图文 72 Broker消息零丢失方案：同步刷盘 + Raft协议主从同步

详情 评论

Broker消息零丢失方案：同步刷盘 + Raft协议主从同步

继《从零开始带你成为JVM实战高手》后，救火队长携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、用了事务消息机制，消息就一定不会丢了吗?

之前我们花了很多的篇幅去讲RocketMQ的事务消息机制，原因无他，就是这个机制是RocketMQ非常核心以及重要的一个功能，可以让我们实现生产消息环节的消息不丢失，而且最重要的是，他可以保证两个业务系统的数据一致性。

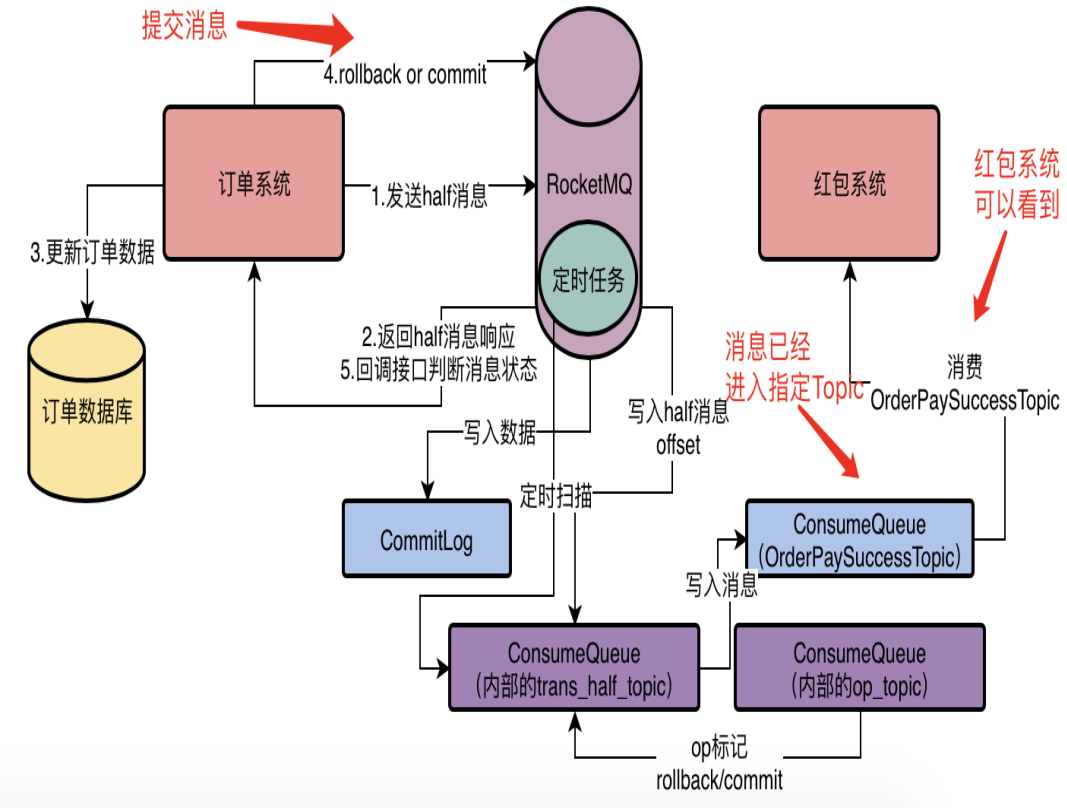
但是现在我们接着来思考，如果我们在生产消息的时候用了事务消息之后，就真的可以保证数据就不会丢失了吗?

那还真是未必

假设咱们现在订单系统已经通过事务消息的机制，通过half消息 + commit的方式，把消息在MQ里提交了

也就是说，现在对于MQ而言，那条消息已经进入到他的存储层了，可以被红包系统看到了

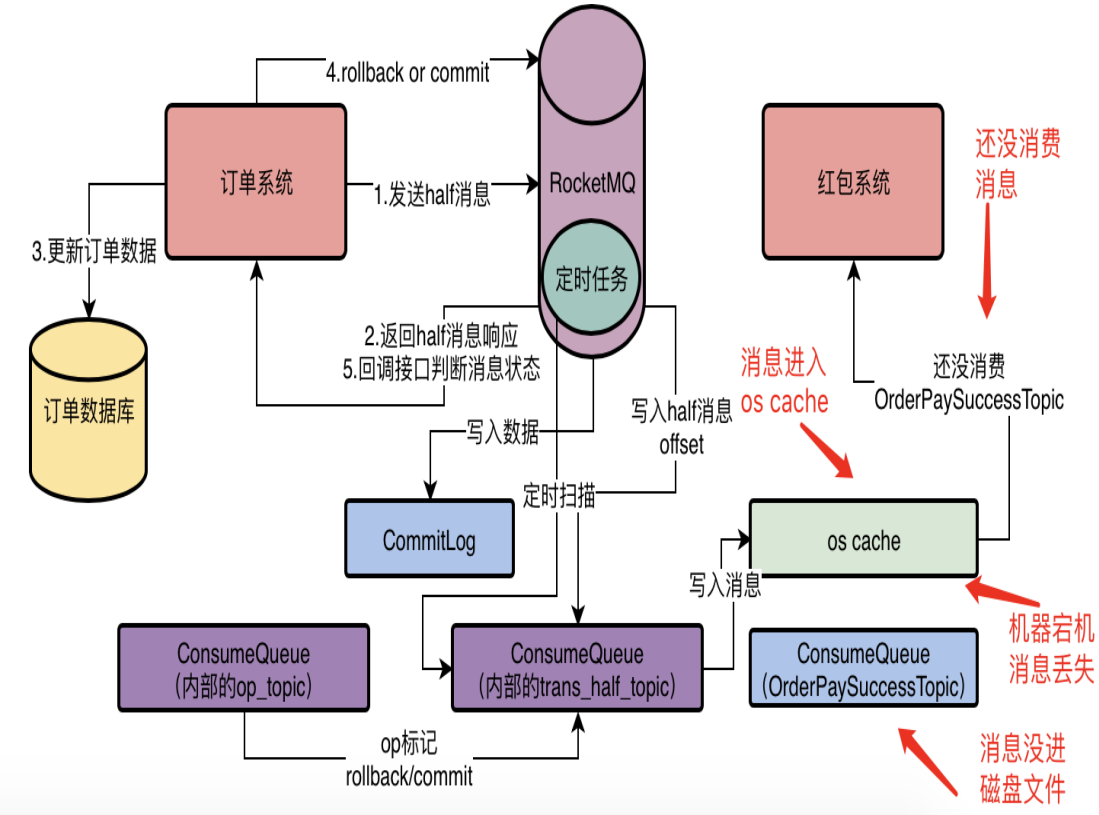
我们看下图



但是我们先稍微等一下，你的这条消息在commit之后，会从half topic里进入OrderPaySuccessTopic中，但是此时仅仅是消息进入了这个你预定的Topic而已，仅仅是可以被红包系统看到而已，此时可能你的红包系统还没来得及去获取这条消息。

然后恰巧在此时，你的这条消息又仅仅停留在os cache中，还没进入到ConsumeQueue磁盘文件里去，然后此时这台机器突然宕机了，os cache中的数据全部丢失，此时必然会导致你的消息丢失，红包系统再没机会读到这条消息了。

我们看下图的示意



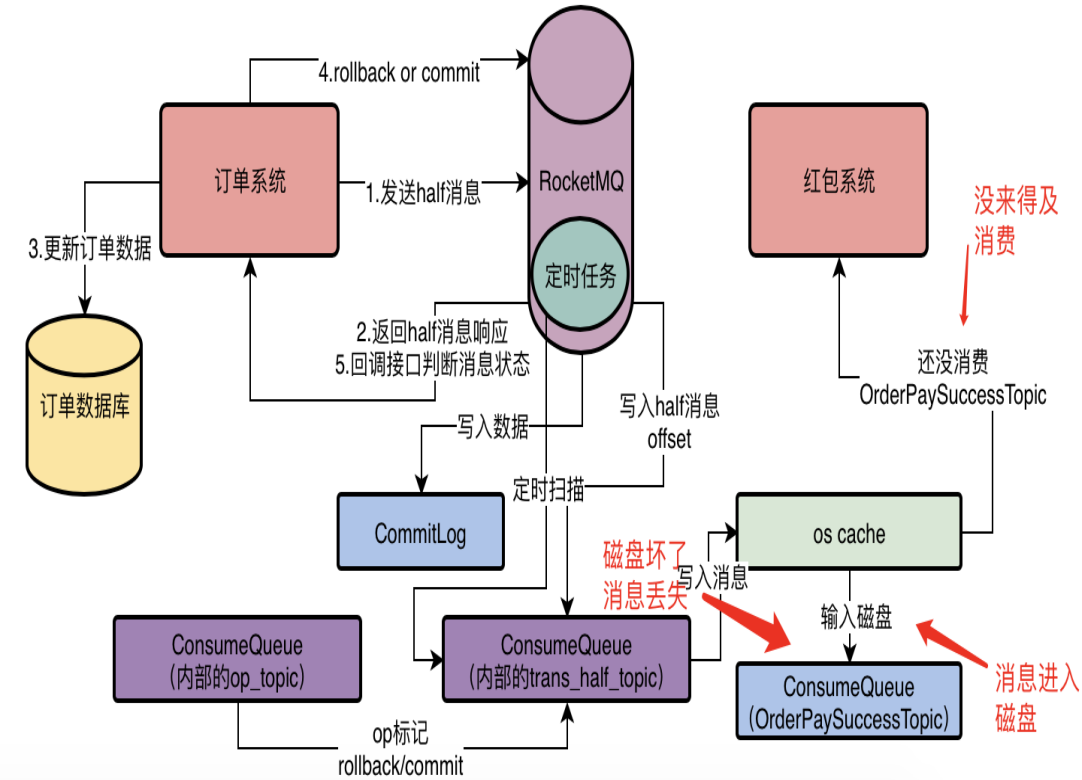
2、就算你走运，消息进了磁盘就不会丢了吗?

那我们接着看，就算我们很走运，比如你的消息已经进入了OrderPaySuccessTopic的ConsumeQueue磁盘文件了，不是停留在os cache里了，此时消息就一定不会丢失了吗?

这也未必

即使消息已经进入磁盘文件了，但是这个时候红包系统还没来得及消费这条消息，然后此时这台机器的磁盘突然就坏了，就会一样导致消息丢失，而且可能消息再也找不回来了，同样会丢失数据。

我们看下图的示意。



3、明确一个前提：保证消息写入MQ不代表不丢失

所以看到这里，我们需要明确一个前提，我们无论是通过比较简单的同步发送消息 + 反复多次重试的方案，还是事务消息的方案，哪怕我们确保消息已经写入MQ成功了，此时也未必消息就不会丢失了。

因为即使你写入MQ成功了，这条消息也大概率是仅仅停留在MQ机器的os cache中，一旦机器宕机内存里的数据都会丢失，或者哪怕消息已经进入了MQ机器的磁盘文件里，但是磁盘一旦坏了，消息也会丢失。

如果消息丢失了，你的红包系统还没来得及消费，那么他就永远没机会消费和派发红包了，所以对于你而言，如果你仅仅只是使用MQ的话，可能不清楚MQ集群内部发生过的一些机器故障，也就不清楚数据丢失的具体原因了。

4、异步刷盘 vs 同步刷盘

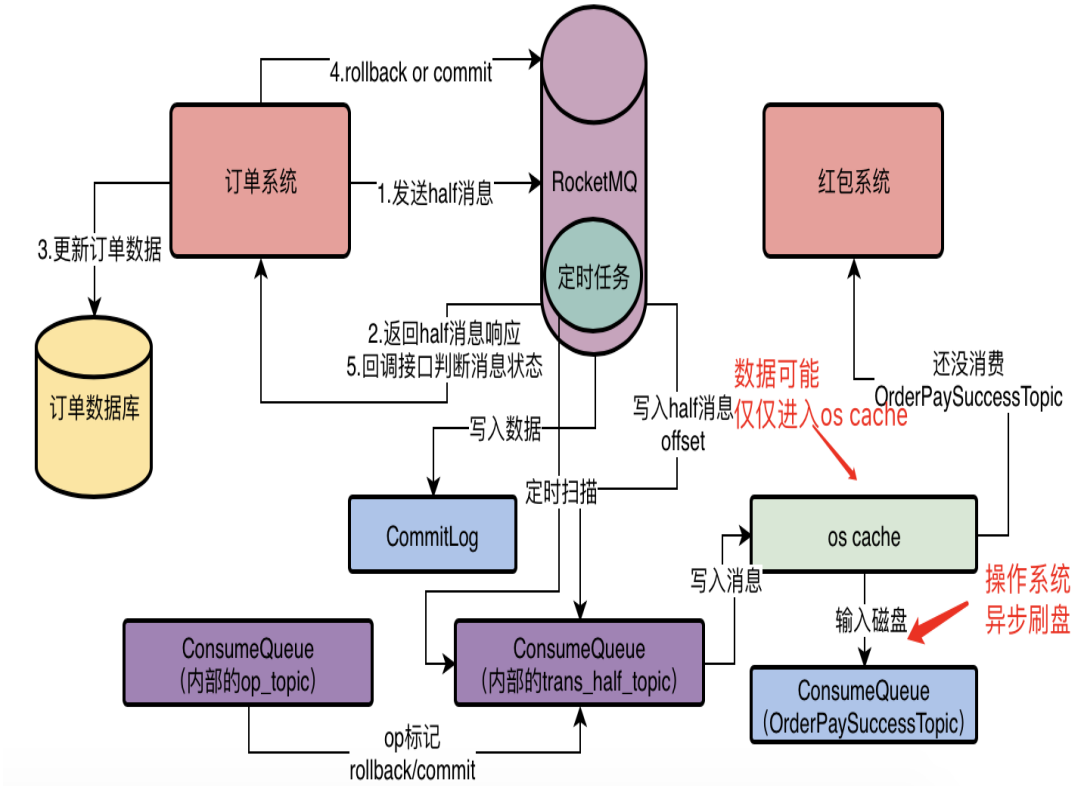
说到这里，我们终于可以进入正题了，到底怎么去确保消息写入MQ之后，MQ自己不要随便丢失数据呢？

解决这个问题的第一个关键点，就是将异步刷盘调整为同步刷盘。

所谓的异步刷盘，就是之前我们一直说的那种模式。

也就是说，你的消息即使成功写入了MQ，他也就在机器的os cache中，没有进入磁盘里，要过一会儿等操作系统自己把os cache里的数据实际刷入磁盘文件中去

我们看下图的示意



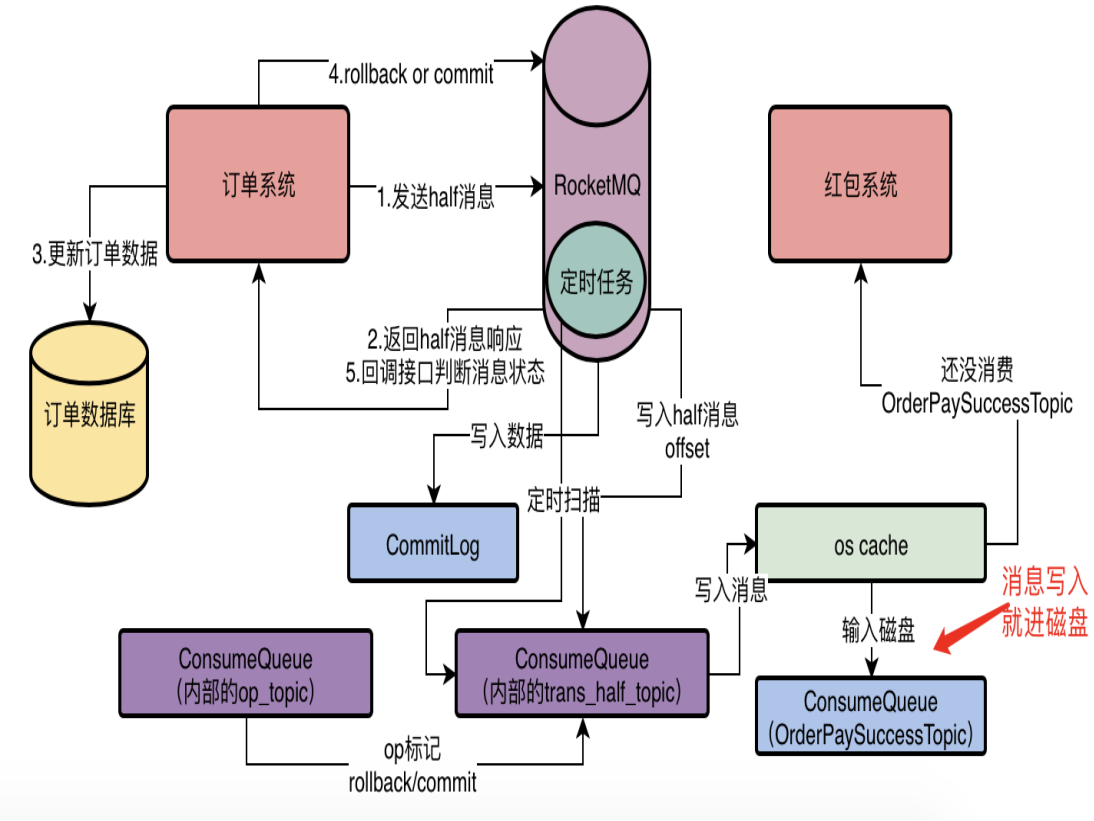
所以在异步刷盘的模式下，我们的写入消息的吞吐量肯定是极高的，毕竟消息只要进入os cache这个内存就可以了，写消息的性能就是写内存的性能，那每秒钟可以写入的消息数量肯定更多了，但是这个情况下，可能就会导致数据的丢失。

所以如果一定要确保数据零丢失的话，可以调整MQ的刷盘策略，我们需要调整broker的配置文件，将其中的flushDiskType配置设置为：SYNC_FLUSH，默认他的值是ASYNC_FLUSH，即默认是异步刷盘的。

如果调整为同步刷盘之后，我们写入MQ的每条消息，只要MQ告诉我们写入成功了，那么他们就是已经进入了磁盘文件了！

比如我们发送half消息的时候，只要MQ返回响应是half消息发送成功了，那么就说明消息已经进入磁盘文件了，不会停留在os cache里。

我们看下图的示意，如果我们使用同步刷盘的策略，那么可以确保写入MQ的消息一定是已经进入磁盘文件了



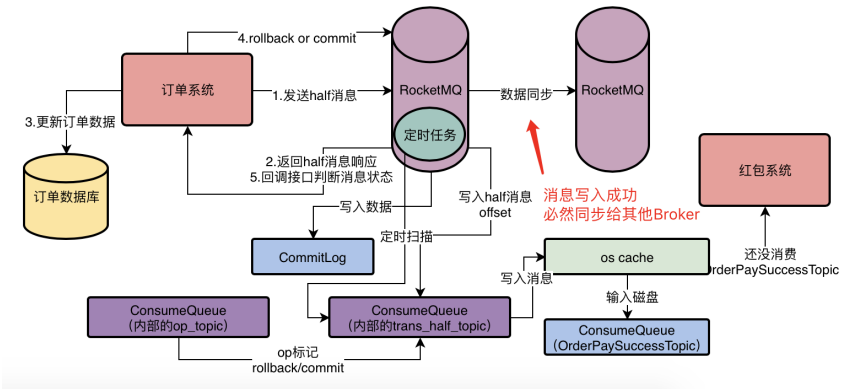
5、如何通过主从架构模式避免磁盘故障导致的数据丢失?

接着我们解决下一个问题，如何避免磁盘故障导致的数据丢失?

其实道理也很简单，我们必须要对Broker使用主从架构的模式

也就是说，必须让一个Master Broker有一个Slave Broker去同步他的数据，而且你一条消息写入成功，必须是让Slave Broker也写入成功，保证数据有多个副本的冗余。

我们看下图的一个示意



这样一来，你一条消息但凡写入成功了，此时主从两个Broker上都有这条数据了，此时如果你的Master Broker的磁盘坏了，但是Slave Broker上至少还是有数据的，数据是不会因为磁盘故障而丢失的。

对于主从同步的架构，我们本来就是讲解了基于DLedger技术和Raft协议的主从同步架构，你如果采用了这套架构，对于你所有的消息写入，只要他写入成功，那就一定会通过Raft协议同步给其他的Broker机器，这里的原理我们之前都已经讲解过了，大家有遗忘的回去看看即可。

6、MQ确保数据零丢失的方案总结

所以通过今天的分析，我们知道了，只要你把Broker的刷盘策略调整为同步刷盘，那么绝对不会因为机器宕机而丢失数据;

只要你采用了主从架构的Broker集群，那么一条消息写入成功，就意味着多个Broker机器都写入了，此时任何一台机器的磁盘故障，数据也是不会丢失的。

最起码只要Broker层面保证写入的数据不丢失，那就一定可以让红包系统消费到这条消息了！

End

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

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**）