ActiveMQ：相对于其它性能较差，版本迭代较慢 不建议使用

RabbitMQ: 吞吐量比kafaka和RocketMQ差一些，并发能力强，延时低是唯一一个达到微秒级的MQ，如果并发量要求不是很高(十万级以下)，首选 具有完备的后台管理系统，缺陷：erlan语言维护成本大

RocketMQ: 阿里根据kafaka开发的

Kafaka: 只提供核心功能，超高的吞吐量，高可用，支持分布式任意扩展，但是可能造成重复消费，常用在日志计算与收集

吞吐量：万级：ActiveMQ、RabbitMQ 十万级：RocketMQ、Kafaka

可用性：都高可用，ActiveMQ+ RabbitMQ—主从架构实现高可用

RocketMQ+kafaka基于分布式架构 一个数据多个副本

时效性：RabbitMQ微秒级 其他毫秒级

消息丢失：ActiveMQ 和 RabbitMQ 丢失的可能性非常低， RocketMQ 和 Kafka 理论上不会丢失

不重复消费

重复发送并不可怕，对重复消费做幂等性校验才是重点

Kafaka:有offset的概念，每个消息写入的时候 都有一个offset与之对应，然后consumer消费之后，每隔一段时间会把消费过的offset提交一下，但是如果还没来的及提交 就异常，会导致部分消息再次消费

幂等性：

1、操作数据库 根据主键查一下 选择update

2、写Redis set 天然幂等

3、Redis记录过滤等

不丢失数据

生产者丢失数据：生产者发送到中间件时 出现网络问题

rabbit提供了事务功能(同步)，在发送数据之前开启事务，如果rabbit没有收到消息 就会报错回滚事务 如果收到了就提交事务—降低吞吐 影响性能

开启confirm模式(异步)：生产者每次写消息都会分配一个唯一ID 如果写入了中间件会回传一个ack消息 如果异常会 回传一个nack消息

RabbitMQ丢失数据：

开启rabbit持久化，1、创建队列时设置为持久化，2、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，即使重启也可以恢复数据，

依然采用confirm模式，只有消息被持久化到磁盘 才会通知ack 否则都返回nack

Kafaka：在重新选举时 造成数据丢失

1. 通过设置参数 给partition至少两个副本
2. 要求leader只有保证有一个follower与自己保持联系
3. 设置acks=all 要求每条数据写入replica 才算写入成功
4. 极端设置 retries=MAX 无限重试

消费端丢失数据：接受到消息 还没处理 挂了

Rabbit提供的ack机制 关闭自动ack 每次处理完消息 再调用api返回ack

Kafaka：关闭自动提交offset 在处理完之后调用API提交

保证顺序性

RabbitMQ：每个队列一个consumer 然后内部使用内存队列

Kafaka：写多个队列 具有相同key的数据 放到同一个队列 然后对于多个线程 每个线程消费一个队列

Kafaka具体parttion是怎么保证有序性的？

保证高可用

RabbitMQ：(非分布式MQ)

普通集群模式：

只能提高消费者的吞吐 没有什么高可用性

镜像集群模式：队列存在多个实例中 每次写入队列都会同步到多个实例，每个节点都包含queue的所有信息

优点：即使出现宕机现象 其他机器可以继续提供服务

缺点：消耗太大，所有消息都要同步所有机器 网络压力大，没有好的方法扩展

Kafaka：基本架构 多个broker组成 一个broker就是一个节点 创建一个topic 可以划分为多个partition 每个partition可以存在不同的broker上面 每个partition存放一部分数据 典型的分布式

通过HA机制 replica副本机制 每个partition的数据都会同步到其他机器上，形成replica副本 然后所有的副本会选举一个leader出来 写的时候会把数据同步到跟随者 读写的时候只能操作leader

高峰期积压削峰

MQ的优缺点：

缺点：

系统可用性降低，MQ故障

系统复杂性变高

一致性问题

Kafaka快的原因：Kafka就是使用的顺序I/O+对Java的内存管理和垃圾回收会有优化

Kafka速度的秘诀在于，它把所有的消息都变成一个批量的文件，并且进行合理的批量压缩，减少网络IO的损耗，通过MMAP提高I/O的速度。写入数据的时候，由于单个Partition（分区）是末尾添加的所以速度最优；读取数据的时候配合sendfile直接暴力输入

了优化写入速度，Kafka采用了两种技术，一种是顺序写入，一种是MMFile

使用磁盘操作有以下几个好处：

1.磁盘顺序读写速度超过内存随机读写。

2.JVM的GC效率低，内存占用大，使用磁盘可以避免这一问题。

3.系统冷启动后，磁盘上的缓存依然可用（内存一旦关机数据就会清空，持久化到磁盘上则不会）

即便是顺序写入磁盘，磁盘的访问速度还是不可能追上内存的。所以Kafka的数据并不是实时的写入硬盘，它充分利用了现代操作系统的分页存储来利用内存，以此来提高I/O效率。Memory Mapped Files（后面简称MMAP）

通过MMAP，进程就可以像读写硬盘一样读写内存（当然是虚拟机内存），也不必关系内存的大小，因为有虚拟内存为我们兜底。使用这种方式可以获取很大的I/O提升，省去了用户空间到内核空间复制的开销

Kafka使用了基于sendfile的Zero Copy提高Web Server静态文件的速度。

Kafka把所有的消息都存放在一个一个的文件中，当消费者需要数据的时候Kafka直接把文件发送给消费者。这就是秘诀所在，比如：10W的消息组合在一起是10MB的数据量，然后Kafka用类似于发文件的方式直接扔出去了，如果消费者和生产者之间的网络非常好（只要网络稍微正常一点10MB根本不是事。。。家里上网都是100Mbps的带宽了），10MB可能只需要1s。所以答案是——10W的TPS，Kafka每秒钟处理了10W条消息。

Kafka是用MMAP作为文件读写方式的，它就是一个文件句柄，所以直接把它传给sendfile；偏移也好解决，用户会自己保持这个offset，每次请求都会发送这个offset。（还记得吗？放在zookeeper中的）；数据量更容易解决了，如果消费者想要更快，就全部扔给消费者。如果这样做一般情况下消费者肯定直接就被 压死了 ；所以Kafka提供了的两种方式——Push，我全部扔给你了，你死了不管我的事情；Pull，好吧你告诉我你需要多少个，我给你多少个。

缺陷：

1. 没有办法删除数据。一次Kafka是不会删除数据的，它只会把所有的数据都保留下来，每个消费者（Consumer）对每个Topic都有一个offset用来表示读取到了第几条数据
2. 不可靠，因为写到MMAP中的数据并没有被真正地写入到硬盘中，操作系统会在程序主动调用flush命令的时候才会把数据真正地写入到硬盘中

Kafka提供了一个参数prducer.type来控制是不是主动flush，如果Kafka写入到MMAP之后就立即flush然后再返回Producer，就叫做同步（sync）；如果Kafka写入到MMAP之后立即返回Producer不调用flush，就叫做异步（async）