top-task-scheduler设计杂谈

# 背景介绍

top-task-scheduler想做成一个分布式的任务执行框架，单一的Master节点负责任务的管理，分发，多个Slave并发的请求任务，执行任务。top-task-scheduler其实只是一个半成品的框架，需要框架的使用者自定义任务，任务的消费接口，任务具体怎么执行与框架无关，框架只负责管理分发任务，管理集群。

# 设计理念

top-task-scheduler是基于waverider的，构建于waverider之上。通过抽象任务（**Job**）的概念，任务在top-task-scheduler其实就是一个javaBean，是对任务的基本描述，如移动数据的任务，并不会将需要移动的数据包含在任务中，只需要提供一些基础的信息，比如源数据库，源表，查询条件，目的数据库，目的表等等，这个样的信息。因为任务会在网络上传输，所以任务必须做到小，可序列化。

每一个任务都有一个执行计划（**JobExecutePlan**），目前一个任务只支持一个执行计划。

目前执行计划分为两种：

1. SimpleJobExecutePlan，简单任务执行计划
2. CronJobExecutePlan，Cron任务执行计划

Master依据任务的执行计划控制任务的分发，回收。

每一个任务都属于一个任务组，每一个Slave可以指定只处理指定的任务组的任务。一个任务组有一个指定的任务消费接口。

每一个任务都有自己的状态：

1. *JOB\_STATUS\_RUNNING 任务正在某个Slave执行*
2. *JOB\_STATUS\_EXECUTED 任务被执行了，但是任务下次还要执行，如周期性任务*
3. *JOB\_STATUS\_FAILED 任务执行失败，框架级别的失败*
4. *JOB\_STATUS\_TIMEOUT任务超时*
5. *JOB\_STATUS\_COMPLETED 任务彻底完成, 任务执行计划指定任务不会在被执行了*
6. *JOB\_STATUS\_WAITING 任务处于等待，等待下次执行*
7. *JOB\_STATUS\_RELOADING 任务处于Reload状态*

# 概要设计



**图3-1 系统概要图**

## Master端

Master端主要负责任务的管理，分发，接收任务结果，任务的Reload，回调等。



**图3-2 Master模块图**

**Master端由如下模块组成：**

1. **JobManager** 任务管理器，主要负责任务的动态添加，删除，任务执行计划安排(依赖**JobExecutePlanManager**)，任务分发服务。
2. **JobProvider** 任务提供器，是业务方提供的任务产生器，JobManager在初始化时，利用JobProvider加载初始任务。目前支持在Master注册多个JobProvider。
3. **JobXXCallback** 任务回调接口，业务方提供的任务回调接口，目前支持的回调如下：
   1. *JOB\_CALLBACK\_CONDITION\_WHEN\_JOB\_EXECUTED, 当任务被执行了并且未完成（有些任务需要执行多次，以及周期执行），执行回调。*
   2. *JOB\_CALLBACK\_CONDITION\_WHEN\_JOB\_COMPLETED，当任务完成时，执行回调*
   3. *JOB\_CALLBACK\_CONDITION\_WHEN\_JOB\_TIMEOUT，当任务超时时，执行回调*
   4. *JOB\_CALLBACK\_CONDITION\_WHEN\_JOB\_FAILED，当任务执行失败(系统级)时，执行回调*
   5. *上面四种的组合，目前组合只支持OR条件*

系统在回调之前，会停止调度需要回调的任务，并从调度器移除，业务提供的回调方法，自行决定怎么处理任务，可以通过JobManager来处理改任务。

1. **JobExecutePlanManager** 任务执行计划管理器，负责调度任务的执行计划，在任务需要执行的时候，通知JobManager。**JobExecutePlanManager**其实是对quartz的包装，通过将top-task-scheduler的任务执行计划转换成quartz job来实现。
2. **PullJobCommandHandler Master**端处理Slave请求任务的命令消费接口，注册到Waverider.MaterNode, 当Slave发送请求任务的命令时，Master通过PullJobCommandHandler，请求JobManager分发任务。
3. **PushResultCommandHandler Master**处理Slave推送任务结果命令的消费接口，注册到Waverider.MaterNode, 当Slave推送任务结果命令时，Master通过**PushResultCommandHandler**，调用JobManager设置任务执行情况。
4. **Waverider.MaterNode Master**依赖的Waverider的Master节点。

细说Master各个模块

## **Slave端**

Slave端干的事情很简单，需要任务了，发送拉取任务的请求到Master，得到任务后，将任务提交到对应的任务组执行器JobGroupExecutor，有任务执行完成了，将结果发送给Master，Slave就是不停的干这事。



**图3-3 Slave模块图**

**Slave端由如下模块组成：**

1. **JobGroupExecutor** 任务组执行器，一个Slave可以注册多个不同的任务组执行器，一个任务组执行器只负责执行对用任务组的任务。
2. **JobConsumer** 任务消费接口，业务方提供的任务消费接口，负责执行任务。
3. **CommandProvider** 命令启动器，负责按需向Master发起拉取任务命令，和按需想Master发送推送任务结果命令，目前做的比较矬，定时扫描，需要改进。
4. **PushJobCommandHandler** Slave端消费Master推送任务命令的消费接口，其实就是将相应的任务提交到具体的JobGroupExecutor。
5. **Waverider.SlaveNode**依赖的Waverider的Slave节点。

细说Slave各个模块

# 测试

目前只是做了一个简单的性能测试，测试的结果如下：



# 反思

单Master能支持的最大任务数，有限。

性能测试没做

Master端其实也需要支持一些周期执行的任务。

# 使用情况

top-monitor在使用，top-monitor的任务并不多，任务执行时间还是有保障的。