图文 70 为什么解决发送消息零丢失方案，一定要使用事务消息方案?

详情 评论

为什么解决发送消息零丢失方案，一定要使用事务消息方案?

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

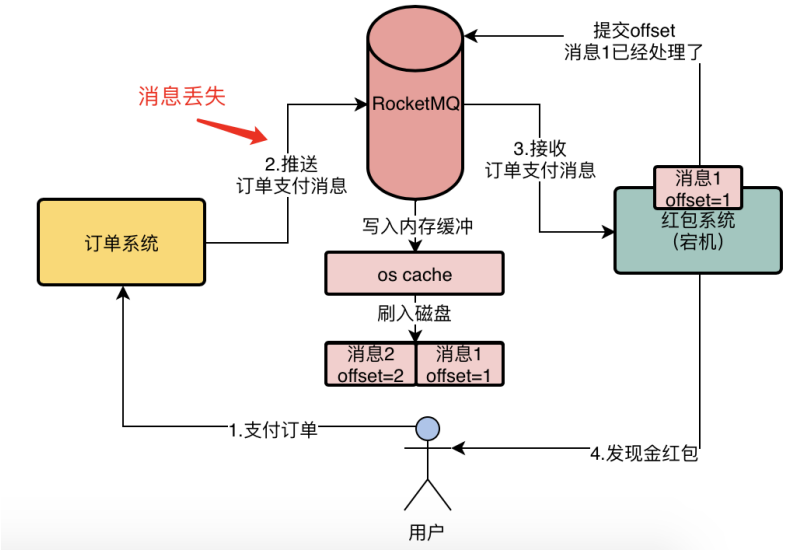
（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、停下脚步，做一点深入思考

我们已经把RocketMQ事务消息的实现流程和底层原理都进行了研究了，接着我们需要思考一个问题:

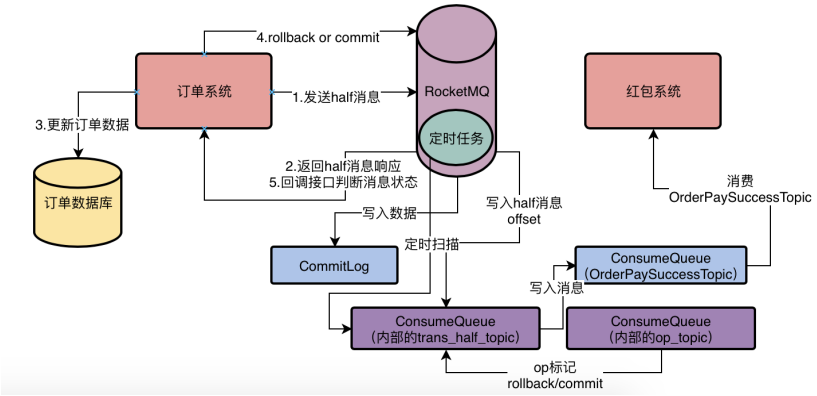
我们最早给大家说在发送消息的时候，可能存在消息的丢失，也就是说，可能消息根本就没有进入到MQ就丢了，我们看下面的图。



然后我们没解释过多的东西直接就切入了RocketMQ事务消息的讲解，其实通过RocketMQ事务消息机制的研究，我相信每个人现在都可以确信一点，如果你使用事务消息机制去发送消息到MQ，一定是可以保证消息必然发送到MQ的，不会丢！

但是我们现在回过头来想想事务消息机制的原理图，大家可以看到下面这个图

这个图的流程可真是复杂啊，先得发送half消息，完了你还得发送rollback or commit的请求，要是中间有点什么问题，MQ还得回调你的接口



我们真的有必要使用这么复杂的机制去确保消息到达MQ，而且绝对不会丢吗？

毕竟这么复杂的机制完全有可能导致整体性能比较差，而且吞吐量比较低，是否有更加简单的方法来确保消息一定可以到达MQ呢？

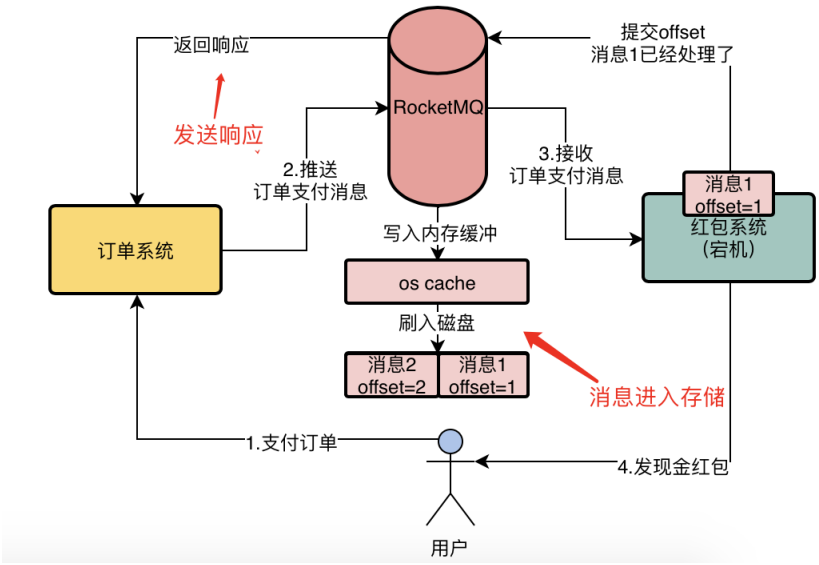
2、一个小思考：能不能基于重试机制来确保消息到达MQ？

想到这里，我觉得很多朋友可能内心已经闪现出一个想法了，就是我们之前觉得发消息到MQ，无非就是觉得可能半路上消息给搞丢了，然后消息根本没进去MQ内部，我们也没做什么额外的措施，这就导致消息找不回来了。

那么我们先搞清楚一个问题，我们发送消息到MQ，然后我们可以等待MQ返回响应给我们，在什么样的情况下，MQ会返回响应给我们呢？

答案显而易见，就是MQ收到消息之后写入本地磁盘文件了，当然这个时候可能仅仅是写入os cache而已，但是只要他写入自己本地存储了，就会返回响应给我们。

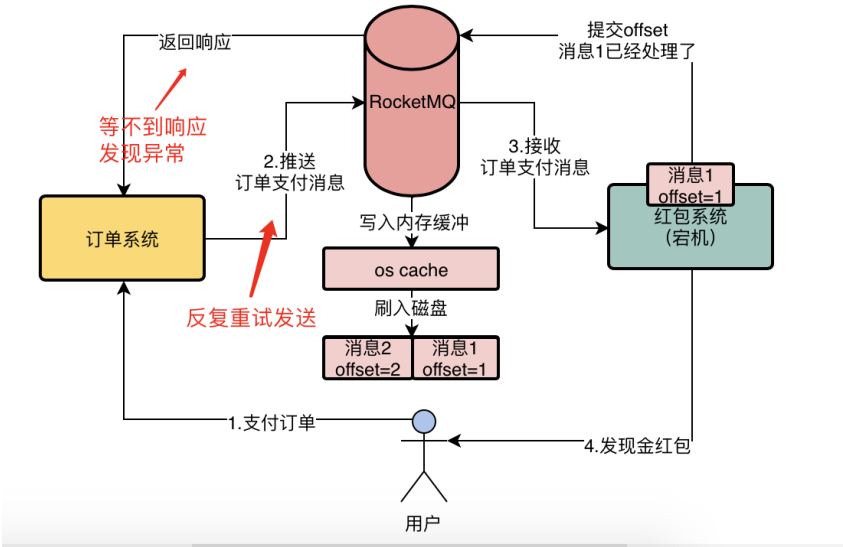
我们看下面的图里的示意



那么只要我们在代码中发送消息到MQ之后，同步等待MQ返回响应给我们，一直等待，如果半路中有网络异常或者MQ内部异常，我们肯定会收到一个异常，比如网络错误，或者请求超时之类的。

如果我们在收到异常之后，就认为消息到MQ发送失败了，然后再次重试尝试发送消息到MQ，接着再次同步等待MQ返回响应给我们，这样反复重试，是否可以确保消息一定会到达MQ?

我们看下图。



理论上似乎在一些短暂网络异常的场景下，我们是可以通过不停的重试去保证消息到达MQ的，因为如果短时间网络异常了消息一直没法发送，我们只要不停的重试，网络一旦恢复了，消息就可以发送到MQ了。

如果要是反复重试多次发现一直没法把消息投递到MQ，此时我们就可以直接让订单系统回滚之前的流程，比如发起退款流程，判定本次订单支付交易失败了。

看起来这个简单的同步发送消息 + 反复重试的方案，也可以做到保证消息一定可以投递到MQ中，大家想想是不是?

确实如此，而且在基于Kafka作为消息中间件的消息零丢失方案中，对于发送消息这块，因为Kafka本身不具备RocketMQ这种事务消息的高级功能，所以一般我们都是对Kafka会采用同步发消息 + 反复重试多次的方案，去保证消息成功投递到Kafka的。

但是如果是在类似我们目前这个较为复杂的订单业务场景中，仅仅采用同步发消息 + 反复重试多次的方案去确保消息绝对投递到MQ中，似乎还是不够的，接下来我们分析一下在复杂业务场景下，这里有什么问题。

3、先执行订单本地事务，还是先发消息到MQ?

要分析这里的问题，我们就需要考虑一个事情：我们应该是先执行订单本地事务，还是先发消息到MQ去?

如果我们先执行订单本地事务，接着再发送消息到MQ的话，看起来伪代码可能是这样的：

```
try {
    // 执行订单本地事务
    orderService.finishOrderPay();
    // 发送消息到MQ去
    producer.sendMessage();
} catch(Exception e) {
    // 如果发送消息失败了，进行重试
    for(int i = 0; i < 3; i++) {
        // 重试发送消息
    }
    // 如果多次重试发送消息之后，还是不行
    // 回滚本地订单事务
    orderService.rollbackOrderPay();
}
```

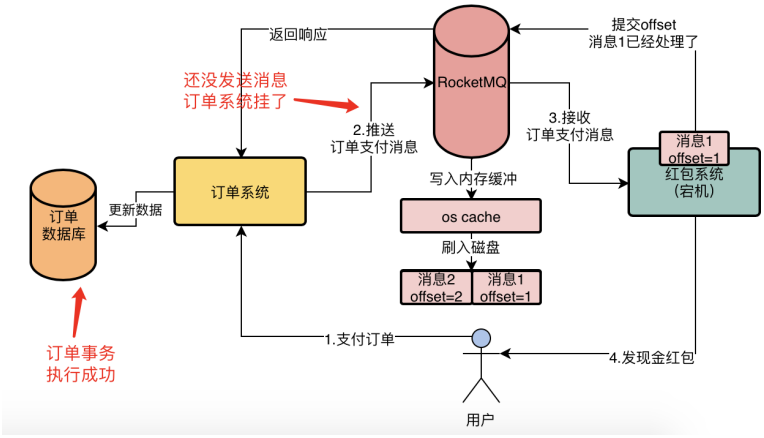
上面那段伪代码看着似乎天衣无缝，先执行订单本地事务，接着发送消息到MQ，如果订单本地事务执行失败了，则不会继续发送消息到MQ了；

如果订单事务执行成功了，发送MQ失败了，自动进行几次重试，重试如果一直失败，就回滚订单事务。

但是这里有一个问题，假设你刚执行完成了订单本地事务了，结果还没等到你发送消息到MQ，结果你的订单系统突然崩溃了！

这就导致你的订单状态可能已经修改为了“已完成”，但是消息却没发送到MQ去！这就是这个方案最大的隐患。

我们看下面的图



如果出现这种场景，那你的多次重试发送MQ之类的代码根本没机会执行！而且订单本地事务还已经执行成功了，你的消息没发送出去，红包系统没机会派发红包，必然导致用户支付成功了，结果看不到自己的红包！

4、把订单本地事务和重试发送MQ消息放到一个事务代码中

我们接着来考虑下一个问题，这个时候有人会想到一个新的想法，如果把订单本地事务代码和发送MQ消息的代码放到一个事务代码中呢？

我们看下面的伪代码示例：

```
// 直接在方法上加入事务注解
@Transactional
public void payOrderSuccess() {
    try {
        // 执行订单本地事务
        orderService.finishOrderPay();
        // 发送消息到MQ去
        producer.sendMessage();
    } catch(Exception e) {
        // 如果发送消息失败了，进行重试
        for(int i = 0; i < 3; i++) {
        // 重试发送消息
        }
        // 如果多次重试发送消息之后，还是不行
        // 抛出异常，去回滚本地订单事务
        throw new XXXException();
    }
}
```

上面这个代码看起来似乎解决了我们的问题，就是在这个方法上加入事务，在这个事务方法中，我们哪怕执行了orderService.finishOrderPay()，但是其实也仅仅执行了一些增删改SQL语句，还没提交订单本地事务。

如果发送MQ消息失败了，而且多次重试还不奏效，则我们抛出异常会自动回滚订单本地事务；

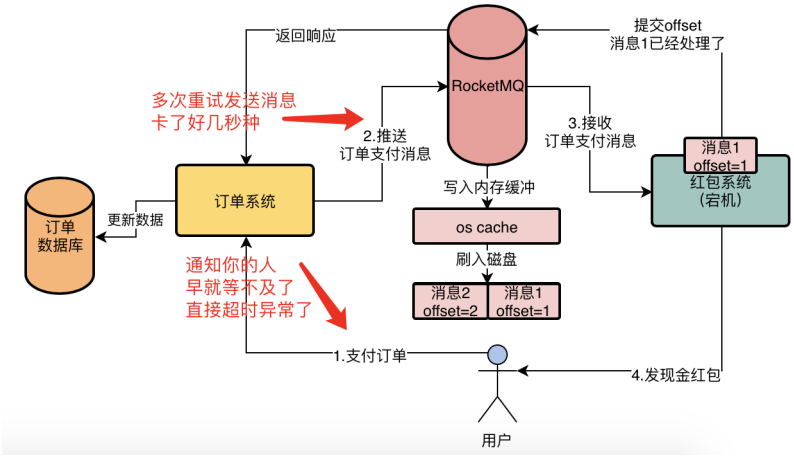
如果你刚执行了orderService.finishOrderPay()，结果订单系统直接崩溃了，此时订单本地事务会回滚，因为根本没提交过。

但是对于这个方案，还是非常的不理想，原因就出在那个MQ多次重试的地方

假设用户支付成功了，然后支付系统回调通知你的订单系统说，有一笔订单已经支付成功了，这个时候你的订单系统卡在多次重试MQ的代码那里，可能耗时了好几秒种，此时回调通知你的系统早就等不及可能都超时异常了。

而且你把重试MQ的代码放在这个逻辑里，可能会导致订单系统的这个接口性能很差

我们看下图的示意



5、再说了，你就一定可以依靠本地事务回滚吗?

除了我们上面说的那个问题之外，我可能还不得不给很多寄希望于订单事务和发送MQ消息包裹在一个事务代码中的朋友，泼一盆冷水，大家觉得我们一定可以依靠本地事务回滚吗?

这还真的未必，我们看下面的代码：

```
// 直接在方法上加入事务注解
@Transactional
public void payOrderSuccess() {
    try {
        // 执行订单本地事务
        orderService.finishOrderPay();
        // 更新Redis缓存
        orderService.updateRedisCache();
        // 更新Elasticsearch数据
        orderService.updateEsData();
        // 发送消息到MQ去
        producer.sendMessage();
    } catch(Exception e) {
        // 如果发送消息失败了，进行重试
        for(int i = 0; i < 3; i++) {
        // 重试发送消息
        }
        // 如果多次重试发送消息之后，还是不行
        // 抛出异常，去回滚本地订单事务
        throw new XXXException();
    }
}
```

大家着重在里面看，我们虽然在方法上加了事务注解，但是代码里还有更新Redis缓存和Elasticsearch数据的代码逻辑，如果你要是已经完成了订单数据库更新、Redis缓存更新、ES数据更新了，结果没法送MQ呢订单系统崩溃了。

虽然订单数据库的操作会回滚，但是Redis、Elasticsearch中的数据更新会自动回滚吗?

不会的，因为他们根本没法自动回滚，此时数据还是会不一致的。所以说，完全寄希望于本地事务自动回滚是不现实的。

6、保证业务系统一致性的最佳方案：基于RocketMQ的事务消息机制

所以分析完了这个同步发送消息 + 反复多次重试的方案之后，我们会发现他实际落地的时候是可以的，但是里面存在一些问题

比如可能会让订单事务执行成功，结果消息没发送出去，或者是订单事务执行成功了，但是反复多次重试发送消息到MQ极为耗时，导致调用你接口的人频繁超时异常。

所以真正要保证消息一定投递到MQ，同时保证业务系统之间的数据完全一致，业内最佳的方案还是用基于RocketMQ的事务消息机制。

因为这个方案落地之后，他可以保证你的订单系统的本地事务一旦成功，那么必然会投递消息到MQ去，通知红包系统去派发红包，保证业务系统的数据是一致的。而且整个流程中，你没必要进行长时间的阻塞和重试。

如果half消息发送就失败了，你就直接回滚整个流程。如果half消息发送成功了，后续的rollback或者commit发送失败了，你不需要自己去卡在那里反复重试，你直接让代码结束即可，因为后续MQ会过来回调你的接口让你判断再次rollback or commit的。

希望大家在学习一个技术的时候，反复给自己多问几个为什么，从各个角度去思考。

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**）
- 互联网Java工程师面试突击（第1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）**