《我想进大厂》之MQ夺命连环11问

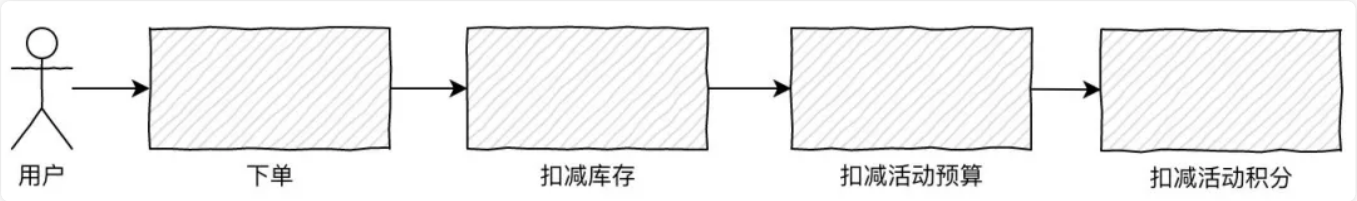
原创 科技缪缪 艾小仙 2020-09-29

收录于话题
#面试大全 29 #文章精选汇总 78

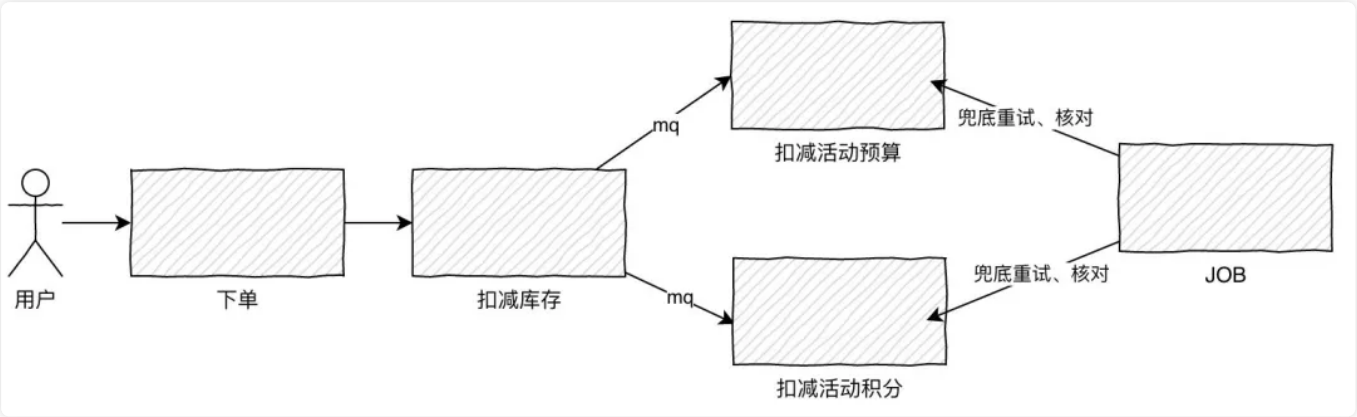
继之前的mysql夺命连环之后，我发现我这个标题被好多套用的，什么夺命zookeeper，夺命多线程一大堆，这一次，开始面试题系列MQ专题，消息队列作为日常常见的使用中间件，面试也是必问的点之一，一起来看看MQ的面试题。

你们为什么使用mq？具体的使用场景是什么？

mq的作用很简单，削峰填谷。以电商交易下单的场景来说，正向交易的过程可能涉及到创建订单、扣减库存、扣减活动预算、扣减积分等等。每个接口的耗时如果是100ms，那么理论上整个下单的链路就需要耗费400ms，这个时间显然是太长了。



如果这些操作全部同步处理的话，首先调用链路太长影响接口性能，其次分布式事务的问题很难处理，这时候像扣减预算和积分这种对实时一致性要求没有那么高的请求，完全就可以通过mq异步的方式去处理了。同时，考虑到异步带来的不一致的问题，我们可以通过job去重试保证接口调用成功，而且一般公司都会有核对的平台，比如下单成功但是未扣减积分的这种问题可以通过核对作为兜底的处理方案。



使用mq之后我们的链路变简单了，同时异步发送消息我们的整个系统的抗压能力也上升了。

那你们使用什么mq？基于什么做的选型？

我们主要调研了几个主流的mq，kafka、rabbitmq、rocketmq、activemq，选型我们主要基于以下几个点去考虑：

1. 由于我们系统的qps压力比较大，所以性能是首要考虑的要素。
2. 开发语言，由于我们的开发语言是java，主要是为了方便二次开发。
3. 对于高并发的业务场景是必须的，所以需要支持分布式架构的设计。
4. 功能全面，由于不同的业务场景，可能会用到顺序消息、事务消息等。

基于以上几个考虑，我们最终选择了RocketMQ。

	Kafka	**RocketMQ**	**RabbitMQ**	**ActiveMQ**
单机吞吐量	10万级	10万级	万级	万级
开发语言	Scala	Java	Erlang	Java
高可用	分布式架构	分布式架构	主从架构	主从架构
性能	ms级	ms级	us级	ms级
功能	只支持主要的MQ功能	顺序消息、事务消息等功能完善	并发强、性能好、延时低	成熟的社区产品、文档丰富

你上面提到异步发送，那消息可靠性怎么保证？

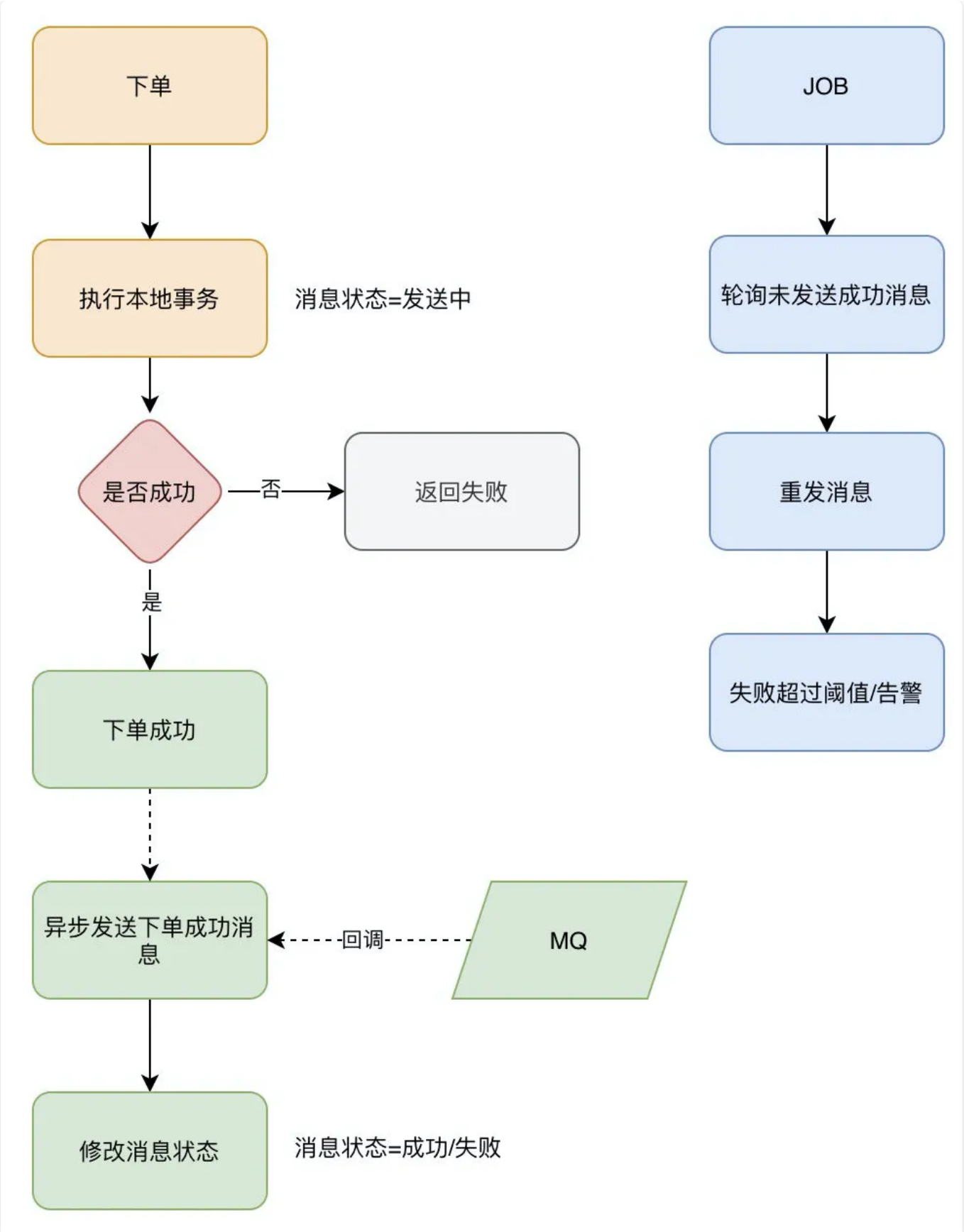
消息丢失可能发生在生产者发送消息、MQ本身丢失消息、消费者丢失消息3个方面。

生产者丢失消息的可能点在于程序发送失败抛异常了没有重试处理，或者发送的过程成功但是过程中网络闪断MQ没收到，消息就丢失了。

由于同步发送的一般不会出现这样使用方式，所以我们就不考虑同步发送的问题，我们基于异步发送的场景来说。

异步发送分为两个方式：**异步有回调和异步无回调**，无回调的方式，生产者发送完后不管结果可能就会造成消息丢失，而通过异步发送+回调通知+本地消息表的形式我们就可以做出一个解决方案。以下单的场景举例。

1. 下单后先保存本地数据和MQ消息表，这时候消息的状态是发送中，如果本地事务失败，那么下单失败，事务回滚。
2. 下单成功，直接返回客户端成功，异步发送MQ消息
3. MQ回调通知消息发送结果，对应更新数据库MQ发送状态
4. JOB轮询超过一定时间（时间根据业务配置）还未发送成功的消息去重试
5. 在监控平台配置或者JOB程序处理超过一定次数一直发送不成功的消息，告警，人工介入。



一般而言，对于大部分场景来说异步回调的形式就可以了，只有那种需要完全保证不能丢失消息的场景我们做一套完整的解决方案。

MQ丢失

如果生产者保证消息发送到MQ，而MQ收到消息后还在内存中，这时候宕机了又没来得及同步给从节点，就有可能导致消息丢失。

比如RocketMQ：

RocketMQ分为同步刷盘和异步刷盘两种方式，默认的是异步刷盘，就有可能导致消息还未刷到硬盘上就丢失了，可以通过设置为同步刷盘的方式来保证消息可靠性，这样即使MQ挂了，恢复的时候也可以从磁盘中去恢复消息。

比如Kafka也可以通过配置做到：

```
acks=all 只有参与复制的所有节点全部收到消息，才返回生产者成功。这样的话除非所有的节点都挂了，消息才会丢失。
replication.factor=N,设置大于1的数，这会要求每个partion至少有2个副本
min.insync.replicas=N，设置大于1的数，这会要求leader至少感知到一个follower还保持着连接
retries=N，设置一个非常大的值，让生产者发送失败一直重试
```

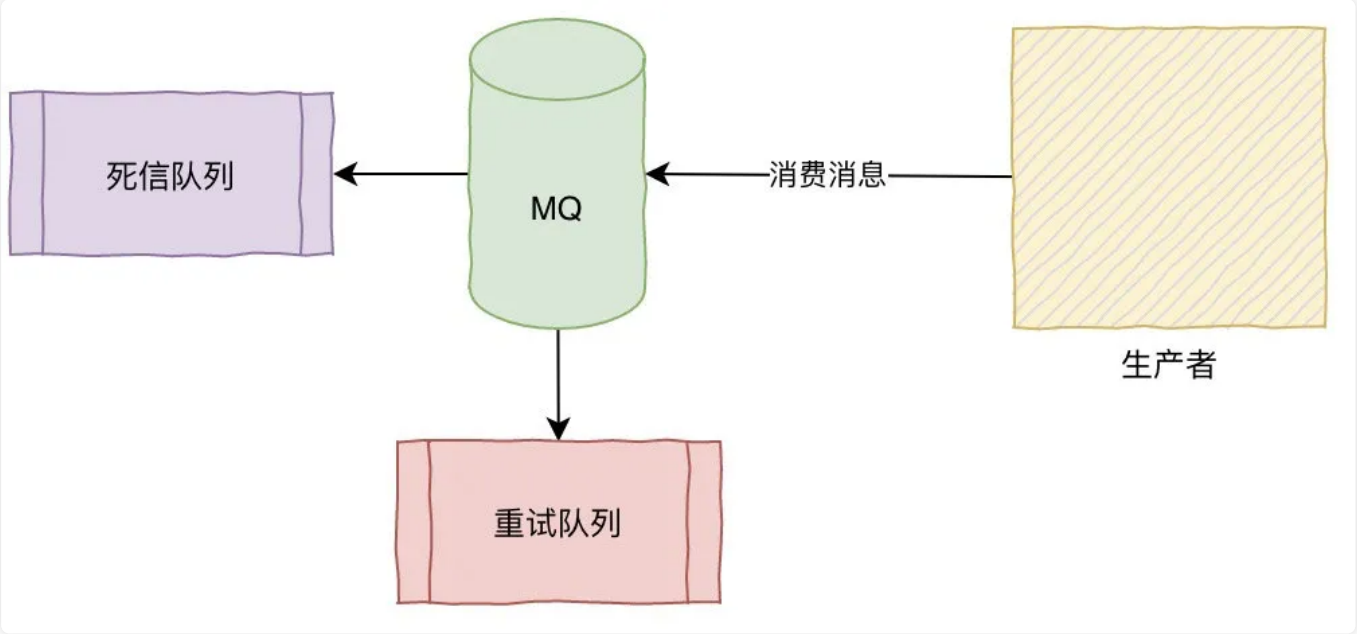
虽然我们可以通过配置的方式来达到MQ本身高可用的目的，但是都对性能有损耗，怎样配置需要根据业务做出权衡。

消费者丢失

消费者丢失消息的场景：消费者刚收到消息，此时服务器宕机，MQ认为消费者已经消费，不会重复发送消息，消息丢失。

RocketMQ默认是需要消费者回复ack确认，而kafka需要手动开启配置关闭自动offset。

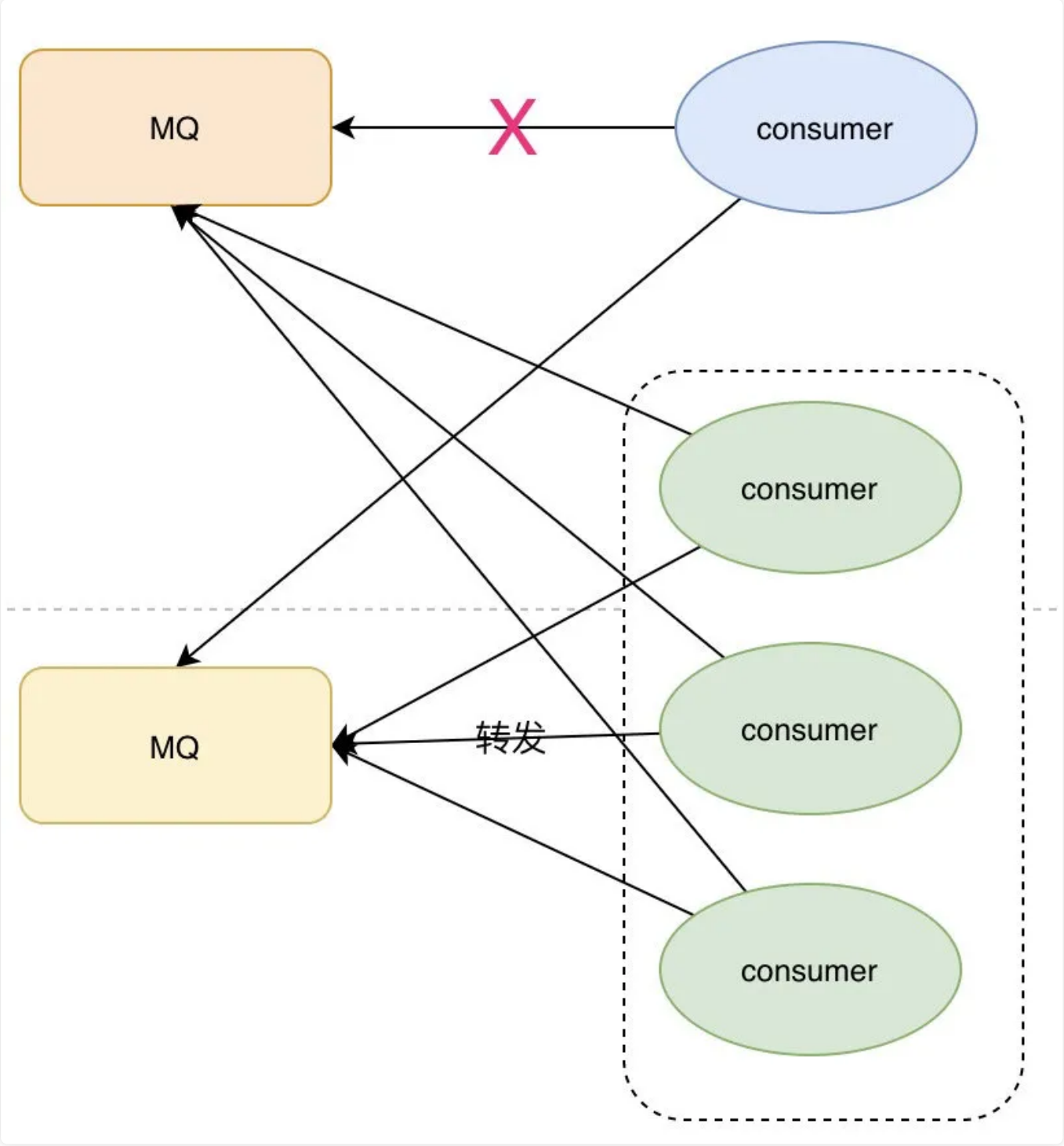
消费方不返回ack确认，重发的机制根据MQ类型的不同发送时间间隔、次数都不尽相同，如果重试超过次数之后会进入死信队列，需要手工来处理了。（Kafka没有这些）



你说到消费者消费失败的问题，那么如果一直消费失败导致消息积压怎么处理?

因为考虑到时消费者消费一直出错的问题，那么我们可以从以下几个角度来考虑：

1. 消费者出错，肯定是程序或者其他问题导致的，如果容易修复，先把问题修复，让consumer恢复正常消费
2. 如果时间来不及处理很麻烦，做转发处理，写一个临时的consumer消费方案，先把消息消费，然后再转发到一个新的topic和MQ资源，这个新的topic的机器资源单独申请，要能承载住当前积压的消息
3. 处理完积压数据后，修复consumer，去消费新的MQ和现有的MQ数据，新MQ消费完成后恢复原状



那如果消息积压达到磁盘上限，消息被删除了怎么办？

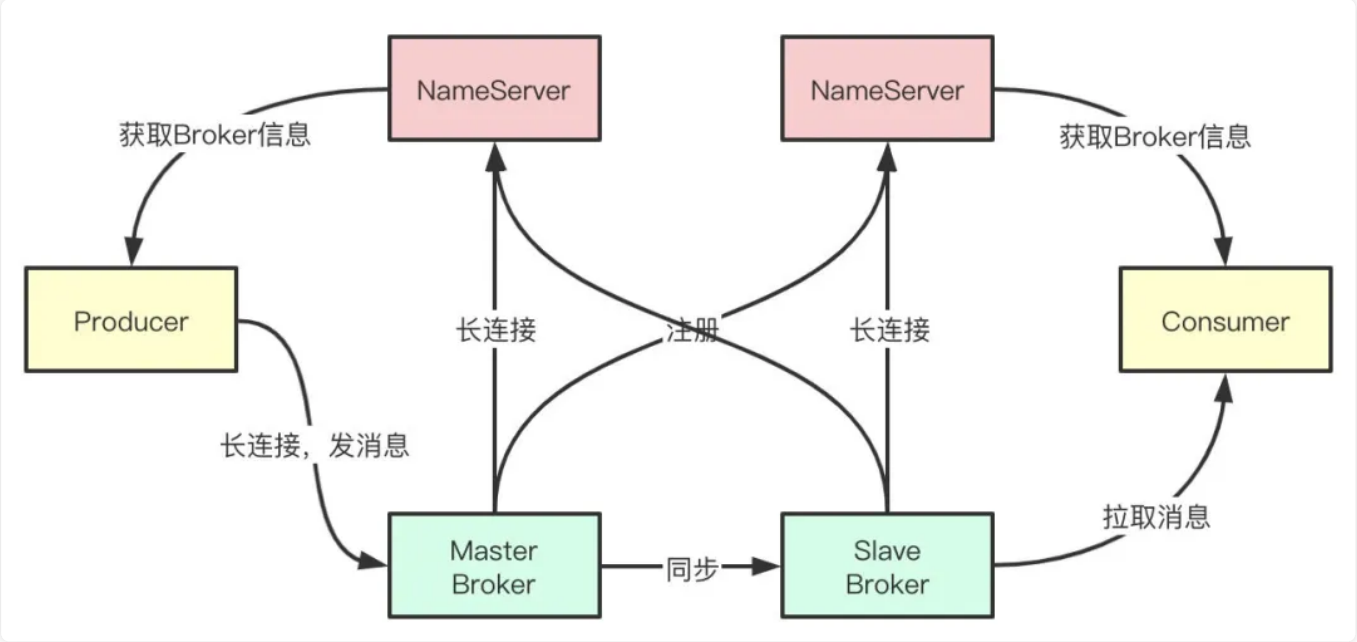
这。。。他妈都删除了我有啥办法啊。。。冷静，再想想。。有了。

最初，我们发送的消息记录是落库保存了的，而转发发送的数据也保存了，那么我们就可以通过这部分数据来找到丢失的那部分数据，再单独跑个脚本重发就可以了。如果转发的程序没有落库，那就和消费方的记录去做对比，只是过程会更艰难一点。

说了这么多，那你说说RocketMQ实现原理吧?

RocketMQ由NameServer注册中心集群、Producer生产者集群、Consumer消费者集群和若干Broker（RocketMQ进程）组成，它的架构原理是这样的：

1. Broker在启动的时候去向所有的NameServer注册，并保持长连接，每30s发送一次心跳
2. Producer在发送消息的时候从NameServer获取Broker服务器地址，根据负载均衡算法选择一台服务器来发送消息
3. Conusmer消费消息的时候同样从NameServer获取Broker地址，然后主动拉取消息来消费



为什么RocketMQ不使用Zookeeper作为注册中心呢?

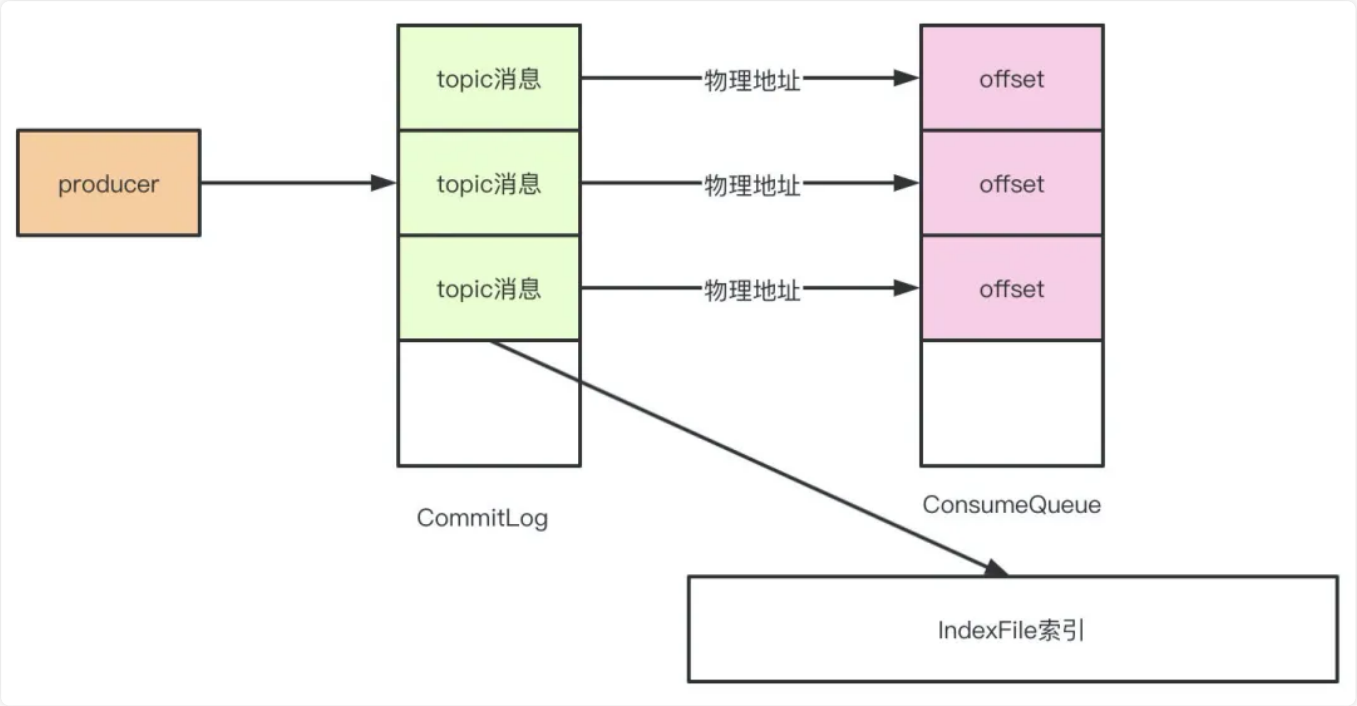
我认为有以下几个点是不使用zookeeper的原因：

1. 根据CAP理论，同时最多只能满足两个点，而zookeeper满足的是CP，也就是说zookeeper并不能保证服务的可用性，zookeeper在进行选举的时候，整个选举的时间太长，期间整个集群都处于不可用的状态，而这对于一个注册中心来说肯定是不能接受的，作为服务发现来说就应该是为可用性而设计。
2. 基于性能的考虑，NameServer本身的实现非常轻量，而且可以通过增加机器的方式水平扩展，增加集群的抗压能力，而zookeeper的写是不可扩展的，而zookeeper要解决这个问题只能通过划分领域，划分多个zookeeper集群来解决，首先操作起来太复杂，其次这样还是又违反了CAP中的A的设计，导致服务之间是不连通的。
3. 持久化的机制来带的问题，ZooKeeper 的 ZAB 协议对每一个写请求，会在每个 ZooKeeper 节点上保持写一个事务日志，同时再加上定期的将内存数据镜像（Snapshot）到磁盘来保证数据的一致性和持久性，而对于一个简单的服务发现的场景来说，这其实没有太大的必要，这个实现方案太重了。而且本身存储的数据应该是高度定制化的。
4. 消息发送应该弱依赖注册中心，而RocketMQ的设计理念也正是基于此，生产者在第一次发送消息的时候从NameServer获取到Broker地址后缓存到本地，如果NameServer整个集群不可用，短时间内对于生产者和消费者并不会产生太大影响。

那Broker是怎么保存数据的呢?

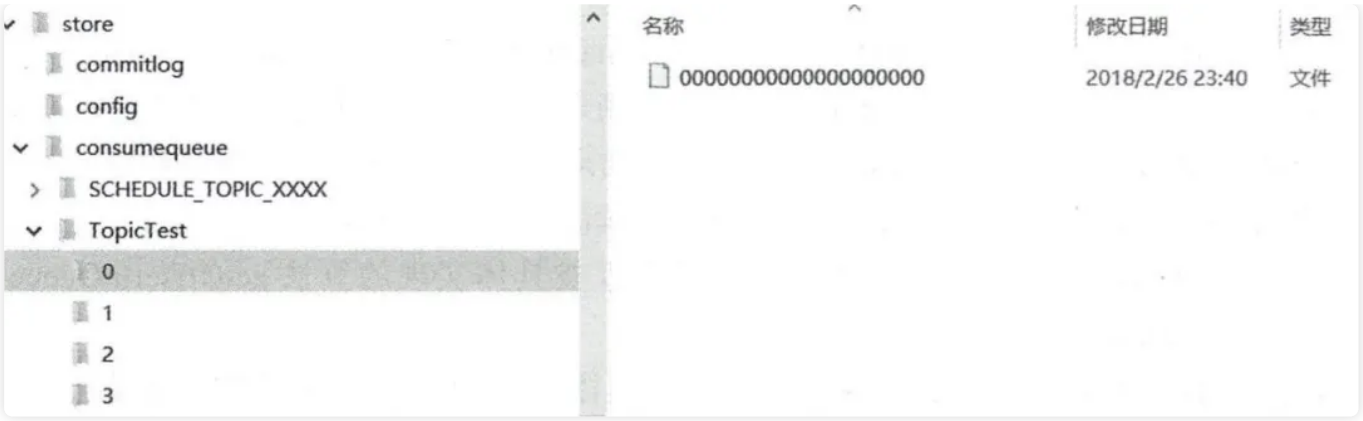
RocketMQ主要的存储文件包括commitlog文件、consumequeue文件、indexfile文件。

Broker在收到消息之后，会把消息保存到commitlog的文件当中，而同时在分布式的存储当中，每个broker都会保存一部分topic的数据，同时，每个topic对应的messagequeue下都会生成consumequeue文件用于保存commitlog的物理位置偏移量offset，indexfile中会保存key和offset的对应关系。



CommitLog文件保存于${Rocket_Home}/store/commitlog目录中，从图中我们可以明显看出来文件名的偏移量，每个文件默认1G，写满后自动生成一个新的文件。

由于同一个topic的消息并不是连续的存储在commitlog中，消费者如果直接从commitlog获取消息效率非常低，所以通过consumequeue保存commitlog中消息的偏移量的物理地址，这样消费者在消费的时候先从consumequeue中根据偏移量定位到具体的commitlog物理文件，然后根据一定的规则（offset和文件大小取模）在commitlog中快速定位。



Master和Slave之间是怎么同步数据的呢?

而消息在master和slave之间的同步是根据raft协议来进行的:

1. 在broker收到消息后，会被标记为uncommitted状态
2. 然后会把消息发送给所有的slave
3. slave在收到消息之后返回ack响应给master
4. master在收到超过半数的ack之后，把消息标记为committed
5. 发送committed消息给所有slave，slave也修改状态为committed

你知道RocketMQ为什么速度快吗?

是因为使用了顺序存储、Page Cache和异步刷盘。

1. 我们在写入commitlog的时候是顺序写入的，这样比随机写入的性能就会提高很多
2. 写入commitlog的时候并不是直接写入磁盘，而是先写入操作系统的PageCache
3. 最后由操作系统异步将缓存中的数据刷到磁盘

什么是事务、半事务消息？怎么实现的?

事务消息就是MQ提供的类似XA的分布式事务能力，通过事务消息可以达到分布式事务的最终一致性。

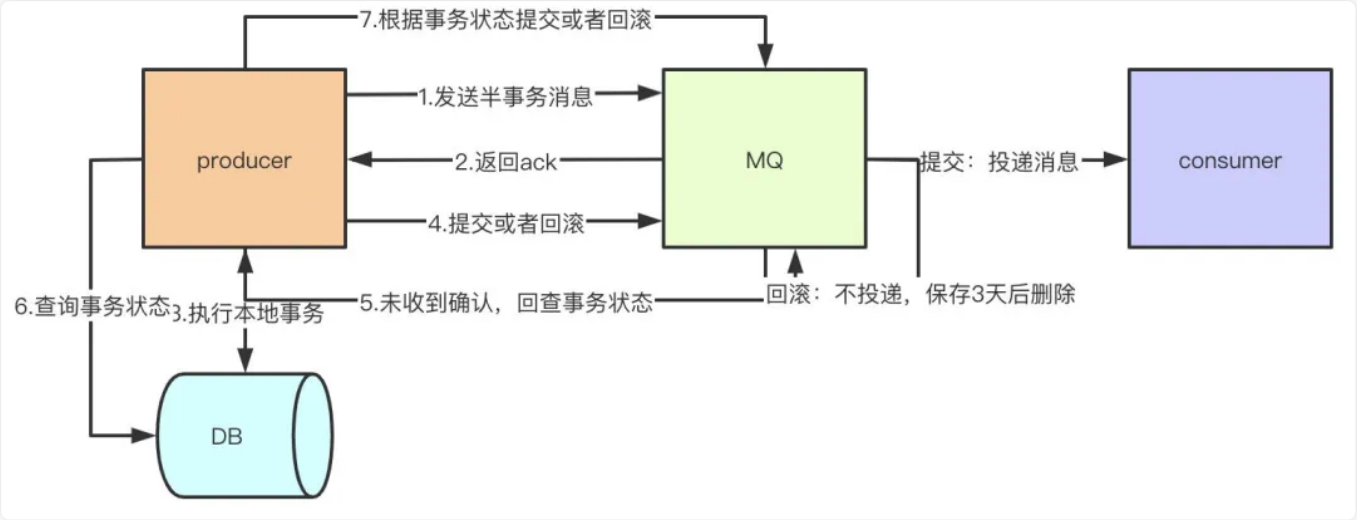
半事务消息就是MQ收到了生产者的消息，但是没有收到二次确认，不能投递的消息。

实现原理如下:

1. 生产者先发送一条半事务消息到MQ

2. MQ收到消息后返回ack确认
3. 生产者开始执行本地事务
4. 如果事务执行成功发送commit到MQ，失败发送rollback
5. 如果MQ长时间未收到生产者的二次确认commit或者rollback，MQ对生产者发起消息回查
6. 生产者查询事务执行最终状态
7. 根据查询事务状态再次提交二次确认

最终，如果MQ收到二次确认commit，就可以把消息投递给消费者，反之如果是rollback，消息会保存下来并且在3天后被删除。



- END -

往期推荐

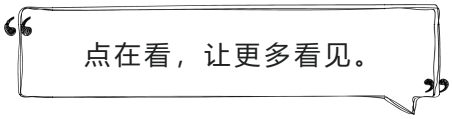
《我想进大厂》之mysql夺命连环13问

再深入一点|binlog和relay-log到底长啥样？

面试官：哪些场景会产生OOM？怎么解决？

构造函数没有返回值是怎么赋值的？

长按二维码识别关注



收录于话题 #面试大全·29个

上一篇

面试官：说说CountDownLatch，CyclicBarrier，Semaphore的原理?

下一篇

《我想进大厂》之Dubbo普普通通9问

喜欢此内容的人还喜欢

我用kafka两年踩过的一些非比寻常的坑

苏三说技术

Redis缓存那点破事 | 绝杀面试官 25 问!

微观技术

大厂动态规划面试汇总，重量级干货，彻夜整理

盼盼编程