个人资料

巴山独钓

levelDB源码分析-SSTable

标签： 存储 算法 c

2012-06-14 17:30 6259人阅读 评论(1) 收藏 举报

SSTable是Bigtable中至关重要的一块，对于LevelDB来说也是如此，对LevelDB的SSTable实现细节的了解也有助于了解Bigtable中一些实现细节。

本节内容主要讲述SSTable的静态布局结构，SSTable文件形成了不同Level的层级结构，至于这个层级结构是如何形成的我们放在后面Compaction一节细说。本节主要介绍SSTable某个文件的物理布局和逻辑布局结构，这对了解LevelDB的运行过程很有帮助。

LevelDB不同层级都有一个或多个SSTable文件（以后缀.sst为特征），所有.sst文件内部布局都是一样的。上节介绍Log文件是物理分块的，SSTable也一样会将文件划分为固定大小的物理存储块Block，但是两者逻辑布局大不相同，根本原因是：Log文件中的记录是Key无序的，即先后记录的key大小没有明确大小关系，而.sst文件内部则是根据记录的Key由小到大排列的，从下面介绍的SSTable布局可以体会到Key有序是为何如此设计.sst文件结构的关键。

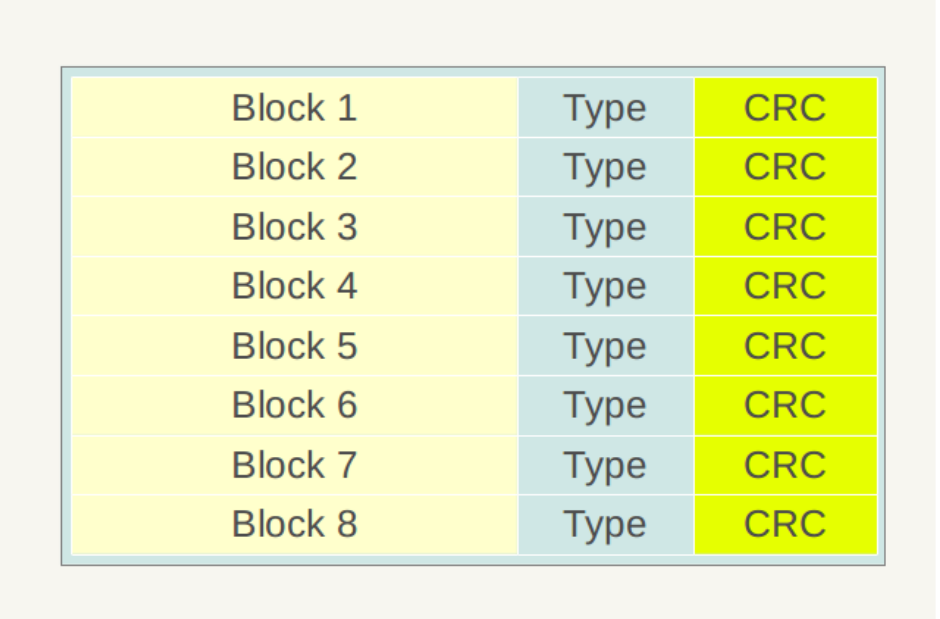

图1 .sst文件的分块结构

图1展示了一个.sst文件的物理划分结构，同Log文件一样，也是划分为固定大小的存储块，每个Block分为三个部分，包括Block、Type和CRC。Block为数据存储区，Type区用于标识Block中数据是否采用了数据压缩算法（Snappy压缩或者无压缩两种），CRC部分则是Block数据校验码，用于判别数据是否在生成和传输中出错。

以上是.sst的物理布局，下面介绍.sst文件的逻辑布局，所谓逻辑布局，就是说尽管大家都是物理块，但是每一块存储什么内容，内部又有什么结构等。图4.2展示了.sst文件的内部逻辑解释。

levelDB源码分析-Slice (2828)

评论排行

memcached源码学习-内 (5)
levelDB源码分析-Skiplist (3)
memcached源码学习-ite (2)
graphviz安装及使用 (2)
levelDB源码分析-Log文件 (2)
[python] 解析源码中的str (1)
Source Insight宏-格式化 (1)
levelDB源码分析-Cache (1)
levelDB源码分析-Arena (1)
源码自动化管理系列一： (1)

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

levelDB源码分析-TableCache
开心乐源: 最新版的sstable文件后缀是ldb

Source Insight宏 - 头文件与源文
dcdcmin: 有用， 谢谢

levelDB源码分析-Log文件
wei_tianzhu: 3q

levelDB源码分析-Log文件
沙扬娜拉的裙裾: 您好！能解释一下这里吗？SkipToInitialBlock()函数中：if (offset_in_...

graphviz安装及使用
Winterto1990: smark

levelDB源码分析-Arena
hualishiri: 这是一个轻量级的垃圾回收器吧，怎么是会内存管理池，那么他怎么进行二次内存分配那？

levelDB源码分析-SSTable：.sst
Joyhen: 牛B，这分析的给力

levelDB源码分析-Cache（LRUC
wang23109203: if (e->refs charge; (*e->deleter)...

levelDB源码分析-Skiplist
wang23109203: 遇到Port这个类时我就糊涂了，能请大神指点，或者推荐一些要看的东西？

levelDB源码分析-SSTable
chenyang2222: hao

欢迎大家交流

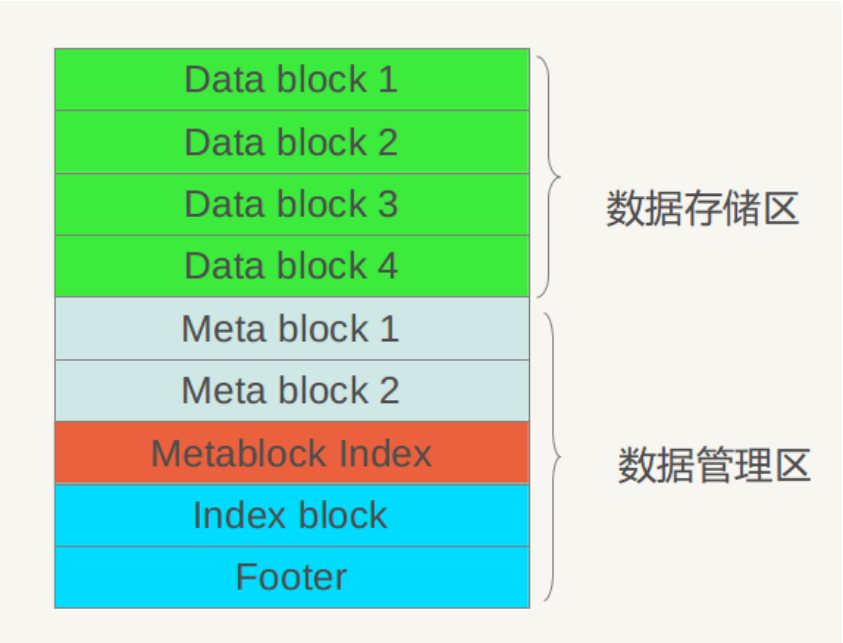

图2 逻辑布局

从图2可以看出，从大的方面，可以将.sst文件划分为数据存储区和数据管理区，数据存储区存放实际的Key:Value数据，数据管理区则提供一些索引指针等管理数据，目的是更快速便捷的查找相应的记录。两个区域都是在上述的分块基础上的，就是说文件的前面若干块实际存储KV数据，后面数据管理区存储管理数据。管理数据又分为四种不同类型：紫色的Meta Block，红色的MetaBlock Index和蓝色的Index block以及一个文件尾部块Footer。

LevelDB 1.2版对于Meta Block尚无实际使用，只是保留了一个接口，估计会在后续版本中加入内容，下面我们看看Index block和文件尾部Footer的内部结构。

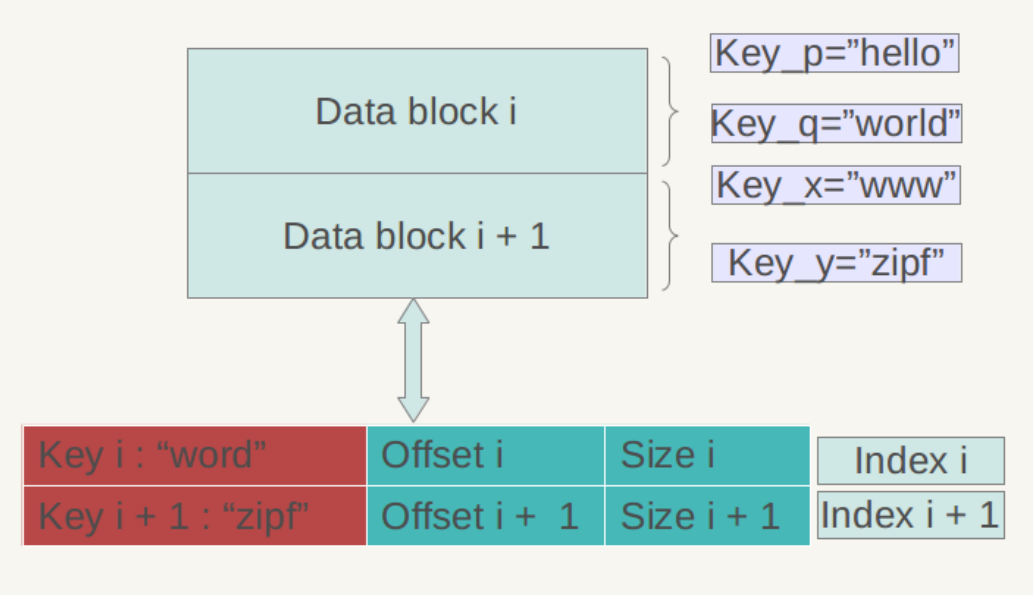

图3 Index block结构

图3是Index block的内部结构示意图。再次强调一下，Data Block内的KV记录是按照Key由小到大排列的，Index block的每条记录是对某个Data Block建立的索引信息，每条索引信息包含三个内容：Data Block中key上限值(不一定是最大key)、Data Block在.sst文件的偏移和大小，以图3所示的数据块i的索引Index i来说：红色部分的第一个字段记载大于等于数据块i中最大的Key值的那个Key，第二个字段指出数据块i在.sst文件中的起始位置，第三个字段指出Data Block i的大小（有时候是有数据压缩的）。后面两个字段好理解，是用于定位数据块在文件中的位置的，第一个字段需要详细解释一下，在索引里保存的这个Key值未必一定是某条记录的Key,以图3的例子来说，假设数据块i 的最小Key="samecity"，最大Key="the best";数据块i+1的最小Key="the fox",最大Key="zoo",那么对于数据块i的索引Index i来说，其第一个字段记载大于等于数据块i的最大Key("the best")，同时要小于数据块i+1的最小Key("the fox")，所以例子中Index i的第一个字段

是：“the c”，这个是满足要求的；而Index i+1的第一个字段则是“zoo”，即数据块i+1的最大Key。

文件末尾Footer块的内部结构见图4，metaindex_handle指出了metaindex block的起始位置和大小；inex_handle指出了index Block的起始地址和大小；这两个字段可以理解为索引的索引，是为了正确读出索引值而设立的，后面跟着一个填充区和魔数（0xdb4775248b80fb57）。

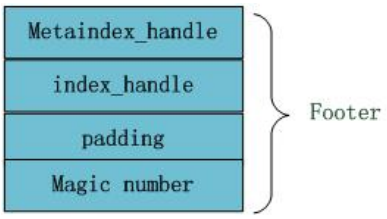

图4 Footer

上面主要介绍的是数据管理区的内部结构，下面我们看看数据区的一个Block的数据部分内部是如何布局的，图5是其内部布局示意图。

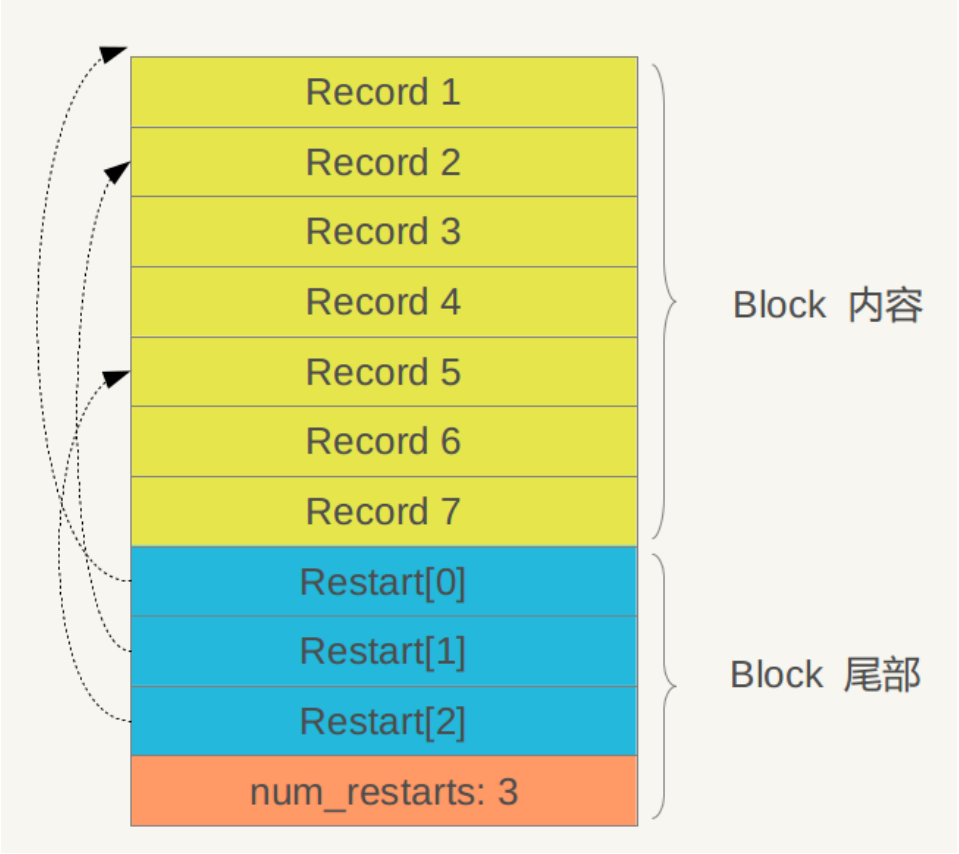

图5 Data Block内部结构

从图中可以看出，其内部也分为两个部分，前面是一个个KV记录，其顺序是根据Key值由小到大排列的，在Block尾部则是一些“重启点”（Restart Point）,其实是一些指针，指出Block内容中的一些记录位置。

“重启点”是干什么的呢？简单来说就是进行数据压缩，减少存储空间。我们一再强调，Block内容里的KV记录是按照Key大小有序的，这样的话，相邻的两条记录很可能Key叠，比如key i=“the car”，Key i+1=“the color”,那么两者存在重叠部分“the c”，y的存储量，Key i+1可以只存储和上一条Key不同的部分“olor”，两者的共同部可以获得。记录的Key在Block内容部分就是这么存储的，主要目的是减少存启点”的意思是：在这条记录开始，不再采取只记载不同的Key部分，而是重的Key值，假设Key i+1是一个重启点，那么Key里面会完整存储“the color”，而略的“olor”方式。但是如果记录条数比较多，随机访问一条记录，需要从头开始行，这样也产生很大的开销，所以设置了多个重启点，Block尾部就是指出哪些重启点的。

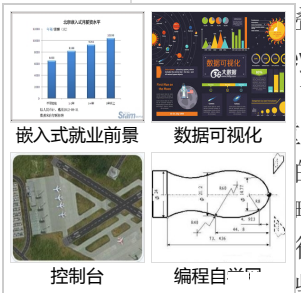

图6 记录格式

在Block内容区，每个KV记录的内部结构是怎样的？图6给出了其详细结构，每个记录包含5个字段：key共享长度，key非共享长度，value长度，key非共享内容，value内容。比如上面的“the car”和“the color”记录，key共享长度5；key非共享长度是4；而key非共享内容则实际存储“olor”；value长度及内容分别指出Key:Value中Value的长度和存储实际的Value值。

上面讲的这些就是.sst文件的全部内部奥秘。

Block格式及相关操作请参阅《levelDB源码分析-SSTable：Block》。

SSTable造作相关请参阅《levelDB源码分析-SSTable：.sst文件的构建与读取》