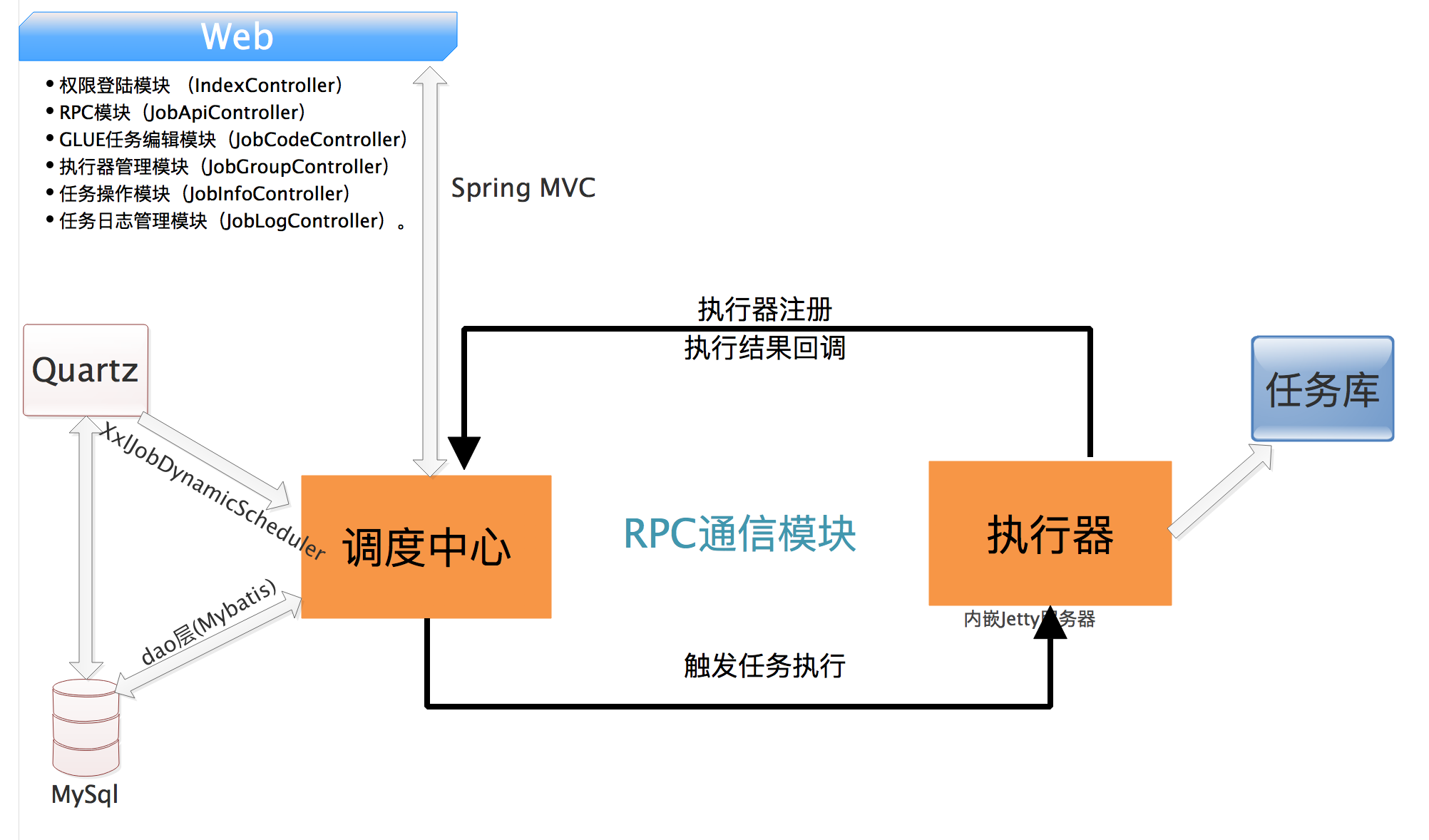
xxl-job V2.0.2 源码分析

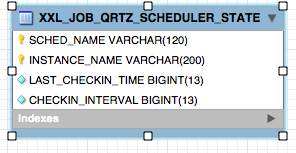
1. 整体架构设计



1. 数据库表分析
2. Quartz 原生11张表分析

（1） 调度器(Scheduler)相关

* qrtz\_scheduler\_state：存储所有节点的Scheduler，会定期检查Scheduler是否失效。



(2) 触发器(Trigger)相关

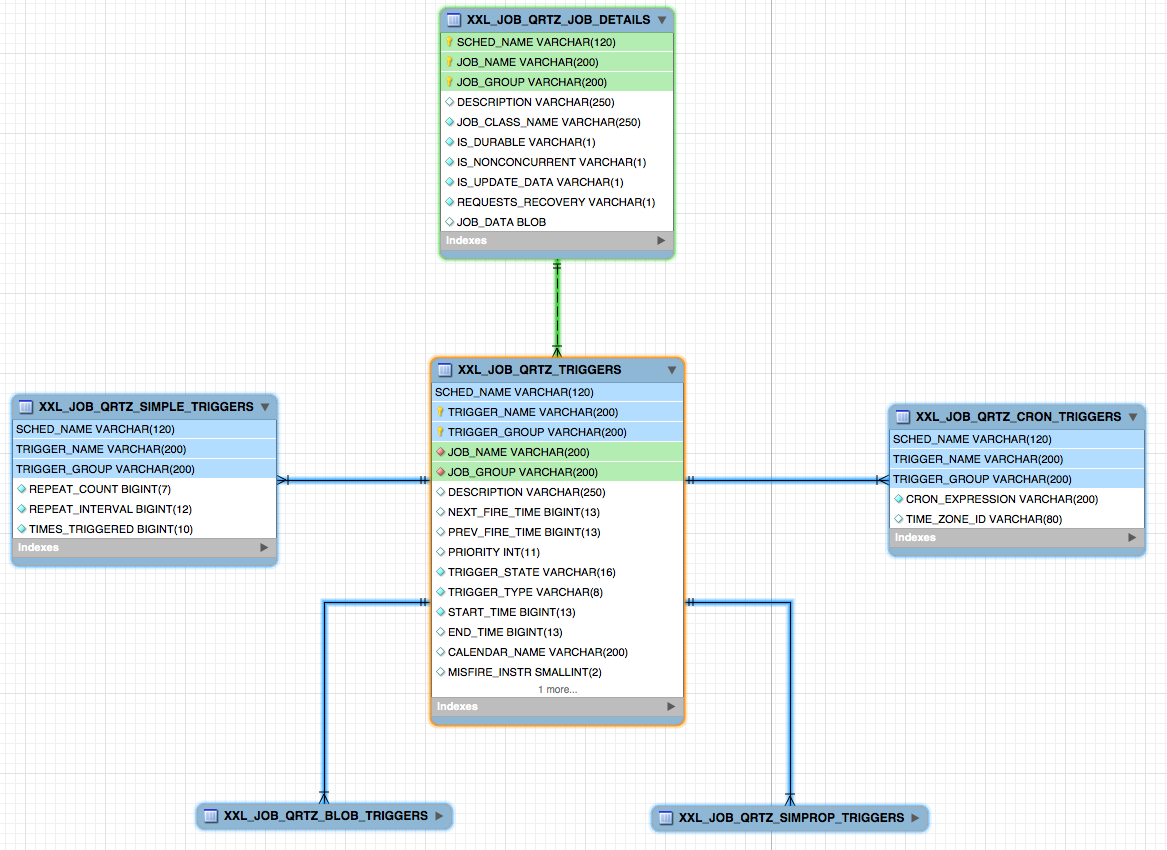
Trigger是用于定义调度时间的元素，即按照什么时间规则去执行什么任务。

Quartz提供的Trigger包括：SimpleTrigger(常用), CronTrigger(常用), DailyTimeIntervalTrigger  和 CalendarIntervalTrigger.

* qrtz\_triggers:存储定义的trigger,并关联job,  即定义：以什么时间规则（trigger）执行什么任务（jobDetails）.
* qrtz\_simple\_triggers：存储SimpleTrigger
* qrtz\_cron\_triggers：存储CronTrigger
* qrtz\_simprop\_triggers：存储DailyTimeIntervalTrigger  和CalendarIntervalTrigger
* qrtz\_blob\_triggers：存储自定义的Trigger
* qrtz\_fired\_triggers：存储已经触发的trigger信息
* qrtz\_paused\_trigger\_grps：存储暂停的trigger 信息
* qrtz\_calendars：日历功能，可以定义一个时间段，可以控制触发器在这个时间段内触发或者不触发

（3） 任务（Job）相关

* qrtz\_job\_details：存储 jobDetails 信息，即每一个已经配置的job 的具体信息



(4) 锁相关

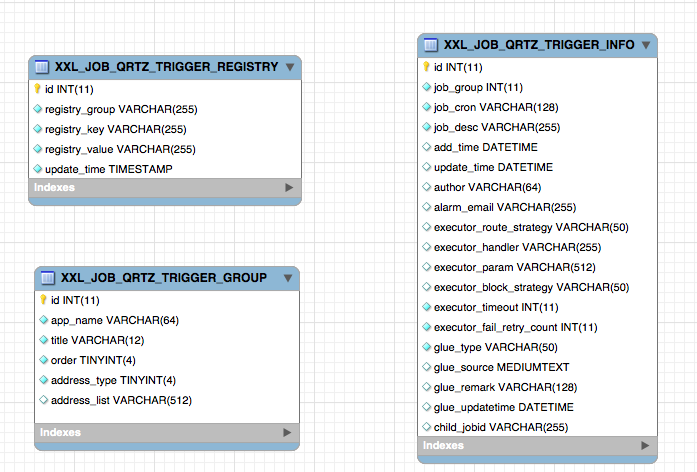
* qrtz\_locks：Quartz提供的锁表，为多个节点调度提供分布式锁，实现分布式调度

1. Xxl-job 新增的5张表

* xxl\_job\_qrtz\_trigger\_group：执行器组信息表，维护执行器组的相关信息。

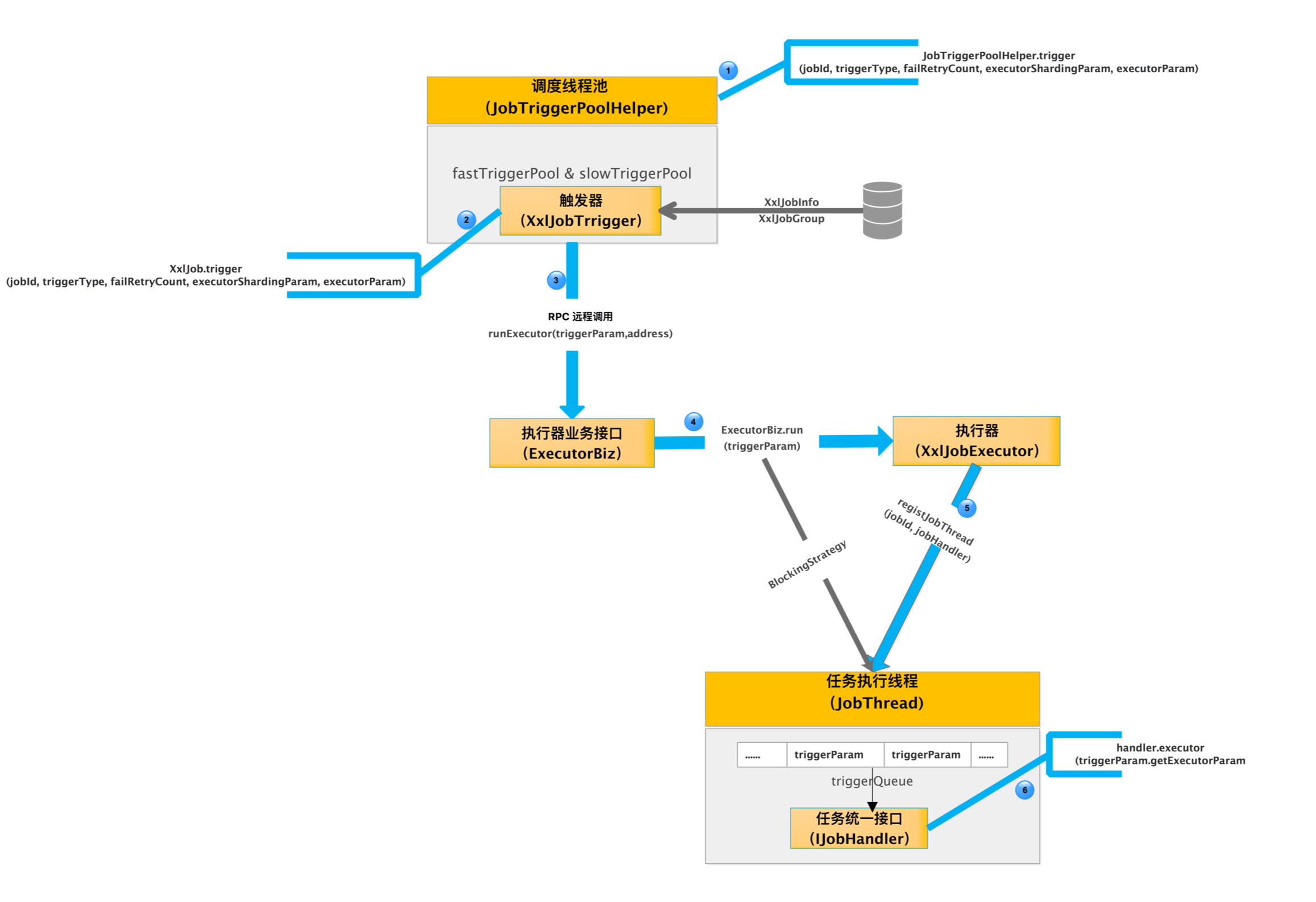
address\_type = 0: 表示自动注册；address\_type = 1：表示手动录入集群地址。

* xxl\_job\_qrtz\_trigger\_registry：执行器注册表，维护在线的执行器和调度中心机器地址信息。执行器在进行“任务注册”时将会周期性维护一条注册记录，即 “机器地址” 和 “执行器组” 的绑定关系，“调度中心”从而可以动态感知每个执行器组在线的机器列表。
* xxl\_job\_qrtz\_trigger\_info：任务调度信息扩展表，用于保存 XXL-JOB 调度任务的扩展信息，如任务分组、任务名、机器地址、执行器、执行入参和报警邮件等
* xxl\_job\_qrtz\_log：调度日志表： 用于保存 XXL-JOB 任务调度的历史信息，如调度结果、执行结果、调度入参、调度机器和执行器等等
* xxl\_job\_qrtz\_logglue：任务 GLUE 日志：用于保存 GLUE 更新历史，用于支持 GLUE 的版本回溯功能



1. 一次手动触发任务执行的过程

用户继承 IJobHandler接口， 创建任务 DemoJobHandler。 然后在Web页面中配置DemoJobHandler 调度相关参数，这些参数保存在info表中，对应数据模型 XxlJobInfo。点击“执行”按钮，会执行 JobInfoController.trigger (int jobId, executorParam) 方法，executorParam 是任务执行所需的入参，不同的任务参数类型和个数各不相同，这个由用户根据具体任务给出。



1. JobTriggerPoolHelper.trigger (int jobId, TriggerTypeEnum triggerType, int failRetryCount, String executorShardingParam, String executorParam)，将调度信息加入调度线程池。

* 为该任务选择快慢线程池（fastTriggerPool slowTriggerPool）
* 在选择好的线程池里执行触发任务执行XxlJobTrigger.trigger (int jobId, TriggerTypeEnum triggerType, int failRetryCount, String executorShardingParam, String executorParam)

1. XxlJobTrigger.trigger (int jobId, TriggerTypeEnum triggerType, int failRetryCount, String executorShardingParam, String executorParam)，触发器触发任务执行。

* 根据 jobId 从数据库表中读取XxlJobInfo（任务调度扩展信息）和 XxlJobGroup（任务所属执行器信息）
* 根据路由策略是否为“分片广播（SHARDING\_BROADCAST）分为两种情况，传入不同的分片参数
* 设置触发参数TriggerParam，获取最终执行该任务的机器地址address
* PRC触发远程执行器执行 XxlJobTrigger.runExecutor (TriggerParam triggerParam, String address)

1. XxlJobTrigger.runExecutor (TriggerParam triggerParam, String address) ,远程触发执行器执行。

* 根据address 创建执行器客户端服务 ExecutorBiz
* 执行 ExecutorBiz.run (triggerParam)

1. ExecutorBiz.run (triggerParam)，远程执行器执行任务。

* 区分不同的任务运行模式，确认 jobThread 和jobHandler
* 处理阻塞处理策略 （同一个任务被频繁调度）,将triggerParam放入队列中
* 执行器（XxlJobExecutor）注册任务执行线程 XxlJobExecutor.registJobThread(triggerParam.getJobId(), jobHandler, removeOldReason)

1. XxlJobExecutor.registJobThread(triggerParam.getJobId(), jobHandler, removeOldReason)

* 启动线程JobThread (int jobId, IJobHandler jobHandler) ,该线程有个 triggerQueue,存放 triggerParam.jobThread从triggerQueue中读取triggerParam，并执行用户定义的任务 handler.execute (triggerParam.getExecutorParam)

1. handler.execute (triggerParam.getExecutorParam) 执行任务。
2. 分片方案总结