分布式事务 第三天

1.1学习目标

目标 RocketMQ事务消息(了解)

第三章 RocketMQ事务消息

RocketMq的简介:

目标:

- rocketmq特点
- 事务消息的概念

出身:阿里巴巴,消息中间件.性能

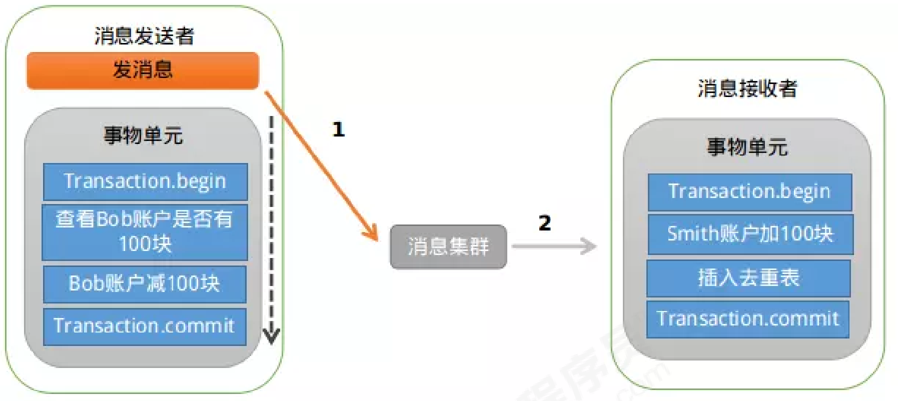
特点:

- 支持事务消息
- 顺序消息
- 定时消息
- 批量消息
- 消息回溯.

事务消息的概念:执行本地事务（Bob账户扣款）和发送异步消息应该保证同时成功或者同时失败

- 首先看下先发送消息的情况，大致的示意图如下：

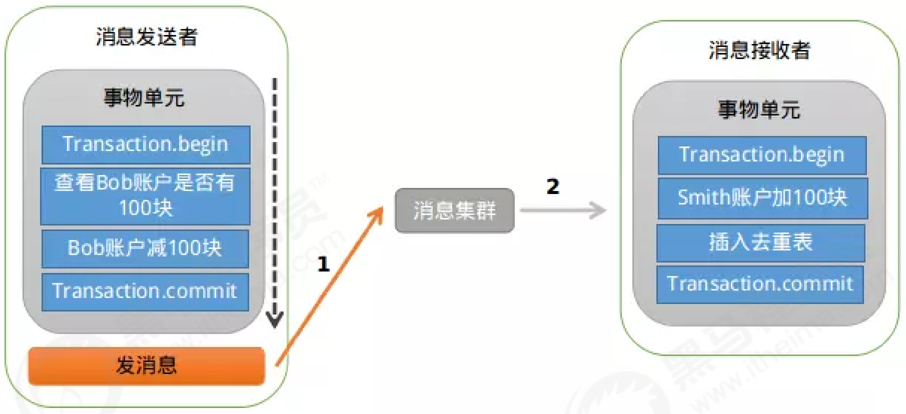
  存在的问题是：如果消息发送成功，但是扣款失败，消费端就会消费此消息，进而向Smith账户加钱。



事务消息：先发送消息

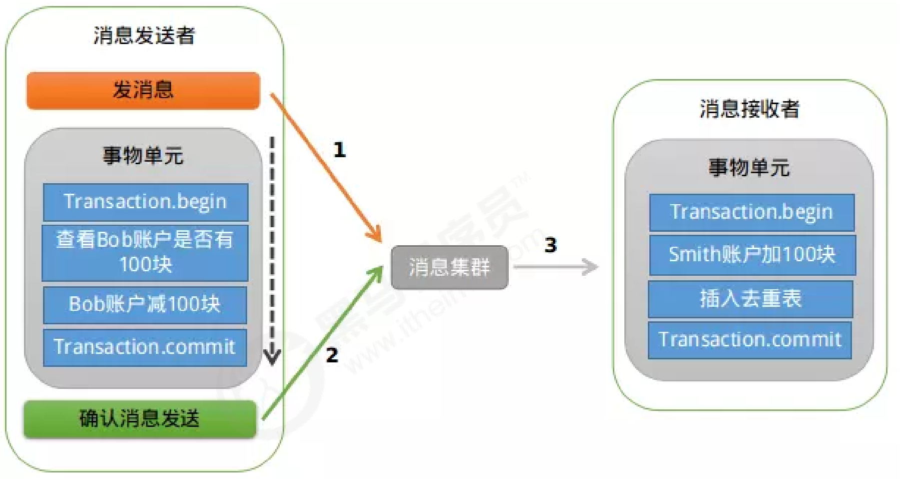
- 先发消息不行，那就先扣款吧，大致的示意图如下：

  如果扣款成功，发送消息失败，就会出现Bob扣钱了，但是Smith账户未加钱。



事务消息-先扣款

解决办法：

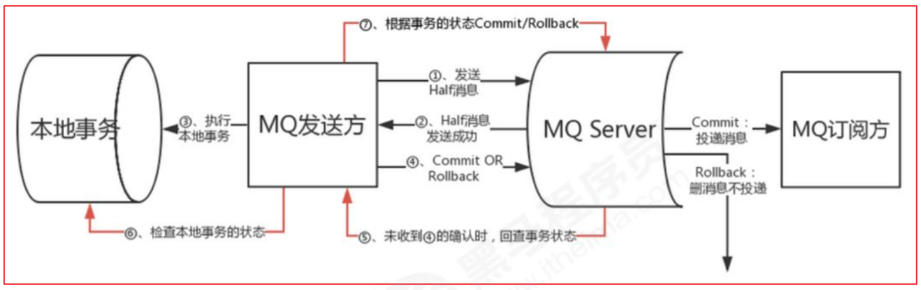


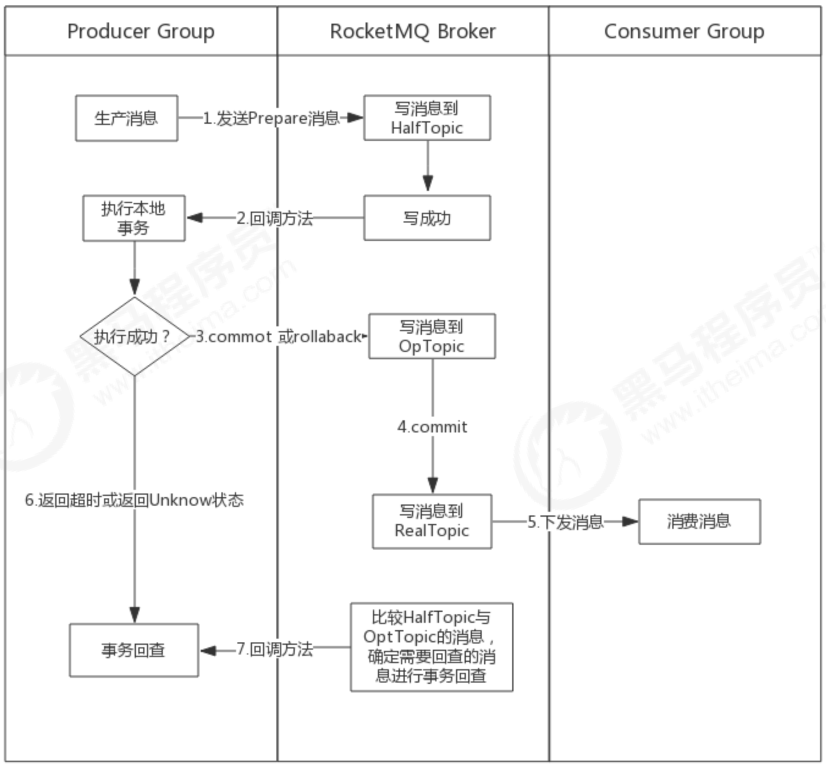
RocketMQ实现发送事务消息

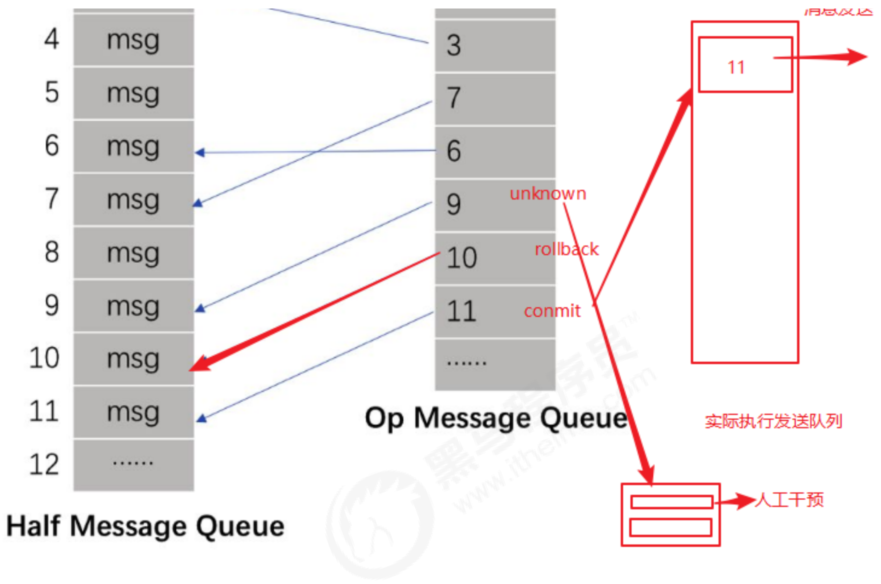
RocketMQ第一阶段发送`Prepared消息`时，会拿到消息的地址，第二阶段执行本地事物，第三阶段通过第一阶段拿到的地址去访问消息，并修改消息的状态。

3.1 RocketMQ事务消息流程

RocketMQ的事务消息，主要是通过消息的异步处理，可以保证本地事务和消息发送同时成功执行或失败，从而保证数据的最终一致性，这里我们先看看一条事务消息从诞生到结束的整个时间线流程：



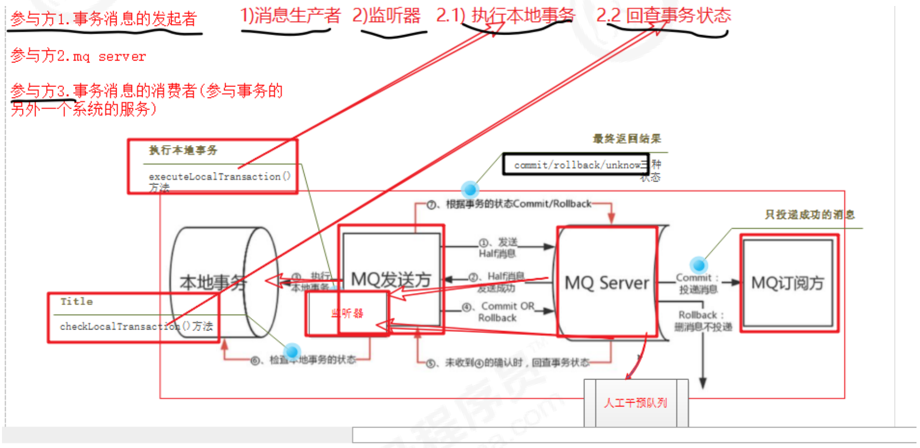




```
事务消息的成功投递是需要经历三个Topic的，分别是：
    Half Topic：用于记录所有的prepare消息
    Op Half Topic：记录已经提交了状态的prepare消息
    Real Topic：事务消息真正的Topic,在Commit后会才会将消息写入该
Topic，从而进行消息的投递
```

3.2 事务消息编写核心步骤

目标:1.知道事务消息编程涉及到的几个角色:



```
1.消息发送者
2.监听器(给发送者用的)
    执行本地事务方法
    回查本地事务状态
3.消息的订阅者
```

2.掌握事务消息编程的核心步骤.(重点)

核心步骤:

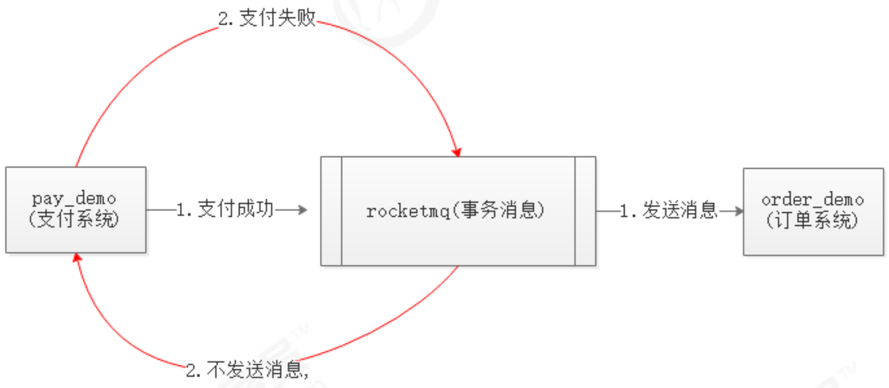
1.消息生产者(即事务发起者)代码编写

- 执行本地事务函数executeLocalTransaction()代码编写
- 回查本地事务状态函数checkLocalTransaction()代码编写

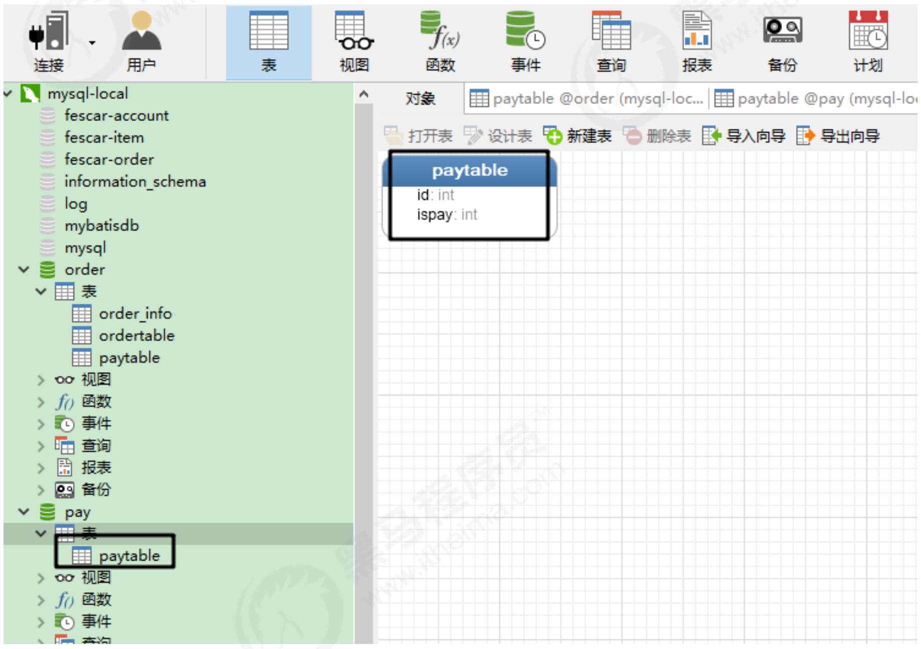
3.消息消费者代码编写

3.3 案例说明:

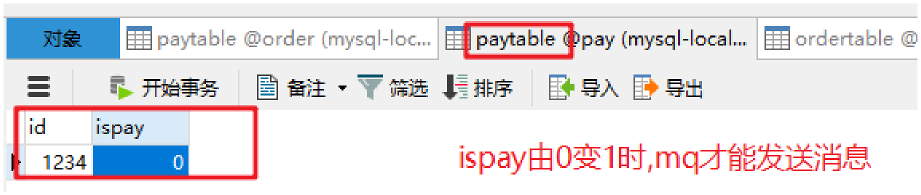
只有支付状态变更为已付款时(paytable表ispay由0变为1时),mq才发送消息,消息的订阅方才可以接受到消息



数据库和表:



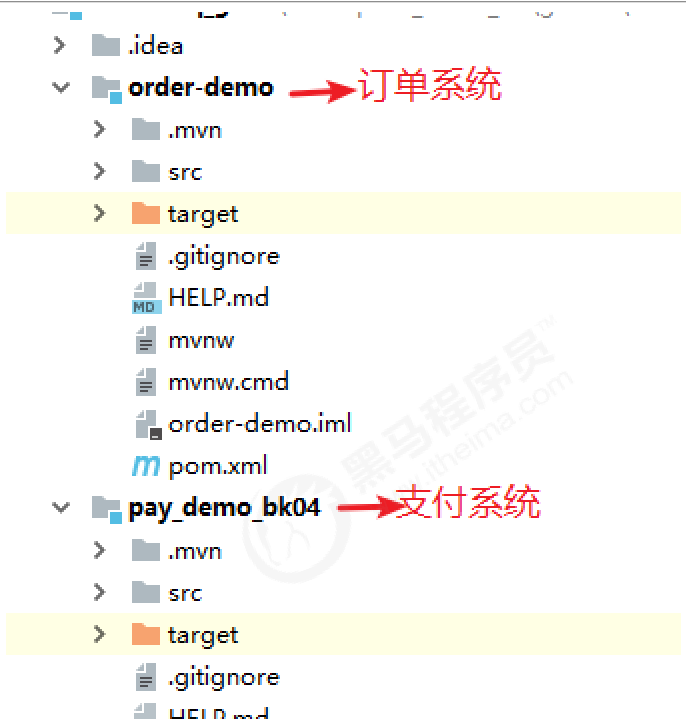
paytable
id: int
ispay: int



id	ispay
1234	0

ispay由0变1时,mq才能发送消息

项目结构:支付系统和订单系统



3.4 消息生产者:PayController中创建消息发送对象,代码如下:

jar包依赖的引入

生产者(事务消息生产者)

监听器

事务id

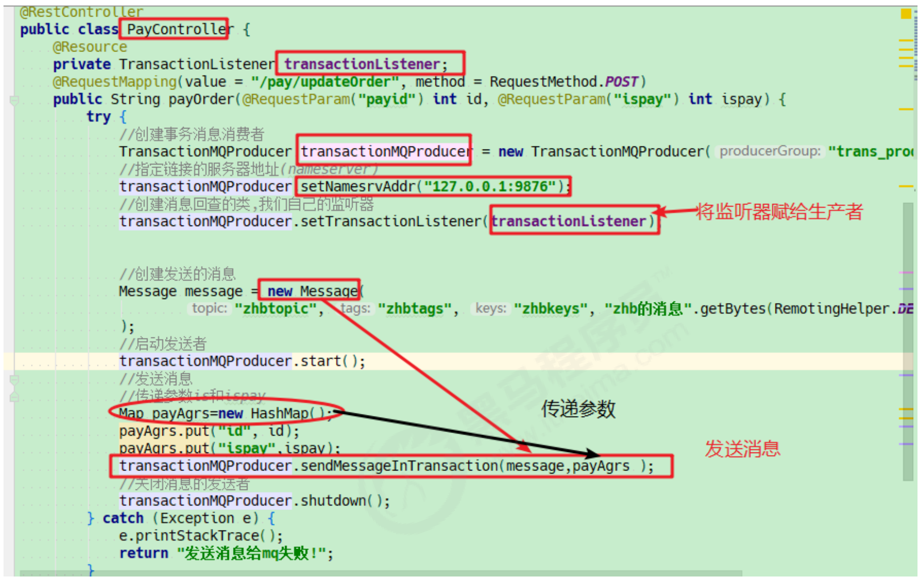
消息体

消息生产者,将监听器装配到消息生产者中.

生成消息体,通过发送方法,将消息体发送出去.

1.引入rocketmq的jar包

```
    <!--整合rokectmq-->
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-spring-boot-
starter</artifactId>
            <version>2.0.3</version>
        </dependency>
```



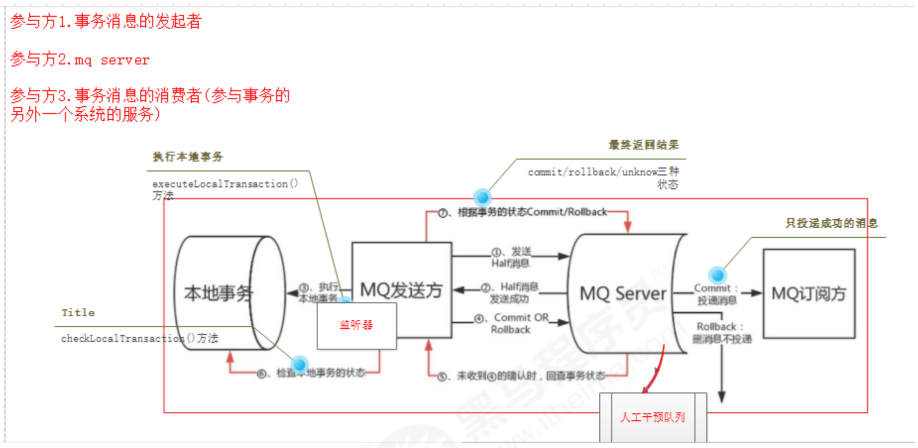
```
@RestController
public class PayController {
    @Resource
    private TransactionListener transactionListener;
    @RequestMapping(value = "/pay/updateOrder", method = RequestMethod.POST)
    public String payOrder(@RequestParam("payid") int id, @RequestParam("ispay") int ispay) {
        try {
            //创建事务消息消费者
            TransactionMQProducer transactionMQProducer = new TransactionMQProducer( producerGroup: "trans_proc
            //指定链接的服务器地址(nameserver)
            transactionMQProducer.setNamesrvAddr("127.0.0.1:9876");
            //创建消息回查的类,我们自己的监听器
            transactionMQProducer.setTransactionListener(transactionListener);


            //创建发送的消息
            Message message = new Message(
                    topic: "zhbtopic",  tags: "zhbtags",  keys: "zhbkeys", "zhb的消息".getBytes(RemotingHelper.DE
            );
            //启动发送者
            transactionMQProducer.start();
            //发送消息
            //传递参数id和ispay
            Map payAgrs=new HashMap();
            payAgrs.put("id", id);
            payAgrs.put("ispay",ispay);
            transactionMQProducer.sendMessageInTransaction(message,payAgrs );
            //关闭消息的发送者
            transactionMQProducer.shutdown();
        } catch (Exception e) {
            e.printStackTrace();
            return "发送消息给mq失败!";
        }
```

上图代码如下：

```
package com.itheimabk04.controller;

import com.itheimabk04.mq.MyTransactionListener;
import com.itheimabk04.service.PayService;
import
org.apache.rocketmq.client.exception.MQClientException;
import
org.apache.rocketmq.client.producer.LocalTransactionState;
import
org.apache.rocketmq.client.producer.TransactionListener;
import
org.apache.rocketmq.client.producer.TransactionMQProducer;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import
org.springframework.web.bind.annotation.RequestParam;
import
org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.*;

@RestController
public class PayController {
    @Resource
    @RequestMapping(value = "/pay/updateOrder", method =
RequestMethod.POST)
    public String payOrder(@RequestParam("payid") int id,
@RequestParam("ispay") int ispay) {
        try {
            //创建事务消息消费者
            TransactionMQProducer transactionMQProducer =
new TransactionMQProducer("trans_producer_group_zhb");
            //指定链接的服务器地址(nameserver)

 transactionMQProducer.setNamesrvAddr("127.0.0.1:9876");
            //创建消息回查的类,我们自己的监听器

 transactionMQProducer.setTransactionListener(transactionL
istener);



            //创建发送的消息
            Message message = new Message(
                    "zhbtopic", "zhbtags", "zhbkeys", "zhb
的消息".getBytes(RemotingHelper.DEFAULT_CHARSET)
            );
            //启动发送者
            transactionMQProducer.start();
            //发送消息
            //传递参数is和ispay
            Map payAgrs=new HashMap();
            payAgrs.put("id", id);
            payAgrs.put("ispay",ispay);

 transactionMQProducer.sendMessageInTransaction(message,pa
yAgrs );
            //关闭消息的发送者
            transactionMQProducer.shutdown();
        } catch (Exception e) {
            e.printStackTrace();
            return "发送消息给mq失败!";
        }
        //如果没有问题,
        return "发送消息给mq成功";
    }

}
```

3.5监听器编写

我们创建一个事务消息生产者TransactionProducer,事务消息发送消息对象是TransactionMQProducer，为了实现本地事务操作和回查，我们需要创建一个监听器，监听器需要实现TransactionListener接口，实现代码如下：



2.代码如下

```
package com.itheimabk04.mq;

import com.itheimabk04.service.PayService;
import
org.apache.rocketmq.client.producer.LocalTransactionState;
import
org.apache.rocketmq.client.producer.TransactionListener;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.common.message.MessageExt;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

@Component
public class MyTransactionListener implements
TransactionListener {

    //记录对应事务消息的执行状态   1:正在执行，2: 执行成功，3: 失败
了
    //对于mq来说,正在事务发起方正在执行查询结果,只要未收到明确的
commit或者rollback,都是未知结果unknow
    //对于mq来说,commit执行成功,才发送消息
    //对于mq来说,事务执行失败了将不再发送消息,并且将消息队列中的
half消息干掉,以免再次扫描到再次回查
    //通过事务的id来辨别不同的事务

    private ConcurrentHashMap<String,Integer> transMap =
new ConcurrentHashMap<String,Integer>();
    //注入payService
    @Resource
    private PayService payService;
    /**
     * 消息发送方执行自身业务操作的方法
     * @param msg 发送方发送的东西
     * @return
     */
    public LocalTransactionState
executeLocalTransaction(Message msg, Object arg) throws
RuntimeException {
            //业务代码写这里

        String transactionId = msg.getTransactionId();
        //设置执行状态为正在执行,state=1
        transMap.put(transactionId, 1);
        //取id和ispay参数
        Map payArgs= (Map) arg;
        Integer id= (Integer) payArgs.get("id");
        Integer ispay= (Integer) payArgs.get("ispay");

        try {
            //控制本地事务
            System.out.println("支付表更新开始");
            payService.updatePayTable(id, ispay);
            System.out.println("支付表更新成功");
            //测试用例1
//              int i=1/0;
           // 测试用例2 测试网络超时状态
//              Thread.sleep(70000);
            System.out.println("更新订单状态");
            System.out.println("订单已更新");
            //执行成功时,返回提交事务消息成功的标识
            transMap.put(transactionId, 2);
//              if(1==1){
//                  return  LocalTransactionState.UNKNOW;
//              }


        }catch (Exception e){
            //发生异常时, 返回回滚事务消息
            //执行成功时,返回提交事务消息成功的标识
            transMap.put(transactionId, 3);
            System.out.println("事务执行失败,事务执行状态为:"+
LocalTransactionState.ROLLBACK_MESSAGE);
            return
 LocalTransactionState.ROLLBACK_MESSAGE;
        }
        System.out.println("事务执行成功,事务执行状态为:"+
LocalTransactionState.COMMIT_MESSAGE);
        return LocalTransactionState.COMMIT_MESSAGE;

    }


    /***
     * 事务超时, 回查方法
     * @return
     */
    @Override
    public LocalTransactionState
checkLocalTransaction(MessageExt msg) {
        //根据transaction的id回查该事务的状态,并返回给消息队列
        //未知状态:查询事务状态,但始终无结果,或者由于网络原因发送不
成功,对mq来说都是未知状态,LocalTransactionState.UNKNOW
        //正确提交返回LocalTransactionState.COMMIT_MESSAGE
        //事务执行失败返回
LocalTransactionState.ROLLBACK_MESSAGE
        String transactionId = msg.getTransactionId();
        Integer state = transMap.get(transactionId);
        System.out.println("回查的事务id
为:"+transactionId+",当前的状态为"+state);

        if (state==2){
            //执行成功,返回commit
            System.out.println("回查结果为事务正确提交,返回状态
为:"+ LocalTransactionState.COMMIT_MESSAGE);
            return  LocalTransactionState.COMMIT_MESSAGE;

        }else if(state==3){
            //执行失败,返回rollback
            System.out.println("回查结果为事务回滚,返回状态
为:"+ LocalTransactionState.ROLLBACK_MESSAGE);
            return
 LocalTransactionState.ROLLBACK_MESSAGE;
        }

            //正在执行
            System.out.println("回查正在执行,返回状态为:"+
LocalTransactionState.UNKNOW);
            return  LocalTransactionState.UNKNOW;

    }
}
```

3.6 事务消息消费

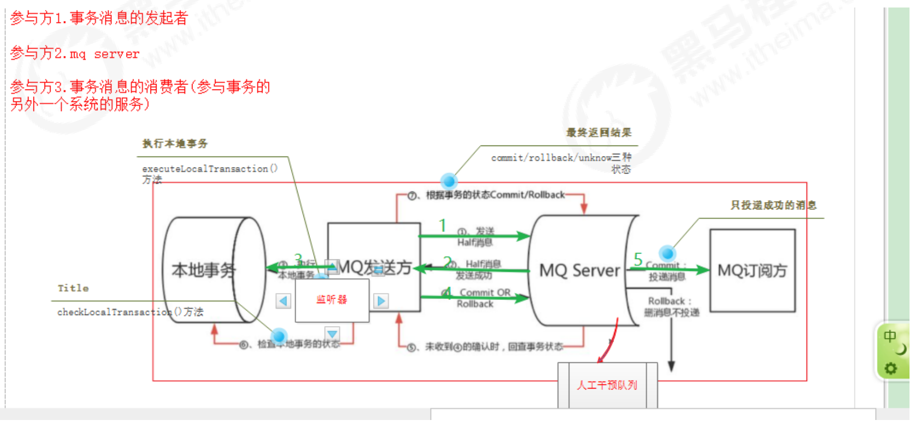
事务消息的消费者和普通消费者一样，这里我们就不做介绍了，直接贴代码：

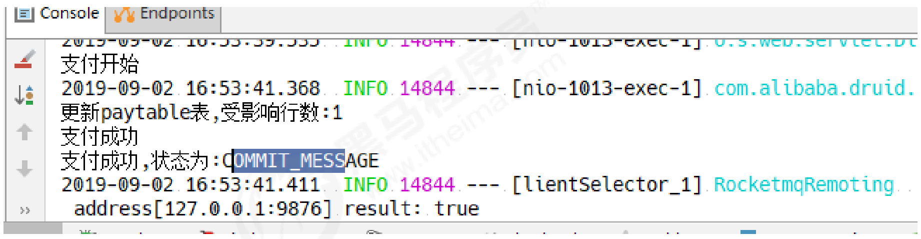
```
package com.itheima.mq;

import
org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
org.apache.rocketmq.client.consumer.listener.ConsumeConcur
rentlyContext;
import
org.apache.rocketmq.client.consumer.listener.ConsumeConcur
rentlyStatus;
import
org.apache.rocketmq.client.consumer.listener.MessageListen
erConcurrently;
import
org.apache.rocketmq.common.consumer.ConsumeFromWhere;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.remoting.common.RemotingHelper;

import java.io.UnsupportedEncodingException;
import java.util.List;

public class TransactionConsumer {
    public static void main(String[] args) throws
Exception{
        //创建消息的消费者
        DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("zhb_trans-client-group");
        //设置要链接的服务器地址(nameserver)
        consumer.setNamesrvAddr("127.0.0.1:9876");
        //设置单次消费的消息的数量
        consumer.setConsumeMessageBatchMaxSize(5);
        //设置消息消费的顺序

 consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FRO
M_FIRST_OFFSET);
        //设置消费者监听哪些消息
        consumer.subscribe("zhbtopic", "zhbtags");
        //进行消息的接收，并返回接收消息的结果
        consumer.registerMessageListener(new
MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus
consumeMessage(List<MessageExt> list,
ConsumeConcurrentlyContext consumeConcurrentlyContext) {

                try {
                    for(MessageExt mes :list){
                        String topic = mes.getTopic();
                        String tags = mes.getTags();
                        String keys = mes.getKeys();
                        String s = new
String(mes.getBody(), "utf-8");
                        String transactionId =
mes.getTransactionId();
transactionid:"+transactionId+",
topic:"+topic+",tags:"+tags+",消息:"+s);
                    }
                }catch( Exception e){
                    e.printStackTrace();
                }

                return
ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
        //启动消费者
        System.out.println("启动完成");
        consumer.start();
    }
}
```

3.7测试

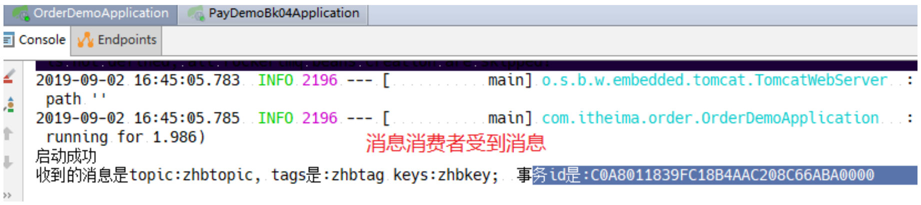
测试用例:

正常:



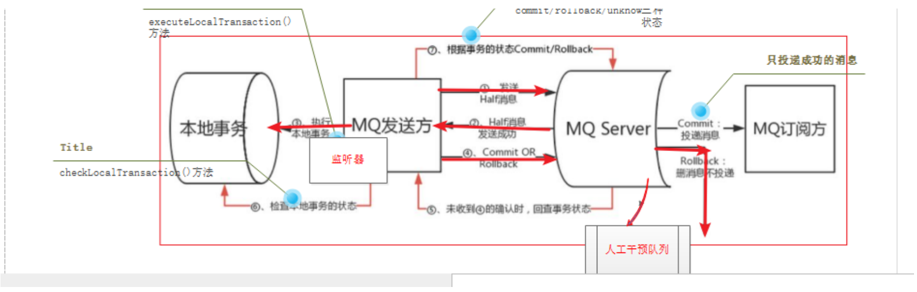


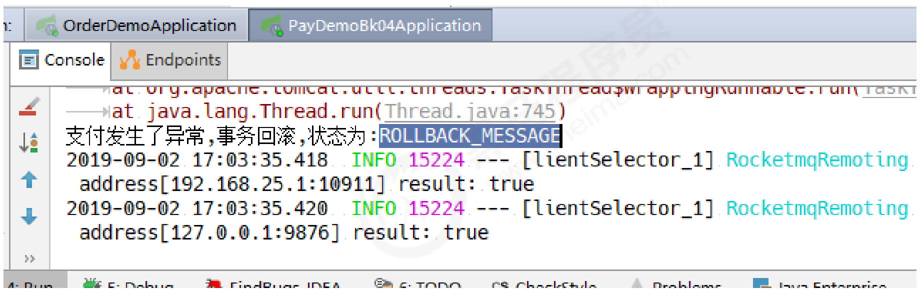
```
Console  Endpoints
2019-09-02 16:53:39.535  INFO 14844 --- [nio-1013-exec-1] o.s.web.servlet.Dt
支付开始
2019-09-02 16:53:41.368  INFO 14844 --- [nio-1013-exec-1] com.alibaba.druid.
更新paytable表,受影响行数:1
支付成功
支付成功,状态为:COMMIT_MESSAGE
2019-09-02 16:53:41.411  INFO 14844 --- [lientSelector_1] RocketmqRemoting
 address[127.0.0.1:9876] result: true
```

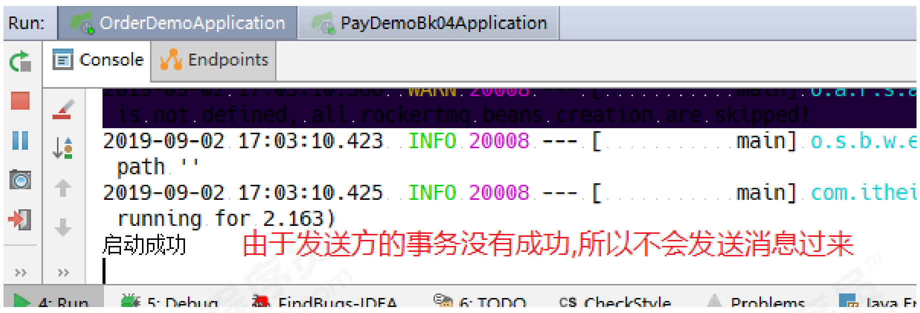


```
OrderDemoApplication  PayDemoBk04Application
Console  Endpoints
2019-09-02 16:45:05.783  INFO 2196 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  :
 path ''
2019-09-02 16:45:05.785  INFO 2196 --- [           main] com.itheima.order.OrderDemoApplication   :
 running for 1.986)
启动成功
收到的消息是topic:zhbtopic, tags是:zhbtag keys:zhbkey;  事务id是:C0A8011839FC18B4AAC208C66ABA0000
```

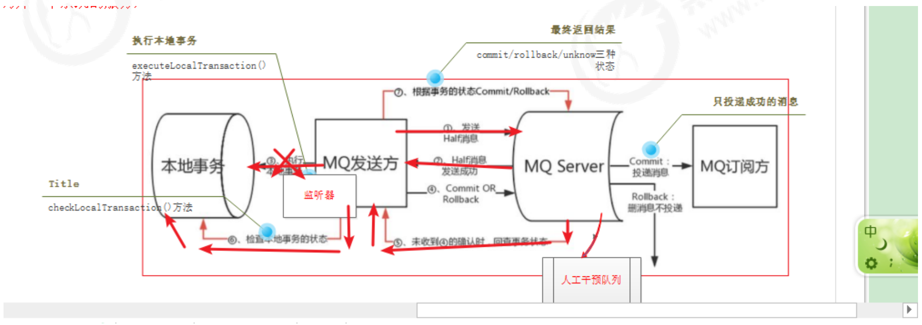
事务失败



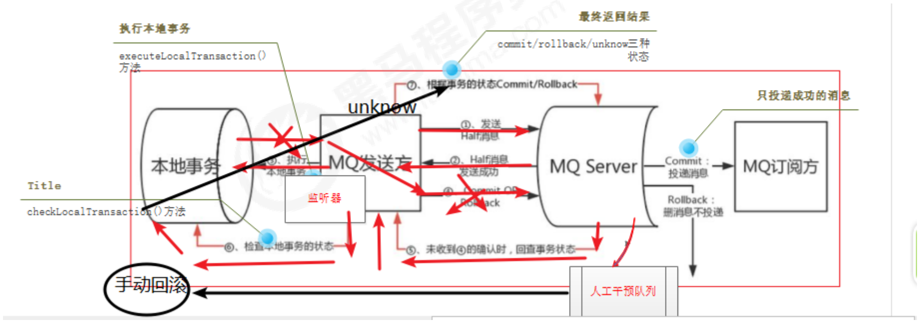

```
    at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskT
    at java.lang.Thread.run(Thread.java:745)
支付发生了异常,事务回滚,状态为:ROLLBACK_MESSAGE
2019-09-02 17:03:35.418  INFO 15224 --- [lientSelector_1] RocketmqRemoting
 address[192.168.25.1:10911] result: true
2019-09-02 17:03:35.420  INFO 15224 --- [lientSelector_1] RocketmqRemoting
 address[127.0.0.1:9876] result: true
```


```
 is not defined, all rockertmq beans creation are skipped!
2019-09-02 17:03:10.423  INFO 20008 --- [           main] o.s.b.w.e
 path ''
2019-09-02 17:03:10.425  INFO 20008 --- [           main] com.ithei
 running for 2.163)
启动成功      由于发送方的事务没有成功,所以不会发送消息过来
```

网络原因,超时

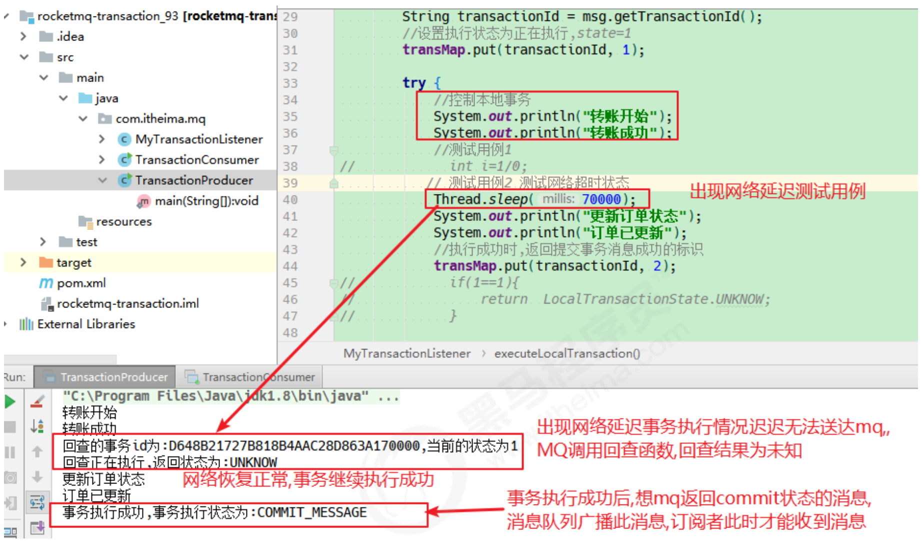


如果事务的执行结果始终不明确(由于网络的原因)



测试结果:

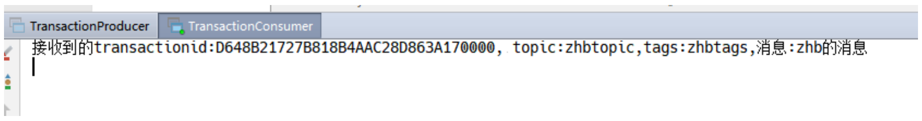
- 执行超时后,mq调用回查方法.返回为未知状态unk
- mq继续调用回查方法,此时网络问题解决,事务执行成功,mq收到事务commit消息


rocketmq-transaction_93 [rocketmq-trans
.idea
src
main
java
com.itheima.mq
MyTransactionListener
TransactionConsumer
TransactionProducer
main(String[]):void
resources
test
target
pom.xml
rocketmq-transaction.iml
External Libraries
MyTransactionListener › executeLocalTransaction()
Run: TransactionProducer TransactionConsumer
出现网络延迟测试用例
出现网络延迟事务执行情况迟迟无法送达mq,,
MQ调用回查函数,回查结果为未知
网络恢复正常,事务继续执行成功
事务执行成功后,想mq返回commit状态的消息,
消息队列广播此消息,订阅者此时才能收到消息

```java
        String transactionId = msg.getTransactionId();
        //设置执行状态为正在执行,state=1
        transMap.put(transactionId, 1);

        try {
            //控制本地事务
            System.out.println("转账开始");
            System.out.println("转账成功");
            //测试用例1
//              int i=1/0;
            // 测试用例2 测试网络超时状态
            Thread.sleep( millis: 70000);
            System.out.println("更新订单状态");
            System.out.println("订单已更新");
            //执行成功时,返回提交事务消息成功的标识
            transMap.put(transactionId, 2);
//              if(1==1){
//                  return  LocalTransactionState.UNKNOW;
//              }

```

```
"C:\Program Files\Java\jdk1.8\bin\java" ...
转账开始
转账成功
回查的事务id为:D648B21727B818B4AAC28D863A170000,当前的状态为1
回查正在执行,返回状态为:UNKNOW
更新订单状态
订单已更新
事务执行成功,事务执行状态为:COMMIT_MESSAGE
```

消费者读取消息:



```
接收到的transactionid:D648B21727B818B4AAC28D863A170000, topic:zhbtopic,tags:zhbtags,消息:zhb的消息
```

事务消息参考地址：http://rocketmq.apache.org/docs/transaction-example/