MQ

(此处的大多是以rabbitmq为例, 其他mq也是类似的, 只是细节不同)

MQ有什么作用/为什么使用MQ (高)

消息队列的本质其实就是一个阻塞队列, 只是在阻塞队列的基础上增加了重试, 消息持久化等等功能.

异步:

- 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
- 如果使用 MQ，那么 A 系统自己本地写库要 3ms, 连续发送 3 条消息到 MQ 队列中，假如耗时 5ms，A 系统从接受一 个请求到返回响应给用户，总时长是 3 + 5 = 8ms; 就非常快

削峰:

- 订单系统中, 在业务高峰期, 下单的太多. 那么直接将生成得订单id放到MQ中, 消息都堆在MQ中. 这就是削峰
- 让新线程/其他服务监听MQ, 然后慢慢地去异步地进行订单创建, 库存扣减等工作. 这些工作会持续到业务的低谷期. 所以叫平谷

解耦:

- A通过接口调用的形式发送数据给BCD, 此时如果E也需要这个数据呢? 如果C不要这个数据呢? 这时候A和其他系统严重耦合.
- 如果A直接发送数据给MQ, 新系统需要数据, 直接监听MQ即可, 旧系统不需要数据了, 就不需要监听MQ了.

- 这样就是谁需要谁去拿, A只用和MQ交互. A不需要考虑给谁发送数据, 也不需要考虑别人调用失败, 或者超时了怎么办.

MQ优缺点 (高)

优点就是: 异步 削峰 解耦

缺点就是:

- 系统的**可用性降低**了: 新引入了MQ, 那么如果MQ挂了, 和MQ相关的服务就崩溃了
- 系统**复杂度提高**了: 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
- **数据一致性问题**: 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

如何保证消息可靠性 (高)

消息丢失的几种情况

- 发送时丢失
  - 生产者发送消息到交换机的过程中丢失了 (消息确认 confirm 机制)
  - 交换机分发消息到队列的过程中丢失了 (return机制)
- MQ宕机
  - 队列已经接收到了消息, 但未持久化, MQ宕机就会丢失消息 (消息持久化以及MQ集群, 保证高可用)

- 消费时丢失
    - 消费者拿到消息后未消费消息就丢失了 (消费者手动ACK)

confirm机制, 发送者确认

- 消息成功投递到交换机，返回ack
- 消息未投递到交换机，返回nack

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。

return, 发送者回执

- 消息投递到交换机了，但是没有路由到队列。返回ACK,及路由失败原因。
- 消息成功从交换机路由到队列, 则不返回任何东西.

项目中配置ConfirmCallback和ReturnCallback. ConfirmCallback就是设置消息未成功投递到交换机, 要做什么, 比如记录日志之类的. ReturnCallback消息没有从交换机路由到队列时触发的回调, 可以记录一下日志.

消息持久化: 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.

消费者ACK: 消费者确认机制，即：消费者处理消息后可以向MQ发送ack回执，MQ收到ack回执后才会删除该消息。

死信队列? 如何导致死信 (中)

死信，顾名思义就是无法被消费的消息. 当一个队列中的消息满足下列情况之一时，可以成为死信（dead letter):

- 消息被拒绝消费
- 消息TTL到期，超时无人消费
- 要投递的队列消息堆积满了，最早的消息可能成为死信

如果该队列配置了dead-letter-exchange属性，指定了一个交换机，那么队列中的死信就会投递到这个交换机中, 而这个交换机称为死信交换机（Dead Letter Exchange,简称DLX)。

死信队列是队列将死信(无法被消费的消息放到死信交换机中)

- 死信队列可以像republish一样作为消息消费失败的兜底方案
    - 在死信队列里面可以对我们的异常消息进行MySQL/Redis持久化，然后人工处理等操作
- 另一方面可以处理消息超时无人消费以及队列满了的问题.

延时队列 (中)

延迟队列指的是消息发送后到mq不会立即被消费，mq会存储对应的延迟消息，而是等待特定时间后，消费者才能拿到这个消息进行消费

比如订单的超时取消, 订单信息被放到mq中, 30分钟未支付订单就取消. 如果使用延时队列, 那监听mq的消费者从mq中直接拿到的就是30分钟未支付订单的信息, 然后直接取消订单. 避免轮询数据库查找超时订单.

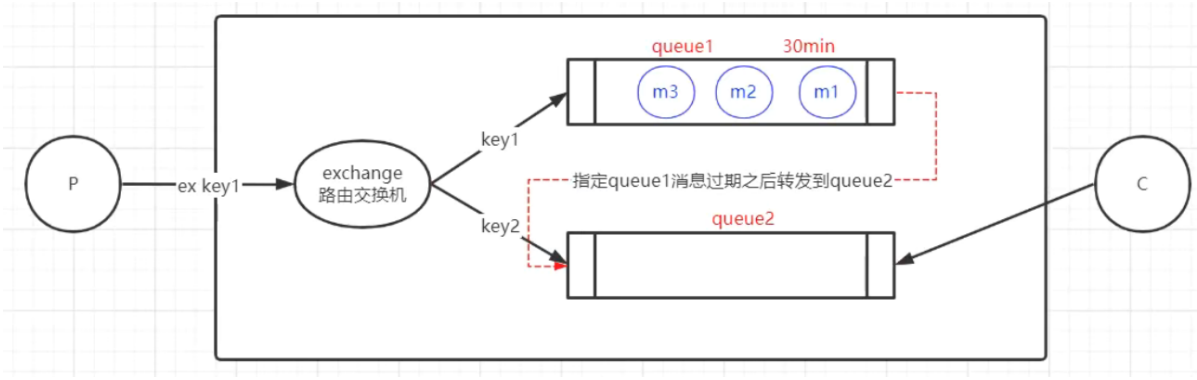
主流mq都支持延时队列, 如RocketMQ、RabbitMQ、Pulsar、Kafka，都支持定时/延时消息

rabbitMQ的延时队列, 需要通过插件或者TTL机制模拟才能实现.

RabbitMQ使用TTL机制模拟延时队列 (中)

TTL就是消息的存活时间。RabbitMQ可以分别队列设置消息存活时间.

- 在创建队列A的时候可以设置队列中消息的存活时间TTL，当生产者发送消息进入到队列A并且在存活时间内没有消费者消费，则此消息就会从当前队列被移除
- **当TTL结束之后，我们可以指定将当前队列A的消息转存到的队列B**
- 那么消费者监听队列B, 从队列B中得到的消息就是延迟消息了.



消息的幂等性 (高)

网络不好的时候, A向队列发送一条消息, 消息发送成功, 队列将消息给B服务, 但是由于网络原因A服务一直未收到ack, 此时A服务重发了消息, 队列又将消息给B服务, 此时就出现了消息重复消费.

或者说MQ向B服务发送了消息, 凡是网络原因, B服务没有能向MQ发送ack, 所以队列将消息重复发送了一遍, 产生了重复消费.

消息重复消费是无法避免的, 所以要做消息的幂等. 幂等性的处理方式如下:

- 唯一约束
  - 如果从MQ拿到数据是要存到数据库，那么可以根据数据创建唯一约束
  - 同样的数据从MQ发送过来之后，当插入数据库的时候，会报违反唯一约束，不会插入成功的。
  - 或者可以先查一次，是否在数据库中已经保存了，如果能查到，那就直接丢弃就好了(**在高并发的情况下，有数据库写入的瓶颈**)
- 消息唯一id
  - 让生产者发送消息时，每条消息加一个全局的唯一id
  - 然后消费时，将该id保存到redis里面
  - 下次再消费时先去redis里面查一下有么有，没有再消费

消息积压如何处理 (中)

当生产者发送消息的速度超过了消费者处理消息的速度, 或者如果消费者因为某些原因持续阻塞，就会导致队列中的消息堆积，直到队列存储消息达到上限。最早接收到的消息，可能就会成为死信，会被丢弃，这就是消息堆积问题

解决消息堆积

- 增加更多消费者，提高消费速度
- 提高单个消息者的处理能力, 在消费者内开启线程池加快消息处理速度
  - 缺点: 消息太多就会开启太多新线程, cpu压力大

如果是bug导致几百万消息持续积压几小时。有如何处理呢?

1. 先修复consumer消费者的bug，以确保其恢复消费速度，然后将现有consumer都停掉
2. 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
3. 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
4. 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
5. 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
6. 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

消息的顺序性 (中)

(该方案适合rabbitmq, rocketmq和kafka有所不同)

为什么要保证顺序

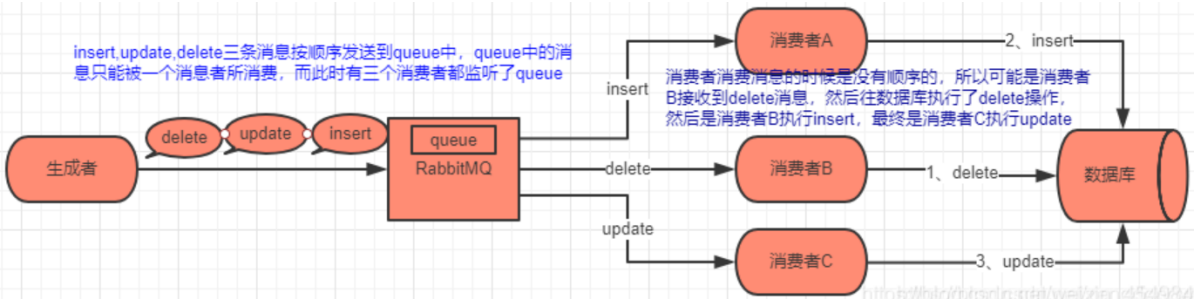
- 消息队列中的若干消息如果是对同一个数据进行操作，这些操作具有前后的关系，必须要按 前后的顺序执行，否则就会造成数据异常。

举例:

- 业务要对某个数据依次做 插入->更新->删除操作，这个顺序必须是这样
- 如果在消费过程中，消息的顺序变成了 删除->插入->更新，那么原本应该被删除的数据，就没有被删除，造成数据的不一致问题

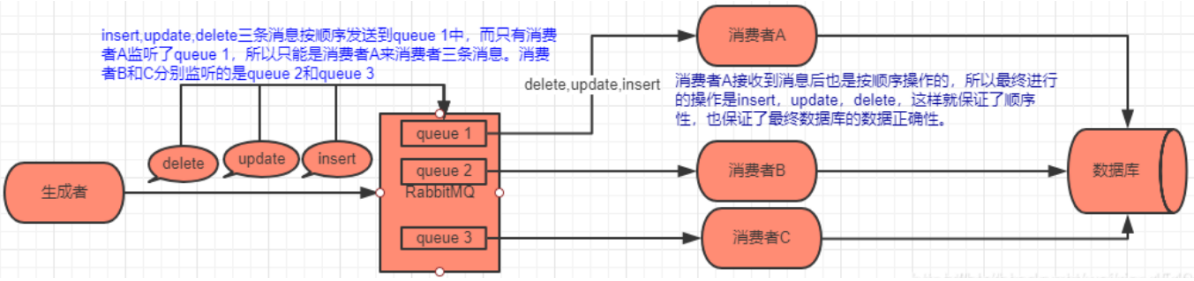
出现顺序错乱的场景

- 一个queue => 多个consumer去消费
- consumer从MQ里面读取数据是有序的， 但是每个consumer的执行时间是不固定的，无法保证先读到消息的 consumer一定先完成操作
- 这样就会出现消息并没有按照顺序执行，造成数据顺序错误



保证消息的顺序消费

- 拆分多个queue，每个queue一个consumer
- 会使队列变多, 造成吞吐量下降
- 这种可以在消费者内部采用多线程的方式去消费



不同的消息中间件对消息顺序性的支持和实现方式有所不同，一些消息中间件如 Kafka、RocketMQ 在设计上就对消息顺序性有较好的支持

- **Kafka**：通过分区和有序消费保证分区内消息顺序性，只要生产者将相关消息发送到同一个分区，消费者按顺序从该分区拉取消息，就能保证消息顺序。
- **RocketMQ**：支持顺序消息，通过将消息发送到特定的队列，并确保消费者按照顺序从队列中获取消息来保证顺序性