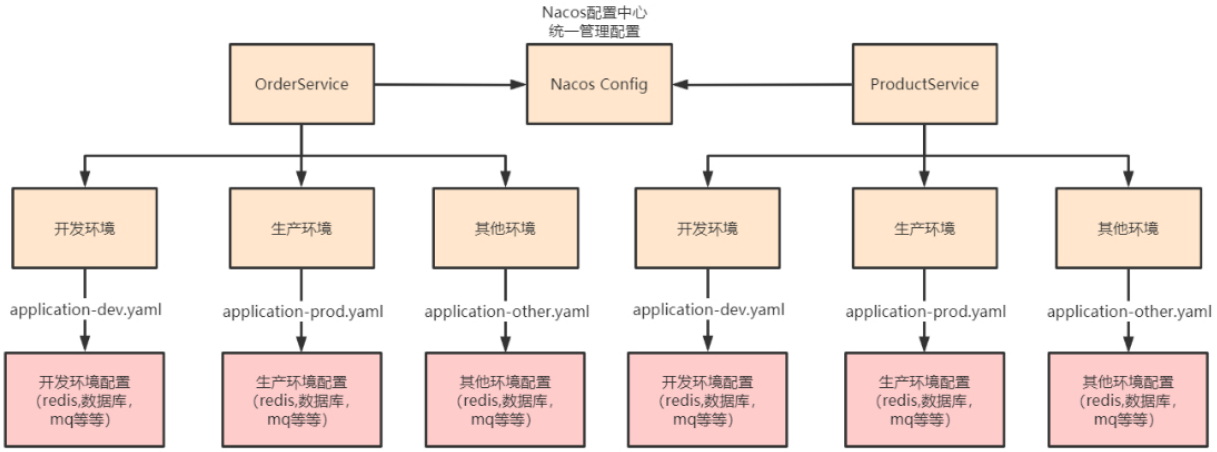
1. Nacos配置中心使用

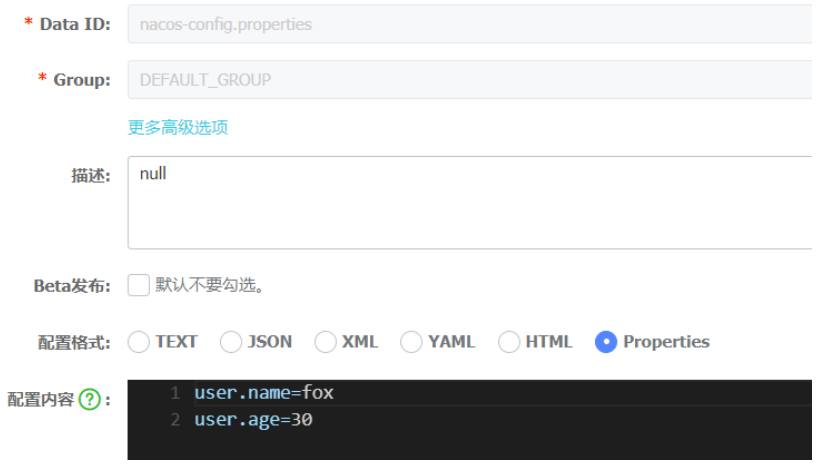
官方文档： https://github.com/alibaba/spring-cloud-alibaba/wiki/Nacos-config

Nacos 提供用于存储配置和其他元数据的 key/value 存储，为分布式系统中的外部化配置提供服务器端和客户端支持。使用 Spring Cloud Alibaba Nacos Config，您可以在 Nacos Server 集中管理你 Spring Cloud 应用的外部属性配置。



1.1 快速开始

准备配置，nacos server中新建nacos-config.properties



1.2 搭建nacos-config服务

通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-config 实现配置的动态变更

1）引入依赖

```
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

2）添加bootstrap.properties

```
spring.application.name=nacos-config
# 配置中心地址
spring.cloud.nacos.config.server-addr=127.0.0.1:8848

# dataid 为 yaml 的文件扩展名配置方式
```

```
# `${spring.application.name}.${file-extension:properties}`
spring.cloud.nacos.config.file-extension=yaml
#profile粒度的配置 `${spring.application.name}-${profile}.${file-extension:properties}`
spring.profiles.active=prod
```

3) 启动服务，测试微服务是否使用配置中心的配置

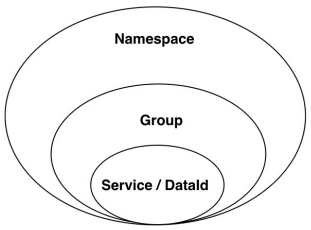
```
@SpringBootApplication
public class NacosConfigApplication {

 public static void main(String[] args) {
 ConfigurableApplicationContext applicationContext = SpringApplication.run(NacosConfigApplication.class, args);
 String userName = applicationContext.getEnvironment().getProperty("common.name");
 String userAge = applicationContext.getEnvironment().getProperty("common.age");
 System.out.println("common name :"+userName+"; age: "+userAge);
 }
}
```

```
2020-07-31 15:33:43.474  WARN 45352 --- [           main] c.a.c.n.c.NacosPropertySourceBuilder     : Ignore the empty
2020-07-31 15:33:43.479  INFO 45352 --- [           main] c.a.c.n.c.NacosPropertySourceBuilder     : Loading nacos da
user.age=30
2020-07-31 15:33:43.481  INFO 45352 --- [           main] b.c.PropertySourceBootstrapConfiguration : Located property
2020-07-31 15:33:43.484  INFO 45352 --- [           main] bat.ke.qq.com.NacosConfigApplication     : No active profil
2020-07-31 15:33:43.634  INFO 45352 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=6
2020-07-31 15:33:43.638  INFO 45352 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'org.spring
2020-07-31 15:33:45.202  INFO 45352 --- [           main] o.s.cloud.commons.util.InetUtils         : Cannot determine
2020-07-31 15:33:45.243  INFO 45352 --- [           main] bat.ke.qq.com.NacosConfigApplication     : Started NacosCon
user name :fox; age: 30
```

1.3 Config相关配置

Nacos 数据模型 Key 由三元组唯一确定, Namespace默认是空串，公共命名空间（public），分组默认是DEFAULT_GROUP

Nacos data model



- **支持配置的动态更新**

```
@SpringBootApplication
public class NacosConfigApplication {

 public static void main(String[] args) throws InterruptedException {
 ConfigurableApplicationContext applicationContext = SpringApplication.run(NacosConfigApplication.class, args);

 while(true) {
 //当动态配置刷新时，会更新到 Enviroment中，因此这里每隔一秒中从Enviroment中获取配置
 String userName = applicationContext.getEnvironment().getProperty("common.name");
  String userAge = applicationContext.getEnvironment().getProperty("common.age");
  System.err.println("common name :" + userName + "; age: " + userAge);
  TimeUnit.SECONDS.sleep(1);
  }

  }

}
```

- **支持profile粒度的配置**

spring-cloud-starter-alibaba-nacos-config 在加载配置的时候，不仅仅加载了以 dataid 为 `${spring.application.name}.${file-extension:properties}` 为前缀的基础配置，还加载了dataid为 `${spring.application.name}-${profile}.${file-extension:properties}` 的基础配置。在日常开发中如果遇到多套环境下的不同配置，可以通过Spring 提供的 `${spring.profiles.active}` 这个配置项来配置。

```
spring.profiles.active=dev
```

- **支持自定义 namespace 的配置**

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的配置的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。

在没有明确指定 `${spring.cloud.nacos.config.namespace}` 配置的情况下， 默认使用的是 Nacos 上 Public 这个 namespace。如果需要使用自定义的命名空间，可以通过以下配置来实现：

```
spring.cloud.nacos.config.namespace=71bb9785-231f-4eca-b4dc-6be446e12ff8
```

- **支持自定义 Group 的配置**

Group是组织配置的维度之一。通过一个有意义的字符串（如 Buy 或 Trade ）对配置集进行分组，从而区分 Data ID 相同的配置集。当您在 Nacos 上创建一个配置时，如果未填写配置分组的名称，则配置分组的名称默认采用 DEFAULT_GROUP 。配置分组的常见场景：不同的应用或组件使用了相同的配置类型，如 database_url 配置和 MQ_topic 配置。

在没有明确指定 `${spring.cloud.nacos.config.group}` 配置的情况下，默认是DEFAULT_GROUP 。如果需要自定义自己的 Group，可以通过以下配置来实现：

```
spring.cloud.nacos.config.group=DEVELOP_GROUP
```

- **支持自定义扩展的 Data Id 配置**

Data ID 是组织划分配置的维度之一。Data ID 通常用于组织划分系统的配置集。一个系统或者应用可以包含多个配置集，每个配置集都可以被一个有意义的名称标识。Data ID 通常采用类 Java 包（如 com.taobao.tc.refund.log.level）的命名规则保证全局唯一性。此命名规则非强制。

通过自定义扩展的 Data Id 配置，既可以解决多个应用间配置共享的问题，又可以支持一个应用有多个配置文件。

```
# 自定义 Data Id 的配置
#不同工程的通用配置 支持共享的 DataId
spring.cloud.nacos.config.sharedConfigs[0].data-id= common.yaml
spring.cloud.nacos.config.sharedConfigs[0].group=REFRESH_GROUP
spring.cloud.nacos.config.sharedConfigs[0].refresh=true

# config external configuration
# 支持一个应用多个 DataId 的配置
spring.cloud.nacos.config.extensionConfigs[0].data-id=ext-config-common01.properties
spring.cloud.nacos.config.extensionConfigs[0].group=REFRESH_GROUP
spring.cloud.nacos.config.extensionConfigs[0].refresh=true

spring.cloud.nacos.config.extensionConfigs[1].data-id=ext-config-common02.properties
spring.cloud.nacos.config.extensionConfigs[1].group=REFRESH_GROUP
spring.cloud.nacos.config.extensionConfigs[1].refresh=true
```

1.4 配置的优先级

Spring Cloud Alibaba Nacos Config 目前提供了三种配置能力从 Nacos 拉取相关的配置。

- A: 通过 `spring.cloud.nacos.config.shared-configs` 支持多个共享 Data Id 的配置
- B: 通过 `spring.cloud.nacos.config.ext-config[n].data-id` 的方式支持多个扩展 Data Id 的配置
- C: 通过内部相关规则(应用名、应用名+ Profile)自动生成相关的 Data Id 配置

当三种方式共同使用时，他们的一个优先级关系是:A < B < C

优先级从高到低：

1) nacos-config-product.yaml 精准配置

2) nacos-config.yaml 同工程不同环境的通用配置

3) ext-config: 不同工程 扩展配置

4) shared-dataids 不同工程通用配置: common2.yml > common1.yml

1.5 @RefreshScope

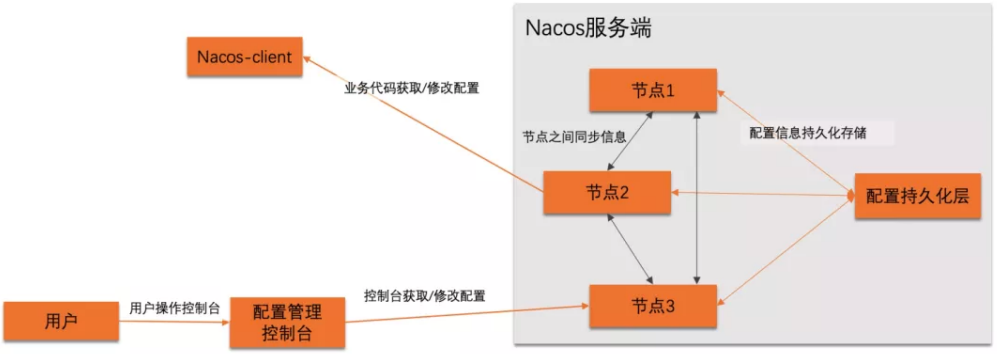
@Value注解可以获取到配置中心的值，但是无法动态感知修改后的值，需要利用@RefreshScope注解

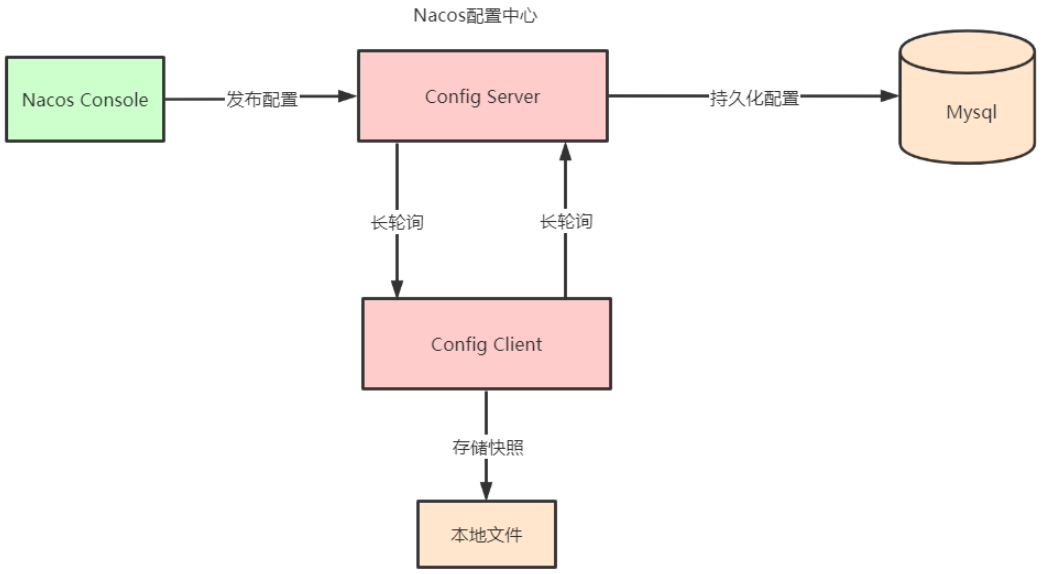
```java
@RestController
@RefreshScope
public class TestController {

 @Value("${common.age}")
 private String age;

 @GetMapping("/common")
 public String hello() {
  return age;
  }

 }
```

2. Nacos配置中心源码分析

https://www.processon.com/view/link/603f3d2fe401fd641adb51f1

2.1 配置中心的架构





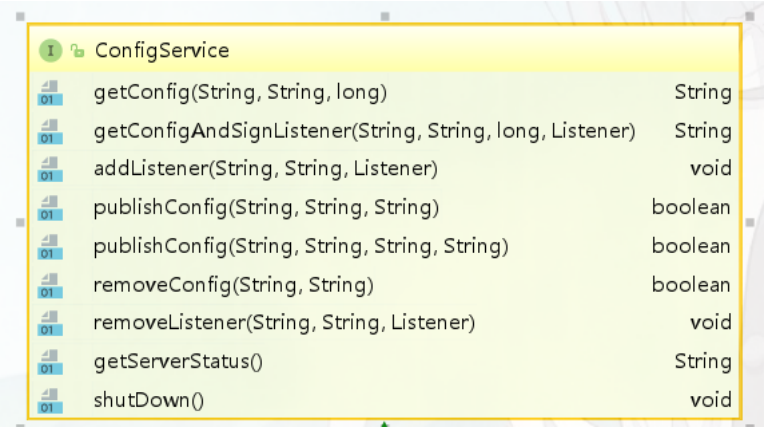
配置中心使用demo

```java
public class ConfigServerDemo {

 public static void main(String[] args) throws NacosException, InterruptedException {
 String serverAddr = "localhost";
 String dataId = "nacos-config-demo.yaml";
 String group = "DEFAULT_GROUP";
```

```
Properties properties = new Properties();
properties.put(PropertyKeyConst.SERVER_ADDR, serverAddr);
//获取配置服务
 ConfigService configService = NacosFactory.createConfigService(properties);
 //获取配置
 String content = configService.getConfig(dataId, group, 5000);
 System.out.println(content);
 //注册监听器
 configService.addListener(dataId, group, new Listener() {
 @Override
 public void receiveConfigInfo(String configInfo) {
 System.out.println("===recieve:" + configInfo);
 }

 @Override
 public Executor getExecutor() {
 return null;
 }
 });

 //发布配置
 //boolean isPublishOk = configService.publishConfig(dataId, group, "content");
 //System.out.println(isPublishOk);
 //发送properties格式
 configService.publishConfig(dataId,group,"common.age=30", ConfigType.PROPERTIES.getType());

 Thread.sleep(3000);
 content = configService.getConfig(dataId, group, 5000);
 System.out.println(content);

// boolean isRemoveOk = configService.removeConfig(dataId, group);
// System.out.println(isRemoveOk);
// Thread.sleep(3000);

// content = configService.getConfig(dataId, group, 5000);
// System.out.println(content);
// Thread.sleep(300000);

 }
}
```

2.2 nacos config client源码分析

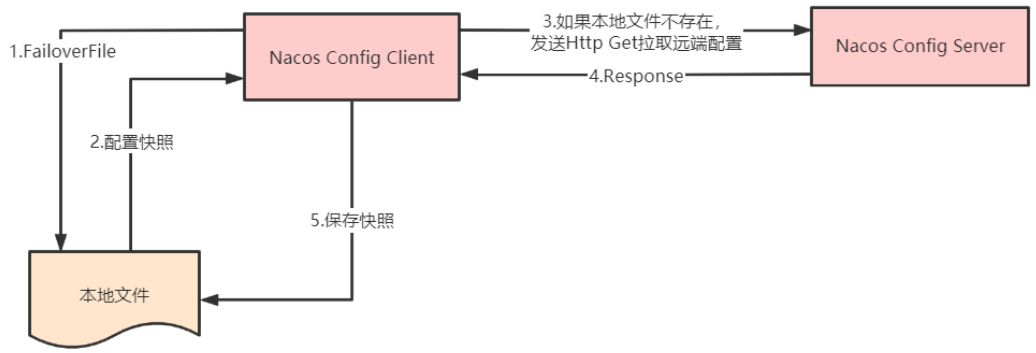
配置中心核心接口ConfigService



2.2.1 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 HTTP GET 方法从远端拉取配置，并保存到本地快照中。当通

过 HTTP 获取远端配置时，Nacos 提供了两种熔断策略，一是超时时间，二是最大重试次数，默认重试三次。

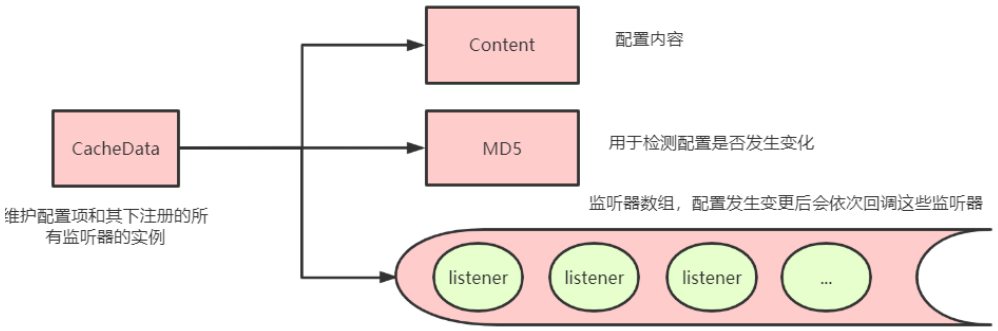


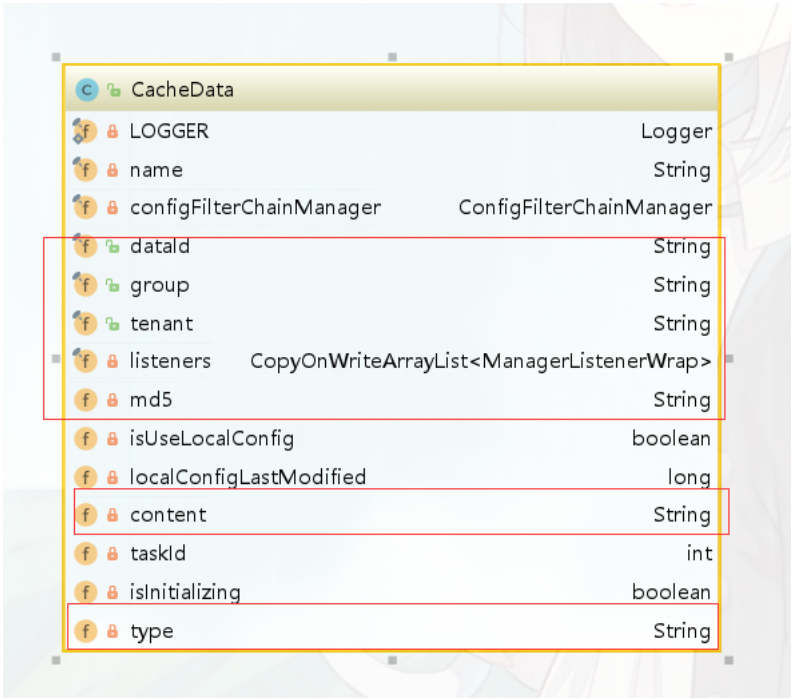
2.2.2 注册监听器

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。

```
NacosConfigService#getConfigAndSignListener
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：

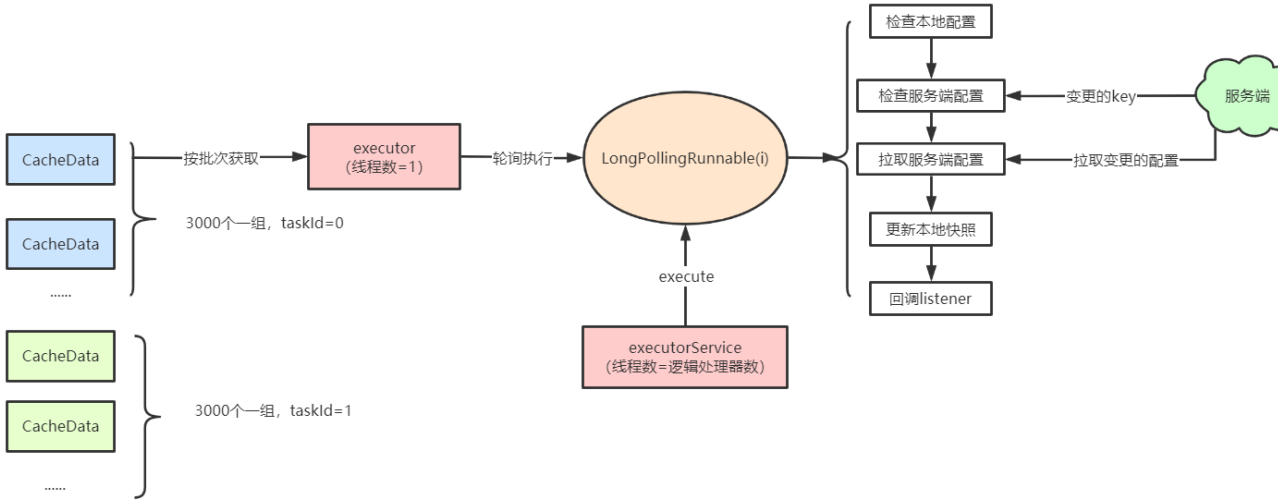




2.2.3 配置长轮询

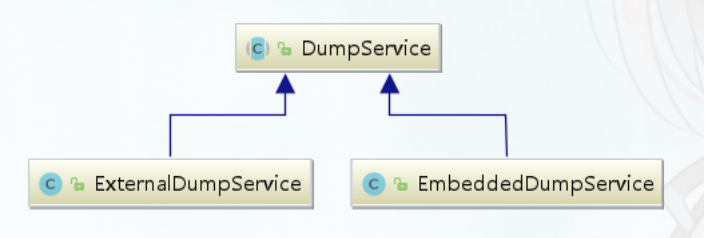
ClientWorker 通过其下的两个线程池完成配置长轮询的工作，一个是单线程的 executor，每隔 10ms 按照每 3000 个配置项为一批次捞取待轮询的 cacheData 实例，将其包装成为一个 LongPollingTask 提交进入第二个线程

池 executorService 处理。



2.3 nacos config server源码分析

2.3.1 配置dump

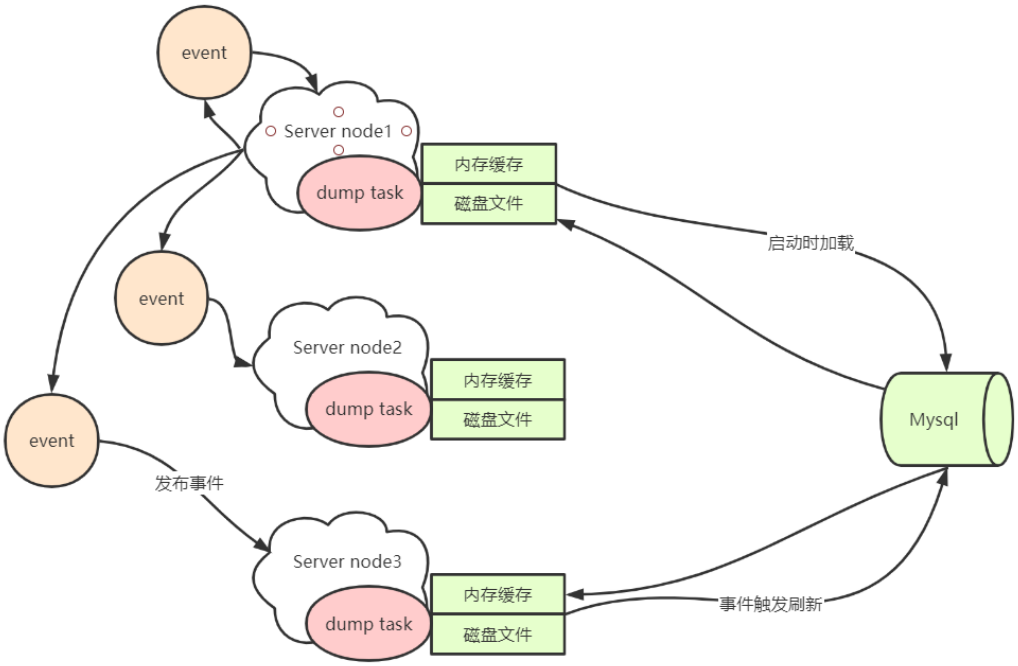


服务端启动时就会依赖 DumpService 的 init 方法，从数据库中 load 配置存储在本地磁盘上，并将一些重要的元信息例如 MD5 值缓存在内存中。服务端会根据心跳文件中保存的最后一次心跳时间，来判断到底是从数据库 dump 全量配置数据还是部分增量配置数据（如果机器上次心跳间隔是 6h 以内的话）。

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。

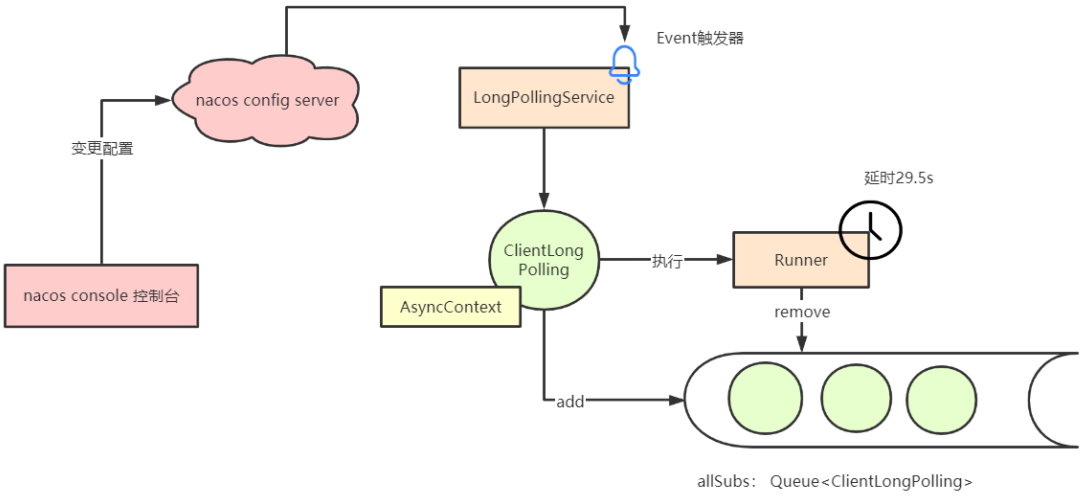
2.3.2 配置发布

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 HTTP 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。



2.3.3 处理长轮询

客户端会有一个长轮询任务，拉取服务端的配置变更，服务端处理逻辑在LongPollingService类中，其中有一个 Runnable 任务名为ClientLongPolling，服务端会将受到的轮询请求包装成一个 ClientLongPolling 任务，该任务持有一个 AsyncContext 响应对象，通过定时线程池延后 29.5s 执行。比客户端 30s 的超时时间提前 500ms 返回是为了最大程度上保证客户端不会因为网络延时造成超时。



文档：06. Alibaba微服务组件Nacos配置中心实...
链接：http://note.youdao.com/noteshare?id=9bf648ba1e0f8dcc4a247c676ea2e72b&sub=9869FC5BC1984CE0B1AFA0E1D92132E6