消息队列

使用场景

异步处理

发送者将消息发送给消息队列之后，不需要同步等待消息接收者处理完毕，而是立即返回进行其它操作。消息接收者从消息队列中订阅消息之后异步处理。

例如在注册流程中通常需要发送验证邮件来确保注册用户身份的合法性，可以使用消息队列使发送验证邮件的操作异步处理，用户在填写完注册信息之后就可以完成注册，而将发送验证邮件这一消息发送到消息队列中。

只有在业务流程允许异步处理的情况下才能这么做，例如上面的注册流程中，如果要求用户对验证邮件进行点击之后才能完成注册的话，就不能再使用消息队列。

流量削锋

在高并发的场景下，如果短时间有大量的请求到达会压垮服务器。

可以将请求发送到消息队列中，服务器按照其处理能力从消息队列中订阅消息进行处理。

应用解耦

如果模块之间不直接进行调用，模块之间耦合度就会很低，那么修改一个模块或者新增一个模块对其它模块的影响会很小，从而实现可扩展性。

通过使用消息队列，一个模块只需要向消息队列中发送消息，其它模块可以选择性地从消息队列中订阅消息从而完成调用。

RocketMQ的一些问题

消费模式

Rocketmq消费分为push和pull两种方式，push为被动消费类型，pull为主动消费类型，push方式最终还是会从broker中pull消息。不同于pull的是，push首先要注册消费监听器，当监听器处触发后才开始消费消息，所以被称为“被动”消费。

消息丢失

当你系统需要保证百分百消息不丢失，你可以使用生产者每发送一个消息，Broker 同步返回一个消息发送成功的反馈消息。即每发送一个消息，同步落盘后才返回生产者消息发送成功，这样只要生产者得到了消息发送生成的返回，事后除了硬盘损坏，都可以保证不会消息丢失。

消息堆积

根据不同的业务实现不同的丢弃任务。

顺序消息

生产者生产消息时指定特定的 MessageQueue ，消费者消费消息时，消费特定的 MessageQueue，其实单机版的消息中心在一个 MessageQueue 就天然支持了顺序消息。注意：同一个 MessageQueue 保证里面的消息是顺序消费的前提是：消费者是串行的消费该 MessageQueue，因为就算 MessageQueue 是顺序的，但是当并行消费时，还是会有顺序问题，但是串行消费也同时引入了两个问题：

> 1. 引入锁来实现串行
> 2. 前一个消费阻塞时后面都会被阻塞

消息重复

- RocketMQ 会出现消息重复发送的问题，因为在网络延迟的情况下，这种问题不可避免的发生，如果非要实现消息不可重复发送，那基本太难，因为网络环境无法预知，还会使程序复杂度加大，因此默认允许消息重复发送
- RocketMQ 让使用者在消费者端去解决该问题，即需要消费者端在消费消息时支持幂等性的去消费消息
- 最简单的解决方案是每条消费记录有个消费状态字段，根据这个消费状态字段来是否消费或者使用一个集中式的表，来存储所有消息的消费状态，从而避免重复消费
- 具体实现可以查询关于消息幂等消费的解决方案

RocketMQ 不使用 ZooKeeper 作为注册中心的原因，以及自制的 NameServer 优缺点

- ZooKeeper 作为支持顺序一致性的中间件，在某些情况下，它为了满足一致性，会丢失一定时间内的可用性，RocketMQ 需要注册中心只是为了发现组件地址，在某些情况下，RocketMQ 的注册中心可以出现数据不一致性，这同时也是 NameServer 的缺点，因为 NameServer 集群间互不通信，它们之间的注册信息可能会不一致
- 另外，当有新的服务器加入时，NameServer 并不会立马通知到 Produer，而是由 Produer 定时去请求 NameServer 获取最新的 Broker/Consumer 信息（这种情况是通过 Producer 发送消息时，负载均衡解决）