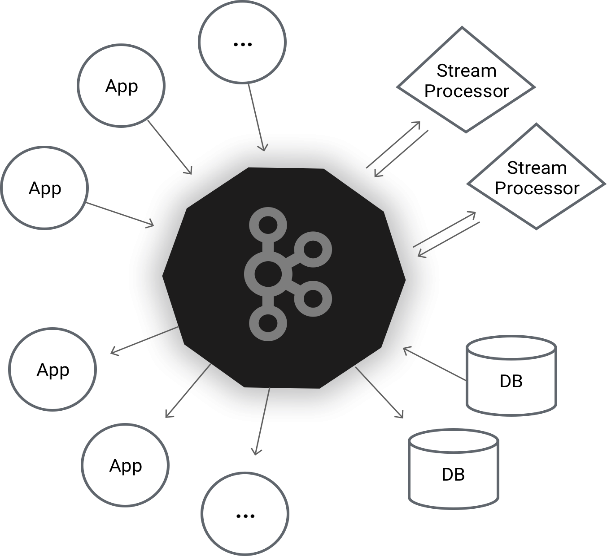
参考：<http://orchome.com/kafka/index>

**kafka中文教程**

本网翻译整理Apache kafka，提供整理Apache kafka的完整学习文档。

### 发布 & 订阅              处理                     存储

数据流，如消息传递系统 高效并实时 数据流安全地在分布式集群中复制存储                              

kafka是用于构建实时数据管道和流应用程序。具有横向扩展，容错，wicked fast（变态快）等优点，并已在成千上万家公司运行。

## 简单说明什么是kafka

Apache kafka是消息中间件的一种，我发现很多人不知道消息中间件是什么，在开始学习之前，我这边就先简单的解释一下什么是消息中间件，只是粗略的讲解，目前kafka已经可以做更多的事情。

举个例子，生产者消费者，生产者生产鸡蛋，消费者消费鸡蛋，生产者生产一个鸡蛋，消费者就消费一个鸡蛋，假设消费者消费鸡蛋的时候噎住了（系统宕机了），生产者还在生产鸡蛋，那新生产的鸡蛋就丢失了。再比如生产者很强劲（大交易量的情况），生产者1秒钟生产100个鸡蛋，消费者1秒钟只能吃50个鸡蛋，那要不了一会，消费者就吃不消了（消息堵塞，最终导致系统超时），消费者拒绝再吃了，”鸡蛋“又丢失了，这个时候我们放个篮子在它们中间，生产出来的鸡蛋都放到篮子里，消费者去篮子里拿鸡蛋，这样鸡蛋就不会丢失了，都在篮子里，而这个篮子就是”kafka“。  
鸡蛋其实就是“数据流”，系统之间的交互都是通过“数据流”来传输的（就是tcp、http什么的），也称为报文，也叫“消息”。  
消息队列满了，其实就是篮子满了，”鸡蛋“ 放不下了，那赶紧多放几个篮子，其实就是kafka的扩容。  
各位现在知道kafka是干什么的了吧，它就是那个"篮子"。

## kafka名词解释

后面大家会看到一些关于kafka的名词，比如topic、producer、consumer、broker，我这边来简单说明一下。

* producer：生产者，就是它来生产“鸡蛋”的。
* consumer：消费者，生出的“鸡蛋”它来消费。
* topic：你把它理解为标签，生产者每生产出来一个鸡蛋就贴上一个标签（topic），消费者可不是谁生产的“鸡蛋”都吃的，这样不同的生产者生产出来的“鸡蛋”，消费者就可以选择性的“吃”了。
* broker：就是篮子了。

大家一定要学会抽象的去思考，上面只是属于业务的角度，如果从技术角度，topic标签实际就是队列，生产者把所有“鸡蛋（消息）”都放到对应的队列里了，消费者到指定的队列里取。

## 怎么样才算真正的学会kafka

最近面试发现，很多人用过kafka，但是没人了解原理，我们可是很注重原理的(PS:要不然怎么知道你真的会呢)。

* kafka节点之间如何复制备份的？
* kafka消息是否会丢失？为什么？
* kafka最合理的配置是什么？
* kafka的leader选举机制是什么？
* kafka对硬件的配置有什么要求？
* kafka的消息保证有几种方式？

。。。。。。 你是否都答得上来？（欢迎大家补充！）

这些问题在下面的文章中都可以找到答案，kafka之所以有这么火热，建议各位一定要看一下。

## 如何学习kafka

还是那句话，学习任何技术，跟学骑自行车一样，不要一开始只关注它的具体细节是什么。先学着怎么骑，骑着骑着就了解大致的原理，这个时候在去看它的原理，会很轻松。

如果你在学习的过程中遇到什么问题，直接评论或者在问题专区中提问，我们在线人员会回复你的问题。

#### 章节与kafka官网对应一致

第一章：开始  
第二章：API  
第三章：kafka的配置  
第四章：kafka如何设计的  
第五章：kafka的实现  
第六章：kafka的常用操作，如扩容，删除和增加topic。  
第七章：安全  
第八章：kafka Connect  
第九章：kafka 流  
第十章：笔记（[kafka命令大全](http://orchome.com/454)）

# Apache Kafka下载

作者：半兽人  
链接：http://orchome.com/5  
来源：OrcHome  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

你可以登录Apache kafka 官方下载。  
<http://kafka.apache.org/downloads.html>

### 注意:别下成源文件了！

带src的是源文件,如：

Source download: kafka-0.10.1.0-src.tgz (asc, md5)

你应该下的是：

Scala 2.11 - kafka\_2.11-0.10.1.0.tgz (asc, md5)

推荐下载scala 2.11版本的。

# kafka入门介绍

## Kafka作为一个分布式的流平台，这到底意味着什么？

我们认为，一个流处理平台具有三个关键能力：

1. 发布和订阅消息（流），在这方面，它类似于一个消息队列或企业消息系统。
2. 以容错的方式存储消息（流）。
3. 在消息流发生时处理它们。

##### 什么是kakfa的优势？

它应用于2大类应用：

1. 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。
2. 构建实时流的应用程序，对数据流进行转换或反应。

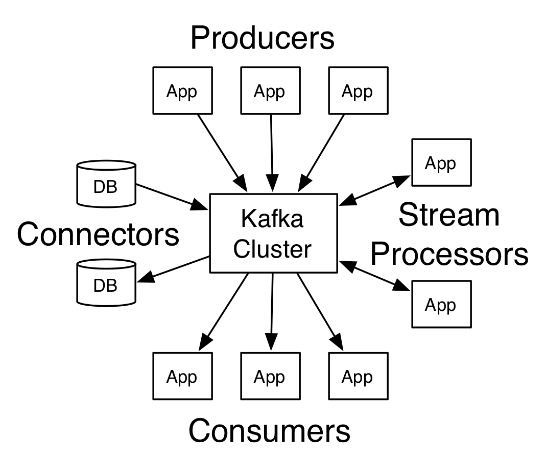
要了解kafka是如何做这些事情的，让我们从下到上深入探讨kafka的能力。

##### 首先几个概念：

1. kafka作为一个集群运行在一个或多个服务器上。
2. kafka集群存储的消息是以topic为类别记录的。
3. 每个消息（也叫记录record，我习惯叫消息）是由一个key，一个value和时间戳构成。

##### kafka有四个核心API：

* 应用程序使用 Producer API 发布消息到1个或多个topic（主题）。
* 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，并生产一个输出流到1个或多个输出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连接到现有的应用程序或数据系统。例如，一个关系数据库的连接器可捕获每一个变化。



Client和Server之间的通讯，是通过一条简单、高性能并且和开发语言无关的[TCP协议](https://kafka.apache.org/protocol.html)。并且该协议保持与老版本的兼容。Kafka提供了Java Client（客户端）。除了Java Client外，还有非常多的[其它编程语言的Client](https://cwiki.apache.org/confluence/display/KAFKA/Clients)。

## 首先来了解一下Kafka所使用的基本术语：

##### Topic

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).

##### Producer

发布消息的对象称之为主题生产者(Kafka topic producer)

##### Consumer

订阅消息并处理发布的消息的种子的对象称之为主题消费者(consumers)

##### Broker

已发布的消息保存在一组服务器中，称之为Kafka集群。集群中的每一个服务器都是一个代理(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。

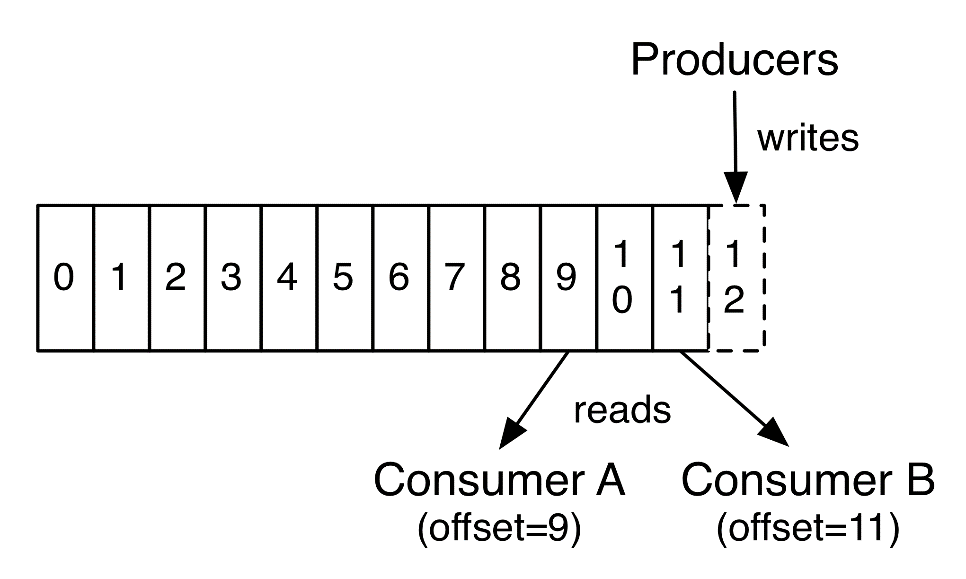
## 话题和日志 (Topic和Log)

让我们更深入的了解Kafka中的Topic。

Topic是发布的消息的类别或者种子Feed名。对于每一个Topic，Kafka集群维护这一个分区的log，就像下图中的示例：



每一个分区都是一个顺序的、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。  


## 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。

## 生产者(Producers)

生产者往某个Topic上发布消息。生产者也负责选择发布到Topic上的哪一个分区。最简单的方式从分区列表中轮流选择。也可以根据某种算法依照权重选择分区。开发者负责如何选择分区的算法。

## 消费者(Consumers)

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。正如下图所示：  


2个kafka集群托管4个分区（P0-P3），2个消费者组，消费组A有2个消费者实例，消费组B有4个。

正像传统的消息系统一样，Kafka保证消息的顺序不变。 再详细扯几句。传统的队列模型保持消息，并且保证它们的先后顺序不变。但是， 尽管服务器保证了消息的顺序，消息还是异步的发送给各个消费者，消费者收到消息的先后顺序不能保证了。这也意味着并行消费将不能保证消息的先后顺序。用过传统的消息系统的同学肯定清楚，消息的顺序处理很让人头痛。如果只让一个消费者处理消息，又违背了并行处理的初衷。 在这一点上Kafka做的更好，尽管并没有完全解决上述问题。 Kafka采用了一种分而治之的策略：分区。 因为Topic分区中消息只能由消费者组中的唯一一个消费者处理，所以消息肯定是按照先后顺序进行处理的。但是它也仅仅是保证Topic的一个分区顺序处理，不能保证跨分区的消息先后处理顺序。 所以，如果你想要顺序的处理Topic的所有消息，那就只提供一个分区。

## Kafka的保证(Guarantees)

* 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 factor）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。

有关这些保证的更多详细信息，请参见文档的设计部分。

## kafka作为一个消息系统

##### Kafka的流与传统企业消息系统相比的概念如何？

传统的消息有两种模式：队列和发布订阅。 在队列模式中，消费者池从服务器读取消息（每个消息只被其中一个读取）; 发布订阅模式：消息广播给所有的消费者。这两种模式都有优缺点，队列的优点是允许多个消费者瓜分处理数据，这样可以扩展处理。但是，队列不像多个订阅者，一旦消息者进程读取后故障了，那么消息就丢了。而发布和订阅允许你广播数据到多个消费者，由于每个订阅者都订阅了消息，所以没办法缩放处理。

kafka中消费者组有两个概念：队列：消费者组（consumer group）允许同名的消费者组成员瓜分处理。发布订阅：允许你广播消息给多个消费者组（不同名）。

kafka的每个topic都具有这两种模式。

##### kafka有比传统的消息系统更强的顺序保证。

传统的消息系统按顺序保存数据，如果多个消费者从队列消费，则服务器按存储的顺序发送消息，但是，尽管服务器按顺序发送，消息异步传递到消费者，因此消息可能乱序到达消费者。这意味着消息存在并行消费的情况，顺序就无法保证。消息系统常常通过仅设1个消费者来解决这个问题，但是这意味着没用到并行处理。

kafka做的更好。通过并行topic的parition —— kafka提供了顺序保证和负载均衡。每个partition仅由同一个消费者组中的一个消费者消费到。并确保消费者是该partition的唯一消费者，并按顺序消费数据。每个topic有多个分区，则需要对多个消费者做负载均衡，但请注意，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。

## kafka作为一个存储系统

所有发布消息到消息队列和消费分离的系统，实际上都充当了一个存储系统（发布的消息先存储起来）。Kafka比别的系统的优势是它是一个非常高性能的存储系统。

写入到kafka的数据将写到磁盘并复制到集群中保证容错性。并允许生产者等待消息应答，直到消息完全写入。

kafka的磁盘结构 - 无论你服务器上有50KB或50TB，执行是相同的。

client来控制读取数据的位置。你还可以认为kafka是一种专用于高性能，低延迟，提交日志存储，复制，和传播特殊用途的分布式文件系统。

## kafka的流处理

仅仅读，写和存储是不够的，kafka的目标是实时的流处理。

在kafka中，流处理持续获取输入topic的数据，进行处理加工，然后写入输出topic。例如，一个零售APP，接收销售和出货的输入流，统计数量或调整价格后输出。

可以直接使用producer和consumer API进行简单的处理。对于复杂的转换，Kafka提供了更强大的Streams API。可构建聚合计算或连接流到一起的复杂应用程序。

助于解决此类应用面临的硬性问题：处理无序的数据，代码更改的再处理，执行状态计算等。

Sterams API在Kafka中的核心：使用producer和consumer API作为输入，利用Kafka做状态存储，使用相同的组机制在stream处理器实例之间进行容错保障。

## 拼在一起

消息传递，存储和流处理的组合看似反常，但对于Kafka作为流式处理平台的作用至关重要。

像HDFS这样的分布式文件系统允许存储静态文件来进行批处理。这样系统可以有效地存储和处理来自过去的历史数据。

传统企业的消息系统允许在你订阅之后处理未来的消息：在未来数据到达时处理它。

Kafka结合了这两种能力，这种组合对于kafka作为流处理应用和流数据管道平台是至关重要的。

批处理以及消息驱动应用程序的流处理的概念：通过组合存储和低延迟订阅，流处理应用可以用相同的方式对待过去和未来的数据。它是一个单一的应用程序，它可以处理历史的存储数据，当它处理到最后一个消息时，它进入等待未来的数据到达，而不是结束。

同样，对于流数据管道（pipeline），订阅实时事件的组合使得可以将Kafka用于非常低延迟的管道；但是，可靠地存储数据的能力使得它可以将其用于必须保证传递的关键数据，或与仅定期加载数据或长时间维护的离线系统集成在一起。流处理可以在数据到达时转换它。

有关Kafka提供的保证，api和功能的更多信息，可继续查阅本网

# Kafka的使用场景

下面是一些关于Apache kafka 流行的使用场景。这些领域的概述，可查看[博客文章](https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying)。

## 消息

kafka更好的替换传统的消息系统，消息系统被用于各种场景（解耦数据生产者，缓存未处理的消息，等），与大多数消息系统比较，kafka有更好的吞吐量，内置分区，副本和故障转移，这有利于处理大规模的消息。

根据我们的经验，消息往往用于较低的吞吐量，但需要低的端到端延迟，并需要提供强大的耐用性的保证。

在这一领域的kafka比得上传统的消息系统，如ActiveMQ或RabbitMQ。

## 网站活动追踪

kafka原本的使用场景：用户的活动追踪，网站的活动（网页游览，搜索或其他用户的操作信息）发布到不同的话题中心，这些消息可实时处理，实时监测，也可加载到Hadoop或离线处理数据仓库。

每个用户页面视图都会产生非常高的量。

## 指标

kafka也常常用于监测数据。分布式应用程序生成的统计数据集中聚合。

## 日志聚合

使用kafka代替一个日志聚合的解决方案。

## 流处理

kafka消息处理包含多个阶段。其中原始输入数据是从kafka主题消费的，然后汇总，丰富，或者以其他的方式处理转化为新主题，例如，一个推荐新闻文章，文章内容可能从“articles”主题获取；然后进一步处理内容，得到一个处理后的新内容，最后推荐给用户。这种处理是基于单个主题的实时数据流。从0.10.0.0开始，轻量，但功能强大的流处理，就进行这样的数据处理了。

除了Kafka Streams，还有Apache Storm和Apache Samza可选择。

## 事件采集

事件采集是一种应用程序的设计风格，其中状态的变化根据时间的顺序记录下来，kafka支持这种非常大的存储日志数据的场景。

## 提交日志

kafka可以作为一种分布式的外部提交日志，日志帮助节点之间复制数据，并作为失败的节点来恢复数据重新同步，kafka的日志压缩功能很好的支持这种用法，这种用法类似于Apacha BookKeeper项目。

作者：半兽人  
链接：http://orchome.com/295  
来源：OrcHome  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。