RocketMQ 面试题

1、多个 MQ 如何选型？

MQ	描述
RabbitMQ	erlang 开发，对消息堆积的支持并不好，当大量消息积压的时候，会导致 RabbitMQ 的性能急剧下降。每秒钟可以处理几万到十几万条消息。
RocketMQ	Java 开发，面向互联网集群化功能丰富，对在线业务的响应时延做了很多的优化，大多数情况下可以做到毫秒级的响应，每秒钟大概能处理几十万条消息。
Kafka	Scala 开发，面向日志功能丰富，性能最高。当你的业务场景中，每秒钟消息数量没有那么多的时候，Kafka 的时延反而会比较高。所以，Kafka 不太适合在线业务场景。
ActiveMQ	Java 开发，简单，稳定，性能不如前面三个。小型系统用也可以，但是不推荐。推荐用互联网主流的。

2、为什么要使用 MQ？

因为项目比较大，做了分布式系统，所有远程服务调用请求都是同步执行经常出问题，所以引入了 MQ。

作用	描述
解耦	系统耦合度降低，没有强依赖关系。
异步	不需要同步执行的远程调用可以有效提高响应时间。
削峰	请求达到峰值后，后端 service 还可以保持固定消费速率消费，不会被压垮。

3、RocketMQ 由哪些角色组成，每个角色作用和特点是什么？

角色	作用
Nameserver	无状态，动态列表；这是和 ZooKeeper 的重要区别之一。ZooKeeper 是有状态的。
Producer	消息生产者，负责发消息到 Broker。
Broker	就是 MQ 本身，负责收发消息、持久化消息等。
Consumer	消息消费者，负责从 Broker 上拉取消息进行消费，消费完进行 ack。

4、RocketMQ 中的 Topic 和 JMS 的 queue 有什么区别？

queue 就是来源于数据结构的 FIFO 队列。而 Topic 是个抽象的概念，每个 Topic 底层对应 N 个 queue，而数据也真实存在 queue 上的。

5、RocketMQ 消费模式有几种？

消费模型由 Consumer 决定，消费维度为 Topic。

- 集群消费

1.一条消息只会被同 Group 中的一个 Consumer 消费

2.多个 Group 同时消费一个 Topic 时，每个 Group 都会有一个 Consumer 消费到数据。

- 广播消费

消息将对一个 Consumer Group 下的各个 Consumer 实例都消费一遍。即即使这些

Consumer 属于同一个 Consumer Group，消息也会被 Consumer Group 中的每个 Consumer 都消费一次。

6、Broker 如何处理拉取请求的？

Consumer 首次请求 Broker。

- Broker 中是否有符合条件的消息
- 有
  - 响应 Consumer。
  - 等待下次 Consumer 的请求。
- 没有
  - DefaultMessageStore#ReputMessageService#run 方法。
  - PullRequestHoldService 来 Hold 连接，每个 5s 执行一次检查 pullRequestTable 有没有消息，有的话立即推送。
  - 每隔 1ms 检查 commitLog 中是否有新消息，有的话写入到 pullRequestTable。
  - 当有新消息的时候返回请求。
  - 挂起 consumer 的请求，即不断开连接，也不返回数据。
  - 使用 consumer 的 offset。

7、RocketMQ 如何做负载均衡？

通过 Topic 在多 Broker 中分布式存储实现。

producer 端

发送端指定 message queue 发送消息到相应的 broker，来达到写入时的负载均衡

- 提升写入吞吐量，当多个 producer 同时向一个 broker 写入数据的时候，性能会下降
- 消息分布在多 broker 中，为负载消费做准备

默认策略是随机选择：

- producer 维护一个 index
- 每次取节点会自增
- index 向所有 broker 个数取余
- 自带容错策略

其他实现：

- SelectMessageQueueByHash
  - hash 的是传入的 args
- SelectMessageQueueByRandom
- SelectMessageQueueByMachineRoom 没有实现

也可以自定义实现 MessageQueueSelector 接口中的 select 方法

```
MessageQueue select(final List<MessageQueue> mqs, final Message msg, final Object arg);
```

consumer 端

采用的是平均分配算法来进行负载均衡。

其他负载均衡算法

平均分配策略(默认)(AllocateMessageQueueAveragely) 环形分配策略

(AllocateMessageQueueAveragelyByCircle) 手动配置分配策略

(AllocateMessageQueueByConfig) 机房分配策略

(AllocateMessageQueueByMachineRoom) 一致性哈希分配策略

(AllocateMessageQueueConsistentHash) 靠近机房策略

(AllocateMachineRoomNearby)

追问：当消费负载均衡 consumer 和 queue 不对等的时候会发生什么？

Consumer 和 queue 会优先平均分配，如果 Consumer 少于 queue 的个数，则会存在部分 Consumer 消费多个 queue 的情况，如果 Consumer 等于 queue 的个数，那就是一个 Consumer 消费一个 queue，如果 Consumer 个数大于 queue 的个数，那么会有部分 Consumer 空余出来，白白的浪费了。

8、消息重复消费

影响消息正常发送和消费的重要原因是网络的不确定性。

引起重复消费的原因

- ACK

正常情况下在 consumer 真正消费完消息后应该发送 ack，通知 broker 该消息已正常消费，从 queue 中剔除

当 ack 因为网络原因无法发送到 broker，broker 会认为词条消息没有被消费，此后会开启消息重投机制把消息再次投递到 consumer

- 消费模式

在 CLUSTERING 模式下，消息在 broker 中会保证相同 group 的 consumer 消费一次，但是针对不同 group 的 consumer 会推送多次

解决方案

- 数据库表

处理消息前，使用消息主键在表中带有约束的字段中 insert

- Map

单机时可以使用 map ConcurrentHashMap -> putIfAbsent guava cache

- Redis

分布式锁搞起来。

9、RocketMQ 如何保证消息不丢失

首先在如下三个部分都可能会出现丢失消息的情况：

- Producer 端
- Broker 端
- Consumer 端

13.1、Producer 端如何保证消息不丢失

- 采取 send()同步发消息，发送结果是同步感知的。
- 发送失败后可以重试，设置重试次数。默认 3 次。

```
producer.setRetryTimesWhenSendFailed(10);
```

集群部署，比如发送失败了的原因可能是当前 Broker 宕机了，重试的时候会发

送到其他 Broker 上。

13.2、Broker 端如何保证消息不丢失

- 修改刷盘策略为同步刷盘。默认情况下是异步刷盘的。

```
FLUSHDISKTYPE = SYNC_FLUSH
```

- 集群部署，主从模式，高可用。

13.3、Consumer 端如何保证消息不丢失

- 完全消费正常后在进行手动 ack 确认。

10、高吞吐量下如何优化生产者和消费者的性能?

开发

- 同一 group 下，多机部署，并行消费
- 单个 Consumer 提高消费线程个数
- 批量消费
    - 消息批量拉取
    - 业务逻辑批量处理

运维

- 网卡调优
- jvm 调优
- 多线程与 cpu 调优
- Page Cache