<https://www.cnblogs.com/yangxiaoyi/p/7359236.html>

 kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理消费者规模的网站中的所有动作流数据。这种动作（网页浏览，搜索和其他用户的行动）是在现代网络上的许多社会功能的一个关键因素。这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。

**1.1 kafka名词解释**

producer：生产者。

consumer：消费者。

topic: 消息以topic为类别记录,Kafka将消息种子(Feed)分门别类,每一类的消息称之为一个主题(Topic)。

broker：以集群的方式运行,可以由一个或多个服务组成，每个服务叫做一个broker;消费者可以订阅一个或多个主题(topic),并从Broker拉数据,从而消费这些已发布的消息。

      每个消息（也叫作record记录,也被称为消息）是由一个key，一个value和时间戳构成。

**1.2 kafka有四个核心API介绍**

应用程序使用producer API发布消息到1个或多个topic中。

应用程序使用consumer API来订阅一个或多个topic,并处理产生的消息。

应用程序使用streams API充当一个流处理器,从1个或多个topic消费输入流,并产生一个输出流到1个或多个topic,有效地将输入流转换到输出流。

connector API允许构建或运行可重复使用的生产者或消费者,将topic链接到现有的应用程序或数据系统。

**1.3 kafka基基原理**

       通常来讲，消息模型可以分为两种：队列和发布-订阅式。队列的处理方式是一组消费者从服务器读取消息，一条消息只有其中的一个消费者来处理。在发布-订阅模型中，消息被广播给所有的消费者，接收到消息的消费者都可以处理此消息。Kafka为这两种模型提供了单一的消费者抽象模型： 消费者组(consumer group)。消费者用一个消费者组名标记自己。

       一个发布在Topic上消息被分发给此消费者组中的一个消费者。假如所有的消费者都在一个组中，那么这就变成了queue模型。假如所有的消费者都在不同的组中，那么就完全变成了发布-订阅模型。更通用的， 我们可以创建一些消费者组作为逻辑上的订阅者。每个组包含数目不等的消费者，一个组内多个消费者可以用来扩展性能和容错。

       并且，kafka能够保证生产者发送到一个特定的Topic的分区上，消息将会按照它们发送的顺序依次加入，也就是说，如果一个消息M1和M2使用相同的producer发送，M1先发送，那么M1将比M2的offset低，并且优先的出现在日志中。消费者收到的消息也是此顺序。如果一个Topic配置了复制因子（replication facto）为N,那么可以允许N-1服务器宕机而不丢失任何已经提交（committed）的消息。此特性说明kafka有比传统的消息系统更强的顺序保证。但是，相同的消费者组中不能有比分区更多的消费者，否则多出的消费者一直处于空等待，不会收到消息。

**1.4 kafka应用场景**

       构建实时的流数据管道，可靠地获取系统和应用程序之间的数据。

       构建实时流的应用程序，对数据流进行转换或反应。

**1.5 主题和日志 (Topic和Log)**

      每一个分区(partition)都是一个顺序的、不可变的消息队列,并且可以持续的添加。分区中的消息都被分了一个序列号,称之为偏移量(offset),在每个分区中此偏移量都是唯一的。Kafka集群保持所有的消息,直到它们过期,无论消息是否被消费了。实际上消费者所持有的仅有的元数据就是这个偏移量，也就是消费者在这个log中的位置。 这个偏移量由消费者控制：正常情况当消费者消费消息的时候，偏移量也线性的的增加。但是实际偏移量由消费者控制，消费者可以将偏移量重置为更老的一个偏移量，重新读取消息。 可以看到这种设计对消费者来说操作自如， 一个消费者的操作不会影响其它消费者对此log的处理。 再说说分区。Kafka中采用分区的设计有几个目的。一是可以处理更多的消息，不受单台服务器的限制。Topic拥有多个分区意味着它可以不受限的处理更多的数据。第二，分区可以作为并行处理的单元，稍后会谈到这一点。

**1.6 分布式(Distribution)**

       Log的分区被分布到集群中的多个服务器上。每个服务器处理它分到的分区。根据配置每个分区还可以复制到其它服务器作为备份容错。 每个分区有一个leader，零或多个follower。Leader处理此分区的所有的读写请求，而follower被动的复制数据。如果leader宕机，其它的一个follower会被推举为新的leader。 一台服务器可能同时是一个分区的leader，另一个分区的follower。 这样可以平衡负载，避免所有的请求都只让一台或者某几台服务器处理。