不要群里问了，Apollo这个坑你现在就记好！

占小狼的博客 昨天

> 作者：绝色天龙
>
> 来源：https://urlify.cn/6reuai

项目用的springboot，连带着配置中心也一直用的spring的配置中心，但是一直以来都有各种问题：

每次变更配置要重启配置中心和应用；各个环境集群的配置文件在不同分支，每个环境要分别手动同步，容易遗漏；无法确定当前应用是否重启过，读取的是最新配置。当然，个人觉得最重要的就是需要重启应用这个，简直就是浪费时间，浪费时间就是浪费生命啊。虽然spring也提供@RefreshScope这个注解来动态更新配置，但是用起来不是很方便，而且不能对散落在代码各处的配置统一处理，对老代码也无能为力，头大。

可能大家忍受够了，运维推了一个apollo配置中心来替换spring的配置中心，主打配置实时同步，真是直击程序员最深处心灵，皆大欢喜。正好以前也看过某位大神的分析，相比较他也是推荐apollo的，所以这边马上就红红火火的切换起来了。

过程

导client的jar包，做一些基本的配置，线上环境的启动参数运维会处理，这边方便本地启动会在配置文件中加一些配置：

项目resource 目录下增加 META-INF/app.properties 文件，文件中增加如下内容

```
app.id=web-admin
# 这里填上用于本地，线上运维会覆盖
apollo.meta=http://172.16.101.11:8080
# 配置的本地缓存目录，这行也可不配置，不配Apollo有默认值target/classes/config-cache
apollo.cacheDir=/opt/data/apollo-config
项目 resource 目录下增加 application.properties 文件， 文件中增加

apollo.bootstrap.enabled=true
apollo.bootstrap.namespaces=application,dev.common
```

"application", "dev.common" 为 应用配置 的namespace ， namespace相关概念移步官方文档，一个namespace相当于一个配置文件。

一切看上去都很顺利，为了验证配置是否实时更新推送，还写了一个监听器，把变化的配置打印出来，大致长这模样

```
@ApolloConfigChangeListener
private void someChangeHandler(ConfigChangeEvent changeEvent) {
    for (String key : changeEvent.changedKeys()) {
        ConfigChange change = changeEvent.getChange(key);
        log.info("Found change - {}", change.toString());
    }
}
```

登录apollo的炫酷管理页面，配置一改一发布，果然日志就啪啪啪打出来了，所以大功告成？还好留了个心眼，这里只是打印了变更事件里面的属性呀，实际的配置值变没变呢？搞了@Value注解的配置，然后变更前后的值都打印了一下，没问题，之前123，现在1234了，变得还挺快。但是测试@ConfigurationProperties注解的配置类的时候就发现问题了，怎么测都不变，配置变更事件是触发的，查看本地缓存也是变了，但是内存中的配置类对象的属性一直不变。网上是在找不到，只好看官方文档，果然发现了坑。。。附官方原话

需要注意的是，@ConfigurationProperties如果需要在Apollo配置变化时自动更新注入的值，需要配合使用EnvironmentChangeEvent或RefreshScope。相关代码实现，可以参考apollo-use-cases项目中的ZuulPropertiesRefresher.java和apollo-demo项目中的SampleRedisConfig.java以及SpringBootApolloRefreshConfig.java

也就是说，默认情况下@ConfigurationProperties注解标注的配置类是不会实时更新的，f**k，这不是坑么，最看重的功能默认不开启。照着官方文档写，结果发现第二种连配置都读不到了，就是@RefreshScope和@ConfigurationProperties一起用的时候会有问题，用的springboot的版本是1.3.2-RELEASE，不知道是不是版本太低的原因，最后用了第一种，还好最后成功了。

```
mponent
f4j
lic class ApolloConfigChanged implements ApplicationContextAware {

 private ApplicationContext applicationContext;

 @ApolloConfigChangeListener
 private void someChangeHandler(ConfigChangeEvent changeEvent) {
     for (String key : changeEvent.changedKeys()) {
         ConfigChange change = changeEvent.getChange(key);
```

```
            log.info("Found change - {}", change.toString());
        }

        // 更新相应的bean的属性值，主要是存在@ConfigurationProperties注解的bean
        this.applicationContext.publishEvent(new EnvironmentChangeEvent(changeEvent.chan
    }

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws Bean
        this.applicationContext = applicationContext;
    }
```

最后

大家从spring转到apollo的时候一定要注意，不要以为所有spring的配置都会变成实时更新的。

其实切换过程中还发生了一点小意外，我用了Idea的JRebel插件来热加载方便测试，但是每次更改类reload之后就会发现apollo的事件监听器列表会增加一个监听器，之后触发的配置更新事件也会被多个监听器处理，如果一直不重启，一直热加载的话，监听器列表会越来越长。。。

近期热文

- 面试官问：MySQL的自增ID用完了，怎么办？
- ArrayList插入1000w条数据之后，我怀疑了jvm...
- 蚂蚁二面，面试官问我零拷贝的实现原理，当场懵了...

最后，分享一份面试宝典《**Java核心知识点整理.pdf**》，覆盖了JVM、锁、高并发、反射、Spring原理、微服务、Zookeeper、数据库、数据结构等等。

获取方式：点“**在看**”，关注公众号并回复 **Java** 领取，更多内容陆续奉上

明天见(。·ω·。)/♡