# [[置顶] Kafka学习总结(一)——Kafka简介](http://blog.csdn.net/gongxinju/article/details/72622204)

标签： [分布式](http://www.csdn.net/tag/%e5%88%86%e5%b8%83%e5%bc%8f" \t "http://blog.csdn.net/gongxinju/article/details/_blank)[kafka-消息发布与订阅](http://www.csdn.net/tag/kafka-%e6%b6%88%e6%81%af%e5%8f%91%e5%b8%83%e4%b8%8e%e8%ae%a2%e9%98%85" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

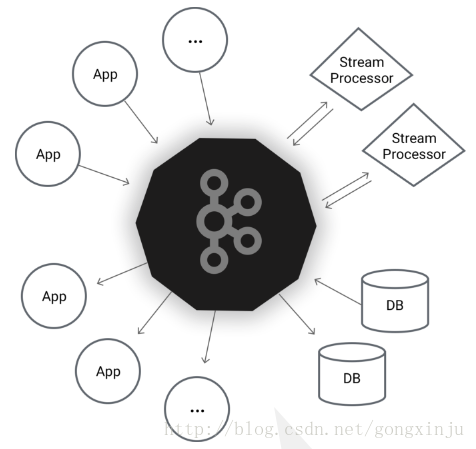
2017-05-22 10:49 216人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/72622204" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/72622204" \l "report" \o "举报)

IMG_256 分类：

kafka-c

版权声明：本文为博主原创文章，未经博主允许不得转载。

**1.1、什么是kafka？**

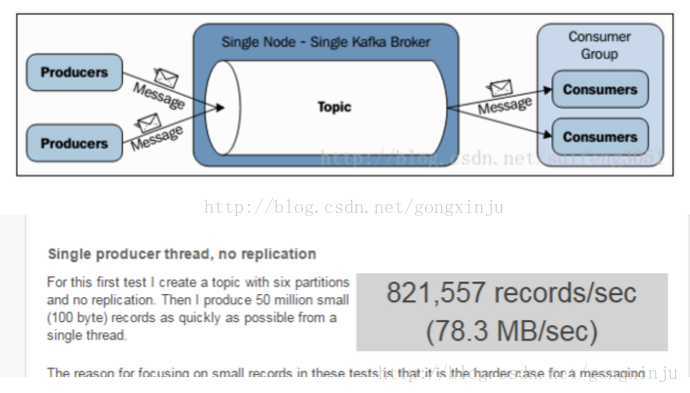
Kafka是由linkedin开发的一个分布式的(发布-订阅)消息系统，使用**[Scala](http://lib.csdn.net/base/scala" \o "Scala知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**编写，它以可水平扩展和高吞吐率而被广泛使用。   


**1.2、Kafka创建背景**

当今社会各种应用系统诸如商业、社交、搜索、浏览等像信息工厂一样不断的生产出各种信息，在**[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**时代，我们面临如下几个挑战：

如何收集这些巨大的信息

如何分析它

如何及时做到如上两点   
以上几个挑战形成了一个业务需求模型，即生产者生产（produce）各种信息，消费者消费（consume）（处理分析）这些信息，而在生产者与消费者之间，需要一个沟通两者的桥梁-消息系统。从一个微观层面来说，这种需求也可理解为不同的系统之间如何传递消息。   


**1.3、使用场景**

一、Message   
对于一些常规的消息系统，kafka是个不错的选择；partions/replication和容错，可以是kafka具有良好的扩展性和性能优势。不过和JMS比，没有“事务性”“消息确认机制”，”消息分组”等企业级特性。

二、Websit   
kafka可以作为”网站活性跟踪”的最佳工具；可以将网页/用户操作信息发送到kafka中。

三、Log Aggregation   
kafka的特性决定它非常适合作为”日志收集”，app可以将操作日志发送到kafka集群中，而不是保存在本地或者DB中。consumer端可以存储和分析日志。

**1.4、kafka主要设计目标**

以时间复杂度为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的访问性能高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒100K条消息的传输支持Kafka Server间的消息分区，及分布式消费，同时保证每个partition内的消息顺序传输同时支持离线数据处理和实时数据处理。

**1.5、为何要用消息队列 (kafka)**

**冗余**   
有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。在被许多消息队列所采用的”插入-获取-删除”范式中，在把一个消息从队列中删除之前，需要你的处理过程明确的指出该消息已经被处理完毕，确保你的数据被安全的保存直到你使用完毕。

**灵活性 & 峰值处理能力**   
在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见；如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。

**可恢复性**   
当体系的一部分组件失效，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。而这种允许重试或者延后处理请求的能力通常是造就一个略感不便的用户和一个沮丧透顶的用户之间的区别。

**送达保证**   
消息队列提供的冗余机制保证了消息能被实际的处理，只要一个进程读取了该队列即可。在此基础上，IronMQ提供了一个”只送达一次”保证。无论有多少进程在从队列中领取数据，每一个消息只能被处理一次。这之所以成为可能，是因为获取一个消息只是”预定”了这个消息，暂时把它移出了队列。除非客户端明确的表示已经处理完了这个消息，否则这个消息会被放回队列中去，在一段可配置的时间之后可再次被处理。

**顺序保证**   
在大多使用场景下，数据处理的顺序都很重要。消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。IronMO保证消息通过FIFO（先进先出）的顺序来处理，因此消息在队列中的位置就是从队列中检索他们的位置

**异步通信**   
很多时候，你不想也不需要立即处理消息。消息队列提供了异步处理机制，允许你把一个消息放入队列，但并不立即处理它。你想向队列中放入多少消息就放多少，然后在你乐意的时候再去处理它们。

**1.6、消息队列对比-ActiveMQ、RabbitMQ、Kafka**

**ActiveMQ**   
重量级的老牌儿MQ，诞生自**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**生态，功能完备，相关的介绍很多，这里不再赘述。

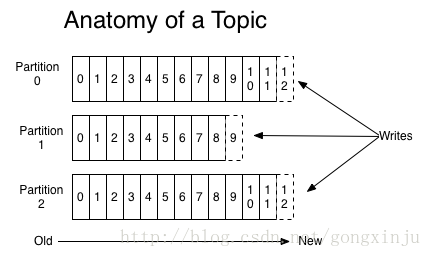
**RabbitMQ**   
同样是老牌儿MQ，基于erlang实现，语言无关，功能完备，诞生自金融领域。

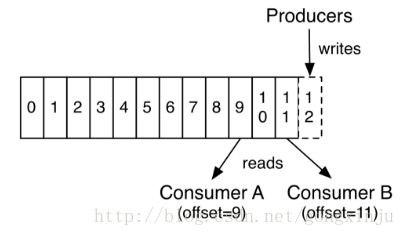
**Kafka**   
MQ中的后起之秀，在很多场景下都超越了前辈，诞生自**[Hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**生态，在大数据的支持方面，目前无人能出其右。

**横向对比**   


**1.7、Kafka组件、术语**

**一、Topics/logs**   
一个Topic可以认为是一类消息，每个topic将被分成多个partition(区)，每个partition在储存层面是append log文件。任何发布到此partition的消息都会被直接追加到log文件的尾部，每条消息在文件中的位置称为offset(偏移量)，offset为一个long整型数字，它是唯一标记一条消息。Kafka并没有提供其他额外的索引机制来存储offset，因为在kafka中几乎不允许对消息进行“随机读写”。

   
在kafka中，即使消息被消费，消息仍然不会被立即删除。日志文件将会根据broker中的配置要求，保留一定的时间之后删除；比如log文件保留2天，那么两天后，文件会被清除，无论其中的消息是否被消费。Kafka通过这种简单的手段，来释放磁盘空间，以及减少消息消费之后对文件内容改动的磁盘IO开支。

   
对于consumer而言，它需要保存消费消息的offset，对于offset的保存和使用，由consumer来控制；当consumer正常消费消息时，offset将会“线性”的向前驱动，即消息将依次顺序被消费。事实上consumer可以使用任意顺序消费消息，它只需要将offset重置为任意值，Offset保存在zookeeper中。

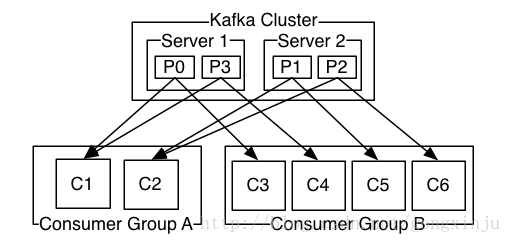
Kafka集群几乎不需要维护任何consumer和producer状态消息，这些信息由zookeeper保存；因此producer和consumer的客户端实现非常轻量级，它们可以随意离开，而不会对集群造成额外的影响。   
Partitions的设计目的有多个。最根本原因是kafka基于文件储存。通过分区，可以将日志内容分散到多个server上，来避免文件尺寸达到单机磁盘的上限，   
每个partitions都会被当前server保存；可以将一个topic切分多任意多个partitions来保存消息。此外越多的partitions意味着可以容纳更多的consumer，有效提升并发消费的能力。

**二、Distribution–partitions**   
一个topic的多个partitions，被分布在kafka集群中的多个server上，每个server负责partitions中消息的 读写操作；此外kafka还可以配置partitions需要备份的个数(replicas)，每个partition将会被备份到多台机器上，以提高可用性。

基于replicated方案，那么就意味着需要对多个备份进行调度，每个partition都有一个server为“leader”；leader负责所有的读写操作，如果leader失效，那么将会有其他follower来接管(成为新的leader)；follower只是单调的和leader跟进，同步消息即可。由此可见作为leader的server承载了全部的请求压力，因此从集群的整体考虑，有多少个partitions就意味着有多少个“leader”，kafka会将“leader”均衡的分散在每个实例上，来确保整体的性能稳定。

* 发送到partitions中的消息将会按照它接收的顺序追加到日志中。
* 对于消费者而言，它们消费消息的顺序和日志顺序一致。
* 如果topic的“replication factor”为n，那么允许n-1个kafka实例失效。

**三、Producers**   
Producer将消息发布到指定的topic中，同时producer也能决定将此消息归属哪个partition；比如基于“round-robin”方式或者通过其他的一些**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**等。

**四、consumers**   
每个consumser属于一个consumer group;反过来说，每个group中可以有多个consumer。发送到Topic的消息，只会被订阅此Topic的每个group中的一个consumer消费。   
如果所有的consumer都有不同的group，那这就是“发布-订阅”，消息将会广播给所有的消费者。   
在kafka中，一个partition中的消息只会被group中的一个consumer消费；每个group中的consumer消息消费相互独立；我们可以认为一个group是一个“订阅”者，一个Topic中的每个partions，只会被一个“订阅者”中的一个consumer消费，不过一个consumer可以消费多个partitions中的消息。kafka只能保证一个partition中的消息被某个consumer消费时，消息是顺序的。事实上，从Topic角度来说，消息仍不是有序的。   
kafka的设计原理决定，对于一个topic，同一个group中不能有多于partitions个数的consumer同时消费，否则将意味着某些consumer将无法得到消费。   


**五、Leader**   
Partition中负责消息读写的节点;Leader是从Partition的节点中随机选取的。每个Partition都会在集中的其中一台服务器存在Leader。一个Topic如果有多个Partition，则会有多个Leader。

**六、ReplicationFactor**   
一个Partition中复制数据的所有节点,包括已经挂了的；数量不会超过集群中broker的数量

**七、isr**   
ReplicationFactor的子集,存活的且和Leader保持同步的节点;

# [[置顶] Kafka学习总结(二)——Kafka设计原理](http://blog.csdn.net/gongxinju/article/details/72599938)

标签： [kafka](http://www.csdn.net/tag/kafka" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

2017-05-21 18:02 147人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/72599938" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/72599938" \l "report" \o "举报)

IMG_256 分类：

kafka消息队列（3） IMG_257

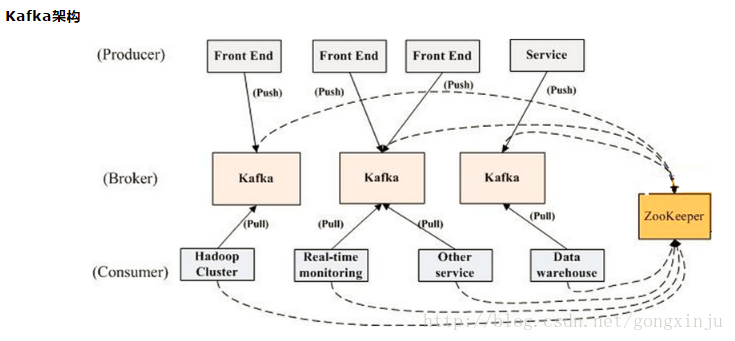
版权声明：本文为博主原创文章，未经博主允许不得转载。

**2.1、kafka的场景和[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**

在我们大量使用分布式**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**、分布式计算集群的时候，是否会遇到这样的一些问题：

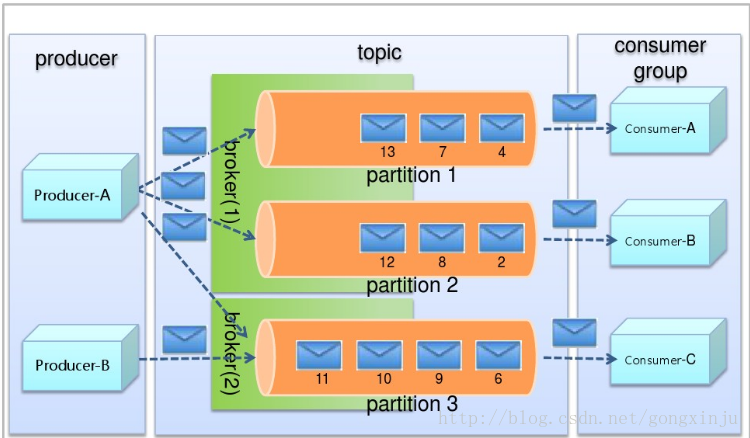
我想分析一下用户行为pageviews，以便我能设计出更好的广告位

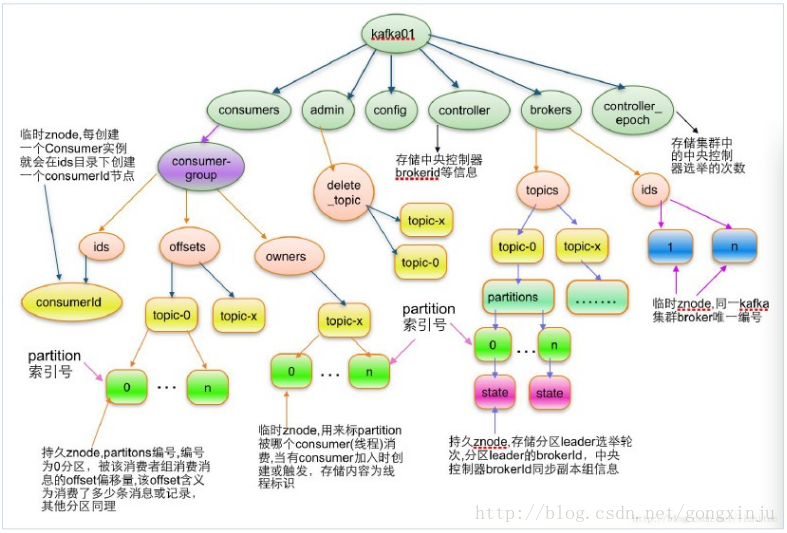
我想对用户的搜索关键词进行统计，分析出当前的流行趋势。这个很有意思，在经济学上有个长裙理论，就是说，如果长裙的销量高了，说明经济不景气了，因为姑娘们没钱买各种丝袜了。

有些数据，我觉得存数据库浪费，直接存硬盘又怕到时候操作效率低。   
kafka在实际应用中被大量的用于日志系统，首先我们要明白什么是消息系统，在kafka官网上对kafka的定义叫：A distributed publish-subscribe messaging system(一个分布式发布-订阅消息传递系统)。publish-subscribe是发布和订阅的意思，所以更准确的说kafka是一个消息订阅和发布的系统。publish- subscribe这个概念很重要，因为kafka的设计理念就可以从这里说起。   


多个broker协同合作，producer和consumer部署在各个业务逻辑中被频繁的调用，三者通过zookeeper管理协调请求和转发。这样一个高性能的分布式消息发布与订阅系统就完成了。图上有个细节需要注意，producer到broker的过程是push，也就是有数据就推送到broker，而   
consumer到broker的过程是pull，是通过consumer主动去拉数据的，而不是broker把数据主动发送到consumer端的。

简单说下整个系统运行的顺序：   
1. 启动zookeeper的server   
2. 启动kafka的server   
3. Producer如果生产了数据，会先找到broker，热后broker再连接zookeeper，然后将数据存放进broker   
4. Consumer如果要消费数据，会先通过zookeeper找对应的broker，然后消费

**kafka集群运行的流程图：**   


**kafka在zookeeper中的数据：**   


**2.2、设计思想**

**consumer group：**各个consumer可以组成一个组，每个消息只能被组中的一个consumer消费，如果一个消息可以被多个consumer消费的话，那么这些consumer必须在不同的组。

**消息状态：** 在Kafka中，消息的状态被保存在consumer中，broker不会关心哪个消息被消费了被谁消费了，只记录一个offset值（指向partition中下一个要被消费的消息位置），这就意味着如果consumer处理不好的话，broker上的一个消息可能会被消费多次。

**消息持久化：** Kafka中会把消息持久化到本地文件系统中，并且保持极高的效率。

**消息有效期：** Kafka会长久保留其中的消息，以便consumer可以多次消费，当然其中很多细节是可配置的。

**批量发送：** Kafka支持以消息集合为单位进行批量发送，以提高push效率。

**push-and-pull：** Kafka中的Producer和consumer采用的是push-and-pull模式，即Producer只管向broker push消息，consumer只管从broker pull消息，两者对消息的生产和消费是异步的。

**Kafka集群中broker之间的关系：** 不是主从关系，各个broker在集群中地位一样，我们可以随意的增加或删除任何一个broker节点。

**负载均衡方面：** Kafka提供了一个 metadata API来管理broker之间的负载（对Kafka0.8.x而言，对于0.7.x主要靠zookeeper来实现负载均衡）。

**同步异步：** Producer采用异步push方式，极大提高Kafka系统的吞吐率（可以通过参数控制是采用同步还是异步方式）。

**分区机制partition：** Kafka的broker端支持消息分区，Producer可以决定把消息发到哪个分区，在一个分区中消息的顺序就是Producer发送消息的顺序，一个主题中可以有多个分区，具体分区的数量是可配置的。分区的意义很重大，后面的内容会逐渐体现。

**离线数据装载：** Kafka由于对可拓展的数据持久化的支持，它也非常适合向**[Hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**或者数据仓库中进行数据装载。

**2.3、Kafka的设计特色**

**Kafka在设计上有以下几个特色：**

消息数据通过磁盘线性存取   
强调吞吐率   
消费状态由消费者自己维护   
分布式

**消息数据通过磁盘线性存取：**

特性应该是Kafka最让惊叹的特性。在我们的认识中，硬盘较内存的数据处理速度(读写)是慢很多的，所以基本所有设计数据处理的程序都是尽量使用内存。然而Kafka的设计者在经过一番调研**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**后，大胆地采用了全硬盘存取消息数据的方案，他们的主要依据是：

硬盘在线性读写的情况下性能优异：   
有测试表明在一个6 7200rpm 的SATA RAID-5 阵列上，线性写可以达到300MB/Sec，而随机写只有50KB/Sec。线性读写之所以可以有这么优异的表现与文件系统是分不开的，因为对写操作，**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**一般会进行缓冲，而对于读操作，操作系统会进行预抓取的缓冲操作，这会极大地提高读取效率。又由于这一层缓存操作是在OS级的，也就意味着即便Kafka挂掉了重启，缓存也不会失效。

减少JVM的GC触发：   
JVM中的对象会占用除实际数据外的较多空间（如类的信息等等），结构不够紧凑，浪费空间。而当内存中维护的消息数据逐渐增多后，GC便会被频繁触发，这会极大影响应用的响应速度。因此，舍弃内存，使用磁盘可以减少GC触发带来的影响。

Kafka论文中与ActiveMQ等消息队列的性能比较也进一步肯定了Kafka的这一设计理念。

**强调吞吐率：**

Kafka的设计初衷便是要能处理TB级的数据，其更强调的是吞吐率。吞吐率表现在读和写两个方面，Kafka在这两个方面都做了很多优化。

写   
前面的文章中我们曾经提到过kafka存储消息数据的形式，如下图： topic-partition–0000.kafka –1024.kafka –2048.kafka kafka在启动时会将该文件夹中的所有文件以channel形式打开，并且只有最后一个kafka文件以读写形式打开，其他都以只读方式打开，新到的消息都直接append到最后一个kafka文件中，这样就实现了顺序写。而前面已经提到，顺序写的性能是极高的，这样写的性能就有了保障。

读   
Kafka在读方面使用了sendfile这个高级系统函数，也即zero-copy技术，感兴趣的同学可以去阅读IBM的文章。 这项技术通过减少系统拷贝次数，极大地提高了数据传输的效率。简单说明如下：

程序读文件内容，调用socket发送内容，过程涉及以下几个步骤   
调用syscall，如read，陷入内核，读文件内容到内核缓存区pagecache   
程序将文件内容由内核缓存区拷贝到用户空间内存调用syscall，如socket write函数，陷入内核，将用户空间的内容拷贝的socket缓冲区内存，准备发送。将socket缓冲区内存拷贝到NIC（Network Interface Controller）缓冲区，发送数据。其中涉及了2次syscall和4次数据拷贝。相信读者一定发现这4次数据拷贝是显然没有必要的，直接把数据从pagecache的内核缓存区读到NIC缓冲区不就可以了吗？sendfile系统函数做的就是这件事情。显然这会极大地提升数据传输的效率。在**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**中对应的函数调用是FileChannle.transferTo。   
另外，Kafka还通过多条数据压缩传输、存取的办法来进一步提升吞吐率。

**消费状态由消费者自己维护：**

Kafka消息数据的消费状态由消费者自己维护，原因这里不再详述了，感兴趣的同学可以去阅读文章开头的参考文献。这里简单说一下这么做的好处：   
去除了服务端维护消费状态的压力。   
提升了消费者存储消费状态的自由度，如存储位置，可以存zookeeper数据库或者HDFS中，根据消费者自身的需求即可。   
针对特殊需求，如消息消费失败，消费者可以回滚而重新消费消息。

**分布式：**

Kafka的broker producer和consumer都是可分布的，其实现是通过zookeeper来维护集群中这三者的信息，从而实现三者的交互，详细实现留待以后再说。

通过上面的介绍相信大家已经对Kafka的设计有了一定的了解。这样的设计完全可以整合offline**[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/gongxinju/article/details/_blank)**处理和online实时处理的需求。对于offline处理，kafka支持将数据批量的迁移到hdfs中，而对于online的实时处理，只要合理的设置consumer处理特性就可以很容易地做到。LinkedIn在其文章中曾举到一个kafka的应用案例，这里与大家分享下。使用kafka来更新索引，当用户更新了自己的数据后，producer便产生一条消息发送到broker，consumer会即时获取该数据，进而更新索引，这样也就可以做到搜索时数据”秒级更新“的特性了。

# [[置顶] Kafka学习总结(三)——Kafka的message存储数据结构](http://blog.csdn.net/gongxinju/article/details/72672375)

2017-05-24 01:20 398人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/72672375" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/72672375" \l "report" \o "举报)

IMG_256 分类：

kafka（2） IMG_257

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)](http://blog.csdn.net/gongxinju/article/details/72672375" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/gongxinju/article/details/72672375" \o "展开)

## 引言

Kafka中的Message是以topic为基本单位组织的，不同的topic之间是相互独立的。每个topic又可以分成几个不同的partition(每个topic有几个partition是在创建topic时指定的)，每个partition存储一部分Message。借用官方的一张图，可以直观地看到topic和partition的关系。  


partition是以文件的形式存储在文件系统中，比如，创建了一个名为page\_visits的topic，其有5个partition，那么在Kafka的数据目录中(由配置文件中的log.dirs指定的)中就有这样5个目录: page\_visits-0， page\_visits-1，page\_visits-2，page\_visits-3，page\_visits-4，其命名规则为<topic\_name>-<partition\_id>，里面存储的分别就是这5个partition的数据。

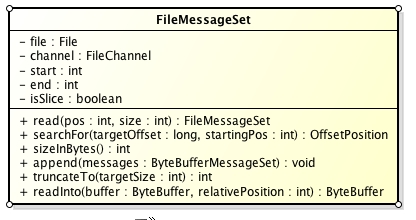
接下来，本文将分析partition目录中的文件的存储格式和相关的代码所在的位置。

## 3.1、Partition的数据文件

Partition中的每条Message由offset来表示它在这个partition中的偏移量，这个offset不是该Message在partition数据文件中的实际存储位置，而是逻辑上一个值，它唯一确定了partition中的一条Message。因此，可以认为offset是partition中Message的id。partition中的每条Message包含了以下三个属性：

* offset
* MessageSize
* data

其中offset为long型，MessageSize为int32，表示data有多大，data为message的具体内容。它的格式和[Kafka通讯协议](http://blog.csdn.net/jewes/article/details/42744855" \o "Kafka通讯协议" \t "http://blog.csdn.net/gongxinju/article/details/_blank)中介绍的MessageSet格式是一致。

Partition的数据文件则包含了若干条上述格式的Message，按offset由小到大排列在一起。它的实现类为FileMessageSet，类图如下：  
  
它的主要方法如下：

* append: 把给定的ByteBufferMessageSet中的Message写入到这个数据文件中。
* searchFor: 从指定的startingPosition开始搜索找到第一个Message其offset是大于或者等于指定的offset，并返回其在文件中的位置Position。它的实现方式是从startingPosition开始读取12个字节，分别是当前MessageSet的offset和size。如果当前offset小于指定的offset，那么将position向后移动LogOverHead+MessageSize（其中LogOverHead为offset+messagesize，为12个字节）。
* read：准确名字应该是slice，它截取其中一部分返回一个新的FileMessageSet。它不保证截取的位置数据的完整性。
* sizeInBytes: 表示这个FileMessageSet占有了多少字节的空间。
* truncateTo: 把这个文件截断，这个方法不保证截断位置的Message的完整性。
* readInto: 从指定的相对位置开始把文件的内容读取到对应的ByteBuffer中。

我们来思考一下，如果一个partition只有一个数据文件会怎么样？

1. 新数据是添加在文件末尾（调用FileMessageSet的append方法），不论文件数据文件有多大，这个操作永远都是O(1)的。
2. 查找某个offset的Message（调用FileMessageSet的searchFor方法）是顺序查找的。因此，如果数据文件很大的话，查找的效率就低。

那Kafka是如何解决查找效率的的问题呢？有两大法宝：1) 分段 2) 索引。

## 3.2、数据文件的分段

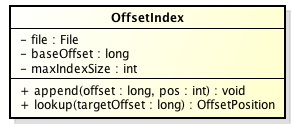
Kafka解决查询效率的手段之一是将数据文件分段，比如有100条Message，它们的offset是从0到99。假设将数据文件分成5段，第一段为0-19，第二段为20-39，以此类推，每段放在一个单独的数据文件里面，数据文件以该段中最小的offset命名。这样在查找指定offset的Message的时候，用二分查找就可以定位到该Message在哪个段中。

## 3.3、为数据文件建索引

数据文件分段使得可以在一个较小的数据文件中查找对应offset的Message了，但是这依然需要顺序扫描才能找到对应offset的Message。为了进一步提高查找的效率，Kafka为每个分段后的数据文件建立了索引文件，文件名与数据文件的名字是一样的，只是文件扩展名为.index。  
索引文件中包含若干个索引条目，每个条目表示数据文件中一条Message的索引。索引包含两个部分（均为4个字节的数字），分别为相对offset和position。

* 相对offset：因为数据文件分段以后，每个数据文件的起始offset不为0，相对offset表示这条Message相对于其所属数据文件中最小的offset的大小。举例，分段后的一个数据文件的offset是从20开始，那么offset为25的Message在index文件中的相对offset就是25-20 = 5。存储相对offset可以减小索引文件占用的空间。
* position，表示该条Message在数据文件中的绝对位置。只要打开文件并移动文件指针到这个position就可以读取对应的Message了。

index文件中并没有为数据文件中的每条Message建立索引，而是采用了稀疏存储的方式，每隔一定字节的数据建立一条索引。这样避免了索引文件占用过多的空间，从而可以将索引文件保留在内存中。但缺点是没有建立索引的Message也不能一次定位到其在数据文件的位置，从而需要做一次顺序扫描，但是这次顺序扫描的范围就很小了。

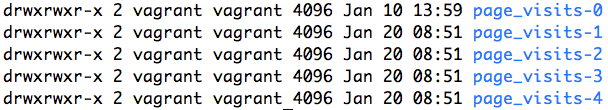
在Kafka中，索引文件的实现类为OffsetIndex，它的类图如下：  


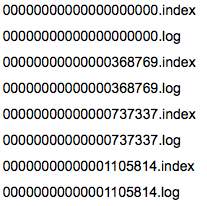
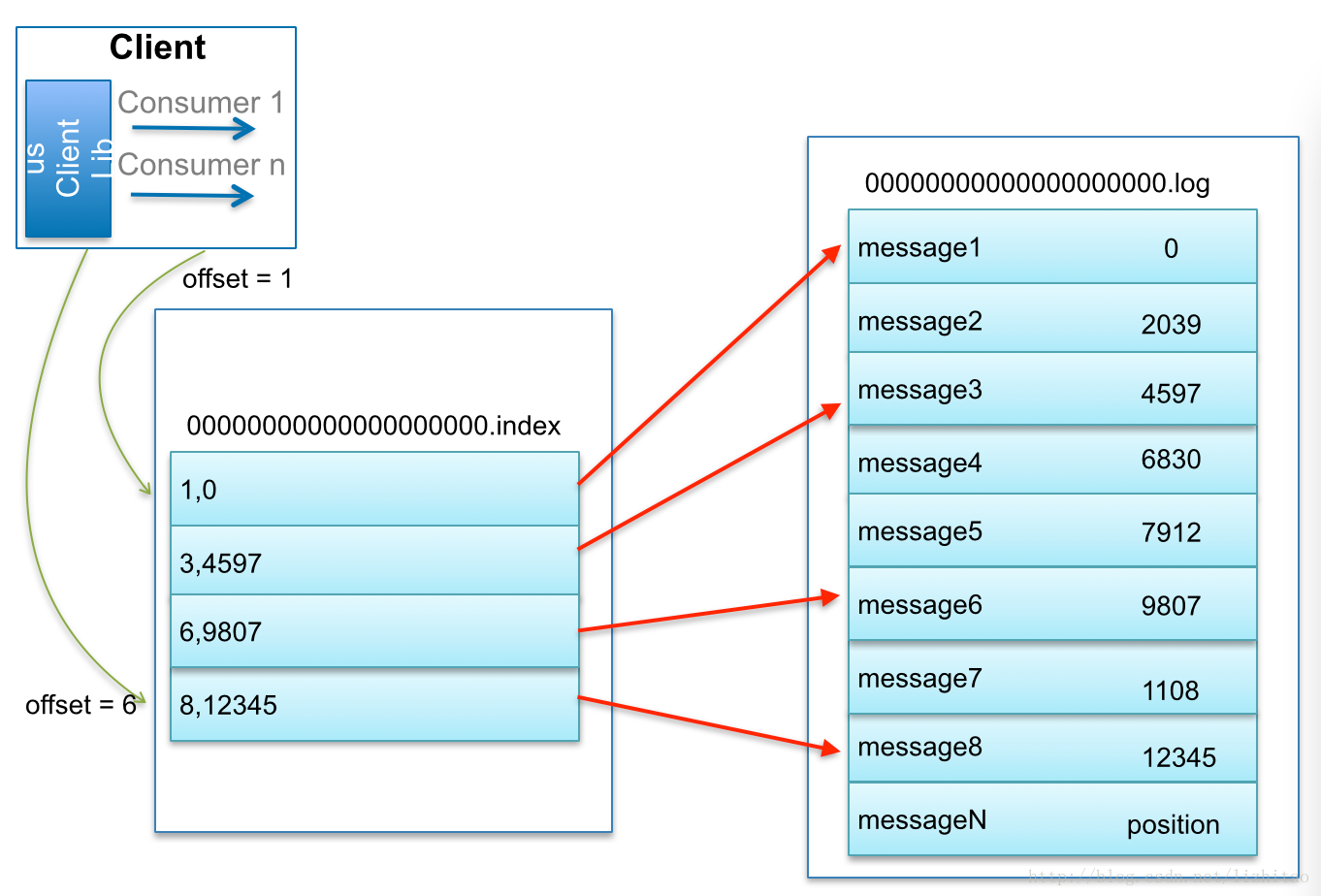
主要的方法有：

* append方法，添加一对offset和position到index文件中，这里的offset将会被转成相对的offset。
* lookup, 用二分查找的方式去查找小于或等于给定offset的最大的那个offset

## 小结

我们以几张图来总结一下Message是如何在Kafka中存储的，以及如何查找指定offset的Message的。

Message是按照topic来组织，每个topic可以分成多个的partition，比如：有5个partition的名为为page\_visits的topic的目录结构为：  


partition是分段的，每个段叫LogSegment，包括了一个数据文件和一个索引文件，下图是某个partition目录下的文件：  
  
可以看到，这个partition有4个LogSegment。展示是如何查找Message的。  
  
比如：要查找绝对offset为7的Message：

1. 首先是用二分查找确定它是在哪个LogSegment中，自然是在第一个Segment中。
2. 打开这个Segment的index文件，也是用二分查找找到offset小于或者等于指定offset的索引条目中最大的那个offset。自然offset为6的那个索引是我们要找的，通过索引文件我们知道offset为6的Message在数据文件中的位置为9807。
3. 打开数据文件，从位置为9807的那个地方开始顺序扫描直到找到offset为7的那条Message。

这套机制是建立在offset是有序的。索引文件被映射到内存中，所以查找的速度还是很快的。

一句话，Kafka的Message存储采用了分区(partition)，分段(LogSegment)和稀疏索引这几个手段来达到了高效性。

# [[置顶] Kafka学习总结(四)——broker、consumer、producer主要配置](http://blog.csdn.net/gongxinju/article/details/72615237)

2017-05-21 22:22 142人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/72615237" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/72615237" \l "report" \o "举报)

IMG_256 分类：

java基础（9） IMG_257 kafka消息队列（3） IMG_258

版权声明：本文为博主原创文章，未经博主允许不得转载。

**1、Broker配置**

#######################Server Basics##############################唯一标识在集群中的ID，要求是正数。

broker.id=0#服务端口，默认9092

port=9092#监听地址，不设为所有地址

host.name=172.168.63.91

# 处理网络请求的最大线程数

num.network.threads=2# 处理磁盘I/O的线程数

num.io.threads=8# 一些后台线程数

background.threads = 4# 等待IO线程处理的请求队列最大数

queued.max.requests = 500

# socket的发送缓冲区（SO\_SNDBUF）

socket.send.buffer.bytes=1048576# socket的接收缓冲区 (SO\_RCVBUF)

socket.receive.buffer.bytes=1048576# socket请求的最大字节数。为了防止内存溢出，message.max.bytes必然要小于

socket.request.max.bytes = 104857600

############################# Topic ############################## 每个topic的分区个数，更多的partition会产生更多的segment file

num.partitions=2# 是否允许自动创建topic ，若是false，就需要通过命令创建topic

auto.create.topics.enable =true# 一个topic ，默认分区的replication个数 ，不能大于集群中broker的个数。

default.replication.factor =1# 消息体的最大大小，单位是字节

message.max.bytes = 1000000

############################# ZooKeeper ############################## Zookeeper quorum设置。如果有多个使用逗号分割

zookeeper.connect=debugo01:2181,debugo02,debugo03# 连接zk的超时时间

zookeeper.connection.timeout.ms=1000000# ZooKeeper集群中leader和follower之间的同步实际

zookeeper.sync.time.ms = 2000

############################# Log ##############################日志存放目录，多个目录使用逗号分割

log.dirs=/var/log/kafka

# 当达到下面的消息数量时，会将数据flush到日志文件中。默认10000#log.flush.interval.messages=10000# 当达到下面的时间(ms)时，执行一次强制的flush操作。interval.ms和interval.messages无论哪个达到，都会flush。默认3000ms#log.flush.interval.ms=1000# 检查是否需要将日志flush的时间间隔

log.flush.scheduler.interval.ms = 3000

# 日志清理策略（delete|compact）

log.cleanup.policy = delete# 日志保存时间 (hours|minutes)，默认为7天（168小时）。超过这个时间会根据policy处理数据。bytes和minutes无论哪个先达到都会触发。

log.retention.hours=168# 日志数据存储的最大字节数。超过这个时间会根据policy处理数据。#log.retention.bytes=1073741824

# 控制日志segment文件的大小，超出该大小则追加到一个新的日志segment文件中（-1表示没有限制）

log.segment.bytes=536870912# 当达到下面时间，会强制新建一个segment

log.roll.hours = 24\*7# 日志片段文件的检查周期，查看它们是否达到了删除策略的设置（log.retention.hours或log.retention.bytes）

log.retention.check.interval.ms=60000

# 是否开启压缩

log.cleaner.enable=false# 对于压缩的日志保留的最长时间

log.cleaner.delete.retention.ms = 1 day

# 对于segment日志的索引文件大小限制

log.index.size.max.bytes = 10 \* 1024 \* 1024#y索引计算的一个缓冲区，一般不需要设置。

log.index.interval.bytes = 4096

############################# replica ############################## partition management controller 与replicas之间通讯的超时时间

controller.socket.timeout.ms = 30000# controller-to-broker-channels消息队列的尺寸大小

controller.message.queue.size=10# replicas响应leader的最长等待时间，若是超过这个时间，就将replicas排除在管理之外

replica.lag.time.max.ms = 10000# 是否允许控制器关闭broker ,若是设置为true,会关闭所有在这个broker上的leader，并转移到其他broker

controlled.shutdown.enable = false# 控制器关闭的尝试次数

controlled.shutdown.max.retries = 3# 每次关闭尝试的时间间隔

controlled.shutdown.retry.backoff.ms = 5000

# 如果relicas落后太多,将会认为此partition relicas已经失效。而一般情况下,因为网络延迟等原因,总会导致replicas中消息同步滞后。如果消息严重滞后,leader将认为此relicas网络延迟较大或者消息吞吐能力有限。在broker数量较少,或者网络不足的环境中,建议提高此值.

replica.lag.max.messages = 4000#leader与relicas的socket超时时间

replica.socket.timeout.ms= 30 \* 1000# leader复制的socket缓存大小

replica.socket.receive.buffer.bytes=64 \* 1024# replicas每次获取数据的最大字节数

replica.fetch.max.bytes = 1024 \* 1024# replicas同leader之间通信的最大等待时间，失败了会重试

replica.fetch.wait.max.ms = 500# 每一个fetch操作的最小数据尺寸,如果leader中尚未同步的数据不足此值,将会等待直到数据达到这个大小

replica.fetch.min.bytes =1# leader中进行复制的线程数，增大这个数值会增加relipca的IO

num.replica.fetchers = 1# 每个replica将最高水位进行flush的时间间隔

replica.high.watermark.checkpoint.interval.ms = 5000

# 是否自动平衡broker之间的分配策略

auto.leader.rebalance.enable = false# leader的不平衡比例，若是超过这个数值，会对分区进行重新的平衡

leader.imbalance.per.broker.percentage = 10# 检查leader是否不平衡的时间间隔

leader.imbalance.check.interval.seconds = 300# 客户端保留offset信息的最大空间大小

offset.metadata.max.bytes = 1024

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116

**2、Consumer主要配置**

##########################Consumer ############################## Consumer端核心的配置是group.id、zookeeper.connect# 决定该Consumer归属的唯一组ID，By setting the same group id multiple processes indicate that they are all part of the same consumer group.

group.id=demo\_group

# 对于zookeeper集群的指定，必须和broker使用同样的zk配置

zookeeper.connect=172.168.63.91:2181,172.168.63.92:2182,172.168.63.93:2182# zookeeper的心跳超时时间，查过这个时间就认为是无效的消费者

zookeeper.session.timeout.ms = 6000# zookeeper的等待连接时间

zookeeper.connection.timeout.ms = 6000# zookeeper的follower同leader的同步时间

zookeeper.sync.time.ms = 2000# 当zookeeper中没有初始的offset时，或者超出offset上限时的处理方式 。# smallest ：重置为最小值 # largest:重置为最大值 # anything else：抛出异常给consumer

auto.offset.reset = largest

# socket的超时时间，实际的超时时间为max.fetch.wait + socket.timeout.ms.

socket.timeout.ms= 30 \* 1000# socket的接收缓存空间大小

socket.receive.buffer.bytes=64 \* 1024#从每个分区fetch的消息大小限制

fetch.message.max.bytes = 1024 \* 1024

# true时，Consumer会在消费消息后将offset同步到zookeeper，这样当Consumer失败后，新的consumer就能从zookeeper获取最新的offset

auto.commit.enable = true# 自动提交的时间间隔

auto.commit.interval.ms = 60 \* 1000

# 用于消费的最大数量的消息块缓冲大小，每个块可以等同于fetch.message.max.bytes中数值

queued.max.message.chunks = 10

# 当有新的consumer加入到group时,将尝试reblance,将partitions的消费端迁移到新的consumer中, 该设置是尝试的次数

rebalance.max.retries = 4# 每次reblance的时间间隔

rebalance.backoff.ms = 2000# 这个参数避免在没有新数据的情况下重复频繁的拉数据。 如果拉到空数据，则多推后这个时间

backoff.increment.ms = 1000

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41

**3、Producer主要配置**

#############################Producer############################## 消费者获取消息元信息(topics, partitions and replicas)的地址,配置格式是：host1:port1,host2:port2，也可以在外面设置一个vip

metadata.broker.list=172.168.63.91:9092,172.168.63.92:9092,172.168.63.93:9092

#消息的确认模式# 0：不保证消息的到达确认，只管发送，低延迟但是会出现消息的丢失，在某个server失败的情况下，有点像TCP# 1：发送消息，并会等待leader 收到确认后，一定的可靠性# -1：发送消息，等待leader收到确认，并进行复制操作后，才返回，最高的可靠性

request.required.acks = 0

# 消息发送的最长等待时间

request.timeout.ms = 10000# socket的缓存大小

send.buffer.bytes=100\*1024# key的序列化方式，若是没有设置，同serializer.class

key.serializer.class# 分区的策略，默认是取模

partitioner.class=kafka.producer.DefaultPartitioner# 消息发送失败后的重试次数

message.send.max.retries = 3# 每次失败后的间隔时间

retry.backoff.ms = 100# 生产者定时更新topic元信息的时间间隔 ，若是设置为0，那么会在每个消息发送后都去更新数据

topic.metadata.refresh.interval.ms = 600 \* 1000

# 异步模式下缓冲数据的最大时间。例如设置为100则会集合100ms内的消息后发送，这样会提高吞吐量，但是会增加消息发送的延时

queue.buffering.max.ms = 5000# 异步模式下缓冲的最大消息数，同上

queue.buffering.max.messages = 10000# 异步模式下，消息进入队列的等待时间。若是设置为0，则消息不等待，如果进入不了队列，则直接被抛弃

queue.enqueue.timeout.ms = -1# 异步模式下，每次发送的消息数，当queue.buffering.max.messages或queue.buffering.max.ms满足条件之一时producer会触发发送。

batch.num.messages=200

# [[置顶] Kafka学习总结(五)——Kafka集群搭建](http://blog.csdn.net/gongxinju/article/details/53415051)

标签： [kafka](http://www.csdn.net/tag/kafka" \t "http://blog.csdn.net/gongxinju/article/details/_blank)[集群](http://www.csdn.net/tag/%e9%9b%86%e7%be%a4" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

2016-11-30 22:59 3283人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/53415051" \l "comments)(3) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/53415051" \l "report" \o "举报)

IMG_256 分类：

kafka消息队列（3） IMG_257

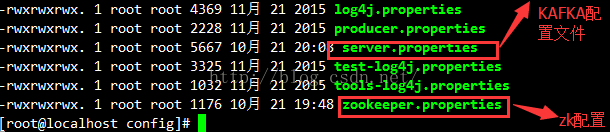
版权声明：本文为博主原创文章，未经博主允许不得转载。

**1、准备好kafka安装包，unzip xxx.zip 解压。**

IMG_258IMG_259

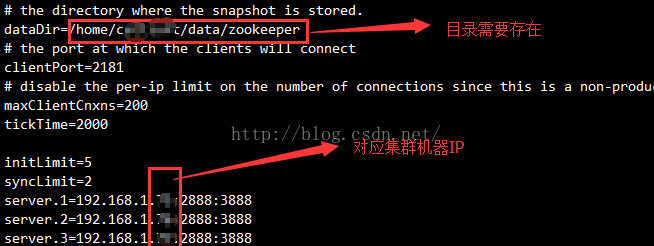
压缩包网盘地址：[http://pan.baidu.com/s/1o8pmH2U](http://pan.baidu.com/s/1o8pmH2U" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

**2、进入到kafka的config目录**

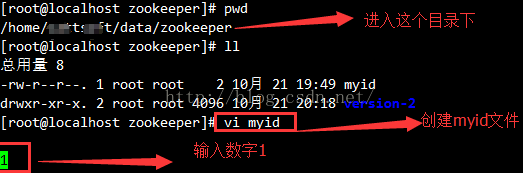
IMG_260

**3、先搭建三个节点zk集群，三台机器。**

修改zookeeper.properties如下配置信息。

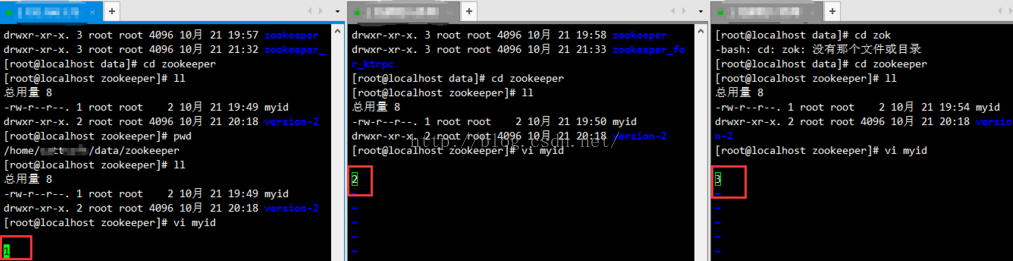
IMG_262

cd /home/xxx/data/zookeeper 目录

IMG_264

每台机器都需要创建myid文件，内容依次为1，2，3

如下是3台集群服务器样例：

IMG_266

**4、三台机器都修改好zookeeper.properties文件和创建好myid文件**

进入kafka目录

IMG_268IMG_269

执行启动zookeeper命令：nohup ./bin/zookeeper-server-start.sh config/zookeeper.properties >logs/zookeeper.log 2>1 &

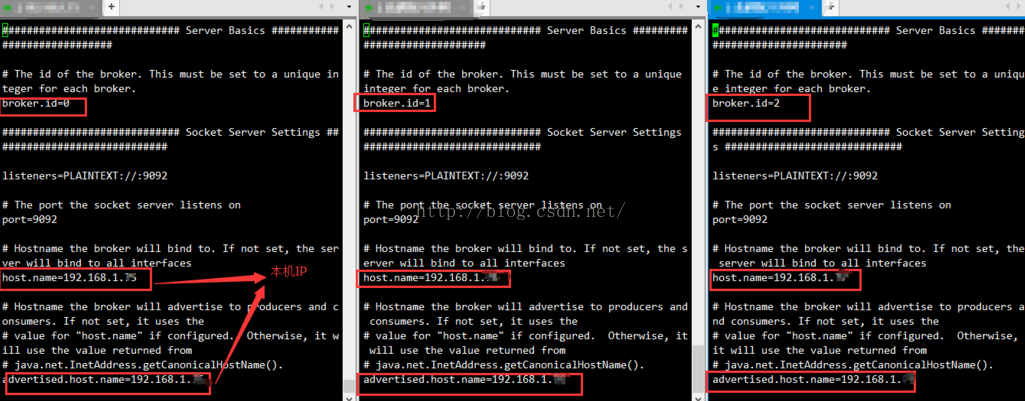
IMG_270IMG_271

三台机器都执行启动命令，查看logs/zookeeper.log 文件，没有报错就说明zookeeper集群启动成功了。

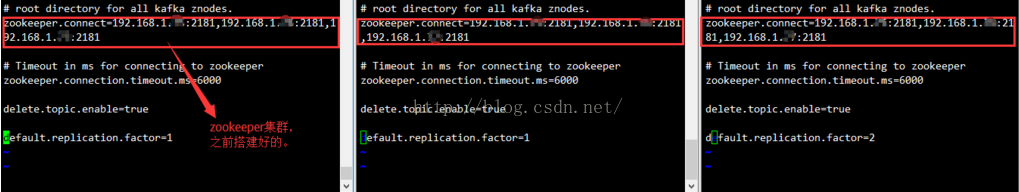
**5、着手搭建kafka集群**

修改server.properties配置文件，三个配置相应修改。

host.name 为各机器的IP地址

IMG_272

配置文件最下方：

IMG_274

**5、启动kafka三个节点**

进入kafka目录，执行如下命令

nohup ./bin/kafka-server-start.sh config/server.properties >logs/kafka.log 2>1 &

**6、测试kafka集群**

1-进入kafka根目录，创建topic--test

bin/kafka-topics.sh --create --zookeeper 192.168.1.xx:2181,192.168.1.xx:2181,192.168.1.xx:2181 --replication-factor 1 --partitions 1 --topic test

IMG_276IMG_277

2-列出已创建的topic列表

bin/kafka-topics.sh --list --zookeeper localhost:2181

3-模拟客户端去发送消息

bin/kafka-console-producer.sh --broker-list 192.168.1.xx:9092,192.168.1.xx:9092,192.168.1.xx:9092 --topic test

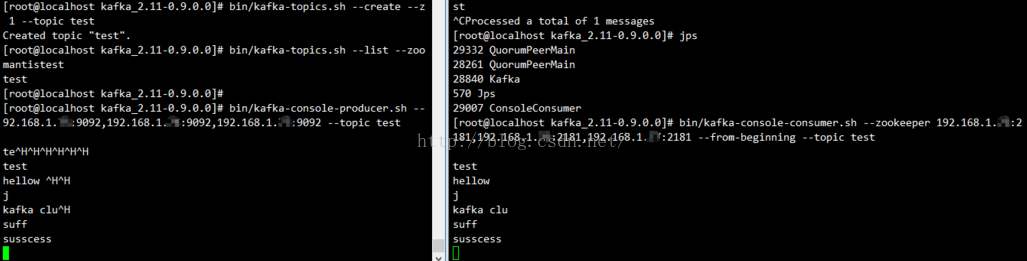
IMG_278IMG_279

4-模拟客户端去接受消息

bin/kafka-console-consumer.sh --zookeeper 192.168.1.xx:2181,192.168.1.xx:2181,192.168.1.xx:2181 --from-beginning --topic test

IMG_280IMG_281

**7、自己测试结果**

IMG_282

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic test

# [[置顶] Kafka学习总结(六)——应用开发](http://blog.csdn.net/gongxinju/article/details/72616636)

2017-05-21 23:11 108人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/72616636" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/72616636" \l "report" \o "举报)

IMG_256 分类：

kafka（2） IMG_257

版权声明：本文为博主原创文章，未经博主允许不得转载。

1. 开发一个生产者应用，如下：

|  |
| --- |
| package com.cattsoft;  import java.util.Properties;  import kafka.javaapi.producer.Producer;  import kafka.producer.KeyedMessage;  import kafka.producer.ProducerConfig;  public class KafkaProducer {  static String message = "kafka message producer!";  public static void main(String[] args) {  System.out.println("生产消息开始!!!");  Properties props = new Properties(); /\* 发送实际数据的socket连接将基于返回的metadata数据信息而建立 \*/  props.setProperty("metadata.broker.list","172.168.10.48:9092,172.168.10.49:9092,172.168.10.54:9092"); /\* 消息的序列化类别。默认编码器输入一个字节byte[]，然后返回相同的字节byte[] \*/  props.setProperty("serializer.class", "kafka.serializer.StringEncoder"); /\* \* 此选项置顶了消息是否在后台线程中异步发送。正确的值： （1） async： 异步发送 （2） sync： 同步发送 \* 通过将producer设置为异步，我们可以批量处理请求（有利于提高吞吐率）但是这也就造成了客户端机器丢掉未发送数据的可能性 \*/  props.setProperty("producer.type", "asyc"); /\* \* 仅仅for sync 0： \* 表示producer从来不等待来自broker的确认信息（和0.7一样的行为）。这个选择提供了最小的时延但同时风险最大（ \* 因为当server宕机时，数据将会丢失）。 1：表示获得leader \* replica已经接收了数据的确认信息。这个选择时延较小同时确保了server确认接收成功。 \* -1：producer会获得所有同步replicas都收到数据的确认,同时时延最大。 \*/  props.put("request.required.acks", "1"); /\* 确认超时时间 \*/  props.put("request.timeout.ms", 1000); /\* broker尽力实现request.required.acks需求时的等待时间，否则会发送错误到客户端 \*/  props.put("request.timeout.ms", 10000); /\* 此项参数可以设置压缩数据的codec，可选codec为：“none”， “gzip”， “snappy” \*/  props.put("compression.codec", "none"); /\* 在设置了压缩的情况下，可以指定特定的topic压缩，为指定则全部压缩 \*/  props.put("compressed.topics", null); /\* 消息发送最大尝试次数 \*/  props.put("message.send.max.retries", 3); /\* 批量消息的数量，仅仅for asyc \*/  props.put("batch.num.messages", 100); /\* \* 当应用async模式时，用户缓存数据的最大时间间隔。例如，设置为100时，将会批量处理100ms之内消息。这将改善吞吐率， \* 但是会增加由于缓存产生的延迟。 \*/  props.put("queue.buffering.max.ms", 5000); /\* producer 缓存的消息的最大数量，仅仅for asyc \*/  props.put("queue.buffering.max.message", 1000); /\* 必须实现kafka.producer.Partitioner，根据Key提供一个分区策略 \*/  props.put("partitioner.class", "kafka.producer.DefaultPartitioner");  ProducerConfig config = new ProducerConfig(props);  Producer<String, String> producer = new Producer<String, String>(config); String topic = "ADAPTER\_QUEUE\_SOLR\_S1000";  try {  int i = 0;  int count = 100;  while (i < count) {  KeyedMessage<String, String> data = new KeyedMessage<String, String>(topic, message);  producer.send(data);  i++;  }  System.out.println("生成消息完成！");  } catch (Exception e) {  e.printStackTrace();  }  producer.close();  }  } |

2、开发一个消费者应用

package com.cattsoft;

import java.util.HashMap;import java.util.List;import java.util.Map;import java.util.Properties;

import kafka.consumer.ConsumerConfig;import kafka.consumer.ConsumerIterator;import kafka.consumer.KafkaStream;import kafka.javaapi.consumer.ConsumerConnector;

public class KafkaConsumer extends Thread {

private final ConsumerConnector consumer;

private final String topic;

public KafkaConsumer(String topic) {

consumer =(ConsumerConnector) kafka.consumer.Consumer

.createJavaConsumerConnector(createConsumerConfig());

this.topic =topic;

}

private ConsumerConfig createConsumerConfig() {

Properties props = new Properties();

/\* 指定zookeeper的连接的字符串 \*/

props.put("zookeeper.connect", "130.60.23.193:2181,130.60.23.194:2181,130.60.23.195:2181");

props.put("group.id", "zk999");

// 如果true,consumer定期地往zookeeper写入每个分区的offset

props.put("auto.commit.enable", "true");

/\* consumer向zookeeper提交offset的频率，单位是秒 \*/

props.put("auto.commit.interval.ms", 60 \* 1000);

/\*

\* zookeeper

\* 会话的超时限制。如果consumer在这段时间内没有向zookeeper发送心跳信息，则它会被认为挂掉了，并且reblance将会产生

\*/

props.put("zookeeper.session.timeout.ms", "50000");

/\* 客户端在建立通zookeeper连接中的最大等待时间 \*/

props.put("zookeeper.connection.timeout.ms", "20000");

/\* rebalance时的最大尝试次数 \*/

/\*

\* 当新的consumer加入到consumer group时，

\* consumers集合试图重新平衡分配到每个consumer的partitions数目。

\* 如果consumers集合改变了，当分配正在执行时，这个重新平衡会失败并重入

\*/

props.put("rebalance.max.retries", "5");

/\* 在重试reblance之前backoff时间 \*/

props.put("rebalance.backoff.ms", "12000");

/\*

\* zookeeper中没有初始化的offset时，如果offset是以下值的回应：

\* smallest：自动复位offset为smallest的offset largest：自动复位offset为largest的offset

\* anything else：向consumer抛出异常

\*/

props.put("auto.offset.reset", "largest");

/\* ZK follower可以落后ZK leader的最大时间 \*/

props.put("zookeeper.sync.time.ms", "1200");

/\* 这个参数避免在没有新数据的情况下重复频繁的拉数据。 如果拉到空数据，则多推后这个时间 \*/

props.put("backoff.increment.ms", 1000);

return new ConsumerConfig(props);

}

public void run() {

Map<String, Integer> topickMap = new HashMap<String, Integer>();

topickMap.put(topic, 1);

Map<String, List<KafkaStream<byte[], byte[]>>> streamMap = consumer.createMessageStreams(topickMap);

KafkaStream<byte[], byte[]> stream = streamMap.get(topic).get(0);

ConsumerIterator<byte[], byte[]> it = stream.iterator();

System.out.println("\*\*\*\*\*\*\*\*\*Results\*\*\*\*\*\*\*\*");

int i = 1;

while (it.hasNext()) {

System.out.println(

Thread.currentThread() + " 接收到的第几个消息 " + i++ + " ------ " + new String(it.next().message()));

try {

Thread.sleep(800);

this.consumer.commitOffsets();

} catch (Exception e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) {

KafkaComsuerUtil consumerThread = new KafkaComsuerUtil("ADAPTER\_QUEUE\_SOLR\_S1000");

consumerThread.start();

}

}

# [[置顶] kafka学习总结(七)——kafka监控-kafkaOffsetMonitor](http://blog.csdn.net/gongxinju/article/details/73277395)

标签： [界面](http://www.csdn.net/tag/%e7%95%8c%e9%9d%a2" \t "http://blog.csdn.net/gongxinju/article/details/_blank)[kafka-java](http://www.csdn.net/tag/kafka-java" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

2017-06-15 11:17 98人阅读 [评论](http://blog.csdn.net/gongxinju/article/details/73277395" \l "comments)(0) [收藏](http://blog.csdn.net/gongxinju/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/gongxinju/article/details/_blank) [举报](http://blog.csdn.net/gongxinju/article/details/73277395" \l "report" \o "举报)

IMG_256 分类：

kafka（2） IMG_257

版权声明：本文为博主原创文章，未经博主允许不得转载。

**1、kafkaOffsetmonitor简述**

　KafkaOffsetMonitor是有由Kafka开源社区提供的一款Web管理界面，这个应用程序用来实时监控Kafka服务的Consumer以及它们所在的Partition中的Offset，你可以通过浏览当前的消费者组，并且每个Topic的所有Partition的消费情况都可以观看的一清二楚。它让我们很直观的知道，每个Partition的Message是否消费掉，有木有阻塞等等。   
　   
　　这个Web管理平台保留的Partition、Offset和它的Consumer的相关历史数据，我们可以通过浏览Web管理的相关模块，清楚的知道最近一段时间的消费情况。

　　该Web管理平台有以下功能：

* 对Consumer的消费监控，并列出每个Consumer的Offset数据
* 保护消费者组列表信息
* 每个Topic的所有Partition列表包含：Topic、Pid、Offset、LogSize、Lag以及Owner等等
* 浏览查阅Topic的历史消费信息

这些功能对于我们开发来说，已经绰绰有余了。

**2、KafkaOffsetMonitor安装部署**

2.1、下载安装包   
百度云下载：[https://pan.baidu.com/s/1hrHfLrA](https://pan.baidu.com/s/1hrHfLrA" \t "http://blog.csdn.net/gongxinju/article/details/_blank)   
github官网下载：[https://github.com/quantifind/KafkaOffsetMonitor/releases/download/v0.2.0/KafkaOffsetMonitor-assembly-0.2.0.jar](https://github.com/quantifind/KafkaOffsetMonitor/releases/download/v0.2.0/KafkaOffsetMonitor-assembly-0.2.0.jar" \t "http://blog.csdn.net/gongxinju/article/details/_blank)

2.2、部署   
KafkaOffsetMonitor的安装部署较为简单，所有的资源都打包到一个JAR文件中了，因此，直接运行即可，省去了我们去配置。这里我们可以新建一个目录单独用于Kafka的监控目录，我这里新建一个kafka\_monitor文件目录，然后我们在准备启动脚本，脚本内容如下所示：

#! /bin/bash

java -cp KafkaOffsetMonitor-assembly-0.2.0.jar \

com.quantifind.kafka.offsetapp.OffsetGetterWeb \

--zk 192.168.174.137:2181 \

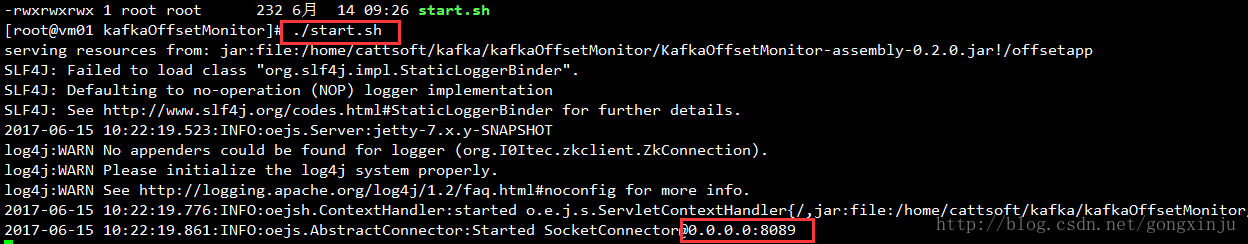
--port 8089 \

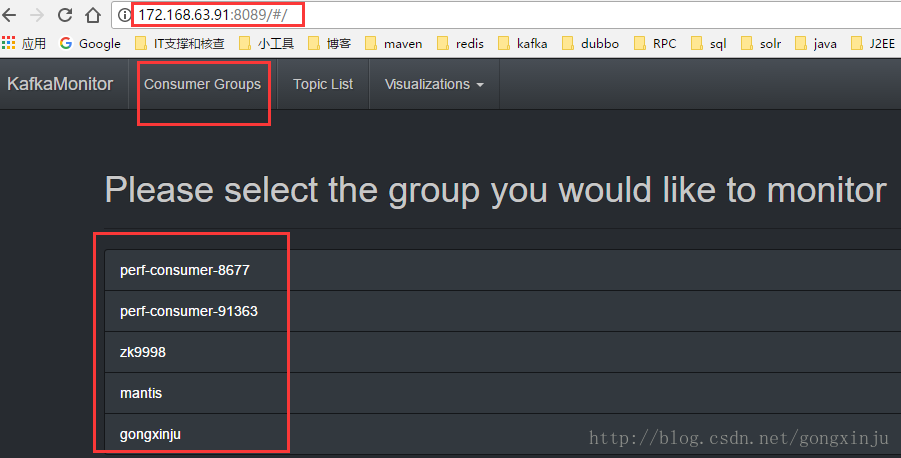
--refresh 10.seconds \

--retain 1.days

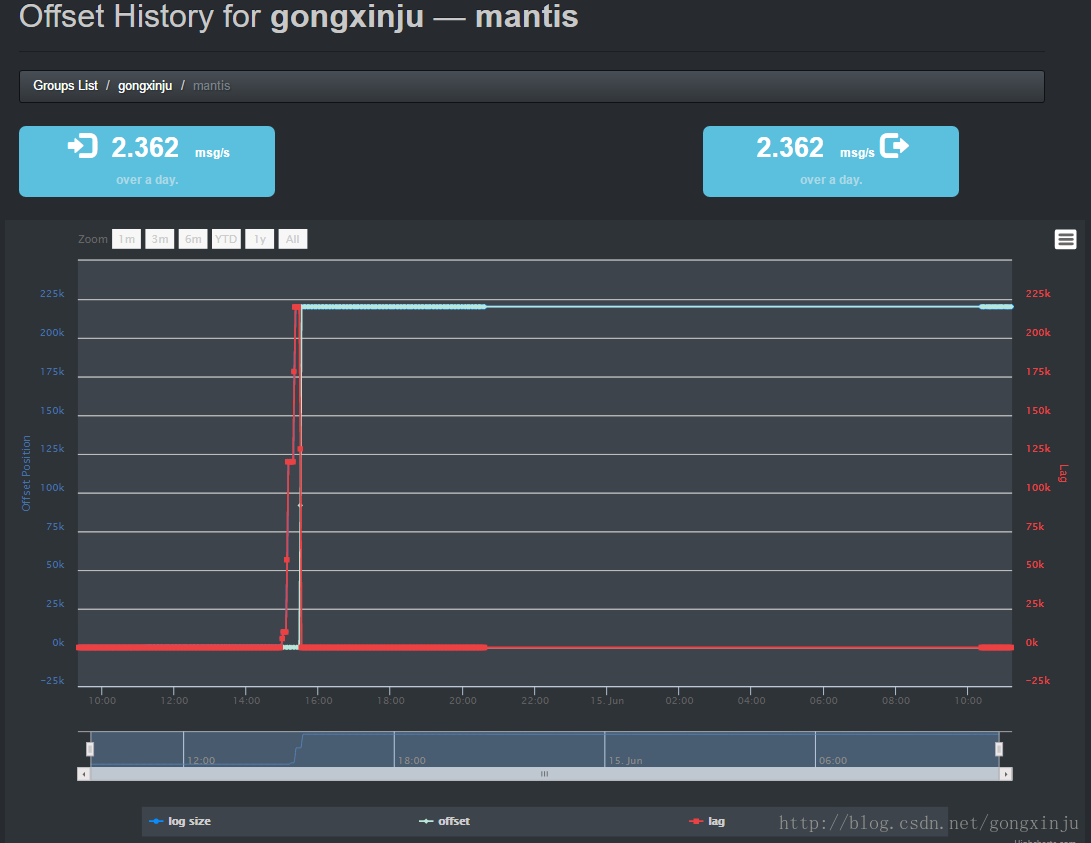
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

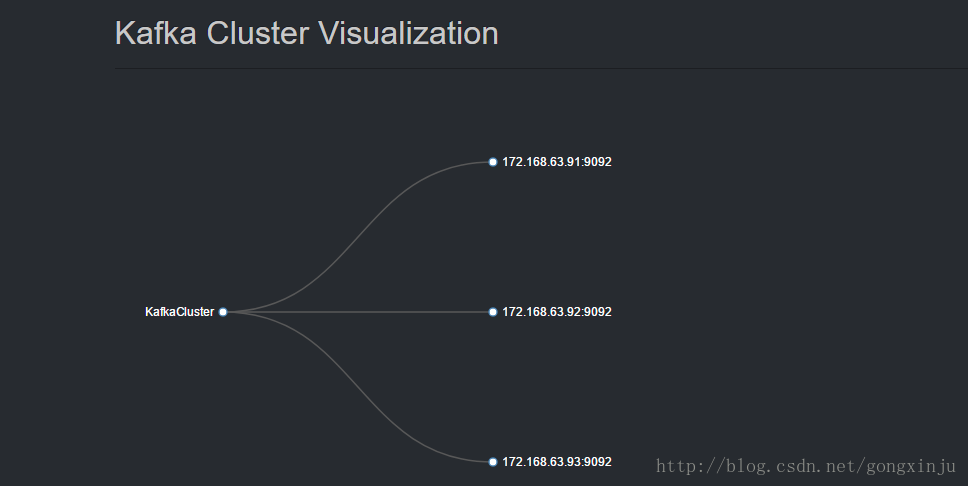


2.3、启动脚本   


3、KafkaOffsetMonitor运行预览   


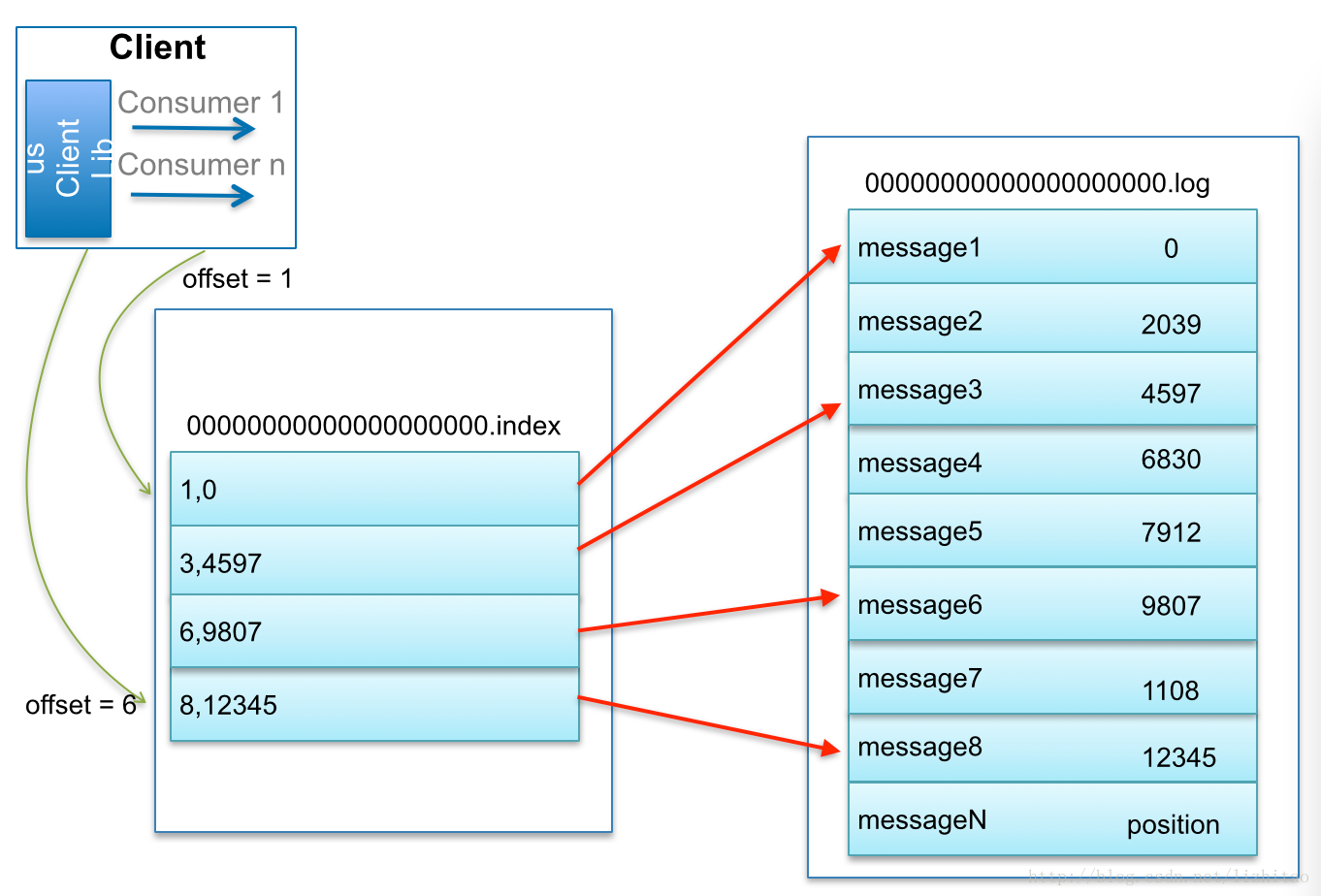






**4、总结**   
图中的一些参数的含义如下：

* Topic：创建Topic名称
* Partition：分区编号
* Offset：表示该Parition已经消费了多少Message
* LogSize：表示该Partition生产了多少Message
* Lag：表示有多少条Message未被消费
* Owner：表示消费者
* Created：表示该Partition创建时间

  
比如：要查找绝对offset为7的Message：

1. 首先是用二分查找确定它是在哪个LogSegment中，自然是在第一个Segment中。
2. 打开这个Segment的index文件，也是用二分查找找到offset小于或者等于指定offset的索引条目中最大的那个offset。自然offset为6的那个索引是我们要找的，通过索引文件我们知道offset为6的Message在数据文件中的位置为9807。
3. 打开数据文件，从位置为9807的那个地方开始顺序扫描直到找到offset为7的那条Message。

这套机制是建立在offset是有序的。索引文件被映射到内存中，所以查找的速度还是很快的。

一句话，Kafka的Message存储采用了分区(partition)，分段(LogSegment)和稀疏索引这几个手段来达到了高效性。