2018.03.14

简述

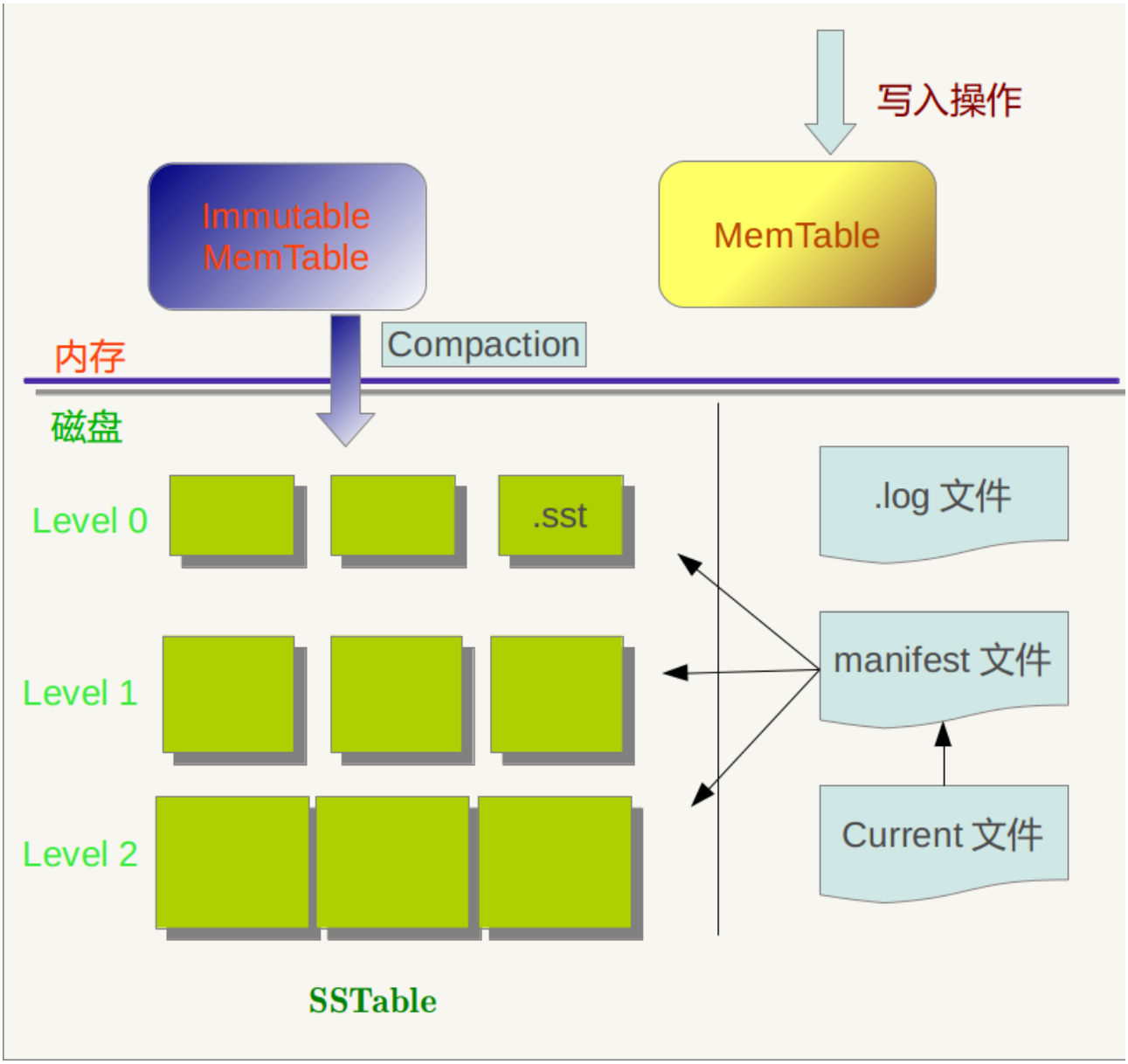
LevelDB可以理解成一个简化版的Tablet Server（即HBase中Region Server）。

特点：
LevelDb有如下一些特点：

1. LevelDb性能非常突出，官方网站报道其随机写性能达到40万条记录每秒，而随机读性能达到6万条记录每秒。总体来说，LevelDb的写操作要大大快于读操作，而顺序读写操作则大大快于随机读写操作。
2. 首先，LevelDb是一个持久化存储的KV系统，和Redis这种内存型的KV系统不同，LevelDb不会像Redis一样狂吃内存，而是将大部分数据存储到磁盘上。
3. 其次，LevleDb在存储数据时，是根据记录的key值有序存储的，就是说相邻的key值在存储文件中是依次顺序存储的，而应用可以自定义key大小比较函数，LevleDb会按照用户定义的比较函数依序存储这些记录。
4. 再次，像大多数KV系统一样，LevelDb的操作接口很简单，基本操作包括写记录，读记录以及删除记录。也支持针对多条操作的原子批量操作。
5. 另外，LevelDb支持数据快照（snapshot）功能，使得读取操作不受写操作影响，可以在读操作过程中始终看到一致的数据。
6. LevelDb还支持数据压缩等操作，这对于减小存储空间以及增快IO效率都有直接的帮助。

架构

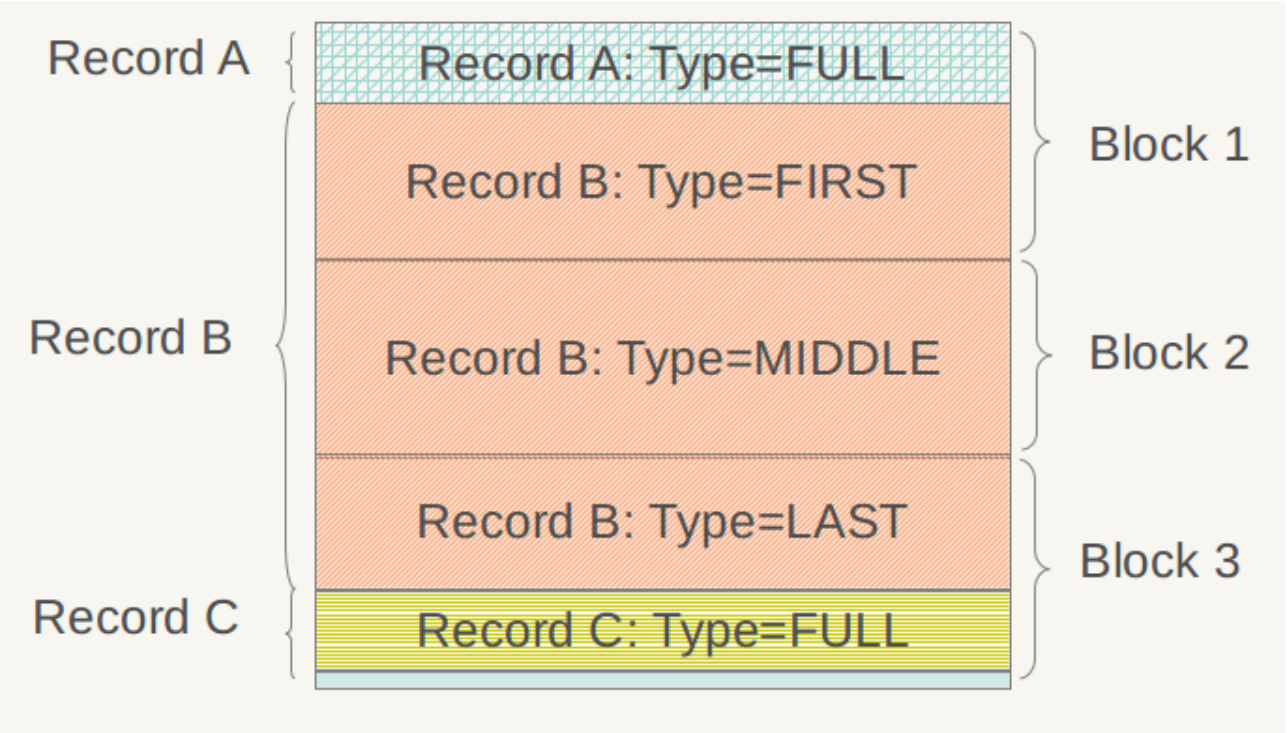
1. MemTable：用户写入时直接写入到内存的MemTable中，本质上是一个按照Key排序的SkipList；
2. Immutable MemTable：只读的MemTable，当MemTable的大小到达阀值（2MB）时变为Immutable MemTable；
3. SSTable文件：磁盘上的数据文件，第i+1层SSTable由第i层的合并生成，第0层的SSTable是从Immutable MemTable直接导出的。必须注意的一点是，第0层的SSTable之间可能存在重复的Key值（**因为是直接从*Immutable MemTable*导出成文件的，因此只能保证同一个*SSTable*，*key*已序并且没有重复**），其他层级就不会。
4. LOG文件：编辑日志
5. mainifest文件：记录SSTable的文件名、所属的层级、最大/小的Key值。
6. Current文件：当前的mainifest文件



文件的物理布局

Log文件

LevelDb对于一个log文件，会把它切割成以32K为单位的物理Block，每次读取的单位以一个Block作为基本读取单位，下图展示的log文件由3个Block构成，所以从物理布局来讲，一个log文件就是由连续的32K大小Block构成的。
一条Record记录存放在一个或者多个Block中，如下图。一个Block中的Record对应了四种状态：Full、First、Middle、Last。



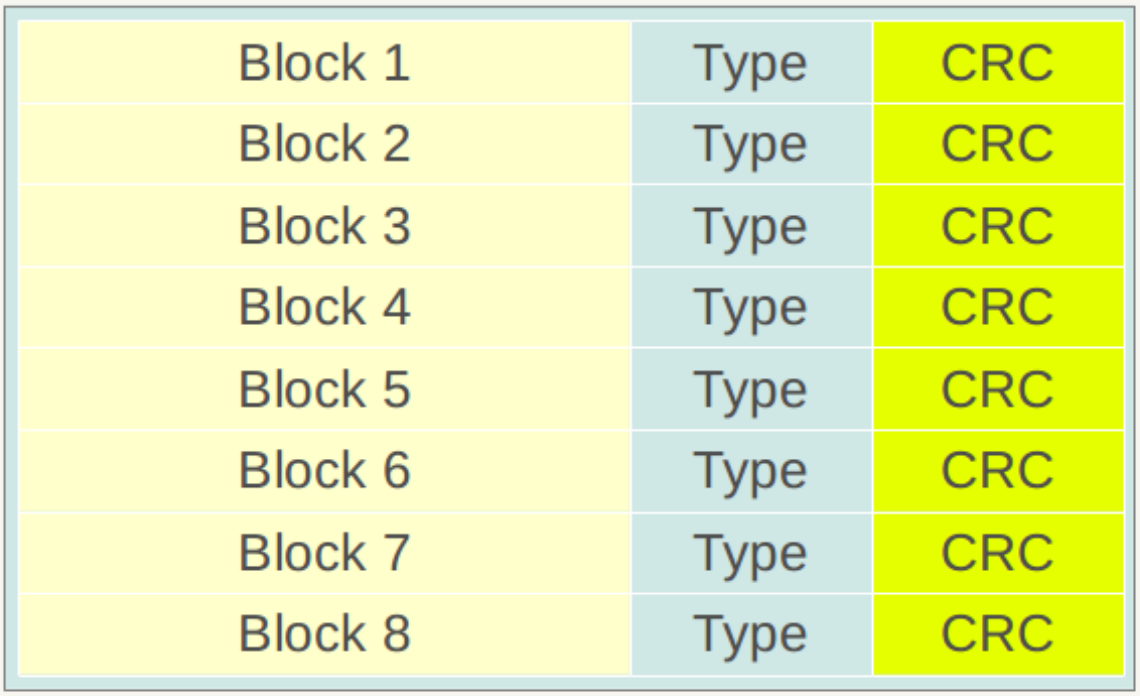
LOG文件Record的结构

类型：FULL/FIRST/MIDDLE/LAST

SSTable

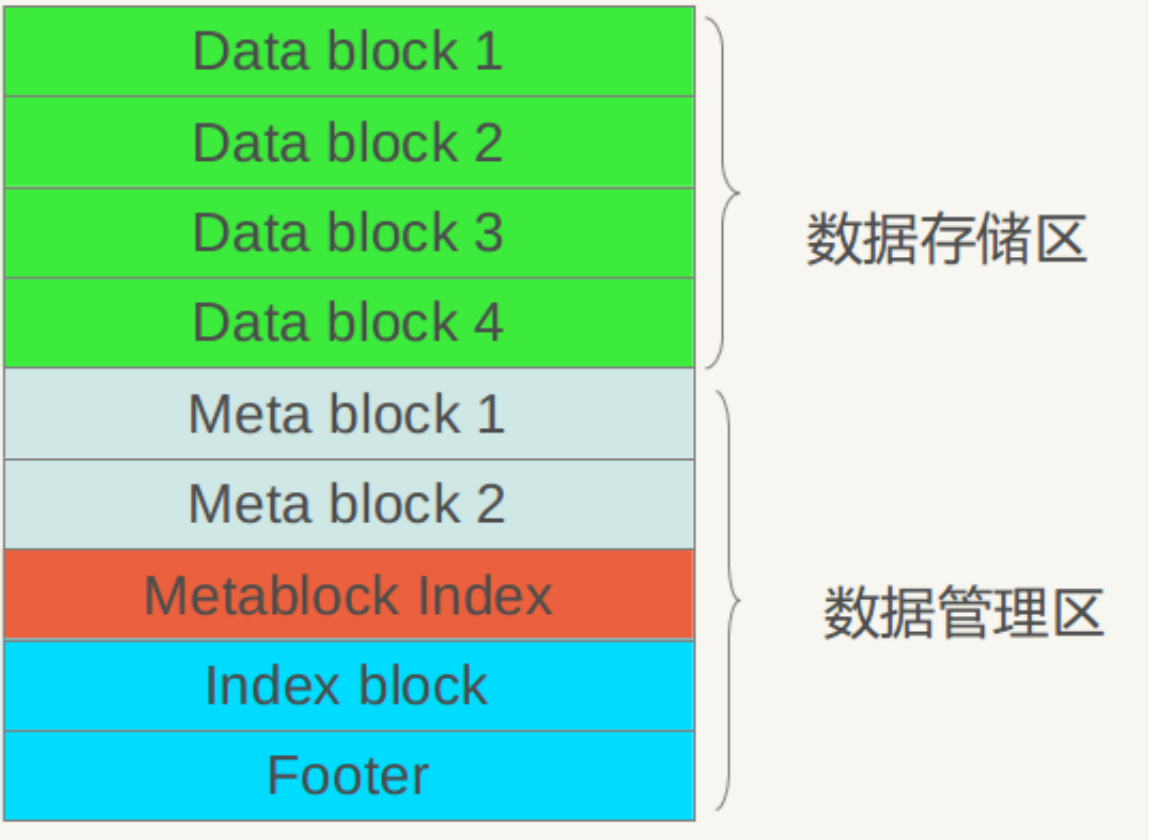
SSTable也一样会将文件划分为固定大小的物理存储块，需要注意：Log文件中的记录是Key无序的，即先后记录的key大小没有明确大小关系，而.sst文件内部则是根据记录的Key由小到大排列的。

每个Block分为三个部分，红色部分是数据存储区， 蓝色的Type区用于标识数据存储区是否采用了数据压缩算法（ Snappy压缩或者无压缩两种 ），CRC部分则是数据校验码，用于判别数据是否在生成和传输中出错。



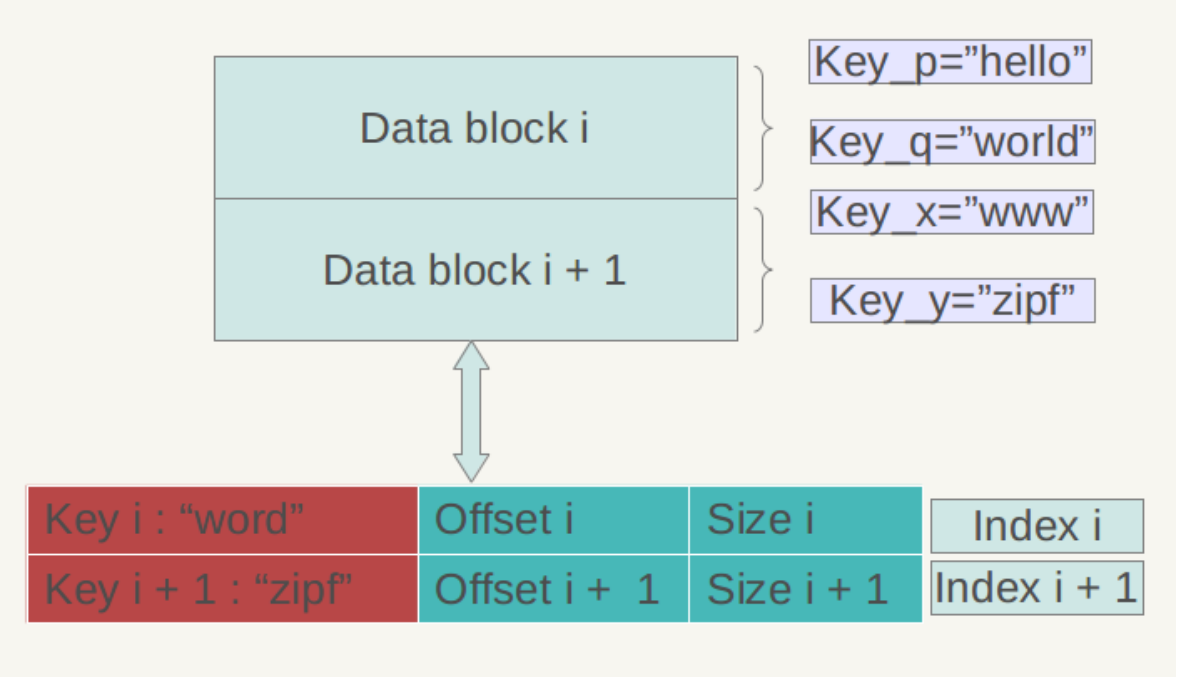
sst文件将每个Block划分为**数据存储区**和**数据管理区**。

- 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。管理数据又分为四种不同类型：紫色的Meta Block，红色的Meta Block 索引和蓝色的数据索引块以及一个文件尾部块。



数据管理区

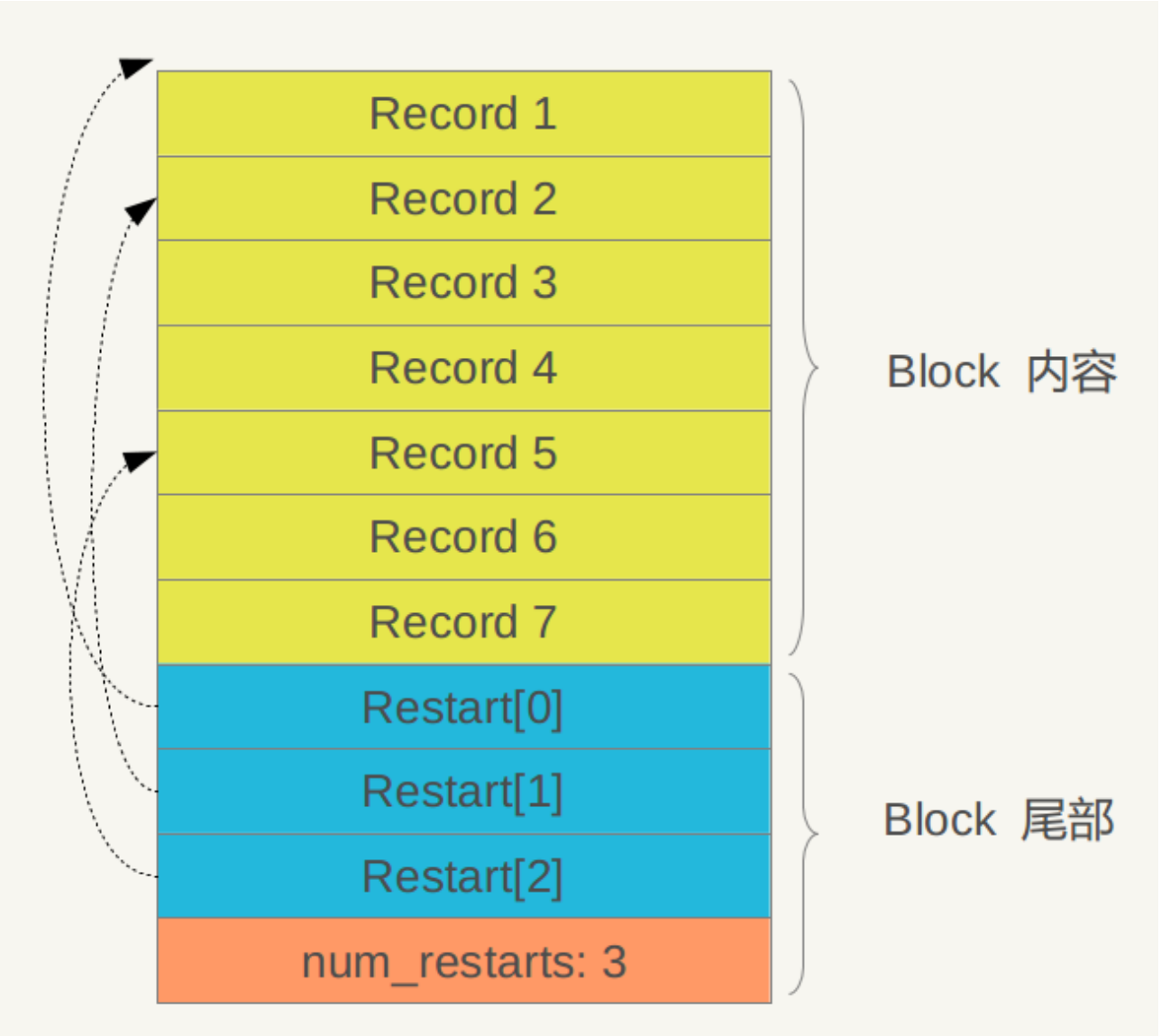
1. Meta Block和Meta Block 索引在LevelDB1.2以前的版本占时没有用；
2. 数据索引:数据索引区的每条记录是对某个Data Block建立的索引信息，每条索引信息包含:
   - 大于等于数据块i中最大的Key值的那个Key，同时要小于数据块i+1的最小key
   - 数据块i在.sst文件中的起始位置
   - 数据块i在.sst文件中的大小



3. Footer块：包括Metaindex_handle（metaindex block的起始位置和大小）、index_handle（index Block的起始地址和大小）、padding（填充区）、Magic number（魔数）

数据区

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）。



记录的格式：

重启点”是干什么的呢？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部分存在重叠，比如key i=“the Car”，Key i+1=“the color”,那么两者存在重叠部分“the c”，为了减少Key的存储量，Key i+1可以只存储和上一条Key不同的部分“olor”，两者的共同部分从Key i中可以获得。记录的Key在Block内容部分就是这么存储的，主要目的是减少存储开销。“重启点”的意思是：在这条记录开始，不再采取只记载不同的Key部分，而是重新记录所有的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”，而不是采用简略的“olor”方式。Block尾部就是指出哪些记录是这些重启点的。

写入操作

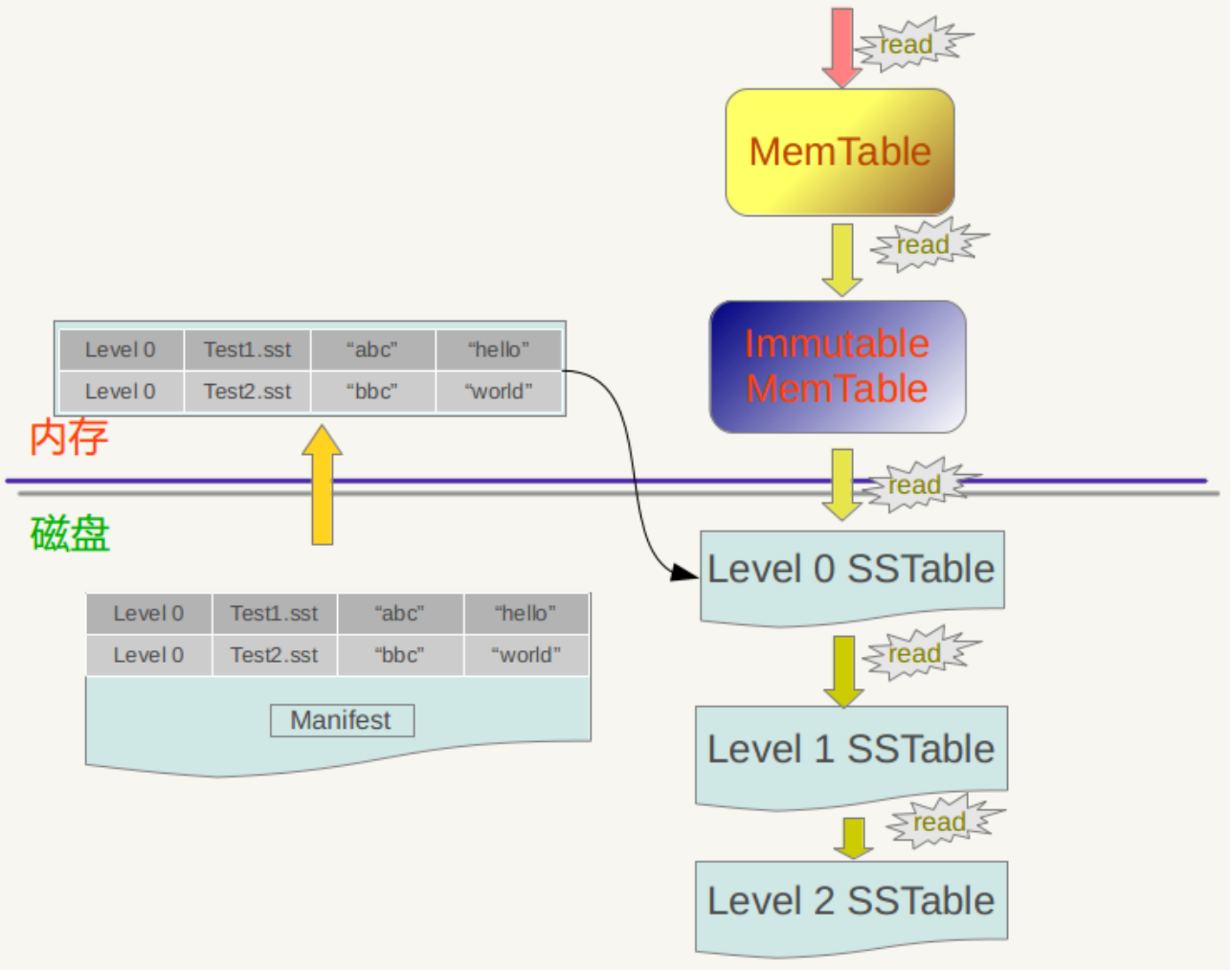
追加一条记录到Log文件，插入记录到MemTable。

读取操作

从最新的数据开始往下找，如果同时在level L和Level L+1找到同一个key，level L的信息一定比level L+1的要新。

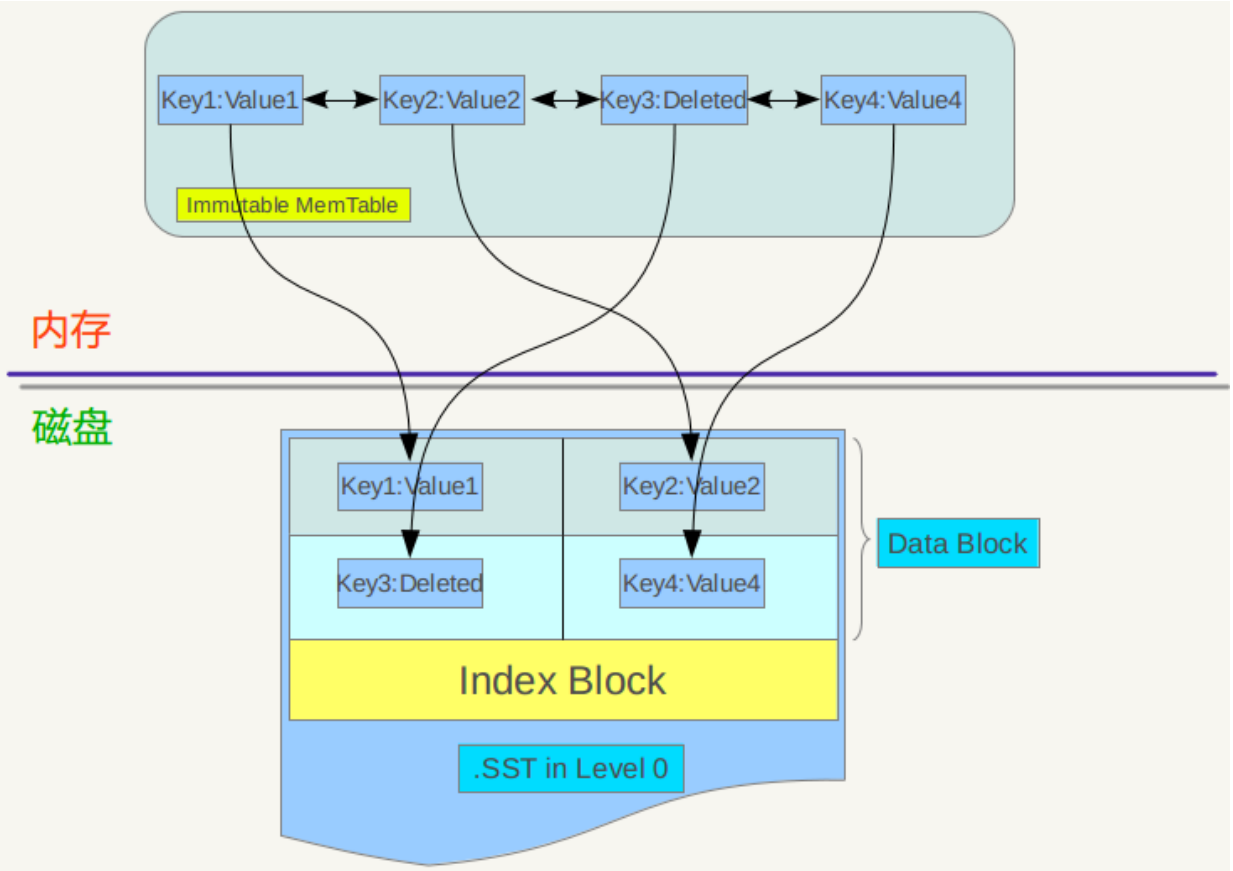
从SSTable文件查Key的过程：

1. 从manifest文件中找出包含Key的所有SSTable文件。之后按照文件的新鲜程度排序，新的文件排在前面，之后依次查找。
2. 非level 0的话，因为这个level的文件之间key是不重叠的，所以只从一个文件就可以找到key对应的value。

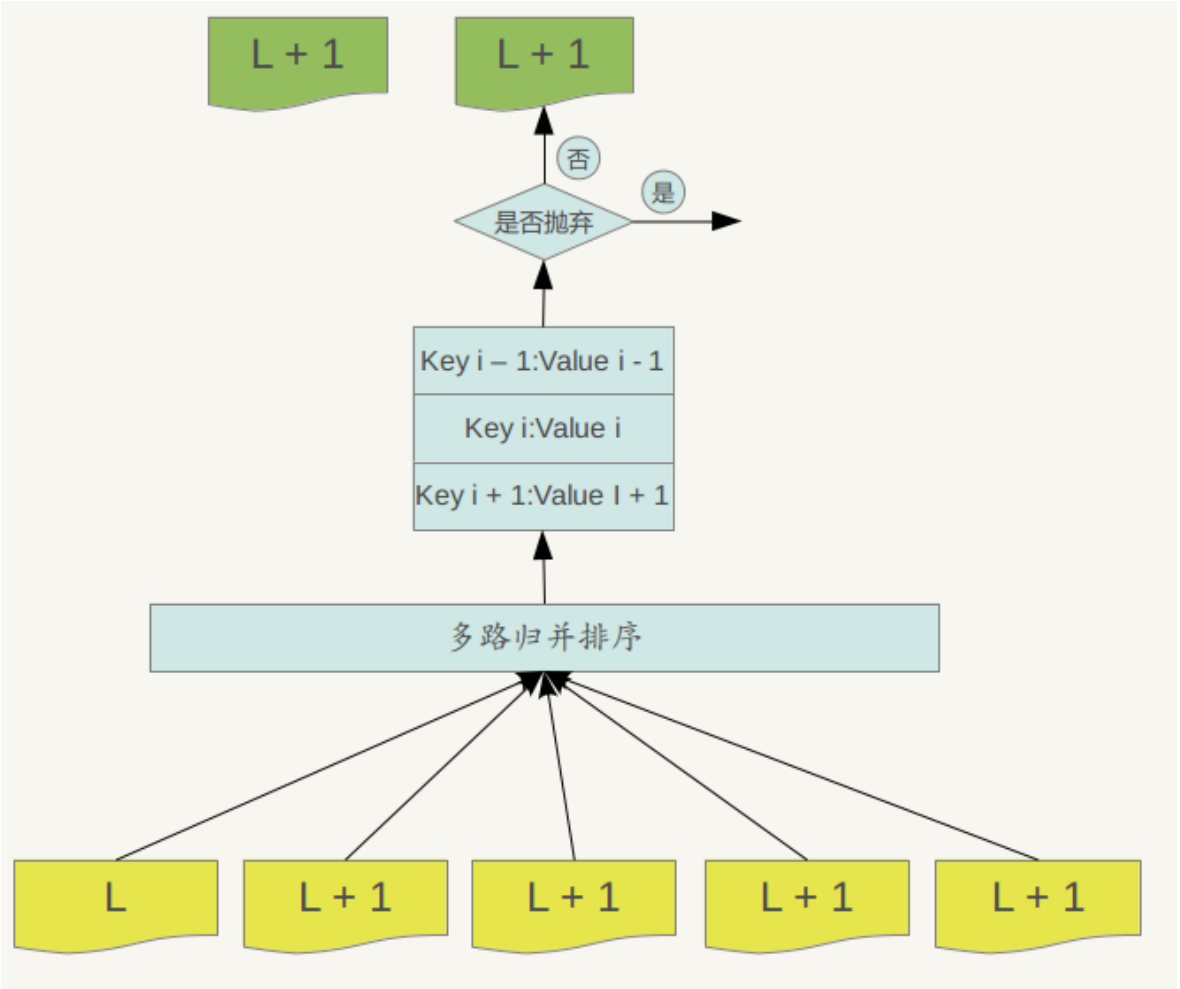


Compaction操作

1. minor Compaction：就是把memtable中的数据导出到SSTable文件中。



2. major compaction：当某个level下的SSTable文件数目超过一定设置值后，levelDb会从这个level的SSTable中选择一个文件（level>0），将其和高一层级的level+1的SSTable文件合并。
    - 在做major compaction的时候，对于大于level 0的层级，选择其中一个文件就行，但是对于level 0来说，指定某个文件后，本level中很可能有其他SSTable文件的key范围和这个文件有重叠，这种情况下，要找出所有有重叠的文件和level 1的文件进行合并，即level 0在进行文件选择的时候，可能会有多个文件参与major compaction。
    - major compaction时选择的SSTable文件是轮流来的
    - 合并算法是多路归并排序
    - LevelDB合并后会判断SSTable有没有保存价值，如果觉得还需要继续保存。（对于层级低于L的文件中如果存在同一Key的记录，那么说明对于Key来说，有更新鲜的Value存在，那么过去的Value就等于没有意义了，所以可以删除。）


L + 1
L + 1
否
是
是否抛弃
Key i – 1:Value i - 1
Key i:Value i
Key i + 1:Value I + 1
多路归并排序
L
L + 1
L + 1
L + 1
L + 1