图文 71 用支付后发红包的案例场景，分析RocketMQ事物消息的代码

详情 评论

用支付后发红包的案例场景，分析RocketMQ事物消息的代码实现细节

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、对RocketMQ的事务消息分析代码落地实现

今天我们来分析一下RocketMQ的事务消息的代码落地实现

既然已经对事务消息的流程、原理以及使用场景都进行了分析，那下一步就是真正看看如何基于RocketMQ提供的Java API进行编码实现了。

我们将会直接基于官方文档提供的事务消息API使用的例子来给大家进行分析，同时我们会把订单系统的业务场景放在里面，加入一些伪代码让大家来参考一下。

对代码的分析我们全部基于注释写在代码里了，大家通过看注释就完全可以理解代码的使用。

2、发送half事务消息出去

```
public class TransactionProducer {

    public static void main(String[] args)
            throws MQClientException, InterruptedException {
        // 这个东西就是用来接受RocketMQ回调的一个监听器接口
        // 这里会实现执行订单本地事务，commit、rollback，回调查询等逻辑
TransactionListener transactionListener = new TransactionListenerImpl();

        // 下面这个就是创建一个支持事务消息的Producer
        // 对这个Producer还得指定要指定一个生产者分组，随便指定一个名字
        TransactionMQProducer producer =
                new TransactionMQProducer("TestProducerGroup");

        // 下面这个是指定了一个线程池，里面会包含一些线程
        // 这个线程池里的线程就是用来处理RocketMQ回调你的请求的
        // 如果大家对线程池不太了解
        // 可以看石杉老师的《互联网Java工程师面试突击第三季》的线程池部分
        ExecutorService executorService = new ThreadPoolExecutor(
                2,
                5,
                100,
                TimeUnit.SECONDS,
                new ArrayBlockingQueue<Runnable>(2000),
                new ThreadFactory() {
                 @Override
                 public Thread newThread(Runnable r) {
                      Thread thread = new Thread(r);
                       thread.setName("TestThread");
                      return thread;
                  }
                 }
         );

         // 给事务消息生产者设置对应的线程池，负责执行RocketMQ回调请求
         producer.setExecutorService(executorService);
         // 给事务消息生产者设置对应的回调函数
         producer.setTransactionListener(transactionListener);
         // 启动这个事务消息生产者
         producer.start();

         // 构造一条订单支付成功的消息，指定Topic是谁
         Message msg = new Message(
                "PayOrderSuccessTopic",
                "TestTag",
                "TestKey",
               ("订单支付消息").getBytes(RemotingHelper.DEFAULT_CHARSET)
         );

         // 将消息作为half消息的模式发送出去
SendResult sendResult = producer.sendMessageInTransaction(msg, null);
    }
}
```

3、假如half消息发送失败，或者没收到half消息响应怎么办？

我们已经看到如何发送half消息了，但是假如发送half消息失败了怎么办呢？

此时我们其实会在执行“producer.sendMessageInTransaction(msg, null)”的时候，收到一个异常，发现消息发送失败了。

所以我们可以用下面的代码去关注half消息发送失败的问题：

```
try {
     SendResult sendResult = producer.sendMessageInTransaction(msg, null);
} catch(Exception e) {
     // half消息发送失败
// 订单系统执行回滚逻辑，比如说触发支付退款，更新订单状态为“已关闭”
}
```

那如果一直没有收到half消息发送成功的通知呢？

针对这个问题，我们可以把发送出去的half消息放在内存里，或者写入本地磁盘文件，后台开启一个线程去检查，如果一个half消息超过比如10分钟都没有收到响应，那就自动触发回滚逻辑。

4、如果half消息成功了，如何执行订单本地事务？

刚才代码里有一个TransactionListener，这个类也是我们自己定义的，如下所示：

```
public class TransactionListenerImpl implements TransactionListener {

     // 如果half消息发送成功了
     // 就会在这里回调你的这个函数，你就可以执行本地事务了
     @Override
     public LocalTransactionState executeLocalTransaction(
             Message msg, Object arg) {
         // 执行订单本地事务
         // 接着根据本地一连串事务执行结果，去选择执行commit or rollback
         try {
              // 如果本地事务都执行成功了，返回commit
             return LocalTransactionState.COMMIT_MESSAGE;
         } catch(Exception e) {
              // 本地事务执行失败，回滚所有一切执行过的操作
              // 如果本地事务执行失败了，返回rollback，标记half消息无效
             return LocalTransactionState.ROLLBACK_MESSAGE;
         }
}

}
```

5、如果没有返回commit或者rollback，如何进行回调?

```
public class TransactionListenerImpl implements TransactionListener {

    // 如果half消息发送成功了
    // 就会在这里回调你的这个函数，你就可以执行本地事务了
    @Override
    public LocalTransactionState executeLocalTransaction(
            Message msg, Object arg) {
        // 执行订单本地事务
        // 接着根据本地一连串事务执行结果，去选择执行commit or rollback
        try {
             // 如果本地事务都执行成功了，返回commit
            return LocalTransactionState.COMMIT_MESSAGE;
        } catch(Exception e) {
             // 本地事务执行失败，回滚所有一切执行过的操作
             // 如果本地事务执行失败了，返回rollback，标记half消息无效
            return LocalTransactionState.ROLLBACK_MESSAGE;
        }
    }

    // 如果因为各种原因，没有返回commit或者rollback
    @Override
    public LocalTransactionState checkLocalTransaction(MessageExt msg) {
        // 查询本地事务，是否执行成功了
        Integer status = localTrans.get(msg.getTransactionId());
        // 根据本地事务的情况去选择执行commit or rollback
        if (null != status) {
             switch (status) {
                 case 0: return LocalTransactionState.UNKNOW;
                 case 1: return LocalTransactionState.COMMIT_MESSAGE;
                 case 2: return LocalTransactionState.ROLLBACK_MESSAGE;
             }
        }
        return LocalTransactionState.COMMIT_MESSAGE;
     }

}
```

6、给大家留下一个小作业

大家在看完今天对RocketMQ的事务消息代码的解释之后，建议大家可以去基于之前自己部署好的RocketMQ，去实验一下这个事务消息的代码

然后请大家自己反复思考一下，在这段代码运行的过程中，各个地方如果出现网络异常，或者是系统突然崩溃了，这套机制是如何确保消息投递稳定运行的。

End

狸猫技术窝其他精品专栏推荐：

《从零开始带你成为JVM实战高手》

《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群？》**（购买后可见）**