

## 05 | 如何确保消息不会丢失？

李玥 · 消息队列高手课



你好，我是李玥。这节课我们来聊聊丢消息的事儿。

对于刚刚接触消息队列的同学，最常遇到的问题，也是最头痛的问题就是丢消息了。对于大部分业务系统来说，丢消息意味着数据丢失，是完全无法接受的。

其实，现在主流的消息队列产品都提供了非常完善的消息可靠性保证机制，完全可以做到在消息传递过程中，即使发生网络中断或者硬件故障，也能确保消息的可靠传递，不丢消息。

绝大部分丢消息的原因都是由于开发者不熟悉消息队列，没有正确使用和配置消息队列导致的。虽然不同的消息队列提供的 API 不一样，相关的配置项也不同，但是在保证消息可靠传递这块儿，它们的实现原理是一样的。

这节课我们就来讲一下，消息队列是怎么保证消息可靠传递的，这里面的实现原理是怎么样的。当你熟知原理以后，无论你使用任何一种消息队列，再简单看一下它的 API 和相关配置项，就能很快知道该如何配置消息队列，写出可靠的代码，避免消息丢失。

## 检测消息丢失的方法

我们说，用消息队列最尴尬的情况不是丢消息，而是消息丢了还不知道。一般而言，一个新的系统刚刚上线，各方面都不太稳定，需要一个磨合期，这个时候，特别需要监控到你的系统中是否有消息丢失的情况。

如果是 IT 基础设施比较完善的公司，一般都有分布式链路追踪系统，使用类似的追踪系统可以很方便地追踪每一条消息。如果没有这样的追踪系统，这里我提供一个比较简单的方法，来检查是否有消息丢失的情况。

**我们可以利用消息队列的有序性来验证是否有消息丢失。**原理非常简单，在 Producer 端，我们给每个发出的消息附加一个连续递增的序号，然后在 Consumer 端来检查这个序号的连续性。

如果没有消息丢失，Consumer 收到消息的序号必然是连续递增的，或者说收到的消息，其中的序号必然是上一条消息的序号 +1。如果检测到序号不连续，那就是丢消息了。还可以通过缺失的序号来确定丢失的是哪条消息，方便进一步排查原因。

大多数消息队列的客户端都支持拦截器机制，你可以利用这个拦截器机制，在 Producer 发送消息之前的拦截器中将序号注入到消息中，在 Consumer 收到消息的拦截器中检测序号的连续性，这样实现的好处是消息检测的代码不会侵入到你的业务代码中，待你的系统稳定后，也方便将这部分检测的逻辑关闭或者删除。

如果是在一个分布式系统中实现这个检测方法，有几个问题需要你注意。

首先，像 Kafka 和 RocketMQ 这样的消息队列，它是不保证在 Topic 上的严格顺序的，只能保证分区上的消息是有序的，所以我们在发消息的时候必须要指定分区，并且，在每个分区单独检测消息序号的连续性。

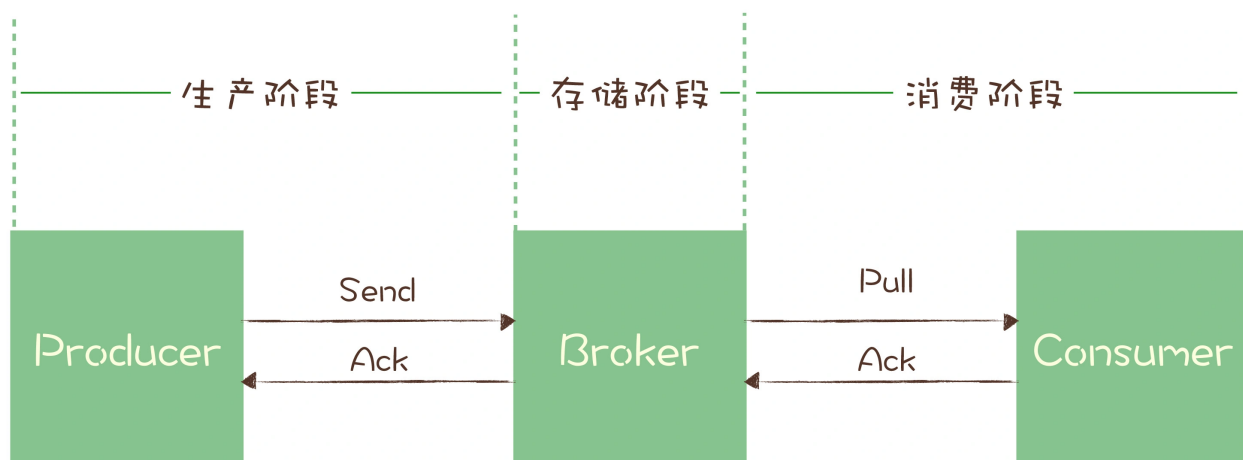
如果你的系统中 Producer 是多实例的，由于并不好协调多个 Producer 之间的发送顺序，所以也需要每个 Producer 分别生成各自的消息序号，并且需要附加上 Producer 的标识，在 Consumer 端按照每个 Producer 分别来检测序号的连续性。

Consumer 实例的数量最好和分区数量一致，做到 Consumer 和分区一一对应，这样会比较方便地在 Consumer 内检测消息序号的连续性。

## 确保消息可靠传递

讲完了检测消息丢失的方法，接下来我们一起来看一下，整个消息从生产到消费的过程中，哪些地方可能会导致丢消息，以及应该如何避免消息丢失。

你可以看下这个图，一条消息从生产到消费完成这个过程，可以划分三个阶段，为了方便描述，我给每个阶段分别起了个名字。



**生产阶段:** 在这个阶段，从消息在 Producer 创建出来，经过网络传输发送到 Broker 端。

**存储阶段:** 在这个阶段，消息在 Broker 端存储，如果是集群，消息会在这个阶段被复制到其他的副本上。

**消费阶段:** 在这个阶段，Consumer 从 Broker 上拉取消息，经过网络传输发送到 Consumer 上。

### 1. 生产阶段


在生产阶段，消息队列通过最常用的请求确认机制，来保证消息的可靠传递：当你的代码调用发消息方法时，消息队列的客户端会把消息发送到 Broker，Broker 收到消息后，会给客户端

返回一个确认响应，表明消息已经收到了。客户端收到响应后，完成了一次正常消息的发送。

只要 Producer 收到了 Broker 的确认响应，就可以保证消息在生产阶段不会丢失。有些消息队列在长时间没收到发送确认响应后，会自动重试，如果重试再失败，就会以返回值或者异常的方式告知用户。


你在编写发送消息代码时，需要注意，正确处理返回值或者捕获异常，就可以保证这个阶段的消息不会丢失。以 Kafka 为例，我们看一下如何可靠地发送消息：

同步发送时，只要注意捕获异常即可。

 复制代码

```
1 try {
2     RecordMetadata metadata = producer.send(record).get();
3     System.out.println("消息发送成功。");
4 } catch (Throwable e) {
5     System.out.println("消息发送失败！");
6     System.out.println(e);
7 }
```

异步发送时，则需要在回调方法里进行检查。这个地方是需要特别注意的，很多丢消息的原因就是，我们使用了异步发送，却没有在回调中检查发送结果。

 复制代码

```
1 producer.send(record, (metadata, exception) -> {
2     if (metadata != null) {
3         System.out.println("消息发送成功。");
4     } else {
5         System.out.println("消息发送失败！");
6         System.out.println(exception);
7     }
8 });
```

## 2. 存储阶段

在存储阶段正常情况下，只要 Broker 在正常运行，就不会出现丢失消息的问题，但是如果 Broker 出现了故障，比如进程死掉了或者服务器宕机了，还是可能会丢失消息的。

**如果对消息的可靠性要求非常高，可以通过配置 Broker 参数来避免因为宕机丢消息。**

对于单个节点的 Broker，需要配置 Broker 参数，在收到消息后，将消息写入磁盘后再给 Producer 返回确认响应，这样即使发生宕机，由于消息已经被写入磁盘，就不会丢失消息，恢复后还可以继续消费。例如，在 RocketMQ 中，需要将刷盘方式 flushDiskType 配置为 SYNC\_FLUSH 同步刷盘。

如果是 Broker 是由多个节点组成的集群，需要将 Broker 集群配置成：至少将消息发送到 2 个以上的节点，再给客户端回复发送确认响应。这样当某个 Broker 宕机时，其他的 Broker 可以替代宕机的 Broker，也不会发生消息丢失。后面我会专门安排一节课，来讲解在集群模式下，消息队列是如何通过消息复制来确保消息的可靠性的。

### 3. 消费阶段

消费阶段采用和生产阶段类似的确认机制来保证消息的可靠传递，客户端从 Broker 拉取消息后，执行用户的消费业务逻辑，成功后，才会给 Broker 发送消费确认响应。如果 Broker 没有收到消费确认响应，下次拉消息的时候还会返回同一条消息，确保消息不会在网络传输过程中丢失，也不会因为客户端在执行消费逻辑中出错导致丢失。

**你在编写消费代码时需要注意的是，不要在收到消息后就立即发送消费确认，而是应该在执行完所有消费业务逻辑之后，再发送消费确认。**

同样，我们以用 Python 语言消费 RabbitMQ 消息为例，来看一下如何实现一段可靠的消费代码：

 复制代码

```
1 def callback(ch, method, properties, body):
2     print(" [x] 收到消息 %r" % body)
3     # 在这儿处理收到的消息
4     database.save(body)
5     print(" [x] 消费完成")
```



```
6      # 完成消费业务逻辑后发送消费确认响应
7      ch.basic_ack(delivery_tag = method.delivery_tag)
8
9  channel.basic_consume(queue='hello', on_message_callback=callback)
```

你可以看到，在消费的回调方法 `callback` 中，正确的顺序是，先是把消息保存到数据库中，然后再发送消费确认响应。这样如果保存消息到数据库失败了，就不会执行消费确认的代码，下次拉到的还是这条消息，直到消费成功。

## 小结

这节课我带大家分析了一条消息从发送到消费整个流程中，消息队列是如何确保消息的可靠性，不会丢失的。这个过程可以分为三个阶段，每个阶段都需要正确的编写代码并且设置正确的配置项，才能配合消息队列的可靠性机制，确保消息不会丢失。

在生产阶段，你需要捕获消息发送的错误，并重发消息。

在存储阶段，你可以通过配置刷盘和复制相关的参数，让消息写入到多个副本的磁盘上，来确保消息不会因为某个 Broker 宕机或者磁盘损坏而丢失。

在消费阶段，你需要在处理完全部消费业务逻辑之后，再发送消费确认。

你在理解了这几个阶段的原理后，如果再出现丢消息的情况，应该可以通过在代码中加一些日志的方式，很快定位到是哪个阶段出了问题，然后再进一步深入分析，快速找到问题原因。

## 思考题

我刚刚讲到，如果消息在网络传输过程中发送错误，由于发送方收不到确认，会通过重发来保证消息不丢失。但是，如果确认响应在网络传输时丢失，也会导致重发消息。也就是说，**无论是 Broker 还是 Consumer 都是有可能收到重复消息的**，那我们在编写消费代码时，就需要考虑这种情况，你可以想一下，在消费消息的代码中，该如何处理这种重复消息，才不会影响业务逻辑的正确性？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (142)



业余草

2019-08-01

一句话，消费做好幂等性即可！

共 9 条评论 >

👍 189



钱

2019-08-20

如何确保消息不会丢失

1: WAL

2: 分布式WAL

除非地球爆炸，否则问题不大。

猜测各种消息队列或者数据库，确保消息不丢只能这么玩，就连人也一样，脑袋记不住那就写下来，怕本子弄潮啦！那就光盘、U盘、磁盘、布头、木头、石头北京、上海、深圳都各写一份。

consumer接到重复消息，那就业务去重，怎么去？

1: 业务处理逻辑本身就是幂等的，那天然就去掉了

2: 业务处理逻辑非幂等，那就消息先去重，根据业务ID(标识消息唯一性的就行)，去查询是否消费过此消息了，消费了，则抛弃，否则就消费

作者回复: 没毛病，很多底层的方法和技术就是通用的

共 4 条评论 >

👍 115



ly

2019-08-02

老师，我有几个理解：

当producer发送消息给blocker的时候（send方法），此方法会在blocker收到消息并正常储存后才返回，此期间应该会阻塞，也就是如果blocker配置同步刷盘，可能会增加调用时间（只能出现对消息敏感的场景）。

另外拉消息的时候，消费者A进行pull后，没有返回确认给blocker就挂了（或者因代码问题导致一直阻塞），这时消息应该还在blocker的，消费者B如果此时pull消息，是否会拉取到刚刚那条给消费者A的消息？衍生的疑问就是两个消费者先后去拉消息是否能拉到同一条消息（在前者未给blocker发确认的前提下）。

对于消费者处理重复消息的问题：一般消息中都会存在一个唯一性的东西，不管是消息队列本

身的msgId还是业务订单号之类的，可以在db中存在一个消费表，对这个唯一性东西建立唯一索引，每次处理消费者逻辑之前先insert进去，让数据库来帮我们排重我觉得是最保险的。

作者回复: 两个消费者先后去拉消息是否能拉到同一条消息？

首先，消息队列一般都会有协调机制，不会让这种情况出现，但是由于网络不确定性，这种情况还是在极小概率下会出现的。

在同一个消费组内，A消费者拉走了index=10的这条消息，还没返回确认，这时候这个分区的消费位置还是10，B消费者来拉消息，可能有2种情况：

1. 超时前，Broker认为这个分区还被A占用着，会拒绝B的请求。
2. 超时后，Broker认为A已经超时没返回，这次消费失败，当前消费位置还是10，B再来拉消息，会给它返回10这条消息。

共 5 条评论 >

👍 70



王立光

2019-08-15

假如消费时由于某种原因，一直没发ack。rocketmq是不是会一直发这条消息，这样导致下面消息都无法被消费？

作者回复: 是的。

rocketmq为了解决这个问题，增加了一个死信队列，对于这种反复投递都无法成功的消息，会被移动到死信队列中，避免卡住其他消息。

共 5 条评论 >

👍 59



kane

2019-08-01

产生重复消息原因：

- (1).发送消息阶段，发送重复的消息
- (2) 消费消息阶段，消费重复的消息。

解决办法：

业务端去重

- 1) 建立一个消息表，consumer消费之前，拿到消息做insert操作，用消息id做唯一主键，重复消费会导致主键冲突。



2) 利用redis, 给消息分配一个全局id, 只要消费过该消息, 将消息以K-V (< id,message >) 形式写入redis, 消费消息之前, 根据key去redis查询是否有对应记录。

共 14 条评论 >

👍 40



南山

2019-10-19

老师, 再看一次这个的时候有个疑问:

1, broker给producer发送ack, 网络原因或者其他原因producer没收到, 这种情况broker这条消息怎么办?

2, broker给producer发送ack, broker怎么知道producer有没有成功收到这条ack信息呢?

作者回复: Broker不知道ack有没有送达到producer, 它也不管是不是送达了。

因为, 这个阶段的消息可靠性是由producer来保证的, 它发了消息, 收到了ack, 那消息就一定送达了。如果没收到ack, 那消息有可能送达了(ack丢失), 也有可能没送达(消息丢失), 对于这种情况, producer一般会重发消息, 确保消息送达。这也就是为什么消息会有重复的原因之一。

共 2 条评论 >

👍 33



陈天柱

2019-10-19

老师您好, 我最近处理一个分布式事务场景, 就是支付成功以后回调处理刷新订单状态和刷新优惠券状态的分布式事务问题。我有一个疑问就是, 假如刷新订单状态失败, 就没有发消息到队列中, 而是在刷新订单状态成功了以后, 再发消息到队列中, 只要保证消息不丢的前提下, 分布式事务能得到保证吗? 换言之, 就是消息不丢的话, 可以保证最终一致性, 事务消息的回查也能保证最终一致性, 两者的概念感觉有一些交集。所以希望老师能抽空帮忙分析一下, 万分感谢🙏

作者回复: 你这个例子中, 最终一致性应该是指: “经过一段时间后, 订单状态和优惠券状态最终会保持一致。”

但是, 你这个场景不太适合使用事务消息来解决, 虽然和我们上节课中的例子相比, 只是把购物车换成了优惠券。但你有没有考虑到, 有人会恶意利用这个短暂的不一致时间来刷优惠券? 比如, 利用下单成功, 但优惠券还没来得及扣减这个时间差, 一个优惠券反复下单?

共 4 条评论 >

👍 26



QQ怪

2019-08-01

建议老师加餐如何做幂等性

作者回复: 不用加餐, 这是教学大纲内的内容, 下节课就会讲到滴。



15



angel txy

2019-08-20

思考题, 从两个环节展开, 生产端到broker, MQ系统内部必须生成一个inner-msg-id, 作为去重和幂等的依据, 这个内部消息ID的特性是:

(1) 全局唯一

(2) MQ生成, 具备业务无关性, 对消息发送方和消息接收方屏蔽

有了这个inner-msg-id, 就能保证上半场重发, 也只有1条消息落到MQ-server的DB中, 实现上半场幂等。

broker到consumer, 由消费端做好幂等性, 比如根据业务流水号去重

作者回复: 这个总结非常到位!

共 7 条评论 >

13



皮蛋

2019-08-04

rocketmq默认失败重试次数是2, 如果2次均失败, 或者说重试次数都失败了, 这种情况一般在实际生产中是怎么处理的?

作者回复: 放到私信队列中, 人工处理。

共 4 条评论 >

12



TH

2019-08-01

幂等性是一种办法, 如果做不到幂等性, 那么在消费端需要存储消费的消息ID, 关键这个ID什么时候存? 如果是消费前就存, 那么消费失败了, 下次消费同样的消息, 是否会认为上次已经成功了? 如果在消费成功后再存, 那么消费会不会出现部分成功的情况? 除非满足事务

ACID特性。

关于消息丢失检查还有一点疑问：如果靠ID连续性来检查，是不是说一个producer只能对应一个consumer？

作者回复：不用，Producer发消息的时候带着ProducerId并要指定分区发送，Consumer消费的时候，需要按照每个Producer来检查序号的连续性。



👍 10



**游弋云端**

2019-08-01

- 1、消费端支持幂等操作，业务上一般有难度；
- 2、消费端增加去冗余机制，例如缓存最新消费成功的N条消息的SN，收到消息后，先确认是否是消费过的消息，如果是，直接应该ACK，并放弃消费。

作者回复：思路是没问题的。



👍 9



**Better me**

2019-08-01

对于思考题，我认为也可以像老师说的那样查看消息是否丢失的方法，如果Producer的某条消息ack相应因为网络故障丢失，那么Producer此时重发消息的唯一标识应该和之前那条消息是一样的，那么只需要在Consumer接受消息前判断是否有相同标识的消息，如果有则拦截。还可以在消费端业务逻辑接口中做幂等判断，前面那种可以做到不侵入到业务代码中，老师看看有没有什么问题

作者回复：非常好！但你需要考虑一下，在分布式环境中“Consumer接受消息前判断是否有相同标识的消息”该如何实现呢？

共 3 条评论 >

👍 8



**史双龙**

2019-08-01

玥哥好，我jio着只要在消费端做好幂等就可以，业务借口最好都要做幂等性校验，

作者回复: 你这结论都是用无数bug换来的呀。



又双叒叕是一年啊

2019-08-31

老师您好我看了这篇文章有几个疑惑希望您帮忙解答下:

1.kafka 和 rocketmq 都不能保证全局topic的顺序性, 只能保证 partition 分区的信息顺序性, 那我需要保证全局唯一性的场景要如何处理比较好, 都有哪些方式可以保证消息的全局顺序性?

文章里您说 可以在 Producer 分别生成各自的消息序号,并且需要附加上 Producer 的标识, 在 Consumer 端按照每个 Producer 分别来检测序号的连续性。

这个句话的意识是说 在 producer 生产消息时要生成一个 全局有序递增的消息id? 然后再 consumer消费时 通过这个id进行消费控制?

因为一个 consumer group 可能包含 多个 consumer , 这个要怎么控制分布式环境下多个consumer消费的顺序呢? 还有consumer也可能发送 rebalance操作导致消费分区发生变化

2.确保消息可靠传递方面

#1.如果 存储阶段 发生极端情况: broker 在收到 消息后还未来得及写入磁盘就发生 broker 掉电宕机情况, 这个消息会丢失?

还是producer端先进行自动重试处理, broker都挂了, 重试肯定会超时失败, 如果达到producer端重试最大次数还是失败, 才会向producer端抛出异常

而 producer端 可以捕获这个异常 进行业务的补偿重试 或 降级 保证消息不丢 (入队列 or redis or 入库), 再进行 后续处理。是这样? 不知道我理解的是否正确

提个建议: 希望老师把关键参数 列一下 或者 给个参考链接 我们去看下不熟悉现找确实比较费劲 不同版本差异也比较大

#2.对于消费阶段

在消费阶段, 你需要在处理完全部消费业务逻辑之后, 再发送消费确认->

这个是在生产阶段都采用 手动 提交方式处理? consumer 每次pull 过程中自动提交会有问题?

比如 文章中例子, 在自动提交情况下 保存db出异常, 在下次 拉取消息时会自动提交刚才的消息, 所以才要求手动提交消费进度?

作者回复: A1: 关于第一个问题, 你可以看一下“08疑难解答”这节课, 里面有专门的一个小结来讲如何用消息队列实现严格顺序。

A2: 你的理解是正确的。关于参数配置, 只要你明白了收发消息的原理, 再去看一下配置的说明, 应

该很容易就理解了。每个消息队列的文档中，都会有专门的一篇文档，列出所有的配置以及每个配置的含义，你可以参考。

A3: 自动提交或者手动提交不是关键问题，关键的问题就是“先执行消费逻辑，再提交消费确认”。比如，RocketMQ Consumer的OnMessage方法，它会在成功执行完这个方法后提交消费确认，如果抛出异常就不会提交消费确认。你只要在这个方法中执行你的消费逻辑，消费错误的时候抛出异常，即使自动提交也能保证我们说的“先执行消费逻辑，再提交消费确认”。



5



凌空飞起的剪刀腿

2020-03-20

李老师您好，我想知道订阅者怎么知道broker有他订阅的消息的，不能一直轮询吧？

作者回复: 是的，常见的消息队列，比如RockMQ、Kafka都是一直轮询的。



4



虚竹

2019-12-24

老师好，关于消息丢失问题，不止发mq，也可能是发es、钉钉、短信、邮件等，如果需要每个业务自己保证消息发出，是比较困难的，  
在考虑应该以微服务的形式提供这些发消息的接口，微服务内部保证消息发送成功，可以通过时间递减等方式重试，一直不成功则  
持久化存储提醒人工操作，这里还可以加一些监控阈值报警异常发送情况，同时这里需要支持web搜索查看某个消息是否发送成功或失败，因为涉及到业务判罚，业务人员比较重视，  
想问下目前业界是否有开源的相关的实现？  
京东内部是如何处理这个问题的？  
另外我的思路是否有可行？  
期待老师帮忙解惑~

作者回复: 增加一个服务也不能解决这个问题，如果调用这个服务失败了怎么办？所以理论上，这个问题是无解的。

实际生产中，还是靠发送方检查发消息的响应来确认消息是否发成功了，并且，还要决定发失败的消息该如何处理。

比如，某些监控类应用，可以把发失败的消息直接丢掉。

再比如，重要的订单类的应用，他们会做多重的降级方案，一般建二个topic，第一个发失败了，换第二个发，再发失败了写数据库。

共 3 条评论 >

👍 4



angel 😊 txy 🇨🇳

2019-08-20

消费端消息丢失，请老师再展开讲一下，比如broker一直收不到消费者的ack，会怎么办？还有消费端消费时抛异常后，卡夫卡处理的？

作者回复：一般来说，消费都是采用pull的模式，消费者主动去broker来拉消息，如果消费者返回ack，Broker就会更新消费位置，如果消费者不返回ack，broker就什么都不做，直到下次消费者再来拉消息，因为消费位置没有更新，所以拉到的还是之前的那批消息，这样就实现了“重试”的效果。

在kafka中，消费位置都是用户代码来手动提交消费位置，所以，也就不存在“消费端消费时抛异常后，卡夫卡处理的”的问题，因为这完全取决于你的代码如何来编写了。

共 3 条评论 >

👍 4



skyun

2019-08-02

老师，我关于事务消息有个疑问：如果生产者在执行完本地事务后向broke提交确认，但是此时broke挂了，提交失败，broke因为挂了也无法进行回查，那么此时这条消息是不是就丢了，从而导致两个系统中数据不一致，还是说这个不一致只是暂时的，等broke重启后，依旧会根据halfMessage进行回查？望解答

作者回复：。如果Broker是集群模式，其他的Broker会替代宕机的Broker来继续进行反查。如果Broker是单节点，只能等到Broker恢复后再继续进行反查。无论哪种模式，消息不会丢，是保存在磁盘上的。

共 2 条评论 >

👍 4



王麒

2019-08-02

每次将消息刷磁盘会不会有性能损耗，有其他持久化方式吗？



作者回复: 一般的建议是用多台Broker配置成集群, 异步刷盘, 复制后再返回发送成功确认, 性能比同步刷盘要好一些。

共 3 条评论 >

 4