开放 / Open　平等 / Equal　协作 / Cooperation　分享 / Share

分布式事务的解决方案及讨论

财产险财物理赔组 2017/12

分布式事务解决方案

目录 Contents

1 paxos

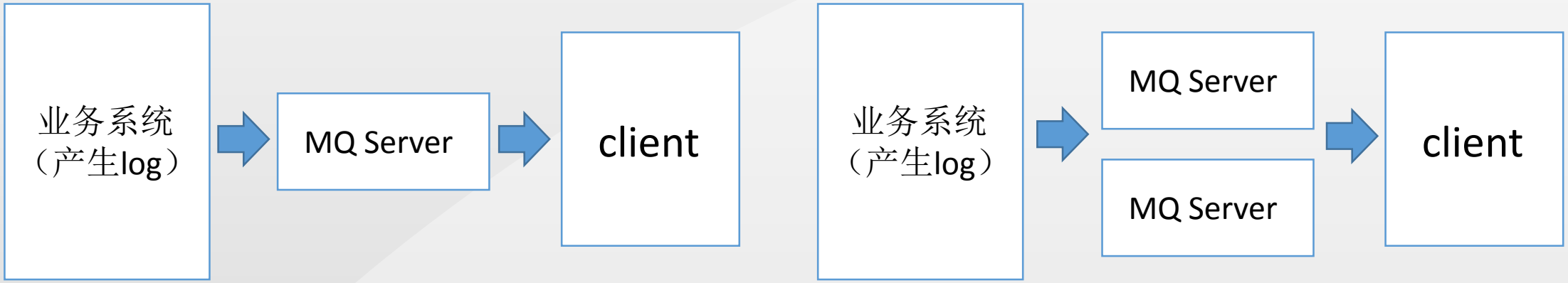
2 TCC和消息队列

3 LCN

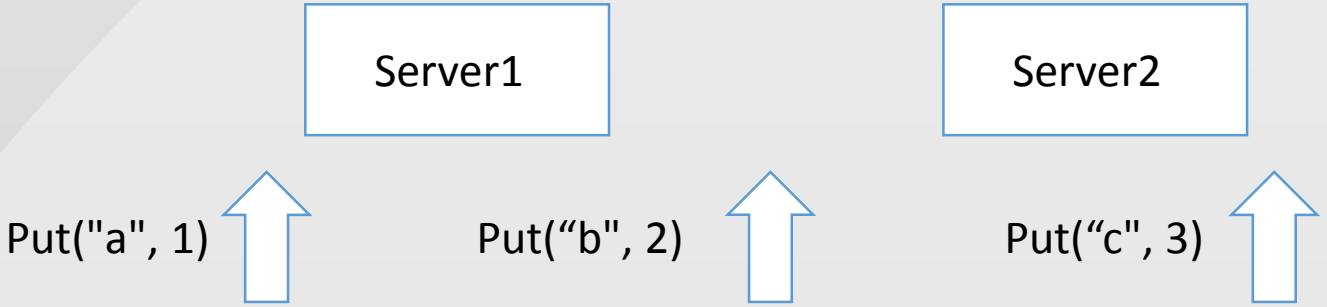
应用场景

状态机场景：每个处室状态一致的状态机上执行一串命令后状态都必须相互一致，即顺序一致性

日志服务器



Db数据更新



执行过程

Proposer 提出提案，提案信息包括提案编号和提议的value;
Acceptor 收到提案后可以接受(accept)提案;
Learner 只能“学习”被批准的议案

(决议的提出与批准)主要分为两个阶段:

1. prepare阶段：

(1). 当Proposer希望提出方案V1，首先发出prepare请求至大多数Acceptor。Prepare请求内容为序列号<SN1>;

(2). 当Acceptor接收到prepare请求<SN1>时，检查自身上次回复过的prepare请求<SN2>

a). 如果SN2>SN1，则忽略此请求，直接结束本次批准过程;

b). 否则检查上次批准的accept请求<SNx，Vx>，并且回复<SNx，Vx>；如果之前没有进行过批准，则简单回复<OK>;

2. accept批准阶段：

(1a). 经过一段时间，收到一些Acceptor回复，回复可分为以下几种:

a). 回复数量满足多数派，并且所有的回复都是<OK>，则Porposer发出accept请求，请求内容为议案<SN1，V1>;

b). 回复数量满足多数派，但有的回复为：<SN2，V2>，<SN3，V3>……则Porposer找到所有回复中SN编号最大的的那个，假设为<SNx，Vx>，则发出accept请求，请求内容为议案<SN1，Vx>;

c). 回复数量不满足多数派，Proposer尝试增加序列号为SN1+，转1继续执行;

(1b). 经过一段时间，收到一些Acceptor回复，回复可分为以下几种:

a). 回复数量满足多数派，则确认V1被接受;

b). 回复数量不满足多数派，V1未被接受，Proposer增加序列号为SN1+，转1继续执行;

(2). 在不违背自己向其他proposer的承诺的前提下，acceptor收到accept 请求后即接受并回复这个请求。

Acceptor 收到 prepare 消息后，如果提案的编号大于它已经回复的所有 prepare 消息，则 Acceptor 将自己上次接受的提案回复给 Proposer，并承诺不再回复小于 n 的提案；

提议ID	提议值	Acceptor				
		A	B	C	D	E
2	a	x	x	x	o	-
5	b	x	x	o	-	x
14	?	-	o	-	x	o
27	?	o	-	o	o	-
29	?	-	o	x	x	-

缺点：
Proposer全局编号
活锁：多个proposer竞争提出提案

应用：
Zookeeper
chubby

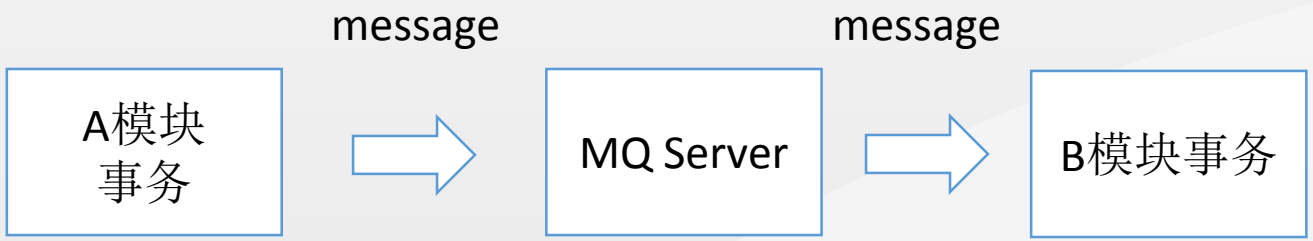
执行原理

TCC:
Try -> Confirm ->Cancel

	Account	Product
Try	T_USER表： balance - 5	T_PRODUCT表 : stock - 1
	T_USER_BALANCE_TCC表 :新增记录，status=1（user_id关联）	T_PRODUCT_TCC表 : 新增记录，status=0（product_id关联）
Confirm	T_USER_BALANCE_TCC表： status=1	T_PRODUCT_TCC表 : status=1
Cancel	T_USER表： balance + 5	T_PRODUCT表 : stock + 1
	T_USER_BALANCE_TCC表： 删除原来记录	T_PRODUCT_TCC表： 删除原来记录

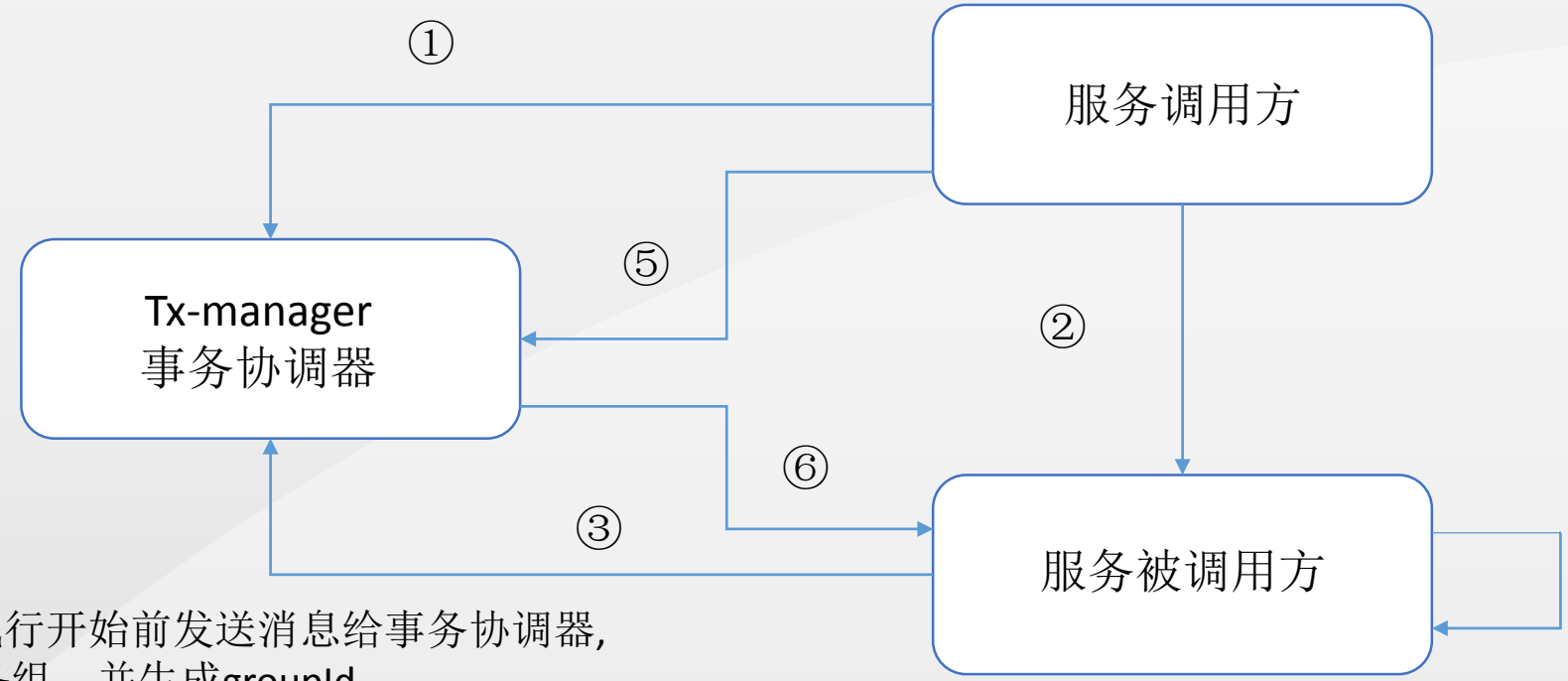
缺点：
1. 开发成本较高，每一个方法都要写try，confirm，cancel三个阶段的方法
2. Try和cancel用put方法，（如balance-5，stock-1），与现状不符

执行原理



业务数据的提交和消息的发送需要为原子服务，也就是
A模块业务数据的事务提交和消息数据的事务提交必须在一个事务里

运行原理



1. 包裹的AOP方法在方法执行开始前发送消息给事务协调器，告诉事务协调器创建事务组，并生成groupId
2. 服务调用方调用服务被调用方，并将groupId和超时时间放入请求头
3. 服务被调用方发送消息给事务协调器，告诉本次服务加入事务组groupId
4. 服务被调用方通过劫持连接池，堵塞住commit的线程，等待事务协调器唤醒
5. 在所有服务被调用方都执行完毕之后，服务调用方发送消息给事务协调器，告知所有方法都正确执行完毕，并关闭当前事务组
6. 事务协调器发送消息给各组件，唤醒各组件之前堵塞的commit线程

核心

连接池拦截过程

代理

LCNTransactionDataSource ⇒ DataSource

LCNDBConnection ⇒ Connection

事务开始：

DatasourceTransactionManager.getTransaction()

↓

DatasourceTransactionManager.doBegin()

↓

LCNTransactionDataSource.getConnection()

返回LCNDBConnection

事务结束：

DataTransactionManager.commit()

↓

DataTransactionManager.processCommit()

↓

DataTransactionManager.doCommit()

↓

LCNDBConnection.commit()

在commit方法里面，执行线程的等待，

并根据最后事务协调器的返回信息，

执行commit或者rollback

补偿

情景：

1. 网络抖动
2. 宕机

```
//start 结束就是全部事务的结束表示,考虑start挂掉的情况
Timer timer = new Timer();
timer.schedule(() -> {
        System.out.println("auto execute ,groupId:" + getGroupId());
        dataSourceService.schedule(getGroupId(), waitTask);
}, getMaxOutTime());

System.out.println("transaction is wait for TxManager notify, groupId : "

waitTask.awaitTask();

timer.cancel();

int rs = waitTask.getState();

System.out.println("lcn transaction over, res -> groupId:"+getGroupId()+"

if (rs == 1) {
    connection.commit();
} else {
    connection.rollback();
}
waitTask.remove();

}
```

Commit时线程等待

线程唤醒后根据status决定是commit还是rollback

补偿的执行逻辑：

```
int rs  = txManagerService.closeTransactionGroup(groupId, state);

long end = System.currentTimeMillis();

final long time = end - start;

if (TxCompensateLocal.current() == null) {
    if (state == 1 && rs == 0) {
        new Thread((HookRunnable) () -> {
                //记录补偿日志
                txManagerService.sendCompensateMsg(groupId, time, info);

        }).start();

    }
}
TxTransactionLocal.setCurrent(null);
logger.info("<---end start transaction");
logger.info("start transaction over, res -> groupId:"+txGroup.getGroupId()
```

所有组件执行完毕，调用方向事务协调器请求关闭事务组

判断关闭事务组的返回值rs，如果为0（即无法通知到所有组件时），执行补偿，发送补偿消息给事务协调器

优势及改进

优势

1. 开发成本低
2. 保证数据一致性

改进空间

1. Spring事务机制的优化(runtime和error异常)
2. 负载均衡的优化

Q&A

开放 / Open

平等 / Equal

协作 / Cooperation

分享 / Share

Thanks !