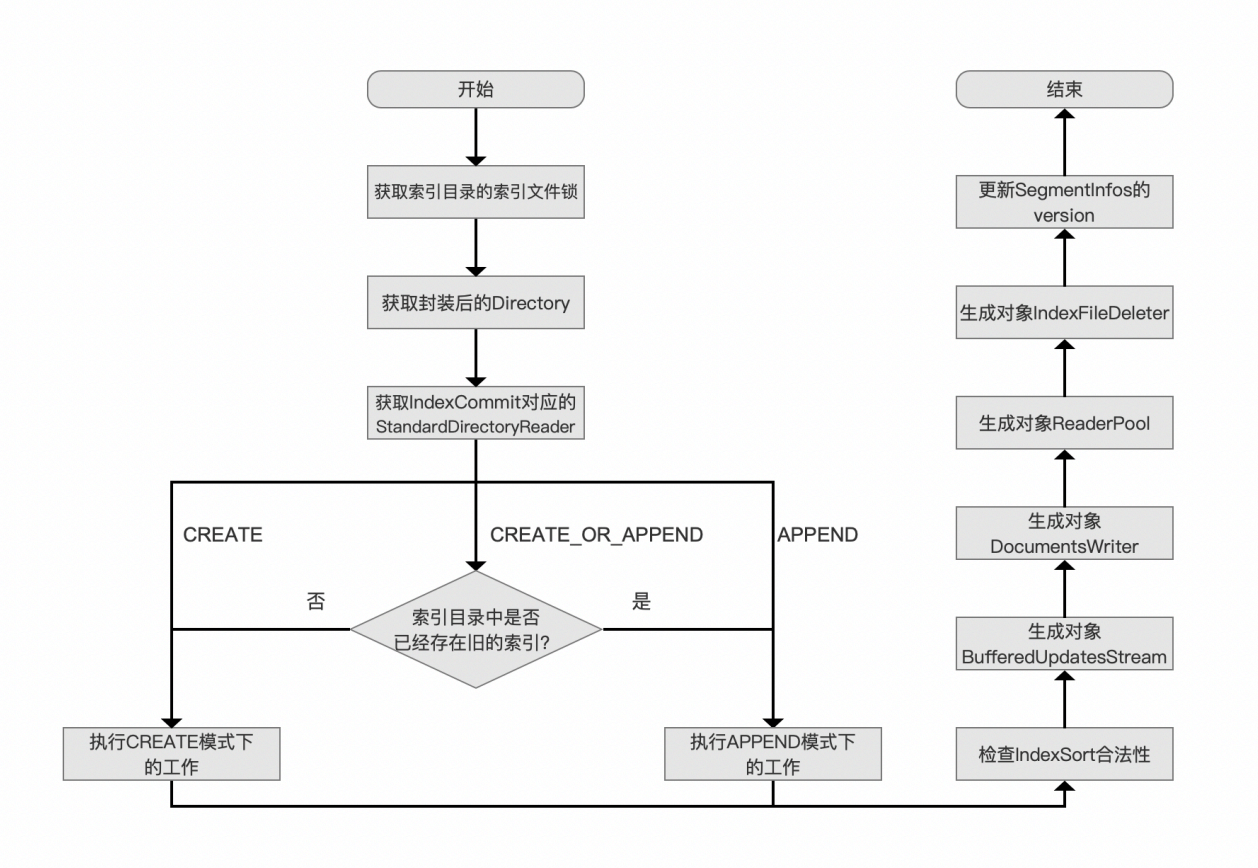


构造IndexWriter对象 (八)

本文承接[构造IndexWriter对象 \(七\)](#)，继续介绍调用IndexWriter的构造函数的流程。

调用IndexWriter的构造函数的流程图

图1：

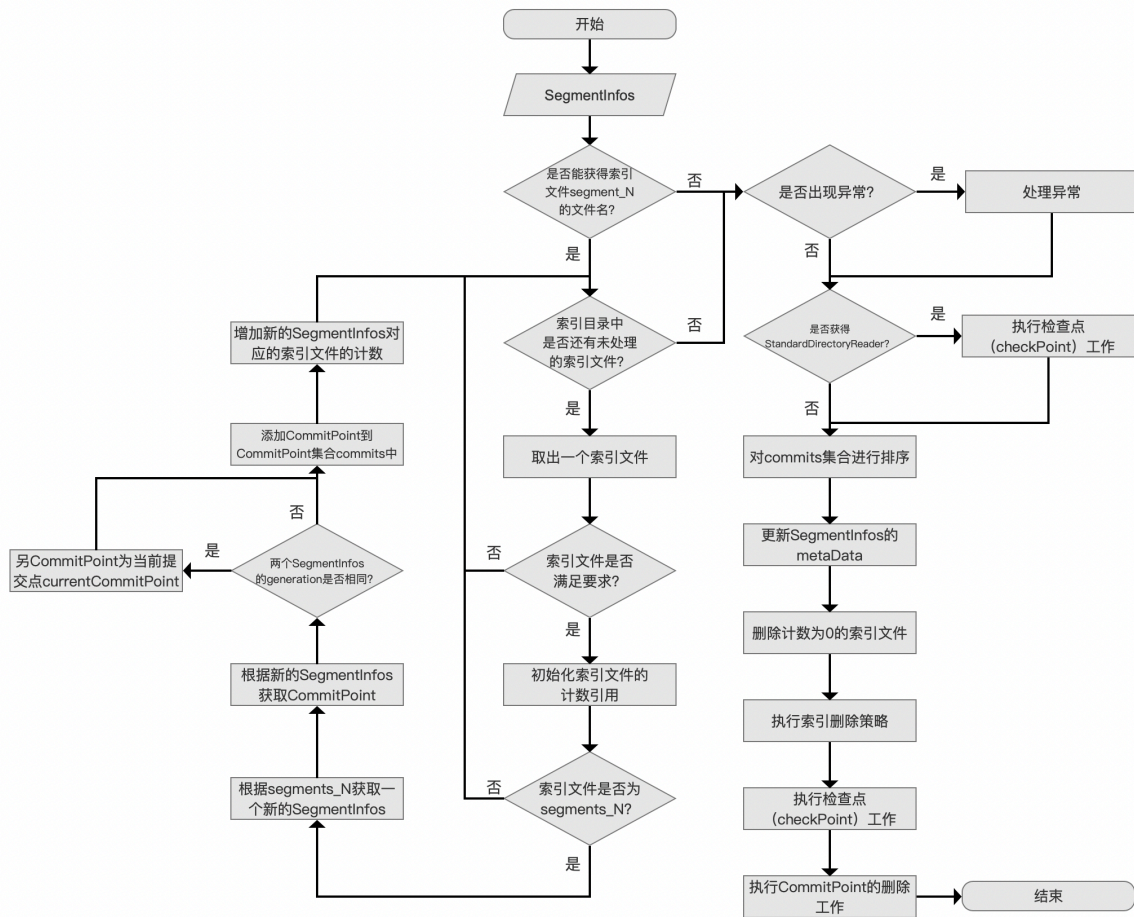


生成对象IndexFileDeleter

我们紧接上一篇文章，继续介绍剩余的流程点，下面先给出IndexFileDeleter的构造函数流程图：

IndexFileDeleter的构造函数流程图

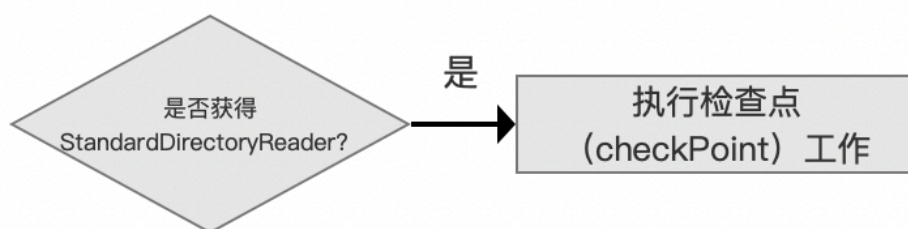
图2：



[点击查看大图](#)

执行检查点（checkPoint）工作

图3：



在介绍当前流程点之前，我们先通过下面的流程图来介绍 执行检查点的工作 这个流程点做了哪些事情。

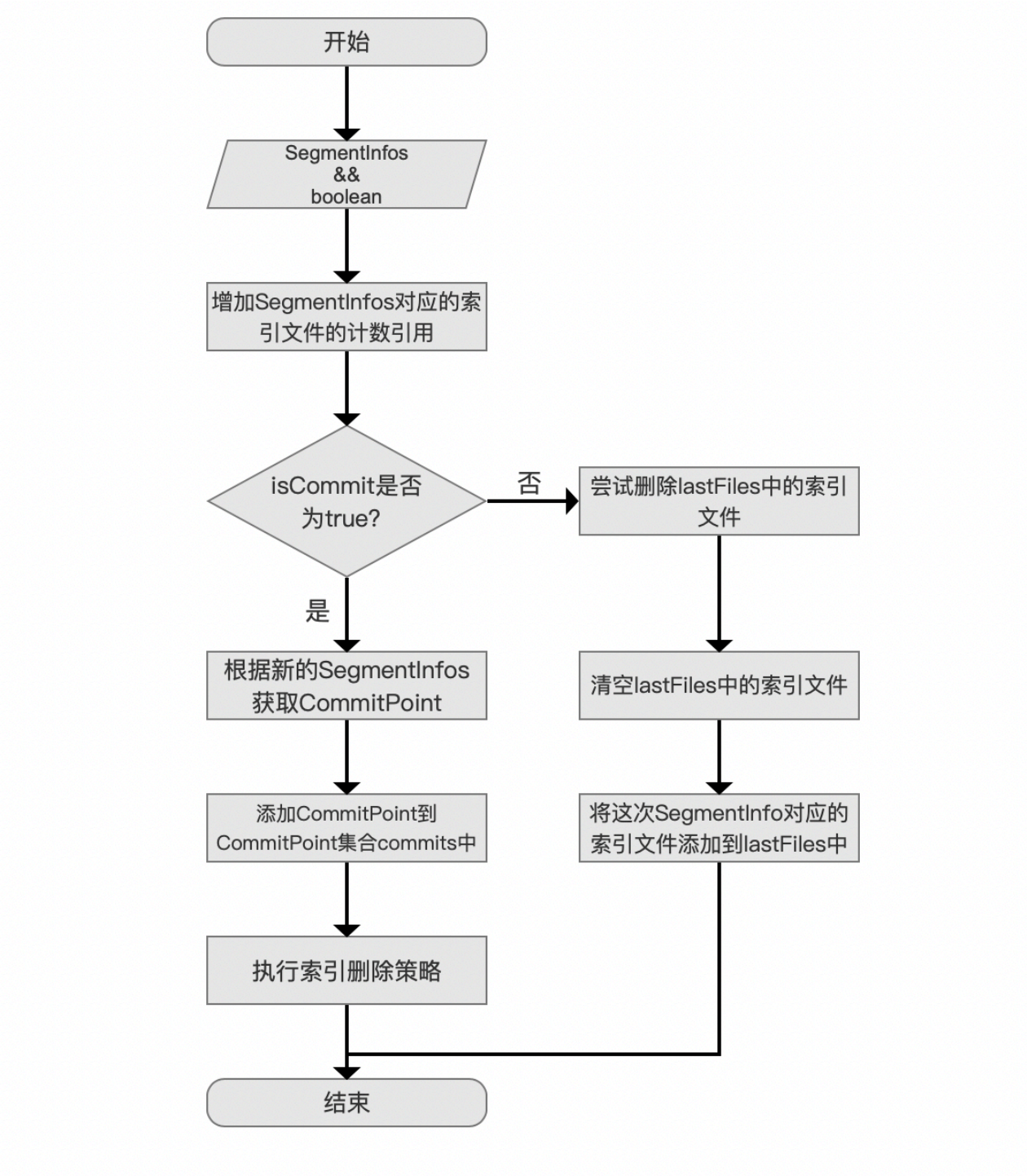
另外 执行检查点（checkPoint）工作 在源码中对应的方法方法定义：

```

1 public void checkpoint(SegmentInfos segmentInfos, boolean isCommit) throws
  IOException {
2     ... ..
3 }
  
```

执行检查点 (checkPoint) 工作的流程图

图4:



准备数据

图5:

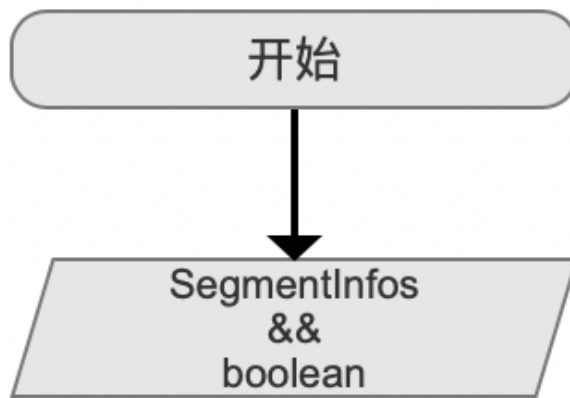
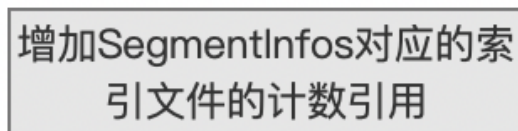


图4流程图的准备数据就是checkPoint方法的两个参数SegmentInfos以及boolean。

增加SegmentInfos对应的索引文件的计数引用

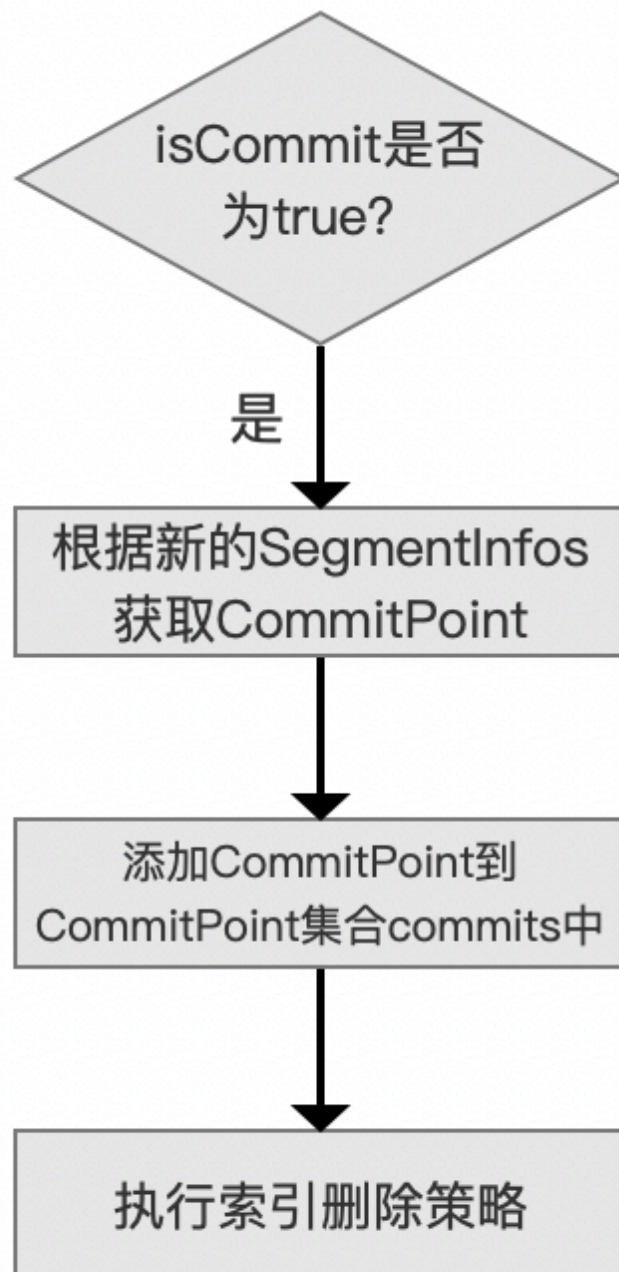
图6:



在后面的流程中，可能会执行索引文件的删除，如果某些索引文件被SegmentInfos引用，那么这些索引文件不应该被删除，防止被删除的方法就是增加SegmentInfos对应的索引文件的计数引用。

当isCommit为true时的处理流程

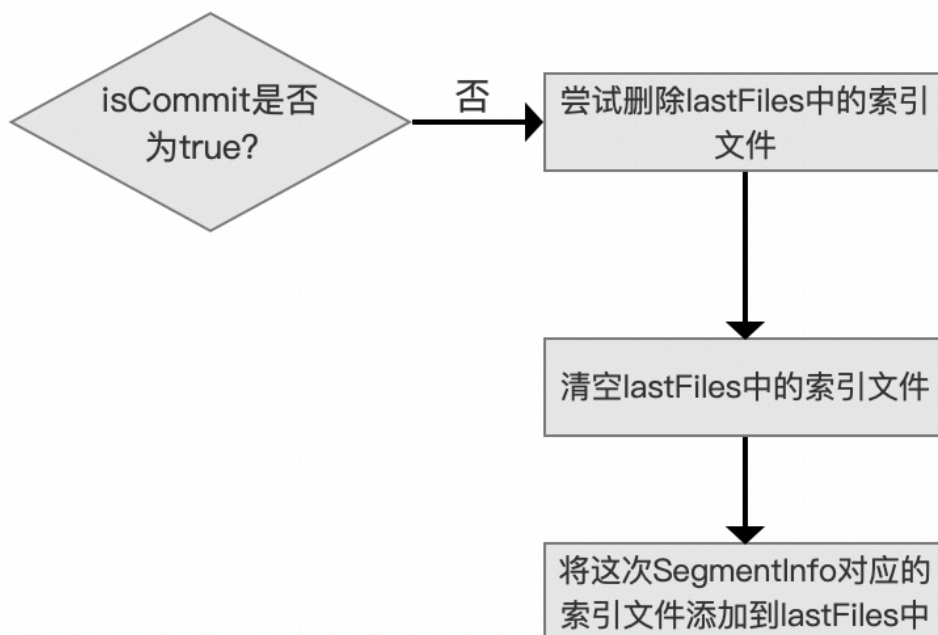
图7:



在执行commit()操作时，也会执行checkPoint的操作，那么此时的isCommit是位true的，在文章[文档提交之commit \(二\)](#)中介绍了这个流程，不赘述。

当isCommit为false时的处理流程

图8：



lastFiles是一个IndexFileDeleter类的成员变量，它用来存放上次执行checkPoint的SegmentInfos中对应的索引文件定义如下：

```
1 private final List<String> lastFiles = new ArrayList<>();
```

所以当isCommit为false时，先尝试删除lastFiles中的索引文件，删除的方式就是减少每一个索引文件的计数引用，如果计数值为0，那么该索引文件就要被删除，否则不删除，最后清空lastFiles中的索引文件后，将这次SegmentInfos对应的索引文件添加到lastFiles中。

结合图6跟图8我们可以发现这种流程的逻辑设计可以使得，即使对同一个SegmentInfos对象执行多次checkPoint的操作时，如果该对象中的段没有发生变化，那么段对应索引文件的计数是不会变的（先增加计数，再减少计数），同样地，如果SegmentInfos中如果增加了段，能通过图6的流程点对该段中的索引文件执行正确的+1计数，如果删除了某个段，能通过图8的流程点 尝试删除lastFiles中的索引文件 对该段中的索引文件执行正确的-1计数。

为什么要通过checkPoint来实现索引文件的删除：

Lucene通过IndexWriter对象中的成员变量SegmentInfos来描述当前IndexWriter对应的索引信息，索引信息的变化通过SegmentInfos对象来反应，但是SegmentInfos对象并不真正的拥有索引文件的数据，它拥有只是这些索引文件的文件名，所以当SegmentInfos对应的信息发生变化时，例如某个段中包含的所有文档满足某个删除操作，该段的信息会从SegmentInfos中移除（段的信息即SegmentCommitInfo，见文章[构造IndexWriter对象（四）](#)关于流程点 获得回滚（rollback）信息的介绍），那么这个段对应的索引文件也应该要删除（如果索引文件的计数引用为0），当然如果其他段同时引用这些索引文件，那么就不会被删除（索引文件的计数引用不为0），也就是为什么图7的流程点 尝试删除lastFiles中的索引文件 为什么要带有 尝试 两个字。

我们回到当前流程点，介绍下为什么获得StandardDirectoryReader后需要执行 执行检查点（checkPoint）工作：

根据图2的流程，我们是在构造IndexFileDeleter对象的流程中第一次调用checkPoint()方法，那么lastFiles中必定不包含任何的数据，并且在源码中调用checkPoint()方法的参数如下所示：

```
1 checkpoint(segmentInfos, false);
```

segmentInfos即StandardDirectoryReader中对应的索引信息（见文章[构造IndexWriter对象_\(四\)_](#)中用StandardDirectoryReader初始化一个新的SegmentInfos对象 流程点的介绍），同时isCommit的值为false，也就说在当前流程点调用checkPoint()方法的目的就是仅仅为了增加segmentInfos对应的索引文件的计数，那么就转变为如下的问题：

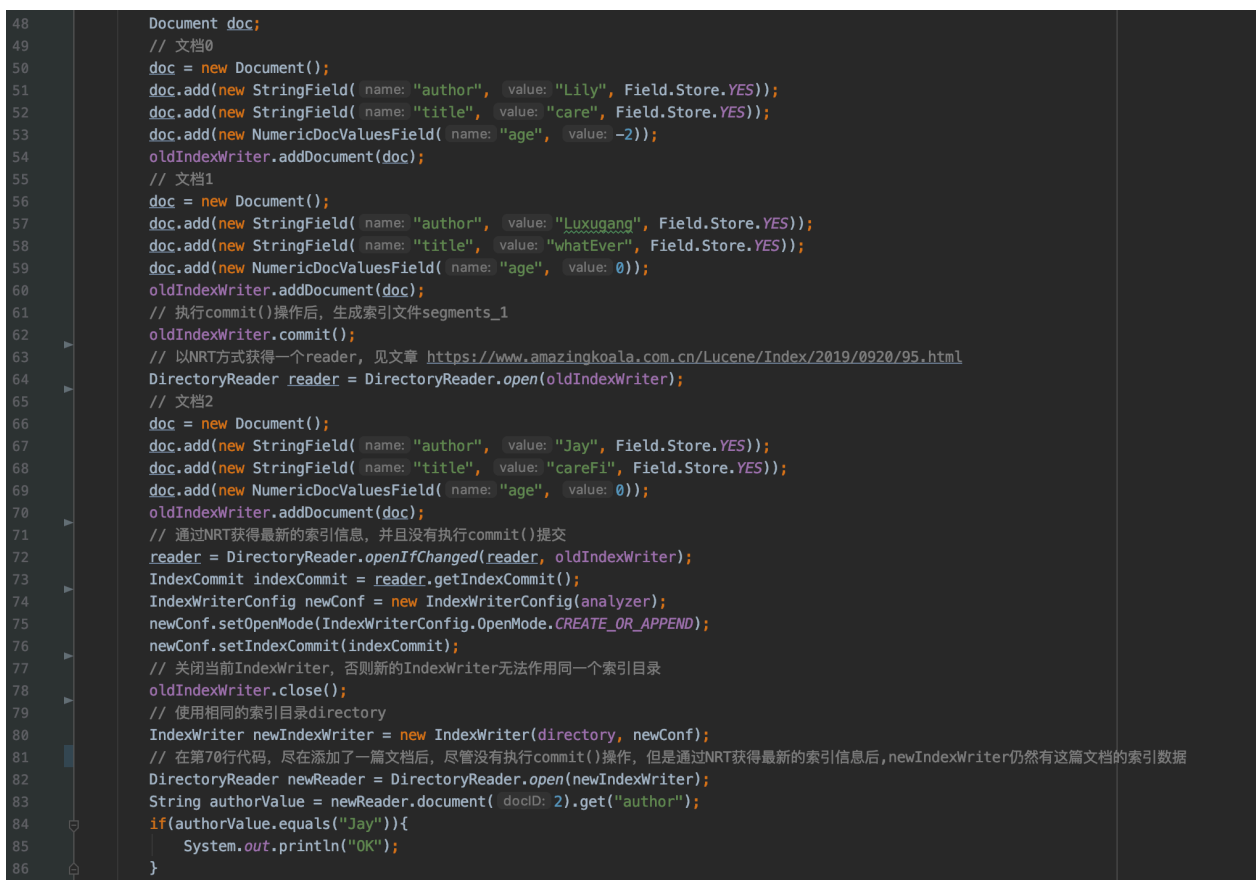
为什么获得StandardDirectoryReader后，需要增加segmentInfos对应的索引文件的计数：

首先给出源码中的解释：

```
1 // Incoming SegmentInfos may have NRT changes not yet visible in the latest
   commit, so we have to protect its files from deletion too:
```

我们用一个例子来介绍上文的注释所描述的场景，完整代码见：<https://github.com/LuXugang/Lucene-7.5.0/blob/master/LuceneDemo/src/main/java/lucene/index/CheckPointInIndexFileDeleter.java>。

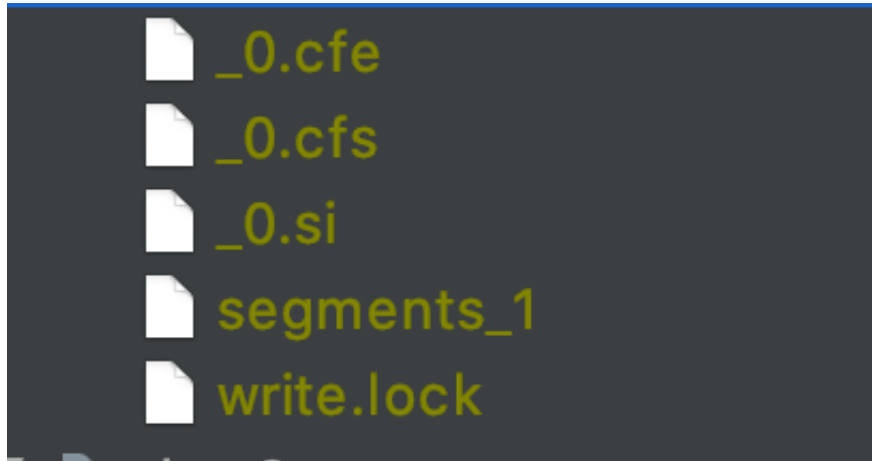
图9：



```
48 Document doc;
49 // 文档0
50 doc = new Document();
51 doc.add(new StringField( name: "author", value: "Lily", Field.Store.YES));
52 doc.add(new StringField( name: "title", value: "care", Field.Store.YES));
53 doc.add(new NumericDocValuesField( name: "age", value: -2));
54 oldIndexWriter.addDocument(doc);
55 // 文档1
56 doc = new Document();
57 doc.add(new StringField( name: "author", value: "Luxugang", Field.Store.YES));
58 doc.add(new StringField( name: "title", value: "whatEver", Field.Store.YES));
59 doc.add(new NumericDocValuesField( name: "age", value: 0));
60 oldIndexWriter.addDocument(doc);
61 // 执行commit()操作后，生成索引文件segments_1
62 oldIndexWriter.commit();
63 // 以NRT方式获得一个reader，见文章 https://www.amazingkoala.com.cn/Lucene/Index/2019/0920/95.html
64 DirectoryReader reader = DirectoryReader.open(oldIndexWriter);
65 // 文档2
66 doc = new Document();
67 doc.add(new StringField( name: "author", value: "Jay", Field.Store.YES));
68 doc.add(new StringField( name: "title", value: "careFi", Field.Store.YES));
69 doc.add(new NumericDocValuesField( name: "age", value: 0));
70 oldIndexWriter.addDocument(doc);
71 // 通过NRT获得最新的索引信息，并且没有执行commit()提交
72 reader = DirectoryReader.openIfChanged(reader, oldIndexWriter);
73 IndexCommit indexCommit = reader.getIndexCommit();
74 IndexWriterConfig newConf = new IndexWriterConfig(analyzer);
75 newConf.setOpenMode(IndexWriterConfig.OpenMode.CREATE_OR_APPEND);
76 newConf.setIndexCommit(indexCommit);
77 // 关闭当前IndexWriter，否则新的IndexWriter无法作用同一个索引目录
78 oldIndexWriter.close();
79 // 使用相同的索引目录directory
80 IndexWriter newIndexWriter = new IndexWriter(directory, newConf);
81 // 在第70行代码，尽在添加了一篇文档后，尽管没有执行commit()操作，但是通过NRT获得最新的索引信息后，newIndexWriter仍然有这篇文档的索引数据
82 DirectoryReader newReader = DirectoryReader.open(newIndexWriter);
83 String authorValue = newReader.document( docID: 2).get("author");
84 if(authorValue.equals("Jay")){
85     System.out.println("OK");
86 }
```

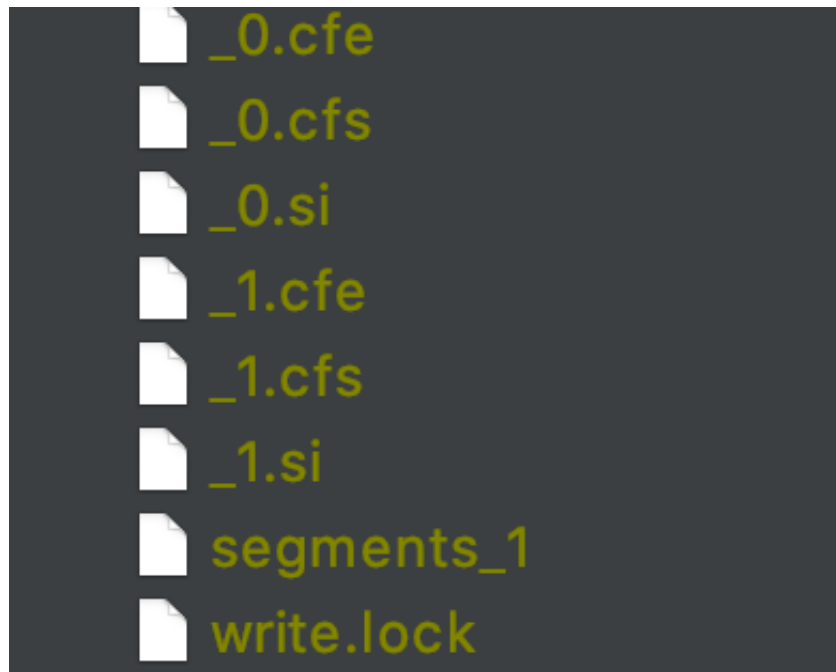
图9中当第62行的oldIndexWriter.commit()操作执行结束后，索引目录中的索引文件如下所示：

图10：



接着执行完第64行的代码后，我们获得了一个NRT的Reader（见文章[近实时搜索NRT（三）](#)的介绍），接着第70行代码结束后，oldIndexWriter新增了一篇文档2，注意的是并没有执行commit()操作（即没有生成新的segments_N文件），随后通过第72行的openIfChange()方法获得一个包含最新索引信息的reader（即StandardDirectoryReader），通过该reader获得一个IndexCommit，IndexCommit中包含了第70行代码增加的索引信息，即图11中以_1为前缀的索引文件，并且在第76行通过IndexWriterConfig.setIndexommit()方法将IndexCommit成为newIndexWriter的配置，在执行完第78行代码后，索引目录中的索引文件如下所示：

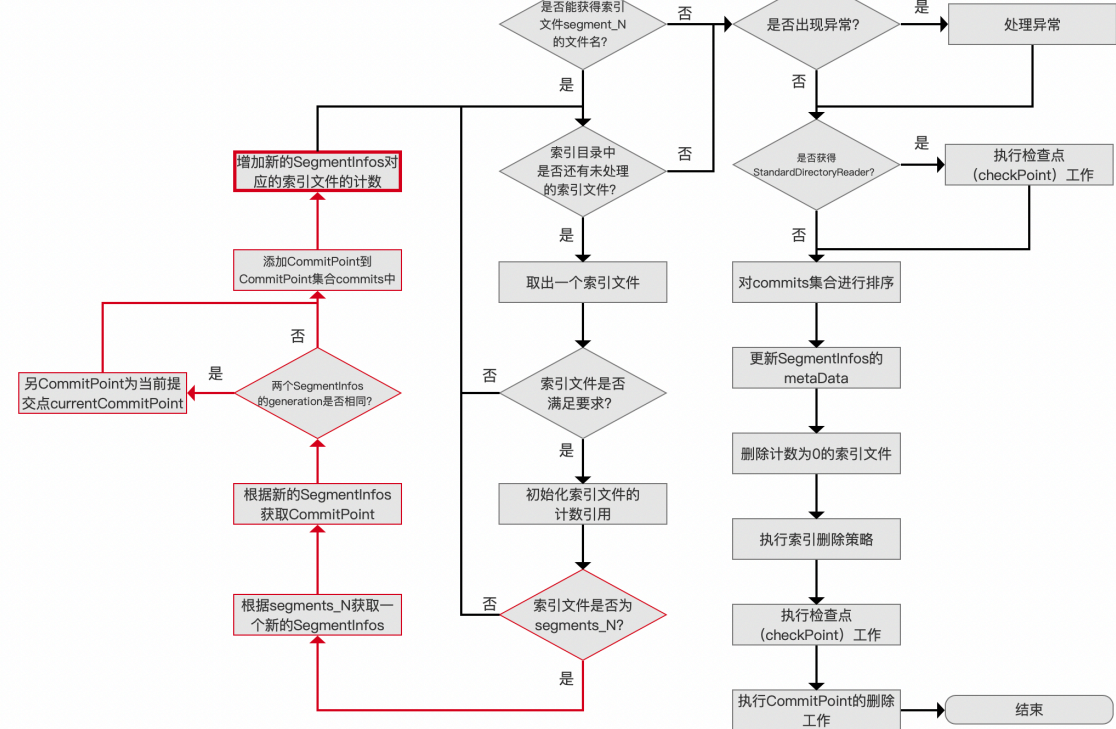
图11：



从图11可以看出，以_1为前缀的索引文件无法被最后的一次commit()可见，即Incoming SegmentInfos may have NRT changes not yet visible in the latest commit，那么在下面的用红色标注的流程中，就无法通过segments_N文件来增加_1.cfe、_1.cfs、_1.si的计数，这些索引文件就是通过NRT changes获得的，在后面的流程中，我们将会知道计数为0的文件都会被删除：

图12：

IndexFileDeleter的构造函数流程图



对commits集合进行排序

图13：



在文章[构造IndexWriter对象（七）](#)中，根据索引目录中的segments_N文件，将对应的segments_N对应的SegmentInfos生成CommitPoint，并且添加到CommitPoint集合commits中，添加的过程是无序的，如果构造中的IndexWriter对象使用的是默认的索引删除策略[KeepOnlyLastCommitDeletionPolicy](#)，那么就无法正确的处理了，所以需要按照从旧到新的提交顺序来排序。

结语

基于篇幅，剩余的内容将在下一篇文章中展开。

[点击](#)下载附件