链滴

RocketMQ 进阶 - 事务消息

作者：jianzh5

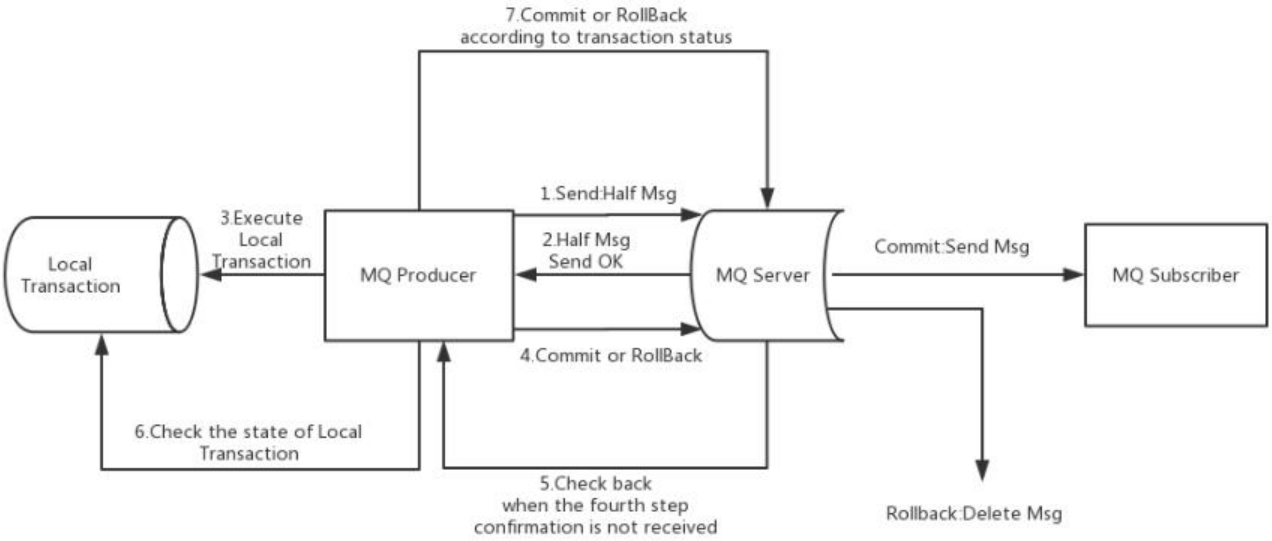
原文链接：https://ld246.com/article/1585790531772

来源网站：链滴

前言

分布式消息选型的时候是否支持事务消息是一个很重要的考量点，而目前只有RocketMQ对事务消息持的最好。今天我们来唠唠如何实现RocketMQ的事务消息！

Apache RocketMQ在4.3.0版中已经支持分布式事务消息，这里RocketMQ采用了2PC的思想来实现提交事务消息，同时增加一个补偿逻辑来处理二阶段超时或者失败的消息，如下图所示。



RocketMQ事务流程概要

RocketMQ实现事务消息主要分为两个阶段：正常事务的发送及提交、事务信息的补偿流程

整体流程为：

- 正常事务发送与提交阶段

1、生产者发送一个半消息给MQServer（半消息是指消费者暂时不能消费的消息）

2、服务端响应消息写入结果，半消息发送成功

3、开始执行本地事务

4、根据本地事务的执行状态执行Commit或者Rollback操作

- 事务信息的补偿流程

1、如果MQServer长时间没收到本地事务的执行状态会向生产者发起一个确认回查的操作请求

2、生产者收到确认回查请求后，检查本地事务的执行状态

3、根据检查后的结果执行Commit或者Rollback操作

补偿阶段主要是用于解决生产者在发送Commit或者Rollback操作时发生超时或失败的情况。

RocketMQ事务流程关键

1、事务消息在一阶段对用户不可见

事务消息相对普通消息最大的特点就是一阶段发送的消息对用户是不可见的，也就是说消费者不能直消费。这里RocketMQ的实现方法是原消息的主题与消息消费队列，然后把主题改成 RMQ_SYS_TRAS_HALF_TOPIC ，这样由于消费者没有订阅这个主题，所以不会被消费。

2、如何处理第二阶段的失败消息?

在本地事务执行完成后会向MQServer发送Commit或Rollback操作，此时如果在发送消息的时候生者出故障了，那么要保证这条消息最终被消费，MQServer会像服务端发送回查请求，确认本地事务执行状态。

当然了rocketmq并不会无休止的的信息事务状态回查，默认回查15次，如果15次回查还是无法得知务状态，RocketMQ默认回滚该消息。

3、消息状态

事务消息有三种状态：

- **TransactionStatus.CommitTransaction**：提交事务消息，消费者可以消费此消息
- **TransactionStatus.RollbackTransaction**： 回滚事务，它代表该消息将被删除，不允许被消费。
- **TransactionStatus.Unknown** ： 中间状态，它代表需要检查消息队列来确定状态。

实现

我们构建这样一个需求：用户请求订单微服务 order-service 接口删除订单（退货），删除订单后需发送消息给用户服务account-service，用户微服务收到消息后会给用户账户增加余额。

这个需求跟钱相关，肯定要保证消息的事务性，接下来我们根据上面的原理实现整个流程。

基础配置

生产者order-servcie和account-service都要引入RocketMQ相关依赖，增加RocketMQ的相关配置

- 引入组件

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
</dependency>
```

- 添加配置

```
# within rocketmq
rocketmq:
  name-server: xxx.xx.x.xx:9876; xxx.xx.x.xx:9876
  producer:
    group: cloud-group
```

发送半消息

order-service在执行删除订单操作时发送一条半消息给MQServer，发送半消息主要是使用rocketMTemplate.sendMessageInTransaction() 方法，发送事务消息。

```
@Override
public void delete(String orderNo) {
    Order order = orderMapper.selectByNo(orderNo);
    //如果订单存在且状态为有效，进行业务处理
    if (order != null && CloudConstant.VALID_STATUS.equals(order.getStatus())) {
```

```
        String transactionId = UUID.randomUUID().toString();
        //如果可以删除订单则发送消息给rocketmq，让用户中心消费消息
        rocketMQTemplate.sendMessageInTransaction("add-amount",
                MessageBuilder.withPayload(
                        UserAddMoneyDTO.builder()
                                .userCode(order.getAccountCode())
                                .amount(order.getAmount())
                                .build()
                )
                .setHeader(RocketMQHeaders.TRANSACTION_ID, transactionId)
                .setHeader("order_id",order.getId())
                .build()
                ,order
        );
    }
}
```

首先先校验一下订单状态，然后发送消息给MQServer，这个逻辑大家都看得懂，主要是关注sendMessageInTransaction() 方法，源码如下：

```
public TransactionSendResult sendMessageInTransaction(String destination, Message<?> message, Object arg) throws MessagingException {
    try {
        if (((TransactionMQProducer)this.producer).getTransactionListener() == null) {
            throw new IllegalStateException("The rocketMQTemplate does not exist TransactionListener");
        } else {
            org.apache.rocketmq.common.message.Message rocketMsg = this.createRocketMqMssage(destination, message);
            return this.producer.sendMessageInTransaction(rocketMsg, arg);
        }
    } catch (MQClientException var5) {
        throw RocketMQUtil.convert(var5);
    }
}
```

该方法有三个参数：

- destination：目的地(主题)，这里发送给 add-amount 这个主题
- message：发送给消费者的消息体，需要使用 MessageBuilder.withPayload() 来构建消息
- arg：参数

注意，这里我们生成了一个transactionId，并放在header中跟消息一起发送（这里实际也可以构造一个对象，放在arg里进行发送），作用后面再讲！

执行本地事务与回查

MQServer收到半消息后会告诉生产者order-service确认收到半消息，这时候order-service需要执本地事务，执行完本地事务后再告诉MQServer本地事务的执行状态，确认消息究竟是Commit还是Rollback。如果在告诉MQServer本地执行状态的时候出异常了还需要让MQServer能够回查到，怎么实这一些列操作呢？

RocketMQ提供了 RocketMQLocalTransactionListener 接口，本地事务监听器，这个接口类的实如下：

```
package org.apache.rocketmq.spring.core;

import org.springframework.messaging.Message;

public interface RocketMQLocalTransactionListener {
    RocketMQLocalTransactionState executeLocalTransaction(Message var1, Object var2);

    RocketMQLocalTransactionState checkLocalTransaction(Message var1);
}
```

第一个方法executeLocalTransaction 为执行本地事务；

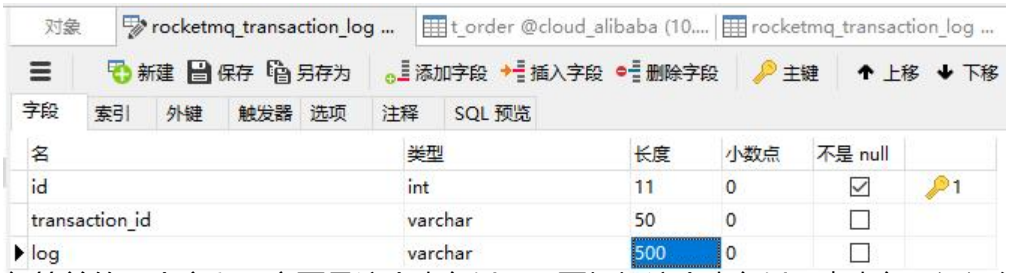
第二个方法checkLocalTransaction 为检查本地事务的执行状态，也就是回查动作。

有了这个接口类我们的执行逻辑清楚了，但是还有个问题：本地事务已经执行完成了，怎么去回查本事务的执行结果呢？

我们可以在执行本地事务的时候同时生成一个事务日志，让本地事务与日志事务在同一个方法中，同添加@Transactional 注解，保证两个操作事务是一个原子操作。**这样如果事务日志表中有这个本地务的信息，那就代表本地事务执行成功，需要Commit，相反如果没有对应的事务日志，则表示没执成功，需要Rollback**

思路既然理顺了，咱们就开撸。

- 首先创建一个日志表



很简单的三个字段，主要是这个事务id，需要根据这个事务id回查事务，还记得我们在发送半消息时成的事务id吗，就是干这个用的！

- 在生产者编写方法实现 RocketMQLocalTransactionListener

```
@Slf4j
@RocketMQTransactionListener
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class AddUserAmountListener implements RocketMQLocalTransactionListener {
    private final OrderService orderService;
    private final RocketMqTransactionLogMapper rocketMqTransactionLogMapper;
    /**
     * 执行本地事务
     */
    @Override
    public RocketMQLocalTransactionState executeLocalTransaction(Message message, Object
arg) {
        log.info("执行本地事务");
        MessageHeaders headers = message.getHeaders();
        //获取事务ID
        String transactionId = (String) headers.get(RocketMQHeaders.TRANSACTION_ID);
        Integer orderId = Integer.valueOf((String)headers.get("order_id"));
        log.info("transactionId is {}, orderId is {}",transactionId,orderId);

        try{
            //执行本地事务，并记录日志
            orderService.changeStatuswithRocketMqLog(orderId, CloudConstant.INVALID_STATUS
transactionId);
            //执行成功，可以提交事务
            return RocketMQLocalTransactionState.COMMIT;
        }catch (Exception e){
            return RocketMQLocalTransactionState.ROLLBACK;
```

```
        }
    }

    /**
     * 本地事务的检查，检查本地事务是否成功
     */
    @Override
    public RocketMQLocalTransactionState checkLocalTransaction(Message message) {

        MessageHeaders headers = message.getHeaders();
        //获取事务ID
        String transactionId = (String) headers.get(RocketMQHeaders.TRANSACTION_ID);
        log.info("检查本地事务,事务ID:{}",transactionId);
        //根据事务id从日志表检索
        QueryWrapper<RocketmqTransactionLog> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("transaction_id",transactionId);
        RocketmqTransactionLog rocketmqTransactionLog = rocketMqTransactionLogMapper.se
ectOne(queryWrapper);
        if(null != rocketmqTransactionLog){
            return RocketMQLocalTransactionState.COMMIT;
        }
        return RocketMQLocalTransactionState.ROLLBACK;
    }
}
```

- 执行本地事务的方法

```
@Transactional(rollbackFor = RuntimeException.class)
@Override
public void changeStatuswithRocketMqLog(Integer id,String status,String transactionId){
    //将订单状态置位无效
    orderMapper.changeStatus(id,status);
    //插入事务表
    rocketMqTransactionLogMapper.insert(
            RocketmqTransactionLog.builder()
                    .transactionId(transactionId)
                    .log("执行删除订单操作")
            .build()
    );
}
```

这一块的代码逻辑都是在生产端，即Order-Server，大家不要搞错了

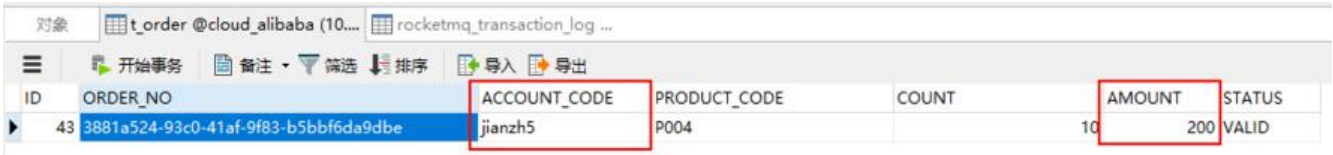
消费消息

Rollback的消息MQServer会给我们处理，我们只要关注Commit状态时消费端可以正常消费即可。在ccount-service监听消息，如果收到消息则给用户账户增加余额。

```
@Slf4j
@Service
@RocketMQMessageListener(topic = "add-amount",consumerGroup = "cloud-group")
@RequiredArgsConstructor(onConstructor = @__(@Autowired) )
public class AddUserAmountListener implements RocketMQListener<UserAddMoneyDTO> {
    private final AccountMapper accountMapper;
```

```
    /**
     * 收到消息的业务逻辑
     */
    @Override
    public void onMessage(UserAddMoneyDTO userAddMoneyDTO) {
        log.info("received message: {}",userAddMoneyDTO);
        accountMapper.increaseAmount(userAddMoneyDTO.getUserCode(),userAddMoneyDTO
getAmount());
        log.info("add money success");
    }
}
```

测试



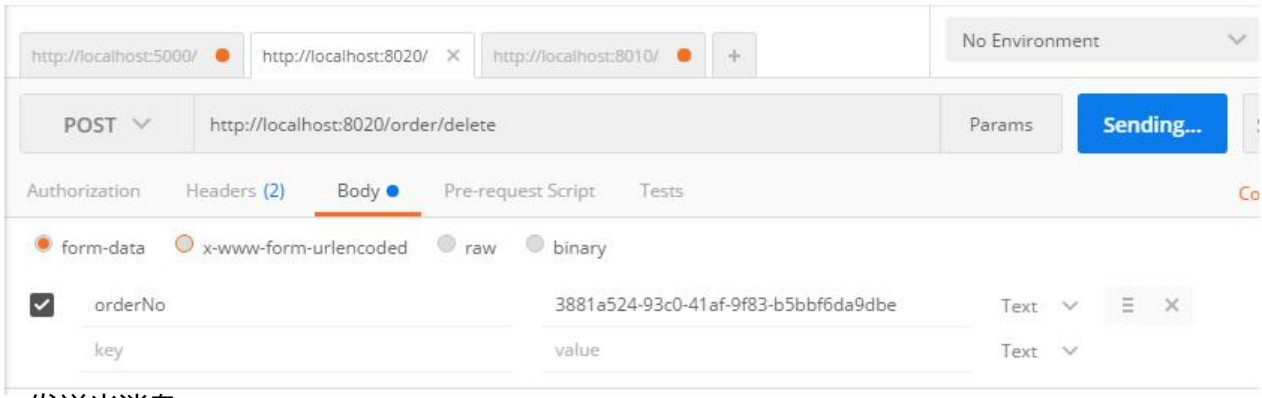
ID	ORDER_NO	ACCOUNT_CODE	PRODUCT_CODE	COUNT	AMOUNT	STATUS
43	3881a524-93c0-41af-9f83-b5bbf6da9dbe	jianzh5	P004	10	200	VALID

订单表有这样一条记录，用户为jianzh5，amount为200

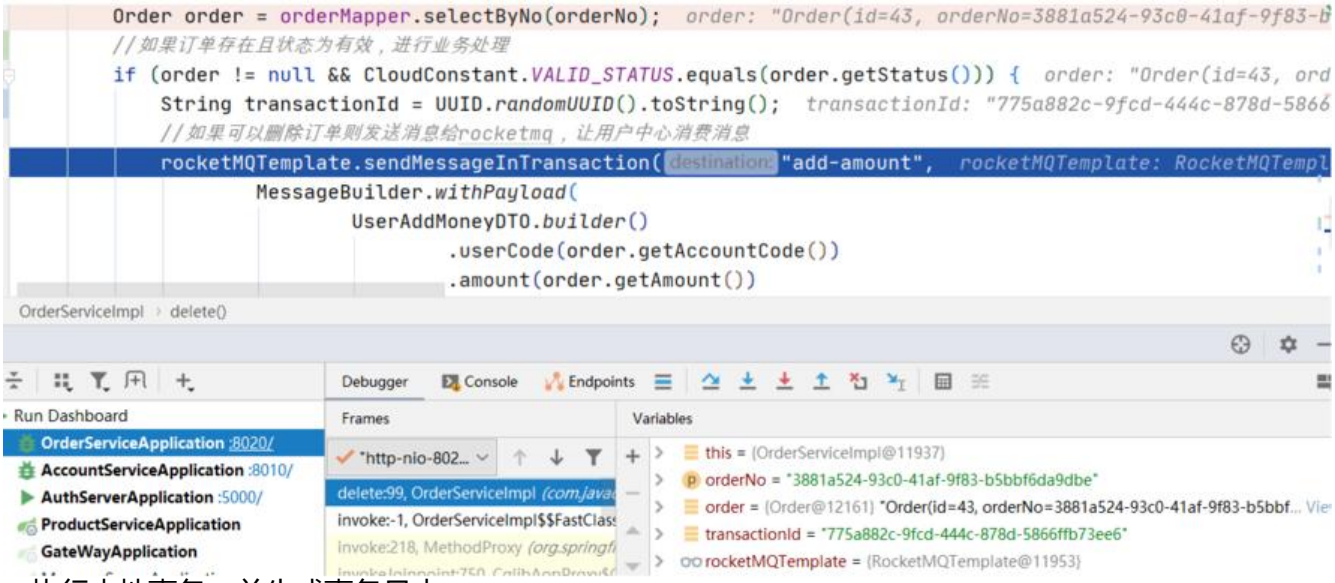
ID	ACCOUNT_CODE	ACCOUNT_NAME	AMOUNT
1	javadaily	JAVA日知录	9847
17	jianzh5	jianzh5	50

用户表的记录，执行完成后jianzh5的账户应该变成250

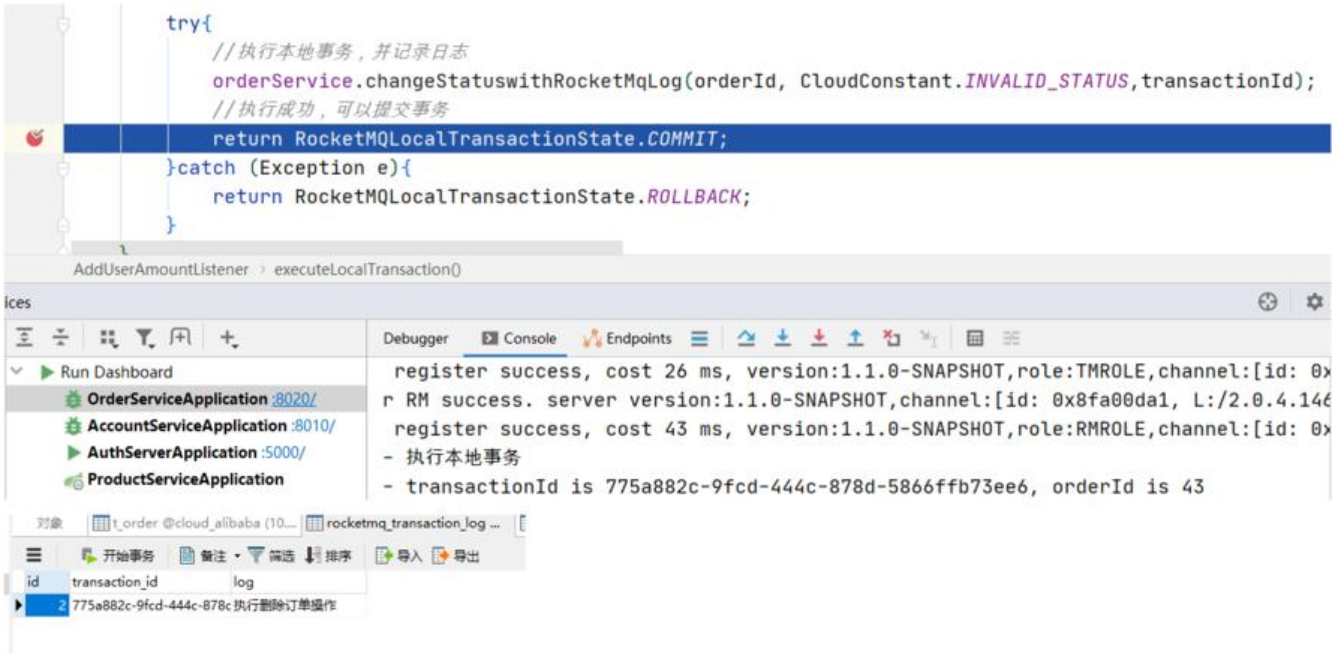
- 调用删除订单接口，删除订单



- 发送半消息



- 执行本地事务，并生成事务日志

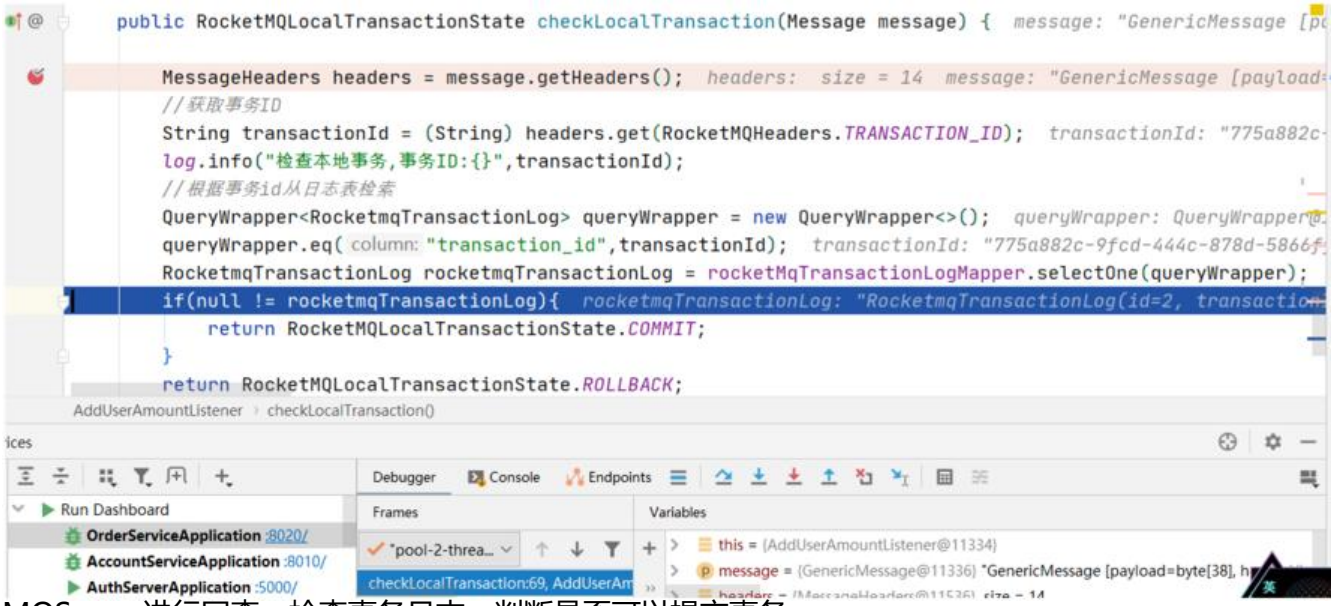


- 模拟异常情况

在发送Commit消息的时候我们用命令杀掉进程taskkill /pid 19748 -t -f，模拟异常！

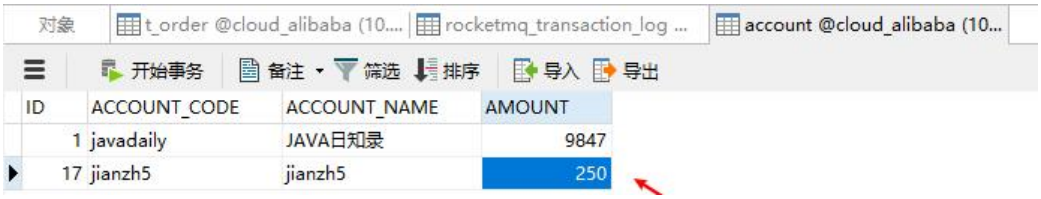
```
D:\project_jianzh5\cloud-aliaba\order-biz\order-service>jps
15652 RemoteMavenServer
19748 OrderServiceApplication
20644 AuthServerApplication
21140 Launcher
24116 Jps
13768
5704 AccountServiceApplication

D:\project_jianzh5\cloud-aliaba\order-biz\order-service>taskkill /pid 19748 -t -f
成功：已终止 PID 19640（属于 PID 19748 子进程）的进程。
成功：已终止 PID 19748（属于 PID 13768 子进程）的进程。
```

- 重新启动order-service，查看是否会执行回查动作



MQServer进行回查，检查事务日志，判断是否可以提交事务

- 消费者消费事务消息，保证事务的一致性



总结

使用RocketMQ实现事务消息的过程还是很复杂的，需要好好理解开头的那张图，只有理解了事务消的交互过程才能编写相应的代码！