05-如何确保消息不会丢失-

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。

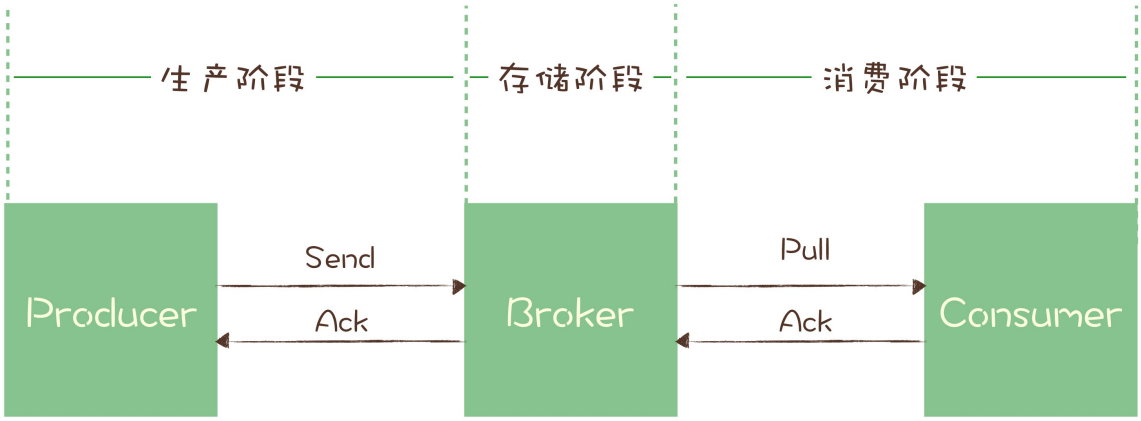
Consumer实例的数量最好和分区数量一致，做到Consumer和分区一一对应，这样会比较方便地在

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。

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

```
    System.out.println(e);
}
```

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。

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

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：

```
def callback(ch, method, properties, body):
```

```
    print(" [x] 收到消息 %r" % body)
    # 在这儿处理收到的消息
    database.save(body)
    print(" [x] 消费完成")
    # 完成消费业务逻辑后发送消费确认响应
    ch.basic_ack(delivery_tag = method.delivery_tag)

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。这个过程可以分为分三个阶段，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。

精选留言：

- 业余草 2019-08-01 08:36:39
  一句话，消费做好幂等性即可！ [8赞]

- QQ怪 2019-08-01 18:43:46
  建议老师加餐如何做幂等性 [6赞]

  作者回复2019-08-02 09:14:43
  不用加餐，这是教学大纲内的内容，下节课就会讲到滴。

- kane 2019-08-01 09:41:56
  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。 [3赞]

- 月下独酌 2019-08-01 09:37:28
  消息需要入库可以靠唯一索引或主键约束，判断为重复的数据无法插入 [3赞]

- ly 2019-08-02 08:22:35
  老师，我有几个理解：
  当produer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。 [2赞]

作者回复2019-08-02 09:30:45
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。

- TH 2019-08-01 19:49:07
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？ [2赞]

作者回复2019-08-02 09:20:58
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。

- DC 2019-08-01 16:21:49
对于重复消息风险的处理代码，必须做好幂等。
有一种场景，消息发出后因为网络问题没有得到响应，此时服务挂掉，也无法重新发起消息，这种情况这个消息算丢失了吧。
思路是在发消息前需要记录消息发送记录，发送完成后标记完成，重启服务后查看发送消息，确无响应的消息，进行重发。不知道我提到的场景是否有问题 [2赞]

作者回复2019-08-02 09:12:19
且听下回分解。

- Better me 2019-08-01 14:16:37
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 [2赞]

作者回复2019-08-02 09:06:56
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？

- 芥末小龙 2019-08-01 09:59:33
  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， [2赞]

  作者回复2019-08-01 11:18:46
  你这结论都是用无数bug换来的呀。

- skyun 2019-08-02 20:01:30
  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 [1赞]

- 游弋云端 2019-08-01 19:22:07
  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。 [1赞]

  作者回复2019-08-02 09:17:03
  思路是没问题的。

- 敬艺 2019-08-01 16:17:24
  一个队列对应多个消费实例的话该如何保证顺序性检查？还是使用reidis 缓存起来，每个实例都去get出来判断？ [1赞]

  作者回复2019-08-02 09:11:51
  在消费端，即使同一个消费组里面有多个实例，只要你的消费代码是按照我们这节课中讲的：“先处理消费业务逻辑，再提交消费成功确认”，就可以保证消费顺序，你可以想一下为什么。

- sun留白 2019-08-01 14:09:48
  依托消息防丢失做的序号，在消费者处理时，先检查序号是否在数据库存在，若存在直接返回。 [1赞]

  作者回复2019-08-02 09:05:54
  也是一种解决思路。

- 许童童 2019-08-01 11:25:07
  消费时，做好幂等性即可。老师可以具体讲一下怎么做幂等性吗？ [1赞]

  作者回复2019-08-02 08:53:19
  且听下回分解。

- Geek_e7834d 2019-08-01 11:22:22
  broker出现重复消息无所谓，最终是consumer来处理。使用Kafka之类的消息队列， 很大原因是速度够快。 所以去重的处理需要速度很快。否则会严重拉低性能。业务逻辑有去重最好。 如果没有。 对于Kafka而言，按照一个consumer一个分区。重复可能出现在一个consumer端, 而可能是重复消息分布在不同的consumer。 对于一个consumer收到重复消息， 有唯一ID容易判断，小于当前ID的可以丢弃（用crc之类的感觉不可行， 重复消息有可能和当前消息隔的很远， 那样要保留多个消息）。 如果不同consumer收到了重复消息， 感觉去重的成本挺高的。要把一段时间内消费过的都记录下来，然后查询。 不知道怎么能以最小的代价来去重。 [1赞]

  作者回复2019-08-02 08:53:09
  且听下回分解，哈。

- Geek_e7834d 2019-08-01 11:00:27
  1. 不丢消息是以系统的performance下降为前提的， Kafka中的至少投递两个broker模式打开后， 会比一个broker确认慢不少。事务对参数似乎做了更多的限制
  2. max.in.flight.requests.per.connection 不等于1就有可能导致失败后重发的无序性。等于1性能又慢了不少。还有别的参数会导致一个partition里的消息也可能不是严格按顺序的吗？
  3. 后面是会有参数调优结合这些需求来进行的主题吗？ [1赞]

  作者回复2019-08-02 08:49:59
  我们的课程还是以讲实现原理为主，没法面面俱到的来讲每一种消息队列的各种配置项，只要掌握了原理，仔细看一下官方文档的配置说明，很容易就知道该怎么配置了。

- linqw 2019-08-01 10:33:06
  1、老师有个疑问检测消息丢失是在还没上线之前做的测试，但是会不会可能在线下没出现消息不一致，但是在线上的时候出现消息丢失了？线上
  检测消息丢失逻辑会关闭，那线上是会有其他的检测机制么？ [1赞]

  作者回复2019-08-01 11:24:47
  这个检测逻辑可以在线上做，不会影响业务的。

- whhbbq 2019-08-01 10:02:33
  请教下，在生产阶段，你需要捕获消息发送的错误，并重发消息。
  那是在catch块里，再次调用发送消息的接口吧？如下
  try {
  RecordMetadata metadata = producer.send(record).get();
  System.out.println(" 消息发送成功。");
  } catch (Throwable e) {
  producer.send(record).get();// 再次发送
  System.out.println(" 消息发送失败！");
  System.out.println(e);
  }
  [1赞]

  作者回复2019-08-01 11:23:30
  发送失败后如何处理需要看业务逻辑，当然主动重试也是一种方式。

- 撒旦的堕落 2019-08-01 09:38:19
  对于幂等 我们项目中有一个 学生报名学习课程 的业务在报名成功后 会往队列中发送消息 消费者接受到消息会进行分配作业 首先我们会往缓存中写入业务的唯一标识 然后进行业务处理 业务处理成功后 发送确认 如果业务处理失败 则删除缓存 当有消息来的时候 我们查询缓存数据库 判断业务是否已经做过 没有 则执行上面流程 有就直接确认消息 [1赞]

- Alexdown 2019-08-01 09:31:15
  『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』是否应该改为『有些消息队列的Producer在长时间没收到消息队列发送的确认响应后，....』 [1赞]

  作者回复2019-08-01 11:16:29
  就是这个意思，同学你这个表述非常清楚明白。