**MQ消息消费重复**

**5.如何保证消息不被重复消费**

分析：这个问题其实换一种问法就是，如何保证消息队列的幂等性？这个问题可以认为是消息队列领域的基本问题。换句话来说，是在考察你的设计能力，这个问题的回答可以根据具体的业务场景来答，没有固定的答案。

回答：先来说一下为什么会造成重复消费？  
其实无论是哪种消息队列，造成重复消费原因其实都是类似的。正常情况下，消费者在消费消息的时候，消费完毕后，会发送一个确认消息给消息队列，消息队列就知道该消息被消费了，就会将该消息从消息队列中删除。只是不同的消息队列发出的确认消息形式不同，例如RabbitMQ是发送一个ACK确认消息，RocketMQ是返回一个CONSUME\_SUCCESS成功标志，kafka实际上有个offet的概念，简单说一下，就是每一个消息都有一个offset，kafka消费过消息后，需要提交offset，让消息队列知道自己已经消费过了。

**那造成重复消费的原因？**，就是因为网络传输等等故障，确认信息没有传送到消息队列，导致消息队列不知道自己已经消费过该消息了，再次将消息分发给其他的消费者。

如何解决？这个问题针对业务场景来答，分以下三种情况：

（1）比如，你拿到这个消息做数据库的insert操作，那就容易了，给这个消息做一个唯一的主键，那么就算出现重复消费的情况，就会导致主键冲突，避免数据库出现脏数据。

（2）再比如，你拿到这个消息做redis的set的操作，那就容易了，不用解决，因为你无论set几次结果都是一样的，set操作本来就算幂等操作。

（3）如果上面两种情况还不行，上大招。准备一个第三方介质，来做消费记录。**以redis为例，给消息分配一个全局id，只要消费过该消息，将<id,message>以K-V形式写入redis.那消费者开始消费前，先去redis中查询有没有消费记录即可。**

**怎么保证消息队列消费的幂等性：**

1，比如数据写库，可以先根据主键查一下，如果这数据都有了，就update

2，比如写redis，那没问题，因为每次都是set，天然幂等性

3，如果不是上面两个场景，那做的稍微复杂一点，需要让生产者发送每条数据的时候，里面加一个全局唯一的id，类似订单id之类的东西，然后消费到了后，先根据这个id去比如redis里查一下，之前消费过吗？如果没有消费过，就处理，然后这个id写redis。如果消费过了，那就别处理了，保证别重复处理相同的消息即可。

4，还有比如基于数据库的唯一键来保证重复数据不会重复插入多条，重复数据拿到了以后我们插入的时候，因为有唯一键约束了，所以重复数据只会插入报错，不会导致数据库中出现脏数据

### 三、如何保证消息的顺序性

##### 1. rabbitmq

拆分多个queue，每个queue一个consumer，就是多一些queue而已，确实是麻烦点；或者就一个queue但是对应一个consumer，然后这个consumer内部用内存队列做排队，然后分发给底层不同的worker来处理

##### 2. kafka

写入一个partition中的数据一定是有序的，生产者在写的时候 ，可以指定一个key，比如指定订单id作为key，这个订单相关数据一定会被分发到一个partition中去。消费者从partition中取出数据的时候也一定是有序的，把每个数据放入对应的一个内存队列，一个partition中有几条相关数据就用几个内存队列，消费者开启多个线程，每个线程处理一个内存队列。

# [RocketMQ各组件介绍](https://www.cnblogs.com/coding400/p/11395158.html)

* [Producer](https://www.cnblogs.com/coding400/p/11395158.html#_label0)
* [Consumer](https://www.cnblogs.com/coding400/p/11395158.html#_label1)
* [Nameserver](https://www.cnblogs.com/coding400/p/11395158.html#_label2)
* [Broker](https://www.cnblogs.com/coding400/p/11395158.html#_label3)

### Producer

消息发布者，支持分布式集群部署。Produer 通过 MQ 负载均衡模块选择相应 Broker 中的 queue 进行消息投递，投递过程支持快速失败并且低延迟

消息消费的角色，支持分布式集群方式部署。支持以push推，pull拉两种模式对消息进行消费。同时也支持集群方式和广播方式的消费，它提供实时消息订阅机制，可以满足大多数用户的需求

[回到目录](https://www.cnblogs.com/coding400/p/11395158.html#_labelTop)

### Nameserver

NameServer：NameServer是一个非常简单的Topic路由注册中心，其角色类似Dubbo中的zookeeper，支持Broker的动态注册与发现。主要包括两个功能：Broker管理，NameServer接受Broker集群的注册信息并且保存下来作为路由信息的基本数据。然后提供心跳检测机制，检查Broker是否还存活；路由信息管理，每个NameServer将保存关于Broker集群的整个路由信息和用于客户端查询的队列信息。然后Producer和Conumser通过NameServer就可以知道整个Broker集群的路由信息，从而进行消息的投递和消费。NameServer通常也是集群的方式部署，各实例间相互不进行信息通讯。Broker是向每一台NameServer注册自己的路由信息，所以每一个NameServer实例上面都保存一份完整的路由信息。当某个NameServer因某种原因下线了，Broker仍然可以向其它NameServer同步其路由信息，Producer,Consumer仍然可以动态感知Broker的路由的信息。

[回到目录](https://www.cnblogs.com/coding400/p/11395158.html#_labelTop)

### Broker

1. Broker主要负责消息的存储、投递和查询以及服务高可用保证，为了实现这些功能，Broker包含了以下几个重要子模块
2. Remoting Module：整个Broker的实体，负责处理来自clients端的请求。
3. Client Manager：负责管理客户端(Producer/Consumer)和维护Consumer的Topic订阅信息
4. Store Service：提供方便简单的API接口处理消息存储到物理硬盘和查询功能。
5. HA Service：高可用服务，提供Master Broker 和 Slave Broker之间的数据同步功能。
6. Index Service：根据特定的Message key对投递到Broker的消息进行索引服务，以提供消息的快速查询。

## RocketMQ 网络部署特点

* NameServer是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。
* Broker部署相对复杂，Broker分为Master与Slave，一个Master可以对应多个Slave，但是一个Slave只能对应一个Master，Master与Slave 的对应关系通过指定相同的BrokerName，不同的BrokerId 来定义，BrokerId为0表示Master，非0表示Slave。Master也可以部署多个。每个Broker与NameServer集群中的所有节点建立长连接，定时注册Topic信息到所有NameServer。 注意：当前RocketMQ版本在部署架构上支持一Master多Slave，但只有BrokerId=1的从服务器才会参与消息的读负载。
* Producer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic 服务的Master建立长连接，且定时向Master发送心跳。Producer完全无状态，可集群部署。
* Consumer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，消费者在向Master拉取消息时，Master服务器会根据拉取偏移量与最大偏移量的距离（判断是否读老消息，产生读I/O），以及从服务器是否可读等因素建议下一次是从Master还是Slave拉取。

结合部署架构图，描述集群工作流程：

* 启动NameServer，NameServer起来后监听端口，等待Broker、Producer、Consumer连上来，相当于一个路由控制中心。
* Broker启动，跟所有的NameServer保持长连接，定时发送心跳包。心跳包中包含当前Broker信息(IP+端口等)以及存储所有Topic信息。注册成功后，NameServer集群中就有Topic跟Broker的映射关系。
* 收发消息前，先创建Topic，创建Topic时需要指定该Topic要存储在哪些Broker上，也可以在发送消息时自动创建Topic。
* Producer发送消息，启动时先跟NameServer集群中的其中一台建立长连接，并从NameServer中获取当前发送的Topic存在哪些Broker上，轮询从队列列表中选择一个队列，然后与队列所在的Broker建立长连接从而向Broker发消息。
* Consumer跟Producer类似，跟其中一台NameServer建立长连接，获取当前订阅Topic存在哪些Broker上，然后直接跟Broker建立连接通道，开始消费消息

# RocketMQ保证消息不丢失

# 概述

**分别从Producer发送机制、Broker的持久化机制，以及消费者的offSet机制来最大程度保证消息不易丢失**

1. 从Producer的视角来看：如果消息未能正确的存储在MQ中，或者消费者未能正确的消费到这条消息，都是消息丢失。
2. 从Broker的视角来看：如果消息已经存在Broker里面了，如何保证不会丢失呢（宕机、磁盘崩溃）
3. 从Consumer的视角来看：如果消息已经完成持久化了，但是Consumer取了，但是未消费成功且没有反馈，就是消息丢失

**从Producer分析：如何确保消息正确的发送到了Broker?**

1. 默认情况下，可以通过同步的方式阻塞式的发送，check SendStatus，状态是OK，表示消息一定成功的投递到了Broker，状态超时或者失败，则会触发默认的2次重试。此方法的发送结果，可能Broker存储成功了，也可能没成功
2. 采取事务消息的投递方式，并不能保证消息100%投递成功到了Broker，但是如果消息发送Ack失败的话，此消息会存储在CommitLog当中，但是对ConsumerQueue是不可见的。可以在日志中查看到这条异常的消息，严格意义上来讲，也并没有完全丢失
3. RocketMQ支持 日志的索引，如果一条消息发送之后超时，也可以通过查询日志的API，来check是否在Broker存储成功

**从Broker分析：如果确保接收到的消息不会丢失?**

1. 消息支持持久化到Commitlog里面，即使宕机后重启，未消费的消息也是可以加载出来的
2. Broker自身支持同步刷盘、异步刷盘的策略，可以保证接收到的消息一定存储在本地的内存中
3. Broker集群支持 1主N从的策略，支持同步复制和异步复制的方式，同步复制可以保证即使Master 磁盘崩溃，消息仍然不会丢失

**从Cunmser分析：如何确保拉取到的消息被成功消费？**

1. 消费者可以根据自身的策略批量Pull消息
2. Consumer自身维护一个持久化的offset（对应MessageQueue里面的min offset），标记已经成功消费或者已经成功发回到broker的消息下标
3. 如果Consumer消费失败，那么它会把这个消息发回给Broker，发回成功后，再更新自己的offset
4. 如果Consumer消费失败，发回给broker时，broker挂掉了，那么Consumer会定时重试这个操作
5. 如果Consumer和broker一起挂了，消息也不会丢失，因为consumer 里面的offset是定时持久化的，重启之后，继续拉取offset之前的消息到本地