,

Spring Cloud Config 总结

Author: 赵立

Creation Date: 2018-05-15

Last Updated:

Document Ref: Spring Cloud Config 总结

Version: 1.0

1. **Title, Subject, Last Updated Date, Reference Number**, **andVersion** are marked by a Word Bookmark so that they can be easily reproduced in the header and footer of documents. When you change any of these values, be careful not to accidentally delete the bookmark. **You can make bookmarks visible by selecting Tools->Options…View and checking the Bookmarks option in the Show region.**
2. To add additional approval lines, press [Tab] from the last cell in the table above.
3. You can delete any elements of this cover page that you do not need for your document. For example, Copy Number is only required if this is a controlled document and you need to track each copy that you distribute.



## 文档控制

**更改记录**

4

| 日期 | 作者 | 版本 | 更改参考 |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

内容目录

[文档控制 2](#_Toc514363541)

[1. Spring Cloud Config概念 4](#_Toc514363542)

[2. 服务端详解 5](#_Toc514363543)

[2.1. 基础架构 5](#_Toc514363544)

[2.2. Git配置仓库 6](#_Toc514363545)

[2.3. 本地仓库 6](#_Toc514363546)

[2.4. 本地文件系统 6](#_Toc514363547)

[2.5. 属性覆盖 6](#_Toc514363548)

[2.6. 安全保护 7](#_Toc514363549)

[2.7. 高可用配置 7](#_Toc514363550)

[3. 客户端详解 8](#_Toc514363551)

[3.1. URI指定配置中心 8](#_Toc514363552)

[3.2. 服务化配置中心 8](#_Toc514363553)

[3.3. 失败快速响应与重试 8](#_Toc514363554)

[3.4. 动态刷新配置 9](#_Toc514363555)

[未结与已结问 10](#_Toc514363556)

[未结问题 10](#_Toc514363557)

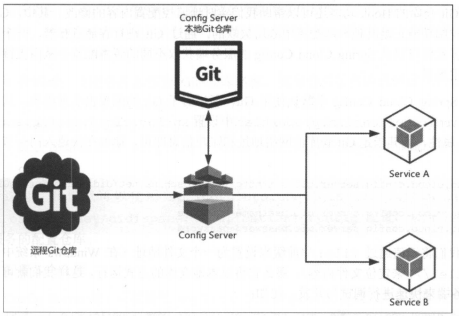
[已结问题 10](#_Toc514363558)

## Spring Cloud Config概念

Spring Cloud Config是Spring Cloud团队创建的一个全新项目，用来为分布式系统中的基础设施和服务应用提供集中化的外部配置支持，它分为客户端和服务段两部分。其中服务端也称为分布式配置中心，它是一个独立的微服务应用，用来链接配置仓库并为客户端提供获取配置信息、加密/解密信息等访问接口；而客户端则是微服务架构中的各个微服务应用或基础设施，他们通过指定的配置中心来管理应用资源与业务相关的配置内容，并在启动的时候从配置中心获取和加载配置信息，所以它除了适用于Spring构建的应用程序之外，也可以在任何其他语言运行的应用程序中使用。

## 服务端详解

### 基础架构



如图所示：

* 远程Git仓库：用来存储配置文件的地方。
* Config Server：分布式配置中心，指定了所要链接的Git仓库位置以及账户、密码等链接信息。
* 本地Git仓库：在Config Server的文件系统中，每次客户端请求获取配置信息时，Config Server从Git仓库中获取最新配置到本地，然后在本地Git仓库中读取并返回，当远程仓库无法获取时，直接将本地内容返回。
* Service A、Service B：具体的微服务应用。它们指定了ConfigServer的地址，从而实现从外部化获取应用自己要用的配置信息，这些应用在启动的时候，会向Server请求获取配置信息来进行加载。

客户端应用从配置管理中获取配置信息遵从下面的执行流程：

1. 应用启动时，根据bootstrap.properties中配置的应用名{application}、环境名{profile}、分支名{label}，向Config Server请求获取配置信息。
2. Config Server根据自己维护的Git仓库信息和客户端传递过来的配置定位信息去查找配置信息。
3. 通过git clone命令将找到的配置信息下载到Config Servier的文件系统中。
4. Config Server创建Spring的SpplicationContext实例，并从Git本地仓库中加载配置文件，最后将这些配置内容读取出来返回给客户端应用。
5. 客户端应用在获得外部配置文件后加载到客户端的ApplicationContext实例，该配置内容的优先级高于客户端的Jar包内部的配置内容，所以在Jar包中重复的内容将不再被加载。

Config Server巧妙的通过git clone将配置信息存于本地，起到了缓存的作用，即使当Git服务端无法访问的时候，依然可以取Config Server中的缓存内容进行使用。

### Git配置仓库

在Spring Cloud Config的服务器，对于配置仓库的默认实现采用了Git。Git非常适用于存储配置内容，它可以非常方便的使用各种第三方工具来对其内容进行管理更新和版本化，同时Git仓库的Hook功能还可以实时地监控配置内容的修改。其中，Git的自身的版本控制功能正是一些配置中心所欠缺的，通过Git进行存储意味着，一个应用的不同部署实现可以从Spring Cloud Config的服务端获取不同的版本配置，从而支持一些特殊的应用场景。

由于Spring Cloud Config中默认使用Git，所以对于Git的配置也非常简单，只需在Config Server的application.properties中设置spring.cloud.config.server.git.uri属性，为其指定Git仓库的网络地址和账户信息即可。

如果将改制通过file://前缀来设置为一个文件地址，那么它将以本地仓库的方式运行，这样就可以脱离Git服务端来快速开发了。

### 本地仓库

在使用了Git或SVN仓库之后，文件都会在Config Server 的本地文件系统中存储一份，这些文件默认会被存储于以config-repo为前缀的临时目录中，比如名为/tmp/config-repo-<随机数>的目录。由于其随机性以及临时目录的特性，可能会有一些不可预知的后果，为了避免将来可能出现的问题，最好的办法就是指定一个固定的位置来存储这些重要的信息。

### 本地文件系统

Spring Cloud Config也提供了一种不使用Git仓库或SVN仓库的存储方式，而是使用本地文件系统的存储方式来保存配置信息。实际方式也非常简单，我们只需要设置属性spring.profiles.active=native，Config Server会默认从应用的src/main/resource目录下搜索配置文件。如果需要指定搜索配置文件的路径，可以通过spring.cloud.config.server.native.searchLocations属性来指定具体的配置文件位置。

### 属性覆盖

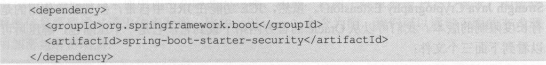
Config Server还有一个“属性覆盖”的特性，它可以让开发人员为所有的应用提供配置属性，直需要通过spring.cloud.config.server.overrides属性来设置键值对的参数，这些参数会以Map的方式加载到客户端的配置中。比如：



通过该属性配置的参数，不会被Spring Cloud的客户端修改，并且Spring Cloud客户端从Config Server中获取配置信息时，都会取得这些配置信息。利用该特性可以方便地为Spring Cloud应用配置一些共同属性或是默认属性。

### 安全保护

由于配置中心存储的内容比较敏感，做一定的安全处理是必需的。为配置中心实现安全保护的方法有很多，比如物理网络限制，OAuth2授权等。不过，由于我们的微服务应用和配置中心都构建于Spring Boot基础上，所以与Spring Security结合使用会更加方便。只需要在配置中心的pom.xml中加入spring-boot-starter-security依赖，不需要做任何其他改动就能实现对配置中心访问的安全保护。



### 高可用配置

当要将配置中心部署到生产环境中时，与服务注册中心一样，我们也希望它是一个高可用的应用。Spring Cloud Config实现服务端的高可用非常简单，主要有以下两种方式。

* 传统模式：不需要为这些服务端做任何额外的配置，只需要遵守一个配置规则，将所有的Config Server都指向同一个Git仓库，这样所有的配置内容就通过统一的共享文件系统来维护。而客户端在指定的Config Server位置时，只需要配置Config Server上层的负载均衡设备地址即可。
* 服务模式：将Config Server作为一个普通的微服务应用，纳入Eureka的服务治理体系中。这样的微服务应用就可以通过配置中心的服务名来获取配置信息，这种方式比起传统的实现模式来说更加有利于维护，因为对于服务端的负载均衡配置和客户端的配置中心指定都通过服务治理机制一并解决。

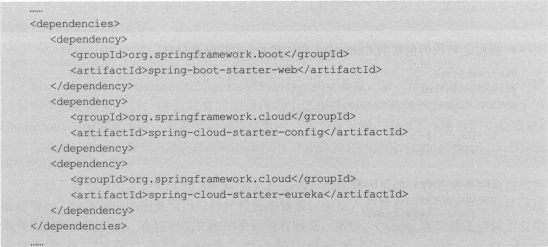
## 客户端详解

### URI指定配置中心

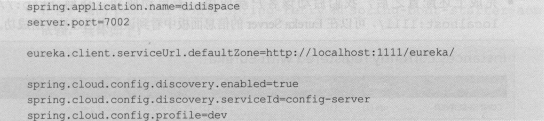
Spring Cloud Config的客户端在启动的时候，默认会从工程的classpath中加载配置信息并启动应用。只有我们配置spring.cloud.config.uri的时候，客户端应用才会尝试连接Spring Cloud Config的服务端来获取远程配置信息并初始化Spring环境配置，同时，将该参数配置在bootstrap.properties、环境变量或是其他优先级高于应用Jar包内的配置信息中，才能正确加载到远程配置。

### 服务化配置中心

config-client的pom.xml中新增spring-cloud-starter-eureka依赖，以实现客户端发现config-server服务，具体配置如下：



在bootstrap.properties中，按如下配置：

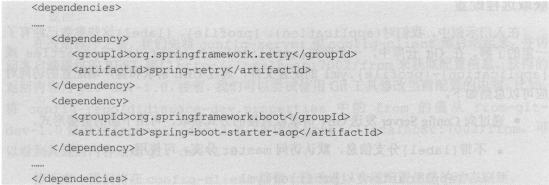


其中，通过eureka.client.serviceUrl.defaultZone参数指定服务注册中心，用于服务的注册与发现；再将spring.cloud.config.discovery.enabled参数设置为true,开启通过服务来访问Config Server的功能；最后利用spring.cloud.config.discovery.serviceId参数来指定Config Server住测的服务名。这里的spring.application.name和spring.cloud.config.profile如之前通过URI的方式访问的时候一样，用户定位Git中的资源。

### 失败快速响应与重试

Spring Cloud Config的客户端会预先加载很多其他信息，然后在开始链接Config Server 进行属性的注入。当构建的应用 较为复杂的时候，可能在连接Config Server之前花费较长的启动时间，而在一些特殊场景下，我们又希望可以快速知道当前应用是否能顺利地从Config Server获取到配置信息，这对在初期建调试环境时，可以减少很多等待启动的时间。要实现客户端优先判断Config Server获取是否正常，并快速响应失败内容。Config客户端还提供了自动重试的功能，在开启重试功能前，先确保已经配置了spring.cloud.config.failFast=true，再进行下面的操作：

在客户端的pom.xml中增加spring-retry和spring-boot-starter-aop依赖

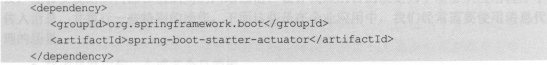


不需要再做其他任何配置。客户端在连接Config Server失败之后，会继续尝试，直到第6次失败后，才返回错误信息。

### 动态刷新配置

通过改造配置内容来实现实时更新。在改造程序之前，我们先将config-server和config-client都启动起来，并访问客户端提供的REST接口<http://localhost:7002/form来获取配置的内容，将config-repo/didispace-dev.properties中的form>的值从from=git-dev-1.0修改为from=git-dev-2.0,再访问<http://localhost:7002/form>，可以看到其返回内容还是git-dev-1.0。

在config-client的pom.xml中新增spring-boot-starter-actuator监控模块。



重新启动config-client，访问一次<http://localhost:7002/form>，可以看到当前的配置信息。

修改Git仓库config-repo/didispace-dev.propertoes文件中form的值。

通过POST请求送到http：//localhost：7002/refresh，可以看到返回的结果。

## 未结与已结问

### 未结问题

| 序号 | 问题 | 解决方案 | 负责人 | 目标日期 | 实际日期 |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 已结问题

| 序号 | 问题 | 解决方案 | 负责人 | 目标日期 | 实际日期 |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |