Kafka消息组件

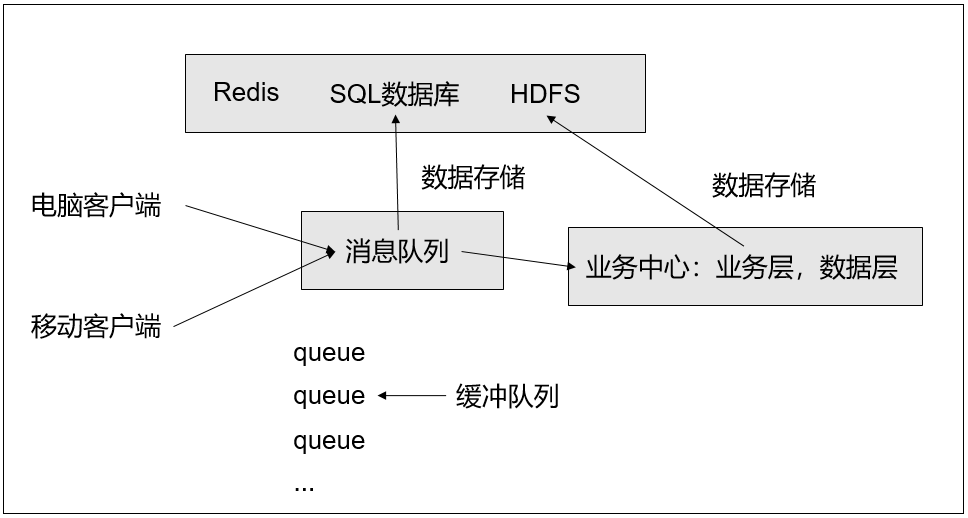
# Kafka简介

## kafka简介

kafka可以说是现在所有开源的消息组件之中性能最高的一款产品，但是同时也需要认识到一个问题：kafka是一项不断继续发展的技术，所以对于其的稳定性永远无法评估。kafka官方网站：<http://kafka.apache.org/>。

**kafka消息框架**

* kafka是分布式发布-订阅消息系统。它最初由LinkedIn公司开发，之后成为Apache项目的一部分。
* kafka是一个分布式的，可划分的，冗余备份的持久性的日志服务。它主要用于处理活跃的流式数据。
* 据LinkedIn统计，最新的数据是每天利用kafka处理的消息超过一万亿条，在峰值时每秒钟会发布超过百万条消息，就算是在内存和CPU都不高的情况下，kafka的速度最高可以达到每秒十万条数据，并且还能持久化存储。



当各个设备发送的消息过多的时候，那么一定会引起数据量的暴涨，如果将这些消息交给处理程序，那么处理程序将无法正确处理，将导致消息数据的丢失，所以使用消息队列有一个最大的功能就是进行数据的缓冲操作。而消息队列有两种处理消息的方式：是直接将消息处理而后保存到持久化设备之中，利用其它的处理程序，例如：storm进行消息的处理。

**JMS与AMQP**

* JMS（Java Message Service、Java消息服务）应用程序接口，是一个Java平台中关于面向 中间件（MOM）的API。
  + JMS消息类型：
    - 点对点（Point-to-Point）队列消息；
    - 发布/订阅（Publish/Subscribe）；
  + JMS支持的数据类型：TextMessage、StreamMessage、MapMessage、ObjectMessage、ByteMessage；
* AMQP（Advanced Message Queuing Protocol），一个提供统一消息服务的应用层标准高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。
* JMS与AMQP区别：
  + AMQP是一种协议，更准确的说是一种binary wire-level protocol（链接协议）。
  + AMQP不从API层进行限定，而是直接定义网络交换的数据格式。这使得实现了AMQP的provider天然型就是跨平台的。

如果要说到消息队列，那么首先自然能够想到的就是JMS（JMS属于Java消息服务，这就是Java原生的操作协议），其中JMS实现的代表性的开源项目：ActiveMQ（这种组件由于其跨越时间太长了，实际上已经不适合于当前高并发的项目使用）。

|  |  |  |  |
| --- | --- | --- | --- |
| 常见消息服务组件 | | | |
|  | ActiveMQ | RabbitMQ | Kafka |
| 所属社区/公司 | Apache | Mozilla Public License | Apache/LinkedIn |
| 开发语言 | Java | Erlang | Java/Scala |
| 支持协议 | OpenWire、STOMQ、REST、XMPP、AMQP | AMQP | 仿AMQP |
| 事物处理 | 支持 | 不支持 | 不支持 |
| 集群 | 支持 | 支持 | 支持 |
| 负载均衡 | 支持 | 支持 | 支持 |
| 动态扩容 | 不支持 | 不支持 | 支持（ZK） |

AMQP是一种不受程序限制的传输的处理协议，而JMS受到程序限制，所以AMQP它的性能和适应性会更高，但是kafka作为一种AMQP的实现有一个最重要的特征：

* RabbitMQ、ActiveMQ有一个特点：消息消费完成后消息就删除；
* kafka特点：所有的消息会自动保存两天的时间。

## kafka工作原理分析

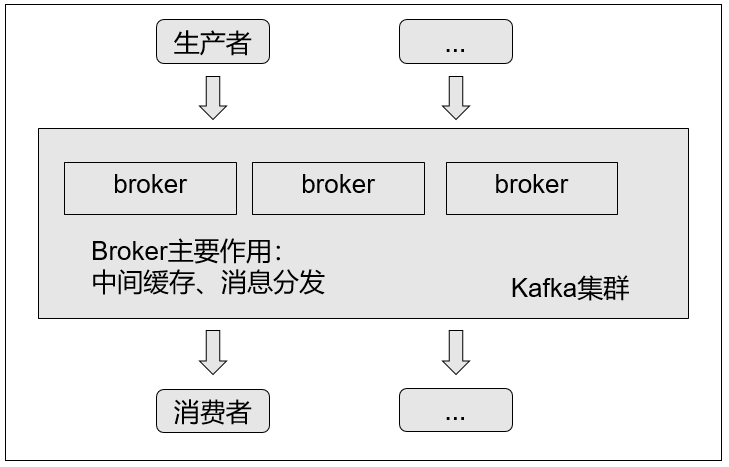
kafka是一款性能很高的消息组件，但是不管如何改变，对于消息组件本身其最基础的组成部分：

* 消息的生产者：负责进行消息信息的推送，推送给指定服务器；
* 消息的消费者：负责通过服务器获取消息的内容；
* 消息服务中间件（服务器）：负责消息的存储，也就是说当消费者来不及处理完全部消息的时候，可以在消息中间件之中进行消息内容的缓冲，所以消息中间件也往往被称为消息队列中间件。

影响整个程序运行的关键因素：程序的设计要合理、CPU要快、内存要大、磁盘转速要快（寻址是成为性能最大的瓶颈），对于消息组件最快的做法就是网络传输也要快，而kafka设计里面讲所有可能影响到程序性能的部分全部都考虑到了。

### kafka架构

* kafka整体采用显示分布式架构，producer、broker(kafka)和consumer都可以有多个；
* producer，consumer实现kafka注册的接口，数据从producer发送到broker，broker承担一个中间缓存和分布的作用。broker分发注册到系统中的consumer；
* broker的作用类似于缓存，即活跃的数据和离线处理系统之间的缓存。客户端和服务器端的通信，最基于简单，高性能，且与编程语言无关的TCP协议。



kafka是基于zookeeper设计，所以对于kafka集群来讲实现就相对容易许多的，同时zookeeper可以保存所有集群主机的信息内容，这也就是说在配置kafka之前一定要首先进行zookeeper配置。

### kafka核心概念

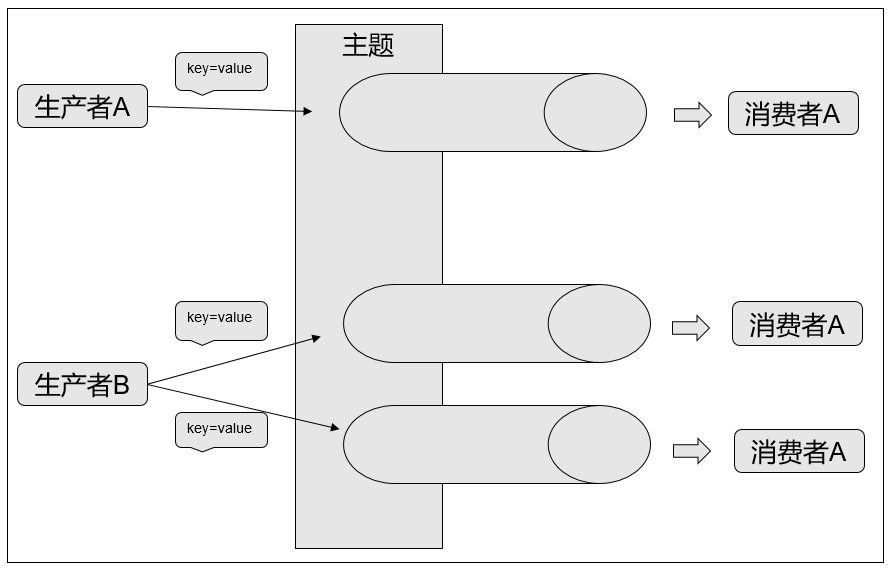
* topic：特指kafka处理的消息源（feeds of messages）的不同分类；
* partition：topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列。partition中的每条消息都会被分配一个有序的id（offset）；
* message：消息，是通信的基本单位，每个producer可以向一个topic（主题）发布一些消息；
* producers：消息和数据生产者，向kafka的一个topic发布消息的过程叫做producers；
* consumers：消息和数据消费者，订阅topics并处理其发布的消息的过程叫consumers；
* broker：缓存代理，kafka集群中的一台或多台服务器同城broker。

如果要想进行消息的处理，很明显所有的消息组件都一定会提供有一个消息主题，所有的消息的生产者根据主题将自己的消息发送并且保存到服务器之中，而消费者也可以通过指定的主题获取消息的内容。这就就可以传递多种消息。

partition：指的是分区，如果你现在配置的主机知识单核CPU，那么你能够进行的合理分区划分就只能够有一个分区，但是如果你的CPU的核心数字可能有16，那么你这台服务器上可以进行的分区操作就可以划分出16个分区，在每一台服务器上，可以有多个分区，而分区划分的最简单的依据：就是根据你CPU的性能来决定。

并不是说一核CPU无法进行多分区配置，只不过要想发挥出最好的性能，那么一定要使用多核的CPU再设置多个分区。再整个的kafka集群里面，所有的分区数量 = 主机的CPU内核数量。

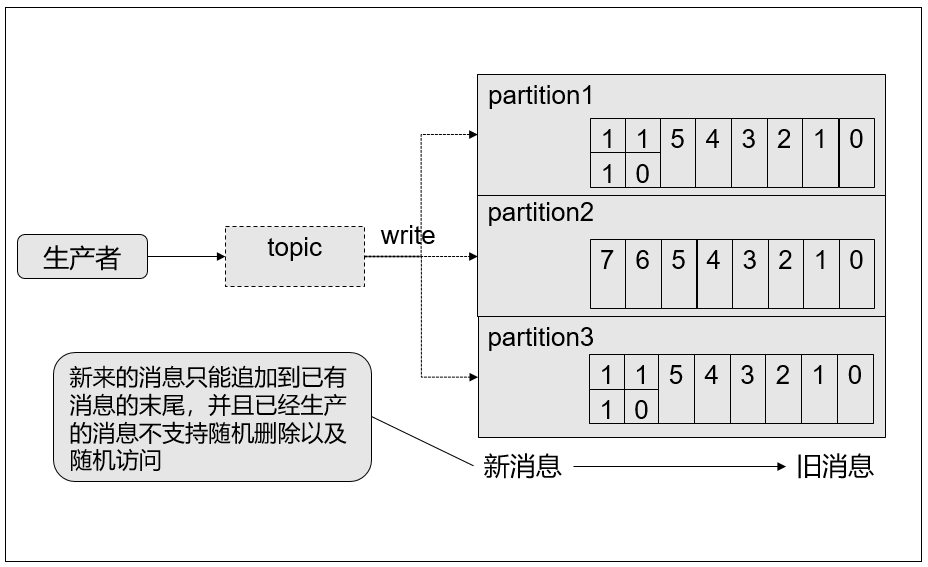
### kafka消息发送流程



在kafka之中消息的发送一定要依据主题进行划分。而每一个主题为了让消息处理更快，专门设置有多个分区，就好比一件工作，你一个人干绝对要比三个人干慢许多。同时在整个kafka里面最新的版本之中支持有key=value结构传输，这样的传输模式对于消费者而言会更加容易处理数据。在进行消费者设计的时候，你的消费者可以使用的数据数量就是你的分区数量，也就是说如果你现在设置了三个分区，那么就表示可以使用三个消费者，反之，如果你只设计了一个分区，那么只能够有一个消费者。

### kafka消息处理流程

如果将消息以随机写的方式存入磁盘，就会按柱面、磁头、扇区的方式进行（寻址过程），缓慢的机械运动（相对内存）会消耗大量时间，导致磁盘的写入速度只能达到内存写入速度的几百分之一。为了规避随机写带来的时间消耗，kafka采取顺序写的方式存储数据。



如果在进行信息的写入的时候，所有的磁盘中的数据保存采用随机的方式进行存储，那么在进行读取的时候就一定会产生性能瓶颈，因为磁盘会出现寻址变慢的情况。

### offset与消息访问

* 消费者通过offset的方式来取得消息数据，利用offset偏移改变消息读取位置，可以实现历史消息读取；
* 为了避免频繁的大量小数据的磁盘IO操作，kafka采用批量读取模式处理消息；
* 在高负载状态下，为防止无效率的字节负载，kafka采用由producer，broker和consumer共享的标准化二进制消息格式，这样数据块就可以在它们之间自由传输，无需转换，降低了字节复制的成本开销。

在整个kafka里面还有一个比较逆天的性能（也是迫切需要的），传统的JMS设计的时候会存在有一个缺陷：当某一个消息消费了之后那么该消息将会被自动删除，而kafka不是，它在进行消息获取之后并不会立刻删除，而是会将消息暂停2天，2天后自动删除。在这样的状态下为了保证kafka的读取性能，单独设计了一个offset，可以理解为当前要操作的消息的下标。如果要想读取历史消息，只需要修改offset的指向即可实现。在一些组件整合过程之中，需要考虑好offset的设计，如果设计不当，那么会造成历史消息重复消费的问题。

在磁盘之中如果要不断的进行各种细小的琐碎操作，那么就有可能造成性能下降，所以在kafka里面专门设计有批量的数据操作，也就是说所有要消费的数据会批量读取，这样就减少了磁盘的操作量，性能也会得到提升。

在很多的消息系统里面由于其可以传输数据类型比较少（字符串），所以在每一次消费的时候都需要去判断数据的类型，这样自然会造成时间复杂度的提升，那么为了解决这样的问题，kafka约定了，你的消息的生产者一定要与消息的消费者协商好所传递的消息数据类型。

### kafka内存应用

* kafka采用了MMAP（Memory Mapped Files，内存映射文件）技术；
  + 现代操作系统都大量使用主存做磁盘缓存，一个现代操作系统可以将内存中的所有剩余空间用作磁盘缓存，而当内存回收的时候几乎没有性能的损失。
* kafka是基于JVM技术的应用：
  + 传统JVM技术缺点：
    - 对象的内存开销非常高，通常是实际要存储数据大小的两倍；
    - 随着数据的增加，java的垃圾收集也会越来越频繁并且缓慢。
  + kafka内存维护：文件系统+页缓存（例如：FIFO、CLOCK）:
    - 不使用进程内缓存，就腾出了内存空间，可以直接利用操作系统的页缓存来实现文件到物理内存的直接映射（容量翻倍），完成映射之后对物理内存的操作在适当的时候会被同步到硬盘上。
    - 由于使用操作系统页缓存，即使kafka重新启动，数据依然可以使用。
    - kafka会将数据写入到持久化日志中而不是刷新到磁盘。实际上它只是转移到了内核的页缓存。

kafka是基于jdk的实现，所以在kafka之中对于内存要想发挥高校，就不能纯粹依靠JVM进行管理，所以kafka还会使用到操作系统的内存空间，这样的好处是即使kafka崩盘了，但是数据不在JVM里面，数据也可以重新立刻恢复。

### 文件传输处理

* 消息的数据接收需要通过broker完成，broker维护的消息日志就是文件的目录，每个文件都是二进制保存，生产者和消费者使用相同的格式来处理。
* 维护公共的格式并允许优化最重要操作：网络传输持久性日志块。
  + 传统文件传输处理操作步骤：
    - 操作系统将数据从磁盘读入到内核空间的页缓存；
    - 应用程序将数据从内核空间读入到用户空间缓存中；
    - 应用程序将数据写回到内核空间到socket缓存中；
    - 操作系统将数据从socket缓冲区复制到网卡缓冲区，以便将数据经网络发出；
  + sendfile系统传输：现代的unix操作系统提供一个优化的代码路径，用于将数据从页缓存传输到socket；在linux中，通过sendfile系统调用来完成的。

文件传输是整个网络操作的核心所在，毕竟消息组件之中最需要有消费者的，而所有的消费者如果要想进行消息的获取，传统的做法一定要通过CPU读取磁盘数据，而后再通过CPU进行网络传输，那么这样的处理中间会经过CPU控制，自然会造成性能下降。