链滴

基于 Kafka 的消息中间件的综述与使用示例 - 中间件结课小论文

作者：qiuming

原文链接：https://ld246.com/article/1610803329340

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20191130.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
1.摘要
本文简要概述了 Kafka 的由来，并详细说明了 Kafka 的架构和设计原则。在充分了解了 Kafka
关原理的基础上，尝试使用虚拟机搭建了一个单机多实例的 Kafka 及其所需的 Zookeeper 集群。最
在已有集群基础上整合 Spring Boot,构建了一个简易的 Kafka 使用用例。
2.研究现状
近年来,随着信息技术与互联网应用的不断发展,全球数据总量也在呈现爆炸式的增长,大数据时代即将
临[1]。任何大型互联网公司都会产生大量的“日志”数据，如社交网站中登录、浏览、点击、喜欢、
享、评论以及网络服务本身的调用栈、调用延迟、错误报告以及一些机器运行指标数据。这些数据被
来越多的用到线上业务，实现诸如基于数据驱动的推荐、广告精准投放的功能。传统的企业级消息系
如 activemq，IBM Websphere MQ，Oracle Enterprise Messaging Service，TIBCO Enterprise
essage Service 已经存在很长时间了，主要作用是消息总线和异步解耦，但它们并不能无缝适配以上
需求。Apache Kafka 起源于 LinkedIn，后来于 2011 年成为 Apache 开源项目，然后于 2012 年成
Apache 项目的第一个类别。它是使用 Scala 和 Java 编写的。Apache Kafka,作为一种分布式的消
系统,具有可水平扩展和高吞吐率而被广泛的使用[2]。
3.Kafka 的概念体系和架构
3.1 主要概念和术语
数据源,在用户行为分析中主要是指移动端日志和 web 日志[3]。一个日志记录也称为记录或消息。当
Kafka 读取或写入数据时，将以事件的形式进行操作。从概念上讲，事件具有键、值、时间戳和可选
元数据标题。下面是一个示例事件：事件的键：“张三”，事件的值：“向李四支付了 200 元”，事
发生的时间戳记录：“2020 年 11 月 29 日，下午 2:06”。
生产者是那些向 Kafka 发布事件的客户端应用程序，而消费者是那些订阅即读取和处理这些事件的客
端应用程序。在 Kafka 中，生产者和消费者之间完全脱钩且彼此不可知，由此 Kafka 实现了高可伸
性。在传统应用开发中，服务之间通过 HTTP 和 RPC 调用实现通信，造成服务之间存在高耦合、请
易阻塞以及请求无缓冲等缺陷。特别是业务逻辑越发复杂，所需服务数量越多。随着服务规模的不断
张以及服务请求量的急剧增加,这些缺陷也被不断地放大从而影响着整个系统的性能和稳定性[4]。
Kafka 中的主题始终是多生产者和多用户的：一个主题可以有零个，一个或多个向其写入事件的生产
，以及零个，一个或多个订阅这些事件的使用者。可以根据需要随时读取主题中的事件-与传统的消
传递系统不同，使用后事件不会被删除。相反，您可以通过按主题配置设置来定义 Kafka 将事件保留
长时间，之后旧的事件将被丢弃。Kafka 的性能相对于数据大小实际上是恒定的，因此长时间存储数
是非常好的。
主题分区，这意味着主题分布在位于不同 Kafka Broker 上的多个存储单元中。数据的这种分布式放
对于可伸缩性非常重要，因为它允许客户端应用程序同时从多个 Broker 读取数据或向多个 Broker
入数据。将新事件发布到主题时，实际上会将其附加到主题的分区之一。具有相同事件密钥，例如，
户 ID 或车辆 ID 的事件被写入相同的分区，并且 Kafka 保证，给定主题分区的任何使用者都将始终
与写入时完全相同的顺序读取该分区的事件。
为了使数据具有容错性和高可用性，在地理区域或数据中心之间，可以复制每个主题，因此，总是有
个代理可以复制数据，以防万一出错。常见的生产设置中复制因子为 3，也就是始终会有三个数据副
。复制将在主题分区级别执行。
3.2 架构体系
Kafka 的架构图（使用 ProcessOn 绘制导出 png 格式）如下：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2021/01/image-b6d0162e.png?imageView2/2/interlace/1/format/jpg"><
p>
Kafka 架构图
Broker: Kafka 作为一个或多个服务器的集群运行，可以跨越多个数据中心或云区域。其中一些服务
构成了存储层，称为代理。其他服务器运行 Kafka Connect 来连续导入和导出数据作为事件流，以将
Kafka 与现有系统集成在一起，例如关系数据库以及其他 Kafka 群集。为了实现关键任务用例，Kafka
群集具有高度的可扩展性和容错能力：如果其任何服务器发生故障，其他服务器将接管其工作，以确
连续运行而不会丢失任何数据。
Consumer/Producer: 用户可以编写分布式应用程序和微服务，即使在网络问题或机器故障的情况下
也可以并行，大规模且以容错的方式读取，写入和处理事件流。Kafka 附带了一些这样的客户端，由
afka 社区提供的数十个客户端进行了扩展 ： 客户端可用于 Java 和 Scala，包括更高级别的 Kafka Str

ams 库，Go，Python，C / C ++ 和许多其他编程语言以及 REST API。
Topic:相当于传统消息系统 MQ 中的一个队列 queue，producer 端发送的 message 必须指定是发到哪个 topic 上.在一个大型的应用系统中，可以根据功能的不同，区分不同的 topic(订单的 topic,登的 topic,金额的 topic 等等)
Partition: 为了实现扩展性，一个非常大的 topic 可以分布到多个 broker（即服务器）上，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有序的 id（offset）。kafka 只保证按一个 partition 中的顺序将消息发给 consumer，不保证一个 t pic 的整体（多个 partition 间）的顺序。
4.Kafka 的设计原则
4.1 存储设计
Kafka 依赖于文件系统来存储和缓存消息。一个主题的每个分区就是逻辑上的一段消息集合。具体到理上，为了防止分区文件过大，Kafka 会将其进一步分成数据段。每次数据往最新的数据段中写，写设定容量后，就会新建一个段文件继续写。此外，为了提高写入性能，Kafka 会将事件记录在内存中行缓存，只有事件数量达到设定值或者缓存数据的大小达到设定值时，才会将数据写入外存中。为了证可靠性，只有数据已写入外存后，才会将其通知给消费者。
和传统的消息系统不同，Kafka 在设计时采用了文件追加的方式来写入消息，即只能在段文件的尾部加新的消息，并且也不允许修改已写入的消息，这种方式属于典型的顺序写盘的操作，顺序写磁盘的度要比随机写内存的速度更块。每个消费者总是顺序的去消费每个分区的数据。创建 topic 时，Kafka 为该 topic 的每个分区在文件系统中创建一个对应的子目录，名字就是-< 分区号 >。所以倘若一个 topic 名为 test，有两个分区，那么在文件系统中 kafka 会创建两个子目录：test-0 和 test-1 每个日志子目录的文件都是由若干组分段构成。每个分段中的日志文件以 log 后缀都有两个对应的索文件：偏移量索引文件（".index"后缀）和时间戳索引文件（以".timeindex"为后缀）。这些索引文帮助 Broker 更快的定位消息所在的物理文件位置。
4.2 传输设计
网络传输是比较消耗时间和系统资源的，因此 Kafka 在网络传输的设计采取了较多的优化。生产者可批量传输数据到 Broker,消费者虽然看起来是逐条消费的，但在 API 实现的底层也是批量拉取一定大的消息供消费者消费。Kafka 在操作系统层面使用页缓存加快磁盘读写，在传统上，一条数据从本地件送到 socket 上通常包含以下几个过程：
(1)从外存中读入数据到操作系统的页缓存。
(2)从页缓存拷贝数据到应用缓冲区中。
(3)从应用缓冲区拷贝到内核缓冲区。
(4)从内核缓冲区拷贝到 socket。
这些过程涉及四次数据拷贝和两次系统调用，非常浪费资源。在 Linux 操作系统中，存在一个 sendfil 零拷贝技术的 API，能够直接将数据从文件传送到 socket 中。利用此 API，可以省去步骤(2)(3)中引的两次数据拷贝和一次系统调用，由此使得 Kafka 可以将数据从 Broker 的段文件中高效的传输给消者，极大提高了系统性能。
4.3 无状态的 Broker
与其他消息队列不同，Kafka 的 Broker 仅提供基于 offset 的读取方式，不会维护各个消费者当前已费消息的 offset 值，而是由消费者各自维护当前读取的进度。消费者读取数据时告诉 Broker 请求消的起始 offset 值，Broker 将之后的消息流式发送过去。从而使 Broker 的设计可以相对简化，不用护过多状态。但是这样做也存在一些问题，Broker 不知道消费者消费到了哪里，也就不知道哪些消需要被清理。对此，Kafka 使用了一个简单的策略，对消息设定有效期，每个主题可以设置不同的有期。这个简单的策略适用于绝大多数情况。大量存储和拉取的设计带来的另外一个重要的好处是消费可以主动选择进行回退消费。这个需求看起来违背了通常消息队列的定义，然而在很多情况下却非常必要。举例来说，当消费者进程由于错误而挂掉后，可以在恢复后有选择的对挂掉前后的数据重新消。这对将 ETL 数据导入 Hadoop 等数据仓库之类的场景非常重要。但是对于 Kafka 来说，消费者只要记住消费到的 offset 即可，下次重启后再从该 offset 后开始拉取。但是对于传统没有大量缓存的息队列来说，可能这部分数据就永远的丢了。
4.4 分布式协调一致
多个生产者和消费者在分布式环境中的行为，对于生产者，其发送数据时，可以将其随机发送到一个区所在 Broker；也可以根据 key 以及作用于 key 上的路由函数，将其发送到某特定分区机器 Broker 上。对于消费者,行为稍复杂。
在 Kafka 中，多个消费者或消费者实例可以组成一个消费者组。它们共享一个公共的 ID，即 group I ，组内的所有消费者协调在一起来消费订阅主题的所有分区。对于消费者组的理解需要明白三点：一.

费者组下可以有一个或多个消费者实例，消费者实例可以是一个进程，也可以是一个线程。二是组 ID 是一个字符串，唯一标识一个消费者组。三是消费组下订阅的主题的每个分区只能分配给组内一个消者。
Kafka 引入了一个高可用的一致性服务 Zookeeper。Zookeeper 的 API 很像文件系统，是以前缀树形式组织的 KV 也就是键值对，K 键是路径，以’/’来区分层次，V 值可以是任何可序列化的值存储该 API 支持创建一个路径、给一个路径设置值、读取路径的值、删除一个路径、列出某个路径下所有节点的值。此外，Zookeeper 还具有以下特性：
(1)客户端可以向某个路径注册一个回调函数，以监听该路径的值或其孩子节点的变动。
(2)路径可以被创建为易失的，即当所有该路径的客户端消失后，该路径及值会被自动的移除。
(3) Zookeeper 使用一致性协议将其数据进行多机备份，使其服务具有高可靠性和高可用性。
Kafka 使用 Zookeeper 完成以下几个任务：
(1)监控 Brokers 和消费者的增删。
(2)当出现 Brokers 或者消费者的增删时，启动消费再平衡任务。
(3)维护消费者的间关系状态，跟踪每个分区的消费偏移量。
具体来说，代理和消费者会将自身元信息注册到 Zookeeper 的注册表中，其中包括：代理的主机名端口以及在其上的主题和分区。每个消费者组都在 Zookeeper 中有一个相关联的所有权注册表和偏量注册表。我们将消费者消费某个分区的行为称为占有，所有权注册表即记录了消费者与其占有的分间的对应关系。其中，路径名标识一个分区，记录值是该分区的拥有者。偏移量记录表记录了该消费组所有订阅的 topic 对应的每个分区的消费进度。
当一个新的消费者组创建时，注册表中没有任何的偏移量记录。这时，使用 Broker 提供的 API，该费者组可以针对每个分区选择从最小的偏移量或者最大的偏移量进行消费。
4.5 数据交付保证
Kafka 提供 3 种消息传输一致性语义：最多 1 次，最少 1 次，恰好 1 次。最少 1 次：可能会重传数，有可能出现数据被重复处理的情况;最多 1 次：可能会出现数据丢失情况;
恰好 1 次：并不是指真正只传输 1 次，只不过有一个机制。确保不会出现“数据被重复处理”和“数丢失”的情况。
Kafka 保证来自于同一个分区的消息是保序的，即 offset 大小顺序，但是不同分区之间的顺序是不保的。
5.Kafka 特性与安全
面向存储的消息队列：意味在近实时的情况下能够将传统消息队列的存储增加几个数量级。实现原理充分利用了磁盘的顺序写和操作系统自身的缓存；此外为了提高访盘、传输效率，使用了文件分段、首索引、零拷贝和批量拉取等技术。
灵活的生产消费方式：基于主题粒度的发布订阅式架构，并且既支持组内多消费者互斥消费，也支持同消费者组间的重复消费。这里面涉及到消息队列的两个设计选择：pull 式消费以及客户端侧存储消进度。为了简化实现，消费时，每个分区最多为一个消费者所消费。
Zookeeper 存储元信息：利用分布式一致性组件 Zookeeper 以注册表的形式存储系统元信息，包括roker 和消费者的存活信息、消费者和分区间的对应关系、每个分区的消费进度等等。Zookeeper 作一个前缀树形式组织 KV、支持发布订阅的高可用组件，可以满足 Kafka 进行消费协调和进度保存的作需求。
简洁强大的消费接口：Kafka 的客户端一般提供两层接口抽象。包括无需关注分区和偏移量信息的高（high-level）简单读写接口，以及可以灵活控制分区组织和消费进度的低层（low-level）接口。<b>
随着越来越多的安全漏洞、数据泄露等问题的爆发,安全正成为系统选型不得不考虑的问题,Kafka 由其安全机制的匮乏,也导致其在数据敏感行业的部署存在严重的安全隐患[5]。在拥有强大性能的同时，afka 在安全方面的问题需要在实际部署应用时特别注意，以免造成生产事故。
6.应用场景
Kafka 有着丰富的使用场景并且已经得到了广泛而深入的应用。例如：在证券交易所，银行和保险中用于实时处理付款和金融交易。在物流和汽车行业，实时跟踪和监视汽车，卡车，车队和货运。在物网应用中，连续捕获和分析来自 IoT 设备或其他设备（例如工厂和风电场）中的传感器数据。在零售酒店和旅游行业以及移动应用程序中，收集并立即响应客户的交互和订单。监测患者的医院护理情况预测病情变化，以确保在紧急情况下及时得到治疗。还可以连接，存储和提供公司不同部门产生的数，用作数据平台，事件驱动的体系结构和微服务的基础。
7.使用示例
7.1 搭建 Kafka 和 Zookeeper 单机多实例环境

本测试环境使用 Linux 操作系统环境，发行版本为 CentOS-7-x86_64-Minimal-1908，虚拟化软件为 VMware Workstation Pro。为虚拟机分配 1GB 内存，硬盘容量 20GB，使用 NAT 模式联网，实体与虚拟机器在同一网段下，可以相互访问，为后续本机访问虚拟机中的 Kafka 集群提供网络基础条件
(1)　搭建虚拟机环境，配置虚拟机静态 IP。
配置 Linux 主机网卡设置静态 IP：vim /etc/sysconfig/network-script/ifcfg-ens33,保存配置重启网络：systemctl restart network，以下是配置详单。
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPADDR=192.168.10.12
NETMASK=255.255.255.0
GATEWAY=192.168.10.2
DNS1=114.114.114.114
DNS2=8.8.8.8
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=30da6508-90d0-4643-8f65-efa88eec2481
DEVICE=ens33
ONBOOT=yes
(2)　获取 Zookeeper 安装包 wget https://xxx/apache-zookeeper-3.6.2-bin.tar.gz
(3)解压安装包 tar –zxvf apache-zookeeper-3.6.2-bin.tar.gz
(4)修改配置文件 zoox.cfg,参考如下：
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/opt/data/zookeeper/zkserver1
clientPort=2181
server.1=127.0.0.1:2888:3888
server.2=127.0.0.1:2889:3889
server.3=127.0.0.1:2890:3890
其中 dataDir 设置三个配置文件写三个不同的目录，事先创建好,在每个目录下创建 myid 文件写入数序号，例如第一个实例写入 1。clienPort 每个实例各不相同，server.x 各个端口不能相同，因为我们单机运行多个实例。
(5)运行 Zookeeper 在其目录里执行命令：
bin/zkServer.sh start conf/zoox.cfg
(6)执行 netstat –ntpl 查看是否程序监听相应端口
(7)获取 Kafka 安装包
wget https://xxx/kafka_2.13-2.6.0.tgz
(8)解压
tar –zxvf kafka_2.13-2.6.0.tgz
(9)修改配置文件
vim config/serverx.properties　需修改的配置：
broker.id=1
listeners=
log.dir=/opt/data/kafka/server1
zookeeper.connect=127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183

(10)启动三个 Kafka 实例，在其目录中执行
bin/kafka-server-start.sh –daemon config/serverx.properties
(11)运行 jps,检查 zookeeper 和 kafka 服务是否正常运行。
7.2 Spring Boot 整合 Kafka 使用用例
使用 IDEA 新建 Spring Boot 项目，勾选 Spring Boot 整合 Kafka 的依赖库，maven 文件截图如下
新建类 KafkaProducer 作为一个 Controller。
创建方法 saycallback 接受用户 get 请求并设置路径参数，将值作为消息发送到 Kafka 集群的 topic1 主题下，注册回调函数用于处理发送结果。
新建类 KafkaConsumer 用于消费 Kafka 集群中的消息。使用 @Component 注解标识此类为组件此类中有一个 KafkaTemplate 类型的属性，该属性值由 spring 自动注入，对象的构造由 kafka 支库自动读取配置文件完成构建。spring 将注入此消费者类到 IOC 容器。
在类中创建方法 getMessage3 接受消息并打印，由注解 @KafkaListener 可知消费者的 id 为 consumer2，消费者组 id 为 consumer-group2，消费的主题为 topic1。通过注解 @KafkaListener 可以定消费的主题，分区以及初始偏移值，也可以指定消费多个
主题。
以下为示例代码和配置文件截图：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-91d3e964.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-6c89b927.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-0ab00e48.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-de8b4682.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-5449c714.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-001a59cf.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-5a7b065e.png?imageView2/2/interlace/1/format/jpg">
环境搭建截图：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-9d52169f.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-cd78e2db.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-b896f80f.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-0c5adef0.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-a8210bbc.png?imageView2/2/interlace/1/format/jpg">

<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-6f163233.png?imageView2/2/interlace/1/format/jpg">
以下是运行结果截图：
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-2e6e9aea.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-46cfcc07.png?imageView2/2/interlace/1/format/jpg">
<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https://b3logfile.com/file/2021/01/image-64d0611d.png?imageView2/2/interlace/1/format/jpg">
8.参考文献
[1]牛牧. 基于 Kafka 的大规模流数据分布式缓存与分析平台[D].吉林大学,2016.
[2]王岩,王纯.一种基于 Kafka 的可靠的 Consumer 的设计方案[J].软件,2016,37(01):61-66.
[3]费秀宏. 基于 Kafka 的日志处理平台的研究[D].吉林大学,2017.
[4]卢帅. 基于 Kafka 的消息发布订阅服务的设计与实现[D].南京大学,2018.
[5]孙元浩.如何构建安全的 Kafka 集群[J].电信网技术,2015(08):10-14.