NeeMo——仗剑走天涯

源码读万遍，编码如有神。

个人资料

受到了风

访问： 11983次
积分： 295
等级： BLOG 2
排名： 千里之外

原创： 16篇
转载： 0篇
译文： 0篇
评论： 2条

levelDB学习笔记 —— table

标签： levelDB table

2013-08-22 21:17 635人阅读 评论(0) 收藏 举报

分类：

数据库（9）

levelDB并不跟beansDB，nessDB一样将key和value分开，而是将key和value一起存放。存放的文件就是.sst文件，而一个文件的结构就是一个table。

table筛选器的阅读顺序应该是

1.block_builder.cc

结合block.cc文件最开始的注释，我们可以得到一个block的结构：许多个entry，最后是一个数组来记录一些文件偏移信息。

这个文件中只有一个Add函数，仔细阅读就会发现每当有block_restart_interval个entry被插入时，就会产生一个restart，因此可以断定这个restart就像nessDB2.0的block的BLOCK_GAP。只不过每个gap记录一个key的具体值，而restart只记录文件偏移我们可以看到，为了节省文件的存储空间，levelDB对key是进行了精心的压缩的，那么为什么还要使用restart来进行记录呢？原因是在Block中的key-value对，是按照key的大小排列的，这样为了加快查找速度，我们可以使用二分查找来快速定位，但是由于key的压缩，使得在读取一个key值时，只能读取到该key的non_shared部分，而shared部分则是存放在上一个key中，这样我们不得不向前递归查找所有的key来得到一个二分查找中的分支key，尽管有人会绞尽脑计来想办法加速这个查找，但是我们为什么不用一个更简单的方法？将Block分成两层，外层进行二分查找，内层进行顺序查找，在外层的开始处，放入完整的key，使得内层的顺序查找变得可能。

2.format.cc

先看BlockHandle类，它里面封装了两个uint64_t类型，根据变量名就可以看出它们分别代表偏移和大小，但是既然是叫BlockHandle，那么肯定跟Block相关，所以里面记录了一个Block在文件中的偏移和它的大小。

再看Footer结构，跟nessDB2.0中的footer名字是一样的，那么应该是放在文件末尾来记录一些信息，它里面的变量为metaindex_handle和index_handle，这些还不知道是啥。这里面还有一个函数，ReadBlock,函数的作用是根据个BlockHandle提供的偏移和大小信息读出一个Block。

3.table_builder.cc

我们发现TableBuilder类里只有一个Rep结构，而再没有其它变量了，所以它的内部变量可能是封装在Rep里了。转到它的定义，发现它里面定义了两个BlockBuilder，即data_block和index_block，index_block跟Footer里的index_handle应该是相同的，但是data_block又出现在哪了呢?

LevelDB 学习笔记 —— u (683)
GDBM学习笔记 (641)
levelDB学习笔记 —— tal (632)

评论排行
字符串加减法(整数，小数 (2)
levelDB学习笔记 —— tal (0)
LevelDB 学习笔记 —— u (0)
nessDB2.0 学习笔记 (0)
nessDB1.0学习笔记 (0)
BeansDB源码剖析——bi (0)
BeansDB源码剖析——re (0)
BeansDB源码剖析——ht (0)
BeansDB学习笔记 (0)
求数组中最长递增子序列 (0)

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
字符串加减法(整数，小数)
受到了风: @liangdaj:放心吧，到时候会缩减条件的。不过小数减法是确实有过的。
字符串加减法(整数，小数)
liangdaj: 如果机试2小时，共3道题，一道题要写400行代码，那就无语了

再看TableBuilder的功能函数，根据函数名，最开始该看的应该是Add函数，即加入一个Entry。上面我们说到一个Block是由许多的Entry组成的，那么这个Entry到底加入到哪里了？跳过if语句，发现entry被加入到Rep的data_block中了，再反过来看if语句if (r->pending_index_entry),不知道这个变量有啥用，先不管，看是如何处理的:这里面最重要的一句就是将一个last_key——这个我们知道，看下面，每当插入一个entry时，这个值就被更新为这个entry的key，连同一个BlockHandle，也就是偏移和大小信息存入到一个BlockBuilder——index_block中，这样我们就可以推断，index_block是一个比data_block高一级的BlockBuilder，它们的关系如同一个Block和一个restart_interval的关系一样，只是逻辑上的拆分，便于分割查找。最后看这个函数的最后一句话，if (estimated_block_size >= r->options.block_size)，意思是说如果插入entry后，这个data_block的size大于了要求的block_size，那么进行Flush。好，再跳到Flush函数来看，它调用了WriteBlock函数。

再看WriteBlock函数，这个函数挺简单的，就是将BlockBuilder中的内容写入文件，同时将其内容清空，并更新它对应的BlockHandle。然后反过来看WriteBlock函数的后面——将pending_index_entry设置为true，结合Add函数，它将pending_handle加入到index_block中，那么很显然这个pending_index_entry跟pending_handle是有密切联系的，仔细分析，如果一个Data Block被加入到了文件中，那么pending_handle就会被设置为这个Block对应的handle，同时pending_index_handle被设置为true，作为pending_handle被更新的标志。pending的意思就是“没有被写入到index_block中”。

好了，经过上面对三个函数Add，Flush，WriteBlock的分析，我们可以得到一个Entry被加入到table文件中的顺序是：首先加入到当前的data_block，如果它满了，那么将这个data_block Flush到文件中，同时生成这个BlockBuilder对应的BlockHandle记录这个block的偏移和大小信息。那么这些信息何时持久化呢？其中一种情况就是在Add下一个Entry时，检测是否有pending_handle存在，也就是判断pengding_index_entry(为什么叫entry？因为一个data_block就是一个index_block的entry)是否为true，如果是，那么将这个data_block的handle作为value，它的最后一个key的某种形式作为key存入。

但是如果data_block的flush仅仅是靠Add中的if来触发，那么如果存入的数据肯定是不完整的。所以这项任务就交给了最后一个功能函数——Finish。来看Finish函数，在看Finish函数之前，需要记住的是，很多的data_block已经在之前就存在于文件中了。首先将data_block中的数据（不满足estimated_block_size >= r->options.block_size）进行了Flush，然后将BlockHandle写入到index_block中，这个操作和Flush之间还有一个操作就是写入一个metaindex_block，但是这个BlockBuilder没啥用。如果将这个写入操作放入到Flush之前，或者index_block的add之后，会使代码更容易理解。然后写入一个index_block，这里面记录了前面data_block的offset和size，最后，metaindex_block和index_block填入Footer，写入到文件末尾。

table文件的结构和写入就介绍完了。最后还需要探讨一下在这样的文件结构中，如何寻找一个key-value对。如果是C语言写的话，或许查找就没有这么难以理解了，但是在C++里总是要有面向对象的思想。table中面向对象的思想是将对Block的遍历操作和对.sst文件的遍历操作也看成是一种对象，及Iterator遍历器。遍历器的概念在OB中也用到了，刚开始看的时候估计会一头雾水，但是理解了其中的核心思想，也没啥了。

iterator.cc
可以将Iterator和STL中的iterator进行一下比较。看Iterator类，Next对应++，Prev对应--，Key和Value的getter则相当于*操作符，SeekToFirst则是.begin，SeekToLast则是.end，Seek则是find。这样一对应，我们就可忽略Iterator的定义，而直接看它内部的具体操作了，就好像vector的iterator和set的iterator的查找定位方式不一样的道理，Block和table的查找定位也是不同的。

block.cc
这个文件的作用就是生成一个在Block中查找遍历的Iterator。至于为什么将table文件分成多个block，除了给出的那3点外，block使得装入内存的文件变少也是一个意料外的惊喜。
根据一个Block文件的结构，前面一堆的有序的Entry，后面是记录每组entry的起始位置。按照在block_builder.cc下的分析，这个Iterator的Seek应该分成内外两层查找，外层使用二分查找，内层使用顺序查找。block.cc中关键的函数是Seek，在二分查找获得一个lower_bound之后，使用SeekToRestartPoint将current_，key_，value_进行重新定位，注意这时候value_的size是空的，这样在while中的ParseNextEntry时就可以得到第一个key，这是需要注意的一个地方。另外，进行二分查找时，由于每个restart的第一个key的non_shared总是为0，所以non_shared其实就是这个key的完整size。

另一个重要的函数是ParseNextEntry，这是根据前一个key得到本key的值和value。不过由于只是比较，每次parse的时候都对value进行赋值，有点浪费。

two_level_iterator.cc

这是table.cc生成的Iterator。根据上面的分析，table_builder会生成一个含有多个data_block和一个记录该data_block的最大key值以及offset和size的index_block，我们已经知道index_block是比data_block高一级的，那么当在table中查找一个key时，我们需要先定位——在哪个data_block里？这时使用index_block的iterator进行upper_bo 到这个data_block后，又可以使用该data_block的iterator进行进一步的查找。这就是所谓的“二层查找 ，其实跟ShardedLRU的思路是一样的，都是为了避免进行大内存的装入，还可以使查找更快。two_le 两个iterator，其实就跟两层for循环一样，外层走得慢，内层走得快，每当外层发生改变时，内层也需要重新赋值，重新生成一个iterator，这就是InitDataBlock。

table.cc

生成一个two_level_iterator，其中最重要的函数是BlockReader。它作为two_level_iterator生成新的内层iterator时的回调block_function_，主要是根据外层传来的BlockHanlde读取table文件，生成一个iterator。

merger.cc

理解了two_level_iterator，merging_iterator也不难理解了。类似于nessDB2.0中的merge，它是对多个iterator进行类似归并排序中的merge。

上一篇 LevelDB 学习笔记 —— utils
下一篇 levelDB学习笔记——Version

我的同类文章

数据库（9）

- levelDB学习笔记——Version 2013-08-30 阅读 1181
- LevelDB 学习笔记 —— utils 2013-08-19 阅读 684
- nessDB2.0 学习笔记 2013-08-18 阅读 610
- nessDB1.0学习笔记 2013-08-17 阅读 605
- BeansDB源码剖析——bitc… 2013-08-14 阅读 1145
- BeansDB源码剖析——rec… 2013-08-14 阅读 839
- BeansDB源码剖析——htre… 2013-08-14 阅读 964
- BeansDB学习笔记 2013-08-14 阅读 944
- GDBM学习笔记 2013-08-08 阅读 641

查看评论

暂无评论

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