**top-task-scheduler Quick Start**

1. **概述：**

top-task-scheduler是一个基于Master-Slave结构的任务调度框架，单节点的Master管理任务并按照任务的执行计划将任务下发到指定Slave上执行，并用任务的执行结果回调业务方提供的回调接口。

1. **明细：**

**任务内容(JobContent)**

由业务方自定义，一个实现JobConent接口的JavaBean，定义自己的业务级的任务内容，由业务方自定的JobConsumer接口处理，任务内容(JobContent)本质上只是一个JavaBean，装载业务方在执行一个自定义任务时需要的一些信息。任务内容(JobContent)由Master端的业务方提供的JobProvider接口产生，并在网络上传输到Slave端由JobConsumer处理。因为任务内容(JobContent)会在网络上传输，所以要求任务内容(JobContent)尽量的小，只传输一些必要的控制信息(任务的策略)。如实现一个更新数据库记录的任务，只需在任务内容(JobContent)指定需要处理那些记录，如id % 10 == 0的记录，而不是将所有的需要处理的数据库记录通过任务内容(JobContent)传输。

**任务结果(JobResult)**

由业务方自定义，一个实现JobResult接口的JavaBean，是业务方的Slave端由JobConsumer处理后返回的一个相应，业务方可以子定义并进行相应的处理。

**任务(Job)**

任务(Job)由系统定义，主要是承载业务方的任务内容(JobContent)和任务结果(JobResult)，当然也承载系统定义的一些数据。业务方的任务内容(JobContent)和任务结果(JobResult)会随着任务(Job)在网络上传输。

任务(Job)是系统调度的最小单元，系统对任务的调度由任务(Job)的执行计划指定。

**任务执行计划(JobExecutePlan)**

任务执行计划(JobExecutePlan)控制着任务的调度，任务什么时间开始执行，是否周期执行，执行周期等。每一个任务都需要设定一个任务执行计划。

任务执行计划有如下两种：

1. **简单任务执行计划(SimpleJobExecutePlan)**

简单任务执行计划，通过设置如下属性来控制任务的调度：

|  |  |  |
| --- | --- | --- |
| **属性** | **控制** | **备注** |
| priority | 任务的优先级 | 0-10，值越大有优先级越高 |
| startTime | 任务首次开始执行的时间 | Date类型 |
| repeatCount | 任务重复执行次数 |  |
| repeatInterval | 任务重复执行周期 | 单位秒 |

这样的执行计划，控制Master在startTime指定的时间将任务分发到相应的Slave上执行，并在startTime + n \* repeatInterval重复执行任务repeatInterval次。

1. **Cron任务执行计划(CronJobExecutePlan)**

Cron任务执行计划，通过设置如下属性来控制任务的调度：

|  |  |  |
| --- | --- | --- |
| **属性** | **控制** | **备注** |
| priority | 任务的优先级 | 0-10，值越大有优先级越高 |
| cron | cron表达式 | String类型 |

这样的执行计划，控制Master在cron表达的执行方式去下发任务到Slave。

**任务分组(JobGroup)**

任务分组，通过将一系列任务统一在一个命名空间下，并可以将指定的任务分组分发到指定的Slave机器上。一个新加入的Slave机器可以指定只消费一个或多个任务分组的任务。

可以设置任务的Key的group字段指定任务的分组。

**任务提供器(JobProvider)**

任务提供器是业务方需要提供的一个或多个服务，通过实现JobProvider接口并注册到Master端，Master在启动时，会调用业务方提供的这个服务产生初始的任务，如果Master开启了定时重新加载任务的功能，并在Master在运行中重现加载任务时调用产生新的任务。业务方的任务提供器实现请参考demo的**DefaultJobProvider**实现。

**任务消费器(JobConsumer)**

任务消费器是业务方需要提供的一个或多个服务，通过实现JobConsumer接口并注册到Slave端，Slave端在接收到Master端分发的任务，Slave调用JobConsumer.work(Job job)方法执行业务相关的任务，work方法具体实现依赖业务方，业务方需要保证work是线程安全的，Slave会在多线程环境下调用，Slave通常会用一个线程池去执行任务。

**任务分组执行器(JobGroupExecutor)**

任务分组执行器，是系统提供的，目前还不支持业务方提供，后期会考虑，业务方直接提供任务分组执行器。Slave端可以配置任务分组执行器，通过指定任务分组执行器消费的任务分组名，并指定对应的任务消费器和资源配额(任务队列容量，最小线程数，最大线程数)。Slave端在接收到Master端分发的任务，通过任务分组执行器调用对应的任务消费器(JobConsumer)的work方法。

任务分组执行器资源配额(**ResourceConfig**)如下表：

|  |  |  |
| --- | --- | --- |
| **配置** | **含义** | **备注** |
| minThread | 线程池最小线程数 | ThreadPoolExecutor的corePoolSize参数 |
| maxThread | 线程池最大线程数 | ThreadPoolExecutor的maximumPoolSize参数 |
| maxQueueSize | 任务队列容量 | 线程池工作队列的容量 |

**任务回调**

任务回调功能是，可以细粒度的针对每一个任务设置回调策略，回调策略有如下几种，以及组合。

任务回调策略：

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。

**任务Reload**

任务回调功能是，可以细粒度的针对每一个任务设置Reload策略，Reload策略有如下几种，以及组合。

任务回调策略：

1. *JOB\_RELOAD\_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Reload。*
2. *JOB\_RELOAD\_CONDITION\_WHEN\_JOB\_COMPLETED，当任务完成时，执行Reload*
3. *JOB\_RELOAD\_CONDITION\_WHEN\_JOB\_TIMEOUT，当任务超时时，执行Reload*
4. *JOB\_RELOAD\_CONDITION\_WHEN\_JOB\_FAILED，当任务执行失败(系统级)时，执行Reload*
5. *上面四种的组合，目前组合只支持OR条件*

系统在Reload之前，会停止调度需要Reload的任务，并从调度器移除，业务提供的Reload方法，可以返回Reload后的任务，如果如果返回非NULL调度器会重新调度返回的Reload后的任务。

注意：

建议使用任务Reload来修复，或动态修改任务。

**任务执行日志(任务执行透明化)**

任务执行日志，可以配置Master将任务执行日志记录到日志文件中，日志记录类似log4j的DailyRollingFileAppender，每天一个日志文件，默认不记录日志。每条日志包含如下几部分信息：

1. 任务基本信息, 标识，状态, 超时设置，任务是否被标记为移除
2. 任务执行计划，任务执行计划，任务下次执行时间
3. 任务执行统计, 分发次数，成功执行次数，执行失败次数，超时次数
4. 任务本次执行统计，本次执行slave信息，执行是否成功，Master分发时间, Master确认本次执行完毕时间，Slave接收到任务的时间，Slave开始执行任务时间，Slave执行任务接收时间

任务的每一次执行都会在日志文件中记录一条记录，记录格式如下表

|  |  |  |
| --- | --- | --- |
| **分组** | **字段** | **注释** |
| 任务基本信息 | id | 内部标识 |
| key | 任务键，包括分组名，任务名  key.group, key.name |
| state | 任务状态   1. Running 2. Executed 3. Done 4. Waiting 5. Reloading |
| timeout | 任务超时时间，Master端， 目前只在Master端有用，单位毫秒 |
| isRemoved | 任务是否被标记移除 |
| 任务执行计划 | plan | 执行计划，复合字段 |
| nextExecuteTime | 任务下次执行时间 |
| 执行统计 | dispatchedCount | Master分发任务次数 |
| executedCount | 成功执行次数(系统级成功) |
| failedCount | 执行失败次数(系统级失败) |
| timeoutCount | 超时次数 |
| 本次执行统计 | jobExecutionInfo | 任务一次执行统计 |

任务执行计划日志格式

|  |  |  |
| --- | --- | --- |
| **执行计划类型** | **字段** | **注释** |
| SimpleJobExecutePlan | simple | 标识 |
| startTime | 首次开始执行时间 |
| repeatCount | 重复执行次数 |
| repeatInterval | 重复周期，单位秒 |
| priority | 优先级 |
| CronJobExecutePlan | cron | 标识 |
| priority | 优先级 |

任务一次执行统计

|  |  |  |
| --- | --- | --- |
| **分组** | **字段** | **注释** |
| 本次执行的Slave | worker | 标识 |
| 执行信息 | isSucceed | 执行是否成功，系统级别 |
| dispatchedTime | Master分发时间 |
| endTime | Master确认执行完成时间 |
| slaveReceivedTime | Slave接收到任务时机 |
| slaveJobStartTime | Slave开始执行任务时间 |
| slaveJobEndTime | Slave执行任务结束时间 |

Slave信息

|  |  |  |
| --- | --- | --- |
| **字段** | **注释** |  |
| id | 内部标识 |  |
| name | 名称 |  |
| ip | IP |  |
| port | 端口 |  |

1. **系统结构：**

****

系统依赖另外一下系统waverider来实现网络通信，waverider是一个基于Master-Slave结构的命令执行框架，Slave主动从Master拉取命令(Command)并执行然后将执行的结果封装在Command返回到Master。

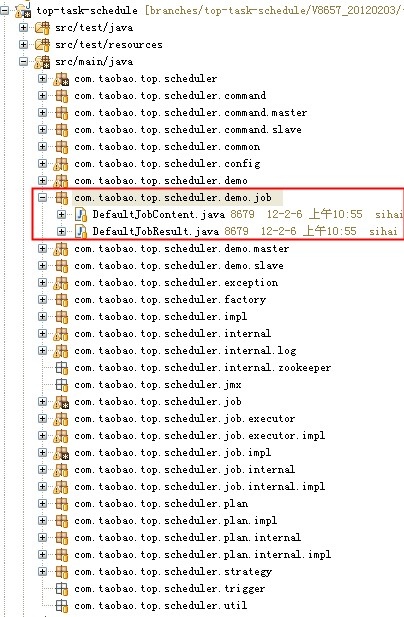
1. **使用实例：**

请参加系统demo包下的Main。

1. 定义自己的JobConent和JobResult，请参考：

com.taobao.top.scheduler.demo.job.DefaultJobContent

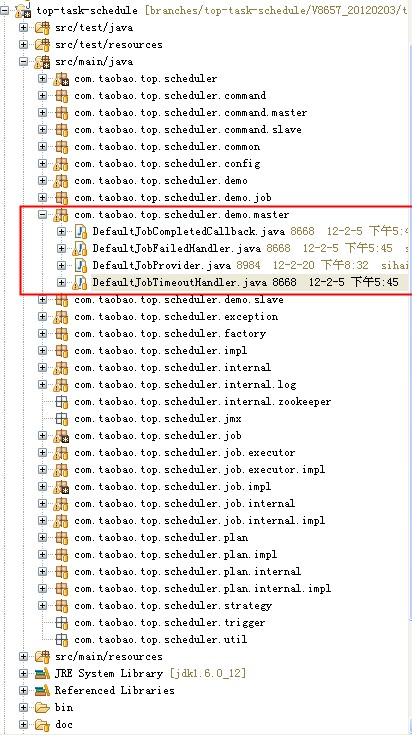
com.taobao.top.scheduler.demo.job.DefaultJobResult



1. Master端

Mater端需要提供如下类：

* 1. 任务产生器，请参考 com.taobao.top.scheduler.demo.master.DefaultJobProvider，可以实现多个任务产生器并注册到Master。
  2. 任务执行完成回调，请参考 com.taobao.top.scheduler.demo.master. DefaultJobCompletedCallback，可以实现多个任务产生器并注册到Master。
  3. 任务执行失败回调，请参考 com.taobao.top.scheduler.demo.master. DefaultJobFailedHandler，可以实现多个任务产生器并注册到Master。
  4. 任务执行超时回调，请参考 com.taobao.top.scheduler.demo.master. DefaultJobTimeoutHandler，可以实现多个任务产生器并注册到Master。



1. Slave端
   1. 任务消费器，请参考 com.taobao.top.scheduler.demo.slave. DefaultJobConsumer，可以实现多个任务产生器并注册到Slave。
2. 启动类

请参考om.taobao.top.scheduler.demo.Main