分布式锁之 zookeeper 篇

作者：BigBigBigPeach
原文链接：https://ld246.com/article/1564317241472
来源网站：链滴

<h2 id="前景回顾">前景回顾</h2>
前面我们在分布式锁 Redis 篇中提到了为什么要使用分布式锁，同时也发现了 redis 单机分布式锁的端。后来我们又分析了 Redis 分布式锁之 RedLock 实现
，解决了单机的 redis 分布式锁问题。但 Redlock 算法虽然强大，但并不是完美的。缺点是算法较复，而且也可能小概率的出现问题。分布式专家 Martin 推荐使用 ZK 实现分布式锁，这篇文章就来分一下。
<h2 id="zookeeper简介">zookeeper 简介</h2>
zookeeper 是一个分布式服务服务，用于维护配置信息，命名，提供分布式同步和提供组服务提供服务注册中心、服务发现等服务。
zk 里面的节点成为 node，我们可以把 node 当成一个“文件夹”。
那么接下来就很好理解了，文件夹有名称，大小，位置，创建时间，修改时间等。且文件夹可以有父级文件夹。那 node 也是类似的
<pre><code class="highlight-chroma"># 创建了一个root，然后进行查看
[zk: localhost:2181 CONNECTED) 20] ls /
[aaa0000000003, ookeeper, aaa0000000002, aaa0000000001, root]
root下面有一个节点
[zk: localhost:2181 CONNECTED) 21] ls /root
[child1]
查看root node 息
[zk: localhost:2181 CONNECTED) 22] get /root
root
# 创建事件id
cZxid = 0x13
ctime = Sun Jul 28 03:10:03 GMT 2019
# 更新事件id
mZxid = 0x13
mtime = Sun Jul 2 03:10:03 GMT 2019
# 此为全局最大id
pZxid = 0x15
cversion = 1
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 4
# 子节点个数
numChildren = 1
</code></pre>
<h2 id="zk分布式锁">zk 分布式锁</h2>
我们要使用 ZK 的分布式锁，必须要借助 zk 的一些特性。我们需要创建一个临时且有序的节点代码如下。当一个节点创建了一个节点之后，需要看一下自己是不是当前最小的临时节点，如果是的

，那么代表自己获得了分布式锁资源，可以进行自己的业务操作，如果不是当前最小的节点，那么需在比自己小的节点上加上一个 watch 时间，监听这个节点的删除事件。
我们来思考一下其中的细节。
<h4 id="为什么需要使用临时的目录呢-">为什么需要使用临时的目录呢？</h4>
因为如果使用永久性的目录，可能在客户端宕机或者网络不通畅的时候出现无法删除的情况。采临时目录，当客户端断开连接时(主动断开或者是网络问题等)，该临时节点会被删除，不会出现脏节。
<h4 id="为什么使用有序节点">为什么使用有序节点</h4>
zk 里面提供了有序节点的功能，这个功能是当你创建一个节点的时候，zk 会在节点名称后面加一个整数。这个整数就像 mysql 等数据库的 int 自增主键，是有序递增的。我们采用有序节点可以作请求的一个排队方式。假设我有 3 个节点来获取分布式锁（顺序 A1,C2,B3），那么当 A 获取了锁之，C 在创建的时候，发现自己的节点不是最小的，那么只能等待。B 在 C 之后创建，自然也获取不到。当 A 断开连接的时候，zk 会删除 A 的临时节点，这个时候 B\C 会重新检查是否是最小节点，来确自己是否需要进行业务处理。
<h4 id="如何进行节点变更通知">如何进行节点变更通知</h4>
上面我们提到了，假设前面的节点已经被释放了，那么怎么去通知其他节点。我们可以利用 zk 的 watch 机制，对自己希望监听的节点进行注册监听，当此节点发生变更的时候，zk 会通知相应的监者。比如上文提到的 A,B,C 三个节点，B,C 可以对 A 进行监听，B 可以对 C 监听。只需要监听比自节点小的节点即可。还有一个优化的点，其实每个节点只需要关注比自己小的一个节点即可/strong>。比如 B 不需要关注 A 节点的变化，只需要关注 C 节点的变化即可。这样减少了节点变化候的通知消息数量，避免每个节点多次去查看当前是否需要去执行。
即使有特殊情况，如 A 还没有断开连接（A 占有的临时节点没有被删除的时候，C 的连接断开了，C 对应的节点被删除，B 收到消息也不会直接执行业务代码，B 会先执行节点检，如果当前的节点还有比自己小的节点，那么会对比自己的小的的节点进行 watch 事件监听。
<h4 id="zk分布式锁有没有什么问题-">zk 分布式锁有没有什么问题？</h4>
我们上面已经分析过监听过程的问题和解决方式。但是当我们再回过头来查看整个过程会发现还有一些问题存在的。
当 A 节点获取节点成功后，发现自己是最小的临时节点，于是开始放心的执行业务逻辑。在还未执行成的时候，与 zk 的连接断开了。zk 删除了 A 对应的临时节点，C 发现自己的节点变成了最小的临时点，于是 C 开始执行业务逻辑....说到这，大家应该知道可能会发生什么了，A、C 在同时操作一个资，结果是不确定的。
解决办法是什么呢？我们最好在执行业务事件的时候，把事件做成幂等函数。另外，像加减金额，最采用 CAS 的思想，如果更新的时候，发现原有的值和现在未更新的值已经不一样了，那就需要特殊断，是否需要继续更新。
<pre><code class="highlight-chroma"># 创建临时且有序的节点。-s即有序，-e即临时
[zk: localhost:2181 CONNECTED) 5] create -s -e /root/addOrder addOrder
Created /root/addOrder0000000002
[zk: localhost:2181 CONNECTED) 6] create -s -e /root/addOrder addOrder
Created /root/addOrder0000000003
</code></pre>
<h2 id="开发使用">开发使用</h2>
开发中如果使用 zk 客户端的原生 api 实现分布式锁可能会比较难受，因为代码一看就比较冗长所以我们可以用 Curator 来实现锁的。具体细节可看 curator 分布式锁 InterProcessMutex
<h2 id="总结">总结</h2>
zk 分布式锁在整个过程上比 redis 的 redlock 算法要易于理解，不需要计算获取锁的时间等操

。但是完美的事情太少，我们在 coding 过程中要考虑一些比较极端的情况。如果出现脏数据、计算误等情况，带来的痛苦就不是多写几行代码能比的了....</p>