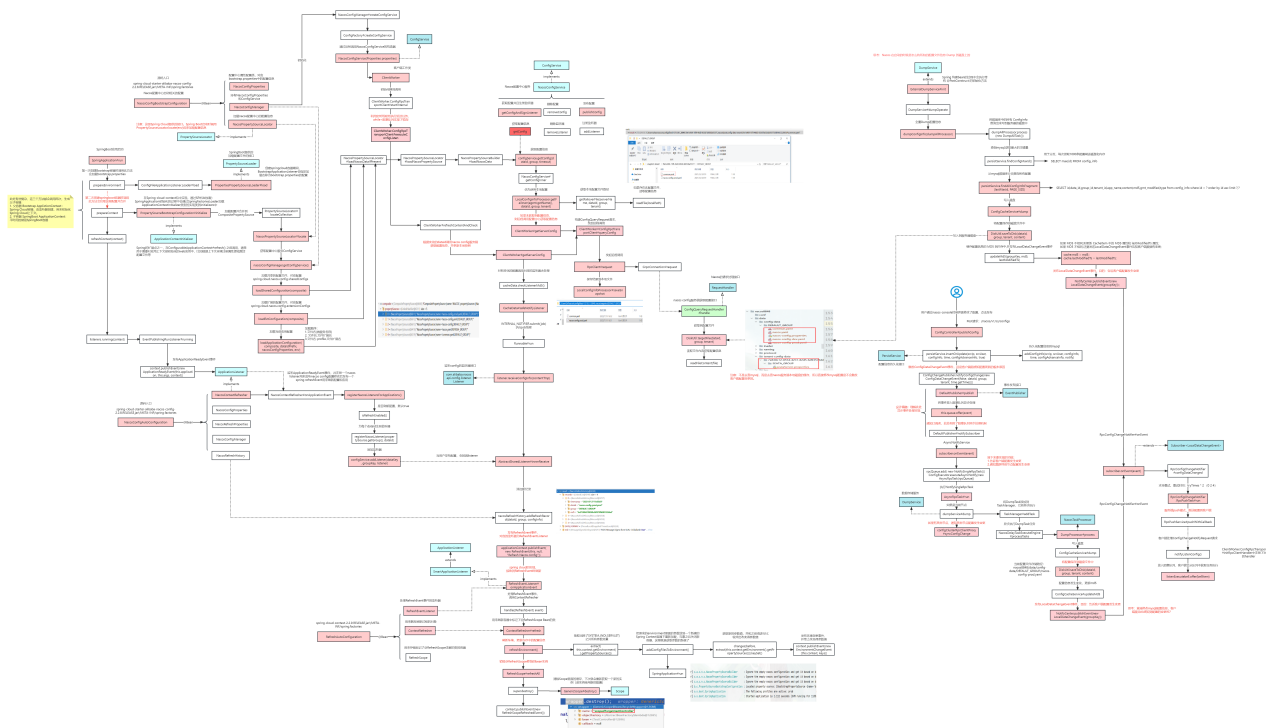
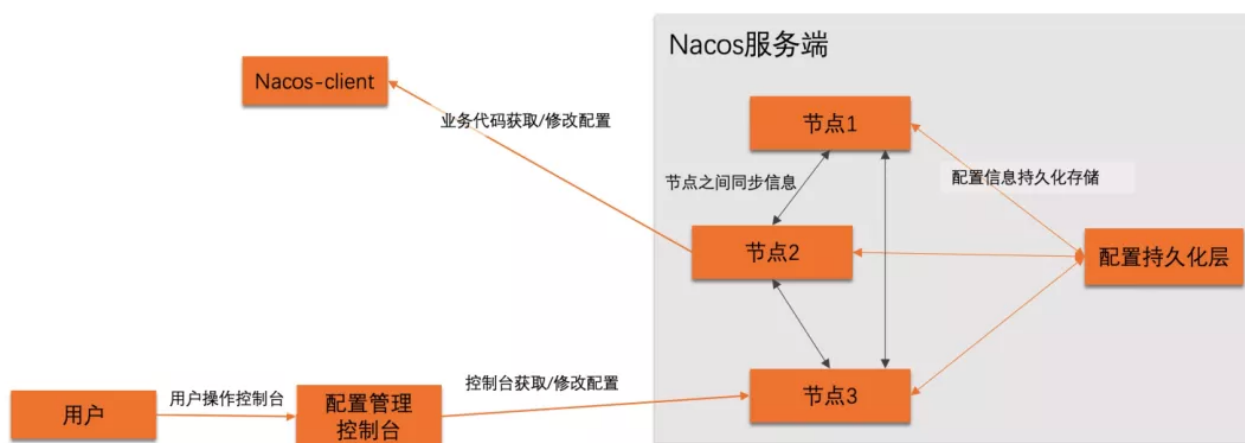


1. Nacos配置中心源码分析

Nacos2.1.0源码分析图: <https://www.processon.com/view/link/62d678c31e08531cf8db16ef>












1.1 配置中心架构



1.2 Config Client源码分析

配置中心核心接口ConfigService

I ConfigService

 01	getConfig(String, String, long)	String
 01	getConfigAndSignListener(String, String, long, Listener)	String
 01	addListener(String, String, Listener)	void
 01	publishConfig(String, String, String)	boolean
 01	publishConfig(String, String, String, String)	boolean
 01	removeConfig(String, String)	boolean
 01	removeListener(String, String, Listener)	void
 01	getServerStatus()	String
 01	shutDown()	void

```
1 public class ConfigServerDemo {
2
3     public static void main(String[] args) throws NacosException, InterruptedException
4     {
5         String serverAddr = "localhost";
6         String dataId = "nacos-config-demo.yaml";
7         String group = "DEFAULT_GROUP";
8         Properties properties = new Properties();
9         properties.put(PropertyKeyConst.SERVER_ADDR, serverAddr);
10        //获取配置服务
11        ConfigService configService = NacosFactory.createConfigService(properties);
12        //获取配置
13        String content = configService.getConfig(dataId, group, 5000);
14        System.out.println(content);
15        //注册监听器
16        configService.addListener(dataId, group, new Listener() {
17            @Override
18            public void receiveConfigInfo(String configInfo) {
19                System.out.println("===recieve:" + configInfo);
20            }
21
22            @Override
23            public Executor getExecutor() {
24                return null;
25            }
26        });
27    }
28 }
```

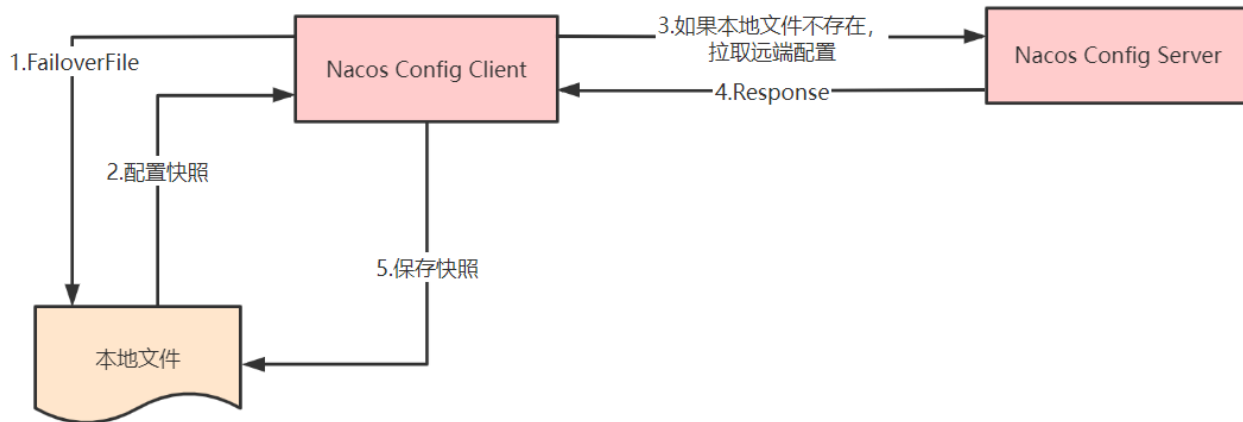
```

24         }
25     });
26
27     //发布配置
28     //boolean isPublishOk = configService.publishConfig(dataId, group, "content");
29     //System.out.println(isPublishOk);
30     //发送properties格式
31     configService.publishConfig(dataId,group,"common.age=30",
ConfigType.PROPERTIES.getType());
32
33     Thread.sleep(3000);
34     content = configService.getConfig(dataId, group, 5000);
35     System.out.println(content);
36
37     //    boolean isRemoveOk = configService.removeConfig(dataId, group);
38     //    System.out.println(isRemoveOk);
39     //    Thread.sleep(3000);
40
41     //    content = configService.getConfig(dataId, group, 5000);
42     //    System.out.println(content);
43     //    Thread.sleep(300000);
44
45     }
46 }

```

获取配置

获取配置的主要方法是 `NacosConfigService` 类的 `getConfig` 方法，通常情况下该方法直接从本地文件中取得配置的值，如果本地文件不存在或者内容为空，则再通过grpc从远端拉取配置，并保存到本地快照中。



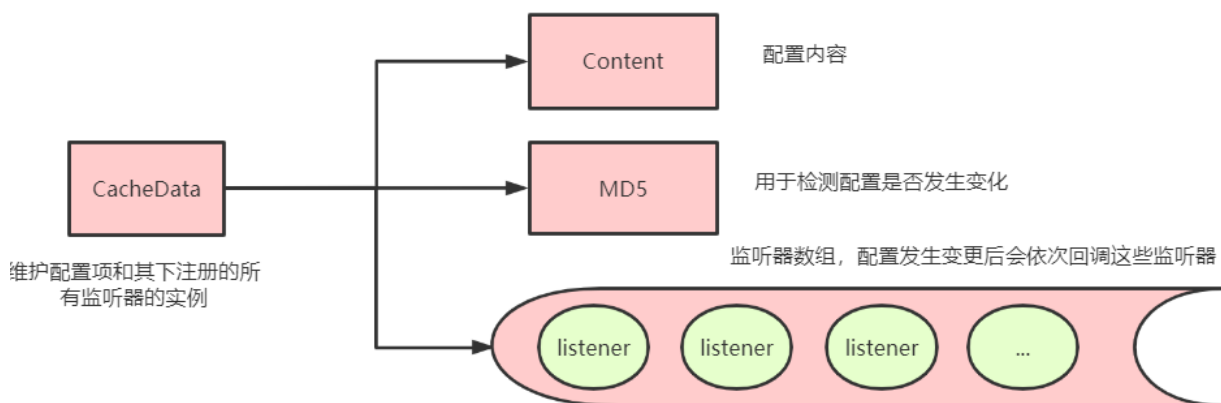
注册监听器

配置中心客户端会通过对配置项注册监听器达到在配置项变更的时候执行回调的功能。

```

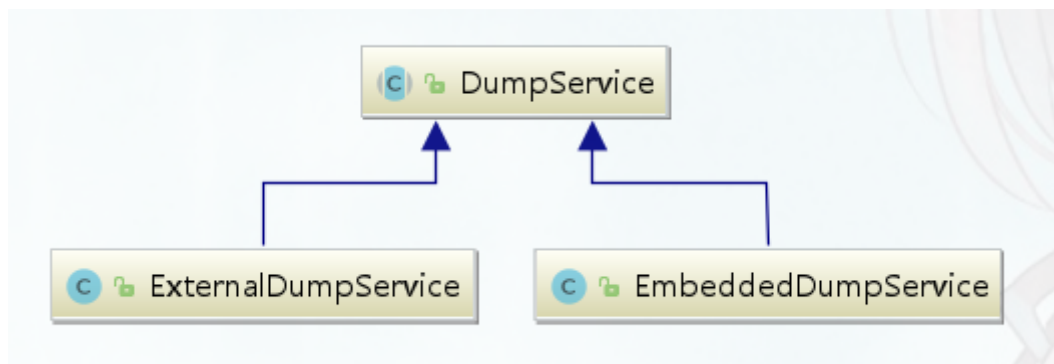
1 ConfigService#getConfigAndSignListener
2 ConfigService#addListener
  
```

Nacos 可以通过以上方式注册监听器，它们内部的实现均是调用 ClientWorker 类的 addCacheDataIfAbsent。其中 **CacheData** 是一个维护配置项和其下注册的所有监听器的实例，所有的 CacheData 都保存在 ClientWorker 类中的原子 cacheMap 中，其内部的核心成员有：



1.3 Config Server源码分析

配置dump



服务端启动时就会依赖 `DumpService` 的 `init` 方法，从数据库中 load 配置存储在本地磁盘上，并将一些重要的元信息例如 MD5 值缓存在内存中。服务端会根据心跳文件中保存的最后一次心跳时间，来判断到底是从数据库 dump 全量配置数据还是部分增量配置数据（如果机器上次心跳间隔是 6h 以内的话）。

全量 dump 当然先清空磁盘缓存，然后根据主键 ID 每次捞取一千条配置刷进磁盘和内存。增量 dump 就是捞取最近六小时的新增配置（包括更新的和删除的），先按照这批数据刷新一遍内存和文件，再根据内存里所有的数据全量去比对一遍数据库，如果有改变的再同步一次，相比于全量 dump 的话会减少一定的数据库 IO 和磁盘 IO 次数。

配置发布

发布配置的代码位于 `ConfigController#publishConfig` 中。集群部署，请求一开始也只会打到一台机器，这台机器将配置插入 MySQL 中进行持久化。服务端并不是针对每次配置查询都去访问 MySQL，而是会依赖 dump 功能在本地文件中将配置缓存起来。因此当单台机器保存完毕配置之后，需要通知其他机器刷新内存和本地磁盘中的文件内容，因此它会发布一个名为 `ConfigDataChangeEvent` 的事件，这个事件会通过 gRPC 调用通知所有集群节点（包括自身），触发本地文件和内存的刷新。

