11

# Kafka

- Kafka概述
- 消息系统
- Kafka组件

# 概述

Kafka最初是由Linkedin公司开发，是一个分布式、分区的、多副本的、多订阅者，基于Zookeeper协调的分布式日志系统（也可以当做MQ（消息队列）系统），常见可以用于web/nginx日志、访问日志，消息服务等等，Linkedin于2010年贡献给了Apache基金会并成为顶级开源项目。

主要应用场景是：日志收集系统和消息系统。

# 应用场景

- 消息投递：能够很好的代替传统的message broker。它提供了更强大的吞吐量，内建分区，复本，容错等机制来解决大规模消息处理型应用程序。
- 用户活动追踪：通过按类型将每个web动作发送到指定topic，然后由消费者去订阅各种topic加以处理，包括实时处理，实时监控，加载到Hadoop或其它离线存储系统用于离线处理等。
- 日志聚合：把物理上分布在各个机器上的离散日志数据聚集到指定区域（如HDFS或文件服务器等）用来处理。

# 设计目标

- 以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上数据也能保证常数时间的访问性能。
- 高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒100K条消息的传输。
- 支持Kafka Server间的消息分区，及分布式消费，同时保证每个partition内的消息顺序传输。
- 同时支持离线数据处理和实时数据处理。
- 支持在线水平扩展

# 消息系统

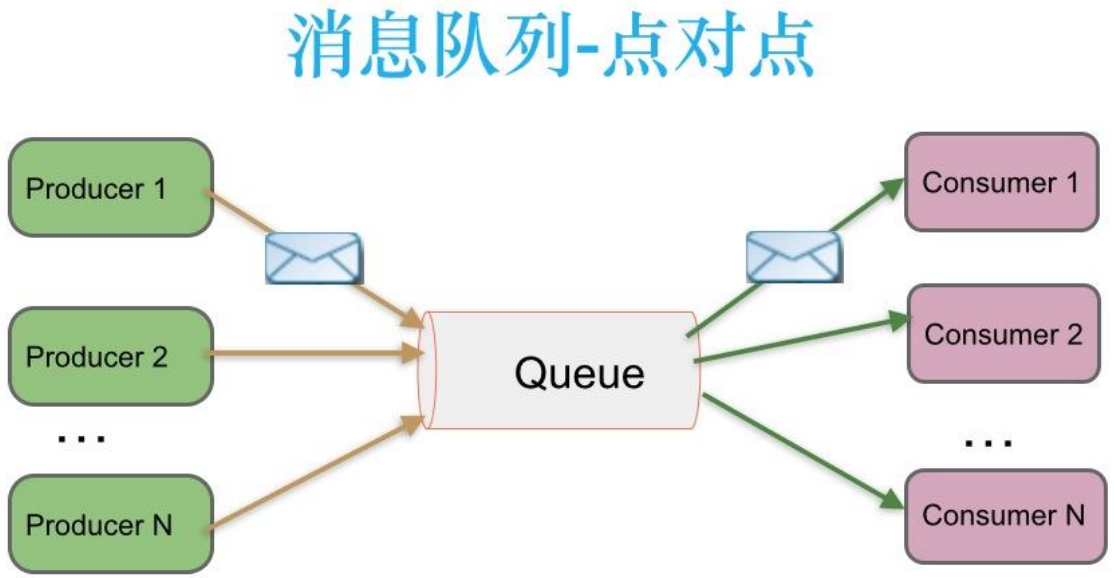
一个消息系统负责将数据从一个应用传递到另外一个应用，应用只需关注于数据，无需关注数据在两个或多个应用间是如何传递的。分布式消息传递基于可靠的消息队列，在客户端应用和消息系统之间异步传递消息。有两种主要的消息传递模式：

- 点对点传递模式
- 发布-订阅模式

大部分的消息系统选用发布-订阅模式。Kafka就是一种发布-订阅模式。

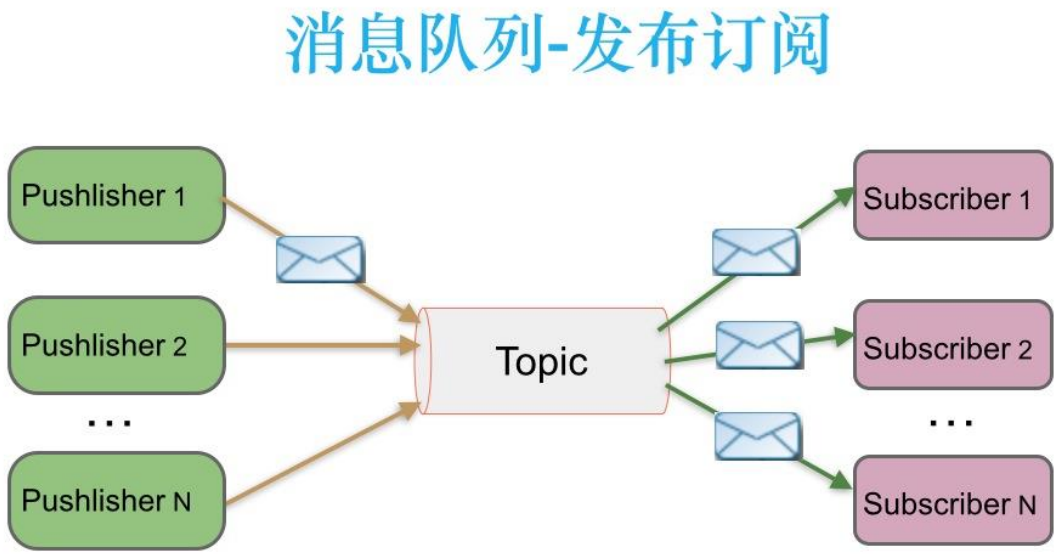
# 点对点消息传递模式

在点对点消息系统中，消息持久化到一个队列中。此时，将有一个或多个消费者消费队列中的数据。但是一条消息只能被消费一次。当一个消费者消费了队列中的某条数据之后，该条数据则从消息队列中删除。该模式即使有多个消费者同时消费数据，也能保证数据处理的顺序。



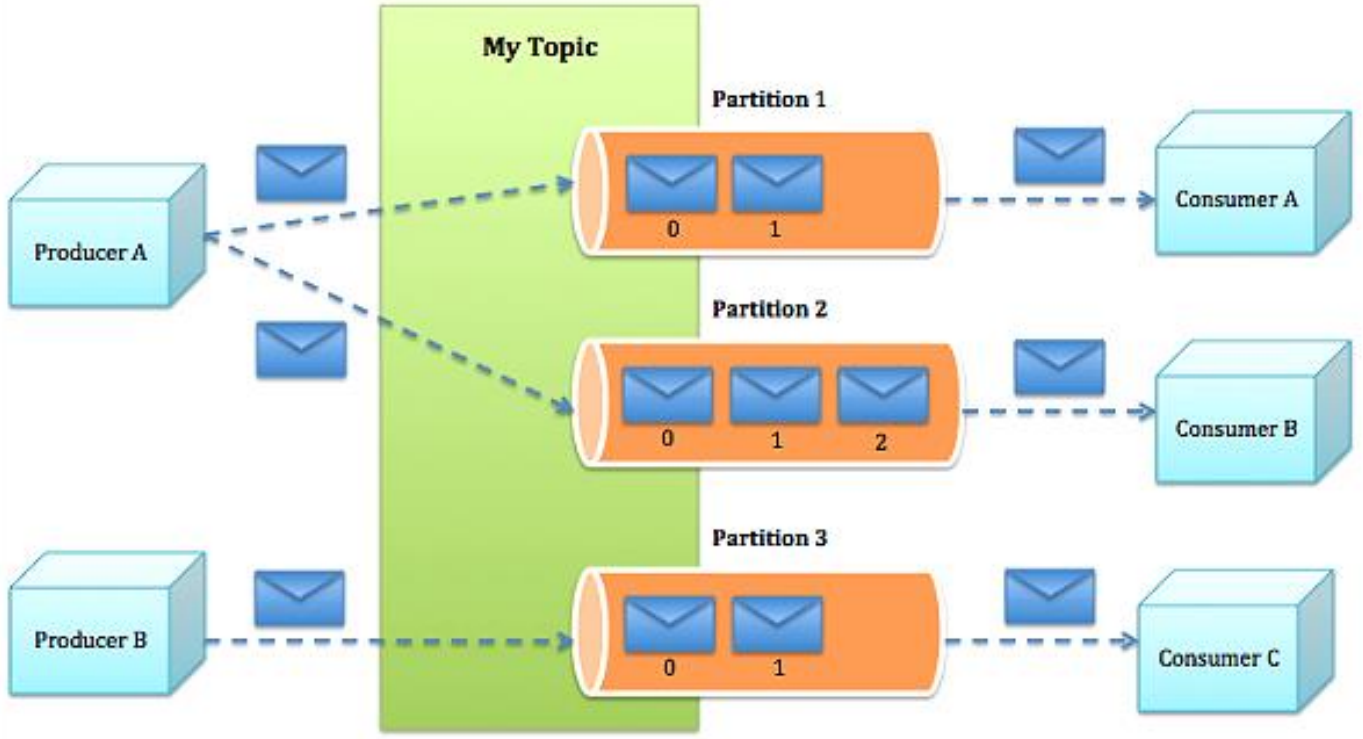
# 发布-订阅消息传递模式

在发布-订阅消息系统中，消息被持久化到一个topic中。与点对点消息系统不同的是，消费者可以订阅一个或多个topic，消费者可以消费该topic中所有的数据，同一条数据可以被多个消费者消费，数据被消费后不会立马删除。在发布-订阅消息系统中，消息的生产者称为发布者，消费者称为订阅者。

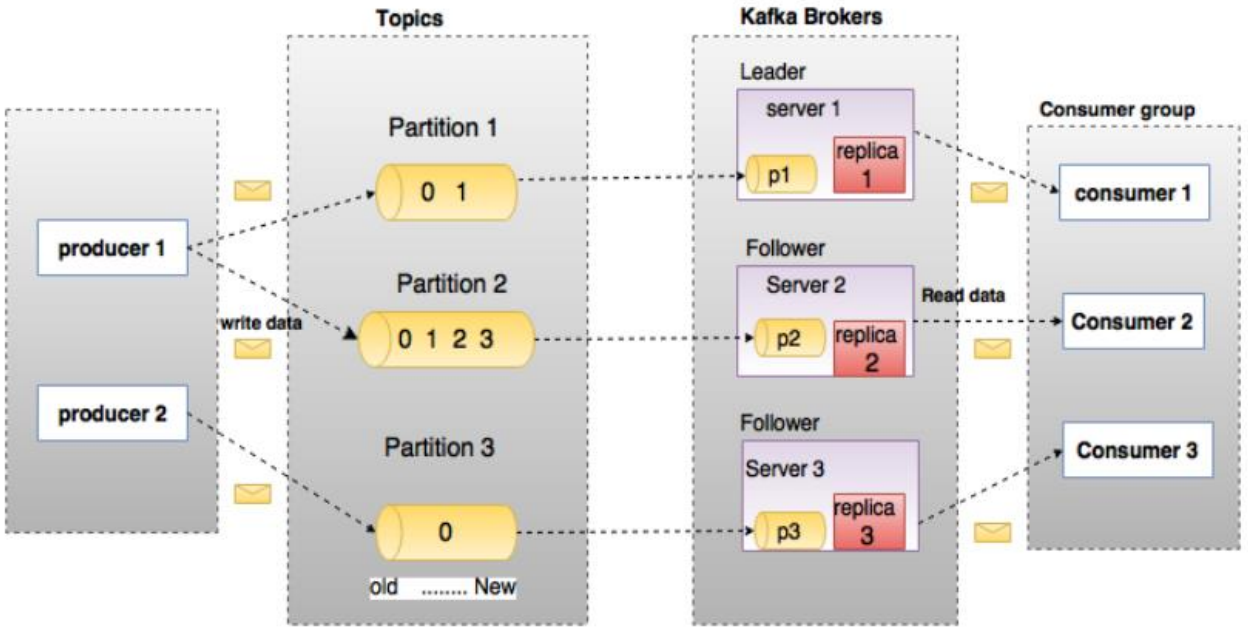


# Kafka组件

Kafka内是分布式的，一个Kafka集群通常包括多个broker。为了均衡负载，将话题分成多个分区，每个broker存储一或多个分区。多个生产者和消费者能够同时生产和获取消息。



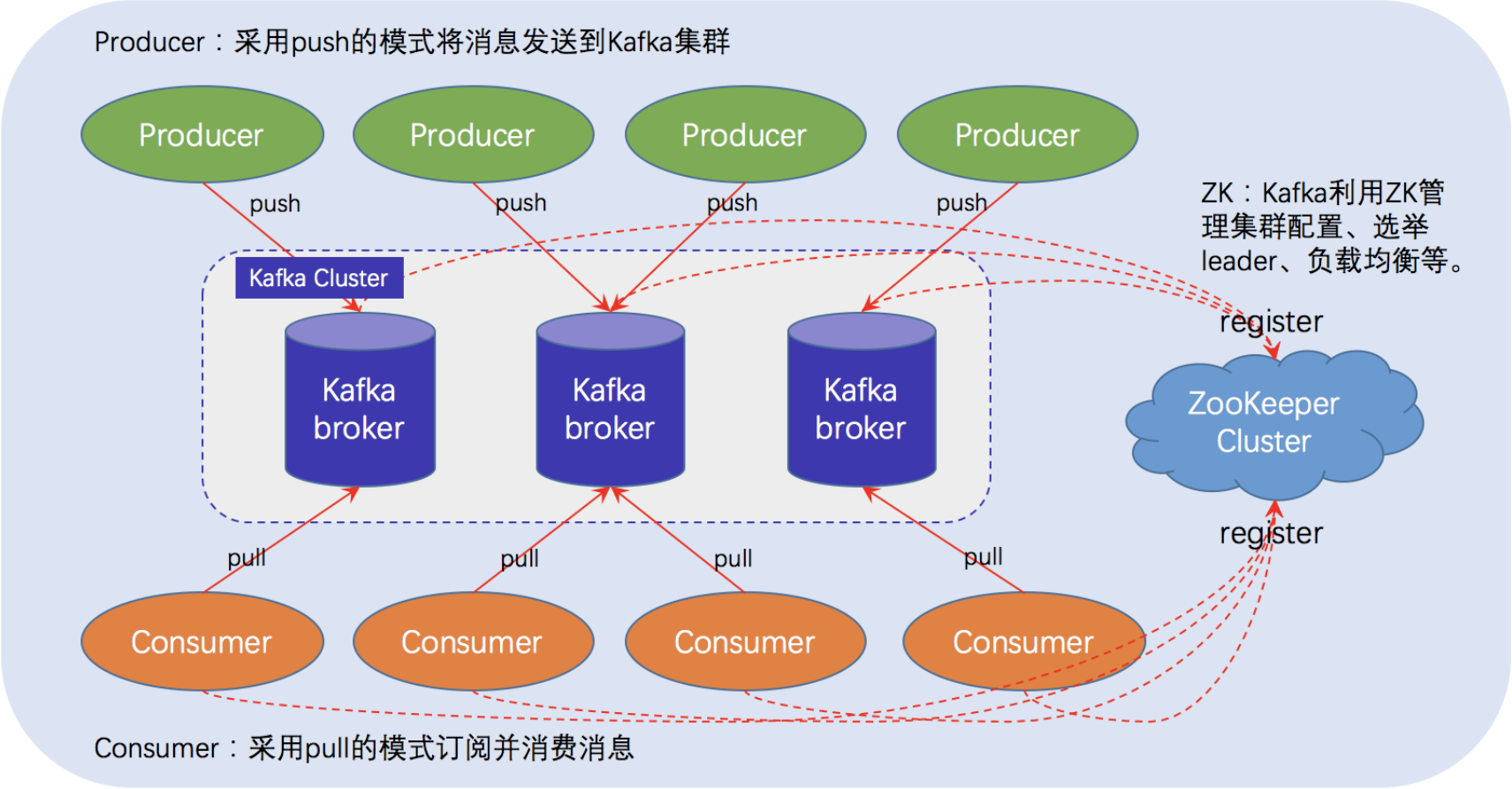
# Kafka组件




一个topic配置了3个partition。Partition1有两个offset：0和1。Partition2有4个offset。Partition3有1个offset。副本的id和副本所在的机器的id恰好相同。

如果一个topic的副本数为3，那么Kafka将在集群中为每个partition创建3个相同的副本。集群中的每个broker存储一个或多个partition。多个producer和consumer可同时生产和消费数据。

# Kafka组件


Producer：采用push的模式将消息发送到Kafka集群
Producer
Producer
Producer
Producer
push
push
push
push
Kafka Cluster
Kafka broker
Kafka broker
Kafka broker
ZK：Kafka利用ZK管理集群配置、选举leader、负载均衡等。
register
ZooKeeper Cluster
register
pull
pull
pull
pull
Consumer
Consumer
Consumer
Consumer
Consumer：采用pull的模式订阅并消费消息

# Producers

生产者即数据的发布者，该角色将消息发布到Kafka的topic中。Broker接收到生产者发送的消息后，broker将该消息追加到当前用于追加数据的segment文件中。生产者发送的消息，存储到一个partition中，生产者也可以指定数据存储的partition。

# Broker

- Kafka 集群包含一个或多个服务器，服务器节点称为broker。
- Broker存储topic的数据。如果某topic有N个partition，集群有N个broker，那么每个broker存储该topic的一个partition。
- 如果某个topic有N个partition，集群有(N+M)个broker，那么其中有N个broker存储该topic的一个partition，剩下的M个broker不存储该topic的partition数据。
- 如果某topic有N个partition，集群中broker数目少于N个，那么一个broker存储该topic的一个或多个partition。在实际生产环境中，尽量避免这种情况的发生，这种情况容易导致Kafka集群数据不均衡。

# Topic

每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）

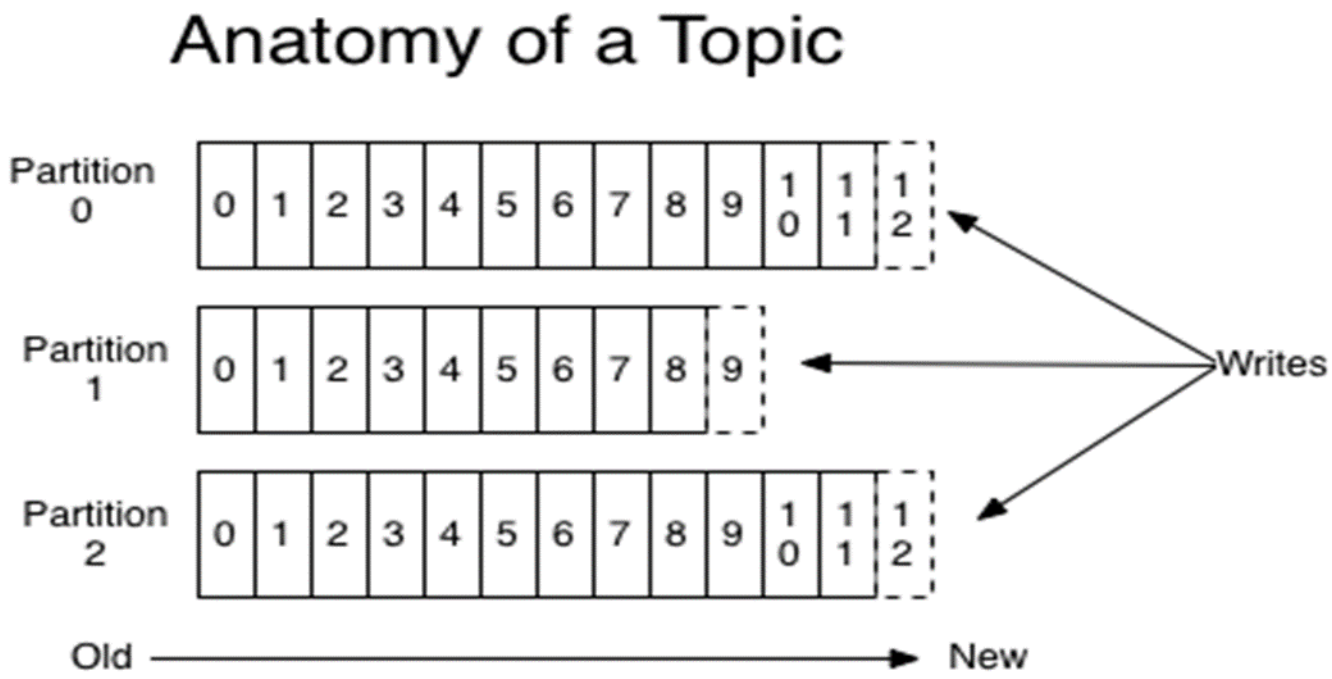
# Topics

Topic在逻辑上可以被认为是一个queue，每条消费都必须指定它的Topic，可以简单理解为必须指明把这条消息放进哪个queue里。为了使得Kafka的吞吐率可以线性提高，物理上把Topic分成一个或多个Partition，每个Partition在物理上对应一个文件夹，该文件夹下存储这个Partition的所有消息和索引文件。创建一个topic时，同时可以指定分区数目，分区数越多，其吞吐量也越大，但是需要的资源也越多，同时也会导致更高的不可用性，Kafka在接收到生产者发送的消息之后，会根据均衡策略将消息存储到不同的分区中。因为每条消息都被追加到该Partition中，属于顺序写磁盘，因此效率非常高（经验证，顺序写磁盘效率比随机写内存还要高，这是Kafka高吞吐率的一个很重要的保证）。

# Topics

一个Topic可以认为是一类消息，每个topic将被分成多partition(区)，每个Partition在存储层面是log文件。任何发布到此Partition的消息都会被直接追加到log文件的尾部，每条消息在文件中的位置称为offset（偏移量），Partition是以文件的形式存储在文件系统中。

# Topics




对于传统的message queue而言，一般会删除已经被消费的消息，而Kafka集群会保留所有的消息，无论其被消费与否。当然，因为磁盘限制，不可能永久保留所有数据（实际上也没必要），因此Kafka提供两种策略删除旧数据。一是基于时间，二是基于Partition文件大小。

# Partition

Topic中的数据分割为一个或多个Partition。每个Topic至少有一个Partition。每个Partition中的数据使用多个Segment文件存储。Partition中的数据是有序的，不同Partition间的数据丢失了数据的顺序。如果Topic有多个Partition，消费数据时就不能保证数据的顺序。在需要严格保证消息的消费顺序的场景下，需要将Partition数目设为1。

# Partition

- Kafka基于文件存储。通过分区，可以将日志内容分散到多个server上,来避免文件大小达到单机磁盘的上限，每个Partiton都会被当前server保存
- 可以将一个Topic切分多任意多个Partitions，来提高保存/消费的效率
- 越多的Partitions意味着可以容纳更多的consumer，有效提升并发消费的能力

# Message

- Message消息是通信的基本单位，每个Producer可以向一个Topic发布一些消息。
- Kafka中的Message是以Topic为基本单位组织的，不同的Topic之间是相互独立的。每个Topic又可以分成几个不同的Partition(每个Topic有几个Partition是在创建Topic时指定的)，每个Partition存储一部分Message。
- Partition中的每条Message包含了以下三个属性：
  - offset　　对应类型：long
  - MessageSize　　对应类型：int32
  - data　　是message的具体内容

# Offset

- 每条消息在文件中的位置称为offset（偏移量）。offset 为一个long型数字，它是唯一标记一条消息。Kafka并没有提供其它额外的索引机制来存储offset，因为在Kafka中几乎不允许对消息进行“随机读写”。
- Partition中的每条Message由offset来表示它在这个Partition中的偏移量，这个offset不是该Message在Partition数据文件中的实际存储位置，而是逻辑上一个值。因此，可以认为offset是Partition中Message的ID。

# Offset

怎样记录每个consumer处理的信息的状态？在Kafka中仅保存了每个Consumer已经处理数据的offset。这样有两个好处：

1）保存的数据量少

2）当Consumer出错时，重新启动

Consumer处理数据时，只需从最近的offset开始处理数据即可。

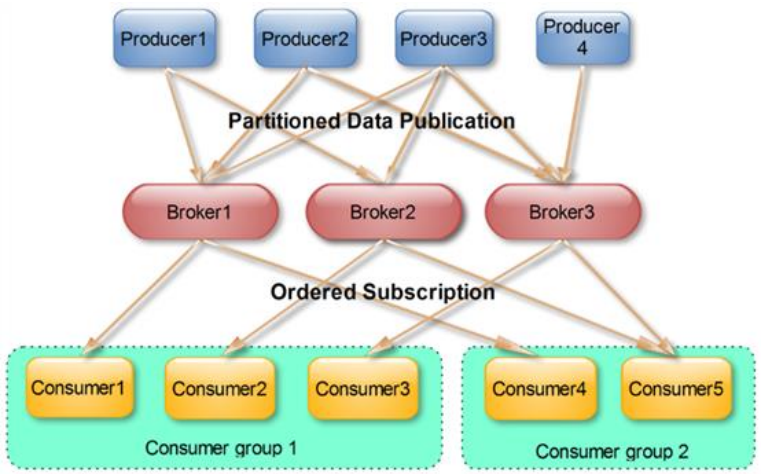
# Consumers

消费者可以从Broker中读取数据。消费者可以消费多个topic中的数据。传统消费一般是通过queue方式（消息依次被感兴趣的消费者接受）和发布订阅的方式（消息被广播到所有感兴趣的消费者）。Kafka采用一种更抽象的方式：消费组（consumer group）来囊括传统的两种方式。首先消费者标记自己一个消费组名。消息将投递到每个消费组中的某一个消费者实例上。如果所有的消费者实例都有相同的消费组，这样就像传统的queue方式。如果所有的消费者实例都有不同的消费组，这样就像传统的发布订阅方式。消费组就好比是个逻辑的订阅者，每个订阅者由许多消费者实例构成(用于扩展或容错)。

# Consumer Group

每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）。

注： Kafka的设计原理决定，对于一个Topic，同一个group中不能有多于Partitions个数的Consumer同时消费，否则将意味着某些Consumer将无法得到消息。




Kafka的设计理念之一就是同时提供离线处理和实时处理。根据这一特性，可以使用Storm这种实时流处理系统对消息进行实时在线处理，同时使用Hadoop这种批处理系统进行离线处理，还可以同时将数据实时备份到另一个数据中心，只需要保证这三个操作所使用的Consumer属于不同的Consumer Group即可。

# Leader

每个Partition有多个副本，其中有且仅有一个作为Leader，Leader是当前负责Partition数据读写的。

# Follower

Follower跟随Leader，所有写请求都通过Leader路由，数据变更会广播给所有Follower，Follower与Leader保持数据同步。如果Leader失效，则从Follower中选举出一个新的Leader。当Follower与Leader挂掉、卡住或者同步太慢，Leader会把这个Follower从“in sync replicas”（ISR）列表中删除，重新创建一个Follower。

# Producer消息路由

- Producer发送消息到Broker时，会根据Paritition机制选择将其存储到哪一个Partition。如果Partition机制设置合理，所有消息可以均匀分布到不同的Partition里，这样就实现了负载均衡。如果一个Topic对应一个文件，那这个文件所在的机器I/O将会成为这个Topic的性能瓶颈，而有了Partition后，不同的消息可以并行写入不同broker的不同Partition里，极大的提高了吞吐率。可以在$KAFKA_HOME/config/server.properties中通过配置项num.partitions来指定新建Topic的默认Partition数量，也可在创建Topic时通过参数指定，同时也可以在Topic创建之后通过Kafka提供的工具修改。
- 在发送一条消息时，可以指定这条消息的key，Producer根据这个key和Partition机制来判断应该将这条消息发送到哪个Parition。Paritition机制可以通过指定Producer的Paritition.class这一参数来指定，该class必须实现kafka.producer.Partitioner接口。

# Push vs. Pull

作为一个消息系统，Kafka遵循了传统的方式，选择由Producer向Broker push消息并由Consumer从Broker pull消息。一些logging-centric system，比如Facebook的Scribe和Cloudera的Flume，采用push模式。事实上，push模式和pull模式各有优劣。

Push模式很难适应消费速率不同的消费者，因为消息发送速率是由Broker决定的。push模式的目标是尽可能以最快速度传递消息，但是这样很容易造成Consumer来不及处理消息，典型的表现就是拒绝服务以及网络堵塞。而pull模式则可以根据Consumer的消费能力以适当的速率消费消息。

对于Kafka而言，pull模式更合适。pull模式可简化Broker的设计，Consumer可自主控制消费消息的速率，同时Consumer可以自己控制消费方式——即可批量消费也可逐条消费。

# Leader Election

Kafka引入了副本策略，同一个Partition可能会有多个Replica，而这时需要在这些Replication之间选出一个Leader，Producer和Consumer只与这个Leader交互，其它Replica作为Follower从Leader中复制数据。

因为需要保证同一个Partition的多个Replica之间的数据一致性（其中一个宕机后其它Replica必须要能继续服务并且即不能造成数据重复也不能造成数据丢失）。如果没有一个Leader，所有Replica都可同时读/写数据，那就需要保证多个Replica之间互相（N×N条通路）同步数据，数据的一致性和有序性非常难保证，大大增加了Replication实现的复杂性，同时也增加了出现异常的几率。而引入Leader后，只有Leader负责数据读写，Follower只向Leader顺序Fetch数据（N条通路），系统更加简单且高效。

# Flume与Kafka区别

Flume (Apache 日志收集系统)，主要功能就是收集同步数据源的数据，并将数据保存到持久化系统中，适合数据来源比较广，数据收集结构比较固定的场景；

Kafka (Apache 分布式消息系统)，主要是作为一个中间件系统的方式存在，适合高吞吐量和负载的情况，可以作为业务系统中的缓存、消息通知系统、数据收集等场景。

一般模式：

Flume(提供数据源) + Kafka（解决sink端挂机问题） + 流式系统