事务处理设计文档

**2014年8月25日**

# 概述

在使用数据分片场景下， 单库下的事务处理无法满足系统的需求，因而需要进行分布式事务处理设计。

# 主要方案对比分析

处理分布式场景下的事务有很多种方案，主要方案如下表所示:

|  |  |  |  |
| --- | --- | --- | --- |
| **主要技术** | **优点** | **缺点** | **适用场景** |
| XA with 2PC(JTA)、 3PC、 | 事务管理最为严格，基本可以确保事务对开发人员透明 | 性能差实现复杂度高可扩展性差 | 对事务要求苛刻，直接面向DB |
| TCC | 事务管理较为严格，基本可以确保事务对开发人员透明 | 性能差可扩展性差实现复杂度高未开放成熟方案与源码 | 对事务要求苛刻，直接面向DB |
| Best Efforts 1PC等 | 事务管理较为严格，基本可以确保事务对开发人员透明 | 性能差可扩展性差实现复杂度高未开放成熟方案与源码 | 对事务要求苛刻，直接面向DB |
| Transaction Conpensation | 采用最终一致性，性能高可扩展性好切换Proxy时，无须修改程序源码 | 对开发人员不透明，需要较多的编码发生异常时可能需要人工干预数据一致性的实时性较弱 | 数据分片有Proxy长事务 |

根据以上对比结果，对于与Mycat结合的事务处理，采用事务补偿方式对事务进行管理。

# 设计

## 设计难点

使用事务补偿方式对事务进行管理的技术难点主要体现在以下两个方面：

1. 失败处理

当事务提交失败时，需要进行回滚操作，如果回滚失败，需要对回滚失败进行处理，而每一步处理都可能继续失败，此问题类似于“拜占庭将军问题”。

1. 事务的隔离

事务执行过程中，需要对事务相关的数据进行隔离，如何有效的隔离，需要根据实际场景，对业务进行加锁。

1. 混合事务

在Mycat中，主子表间可以配置刚性事务，如果刚性事务与柔性事务混合在一起，处理比较困难。

## 设计说明

事务补偿方式采用BASE事务模型，使用事务补偿方式对事务进行管理，牺牲了高一致性，获得高性能，提升了可用性，事务补偿方式管理的事务可以视为一种柔性事务。

采用BASE事务与ACID事务的比较：

|  |  |  |
| --- | --- | --- |
|  | **ACID** | **BASE** |
| 原子性 | 则成功则全部成功；若失败，则全部回滚 | 可能全部成功，也可能部分成功。如果失败，则回滚，若回滚失败，进行异常处理（进行监控并人工处理） |
| 一致性 | 在事务开始或结束时，数据库应该在一致状态 | 在事务开始或结束时，数据库可能在一致状态，也可能不在一致状态，但最终将达到一致状态 |
| 隔离性 | 在数据库事务进行隔离 | 采用业务锁进行隔离或不隔离 |
| 持久化 | 若事务完成，则不能返回 | 若事务完成，则不能返回 |

BASE事务处理设计包含三部分：事务管理接口，事务监控任务&异常事务展现，以及校对任务，如下图所示:



对于混合事务，将事务切分为两个事务，对刚性事务不进行回滚，特殊处理之。

**事务处理接口伪代码**(部分伪码，仅供参考):

Try{

 记业务Log

While(有待处理SQL){

 判断是否有业务锁，若有业务锁则加业务锁

 执行SQL

 反写执行成功标记位

}

获取待处理锁（锁的顺序与业务SQL顺序相反）

While(有待处理锁){

 解锁

}

}catch(异常){

记录Log

回滚并解锁

抛出异常或返回错误信息

}finally{

记录快照

后续处理

}

function 记业务Log(String SQL){

 写log

 若写失败，抛TLogException

}

function 加锁(String SQL){

 加锁

 若写失败，则抛出TLockException

}

function回滚并解锁(List<RollableSQL> sqls){

 回滚

 若失败，则抛出TLockException

}

## 风险与规避措施

### 风险

**隔离性风险：**

1, 对于部分无法隔离的数据，可能造成业务数据的错误，如减库存失败且无法隔离的情况下，可能造成超卖

2, SQL执行失败以后，系统有新数据进入系统，当重新执行时，可能影响后进入系统的数据。

**一致性风险：**

部分不一致场景客户可能无法忍受。

**非幂等SQL风险：**

对于非幂等sql, 多次执行将造成数据错误。

### 风险规避方案

通过数据校对任务检测错误数据。

## 约束

单据的删除需要使用逻辑删除

查询业务数据时需要区分被锁定的数据与未被锁定的数据