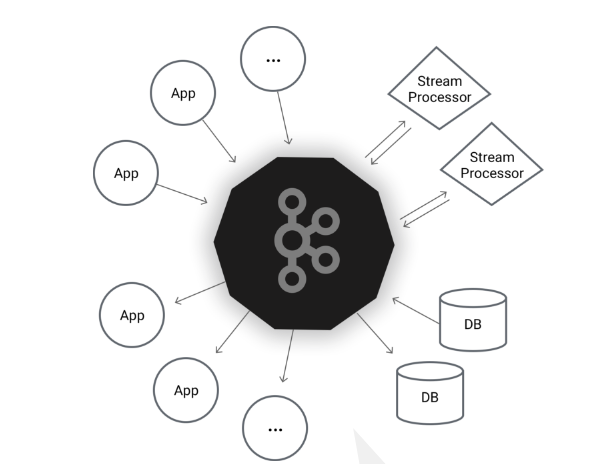
**kafka**

# 概述



kafka是一个分布式流平台。

1. 允许发布和订阅流式数据。在这一点上类似于MQ系统
2. 能够以容错式的方式存储流式数据
3. 能够实时处理流式数据。

使用场景：

构建实时流数据管道可以在系统和应用件可靠获取数据。

构建实时流数据处理应用来处理和响应实时流数据。

* apache kafak是一个开源消息系统，由scala写成。是由Apache软件基金会开发的一个开源消息系统项目。
* kafka最初是由LinkedIn开发，并于2011年初开源。2012年10月从Apache incubator毕业。该项目的目标是为处理实时数据提供一个统一、高通量、低等待的平台。
* kafka是一个分布式消息队列(生产者、消费者的功能)。它提供了类似于JMS的特性、但是在设计实现上完全不同，此外它并不是JMS规范的实现。
* kafka对消息保存时根据Topic进行归类，发送消息者称为Producer，消息接收者称为Consumer，此外kafka集群有多个kafka实例组成，每个实例(server)成为broker。
* 无论是kafka集群，还是producer和consumer都依赖于zookeeper集群保存一些meta信息，来保证系统可用性。

# kafka和其它MQ的区别

1. kafka没有消息确认机制
2. kafka 只有发布订阅模式
3. 吞吐量：kafka具有高吞吐量，内部采用消息的批处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高。
4. 可靠性：MQ支持对消息的可靠的传递，支持事物，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。
5. 抗压性：kafka在大数据量下依然保持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复杂度的访问性能。而MQ则会出现性能下降。
6. 是否支持离线数据处理和实时数据处理：

kafka同时支持离线数据处理和实时数据处理。

1. 扩展性比较：kafka支持在线水平扩展。
2. 消息安全性：kafka可以保证在绝大多数情况下消息的安全性。当消息算作commit的时候才会暴露给consumer，并保证at-least-once的投递原则。
3. 可用性，稳定容错性：kafka天然支持HA，leader挂了，follower自动选举出新的leader
4. 适用场景：kafka是分布式消息系统。mq是基础的消息系统。kafka适用于流式计算。如果需要消息可靠性传输，使用MQ，如果处理海量日志数据等，使用kafka。如果需要同时处理在线应用(消息)和离线应用(数据文件，日志)

# 使用场景

## 消息队列的使用

相比传统消息系统，kafka具有更高的吞吐量，内建分组，复制和容错机制。可以将kafka当做传统消息系统一样来使用。

## 网站活动跟踪

Kafka的初始目标是能够将用户活动流管道重建为一组实时发布 - 订阅的源。也就是说网站活动例如浏览页面，搜索或者其它用户可能的操作会被publish到中央的topics(每个活动类型对应一个topic)，这些源可以被订阅取到活动数据然后做一系列事情,比如实时计算或者实时监控或者存储到hadoop等离线数据仓库系统做离线处理(比如经过离线计算后生成报表之类)。

活动跟踪通常量比较大，因为为每个用户页面视图生成许多活动消息。

## 度量

卡夫卡通常用于运行监控数据。这涉及从分布式应用程序汇总统计数据以生成操作数据的集中式源。

## 日志聚合

许多人使用Kafka作为日志聚合解决方案的替代品。日志聚合通常从服务器收集物理日志文件，并将其置于中央位置（可能是文件服务器或HDFS）进行处理。Kafka抽象了提取文件的细节，并将日志或事件数据作为消息流进行更清晰的抽象。这样可以实现更低延迟的处理，并且更容易支持多个数据源和分布式数据消耗.。与Scribe或Flume等以日志为中心的系统相比，Kafka提供同样出色的性能，由复制产生的更强大的持久性保证以及更低的端到端延迟。

## 流式处理

Kafka的许多用户在处理管道中处理数据，这些数据由多个阶段组成，其中原始输入数据从Kafka主题中消耗，然后聚合，丰富或以其他方式转化为新的主题，以供进一步消费或后续处理。例如，用于推荐新闻文章的处理流水线可以从RSS提要抓取文章内容并将其发布到“文章”主题; 进一步的处理可以对这个内容进行归一化或者重复删除，并且将已清理的文章内容发布到新的主题; 最终处理阶段可能会尝试将这些内容推荐给用户。这种处理流水线基于各个主题创建实时数据流的图表。从0.10.0.0开始，这是一个轻量但功能强大的流处理库，称为Kafka Streams 在Apache Kafka中可用于执行如上所述的数据处理。除了Kafka Streams之外，替代性的开源流处理工具还包括Apache Storm和 Apache Samza。

## 事件源(Event sourcing)

Event sourcing 是一种应用程序设计的风格，即状态改变以时间排序的序列进行记录。Kafka支持超大日志数据，使它可以成为基于Evnet sourcing设计的应用程序的优秀后端。

## 提交日志(Commit Log)

Kafka可以作为分布式系统的一种外部提交日志

日志有助于复制节点之间的数据，并作为失败节点恢复数据的重新同步机制。Kafka中的日志压缩功能有助于支持这种用法。在这个用法中，Kafka与Apache BookKeeper项目类似。

# API

Producer api：允许应用程序作为生产者发送流式数据到kafka集群中的topics

Consumer api: 允许应用程序作为消费者从kafka集群中的topics读取数据

Streams api：允许应用程序将输入的流式数据转换然后再次发送到指定的topics

Connect api：允许连接器从某个源系统或应用程序持续拉数据到kafka或者从kafka推送数据到某个消费系统或应用程序。

AdminClient api：允许管理和监控topics,brokers以及其它kafka对象。

## Producer api

允许应用程序作为生产者发送流式数据到kafka集群中的topics

定义：public class KafkaProducer<K,V> extends java.lang.Object implements Producer<K,V>

Producer是线程安全的，多线程环境中，推荐使用一个单一的producer，将会比每个线程一个producer的速度要快。

示例如下：

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("acks", "all");

props.put("retries", 0);

props.put("batch.size", 16384);

props.put("linger.ms", 1);

props.put("buffer.memory", 33554432);

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer<>(props);

for (int i = 0; i < 100; i++)

producer.send(new ProducerRecord<String, String>("my-topic", Integer.toString(i), Integer.toString(i)));

producer.close();

1. send方法是异步的。调用的时候会将一条记录添加到发送缓冲区，然后立即返回。这种方式允许生产者批量发送数据以提高效率。
2. acks

0：这意味着生产者producer不等待来自broker同步完成的确认就继续发送下一条（批）消息。此选项提供最低的延迟但最弱的耐久性保证，因为其没有任何确认机制。

1:意味着producer在leader已成功收到的数据并得到确认后发送下一条消息。等待leader的确认后就返回，而不管partion的follower是否已经完成。

2: 这意味着producer在follower副本确认接收到数据后才算一次发送完成。此选项提供最好的耐久性，我们保证没有信息将丢失，只要至少一个同步副本保持存活

如果发送请求失败，生产者会自动重试，尽管已经设置restries为0？(不确定)。允许重试将会导致有可能对同一条消息多次传输。

producer为每一个partition维护一个未发送记录的缓冲区。这些bufers各自的大小可以通过batch.size指定。将该值设大了经会导致批量提交的效率但也会更多的占用内存。

默认情况下，一个buffer是可以立即发送的，尽管还有未使用的空间。如果你想降低请求的次数，可以设置linger.ms为比0更大的值。linger.ms表示生产者在前一次发送之后等待多久后再次发送，这样做可以使更多的消息批量提交以提高缓存利用率。

在上面的示例代码中，如果有100条消息产生，在设置linger.ms为1ms的情况下，所有100条数据将会在一起请求中被发送。注意：及时到达的批量消息也会批量被提交尽管linger.ms被设置为0。因此在负载比较重的情况下可以忽略linger.ms的设置。但不管怎样，设置linger.ms大于0的某个值可以负载比较小的情况起到提高发送效率的作用。

buffer.memory控制了producer分配所有buffers时的总内存大小。如果记录的发送速度比发送到服务器的速度快，那么这个缓冲区空间将被耗尽。当缓冲区空间耗尽时，另外的发送调用将被阻塞。 阻塞时间的阈值由max.block.ms决定，在此之后它将引发TimeoutException。

key.serializer和value.serializer指示如何将用户提供的ProducerRecord键和值对象转换为字节。 您可以使用包含的ByteArraySerializer或StringSerializer作为简单的字符串或字节类型。

从Kafka 0.11开始，KafkaProducer支持另外两种模式：幂等生产者和事务生产者

幂等生产者增强了kafka的发送机制，从至少发送一次精确到只发送一次。也就是说生产者重试将不会导致发送多份副本。

事务性生产者允许应用程序原子性的发送消息到多个分区（或主题）。

如何开启幂等性支持：

设置enable.idempotence为true，如果设置enable.idempotence为true，retries配置将会默认设为Integer.MAX\_VALUE并且acks 配置将会默认设置为all。

幂等生产者没有API改变，所以现有的应用程序不需要被修改来利用这个特征。

为了利用幂等性的生产者。应用程序级别的代码应该避免重复发送。也就是说，如果一个应用程序想要使用幂等性生产者，就不要去配置retries属性。另外，如果send(ProducerRecord)甚至在无限重试的情况下返回一个错误（例如，如果消息在发送之前在缓冲区中到期），那么建议关闭生产者并检查最后生成的消息的内容以确保 它不重复。最后，幂等生产者只能保证在单个会话中发送的消息具有同等性。

要使用事务性生产者和伴随的API，您必须设置transactional.id配置属性。 如果设置了transactional.id，则idempotence将自动启用，同时生产者配置将依赖于哪个幂等。

此外，交易中包含的主题应该配置为耐用性(replication.factor应该至少为3，而这些主题的min.insync.replicas应该设置为2.)。最后，为了使事务保证从端到端实现，消费者必须配置为只读取提交的消息。

### producer的同步异步机制：

通过配置producer.type的值来确定是异步还是同步，默认为同步。async/sync 默认是sync。

如果设置为异步，那么提供了批量发送的功能,当满足以下其中一个条件的时候就触发发送。

1. batch.num.messages 异步发送 每次批量发送的条目 ；
2. queue.buffering.max.ms 异步发送的时候 发送时间间隔 单位是毫秒。

其次，异步发送消息的实现很简单，客户端消息发送过来以后，先放入到一个队列中然后就返回了。Producer再开启一个线程（ProducerSendThread）不断从队列中取出消息，然后调用同步发送消息的接口将消息发送给Broker。

### 分区partion发送

为了负载均衡一个topic可能会有多个partition，不同的partition存在在不同的broker里面，因此可以设定一定的partition的规则来确定什么样的消息发送到那个partition当中，代码如下：

public class CustomizePartitioner implements Partitioner{

/\*\*

\* 返回分区索引编号

\* @param key sendMessage时，输出的partKey

\* @param numPartitions topic中的分区总数

\* @return

\*/

@Override

public int partition(Object key, int numPartitions) {

System.out.println("key:" + key + " numPartitions:" + numPartitions);

String partKey = (String)key;

if ("part2".equals(partKey))

return 2;

return 0;

}

}

partition当中的metadata.broker.list：

该选项用于存放broker的元信息，官方翻译如下：

这个选项是用于一个producer启动的时候，在启动的时候producer会通过这个选项配置的broker的地址去获取元信息（topics, partitions and replicas）。它的格式如下，host1:port1,host2:port2。这个list可以是一个broker集合的子集。

需要注意的是producer是如何动态获取集群中的broker信息的变化呢，它又没有和zookeeper进行交互?

1,producer没有直接和zookeeper进行通信，但是broker集群会和zookeeper进行进行通信，然后broker集群会把元信息返回给producer；

2,producer在调用send方法的时候会去定时的刷新metadata信息（这自己又些以为，不太明白producer的定时刷新的机制）

3，由于在调用send之前可能会刷新metadata信息，因此可能会有一些延迟。如果不想要该延迟，把topic.metadata.refresh.interval.ms值改为-1，这样只有在发送失败时，才会重新刷新。Kafka的集群中如果某个partition所在的broker挂了，可以检查错误后重启重新加入集群，手动做rebalance，producer的连接会再次断掉，直到rebalance完成，那么刷新后取到的连接着中就会有这个新加入的broker。

除了上面说的之外，kafka还提供了新的producer写法，见http://kafka.apache.org/documentation.html#producerapi