# 简单分析

吞吐量  
kafka吞吐量更高：  
　　1）Zero Copy机制，内核copy数据直接copy到网络设备，不必经过内核到用户再到内核的copy，减小了copy次数和上下文切换次数，大大提高了效率。  
　　2）磁盘顺序读写，减少了寻道等待的时间。  
　　3）批量处理机制，服务端批量存储，客户端主动批量pull数据，消息处理效率高。  
　　4）存储具有O(1)的复杂度，读物因为分区和segment，是O(log(n))的复杂度。  
　　5）分区机制，有助于提高吞吐量。  
可靠性  
rabbitmq可靠性更好：  
　　1）确认机制（生产者和exchange，消费者和队列）;  
　　2）支持事务，但会造成阻塞;  
　　3）委托（添加回调来处理发送失败的消息）和备份交换器（将发送失败的消息存下来后面再处理）机制;  
高可用  
　　1）rabbitmq采用mirror queue，即主从模式，数据是异步同步的，当消息过来，主从全部写完后，回ack，这样保障了数据的一致性。  
　　2）每个分区都可以有一个或多个副本，这些副本保存在不同的broker上，broker信息存储在zookeeper上，当broker不可用会重新选举leader。  
　　kafka支持同步负载消息和异步同步消息（有丢消息的可能），生产者从zk获取leader信息，发消息给leader，follower从leader pull数据然后回ack给leader。  
**负载均衡**  
　　1）kafka通过zk和分区机制实现：zk记录broker信息，生产者可以获取到并通过策略完成负载均衡；通过分区，投递消息到不同分区，消费者通过服务组完成均衡消费。  
　　2）rabbitmq需要外部支持。  
模型  
　**1）rabbitmq：**  
　　　　producer,broker遵循AMQP（exchange，bind，queue），consumer;  
　　　　broker为中心，exchange分topic,direct,fanout和header，路由模式适合多种场景;  
　　　　consumer消费位置由broker通过确认机制保存;  
**2）kafka：**  
　　　　producer，broker，consumer，未遵循AMQP;  
　　　　consumer为中心，获取消息模式由consumer自己决定;  
　　　　offset保存在消费者这边，broker无状态;  
　　　　消息是名义上的永久存储，每个parttition按segment保存自己的消息为文件（可配置清理周期）；  
　　　　consumer可以通过重置offset消费历史消息;  
　　　　需要绑定zk;

# **kafka和rabbitmq全面对比分析**

|  |  |  |
| --- | --- | --- |
| **对比项** | **kafka** | **rabbitmq** |
| 开发语言 | scala,Java | erlang |
| 是否支持多租户 | 2.x.x支持多租户 | 支持多租户 |
| 是否支持topic优先级 | 不支持 | 支持 |
| 是否支持消息全局有序 | 不支持 | 支持 |
| 是否支持消息分区有序 | 支持 | 支持 |
| 是否内置监控 | 无内置监控 | 内置监控 |
| 是否支持多个生产者 | 一个topic支持多个生产者 |  |
| 是否支持多个消费者 | 一个topic支持多个消费者 |  |
| 是否支持一个分区多个消费者 | 不支持 | 不支持 |
| 是否支持JMX | 支持 | 不支持(非java语言编写) |
| 是否支持加密 | 支持 | 支持 |
| 消息队列协议支持 | 仅支持自定义协议 | 支持AMQP、MQTT、STOMP协议 |
| 客户端语言支持 | 支持多语言客户端 | 支持多语言客户端 |
| 是否支持消息追踪 | 不支持消息追踪 | 支持消息追踪 |
| 是否支持消费者推模式 | 不支持消费者推模式 | 支持消费者推模式 |
| 是否支持消费者拉模式 | 支持消费者拉模式 | 支持消费者拉模式 |
| 是否支持广播消息 | 支持广播消息 | 支持广播消息 |
| 是否支持消息回溯 | 支持消息回溯，因为消息持久化，消息被消费后会记录offset和timstamp | 不支持，消息确认被消费后，会被删除 |
| 是否支持消息数据持久化 | 支持消息数据持久 | 支持消息数据持久 |
| 是否支持消息堆积 | 支持消息堆积，并批量持久化到磁盘 | 支持阈值内的消息对接，无法支持较大的消息堆积 |
| 是否支持流量控制 | 支持控制用户和客户端流量 | 支持生产者的流量控制 |
| 是否支持事务性消息 | 支持 | 不支持 |
| 元数据管理 | 通过zookeeper进行管理 | 支持消息数据持久 |
| 默认服务端口 | 9200 | 5672 |
| 默认监控端口 | kafka web console 9000;kafka manager 9000; | 15672 |
| 网络开销 | 相对较小 | 相对较大 |
| 内存消耗 | 相对较小 | 相对较大 |
| cpu消耗 | 相对较大 | 相对较小 |

# 总结

在实际生产应用中，通常会使用kafka作为消息传输的数据管道，rabbitmq作为交易数据作为数据传输管道，主要的取舍因素则是是否存在丢数据的可能；rabbitmq在金融场景中经常使用，具有较高的严谨性，数据丢失的可能性更小，同事具备更高的实时性；而kafka优势主要体现在吞吐量上，虽然可以通过策略实现数据不丢失，但从严谨性角度来讲，大不如rabbitmq；而且由于kafka保证每条消息最少送达一次，有较小的概率会出现数据重复发送的情况;