近实时搜索NRT（一）

Lucene提供了近实时搜索NRT（near real time）的功能，它描述了索引信息发生改变后，不需要执行commit操作或者关闭IndexWriter（调用IndexWriter.close()方法）就能使得这些更改的信息**很快（quickly）**变得可见。

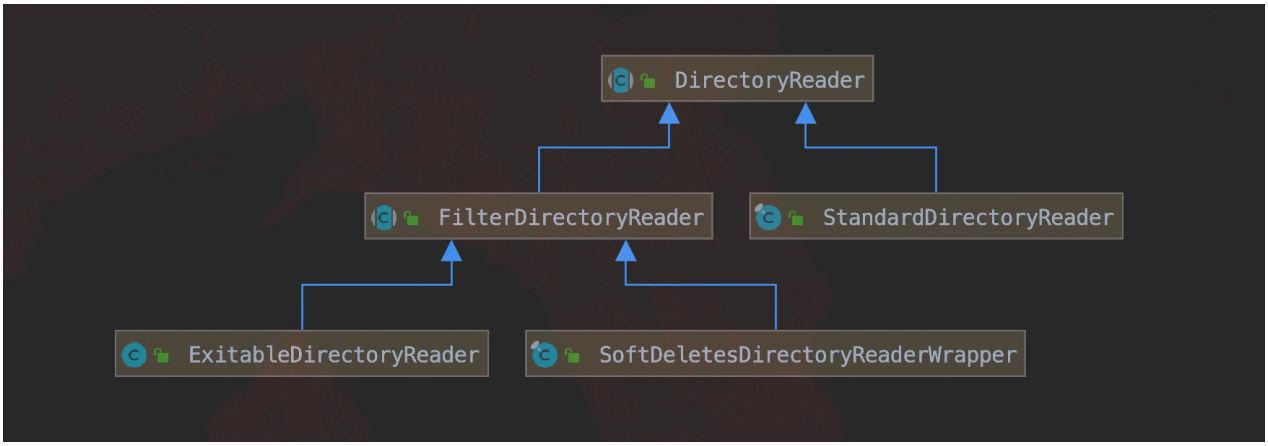
"很快"意味着不是马上变得可见，Lucene无法保证更改索引信息后，在**某个确定的时间**之后，使得最新的索引信息变得可见，这取决具体的业务。

近实时搜索NRT通过DirectoryReader来实现，故下面的文章中将会介绍DirectoryReader的实现原理。

DirectoryReader

通过DirectoryReader，我们可以读取索引目录中已经提交（执行commit操作）或者未提交（执行flush操作）的段的信息，IndexSearcher通过DirectoryReader包含的信息来进行查询，它是一个抽象类，其类图如下所示，在下面的内容中会详细介绍DirectoryReader如何生成的：

图1：



上图中，DirectoryReader有两个子类StandardDirectoryReader、FilterDirectoryReader，其中StandardDirectoryReader是Lucene7.5.0中默认的DirectoryReader具体实现，FilterDirectoryReader作为一个抽象类，它的两个具体实现ExitableDirectoryReader、SoftDeletesDirectoryReaderWrapper通过封装其他的DirectoryReader对象，来实现功能扩展。下面我们一一介绍图1中的每一个类。

StandardDirectoryReader

StandardDirectoryReader作为Lucene7.5.0中默认且唯一的DirectoryReader类的具体实现，我们需要详细的来了解它，该类的对象可以通过个调用以下四个方法来获得：

- 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)

其中通过调用方法三&&方法四的方法实现了NRT功能，而方法一&&方法二则没有,下文将会描述它们之间的差异。

获取StandardDirectoryReader对象的流程图

尽管提供了4种方法，但这些方法的实现原理都可以在一张流程图中表示：

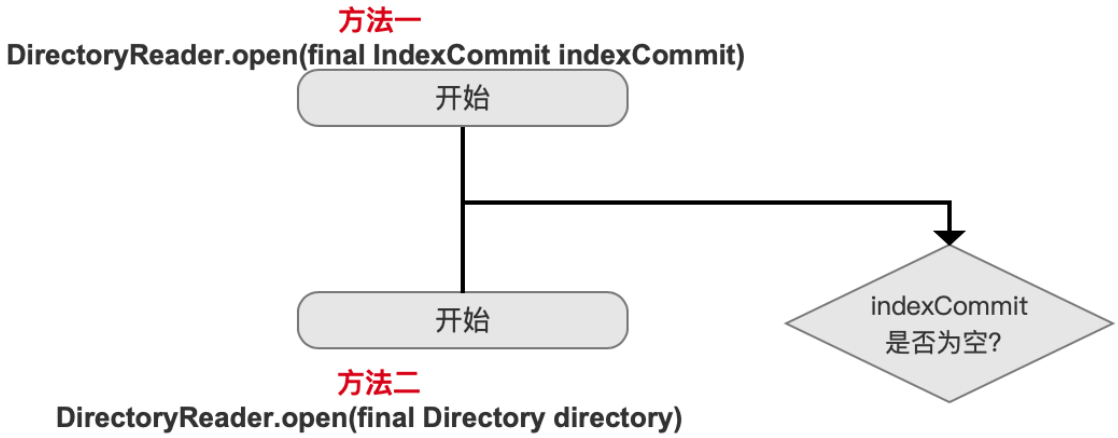
图2：

点击查看大图

我们先介绍方法一&&方法二的所有流程点。

统一入口

图3：



IndexCommit是什么：

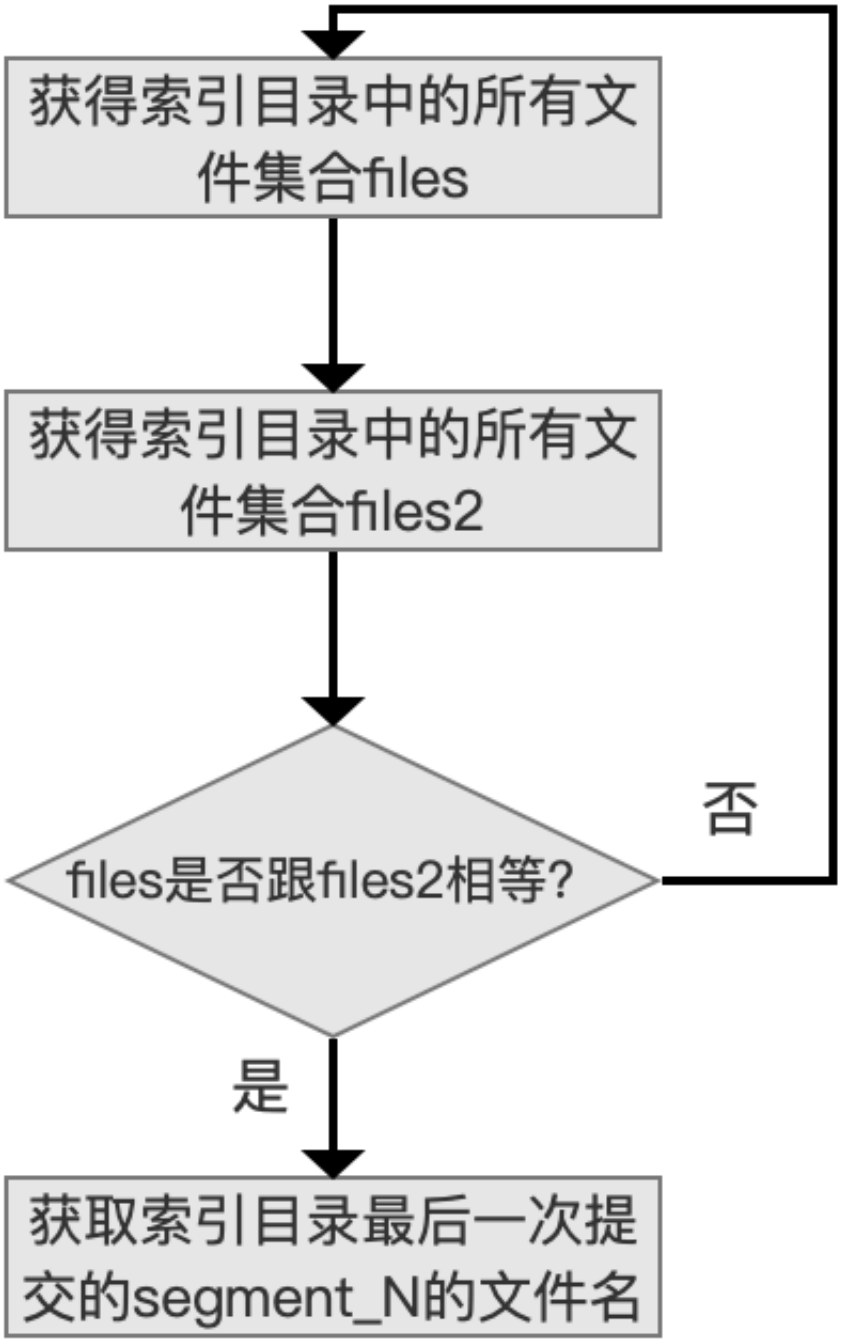
- 执行一次提交操作（执行commit)方法）后，这次提交包含的所有的段的信息用IndexCommit来描述，其中至少包含了两个信息，分别是segment_N文件跟Directory

如何获得IndexCommit：

- 在文档提交之commit（二）的文章中，我们提到了一种索引删除策略SnapshotDeletionPolicy，在每次执行提交操作后，我们可以通过主动调用SnapshotDeletionPolicy.snapshot()来实现快照功能，而该方法的返回值就是IndexCommit

获取segment_N文件

图4：



该流程只有方法二才会执行，图4中的多个流程，执行它们的最终目的是为了获得索引目录（根据方法二的参数Directory对象获得索引目录）中的segment_N文件。

在图4的流程中，Lucene连续两次获取索引目录中的所有文件，获得两个文件名集合file、file2，在分别对file、file2进行排序后，通过比较两个集合是否包含相同的文件名来判断是否当前索引目录是否有频繁的新的写入操作，如果有，那么通过重试的方法，直到file、file2是相同的文件集合，由于这段代码较为简洁，故直接给出：

```
String files[] = directory.listAll();
String files2[] = directory.listAll();
Arrays.sort(files);
Arrays.sort(files2);
if (!Arrays.equals(files, files2)) {
// listAll() is weakly consistent, this means we hit "concurrent modification
exception"
continue;
}
```

在上面的代码中，其中directory即方法二的参数。

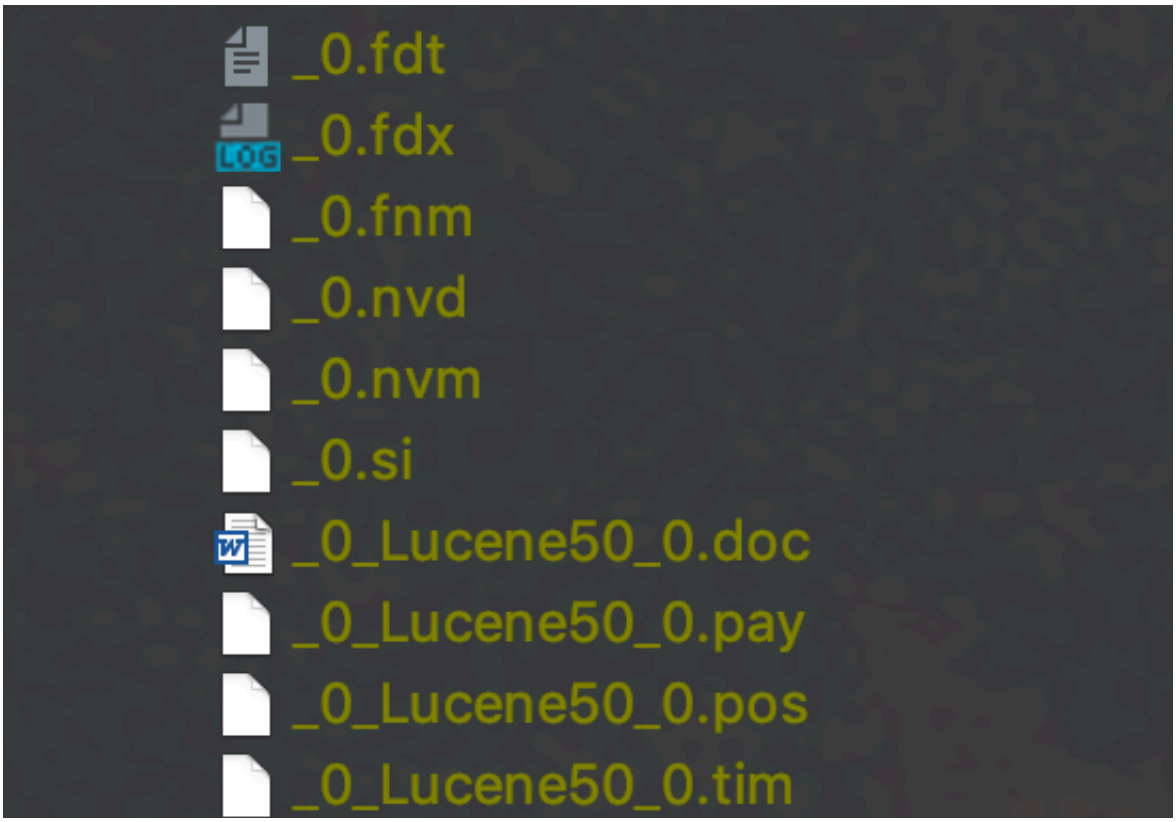
从上面的代码我们可以看出哪些信息：

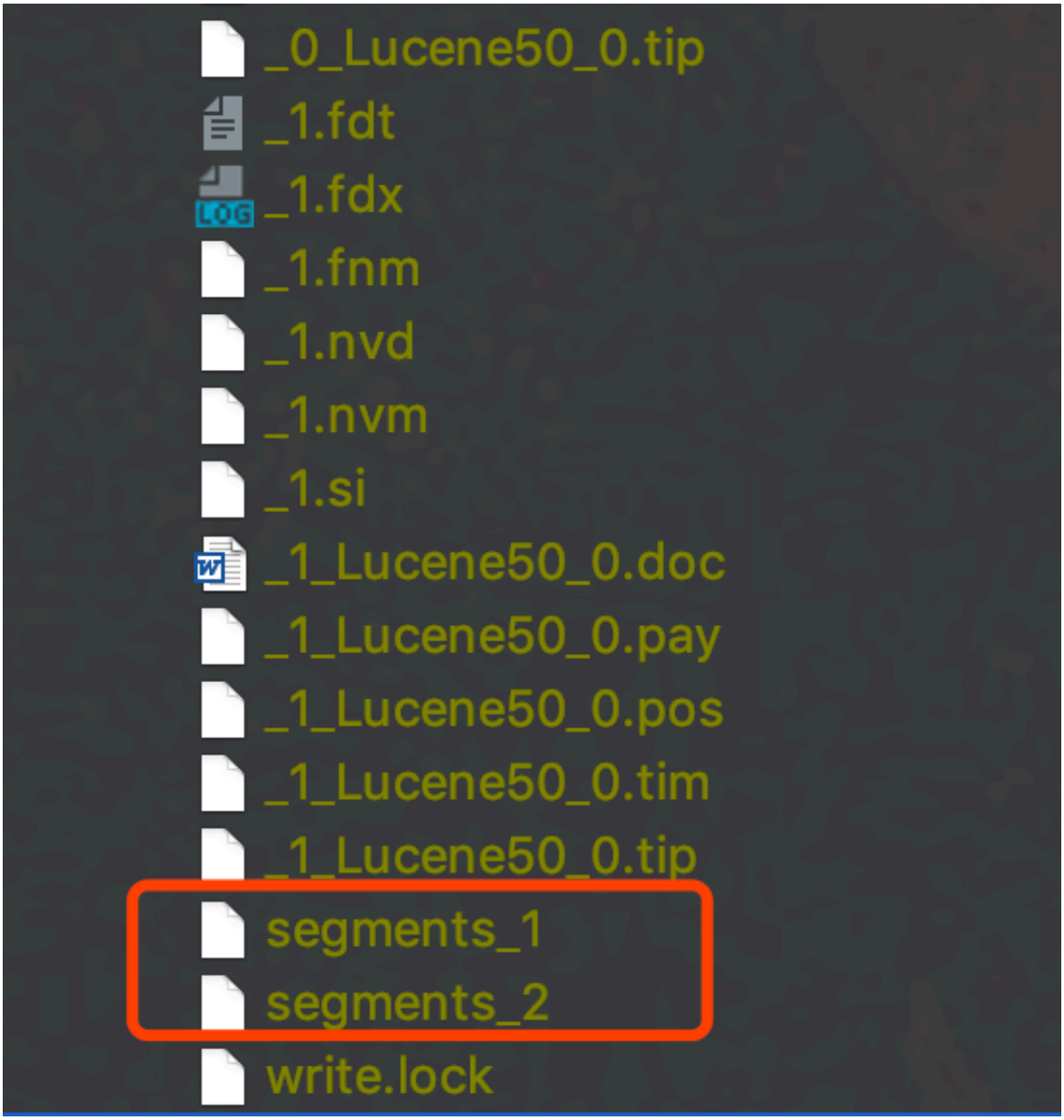
- 信息一：在多线程下，当频繁的有修改索引信息的操作时，获取一个读取索引文件的操作可能会有较高的延迟（files跟files2一直无法相等）
- 信息二：从图1的流程中可以看出，方法二的流程并没有同步操作，故即使在某一时候files跟files2相等，跳出图4的重试操作，有可能索引信息马上被别的线程修改了，故在不同步索引修改方法（见文档的增删改）的前提下，不一定能获得最新的索引信息，该方法至少能保证获得调用方法二之前的索引信息

如何根据文件集合files来获得最后一次提交的segment_N文件：

- 每次提交都会生成segment_N，其中N是一个递增的值，它描述了一个段的编号，即最新的提交对应的N值是最大的，Lucene通过遍历files中的每一个文件名，取出前缀为"segment"，并且段编号最大的文件，该文件即最后一次提交的segment_N文件，图5中，执行了两次提交操作，并采用索引删除策略NoDeletionPolicy（见文档提交之commit（二）），故索引目录中保留了2个segment_N文件，并且最后一次提交的segment_N文件是segment_2

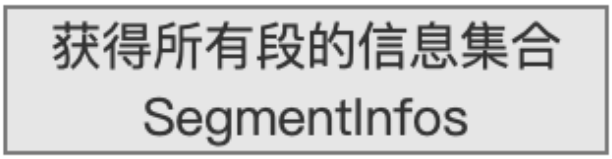
图5：





获得所有段的信息集合SegmentInfos

图6：

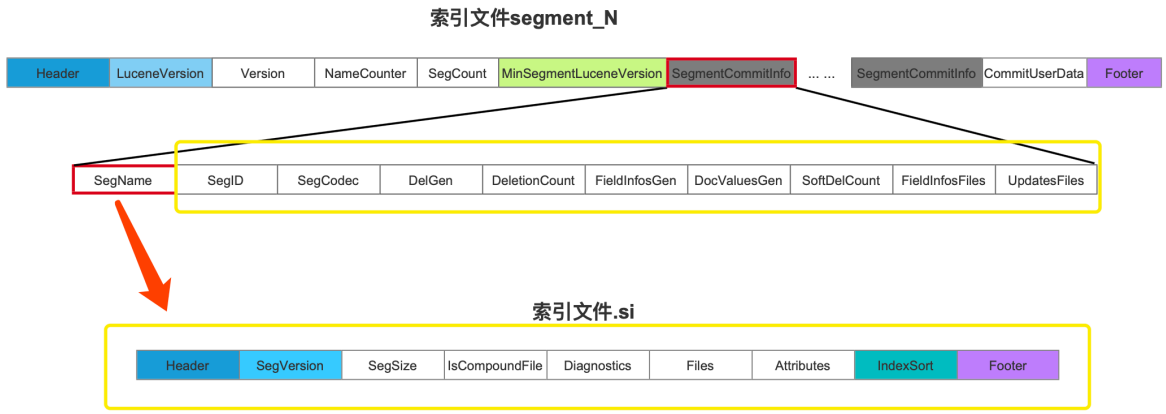


在前面的流程中，无论是方法一还是方法二，到达此流程点时，已经获得了segment_N文件，那在图6的流程中，我们根据segment_N获取段的信息集合segmentInfos，该集合在前面的文章中已经多次介绍，它包含的一个重要信息是一个链表，链表元素是每一个段的信息：

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

SegmentCommitInfo的数据分散在两个索引文件中，如图7所示：

图7：



点击查看大图

图7中，两块黄色的框标注的内容即SegmentCommitInfo的信息，在读取segment_N阶段，先读取出SegmentCommitInfo的第一块数据，**然后根据第一块数据的中的SegName（该字段的含义见segment_N，生成segment_N的时机见文档提交之commit（二））从索引文件.si（生成索引文件.si的时机见文档提交之flush（三））中读取出SegmentCommitInfo的第二块数据**，两块数据组成完整的SegmentCommitInfo的信息，在源码中，这些信息即SegmentCommitInfo类的成员变量。

获得StandardDirectoryReader

图8：

在前面的流程中，我们获得了每一个段的SegmentCommitInfo，在Lucene中，将SegmentCommitInfo再次封装为SegmentReader，然后将所有段对应的SegmentReader最后封装为StandardDirectoryReader。

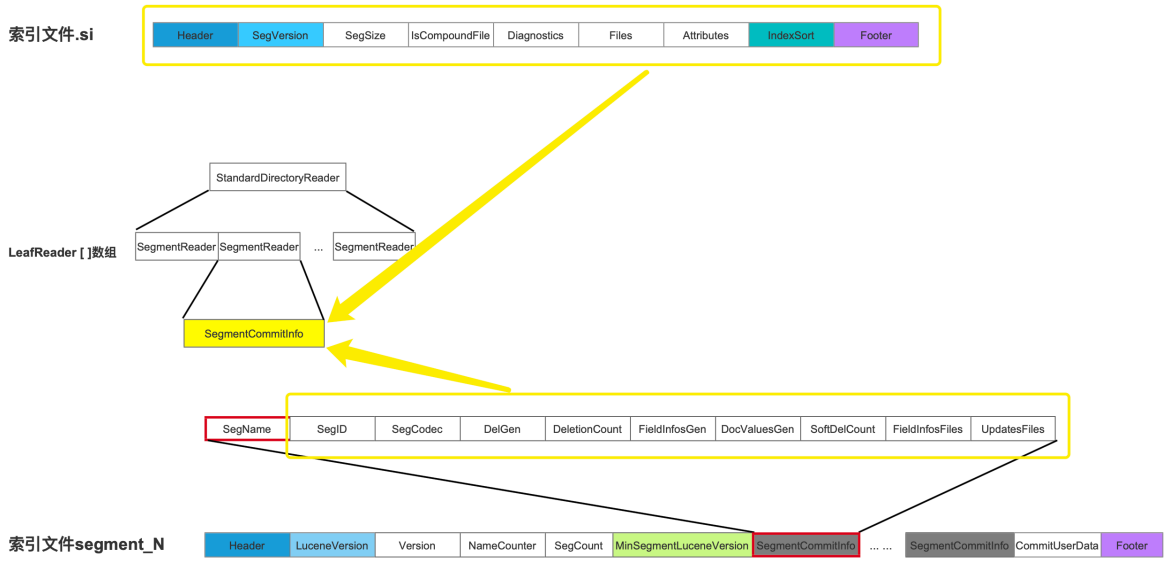
SegmentReader

SegmentReader类不展开作介绍了，我们只需要知道该类描述的是一个段的所有的信息，该信息通过封装的SegmentCommitInfo来描述。

StandardDirectoryReader

所有段对应的SegmentReader集合以数组方式LeafReader[ ]被封装在StandardDirectoryReader中，如下图所示：

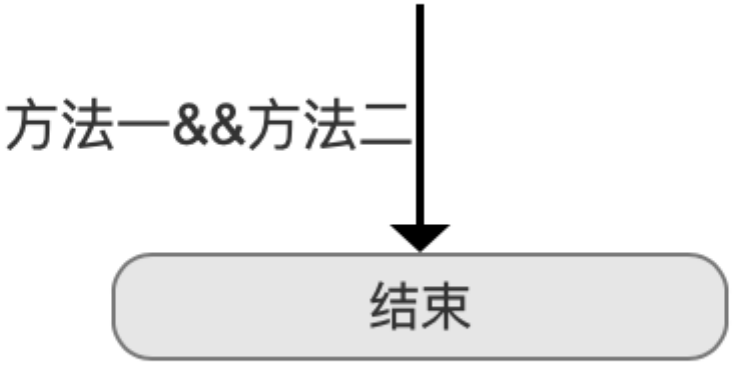
图9：



点击查看大图　　图9中描述了StandardDirectoryReader中包含的最重要的一些数据，其中LeafReader[ ]数组中的元素个数跟索引文件segment_N中的黑框字段SegmentCommitInfo的个数是一致的。

结束

图10：



对于方法一&&方法二，严格的来讲，至此我们获得了流程点`获得所有段的信息集合SegmentInfos`之前索引目录中最新的索引数据，由于其他线程可能通过IndexWriter并发的执行更改索引的操作，所以在多线程下，方法一&&跟方法二并不能实现NRT的功能。

结语

基于篇幅，剩余的内容在一篇文章中展开。

点击下载附件