开源之夏项目申请书

项目名称：Sermant 动态配置能力增强——支持 Apollo 配置中心

项目导师：李来

申请人：陈振洋

日期：2025.6.1

邮箱：andy271828@163.com

目录

一、 项目背景 1

1.1 Sermant 启动流程 1

1.2 配置中心的接入 2

二、 实现方案 3

三、 项目规划 5

四、 可行性分析 5

一、项目背景

1.1 Sermant 启动流程

Sermant 是基于 Java 字节码增强技术的无代理服务网格，为宿主应用程序提供服务治理功能，以解决大规模微服务场景中的服务治理问题。Sermant 由注入宿主应用的 JavaAgent、控制面板 BackendService 以及通过插件接入的配置中心组成，其中配置中心通过插件为各 Agent 和宿主应用实时推送配置，实现对服务运行时行为的集中控制与动态治理。

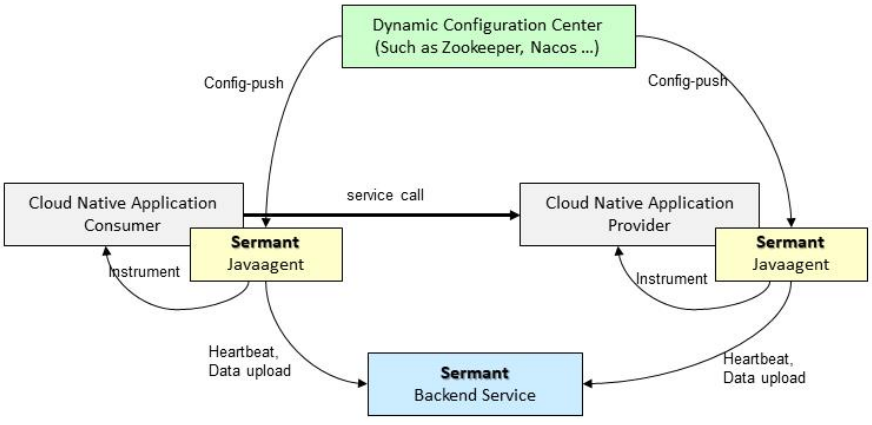

图 1. Sermant 结构图

Sermant 启动流程负责初始化核心框架与插件体系，建立字节码增强环境。

Sermant 以 AgentLauncher#launchAgent 为启动入口，首先通过 installGodLibs 将基础类（god 包）注入 BootstrapClassLoader，确保基础类的可见性。然后调用 installAgent，完成核心模块初始化。

其中插件系统在 PluginSystemEntrance#initialize 中完成初始化，主要流程有：

解析插件配置文件，构建插件类加载器，加载插件配置与服务实现，注册类加载器到 PluginClassFinder，以支持后续字节码增强定位类，最后将插件注册到全局 PLUGIN_MAP。

1.2 配置中心的接入

配置中心插件接入配置中心，并为 SermantAgent 插件和宿主进程提供动态读取配置的功能。

首先在 core 中定义了 DynamicConfigService 作为配置中心的抽象类，实现基本的对配置进行增删改查和订阅功能，其中，订阅可以使用 addConfigListener 单 key 订阅也可以使用 addGroupListener 实现组订阅。

当在配置中心变动时，sermant 的配置中心客户端会通过原生监听机制；客户端会调用绑定的回调方法，构造出一个 DynamicConfigEvent 对象，统一表达变更事件，传给 DynamicConfigListener#process 方法；其中 DynamicConfigListener 由不同插件定制化实现，启动时向 Service 注册。

BufferedDynamicConfigService 实现了 DynamicConfigService 接口，通过私有属性维护一个实现真实逻辑的 DynamicConfigService，从而启动配置好的 DynamicConfigService。BufferedDynamicConfigService 会在初始化时，基于动态配置的 serviceType 确定启动的服务，包括 Kie、Nacos 和 zookeeper(默认)。

以 ZookeeperDynamicConfigService 为例，ZooKeeperClient 是对 Zookeeper 原生客户端的封装类，主要负责底层连接、节点操作与监听管理。在初始化时基于配置创建 Zookeeper 连接，并提供常见的增删改查操作方法；ZooKeeperBufferedClient 提供了简化封装，由于 zk 的结构清晰，这部分没有做缓存，同层的 nacos 则做了缓存处理。在 ZooKeeperDynamicConfigService 中则实现了 DynamicConfigService 的抽象方法，提供了事件转换方法，并基于传入 Listener 构建匿名内部类，封装提供给 ZooKeeperBufferedClient 处理。

在此基础上，Sermant 插件需要通过 core 模块定义的 ConfigSubscriber 接口和 AbstractGroupConfigSubscriber 抽象类来接入配置中心。subscribe 采用了模板模式由子类定义订阅启动的判断和构建对配置的订阅，并返回 Listener。

```
@Override
public boolean subscribe() {
    if (dynamicConfigService == null) {
        LOGGER.severe( msg: "dynamicConfigService is null, fail to subscribe!");
        return false;
    }
    if (!isReady()) {
        LoggerFactory.getLogger().warning( msg: "The group subscriber is not ready, may be service name is null");
        return false;
    }
    final Map<String, DynamicConfigListener> subscribers = buildGroupSubscribers();
    if (subscribers != null && !subscribers.isEmpty()) {
        for (Map.Entry<String, DynamicConfigListener> entry : subscribers.entrySet()) {
            dynamicConfigService.addGroupListener(entry.getKey(), entry.getValue(), ifNotify: true);
            printSubscribeMsg(entry.getKey());
        }
    }
    return true;
}
```

图 2. subscribe 函数

对于宿主进程，Sermant 会通过 ConfigHodler 为宿主 springboot 提供配置源，DynamicConfigInitializer 利用 core 提供的 DefaultGroupConfigSubscriber 进行组订阅（group 为 service 维度，如 service=abc），初始配置会回调 ConfigListener，后者在处理过程中将配置提供给 ConfigHodler。

ConfigHolder 创建时，通过 SPI 加载当前 PluginClassLoader 下的 ConfigSource 实现，当 ConfigHolder 收到配置变更事件，调用 DefaultDynamicConfigSource 处理，处理完成后执行配置热更新。

```
public void resolve(DynamicConfigEvent event) {
    executorService.submit(() -> {
        boolean isNeedRefresh = false;
        for (ConfigSource configSource : configSources) {
            isNeedRefresh |= doAccept(configSource, event);
        }
        if (isNeedRefresh) {
            notifier.refresh(event);
        }
    });
}
```

图 3. resolve 代码

热配置需要 SpringEventPublisher 在 setApplicationEventPublisher 阶段，向 ConfigHolder 注册监听。ConfigHolder#addListener 中使用 RefreshNotifier 管理 ConfigSource 更新后需要感知配置变化的 Listener，在 notifier.refresh()中，刷新 RefreshScope 下的 Bean。

二、实现方案

在 1.2 的现有流程中，DynamicConfigService 参与了整体的配置注册和更新流程，具体实现对于下游是透明的，因此完成 Sermant 动态配置服务对 Apollo 配置中心的支持，主要改动点在于新建一个 ApolloDynamicConfigService 的实现类，并实现基于配置规则的加载满足 DynamicConfigService 接口功能和 Apollo 客户端特性的对接，同时 Apollo 提供了 OpenAPI 允许开发者通过编程方式管理配置中心，实现配置的增删改查等操作，提供了标准化的认证机制（通过 accessToken），支持对应用、集群、命名空间等维度的配置管理，包括创建、修改、删除配置项，以及发布配置变更，在开放过程中也需要引入来实现增删改操作。

新建一个 ApolloDynamicConfigService，并在 BufferedDynamicConfigService 构造函数中接入，在 start 函数中读取 agent 配置如 appId、metaServer、namespaces、openApiToken 等；初始化 ApolloClientWrapper 为每个 namespace 缓存 Config，对每个 namespace 添加一次 ConfigChangeListener；初始化 ApolloOpenApiClient，可复用单例 OkHttpClient 和 JSON 序列化器。此外，在 stop 中实现资源的关闭。

```
public BufferedDynamicConfigService() {
    // Different implementations are initialized depending on the configuration
    switch (CONFIG.getServiceType()) {
        case KIE:
            service = new KieDynamicConfigService();
            break;
        case NACOS:
            service = new NacosDynamicConfigService();
            break;
        default:
            service = new ZooKeeperDynamicConfigService();
    }
}
```

图 4. 委派 DynamicConfigService 实现类

在分层和设计上，需要考虑 Apollo 的客户端和 open API 方式分别实现监听和配置修改上传，需要分别实现客户端的 client 和 open API 服务，并在 BufferedClient 中统一接入。考虑在 Apollo 中，namespace 是非常大的粒度，不适合直接对映到 Sermant 的 group 概念，因此需要在 BufferedClient 中设计数据结构存储 group 和对应的 Listener，当客户端接收到请求时，转换并派发内部事件，并通过该 map 找到字段对应的 Listener 实现 process。

除此之外，在 NacosDynamicConfigService 还需要设计实现以下 DynamicConfigService 中的抽象函数函数：

1. doGetConfig(Stringkey,Stringgroup);获取某个配置项的值，其中 group 在 Apollo 中应对应为 namespace 下的配置项，可以先根据 namespace 监听获取配置，并从中取值返回 Optional<String>。doListKeysFromGroup(Stringgroup) 可参考上述方式，基于 namespace 获取配置，并根据缓存得到对应的 key 和 value。

2. doPublishConfig(Stringkey,Stringgroup,Stringcontent);用于设置一个 key 的值为 content。由于 Apollo 客户端只读特性，需要通过调用 Apollo 的 OpenAPI 实现配置的写入与发布。判断目标 key 是否已存在，执行新增或更新操作，并在写入后调用发布接口使配置生效。

3. doRemoveConfig(Stringkey,Stringgroup);通过 Apollo 的 OpenAPI 实现配置项的删除。将 group 映射为对应的 namespace，调用 OpenAPI 接口删除指定 key，适用于动态配置的逻辑下线或策略撤回场景。

4. doAddConfigListener(Stringkey,Stringgroup,DynamicConfigListenerlistener);通过 Apollo 客户端获取对应 namespace 下的 Config 实例，并注册一个 ConfigChangeListener。由于 Apollo 本身不支持单 key 订阅，需要在变更回调中手动筛选出目标 key，仅当该 key 变更时才触发 DynamicConfigListener.process()，实现逻辑层面的精确监听。

5. doRemoveGroupListener(Stringgroup);在本地维护的监听器 map 中移除 namepsace 下的配置，doRemoveConfigListener(Stringkey,Stringgroup);同样需要删除单 key 对应的所有监听器，可以考虑采用二级 Map 来维护本地的 Listeners，便于维护，同时需要注意线程安全。

6. doListKeysFromGroup(String group)在指定 namespace 上注册一个监听器，在回调中遍历所有变化 key，逐条构造 DynamicConfigEvent 并传入 listener.process()。doAddGroupListener(Stringgroup,DynamicConfigListenerlistener);在指定 namespace 上注册一个监听器，在回调中遍历所有变化 key，逐条构造 DynamicConfigEvent 并传入 listener.process()。

其中，由于监听事件是低频率的事件，Listener 采用单线程线程池；事件监听需要在 namespace 粒度，在 Apollo 回调过程中对每个发生变变更的 key 分派触发 DynamicConfigEvent。

三、项目规划

本人在暑假有充分的时间，并且保持每周至少一次和导师沟通。

时间	目标
7.1-7.6	进一步熟悉代码，和导师沟通确认开发思路和排期
7.7-7.20	开发实现 ApolloDynamicConfigService 的核心逻辑
7.21-8.3	完成 ApolloDynamicConfigService 在整体链路中的接入和适配
8.4-8.17	完成单测、自测，留出时间 debug
8.18-8.31	通过 Benchmark 测试验证万级配置项的监听与推送性能，确保生产可用性。
9.1-9.30	继续优化系统，编写说明和文档示例

四、自我介绍

本人计算机基础扎实，熟悉 Java 语言和 JVM 的类加载等机制，熟悉 zookeeper，了解 Nacos 和 Apollo 配置中心。同时在滴滴出行、度小满金融等企业有真实项目的实习经验，编码能力较强，并保持良好的文档撰写和测试习惯。本人在暑假期间有充分的时间投入，也对本项目非常感兴趣，希望能通过这次机会接触开源社区，并在项目结束之后持续在社区中做出贡献。