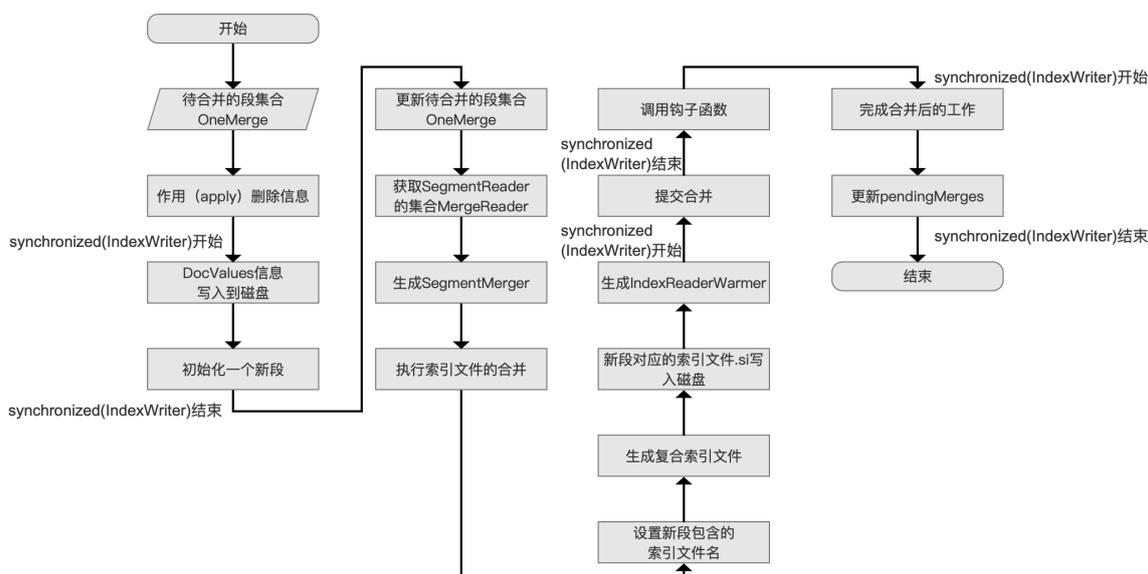


执行段的合并 (四)

本文承接[执行段的合并 \(三\)](#)，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](#)

生成IndexReaderWarmer

在前面流程中我们了解到，合并后生成的新段已经包含了所有固定的索引信息及部分删除信息，故在当前流程点，我们可以生成该段对应的SegmentReader对象，并将该对象添加到ReadPool（见[执行段的合并 \(二\)](#)）中，这便是生成IndexReaderWarmer的过程。

删除信息包括了被删除的文档号跟变更的DocValues信息。

SegmentReader对象中包含的内容在[SegmentReader](#)系列文章中介绍，不赘述

固定的索引信息是哪些：

- [索引文件.nvd、.nvm、.pos、.pay、.doc、.tim、.tip、.dim、.dii、.tvx、.tvd、.fdx、.fdt](#)中包含的信息

为什么是包含了部分删除信息：

- 执行段的合并是Lucene安排一个新的线程执行的并发操作，在合并的过程中，其他执行[文档增删改](#)的线程可能生成了新的删除信息，并且新的删除信息会在随后 **提交合并** 的流程中作用当前的新段

为什么要生成IndexReaderWarmer：

- 首先要说的是，在合并阶段生成IndexReaderWarmer需要通过 [IndexWriterConfig.setMergedSegmentWarmer\(\)](#)方法设置，默认不使用该功能

- 由于执行段的合并是并发操作，使得可以并发的提前读取新段的内容，即获得SegmentReader对象（生成IndexReaderWarmer的主要目的），其他线程执行[近实时搜索NRT](#)时无需等待合并操作结束后再去获得SegmentReader对象，要知道获得SegmentReader对象的过程是I/O操作，故可以降低NRT搜索的延迟

提交合并

在介绍 提交合并 流程前，我们先介绍下MergeState，在[执行段的合并（三）](#)的文章中我们只介绍了该对象的生成时机，即图1的 生成SegmentMerger 流程，由于在当前 提交合并 的流程中将会用到该对象，故才在此流程点展开介绍。

MergeState

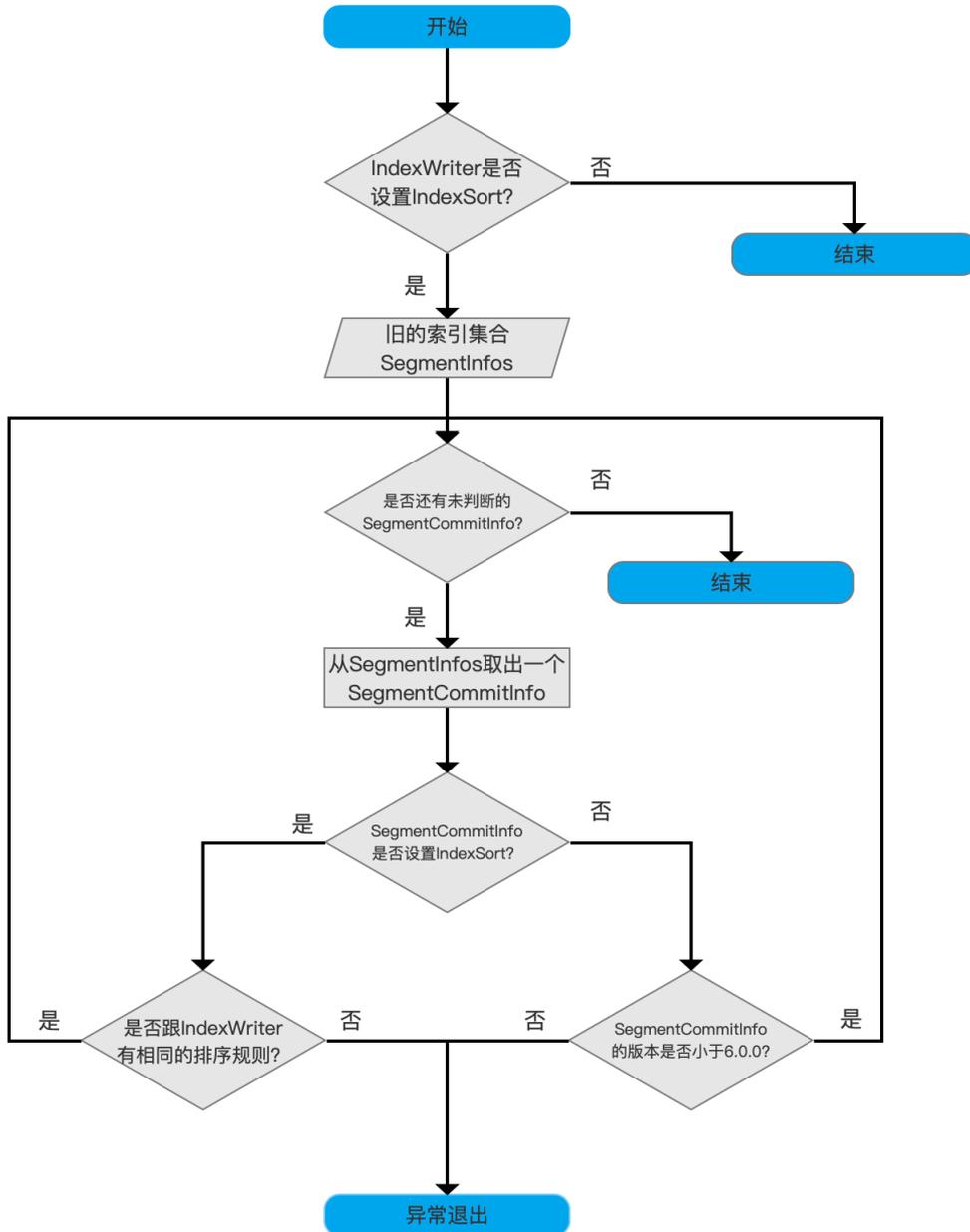
MergeState维护了在段的合并过程中一些共同的状态（common state），在本篇文章中我们只关心在生成MergeState的过程中完成的几个任务，根据IndexWriter是否设置了IndexSort（见文章[索引文件之si](#)中关于IndexSort的介绍）可以将任务划分为如下两类：

- 设置了IndexSort
 - 任务一：对每一个待合并的段进行段内排序
 - 任务二：对合并后的新段进行段内排序
 - 任务三：获得所有待合并的段的被删除的文档号与段内真正的文档号的映射DocMap[]
- 未设置IndexSort
 - 任务三：获得所有待合并的段的被删除的文档号与段内真正的文档号的映射DocMap[]

在介绍每一个任务前，我们首先介绍下在初始化IndexWriter对象的过程中段内排序的非法检查的流程（见源码中的[validateIndexSort\(\)](#)方法），如果通过[IndexWriterConfig.setIndexSort\(Sort\)](#)设置了段内排序，那么每次flush后生成的段，它包含的文档（document）是按照参数Sort排序的，并且如果IndexWriter对象需要读取旧的索引，即不是该IndexWriter对象生成的索引，那么需要检查旧的索引中所有段的段内排序规则，判断过程如下所示：

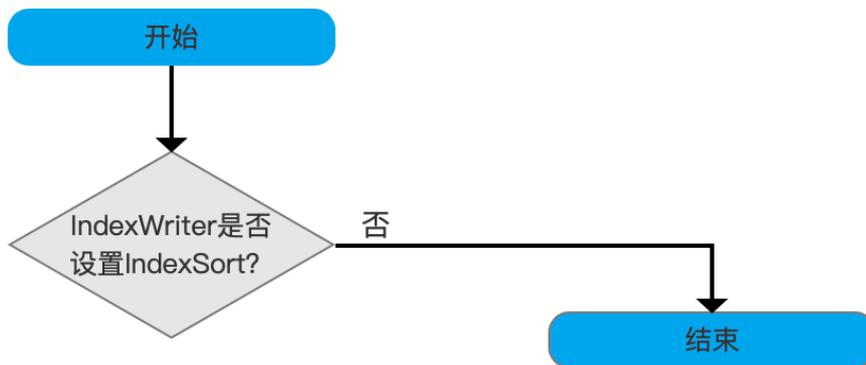
初始化IndexWriter对象的过程中段内排序的非法检查流程图

图2：



IndexWriter没有设置IndexSort

图3:



如果IndexWriter没有设置IndexSort, 那么不需要对旧索引中的段的段内排序规则进行非法检查, 同时在合并阶段只需要执行任务三。

旧的索引集合SegmentInfos

图4:



IndexWriter读取索引目录中的内容时, 默认总是只读取最后一次提交对应的索引信息, 即只读索引文件名segment_N, N值最大的那个, 通过segment_N文件获取到旧的索引集合SegmentInfos, SegmentInfos中包含的一个重要的信息就是一个存放SegmentCommitInfo的链表:

```
private List<SegmentCommitInfo> segments = new ArrayList<>();
```

索引目录中为什么可能会有多个segment_N文件:

- 该内容已在文章[索引文件之segments_N](#)中介绍, 不赘述

如何根据segment_N文件获取到旧的索引集合SegmentInfos:

- 通过索引文件segment_N的数据结构一目了然, 如下所示:

图5:

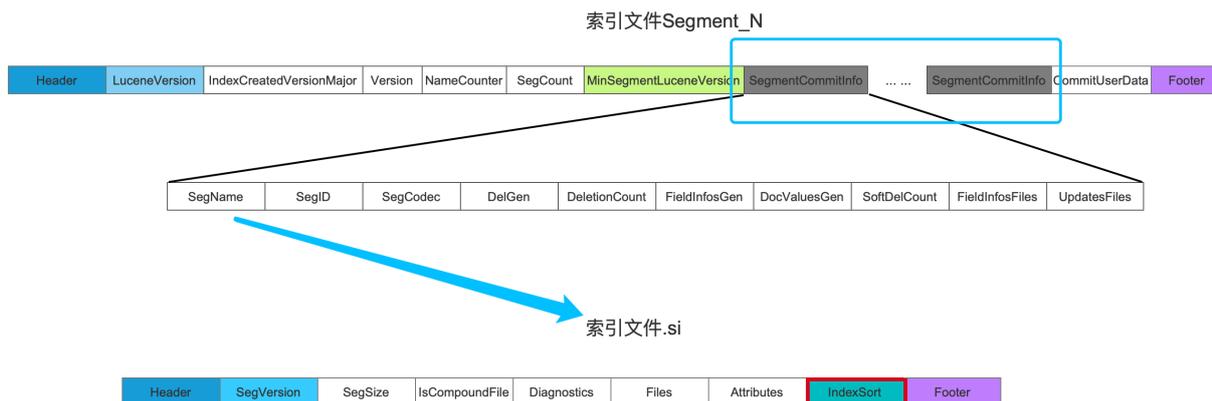
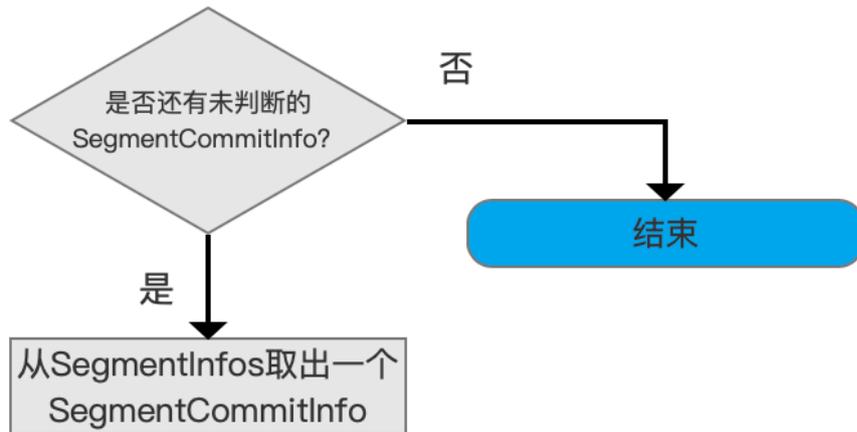


图5中, 根据索引文件segment_N就可以找到这次提交对应的所有段的信息 (除了删除信息), 即蓝色框标注的SegmentCommitInfo。

另外图5中蓝色箭头描述的是, 通过Segname就可以从索引目录中找到对应的索引文件.s, SegName为一个段的段命

依次处理每一个段

图6:



从SegmentInfos中依次取出SegmentCommitInfo，当所有的SegmentCommitInfo都处理结束后退出。

获取一个段的段内排序规则

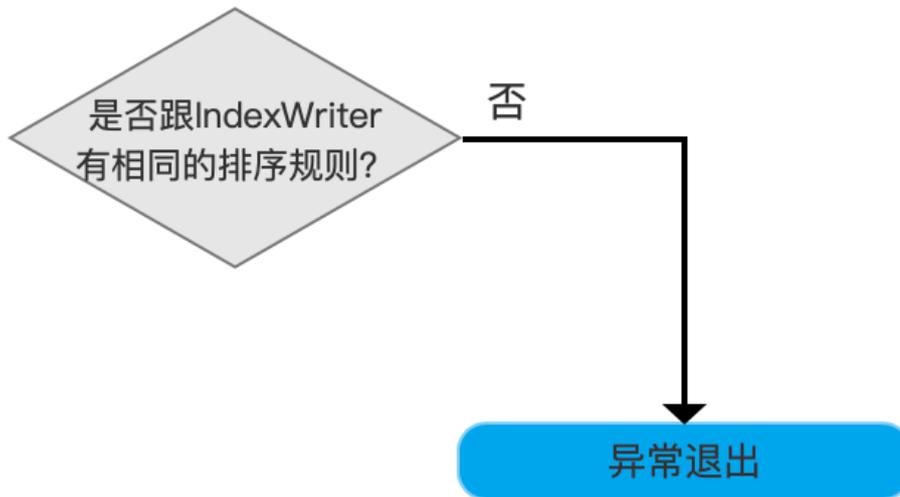
图7:



在图5中，索引文件.si中红框标注的IndexSort就是一个段的段内排序规则，如果该字段不为空，说明该段设置了段内排序规则。

处理设置了段内排序的段

图8:



如果该段的段内排序规则跟IndexWriter设置的不一致，那么无法初始化IndexWriter，并且抛出如下的异常：

```

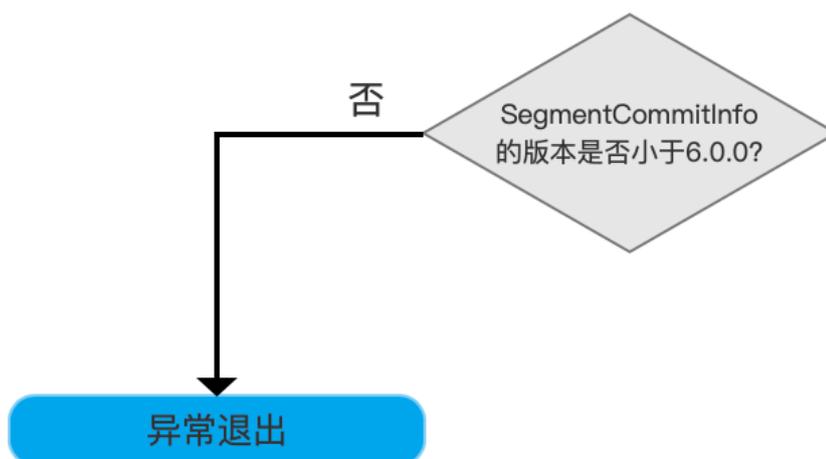
throw new IllegalArgumentException("cannot change previous indexSort=" +
segmentIndexSort + " (from segment=" + info + ") to new indexSort=" +
indexSort);
  
```

上述的异常中， segmentIndexSort是段的段内排序规则， IndexSort是IndexWriter设置的排序规则， info是该段对应的SegmentCommitInfo。

处理旧索引中的任意一个段时发生异常退出都会导致IndexWriter无法初始化。

处理未设置段内排序的段

图9：



如果一个段未设置段内排序，并且生成该段的Lucene版本号大于等于6.0.0，那么无法初始化IndexWriter，并且抛出如下的异常：

```

throw new CorruptIndexException("segment not sorted with indexSort=" +
segmentIndexSort, info.info.toString());
  
```

上述的异常中，segmentIndexSort是段的段内排序规则，info.info描述的是该段对应的索引文件.si信息。

从图8、图9可以看出，设置了段内排序的IndexWriter只能处理下面两种情况的旧的索引：

- 与IndexWriter有相同段内排序规则
- 未设置段内排序，并且版本号小于6.0.0

如何读取未设置段内排序，并且版本号大于等于6.0.0的旧索引

- 可以通过[IndexWriter.addIndexes\(CodecReader... readers\)](#)方法实现，该方法的参数readers为旧的索引的句柄。

调用IndexWriter.addIndexes(CodecReader... readers)方法的过程实际是将旧的索引进行合并，将新生成的段添加到IndexWriter中，该方法中的合并过程也会生成MergeState，**并且只有这种情况下以及处理版本号小于6.0.0的旧索引才会执行上文中提到的任务一**，而这正是先介绍初始化IndexWriter对象的过程中段内排序的非法检查流程的原因😞。

结语

生成MergeState过程中的三个任务的内容将在下一篇文章中展开。

[点击](#)下载附件