
分布式配置中心

课程讲义

主讲：Reythor 雷

2021

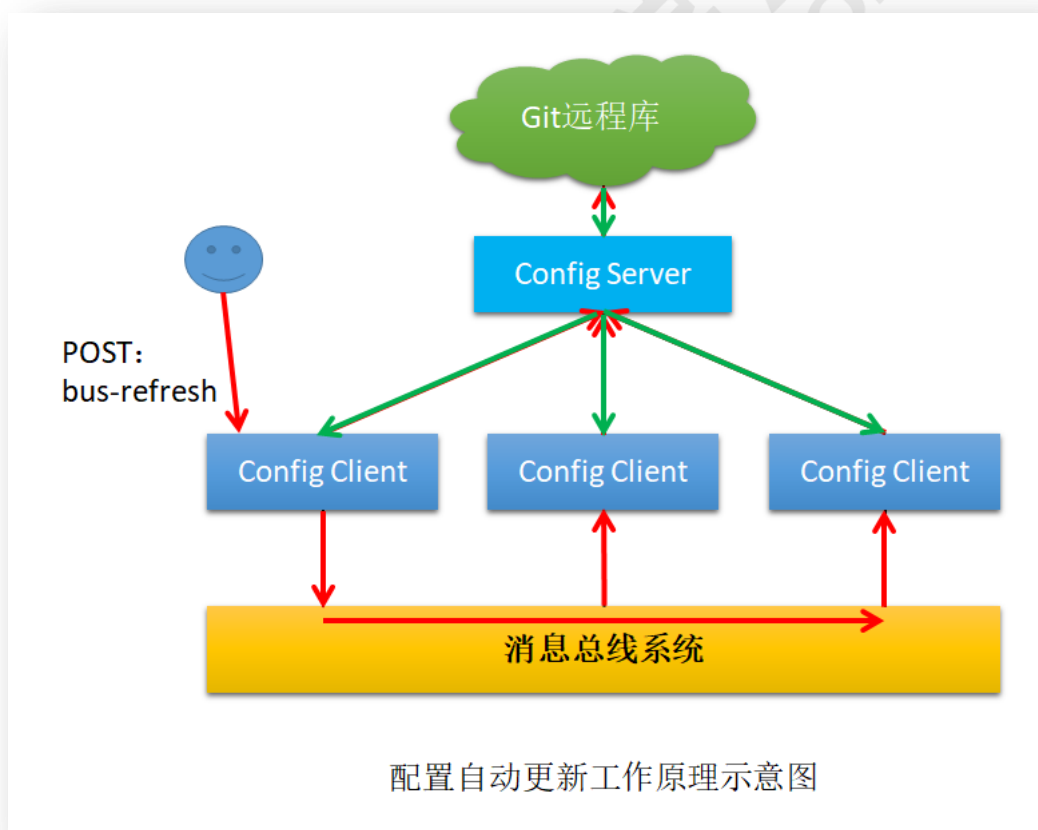
分布式配置中心

1.1 配置中心功能

集群中每一台主机的配置文件都是相同的，对配置文件的更新维护就成为了一个棘手的问题，Nacos 是可以对 Spring Cloud 中各个微服务配置文件进行统一维护管理的配置中心。

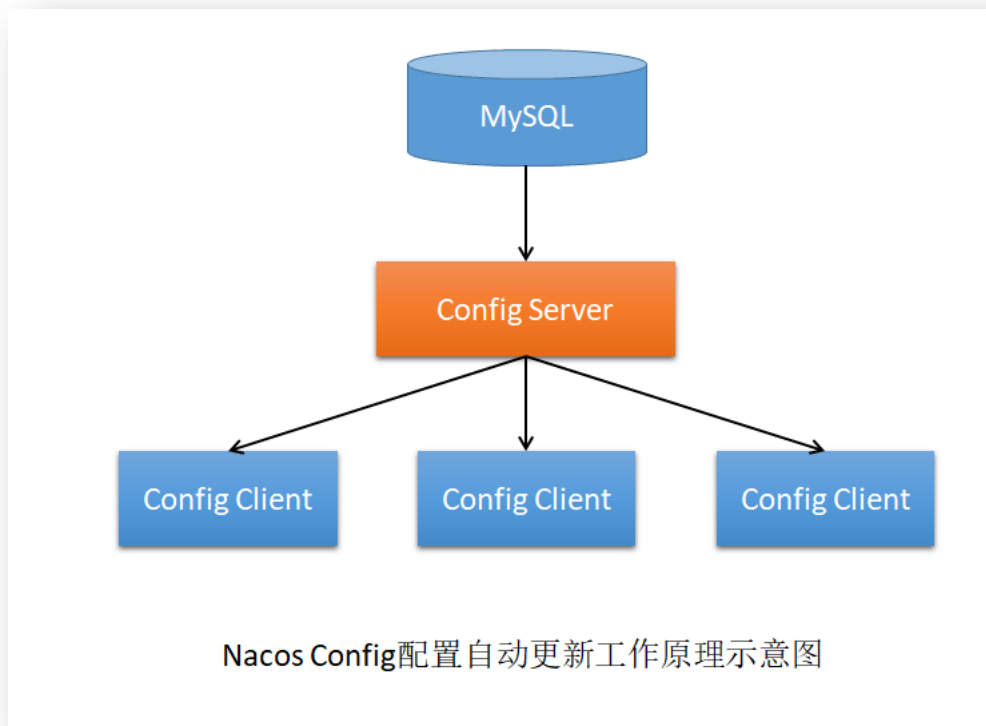
1.2 常见配置中心工作原理

1.2.1 Spring Cloud Config

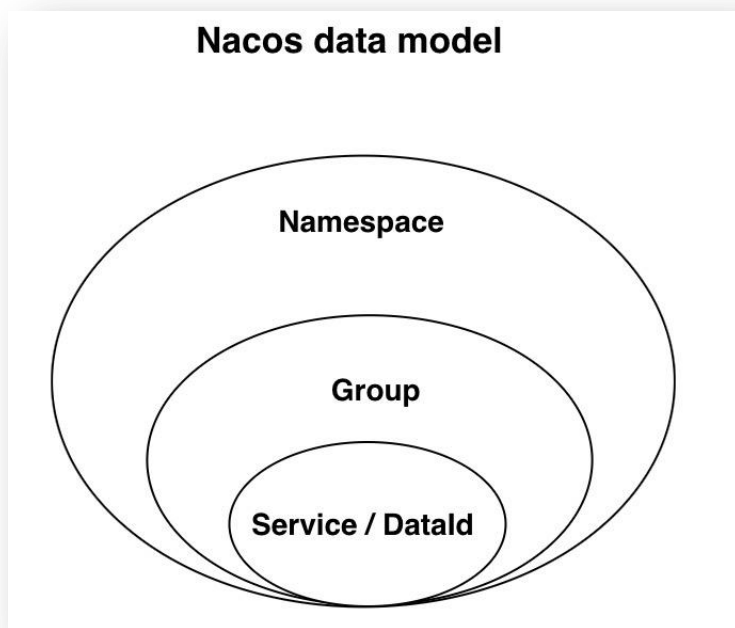


1.2.2 Nacos Config

(1) 系统架构



(2) 数据模型

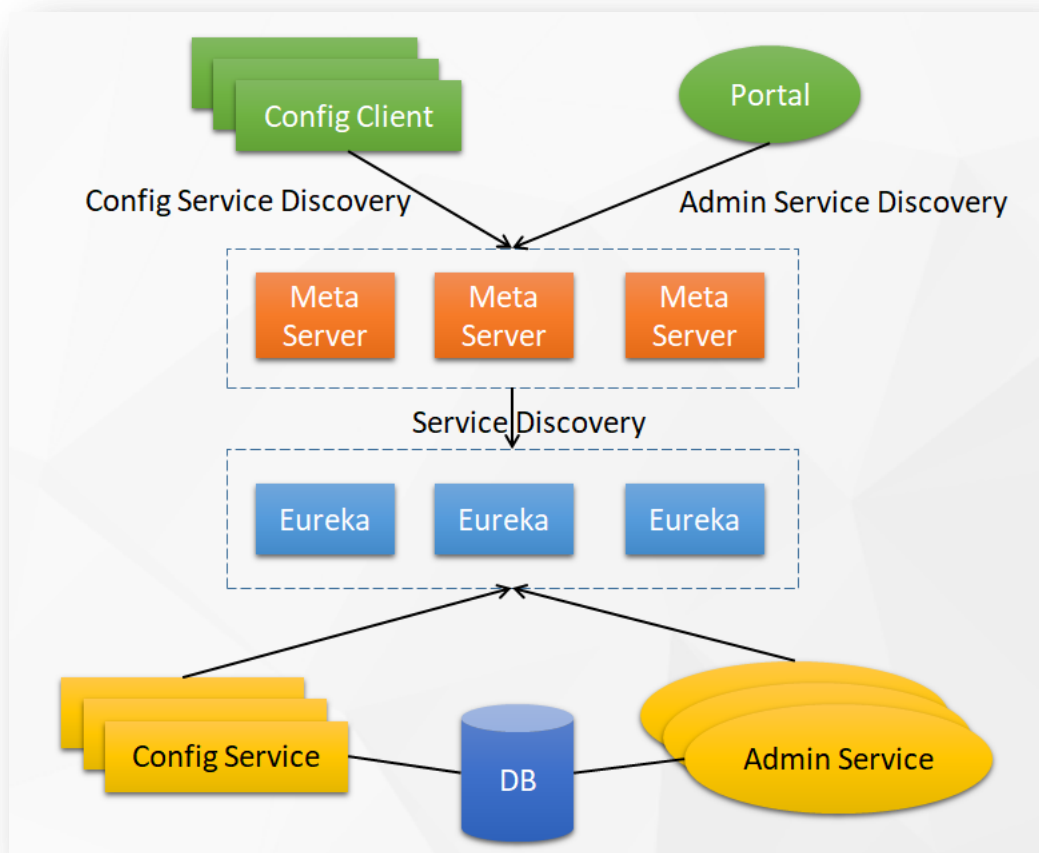


在后面阅读 Nacos Config 源码过程中会看到一个概念：**tenant**。中文意思是租户，房客。其实，**tenant** 就是 **namespace**，是 `bootstrap.yml` 文件属性 `spring.cloud.nacos.config` 中指定的 **namespace**。在代码中为了区分 `spring.cloud.nacos.discovery` 中指定的 **namespace**，所以起了 **tenant** 这个名字。

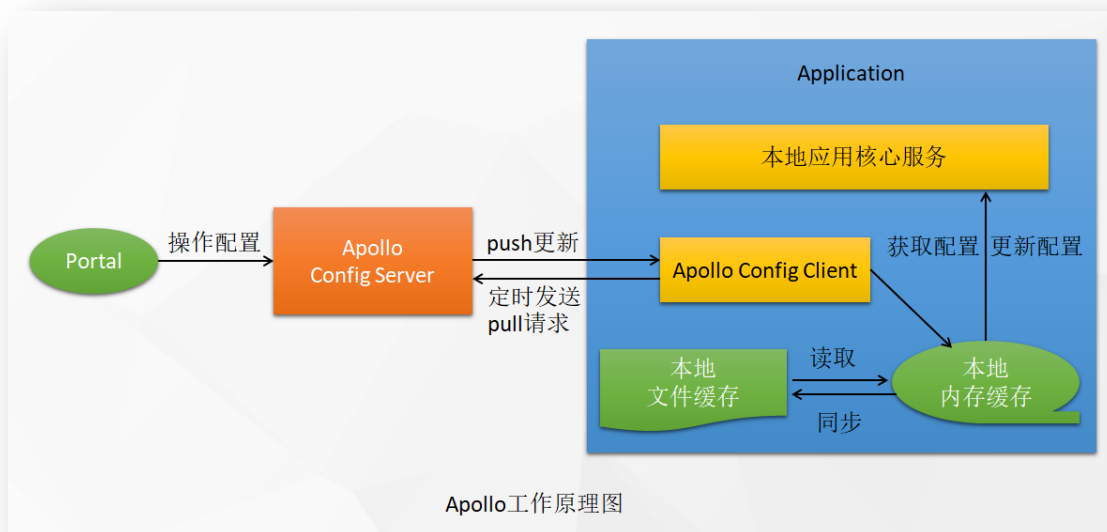
1.2.3 Apollo

Apollo 是由携程推出的一款开源的分布式配置中心。

(1) 系统架构



(2) 工作原理



1.2.4 对比

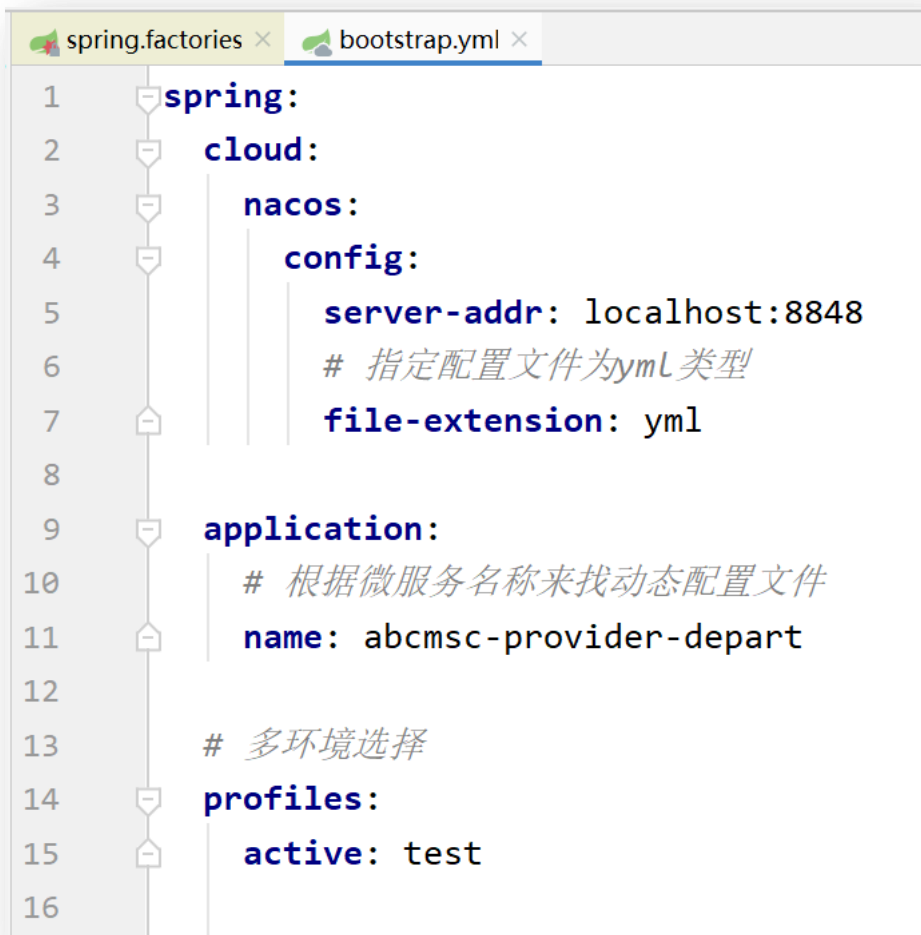
- **系统架构复杂度：**Nacos Config 最为简单，无需消息总线系统，无需 Eureka 等。而 Apollo 与 Spring Cloud Config 系统搭建成本及复杂度较 Nacos 要高很多。
- **羊群效应：**对于 Spring Cloud Config，Config Client 需要提交配置更新请求。当微服务系统很庞大时，任何一个 Config Client 的更新请求的提交，都会引发所有“Bus 在线 Config Client”的配置更新请求的提交，即会引发羊群效应。这将会导致 Config Client 的效率下降，导致整个系统的效率下降。而 Nacos Config 与 Apollo 则是“定点更新”，谁的配置更新了向谁推送。
- **自动感知配置更新：**Spring Cloud Config 是 Config Client 不提交请求，其是无法感知配置更新的。但 Nacos 与 Apollo 则是，当 Config Server 中的配置文件发生了变更，Config Client 会自动感知到这个变更，无需 Config Client 端的用户做任何操作。
- **配置文件类型：**Nacos Config 与 Spring Cloud Config 配置文件支持比较多的类型，支持 yaml、text、json、xml、html、Properties 等，但 Apollo 只支持 xml、text、Properties 类型，不支持 yaml。

1.3 配置文件的加载

Nacos Config Client 在启动时是如何将远程配置中心 Nacos Config Server 中的配置文件加载到本地的呢？这里要解决这个问题。

1.3.1 源码基础

(1) 基本配置说明



```

1  spring:
2    cloud:
3      nacos:
4        config:
5          server-addr: localhost:8848
6          # 指定配置文件为yml类型
7          file-extension: yml
8
9    application:
10     # 根据微服务名称来找动态配置文件
11     name: abcmvc-provider-depart
12
13     # 多环境选择
14   profiles:
15     active: test
16
  
```

nacos config client 要加载的配置文件有三种：

- 自身配置文件
- 共享配置文件
- 扩展配置文件

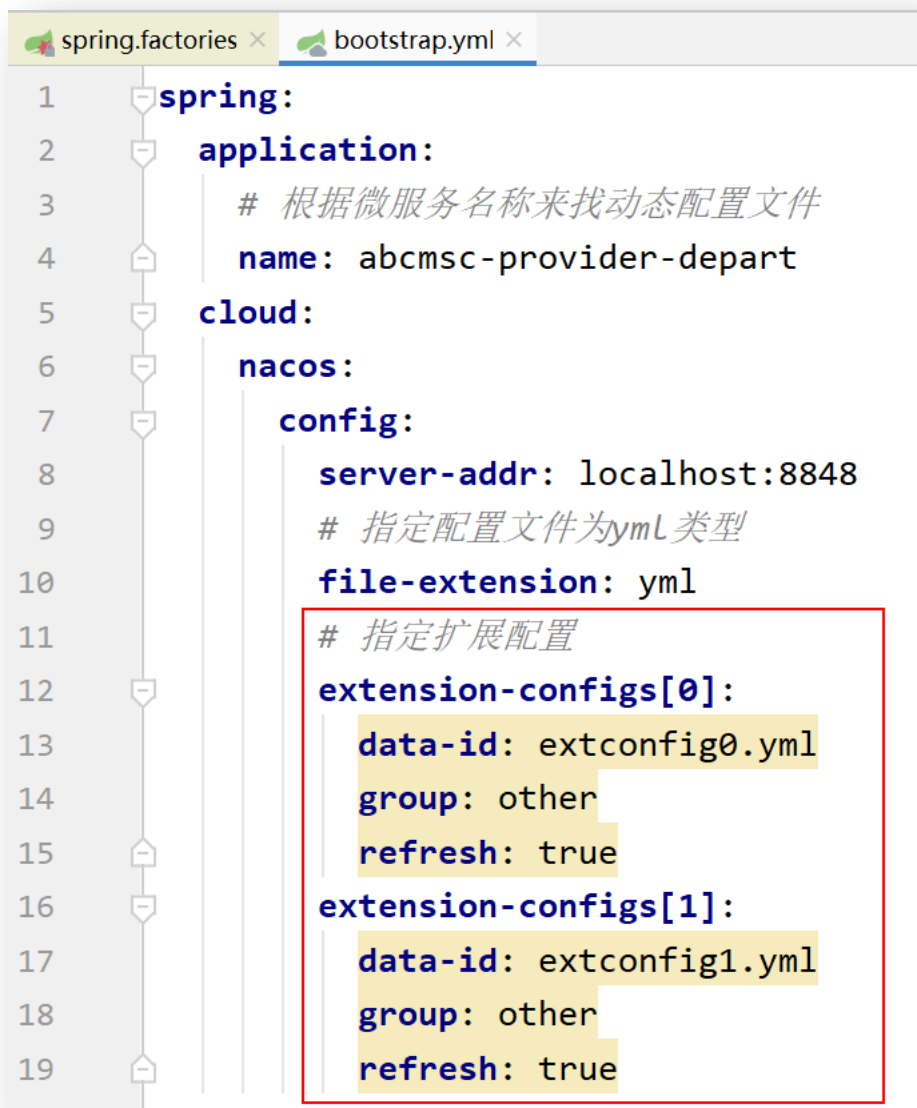
(2) 共享配置说明

```

spring:
  application:
    # 根据微服务名称来找动态配置文件
    name: abcm-sc-provider-depart
  cloud:
    nacos:
      config:
        server-addr: localhost:8848
        # 指定配置文件为yaml类型
        file-extension: yaml
        # 指定共享配置
        # shared-configs: shareconfig0.yaml, shareconfig1.yaml
        # 以上共享配置等价于如下共享配置
        shared-configs[0]:
          data-id: shareconfig0.yaml
          refresh: true
        shared-configs[1]:
          data-id: shareconfig1.yaml
          refresh: true

```


(3) 扩展配置说明



```

1  spring:
2    application:
3      # 根据微服务名称来找动态配置文件
4      name: abcmvc-provider-depart
5    cloud:
6      nacos:
7        config:
8          server-addr: localhost:8848
9          # 指定配置文件为yaml类型
10         file-extension: yaml
11         # 指定扩展配置
12         extension-configs[0]:
13           data-id: extconfig0.yaml
14           group: other
15           refresh: true
16         extension-configs[1]:
17           data-id: extconfig1.yaml
18           group: other
19           refresh: true

```

(4) 加载顺序说明

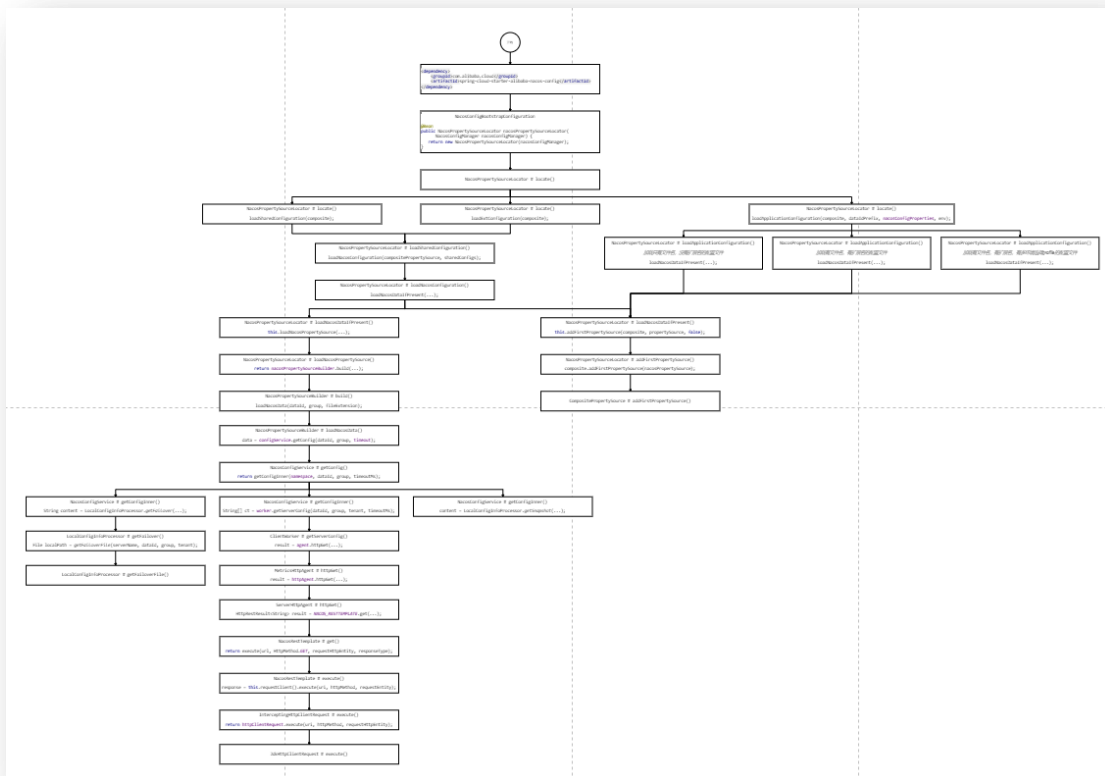
以上三类配置文件的加载顺序为，共享配置 -> 扩展配置 -> 当前应用自身配置，如果存在相同属性配置了不同属性值，则后加载的会将先加载的给覆盖。即优先级为：共享配置 < 扩展配置 < 应用自身配置。

对于每种配置文件的加载过程，又存在三种可用选择：在应用本地存在同名配置，远程配置中心存在同名配置，本地磁盘快照 snapshot 中存在同名配置。这三种配置的优先级为：本地配置 > 远程配置 > 快照配置。只要前面的加载到了，后面的就不再加载。

若要在应用本地存放同名配置，则需要存放到当前用户主目录下的 `nacos\config\fixed-localhost_8848_nacos\data\config-data\{groupId}` 目录中。

若开启了配置的快照功能，则默认会将快照记录在当前用户主目录下的 `nacos\config\fixed-localhost_8848_nacos\snapshot\{groupid}` 目录中。

1.3.2 源码跟踪



1.4 配置文件的动态更新

当远程 Nacos Config Server 中的配置信息发生了变更，Nacos Config Client 是如何感知到的呢？这里就来解决这个问题。

1.4.1 长轮询模型

Nacos Config Server 中配置数据的变更，Nacos Config Client 是如何知道的呢？Nacos Config Server 采用了长轮询模型实现的变更通知。

一般情况下 Server 端数据的变更若要使 Client 感知到，可以选择两种模型：

- **Push 模型**：当 Server 端的数据发生了变更，其会主动将更新推送给 Client。Push 模型适合于 Client 数量不多，且 Server 端数据变化比较频繁的场景。其实时性较好，但其需要维护长连接，占用系统资源。
- **Pull 模型**：需要 Client 定时查看 Server 端数据是否更新。其实时性不好，且可能会产生

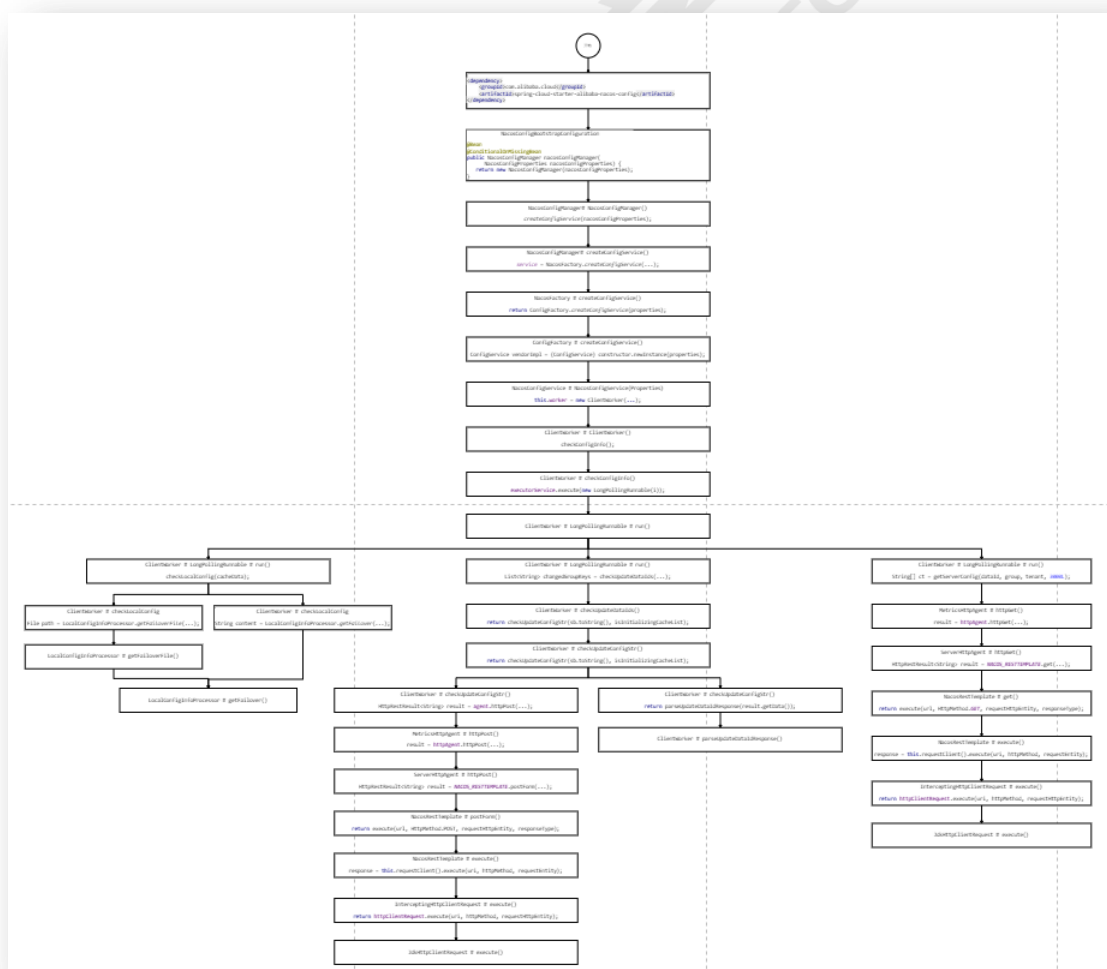
数据更新的丢失。

长轮询模型整合了 Push 与 Pull 模型的优势。Client 仍定时发起 Pull 请求，查看 Server 端数据是否更新。若发生了更新，则 Server 立即将更新数据以响应的形式发送给 Client 端；若没有发生更新，Server 端不会发送任何信息，但其会临时性的保持住这个连接一段时间。若在此时间段内，Server 端数据发生了变更，这个变更就会触发 Server 向 Client 发送变更结果。这次发送的执行，就是因为长连接的存在。若此期间仍未发生变更，则放弃这个连接。等待着下一次 Client 的 Pull 请求。

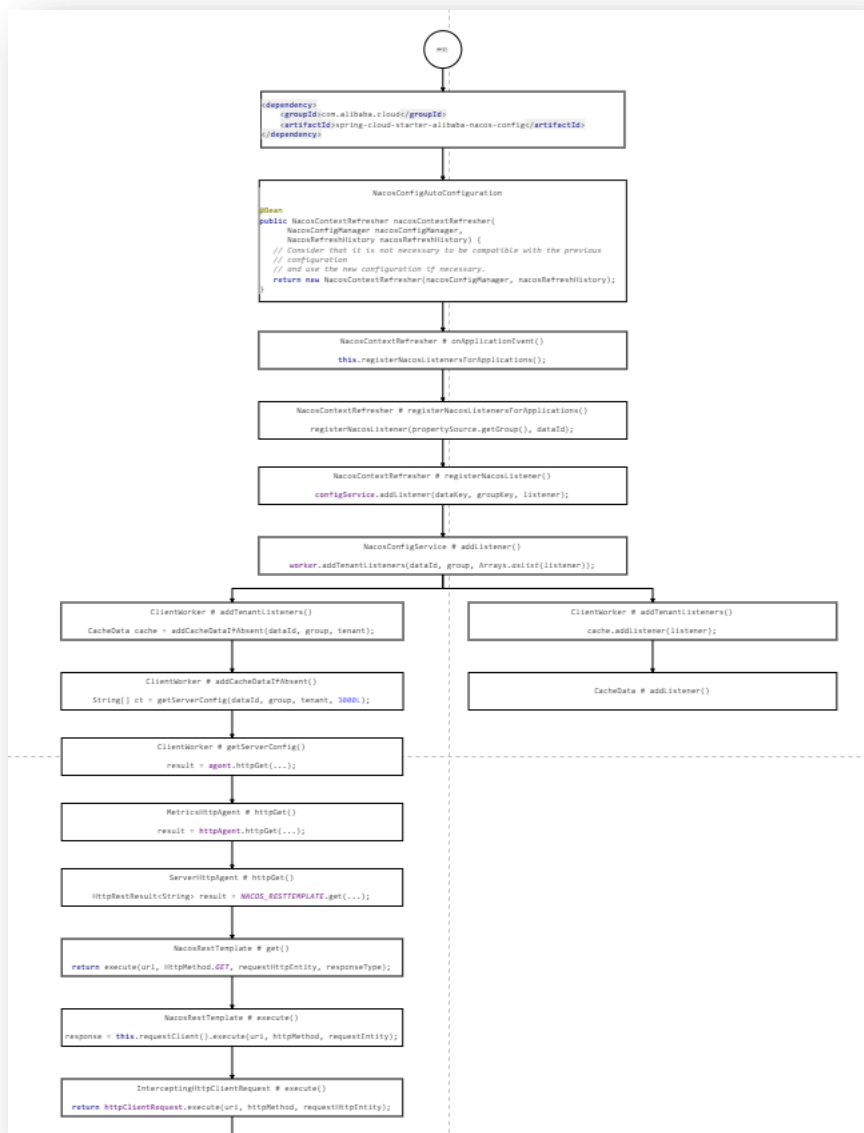
长轮询模型，是 Push 与 Pull 模型的整合，既减少了 Push 模型中长连接的被长时间维护的时间，又降低了 Pull 模型实时性较差的问题。

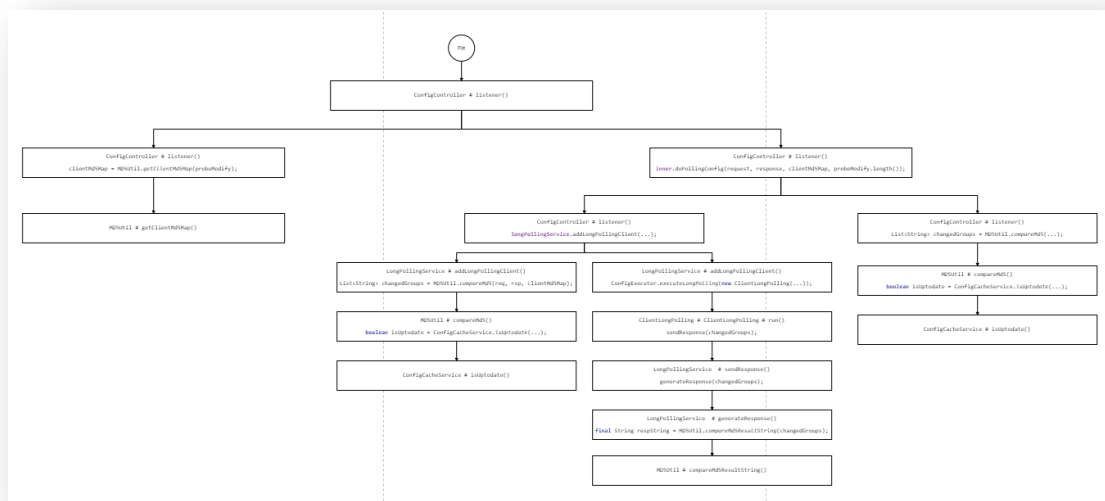
1.4.2 config client 定时发出更新检测

NacosConfigService -> ClientWorker -> 启动一个定时任务
cacheMap CacheData



1.4.3 config client 将更新同步到应用实例





1.4.5 config server 感知配置变更后通知 client

