# 官网

https://nacos.io/

# 官网文档

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

# 下载地址

https://github.com/alibaba/nacos/releases

# 快速开始

https://nacos.io/zh-cn/docs/v2/quickstart/quick-start.html

# 基本概念

Nacos是阿里巴巴开源的一款支持服务注册与发现，配置管理以及微服务管理的组件。用来取代以前常用的注册中心（zookeeper , eureka等等），以及配置中心（spring cloud config等等）。Nacos是集成了注册中心和配置中心的功能，做到了二合一。

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。

## 地域

## 可用区

## 接入点

## 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

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

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

配置集

配置集ID

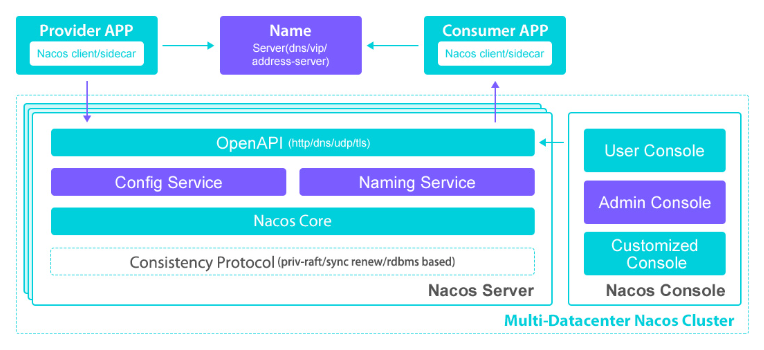
配置分组

配置快照

# 发布时间

2018年

# 架构



# 特性

## ****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（service-level agreement 服务级别协议） 以及最首要的 metrics 统计数据。

# 优点

1、开箱即用，适用于[dubbo](https://so.csdn.net/so/search?q=dubbo&spm=1001.2101.3001.7020)，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。

# 默认端口号

8848

# 环境搭建（单机方式 windows环境）

## 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](https://maven.apache.org/index.html)环境，请确保是在以下版本环境中安装使用:

（1）64 bit OS，支持 Linux/Unix/Mac/Windows，推荐选用 Linux/Unix/Mac。

（2）64 bit JDK 1.8+；[下载](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) & [配置](https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/)。

（3）Maven 3.2.x+；[下载](https://maven.apache.org/download.cgi) & [配置](https://maven.apache.org/settings.html)。

## 2、下载安装包

下载地址: https://github.com/alibaba/nacos/releases

下载zip格式的安装包，然后进行解压缩操作。

## 3、启动nacos

（1）切换目录

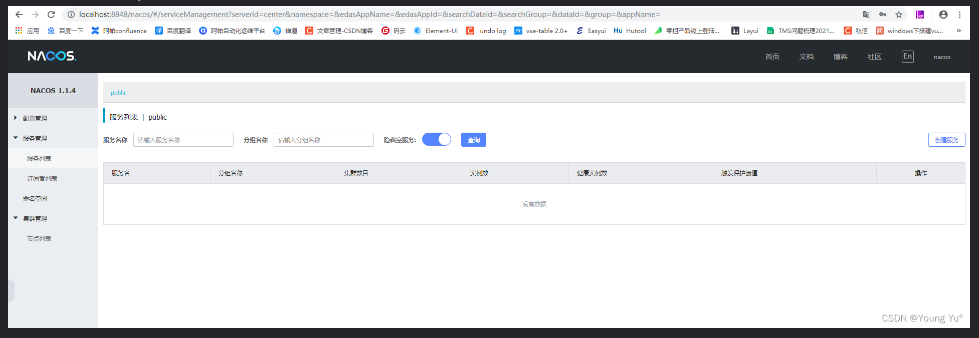
> cd nacos/bin

（2）命令启动 Linux 单实例非集群模式启动命令

> startup.cmd -m standalone

## 4、访问nacos

打开浏览器输入http://localhost:8848/nacos，即可访问服务， 默认密码是nacos/nacos。



# linux环境安装Nacos

https://nacos.io/zh-cn/docs/v2/quickstart/quick-start.html

## 启动命令

sh startup.sh -m standalone

standalone代表着单机模式运行，非集群模式

## 关闭命令

sh shutdown.sh

# 控制台手册

## 基本概念

Nacos 控制台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，将提供包括下列基本功能：

### 1、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

### 2、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

### 3、命名空间

### 4、登录管理

## 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。

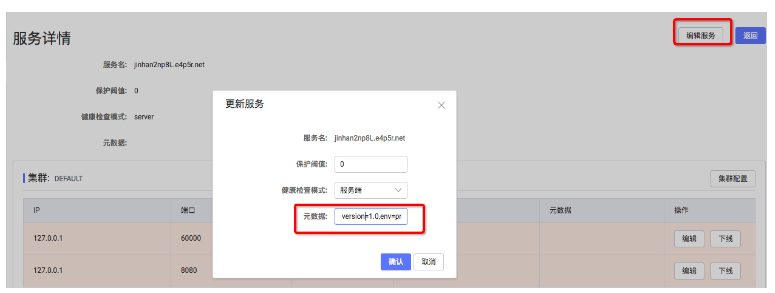
#### 服务流量权重支持及流量保护

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。

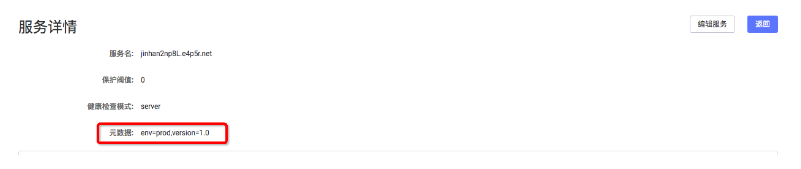


#### 服务元数据管理

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。

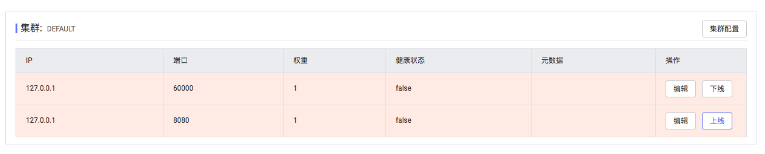


点击确认，就可以在服务详情页面，看到服务的元数据已经更新了。



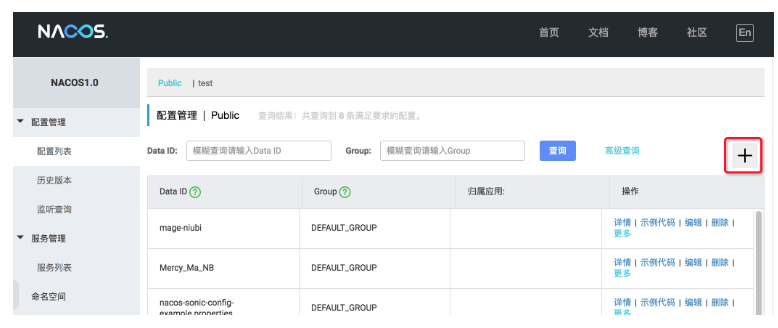
#### 服务优雅上下线

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

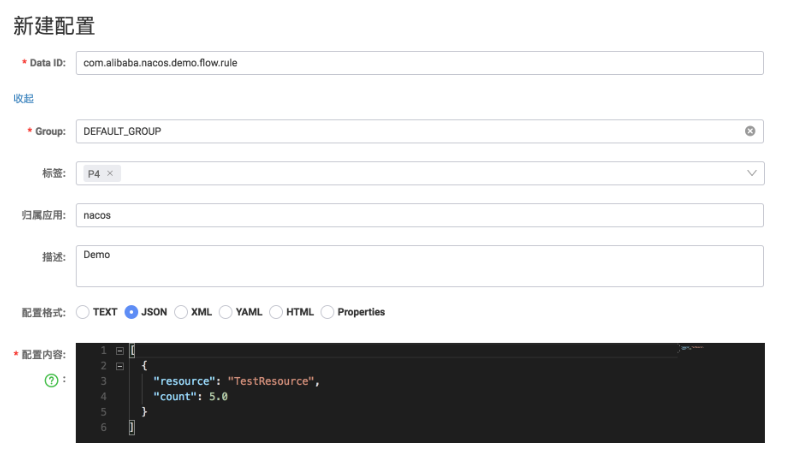
Nacos支持基于Namespace和Group的配置分组管理，以便用户更灵活的根据自己的需要按照环境或者应用、模块等分组管理微服务以及Spring的大量配置，在配置管理中主要提供了配置历史版本、回滚、订阅者查询等核心管理能力。



#### 多配置格式编辑器

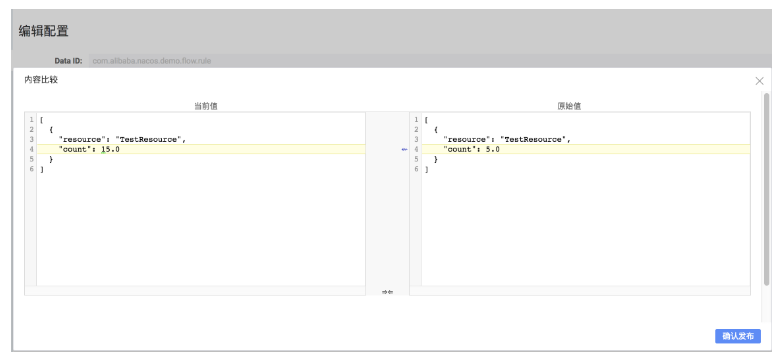
Nacos支持 YAML、Properties、TEXT、JSON、XML、HTML 等常见配置格式在线编辑、语法高亮、格式校验，帮助用户高效编辑的同时大幅降低格式错误带来的风险。

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。



#### 编辑DIFF

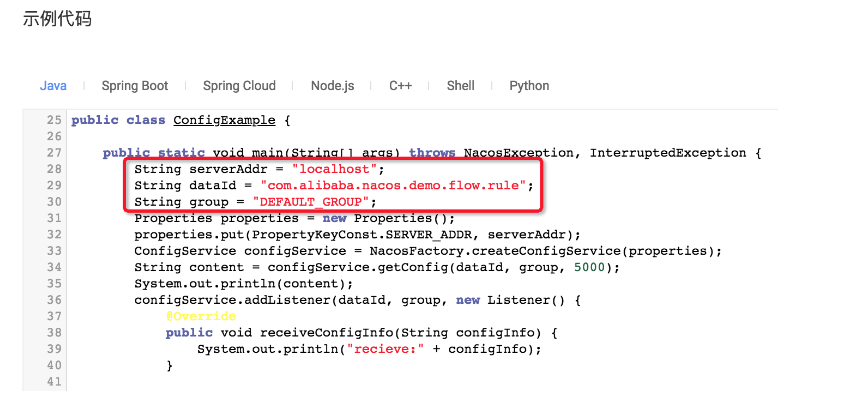
Nacos支持编辑DIFF能力，帮助用户校验修改内容，降低改错带来的风险。



#### 示例代码

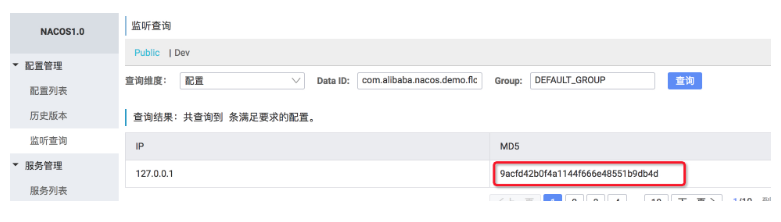
Nacos提供示例代码能力，能够让新手快速使用客户端编程消费该配置，大幅降低新手使用门槛。





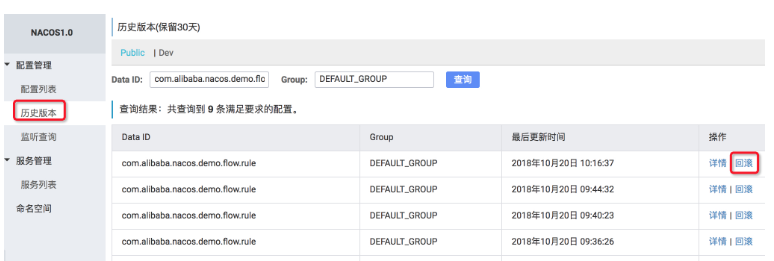
#### 监听查询

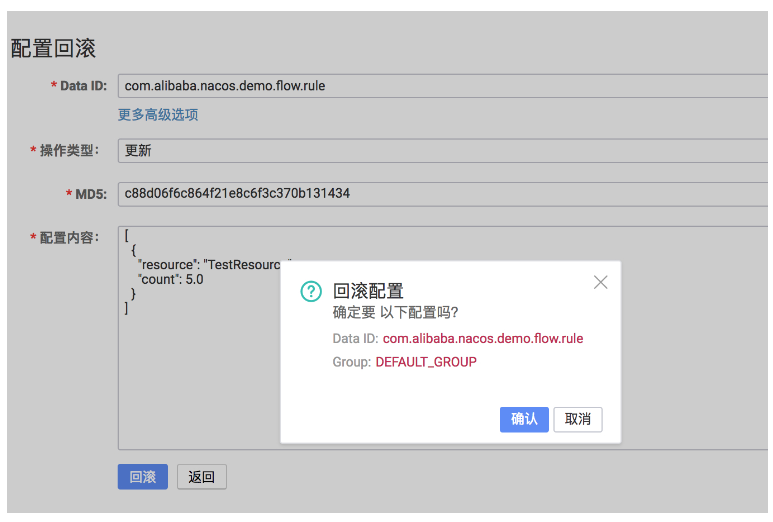
Nacos提供配置订阅者即监听者查询能力，同时提供客户端当前配置的MD5校验值，以便帮助用户更好的检查配置变更是否推送到 Client 端。



#### 配置的版本及一键回滚

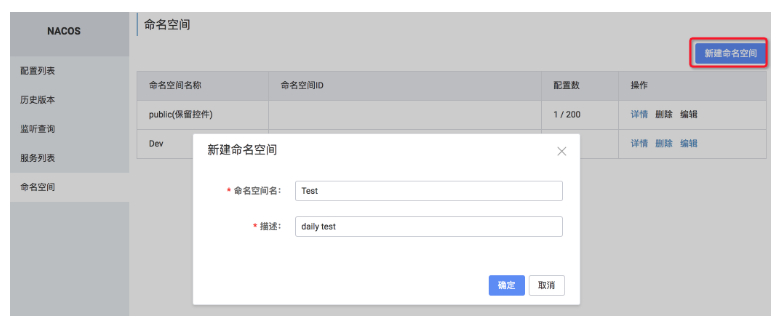
Nacos通过提供配置版本管理及其一键回滚能力，帮助用户改错配置的时候能够快速恢复，降低微服务系统在配置管理上的一定会遇到的可用性风险。





### 命名空间管理

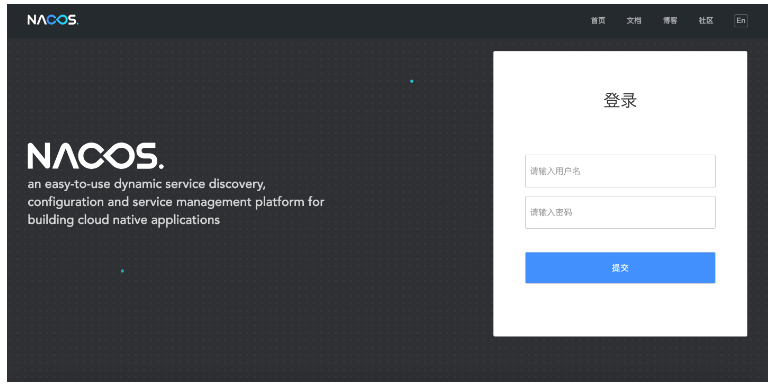
Nacos 基于Namespace 帮助用户逻辑隔离多个命名空间，这可以帮助用户更好的管理测试、预发、生产等多环境服务和配置，让每个环境的同一个配置（如数据库数据源）可以定义不同的值。





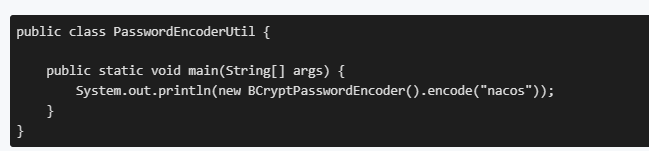
### 登录管理

Nacos0.8 版本支持简单登录功能，默认用户名/密码为： nacos/nacos。

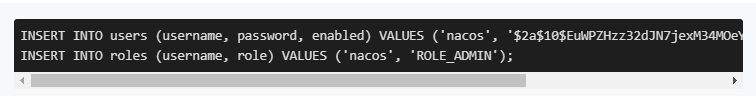


#### 修改默认用户名/密码方法

1、生成加密密码， 在com.alibaba.nacos.console.utils.PasswordEncoderUtil.main函数中，将 nacos 改成你要改成的密码，运行即可得到加密有算法。注意盐值是随机的，所以生成密码每次可能不一样，请不要担心。



2、创建用户名或者密码的时候，用指定用户名密码即可。



### 关闭登录功能

由于部分公司自己开发控制台，不希望被nacos的安全filter拦截。因此nacos支持定制关闭登录功能找到配置文件 ${nacoshome}/conf/application.properties， 替换以下内容即可。



### 会话时间

默认会话保持时间为30分钟。30分钟后需要重新登录认证。 暂时不支持修改该默认时间。

# Nacos默认使用的数据库

## 说明

Nacos可以不用单独配置数据库信息也可以启动。

## 默认启动模式

Nacos默认使用内嵌数据库（Derby）启动，这种模式下：

1、不需要配置外部数据库

2、数据存储在本地文件系统中

3、适合单机部署和开发测试环境

## 内嵌数据库模式

1、使用Derby数据库

2、数据文件位置：${nacos.home}/data/derby-data/

3、配置信息存储在本地文件中

4、重启后数据不会丢失

## 使用场景

内嵌数据库适用于：

1、开发测试环境

2、单机部署

3、快速体验Nacos功能

# Naocs配置数据库

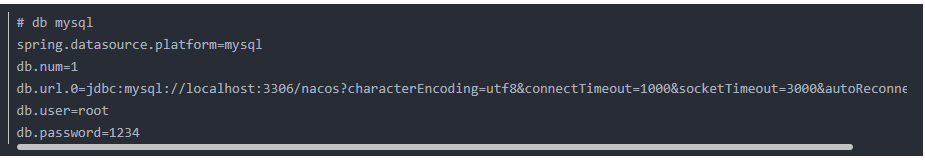
nacos支持配置多个数据库，通过 db.num 和 db.url.index的配置来控制。nacos配置mysql数据库只需如下三点即可完成：

1、安装数据库，且确保版本5.6.5+

2、使用 conf/nacos-mysql.sql 文件初始化数据库

3、修改conf/application.properties文件，增加mysql支持。

下面给出只配置一个数据库的mysql支持配置：



# 服务端开启鉴权

## 参考网址

https://nacos.io/zh-cn/docs/v2/guide/user/auth.html

## 基本概念

1、2.2.2版本之前的Nacos默认控制台，无论服务端是否开启鉴权，都会存在一个登录页。

2、Nacos自**2.2.2**版本开始，在未开启鉴权时，默认控制台将不需要登录即可访问，同时在控制台中给予提示，提醒用户当前集群未开启鉴权。

3、在用户开启鉴权后，控制台才需要进行登录访问。 同时针对不同的鉴权插件，提供新的接口方法，用于提示控制台是否开启登录页；同时在2.2.3版本后，Nacos可支持关闭开源控制台，并引导到用户自定义的Nacos控制台。

## 参数

### nacos.core.auth.enabled

是否开启鉴权功能，默认值：false

### nacos.core.auth.system.type

鉴权类型，默认nacos

### nacos.core.auth.plugin.nacos.token.secret.key

# nacos修改密码

https://blog.csdn.net/HaHa\_Sir/article/details/122284868

# nacos的配置信息

## spring.main.allow-bean-definition-overriding

这个配置控制是否允许**Bean定义覆盖**，即当Spring容器中存在同名Bean时的处理策略。

## spring.cloud.nacos.discovery.server-addr

这个配置用于指定**Nacos服务器的地址**，告诉Spring Cloud应用去哪里注册服务和发现其他服务。

这个配置是Spring Cloud Nacos集成的基础，正确配置后应用就能实现服务注册与发现功能。

## spring.cloud.nacos.config.server-addr

这个配置用于指定**Nacos配置中心服务器的地址**，告诉Spring Cloud应用从哪里获取配置信息。

这个配置是实现配置中心功能的基础，正确配置后应用就能从Nacos动态获取和更新配置。

## spring.cloud.nacos.config.namespace

用于指定 Nacos 配置中心的命名空间（Namespace），通过合理使用命名空间，可以确保不同环境、项目或租户的配置互不干扰，提升系统的安全性和可维护性。配置时需注意：

1、使用 Nacos 生成的 **Namespace ID**（非名称）。

2、结合 Spring Profiles 实现环境动态切换。

3、避免直接使用默认的 public 命名空间。

## spring.cloud.nacos.discovery.enabled

spring.cloud.nacos.discovery.enabled 是 Spring Cloud Nacos（一个基于 Nacos 实现的服务注册与发现组件）中的一个配置属性，用于控制是否启用 Nacos 的服务发现功能。

当 spring.cloud.nacos.discovery.enabled 的值为 true 时，表示启用 Nacos 的服务发现功能。

spring.cloud.nacos.discovery.enabled 的值为 false 时，表示禁用 Nacos 的服务发现功能。

作用：Nacos 的服务发现功能允许服务提供者将自己注册到注册中心，并允许服务消费者从注册中心获取可用的服务实例信息。通过配置 spring.cloud.nacos.discovery.enabled 属性，可以灵活地控制是否使用 Nacos 的服务发现功能。

## spring.cloud.nacos.discovery.register-enabled

用于控制是否启用 Nacos 的服务注册功能

## spring.cloud.nacos.config.enabled

用于控制是否启用 Nacos 的配置中心功能。

当 spring.cloud.nacos.config.enabled 的值为 true 时，表示启用 Nacos 的配置中心功能

当 spring.cloud.nacos.config.enabled 的值为 false 时，表示禁用 Nacos 的配置中心功能。

作用：

Nacos 提供了一个集中式的配置中心，可以管理和动态刷新应用程序的配置信息。通过配置 spring.cloud.nacos.config.enabled 属性，可以决定是否使用 Nacos 的配置中心功能。

## spring.cloud.nacos.config.extension-configs

用于指定额外的扩展配置文件。它允许你在应用程序中加载多个不同路径的配置文件，并将它们合并到应用程序的配置中。spring.cloud.nacos.config.extension-configs 允许你指定一个包含多个配置文件路径的列表，这些配置文件会被加载并应用于应用程序的配置中。每个路径可以是本地文件系统上的文件路径（使用 file: 前缀）或类路径下的文件路径（使用 classpath: 前缀）。

## spring.cloud.nacos.discovery.heart-beat-timeout

这个配置用于设置**心跳超时时间**，即服务实例向Nacos发送心跳的超时阈值。

这个配置直接影响服务发现的实时性和准确性，需要根据实际环境合理调整。

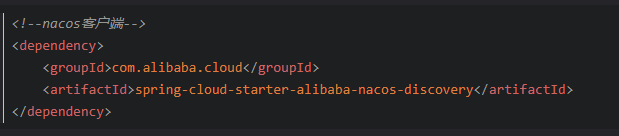
## ribbon.nacos.enabled

ribbon.nacos.enabled=true 是 Spring Cloud Ribbon 中用于启用或禁用基于 Nacos 的负载均衡的配置项。

当将 ribbon.nacos.enabled 设置为 true 时，表示启用了基于 Nacos 的负载均衡功能。这意味着 Ribbon 将通过 Nacos 服务器来获取注册在 Nacos 上的服务实例列表，并根据负载均衡策略选择目标服务实例来分发请求。

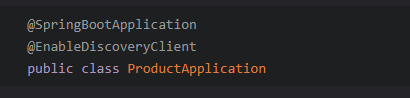
# 客户端nacos使用

## 1、在pom.xml中添加nacos的依赖



spring-cloud-starter-alibaba-nacos-discovery

## 2、在主类上添加@EnableDiscoveryClient注解



说明：这个可以省略

## 3、在application.yml中添加nacos服务的地址

文本

中度可信度描述已自动生成

## 4、启动客户端服务，观察nacos服务控制面板



## 5、元数据

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。



可获取服务的IP、端口号等信息。

# 服务调用



通过restTemplate类去调用。

# Springcloud整合Nacos（重点）

## 参考文档

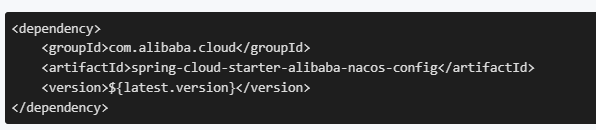
https://github.com/alibaba/spring-cloud-alibaba/wiki/Nacos-config

## 1、前提条件

您需要先下载 Nacos 并启动 Nacos server。

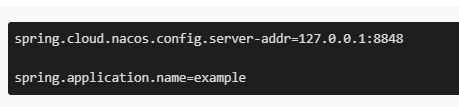
## 2、启动配置管理

### 1、添加依赖



spring-cloud-starter-alibaba-nacos-config

### 2、在 bootstrap.properties文件中配置 Nacos server 的地址和应用名



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：



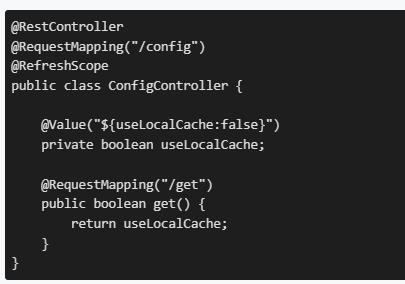
prefix 默认为 spring.application.name 的值，也可以通过配置项 spring.cloud.nacos.config.prefix来配置。

spring.profiles.active 即为当前环境对应的 profile，详情可以参考 Spring Boot文档。

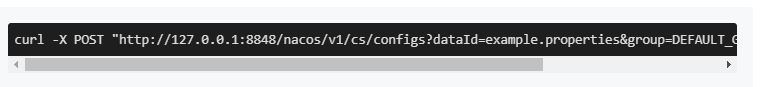
**注意：当**spring.profiles.active**为空时，对应的连接符**-**也将不存在，dataId 的拼接格式变成**${prefix}.${file-extension}

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。

### 3、通过 Spring Cloud 原生注解 @RefreshScope 实现配置自动更新

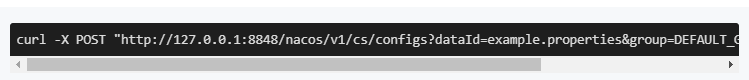


4、首先通过调用 [Nacos Open API](https://nacos.io/zh-cn/docs/open-api.html) 向 Nacos Server 发布配置：dataId 为example.properties，内容为useLocalCache=true



5、运行 NacosConfigApplication，调用 curl http://localhost:8080/config/get，返回内容是 true。

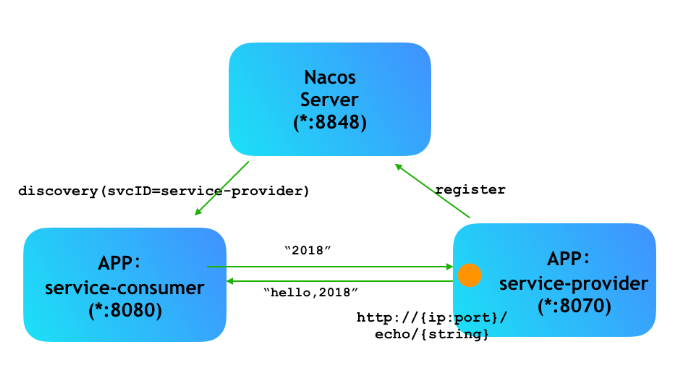
6、再次调用 [Nacos Open API](https://nacos.io/zh-cn/docs/open-api.html) 向 Nacos server 发布配置：dataId 为example.properties，内容为useLocalCache=false



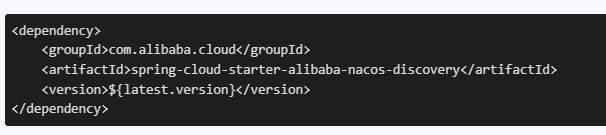
7、再次访问 http://localhost:8080/config/get，此时返回内容为false，说明程序中的useLocalCache值已经被动态更新了。

## 3、启动服务发现

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:



### 1、添加依赖

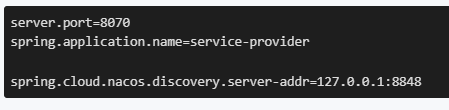


spring-cloud-starter-alibaba-nacos-discovery

### 2、配置服务提供者

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。

在 application.properties 中配置 Nacos server 的地址：



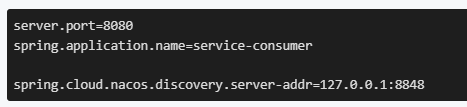
通过 Spring Cloud 原生注解 @EnableDiscoveryClient 开启服务注册发现功能。



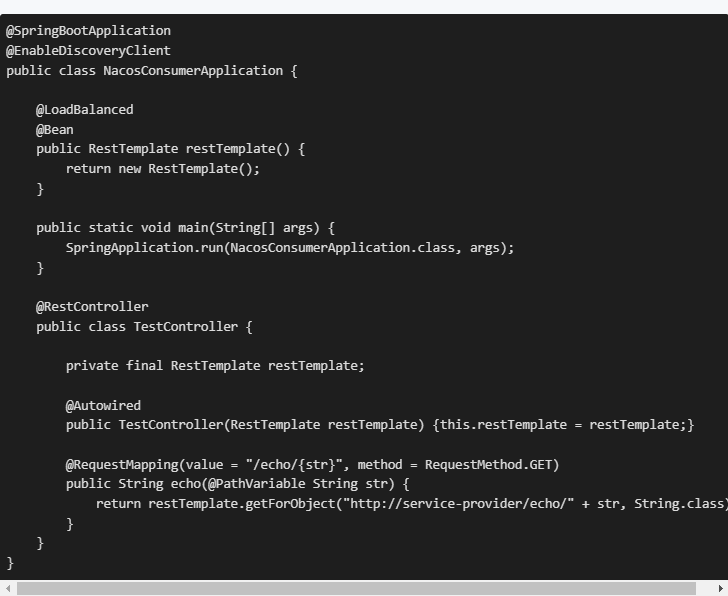
### 3、配置服务消费者

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。

在 application.properties 中配置 Nacos server 的地址



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](https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-ribbon.html) 的集成。



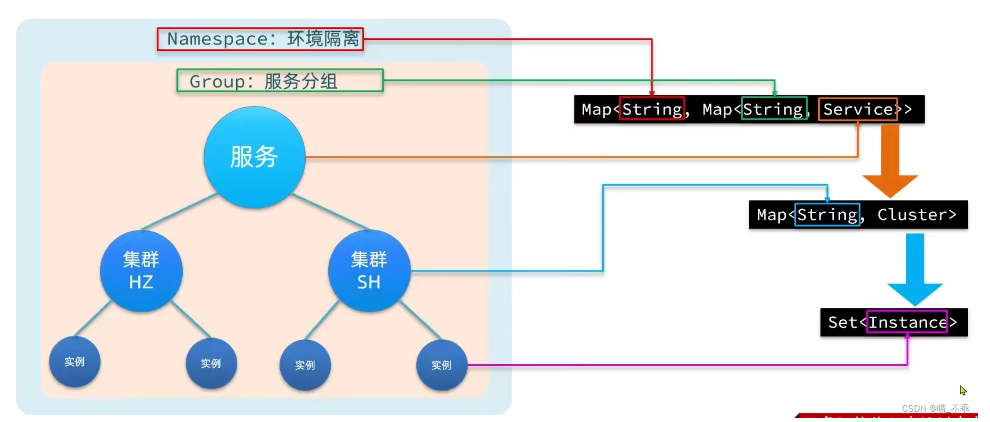
启动 ProviderApplication 和 ConsumerApplication ，调用 http://localhost:8080/echo/2018，返回内容为 Hello Nacos Discovery 2018。

# Open API指南

https://nacos.io/zh-cn/docs/open-api.html

# Nacos服务注册表结构

Nacos最外层是namespace隔离环境，然后是group对服务进行分组，然后就是服务，一个服务下有多个集群，集群下有多个实例。对应Java代码，Map<String,Map<String,Service>>，最外层的key是namespaceId，值是map，内部map大的key是group，拼接serviceName（group@@serviceName），值是service对象，service对象内部又是一个map，key是集群名称，值是Cluster对象，Cluster对象内部维护了实例对象集合。



# 消费者是如何调用提供者的

通过创建RestTemplate对象来实现。

# 为什么要定义bootstrap.yml文件

此文件优先级比较高，可以在服务启动的时候读取配置中心的数据。

# Nacos心跳时间配置及服务快速上下线

https://blog.csdn.net/jxlhljh/article/details/119875072

# Nacos配置中心宕机，还能读取到配置吗

可以，客户端读取到配置信息以后会缓存在本地内存，因此可以在内存中拿到数据。

# 如何确定实例状态

通过发送心跳包，5秒发送一次，如果15秒没有回应，则说明服务出现了问题，如果30秒后没有回应，则说明服务已经停止。

# Nacos CAP

Nacos的 CP 和 AP 架构的选择，取决于我们配置的服务实例是临时实例还是持久实例。



# Nacos和Eureka区别

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 1、接口方式

Nacos与Eureka都对外暴露了Rest风格的API接口，用来实现服务注册、发现等功能。

## 2、实例类型

Nacos的实例有永久和临时实例之分，而Eureka只支持临时实例。

## 3、健康检测

Nacos对临时实例采用心跳模式检测，对永久实例采用主动请求来检测，Eureka只支持心跳模式。

## 4、服务发现

Nacos支持定时拉取和订阅推送两种模式，Eureka只支持定时拉取模式。

# Nacos如何支撑阿里内部数十万服务注册压力

Nacos内部接收到注册的请求时，不会立即写数据，而是将服务注册的任务放入一个阻塞队列就立即响应给客户端。然后利用线程池读取阻塞队列中的任务，异步来完成实例更新，从而提高并发写能力。

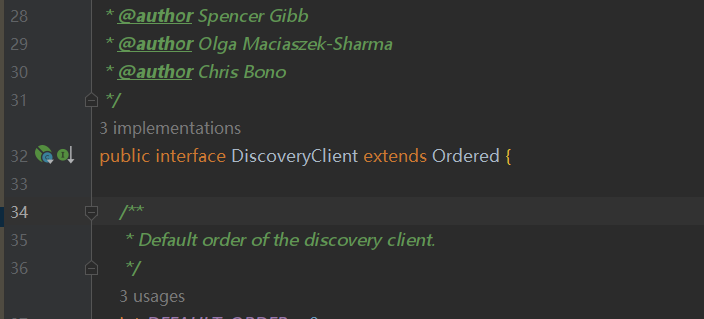
# 接口

## DiscoveryClient

### 基本说明

表示通常可用于发现服务（例如 Netflix Eureka 或 consul.io）的读取操作。

### 源码



### 方法

#### List<ServiceInstance> getInstances(String serviceId);

获取实例

# 类

## ClientWorker

## NotifySubscriberRequest

### 基本概念

NotifySubscriberRequest类是Nacos中用于实现通知订阅的请求类，主要用于向服务端发送订阅通知的请求。

该类包含以下主要属性：

serviceName：String类型，表示服务名。

groupName：String类型，表示服务所属组名。

clusters：String类型，表示要订阅的集群。

udpPort：int类型，表示UDP端口号。

clientIP：String类型，表示客户端IP地址。

namespaceId：String类型，表示命名空间ID。

encode：String类型，表示编码格式。

app：String类型，表示应用名。

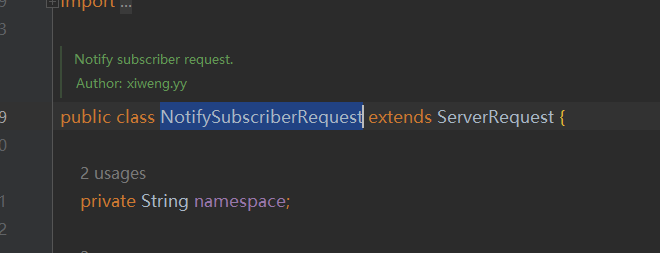
instanceHeartbeatInterval：long类型，表示实例心跳间隔时间。

tenant：String类型，表示租户信息。

以上属性可以通过构造函数或setter方法进行设置和获取。

当客户端订阅服务时，会通过NotifySubscriberRequest对象向Nacos服务端发送订阅请求，如果服务端有相关服务的配置发生变更，就会向订阅该服务的客户端发送通知消息，并将相关配置信息返回给订阅者。

### 源码



### 方法

#### public ServiceInfo getServiceInfo()

#### public String getServiceName()

#### public String getGroupName()

## PropertyKeyConst

### 基本概念

PropertyKeyConst 类是 Nacos（一个开源的动态服务发现、配置和服务管理平台）中的一个工具类，主要用于定义 Nacos 配置中的属性键（Property Key），在 Nacos 中，配置信息通过属性键-属性值的方式进行存储和获取。PropertyKeyConst 类集中定义了一系列常量，用于表示 Nacos 配置中常用的属性键。这样做的目的是为了方便在代码中引用这些属性键，并避免硬编码。开发者可以直接使用这些常量来获取相应的属性值，而无需手动输入属性键的字符串。

### 主要功能

#### 1、定义属性键常量

PropertyKeyConst 类定义了一些常用的属性键常量，例如NAMESRV\_ADDR、NAMESPACE、SERVER\_ADDR等。这些常量对应了 Nacos 配置中的不同属性，开发者可以直接使用这些常量来访问或修改相应的属性值。

#### 2、提供默认属性键

PropertyKeyConst 类还定义了一些默认的属性键常量，用于指定默认的配置属性。例如，DEFAULT\_NAMESPACE 常量指定了默认的命名空间。

#### 总结

通过使用 PropertyKeyConst 类提供的常量，开发者可以更方便地访问 Nacos 配置中的属性，并降低硬编码的风险。此外，由于这些常量是在类级别上定义的，因此可以提供更好的类型安全性和可维护性。

## HttpClientBeanHolder

### 基本概念

主要用于管理 HttpClient 实例。在 Nacos 中，网络通信模块使用 HttpClient 来发送 HTTP 请求和接收响应。为了保证 HttpClient 的线程安全性和高效性，Nacos 使用了连接池技术，即通过 HttpClient 创建连接池来管理网络连接。而 HttpClientBeanHolder 类就是用来持有 HttpClient 实例，并提供相关操作的工具类。

### 主要功能

1、创建HttpClient实例

HttpClientBeanHolder 类会在初始化过程中创建一个 HttpClient 实例，并设置一些默认的配置，例如超时时间、最大连接数等。

2、获取 HttpClient 实例

HttpClientBeanHolder 提供了静态方法 getHttpClient() 来获取 HttpClient 实例。这个方法会返回创建的 HttpClient 对象，如果实例不存在，则会调用 createHttpClient() 方法来创建新的实例。

3、关闭 HttpClient 实例

HttpClientBeanHolder 也提供了静态方法 destroyHttpClient() 来关闭 HttpClient 实例。这个方法会关闭连接池并释放资源。

### 总结

通过使用 HttpClientBeanHolder 类，Nacos 可以在需要的时候获取 HttpClient 实例，以便进行网络通信。这样可以有效地提高网络请求的性能和效率，并提供线程安全性。

## PushReceiver

### 基本概念

在 Nacos 中，PushReceiver 类是一个用于接收推送消息的类。Nacos 是一个动态服务发现、配置和服务管理平台，它支持实时推送配置和服务变更通知给客户端。PushReceiver 类是 Nacos 客户端中的一个组件，主要用于接收来自 Nacos 服务器的推送消息，并将消息交给相应的处理器进行处理。

### 主要功能

#### 1、建立长连接

PushReceiver 在初始化过程中会与 Nacos 服务器建立一个长连接，以便实时接收推送消息。它使用 HTTP 协议进行通信，并通过长轮询方式获取服务器的推送消息。

#### 2、接收推送消息

一旦建立了与服务器的长连接，PushReceiver 就会持续监听服务器的推送消息。当有新的消息到达时，PushReceiver 将会立即接收并将消息传递给相应的处理器进行处理。

#### 3、处理推送消息

PushReceiver 将接收到的推送消息传递给注册的消息处理器进行处理。消息处理器可以是针对不同类型消息的具体业务逻辑，例如配置变更通知处理器、服务变更通知处理器等。开发者可以根据自己的需求实现自定义的消息处理器，并将其注册到 PushReceiver 中。

### 总结

通过使用 PushReceiver 类，Nacos 客户端能够实时接收到服务器推送的配置变更和服务变更通知，从而能够及时地响应和处理这些变更，保持与 Nacos 服务器的数据同步。

## NacosWatch

### 基本概念

NacosWatch 类是 Nacos 客户端库中的一个重要组件，它用于监听 Nacos 服务器上配置项的变化，并在配置变化发生时触发相应的回调或处理逻辑。

### 作用

#### 1、配置监听

NacosWatch 的主要作用是监听 Nacos 服务器上的配置项变化。Nacos 是一个服务发现和配置管理平台，可以用于动态管理应用程序的配置信息。NacosWatch 通过与 Nacos 服务器建立连接并订阅特定的配置项，持续地监测这些配置项的变化。

#### 2、实时通知

当 Nacos 服务器上的配置项发生变化时，NacosWatch 能够实时接收到通知。这使得应用程序能够及时感知到配置的更新，以便采取相应的动作。例如，当配置项的值被修改时，NacosWatch 可以立即通知应用程序进行相应的更新操作。

#### 3、触发回调

NacosWatch 允许开发人员注册回调函数，当配置发生变化时自动触发这些回调。通过这种方式，应用程序可以定义自己的逻辑来处理配置变化。比如，可以在配置变化时重新加载配置、更新缓存、重启相关服务等。

#### 4、支持多种配置类型

NacosWatch 支持监听不同类型的配置，包括普通的文本配置、Properties 配置、JSON 配置等。应用程序可以根据需要选择适合自己的配置类型，并使用 NacosWatch 监听其变化。

#### 5、高可靠性

NacosWatch 具备高可靠性，它会在网络故障或连接中断后自动进行重连，以保证与 Nacos 服务器的连接稳定。当恢复连接时，NacosWatch 会重新订阅配置项，并继续监测配置的变化。

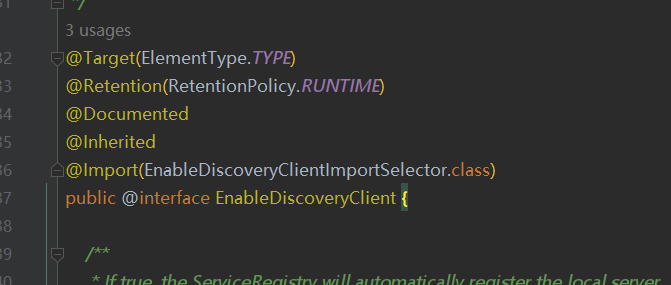
### 总结

NacosWatch 类提供了对 Nacos 服务器上配置项变化的监听功能，可以帮助应用程序实现实时感知和处理配置的更新。它是构建基于 Nacos 的动态配置管理系统的关键组件之一。开发人员可以使用 NacosWatch 来简化配置管理的过程，并使应用程序具备更高的灵活性和实时性。

# 注解

## @EnableDiscoveryClient

### 源码



### 属性

#### autoRegister

默认是true，自动注册服务到本地服务器。

### 作用

能够让注册中心发现，扫描到该服务。从Spring Cloud Edgware开始，@EnableDiscoveryClient 或@EnableEurekaClient **可省略**，只需加上相关依赖，并进行相应配置，即可将微服务注册到服务发现组件上。

## @RefreshScope

### 基本概念

@RefreshScope 是 Spring Cloud 提供的一个注解，用于标识需要在运行时刷新的 Bean。当应用中的配置发生变化时，可以通过发送 POST 请求到 /actuator/refresh 端点来触发配置的刷新。但是并非所有的 Bean 都会在配置刷新后自动更新，而是需要使用 @RefreshScope 注解来标识需要刷新的 Bean。

### 作用范围

1、应用于 Spring Boot 的 @Configuration 类或 @Bean 方法上。

2、可以应用于带有 @Controller 或其他 @Component 注解的类上。

### 刷新方法

当应用的配置发生变化并且触发了 /actuator/refresh 端点后，被 @RefreshScope 注解标记的 Bean 将会重新创建，从而使得它们能够使用最新的配置值。

### 注意事项

1、@RefreshScope 注解只适用于被 Spring 容器管理的 Bean，例如由 @ComponentScan 扫描到的 Bean。

2、对于直接使用 new 关键字创建的对象，以及非容器管理的对象，无法应用 @RefreshScope 注解。

通过使用 @RefreshScope 注解，我们可以动态地刷新配置，而无需重启应用。这对于需要在不停机的情况下更新配置的场景非常有用，例如微服务架构中的配置管理。

## @LoadBalanced

### 基本概念

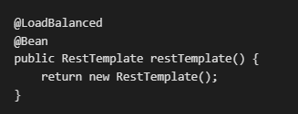
@LoadBalanced 是 Spring Cloud 提供的一个注解，用于标识带有负载均衡能力的 RestTemplate 或 WebClient。在微服务架构中，通常会有多个相同功能的服务实例运行在不同的主机上，用于提供高可用性和扩展性。负载均衡是一种将请求分发到多个服务实例的机制，以实现负载均衡和故障恢复。

### 作用范围

可以应用于 Spring Boot 的 RestTemplate 或 WebClient Bean 上。当使用 @LoadBalanced 注解标识了 RestTemplate 或 WebClient 后，它们将具备负载均衡的能力。在发送请求时，RestTemplate 或 WebClient 会自动选择目标服务的一个实例进行调用。如果实例不可用，负载均衡机制会自动切换到可用的实例。

### 配置方式

1、在使用 RestTemplate 时，只需在创建 RestTemplate Bean 的方法上添加 @LoadBalanced 注解即可。



2、在使用 WebClient 时，可以通过 WebClient.builder().baseUrl(baseUrl).build() 创建实例，并在调用相关方法之前调用 .filter(loadBalancerExchangeFilterFunction) 方法来添加负载均衡功能。

### 总结

通过使用 @LoadBalanced 注解，我们可以很方便地为 RestTemplate 或 WebClient 添加负载均衡的能力，从而实现在多个服务实例之间分发请求，并提高应用的可用性和性能。

# Nacos常见问题

https://nacos.io/zh-cn/docs/faq.html

## 1、Nacos如何支持多环境

在日常使用中常常需要不同的环境，比如日常，预发，线上环境，如果是逻辑隔离可以使用命名空间，Nacos支持命名空间来支持多环境隔离，可以在Nacos控制台创建多个命名空间。如果需要物理隔离，就要部署多套Nacos环境。

# 问题

## 1、Spring Cloud LoadBalancer 当前使用默认缓存。 虽然此缓存实现对开发和测试很有用，但建议在生产中使用 Caffeine 缓存。您可以切换到使用 Caffeine 缓存，方法是将它和 org.springframework.cache.caffeine.CaffeineCacheManager 添加到类路径中。

## 2、发布失败，请检查参数是否正确

## 3、ERROR 16288 [com.alibaba.nacos.client.naming.updater] com.alibaba.nacos.client.security.SecurityProxy [SecurityProxy] login http request failed url: http://localhost:8848/nacos/v1/auth/users/login, params: {username=nacos}, bodyMap: {password=nacos}, errorMsg: Server returned HTTP response code: 500 for URL: http://localhost:8848/nacos/v1/auth/users/login?username=nacos

错误信息显示是在登录 Nacos 配置中心时发生了问题，返回了一个 HTTP 500 的错误码。这个错误通常表示服务器内部发生了错误，导致无法正常处理请求。

解决方法：

1、检查 Nacos 服务是否正常启动。

2、检查 Nacos 配置文件：检查 Nacos 配置文件中的相关配置项是否正确。特别是 nacos.security.proxy.username 和 nacos.security.proxy.password 这两个配置项，确保它们的值与 Nacos 配置中心的用户名和密码匹配。

3、检查网络连接：确保网络连接正常，能够正常访问 Nacos 服务所在的主机。尝试使用命令行工具（如ping命令）检查与 Nacos 服务主机的连通性。

4、检查 Nacos 配置中心的日志：查看 Nacos 配置中心的日志，查找是否有其他错误或异常信息提供更多线索。

5、是否存在其他冲突：确保没有其他应用程序或进程占用了 Nacos 所使用的端口。如果存在冲突，请修改 Nacos 的端口配置。

6、更新 Nacos 版本：如果以上步骤都没有解决问题，考虑升级或降级 Nacos 的版本，以修复可能存在的版本兼容性或 bug 导致的问题。

## 4、ERROR 18856 [com.alibaba.nacos.client.naming.security] com.alibaba.nacos.client.security.SecurityProxy login failed: {"code":500,"message":"caused: Cannot invoke \"com.alibaba.nacos.plugin.auth.impl.jwt.NacosJwtParser.getExpireTimeInSeconds(String)\" because \"this.jwtParser\" is null;","header":{"header":{"Accept-Charset":"UTF-8","Authorization":"Bearer","Connection":"close","Content-Length":"142","Content-Security-Policy":"script-src 'self'","Content-Type":"text/html;charset=UTF-8","Date":"Mon, 10 Jul 2023 01:01:22 GMT","Vary":"Access-Control-Request-Headers"},"originalResponseHeader":{"Authorization":["Bearer"],"Connection":["close"],"Content-Length":["142"],"Content-Security-Policy":["script-src 'self'"],"Content-Type":["text/html;charset=UTF-8"],"Date":["Mon, 10 Jul 2023 01:01:22 GMT"],"Vary":["Access-Control-Request-Headers","Access-Control-Request-Method","Origin"]},"charset":"UTF-8"}}

解决方法：需要把配置文件中的nacos用户名和密码删除就可以了。

5、