LCN(Lock Control Notify) 一种基于Java代理协调技术的分布式事务系统

lorne

wangliang@codingapi.com

www.txlcn.org

NO.JN20200102

摘要:LCN分布式事务系统通过代理数据资源，通TxManager(事务管理器)协调来完成对事务的统一控制，这样的操控方式使得框架对业务嵌入性非常低，在对本地代理资源的同时也通过排它锁防止其他人的访问，从而也保障了事务的隔离性。

名词解释:

TC:Transaction Client 代表事务客户端，对应流程中是对事务发起方与事务参与方的泛指。

TM:TransactionManager，也简称为TxManager是事务管理器。

1. 简介

目前对于分布式事务的相关依据主要还是BASE[1]理论，尽管已经存在像TIDB[2]一样全面支持ACID[3]的NewSQL[4]数据库，但由于相关技术还尚未完全普及，以及对其他数据库环境的混合使用，也导致不能完全靠一种数据库技术来实现分布式系统的所有事务问题。分布式事务对金融领域来说更为重要，随着分布式计算的普及，业务的扩张，金融业务的公司都将自己的业务服务以分布式的形式部署在网络的多个节点上，金融公司对用户资金控制的数据一致性要求度是特别高的，但由于金融公司拥有众多的业务系统，分散的数据以及多样性的框架，使其对数据一致性的控制变的更加困难。LCN是由Lock Control Notify三个单词的首字母组合而来，他们含义为是:Lock:是指通过获取到资源的控制器，形成对其他访问的排他性;Control:是通过控制数据的写入而达到对事务的提交与回滚.Notify:通过消息来协调控制各节点的数据事务操作。

2. LCN的原理介绍

2.1. LCN组件介绍

TransactionClient(TC):

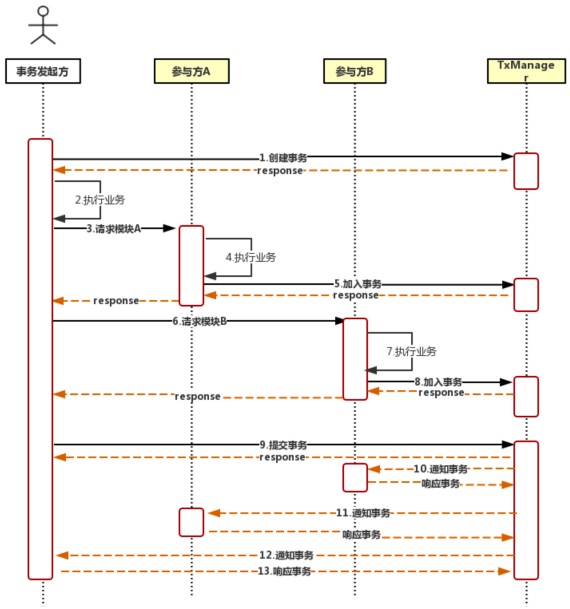
主要是对资源做代理控制锁定以及配合TxManager对事务做提交回滚。

TxManager(TM):

事务协调管理器，简写TM有时候也称作为TransactionManager，是负责控制整个事务保存一致

性的控制节点，也记录事务中可能出现的特殊情况与各阶段的操作日志数据用于调试追溯。

2.2. 事务的协调原理



上图是对分布式事务的协调控制流程图

步骤介绍：

步骤1：

事务发起方在开始执行业务之前将先调用TM通知创建事务，事务中包含关键的事务信息如节点信息、全局事务Id标示等信息，发起方将等待TM响应数据以后再执行步骤2。

步骤2：

当接受到TM创建的事务消息以后，事务发起方则开始执行自己的业务代码，在执行业务的过程就会开始对各个参与方的调用，在调用时会将TM创建的事务标示信息传递到参与方中。

步骤3:

事务发起方开始发起对事务参与方的调用，调用时将会把事务标志信息传递给参与方。

步骤4:

当参与方接受到来自发起方的请求之后就开始执行本地的业务。在这里执行完业务后不会立即对本次的本地事务做处理，实际上在系统中是代理了连接对象没有对其做真正的事务操作。具体的代理步骤将会在下面的代理机制中解释。

步骤5：

当参与方A完成了本地业务的执行以后，再请求TM加入到本次事务中，提交的主要信息有本地的日志标记信息与节点信息，在请求TM前会先开启超时等待任务，然后再发起加入事务指令给TM，加入事务受系统控制。

步骤6、7、8：

该步骤原理对账3、4、5一样，只是参与方不同而已。

步骤9：

当发起方完成所有的参与方调用以后，则根据最终的事务情况来通知TM提交事务，该步骤需要确认TM接受到消息，提交事务分为：commit、rollback两种，当提交完成事务以后也会与参与方一样开始进入等待超时机制。TM通知的流程是会遍历参与方逐步通知确认，全部通过后再通知发起方，详细原理见TM原理介绍。

步骤10：

TM通知参与方节点做事务的commit或rollback操作，对于TM通知成功的原则是需要将消息通知到参与方，参与方完成事务的执行后再将消息反馈给TM，TM接受到消息以后再开始调用下一个参与方的事务通知。这里存在超时机制与重试机制和切换模块事务机制，详细介绍可参考下面的内容。

步骤11：

该步骤原理同步骤10一样，只是参与方不同而已。

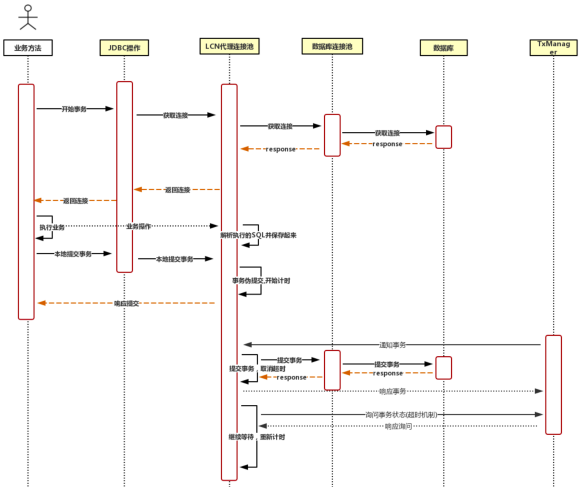
步骤12：

通知发起方事务，该步骤是TM已经完成了对所有参与方的通知确认后，再通知发起方提交事务的操作步骤，发起方根据事务状态完成对事务的操作。

步骤13：

当发起方完成了事务操作则会通知TM事务已提交完成，这时候对于TM来说才是所有事务的完成，该消息无需TM有无接受到，都将在发送完成以后响应数据给调用者。由于事务已提交TM再次发起请求，也会由于SQL日志已清理而无法重复提交。

2.3. TC代理控制原理(强一致)



上图是TC代理控制处理的流程图

步骤介绍：

业务方法(开始事务)

开始事务是指参与方或者发起方开始执行自己的业务时开启事务操作，对应JDBC来说就需要去获取Connection连接对象，在这一步是将会开始访问JDBC[5]接口获取连接对象。

JDBC操作(获取连接)

这里代表的是通过JDBC协议向连接池获取Connection连接对象的请求。

LCN代理连接池(获取连接)

这里是指通过LCN代理连接池继续将请求传递，向数据库连接池层获取Connection连接对象。

数据库连接池(获取连接)

这里是数据库连接池继续向数据库层获取连接请求传递。

LCN代理连接池(代理连接)

代理连接是指LCN将返回的连接对象创建代理对象，然后在返回给调用方。代理对象主要是为了获取到实际要执行的业务操作和控制数据的“伪提交”代理控制方，这里是一个抽象的接口，目前代表的是关系型数据库的代理实现。

业务方法(执行业务)

业务方法在执行业务的时候会对资源做操作，当操作资源的时候就会发送数据给LCN代理的对象。

业务方法(业务操作)

随着业务的操作，则将控制数据传递给了LCN代理连接池。

LCN代理连接池(解析执行的SQL并保存起来)

LCN接受到了业务方法的操作然后对其做业务解析操作，这里由于是关系型数据库，解析得到将是SQL语句。对SQL有两点主要事项：1、所有的赋值对象将通过java来赋值而不能直接使用数据库函数，就如mysql的now()函数一样，那么再对其做补偿业务的情况下时，会出现记录的时间会与实际发生时间不一致的情况，就违背了幂等性;2、若是insert语句且是数据库主键自增类型的得需要获取到插入以后的主键key信息。

业务方法(提交事务)

业务方法完成了所有的业务操作，开始发起提交本地的事务提交请求。

LCN代理连接池(事务伪提交，开始超时机制)

LCN代理连接对象接受到提交事务请求时会阻止事务的提交，但是会将解析到的业务操作日志数据(SQL)存储到本地，再开始启用倒计时等待TM的消息通知，然后再返回提交成功给业务方法。

LCN代理连接池(响应提交)

LCN代理连接池将响应完提交事务以后将结果反馈给参与业务方，业务方法再继续执行业务，将数据反馈给调用者。

TxManager(通知事务)

当接受到TM的事务通知时，则会根据事务状态做对应的事务操作。

LCN代理连接池(提交事务)

当事务提交操作时，TC则会先创建对本次操作日志步骤的删除操作，对应关系型数据库来说就是对记录的业务SQL做删除操作，将该操作步骤也加入到待提交的本地事务中，然后一同与本地事务提交。当接受到的是回滚操作，那么则会将删除操作加入到事务中然后一同回滚，这样的方式主要是为了在事务的回滚或提交时都会将删除日志操作与本地事务一起处理。注意：在执行delete语句的时候会根据实际记录的日志数来判断是否执行成功，若没有执行成功则肯定是由于日志数据没有保存成功导致的影响行数不匹配，此时会继续保存日志，若2次尝试后依旧无法保存则直接抛出JDBC异常。当事务正在提交或回滚后也不会出现重复提交的问题，因为重复提交的执行日志已经在

事务提交是清理了，仅当需要补偿的时候才会保留本地的业务日志数据。当执行提交事务出现异常的时候也会重试，若重试依旧无法提交事务则直接作为补偿情况，此时将返回给TM事务须补偿的状态，也将取消对本地的超时任务。

LCN代理连接池(响应事务)

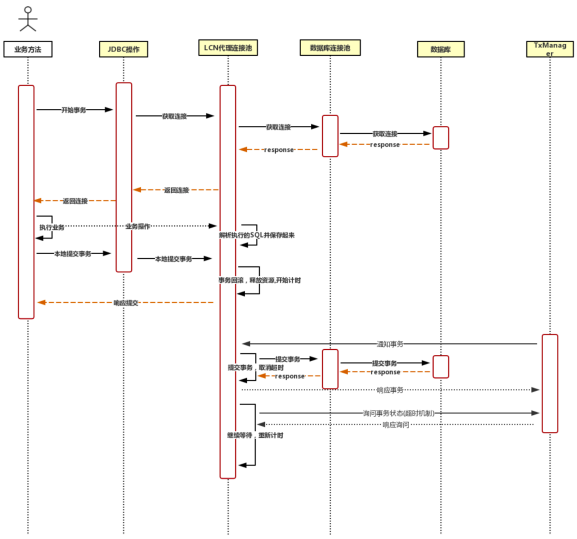
当LCN事务提交以后就开始响应事务通知给TM，通知TM本次事务已经完成。

LCN代理连接池(询问事务组)

若TM长时间没有请求到LCN代理连接池的话，那么LCN代理连接池将会触发超时请求机制，则会主动去联系TM，若访问不到TM，则会再重试访问一次请求，若还不能访问则会根据TM的集群部署特点，请求其他的TM节点，若全都访问不到，只可能是TM故障或者网络故障，这样的情况下LCN代理连接将执行回滚事务，但是不删除日志数据。当再次联系上TM时可检测日志数据请求状态再做事务提交或回滚请求。若可以联系上TM则TM会响应一个继续等待的消息，则TC将继续等待TM通知。因此设置TC的等待超时时间时可以是短暂的，因为在询问到TM若未完成的事务或者没有通知到的事务时TM都将通知TC继续等待。

当TC请求到的TM状态已经是补偿状态时，则会唤醒TM的继续请求指令，然后TC将继续进入计时等待状态。

2.4. TC代理控制原理(柔性事物)



柔性事务主要差异的地方在于本地事务提交和通知事务的差异上。

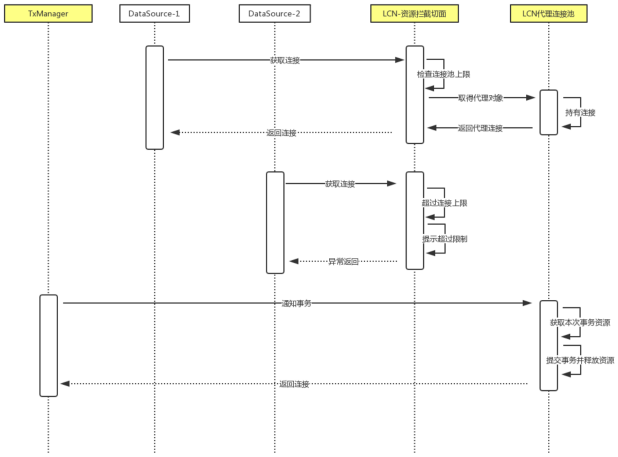
LCN代理连接池(事务回滚，释放资源，开始计时)

在本体事务提交的时候，会先通过TM获取到数据排他性的全局锁，然后在执行本地事务的回滚操作，然后再开始在本地事务中执行日志SQL存储提交，再执行定时任务计时。

LCN代理连接池(提交事务)

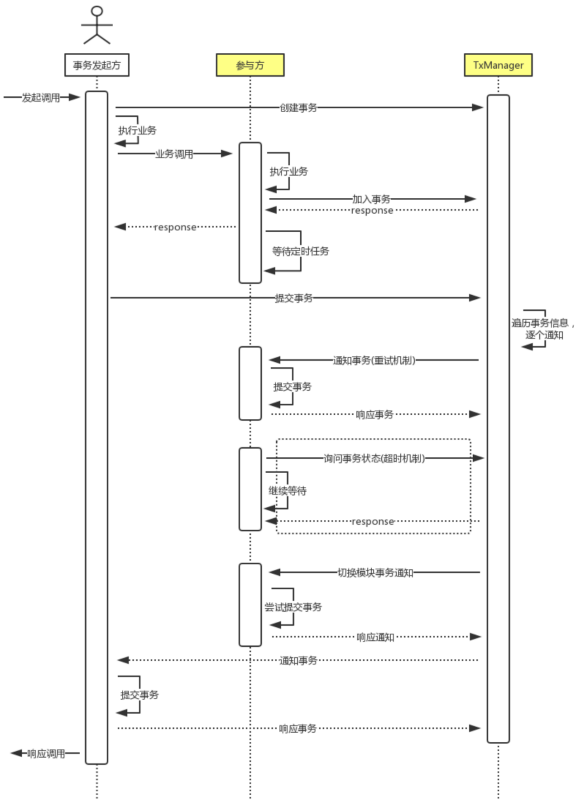
在TM通知事务以后将开始执行本地事务的提交，本地事务的提交将会查询出执行的SQL然后执行业务，再将插入SQL和删除日志的SQL在同一事物中一同执行。若是回滚操作的话，则直接就是删除执行的SQL，这只是与强一致差异性的地方，其他操作可参考强一致事务流程。

2.5. TC代理连接池优化



一个分布式事务的执行过程中会占用各个节点的资源，为了给非分布式事务的操作余留资源，在LCN的代理资源连接上可以设置上限值。那么单个模块参与分布式事务的资源连接将会被控制在这个范围内，且为了支持多数据源的操作，代理连接切面以及代理连接池是不会对数据源做限制，多数据源的业务也是可同时进入到代理控制中。在通知事务的步骤时将通过事务标示来获取满足条件的代理资源列表，然后逐步完成事务的提交并释放资源。

2.6. TM的原理介绍



步骤介绍

创建事务

创建事务标志着一个分布式事务的创建，创建事务组操作是由事务发起方发起的请求，在创建事务操作时将会创建一个全局事务标示id信息。

加入事务

加入事务是在TC执行完本地业务后开始调用的操作，该请求将携带传递过来的事务标示Id信息及操作日志记录标示信息给TM。

提交事务

提交事务是发起方本地执行完所有业务以后根据当前的事务状态判断而定的提交状态，若业务正常则提交commit请求给TM，若失败则提交rollback给TM，当然用户也可自行通过硬编程的方式控制事务的手动回滚。

遍历事务信息逐个通知

当TM接受到请求以后开始遍历该事务下的所有参与方，然后开始逐个通知事务的最终状态。

通知事务(重试机制)

通知各个节点事务状态，并确认节点执行成功以后方可标记状态再继续通知其他模块。若TC不响应数据，则会再重试通知一次若还不响应，则会寻找负载模块通知，若负载模块通知成功则标记通知成功，然后继续遍历参与方通知。若负载模块由于锁原因导致提交失败，则TM将继续通知记录节点，尝试直到超过超时时间，若到超时时间还未能请求通则不再尝试，标记为待补偿状态，并通知运维，再继续通知其他未通知的参与模块。

询问事务状态(超时机制)

超时机制会有二种情况：

1、访问不通TM ：

由于TM长时间未能通知到，此类情况往往是因为TM故障或网络导致的。由于TM也是集群状态，因此若TC超时访问不通时，则可切换在线的其他TM节点继续请求，若依旧请求不通则会回滚本地事务，但保留着本地的SQL记录数据。

2、TM正常响应：

若获取到了TM的在线消息，则延长超时时间继续等待TM下发指令。

切换模块事务通知

切换模块事务通知是由于上述所说在通知事务时2次不成功则会通知其他负载模块。通知到其他模块后则要求其模块对事务做对应事务状态操作，若可以操作成功则正常响应给TM。然后TM也将状态标记为已通知，然后再继续通知其他节点。若通知执行失败再重复上述的循环重试机制。

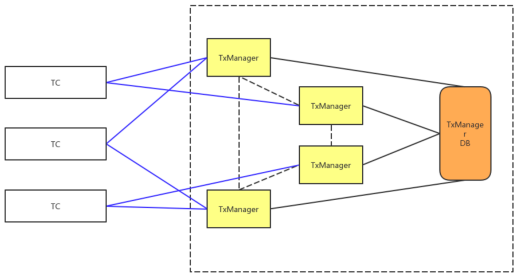
通知事务(发起方)：

在通知完成所有的事务以后则将通知事务发起方最终的事务通知结果。通知结果是提交或者是回滚也会影响到事务发起方最终操作，发起方事务执行完成以后则需反馈给TM提交完成消息。

响应事务(发起方)：

事务发起方完成了业务以后则需要响应消息给TM .同样若TM迟迟收不到响应也会向其他负载节点询问情况，操作机制与参与方一样，若TM始终接受不到发起方的成功提交消息也会记录补偿，并通知运维，但作为发起方则只需要发送完消息以后就可以直接响应数据给调用方了。

2.7. TM的高可用机制



各个TM之间将基于P2P网络互相彼此连接着，每当任何一个TC节点连接上来以后，TM都会通知他其他TM的节点信息，每个TC为了高可用必须连接2个以上的有效TM资源。当TC仅有一个连接对象时则会询问TM更多的资源，然后去连接保持多个在线资源。2个以上的资源配置可由用户自行配置数量，非强制根据实际的TM实例数量来定义。每当TM启动时会先检查本地的事务数据，通知响应模块对应的事务状态操作。TM也将拥有轮训监控机制，定时跟踪那些长时间没有完成的事务操作。

TM作为分布式事务的控制方，因此对TM的分布式部署与集群化将是重要的保障机制。

TM的事务数据需持久化到硬盘，并且可支持分布式部署。可采用redis集群、mysql集群等方案。

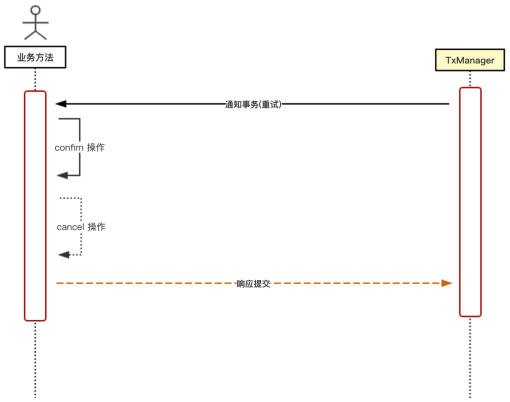
2.8. TC与TM的通讯方式

为了简化TC与TM的通讯消息量，他们之间直接基于TCP Socket的自定义通讯协议，可采netty[6]框架，每个TC配置资源连接一个或多个地址，当连接上以后将TM将会返回更多其他节点的TM连接地址，为了TC与TM的稳定性，他们将需要通过心跳数据包确保双方是否在线，当TC检测到连接断开以后，TC将要继续寻找其他连接，需保持在其稳定的连接数量要求之上。

3. 对NoSQL[7]的支持与扩展

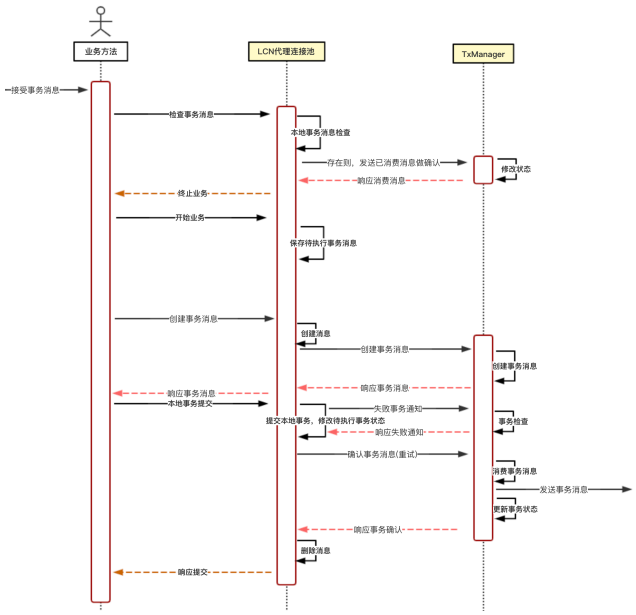
本文大量篇幅都是以关系型数据库为题展开的描述，实际LCN分布式解决方案不仅仅致力于解决关系型数据库，针对与NoSQL的目前我们有两种方案，一种是针对与NoSQL的TCC[8]方案、第二种是基于MQ[9]事务消息队列模式。

3.1. TCC事务



TCC的关键步骤：Try Confim Cancel ，Try操作实际在执行业务的时候就已经执行了。 Confim与Cancel则是在通知事务的时候由事务发起方来决定，当事务需要提交时则会通知所有参与节点提交事务，若事务迟迟不响应时则会重试一次，若依旧不能响应则会切换到其负责模块通知，若在超时时间内未能效应则TM会记录事务补偿，然后也会通知给运维。在事务协调阶段与LCN是一致的。

3.2. MQ事务消息



目前阿里的RabbitMQ[10]是支持事务消息的但是其他的MQ并不支持事务消息，为了统一事务消息则将基于TM来完成事务消息的封装。在事务消息模式下TM将充当消息的发送者的角色。

步骤介绍:

MQ消息(接受事物消息):

由于模块订阅了该消息，当模块接受事务消息后首先会先进入事务消息检查的任务。

LCN代理连接池(本地事务消息检查):

首先LCN代理层将先拦截到MQ消息，然后对其内容做检查，先查询本地的事务消息中是否已经包含本次消息，若已经包含则表明是业务已经执行过，若是已经执行过的状态则将会通知TM，若TM的状态未更改则会更改状态，若已更改则可无视该消息。若该消息在本地中尚未存在，则可直接返回业务，然后正常执行业务。

业务方法(开始业务):

业务方法将开始执行自己本地业务逻辑。

LCN代理连接池(保存待执行事务消息):

这里将会创建本地的事务消息语句，然后添加到要执行的事务中，但并不直接提交事务。

业务方法(创建事务消息):

该方法是指业务方法在执行业务过程中有需要发送事务消息给其他模块，这里是代表执行发送消息的代码块。

LCN代理连接池(创建消息):

当首先会在本地创建一条新的待发送的事务消息，这个操作也将与本地事务在一个事务中操作，事务的提交也将与本地事务的提交一致，若在执行插入操作时出现异常，则可能是有数据库悲观锁导致的异常，表明其他模块正在执行该业务，因此将抛出异常，提示回滚。当成功完成了本地消息的语句执行后将会去通知TM ,若TM接受不到消息，则会重试，若重试还不能接受到消息则会切换TM,若切换了以后依旧无法访问通，则直接抛出异常，本地事务也要回滚。

TxManager(创建事务消息):

TM将会也在本地存储一条事务消息数据。然后设置状态为待发送。该消息务必要通知到TC若通知不到，则TC最终将抛出业务异常。

LCN代理连接池(提交本地事务，修改待执行事务状态):

若事务正常执行完业务，则尝试提交本地事务。

LCN代理连接池(失败事务通知):

由于提交事务分为commit与rollback，当本次业务执行失败时，则会通知TM该业务已执行失败，若当TM接受到失败事务通知时，先判断事务状态是否已执行过，若执行过则TM将直接返回，若事务未执行则记录补偿，且将事务消息通知次数+1。这里通知不到TM则也不是特别重要，因为本次事务也是失败的已回滚。

思考:事务失败时为什么还可能会出现事务执行过的情况呢?

回答:虽然在执行业务前已经做了两次检查，但是若数据库采用的是乐观锁机制则会在这里出现异常，这也表明是数据已执行过的情况。

该通知中的内容因为业务具体执行的过程而定，若该步骤操作中即发起了一次事务消息，又执行了事务的消费，那么该失败事务通知中将提现出两个事务标示来。

LCN代理连接池(确认事务消息(重试)):

当事务本地执行成功则将会通知TM本次事务已经执行完毕，本次确认的事务消息内容也会根据实际执行的业务而定，1有无创建事务消息(可多条)，2有无消费事务消息。该操作将存在重试机制。这里通知不到TM则会做2次尝试，分别是二次尝试与切换模块尝试，若均无法访问则放弃请求，该步骤有无成功执行将会影响到下面的删除消息操作。

TxManager(确认事务消息 步骤):

TM接受到指令以后会根据内容来操作，若有确认消息的则会执行确认消息然后再发生事务消息，若是事务消费确认，则会修改事务状态为已执行。

为什么上面步骤说该步骤不成功不是特别影响呢?

1、当TM介绍不到TC的确认事务消息时，则TC也不会删除待发送事务消息的内容，那么此时TC依旧会通过定时机制触发对TM的调用。因此不必担心TM暂时性的接收不到消息。

2、若是未通知到消费事务消息，则由于事务已提交本地事务遗留痕迹，那么当TM再次通知到TC的时候也将不会重复执行任务。且此时TC也会再次通知TM变更状态。

当接受到确认发送事务消息时若该消息被重复调用过也会重复发送，由于我们做到了本地事务防止重复执行的操作，固不担心业务的重复调用。

LCN代理连接池(删除消息):

仅当TM成功响应TC确认事务消息步骤后才会执行删除业务，若极端情况下TC执行失败了，也不影响则重复发送消息给TM，TM再重复发送即可。

4. 总结

LCN系统在非极端场景下（TM宕机、TC宕机、网络异常、数据库宕机）支持事务的幂等性与强一致性，理论上其性能相比本地事务要逊色，但是相比逆向SQL方式及不可控代理模式来说更有优势。LCN框架可支持多种数据源的适配、对各种DB框架、各种数据库的兼容适配，且无需针对各种数据库做适配程序，也可支持存储过程的执行，而且可包容TCC、MQ等其他模式的兼容。

参考文献:

[1] BASE ACP :https://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/

[2] TIDB : https://pingcap.com/docs-cn/stable/overview/

[3] ACID : https://en.wikipedia.org/wiki/ACID

[4] NewSQL :https://en.wikipedia.org/wiki/NewSQL

[5] JAVA JDBC:https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

[6] Netty https://netty.io/

[7] NoSQL https://en.wikipedia.org/wiki/NoSQL

[8] TCC:https://www.enterpriseintegrationpatterns.com/patterns/conversation/TryConfirmCancel.html

[9] MQ:https://en.wikipedia.org/wiki/Message_queue

[10] RabbitMQ: https://en.wikipedia.org/wiki/RabbitMQ