项目申请书

项目名称： k8s任务与k8s集群链接优化

项目主导师：王兴杰

申请人： 陈文涛

日期： 2025.05.23

邮箱： 2738035238@qq.com

1.项目背景

1.1项目基本需求

- 项目简述：

目前DolphinScheduler中的worker在接受到master下发的k8s任务运行指令时会根据k8s的信息创建一个k8s链接，通过该链接与k8s集群之间保持联系。但当任务并发量大时，k8s集群要为每一个k8s任务维护一个链接，现有方案会造成信息同步效率差甚至与k8s集群失联。本课题针对该问题提

出优化方案，为一个DolphinScheduler的worker节点创建一个链接池（考虑会有多个k8s集群的情况，一个k8s集群一个链接），为每一个k8s集群创建一个公用链接，该worker中所有k8s节点通过链接池中的链接与k8s集群交互。该方案可以减少k8s对链接维护的成本，可以提高k8s任务类型并发量，提高状态同步效率。

- 项目产出要求：

a. 完善k8s任务 e2e测试

b. 完成k8s链接池部分unitTest测试

c. 本地测试阶段，完成k8s任务并行验证

1.2项目相关仓库

a. dolphinscheduler主仓库：https://github.com/apache/dolphinscheduler

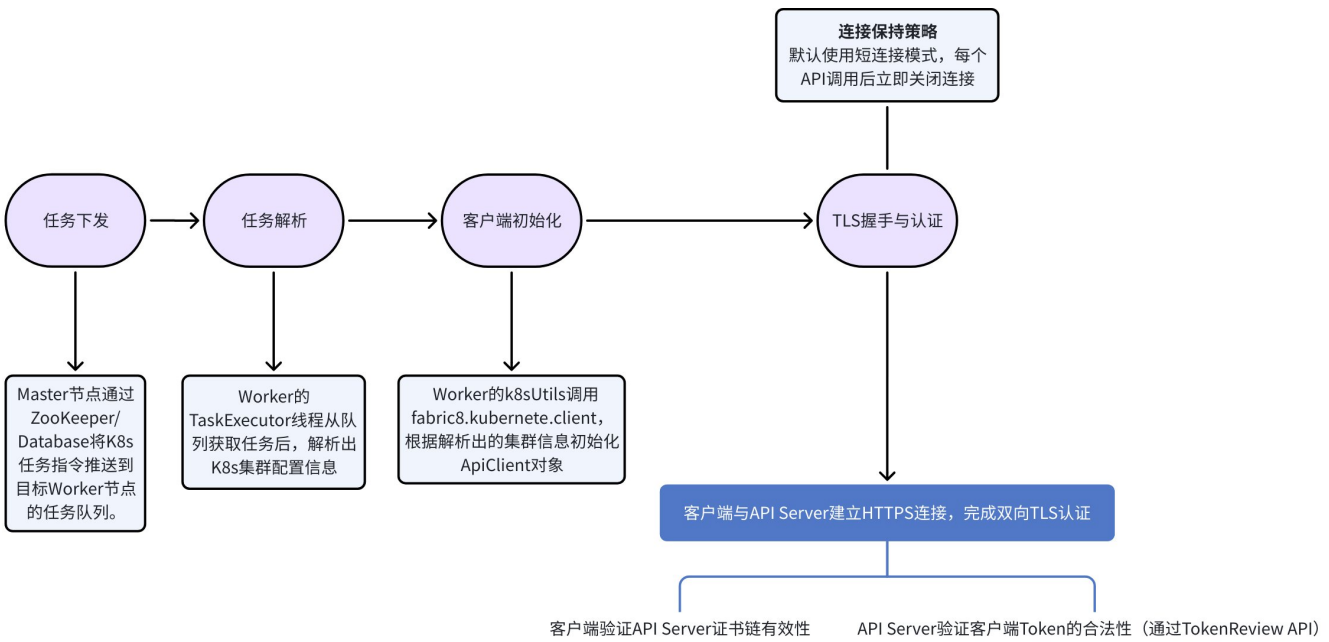
b. k8s主仓库：https://github.com/kubernetes/kubernetes

2.技术方法及可行性

2.1 问题原因阐述

当DolphinScheduler的Master节点向Worker节点下发K8s任务时，每个Worker会为每个K8s任务单独创建一个独立的K8s集群连接。Worker通过该连接与K8s集群交互（如创建Pod、查询任务状态等），每个任务独立维护自己的连接通道。然而当遭遇高并发环境时，就容易产生问题，例如当任务并发量极大时，K8s集群需要为每个任务维护一个独立连接，导致连接数与任务数成线性关系增长，对于k8s服务端来说，集群的API Server需要处理大量并发连接，消耗大量内存、CPU和网络带宽，可能超出其处理能力上限，同时也会因为负载上限从而导致效率降低，甚至因为连接数过高而导致k8s拒绝连接，造成worker与k8s集群失联，任务执行失败。情况类似于 “为每个网页请求单独建立一条专用光纤线路”，而非复用共享线路。在低并发时尚可运行，但高并发时线路成本和维护成本会指数级上升。

2.2 传统k8s连接方案



DolphinScheduler的Master节点向Worker节点下发K8s任务时，每个Worker会为每个K8s任务单独创建一个独立的K8s集群连接

- **设计原理**

代码块

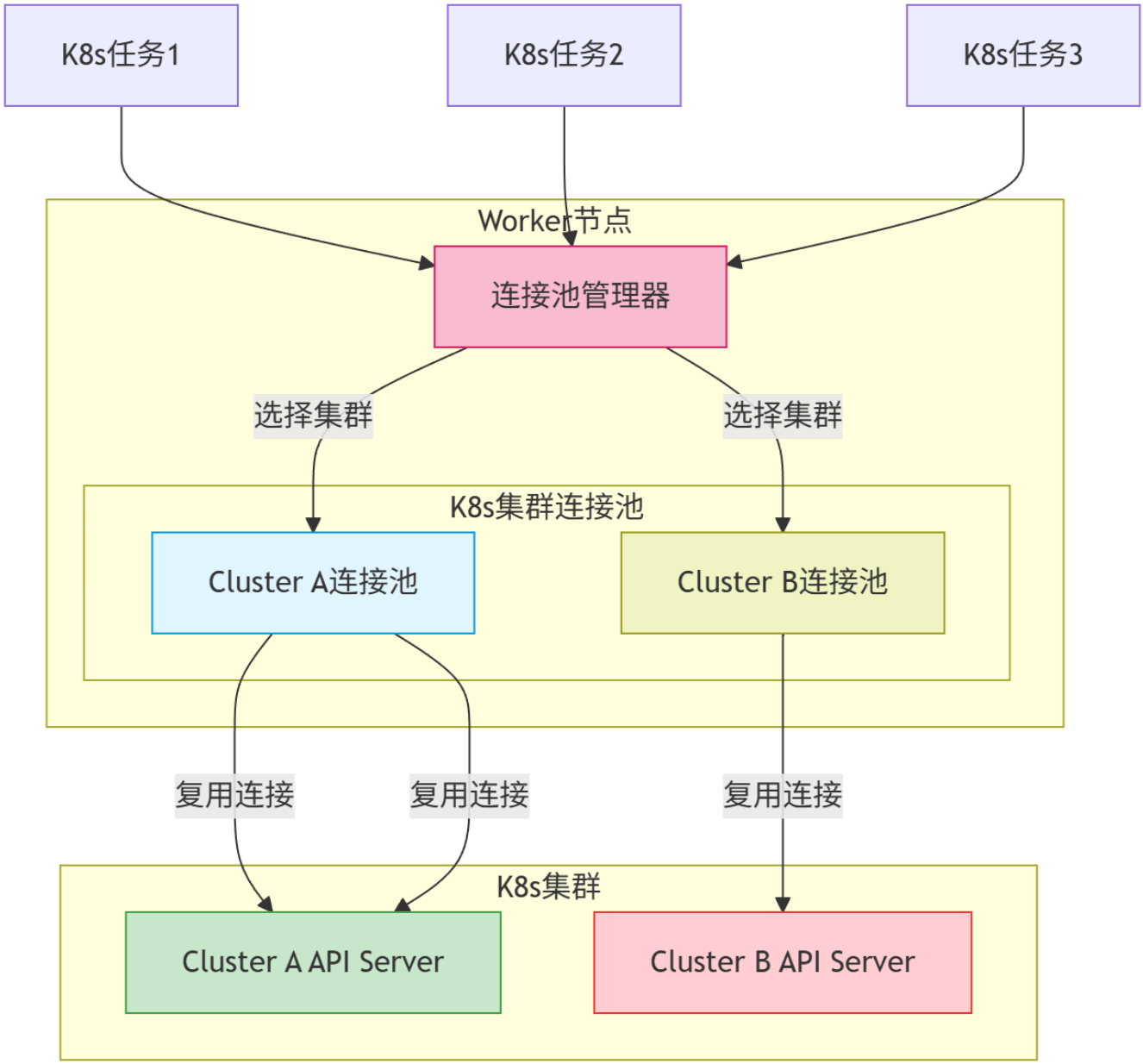
```
1   public void buildClient(String configYaml) {
2       try {
3           Config config = Config.fromKubeconfig(configYaml);
4           client = new KubernetesClientBuilder().withConfig(config).build();
5       } catch (Exception e) {
6           throw new TaskException("fail to build k8s ApiClient", e);
7       }
8   }
```

在DolphinScheduler中，每个Kubernetes任务需要与K8s API Server交互（如创建Pod、查询状态）。当前代码（`K8sUtils`类）的`buildClient`方法会为每个任务创建一个新的`KubernetesClient`实例，独立创建完整的HTTPS连接链，任务结束后立即关闭连接。

- **存在局限性：**

1. **连接池冗余：**每个`KubernetesClient`实例底层使用独立的`OkHttpClient`，各自维护连接池。
2. **资源浪费：**大量短连接导致TCP握手/TLS握手重复，增加延迟和CPU消耗。
3. **高并发瓶颈：**
   a. K8s API Server的默认并发限制（如`--max-requests-inflight=400`）容易被突破。
4. **状态同步延迟：**长轮询机制在连接数过多时，K8s API Server的请求排队延迟呈指数级增长

2.3 k8s连接池架构设计



由于当前 `K8sUtils` 类中的 `buildClient` 方法每次调用都会创建新的 `KubernetesClient` 实例，导致以下问题：

1. 相同集群配置下产生多个客户端实例
2. 底层OkHttp3连接池无法跨任务复用
3. 连接数随任务数线性增长

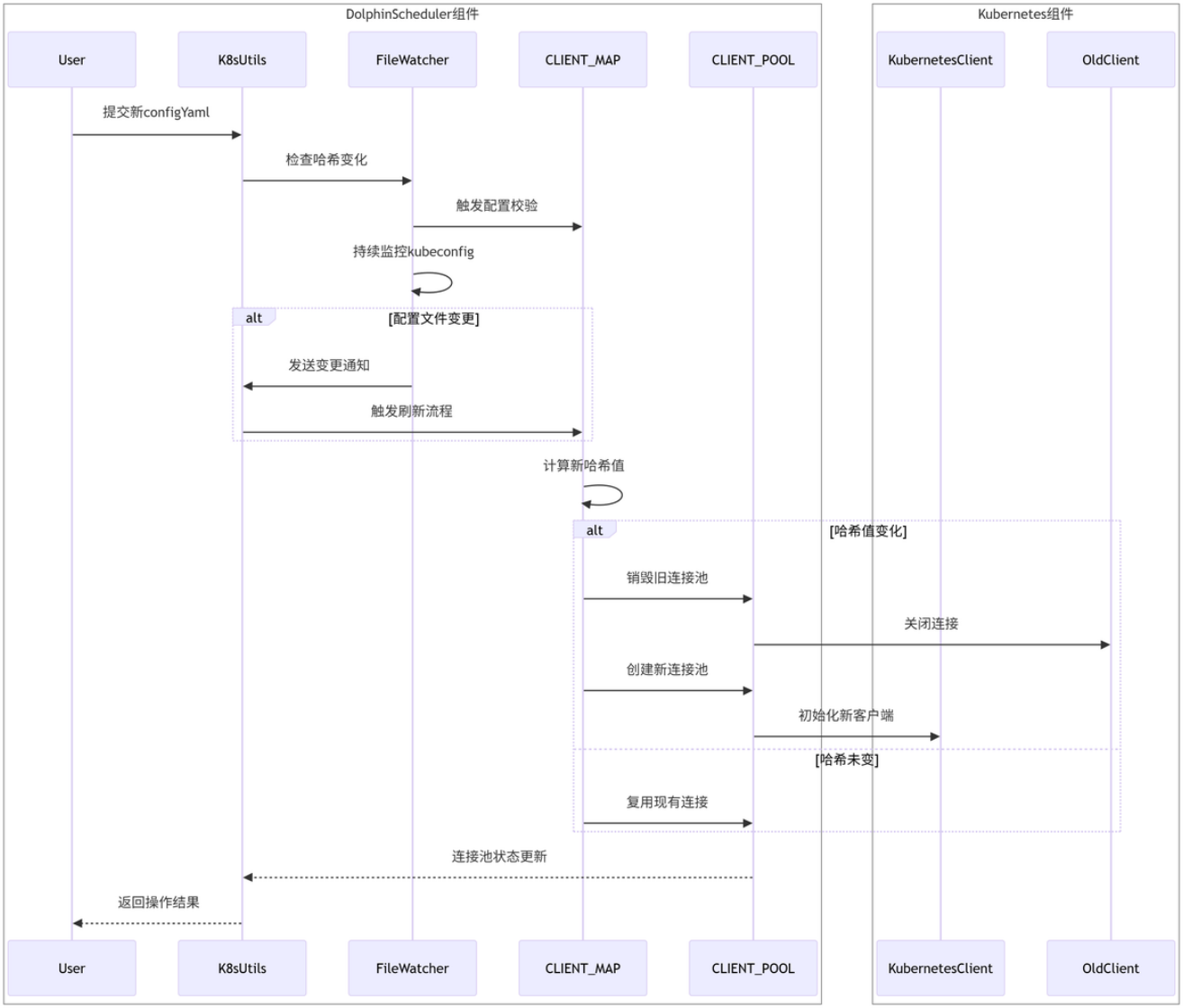
于是考虑在dolphinscheduler的k8sUtils中，对kubernetesClientBuilder创建的基于OkHttps的 `kubernetesClient` 进行修改，来复用其底层存在的https连接池。

与此同时，也要考虑，当k8s的配置 `config` 更新后，应当及时对worker中的连接进行 `关闭->更新->重启` 。

1. **连接池底层**
    - 在连接中封装K8s API客户端及元数据（集群名称、最后使用时间）
    - 对连接配置进行校验，确保配置信息一致从而实现复用
    - 使用 `ConcurrentHashMap` 缓存客户端，键作为集群配置哈希值
2. **任务路由逻辑**
    - 根据任务元数据（如Namespace标签）选择对应的连接池
    - 采用“**借出-使用-归还**”模式，任务执行前从池中获取连接，完成后立即归还（避免泄漏）
3. **证书更新**
    - 定期监听证书的变化，使用hash对文件进行校验，当发现变化后及时更新

3. 项目实现细节梳理

3.1 设计细节



- 相同 `configHash` 的KubernetesClient共享同一个OkHttpClient实例
- 连接池参数通过Builder显式设置
- 通过工厂类实现证书热更新

---

1. 创建自定义HttpClient工厂

由于KubernetesClientBuilder所支持的参数配置是是 `HttpClient.Factory` 类型，所以需要提前创建一个自定义的 `HttpClient.Factory` 类

代码块

```
1  public class SharedHttpClientFactory implements HttpClient.Factory {
2      private static final ConcurrentHashMap<String, OkHttpClient> CLIENT_POOL=
   new ConcurrentHashMap<>();
3      private final String configHash;
4
5      public SharedHttpClientFactory(String configHash) {
```

```
6           this.configHash = configHash;
7       }
8
9       @Override
10      public OkHttpClient createHttpClient(Config config) {
11          return CLIENT_POOL.compute(configHash, (key, client) -> {
12              if (client == null) {
13                  return new OkHttpClient.Builder()
14                      .connectionPool(new ConnectionPool(200, 15,
    TimeUnit.MINUTES))
15                      .connectTimeout(30, TimeUnit.SECONDS)
16                      .build();
17              }
18              return client;
19          });
20      }
21
22      // 配置变更时关闭旧客户端
23      public static void refreshClient(String configHash) {
24          OkHttpClient oldClient = CLIENT_POOL.remove(configHash);
25          if (oldClient != null) {
26              oldClient.dispatcher().executorService().shutdown();
27              oldClient.connectionPool().evictAll();
28          }
29      }
30  }
```

2. 对k8sUtils进行重构

代码块

```
1   public class K8sUtils {
2   // 创建一个clientEntry类，String是集群元数据信息的HASH
3   public class K8sUtils {
4     private static final ConcurrentHashMap<String, ClientEntry> CLIENT_MAP = new
    ConcurrentHashMap<>();
5     private KubernetesClient client;
6
7       private static class ClientEntry {
8           private final KubernetesClient client;
9           private final String configYaml;
10          private volatile long lastAccessed;
11
12          ClientEntry(KubernetesClient client, String configYaml) {
13              this.client = client;
14              this.configYaml = configYaml;
```

```
15              this.lastAccessed = System.currentTimeMillis();
16          }
17
18          void updateLastAccessed() {
19              this.lastAccessed = System.currentTimeMillis();
20          }
21      }
22
23  //...
24
25  // 重构createjob方法
26      // 在执行操作前检查连接有效性
27      // 证书变化时触发客户端client刷新
28  // 重构buildClient方法
29  // 加入对configyaml的hash校验，以及通过工厂类中的连接池进行维护client连接
30      private static String computeHash(String configYaml) {
31          return Hashing.sha256().hashString(configYaml,
    StandardCharsets.UTF_8).toString();
32          }
33      }
34      // 配置变更时触发刷新
35          // 配置文件监听扩展
36      // 增加连接回收机制
37      }
38  }
```

3.2 其他设计补充

1. **哈希校验机制**：采用SHA-256对kubeconfig内容哈希，确保配置变更可检测
2. **双层清理策略**：
   - 主动清理：检测到配置变更时立即关闭旧连接
   - 被动回收：30分钟未使用的连接自动释放
3. **令牌预检机制**：关键操作执行前校验证书有效期，提前7天预警

4.规划

1. **项目开发的第一阶段（7月1日-8月15日）**
   - 完成DolphinScheduler的k8s任务连接池的初步的代码编写
   - 对实现了连接池后的DolphinScheduler进行单机部署测试

i. **混沌测试场景**：
   - 在任务执行过程中随机变更kubeconfig
   - 模拟API Server的滚动重启

ii. **长连接测试**：持续运行24小时，验证连接泄漏情况

- 实现正式集群化调度时的DolphinScheduler部署测试，校验是否有问题

2. **项目开发的第二阶段（8月16日-9月30日）**
   - 使用用例测试项目开发内容
   - 对总体的工作内容进行总结，编写相应的开发文档
   - 进一步完善可以改进或补充的工作