分布式事物

作者：0ziyu0
原文链接：https://ld246.com/article/1554125032883
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

一、产生背景
　　1、基于微服务环境下，将系统中的业务进行拆分成不同的子模块来构建成不同的服务。在同的服务中其数据库连接和表结构相互独立运行、互不影响，此时夸模块之间的调用产生异常需要手回滚，从而增加了业务代码的复杂度。（eg：有 2 个微服务 A 和 B，当 A 调用 B，B 服务成功将数写入表中，A 继续执行业务逻辑发生异常，则需要手动再次调用 B 服务将保存的数据删除掉，可以试下如果出现多个微服务调用其中最后一个服务产生异常，此时需要回滚前面所有数据。）　　
二、Base 理论和 CAP 理论
　　1.ACID 原理
　　原子性（Atomicity） 原子性是指事务是一个不可再分割的工作单元，事务中的操作要都发生，要么都不发生。
　　一致性（Consistency）一致性是指在事务开始之前和事务结束以后，数据库的完整性束没有被破坏。
　　隔离性（Isolation）多个事务并发访问时，事务之间是隔离的，一个事务不应该影响其事务运行效果。
　　持久性（Durability）意味着在事务完成以后，该事务所对数据库所作的更改便持久的存在数据库之中，并不会被回滚。
　　注意：传统的 ACID 只能保证单个微服务事务。
　　2.CAP 理论
　　Consistency：(强)一致性，在分布式系统中的所有数据备份，在同一时刻是否同样的。（等同于所有节点访问同一份最新的数据副本）。
　　Availability：可用性，在集群中一部分节点故障后，集群整体是否还能响应客户端的读请求。（对数据更新具备高可用性）。
　　Partition tolerance：分区容忍性，相当于对通信的时限要求。系统如果不能在时限内成数据一致性，就意味着发生了分区的情况，必须就当前操作在 C 和 A 之间做出选择。
　　注意：对于分布式数据系统，分区容忍性是基本要求，否则就失去了价值。目前常用的分布事务大多使用补偿机制将一致性改成最终一致性(5 分钟 batch[忽略本括号中])。
 　　最终一致性需要一个时间窗口在用户感知到的时间内将数据同步。
　　3.BASE 理论
　　Basically Available：基本可用，分布式系统在出现不可预知故障的时候，允许损失部可用性（不等价于系统不可用）。
<pre><code class="highlight-chroma">　　*　响应时间上的损失：正常情况下，一个在线搜索引擎需要0.5秒内返回给用户相应的查询结，但由于出现异常（比如系统部分机房发生断电或断网故障），查询结果的响应时间增加到了1~2秒。
　　*　功能上的失：正常情况下，在一个电子商务网站上进行购物，消费者几乎能够顺利地完成每一笔订单，但是在些节日大促购物高峰的时候，由于消费者的购物行为激增，为了保护购物系统的稳定性，部分消费者能会被引导到一个降级页面。
</code></pre>
　　Soft-state：软状态/柔性事务，软状态是指允许系统存在中间状态，并且该中间状态不影响系统整体可用性。即允许系统在不同节点间副本同步的时候存在延时。
　　Eventual Consistency：最终一致性，系统中所有的数据副本，在经过一段时间的同步，最终能够达到一个一致的状态。因此，最终一致性的本质是需要系统保证最终数据能够达到一致，不需要实时保证系统数据的强一致性。
　　注意：BASE 理论源于对大规模互联网系统分布式实践的结论，是基于 CAP 定理逐步演化来，ACID 是传统数据库常用的概念设计，追求强一致性模型。。
三、刚性事务和柔性事务
 　　1.刚性事务：遵循 ACID 原则，强一致性。
　　2.柔性事务：遵循 BASE 理论，最终一致性；与刚性事务不同，柔性事务允许一定时间内，同节点的数据不一致，但要求最终一致。
　　3.柔性事务分类：两阶段型、补偿型、异步确保型、最大努力通知型。
四、实现原理
  　　1.两阶段提交协议：
<pre><code class="highlight-chroma">*　第一阶段(准备阶段)：协调者向参与者发起指令，参与者评估自己的状态，如果参与者评估指

</code></pre>

可以完成，则会写redo或者undo日志，让后锁定资源，执行操作，但并不提交。
* 第二阶段(提交段):如果每个参与者明确返回准备成功，则协调者向参与者发送提交指令，参与者释放锁定的资源，何任何一个参与者明确返回准备失败，则协调者会发送中指指令，参与者取消已经变更的事务，释放定的资源。
</code></pre>
　　缺点：1.如果协调者宕机，参与者没有协调者指挥，则会一直阻塞。2.协调者没有一个超时间设定。
　　 三阶段提交协议：
<pre><code class="highlight-chroma">* 第一阶段（询问阶段）：协调者询问参与者是否可以完成指令，协调者只需要回答是还是不是而不需要做真正的操作。
</code></pre>
　　　　优点：通过超时机制解决了阻塞的问题，不会阻塞和永远锁定资源。
 　　2.补偿性：在一个长事务（long-running）中，一个由两台服务器一起参与的事务服务器 A 发起事务，服务器 B 参与事务，A 的事务如果执行顺利，那么事务 A 就先行提交，如果事务 B 也执行顺利，则事务 B 也提交，整个事务就算完成。如果事务 B 执行失败，事务 B 本身回滚，这事务 A 已经被提交，所以需要执行一个补偿操作，将已经提交的事务 A 执行的操作作反操作，恢复未执行前 事务 A 的状态。
　　3.异步确保型：将一些同步阻塞的事务操作变为异步的操作，避免对数据库事务的争用，典例子是热点账户异步记账、批量记账的处理。
　　4.最大努力型：通过通知服务器（消息通知）进行，允许失败，有补偿机制（或重发机制）

五、LCN 实现原理

LCN 客户端（发起方和参与方都必须要注册到事务协调者中）建立一个长连接。
事务发起方调用参与方接口之前会向 Tx-Manager 事务协调者创建一个事务分组 id。
事务发起方调用参与方接口时，会在请求头中存放该事务的分组 id 传给参与方。
如果参与方服务获取到请求头中有对应的事务分组 id，当参与方服务业务逻辑代码执行完毕后，采用假关闭 JDBC 连接，不会向参与方服务提交该事务
参与方在发起方执行成功后，才会提交事务。

六、其他常见解决方案
　　1.使用阿里巴巴的 TCC 补偿框架
　　2.基于可靠的消息模式（RabbitMQ）
　　3.使用阿里的 GTS 框架解决
　　4.使用本文介绍的 LCN 框架
　　注意：单一项目中采用 Jta+Atomikos
七、LCN 框架代码实现(核心类)
　　详细代码已上传：http://47.93.254.162/open
项目架构
<img src="https://ld246.com/images/img-loading.svg" alt="a.png" data-src="https://b3logfil.com/file/2019/04/a-9176238a.png?imageView2/2/interlace/1/format/jpg">
<pre><code class="highlight-chroma">1、tx-manager
</code></pre>
<img src="https://ld246.com/images/img-loading.svg" alt="b.png" data-src="https://b3ogfile.com/file/2019/04/b-805a345b.png?imageView2/2/interlace/1/format/jpg">
配置注册中心地址和 redis
2、事务发起方添加 @TxTransaction(isStart=true)即可
<img src="https://ld246.com/images/img-loading.svg" alt="c.png" data-src="https://b3logfil.com/file/2019/04/c-ee4355c5.png?imageView2/2/interlace/1/format/jpg">
3、事务参与方 @TxTransaction

<img src="https://ld246.com/images/img-loading.svg" alt="d.png" data-src="https://b3logfile.com/file/2019/04/d-232a9053.png?imageView2/2/interlace/1/format/jpg">
注意在 tx-manager 中默认 tm.compensate.maxWaitTime=50000 设置默认等待 5 秒钟，根据具业务处理时间设定
原理简介：当事务发起方由 tx-manage 在 redis 中获取一个唯一 id，当调用事务参与方时会将该 id 给调用方，此时会将 jdbc 的连接信息交给 tx-manage 进行管理,当参与方调用结束时，tx-manage 会提交该事务，当事务调用方执行完毕、没有异常抛出并 tx-manage 没有超时的情况下会更具唯一 id 将参与方和调用方的 jdbc 连接进行提交，若有一方抛出异常则都不会提交事务。</p>