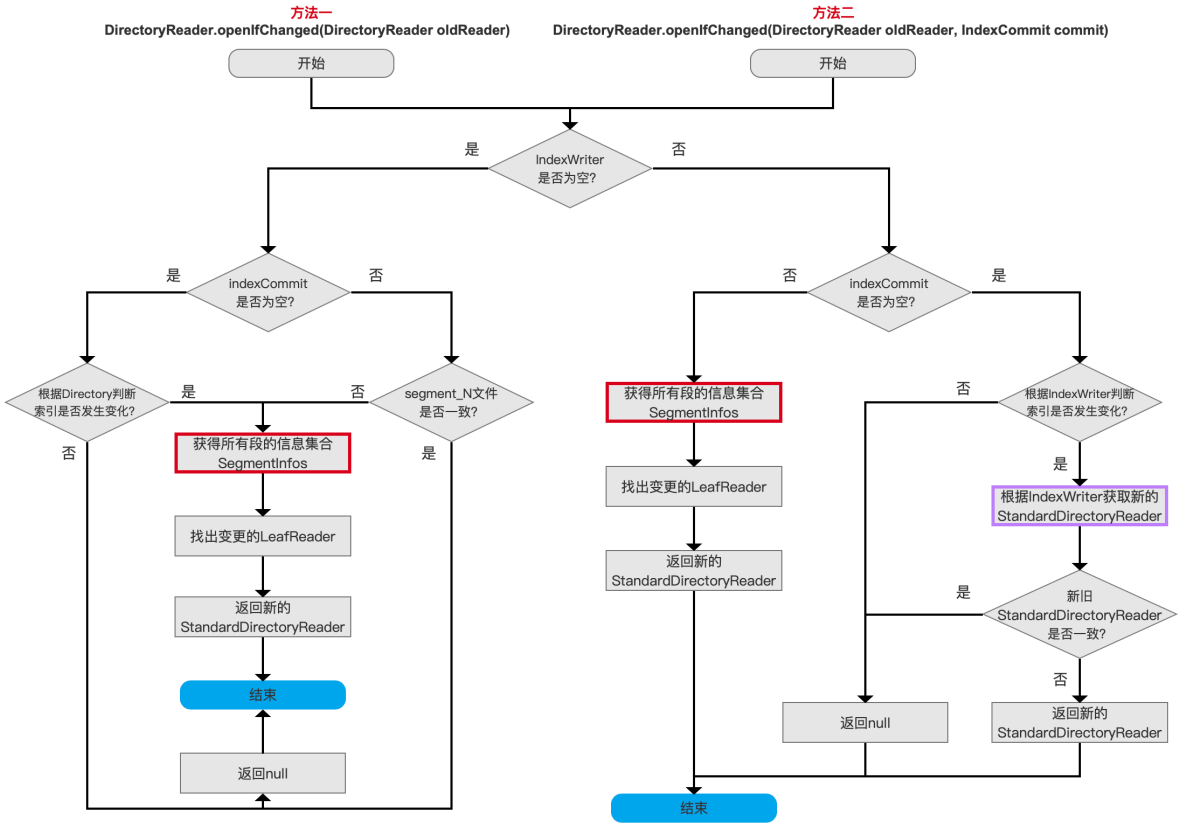
近实时搜索NRT（五）

本文承接近实时搜索NRT（四），继续依次介绍每一个流程点，阅读本文章需要看过文档的增删改、文档提交之flush的系列文章。

openIfChange方法的流程图

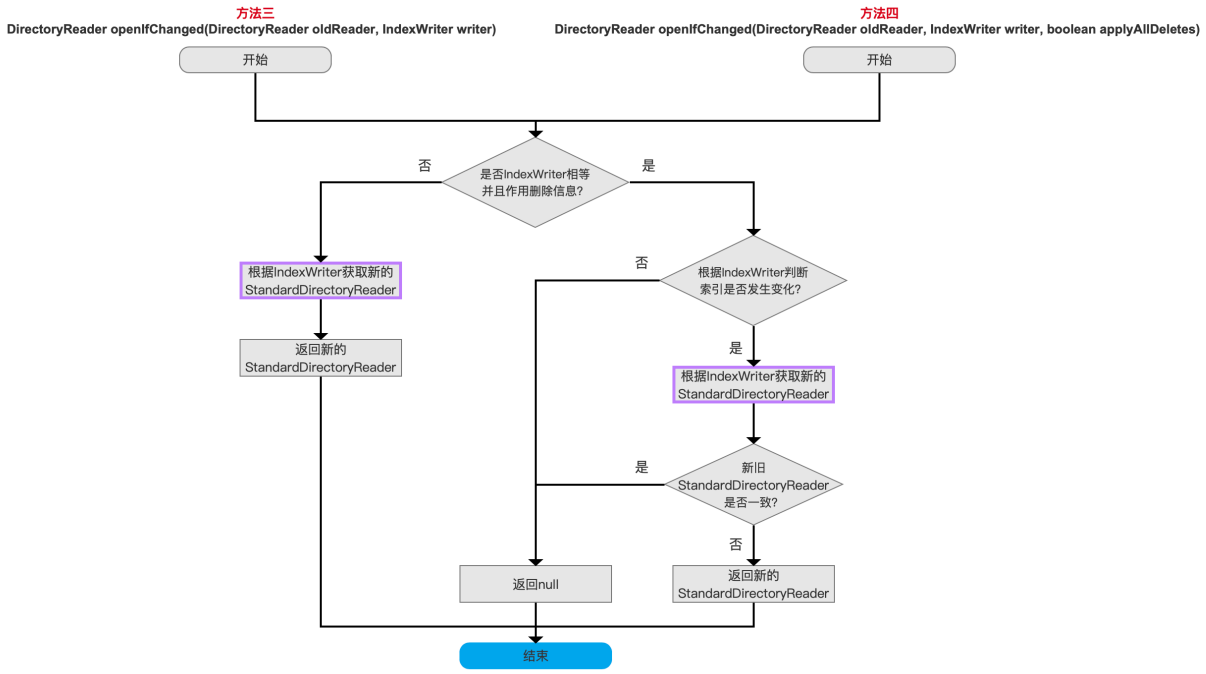
其中openIfChange的方法一&&方法二、方法三&&方法四的逻辑需要用图1、图2两个流程图展现：

图1：



点击查看大图

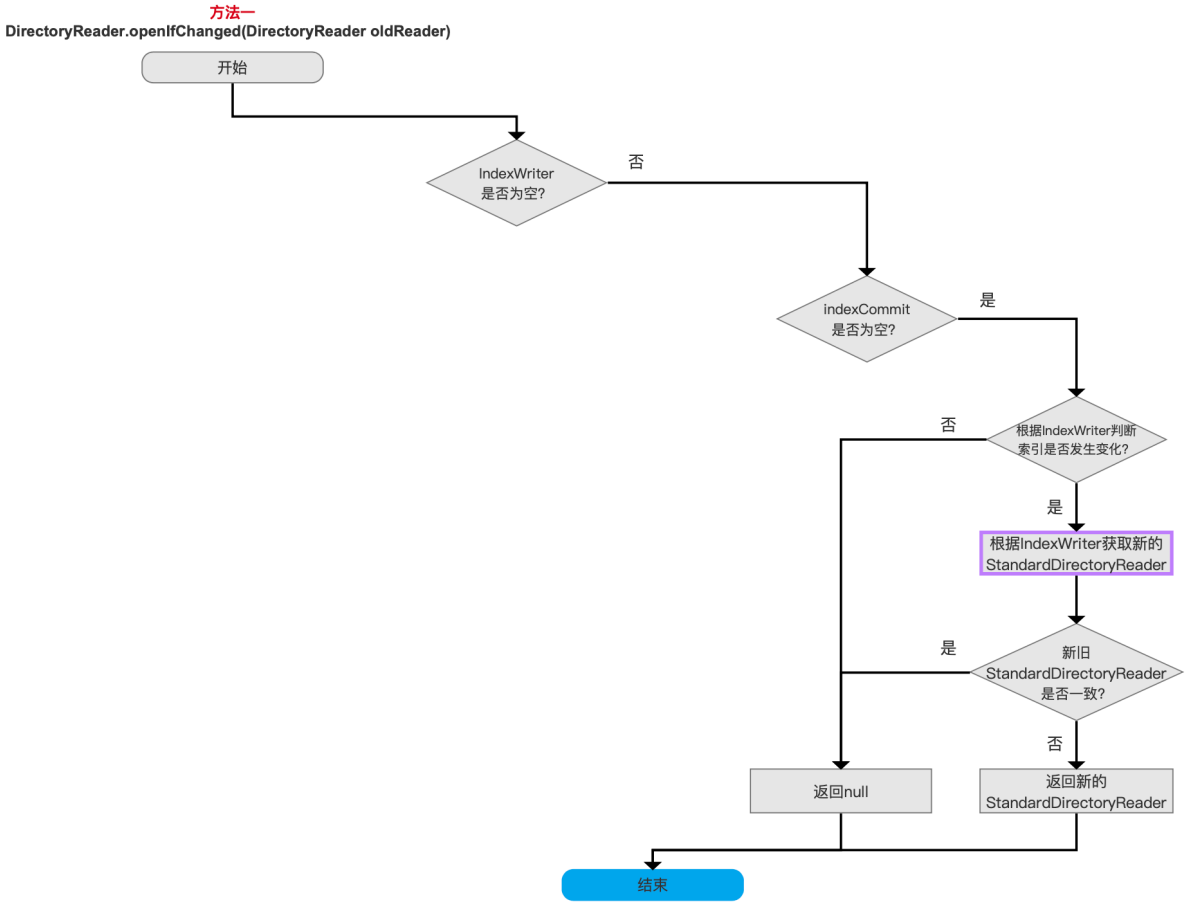
图2：



[点击](#)查看大图

从IndexWriter中获取StandardDirectoryReader

图3:



如果oldReader是通过open方法（见近实时搜索NRT（一）、近实时搜索NRT（四））中的方法三或者方法四获得，那么oldReader就持有IndexWriter对象（见近实时搜索NRT（三）），当调用了openIfChange的方法一，该方法就会执行图3中的流程，这种方法属于NRT。

图3的流程描述了这么一个过程：根据oldReader中持有的IndexWriter判断索引是否发生变化，即流程点`根据IndexWriter判断索引是否发生变化?`，发生变化则需要重新生成StandardDirectoryReader，如果生成了新的StandardDirectoryReader，那么需要再次跟oldReader作比较，即流程点`新旧StandardDirectoryReader是否一致?`，如果不相同，那么返回这个新的StandardDirectoryReader，否则返回null。

根据IndexWriter判断索引是否发生变化需要判断下面四个条件，`依次`同时满足视为没有发生变化：

- 条件一：oldReader与IndexWriter他们俩分别持有的SegmentInfos的版本号Version必须相同
  - SegmentInfos的介绍见近实时搜索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的介绍见近实时搜索NRT（三）
  - 版本号不相同意味着执行了索引目录中一个或多个旧段发生了更改，或者生成了一个或多个新段，或者生成了还未提交的索引信息（flush操作）
- 条件二：目前没有正在处理的文档，该条件由四个子条件组成，所有子条件都满足视为满足条件二
  - 子条件一：numDocsInRAM的值为0
    - 每当DWPT处理一篇文档，该值会加一（见文档的增删改的系列文章），在DWPT生成一个段后，该值会减去DWPT中包含的文档数（见文档提交之flush的系列文章），如果numDocsInRAM的不为0，说明目前有些DWPT中包含的文档信息还未生成对应的段，也就说明了oldReader需要更新
  - 子条件二：全局删除队列deleteQueue（见文档的增删改（下）（part 2））中的结点个数为0
    - 不满足子条件二说明还有未处理的删除信息，这些删除信息可能会作用（apply）索引目录中已有的段，如果结点个数不为0，说明没有一个DWPT开始执行生成段的工作（见文档提交之flush（六）），也就是没有flush操作（主动flush或者自动flush），也就是索引信息将要变化，oldReader需要更新
  - 子条件三：ticketQueue（见文档提交之flush（四））中的元素个数为0
    - ticketQueue不为0，说明还有某些段没有被发布（发布的含义见文档提交之flush（五））
  - 子条件四：pendingChangesInCurrentFullFlush的值为false
    - 由于fullFlush操作（见文档提交之flush（一））跟openIfChange操作可以是并行的，pendingChangesInCurrentFullFlush用来保证执行完fullFlush才能使得openIfChange来判断索引是否发生变化
- 条件三：所有的删除信息都已经作用到其他的段
  - 在文档提交之flush（五）中我们介绍了`发布生成的段`的流程，该流程中从ticketQueue中依次取出FlushTicket，随后执行的两个工作分别是将删除信息作用到其他的段和将DocValues更改信息作用到其他的段，只有这个两个工作完成了，才能保证段的信息不会再被改变，否则段可能会被改变，那么oldReader需要更新
- 条件四：所有的DocValues更改信息已经作用到其他的段
  - 同条件三

需要满足条件二、条件三、条件四正是说明了我们重新获得StandardDirectoryReader中包含了未提交（commit操作）的索引信息，所以这种方法属于NRT（见近实时搜索NRT（一）文章中对NRT的定义）。

为什么需要同时满足上面的四个条件（重要）：

- 任何对段进行更改的操作都会更改segmentInfos的版本号Version，所以在单线程下，实际上我们只需要判断Version就能知道索引目录中的索引信息是否发生更改，然而在多线程下，其他线程可能正在执行文档的增删改操作，而这些操作就可以通过条件二、条件三、条件四来判断（至于为什么通过这三个条件判断，我想阅读过文档的增删改以及文档提交之flush的系列文章才能明白，无法通过一两句话解释清楚），故在多线程下，如果一个正在执行方法一的线程仅仅判断segmentInfos的版本号Version无法获得实时性较好的StandardDirectoryReader
- 所以这也是为什么上文中提到，满足这四个条件是`依次`同时满足，并且判断版本号是第一个判断条件

生成了新的StandardDirectoryReader后，为什么还要跟oldReader比较是否一致（相同）：

- 该问题即图3中的`新旧StandardDirectoryReader是否一致?`的流程点，它的判断方法是通过比较新旧两个StandardDirectoryReader他们分别持有的segmentInfos的版本号Version是否一致
- 首先判断的目的是为了保证方法返回值的正确性，因为openIfChange方法的返回值只有两种情况，要么null要么不是null，null说明了索引信息没有发生变化，反之发生了更改，也就是新的StandardDirectoryReader跟oldReader包含相同的索引信息

为什么已经在流程点`根据IndexWriter判断索引是否发生变化?`已经判断出索引信息发生变化，还会出现新的StandardDirectoryReader跟oldReader包含相同的索引信息（重要）：

- 上文中我们提到根据IndexWriter判断索引是否发生变化需要依次判断四个条件，我们考虑这么一种情况，如果满足条件一，但是不满足条件二、条件三、条件四中的一个或多个，这种情况属于调用openIfChange()方法的线程A并未修改索引信息，其他线程正在修改索引信息，如果其他线程执行更改索引的操作并没有成功更改索引内容（例如执行删除文档的操作，然后所有段中的文档没有一篇满足删除的条件，或者添加文档失败等等），意味着segmentInfos的版本号Version不会更改，那么更改操作结束后，线程一重新获得的StandardDirectoryReader跟oldReader包含相同的索引信息仍旧相同的，那么此时返回null

结语

至此我们介绍完了图1中的所有流程，由于图2中的流程点与图1中的流程点是重复的，所以不对图2的流程展开介绍，那么Lucene提供的性能更高获得StandardDirectoryReader对象的openIfChange()方法就都介绍结束了。

在下一篇文章中，我们将会继续DirectoryReader的其他子类（见近实时搜索NRT（一）的图1）。

点击下载附件