RocketMQ发送消息之事务消息

先写库还是先发消息?

首先，咱们来看一下工作场景，订单ID1030被创建后要保存到数据库，同时该1030订单通过MQ投递给其他系统进行消费。如果要保证订单数据入库与消息投递状态要保证最终一致，要怎么做？这里有两种常见做法：

第一种，先写库，再发送数据

```
//伪代码
//插入1030号订单
orderDao.insert(1030,order);
//向1030号订单新增3条订单明细，10081-10083，
orderDetailDao.insert(10081,1030,orderDetail1);
orderDetailDao.insert(10082,1030,orderDetail2);
orderDetailDao.insert(10083,1030,orderDetail3);
//向MQ发送数据，如果数据发送失败
SendResult result = producer.send(orderMessage)
if(result.getState().equals("SEND_OK"))){
   connection.commit();
}else{
   connection.rollback();
}
```

如果生产者发送消息时，因为网络原因导致10秒消息才返回SendResult结果，这就意味这10秒内数据库事务无法提交，大量并发下，数据库连接资源会在这10秒内迅速耗尽，后续请求进入连接池等待状态，最终导致系统停止响应。

第二种，先发消息，再写库

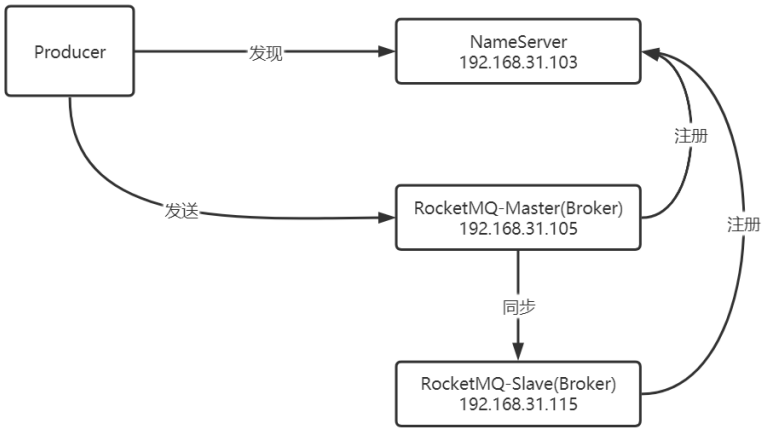
```
//伪代码
//向MQ发送数据，如果数据发送失败
SendResult result = producer.send(orderMessage)
if(result.getState().equals("SEND_OK"))){
   //插入1030号订单
   orderDao.insert(1030,order);
   //向1030号订单新增3条订单明细，10081-10083，
   orderDetailDao.insert(10081,1030,orderDetail1);
   orderDetailDao.insert(10082,1030,orderDetail2);
  orderDetailDao.insert(10083,1030,orderDetail3);
  connection.commit;
}
```

问题更严重，因为消息已经被发送了，消费者可以立即消费，比如下游消费者为1030订单自动设置了“快递信息”，可是如果后续orderDao向数据库插入数据产生异常导致业务失败。我们还需要再次发送“取消1030订单”的消息把下游1030订单分配的“快递信息”给撤销，这些都是在业务层面上的额外处理，这无疑提高了对程序员的要求与处理的难度。

那有没有什么方式可以既不阻塞数据库事务，也能保证最终一致性呢？有，RocketMQ提供了事务消息可以保障应用本地事务与MQ最终一致性。

案例实践

架构拓扑



代码分析

MessageType4-发出事务消息代码

```
public class MessageType4 {
    public static void main(String[] args) throws MQClientException, InterruptedException,
UnsupportedEncodingException {

        //事务消息一定要使用TransactionMQProducer事务生产者创建
        TransactionMQProducer producer = new
TransactionMQProducer("transaction_producer_group");
        //从NameServer获取配置数据
        producer.setNamesrvAddr("192.168.31.103:9876");

        //CachedThreadPool线程池用于回查本地事务状态
        ExecutorService cachedThreadPool = Executors.newCachedThreadPool(new
ThreadFactory() {
            @Override
            public Thread newThread(Runnable r) {
                Thread thread = new Thread(r);
                thread.setName("check-transaction-thread");
                return thread;
            }
        });
        //将生产者与线程池绑定
        producer.setExecutorService(cachedThreadPool);
        //绑定事务监听器，用于执行代码
        TransactionListener transactionListener = new OrderTransactionListenerImpl();
        producer.setTransactionListener(transactionListener);
        //启动生产者
        producer.start();

        //创建消息对象
```

```
        Message msg =
                new Message("order","order-1030" ,
                        "1030", "1030订单与明细的完整JSON数据（略）".getBytes());
        //一定要调用sendMessageInTransaction发送事务消息
        //参数1：消息对象
        //参数2：其他参数，目前用不到
        producer.sendMessageInTransaction(msg, null);
    }
}
```

TransactionListenerImpl-处理本地事务业务代码

```
public class OrderTransactionListenerImpl implements TransactionListener {
    @Override
    //执行本地事务代码
    public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
        log.info("正在执行本地事务,订单编号:" + msg.getKeys());
        /* 伪代码
        try{
            //插入1030号订单
            orderDao.insert(1030,order);
            //向1030号订单新增3条订单明细，10081-10083，
            orderDetailDao.insert(10081,1030,orderDetail1);
            orderDetailDao.insert(10082,1030,orderDetail2);
            orderDetailDao.insert(10083,1030,orderDetail3);
            connection.commit();
            //返回Commit，消费者可以消费1030订单消息
            return LocalTransactionState.COMMIT_MESSAGE;
        }catch(Exception e){
            //返回Rollback，Broker直接将数据删除，消费者不能收到1030订单消息
            connection.rollback();
            return LocalTransactionState.ROLLBACK_MESSAGE;
        }
        */
        log.info("模拟网络中断，Broker并未收到生产者本地事务状态回执，返回UNKNOW");
        return LocalTransactionState.UNKNOW;
    }

    @Override
    //会查本地事务处理状态
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {

        String keys = msg.getKeys();
        log.info("触发回查，正在检查" + keys + "订单状态");
        /* 伪代码

        Order order = orderDao.selectById(1030);
        if(order != null){
            //查询到记录，代表数据库已处理成功，回查返回Commit，消费者可以消费1030订单消
息
```

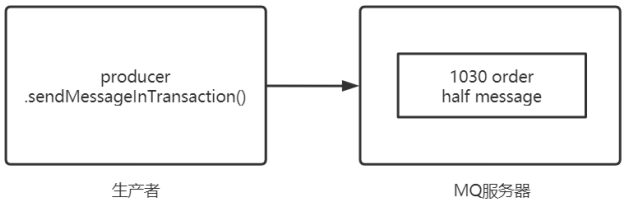
```
            return LocalTransactionState.COMMIT_MESSAGE;
        }else{
            //未查询到记录，代表数据库处理失败，回查返回Rollback，Broker直接将数据删除，消费者不能收到1030订单消息
            return LocalTransactionState.ROLLBACK_MESSAGE;
        }
        */
        log.info("回查结果，" + keys + "订单已入库，发送Commit指令");
        return LocalTransactionState.COMMIT_MESSAGE;
    }
}
```

实验了解RocketMQ事务执行过程

标准流程

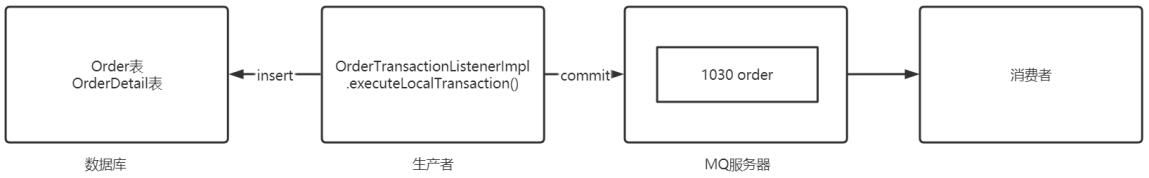
1. producer.sendMessageInTransaction(msg, null); 执行成功

此时1030订单消息已被发送到MQ服务器（Broker），不过该消息在Broker此时状态为“half-message”，相当于存储在MQ中的“临时消息”，此状态下消息无法被投递给消费者。



2. 生产者发送消息成功后自动触发 OrderTransactionListenerImpl.executeLocalTransaction()执行本地事务。

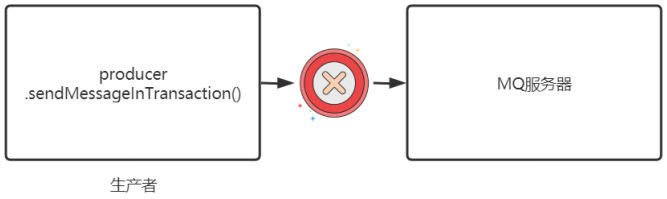
当消息发送成功，紧接着生产者向本地数据库写数据，数据库写入后提交commit，同时executeLocalTransaction方法返回COMMIT_MESSAGE，生产者会再次向MQ服务器发送一个commit提交消息，此前在Broker中保存1030订单消息状态就从“half-message”变为"已提交"，broker将消息发给下游的消费者处理。




```
public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
    /* 伪代码
    try{
        orderDao.insert(1030,order);
        //向1030号订单新增3条订单明细，10081-10083，
        orderDetailDao.insert(10081,1030,orderDetail1);
        orderDetailDao.insert(10082,1030,orderDetail2);
        orderDetailDao.insert(10083,1030,orderDetail3);
        connection.commit();
        //返回Commit，消费者可以消费1030订单消息
        return LocalTransactionState.COMMIT_MESSAGE;
    }catch(Exception e){...}
}
```

异常流程1： producer.sendMessageInTransaction(msg, null); 执行失败，抛出异常

此时没有任何消息被发出，本地事务也不会执行，除了报错外不会产生任何不一致。

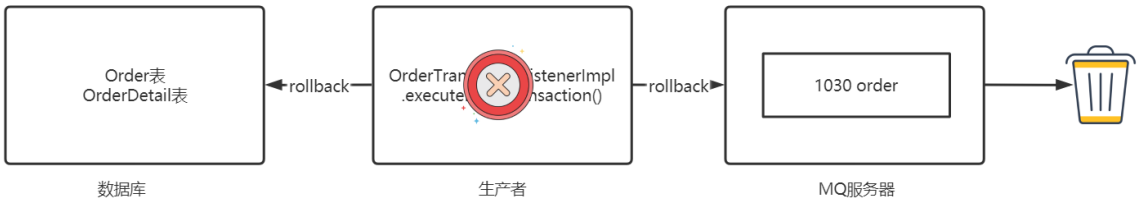


异常流程2： producer.sendMessageInTransaction(msg, null); 执行成功，本地事务执行失败

OrderTransactionListenerImpl：

```
public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
    /* 伪代码
    try{
        //插入1030号订单
        orderDao.insert(1030,order);
        //插入失败
        orderDetailDao.insert(10081,1030,orderDetail1);
        ...
    }catch(Exception e){
        //返回Rollback，Broker直接将数据删除，消费者不能收到1030订单消息
        connection.rollback();
        return LocalTransactionState.ROLLBACK_MESSAGE;
    }
}
```

此时本地事务执行rollback回滚，数据库数据被撤销，同时executeLocalTransaction方法返回ROLLBACK_MESSAGE代表回滚，生产者会再次向MQ服务器发送一个rollback回滚消息，此前在Broker中保存1030订单消息就会被直接删除，不会发送给消费者，本地事务也可以保证与MQ消息一致。



异常流程3： producer.sendMessageInTransaction(msg, null); 执行成功，本地事务执行成功，但给Broker返回Commit消息时断网了，导致broker无法收到提交指令。

OrderTransactionListenerImpl：

```
public LocalTransactionState executeLocalTransaction(Message msg, Object arg) {
    /* 伪代码
    try{
        orderDao.insert(1030,order);
        //向1030号订单新增3条订单明细，10081-10083，
        orderDetailDao.insert(10081,1030,orderDetail1);
        orderDetailDao.insert(10082,1030,orderDetail2);
        orderDetailDao.insert(10083,1030,orderDetail3);
        connection.commit();
        //返回Commit时网络中断
        return LocalTransactionState.COMMIT_MESSAGE;
```

```
    }catch(Exception e){...}
}
```

此时本地数据库订单数据已入库，但MQ因为断网无法收到生产者的发来的“commit”消息，1030订单数据一直处于“half message”的状态，消息无法被投递到消费者，本地事务与MQ消息的一致性被破坏。

RocketMQ为了解决这个问题，设计了回查机制，对于broker中的half message，每过一小段时间就自动尝试与生产者通信，试图调用通OrderTransactionListenerImpl.checkLocalTransaction()方法确认之前的本地事务是否成功。

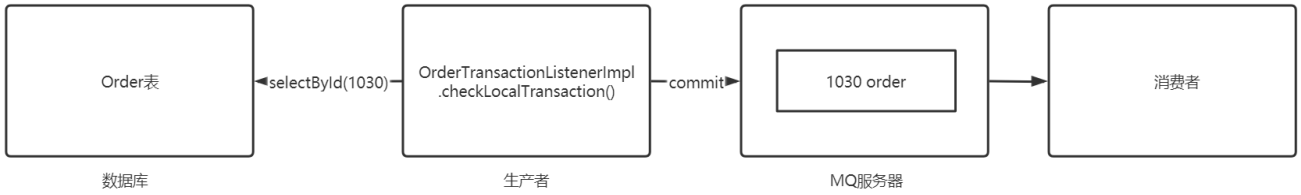
```
//会查本地事务处理状态
public LocalTransactionState checkLocalTransaction(MessageExt msg) {

    String keys = msg.getKeys();
    log.info("触发回查，正在检查" + keys + "订单状态");
    /* 伪代码

    Order order = orderDao.selectById(1030);
    if(order != null){
        //查询到记录，代表数据库已处理成功，回查返回Commit，消费者可以消费1030订单消息
        return LocalTransactionState.COMMIT_MESSAGE;
    }else{
        //未查询到记录，代表数据库处理失败，回查返回Rollback，Broker直接将数据删除，消费者不能收到1030订单消息
        return LocalTransactionState.ROLLBACK_MESSAGE;
    }
    */
    log.info("回查结果，" + keys + "订单已入库，发送Commit指令");
    return LocalTransactionState.COMMIT_MESSAGE;
}
```

由MQ服务器主动发起，生产者调用OrderTransactionListenerImpl.checkLocalTransaction()检查之前数据库事务是否完成。

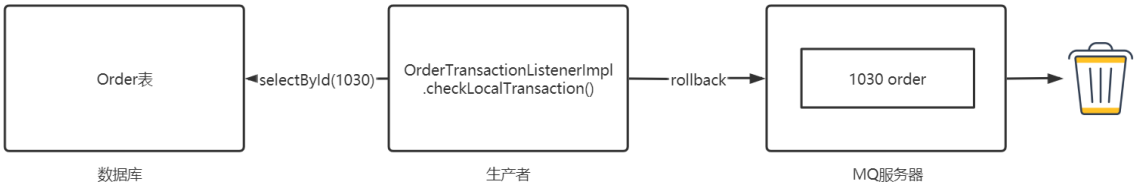
checkLocalTransaction()查询到订单数据，说明之前的数据库事务已经完成，返回COMMIT_MESSAGE，这样Broker中的1030订单消息就可以被发送给消费者进行处理。



运行结果：

22:31:35.670 [main] INFO com.itlaoqi.rocketmq.mtype.OrderTransactionListenerImpl - 正在执行本地事务,订单编号:1030
22:31:35.672 [main] INFO com.itlaoqi.rocketmq.mtype.OrderTransactionListenerImpl - 模拟网络中断，Broker并未收到生产者本地事务状态回执，返回UNKNOW
22:31:45.995 [check-transaction-thread] INFO com.itlaoqi.rocketmq.mtype.OrderTransactionListenerImpl - 触发回查，正在检查1030订单状态
22:31:45.996 [check-transaction-thread] INFO com.itlaoqi.rocketmq.mtype.OrderTransactionListenerImpl - 回查结果，1030订单已入库，发送Commit指令

checkLocalTransaction()未查询到订单数据，说明之前的数据库事务没有处理成功，返回ROLLBACK_MESSAGE，这样Broker中的1030订单消息就会被删除。



RocketMQ事务消息执行执行流程