jewes的专栏

个人资料

jewes

访问： 254112次
积分： 2983
等级： BLOG 5
排名： 第8600名

原创： 71篇 转载： 1篇
译文： 0篇 评论： 66条

Kafka的Log存储解析

标签： kafka

2015-01-21 17:11 11215人阅读 评论(6) 收藏 举报

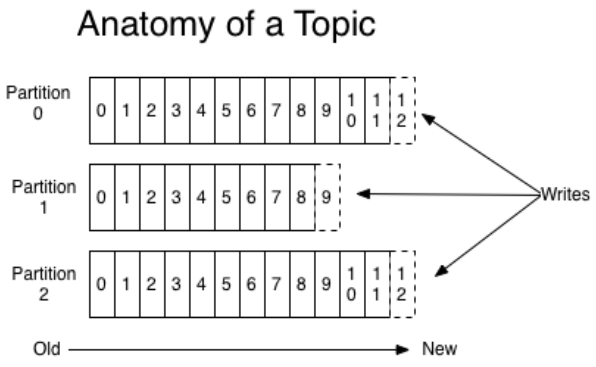
分类： Kafka（3）

Kafka的Log存储解析

标签（空格分隔）： kafka

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。

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：

Kafka的Log存储解析 (11186)
Kerberos认证流程详解 (10555)
扩大无线的覆盖范围 - 谈 (9733)
Kafka的通讯协议 (8667)
NPIV - 连接虚拟机与存储 (6693)
静态和动态链接 (6680)
【已更新】为什么我用支 (6398)
最常用的也是最容易忘记 (5455)

评论排行

初来乍到：设置CSDN头 (10)
Kerberos认证流程详解 (10)
NPIV - 连接虚拟机与存储 (7)
Kafka的Log存储解析 (6)
静态和动态链接 (5)
Spark RDD API详解(一) (5)
Get Started With Continu (3)
MAC：解决Mac雪豹开机 (2)
亲历2013中国产品经理大 (2)
最常用的也是最容易忘记 (2)

推荐文章

* 2016 年最受欢迎的编程语言是什么？
* Chromium扩展（Extension）的页面（Page）加载过程分析
* Android Studio 2.2 来啦
* 手把手教你做音乐播放器（二）技术原理与框架设计
* JVM 性能调优实战之：使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

最新评论

Kafka的Log存储解析
mulangren1988: 不错，解惑了

NPIV - 连接虚拟机与存储的桥梁
FOX0406: 说的清楚，学习了。

静态和动态链接
记忆力不好: 动态链接为什么是装入时，而不是运行时或者调用时

Kafka SocketServer源代码分析
jewes: @qwe564217192:Kafka是开源的，可以从其官网上下载到源码。

Kafka SocketServer源代码分析
风吹裤衩轻飞扬丶: 你有这个源码嘛？没有接触过这方面的知识现在要上项目了怎么快速学习？

Spark RDD API详解(一) Map和F
blacklee123: http://homepage.cs.latrobe.edu.au/

Spark RDD API详解(一) Map和F
xmh8023: 解释的好，给你点个赞

Kerberos认证流程详解
jewes: @heart2header:谢谢！

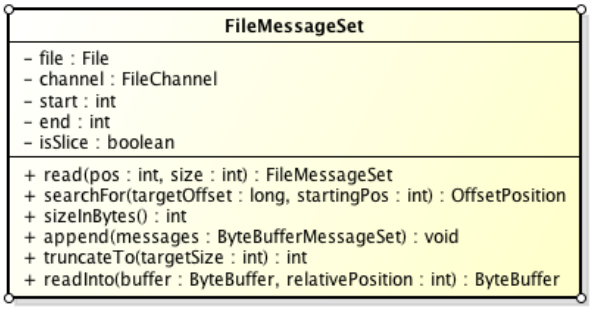
Kerberos认证流程详解
heart2header: 很好的帖子，我也是自己总结下Kerberos，但是显然你做的更好。我就直接收藏了。

Spark RDD API详解(一) Map和F
guotong1988: 很有用

- 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。

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。

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。

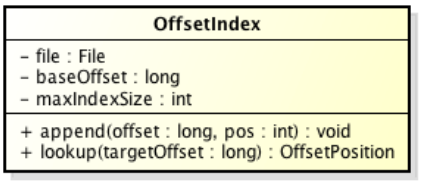
- 相对offset：因为数据文件分段以后，每个数据文件的起始offset不为0，相对offset表示这条Message相对于其所属数据文件中最小的offset的大小。举例，分段后的一个数据文件的offset是从20开始，那么offset为25的

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：

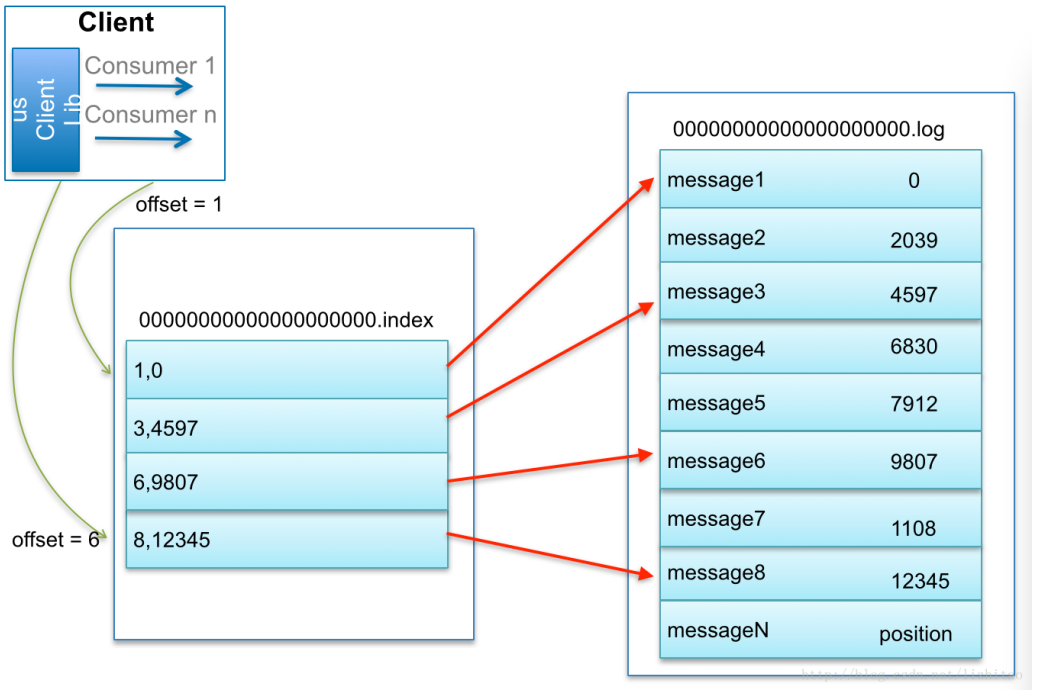
```
drwxrwxr-x 2 vagrant vagrant 4096 Jan 10 13:59 page_visits-0
drwxrwxr-x 2 vagrant vagrant 4096 Jan 20 08:51 page_visits-1
drwxrwxr-x 2 vagrant vagrant 4096 Jan 20 08:51 page_visits-2
drwxrwxr-x 2 vagrant vagrant 4096 Jan 20 08:51 page_visits-3
drwxrwxr-x 2 vagrant vagrant_4096 Jan 20 08:51 page_visits-4
```

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：

```
00000000000000000000.index
00000000000000000000.log
00000000000000368769.index
00000000000000368769.log
00000000000000737337.index
00000000000000737337.log
00000000000001105814.index
00000000000001105814.log
```

可以看到，这个partition有4个LogSegment。

借用博主@lizhitao博客上的一张图来展示是如何查找Message的。



比如，要查找绝对offset为7的Message：

二分查找确定它是在哪个LogSegment中，自然是在第一个Segment中。

egment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
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

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。

上一篇 Kafka Producer相关代码分析
下一篇 谈谈对CAP定理的理解

我的同类文章

Kafka（3）

- Kafka Producer相关代码分析 2015-01-17 阅读 4060
- Kafka SocketServer源代码... 2015-01-04 阅读 3419
- Kafka的通讯协议 2015-01-15 阅读 8667

猜你在找

查看评论

6楼 mulangren1988 2016-08-23 11:38发表


不错，解惑了

5楼 柳年思水 2015-11-03 14:49发表


支持，楼主总结的很不错，非常感谢

4楼 周小虎_ 2015-09-30 10:41发表


真心好文，讲解的条理非常清晰，已经转载并注明出处，谢谢

3楼 chenking666 2015-09-09 11:22发表


楼主讲的非常给力，学习了

2楼 longlongkong 2015-05-13 11:36发表


楼主讲的非常给力，谢谢！

1楼 fawen18 2015-04-19 08:41发表


楼主讲的非常给力，谢谢分享

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap