Kafka、ActiveMQ、RabbitMQ、RocketMQ 都有什么优点和缺点？

消息队列的作用

解耦、异步、削峰。

通过一个 MQ，Pub/Sub 发布订阅消息，A 系统就跟其它系统彻底解耦了。

Kafka、ActiveMQ、RabbitMQ、RocketMQ 有什么优缺点？

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类的系统来进行实时数据计算、日志采集等场景
topic 数量对吞吐量的影响			topic 可以达到几百/几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器下，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，需要增加更多的机器资源
时效性	ms 级	微秒级，这是 RabbitMQ 的一大特点，延迟最低	ms 级	延迟在 ms 级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据	基本不丢	经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用

最早大家都用 ActiveMQ，但没经过大规模吞吐量场景的验证，社区也不是很活跃。

RabbitMQ，erlang 开源的，比较稳定的支持，活跃度也高；

RocketMQ 阿里出品，但社区可能有突然黄掉的风险。目前 已捐给 Apache，但 GitHub 上的活跃度其实不算高。

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，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。

如何保证消息队列的高可用？

RabbitMQ 的高可用性

RabbitMQ 是比较有代表性的，因为是**基于主从**（非分布式）做高可用性的，我们就以 RabbitMQ 为例子讲解第一种 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。

单机模式

单机模式，就是 Demo 级别的。

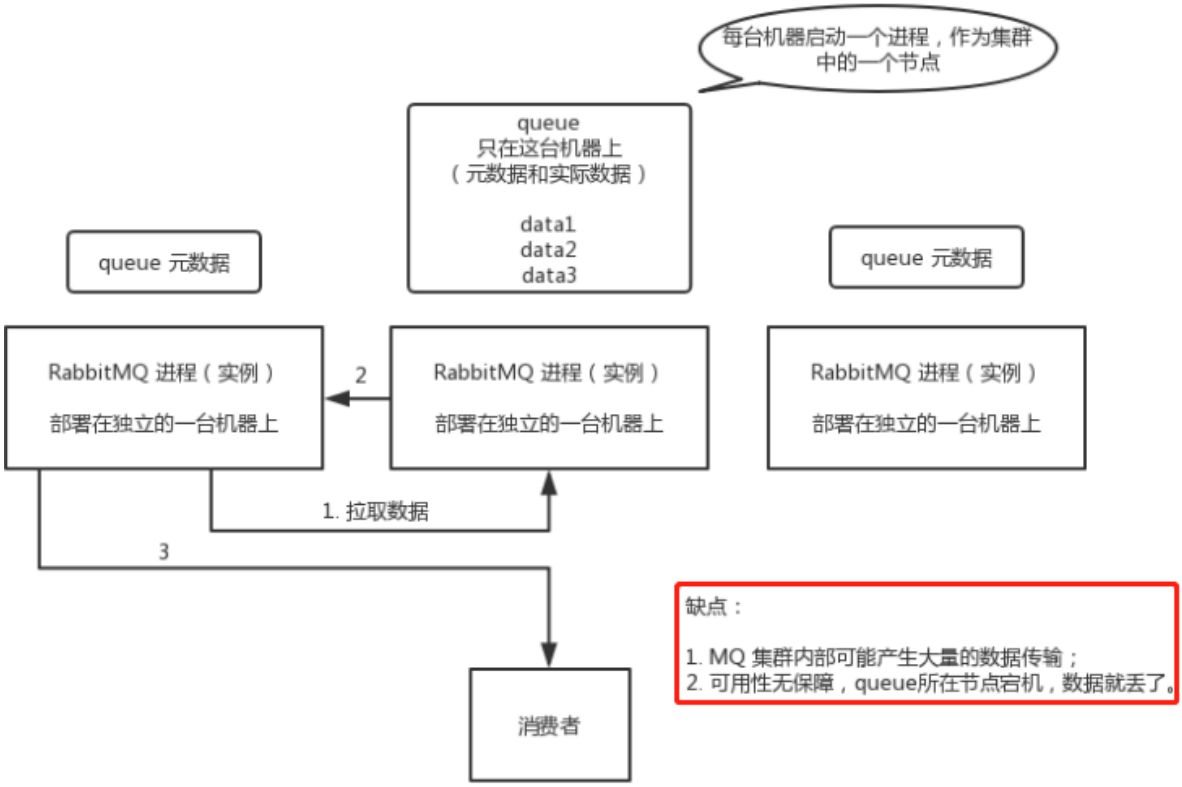
普通集群模式（无高可用性）

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。你**创建的 queue，只会放在一个 RabbitMQ 实例上**，但是每个实例都同步 queue 的元数据（元数据可以认为是 queue 的一些配置信息，通过元数据，可以找到 queue 所在实例）。你消费的时候，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。

这方案主要是提高吞吐量的，单纯的机子性能累加。

缺点：

1. MQ集群内部可能产生大量数据传输。
2. 可用性无保障，queue所在结点宕机，数据会丢失。



镜像集群模式（高可用性）

镜像就复制多份的意思。在镜像集群模式下，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**，就是说，每个 RabbitMQ 节点都有这个 queue 的一个**完整镜像**，包含 queue 的全部数据的意思。然后每次你写消息到 queue 的时候，都会自动把**消息同步**到多个实例的 queue 上。

如何开启这个镜像集群模式呢？管理控制台新增一个**镜像集群模式**的策略，指定的时候是可以要求数据同步到所有节点的，也可以要求同步到指定数量的节点，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。

好处在于，不怕某台机子宕机。

坏处在于

- 第一，性能开销大，消息需要同步到所有机器上，导致网络带宽压力和消耗很重。
- 第二，没有扩展性，如果这个 queue 的数据量很大，大到这个机器上的容量无法容纳了，此时该怎么办呢？

Kafka

Kafka 一个最基本的架构认识：由多个 broker 组成，每个 broker 是一个节点；你创建一个 topic，这个 topic 可以划分为多个 partition，每个 partition 可以存在于不同的 broker 上，每个 partition 就放一部分数据。

HA 机制，通俗的说就是主从复制，leader 宕机就从follower选举一个新的leader。写读都是去leader。

写数据的时候写leader 先持久化，然后follower主动同步后发送ack，收到了所有ack，才会会返回写成功的消息给生产者，这个消息才会被消费者读到

Kafka 会均匀地将一个 partition 的所有 replica 分布在不同的机器上，这样才可以提高容错性。

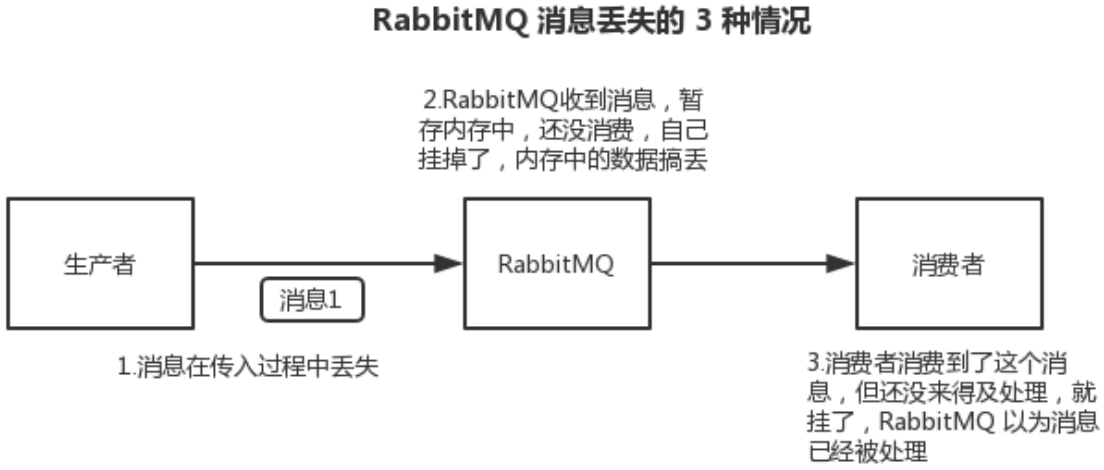
如何保证消息不被重复消费？（消息消费的幂等性）

场景：消费者已经消费了，但还没来得及通知kafka（ offset 并没有提交）就宕机了，重启的时候，由于之前的 offset 没有提交成功，数据会再次传过来。

解决的办法就消费者这边做一些去重的操作。

- **先查一下数据库**，已经有了就update
- **redis的Set**，有去重功能，天然幂等性。
- **每条数据加上的全局唯一的 id**，消费时候 id 去比如 Redis 里查一下。
- 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。因为有唯一键约束了，重复数据插入只会报错，不会导致数据库中出现脏数据。

如何保证消息的可靠性传输？（丢失）



生产者弄丢了数据

情况

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，因为网络问题啥的，都有可能。

解决

两个办法

1. 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。**事务是使吞吐量会下来，因为太耗性能。**
2. **设置开启 `confirm` 模式**，这样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，如果 RabbitMQ 没能处理这个消息，会回调你的一个 `nack` 接口，告诉你这个消息接收失败，你可以重试。还可以过重发。

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**异步回调**。所以一般在生产者这块**避免数据丢失**，都是用 `confirm` 机制的。

RabbitMQ 弄丢了数据

情况

RabbitMQ 一挂，内存里的数据就会全部丢失。

解决

开启 RabbitMQ 的持久化，就是消息写入之后会持久化到磁盘， RabbitMQ 挂了，**恢复之后会自动读取之前存储的数据**。除非极其罕见的是，RabbitMQ 还没持久化，自己就挂了，**可能导致少量数据丢失**，但是这个概率较小。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。

设置持久化有**两个步骤**：

- 创建 queue 的时候将其设置为持久化
  这样就可以保证 RabbitMQ 持久化 queue 的元数据，但是它是不会持久化 queue 里的数据的。
- 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。

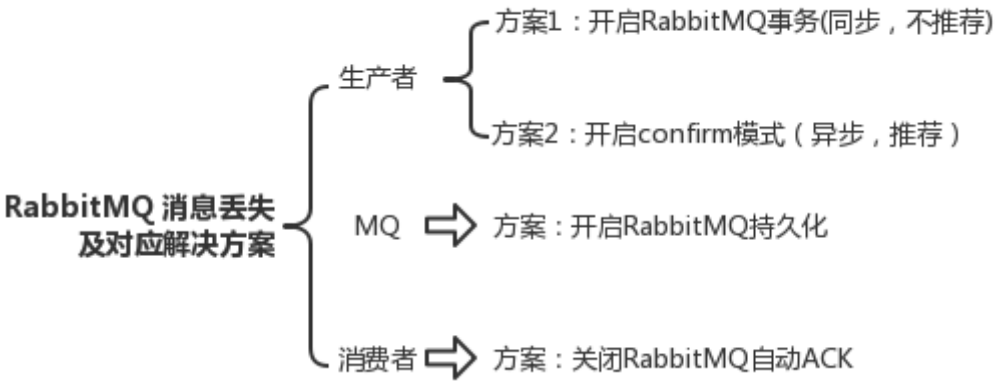
消费端弄丢了数据

情况

刚消费到，还没处理，结果进程挂了。RabbitMQ 认为你都消费了，这数据就丢了。

解决

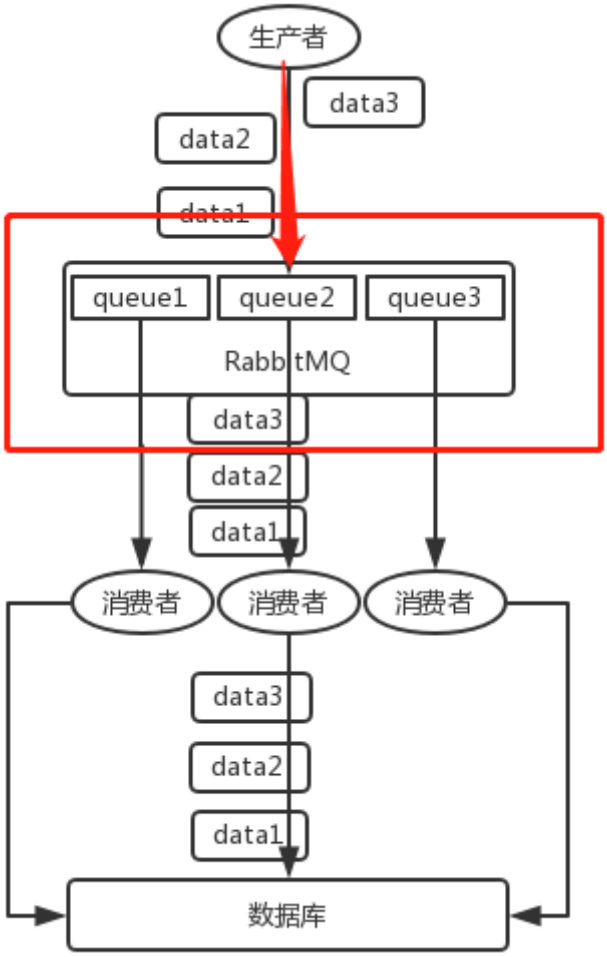
**关闭 RabbitMQ 的自动 `ack**，可以通过一个 api 来调用就行，然后每次你自己代码里确保处理完的时候，再在程序里ack` 一把。



如何保证消息的顺序性？

RabbitMQ

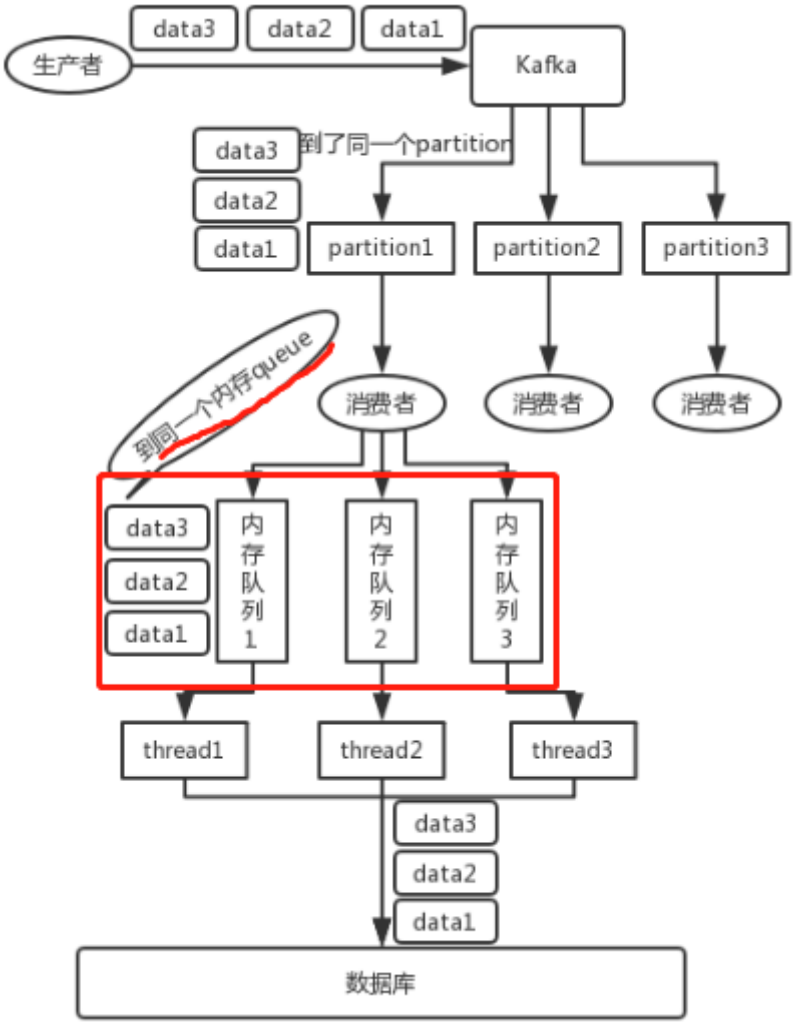
一个 queue，多个 consumer，**每个消费者分别**拿到了123条数据，每个消费者处理速度不一样，这样就乱了。



解决：拆分多个 queue，**每个消费者对应一个queue，把需要保证顺序的数据都发到一个queue里面。**就是多一些 queue 而已。

Kafka

本身就有多个queue，它叫partition。指定了某个订单 id 作为 key，那么这个订单相关的数据，一定会被分发到同一个 partition 中去，而且这个 partition 中的数据一定是有顺序的。**但是，我们在消费者里可能会搞多个线程来并发处理消息123。**



解决：写 N 个内存 queue，具有相同 key 的数据都到同一个内存 queue；然后对于 N 个线程，每个线程分别消费一个内存 queue 即可，这样就能保证顺序性。

总结：问题都出现在多个处理者处理的时候，会出现消息乱序。RabbitMQ是分发给消费者时，Kafka是消费者顺序拿到后分发给线程时，处理手段思路一样，**多个处理者每个处理者对应不同的queue,把需要保证顺序的数据都发到一个queue里面.**

延时以及过期失效怎么办？消息持续积压几小时怎么办？

大量消息在 mq 里积压了几个小时了还没解决

一般这个时候，只能临时**紧急扩容**了，具体操作步骤和思路如下：

- 先修复 consumer 的问题，确保其恢复消费速度，然后将现有 consumer 都停掉。
- 新建一个 topic，partition 是原来的 10 倍，临时建立好原先 10 倍的 queue 数量。
- 然后写一个临时的分发数据的 consumer 程序，这个程序部署上去消费积压的数据，**消费之后不做耗时的处理**，直接均匀轮询写入临时建立好的 10 倍数量的

queue。（为什么要这一步？即便紧急扩容十倍，也会要一定的时间,先把它拿出 别压在mq里）

- 接着临时征用 10 倍的机器来部署 consumer，每一批 consumer 消费一个临时 queue 的数据。这种做法相当于是临时将 queue 资源和 consumer 资源扩大 10 倍，以正常的 10 倍速度来消费数据。
- 等快速消费完积压数据之后，**得恢复原先部署的架构**，**重新**用原先的 consumer 机器来消费消息。

mq 都快写满了

临时写程序，接入数据来消费，**消费一个丢弃一个，都不要了**，快速消费掉所有的消息。然后走第二个方案，到了晚上再补数据吧。

mq 中的消息过期失效了

假设你用的是 RabbitMQ，RabbtiMQ 是可以设置过期时间的，也就是 TTL。一般这个是不允许设置的，如果你设置了，这就不是说数据会大量积压在 mq 里，而是**大量的数据会直接搞丢**。

设计一个消息队列

- 首先这个 mq 得支持可伸缩性吧，就是需要的时候快速扩容，就可以增加吞吐量和容量，那怎么搞？设计个分布式的系统呗，参照一下 kafka 的设计理念，broker -> topic -> partition，每个 partition 放一个机器，就存一部分数据。如果现在资源不够了，简单啊，给 topic 增加 partition，然后做数据迁移，增加机器，不就可以存放更多数据，提供更高的吞吐量了？
- 其次你得考虑一下这个 mq 的数据要不要落地磁盘吧？那肯定要了，落磁盘才能保证别进程挂了数据就丢了。那落磁盘的时候怎么落啊？顺序写，这样就没有磁盘随机读写的寻址开销，磁盘顺序读写的性能是很高的，这就是 kafka 的思路。
- 其次你考虑一下你的 mq 的可用性啊？这个事儿，具体参考之前可用性那个环节讲解的 kafka 的高可用保障机制。多副本 -> leader & follower -> broker 挂了重新选举 leader 即可对外服务。
- 能不能支持数据 0 丢失啊？可以的，参考我们之前说的那个 kafka 数据零丢失方案。