internal Server Error	服务器内部错误
200	OK	正常

示例

- 请求示例

```
1. http:serverIp:8848/nacos/v1/cs/configs?dataId=dataIdparam&group=groupParam
2.
```

- 返回示例

```
1. true
```

服务发现API

注册实例

描述

注册一个实例到服务。

请求类型

PUT

请求路径

```
1. /nacos/v1/ns/instance
```

请求参数

名称	类型	是否必选	描述
ip	字符串	是	服务实例IP

port	int	是	服务实例port
tenant	字符串	否	租户ID
weight	double	否	权重
enable	boolean	否	是否上线
healthy	boolean	否	是否健康
metadata	字符串	否	扩展信息
clusterName	字符串	否	集群名
serviceName	字符串	是	服务名

示例请求

```
1. curl -X PUT '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.3&encoding=GBK&tenant=n1''
```

示例返回

ok

删除实例

描述

删除服务下的一个实例。

请求类型

DELETE

请求路径

```
1. /nacos/v1/ns/instance
```

请求参数

名称	类型	是否必选	描述
serviceName	字符串	是	服务名
ip	字符串	是	服务实例IP
port	int	是	服务实例port
cluster	字符串	是	集群名称
tenant	字符串	否	租户ID

示例请求

```
curl -X DELETE 127.0.0.1:8848/nacos/v1/ns/instance?serviceName=nacos.test.1&ip=1.1.1.1&port=8888&cluster=TEST1
```

示例返回

ok

修改实例

描述

修改服务下的一个实例。

请求类型

POST

请求路径

```
/nacos/v1/ns/instance
```

请求参数

名称	类型	是否必选	描述
serviceName	字符串	是	服务名
ip	字符串	是	服务实例IP
port	int	是	服务实例port
cluster	字符串	是	集群名称
tenant	字符串	否	租户ID
weight	double	否	权重
metadata	JSON	否	扩展信息

示例请求

```
curl -X POST 127.0.0.1:8848/nacos/v1/ns/instance?
serviceName=nacos.test.1&ip=1.1.1.1&port=8888&cluster=TEST1&weight=8&metadata={}
```

示例返回

ok

查询实例列表

描述

查询服务下的实例列表

请求类型

GET

请求路径

```
/nacos/v1/ns/instance/list
```

请求参数

名称	类型	是否必选	描述
serviceName	字符串	是	服务名
tenant	字符串	否	租户ID
clusters	字符串，多个集群用逗号分隔	否	集群名称
healthyOnly	boolean	否，默认为false	是否只返回健康实例

示例请求

```
curl -X GET 127.0.0.1:8848/nacos/v1/ns/instance/list?serviceName=nacos.test.1
```

示例返回

```
{
    "dom": "nacos.test.1",
    "cacheMillis": 1000,
    "useSpecifiedURL": false,
    "hosts": [{
        "valid": true,
        "marked": false,
        "instanceId": "10.10.10.10-8888-DEFAULT-nacos.test.1",
        "port": 8888,
        "ip": "10.10.10.10",
        "weight": 1.0,
        "metadata": {}
    }],
```

```
    "checksum": "3bbcf6dd1175203a8afdade0e77a27cd1528787794594",
    "lastRefTime": 1528787794594,
    "env": "",
    "clusters": ""
}
```

查询实例详情

描述

查询一个服务下个某个实例详情。

请求类型

GET

请求路径

```
/nacos/v1/ns/instance
```

请求参数

名称	类型	是否必选	描述
serviceName	字符串	是	服务名
ip	字符串	是	实例IP
port	字符串	是	实例端口
tenant	字符串	否	租户ID
clusters	字符串，多个集群用逗号分隔	否	集群名称
healthyOnly	boolean	否，默认为false	是否只返回健康实例

示例请求

```
curl -X GET '127.0.0.1:8848/nacos/v1/ns/instance?
serviceName=nacos.test.2&ip=10.10.10.10&port=8888&cluster=DEFAULT'
```

示例返回

```
{
    "metadata": {},
    "instanceId": "10.10.10.10-8888-DEFAULT-nacos.test.2",
    "port": 8888,
    "service": "nacos.test.2",
```

```
    "healthy": false,
    "ip": "10.10.10.10",
    "clusterName": "DEFAULT",
    "weight": 1.0
}
```

> 原文: *https://nacos.io/zh-cn/docs/open-API.html*

- 部署手册
- 集群部署说明
- 运维API
- 命令行手册
- 控制台手册

Nacos支持三种部署模式

- 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。

单机模式下运行Nacos

Linux/Unix/Mac

- Standalone means it is non-cluster Mode. *sh startup.sh -m standalone

Windows

cmd startup.cmd或者双击 startup.cmd 文件

集群模式下运行Nacos

Linux/Unix/Mac

sh startup.sh

Deploy Nacos in Multi-Cluster Mode

Nacos支持NameServer路由请求模式，通过它您可以设计一个有用的映射规则来控制请求转发到相应的集群，在映射规则中您可以按命名空间或租户等分片请求…

TODO

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

集群模式部署

这个快速开始手册是帮忙您快速在你的电脑上，下载安装并使用Nacos，部署生产使用的集群模式。

1.预备环境准备

请确保是在环境中安装使用:

- 64 bit OS Linux/Unix/Mac，推荐使用Linux系统。
- 64 bit JDK 1.8+; 下载.配置。
- Maven 3.2.x+; 下载.配置。

2.下载源码或者安装包

你可以通过两种方式来获取 Nacos。

从 Github 上下载源码方式

```
unzip nacos-source.zip
cd nacos/
mvn -Prelease-nacos clean install -U
cd nacos/distribution/target/nacos-server-0.2.1/nacos/bin
```

下载编译后压缩包方式

下载地址

zip包

tar.gz包

```
  unzip nacos-server-0.2.1.zip 或者 tar -xvf nacos-server-0.2.1.tar.gz
  cd nacos/bin
```

3.配置集群配置文件

在nacos的解压目录nacos/的conf目录下，有配置文件cluster.conf，请每行配置成ip:port。

```
# ip:port
200.8.9.16:8848
200.8.9.17:8848
200.8.9.18:8848
```

4.配置mysql数据库

生产使用建议至少主备模式，或者采用高可用数据库。

初始化mysql数据库

sql语句源文件

application.properties 配置

application.properties配置文件

5.启动服务器

Linux/Unix/Mac

启动命令(在没有参数模式，是集群模式)：

```
sh startup.sh
```

6.服务注册&发现和配置管理

服务注册

```
curl -X PUT 'http://127.0.0.1:8848/nacos/v1/ns/instance?serviceName=nacos.naming.serviceName&ip=20.18.7.10&port=8080&#39;
```

服务发现

```
curl -X GET 'http://127.0.0.1:8848/nacos/v1/ns/instances?serviceName=nacos.naming.serviceName&#39;
```

发布配置

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test&content=helloWorld&#34;
```

获取配置

```
curl -X GET "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test&#34;
```

5.关闭服务器

Linux/Unix/Mac

```
sh shutdown.sh
```

> 原文：*https://nacos.io/zh-cn/docs/cluster-mode-quick-start.html*

IN PLAN with Nacos 0.7.0

> 原文: *https://nacos.io/zh-cn/docs/managementAPI.html*

IN PLAN with Nacos 0.7.0

> 原文: *https://nacos.io/zh-cn/docs/CLI-guide.html*

Dubbo Nacos 0.3.0发布，代码仓库在https://github.com/alibaba/nacos，该版本主要旨在于增强对于服务列表，健康状态管理，服务治理，分布式配置管理等方面的管控能力，以便进一步帮助用户降低管理微服务应用架构的成本，在第一版的 UI 功能规划中，将提供包括下列基本功能：

- 服务管理
    - 服务列表及服务健康状态展示
    - 服务元数据存储及编辑
    - 服务流量权重的调整
    - 服务优雅上下线
- 配置管理
    - 多种配置格式编辑
    - 编辑DIFF
    - 示例代码
    - 推送状态查询
    - 配置版本及一键回滚
- 命名空间

特性详解

服务管理

开发者或者运维人员往往需要在服务注册后，通过友好的界面来查看服务的注册情况，包括当前系统注册的所有服务和每个服务的详情。并在有权限控制的情况下，进行服务的一些配置的编辑操作。Nacos在这个版本开放的控制台的服务发现部分，主要就是提供用户一个基本的运维页面，能够查看、编辑当前注册的服务。

服务列表管理

服务列表帮助用户以统一的视图管理其所有的微服务以及服务健康状态。整体界面布局是左上角有服务的搜索框和搜索按钮，页面中央是服务列表的展示。服务列表主要展示服务名、集群数目、实例数目、健康实例数目和详情按钮五个栏目。

在服务列表页面点击详情，可以看到服务的详情。可以查看服务、集群和实例的基本信息。

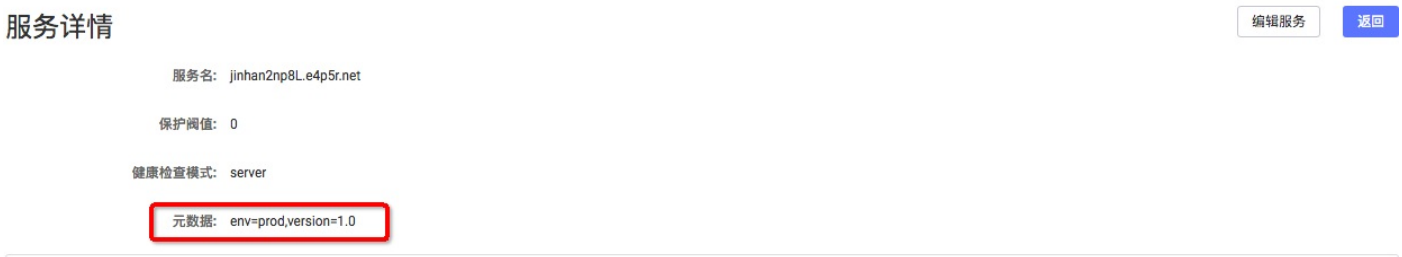
服务流量权重支持及流量保护

Nacos 为用户提供了流量权重控制的能力，同时开放了服务流量的阈值保护，以帮助用户更好的保护服务服务提供者集群不被意外打垮。如下图所以，可以点击实例的编辑按钮，修改实例的权重。如果想增加实例的流量，可以将权重调大，如果不想实例接收流量，则可以将权重设为0。

服务元数据管理

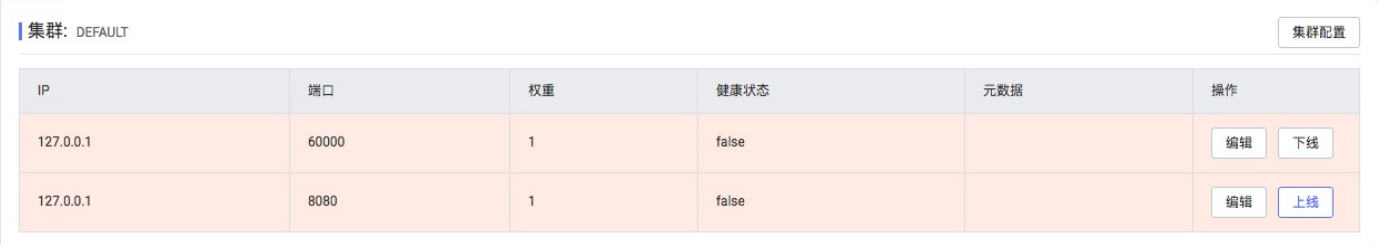
Nacos提供多个维度的服务元数据的暴露，帮助用户存储自定义的信息。这些信息都是以K-V的数据结构存储，在控制台上，会以k1=v1,k2=v2这样的格式展示。类似的，编辑元数据可以通过相同的格式进行。例如服务的元数据编辑，首先点击服务详情页右上角的“编辑服务”按钮，然后在元数据输入框输入：version=1.0,env=prod。

点击确认，就可以在服务详情页面，看到服务的元数据已经更新了。



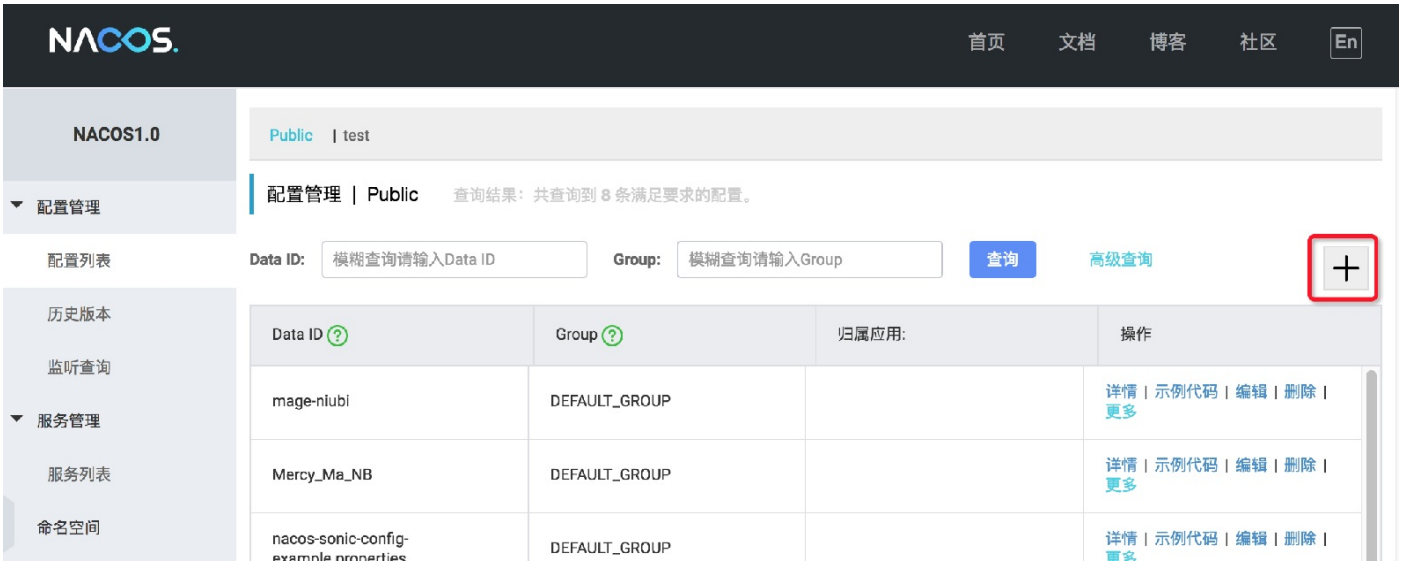
服务优雅上下线

Nacos还提供服务实例的上下线操作，在服务详情页面，可以点击实例的“上线”或者“下线”按钮，被下线的实例，将不会包含在健康的实例列表里。



配置管理

Nacos支持基于Namespace和Group的配置分组管理，以便用户更灵活的根据自己的需要按照环境或者应用、模块等分组管理微服务以及Spring的大量配置，在配置管理中主要提供了配置历史版本、回滚、订阅者查询等核心管理能力。

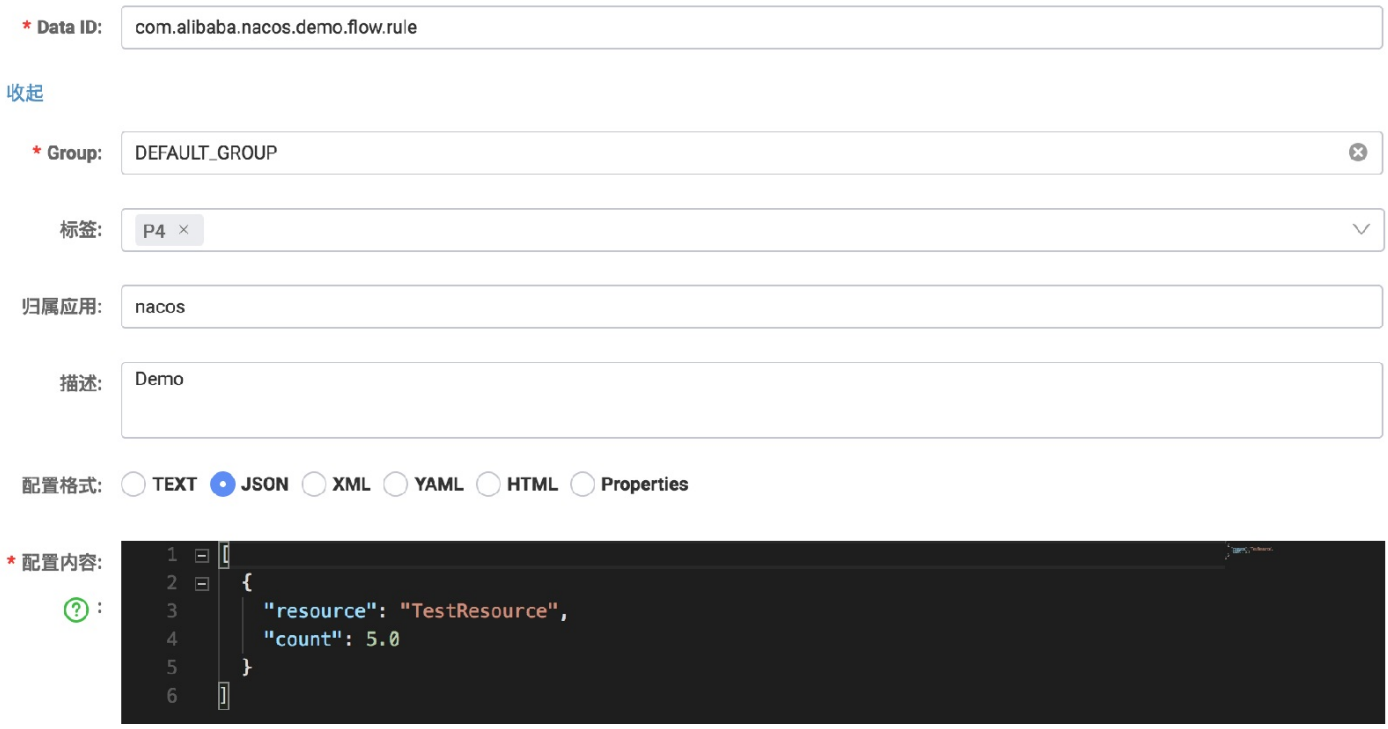


多配置格式编辑器

Nacos支持 YAML、Properties、TEXT、JSON、XML、HTML 等常见配置格式在线编辑、语法高亮、格式校验，帮助用户高效编辑的同时大幅降低格式错误带来的风险。

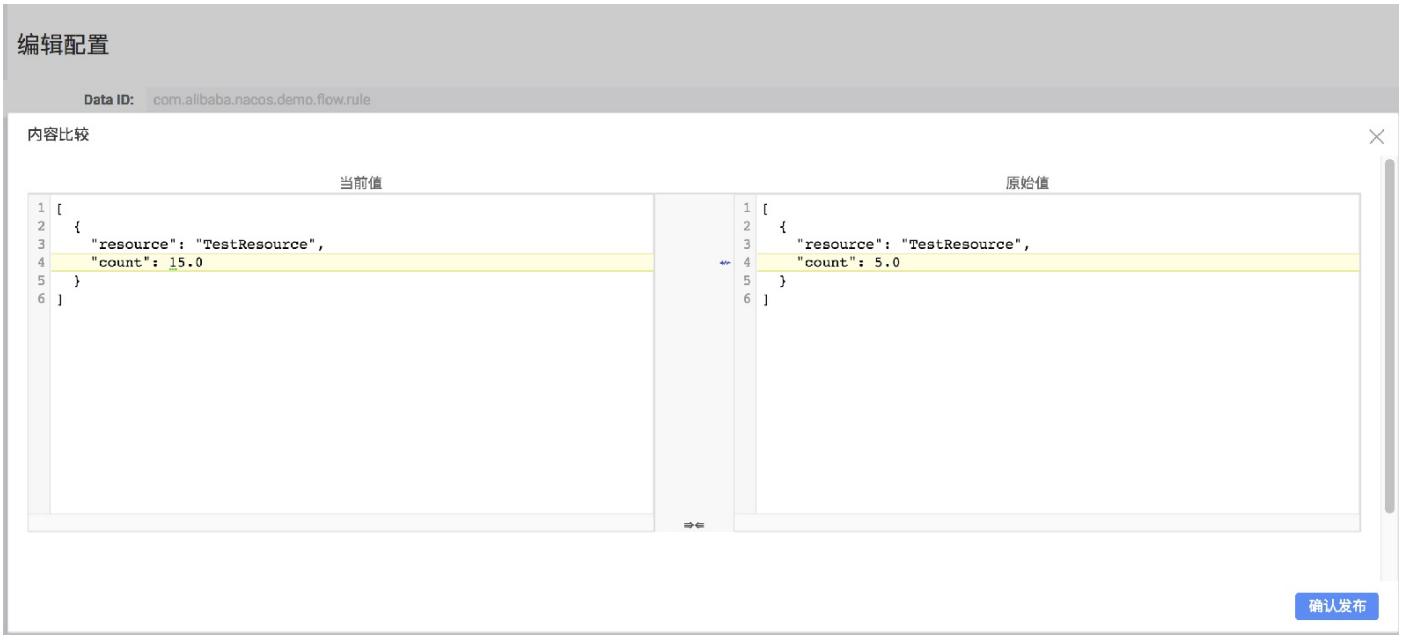
Nacos支持配置标签的能力，帮助用户更好、更灵活的做到基于标签的配置分类及管理。同时支持用户对配置及其变更进行描述，方面多人或者跨团队协作管理配置。

新建配置



编辑DIFF

Nacos支持编辑DIFF能力，帮助用户校验修改内容，降低改错带来的风险



示例代码

Nacos提供示例代码能力，能够让新手快速使用客户端编程消费该配置，大幅降低新手使用门槛。

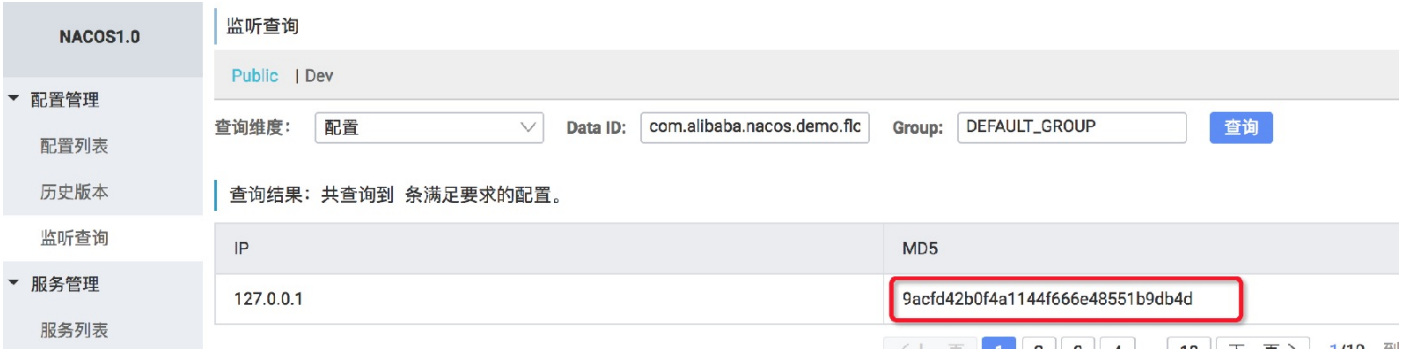
示例代码

Java | Spring Boot | Spring Cloud | Node.js | C++ | Shell | Python

```
25 public class ConfigExample {
26
27     public static void main(String[] args) throws NacosException, InterruptedException {
28         String serverAddr = "localhost";
29         String dataId = "com.alibaba.nacos.demo.flow.rule";
30         String group = "DEFAULT_GROUP";
31         Properties properties = new Properties();
32         properties.put(PropertyKeyConst.SERVER_ADDR, serverAddr);
33         ConfigService configService = NacosFactory.createConfigService(properties);
34         String content = configService.getConfig(dataId, group, 5000);
35         System.out.println(content);
36         configService.addListener(dataId, group, new Listener() {
37             @Override
38             public void receiveConfigInfo(String configInfo) {
39                 System.out.println("recieve:" + configInfo);
40             }
41
```

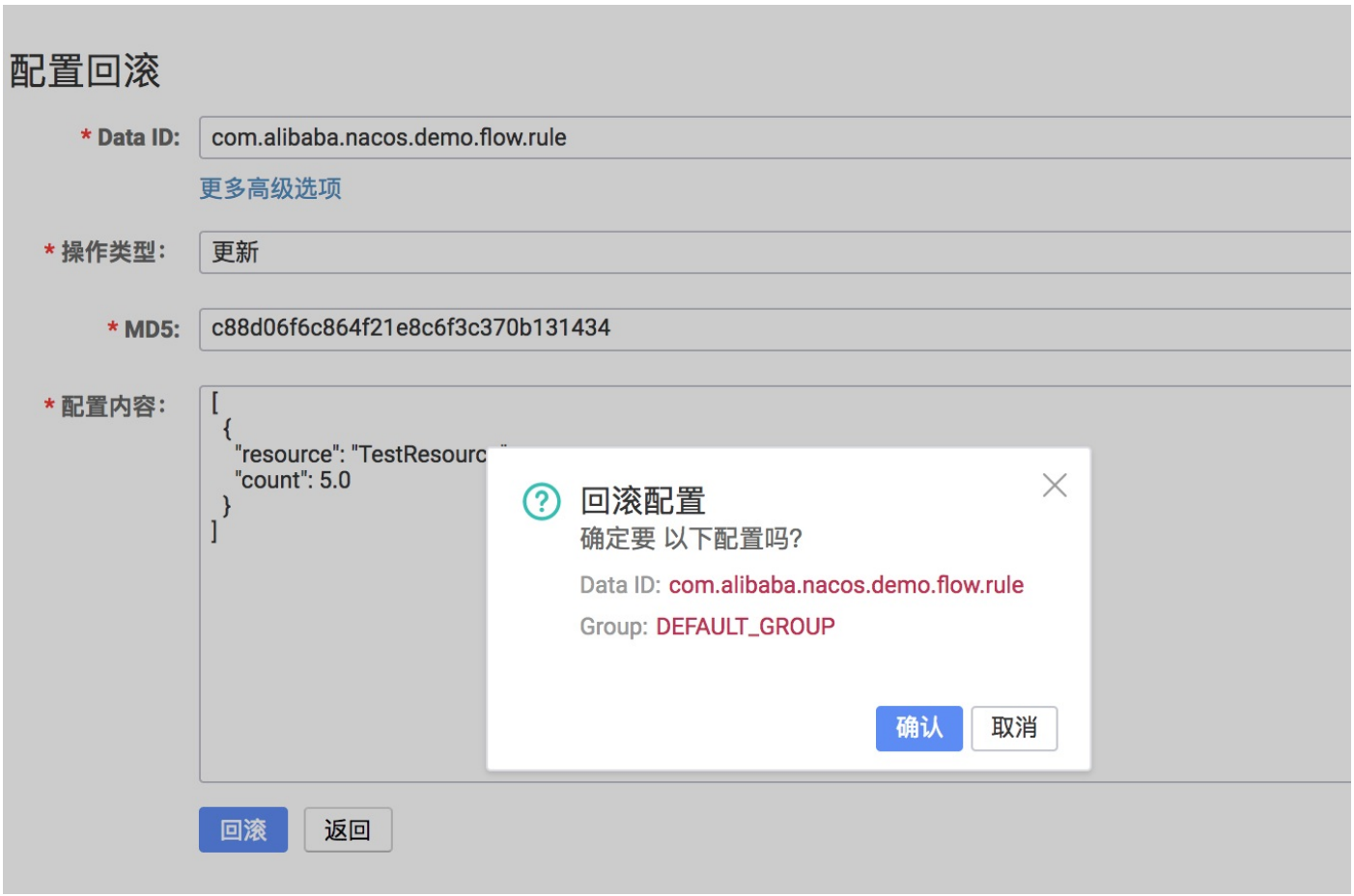
监听者查询

Nacos提供配置订阅者即监听者查询能力，同时提供客户端当前配置的MD5校验值，以便帮助用户更好的检查配置变更是否推送到 Client 端。



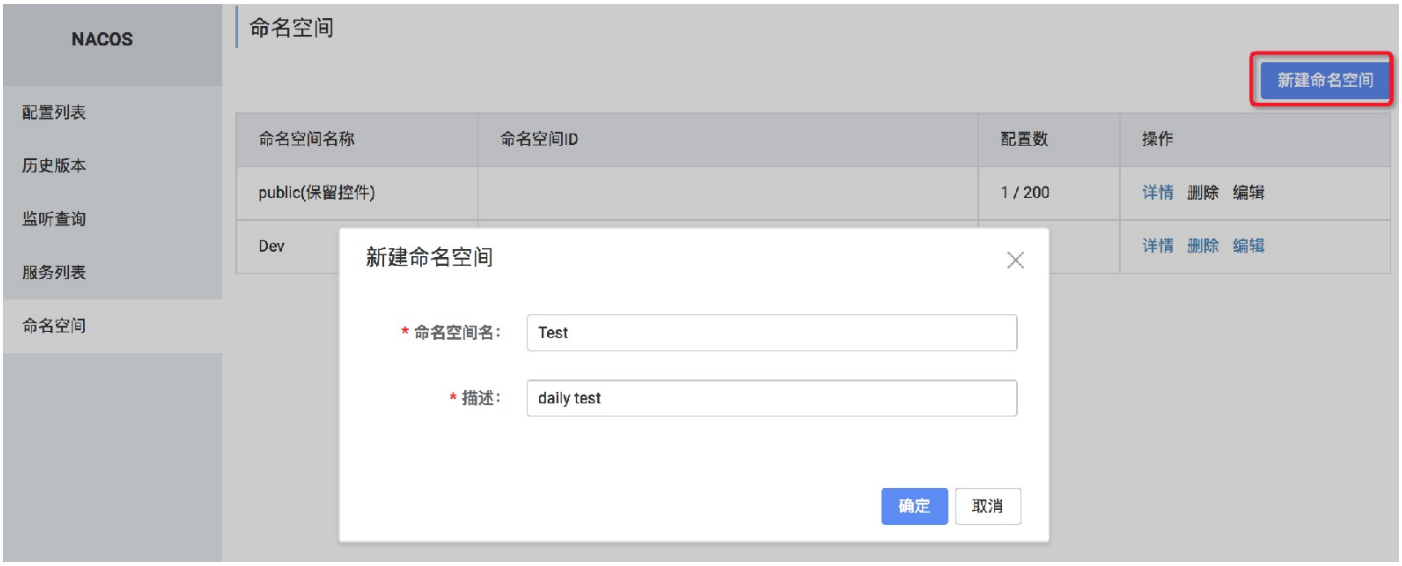
配置的版本及一键回滚

Nacos通过提供配置版本管理及其一键回滚能力，帮助用户改错配置的时候能够快速恢复，降低微服务系统在配置管理上的一定会遇到的可用性风险。



命名空间管理

Nacos 基于Namespace 帮助用户逻辑隔离多个命名空间，这可以帮助用户更好的管理测试、预发、生产等多环境服务和配置，让每个环境的同一个配置（如数据库数据源）可以定义不同的值。



社区参与的前端共建

在Nacos前端风格、布局的讨论中，社区踊跃投票，最终选择了这套经典黑白蓝风格的皮肤，并且通过我们UED程瑶同学的设计，布局，让交互变得十分自然流畅。

在控制台的开发之前我们通过社区招募到了很多前端同学一起参与了前端代码的开发，在此尤其感谢李晨、王庆、王彦民同学在Nacos前端开发过程中的大力支持！

坚持社区化发展，欢迎加入并贡献社区

> *DISS is cheap, show me your hand*比吐槽更重要的是搭把手，参与社区一起发展*Nacos*

要加入Nacos 微信社区讨论 Nacos 产品的演进，你可以通过扫“超哥”的微信二维码，让“超哥” 帮你拉入“Nacos社区交流群”

更多与 Nacos 相关的开源项目信息

- Nacos
- Nacos Spring Project
- Nacos Spring Boot
- Spring Cloud Alibaba
- Dubbo
- Sentinel
- Spring Cloud
- Nepxion Discovery

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

- 贡献源码
- Nacos有奖活动介绍
- pull request模板
- 如何提交问题报告
- Nacos规划
- Nacos支持SpringCloud生态
- nacos支持dubbo生态
- Nacos支持k8s
- nacos支持istio

如何贡献

我们非常欢迎您的贡献和加入，无论是微不足道的清理或大的新功能。我们希望为每个编程语言提供高质量、有良好文档的代码。

这也不是代码是唯一有贡献项目的方式。我们非常重视文档、与其他项目的集成，并欣然接受这些方面的改进。

联系gitter https://gitter.im/alibaba/nacos

邮件列表

邮件列表是讨论与Dubbo相关的几乎所有内容的推荐方式。有关如何订阅的详细文档，请参阅此指南。

- subscribe@googlegroups.comg"">dev-nacos@googlegroups.com: 开发邮件列表，如果您在使用或开发Nacos时遇到任何问题，可以在此处提问。
- subscribe@googlegroups.com"">commits-nacos@googlegroups.com: 所有提交都将发送到此邮件列表。如果您对Nacos的开发感兴趣，可以订阅它。
- subscribe@googlegroups.com"">users-nacos@googlegroups.com: 所有Github issues更新和[pull request](https://github.com/alibaba/nacos/pulls)更新将发送到此邮件列表。
- 有任何问题可以联系nacos_dev@linux.alibaba.com。

贡献流程

这是贡献者的工作流程的大致轮廓：

fork当前存储github库。创建一个分支，作为贡献的基础。 这通常是master分支。做出一些变更提交。确保提交消息的格式正确（见下文）。推送变更到你的fork仓库中。按照拉取请求模板中的清单进行操作在发送拉取请求之前，请将您的fork仓库与远程存储库同步。 这将使您的拉取请求变得简单明了。 见下面的指南：

```
1. git remote add upstream git@github.com:alibaba/nacos.git
2. git fetch upstream
3. git rebase upstream/master
4. git checkout -b your_awesome_patch
5. ... add some work
6. git push origin your_awesome_patch
```

提交pull request 到 alibaba/nacos，等待回复。如果回复的慢，请无情的催促。

贡献代码

请提交代码时候，检查以下内容：

如果变化不大，请编写一些覆盖新功能的单元测试。

如果你正在引入一个全新的特性或API，那么首先启动wiki并在基本设计上达成共识，再开始投入。

我们的工作是及时跟进补丁。如果我们没有及时跟进，请无情的催促我们。

成为贡献者

We are always interested in adding new contributors. What we look for are series of contributions, good taste and ongoing interest in the project. If you are interested in becoming a committer, please let one of the existing committers know and they can help you walk through the process.

Nowadays,we have several important contribution points:

我们会积极纳入新的贡献者。我们更关注的是一系列的持续贡献，良好的品味和对项目维护的持续兴趣。如果你想成为一个提交者（Committer），请让一个现有的提交者(Committer)知道，他们会帮助你通过贡献加入我们。

现在，我们有几个重要的贡献点:

Wiki & JavaDoc

Nacos Console

Nacos SDK(C++.Net\Php\Python\Go\Node.js)

前提

如果你想贡献以上的项，请你必须遵守我们的一些先决条件:

可读性，一个API必须具有JavaDoc，一些非常重要的方法也必须有JavaDoc。

可测性，关于测试过程的单元测试覆盖率（80%）

可维护性，可满足我们的PMD spec，以及至少3个月的更新频率

可部署性，我们可以鼓励您部署到maven repository

> 原文: *https://nacos.io/zh-cn/docs/contributing.html*

I. 活动任务列表

- 阅读官网文案，找官网bug，给中、英文官网提建设性建议
- 阅读中英文档，找文档bug，给中、英文档改善提建议（尤其是关注英文翻译不好的地方，因为英文都是我们程序员自己撸的）
- 尝试下代码 ->编译打包 -> 启动Nacos server -> 停止Nacos server流程，提出改进意见
- 尝试配置以及启动多节点 Nacos 集群模式任务，提改进意见
- 尝试使用Nacos Java SDK，给Java SDK提改进建议
- 尝试使用Nacos Open API，给OpenAPI提改进建议
- 尝试根据《如何贡献Nacos文档》试一下 贡献流程，给贡献者流程提建议
- 给Nacos提需求、发展计划、想法和要求等

II. 活动参与方式

- 扫描 “超哥” 微信2微码，让超哥帮助加入 “Nacos社区交流群”

- 选择 (I) 中的一个或者多个体验任务
- 发现问题或者BUG之后，按照(III)中的《问题Report方式》，发一个相应的 github issue，并指派给 @github账号xuechaos

III. 问题Report方式

- 根据issues模板，在github的nacos仓库，中提交问题。
- 代码仓库问题，提交pull request修复，通过review后，合并进入主干，就成为了Contributor。
-
- https://github.com/alibaba/nacos
- https://github.com/nacos-group/nacos-group.github.io

IV. 任务奖励

- 我们正在为参与并作出突出贡献的小伙伴定制一些带有Nacos Logo的小礼品，会考虑快递给过程中有突出贡献的小伙伴。
- 礼物虽轻，但希望能表达对你们的帮助的感激之情的万一。
- 因资金有限，我们目前只能提供40份礼品，所以我们这次限制报名参与人数为40人，先到先得。
- 如果你愿意，希望收到礼物的小伙伴能发一张与礼物的合影（可以隐藏面部）给我们，作为后续我们社区活动的照片素材。

V. 其它说明

- 我们不确定每个建议最后都会被采用，但是我们尽量会说明我们基于何种考虑，您的建议我们最后没有采用。
- 尽量通过邮件列表或者report issue的方式，而不是在微信群里report问题，以便将我们的沟通过程文档化和更容易沉淀。

> 原文: *https://nacos.io/zh-cn/docs/activity.html*

Please do not create a Pull Request without creating an issue first.

What is the purpose of the change

XXXXX

Brief changelog

XX

Verifying this change

XXXX

Follow this checklist to help us incorporate your contribution quickly and easily:

- [ ] Make sure there is a Github issue filed for the change (usually before you start working on it). Trivial changes like typos do not require a Github issue. Your pull request should address just this issue, without pulling in other changes - one PR resolves one issue.
- [ ] Format the pull request title like `[ISSUE #123] Fix UnknownException when host config not exist`. Each commit in the pull request should have a meaningful subject line and body.
- [ ] Write a pull request description that is detailed enough to understand what the pull request does, how, and why.
- [ ] Write necessary unit-test to verify your logic correction, more mock a little better when cross module dependency exist. If the new feature or significant change is committed, please remember to add integration-test in test module.
- [ ] Run `mvn -B clean apache-rat:check findbugs:findbugs` to make sure basic checks pass. Run `mvn clean install -DskipITs` to make sure unit-test pass. Run `mvn clean test-compile failsafe:integration-test` to make sure integration-test pass.
- [ ] If this contribution is large, please file an Apache Individual Contributor License Agreement.

> 原文: *https://nacos.io/zh-cn/docs/pull-request.html*

如何提交问题报告

如果Nacos项目的任何部分存在问题或文档问题，请通过opening an issue告诉我们。我们非常认真地对待错误和错误，在产品面前没有不重要的问题。不过在创建错误报告之前，请检查是否存在报告相同问题的issues。

为了使错误报告准确且易于理解，请尝试创建以下错误报告：

- 具体到细节。包括尽可能多的细节：哪个版本，什么环境，什么配置等。如果错误与运行Nacos服务器有关，请附加Nacos日志（具有Nacos配置的起始日志尤为重要）。
- 可复现。包括重现问题的步骤。我们理解某些问题可能难以重现，请包括可能导致问题的步骤。如果可能，请将受影响的Nacos数据目录和堆栈strace附加到错误报告中。
- 不重复。不要复制现有的错误报告。

在创建错误报告之前，最好阅读下Elika Etemad关于提交好错误报告的文章，相信 会给你启发。

我们可能会要求您提供更多信息以查找错误。将关闭重复的错误报告。

nacos-issuefiling-good-bugs

> 原文：*https://nacos.io/zh-cn/docs/how-to-reporting-bugs.html*

我们计划从 Nacos 0.8.0 开始将其做到生产可用状态。在这个版本之前，我们建议您仅将其用于开发和测试环境。我们目前的计划是努力在未来6~8个月内将Nacos演进到生产可用的版本。当然计划可能因为各种因素影响而做调整，包括根据社区的声音进行优先级调整等，但整体应该不会超过1年的时间。

以下是未来1年我们的主要路线图与计划。

Nacos 1.0

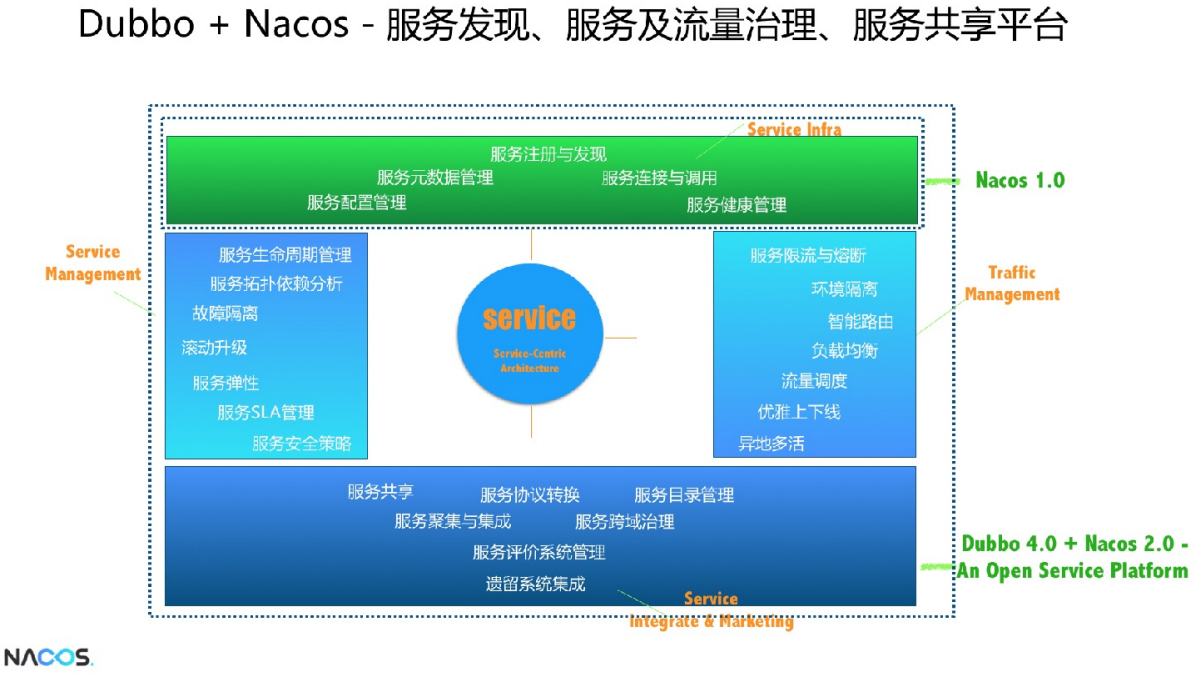
主要目标有两个：

- 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。
  以下是大致的计划：
- 0.1 Basic Nacos server and simple OpenAPI and Java SDK;
- 0.2 - 0.3 Seamless support for Kubernetes, Service Mesh and Spring Cloud service discovery and configuration management;
- 0.4 - 0.5 Build an easy-to-use Web UI/User Console;
- 0.6 - 0.7 High availability, ease of use, monitoring and alert etc.;
- 0.8 Production ready;
- 0.9 Large scale performance tuning and benchmark;
- 1.0 GA for large scale production.

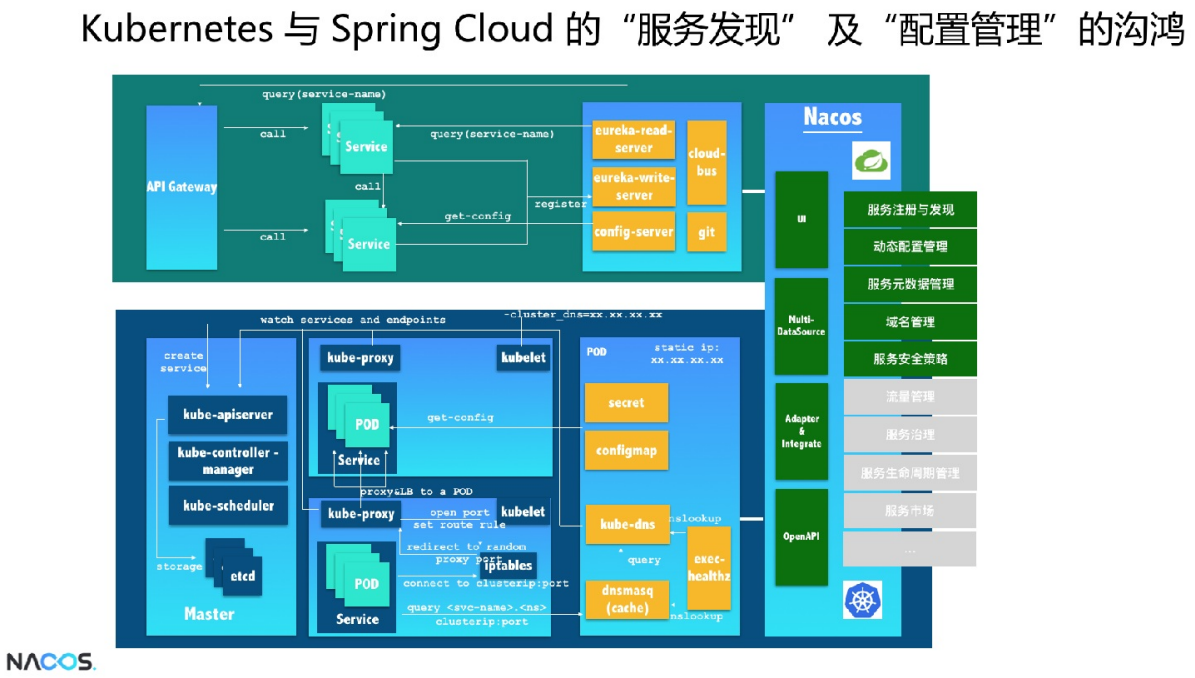
Nacos 2.0

主要关注在统一服务管理、服务共享及服务治理体系的开放的服务平台的建设上，主要包括两个方面：

- Dubbo 4.0 + Nacos 2.0 开放的服务平台



- Kubernetes + Spring Cloud 统一服务管理



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

IN PLAN with Nacos 0.2.0

> 原文: *https://nacos.io/zh-cn/docs/use-nacos-with-springcloud.html*

IN PLAN with Nacos 0.3.0

> 原文: *https://nacos.io/zh-cn/docs/use-nacos-with-dubbo.html*

IN PLAN with Nacos 0.4.0

> 原文: *https://nacos.io/zh-cn/docs/use-nacos-with-kubernetes.html*

IN PLAN with Nacos 0.6.0

> 原文: *https://nacos.io/zh-cn/docs/use-nacos-with-istio.html*

Contact

Nacos Gitter-https://gitter.im/alibaba/nacos

Nacos weibo-https://weibo.com/u/6574374908

Nacos segmentfault-https://segmentfault.com/t/nacos

Mailing list

Mailing list is recommended for discussing almost anything related to Nacos. Please refer to this?guide?for detailed documentation on how to subscribe to our mailing lists.

- dev-nacos@googlegroups.com: The develop mailing list. You can ask questions here if you encounter any problem when using or developing Nacos.
- commits-nacos@googlegroups.com: All commits will be sent to this mailing list. You can subscribe to it if you are interested in Nacos' development.
- users-nacos@googlegroups.com: All Github?issue?updates and?pull request?updates will be sent to this mailing list.
- nacos_dev@linux.alibaba.com.

> 原文: *https://nacos.io/zh-cn/docs/community.html*