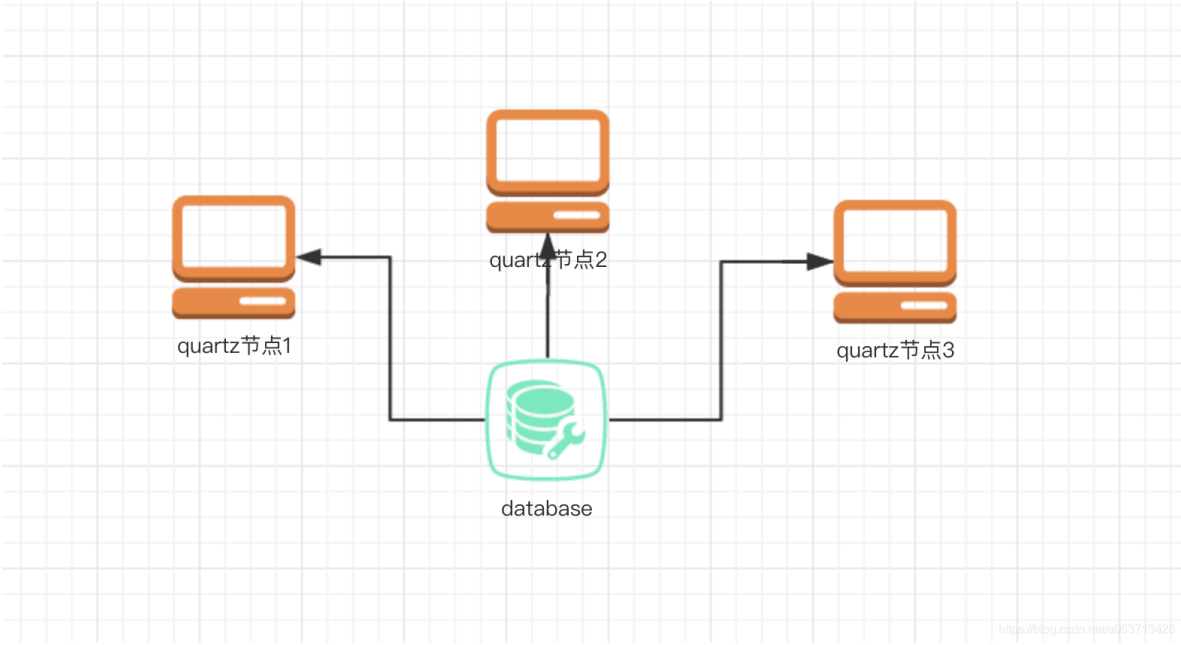
分布式调度

quartz 和 Elastic-Job

1.quartz：

quartz 的常见集群方案如下，通过在数据库中配置定时器信息， 以数据库悲观锁的方式达到同一个任务始终只有一个节点在运行。



优点：

- 保证节点高可用 （HA）， 如果某一个节点挂了， 其他节点可以顶上；

缺点：

- 同一个任务只能有一个节点运行，其他节点将不执行任务，性能低，资源浪费；
- 当碰到大量短任务时，各个节点频繁的竞争数据库锁，节点越多这种情况越严重。性能会很低下；
- quartz 的分布式仅解决了集群高可用的问题，并没有解决任务分片的问题，不能实现水平扩展；

问题：

- 问题一：调用API的的方式操作任务，不人性化；
- 问题二：需要持久化业务QuartzJobBean到底层数据表中，系统侵入性相当严重。
- 问题三：调度逻辑和QuartzJobBean耦合在同一个项目中，这将导致一个问题，在调度任务数量逐渐增多，同时调度任务逻辑逐渐加重的情况加，此时调度系统的性能将大大受限于业务；
- 问题四：quartz底层以“抢占式”获取DB锁并由抢占成功节点负责运行任务，会导致节点负载悬殊非常大。

2.Elastic-Job：

elastic-job 是由当当网基于quartz 二次开发之后的分布式调度解决方案， 由两个相对独立的子项目Elastic-Job-Lite和Elastic-Job-Cloud组成 。

Elastic-Job-Lite定位为轻量级无中心化解决方案，使用jar包的形式提供分布式任务的协调服务。

Elastic-Job-Cloud使用Mesos + Docker(TBD)的解决方案，额外提供资源治理、应用分发以及进程隔离等服务 **亮点**：

- 基于quartz 定时任务框架为基础的，因此具备quartz的大部分功能

- 使用zookeeper做协调，调度中心，更加轻量级
- 支持任务的分片(所谓分片就是分机器)
- 支持弹性扩容， 可以水平扩展， 当任务再次运行时，会检查当前的服务器数量，重新分片，分片结束之后才会继续执行任务
- 失效转移，容错处理，当一台调度服务器宕机或者跟zookeeper断开连接之后，会立即停止作业，然后再去寻找其他空闲的调度服务器，来运行剩余的任务
- 提供运维界面，可以管理作业和注册中心。

> 分片的好处：
>
> Example：你们公司刚成立，只有几十万用户，需要做一个用户订单每日统计分析，你每天凌晨触发任务，只用一个小时就跑完了。过了一年你们的用户量暴增，3000万。你们的分析任务更加复杂了，这个时候跑批任务需要6个小时，等产品经理上班来要结果的时候你发现还没跑完，他被老板骂了一顿，然后回来怼你。
>
> 你虽然受气了心理很难受，但是确实是你的锅。于是在想你有这么多机器为啥非得在一台上跑呢，咋样才能将任务分解上多台机器上独立执行。

不足：

- 异构语言不支持
- 目前采用的无中心设计，难于支持多语言，后面需要考虑调度中心的可行性。
- 监控体系有待提高，目前只能通过注册中心做简单的存活和数据积压监控，未来需要做的监控部分有： 增加可监控维度，如作业运行时间等。 基于JMX的内部状态监控。 基于历史的全量数据监控，将所有监控数据通过flume等形式发到外部监控中心，提供实时分析功能。
- 不能支持多种注册中心。
- 需要增加任务工作流，如任务依赖，初始化任务，清理任务等。
- 失效转移功能的实时性有待提升。
- 缺少更多作业类型支持，如文件，MQ等类型作业的支持。