		kafka	RocketMQ	RabbitMQ
定位	设计定位	系统间的数据流管道，实时数据处理。 例如：常规的消息系统、网站活性跟踪，监控数据，志收集、处理等	非日志的可靠消息传输。 例如：订单，交易，充值，流计算，消息推送，日流式处理，binglog分发等	可靠消息传输，和RocketMQ类似。
基础对比	成熟度	日志领域成熟	成熟	成熟
	所属社区 / 公司	Apache	Alibaba开发，已加入到Apache下	Mozilla Public License
	社区活跃度	高	中	高
	API完备性	高	高	高
	文档完备性	高	高	高
	开发语言	Scala	Java	Erlang
	支持协议	一套自行设计的基于TCP的二进制协议	自己定义的一套 (社区提供 JMS一不成熟)	AMQP
	客户端语言	C/C++、Python、Go、Erlang、.NET、Ruby、Node.js、PHP等	Java	Java、C、 C++、 Python、 PHP、Perl 等
	持久化方式	磁盘文件	磁盘文件	内存、文件
可用性、靠性比较	部署方式	单机 / 集群	单机 / 集群	单机 / 集群
	集群管理	zookeeper	name server	
	选主方式	从ISR中自动选举一个leader	不支持自动选主。通过设定brokername、brokerid实现，brokername相同，brokerid=0时为maser，其他为slave	最早加入集群的broker
	可用性	非常高 分布式、主从	非常高 分布式、主从	高 主从，采用镜像模式实现，数据量大时可能产生性能瓶颈
	主从切换	自动切换 N个副本，允许N-1个失效；master失效以后自动从ISR中选择一个主；	不支持自动切换 master失效以后不能向master发送信息，consumer大概30s（默认）可以感知此事件，此后从slave消费；如果master无法恢复，异步复制时可能出现消分信息丢失	自动切换 最早加入集群的slave会成为master；因为新加入的slave不会同步master之前的数据，所以可能会出部分数据丢失
	数据可靠性	很好 支持producer单条发送、同步刷盘、同步复制，但这场景下性能明显下降。	很好 producer单条发送，broker端支持同步刷盘、异步刷盘，同步双写，异步复制。	好 producer支持同步 / 异步ack，支持队列数据持久化，镜像模式中支持主从同步
	消息写入性能	非常好 每条10个字节测试：百万条/s	很好 每条10个字节测试：单机单broker约7w/s，单机3broker约12w/s	RAM约为RocketMQ的1/2， Disk的性能约为RAM性能的1/3
	性能的稳定性	队列/分区多时性能不稳定，明显下降。 消息堆积时性能稳定	队列较多、消息堆积时性能稳定	消息堆积时，性能不稳定、明显下降
	单机支持的队列数	单机超过64个队列/分区，Load会发生明显的飙高现象，队列越多，load越高，发送消息响应时间变长	单机支持最高5万个队列，Load不会发生明显变化	依赖于内存
	堆积能力	非常好 消息存储在log中，每个分区一个log文件	非常好 所有消息存储在同一个commit log中	一般 生产者、消费者正常时，性能表现稳定；消费者不消费时，性能不稳定
	复制备份	消息先写入leader的log，followers从leader中pull，pull到数据以后先ack leader，然后写入log中。 ISR中维护与leader同步的列表，落后太多的follwer被删除掉	同步双写 异步复制：slave启动线程从master中拉数据	普通模式下不复制； 镜像模式下：消息先到master，然后写到slave上。入集群之前的消息不会被复制到新的slave上。
	消息投递实时性	毫秒级 具体由consumer轮询间隔时间决定	毫秒级 支持pull、push两种模式，延时通常在毫秒级	毫秒级
功能对比	顺序消费	支持顺序消费 但是一台Broker宕机后，就会产生消息乱序	支持顺序消费 在顺序消息场景下，消费失败时消费队列将会暂停	支持顺序消费 但是如果一个消费失败，此消息的顺序会被打乱
	定时消息	不支持	开源版本仅支持定时Level	不支持
	事务消息	不支持	支持	不支持
	Broker端消息过滤	不支持	支持 通过tag过滤，类似于子topic	不支持
	消息查询	不支持	支持 根据MessageId查询 支持根据MessageKey查询消息	不支持
	消费失败重试	不支持失败重试 offset存储在consumer中，无法保证。 0.8.2版本后支持将offset存储在zk中	支持失败重试 offset存储在broker中	支持失败重试
	消息重新消费	支持通过修改offset来重新消费	支持按照时间来重新消息	
	发送端负载均衡	可自由指定	可自由指定	需要单独loadbalancer支持
	消费并行度	消费并行度和分区数一致	顺序消费：消费并行度和分区数一致 乱序消费：消费服务器的消费线程数之和	镜像模式下其实也是从master消费
	消费方式	consumer pull	consumer pull / broker push	broker push
	批量发送	支持 默认producer缓存、压缩，然后批量发送	不支持	不支持
	消息清理	指定文件保存时间，过期删除	指定文件保存时间，过期删除	可用内存少于40%（默认），触发gc，gc时找到相同的两个文件，合并right文件到left。
	访问权限控制	无	无	类似数据库一样，需要配置用户名密码
运维	系统维护	Scala语言开发，维护成本高	java语言开发，维护成本低	Erlang语言开发，维护成本高
	部署依赖	zookeeper	nameserver	Erlang环境
	管理后台	官网不提供，第三方开源管理工具可供使用；不用重新开发	官方提供，rocketmq-console	官方提供rabbitmqadmin
	管理后台功能	**Kafka Web Conslole** Brokers列表：Kafka 集群中 Topic列表，及对应的Partition、LogSize等信息；Topic对应的Consumer Groups、Offset、Lag等信息； 生产和消费流量图、消息预览 **KafkaOffsetMonitor：** Kafka集群状态：Topic、Consumer Group列表；图形展示topic和consumer之间的关系；图形化展示consumer的Offset、Lag等信息 **Kafka Manager** 管理几个不同的集群；监控集群的状态(topics, brokers, 副本分布, 分区分布)；产生分区分配(Generate partition assignments)基于集群的当前态，重新分配分区	Cluster、Topic、Connection、NameServ、Messa Broker、Offset、Consumer	overview、connections、channels、exchanges、queues、admin
总结	优点	1、在高吞吐、低延迟、高可用、集群热扩展、集群容错上有非常好的表现； 2、producer端提供缓存、压缩功能，可节省性能，提高效率， 3、提供顺序消费能力 4、提供多种客户端语言 5、生态完善，在大数据处理方面有大量配套的设施。	1、在高吞吐、低延迟、高可用上有非常好的表现 消息堆积时，性能也很好。 2、api、系统设计都更加适在业务处理的场景。 3、支持多种消费方式。 4、支持broker消息过滤。 5、支持事务。 6、提供消息顺序消费能力；consumer可以水平扩展，消费能力很强。 7、集群规模在50台左右，单日处理消息上百亿；历经过大数据量的考验，比较稳定可靠。	1、在高吞吐量、高可用上较前两者有所不如。 2、支持多种客户端语言；支持amqp协议。 3、由于erlang语言的特性，性能也比较好；使用RAM模式时，性能很好。 4、管理界面较丰富，在互联网公司也有较大规模的应用。
	缺点	1、消费集群数目受到分区数目的限制。 2、单机topic多时，性能会明显降低。 3、不支持事务	1、相比于kafka，使用者较少，生态不够完善。消息堆积、吞吐率上也有所不如。 2、不支持主从自动切换，master失效后，消费者要一定的时间才能感知。 3、客户端只支持java	1、erlang 语言难度较大，集群不支持动态扩展。 2、不支持事务、消息吞吐能力有限 3、消息堆积时，性能会明显降低