

Kafka是linkedin开源的MQ系统，主要特点是基于Pull的模式来处理消息消费，追求高吞吐量，一开始的目的就是用于日志收集和传输，0.8开始支持复制，不支持事务，适合产生大量数据的互联网服务的数据收集业务。

Kafka是Apache下的一个子项目，是一个高性能跨语言分布式发布/订阅消息队列系统，而Jafka是在Kafka之上孵化而来的，即Kafka的一个升级版。

具有以下特性：

快速持久化，可以在O(1)的系统开销下进行消息持久化；

高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统，Broker、Producer、Consumer都原生自动支持分布式，自动实现负载均衡；

支持Hadoop数据并行加载，对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。

Kafka通过Hadoop的并行加载机制统一了在线和离线的消息处理。Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。

RabbitMQ是使用Erlang语言开发的开源消息队列系统，基于AMQP协议来实现。AMQP的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。AMQP协议更多用在企业系统内，对数据一致性、稳定性和可靠性要求很高的场景，对性能和吞吐量的要求还在其次。

结合erlang语言本身的并发优势，支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正是如此，使的它变的非常重量级，更适合于企业级的开发。

性能较好，但是不利于做二次开发和维护。

RocketMQ是阿里开源的消息中间件，它是纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。RocketMQ思路起源于Kafka，但并不是Kafka的一个Copy，它对消息的可靠传输及事务性做了优化，目前在阿里集团被广泛应用于交易、充值、流计算、消息推送、日志流式处理、binglog分发等场景。

Redis作为一个基于内存的K-V数据库，其提供了消息订阅的服务，可以当作MQ来使用，目前应用案例较少，且不方便扩展。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。

入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于 Redis。

ActiveMQ历史悠久的开源项目，是Apache下的一个子项目。已经在很多产品中得到应用，实现了JMS1.1规范，可以和spring-jms轻松融合，实现了多种协议，不够轻巧（源代码比RocketMQ多），支持持久化到数据库，对队列数较多的情况支持不好。

ZeroMQ只是一个网络编程的Pattern库，将常见的网络请求形式（分组管理，链接管理，发布订阅等）模式化、组件化，简而言之socket之上、MQ之下。对于MQ来说，网络传输只是它的一部分，更多需要处理的是消息存储、路由、Broker服务发现和查找、事务、消费模式（ack、重投等）、集群服务等。

号称最快的消息队列系统，尤其针对大吞吐量的需求场景。

扩展性好，开发比较灵活，采用C语言实现，实际上只是一个socket库的重新封装，如果做为消息队列使用，需要开发大量的代码。ZeroMQ仅提供非持久性的队列，也就是说如果down机，数据将会丢失。其中，Twitter的Storm中使用ZeroMQ作为数据流的传输。

RabbitMQ/Kafka/ZeroMQ 都能提供消息队列服务，但有很大的区别。

在面向服务架构中通过消息代理（比如 RabbitMQ / Kafka等），使用生产者-消费者模式在服务间进行异步通信是一种比较好的思想。

因为服务间依赖由强耦合变成了松耦合。消息代理都会提供持久化机制，在消费者负载高或者掉线的情况下会把消息保存起来，不会丢失。就是说生产者和消费者不需要同时在线，这是传统的请求-应答模式比较难做到的，需要一个中间件来专门做这件事。其次消息代理可以根据消息本身做简单的路由策略，消费者可以根据这个来做负载均衡，业务分离等。

缺点也有，就是需要额外搭建消息代理集群（但优点是大于缺点的 ） 。

ZeroMQ 和 RabbitMQ/Kafka 不同，它只是一个异步消息库，在套接字的基础上提供了类似于消息代理的机制。使用 ZeroMQ 的话，需要对自己的业务代码进行改造，不利于服务解耦。

RabbitMQ 支持 AMQP（二进制），STOMP（文本），MQTT（二进制），HTTP（里面包装其他协议）等协议。Kafka 使用自己的协议。

Kafka 自身服务和消费者都需要依赖 Zookeeper。

RabbitMQ 在有大量消息堆积的情况下性能会下降，Kafka不会。毕竟AMQP设计的初衷不是用来持久化海量消息的，而Kafka一开始是用来处理海量日志的。

总的来说，RabbitMQ 和 Kafka 都是十分优秀的分布式的消息代理服务，只要合理部署，不作，基本上可以满足生产条件下的任何需求。

关于这两种MQ的比较，网上的资料并不多，最权威的是kafka的提交者写一篇文章。http://www.quora.com/What-are-the-differences-between-Apache-Kafka-and-RabbitMQ

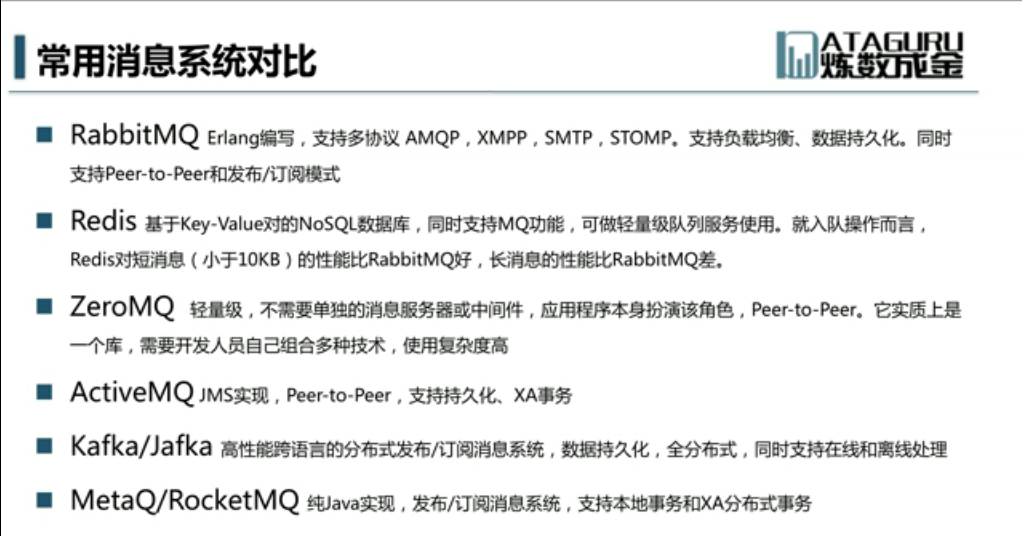
里面提到的要点：

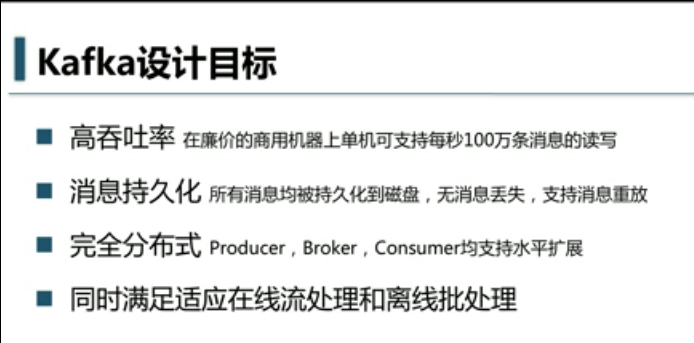
1、 RabbitMq比kafka成熟，在可用性上，稳定性上，可靠性上，RabbitMq超过kafka

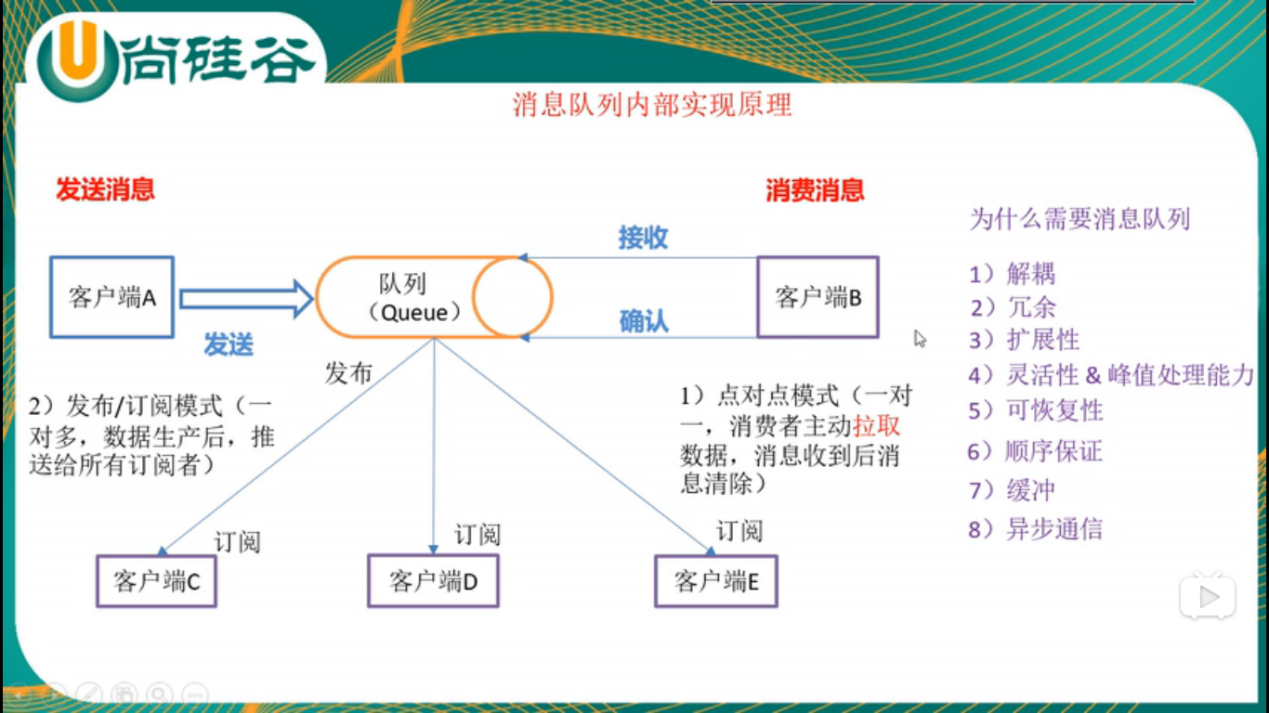
2、 Kafka设计的初衷就是处理日志的，可以看做是一个日志系统，针对性很强，所以它并没有具备一个成熟MQ应该具备的特性

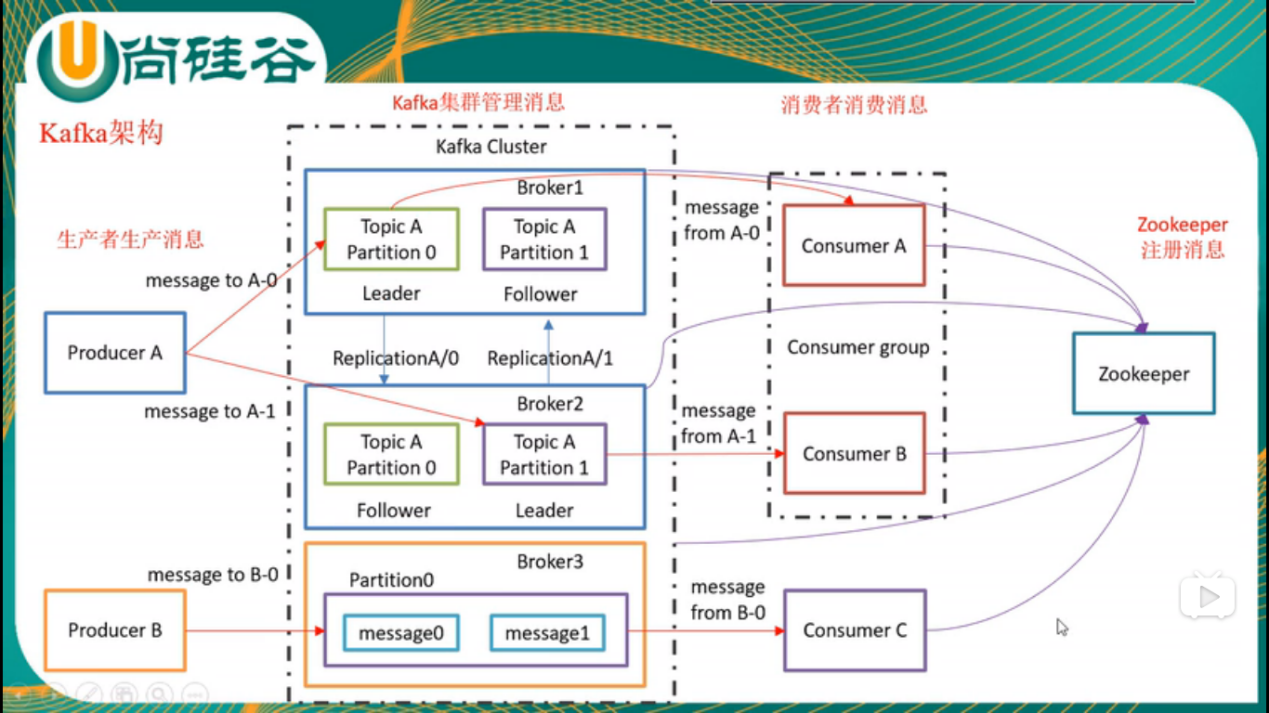
3、 Kafka的性能（吞吐量、tps）比RabbitMq要强，这篇文章的作者认为，两者在这方面没有可比性。

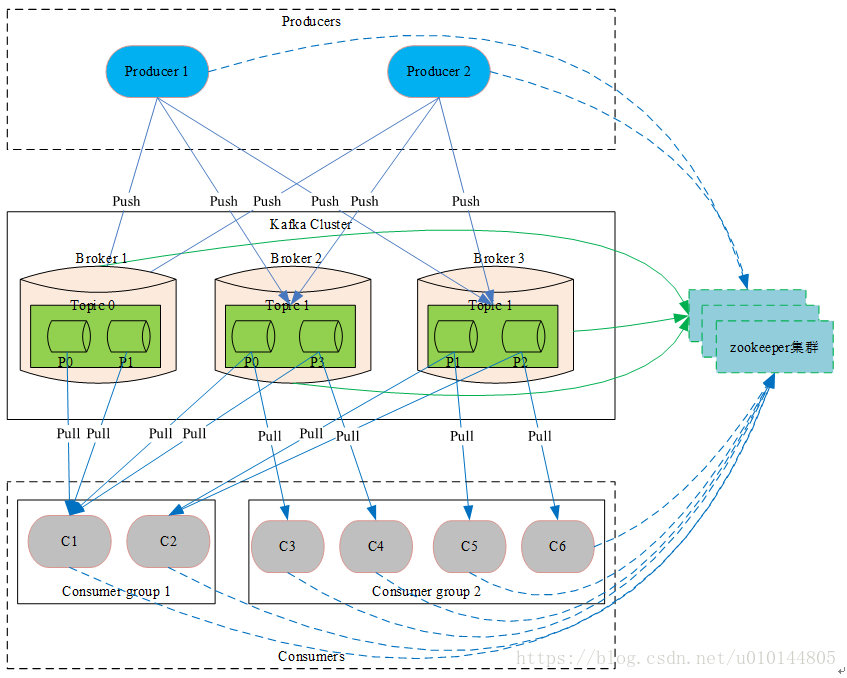


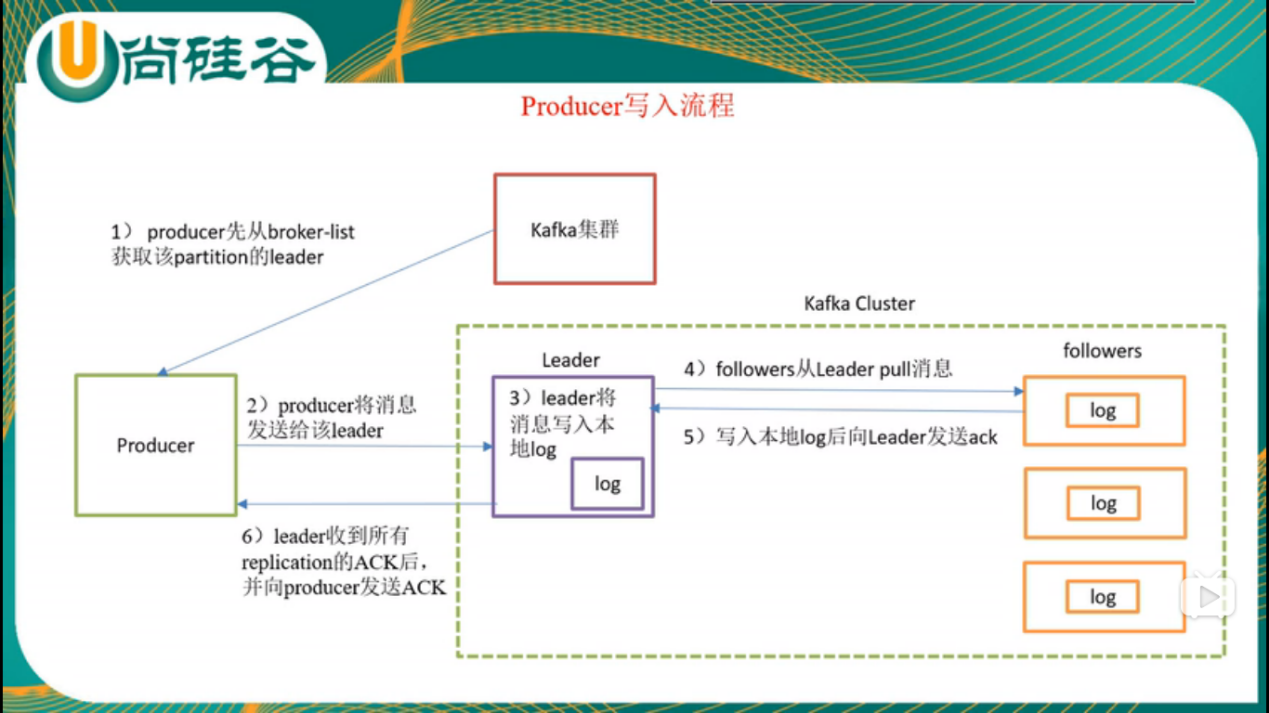












Kafka的ack机制，指的是producer的消息发送确认机制，这直接影响到Kafka集群的吞吐量和消息可靠性。而吞吐量和可靠性就像硬币的两面，两者不可兼得，只能平衡。

ack有3个可选值，分别是1，0，-1。

ack=1，简单来说就是，producer只要收到一个分区副本成功写入的通知就认为推送消息成功了。这里有一个地方需要注意，这个副本必须是leader副本。只有leader副本成功写入了，producer才会认为消息发送成功。

注意，ack的默认值就是1。这个默认值其实就是吞吐量与可靠性的一个折中方案。生产上我们可以根据实际情况进行调整，比如如果你要追求高吞吐量，那么就要放弃可靠性。

ack=0，简单来说就是，producer发送一次就不再发送了，不管是否发送成功。ack=-1，简单来说就是，producer只有收到分区内所有副本的成功写入的通知才认为推送消息成功了。