# 统一配置管理

## 认识统一配置管理

1. 在Spring Boot项目中，默认会提供一个application.properties或application.yml文件，我们可以把一些全局性的配置或者需要动态维护的配置写入该文件，比如数据库连接、功能开关、限流阈值、服务器地址等。为了解决不同环境下服务连接配置等信息的差异，Spring Boot还提供了基于spring.profiles.active={profile}的机制来实现不同环境的切换。
2. 随着单体架构向服务化结构及微服务架构的演进，各个应用自己独立维护的本地配置的方式开始显露出它的不足之处。

·配置的动态更新：在实际应用中会有动态更新配置的需求，比如修改服务器连接

地址、限流的配置等。在传统模式下，需要手动修改配置文件并且重启应用才能

生效，这种方式效率太低，重启也会导致服务暂时不可用。

·配置集中式管理：在微服务架构中，某些核心服务为了保证高性能会部署上百个

节点，如果在每个节点中都维护一个配置文件，一旦配置文件中的某个属性需要

修改，可想而知，工作量是巨大的。

·配置内容的安全性和权限：配置文件随着源代码统一提交到代码库中，容易造成

生产环境配置信息的数据泄漏。

·不同部署环境下配置的管理：前面提到过通过profile机制来管理不同环境下的

配置，这种方式对于日常维护来说比较烦琐。

统一配置管理就是弥补上述不足的方式，简单来说，最基本的方法是把各个应用系统中的某些配置放在一个第三方中间件上进行统一维护。然后，对于统一配置中心的数据的变更需要推送到相应的服务节点实现动态更新。所以在微服务架构中，配置中心也是一个核心组件。

1. 配置中心的开源解决方案有很多，比如ZooKeeper、Disconf、Apollo、Spring Cloud Config、QConf、Nacos等。不管是哪一种解决方案，它的核心功能是不会变的。

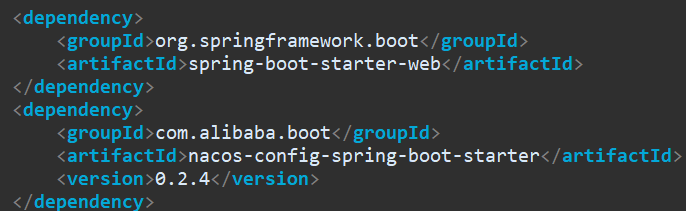
## Nacos配置中心简介

Nacos是Alibaba开源的中间件，在前面的学习中讲解了如何使用Nacos实现服务注册与发现。在Nacos的架构图中有两个模块，分别是Config Service和Naming Service。其中Config Service就是Nacos用于实现配置中心的核心模块，它实现了对配置的CRUD、版本管理、灰度管理、监听管理、推送轨迹、聚合数据等功能。

## Nacos集成Spring Boot实现统一配置管理

Nacos是一个独立组件，它可以独立部署和应用，在继承到Spring Cloud之前，我们可以结合Spring Boot来实现统一配置管理。

1. **项目准备**
2. 创建一个Spring Boot工程spring-boot-nacos-config，并添加依赖



1. 在application.properties中添加Nacos Server的地址



1. 创建NacosConfigController，用于从Nacos Server动态读取配置



@NacosValue：设置属性的值，其中info表示key，而Local Hello World代表默认值。也就是说，如果key不存在，则使用默认值。这是一种高可用策略，在实际应用中，我们需要尽可能考虑到在配置中心不可用的情况下如何保证服务的可用性。

@NacosPropertySource：用于加载dataId为example的配置源，authRefreshed表示开启自动更新。

1. 启动Nacos服务。
2. **创建配置**

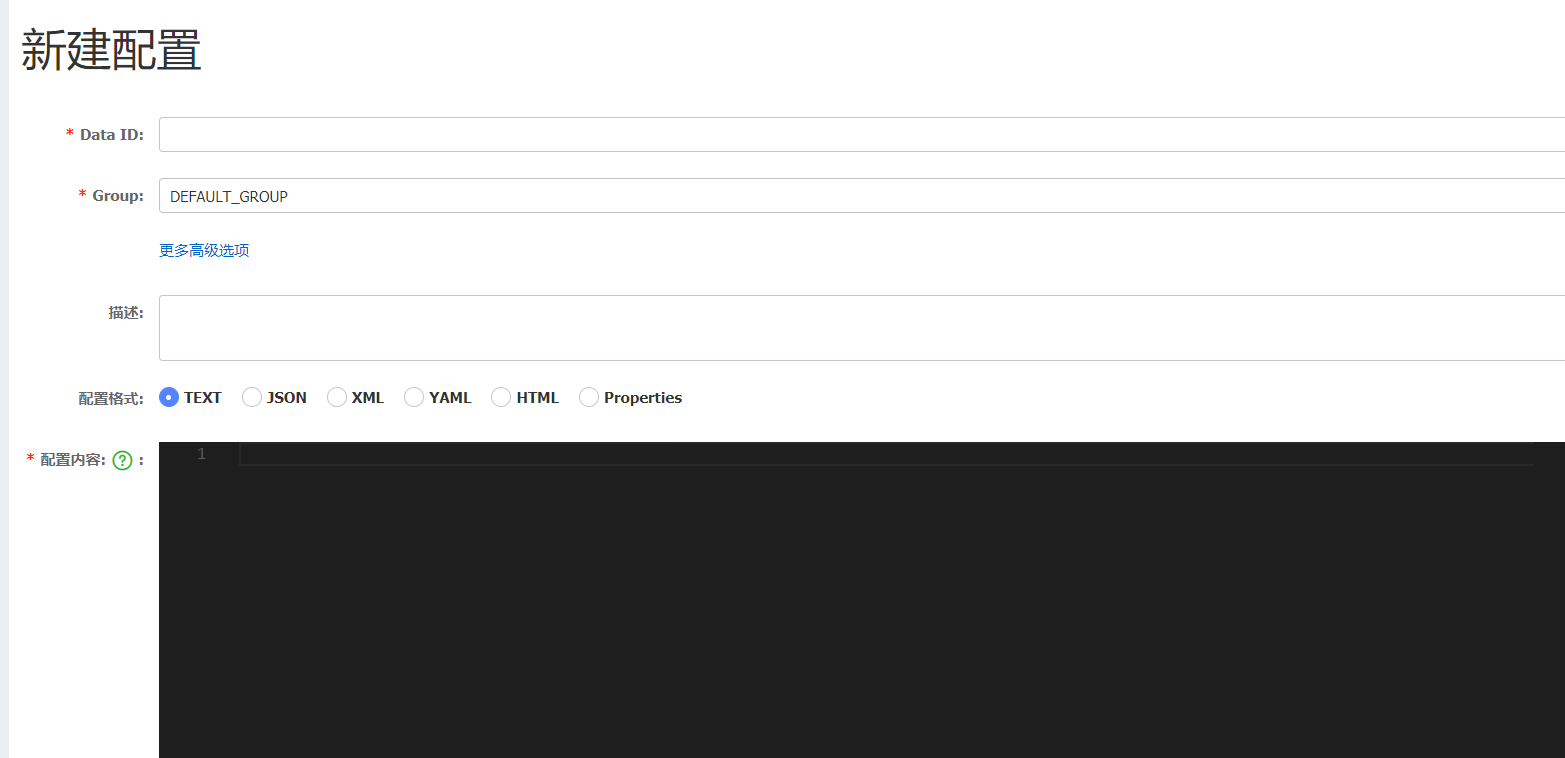
创建配置有两种方式：

·在Nacos控制台上创建

·使用Open API创建

1. 使用Nacos控制台创建

登录Nacos控制台，进入【配置管理】-->【配置列表】，单击右上角的【+】进入如下界面：



配置字段说明如下，其中Data ID和Group会在后续章节详细说明。

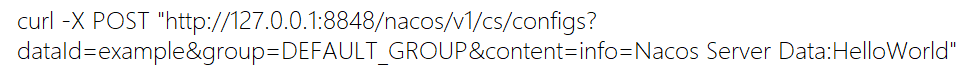
·Data ID：表示Nacos中某个配置集的ID，通常用于组织划分系统的配置集。

·Group：表示配置所属的分组。

·配置格式：当前配置内容所遵循的格式。

1. Open API创建方式

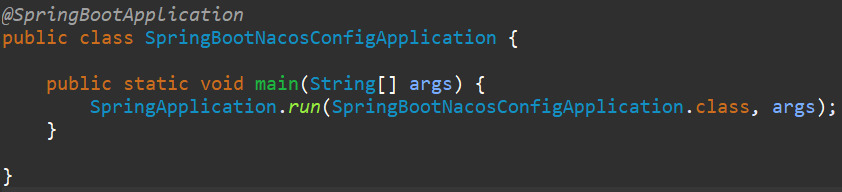
通过如下指令进行配置的创建：



使用curl执行可能会报505错误，可以使用Postman来完成。

1. **启动服务并进行测试**

启动Spring Boot项目



使用curl命令进行测试：



可见，使用的是Nacos配置的数据。如果通过控制台或者Open API方式修改配置的数据，重新执行curl命令将会获得最新的结果。（如果将数据删除则依然可以获得）

# Spring Cloud Alibaba Nacos Config

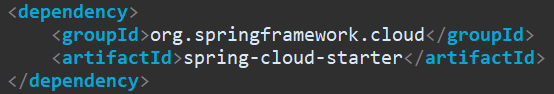
## 认识Spring Cloud Alibaba Nacos Config

Spring Cloud Config是Spring Cloud生态中统一配置管理的组件，它为外部化配置提供了服务端和客户端支持，包含Config Server和Config Client两部分。而Spring Cloud Alibaba Nacos Config是Config Server和Config Client的替代方法。下面将演示如何基于Spring Cloud生态来集成Nacos实现配置中心。

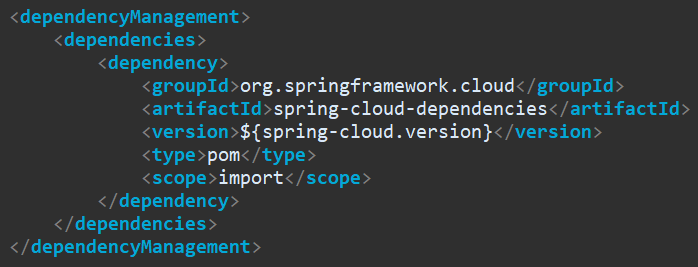
## Nacos Config的基本应用

在Spring Cloud生态下Nacos Config的使用步骤如下：

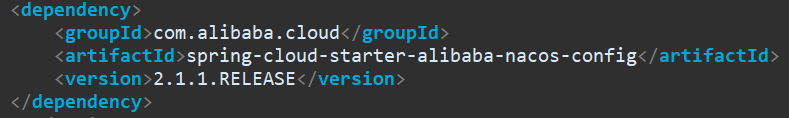
1. 创建一个名为spring-cloud-nacos-config的Spring Boot类型项目，添加spring-cloud-starter依赖



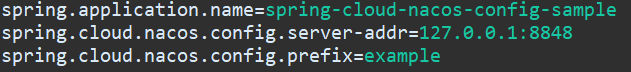
对其进行版本管理



1. 添加spring-cloud-starter-alibaba-nacos-config依赖



1. 在resources目录下创建bootstrap.properties文件，并在bootstrap.properties中添加Nacos Server的连接地址



spring.cloud.nacos.config-server-addr：Nacos配置中心的地址。如果地址是域名，配置的方式应该是域名:port，即便是80端口，也需要将80端口带上。

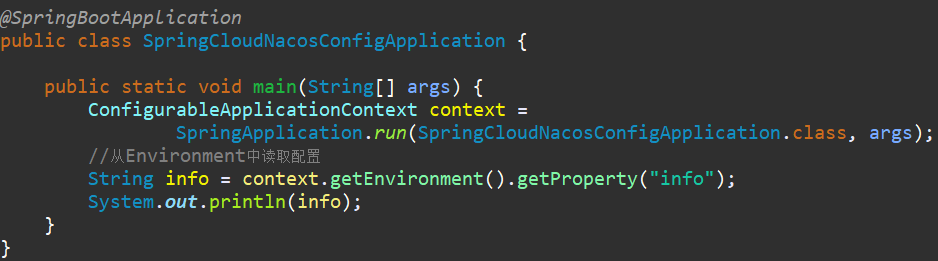
spring-cloud.nacos.config.prefix：Nacos配置中心上的DataID的前缀

需要注意，这些配置项是放在bootstrap.properties文件中的。在Spring Boot中有两种上下文配置，一种是bootstrap，另一种是application。Bootstrap是应用程序的父上下文，也就是说boostrap加载优先于application。由于在加载远程配置之前，需要读取Nacos配置中心的服务地址信息，所以Nacos服务地址等属性配置需要放在bootstrap.properties文件中。

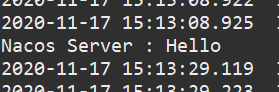
1. 在Nacos Console中创建如下配置



1. 在启动类中，读取配置中心的数据



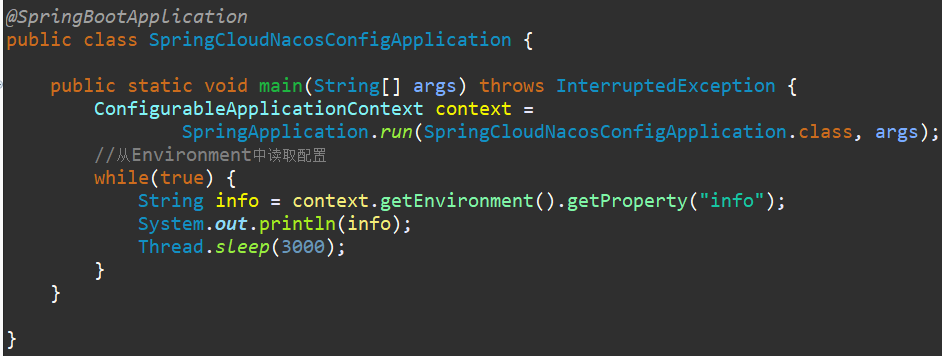
启动，查看是否能够获取在Nacos控制台中配置的数据：



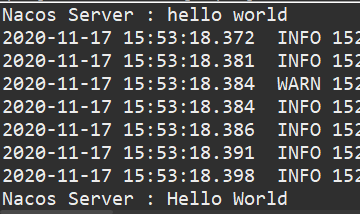
获取成功。

## 动态更新配置

配置中心必然需要支持配置的动态更新，也就是在配置中心上修改配置值之后，应用程序需要感知值的变化。下面通过一段代码来验证动态更新的实现：



通过一个while循环不断读取info属性，当info属性发送变化时，会输出最新的配置信息。



## 基于DataID配置YAML的文件扩展名

**1、认识匹配DataID的规则**

Spring Cloud Alibaba Nacos Config从Nacos Config Server中加载配置时，会匹配DataID。在Spring Cloud Nacos的实现中，DataID默认规则如下：

**${prefix}-${spring.profile.active}.${file-extension}**

·在默认情况下，会去Nacos服务器上加载DataID以${spring.app lication.name}.${f

ile.extension:properties为前缀的基础配置。比如在学习Spring Cloud Alibaba Na

cos Config时，在bootstrap.proeprties文件中配置了属性spring.application.name

=spring-cloud-nacos-config-sample，在不通过spring.cloud.nacos.config.prefix=exa

mpe属性，则会加载DataID为spring-cloud-nacos-config-sample.properties的配置

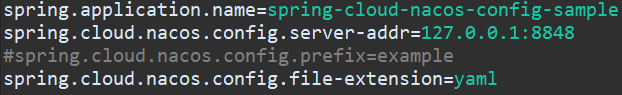
信息。如果明确指定了spring.cloud.nacos.config.prefix=example属性，则会加载DataID=example的配置。

·spring.profile.active表示多环境支持，下节会详细讲解。

**2、使用YAML格式的配置**

在实际应用中，如果在Nacos中配置的是YAML格式的数据，Nacos Config也提供了YAML配置格式的支持，执行步骤如下：

1. 在bootstrap.proeprties中声明spring.cloud.nacos.config.file-extension=yaml



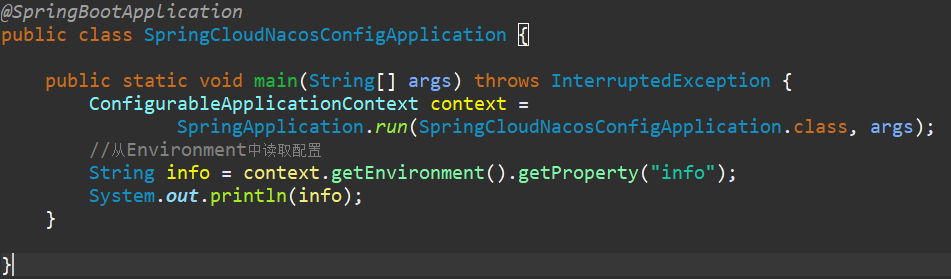
同时为了演示默认DataID的匹配规则，将spring.cloud.nacos.config.prefix属性注释，表示使用默认的DataID匹配规则。

1. 在Nacos控制台增加如下配置，其中配置格式为yaml格式

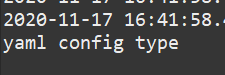


注意：配置内容的格式要遵循yaml格式。

1. 运行启动方法，



获得结果如下：



获取成功。

## 不同环境的配置切换

**1、Spring Boot中的多环境切换**

在Spring Boot中，可以基于spring.profiles.active实现不同环境下的配置切换，这在实际工作中用得比较多。很多公司都会有开发环境、测试环境、预生产环境、生产环境等，服务部署在不同的环境下，有些配置是不同的，所以我们希望能够通过一个属性非常方便地指定当前应用部署的环境，根据不同的环境加载对应的配置。基于Spring Boot项目的多环境支持配置步骤如下：

1. 在resources目录下根据不同环境创建不同的配置文件。

·application-dev.properties

·application-test.properties

·application-prod.properties

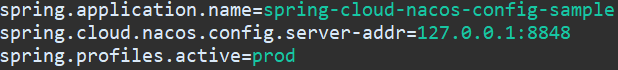
分别表示开发环境、测试环境、生产环境。

1. 在application.properties配置文件中通过spring-profiles.active=${env}来指定当前使用哪个环境的配置，如果${env}的值为prod，表示使用application-prod来指定使用的环境配置。

**2、DataID对多环境切换的支持**

在Spring Cloud Alibaba Nacos Config中加载Nacos Config Server中的配置时，不仅会加载DataID以${spring.application.name}.${file-extension:properties}为前缀的基础配置，还会加载DataID为${spring.application.name}.${profile}.${file-extension:properties}的基础配置，这样的方式为不同环境的切换提供了非常好的支持。配置方式和Spring Boot相同，具体的实现步骤如下：

1. 在bootstrap.properties中声明spring.profiles.active=prod，需要注意的是必须要在boostrap.properties中声明。

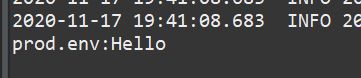


1. 在Nacos控制台上新增两个DataID的配置项





1. 运行启动方法。如果spring.profile.active=prod，将会获得以下结果：

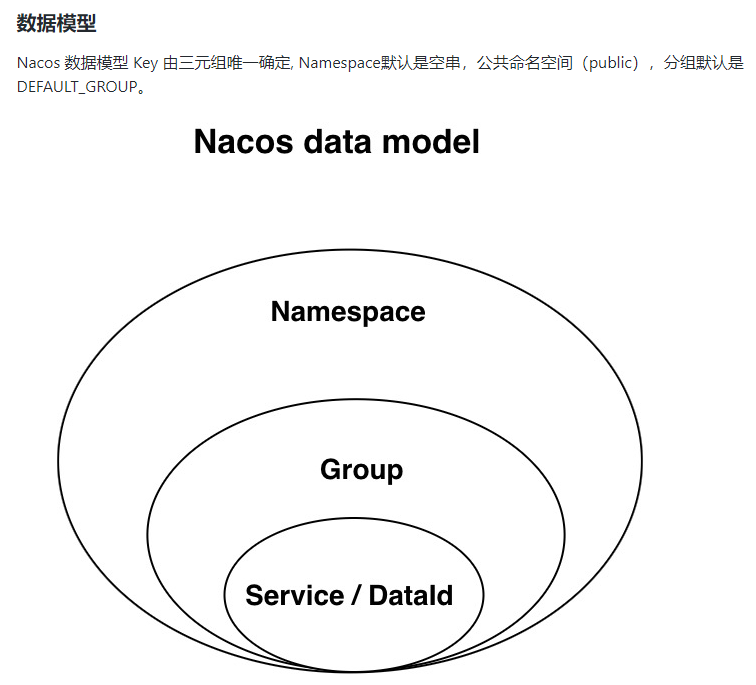


可以发现，基于Nacos Config实现不同环境的切换和本地配置的不同环境的切换没有任何区别，如果我们需要切换到测试环境，只需要修改spring.profiles.active=test即可。不过这个属性要写死在bootstrap.properties文件中的，修改起来显得很麻烦。通常的做法是通过-Dspring.profiles.active=${profile}参数来指定环境，已达到灵活切换的目的。

# Nacos Config中的 Namespace和Group

## 认识Namespance和Group

在本章前面的学习中，使用Nacos Config时都采用默认的Namespace:public和Group:DEFAULT\_GROUP，从名字可以基本能够猜测到它们的作用。下图是Nacos提供的数据模型，它的数据模型Key是由三元组来进行唯一确定的。



其中Namespace用于解决多环境及多用租户数据的隔离问题，比如在多套不同的环境下，可以根据指定的环境创建不同的Namespace，实现多环境的隔离，或者在多用户的场景中，每个用户可以维护自己的Namespace，实现每个用户的配置数据和注册数据的隔离。需要注意的是，在不同的Namespace下，可以存在相同的Group或DataId。

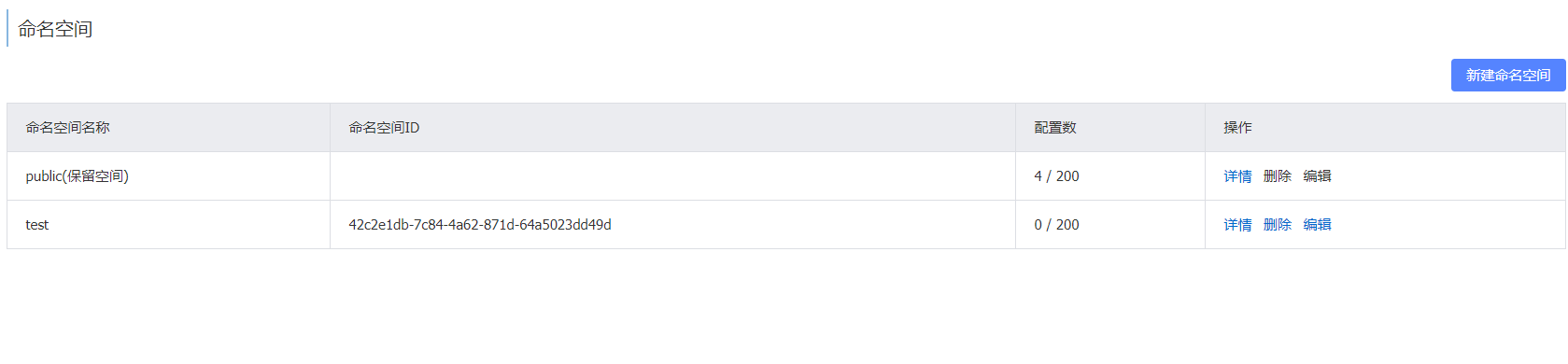
Group是Nacos中用来实现DataID分组管理的机制，它可以实现不同的@Service/DataID的隔离。对于Group的用法，其实没有固定的规定，比如它可以实现不同环境下的DataID分组，也可以实现不同应用或组件下使用相同配置类型的分组，比如database\_url。

官方的建议是，通过Namespace来区分不同的环境，而Group进行基本的定调，避免使用上的混乱。

## 自定义Namespace和Group

认识了Namespace和Group后，下面讲以下Spring Cloud Alibaba Nacos Config如何实现自定义Namespace和Group。

1. **Namespace**
2. Nacos控制台的【命名空间】下，创建一个命名空间，如下创建了一个名为test的命令空间：



1. 命名空间创建好后，在bootstrap.properties中添加配置：



其值对应的是命名空间的ID，表示使用该命名空间中的DataID。

1. **Group**

Group不需要提前创建，只需要在创建的时候指定，配置方法如下

1. 在Nacos控制台的“新建配置”界面中指定配置所属的Group



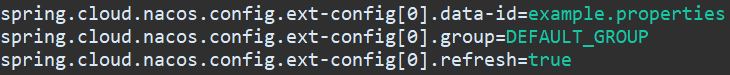
1. 在bootstrap.properties中添加如下配置即可：



表示使用该组的DataID

**3、DataID**

DataID是Nacos中某个配置集的ID，它通常用于组织划分系统的配置集。在前面的示例中都是通过配置文件的名字来进行配置的划分的，也可以通过Java包的全路径来划分，主要取决于DataID的使用维度。同时Spring Cloud Alibaba Nacod Config同样支持自动以DataID配置。



表示在项目中引入使用名为example.properties且分组为DEFAULT\_GROUP的DataID，且动态刷新感知最新的配置值。上述配置中可以看到spring.cloud.nacos-config.ext-config[n]支持多个DataID的扩展配置，包含三个属性：data-id、group、refresh。其中：

·spring.cloud.nacos-config.ext-config[n].data-id指定Nacos Config的DataID

·spring.cloud.nacos-config.ext-config[n].group指定DataID所在的组

·spring.cloud.nacos-config.ext-config[n].refresh控制DataId在配置发送变更时

是否动态刷新以感知最新的配置值。默认是false，也就是不会实现动态刷

新。

在使用过程中，有两个注意点：

·spring.cloud.nacos.config.ext-config[n].data-id属性的值必须要带文件的扩展

名，可以支持properties、yaml、json等。

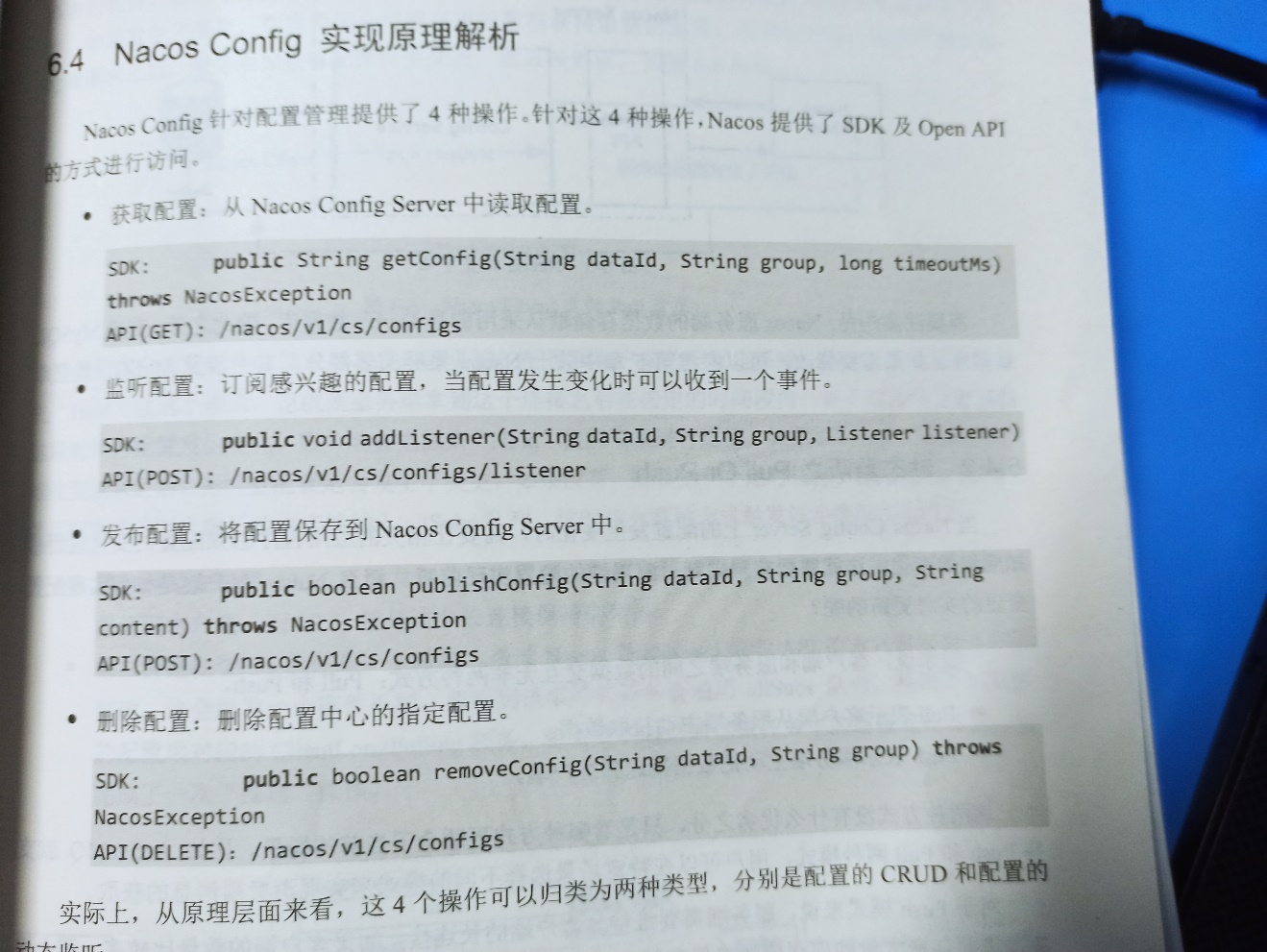
·spring.cloud.nacos.config.ext-config[n].data-id配置多个DataID时，n的值越

大，优先级越高。

通过自定义扩展的DataID配置，既可以解决多个应用的配置共享问题，在支持一个应用有多个配置文件的情况。需要注意的是，在ext-config和${spring.application.name}.${file.extension:properties}都存在的情况下，优先级高的是后者。

# Nacos Config实现原理解析

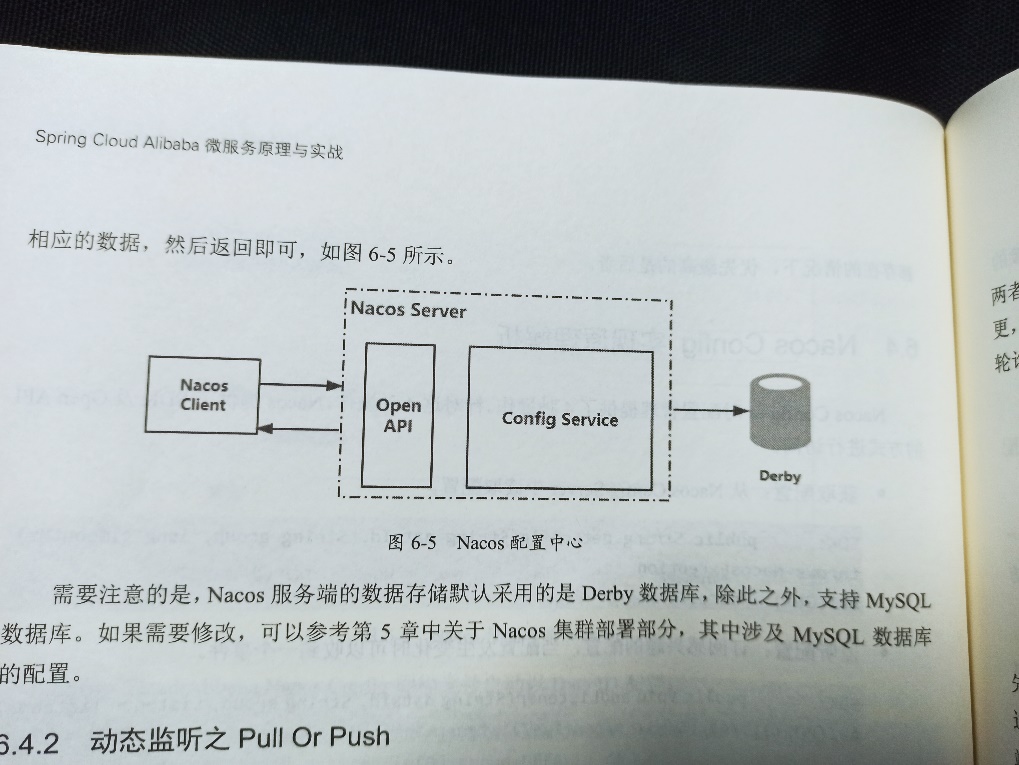
Nacos Config针对配置管理提供了4种操作，针对这4种操作，Nacos提供了SDK及Open API的方式访问。



实际上，从原理层面来看，这4个操作可以归类为两种类型，分别是配置的CRUD和配置的动态监听。

## 配置的CURD

对于Nacos Config来说，其实就是提供了配置的集中式管理功能，然后对外提供CRUD的访问接口使得应用系统可以完成配置的基本操作。这种场景并不复杂，对于服务端来说，无非就是配置如何存储，以及是否需要持久化，对于客户端来说，就是通过接口从服务器端查询到相应的数据，然后返回即可。如下图：



需要注意的是，Nacos服务端的数据默认采用的是Derby数据库，除此之外，支持MySQL数据库。如果需要修改，可以参考上一章节中关于Nacos集群部署部分，其中涉及MySQL数据库的配置。

## 动态监听

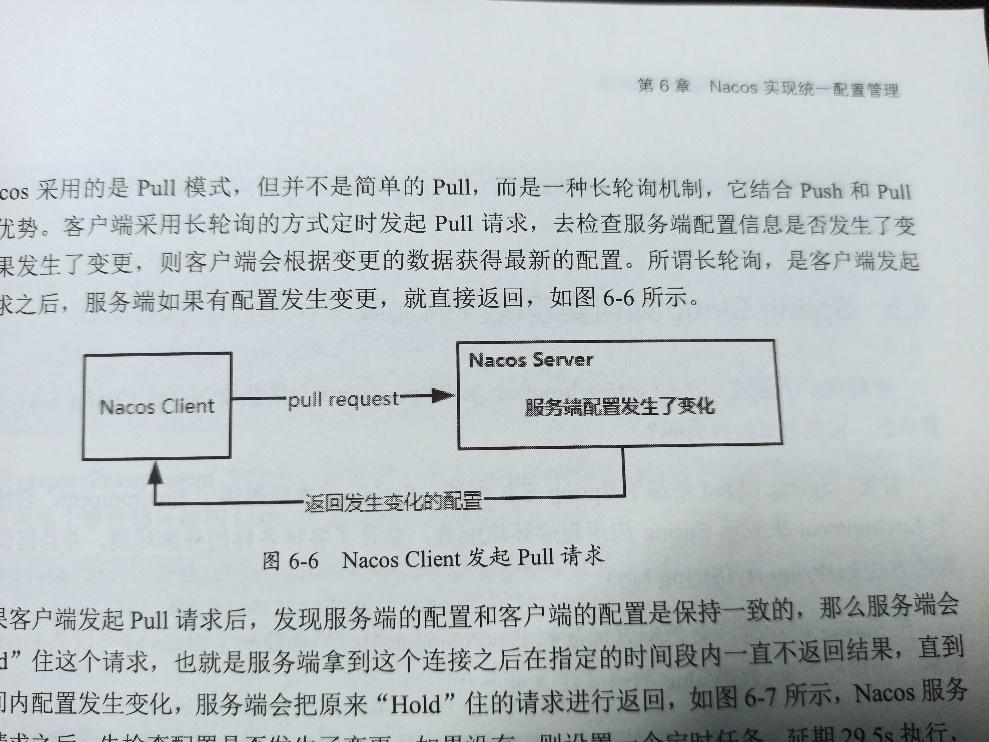
当Nacos Config Server上的配置文发送变化时，需要让相关的应用程序感知配置的变化进而感知应用的变化，这就需要客户端针对感兴趣的配置实现监听。一般来说，客户端和服务端之间的数据交互无非两种方式：Pull（拉取）和Push（推送）。

·Pull表示客户端从服务端主动拉取数据

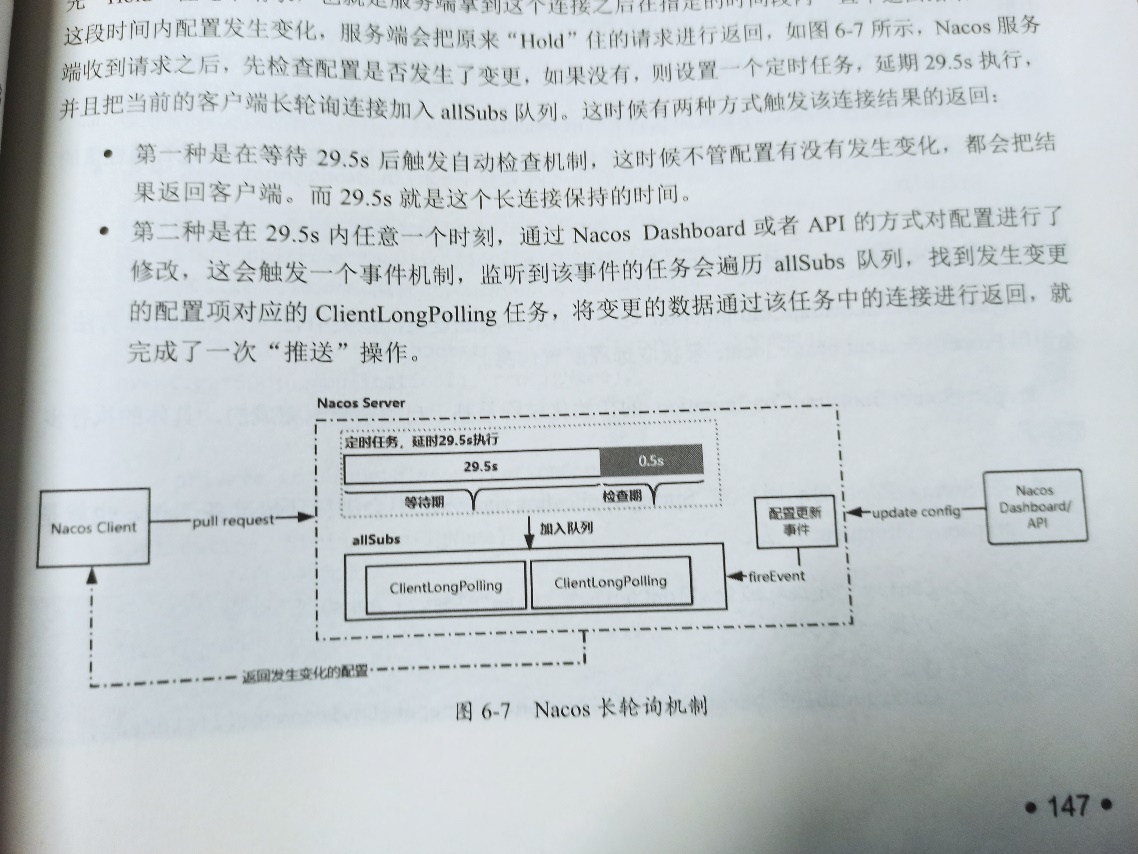
·Push表示服务端主动把数据推送到客户端

1. 这两种方式没有什么优劣之分，只是看哪种方式更适合于当前的场景。比如ActiveMQ就支持Pull和Push两种模式，用户可以在特定场景选择不同的模式来实现消费端消息的获取。
2. 对于Push模式来说，服务端需要维持与客户端的长连接，如果客户端的数量比较多，那么服务端需要耗费大量的内存资源来保存每个连接，并且为了检测连接的有效性，还需要心跳机制来维持每个连接的状态。在Pull模式下客户端需要定时从服务端拉取一次数据，由于定时任务会存在一定的时间间隔，所以不能保证数据的实时性。并且在服务端配置长时间不更新的情况下，客户端的定时任务会做一些无效的PULL。

Nacos采用的Pull模式，但并不是简单的Pull，而是一种长轮询机制，它结合Push和Pull两者的优势。客户端采用长轮询的方式定时发起Pull请求，去检查服务端配置信息是否发送了变更，如果发送了变更，则客户端会根据变更的数据获得最新的配置。所谓的长轮询，是客户端发起轮询请求之后，服务端如果有配置发送变更，就直接返回。如下图所示：



如果客户端发起Pull请求后，发现服务端的配置和客户端的配置是保持一致的，那么服务端会先“Hold”住这个请求，也就是服务端拿到这个连接之后在指定的时间段内一致不返回结果，知道这段时间内配置发生变化，服务端会把“Hold”住的请求进行返回，如下图所示：



Nacos服务端收到请求之后，先检查配置是否发送了变更，如果没有，则设置一个定时任务，延期29.5s执行，并且把当前的客户端长轮询连接加入allSubs队列。这时候有两种方式触发该连接结果的返回：

·第一种是在等待29.5s后触发自动检查机制，这时候不管配置有没有发送变化，

都会把结果返回客户端。而29.5s就是这个长连接保持的事件。

·第二种是在29.5s内任意一个时刻，通过Nacos Config Dashboard或者API的方式

对配置进行了修改，这会触发一个事件机制，监听到该事件的任务会遍历allSubs

队列，找到发生变更的配置项对应的ClientLongPolling任务，将变更的数据通过

该任务中的连接进行返回，就完成了一次“推送”的过程。

这样既能够保证客户端实时感知配置的变化，也降低了服务端的压力。其中，这个长连接的会话超时时间默认是30s。

# Spring Cloud如何实现配置的加载

## 分析

1. 前面演示了通过context.getEnvironment.getProperties(“info”)来获取Nacos Config Server上的配置信息，它是怎么做到的呢？首先，Spring Cloud是基于Spring来扩展的，Spring原本就提供了Environment，它抽象了一个Environment来表示Spring应用程序环境配置，整合了各种各样的外部环境，并且提供统一访问的方法getProperty(String key)。
2. 在Spring启动时，会把配置加载到Environment中。当创建一个Bean时可以从Environment中把一些属性值通过@Value的形式注入业务代码。而在Spring Cloud中，要实现统一配置管理并且动态刷新配置，需要解决的问题有：

·如何将远程服务器上的配置加载到Environment

·配置变更时，如何将新的配置更新到Environment中，保证配置变更时可以进

行属性值的动态刷新。

要想知道如何解决这两个问题，就需要研究PropertySourceBootstrapConfiguration

和PropertySourceLocator这两个类。