LTS 业务场景

首先需要知道的是LTS主要支持以下三种任务类型：

* 实时任务　立即执行的任务
* 定时任务　延迟执行的任务,譬如3个小时之后给用户发短信
* Cron任务　周期性的任务，0 0/1 * * * ?

所以根据这三种任务类型可以大致分为以下几个场景。

1. 实时任务，实时执行

这种场景下，当任务量比较小的时候，单机都可以完成的时候，自己采用线程池或者去轮训数据库取任务的方式(或者其他方式)就可以解决。但如果是任务执行时间比较长或者任务量比较大，单机不足以满足需求的时候，就需要自己去做分布式的功能，还有很重要的是，怎么做容错，怎么保证同一个任务只被一个节点执行，怎么解决执行失败异常的情形等等，你就需要自己去做很多事情，头可能就大了。这时候LTS就派上用场了，因为这些问题LTS都帮你解决了，你只需关注你的业务逻辑，而不用为上面的这些事情而烦恼。

当然这时候有人可能会想到如果用MQ去解决，利用MQ的异步去解耦，也同时可以实现分布还有容错等。当然有时候是可以的，为什么说是可以的呢，因为LTS的架构也和MQ的类似，JobClient相当于MQ的Producer，JobTracker相当于MQ的Broker，TaskTracker相当于MQ的Consumer，经过我这么一说，是不是觉得貌似是很像哈。但是我又为什么说是有时候是可以的呢，而不是一定是可以的呢，因为如果你同一个任务(消息) 提交MQ两次，MQ队列中有两条同样的任务消息，那么当你这个任务不能有两个节点同时执行的时候(同时执行一个任务可能出现各种问题)，MQ就不能满足了，因为他不知道你这两条消息是同一个任务，它会把这两条消息可能会发给两个不同的节点同时执行(或者同一个节点的不同线程去执行)，这时候你就需要自己去做一些事情去保证同一个任务不能同时被两个线程(或节点)去执行问题，这时候头又大了，那么LTS又派上用场了，以为LTS就可以保证这一点。

说到任务调度，很多人一下就想到了Quartz，对于这种实时任务的情况，Quartz是不太适合的，它也不能很好的解决故障转移(譬如执行中的节点突然停掉了，Quartz不能将这个执行中的任务立马分配给其他节点执行，最多设置了Quartz的可恢复性，在停掉的节点重启之后重新执行该任务，但如果这个节点再也不启动起来了呢？那就只能呵呵了)等问题，这类场景Quartz就不做比较了。

有些人可能问，说了这么多，你倒是举个例子呀。嗯，我举几个例子：1. 发短信验证码，这种可以用MQ去实现，也可以单机去实现(如果你量不大的话)，当然LTS也是可以的, 如果你量非常非常大的话，建议还是用性能比较好的MQ代替 2. 实时的给在线用户算数据，触发者是用户自己(自己手动点)，但是算任务的只能同时由一个线程去执行，这是就可以用LTS了。

2. 定时任务，某个时间点触发

这种场景下, 和实时任务相比，只有一个不同，就是要指定一个时间点去执行，可能是1个小时之后，可能是1天之后，也可能是1天1小时之后。有些人可能用轮训的业务数据库的方式去解决，轮训业务数据库有什么问题呢，当然你数据量很小我就不说了。如果你数据量还比较大的情况下，轮训数据库，势必会影响业务查询，如果有其他业务查询的话。还有就是对于分布式的支持不是很好，还有

当表中存在多种不同执行(延迟)时间的任务，这个轮训频率就比较关键了，太短，影响性能，太长，影响业务，执行不及时，导致任务执行延迟太久，等等。

这时候如果用MQ，虽然有些MQ支持延迟队列(RabbitMQ,RocketMQ等)。但他们都是指定的一些特定的Level级别延迟，但是不支持任意时间精度，譬如 1min, 5min , 10min 等等，但如果是7分钟，或者20天之后呢。如果MQ支持任意时间精度，那么它的性能就只能呵呵了，这种情况MQ就排除了，但是如果MQ的这些特定的Level刚好满足你的需求，那么选MQ也是可以的。

再说说Quartz吧，Quartz可以支持定时任务，支持某个时间点触发，也支持集群，它在架构上是分布式的，没有负责几种管理的节点。Quartz是通过数据库行级锁的方式实现多线程之间任务争用的问题。行锁有哪些特点呢，开销大，加锁慢，会出现死锁，并发度相比表级锁，页级锁高一点。但是在任务量比较大的时候，并发度较大的时候，行级锁就显得比较吃力了，而且很容易发生死锁。那么LTS是怎么解决并发性的问题的呢，LTS采用预加载和乐观锁的方式，批量的将部分要执行的任务预加载到内存中，所以在取任务的时候只需要两步: 1. 从内存中取到一个任务，当然内存中保证同一个线程拿到同一个任务是很容易的也是很高效的， 2. 将拿到的这个任务对应的数据库记录锁住，那么这里采用乐观锁，CAS的方式去修改记录(如果任务已经被别的节点拿走了，那么重新执行1，2步，这种已经被别的节点拿走的情况，主要是在多个JobTracker的情形下，单个JobTracker不会出现这种情况，但是在多个JobTracker下，内存中的预加载数据采用不同步长的方式来减小两个JobTracker内存中数据重复的概率，很好的解决了这个问题，这里稍微提下)，所以这个时候LTS相对于Quartz的优势一下就体现出来了。还有就是上面说的Quartz对故障转移做的不是很好。还有就是当Quartz对应的MySQL数据库挂了，这时候问题就来了客户端提交的任务提交不成功了，那么有人会想将这些数据保存在内存中，等MySQL重启起来了再重试提交，那么如果你当前节点也挂了呢，你内存中的数据就会全部丢失了，所以这时候你需要自己额外的去做一些失败任务本地持久化的工作，不过如果你用LTS的话，LTS支持FailStore，任务提交失败了，自动帮你本地持久化，然后待JobTracker可用的时候重试，不管你是JobTracker挂了，还是JobTracker对应的数据库挂了，都是ok的。

举个例子吧，在一个小时之后给某些用户发短信，或者当用户点击退款操作之后，从点击退货的这个时间点开始，n天后将这个退款关闭。

3. 周期性任务

这种场景下，和定时任务相比，不一样的地方，就只有，这个是会重复执行的，相当于重复执行的定时任务。所以这种场景下的对比，可以继续考照 定时任务的对比。

LTS 在这种场景下提供的特性有，提供统一的后台监控和后台管理。当某次定时任务执行失败，会执行重试操作，并提供执行日志。