## EasySchedulerr大数据调度系统架构分析

## 导语

EasyScheduler是易观平台自主研发的大数据分布式调度系统。主要解决数据研发ETL 错综复杂的依赖关系，而不能直观监控任务健康状态等问题。EasyScheduler以DAG流式的方式将Task组装起来，可实时监控任务的运行状态，同时支持重试、从指定节点恢复失败、暂停及Kill任务等操作。

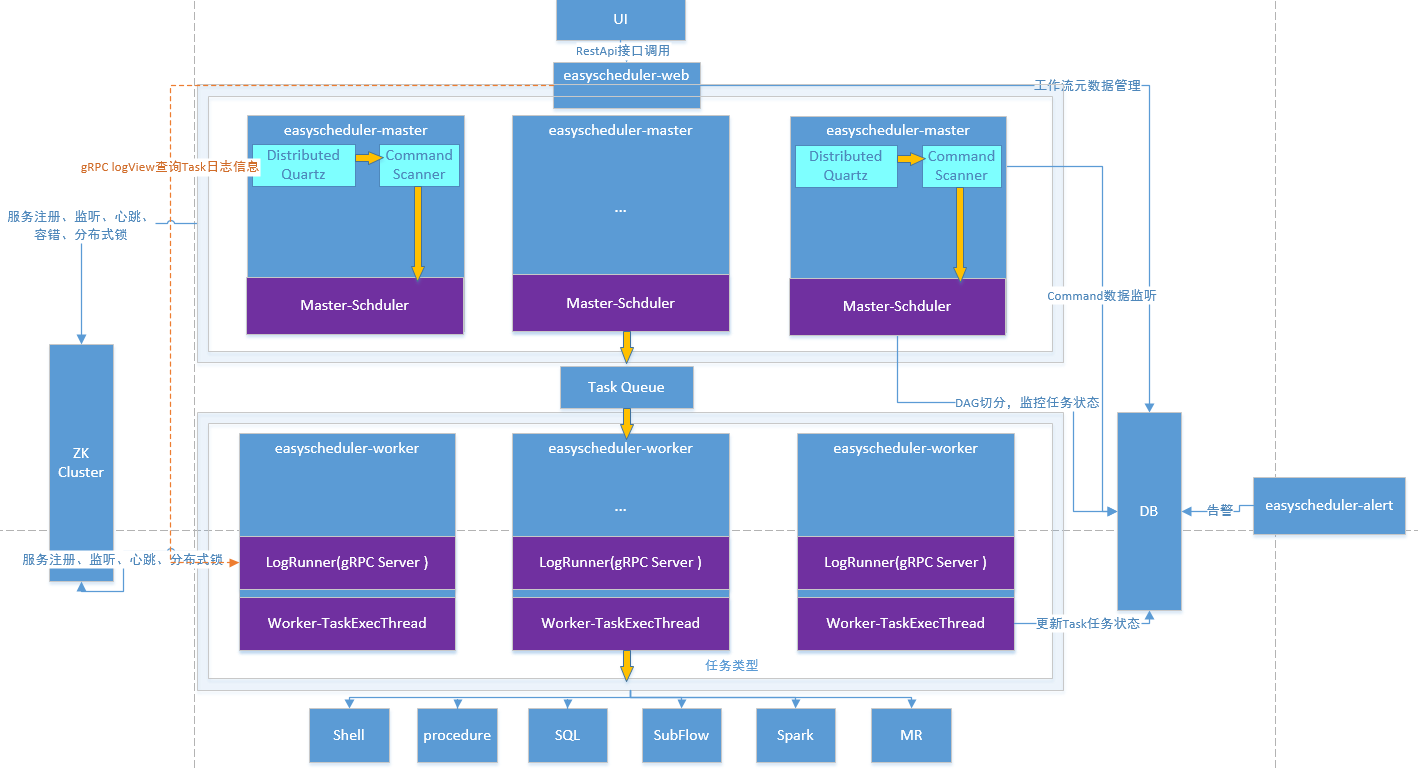
## 背景

任务调度系统在大数据平台当中是一个核心的基础设施，由于数据处理流程常常具有很长的依赖链条，因此依赖单机的crontab等单纯依赖时间调度的方式，往往存在很大的弊端，如依赖不清晰，出错难以查找等问题，因此，我们调研了市面上流行的调度系统



鉴于易观日处理数据30TB，复杂的 ETL依赖关系，我们开发了易观自己的大数据分布式调度系统EasyScheduler。

## 总架构设计



EasyScheduler设计围绕四个服务展开，UI、Web、Server和Alert。

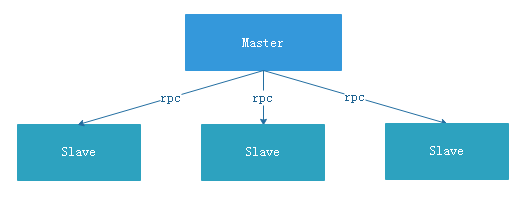
* UI : 使用易观封装的Vue及jsplumb组件开发
* Web：使用SpringBoot提供Rest Api和UI分离交互
* Server : Scheduler调度及分布式任务执行引擎
* Alert：告警微服务

以下将详细介绍Server的设计思想和遇到的问题

## 去中心化vs 中心化

### 中心化思想

中心化的设计理念比较简单，分布式集群中的节点按照角色分工，大体上分为两种角色：

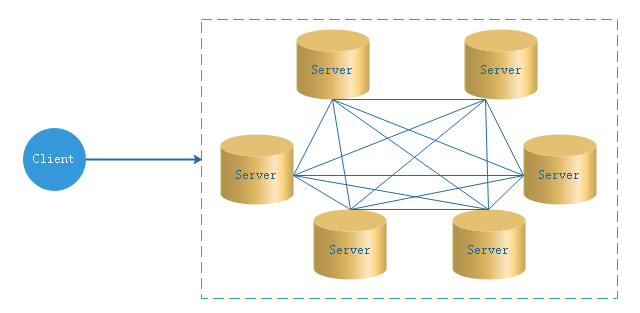


* Master的角色主要负责任务分发并监督Slave的健康状态，可以动态的将任务均衡到Slave上，以致Slave节点不至于“忙死”或”闲死”的状态。
* Worker的角色主要负责任务的执行工作并维护和Master的心跳，以便Master可以分配任务给Slave。

中心化思想设计存在的问题：

* 一旦Master出现了问题，则群龙无首，整个集群就崩溃。为了解决这个问题，大多数Master/Slave架构模式都采用了主备Master的设计方案，可以是热备或者冷备，也可以是自动切换或手动切换，而且越来越多的新系统都开始具备自动选举切换Master的能力,以提升系统的可用性。
* 另外一个问题是如果Scheduler在Master上，虽然可以支持一个DAG中不同的任务运行在不同的机器上，但是会产生Master的过负载。如果Scheduler在Slave上，则一个DAG中所有的任务都只能在某一台机器上进行作业提交，则并行任务比较多的时候，Slave的压力可能会比较大。

### 去中心化



去中心化设计里，通常没有Master/Slave的概念，所有的角色都是一样的，地位是平等的，全球互联网就是一个典型的去中心化的分布式系统，联网的任意节点设备down机，都只会影响很小范围的功能。

去中心化设计的核心设计在于整个分布式系统中不存在一个区别于其他节点的”管理者”，因此不存在单点故障问题。但由于不存在” 管理者”节点所以每个节点都需要跟其他节点通信才得到必须要的机器信息，而分布式系统通信的不可靠行，则大大增加了上述功能的实现难度。

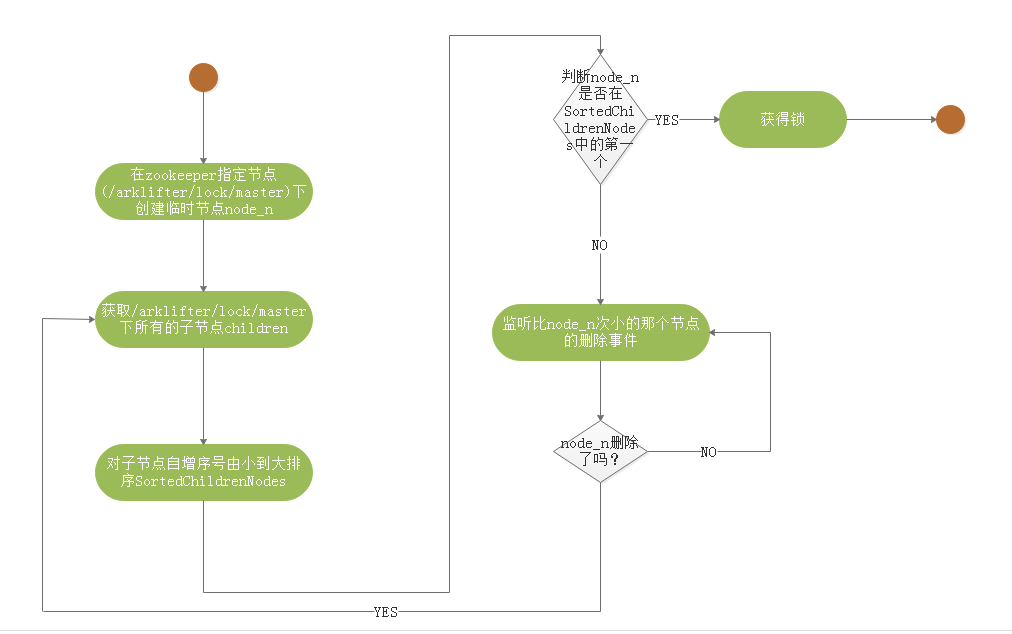
实际上，真正去中心化的分布式系统并不多见。反而动态中心化分布式系统正在不断涌出。在这种架构下，集群中的管理者是被动态选择出来的，而不是预置的，并且集群在发生故障的时候，集群的节点会自发的举行"会议"选举新的"管理者"主持工作。最典型的案例就是ZooKeeper及Go语言实现的Etcd。

EasyScheduler的去中心化是Master/Worker注册到Zookeeper中，实现Master集群和Worker集群无中心，并使用Zookeeper分布式锁来选举其中的一台Master或Worker为“管理者”来执行任务。

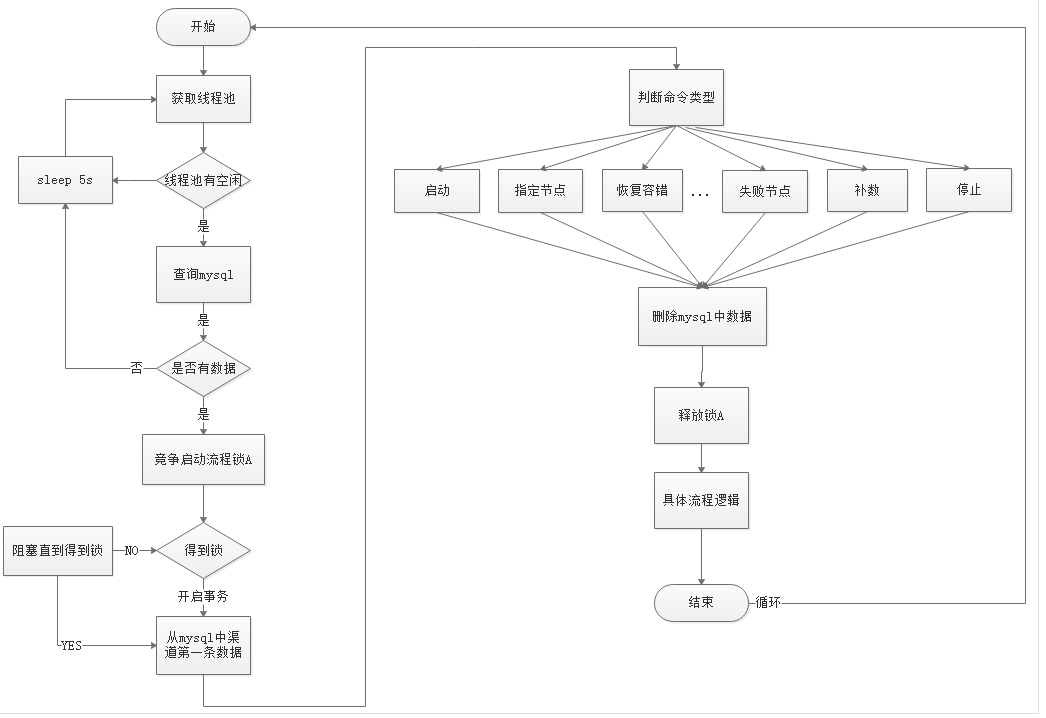
## 分布式锁实践

EasyScheduler使用Zookeeper分布式锁来实现同一时刻只有一台Master执行Scheduler，或者只有一台Worker执行任务的提交。

获取分布式锁的核心流程算法如下：

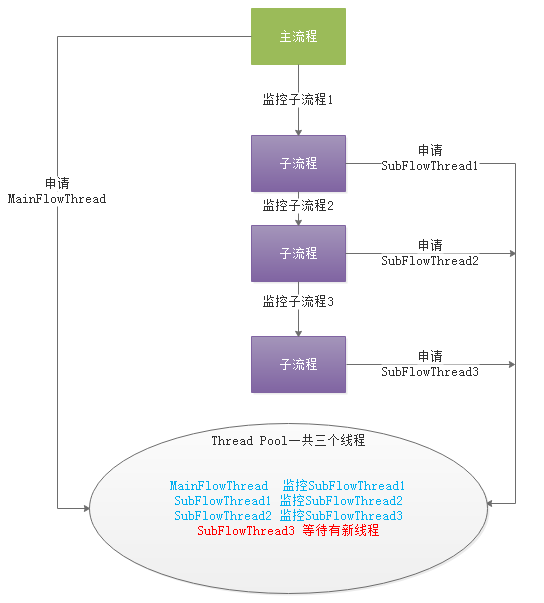


EasyScheduler中Scheduler线程分布式锁实现流程图：



## 线程不足，循环等待问题

* 如果一个DAG中没有子流程，则如果Command中的数据条数大于线程池设置的阈值，则直接流程等待或失败。
* 如果一个大的DAG中嵌套了很多子流程，如下图：



则会产生“死等”状态。MainFlowThread等待SubFlowThread1结束，

SubFlowThread1等待SubFlowThread2结束，SubFlowThread2等待SubFlowThread3结束，而SubFlowThread3等待线程池有新线程，则整个DAG流程不能结束，从而其中的线程也不能释放。这样就形成的子父流程循环等待的状态。此时除非启动新的Master来增加线程来打破这样的”僵局”，否则调度集群将不能再使用。

对于启动新Master来打破僵局，似乎有点差强人意，于是我们提出了以下三种方案来降低这种风险：

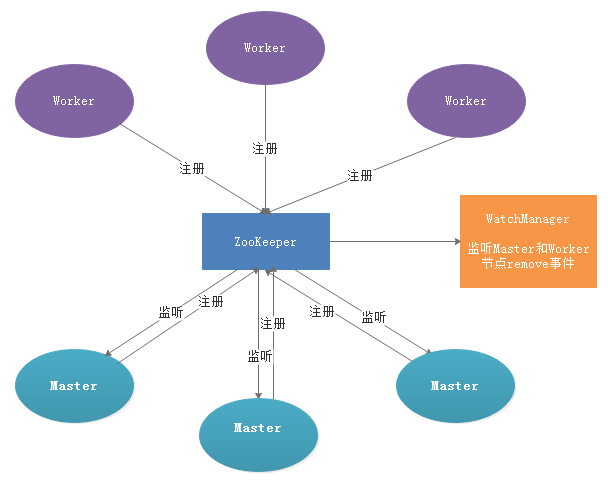
1. 计算所有Master的线程总和，然后对每一个DAG需要计算其需要的线程数，也就是在DAG流程执行之前做预计算。因为是多Master线程池，所以总线程数不太可能实时获取。
2. 对单Master线程池进行判断，如果线程池已经满了，则让线程直接失败。
3. 增加一种资源不足的Command类型，如果线程池不足，则将主流程挂起。这样线程池就有了新的线程，可以让资源不足挂起的流程重新唤醒执行。

注意：Master Scheduler线程在获取Command的时候是FIFO的方式执行的。

于是我们选择了第三种方式来解决线程不足的问题。

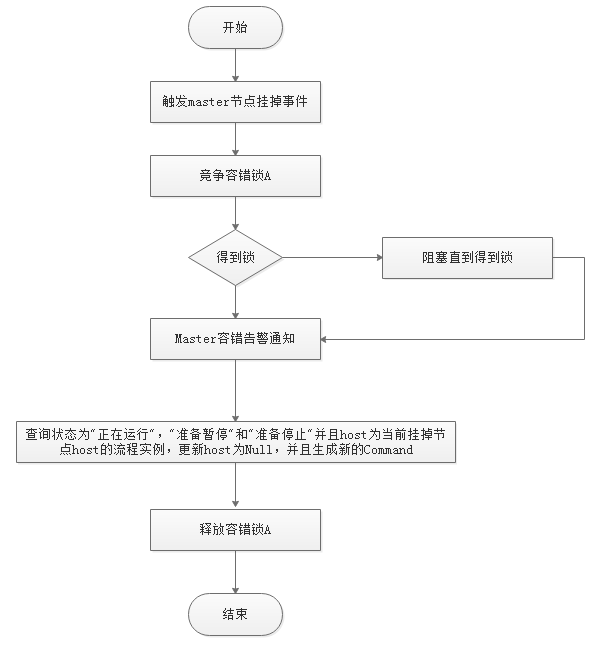
## 容错设计

EasyScheduler容错设计依赖于Zookeeper的Watcher机制，实现原理如图：



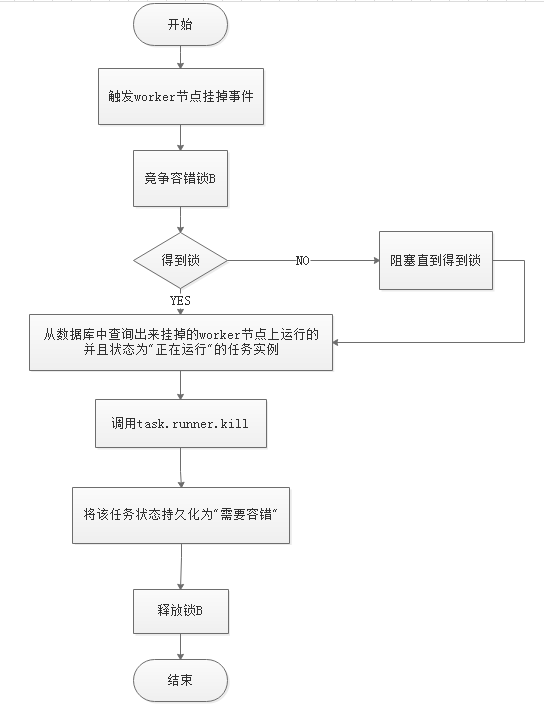
Master监控其他Master和Worker的目录，如果监听到remove事件，则会根据具体的业务逻辑进行流程实例容错或者任务实例容错。

Master容错流程图：



ZooKeeper Master容错完成之后则重新由EasyScheduler中Scheduler线程调度，遍历 DAG 找到”正在运行”和“提交成功”的任务，对”正在运行”的任务监控其任务实例的状态，对”提交成功”的任务需要判断Task Queue中是否已经存在，如果存在则同样监控任务实例的状态，如果不存在则重新提交任务实例。

Worker容错流程图：



Master Scheduler线程一旦发现任务实例为” 需要容错”状态，则接管任务并进行重新提交。

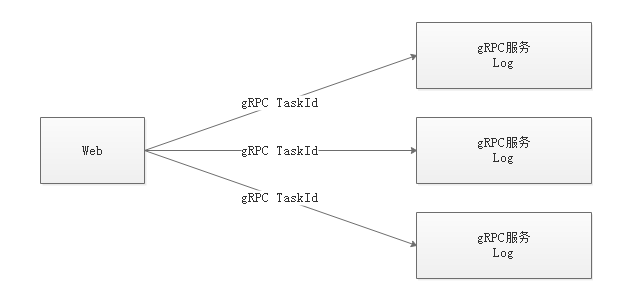
注意：由于” 网络抖动”可能会使得节点短时间内失去和zk的心跳，从而发生节点的remove事件。对于这种情况，我们使用最简单的方式，那就是节点一旦和zk发生超时连接，则直接将Master或Worker服务停掉。

## Logback和gRPC实现日志访问

由于Web和Worker不一定在同一台机器上，所以查看日志不能像查询本地文件那样。有两种方案：

* 将日志放到ES搜索引擎上
* 通过gRPC通信获取远程日志信息

介于考虑到尽可能的EasyScheduler的轻量级性，所以选择了gRPC实现远程访问日志信息。



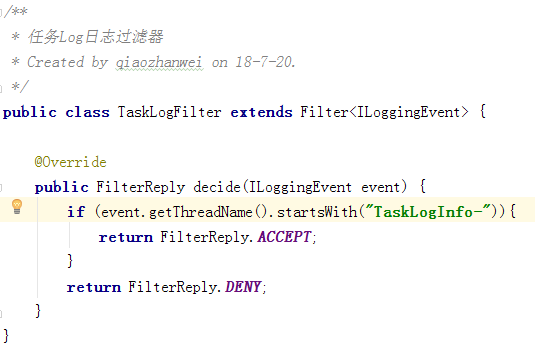
我们使用自定义Logback的FileAppender和Filter功能，实现每个任务实例生成一个日志文件。

FileAppender实现如下：



以…/流程定义id/流程实例id/任务实例id.log的形式生成日志。

过滤匹配以TaskLogInfo开始的线程名称：



## 总结

本章从调度出发，介绍了易观自主研发的大数据分布式调度系统。

EasyScheduler在易观数据平台起着中流砥柱的作用。本章着重介绍了EasyScheduler的架构原理及实现思路。