|  |  |  |
| --- | --- | --- |
|  | RocketMq | Kafka |
| 吞吐量TPS | 大(10w级) | 极大(10w+级) |
| 开发语言 | Java | Scala |
| 集群方式 | 支持多Master-slave  分布式集群,自带namesrv管理,  非常方便扩展,且4.8以后的DLedger支持主从自动切换(启用DLedger会一定程度上影响吞吐量) | 无状态集群,每台机器既是master也是slave,依靠外部zookeeper维护集群,方便扩展  最新2.8版本自研了注册中心,但效果如何还不确定 |
| 负载均衡 | 支持非常好,由namesrv管理集群的每个成员,发送消息会均衡发送到broker们上去,消费消息时也有均衡策略 | 支持,由分区首领将任务分配到不同的broker上去 |
| 管理界面 | 有 | 有 |
| 可用性 | 非常高(分布式) | 非常高(分布式) |
| Topic数量对吞吐量的影响 | Topic达到几百上千的时候会有一定影响 | Topic达到成百上千的时候吞吐量会大幅下降 |
| 时效性 | ms | ms |
| 消息可靠性 | 配置参数可做到0丢失 | 配置参数可做到0丢失 |
| 功能特性 | 特性完备,支持如事务消息,顺序消息,死信队列,延迟消息等等 | 功能单一,只有基本发送消费,但是可以在业务代码层面实现rocket拥有的这些功能 |
| 消息存储 | 大量堆积 | 大量堆积 |
| 订阅形势和消息分发 | 基于topic以及按照topic进行正则匹配的发布订阅模式 | 基于topic以及按照topic进行正则匹配的发布订阅模式 |
| 客户端支持语言 | Java、C/C++、Go、Python | java、Python、Ruby、PHP、C#、JavaScript、Go |
| 高可用 | 依赖于NameServer的集群管理，基于Broker（RocketMQ进程）的主从复制，支持一主一从或者一主多从。RocketMQ有两种复制方式：一种是同步复制，消息同步双写到主从节点上，主从都写成功，才返回“写入成功”给客户端；另外一种是异步复制，消息发送到主节点上，就返回“写入成功”给客户端，消息再异步复制到从节点。主从节点之间不支持故障切换，只有主节点提供写入，从节点只提供消费，主节点宕机情况下只能消费不能生产消息。牺牲可用性保证数据一致性（只有硬盘损坏情况下才会丢失消息）  基于Dledger的集群，Dledger 在写入消息的时候，要求至少消息复制到半数以上的节点之后，才给客户端返回写入成功，并且支持通过选举进行故障。在选举过程中，无法集群无法对外提供服务。写入性能相较异步差，且资源利用率低。 | 依赖于Zookeeper的集群管理，复制的基本单位是分区，每个分区的副本构成复制集群，Broker只是分区的容器，不分主从关系，分区间采用一主多从配置，副本间的复制方式为异步复制，但是写入时，并不会马上返回成功，需要等待足够多的副本复制成功再返回成功，足够多个数需要自己配置，基于性能、可用性、一致性灵活取舍 |
| 数据可靠性 | RocketMQ提供同步刷盘与异步刷盘策略，同步刷盘情况下可保证数据一定不丢，异步刷盘时，如果整个集群宕机，且消息均为落盘，会出现消息丢失。 | Kafka客户端默认采取消息批量发送的方式，生产者宕机可能导致消息丢失。  分区间的刷盘方式默认为异步刷盘，依赖多个副本保证数据可靠性，可配置为同步刷盘保证绝对的数据可靠性（影响性能） |
| io | 单服务的所有消息均顺序追加在单个文件上，文件默认滚动大小为1K，不受topic与queue数量影响，由于只写一个文件，可能无法充分发挥整个硬盘的性能 | 每个分区对应一个文件，单文件上采用顺序写方式追加数据，多topic时，多文件的顺序写会演变成随机io，故性能随topic数量的增长先增后减，适当的topic数量可充分利用硬盘性能 |
| 失败重试 | 支持梯度时间级别的消息消费失败重试 | 不支持 |
| 定时消息 | 开源版支持梯度级别的延时消息  商业版支持任意精度级别的定时消息 | 不支持 |
| 事务消息 | 提供基于2PC保证最终一致性的事务消息特性 | 不支持 |

|  |  |  |
| --- | --- | --- |
| 特性 | Rocket | Kafka |
| 顺序消息 | 支持,包括Topic级别,messageQueue级别的顺序 | 没有特定的api,可以通过指定partion发送消息和单partion的topic来实现 |
| 延时消息 | 支持 | 不支持,但可以手动实现 通过实现一个代理服务(多延时级别的topic,类似rocket),处理和转发延时消息 |
| 事务消息 | 支持,属于偏向业务上的事务 | 支持类似sql的事务,即多条消息一起发,可以保证同时成功或者同时失败 可以手动实现类似于rocket的事务消息 |
| 消息过滤 | 支持按tag和sql92标准过滤 | 不支持,但可以通过细分topic来处理 |
| 死信队列 | 不支持 | 不支持,但可以手动设定一个特殊的topic作为死信队列,在业务层面手动将消息转发到死信队列 |
|  |  |  |
|  |  |  |

集群机制:

Rocket是区分broker的身份,分master和slave,master broker宕机后是从slave broker中选一个新的master,kafka的同一个brokerId的broker上的数据都是一样的,由master broker同步到slave

Kafka不区分broker的身份,但区分partion的身份,kafka的一个Topic可以有多个patrion,机制等同于rocket的messageQueue,不同的是Kafka的partion可以有多个复制partion.多partion的情况下,一个是leader,其他的都是follower.leader partion宕机后,会从多个follower中选出新的leader. Kafka的每个broker上的数据都是不一样的,数据同步是partion级的

**Kafka** 原由scala开发,现在已慢慢主要构成为Java,吞吐量极高,支持各种语言,但是功能特性单一,不支持多数高级功能,不适应复杂业务场景,主要用于日志收集与传输场景.

**RocketMq**从综合功能上来看是最强的,支持各种高级功能,高可用,高并发.支持大量消息堆积,支持主从自动切换,由java开发,易于维护和二次开发,支持java,Python,C/C++,Go,不支持ruby.

可视化工具分析:

**Rocketmq: 可视化工具为RocketMq-external(最常用的,也是功能最强的)**

1.有中英文可以选择,方便查看

2.驾驶舱可以看到broker和topic的一些统计数据,top10和最近五分钟的(柱状图及折线图)

3.可以指定集群查看集群上各个broker的统计数据,如消息生产数量,消费数量等,以及broker的状态和详细配置信息

4.可以查看所有的topic,对其进行管理和配置信息的修改(增删改查)

5.可以查看和管理所有消费者,查看每一个消费者的配置和消费数据,如消息位点重置

6.可以按主题筛选所有的生产者

7.可以按主题,时间段,messageKey,MessageId筛选查看所有的消息

**Kafka: 可视化工具为kafkaManager(最常用的)**

1.管理多个集群

2.轻松检查群集状态（主题，消费者，偏移，代理，副本分发，分区分发）

3.运行首选副本选举

4.使用选项生成分区分配以选择要使用的代理

5.运行分区重新分配（基于生成的分配）

6.使用可选主题配置创建主题（0.8.1.1具有与0.8.2+不同的配置）

7.删除主题（仅支持0.8.2+并记住在代理配置中设置delete.topic.enable = true）

8.主题列表现在指示标记为删除的主题（仅支持0.8.2+）

9.批量生成多个主题的分区分配，并可选择要使用的代理

10.批量运行重新分配多个主题的分区

11.将分区添加到现有主题

12.更新现有主题的配置

消息队列高级功能特性与使用场景分析:

**1.消息重试机制**:在响应端(如消费者)返回消息重试的响应后(或者没有返回ack),消息队列会按照相应的重试规则进行重投

**2.事务消息:**

使用场景比如AB转账问题,这种不追求强一致性只需最终一致性的场景,非常适合事务消息

开始执行A扣钱逻辑,并同时发送B加钱的消息到broker,此时发送的是半消息,等到本地事务执行成功(rocket中事务状态会定时回查),才会发送确认给broker使之前给B加钱的消息生效,保证最终一致性.

类似X/Open XA的分布式事务功能,以达到事务最终一致性状态.比如在rocketmq中,如果选择发送事务消息,那发送之后,异步线程去处理对应事务定义的逻辑,同时会发送一条半消息(消费者看不到的消息)到broker,如果事务逻辑处理成功,就会再发送一条半消息使之前的半消息变成正常可以被消费的消息,如果事务逻辑处理失败,经历一定重试次数后,会删掉之前那条半消息,也就是如果事务逻辑处理失败了,那么就会回滚发送消息的逻辑.

**3.定时消息/延时消息:** 在指定的时间才发出消息

适用于订单超时取消的场景,避免了定时扫描数据库给数据库和服务器的压力,将压力转移到MQ上.也无需手写定时器,降低了业务复杂度

**4.顺序消息:** 需要严格按照顺序消费的场景,比如对数据库的操作,如果sql执行顺序混乱可能会造成数据与预期不符,这种情况需要严格保证消息消费的顺序性.

**5.消息过滤:** 可以精确按需获取细分模块下的消息

**6.死信队列:**

1.如电商场景中订单超时自动取消,一般来说是设置定时任务去轮询,或者直接延迟队列去做,但是这在数据特别大的情况下对服务器压力都很大.

死信队列用法-->用户提交订单后,发送一条消息并设置过期时间为半个小时,如果超时这条信息就会变成死信,并被转发到死信队列中.此时可以监听这条死信队列,

然后查询订单状态,如果还是未支付则直接取消订单.

1. 正常消息在被消费时程序出现异常,一直消费失败,此时消息就会转向死信队列,这样有助于排查异常问题,也保证数据不会丢失.