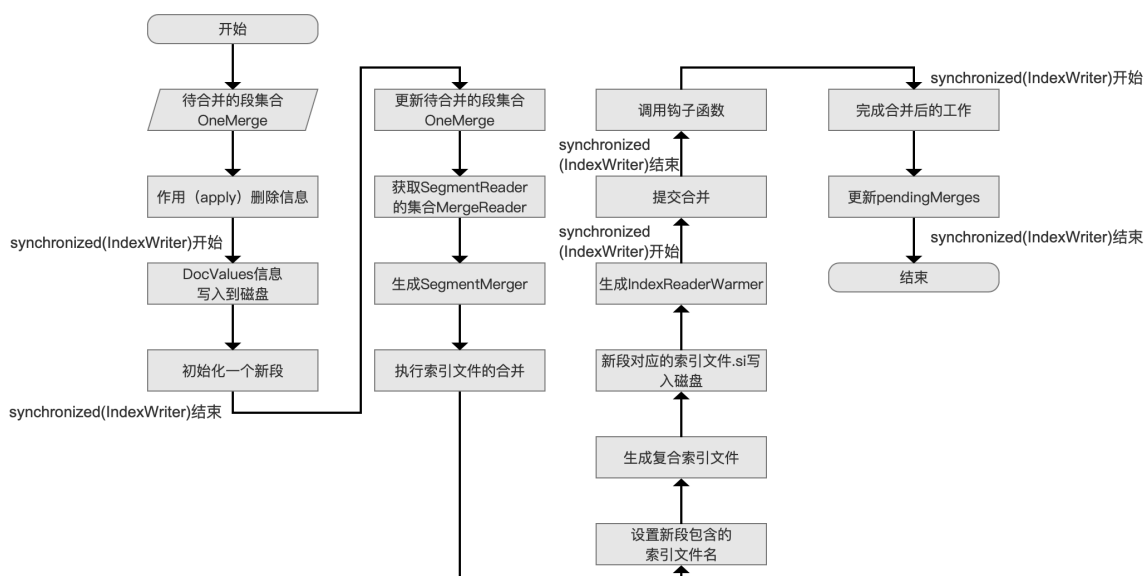


## 执行段的合并（二）

本文承接[执行段的合并（一）](#)，继续介绍执行段的合并的剩余的流程，下面先给出执行段的合并的流程图：

图1：



[点击查看大图](#)

## DocValues信息写入到磁盘

在上一个流程点 `作用 (apply) 删除信息` 执行结束后，待合并的段中新增的删除信息目前还存储在内存中，此时需要持久化DocValues信息，即写入到磁盘。

删除信息可以分为下面两类：

- 被删除的文档号：这类删除信息使用[FixedBitSet](#)存储，按照Term进行删除、按照Query进行删除、更新文档操作这三种操作找出的文档号都是被删除的文档号
- DocValues信息：这类删除信息使用链表存储，这里不展开介绍，在以后介绍软删除的文章中会展开，更新DocValues域的操作会产生这类删除信息

在源码中，该流程点有以下的TODO注释：

```
TODO: we could fix merging to pull the merged DV iterator so we don't have to
move these updates to disk first, i.e. just carry them in memory:
```

也就是说没有必要在这个流程将变更的DocValues信息写入到磁盘，由于由于执行段的合并跟[文档的增删改](#)，文档提交（[commit](#)、[flush](#)）是并发的操作，DocValues还有可能被更新，故在以后的版本，将不需要在这个位置执行该流程，而是跟处理被删除的文档号一样，通过OneMerge存储（仅仅是作者的猜测，至少被删除的文档号是这么做的），OneMerge中包含的信息见文章[执行段的合并（一）](#)

中的介绍，在后面的流程中会介绍处理被删除的文档号的过程。

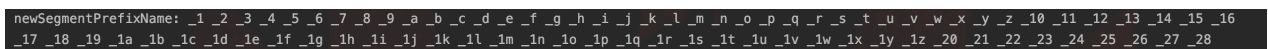
至于DocValues信息从内存持久化到磁盘的过程，在以后介绍软删除的文章中会展开，在这篇文章中我们只需要知道，当前流程执行结束后，会生成新的`.dvd`、`.dvm`的索引文件。

## 初始化一个新段

新段`newSegment`即待合并的段合并后的目标段（target segment），在合并操作中初始化一个新段的过程有以下五个步骤：

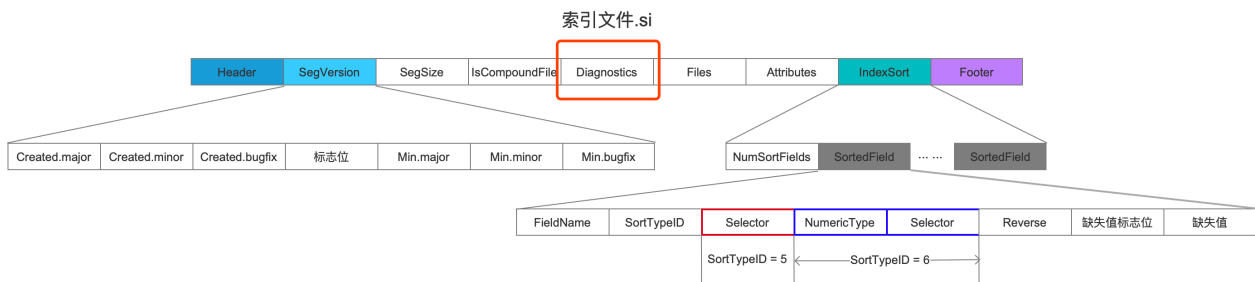
- 步骤一之获得新段的段名前缀：使用jdk提供的`Long.toString(count, Character.MAX_RADIX)`方法来获得，demo看这里：<https://github.com/LuXugang/Lucene-7.5.0/blob/master/LuceneDemo/src/main/java/lucene/index/NewSegmentTest.java>，下图的例子显示的是连续获取80个段名前缀，可以看出命名方式就是0~9、a~z的组合值：

图2：



- 步骤二之生成`SegmentInfo`对象：在当前阶段`SegmentInfo`对象中的变量都是初始化的数据，在后面的流程中会不断的更新`SegmentInfo`对象中的信息
  - **SegmentInfo对象是什么：**
    - [索引文件si](#)中包含的索引信息读取到内存后就用`SegmentInfo`对象来描述，反之生成[索引文件si](#)的过程就是将`SegmentInfo`对象中的信息持久化到磁盘，所以`SegmentInfo`对象中的信息如图3所示
- 步骤三之初始化`SegmentInfo`对象中的Diagnostics：如果是通过flush生成一个新的`SegmentInfo`对象，那么会将以下的信息初始化Diagnostics：
  - 初始化的Diagnostics包含的字段：
    - `os`：运行Lucene所在的操作系统，版本号，架构，比如操作系统为Mac OS X，版本号为10.14.3，架构为x86\_64
    - `java`：java的发行商，版本号，JVM的版本号
    - `version`：Lucene的版本号，比如7.5.0
    - `source`：生成当前segment是由什么触发的，flush、commit、merge、addIndexes(facet)
    - `timestamp`：生成当前segment的时间戳
  - 由于是通过merge生成的`SegmentInfo`对象，所以会额外多两个字段：
    - `mergeMaxNumSegments`：该字段在`forceMerge`中会用到，这里不开展解释
    - `mergeFactor`：新段是由mergeFactor个旧段合并生成的
  - Diagnostics在[索引文件si](#)中的位置如下图所示，红框标注：

图3：



- 步骤四之生成SegmentCommitInfo对象：根据SegmentInfo对象生成一个SegmentCommitInfo对象，该对象不展开介绍，已经解释过好多次了，同样的，在这个阶段，SegmentCommitInfo对象中的变量都是初始化，在后面的流程中会更新
- 步骤五之更新OneMerge：在[执行段的合并（一）](#)中我们介绍到，OneMerge在后面的流程中，它包含的变量会逐步更新，在这里OneMerge中的SegmentCommitInfo会被更新，即新段的信息被更新到OneMerge中

## 更新待合并的段集合OneMerge

在介绍该流程之前，我们先讲述下Lucene中两个很重要的类ReadersAndUpdates、ReaderPool。

### ReadersAndUpdates

ReadersAndUpdates用来维护一个段的信息，比如删除信息的更新，段的复用（[NRT近实时搜索](#)）等，查询、合并操作都会用到ReadersAndUpdates，它包含的几个重要的变量如下所示：

- SegmentCommitInfo info：该字段描述了一个段的完整索引信息（除了删除信息），见[近实时搜索NRT（四）](#)
- SegmentReader reader：当需要读取段的索引信息时，我们可以复用该对象，降低读取开销（复用、提高读取性能的概念见文章[SegmentReader（一）](#)）
- PendingDeletes pendingDeletes：上文中我们说到删除信息被分为两类，被删除的文档号和DocValues信息，PendingDeletes对象中包含了一个[FixedBitSet](#)对象来存储被删除的文档号
- boolean isMerging：该字段用来描述当前段是否正在执行合并操作
- Map<String,List<DocValuesFieldUpdates>> pendingDVUpdates：如果当前段中的DocValues信息需要更新，那么DocValues信息用该Map容器存放
- Map<String,List<DocValuesFieldUpdates>> mergingDVUpdates：如果当前段中的DocValues信息需要更新，但是当前段正在更新，那么DocValues信息会先用pendingDVUpdates存放，同时用该Map容器存放

### ReaderPool

ReaderPool中包含了一个容器，其定义如下：

```
private final Map<SegmentCommitInfo,ReadersAndUpdates> readerMap = new
HashMap<>();
```

ReaderPool是IndexWriter的变量，所以ReaderPool的作用是在持有IndexWriter的情况下能通过SegmentCommitInfo找到每一个段的ReadersAndUpdates，故IndexWriter、ReaderPool、ReadersAndUpdates三者的关系如下：

图4:

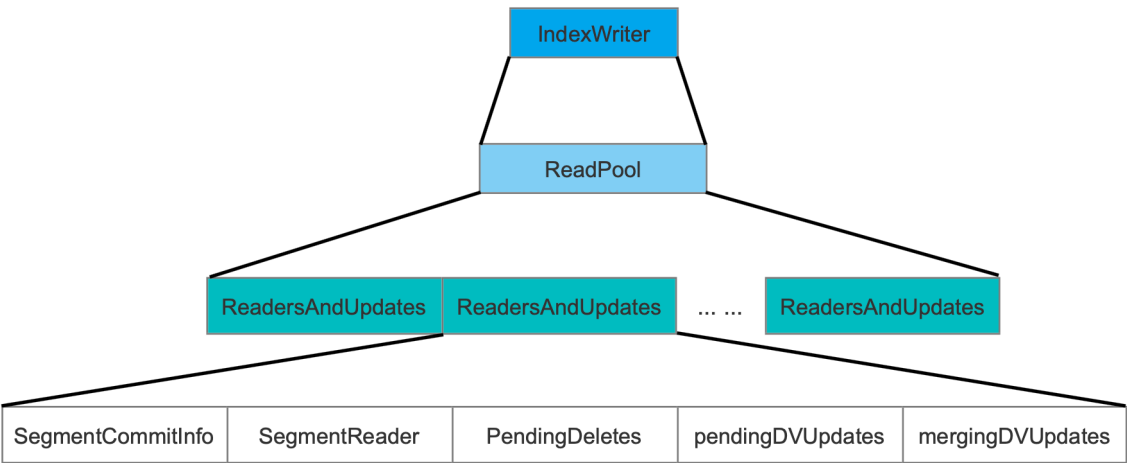


图4中ReadersAndUpdates的个数即当前索引目录中段的个数。

**ReadersAndUpdates在什么时候生成：**

- 作用删除信息：如果当前段需要被作用删除信息，如上文描述的，删除信息会被存储到当前段的ReadersAndUpdates或pendingDVUpdates或mergingDVUpdates中，故如果ReaderPool中没有该段的ReadersAndUpdates，那么就会生成ReadersAndUpdates，生成的时间点在下方的流程图中红框标注：

图5:

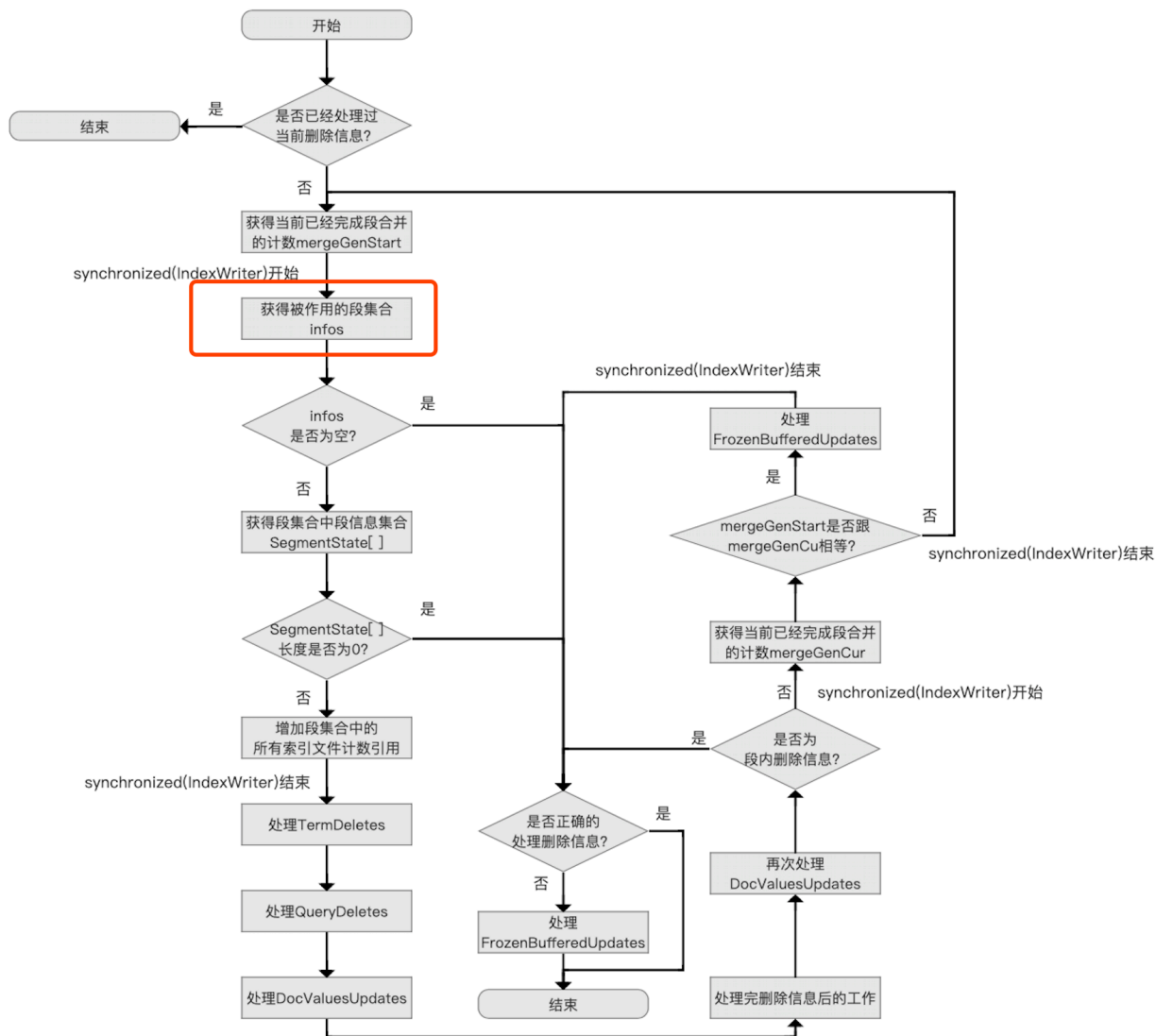


图5的流程图为[文档提交之flush \(七\)](#)中的处理删除信息的流程图。

- 合并阶段：合并期间需要合并删除信息，故同样使用ReadersAndUpdates来获得每一个待合并的段删除信息，如果某个段在ReadPool中没有ReadersAndUpdates对象，那么先生成该ReadersAndUpdates对象

我们回到流程点 **更新待合并的段集合OneMerge**，在当前流程点我们需要更新OneMerge中的两个变量，如下所示，OneMerge中包含的信息见文章[执行段的合并 \(一\)](#)中的介绍：

- List<SegmentReader> readers：readers中的每一个SegmentReader描述的是某个待合并的段的信息，并且SegmentReader是通过ReadersAndUpdates获得的
- List<Bits> hardLiveDocs：hardLiveDocs中的每一个Bits描述的是某个待合并的段中被标记为删除的文档号集合，并且hardLiveDocs是通过[SegmentReader](#)获得的

## 获取SegmentReader的集合MergeReader

该流程会涉及软删除的概念，基于篇幅，将在下一篇文章中展开。

## 结语

[点击下载附件](#)