

LogMergePolicy

本篇文章介绍索引文件的合并策略，某次提交(commit)或者刷新(flush)的所有索引文件属于一个新的段(Segment)，所以也可以称为段合并(Segment Merge)。当IndexWriter索引中的数据有任意修改动作，它会调用findMerges(...)方法通过某个合并策略(MergePolicy)来找出需要合并的段集，如果需要合并，那么合并策略会返回一个oneMerge的集合，oneMerge的个数描述了IndexWriter需要执行合并的次数，单个oneMerge中包含了需要合并为一个新段(New Segment)的段集合。Lucene4版本开始，默认的合并策略为TieredMergePolicy，之前是LogMergePolicy，根据实际的业务需要，某些场景需要用到LogMergePolicy，故本篇文章先介绍这种策略。

LogMergePolicy的一些参数

DEFAULT_NO_CFS_RATIO

默认值为0.1，如果我们的索引文件设置为复合文件(compound file)，那么.doc、.pos、.fdx等等索引文件在执行段合并后会生成一个复合文件，这个文件可能会很大，所以在合并前会判断将要生成的复合文件的大小是否会超过总的索引文件的10%，如果超过，那么合并后就不会生成复合文件。

mergeFactor(可配置)

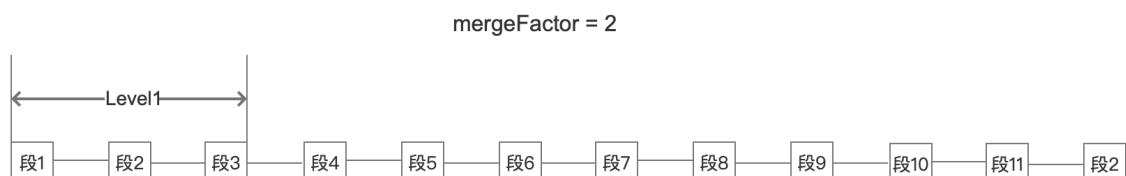
mergeFactor描述了IndexWriter执行一次合并操作的段的个数，即上文中提到的单个oneMerge中段集合大小。

level

合并过程中根据策略(后面会介绍)会将连续的段分为不同的层级(level)，遍历每一个层，然后在每一层中继续遍历每一个段，找到mergeFactor个段作为一个oneMerge(前提是满足合并条件，下文会介绍)。

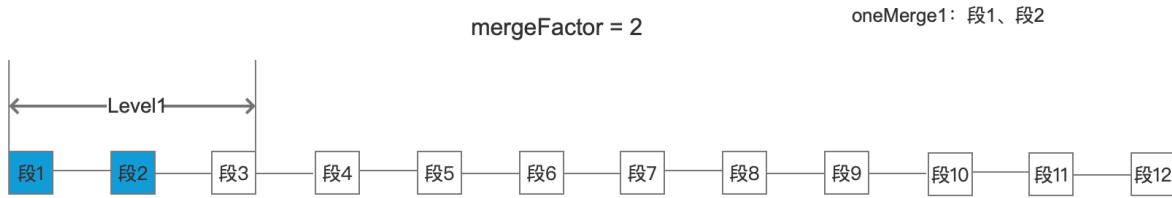
- 根据策略找到第一层

图1：



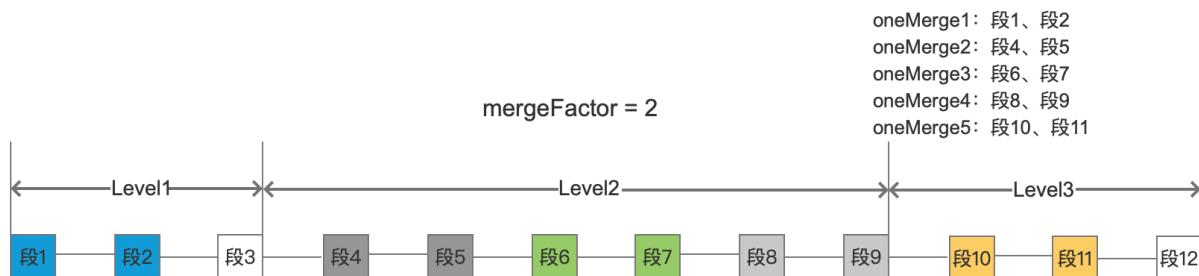
- 在第一层中遍历找到 mergeFactor个段，当前例子中mergeFactor的值为2，段1跟段2作为一个oneMerge，这里称为oneMerge1。由于第一层中的只有一个段3，所以无法生成一个oneMerge。

图2：



- 同理根据策略(后面会介绍)找到第二层、第三层。并且生成5个oneMerge，这些oneMerge就会交给IndexWriter，IndexWriter会执行5次合并操作。

图3：



所以上面是两层遍历，时间复杂度为 $O(n^2)$ 。

minMergeSize、maxMergeSize(可配置)

- maxMergeSize：某个段超过该值，这个段永远会被合并。
- minMergeSize：该值用来分层，关键的作用是处理数量很多的小段。如果minMergeSize设置的比较大，那么会导致划分很多的层，很多层意味着性能的降低，因为找出oneMerge是 $O(n^2)$ 的时间复杂度，如果这个值设置的较为合适，那么会将许多的小段划分为一层，意味着两层循环的外层循环次数会减少，那么就能有效的将这些小段生成一个oneMerge。

maxMergeDocs(可配置)

默认值为Integer.MAX_VALUE，如果段中的文档数超过这个值，那么该段永远会被合并。

calibrateSizeByDeletes(可配置)

该值描述了是否要标记段中的被删除的文档，如果该值为true，那么被删除的文档的个数或大小不会作为该段的一部分。

段的大小(Segment Size)

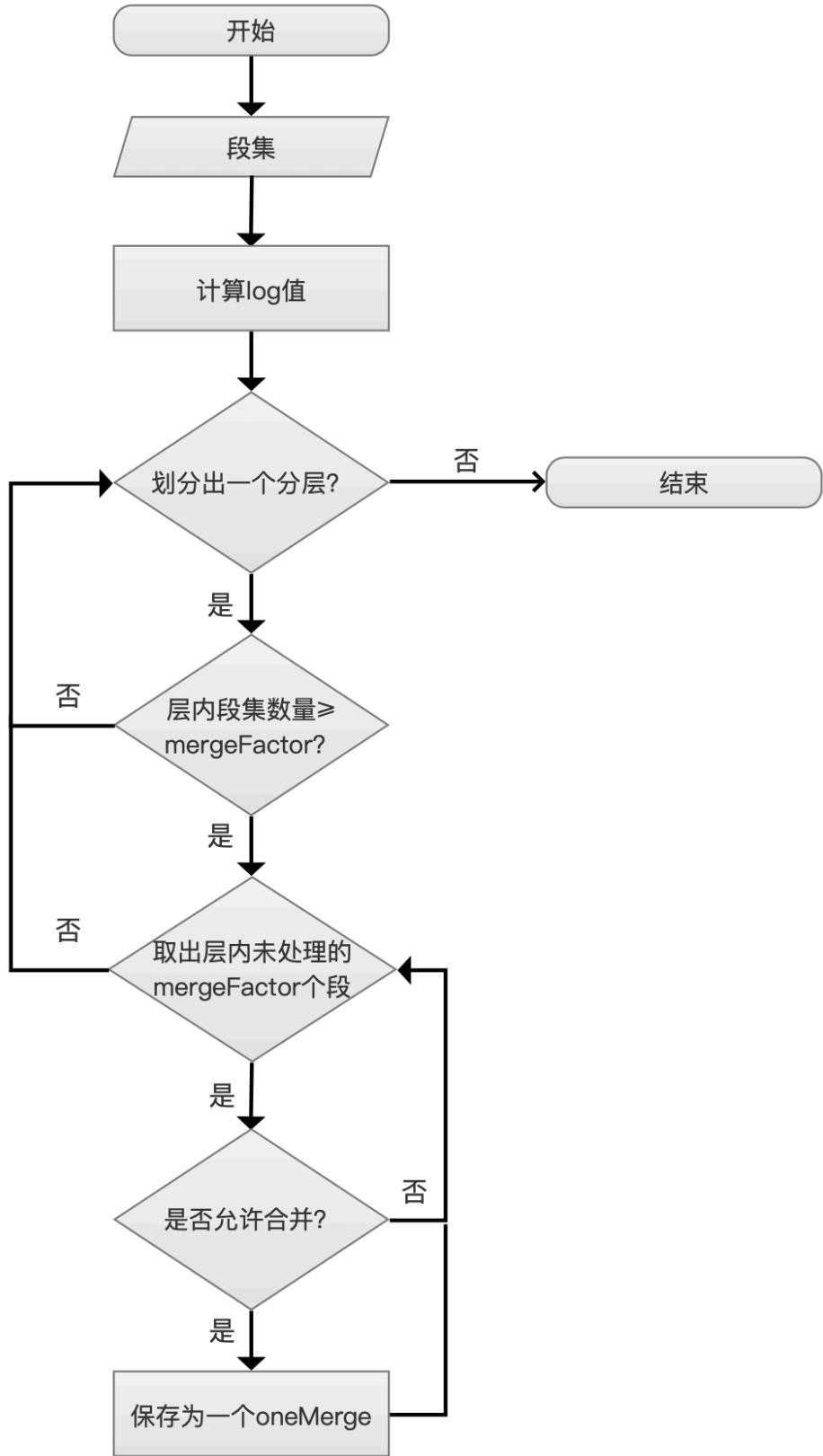
有两种方式来描述一个段的大小

- 文档数量：一个段中的文档数量可以用来描述段的大小，根据calibrateSizeByDeletes的值判断被删除的文档号是否也作为段的大小的一部分，例如LogDocMergePolicy就使用了该方法来计算段的大小
- 索引文件大小：一个段中包含的所有索引文件大小总和，在Lucene7.5.0版本中除了

LogDocMergePolicy，其他的合并策略都使用该方法

流程图

图4：



LogByteSizeMergePolicy跟LogDocMergePolicy两种合并策略都属于LogMergePolicy，他们使用相同的逻辑执行段的合并，差别在于对Segment Size有不同的定义，上文中已经介绍，下文中以**LogDocMergePolicy**来作为例子进行介绍，即**Segment Size**描述了一个段中包含的文档个数。

开始

图5：

开始

当IndexWriter对索引有任意的更改都会调用合并策略。

段集

图6：

段集

IndexWriter会提供一个段集(段的集合)给合并策略。

计算log值

图7：

计算log值

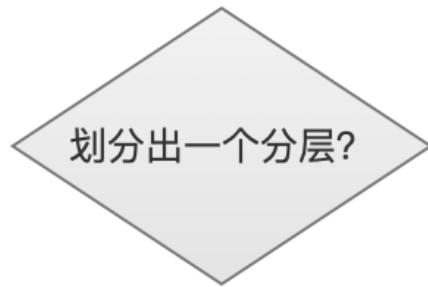
由于每个段中的文档号数量各不相同，最小值跟最大值可能相差很大，使用MergePolicy可能使得永远无法进行合并操作(看完下文中的合并策略就明白为什么啦)，所以计算log值作为一个平滑操作，故我们需要的并不是每个段的真实大小值，而是段之间的相对大小值，类似归一化处理。计算公式如下：

$$\log(\text{SegmentSize}) / (\text{mergeFactor})$$

SegmentSize跟mergeFactor在上文中已经介绍了。

划分出一个分层？

图8:



在上文中介绍level时，我们知道了段集会被划分为多个层级，这里会介绍如何划分出一个层级，划分规则依据两个参数：

- maxLevel: 该值即未处理过的段集中log最大值。
- LEVEL_LOG_SPAN: 默认值为0.75, 该值不可配置，你要是问我为什么是0.75,而不是其他值，我也不知道~ 😅
- levelBottom: 该值为一个差值: $\text{maxLevel} - \text{LEVEL_LOG_SPAN}$

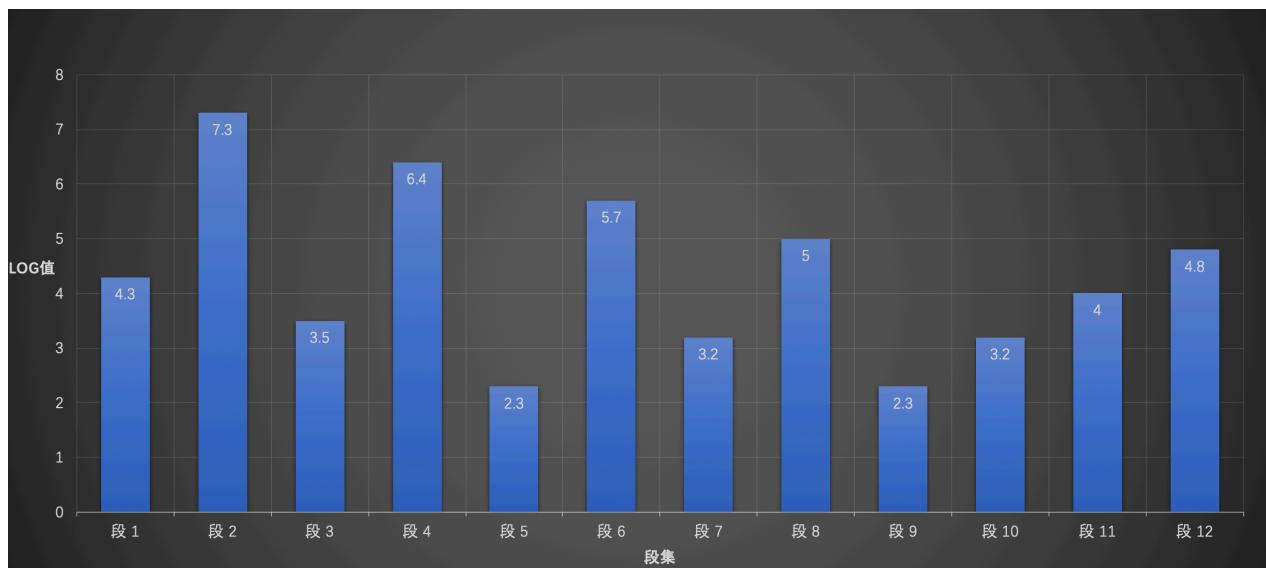
选出一个层级的逻辑分为三个步骤：

- 步骤一：找出未处理的段集中的maxLevel
- 步骤二：从最后一个段集开始往前遍历，找到第一个大于等于levelBottom，遍历到maxLevel停止
- 步骤三：未处理的段集中第一个段到levelBottom所在的段的区间范围内的所有段处理为一个层级

例子

下面的例子中，颜色为蓝色的柱状为未处理的段集。

图9:



第一层

- 步骤一

找出未处理的段集(段1~段12)中的maxLevel: 即段2。

- 步骤二

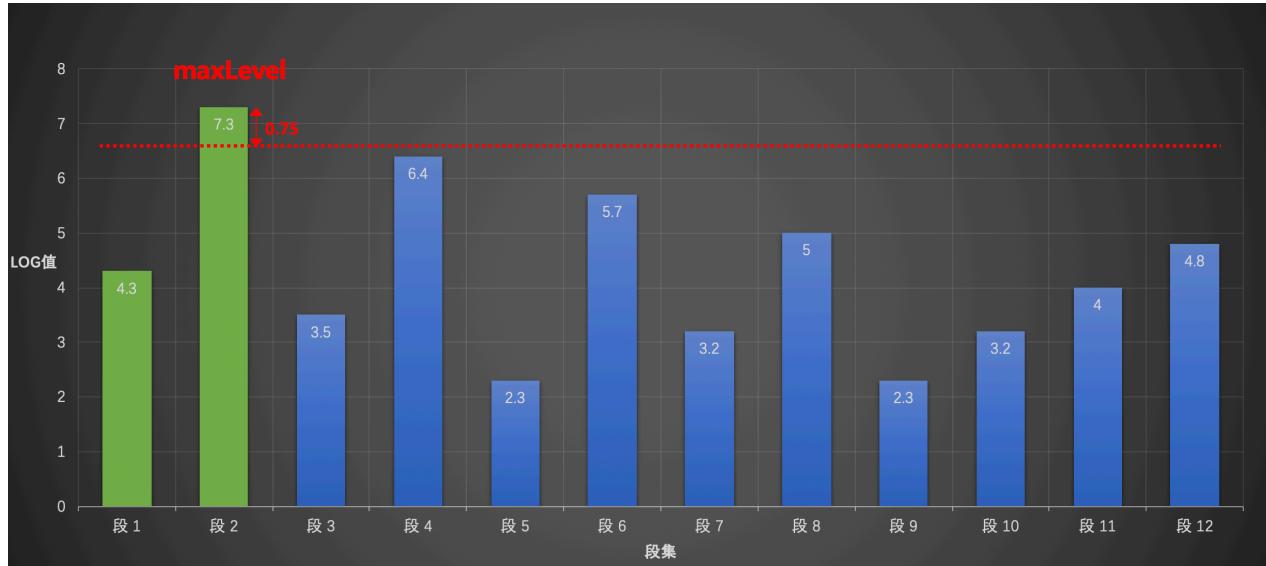
从后往前遍历，找到第一个大于levelBottom：即段2。

```
levelBottom = (maxLevel - LEVEL_LOG_SPAN), levelBottom = (7.3 - 0.75), 即  
levelBottom = 6.55
```

- 步骤三

未处理的段集中第一个段(段1)到levelBottom所在的段的区间范围内的所有段集为一个层级：段1~段2。

图10：



当划分出一个层级以后就可以进入下一步流程了，但是这里我们直接将其他所有的层级都先标记出来吧。

第二层

- 步骤一

找出未处理的段集(段3~段12)中的maxLevel：即段4。

- 步骤二

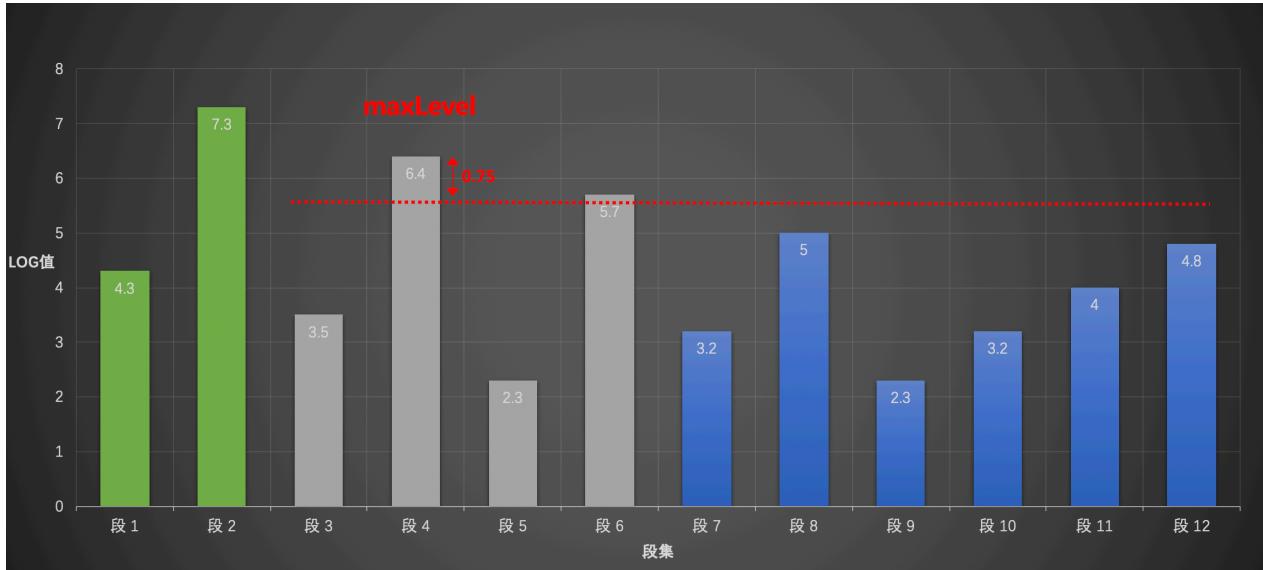
从后往前遍历，找到第一个大于levelBottom：即段6。

```
levelBottom = (maxLevel - LEVEL_LOG_SPAN), levelBottom = (6.4 - 0.75), 即  
levelBottom = 5.65
```

- 步骤三

未处理的段集中第一个段(段3)到levelBottom所在的段的区间范围内的所有段集为一个层级：段3~段6。

图11：



第三层

- 步骤一

找出未处理的段集(段7~段12)中的maxLevel：即段8。

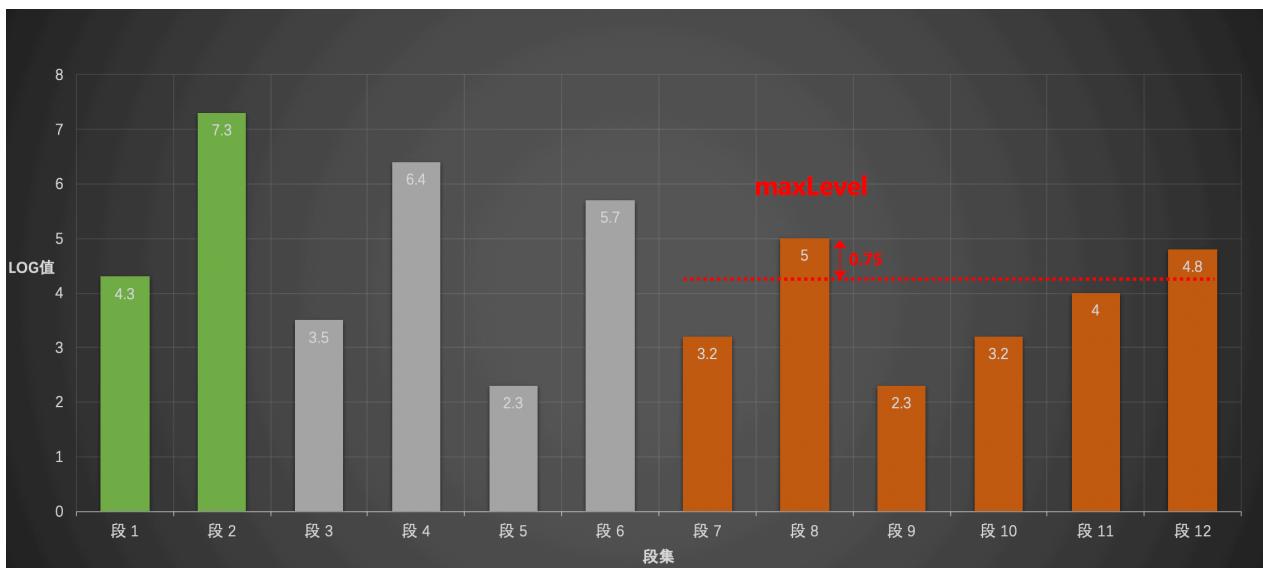
- 步骤二

从后往前遍历，找到第一个大于levelBottom：即段12。

```
levelBottom = (maxLevel - LEVEL_LOG_SPAN), levelBottom = (5 - 0.75), 即
levelBottom = 4.25
```

- 步骤三 未处理的段集中第一个段(段7)到levelBottom所在的段的区间范围内的所有段集为一个层级：段7~段12。

图12：



层内段集数量 \geq mergeFactor?

图13：



mergeFactor定义了每次合并需要的段的个数，如果如果当前层内的段子集数量小于该值，那么就不用进行层内遍历了。

取出层内未处理的mergeFactor个段

图14:

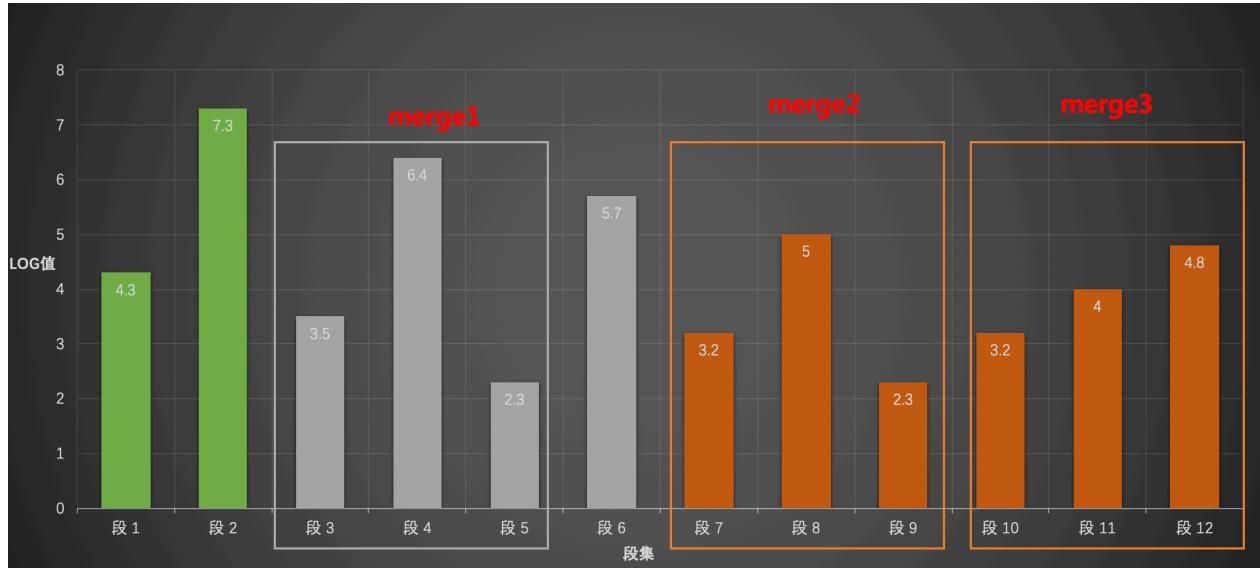


当前层内的段子集个数不小于mergeFactor，那么顺序遍历层内的段子集，每次取出mergeFactor个段，并处理为一个merge，某些层内可以取出多个merge。

例子

下面是当MergeFactor = 3 时，每一层可以取出的的merge：

图15:



在第一层中，由于层内段子集个数小于MergeFactor，所以无法生产一个merge，同理在第二层中，可以获得一个merge，即merge1，并且段6无法生成新的merge，同理在第三层中可以生成2个merge，即merge2、merge3。注意的是，这里的merger1、merge2、merge3并不是最后的允许合并的段，即不是oneMerge。

是否允许合并？

图16：



在上面的流程中，我们在一个层级内获得一个或多个merge，需要继续判断这些merge是否允许生成oneMerge，即真正提供给IndexWriter执行合并的段集。

当IndexWriter获得一个oneMerge后，会使用后台线程对oneMerge中的段进行合并，那么这时候索引再次发生更改时，IndexWriter会再次调用MergePolicy，可能会导致某些已经正在合并的段被处理为一个新的oneMerge，为了防止重复合并，在生成新的oneMerge前需要判断某些段是否允许合并。

判断的方法有下面几个步骤：

- 步骤1：后台合并的线程会将正在合并的段添加到Set对象中，在IndexWriter调用合并策略时传入
- 步骤2：MergePolicy在层内取出mergeFactor个段的过程中，会逐个判断每一个段是否在Set对象中（相同的引用），接着还要判断一个段的大小是否大于等于maxMergeSize，正如上文中介绍的，超过maxMergeSize的段是不需要合并的，并且这个merge中的段集此次都会被认为是不允许合并的

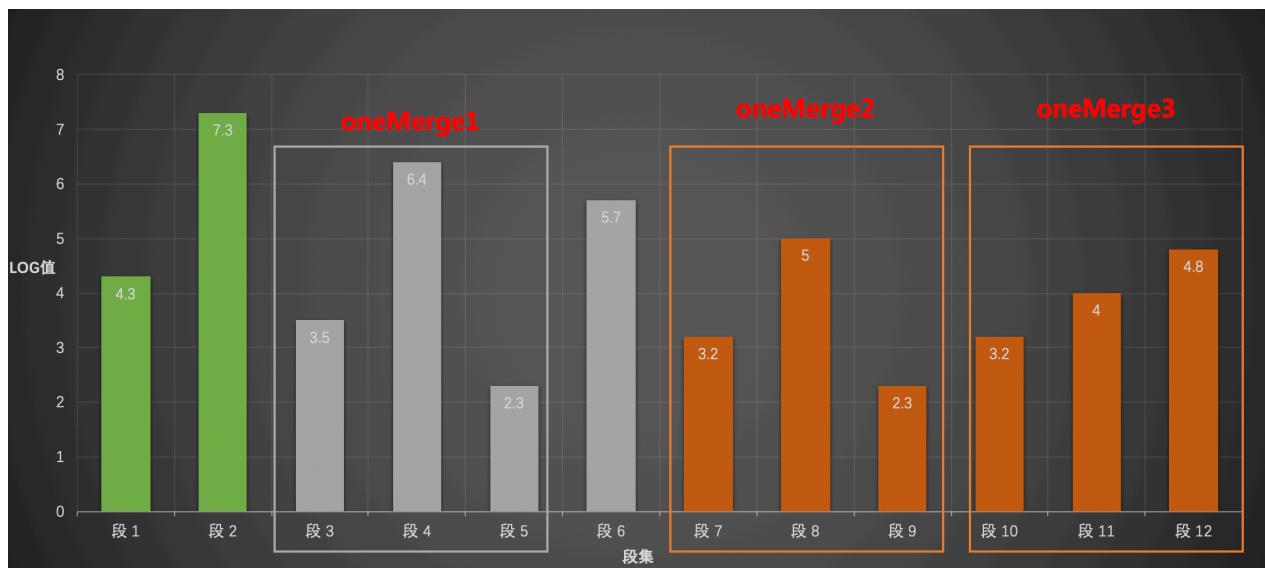
保存为一个oneMerge

图17:

保存为一个oneMerge

在上一步流程中一个merge被认为是允许合并后，那么这个merge处理为一个oneMerge。如果上一步流程中所有的merge都允许合并，那么这些merge就会分别处理为一个oneMerge。

图18:



结束

图19:

结束

每个层级生成的多个oneMerge添加到一个集合中，将这个集合交给IndexWriter，IndexWriter会开启一个后台线程去执行合并操作。

结语

本篇文章介绍了段的合并策略LogMergePolicy，LogMergePolicy并不负责段的合并，而是找出可以合并的段集，然后IndexWriter会开启后台线程合并这些段集，而这个后台线程在源码中即MergeScheduler，在后面的文章中会介绍MergeScheduler是如何实现段的合并。另外在MergePolicy中，还有一些其他的方法，例如findForcedMerges(...)、findForcedDeletesMerges(...)等等方法，在以后介绍IndexWriter时，会详细介绍。

[点击下载](#)Markdown文件