**消息队列**

目录

[消息队列 1](#_Toc41847471)

[1. 消息队列的7个连环炮​ 1](#_Toc41847472)

[2. 你为什么使用消息队列 2](#_Toc41847473)

[2.1 核心原因一：解耦 2](#_Toc41847474)

[2.2 核心原因二：异步 3](#_Toc41847475)

[2.3 核心原因三：削峰 3](#_Toc41847476)

[3. 消息队列有什么优点和缺点 4](#_Toc41847477)

[3.1 优点 4](#_Toc41847478)

[3.2 缺点 5](#_Toc41847479)

[4. kafka、activemq、rabbitmq、rocketmq都有什么优点和缺点 6](#_Toc41847480)

[5. 如何保证消息队列的高可用啊 7](#_Toc41847481)

[5.1 单机模式 7](#_Toc41847482)

[5.2 普通集群模式 7](#_Toc41847483)

[5.3 镜像集群模式 8](#_Toc41847484)

[6 如何保证消息不被重复消费（幂等性） 9](#_Toc41847485)

[7 如何保证消息的可靠性传输（不丢失） 10](#_Toc41847486)

[7.1 生产者弄丢了数据 10](#_Toc41847487)

[7.2 rabbitmq弄丢了数据 10](#_Toc41847488)

[7.3 消费端弄丢了数据 11](#_Toc41847489)

[8 如何保证消息的顺序性 12](#_Toc41847490)

# 消息队列

## 1. 消息队列的7个连环炮​

1. 那你们为什么使用消息队列啊？

2. 那你说说用消息队列都有什么优点和缺点？

3. kafka、activemq、rabbitmq、rocketmq都有什么区别？

4. 那你们是如何保证消息队列的高可用啊？

5. 如何保证消息不被重复消费啊？如何保证消费的时候是幂等的啊？

6. 如何保证消息的可靠性传输啊？要是消息丢失了怎么办啊？

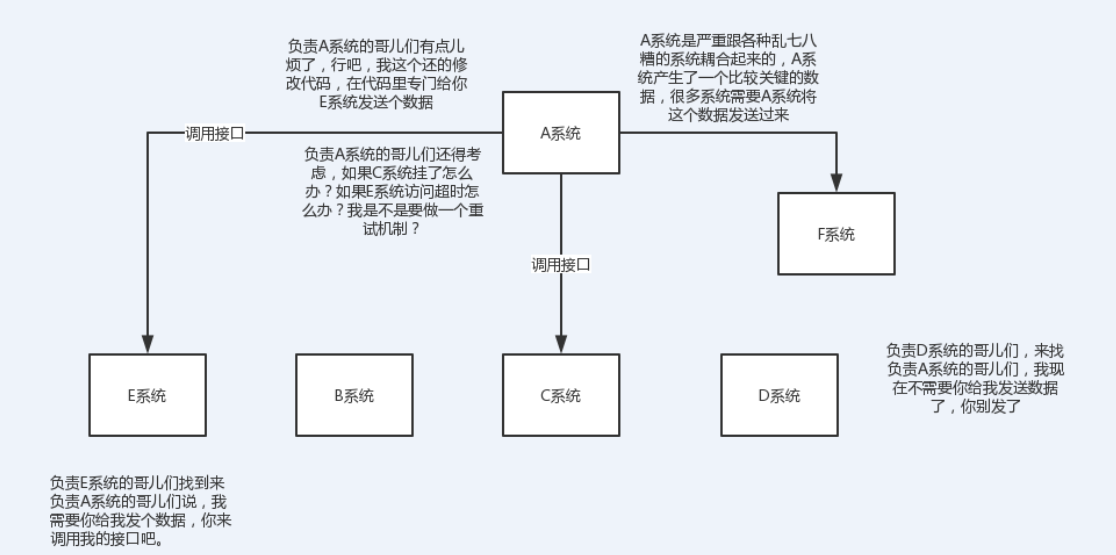
7. 那如何保证消息的顺序性？

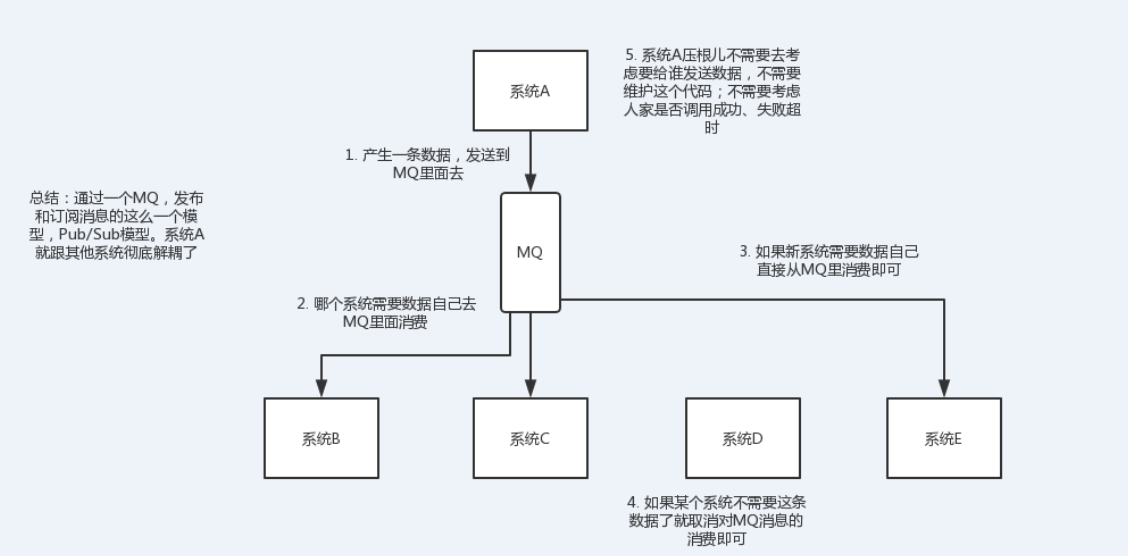
## 2. 你为什么使用消息队列

存在某些业务场景，不用MQ可能会很麻烦，用了MQ之后带给了你很多的好处。消息队列使用场景有很多，但是比较核心的有3个：**解耦、异步、削峰**。

### 2.1 核心原因一：解耦

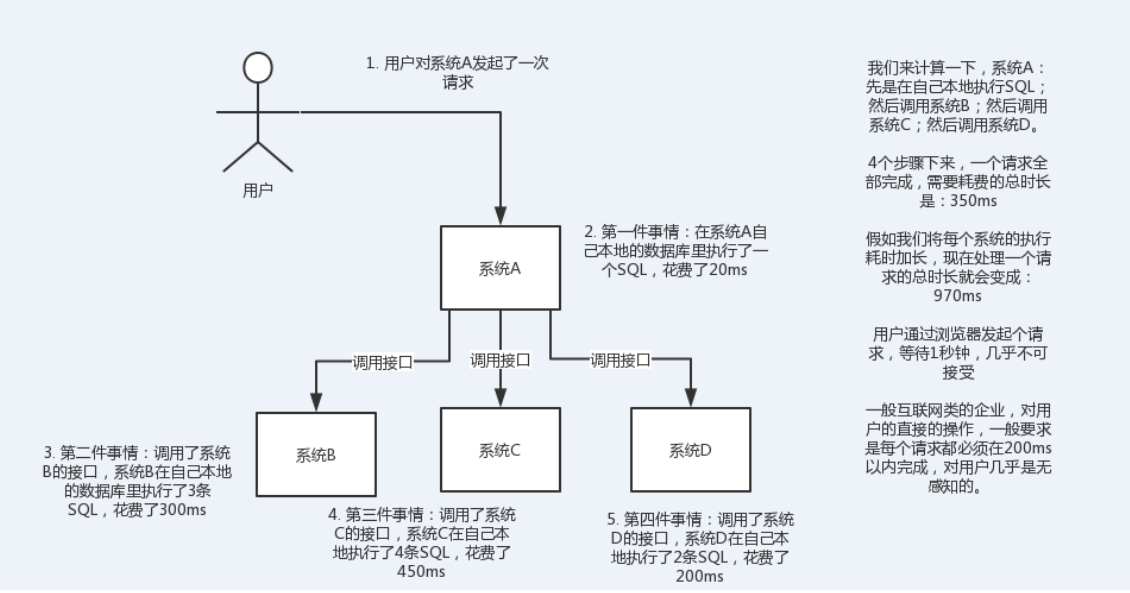
场景：如果一个系统或者一个模块，调用了多个系统或者模块，存在一对多的关系，互相之间的调用很复杂，维护起来很麻烦。但是其实这个调用是不需要直接同步调用接口的，此时可以运用MQ去进行系统的解耦，如下图所示：

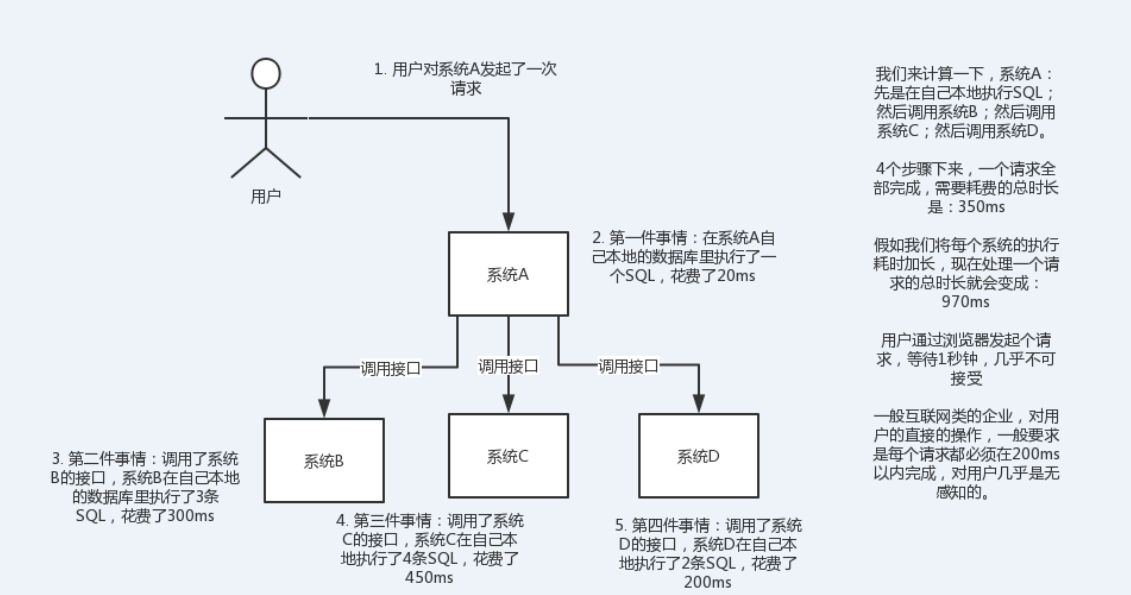




### 2.2 核心原因二：异步

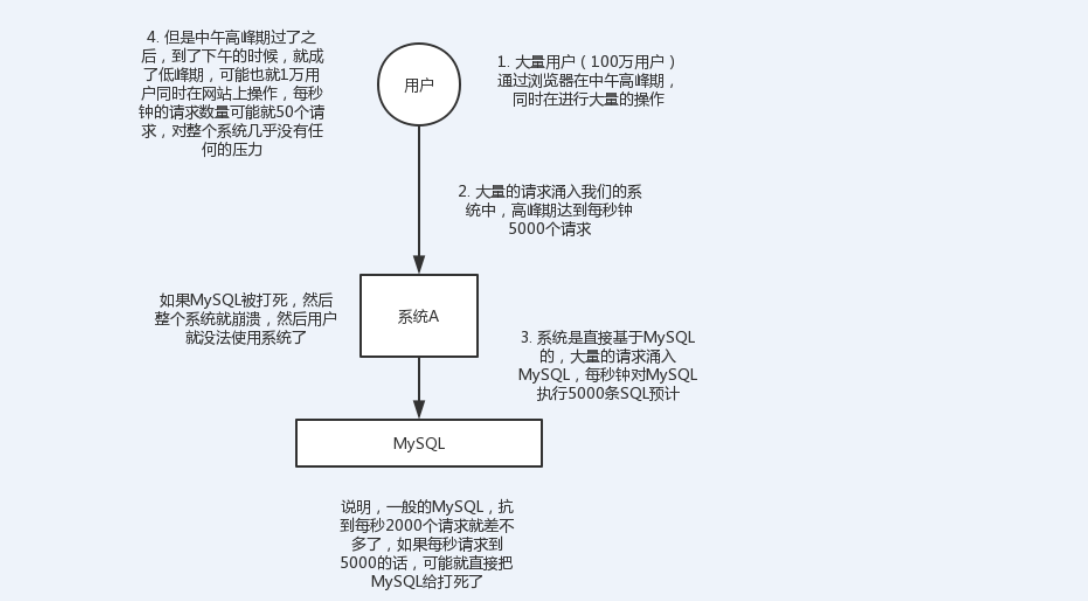
场景： A系统接收一个请求，需要在自己本地写库，还需要在BCD三个系统写库，自己本地写库要3ms，BCD三个系统分别写库要300ms、450ms、200ms。最终请求总延时是3 + 300 + 450 + 200 = 953ms，接近1s，用户的感觉是慢死了**。**

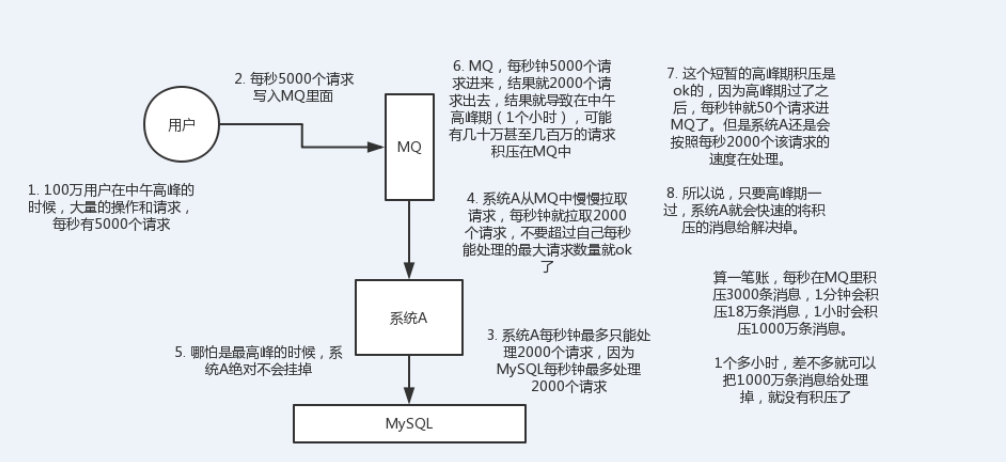
****

****

### 2.3 核心原因三：削峰

场景：每天0点到11点，A系统风平浪静，每秒并发请求数量就100个。结果每次一到11点~1点，每秒并发请求数量突然会暴增到1万条。但是系统最大的处理能力就只能是每秒钟处理1000个请求啊，此时就可以使用MQ来削峰。





## 3. 消息队列有什么优点和缺点

### 3.1 优点

在特殊场景下有其对应的好处，解耦、异步、削峰。

### 3.2 缺点

#### 3.2.1 系统可用性降低

系统引入的外部依赖越多，越容易挂掉，本来你就是A系统调用BCD三个系统的接口就好了，人ABCD四个系统好好的，没啥问题，你偏加个MQ进来，万一MQ挂了咋整？MQ挂了，整套系统崩溃了。

#### 3.2.2 系统复杂性提高

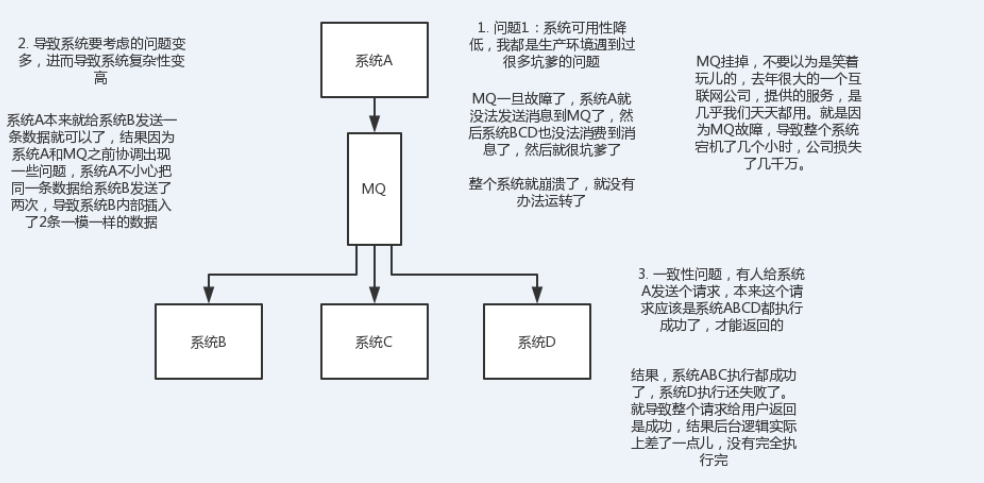
硬生生加个MQ进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？问题一大堆，需要处理

#### 3.2.3 一致性问题

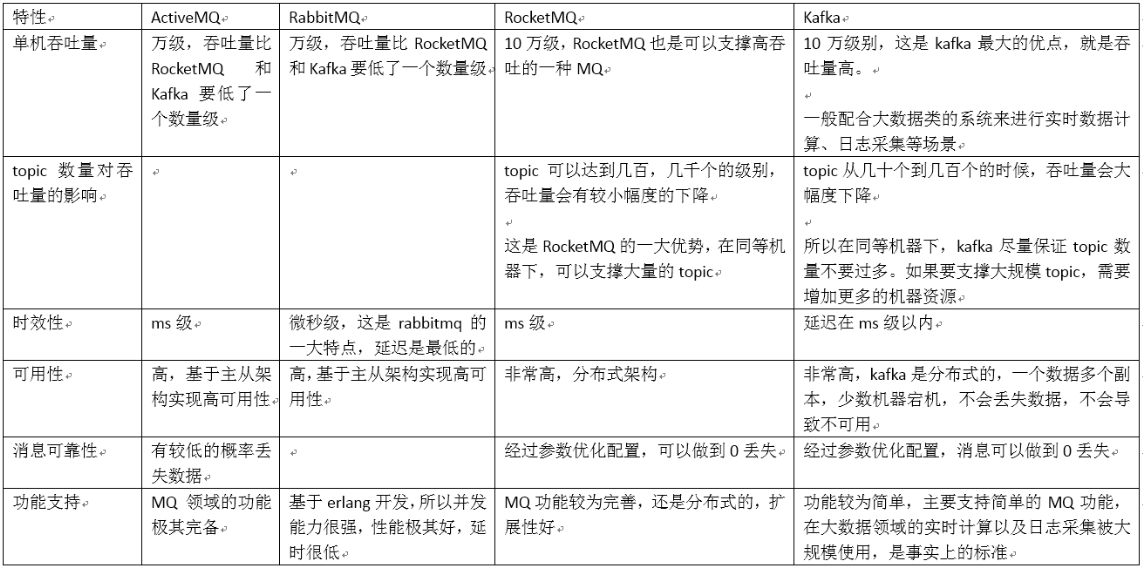
A系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是BCD三个系统那里，BD两个系统写库成功了，结果C系统写库失败了，咋整？你这数据就不一致了。

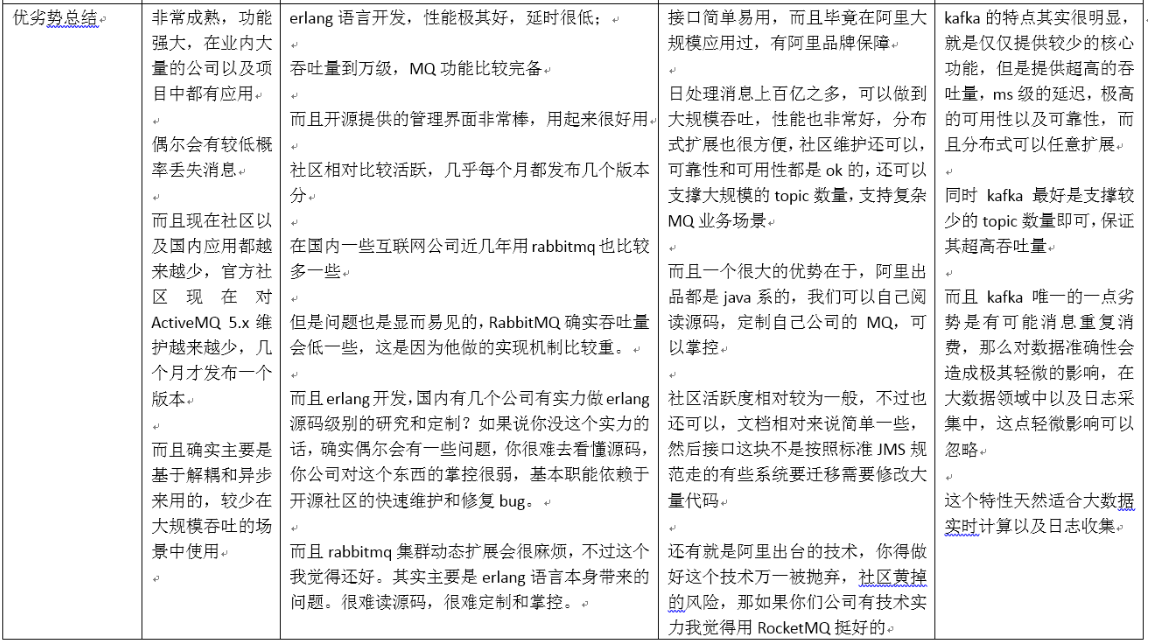
总结：

所以消息队列实际是一种非常复杂的架构，你引入它有很多好处，但是也得针对它带来的坏处做各种额外的技术方案和架构来规避掉，最后可能导致系统复杂度提升了，但是关键时刻该用还是得用的。



## 4. kafka、activemq、rabbitmq、rocketmq都有什么优点和缺点





综上所述：

最早大家都用ActiveMQ，但是现在互联网公司确实用的不太多了，大规模吞吐量场景不大适应，社区也不是很活跃，出了bug问题很难办；

后来大家开始用RabbitMQ，但是确实erlang语言阻止了大量的java工程师去深入研究和掌控他，对公司而言，几乎处于不可控的状态，但是确实人是开源的，比较稳定的支持，活跃度也高；

不过现在确实越来越多的公司，会去用RocketMQ，确实很不错，但是我提醒一下自己想好社区万一突然黄掉的风险，对自己公司技术实力有绝对自信的，我推荐用RocketMQ，否则回去老老实实用RabbitMQ吧，人是活跃开源社区，绝对不会黄，所以中小型公司，技术实力较为一般，技术挑战不是特别高，用RabbitMQ是不错的选择；大型公司，基础架构研发实力较强，用RocketMQ是很好的选择

如果是大数据领域的，实时计算、日志采集等场景，用Kafka是业内标准的，绝对没问题，社区活跃度很高，绝对不会黄，何况几乎是全世界这个领域的事实性规范

## 5. 如何保证消息队列的高可用啊

非常大的互联网公司，非常核心的系统，如果没考虑MQ如何保证高可用，如果MQ挂了怎么办，导致几个小时系统不可用，公司损失几千万是有可能的。

RabbitMQ是比较有代表性的，以此讨论MQ的高可用性怎么实现。rabbitmq有三种模式：单机模式，普通集群模式，镜像集群模式。

### 5.1 单机模式

就是demo级别的，一般就是你本地启动了玩玩儿的，没人生产用单机模式。

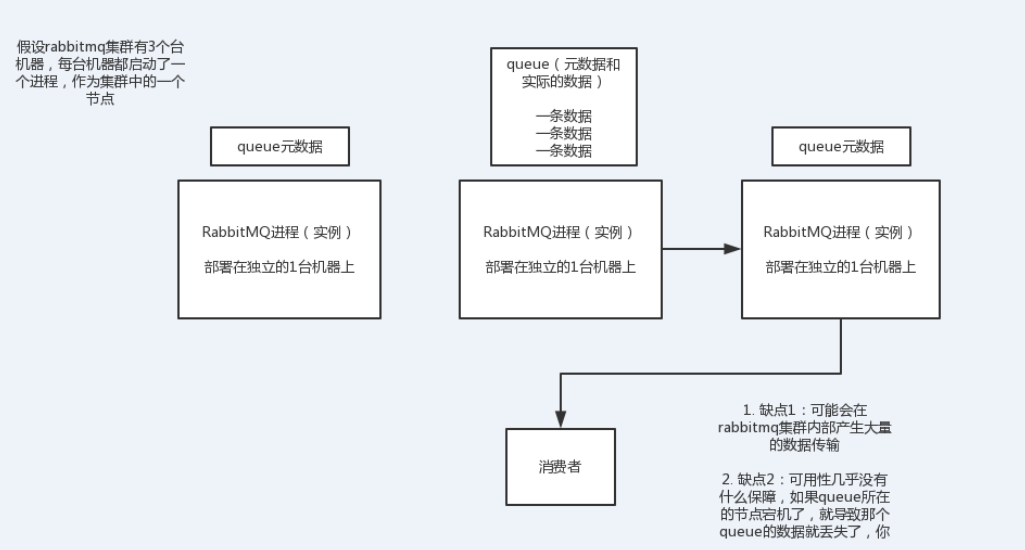
### 5.2 普通集群模式

意思就是在多台机器上启动多个rabbitmq实例，每个机器启动一个。但是你创建的queue，只会放在一个rabbtimq实例上，但是每个实例都同步queue的元数据。实际上如果连接到了另外一个实例，那么那个实例会从queue所在实例上拉取数据过来。

这种方式确实很麻烦，也不怎么好，没做到所谓的分布式，就是个普通集群。因为这导致你要么消费者每次随机连接一个实例然后拉取数据，要么固定连接那个queue所在实例消费数据，前者有数据拉取的开销，后者导致单实例性能瓶颈。

如果那个放queue的实例宕机了，会导致接下来其他实例就无法从那个实例拉取，如果你开启了消息持久化，让rabbitmq落地存储消息的话，消息不一定会丢，得等这个实例恢复了，然后才可以继续从这个queue拉取数据。

所以这个事儿就比较尴尬了，这就没有什么所谓的高可用性可言了，这方案主要是提高吞吐量的，就是说让集群中多个节点来服务某个queue的读写操作。



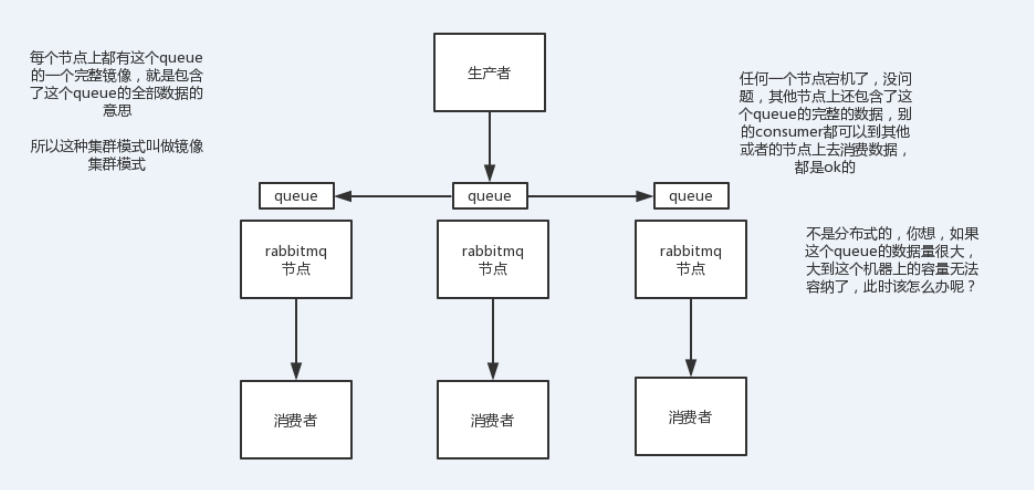
### 5.3 镜像集群模式

镜像（Mirroring）是冗余的一种类型，一个磁盘上的数据在另一个磁盘上存在一个完全相同的副本即为镜像。

这种模式，才是所谓的rabbitmq的高可用模式，跟普通集群模式不一样的是，你创建的queue，无论元数据还是queue里的消息都会存在于多个实例上，然后每次你写消息到queue的时候，都会自动把消息到多个实例的queue里进行消息同步。

这样的话，好处在于你任何一个机器宕机了，没事儿，别的机器都可以用。坏处在于，第一，这个性能开销也太大了吧，消息同步所有机器，导致网络带宽压力和消耗很重！第二，这么玩儿，就没有扩展性可言了，如果某个queue负载很重，你加机器，新增的机器也包含了这个queue的所有数据，并没有办法线性扩展你的queue。

那么怎么开启这个镜像集群模式呢？其实很简单rabbitmq有很好的管理控制台，就是在后台新增一个策略，这个策略是镜像集群模式的策略，指定的时候可以要求数据同步到所有节点的，也可以要求就同步到指定数量的节点，然后你再次创建queue的时候，应用这个策略，就会自动将数据同步到其他的节点上去了。



## 6 如何保证消息不被重复消费（幂等性）

首先就是比如rabbitmq、rocketmq、kafka，都有可能会出现消费重复消费的问题。因为这问题通常不是mq自己保证的，是要给我们保证的。

其实重复消费不可怕，可怕的是你没考虑到重复消费之后，怎么保证幂等性。

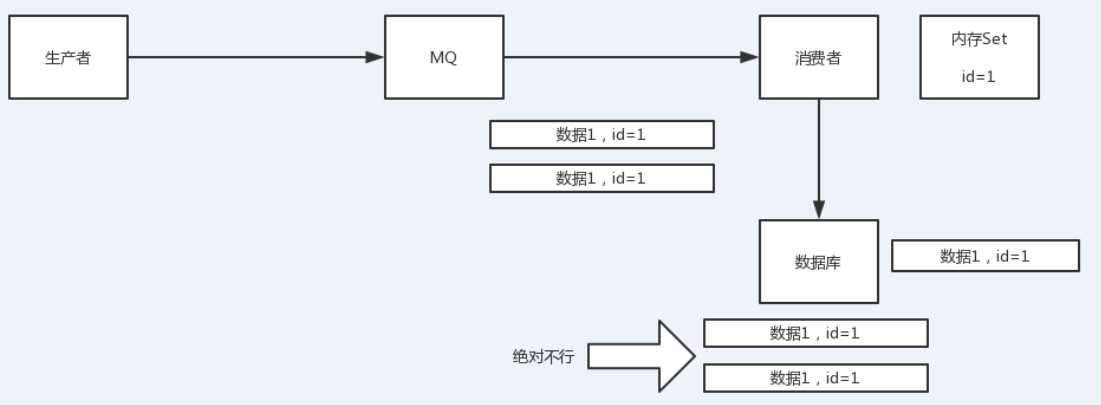
举个例子吧。假设你有个系统，消费一条往数据库里插入一条，要是你一个消息重复两次，你不就插入了两条，这数据不就错了？但是你要是消费到第二次的时候，自己判断一下已经消费过了，直接扔了，不就保留了一条数据？一条数据重复出现两次，数据库里就只有一条数据，这就保证了系统的幂等性。

**幂等性，**我通俗点说，就一个数据，或者一个请求，给你重复来多次，你得确保对应的数据是不会改变的，不能出错**。**

那么怎么保证消息队列消费的幂等性？，其实还是得结合业务来思考，我这里给几个思路：

1. 比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update一下好吧。
2. 比如你是写redis，那没问题了，反正每次都是set，天然幂等性
3. 比如你不是上面两个场景，那做的稍微复杂一点，你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，然后你这里消费到了之后，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，你就处理，然后这个id写redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。

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。



## 7 如何保证消息的可靠性传输（不丢失）

mq有个基本原则，就是数据不能多一条，也不能少一条，不能多，就是刚才说的重复消费和幂等性问题。不能少，就是说这数据别搞丢了。

这个丢数据，要么是mq自己弄丢了，要么是我们生产和消费的时候弄丢了。如下以rabbitmq为例子：

### 7.1 生产者弄丢了数据

生产者将数据发送到rabbitmq的时候，可能数据就在半路给搞丢了，因为网络啥的问题，都有可能。

此时可以选择用rabbitmq提供的事务功能，就是生产者发送数据之前开启rabbitmq事务（channel.txSelect），然后发送消息，**如果消息没有成功被rabbitmq接收到**，那么生产者会收到异常报错，此时就可以回滚事务（channel.txRollback），然后重试发送消息；**如果收到了消息**，那么可以提交事务（channel.txCommit）。但是问题是，rabbitmq事务机制一搞，基本上吞吐量会下来，因为太耗性能。

所以一般来说，如果你要确保说写rabbitmq的消息别丢，可以开启confirm模式，在生产者那里设置开启confirm模式之后，你每次写的消息都会分配一个唯一的id，然后如果写入了rabbitmq中，rabbitmq会给你回传一个ack消息，告诉你说这个消息ok了。如果rabbitmq没能处理这个消息，会回调你一个nack接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息id的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。

事务机制和cnofirm机制最大的不同在于，事务机制是同步的，你提交一个事务之后会阻塞在那儿，但是confirm机制是异步的，你发送个消息之后就可以发送下一个消息，然后那个消息rabbitmq接收了之后会异步回调你一个接口通知你这个消息接收到了。

所以一般在生产者这块避免数据丢失，都是用confirm机制的。

### 7.2 rabbitmq弄丢了数据

就是rabbitmq自己弄丢了数据，这个你必须开启rabbitmq的持久化，就是消息写入之后会持久化到磁盘，哪怕是rabbitmq自己挂了，恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是rabbitmq还没持久化，自己就挂了，可能导致少量数据会丢失的，但是这个概率较小。

设置持久化有两个步骤：

第一个是创建queue的时候将其设置为持久化的，这样就可以保证rabbitmq持久化queue的元数据，但是不会持久化queue里的数据；第二个是发送消息的时候将消息的deliveryMode设置为2，就是将消息设置为持久化的，此时rabbitmq就会将消息持久化到磁盘上去。必须要同时设置这两个持久化才行，rabbitmq哪怕是挂了，再次重启，也会从磁盘上重启恢复queue，恢复这个queue里的数据。

而且持久化可以跟生产者那边的confirm机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者ack了，所以哪怕是在持久化到磁盘之前，rabbitmq挂了，数据丢了，生产者收不到ack，你也是可以自己重发的。

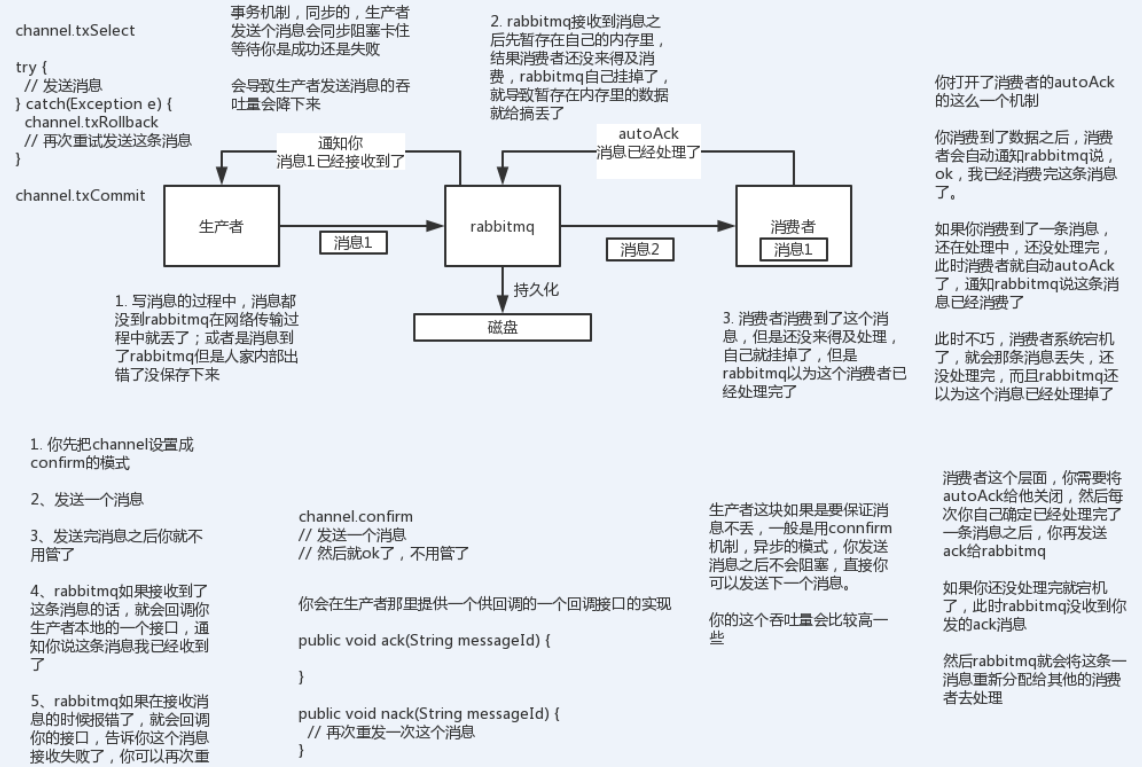
哪怕是你给rabbitmq开启了持久化机制，也有一种可能，就是这个消息写到了rabbitmq中，但是还没来得及持久化到磁盘上，结果不巧，此时rabbitmq挂了，就会导致内存里的一点点数据会丢失。

### 7.3 消费端弄丢了数据

rabbitmq如果丢失了数据，主要是因为你消费的时候，刚消费到，还没处理，结果进程挂了，比如重启了，那么就尴尬了，rabbitmq认为你都消费了，这数据就丢了。

这个时候得用rabbitmq提供的ack机制，简单来说，就是你关闭rabbitmq自动ack，可以通过一个api来调用就行，然后每次你自己代码里确保处理完的时候，再程序里ack一把。这样的话，如果你还没处理完，不就没有ack？那rabbitmq就认为你还没处理完，这个时候rabbitmq会把这个消费重新分配给consumer去处理，消息是不会丢的。

【总结】：

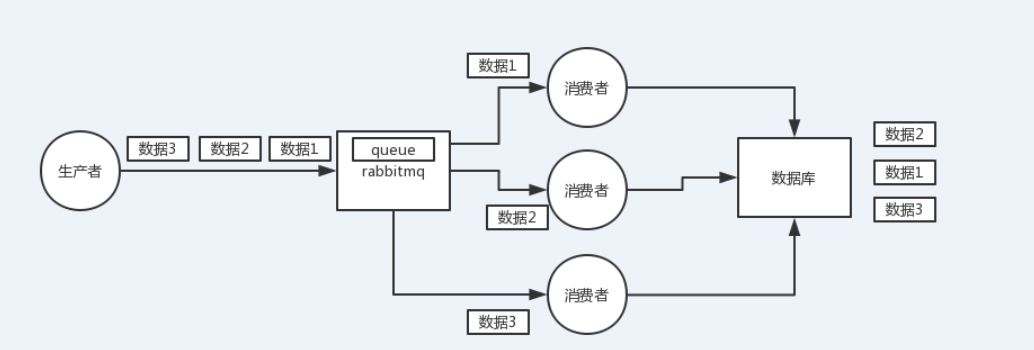


## 8 如何保证消息的顺序性

你在mysql里增删改一条数据，对应出来了增删改3条binlog，接着这三条binlog发送到MQ里面，到消费出来依次执行，起码得保证人家是按照顺序来的吧？不然本来是：增加、修改、删除；你楞是换了顺序给执行成删除、修改、增加，不全错了么。

本来这个数据同步过来，应该最后这个数据被删除了；结果你搞错了这个顺序，最后这个数据保留下来了，数据同步就出错了。

先看看顺序会错乱的场景：rabbitmq：一个queue，多个consumer，这不明显乱了



那如何保证消息的顺序性呢？简单

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。

