# 百万级任务调度系统实践

兴盛优选 / 陈奉刚

InfoQ

# 精彩继续！
# 更多一线大厂前沿技术案例

广州站

QCon

全球软件开发大会

时间：2022年7月31-8月1日

地点：广州·万富希尔顿酒店

扫码查看大会详情>>

北京站

GMTC

全球大前端技术大会

时间：2022年8月

地点：北京·国际会议中心

扫码查看大会详情>>

北京站

QCon

全球软件开发大会

时间：2022年9月

地点：北京·国际会议中心

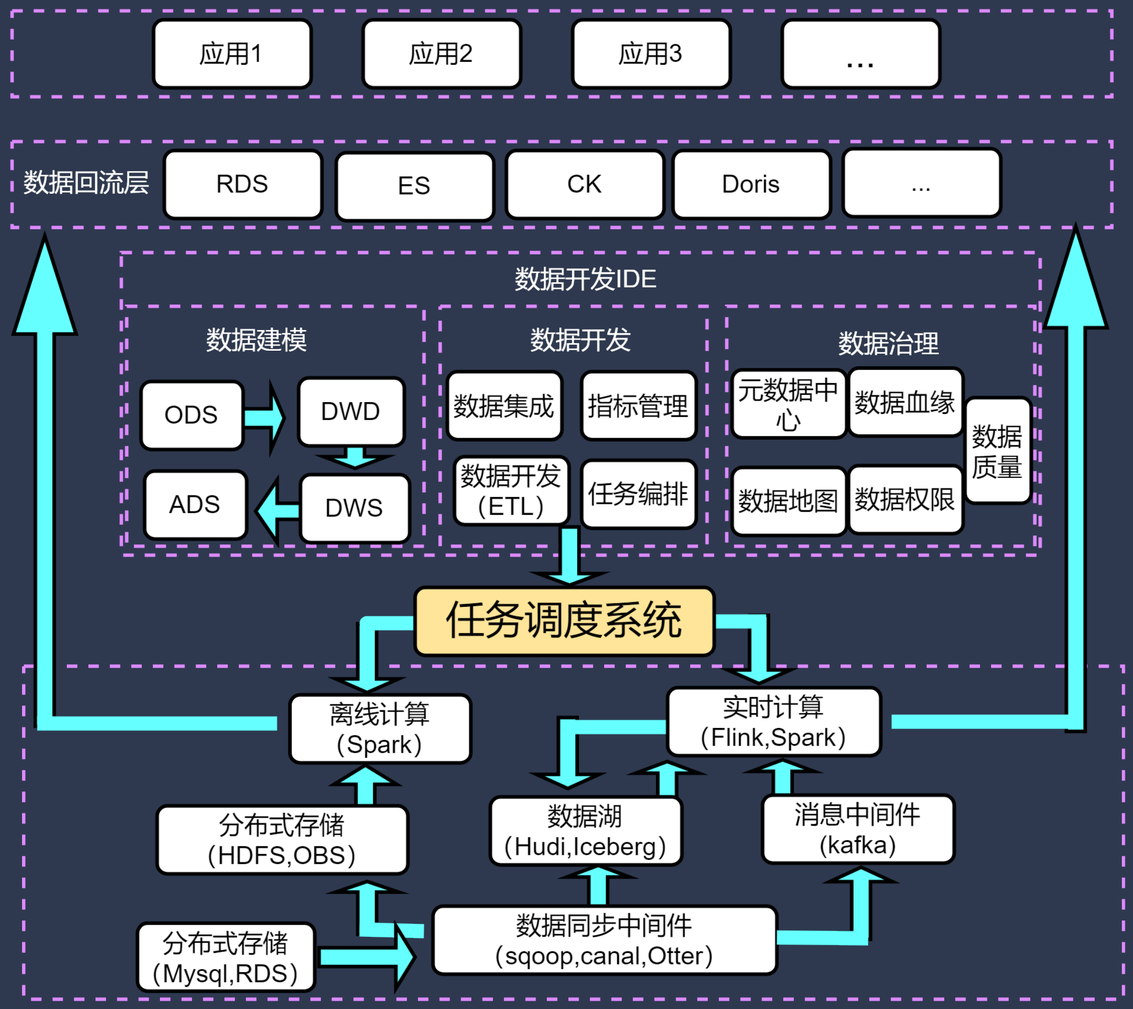
扫码查看大会详情>>

# 目录


# 任务调度的背景&系统挑战

- 兴盛优选任务调度系统的设计背景
- 系统的目标和挑战
- 开源任务调度系统的分析

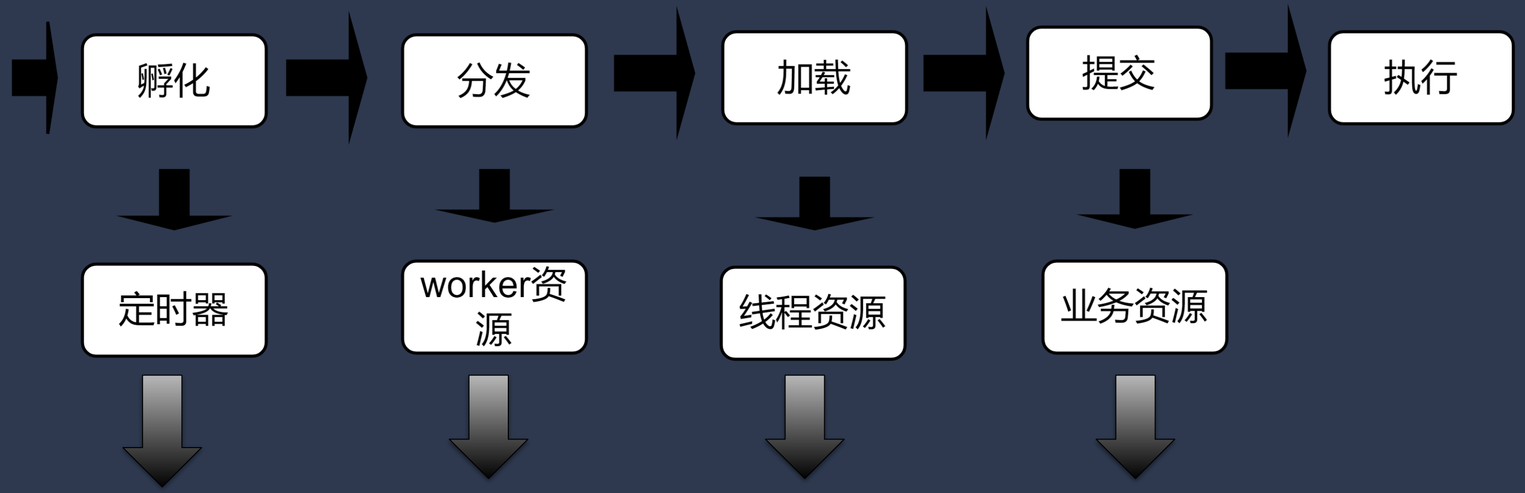
# 01 兴盛优选任务调度的背景




## 背景

- 随着业务发展，任务急剧增加，原有的调度出现明显的延时和调度卡死的情况，经常出现“起夜”现象，不得不重新规划调度。
- 为了简化大数据运维，需统一管控公司的大数据任务(包括离线调度任务，实时运行任务等)；需统一管控数据权限，Yarn队列资源。
- 需要解决流批之间的依赖关系，比如销售预测场景中，需基于每小时的商品销售量来推测当天的该商品销售量。因为无法预知实时任务数据是否有延时，因此推测任务(离线任务)的触发时间成为难点

# 01 并发情况下任务的精准调度




## 挑战点

- 任务数量大，怎样提升并发能力，保障所有的任务都能正确调度
- 任务从孵化到执行流程繁多，影响大，怎么保障准确准时调度
- 资源不足怎么保障高优业务

# 01 异常情况下的Exactly once

## 集群是不稳定的

不可预期节点故障

快速的版本迭代

随时bug修复

弹性资源的回收

… 其他

## 业务无法保障100%的幂等性

at-least-once

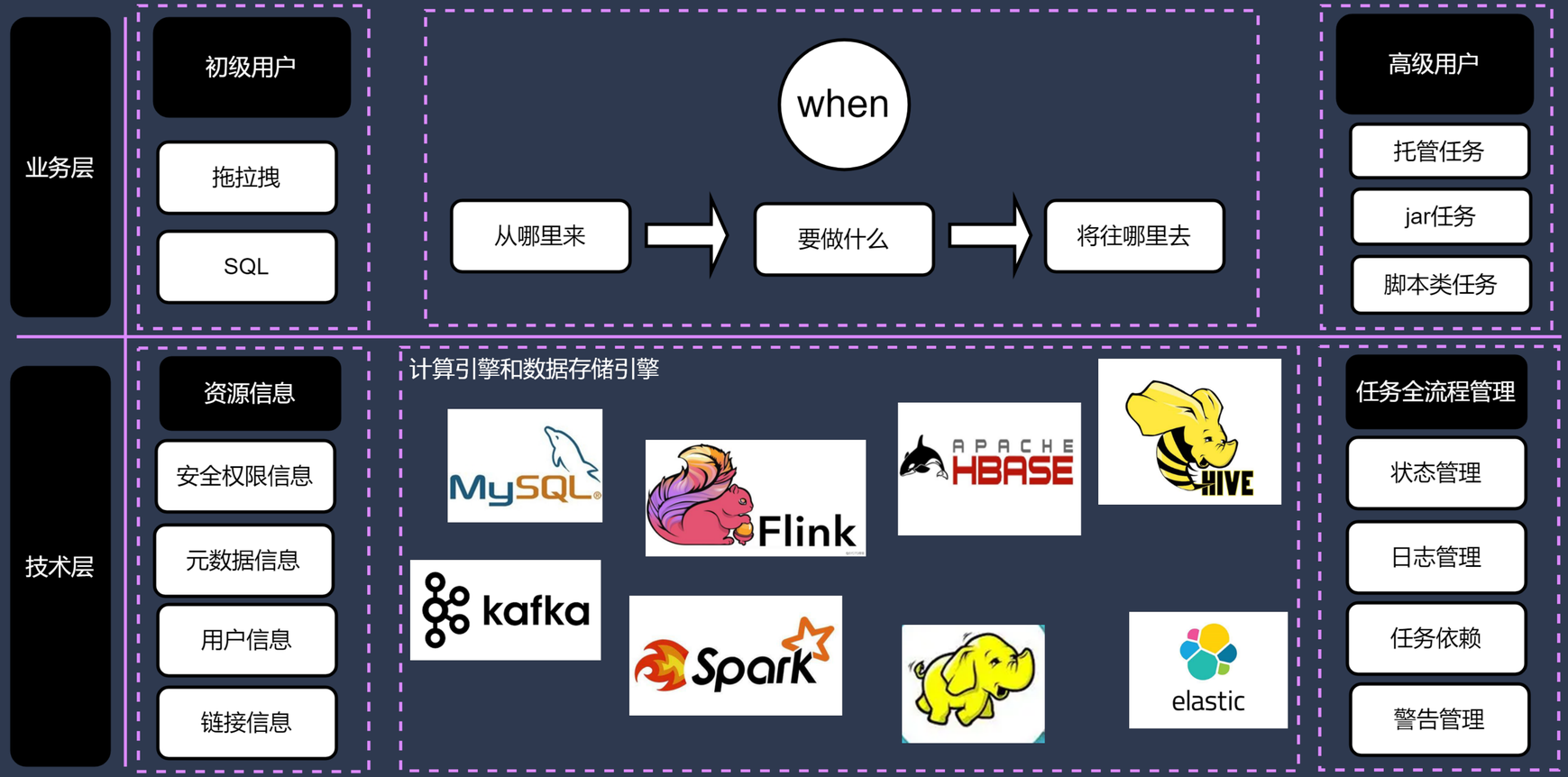
exactly-once

## 挑战

故障自动转移

尽可能“断点续传”

# 01 多类型任务情况下的易用性




## 目标

- 多种异构计算引擎的常用算子图形化支持拖拉拽实现任务快速的生成和编排
- 提供API被其他系统集成简化任务管理

## 挑战点

- 抽象化技术
- 体系化平台
- 标准化业务使用流程

# 01 常见开源调度系统的优缺点分析

**Airflow**

**脚本式开发**

**社区活跃，功能齐全**

内部采用扫描式加载调度任务，存在明显的延时；运行时解析任务脚本，处理能力有限；另外Airflow没有对外提供API接口，需要将代码上传到调度目录才能加载。

**Elastic-job**

**订阅式调度**

**直接嵌入代码，能实时获取结果**

用户注入任务，由调度系统通过消息系统定时触发业务逻辑，所以比较适合Java，Python，Shell等可编程的任务。

**Azkaban**

**配置化调度**

**与Hadoop生态圈结合紧密**

只有DAG内部依赖，没有DAG之间的依赖；UI界面无法直接添加任务。

**Dolphin Scheduler**

**无法实现流批一体**

**第一个完整依赖关系的图形化调度**

分布式架构；支持DAG之间依赖；支持资源组；提供依赖完整的DAG为基础的调度体系。但目前架构无法实现流批任务统一管理

# 01 基于DolphinScheduler自研的原因

## DolphinScheduler的优势

可视化的操作，能够完美解决workflow之间的依赖

分布式的架构

提供了大量的开箱即用的算子

基于DAG为单位调度

## 业务目标

实现流批统一集群管理，建立批流依赖关系

高并发下任务准时调度

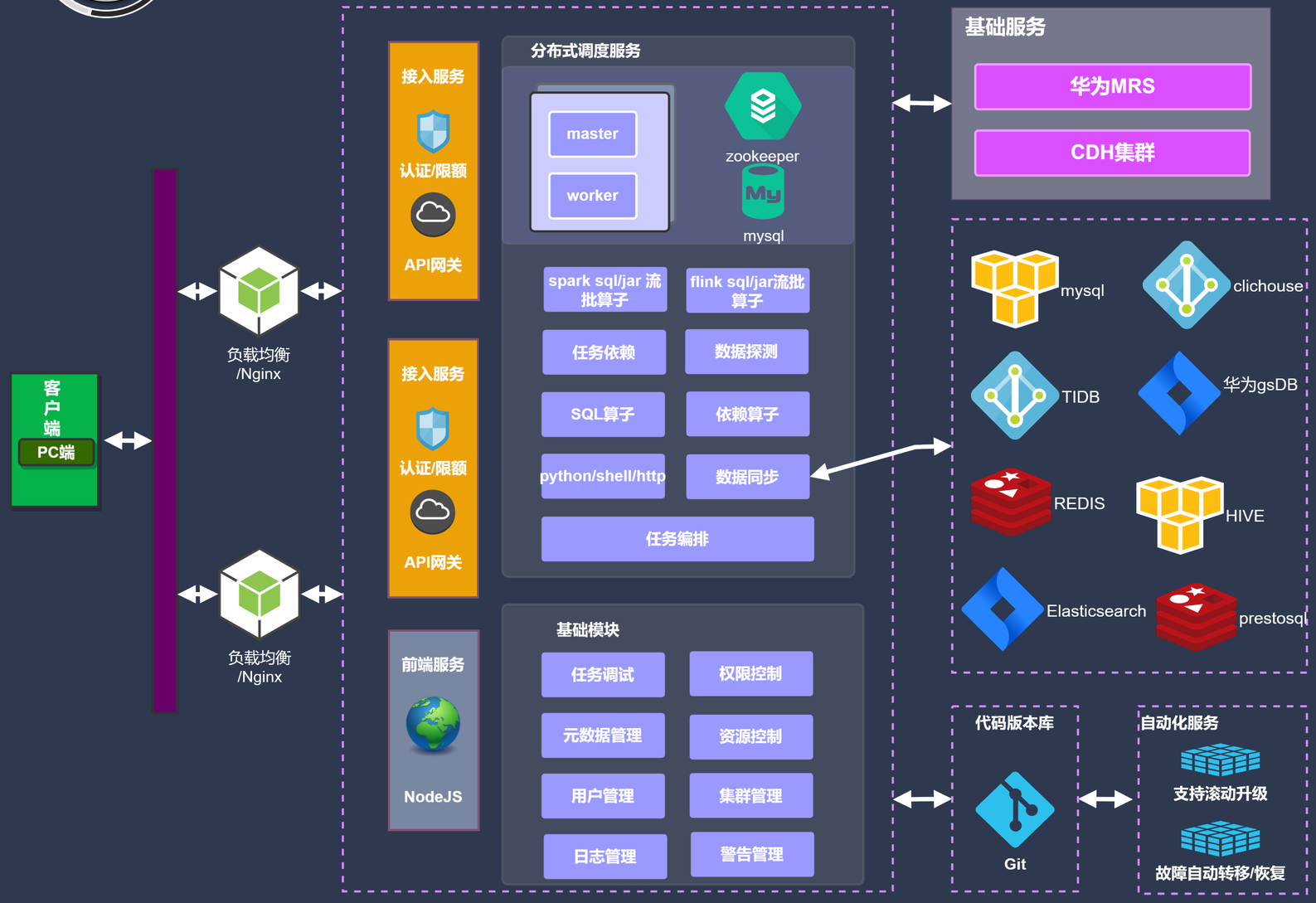
故障的优雅处理以及优雅升级

**基于开源自研**


# 分布式任务调度系统的实践

- 整体设计
- 并发增强
- 调度准时性增强
- 优雅故障处理
- 算子优化

# 02 太阳神调度系统的整体设计




## 核心模块

**master**：提供任务调度，故障处理等；

**worker**：提供任务实例的执行；

**算子**：某一类型的任务抽象。

## 其他模块

**调试模块**：提供Sparksql，Flinksql的调试功能；

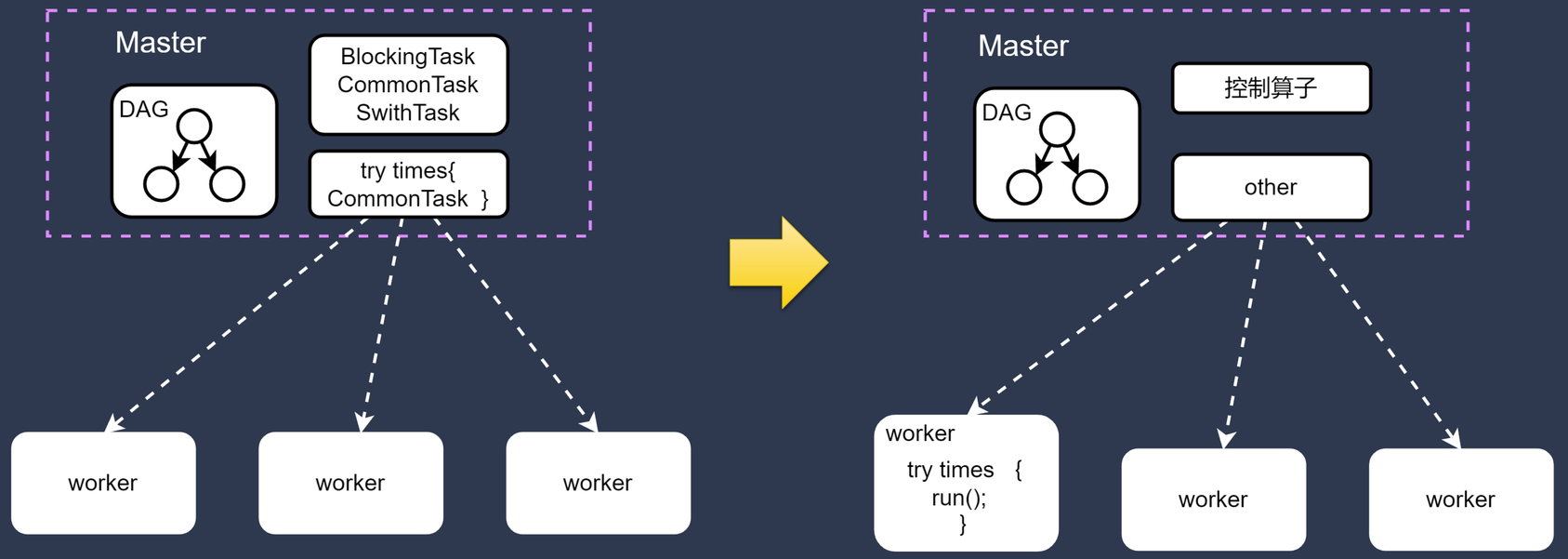
**日志模块**：日志服务；

**告警模块**：警告服务；

**版本管理**：提供任务脚本版本管理；

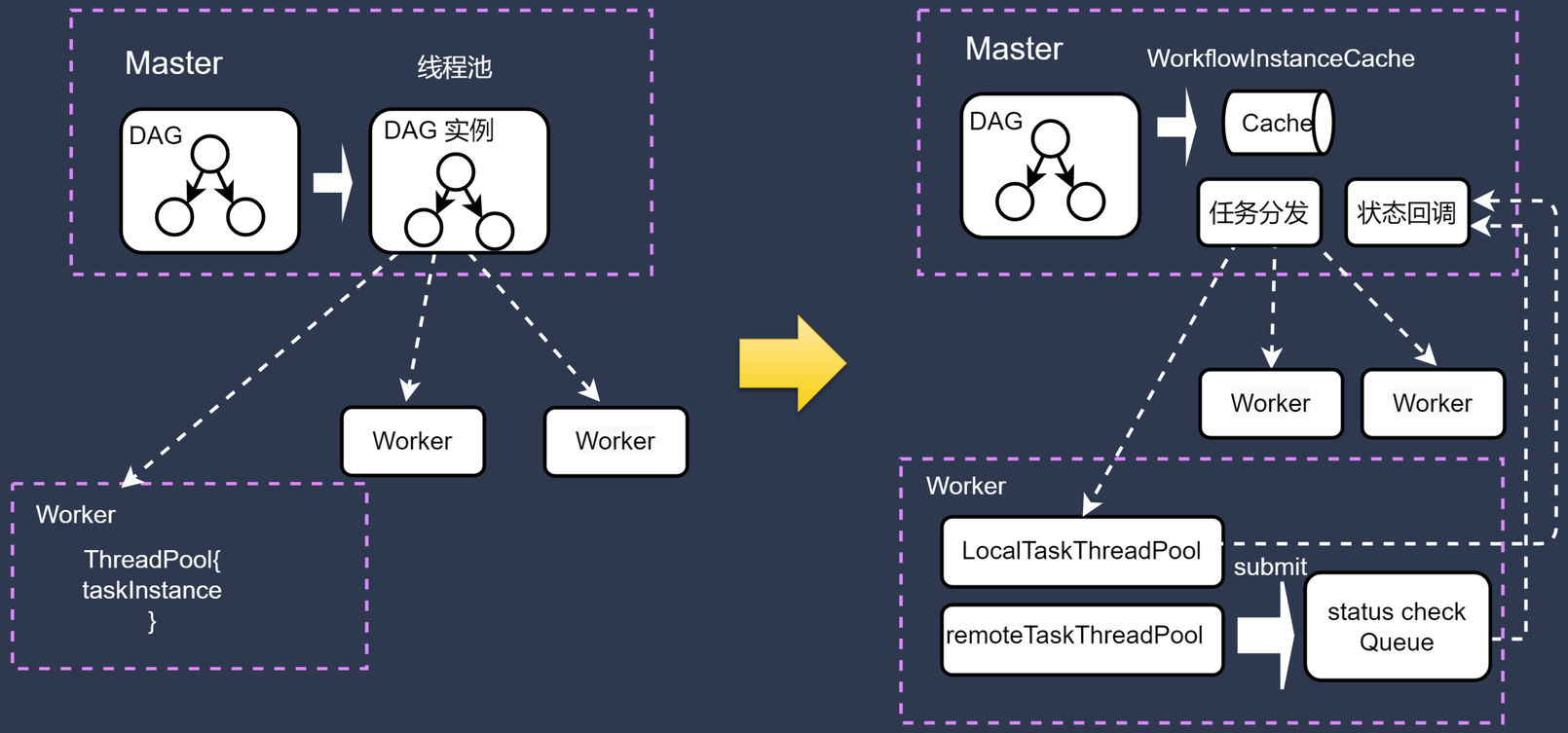
**资源，权限管理模块**：提供基础资源的计算资源，数据资源信息，用户权限信息。

# 02 任务的孵化和执行分离，简化Master职责




- Master只负责流程控制算子
  - 开始算子
  - 结束算子
  - Switch算子
- 优点
  - 职责更明确，Master只负责分发即可
  - 重试下，减少通信次数
  - Master故障时只需要恢复Workflow instance 即可
  - 提升了Master稳定性
  - Master只关心workflow实例状态，Worker只关心task 实例

# 02 减少线程占用提升系统单实例的并发能力




## Master优化

- 由线程监控workflow instance 状态改为队列等待回调。

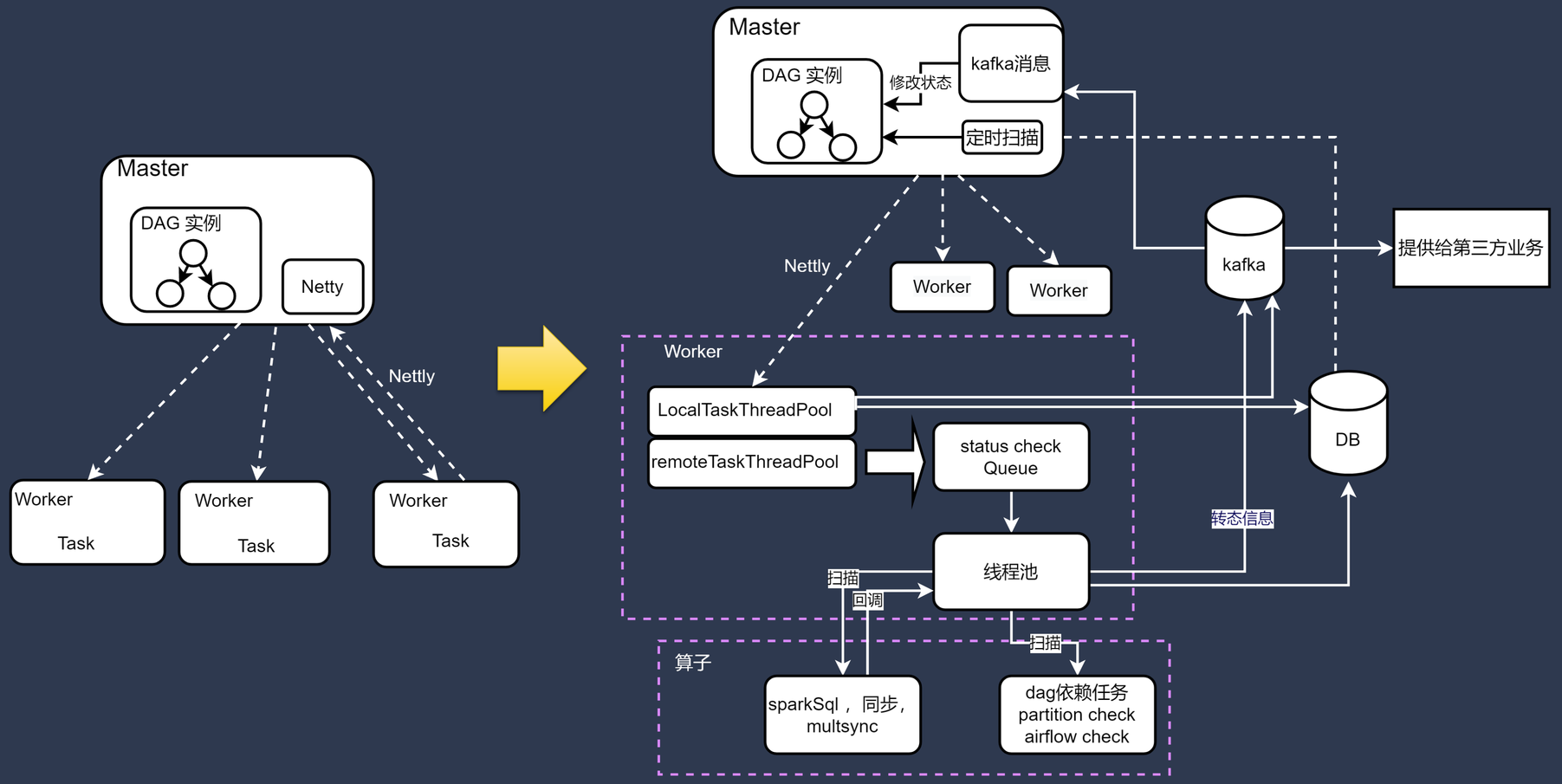
## Worker优化

- 将任务分为Local和Remote任务，Local本地执行，比如Python，Shell等，Remote任务状态由远程服务托管比如Yarn类，Check类，依赖类任务。
- Remote task 提交后，由状态监测队列和线程管理。

## 优化后效果

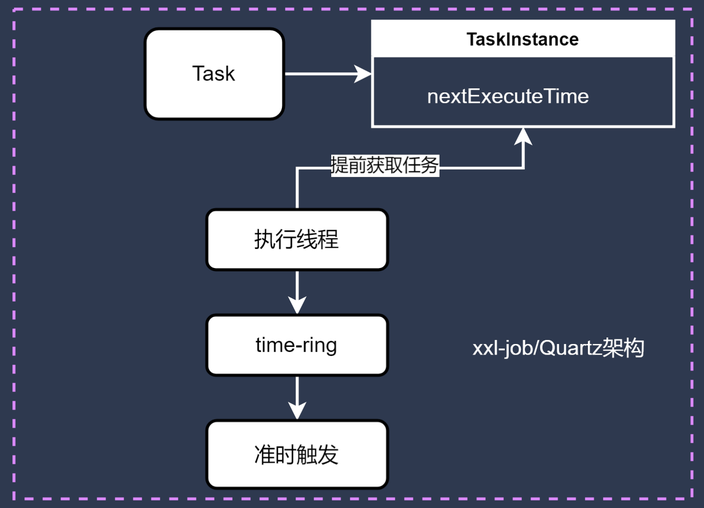
- 单台Master 管理workflow实例能力由100(默认) 提升数十万
- 单台Worker对Yarn任务并发能力提升60倍(Yarn任务平均耗时10min算)
- 采用状态监控类型后，支撑了实时任务和离线任务共同部署

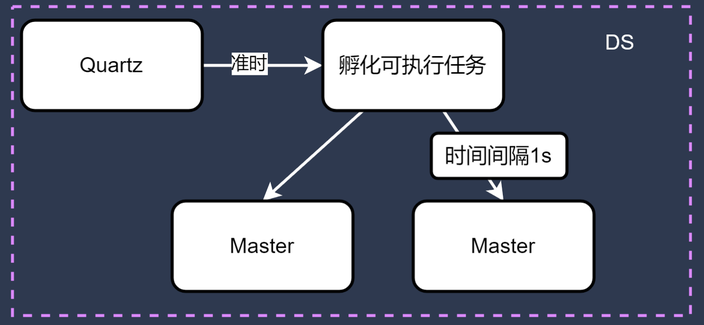
# 02 基于任务状态回调提升系统单实例的并发能力和准时性

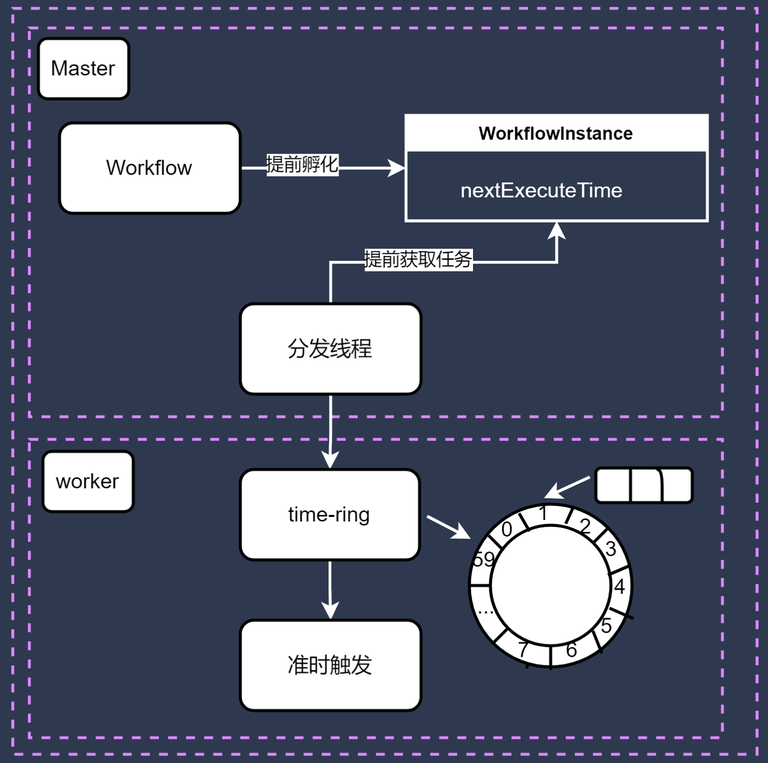



- 引入Kafka
  - 将第三方托管任务状态返还给第三方
  - 利用Kafka能长时间缓存状态，为Master故障恢复提供信息。
- 引入Mysql
  - 防僵死：防止消息遗漏产生状态未更新
  - 量小，频次低：5分钟级别，另只扫描超过5min未完成的的任务实例。
- 在算子中引入回调
  - 减少扫描，提升并发度和响应速度

# 02 提前孵化和时间转盘提升准时性和减少数据库扫描



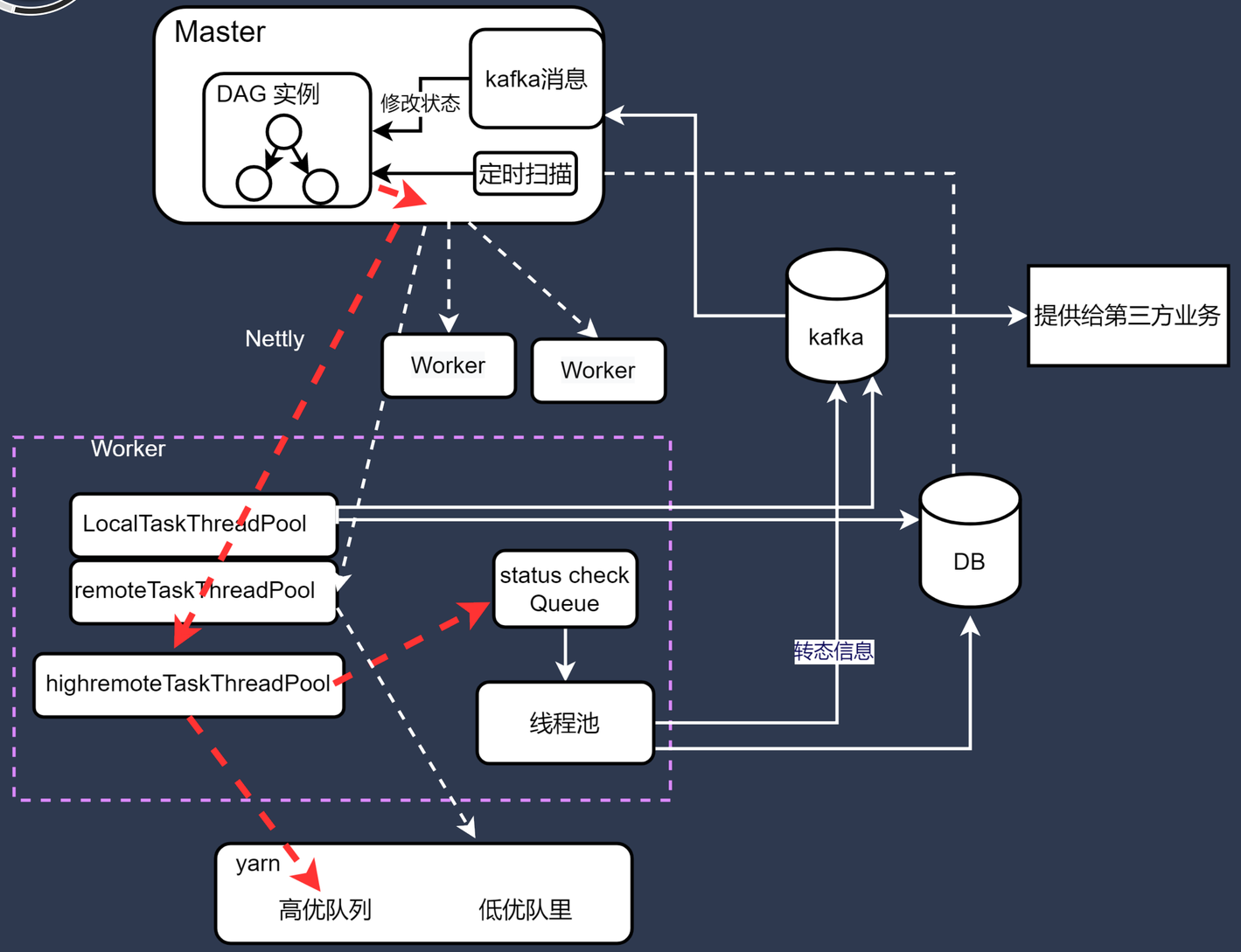

## 预孵调度器

- 通过提前将执行任务并提前分发，减少Master处理和分发过程导致的延时。
- 减少扫库的频率，提升的分发的效率。
- 多余时间通过时间转盘消除，时间转盘每秒转动一次。
- 调度器提前分发，时间转盘消除多余，实现了准时触发

## 优点

- 提前30s获取任务，单Master减少15倍扫描数据库，随着Master功能简化，处理性能提升，假设Master部署数减半，扫描次数减少30倍。
- 降低调度延时，减少了Master和网络的影响

# 02 设置不同优先级队列保障高优任务的执行



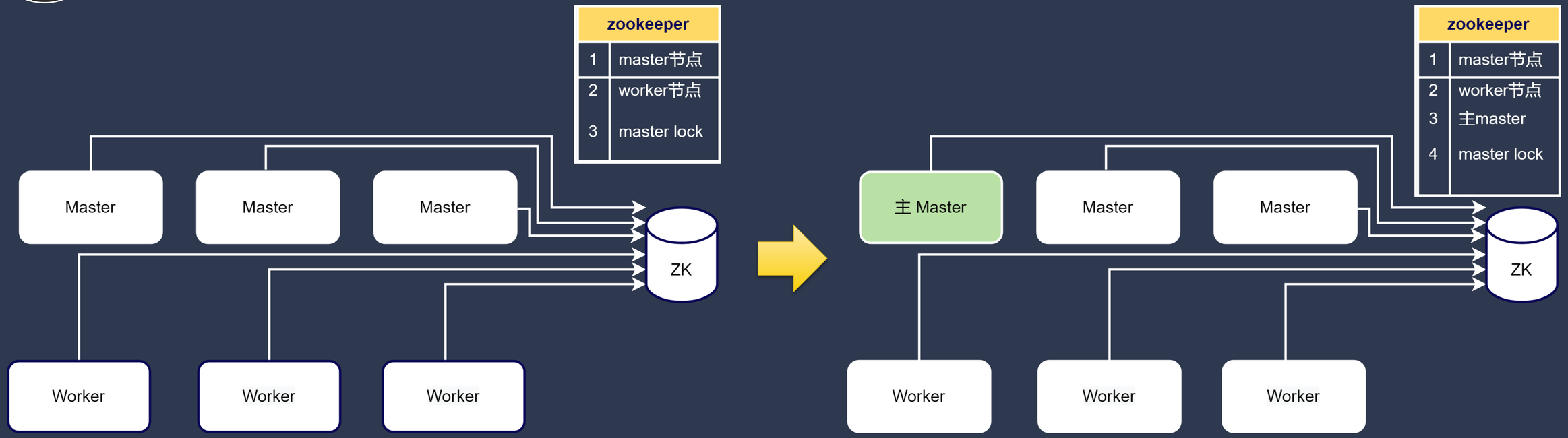

## 挑战点

优先级高的任务无法让正在运行的任务释放资源

## 改进

- Master节点：分发的时候根据任务的优先级排序进行顺序分发。
- Worker节点：划分多类任务的线程池，每一类消耗不影响其他类的任务。
- Yarn管理器：提供高优任务队列任务以组为单位提交，每个组在yarn资源上有高，中，低三个队列。

# 02 新增主Master增强故障处理的能力



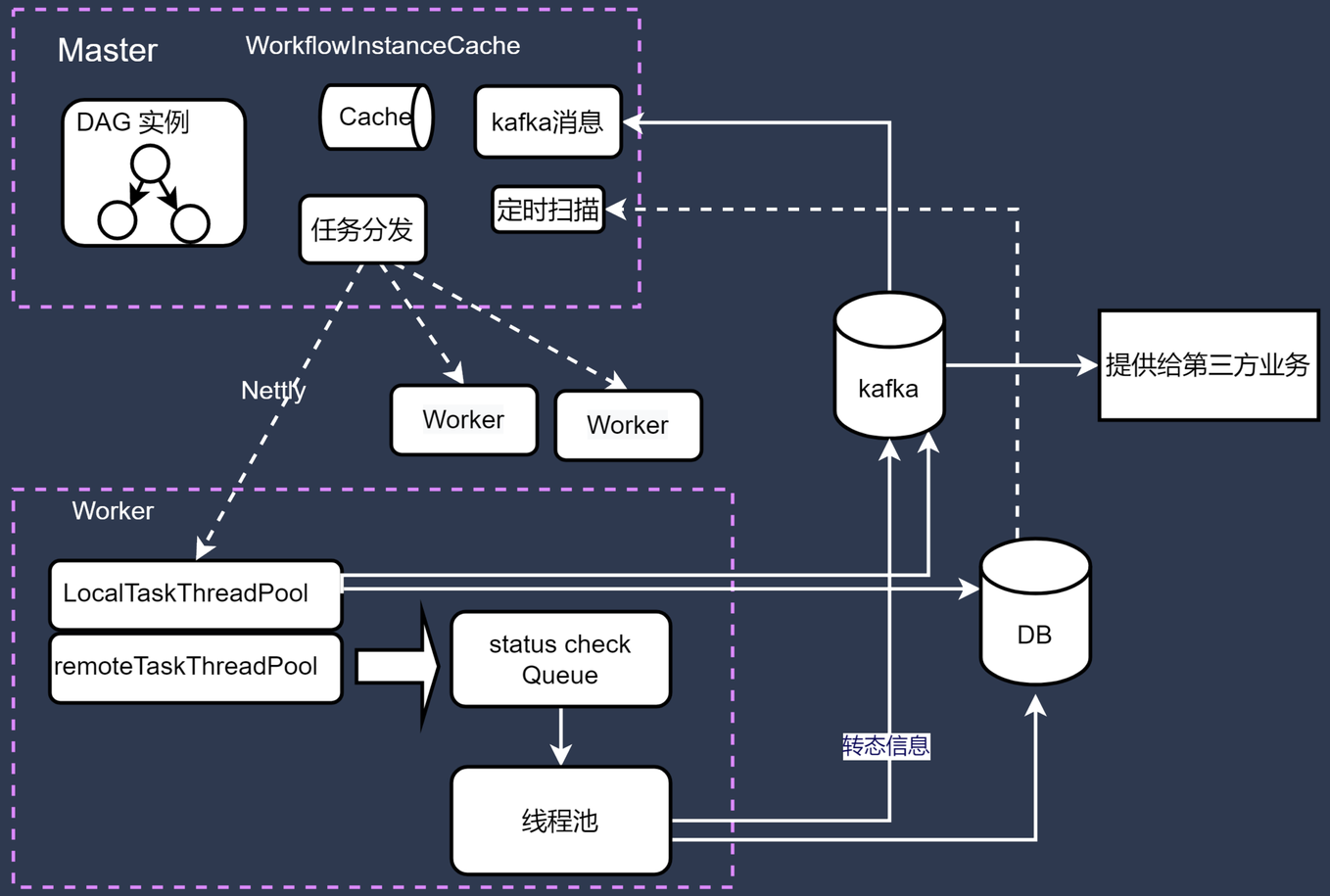

## 原有架构的故障恢复缺点

- 多个master出于同等状态，收到故障消息后通过分布式锁争抢故障处理。
- Master在处理故障的同时还处理其他的任务，影响故障处理的时延。

## 调整后的故障恢复流程

- 多个master里面选出主master，专门用于故障处理。
- 当Master或Worker故障时，主Master停止孵化workflow，优先处理故障。

# 02 Master和Worker分离实现故障的定点恢复




机器故障应尽可能“再续前缘”

## Master故障处理

- 主Master故障：先切主，然后执行下一步。
- 非主Master故障：找到故障机器的workflow 实例，并在主master恢复状态信息即可，不用发送给worker。

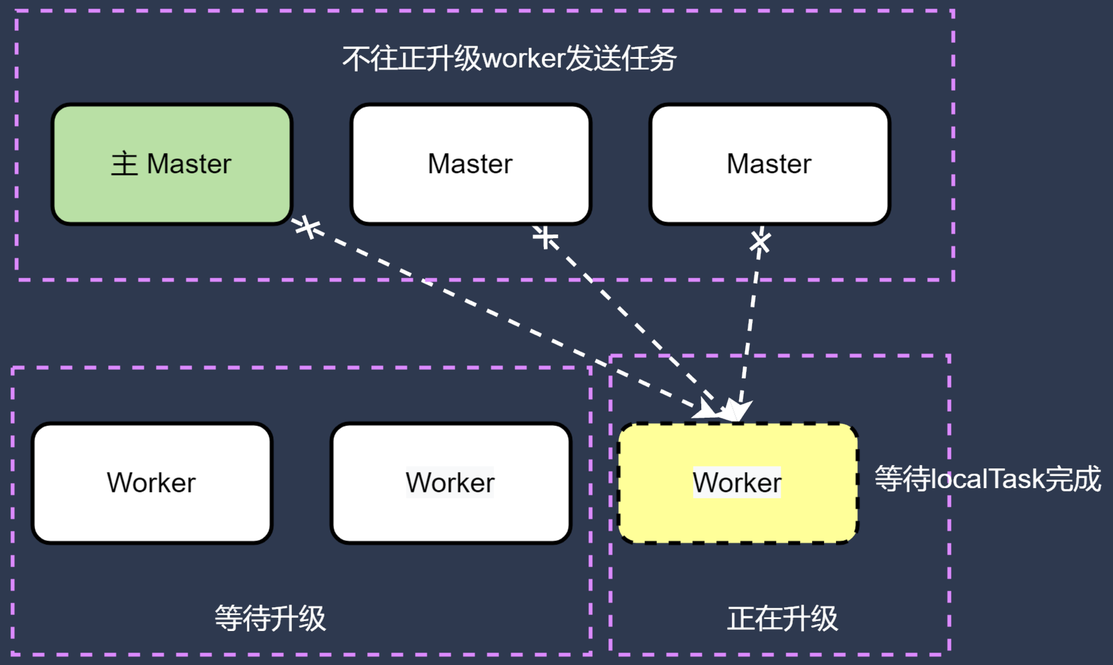
## Worker故障处理

- 主Master监听故障后，找到故障节点的运行的任务，重新随机发送。
- LocalTask：因为在故障机器上运行，需要重新构建。
- RemoteTask：因为在远端运行，从远端资源管理器获取该任务状态即可。

## 优点

- 当前业务本身95%是Remote任务，所以该部分任务无需重跑，仅部分任务需要重新构建重跑，提升了故障恢复的效率。

# 02 Master和Worker的滚动更新保障系统优雅升级



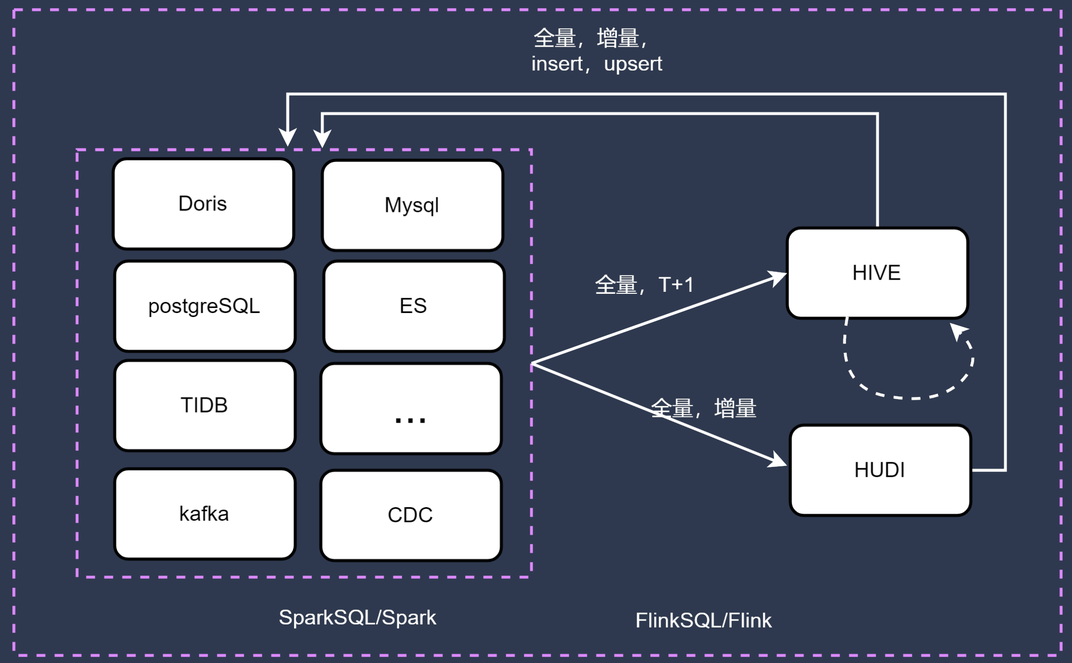

## Master和Worker的滚动升级逻辑

- Master滚动升级：master的状态存储在kafka和数据库当中，单个节点升级的过程如故障处理一样。
- Worker滚动升级：worker节点本地存储了Local任务的执行状态，升级的时候master暂停往该节点发送任务，待Local任务结束即可升级并恢复任务处理。

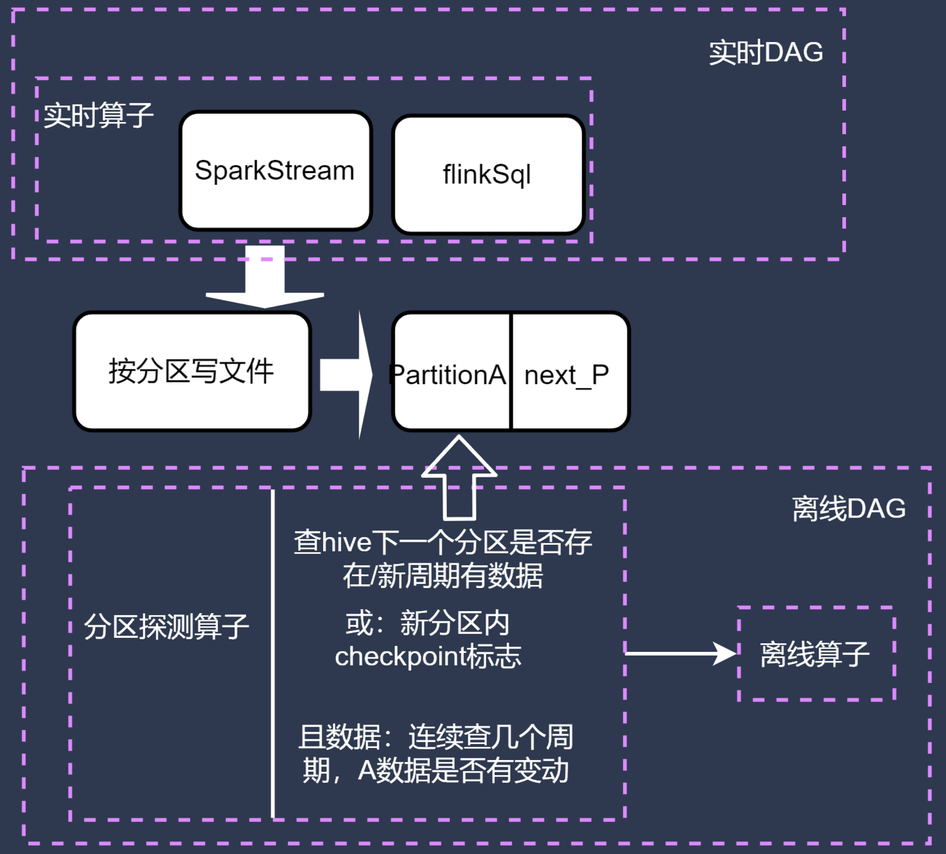
## 效果

- 共经历2个大版本，10多个小版本，另外10多个补丁或紧急需求，均无业务感知

# 02 多种类型任务算子增强支持提升用户易用性




- 基于SparkSQL重构了Sqoop数据互导；支持全量/T+1写Hive；全量/增量入库Hudi；全量/增量/insert/upsert写其他数据库存储。
- 提供SparkSql 对hive的计算。
- 基于FlinkSQL 提供了实时写算子能力。
- 提供MutilInput对Mysql分库分表数据的汇聚

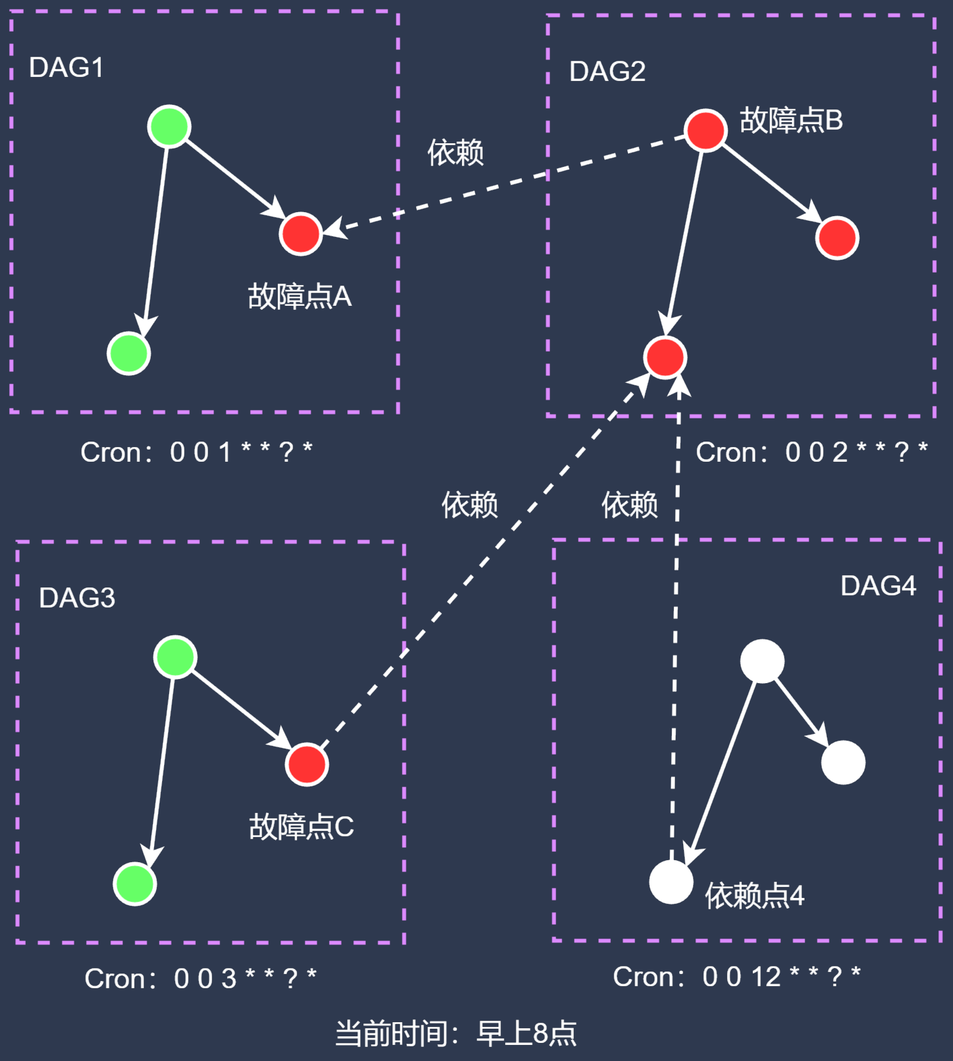



- 通过分区探测算子将实时任务和离线任务建立了联系。


# 未来规划

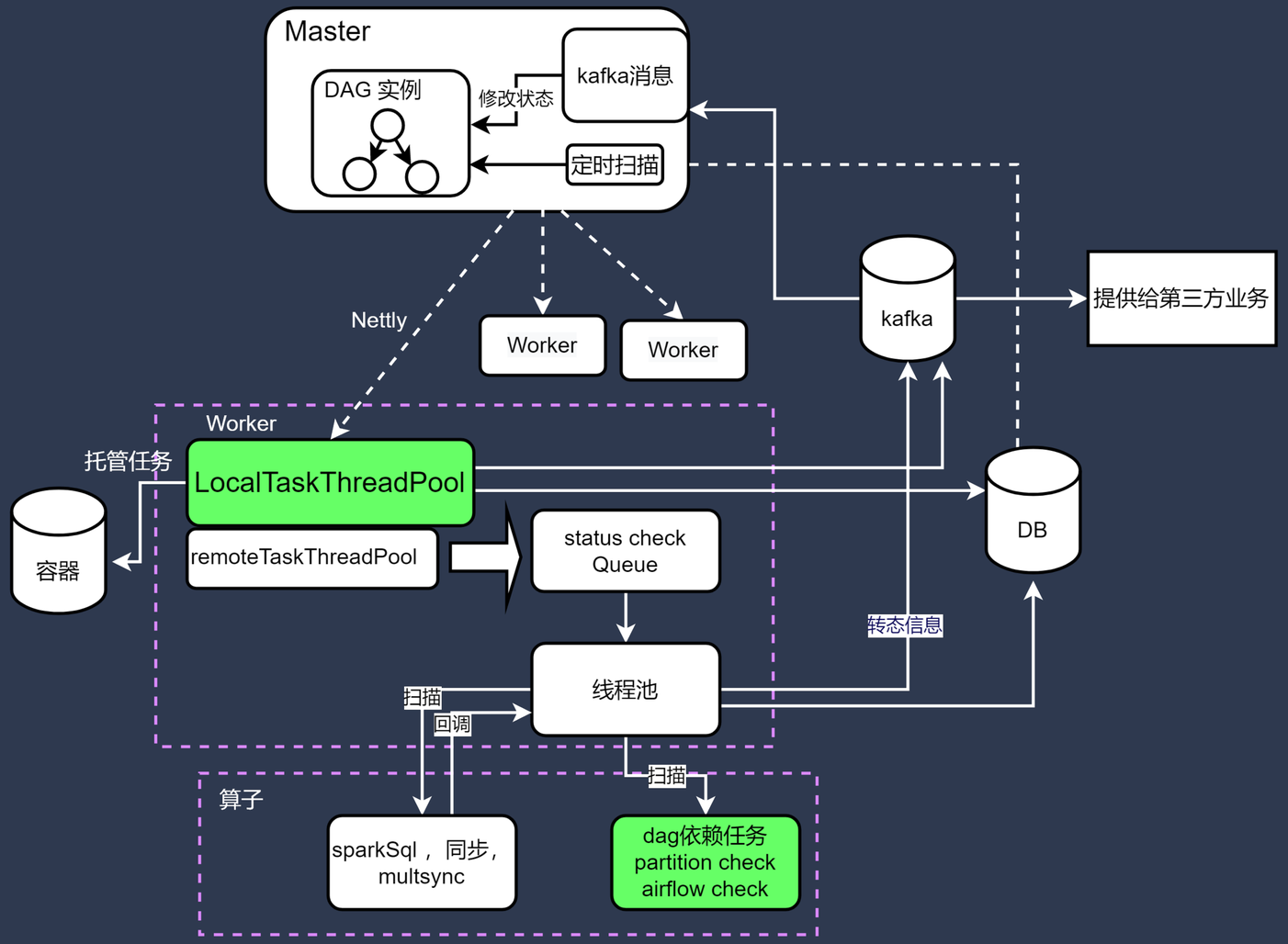
- 基于DAG血缘的任务串联恢复
- 基于异步回调进一步性能提升
- 基于k8s实现Worker节点的弹性伸缩

# 03 基于DAG血缘的任务串联恢复




- 正常调度很难做的之上而下的触发式调度
  - 调度系统采用DAG为基本调度单位，有独自的调度周期。因周期不同，时间点的不同，很难做到至上而下的触发式调度，目前采用检查方式。
- 故障无法避免
  - 调度只能保证任务能启动，但无法控制任务本身错误。
- DAG基本的重跑
  - 基于DAG的血缘关系，可进行DAG之间的重跑，如图，只触发A点重跑，后自动触发B点运行，最后到C点。
- 注意点
  - 依赖点如果没有达到调度时间，将不触发，等待DAG自行触发。如DAG4

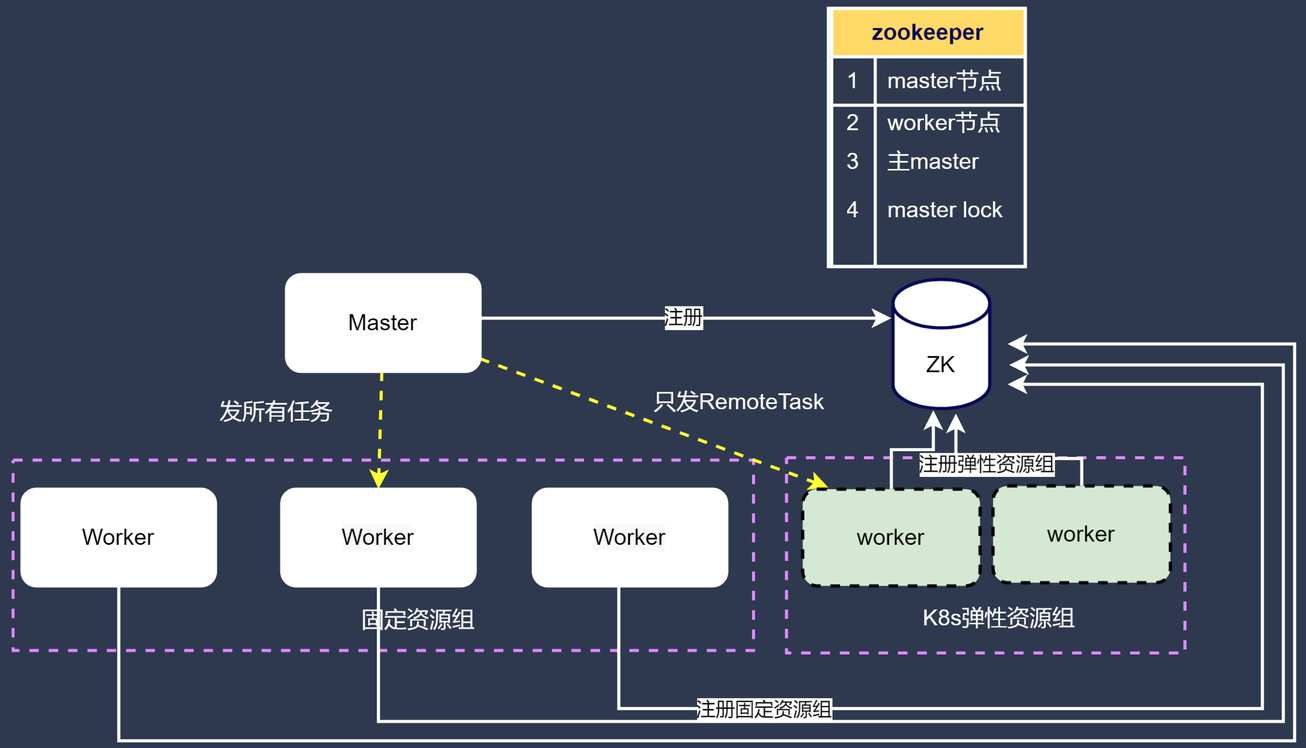
# 03 本地任务容器化和任务状态回调进一步提升稳定性和性能




Local任务采用容器化运行，与Worker节点分离简化运行环境的维护以及缩短故障恢复的时间。

更多算子的任务状态采用回调方式替换常驻线程轮询，降低资源的开销。

# 03 基于k8s实现Worker节点的弹性伸缩


- 基于使用情况，任务有高低峰谷，在高峰期，可以实现弹性Worker

- Master实时收集各worker节点的任务积压情况，根据积压情况自适应进行弹性伸缩。
- 镜像存储在同机房的集群节点上，缩短容器的启动时间

# TGO 鲲鹏会 – 助力企业赢得数字化先机

TGO 鯤鵬會

tgo.infoq.cn

TGO 鲲鹏会是科技领导者同侪学习社区，致力于把技术领导者和专家连接在一起，通过领导力峰会、专属小组活动、闭门沙龙等形式，为所有“孤军奋战”的科技管理者获得自身的成长和职业的发展。

TGO鲲鹏会目前拥有1600+位高端会员。在北京、上海、深圳、广州、杭州、南京、成都、厦门、台北、硅谷、武汉、苏州等全球十二个城市设立分会。




# Thanks

兴盛优选技术官方微信公众号