# 消息队列面试题

**生产/消费者模式**

一个服务生产数据（发起请求），一个服务消费数据（处理请求)，服务间解耦; 共享区存放中间数据。共享区满会阻塞生产者继续发消息；共享区空会阻塞消费者消费消息。

**发布订阅模式**

生产者指定主题(topic)和标签(tag)发送消息，接收者根据主题（topic）和标签(tag)接受消息

**推模式和拉模式**

生产者发消息一定是推模式

推模式: MQ将消息推给消费者，触发消费者的监听方法，对消息进行处理；实时性高，消息多可能导致缓冲区溢出

拉模式：每隔几秒消费者调请求，主动去MQ获取，有延迟，消费不过来慢点拉

**广播\集群模式**

集群：一条消息消费组(微服务)中只有一个消费者会消费

广播：一条消息消费组中所有消费者都会去消费

**MQ应用场景**

应用解耦：员工入职后，用MQ通知多个下游系统处理,入职系统和其他系统低耦合，宕机/异常不会相互影响

异步：前端表格数据导出时，后端先发送导出消息,接口返回"到导出页面下载"，自己接收MQ后异步处理，避免前端长时间等待

流量削峰(限流)：秒杀中大量请求时，请求先进入MQ,再按顺序进行处理，如果超出队列最大值，也可直接返回404

**消息重复**

重复消费原因：由于网络原因ack未发送给生产者/MQ导致消息重复发送

重复发送的后果：重复消费(重复扣款)

**MQ防止重复消费（幂等性）**

幂等性：保证消息唯一，重复发送后，也只能消费一次，将重复消息丢弃

rocketMQ内部幂等：每个消息有唯一的消息IDmessageId, 作为幂等判断，重复收到则丢弃

业务上的幂等：将消息落地，每次接收消息去缓存中查询业务id是否存在

**消息队列如何保证顺序消费**

将MQ分成多个队列，有顺序的消息放入同个队列，让同个消费者去消费(锁/指定消费模式)

**MQ工作流程**

生产者发消息给 MQ；

MQ将消息持久化,并发 ACK给生产者；发消息给消费者；

消费者收到消息后发 ACK 给 MQ；

MQ接收ACK后，更新消息状态；

**消息基于什么传输**  
TCP连接的创建和销毁开销大，影响性能，RabbitMQ使用信道传输数据。信道是TCP连接内的虚拟连接，没有信道数量限制

**为什么不将所有消息持久化**

持久化会降低MQ的性能，如果对性能要求高，应该对关键消息做持久化

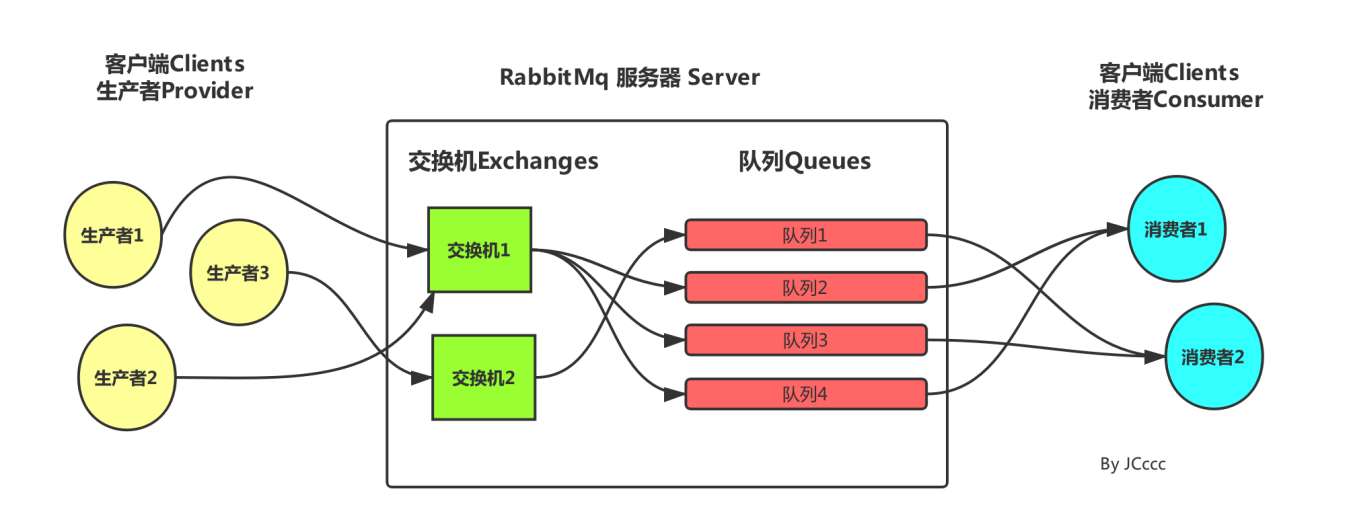
**Rabbitmq中的持久化**

队列持久化：重启mq后队列依然存在

消息持久化：重启mq后消息依然存在

交换机持久化：重启mq后交换机依然存在

**RabbitMq工作流程**



**消息可靠性保证(防止消息丢失)**

生产者丢消息：由于网络原因没发到MQ

使用事务消息：发送异常就回滚

确认机制：MQ收到消息后，发送ack给生产者，生产者长时间未收到，则认为失败，进行重发

**消息队列丢消息**：已发送ack给生产者，MQ重启/宕机

持久化机制：改成持久化后再给生产者发送ack

主从复制机制：主节点的消息会同步给从节点，主节点宕机，从节点也能提供服务

**消费者丢消息**：

自动提交：处理完消息后，自动发送ack给MQ，但消费逻辑异常可能导致被错误确认

手动提交：等处理完消息后，手动显示调用发送ack发给MQ，适合可靠性高的场景

重试队列：当消费失败后将消息发到重试队列，进行重发，多次重发无效后加入死信队列

**事务消息**

解决本地事务和发消息同时成功/失败，本地事务执行成功后才发送消息，如果发送失败本地事务需要回滚

使用场景：异步更新数据，对实时性要求不太高的场景

分布式事务实现有2PC（二阶段提交）、TCC（Try-Confirm-Cancel）和事务消息

生产者在消息队列上开启事务，给MQ发一个“半消息”，包含完整消息内容，事务提交前，对消费者不可见.

半消息发成功后，生产者执行本地事务，成功则提交事务消息，消费者就能看消息;失败，则回滚事务消息，消费者不会收到消息。

**提交/回滚事务消息失败了怎么办？**

RocketMQ使用事务反查机制。生产者在提交/回滚事务消息发生网络异常，MQ没有收到请求，MQ会定期去生产者上反查该事务对应本地事务的状态，根据反查结果决定提交/回滚。

反查机制需要业务代码实现反查本地事务状态的接口，返回本地事务是执行成功还是失败。

**事务中的消息发送**

单条消息放到最后进行发送，发送失败事务会回滚，事务失败也不会发送MQ

多条消息,如果一条成功发送，一条失败，事务也会回滚。将消息入库，消息ID存到threadLocal中，等事务提交后再发送(重写事务管理器，每次提交后去threadlocal查询待发送的消息ID,进行发送，发送失败则定时重发)

**消息堆积的解决方式**

监控消息，增加消费者节点，消费者多线程处理消息

优化消费逻辑，设置优先级，提高重要消息消费等级

长时间无法消费的消息，进行重试发送，多次后加入死信队列

**MQ区别**

