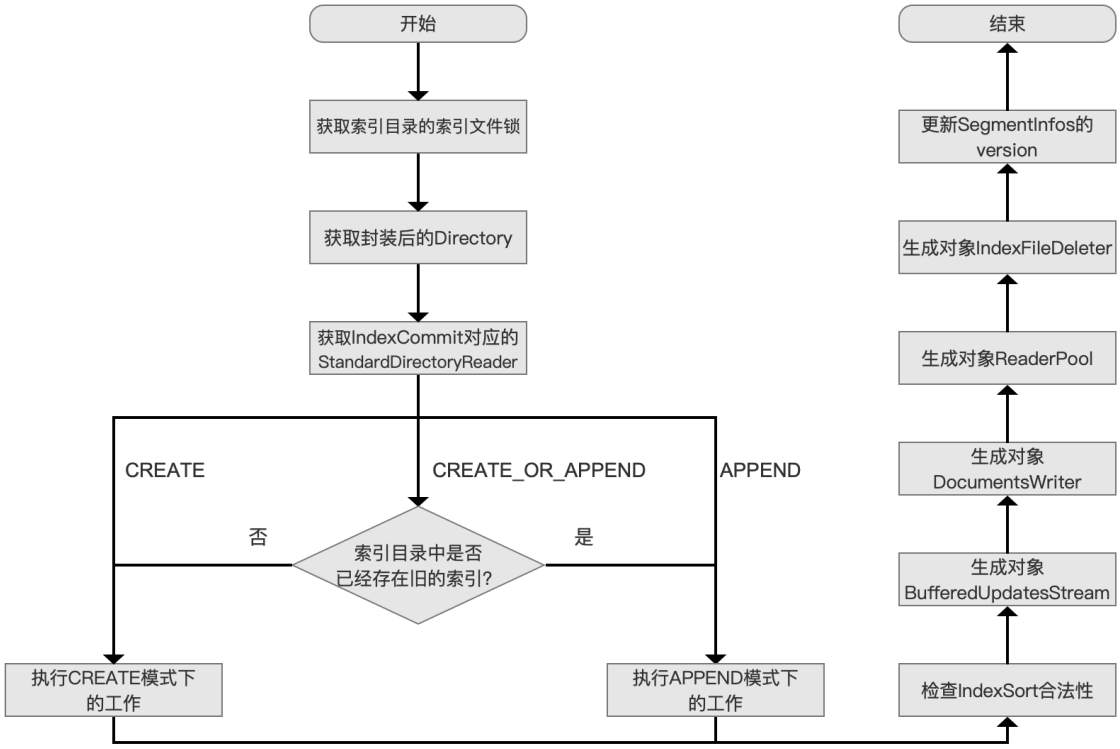
构造IndexWriter对象（四）

本文承接构造IndexWriter对象（三），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：

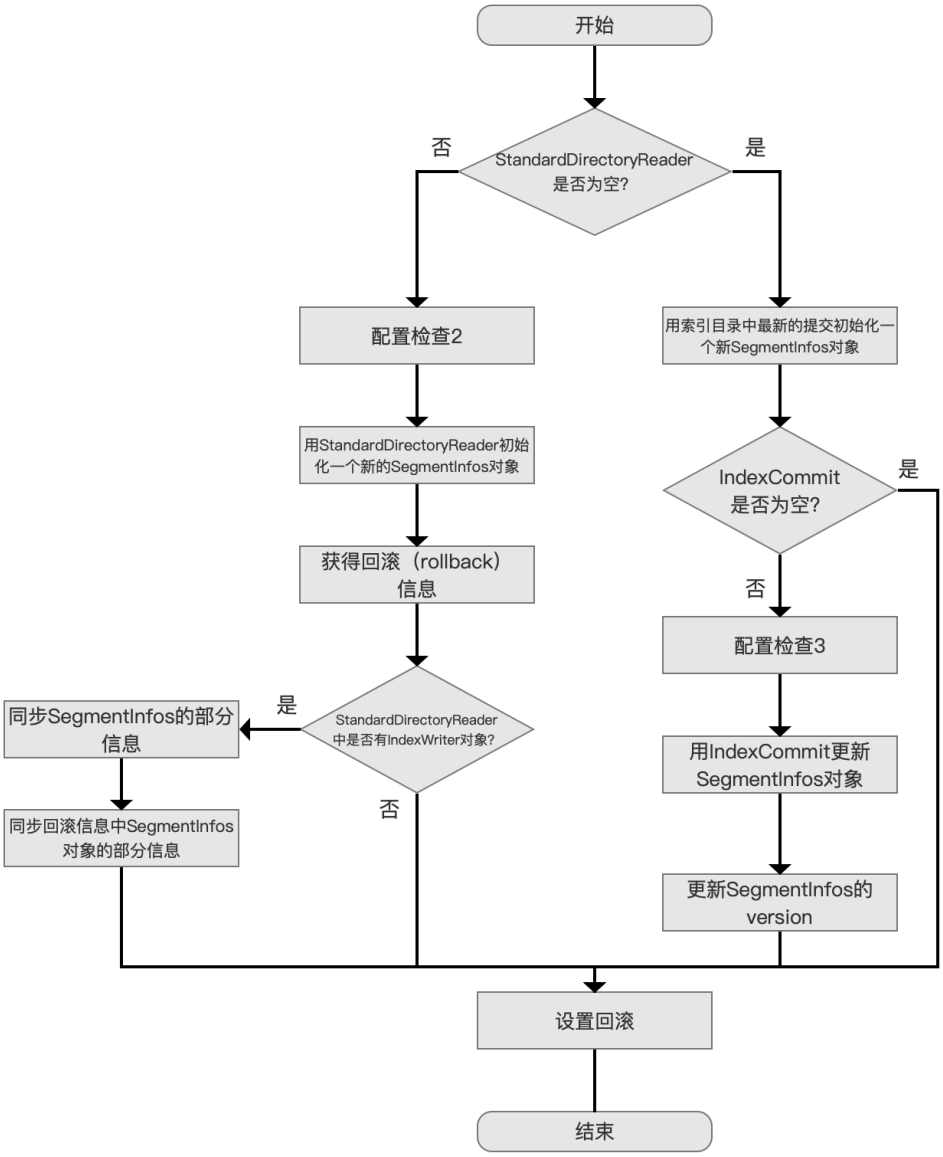


根据不同的OpenMode执行对应的工作

在上一篇文章中，我们介绍了执行CREATE模式下的工作，故继续介绍执行APPEND模式下的工作。

执行APPEND模式下的工作的流程图

图2：



StandardDirectoryReader是否为空?

图3：

在图1的流程点`获取IndexCommit对应的StandardDirectoryReader`，如果用户通过IndexWriterConfig.setIndexCommit(IndexCommit commit)设置了IndexCommit，那么Lucene会尝试根据该IndexCommit获得一个StandardDirectoryReader，它描述了IndexCommit中包含的索引信息（主要是SegmentInfos对象的信息，见构造IndexWriter对象（三）关于SegmentInfos对象的介绍）。

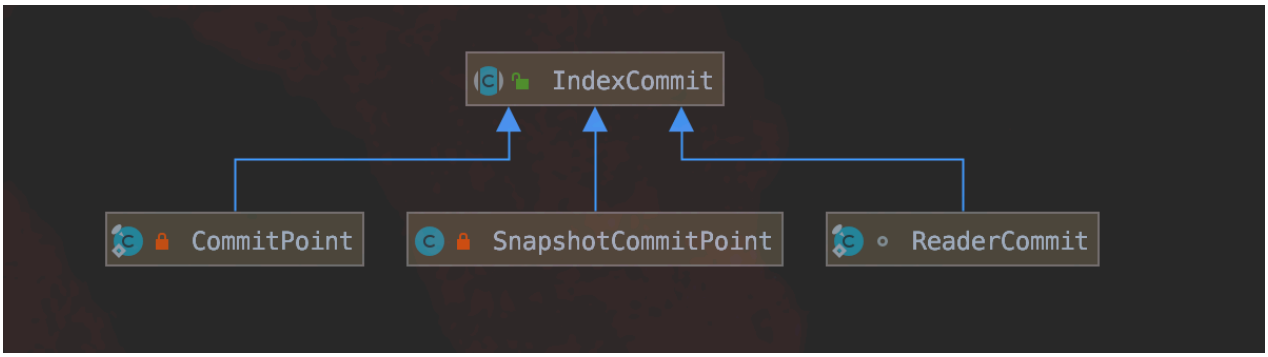
StandardDirectoryReader为空的情况有两种：

- 用户没有设置IndexCommit
- 用户设置了IndexCommit，但是IndexCommit中没有StandardDirectoryReader对象的信息

为什么IndexCommit中可能会没有StandardDirectoryReader对象的信息：

IndexCommit类其实是一个抽象类，它的类图关系如下所示：

图4：



ReaderCommit、CommitPoint

ReaderCommit类跟CommitPoint类中包含的内容如下图所示

图5：

```
static final class ReaderCommit extends IndexCommit {
  private String segmentsFileName;
  Collection<String> files;
  Directory dir;
  long generation;
  final Map<String,String> userData;
  private final int segmentCount;
  private final StandardDirectoryReader reader;
```

```
final private static class CommitPoint extends IndexCommit {

  Collection<String> files;
  String segmentsFileName;
  boolean deleted;
  Directory directoryOrig;
  Collection<CommitPoint> commitsToDelete;
  long generation;
  final Map<String,String> userData;
  private final int segmentCount;
```

故如果是ReaderCommit对象，那么就可以获得StandardDirectoryReader对象，而ReaderCommit对象则是通过StandardDirectoryReader.getIndexCommit()方法获得，由于该方法的实现很简单，故直接给出：

图6：

```
public IndexCommit getIndexCommit() throws IOException {
  ensureOpen();
  return new ReaderCommit( reader: this, segmentInfos, directory);
}
```

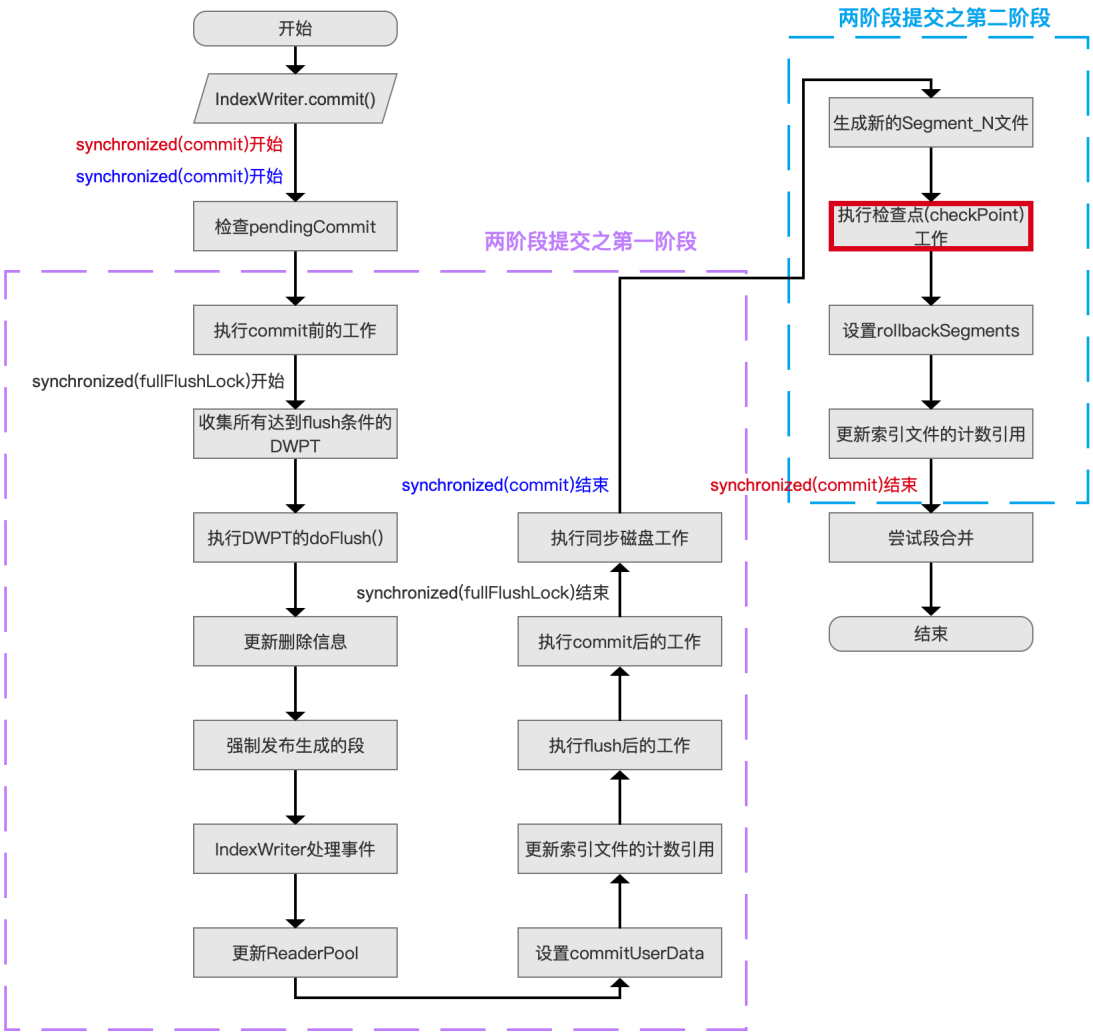
图6为StandardDirectoryReader类中的方法，可以看出，StandardDirectoryReader对象通过调用getIndexCommit()方法，构造了一个新的ReaderCommit对象，并且将自己（this指针）作为ReaderCommit的成员变量之一，即图5中红框标注。

尽管无法通过CommitPoint对象中获得StandardDirectoryReader对象，但这里仍然要说下CommitPoint对象是什么生成的。

CommitPoint对象生成点有两处：

- 生成IndexFileDeleter对象期间：这个时机点即图1中的流程点`生成对IndexFileDeleter`，故我们这里先暂时不展开
- 执行commit()操作期间：图7是执行IndexWriter.commit()的两阶段提交的流程图，并且红框标注的流程点为CommitPoint对象生成的时机，该流程点的具体介绍可以看文档提交之commit（二）

图7：



SnapshotCommitPoint

SnapshotCommitPoint实际是采用装饰者模式来实现额外的功能，该类中的成员变量很少，故直接给出：

图8：

```
/** Wraps a provided {@link IndexCommit} and prevents it
 *  from being deleted. */
private class SnapshotCommitPoint extends IndexCommit {
```

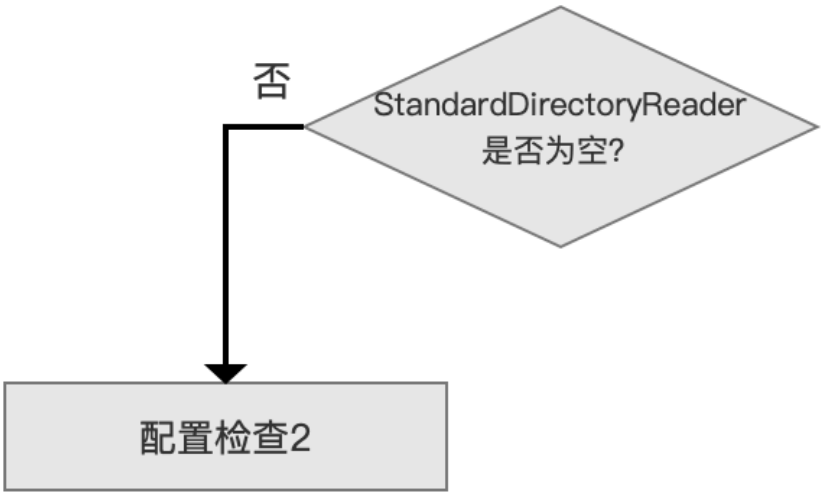
根据图8可以看出，它封装了另一个IndexCommit对象（见图7的红色标注的流程在文章文档提交之commit（二）中的介绍），实际上该对象就是 CommitPoint。

至于它实现了哪些额外的功能，在后面的流程中会展开介绍，这里我们只需要知道它同CommitPoint一样，无法提供StandardDirectoryReader对象。

由于SnapshotCommitPoint封装了CommitPoint对象，所以它的生成时机同样在图7中的红色标注流程点。

配置检查2

图9：



我们先介绍下能获得StandardDirectoryReader对象的情况，需要对用户的配置信息进行三项检查，如下所示

图10：

```
if (reader.directory() != commit.getDirectory()) {
  throw new IllegalArgumentException("IndexCommit's reader must have the same directory as the IndexCommit");
}
```

图10中，reader即StandardDirectoryReader对象、commit即IndexCommit对象、这里没什么好介绍的。

用StandardDirectoryReader初始化一个新的SegmentInfos对象

图11：

在文章构造IndexWriter对象（三）中我们介绍`执行CREATE模式下的工作`时说到，该模式下初始化一个新的SegmentInfos对象时，它不包含任何的索引信息，而在APPEND模式下，则是用StandardDirectoryReader中的索引信息来初始化一个新的SegmentInfos对象，即所谓的"追加"。

获得回滚（rollback）信息

图12：

上文中根据IndexCommit获得的StandardDirectoryReader，它包含的SegmentInfos在后面的流程中将会作为回滚内容，而在这个流程中，最重要的一步是检查SegmentInfos中包含的索引信息对应的索引文件是否还在索引目录中。

为什么要执行这个检查：

我们首先根据下面两张图来介绍下SegmentInfos对象跟索引文件segments_N、索引文件.si以及其他索引文件的关系：

图14：

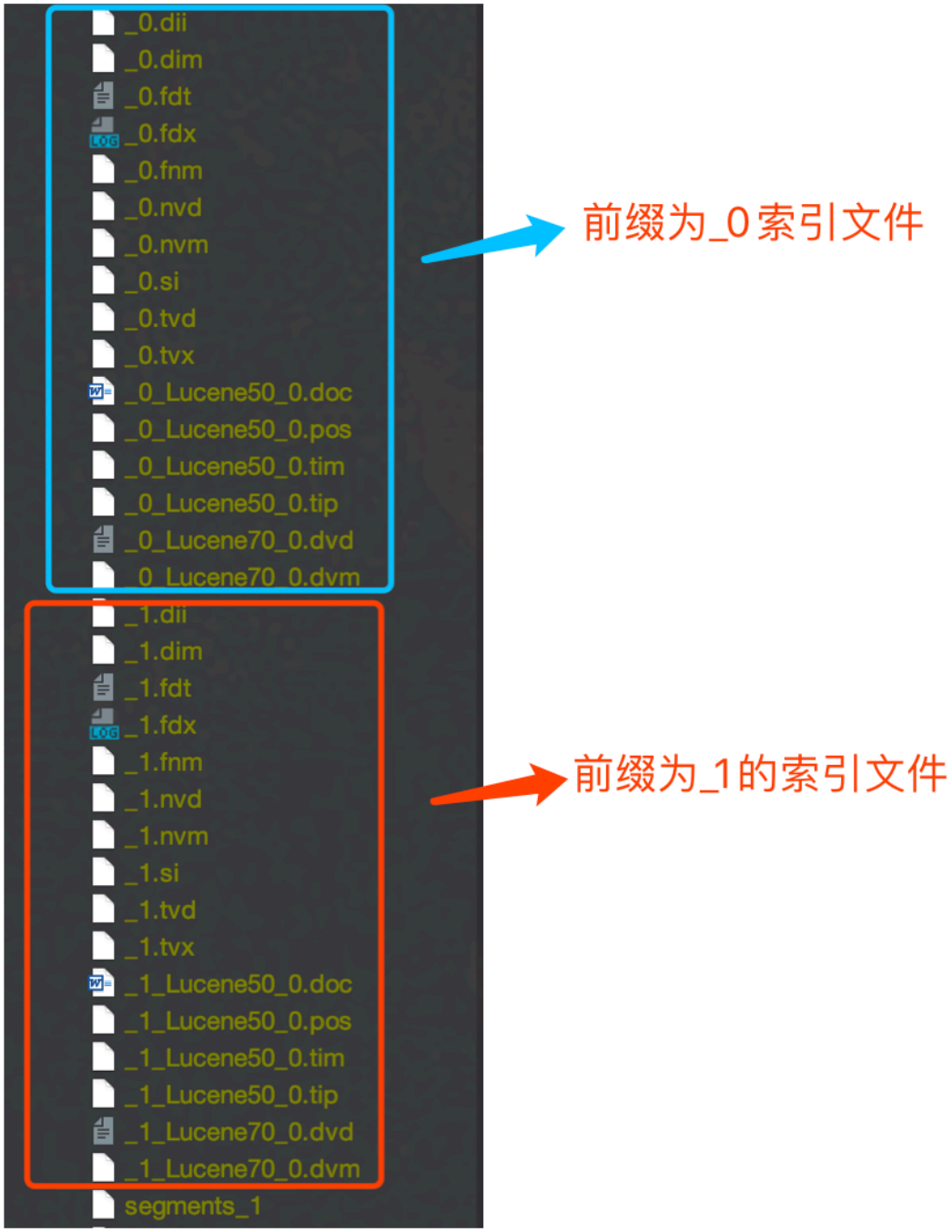


图14为索引目录中的索引文件，并且存在两个段。

图15：

SegmentInfos对象是索引文件segments_N和索引文件.si在内存中的表示，图14中的提交中包含了两个段，即以_0跟_1开头的两个段，所以索引文件segments_1中有两个SegmentCommitInfo字段，接着根据SegmentCommitInfo中的SegName字段，该字段的值描述的是该段对应的所有索引文件的前缀值（见文章索引文件.si中的介绍），即_0，那么就可以在索引目录中找到 索引文件_0.si，而在 索引文件_0.si的Files字段（图15中红框标注）中存储了其他索引文件的名字，同样地根据这些索引文件的名字在索引目录中读取到所有的索引信息。

另外图15中SegmentCommitInfo中的两个字段FieldInfosFies、UpdatesFiles也是存储了索引文件的名字，当一个段中的DocValues发生变更时，变更的信息也用索引文件描述，并且索引文件的名字存储在这两个字段里。

从上文的描述可以看出，尽管我们通过IndexCommit可以获得SegmentInfos信息，但是该对象只是描述了它对应的索引文件有哪些，并不具有这些索引文件真正的数据，故可能在获得IndexCommit之后，索引又发生了变化，例如又出现了新的提交，那么根据**默认的索引删除策略**（见文档提交之commit（二）中关于`执行检查点(checkPoint)工作`d的介绍），segments_1文件就会被删除，当执行回滚操作时就无法获得真正的索引数据。如果出现在这个情况，那么在当前流程点会抛出如下的异常：

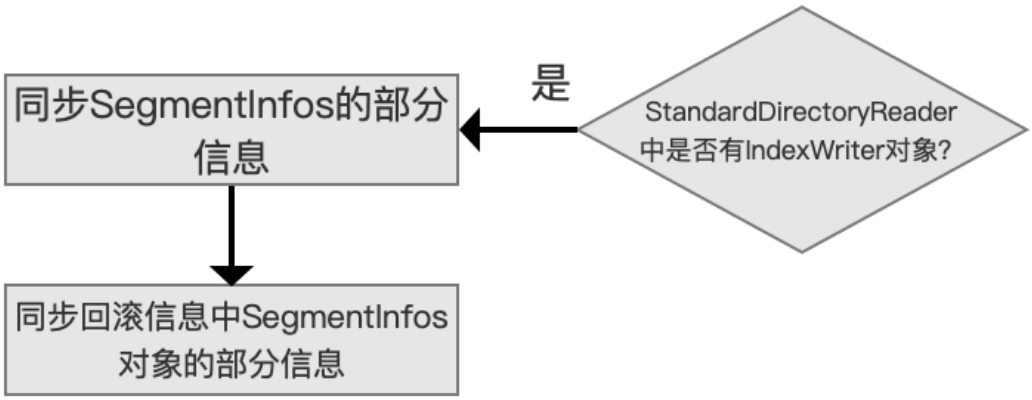
```
            throw new IllegalArgumentException("the provided reader is stale: its prior commit file \"" + segmentInfos.getSegmentsFileName() + "\" is missing from index");
```

其中segmentInfos.getSegmentsFileName()指的就是segment_1文件。

感兴趣的同学可以执行这个demo：https://github.com/LuXugang/Lucene-7.5.0/blob/master/LuceneDemo/src/main/java/lucene/index/StaleIndexCommit.java ，该demo模拟了上述异常的抛出。

同步SegmentInfos以及回滚信息中SegmentInfos中的部分信息

图16：



为什么StandardDirectoryReader中可能没有IndexWriter对象：

在近实时搜索NRT系列文章中我们说到可以通过下面四种open()方法获得一个StandardDirectoryReader，其中：

- 方法一：DirectoryReader.open(final Directory directory)
- 方法二：DirectoryReader.open(final IndexCommit indexCommit)
- 方法三：DirectoryReader.open(final IndexWriter indexWriter)
- 方法四：DirectoryReader.open(final IndexWriter indexWriter, boolean applyAllDeletes, boolean writeAllDeletes)

其中通过方法一根方法二获得的StandardDirectoryReader对象中是没有IndexWriter对象的，即使方法二的参数indexCommit对象中有IndexWriter对象。

为什么持有（引用）IndexWriter对象的StandardDirectoryReader需要执行图16中的两个同步操作：

源码中是这么说的：

```
// In case the old writer wrote further segments (which we are now dropping)
```

其中old Writer指的就是StandardDirectoryReader中的IndexWriter对象，上述注释的意思是为了能处理old writer可能生成的新提交（一个或多个），并且该提交是需要丢弃的。

该注释的详细意思就是：我们使用的IndexCommit参数对应的索引信息可能不是old writer最新的提交对应的索引信息，那么比IndexCommit更加新的的提交（一个或多个）都应该丢弃（dropping），为了能正确的处理那些应该被丢弃的段，我们需要上面图16中的两个更新操作。

为什么执行图16的两个更新操作就能正确的处理那些应该被丢弃的段：

处理的时机点在后面的流程中，到时候再介绍。

另外图16中两个同步的内容即version、counter、generation，在构造IndexWriter对象（三）文章中介绍了这三个值，这里不赘述。

设置回滚

该流程跟构造IndexWriter对象（三）文章中的中的`设置回滚`是相同的意思，将流程点`获得回滚（rollback）信息`的信息更新到rollbackSegments。

在设置回滚后，下一次commit前出现任何的错误，都可以回到当前设置的回滚状态，如果某次提交成功了，那么rollbackSegment会被重新设置该次提交。

结语

基于篇幅，剩余的内容将在下篇文章中展开。

点击下载附件