[一、Job配置 1](#_Toc8781)

[二、JobScheduler初始化流程 2](#_Toc18962)

[三、JobScheduler调度流程 3](#_Toc12195)

[四、AbstractElasticJobExecutor调度流程 4](#_Toc19331)

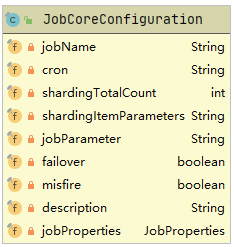
[五、任务执行事件上报流程 7](#_Toc10394)

[六、Failover机制 8](#_Toc31710)

[七、控制台常用操作 9](#_Toc3173)

[八、ZK数据结构 11](#_Toc7688)

一、Job配置

JobCoreConfiguration

jobName: String

cron: String

shardingTotalCount: int

shardingItemParameter: String, 0=a,1=b,2=c

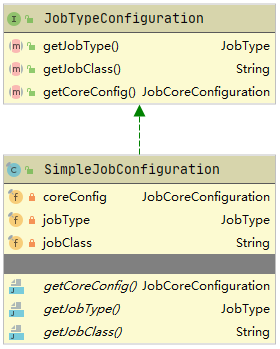
jobParameter: String, 作业自定义参数

failover: boolean

misfire: boolean

description: String

jobProperties: JobProperties, job\_exception\_handler(作业异常处理器)与 executor\_service\_handler(线程池服务处理器)

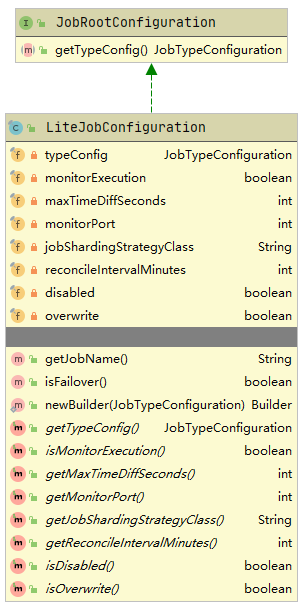


SimpleJobConfiguration:

coreConfig: JobCoreConfiguration

jobType: JobType, SIMPLE, DATAFLOW, SCRIPT

jobClass: String

LiteJobConfiguration：

typeConfig: JobTypeConfiguration

monitorExecution: boolean, 监控作业运行状态

maxTimeDiffSeconds: int, 最大允许的本机与注册中 心的时间误差秒数

monitorPort: int, 作业监控端口用于dump作业信息

jobShardingStrategyClass: String

reconcileIntervalMinutes: int

disabled: boolean

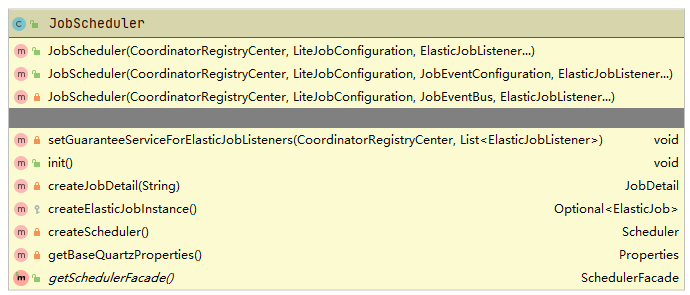
overwrite: boolean

LiteJobConfiguration[SimpleJobConfiguration[JobCoreConfiguration]]

LiteJobConfiguration[DataflowJobConfiguration[JobCoreConfiguration]]

LiteJobConfiguration[ScriptJobConfiguration[JobCoreConfiguration]]

二、JobScheduler初始化流程



1. 添加作业至本地注册表，存于内存



1. 初始化GuaranteeService。提供如下操作：

registerStart: 依次注册各个分片至guarantee/started/{shardingItemNumber}

isAllStarted: 判断是否所有分片均已启动，首先判断guarantee/started节点是否存在，再统计guarantee/started的子节点数量是否等于shardingTotalCount

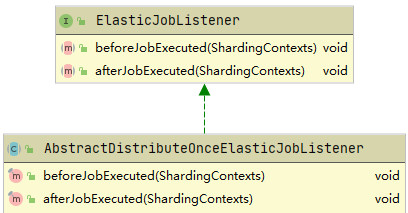
clearAllStartedInfo: 删除guarantee/started节点

registerComplete: 依次注册各个分片至guarantee/completed/{shardingItemNumber}

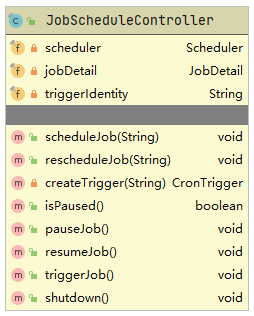
isAllCompleted: 判断是否所有分片均已执行完成，首先判断guarantee/completed节点是否存在，再统计guarantee/completed的子节点数量是否等于shardingTotalCount

clearAllCompletedInfo: 删除guarantee/completed节点

1. 将GuaranteeService绑定至AbstractDistributeOnceElasticJobListener，仅单一节点执行此监听器



beforeJobExecuted --> registerStart --> isAllStarted ? --> doBeforeJobExecutedAtLastStarted afterJobExecuted --> registerComplete --> isAllCompleted ? --> doAfterJobExecutedAtLastCompleted

1. 若/config节点不存在或开启了overwrite设置，则持久化本地LiteJobConfiguration至/config节点，
2. 更新注册表的任务分片数
3. 使用Quartz Scheduler与JobDetail初始化JobScheduleController
4. 依次启动所有Listener



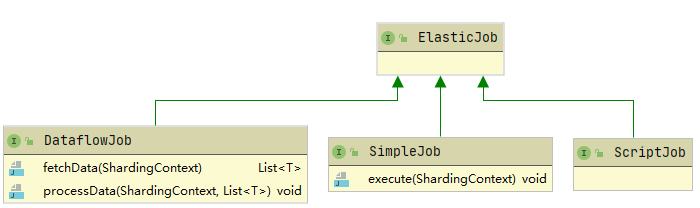
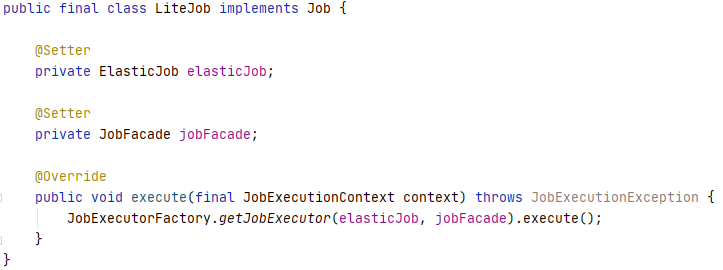
1. LeaderService选主
2. 若配置disable为true，/servers/{ip}节点写入DISABLED，表示作业未启用
3. 创建临时节点/instances/{ip}@-@{pid}
4. 创建/leader/sharding/necessary节点，表示需要重新分配分片
5. 根据配置monitorExecution选择是否开启MonitorService
6. 开启reconcileService，固定间隔执行一次，若当前为主节点，且存在掉线节点，则创建/leader/sharding/necessary节点，表示需要重新分配分片
7. 根据cron配置创建Quartz Trigger，开始执行Job

三、JobScheduler调度流程

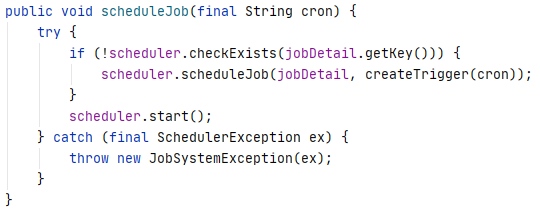
1、创建Quartz JobDetail



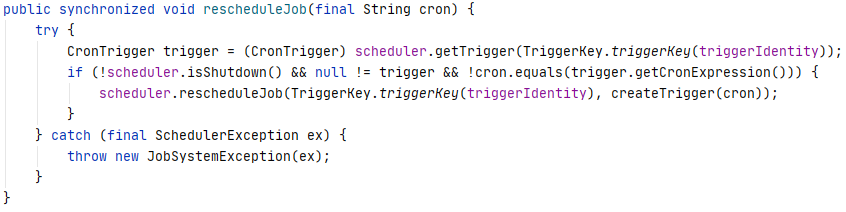
并往LiteJob中写入ElasticJob实例与JobFacade。JobFacade主要提供针对zk job和sharding相关节点的操作。LiteJob作为主任务，负责调度用户自定义任务elasticJob。



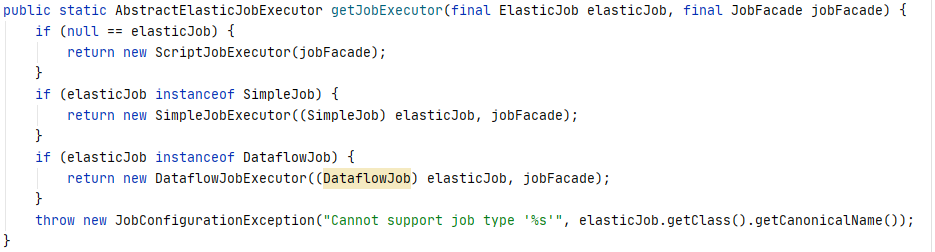
1. JobScheduleController使用Quartz Scheduler开始作业调度执行

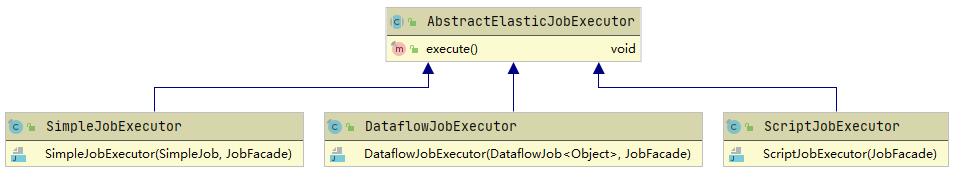


JobScheduleController同时提供了reschedule操作，可供动态修改调度规则



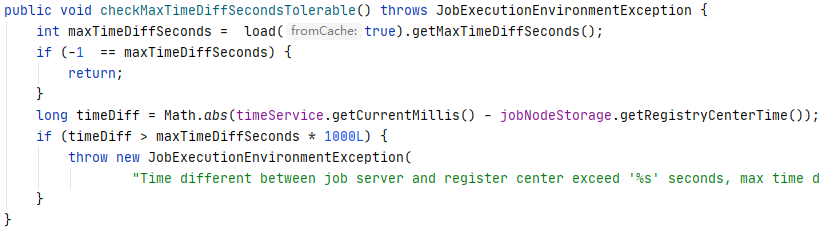
1. 获取不同类型Job对应的JobExecutor，并执行



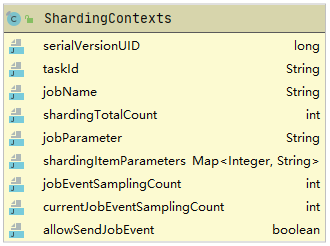


四、AbstractElasticJobExecutor调度流程

1、使用JobFacade检查本机与注册中心的时间误差秒数是否在允许范围。



1. 使用JobFacade获取任务运行上下文，ShardingContexts



JobFacade在获取ShardingContexts前首先进行failover操作。首先遍历/sharding的所有子节点，判断是否存在/sharding/{shardingItem}/failover节点，若存在，表示此分片需要进行failover操作，/sharding/{shardingItem}/failover节点内存储了执行此failover操作的instanceId。获取与本地instanceId匹配的failover分片后，构造ShardingContexts。

ShardingContexts构造过程见4.3。

若无需进行failover操作，主节点会尝试根据分片策略进行分片操作，见4.4。

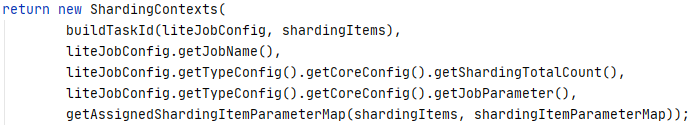
从zk获取本机实例对应的所有分片。首先遍历/sharding节点下的所有子节点，获取/sharding/{shardingItem}/instance存储的实例id，获取与本机实例id对应的分片。若获取的分片/sharding/{shardingItem}下存在failover节点，表示此分片已被失效转移，此分片需要被排除。若获取的分片/sharding/{shardingItem}下存在disable节点，表示此分片已被禁用，此分片需要被排除。通过获取的分片构造ShardingContexts。

1. 通过ExecutionContextService构造ShardingContexts



若job配置中monitorExecution打开，则遍历分片列表，若/sharding/{shardingItem}下存在running节点，表示此分片任务还在执行，需要移除此分片。

建立shardingItem与shardingItemParameter的对应关系，构建ShardingContexts。



1. 根据分片策略进行分片操作

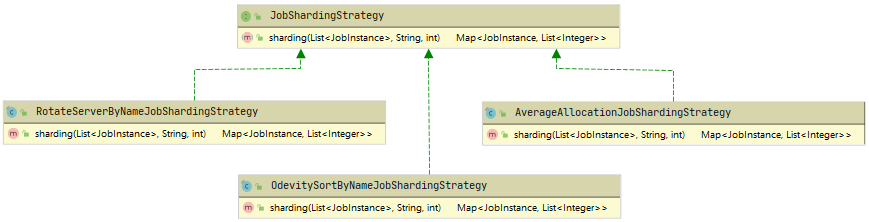
首先判断是否需要进行任务分片，若存在/leader/sharding/necessary节点且zk中记录的可用实例不为空，则需要进行分片。

若当前没有主节点，则使用LeaderLatch进行主节点选举。主节点负责任务分片，其余节点阻塞直到分片完成，分片完成后会移除/leader/sharding下的necessary节点与processing节点。

若job配置中monitorExecution打开，则主节点还需要等待/sharding下的所有分片任务执行完成。

主节点开始进行任务分片操作，并在/leader/sharding下创建processing节点，表示分片操作正在进行。接下来主分片会进行正式分片前的清理，清除/sharding/ {shardingItem}下的instance节点，若分片前配置已经更改，且新分片数变小，则删除多余的分片/sharding/ {shardingItem}。根据配置jobShardingStrategyClass获取相应分片策略执行，并将分片结果写入/sharding/{shardingItem}/instance节点中，完成后删除/leader/ sharding/necessary与/leader/sharding/processing节点。

1. 分片策略

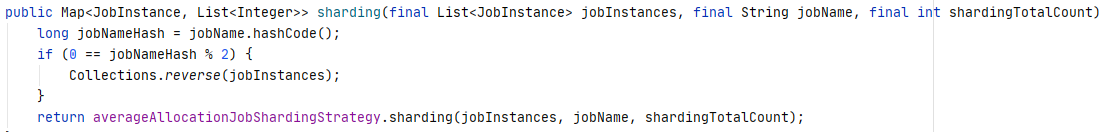


AverageAllocationJobShardingStrategy:

对于0,1,2,3共三个分片，分配给3个实例：[0,3], [1], [2]

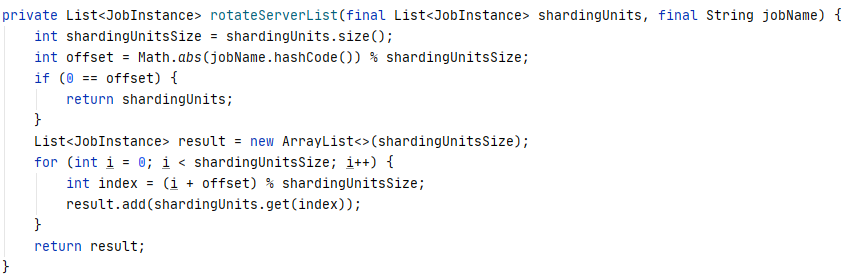
OdevitySortByNameJobShardingStrategy:

根据jobName哈希值的奇偶，对作业列表进行升降排序，之后应用AverageAllocationJob ShardingStrategy策略进行分片。



RotateServerByNameJobShardingStrategy:

通过jobName的哈希值对作业列表进行rotate操作，之后使用AverageAllocation JobShardingStrategy策略进行分片



若jobName哈希值余数为0，不该变作业列表的顺序，若余数为1，则原本[0, 1, 2]的顺序变为[1,2,0]。

1. 设置misfire分片

若任务配置中monitorExecution开启，则可监控作业运行状态，若此时仍有分片任务正在执行，即存在/sharding/{shardingItem}/running节点，则添加misfire节点/sharding/ {shardingItem}/misfire，表示此分片任务misfire。此轮任务调度结束。

1. 执行任务

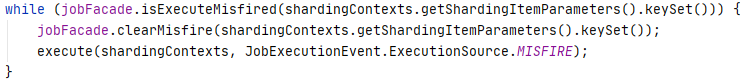
若任务配置中monitorExecution开启，创建/sharding/{shardingItem}/running节点，表示分片任务开始执行。此轮任务执行完毕后移除/sharding/{shardingItem}/running节点，若执行的是failover任务，则移除sharding/{shardingItem}/failover节点。

依次执行每个分片任务：



1. 执行misfire分片任务

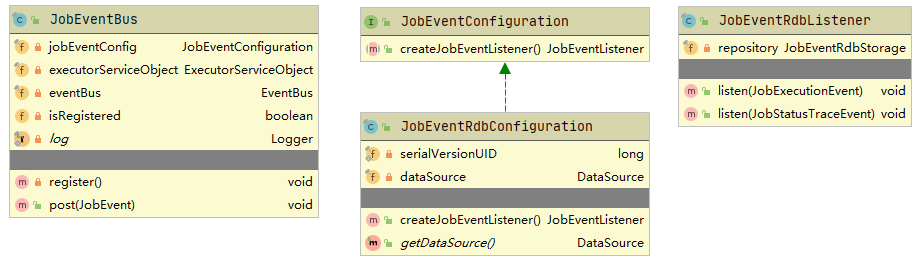
首先删除/sharding/{shardingItem}/misfire节点，再把分片任务全部执行一次。



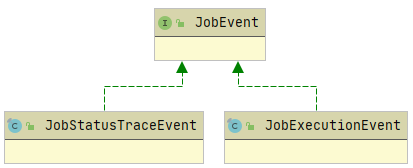
1. 执行作业分片失效转移

若存在/leader/failover/items节点，且该节点存在子节点，表示存在失效分片，需要执行分片失效转移。每个实例依次竞争/leader/failover/latch，竞争成功实例取出/leader/failover/ items下的第一个节点，获取分片编号，并删除该节点。创建/sharding/{shardingItem}/failover节点，并写入本机实例id。使用Quartz scheduler执行一次任务。

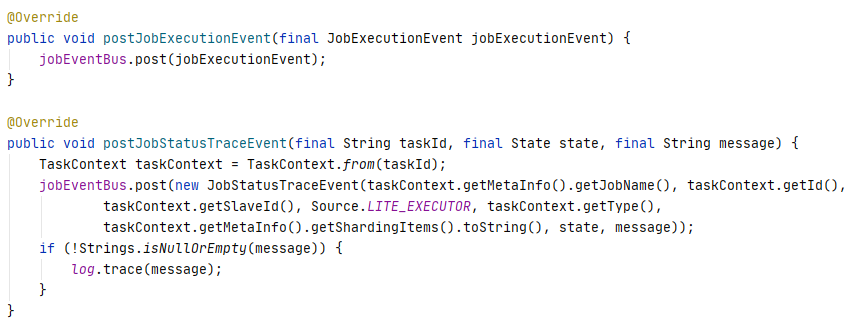
五、任务执行事件上报流程



elasticjob提供基于数据库的执行事件存储机制。JobEventBus负责事件分发和监听器注册，事件分发基于guava AsyncEventBus。JobEventConfiguration负责创建一个事件监听器，目前为JobEventRdbListener，负责往数据库内写入事件记录。目前包含两类事件：JobExecutionEvent和JobStatusTraceEvent，前者记录任务执行的最终状态，后者记录任务执行的整个流程。



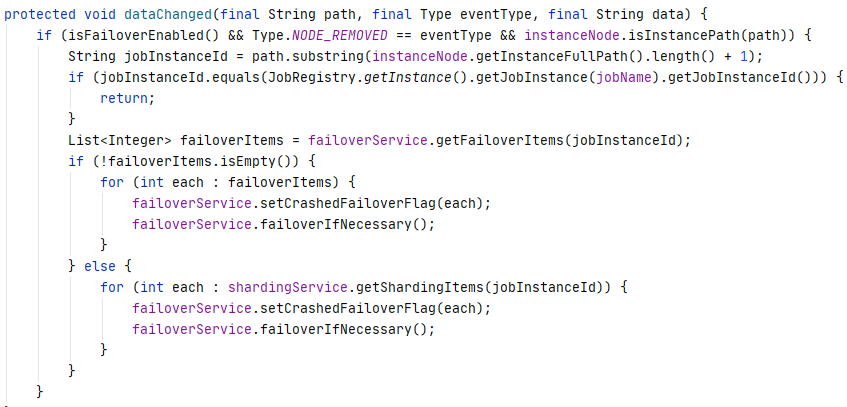
LiteJobFacade实例持有JobEventBus实例，提供发布事件接口，JobExecutor在其各个执行点发布对应事件，JobEventRdbListener负责接收并持久化这些事件。



1. Failover机制

FailoverListenerManager负责处理failover事务，注册两个监听器，分别为JobCrashedJob Listener与FailoverSettingsChangedJobListener，前者处理节点离线事件，后者处理failover配置变更事件。

1. JobCrashedJobListener，负责监听/instances下的子节点删除事件。若实例掉线，对应的临时节点/instances/{instanceId}被删除，此listener会将实例对应的所有分片添加至/leader/failover/items/ {shardingItem}，表示此分片需要执行failover操作。



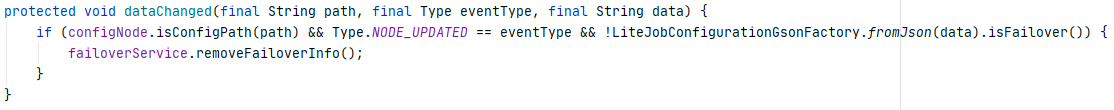
各个实例会竞争对/leader/failover/items的处理权，竞争成功的实例会获取/leader/failover/items下的第一个分片，并删除此节点/leader/failover/items/ {shardingItem}。同时创建节点/sharding/{shardingItem}/failover，并往此节点中写入本地实例id，表示此分片的failover由本地实例处理。



最后会触发一次任务，处理失效分片，见4.2。



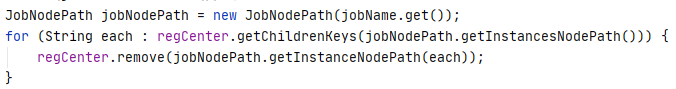
2、FailoverSettingsChangedJobListener负责监听/config节点的内容变更，若配置中failover设置被关闭，会依次移除所有sharding下的failover节点，/sharding/{shardingItem}/failover，关闭failover操作。



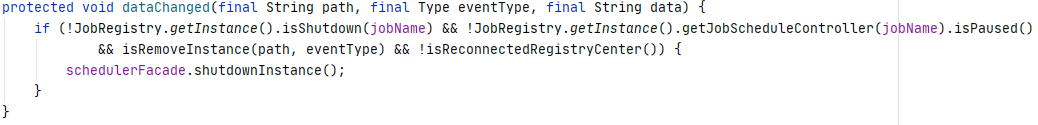
七、控制台常用操作

1、shutdown

shutdown操作会通过控制台删除/instances下的所有节点

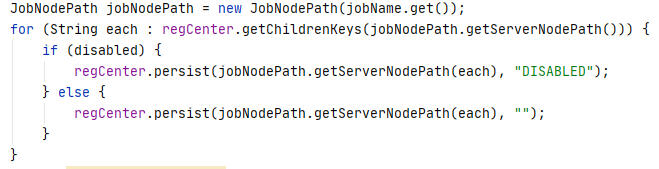


InstanceShutdownStatusJobListener监听到节点下线，且下线实例id为本地实例，会直接停掉本地的Quartz定时任务，若本地实例为主，会删掉节点/leader/election/instance让出主



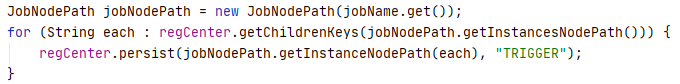
2、disable

disable操作会通过控制台往/servers下的所有子节点/servers/{ip}写入DISABLED标志，表示此ip上的所有实例均被禁用，客户端进行作业调度时，读取到DISABLED标志，会停止获取作业分片，跳过此次调度。

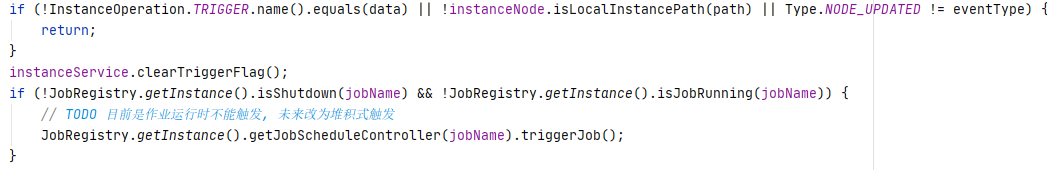


1. trigger

trigger操作会立即触发一次任务，控制台往/instances下的所有节点/instances/{id}写入TRIGGER标志，

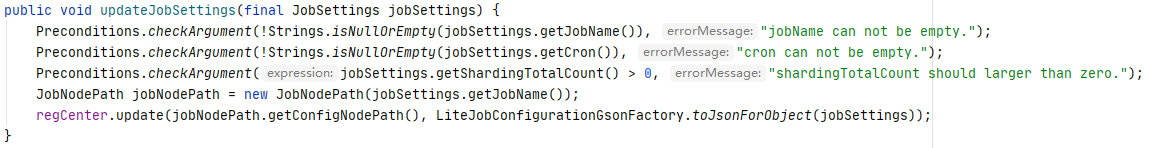


JobTriggerStatusJobListener会监听节点/instances/{id}的更新事件，若更新内容为TRIGGER且实例id为本地实例id，则立即触发一次作业调度

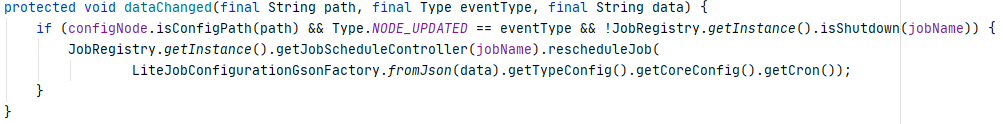


1. 作业配置更新

控制台往/config节点写入更改后的配置，



CronSettingAndJobEventChangedJobListener负责监听/config节点变化，并重新调度任务



1. ZK数据结构

---**config**

---**instances** (作业运行实例信息，子节点是当前作业运行实例的id)

-----|{ip}@--@{pid} (临时节点)

-----|{ip}@--@{pid}

---**shardings**

-----|**0**(分片编号)

----------|instance(执行该分片项的作业运行实例id)

----------|running(临时节点，任务是否正在执行, 仅配置monitorExecution时有效)

----------|failover(临时节点，如果该分片项被失效转移分配给其他作业服务器，则此节点值记录执行此分片的实例id)

----------|misfire(表示有任务被错过，需要重新执行)

----------|disabled(表示此分片被禁用)

---**servers**

-----|{ip}

-----|{ip}

---**leader**

-----|**election**

----------|instance(当前主实例id)

----------|latch(主实例选举分布式锁)

-----|**sharding**

----------|necessary(表示需要重新分片，实例上下线时会开启此标志)

----------|processing(分片正在进行中，分片完成后会删除此节点)

-----|**failover**

----------|**items**

---------------|latch(各实例竞争失效分片时占用的分布式锁)

---------------|0(分片编号，表示此分片已失效，实例竞争到此分片后会删除此节点)

---------------|1(分片编号)