**Kafka 入门**

## Apache Kafka 概述

在大数据中，使用了大量的数据。 关于数据，我们有两个主要挑战。第一个挑战是如何收集大量的数据，第二个挑战是分析收集的数据。 为了克服这些挑战，您必须需要一个消息系统。

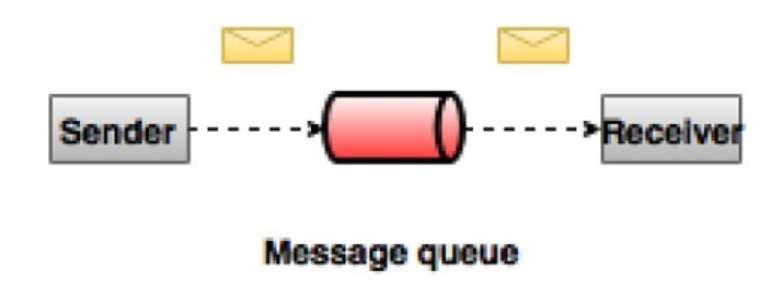
Kafka专为分布式高吞吐量系统而设计。 Kafka往往工作得很好，作为一个更传统的消息代理的替代品。 与其他消息传递系统相比，Kafka具有更好的吞吐量，内置分区，复制和固有的容错能力，这使得它非常适合大规模消息处理应用程序

**什么是消息系统**

消息系统负责将数据从一个应用程序传输到另一个应用程序，因此应用程序可以专注于数据，但不担心如何共享它。 分布式消息传递基于可靠消息队列的概念。 消息在客户端应用程序和消息传递系统之间异步排队。 有两种类型的消息模式可用 - 一种是点对点，另一种是发布 - 订阅(pub-sub)消息系统。 大多数消息模式遵循**pub-sub。**

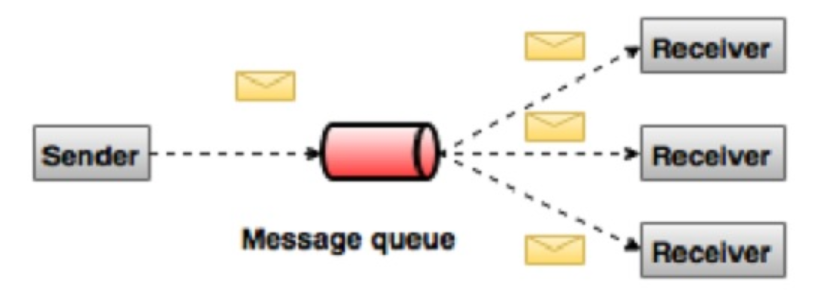
**点对点消息系统**

在点对点系统中，消息被保留在队列中。 一个或多个消费者可以消耗队列中的消息，但是特定消息只能由最多一个消费者消费。 一旦消费者读取队列中的消息，它就从该队列中消失。 该系统的典型示例是订单处理系统，其中每个订单将由一个订单处理器处理，但多个订单处理器也可以同时工作。

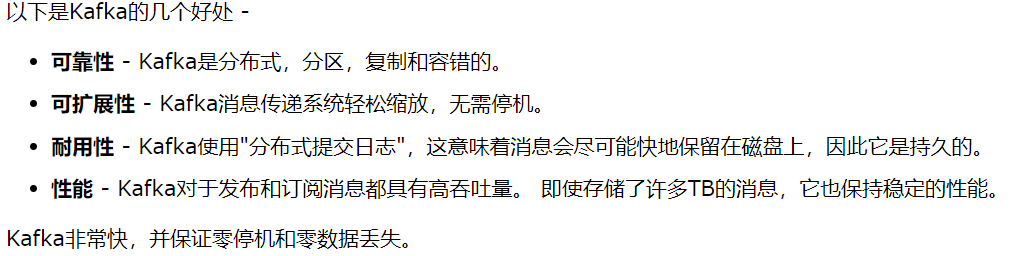


**发布 - 订阅消息系统**

在发布 - 订阅系统中，消息被保留在主题中。 与点对点系统不同，消费者可以订阅一个或多个主题并使用该主题中的所有消息。 在发布 - 订阅系统中，消息生产者称为发布者，消息使用者称为订阅者。 一个现实生活的例子是Dish电视，它发布不同的渠道，如运动，电影，音乐等，任何人都可以订阅自己的频道集，并获得他们订阅的频道时可用。



**好处**



Kafka是一个统一的平台，用于处理所有实时数据Feed。 Kafka支持低延迟消息传递，并在出现机器故障时提供对容错的保证。 它具有处理大量不同消费者的能力。 Kafka非常快，执行2百万写/秒。 Kafka将所有数据保存到磁盘，这实质上意味着所有写入都会进入操作系统(RAM)的页面缓存。 这使得将数据从页面缓存传输到网络套接字非常有效。

## Kafka基础

首先让我们看几个基本的消息系统术语：

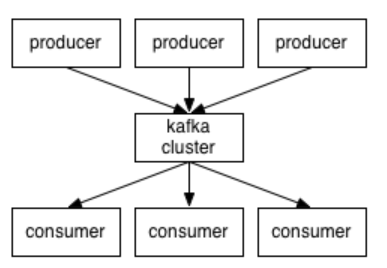
1.Kafka将消息以topic为单位进行归纳。

2.将向Kafka topic发布消息的程序成为producers.

3.将预订topics并消费消息的程序成为consumer.

4.Kafka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.

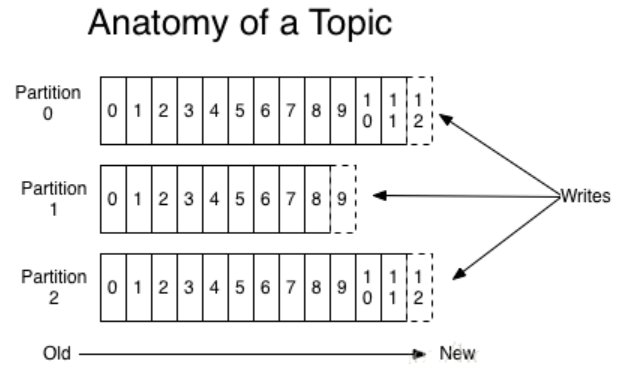
producers通过网络将消息发送到Kafka集群，集群向消费者提供消息，如下图所示：



客户端和服务端通过TCP协议通信。Kafka提供了Java客户端，并且对多种语言都提供了支持。

**Topics 和Logs：**

一个topic是对一组消息的归纳。对每个topic，Kafka 对它的日志进行了分区，如下图所示：



每个分区都由一系列有序的、不可变的消息组成，这些消息被连续的追加到分区中。分区中的每个消息都有一个连续的序列号叫做offset,用来在分区中唯一的标识这个消息。

在一个可配置的时间段内，Kafka集群保留所有发布的消息，不管这些消息有没有被消费。比如，如果消息的保存策略被设置为2天，那么在一个消息被发布的两天时间内，它都是可以被消费的。之后它将被丢弃以释放空间。Kafka的性能是和数据量无关的常量级的，所以保留太多的数据并不是问题。

实际上每个**consumer**唯一需要维护的数据是消息在日志中的位置，也就是offset.这个offset有consumer来维护：一般情况下随着consumer不断的读取消息，这offset的值不断增加，但其实consumer可以以任意的顺序读取消息，比如它可以将offset设置成为一个旧的值来重读之前的消息。

将日志分区可以达到以下目的：首先这使得每个日志的数量不会太大，可以在单个服务上保存。另外每个分区可以单独发布和消费，为并发操作topic提供了一种可能。

**分布式**

每个分区在Kafka集群的若干服务中都有副本，这样这些持有副本的服务可以共同处理数据和请求，副本数量是可以配置的。副本使Kafka具备了容错能力。

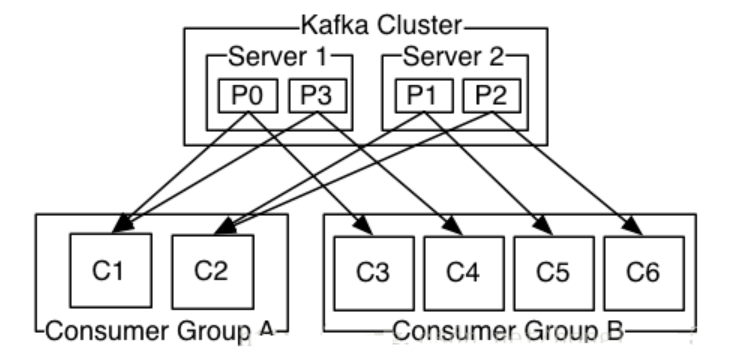
每个分区都由一个服务器作为“leader”，零或若干服务器作为“followers”,leader负责处理消息的读和写（zookeeper的followers也可以负责读写），followers则去复制leader.如果leader down了，followers中的一台则会自动成为leader。集群中的每个服务都会同时扮演两个角色：作为它所持有的一部分分区的leader，同时作为其他分区的followers，这样集群就会据有较好的负载均衡。

**Producers**

Producer将消息发布到它指定的topic中,并负责决定发布到哪个分区。通常简单的由负载均衡机制随机选择分区，但也可以通过特定的分区函数选择分区。使用的更多的是第二种。

**Consumers**

发布消息通常有两种模式：队列模式（queuing）和发布-订阅模式(publish-subscribe)。队列模式中，consumers可以同时从服务端读取消息，每个消息只被其中一个consumer读到；发布-订阅模式中消息被广播到所有的consumer中。Consumers可以加入一个consumer 组，共同竞争一个topic，topic中的消息将被分发到组中的一个成员中。同一组中的consumer可以在不同的程序中，也可以在不同的机器上。如果所有的consumer都在一个组中，这就成为了传统的队列模式，在各consumer中实现负载均衡。如果所有的consumer都不在不同的组中，这就成为了发布-订阅模式，所有的消息都被分发到所有的consumer中。更常见的是，每个topic都有若干数量的consumer组，每个组都是一个逻辑上的“订阅者”，为了容错和更好的稳定性，每个组由若干consumer组成。这其实就是一个发布-订阅模式，只不过订阅者是个组而不是单个consumer。



由两个机器组成的集群拥有4个分区 (P0-P3) 2个consumer组. A组有两个consumerB组有4个

相比传统的消息系统，Kafka可以很好的保证有序性。传统的队列在服务器上保存有序的消息，如果多个consumers同时从这个服务器消费消息，服务器就会以消息存储的顺序向consumer分发消息。虽然服务器按顺序发布消息，但是消息是被异步的分发到各consumer上，所以当消息到达时可能已经失去了原来的顺序，这意味着并发消费将导致顺序错乱。为了避免故障，这样的消息系统通常使用“专用consumer”的概念，其实就是只允许一个消费者消费消息，当然这就意味着失去了并发性。

在这方面Kafka做的更好，通过分区的概念，Kafka可以在多个consumer组并发的情况下提供较好的有序性和负载均衡。将每个分区分只分发给一个consumer组，这样一个分区就只被这个组的一个consumer消费，就可以顺序的消费这个分区的消息。因为有多个分区，依然可以在多个consumer组之间进行负载均衡。注意consumer组的数量不能多于分区的数量，也就是有多少分区就允许多少并发消费。

Kafka只能保证一个分区之内消息的有序性，在不同的分区之间是不可以的，这已经可以满足大部分应用的需求。如果需要topic中所有消息的有序性，那就只能让这个topic只有一个分区，当然也就只有一个consumer组消费它。

## 工作流程

### 发布 - 订阅消息的工作流程

* 生产者定期向主题发送消息。
* Kafka代理存储为该特定主题配置的分区中的所有消息。 它确保消息在分区之间平等共享。 如果生产者发送两个消息并且有两个分区，Kafka将在第一分区中存储一个消息，在第二分区中存储第二消息。（**负载均衡**）
* 消费者订阅特定主题。
* 一旦消费者订阅主题，Kafka将向消费者提供主题的当前偏移，并且还将偏移保存在Zookeeper系综中。
* 消费者将定期请求Kafka(如100 Ms)新消息。
* 一旦Kafka收到来自生产者的消息，它将这些消息转发给消费者。
* 消费者将收到消息并进行处理。
* 一旦消息被处理，消费者将向Kafka代理发送确认。
* 一旦Kafka收到确认，它将偏移更改为新值，并在Zookeeper中更新它。 由于偏移在Zookeeper中维护，消费者可以正确地读取下一封邮件，即使在服务器暴力期间。
* 以上流程将重复，直到消费者停止请求。
* 消费者可以随时回退/跳到所需的主题偏移量，并阅读所有后续消息。

### 队列消息/用户组的工作流程

在队列消息传递系统而不是单个消费者中，具有相同组ID 的一组消费者将订阅主题。 简单来说，订阅具有相同 Group ID 的主题的消费者被认为是单个组，并且消息在它们之间共享

* 生产者以固定间隔向某个主题发送消息。
* Kafka存储在为该特定主题配置的分区中的所有消息，类似于前面的方案。
* 单个消费者订阅特定主题，假设 Topic-01 为 Group ID 为 Group-1 。
* Kafka以与发布 - 订阅消息相同的方式与消费者交互，直到新消费者以相同的组ID 订阅相同主题 Topic-01  1 。
* 一旦新消费者到达，Kafka将其操作切换到共享模式，并在两个消费者之间共享数据。 此共享将继续，直到用户数达到为该特定主题配置的分区数。
* 一旦消费者的数量超过分区的数量，新消费者将不会接收任何进一步的消息，直到现有消费者取消订阅任何一个消费者。 出现这种情况是因为Kafka中的每个消费者将被分配至少一个分区，并且一旦所有分区被分配给现有消费者，新消费者将必须等待。
* 此功能也称为使用者组。 同样，Kafka将以非常简单和高效的方式提供两个系统中最好的。

### ZooKeeper的作用

Apache Kafka的一个关键依赖是Apache Zookeeper，它是一个分布式配置和同步服务。 Zookeeper是Kafka代理和消费者之间的协调接口。 Kafka服务器通过Zookeeper集群共享信息。 Kafka在Zookeeper中存储基本元数据，例如关于主题，代理，消费者偏移(队列读取器)等的信息

由于所有关键信息存储在Zookeeper中，并且它通常在其整体上复制此数据，因此Kafka代理/ Zookeeper的故障不会影响Kafka集群的状态。 Kafka将恢复状态，一旦Zookeeper重新启动。 这为Kafka带来了零停机时间。Kafka代理之间的领导者选举也通过使用Zookeeper在领导者失败的情况下完成。