链滴

SkyWalking Java Agent 配置文件加载

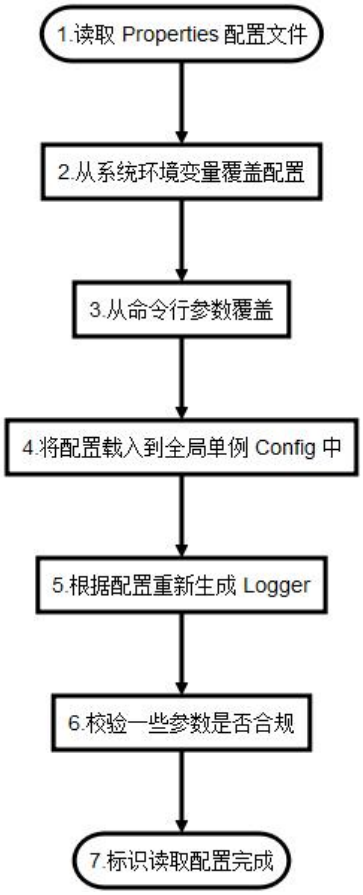
作者：noelcliu
原文链接：https://ld246.com/article/1669955941490
来源网站：链滴

1. SkyWalking Java Agent 总览
2. SkyWalking Java Agent 配置文件加载
3. SkyWalking Java Agent 插件加载
4. SkyWalking Java Agent 插件详解
5. SkyWalking Java Agent 自定义插件
6. SkyWalking Java Agent 微内核剖析
7. Skywalking Java Agent 服务详解

项目源码注释地址：https://github.com/ChuckChen123/skywalking-java/tree/chuck_learn

回顾

在 SkyWalking Java Agent 总览 中我们了解到 SkyWalking Java Agent 读取配置文件的流程如下：



总体可以分为两个步骤：

1. 从各渠道读取配置并合并
2. 做一些初始化及校验

接下来我会从这两个步骤来看看源码是如何实现的。

配置读取

读取配置文件

从 premain() 进入， SnifferConfigInitializer.initializeCoreConfig(agentArgs) 这一行代码进行了置加载，进入 initializeCoreConfig 方法，加载配置文件源代码如下：

```
public class SnifferConfigInitializer {

    ...

    public static void initializeCoreConfig(String agentOptions) {
        // 初始化 Properties 文件
        AGENT_SETTINGS = new Properties();
        // 1.读取配置文件
        try (final InputStreamReader configFileStream = loadConfig()) {
            AGENT_SETTINGS.load(configFileStream);
            for (String key : AGENT_SETTINGS.stringPropertyNames()) {
                String value = (String) AGENT_SETTINGS.get(key);
```

```
            // 处理占位符
            AGENT_SETTINGS.put(key, PropertyPlaceholderHelper.INSTANCE.replacePlaceholde
s(value, AGENT_SETTINGS));
        }

    } catch (Exception e) {
        LOGGER.error(e, "Failed to read the config file, skywalking is going to run in default c
nfig.");
    }

    ...

  }

  ...

  // 读取配置文件
  private static InputStreamReader loadConfig() throws AgentPackageNotFoundException, C
nfigNotFoundException {
    // 从系统环境变量中读取配置文件路径 --- skywalking_config
    String specifiedConfigPath = System.getProperty(SPECIFIED_CONFIG_PATH);
    // 如果没有配置则用默认地址 --- /config/agent.config
    File configFile = StringUtil.isEmpty(specifiedConfigPath) ? new File(
        AgentPackagePath.getPath(), DEFAULT_CONFIG_FILE_NAME) : new File(specifiedConfi
Path);

    if (configFile.exists() && configFile.isFile()) {
        try {
            LOGGER.info("Config file found in {}.", configFile);

            return new InputStreamReader(new FileInputStream(configFile), StandardCharsets.
TF_8);
        } catch (FileNotFoundException e) {
            throw new ConfigNotFoundException("Failed to load agent.config", e);
        }
    }
    throw new ConfigNotFoundException("Failed to load agent.config.");
  }
}
```

可以看到，程序会先从系统环境变量中读取 skywalking_config 这个配置，来看是否有配置过 config 文件的路径，如果没有配置，则用默认路径 /config/agent.config，然后使用 UTF-8 的编码格式将置文件读取成 InputStreamReader，最后由 Properties 对象进行解析，解析完成之后，进行了占位的处理，这样整个 Properties 文件的处理就结束了。

读取系统环境变量

配置文件读取完成之后，会进行系统环境变量的处理，相关代码如下：

```
public class SnifferConfigInitializer {

  ...

  public static void initializeCoreConfig(String agentOptions) {
```

```
        ...

        try {
            // 2.从系统环境变量覆盖配置
            overrideConfigBySystemProp();
        } catch (Exception e) {
            LOGGER.error(e, "Failed to read the system properties.");
        }

        ...

    }

    // 处理系统变量配置
    private static void overrideConfigBySystemProp() {
        Properties systemProperties = System.getProperties();
        for (final Map.Entry<Object, Object> prop : systemProperties.entrySet()) {
            String key = prop.getKey().toString();
            if (key.startsWith(ENV_KEY_PREFIX)) {
                // 获取 skywalking. 开头的配置信息，然后截取后半段进行覆盖
                String realKey = key.substring(ENV_KEY_PREFIX.length());
                AGENT_SETTINGS.put(realKey, prop.getValue());
            }
        }
    }

    ...

}
```

这里直接调用了内部私有方法 overrideConfigBySystemProp，直接获取了所有的系统配置，然后判 key 是否以 skywalking. 开头（ENV_KEY_PREFIX 的值为 skywalking.），拿到对应的 key 和 value 之后，我们可以看到，代码中直接将 key 和 value存到了存储 Properties 配置的 AGENT_SETTINGS 中。还记得总览说过的优先级顺序吗，命令行参数 > 系统环境变量 > 配置文件，这里代码使用了低先级优先加载，然后后面配置覆盖前面配置的方法来完成这个优先级的实现，这就是为什么会直接存到 AGENT_SETTINGS 的原因。

系统环境变量的处理结束了，让我们看看命令行参数的处理。

读取命令行参数

处理命令行参数相关源码如下：

```
public class SnifferConfigInitializer {

    ...

    public static void initializeCoreConfig(String agentOptions) {

        ...

        // 3.从命令行参数覆盖
        overrideConfigByAgentOptions(agentOptions);
```

```
        ...

    }

    // 处理命令行参数
    private static void overrideConfigByAgentOptions(String agentOptions) throws IllegalArgu
entException {
        for (List<String> terms : parseAgentOptions(agentOptions)) {
            if (terms.size() != 2) {
                throw new IllegalArgumentException("[" + terms + "] is not a key-value pair.");
            }
            AGENT_SETTINGS.put(terms.get(0), terms.get(1));
        }
    }

    ...

}
```

和处理系统环境变量类似，处理命令行参数也是调用了一个内部私有方法处理 —— overrideConfigBAgentOptions，可以看到，它将传入 initializeCoreConfig 的 agentOptions 传入到了 overrideConigByAgentOptions 中，其实 agentOptions 是来自于 premain 方法的，类似于 main 方法的 args 然后代码对 agentOptions 进行了处理，其实也就是一般的字符串分割，最后覆盖存入了 AGENT_SETINGS。

存储 Config

到这里就说明所有的配置已经加载处理完了，这一步是要将最终的结果存入到 Config 中，代码如下：

```
public class SnifferConfigInitializer {

    ...

    public static void initializeCoreConfig(String agentOptions) {

        ...

        // 4.覆盖全局 Config 类中的静态变量
        initializeConfig(Config.class);

        ...

    }

    public static void initializeConfig(Class configClass) {
        if (AGENT_SETTINGS == null) {
            LOGGER.error("Plugin configs have to be initialized after core config initialization.");
            return;
        }
        try {
            ConfigInitializer.initialize(AGENT_SETTINGS, configClass);
        } catch (IllegalAccessException e) {
            LOGGER.error(e,
```

```
                    "Failed to set the agent settings {}"
                        + " to Config={} ",
                    AGENT_SETTINGS, configClass
            );
        }
    }

    ...

}
```

这一步的操作其实就是将 AGENT_SETTINGS 中的内容写入到 Config 类中，Config 类是 SkyWalkin Java Agent 中的配置保存类，内部都是一些静态变量，这一步就是填充这些变量。

初始化及校验

加载和填充 Config 完毕之后，还需要做一些其他事情，代码如下：

```
public class SnifferConfigInitializer {

    ...

    public static void initializeCoreConfig(String agentOptions) {

        ...

        // 5.配置 Config 中的配置的 Log Solver
        configureLogger();
        // 6.因为初始化的时候配置了默认 Solver，现在 Solver 改变，需要重新生成 Logger
        LOGGER = LogManager.getLogger(SnifferConfigInitializer.class);

        // 7.校验一些参数是否合规
        if (StringUtil.isEmpty(Config.Agent.SERVICE_NAME)) {

            ...

        }

        ...

        // 8.将 Config Completed 设置为 true，表示配置已经读取完成
        IS_INIT_COMPLETED = true;
    }

    ...

}
```

由于 SkyWalking Java Agent 中的 Log 是自己实现的，它有两种 Log，一种是 JsonLogger，另外种是 PatternLogger，由于在配置文件中是支持配置使用哪一种 Logger 的，但是在 SnifferConfigIni ializer 还没有读取配置的时候，需要用到 Log 功能，所以在读取配置文件之前会默认初始化一个 Patt rnLogger，等到读取完配置文件之后，再重新按照新的配置进行 Logger 的初始化，也就是第 6 步处理完 Log 问题之后，会执行一些参数的校验，具体的校验可以去看源代码细节，最后会将 IS_INIT_ OMPLETED 设为 true，表示配置初始化已经完成了。

总结

那么对 SkyWalking Java Agent 的读取配置分析就结束了，可以看到对于配置的读取并不是特别复，SkyWalking Java Agent 并没有考虑配置文件来自于不同服务器或者网络的这种情况，毕竟是 Age t，应该暂时是还不需要这种场景。

接下来就是 SkyWalking Java Agent 比较核心的功能了 —— 插件加载。