近实时搜索NRT（三）

在近实时搜索NRT（二）的文章中我们提到，Lucene提供了四种方法来获得StandardDirectoryReader对象，这里先简单总结下这四种open方法获取StandardDirectoryReader的差异：

- 方法一：DirectoryReader.open(final Directory directory)
  - 根据索引目录Directory中N值最大的segment_N文件（即最近的一次commit）获取已经提交的索引信息，未commit的文档的索引信息无法被读取，即使已经flush但未提交的文档的索引信息也无法被读取，**故这种获取StandardDirectoryReader的方法`不属于`NRT**
- 方法二：DirectoryReader.open(final IndexCommit indexCommit)
  - 我们如果使用封装了索引删除策略NoDeletionPolicy的SnapshotDeletionPolicy，我们可以记录每一次的提交，即IndexCommit（见近实时搜索NRT（一）），在索引目录中会生成多个segment_N文件，使用此方法可以获得任意一次提交的索引信息，可以用来回溯操作，**故这种获取StandardDirectoryReader的方法`不属于`NRT**
- 方法三：DirectoryReader.open(final IndexWriter indexWriter)
  - 该方法实际调用了方法四，区别在于另方法四中的两个参数applyAllDeletes为**true**、writeAllDeletes为**false**
- 方法四：DirectoryReader.open(final IndexWriter indexWriter, boolean applyAllDeletes, boolean writeAllDeletes)
  - 由于在生成IndexWriter对象阶段会先读取索引目录中已有的索引信息（旧的索引信息），并且更改索引（新的索引信息）需要通过IndexWriter对象，故通过IndexWriter对象能获得索引目录中所有的索引信息，**故方法三跟方法四获取StandardDirectoryReader的方法`属于`NRT**

在近实时搜索NRT（一）、近实时搜索NRT（二）中我们了解到，无论调用哪一个上述介绍的方法，其相同的逻辑都是将一个段的信息segmentCommitInfo封装为一个LeafReader，最后将多个LeafReader封装为StandardDirectoryReader，当索引目录中的索引信息发生更改时，我们可以通过**重新调用**上述的方法来获得最新的StandardDirectoryReader，但是基于下面的几个考虑，Lucene提供了性能更高的openIfChanged方法来获得最新的StandardDirectoryReader：

- 只将部分发生更改的段生成LeafReader，即仅替换StandardDirectoryReader中的那些发生变更的LeafReader
- 如果索引信息没有发生变化，那么就直接返回StandardDirectoryReader，而不用执行上述四个方法的所有流程

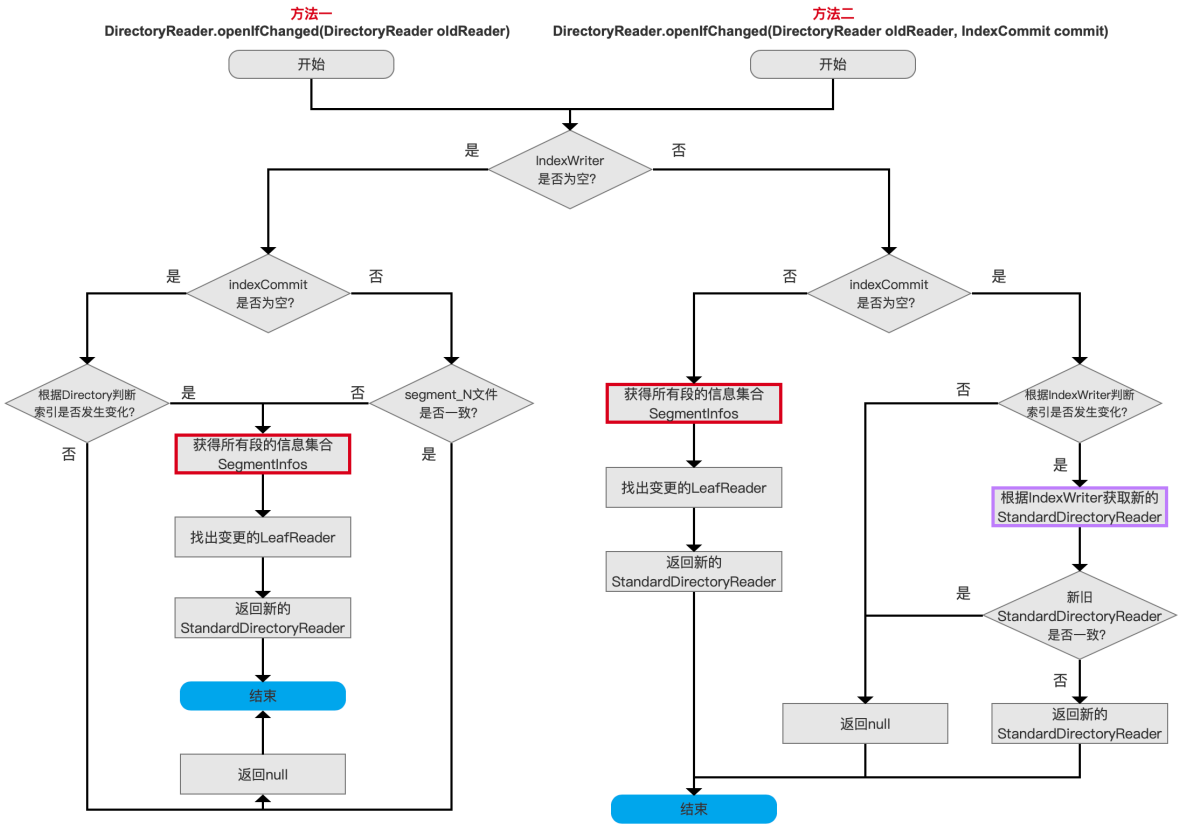
Lucene7.5.0中提供了以下四种openIfChange方法，这四种方法：

- 方法一：DirectoryReader.openIfChanged(DirectoryReader oldReader)
- 方法二：DirectoryReader.openIfChanged(DirectoryReader oldReader, IndexCommit commit)
- 方法三：DirectoryReader openIfChanged(DirectoryReader oldReader, IndexWriter writer)
- 方法四：DirectoryReader openIfChanged(DirectoryReader oldReader, IndexWriter writer, boolean applyAllDeletes)

openIfChange方法的流程图

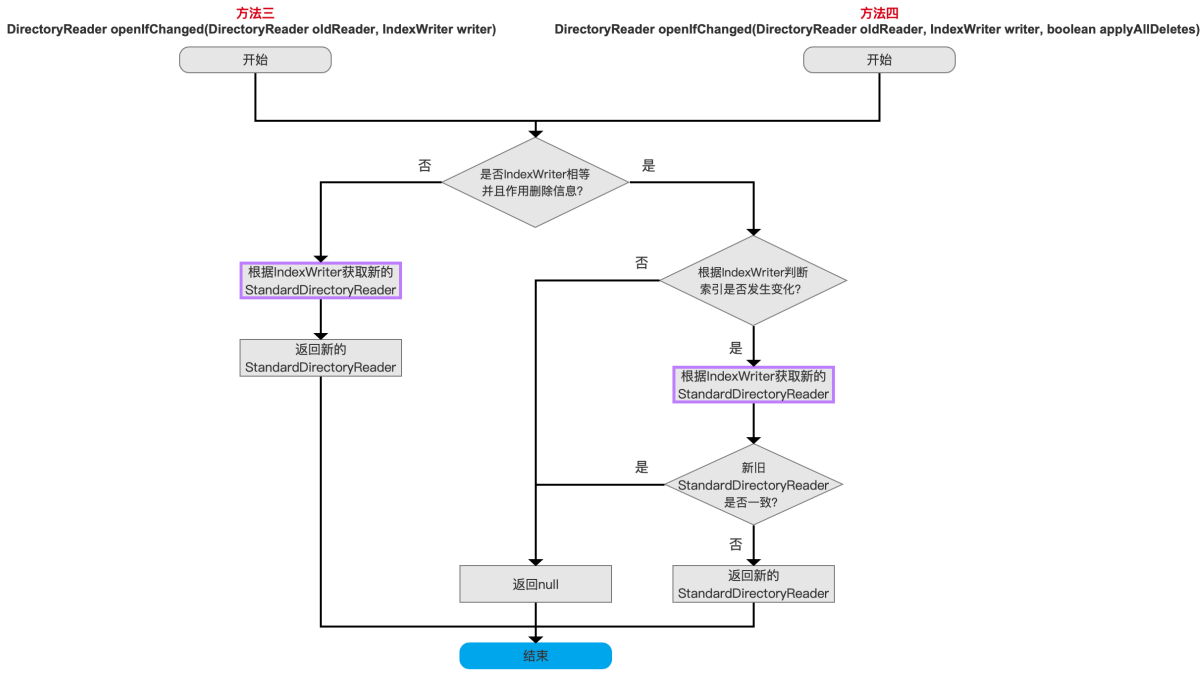
其中openIfChange的方法一&&方法二、方法三&&方法四的逻辑需要用两个流程图图1、图2展现：

图1：



点击查看大图

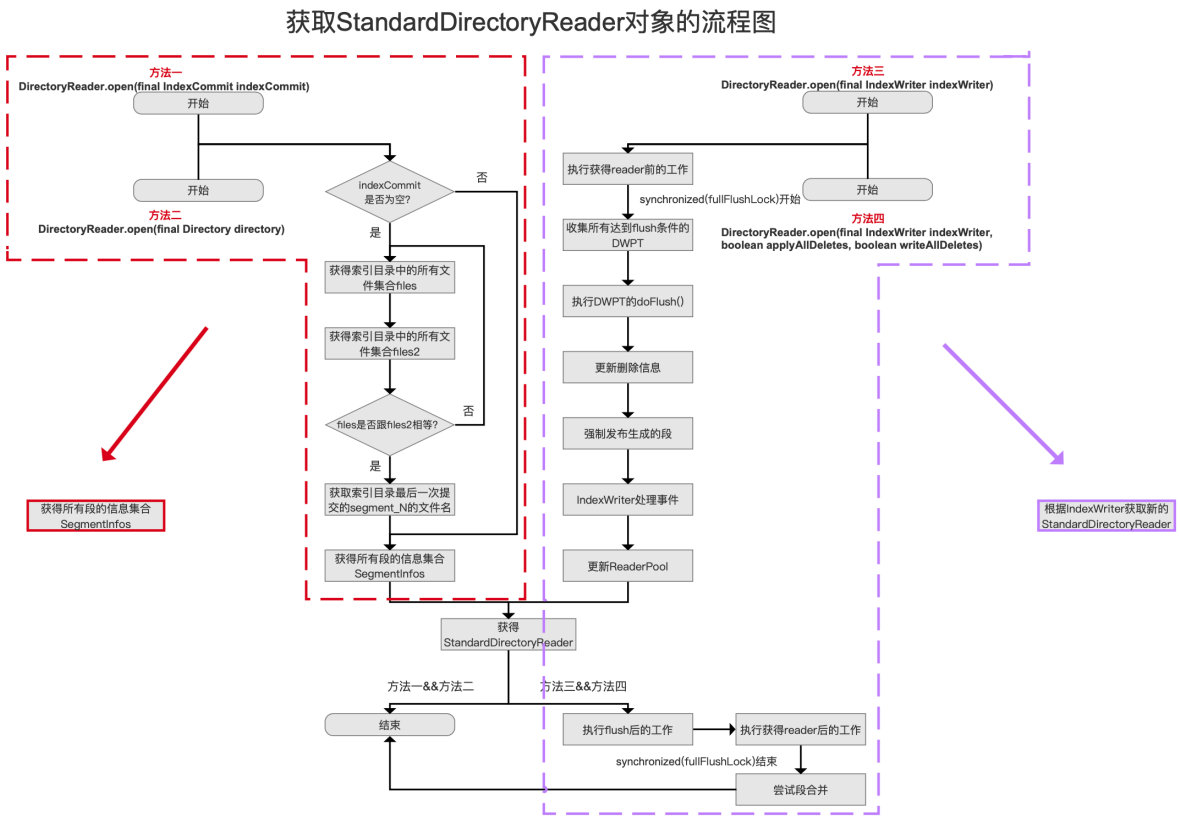
图2：



点击查看大图

在介绍每一个流程点之前，我们先大致先说下图1、图2的流程，两个流程图总体描述了旧的StandardDirectoryReader，即oldReader（DirectoryReader的子类，见近实时搜索NRT（一））在经过一系列的流程后，判断是否需要生成新的StandardDirectoryReader，如果不需要那么返回null，否则生成一个新的StandardDirectoryReader，生成的新的StandardDirectoryReader的方法有两个，即图1、图2中用**红色**跟**紫色**标注的两个流程点，而这两种获取新的StandardDirectoryReader的方法即近实时搜索NRT（一）、近实时搜索NRT（二）介绍的四种open方法，如下图所示：

图3：

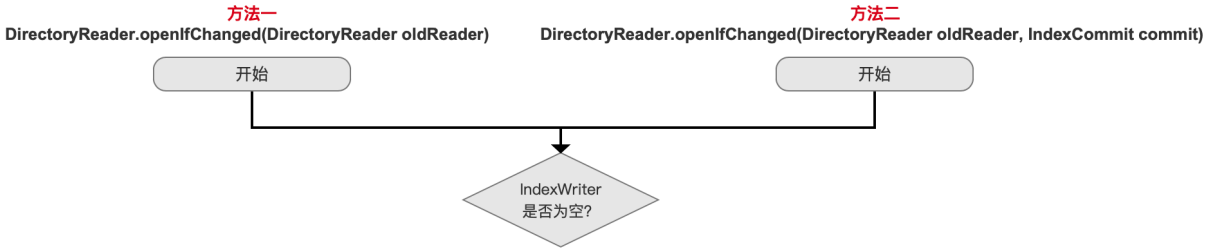


点击查看大图

下面我们先介绍图1中的每一个流程点。

IndexWriter是否为空?

图4：



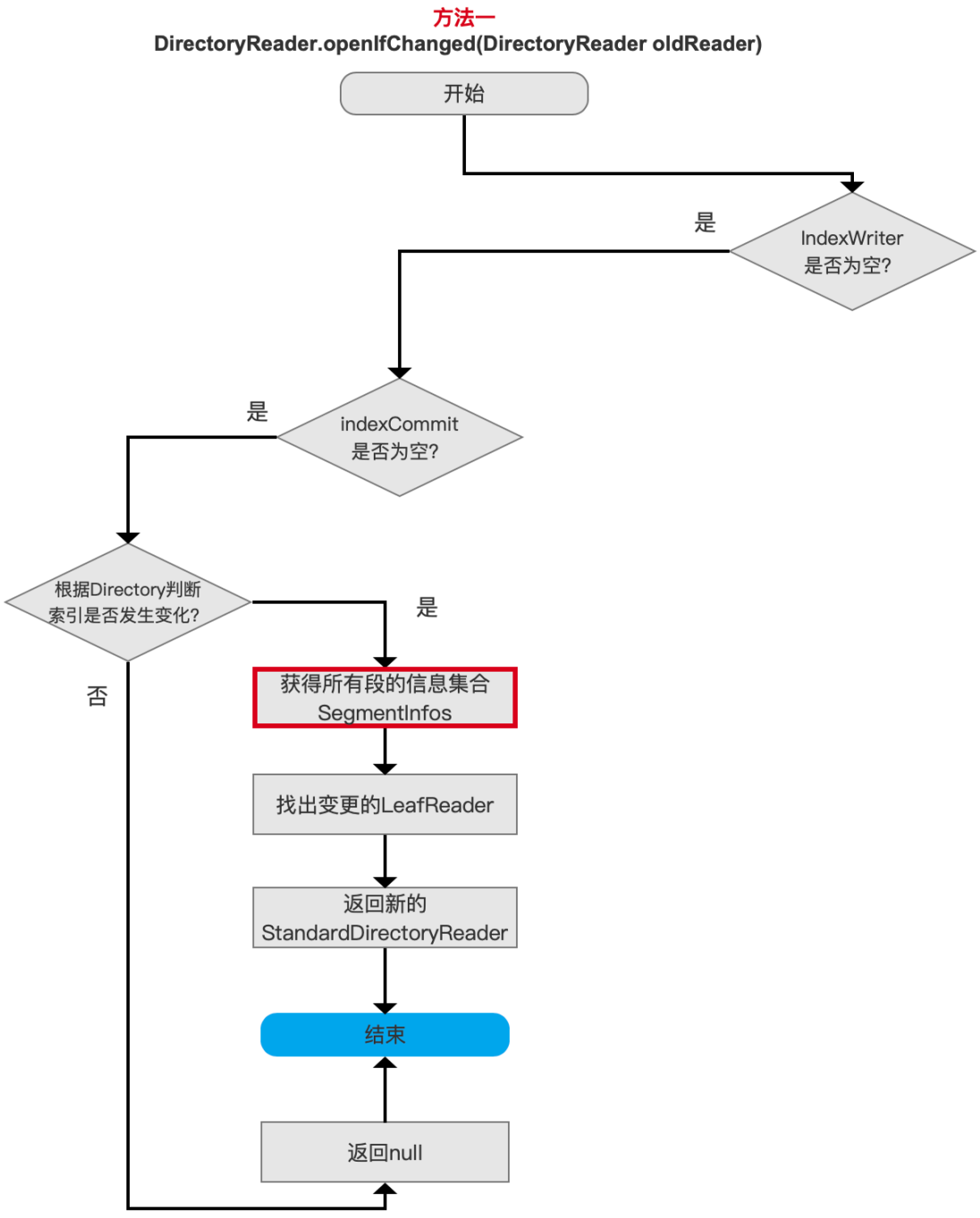
IndexWriter是否为空描述的是旧的StandardDirectoryReader，即oldReader中是否持有IndexWriter对象的引用。

如何判断oldReader中是否持有IndexWriter对象的引用：

- 上文中我们知道四种获取StandardDirectoryReader的open方法中，方法三&&方法四的参数带有IndexWriter对象，故通过这两个方法获得的oldReader中会持有IndexWriter对象的引用，同理，通过方法一&&方法二获得的oldReader中不持有IndexWriter对象的引用

根据Directory判断索引是否发生变化?

图5：



如果oldReader是通过open方法中的方法一或者方法二获得的，并且调用openIfChange的方法一，由于该方法没有indexCommit参数，故该方法会执行图5中的流程。

方法一只提供了一个oldReader的参数，并且没有持有IndexWriter对象的引用，所以判断是否需要获取新的StandardDirectoryReader的方式为根据索引目录中最后一次commit，对比最后一次commit对应的SegmentInfos对象跟oldReader中的SegmentInfos的版本号Version是否一致，如果不一致，说明索引信息在生成oldReader之后发生了变化，那么我们需要重新生成一个新的StandardDirectoryReader，否则返回null。

SegmentInfos是什么：

- 见近实时搜索NRT（一）文章中的关于流程点`获得所有段的信息集合SegmentInfos`的介绍

SegmentInfos的版本号Version是什么：

- SegmentInfos的版本号描述了内存中的SegmentInfos变更状态，内存中的任意一个段发生变化都会增加版本号，当执行了一次commit后，这次commit的对应的SegmentInfos的版本号就被写入到segment_N文件中，如下图所示：

段的什么变化会引起版本号Version的变化：

- 这块内容十分重要，由于openIfChange的方法三&&方法四会也会涉及版本号的内容，故我们留到后面的文章介绍，这里先挖个坑

图6：

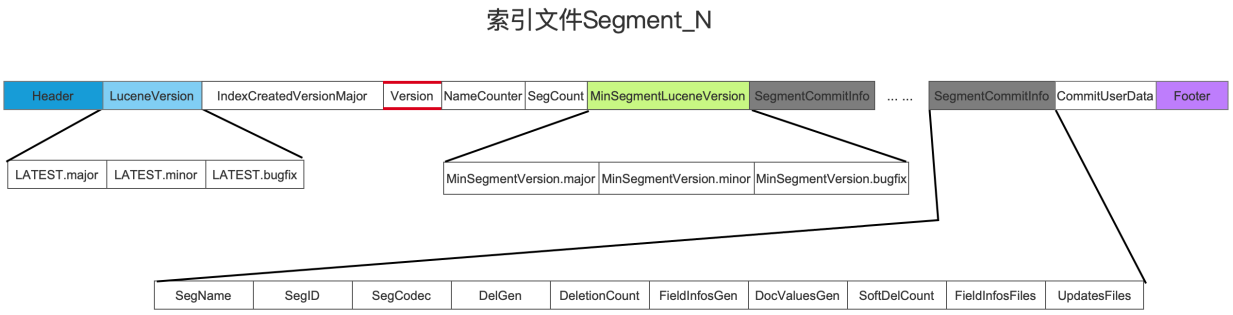


图6中，红框标注的为某次提交对应的SegmentInfos的版本号。

如何根据最后一次commit获得SegmentInfos：

- 见近实时搜索NRT（一）文章中的关于流程点`获取segment_N文件`、`获得所有段的信息集合SegmentInfos`的介绍

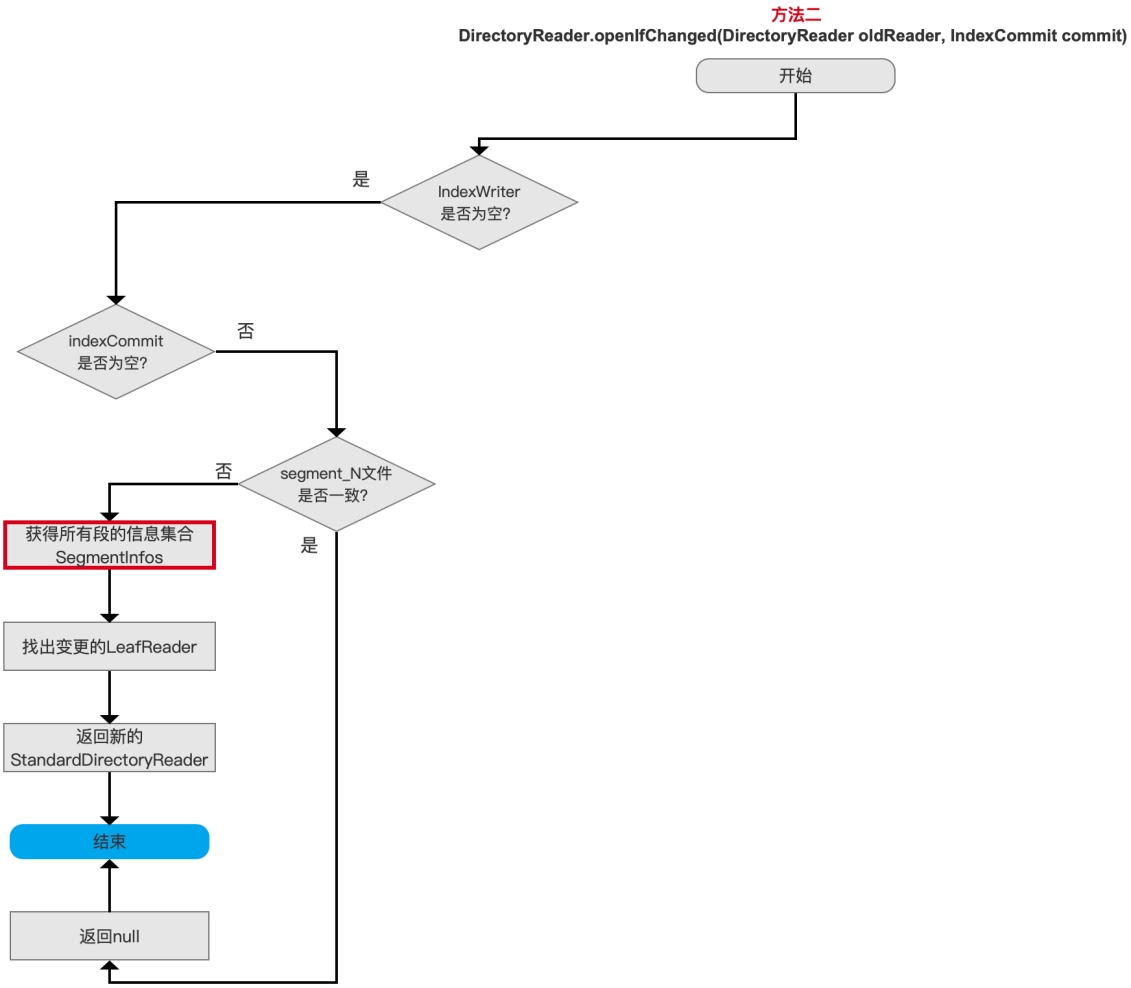
图5中如果根据Directory判断索引发生了变化，那么通过图3的流程获得一个新的StandardDirectoryReader。

如何获得新的StandardDirectoryReader：

- 当Version不一致时，说明oldReader中的SegmentInfos跟内存中的SegmentInfos信息不一致，那么需要获得新的StandardDirectoryReader，即图5中的`找出变更的LeafReader`流程点，由于后面的流程也有该操作，我们留到下一篇文章介绍，这里先挖个坑

segment_N文件是否一致?

图7：



如果oldReader是通过open方法中的方法一或者方法二获得的，并且调用openIfChange的方法二，由于该方法有indexCommit，那么该方法会执行图7中的流程。

方法二描述的是根据参数IndexCommit来获得对应的StandardDirectoryReader，另外通过比较oldReader中对应的segment_N文件名是否跟方法二的参数IndexCommit中的segment_N文件名是否相同来判断是否需要获得新的StandardDirectoryReader。

如何通过oldReader获得对应的segment_N文件：

- oldReader即StandardDirectoryReader类的对象，该类中并没有一个存放segment_N的变量，而是通过StandardDirectoryReader类中的SegmentInfos对象来获得segment_N文件，SegmentInfos对象在上文中已介绍，该对象中有一个lastGeneration，Lucene通过字符串拼接（"segment_lastGeneration"）的方式获得segment_N文件的文件名

lastGeneration什么时候被赋值的：

- 在文档提交之commit（二）的`生成新的Segment_N文件`流程中我们说到，该流程会生成一个generation用来设置这次提交生成的segment_N的N值，同时generation值会赋值给lastGeneration

另外IndexCommit类中有一个变量叫做segmentsFileName，该变量用来保存segment_N的文件名。

当判断出segment_N的文件名不一致，那么我们需要重新获得StandardDirectoryReader，即图7中的`找出变更的LeafReader`流程点，同样的，详细的获得过程在下一篇文章中展开。

结语

基于篇幅，剩余的流程点将在下一篇文章中展开。

点击下载附件