构造IndexWriter对象（一）

该系列文章将会介绍构造一个IndexWriter对象的流程，该流程总体分为下面三个部分：

- 设置索引目录Directory
- 设置IndexWriter的配置信息IndexWriterConfig
- 调用IndexWriter的构造函数

设置索引目录Directory

Directory用来维护索引目录中的索引文件，定义了`创建`、`打开`、`删除`、`读取`、`重命名`、`同步`(持久化索引文件至磁盘)、`校验和`（checksum computing）等抽象方法，索引目录中不存在多级目录，即不存在子文件夹的层次结构(no sub-folder hierarchy)，另外Directory的具体内容已经在Directory系列文章中介绍，这里不赘述。

设置IndexWriter的配置信息IndexWriterConfig

在调用IndexWriter的构造函数之前，我们需要先初始化IndexWriter的配置信息IndexWriterConfig，IndexWriterConfig中的配置信息按照可以分为两类：

- 不可变配置（unmodifiable configuration）：在实例化IndexWriter对象后，这些配置不可更改，即使更改了，也不会生效，因为仅在IndexWriter的构造函数中应用一次这些配置
- 可变配置（modifiable configuration）：在实例化IndexWriter对象后，这些配置可以随时更改

不可变配置

不可变配置包含的内容有：OpenMode、IndexDeletionPolicy、IndexCommit、Similarity、MergeScheduler、Codec、DocumentsWriterPerThreadPool、ReaderPooling、FlushPolicy、RAMPerThreadHardLimitMB、InfoStream、IndexSort、SoftDeletesField，下面我们一一介绍这些不可变配置。

OpenMode

OpenMode描述了在IndexWriter的初始化阶段，如何处理索引目录中的已有的索引文件，这里称之为旧的索引，OpenMode一共定义了三种模式，即：CREATE、APPEND、CREATE_OR_APPEND。

- CREATE：如果索引目录中已经有旧的索引（根据Segment_N文件读取旧的索引信息），那么会覆盖（Overwrite）这些旧的索引，但注意的是新的提交（commit）生成的Segment_N的N值是旧索引中最后一次提交生成的Segment_N的N值加一后的值，如下所示：

图1：

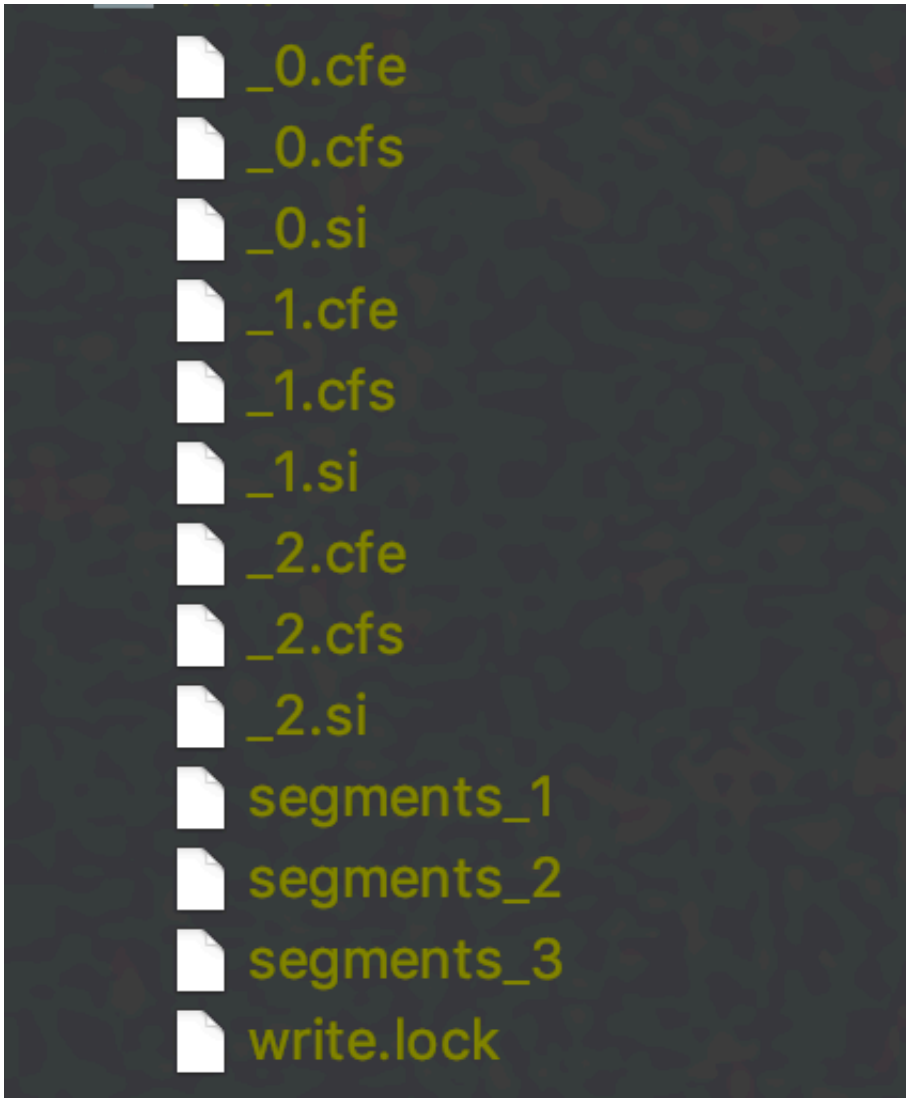
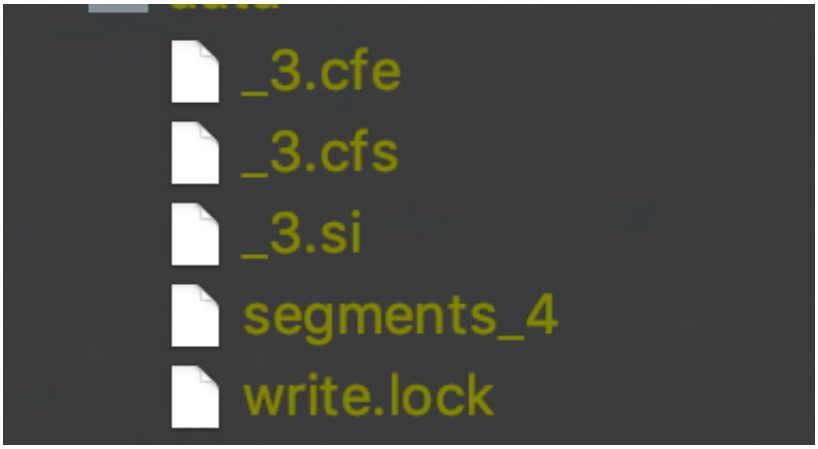


图1为索引目录中的旧的索引，并且有三个Segment_N文件，即segments_1、segments_2、segments_3。

图2：



接着我们通过CREATE打开图1的索引目录，并且执行了一次commit操作后，可以看出旧的索引信息被删除了（_0.cfe、_0.cfs、_0.si、_1.cfe、_1.cfs、_1.si、_2.cfe、_2.cfs、_2.si被删除了），并且新的提交（commit）生成的Segment_N（segment_4）的N值是旧索引中最后一次提交生成的Segment_N（segment_3）的N值加一后的值。

- APPEND：该打开模式打开索引目录会先读取索引目录中的旧索引，新的提交操作不会删除旧的索

引，注意的是如果索引目录没有旧的索引（找不到任何的Segment_N文件），并且使用当前模式打开则会报错，报错信息如下：

```
throw new IndexNotFoundException("no segments* file found in " + directory + ":
files: " + Arrays.toString(files));
```

上述的异常中，directory即上文提到的索引目录Directory，而Arrays.toString(files)用来输出索引目录中的所有文件。

- CREATE_OR_APPEND：该打开模式会先判断索引目录中是否有旧的索引，如果存在旧的索引，那么相当于APPEND模式，否则相当于CREATE模式

OpenMode可以通过IndexWriterConfig.setOpenMode(OpenMode openMode)方法设置，默认值为CREATE_OR_APPEND。

IndexDeletionPolicy

IndexDeletionPolicy是索引删除策略，该策略用来描述当一个新的提交生成后，如何处理上一个提交，在文档提交之commit（二）的文章中详细介绍了几种索引删除策略，这里不赘述。

IndexCommit

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，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。

如果设置了IndexCommit，那么在构造IndexWriter对象期间，会先读取IndexCommit中的索引信息，IndexCommit可以通过IndexWriterConfig.setIndexCommit(IndexCommit commit)方法设置，默认值为null。

另外IndexCommit的更多的用法见近实时搜索NRT系列文章。

Similarity

Similarity描述了Lucene打分的组成部分，在查询原理系列文章中详细介绍了Lucene如何使用BM25算法实现对文档的打分，这里不赘述。

Similarity可以通过IndexWriterConfig.setSimilarity(Similarity similarity)方法设置，默认使用BM25。

MergeScheduler

MergeScheduler即段的合并调度策略，用来定义如何执行一个或多个段的合并，比如并发执行多个段的合并任务时的执行先后顺序，磁盘IO限制，Lucene7.5.0中提供了三种可选的段的合并调度策略，见文章MergeScheduler。

MergeScheduler可以通过IndexWriterConfig.setMergeScheduler(MergeScheduler mergeScheduler)方法设置，默认使用ConcurrentMergeScheduler。

Codec

Codec定义了索引文件的数据结构，即描述了每一种索引文件需要记录哪些信息，以及如何存储这些信息，在索引文件的专栏中介绍了所有索引文件的数据结构，这里不赘述。

Codec可以通过IndexWriterConfig.setCodec(Codec codec)方法设置，在Lucene7.5.0版本中，默认使用Lucene70Codec。

DocumentsWriterPerThreadPool

DocumentsWriterPerThreadPool是一个逻辑上的线程池，它实现了类似Java线程池的功能，在Java的线程池中，新来的一个任务可以从ExecutorService中获得一个线程去处理该任务，而在DocumentsWriterPerThreadPool中，每当IndexWriter要添加文档，会从DocumentsWriterPerThreadPool中获得一个ThreadState去执行，故在多线程（持有相同的IndexWriter对象引用）执行添加文档操作时，每个线程都会获得一个ThreadState对象，DocumentsWriterPerThreadPool以及ThreadState的具体介绍可以看文章文档的增删改（中），这里不赘述。

如果不是深度使用Lucene，应该不会去调整这个配置吧。。。

ReaderPooling

ReaderPooling该值是一个布尔值，用来描述是否允许共用（pool）SegmentReader，共用（pool）可以理解为缓存，在第一次读取一个段的信息时，即获得该段对应的SegmentReader，并且使用ReaderPool（见执行段的合并（二）中关于ReaderPool的介绍）来缓存这些SegmentReader，使得在处理删除信息（删除操作涉及多个段时效果更突出）、NRT搜索时可以提供更好的性能，至于为什么共用/缓存SegmentReader能提高性能见文章SegmentReader（一）。

ReaderPooling可以通过IndexWriterConfig.setReaderPooling(boolean readerPooling)方法设置，默认值为true。

FlushPolicy

FlushPolicy即flush策略，准确的说应该称为 自动flush策略，因为flush分为自动flush跟主动flush（见文档提交之flush（一）），即显式调用IndexWriter.flush( )方法，FlushPolicy描述了IndexWriter执行了增删改的操作后，将修改后的索引信息写入磁盘的时机（实际是存入磁盘缓存，见文档提交之commit（一）中关于`执行同步磁盘工作`的介绍）。

Lucene7.5.0版本中，有且仅有一个FlushPolicy：FlushByRamOrCountsPolicy，可以看文章文档的增删改（中）中关于FlushByRamOrCountsPolicy的详细介绍。

RAMPerThreadHardLimitMB

该配置在后面介绍可变配置中的MaxBufferedDocs、RAMBufferSizeMB时一起介绍。

InfoStream

InfoStream用来在对Lucene进行调试时实现debug输出信息，在业务中打印debug信息会降低Lucene的性能，故在业务中使用默认值就行，即不输出debug信息。

InfoStream可以通过IndexWriterConfig.setInfoStream(InfoStream infoStream)方法设置，默认值为NoOutput。

IndexSort

IndexSort描述了在索引阶段如何对segment内的文档进行排序，排序的好处及其实现方式见文章Collector（三）中的**预备知识**及文章文档提交之flush（三）中的**sortMap**。

IndexSort可以通过IndexWriterConfig.setIndexSort(Sort sort)方法设置，默认值为null。

SoftDeletesField

SoftDeletesField用来定义哪些域为软删除的域，关于软删除的概念在后面的文章中会用多篇文章的篇幅介绍，这里暂不展开。

IndexSort可以通过IndexWriterConfig.setSoftDeletesField(String softDeletesField)方法设置，默认值为null。

可变配置

可变配置包含的内容有：MergePolicy、MaxBufferedDocs、RAMBufferSizeMB、MergedSegmentWarmer、UseCompoundFile、CommitOnClose、CheckPendingFlushUpdate。

结语

基于篇幅，我们将在下一篇文章中介绍可变配置的内容。

点击下载附件