Apache Lucene 分析

李克西 Krse Lee
21551081
浙江大学软件学院

2016.1.4

目录

Lucene

开发者	Apache Software Foundation
稳定版本	5.3.0（2015年8月24日，4个月前）
开发状态	活跃
编程语言	Java
操作系统	Cross-platform
类型	搜索及全文检索
许可协议	Apache许可证 2.0
网站	lucene.apache.org

Part 1 Apache Lucene

Apache Lucene

- Lucene 是一套用于全文检索的开源程序库，由Apache软件基金会支持和提供。
- 功能：全文索引和检索
- 编写语言：Java
- 衍生项目：Mahout，Solr

使用Lucene检索

药生产废水的处理方法	C02F9/08(2006.01)I	C02F9/08(2006.01)I;C02F103/36(2006.01)N	珠海保税区丽珠合成制药		
测的胶体金共价标记方法	G01N33/532(2006.01)I	G01N33/532(2006.01)I	珠海丽珠试剂股份有限公司	曾敏霞;李重	
端电气装配接地装置	H02B1/16(2006.01)I	H02B1/16(2006.01)I	珠海许继电气有限公司	汤定阳;魏浩铭;胡兵;羊	
传动结构	H01H33/666(2006.01)I	H01H33/666(2006.01)I	珠海许继电气有限公司	周斌;黄楷涛;黄冬喜	2015.05.2
型控制小室	H02B1/052(2006.01)I	H02B1/052(2006.01)I	珠海许继电气有限公司	周斌;黄楷涛;刘明清	2015.05.2
GPRS模块结构	H04W88/02(2009.01)I	H04W88/02(2009.01)I	珠海许继电气有限公司	张维;孙玮	2015.05.20
端电气装配接地装置	H02B1/16(2006.01)I	H02B1/16(2006.01)I	珠海许继电气有限公司	汤定阳;魏浩铭;胡兵;羊	
无线公网通信系统的状态监测方法	H02J13/00(2006.01)I	H02J13/00(2006.01)I	珠海许继电气有限公司	潘希;田巍巍	
构的储液器	F25B43/00(2006.01)I	F25B43/00(2006.01)I	珠海华宇金属有限公司	刘富强;廖鹏飞;柳林林	
B01D50/00(2006.01)I	B01D50/00(2006.01)I	珠海金鸡化工有限公司	蔡杨勇;冯明东;周建新;余军生;黄松贵		
面碗纸及其生产工艺	D21H11/10(2006.01)I	D21H11/10(2006.01)I;D21H11/04(2006.01)I	珠海华丰纸业有限公司	季	
饲料及其应用	A23K1/18(2006.01)I	A23K1/18(2006.01)I;A23K1/10(2006.01)I;A23K1/14(2006.01)I;A23K1/16			
南美白对虾配合饲料	A23K1/18(2006.01)I	A23K1/18(2006.01)I;A23K1/10(2006.01)I;A23K1/14(2006.01)I;			
纯化的方法	C07K14/59(2006.01)I	C07K14/59(2006.01)I;C07K1/34(2006.01)I	上海丽珠制药有限公司	万龙岩;陈岚	
锡导烟罩结构	B08B15/04(2006.01)I	B08B15/04(2006.01)I;C23C2/00(2006.01)I	珠海中精机械有限公司	周兴和	2
的锡叉压线结构	B25B11/00(2006.01)I	B25B11/00(2006.01)I	珠海中精机械有限公司	周兴和	2013.09.11
锡导烟罩结构	C23C2/08(2006.01)I	C23C2/08(2006.01)I;C23C2/38(2006.01)I	珠海中精机械有限公司	周兴和	2
保健配合饲料	A23K1/18(2006.01)I	A23K1/18(2006.01)I;A23K1/10(2006.01)I;A23K1/14(2006.01)I;A23K1/16			
饲料及其应用	A23K1/18(2006.01)I	A23K1/18(2006.01)I;A23K1/16(2006.01)I	珠海海龙生物科技有限公司	负	
用配合饲料及其制备方法	A23K1/04(2006.01)I	A23K1/04(2006.01)I;A23K1/10(2006.01)I;A23K1/16(2006.01)I;			

使用Lucene

```
public void testLuceneSearch(){
    Properties prop = new Properties();

    String[] areas = AreasInfo.getAreasArray();

    List<String> areaslist = Arrays.asList(areas);

    prop.put("--indexpath", "index/testIndex");
    prop.put("--threshold", 0.8);
    prop.put("--tops", 80000);
    prop.put("--target", "apply_man");
    prop.put("--docs", "true");
    prop.put("--areas", areaslist);
    prop.put("--key", "珠海");
    try {
        Map<String,Object> map = lucene.doSearch(prop);
        List<List> list = (List<List>) map.get("result");
        int count = 0;
        for(List<String> l : list){
            System.out.print(prop.get("total")+"\t"+count++ + "\t");
            for(String s : l ){
                System.out.print(s+"\t");
            }
            System.out.println();
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

使用Lucene（续）

```
public Map<String, Object> doSearch(Properties prop) {
    Map<String, Object> result = null;
    try {
        Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_47);
        QueryParser parser = new QueryParser(Version.LUCENE_47, (String) prop.get("--target"), analyzer);
        Query query = parser.parse((String) prop.get("--key"));
        // 检索
        result = search(query, prop);

    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

使用Lucene（续）

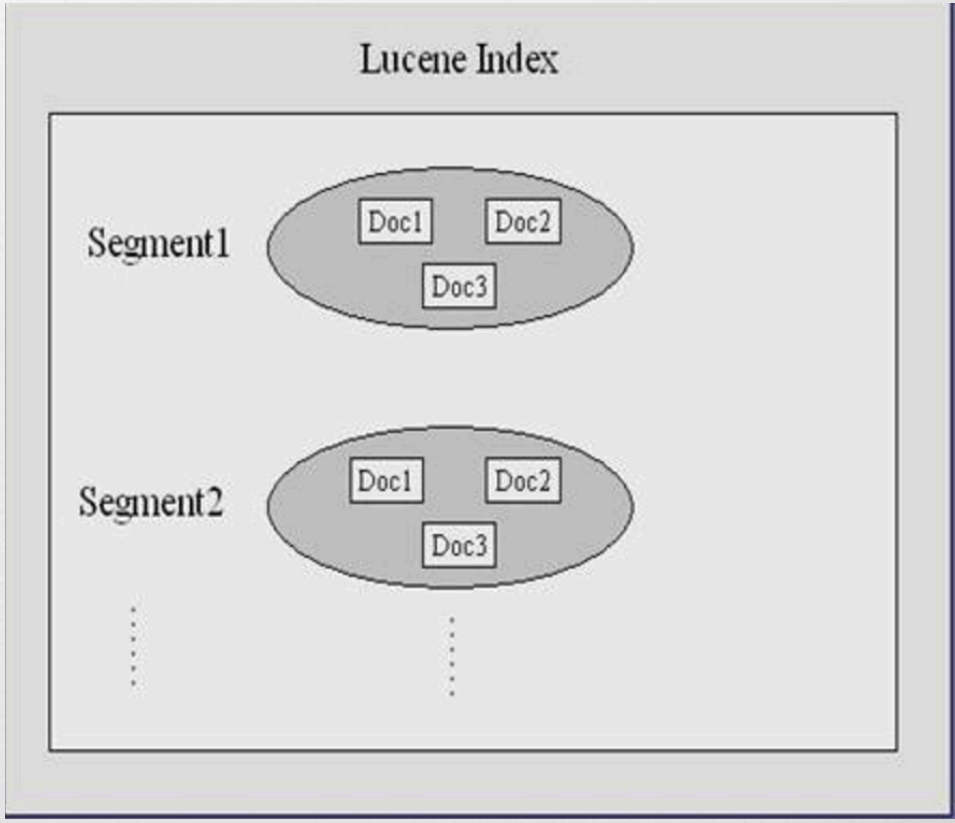
```
private Map<String, Object> search(Query query, Properties prop) throws Exception {
    Directory dire = FSDirectory.open(new File((String) prop.get("--indexpath")));
    IndexReader ir = DirectoryReader.open(dire);
    IndexSearcher is = new IndexSearcher(ir);
    Map<String, Object> result = new HashMap<String, Object>();

    double threshold = (double) prop.get("--threshold");
    int topNums = (int) prop.get("--tops");
    List<String> areas = (List<String>) prop.get("--areas");
    TopDocs td = is.search(query, null, topNums);
    // 将总文档数目返回
    result.put("total", td.totalHits);

    ScoreDoc[] sds = td.scoreDocs;
    Document d;
    @SuppressWarnings("rawtypes")
    List<List> docList = new LinkedList<List>();

    for (ScoreDoc sd : sds) {
        d = is.doc(sd.doc);
        if (sd.score > threshold) {
            List<String> infos = new LinkedList<String>();
            for (String area : areas) {
                infos.add(d.get(area));
            }
            docList.add(infos);
        }
    }
    ir.close();
    dire.close();

    result.put("result", docList);
    return result;
}
```

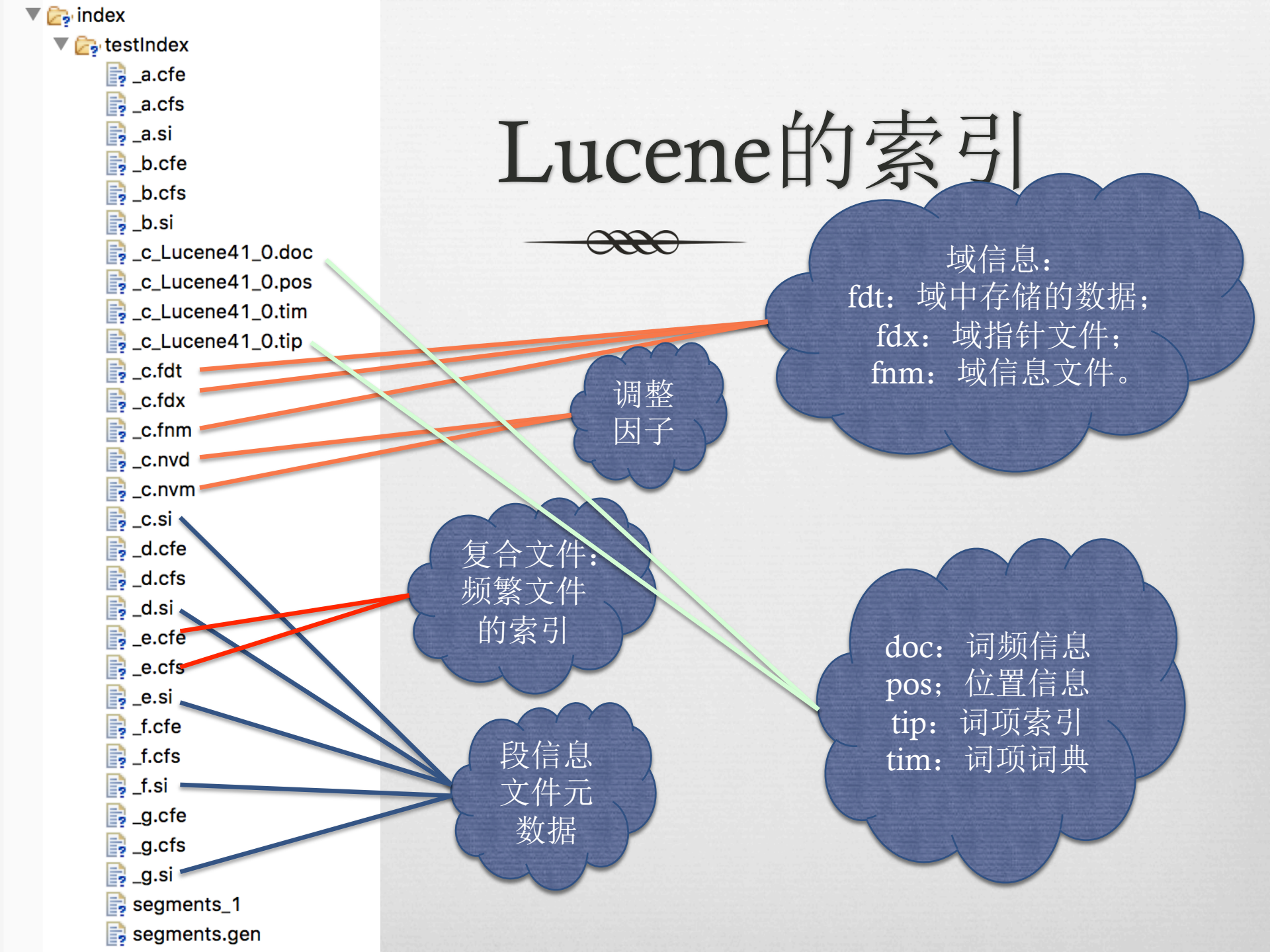
Part 2 Lucene Index过程

概述
IndexWriter的构建
添加文档到索引
缓存控制机制
关闭IndexWriter

Lucene索引层次

- **索引**：对文档集合建立的倒排索引，由段组成；
- **段**：组成索引的单元，由文档组成，段与段相互独立，可以合并；
- **文档**：索引的基本单位，新添加的文档单独保存在一个段中，随着段的合并，不同的文档会合并到一个段中。
- 域：文档的组成部分，如标题、作者、时间、正文等




index
testIndex
_a.cfe
_a.cfs
_a.si
_b.cfe
_b.cfs
_b.si
_c_Lucene41_0.doc
_c_Lucene41_0.pos
_c_Lucene41_0.tim
_c_Lucene41_0.tip
_c.fdt
_c.fdx
_c.fnm
_c.nvd
_c.nvm
_c.si
_d.cfe
_d.cfs
_d.si
_e.cfe
_e.cfs
_e.si
_f.cfe
_f.cfs
_f.si
_g.cfe
_g.cfs
_g.si
segments_1
segments.gen
Lucene的索引
域信息：
fdt：域中存储的数据；
fdx：域指针文件；
fnm：域信息文件。
调整因子
复合文件：频繁文件的索引
段信息文件元数据
doc：词频信息
pos；位置信息
tip：词项索引
tim：词项词典

Index模块的任务

- 在磁盘上创建索引文件
- 将文档内容添加到索引中去
- 将可以合并的段进行合并
- 保证线程安全

- 流程：将文档映射成Document对象，添加到内存缓冲区的索引结构中，如果缓冲区满，就将缓冲区的索引写入磁盘；所有的文档添加完后，将缓冲区中剩余的部分写入磁盘。

使用Lucene建立索引

```
/**************************************************************************
 * @param prop
 *             --indexpath index的存储路径;
 *             --filelist 需要建立索引的文档路径列表;
 *             --areas 信息域集合;
 *             --ex 分割域之间的分隔符;
 *************************************************************************/
public void createIndex(Properties prop) throws Exception {
    Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_47);
    Directory dire = FSDirectory.open(new File((String) prop.get("--indexpath")));
    IndexWriterConfig iwc = new IndexWriterConfig(Version.LUCENE_47, analyzer);
    IndexWriter iw = new IndexWriter(dire, iwc);
    System.out.println(">>>create index start");
    addFiles(iw, prop);

    iw.close();
}
```

使用Lucene建立索引（续）

```
private void addFile(IndexWriter iw, File file, Properties prop) throws Exception {

    FileInputStream fis = new FileInputStream(file);
    InputStreamReader isr = new InputStreamReader(fis);
    BufferedReader br = new BufferedReader(isr);

    String line = null;
    List<String> areas = (List<String>)prop.get("--areas");
    String ex = (String)prop.get("--ex");
    int index = 0;
    while ((line = br.readLine()) != null) {

        Document doc = new Document();
        String str[] = line.split(ex);
        if(areas.size()!= str.length) continue;
        System.out.println(index+"\t"+str[2]);
        for(int i=0;i<areas.size();i++){
            doc.add(new Field(areas.get(i), str[i], TextField.TYPE_STORED));
        }

        index++;
        iw.addDocument(doc);
    }
    br.close();
    isr.close();
    fis.close();
    iw.commit();
}
```

构建Index Writer

Index步骤

- 构建解析器
- 设置索引路径
- 构建IndexWriter
- 构建文档Document并添加到IndexWriter
- 提交IndexWriter
- 关闭IndexWriter

IndexWriter

- 任务：将Document写入到索引文件中
- 安全性：保证线程安全
- 接受参数：分析器Analyser，索引路径Directory
- 构造过程：
  - 解析参数
  - 申请锁
  - 读取或创建索引
  - 创建段和索引提交器IndexCommit
  - 创建DocumentsWriter、IndexFileDeleter
  - 释放锁

构造IndexWriter：解析参数

```
public IndexWriter(Directory d, IndexWriterConfig conf) throws IOException {
  conf.setIndexWriter(this); // prevent reuse by other instances
  config = new LiveIndexWriterConfig(conf);
  directory = d;
  analyzer = config.getAnalyzer();
  infoStream = config.getInfoStream();
  mergePolicy = config.getMergePolicy();
  mergePolicy.setIndexWriter(this);
  mergeScheduler = config.getMergeScheduler();
  codec = config.getCodec();

  bufferedUpdatesStream = new BufferedUpdatesStream(infoStream);
  poolReaders = config.getReaderPooling();
```

analyzer: 解析器，用于文本的处理，如分词
directory: 索引路径
mergePolicy: 用于执行原始段合并操作（指定要合并的段，以及如何合并）
mergeScheduler: 段合并调度器，将mergePolicy指定的段进行合并
codec: 用于索引文件的编码和解码

IndexWriter构造：申请锁、创建或打开索引

```
writeLock = directory.makeLock(WRITE_LOCK_NAME);

if (!writeLock.obtain(config.getWriteLockTimeout())) // obtain write lock
  throw new LockObtainFailedException("Index locked for write: " + writeLock);

boolean success = false;
try {
  OpenMode mode = config.getOpenMode();
  boolean create;
  if (mode == OpenMode.CREATE) {
    create = true;
  } else if (mode == OpenMode.APPEND) {
    create = false;
  } else {
    // CREATE_OR_APPEND - create only if an index does not exist
    create = !DirectoryReader.indexExists(directory);
  }
```

- 申请写锁来防止多线程脏写
- 如果索引存在就打开索引文件，不存在则创建索引

IndexWriter构造：创建段、索引提交器

```
// If index is too old, reading the segments will throw
// IndexFormatTooOldException.
segmentInfos = new SegmentInfos();

boolean initialIndexExists = true;

if (create) {
  // Try to read first.  This is to allow create
  // against an index that's currently open for
  // searching.  In this case we write the next
  // segments_N file with no segments:
  try {
    segmentInfos.read(directory);
    segmentInfos.clear();
  } catch (IOException e) {
    // Likely this means it's a fresh directory
    initialIndexExists = false;
  }

  // Record that we have a change (zero out all
  // segments) pending:
  changed();
} else {
  segmentInfos.read(directory);
```

创建段，段是索引的基本组成单元，将索引写入磁盘是通过将一个个段写入磁盘来完成的

```
    IndexCommit commit = config.getIndexCommit();
    if (commit != null) {
      // Swap out all segments, but, keep metadata in
      // SegmentInfos, like version & generation, to
      // preserve write-once.  This is important if
      // readers are open against the future commit
      // points.
      if (commit.getDirectory() != directory)
        throw new IllegalArgumentException("IndexCommit's direc
      SegmentInfos oldInfos = new SegmentInfos();
      oldInfos.read(directory, commit.getSegmentsFileName());
      segmentInfos.replace(oldInfos);
      changed();
      if (infoStream.isEnabled("IW")) {
        infoStream.message("IW", "init: loaded commit \"" + com
      }
    }
  }

  rollbackSegments = segmentInfos.createBackupSegmentInfos();
```

创建索引提交器和回滚用的备份段

IndexWriter构造：创建DocumentWriter

```
// start with previous field numbers, but new FieldInfos
globalFieldNumberMap = getFieldNumberMap();
config.getFlushPolicy().init(config);
docWriter = new DocumentsWriter(this, config, directory);
eventQueue = docWriter.eventQueue();
```

- IndexWriter添加文档后依靠DocumentWriter的方法将文档添加到索引中。

IndexWriter构造：创建索引删除器

```
// Default deleter (for backwards compatibility) is
// KeepOnlyLastCommitDeleter:
synchronized(this) {
  deleter = new IndexFileDeleter(directory,
                                 config.getIndexDeletionPolicy(),
                                 segmentInfos, infoStream, this,
                                 initialIndexExists);
}

if (deleter.startingCommitDeleted) {
  // Deletion policy deleted the "head" commit point.
  // We have to mark ourself as changed so that if we
  // are closed w/o any further changes we write a new
  // segments_N file.
  changed();
}

if (infoStream.isEnabled("IW")) {
  infoStream.message("IW", "init: create=" + create);
  messageState();
}
```

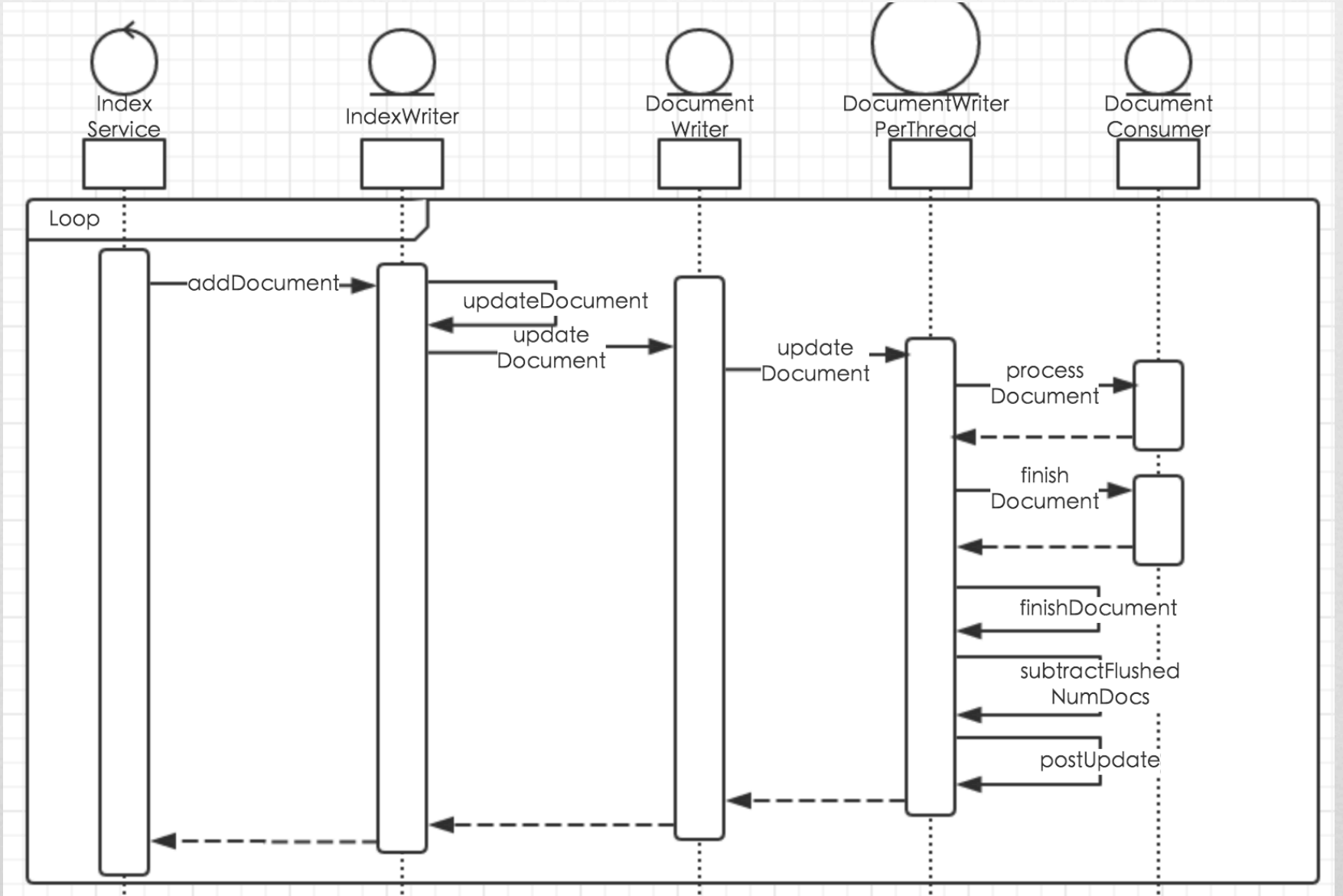
- 告诉IndexFileDeleter索引路径、段信息等信息
- 启动deleter

IndexWriter构造：完成构造，释放锁

```
    success = true;

  } finally {
    if (!success) {
      if (infoStream.isEnabled("IW")) {
        infoStream.message("IW", "init: hit exception on init; releasing write lock");
      }
      IOUtils.closeWhileHandlingException(writeLock);
      writeLock = null;
    }
  }
```

Lucene Index
添加文档到索引

IndexWriter添加文档过程



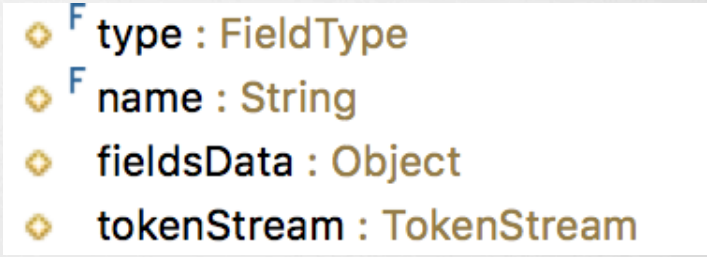
Document类

- 一篇文档对应一个Document，结构化的文本数据也可以是一个结构体对应一个Document。
- fields：文档的域集合，通过操作这个List来增加、删除、获取一个个域


```
while ((line = br.readLine()) != null) {

    Document doc = new Document();
    String str[] = line.split(ex);
    if(areas.size()!= str.length) continue;
    System.out.println(index+"\t"+str[2]);
    for(int i=0;i<areas.size();i++){
        doc.add(new Field(areas.get(i), str[i], TextF
    }

    index++;
    iw.addDocument(doc);
}
```

Field类

- 实现接口：IndexableField，组成Document类。
- 主要成员变量：
  - type：域类型（Store或者Index）
  - name：域名称（标题、作者、时间、正文、结束语）
  - fieldsData：域值，一般为String
  - tokenStream：可选，是通过analyzer分词后得到的单词流。

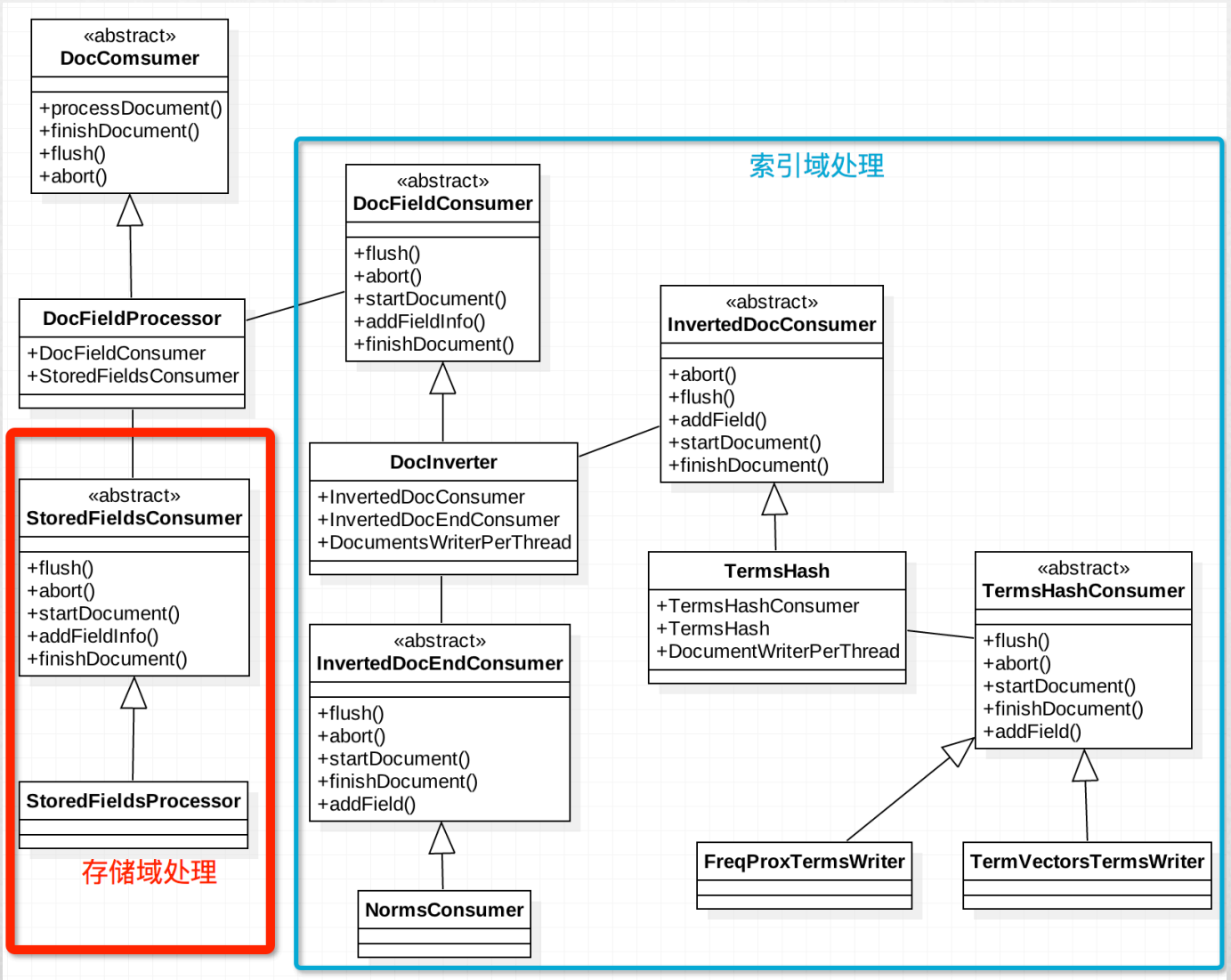


DocConsumer

```
abstract class DocConsumer {
  abstract void processDocument(FieldInfos.Builder fieldInfos) throws IOException;
  abstract void finishDocument() throws IOException;
  abstract void flush(final SegmentWriteState state) throws IOException;
  abstract void abort();
}
```

- DocConsumer是将文档最终写入到缓存中的实体，其实现类是DocFieldProcesser
- 将文档写入索引并不是由DocConsumer单独完成的，而是一系列链式的对象共同承担

Consumer链式对象



各个Consumer的作用

- DocConsumer <= DocFieldProcessor
  - DocFieldConsumer <= DocInverter
    - InvertedDocConsumer <= TermsHash
      - TermsHashConsumer <= FreqProxTermsWriter

        负责写freq 、 prox 信息
      - TermsHashConsumer <= TermVectorsTermsWriter

        负责写tvx、tvd、tvf信息
    - InvertedDocEndConsumer <= NormsWriter

      负责写nrm信息
  - StoredFieldsConsumer <= StoredFieldsProcessor

    负责写段信息si

缓存控制

什么时候将缓存索引写入磁盘？

- 为了提高索引的速度,Lucene 对很多的数据进行了缓存,使一起写入磁盘（解决高速设备与低速设备不匹配问题）。
- 每次向缓冲区索引添加一个Document对象时，都会检查缓冲区大小，如果触发阈值，就会将缓冲区内容写入磁盘。

什么时候将缓存索引写入磁盘？（续）

- DocumentWriter.updateDocuments( )

```
boolean updateDocuments(final Iterable<? extends Iterable<? extends IndexableField>> docs, final An
                        final Term delTerm) throws IOException {
  boolean hasEvents = preUpdate();

  final ThreadState perThread = flushControl.obtainAndLock();
  final DocumentsWriterPerThread flushingDWPT;

  try {
    if (!perThread.isActive()) {
      ensureOpen();
      assert false: "perThread is not active but we are still open";
    }
    ensureInitialized(perThread);
    assert perThread.isInitialized();
    final DocumentsWriterPerThread dwpt = perThread.dwpt;
    final int dwptNumDocs = dwpt.getNumDocsInRAM();
    try {
      final int docCount = dwpt.updateDocuments(docs, analyzer, delTerm);
      numDocsInRAM.addAndGet(docCount);
    } finally {
      if (dwpt.checkAndResetHasAborted()) {
        if (!dwpt.pendingFilesToDelete().isEmpty()) {
          putEvent(new DeleteNewFilesEvent(dwpt.pendingFilesToDelete()));
        }
        subtractFlushedNumDocs(dwptNumDocs);
        flushControl.doOnAbort(perThread);
      }
    }
    final boolean isUpdate = delTerm != null;
    flushingDWPT = flushControl.doAfterDocument(perThread, isUpdate);
  } finally {
    perThread.unlock();
  }

  return postUpdate(flushingDWPT, hasEvents);
}
```

这两个函数都会检查缓冲区文件大小是否达到阈值

什么时候将缓存索引写入磁盘？（续）

- DocumentWriter.preUpdate( )

```
private boolean preUpdate() throws IOException {
  ensureOpen();
  boolean hasEvents = false;
  if (flushControl.anyStalledThreads() || flushControl.numQueuedFlushes() > 0) {
    // Help out flushing any queued DWPTs so we can un-stall:
    if (infoStream.isEnabled("DW")) {
      infoStream.message("DW", "DocumentsWriter has queued dwpt; will hijack this thread to flu
    }
    do {
      // Try pick up pending threads here if possible
      DocumentsWriterPerThread flushingDWPT;
      while ((flushingDWPT = flushControl.nextPendingFlush()) != null) {
        // Don't push the delete here since the update could fail!
        hasEvents |= doFlush(flushingDWPT);
      }

      if (infoStream.isEnabled("DW")) {
        if (flushControl.anyStalledThreads()) {
          infoStream.message("DW", "WARNING DocumentsWriter has stalled threads; waiting");
        }
      }

      flushControl.waitIfStalled(); // block if stalled
    } while (flushControl.numQueuedFlushes() != 0); // still queued DWPTs try help flushing

    if (infoStream.isEnabled("DW")) {
      infoStream.message("DW", "continue indexing after helping out flushing DocumentsWriter is
    }
  }
  return hasEvents;
}
```

什么时候将缓存索引写入磁盘？（续）

- DocumentWriter.postUpdate( )
- 可以看到，preUpdate和postUpdate都会通过调用doFlush方法获取响应事件hasEvents

```
private boolean postUpdate(DocumentsWriterPerThread flushingDWPT, boolean hasEvents) throws IO
  hasEvents |= applyAllDeletes(deleteQueue);
  if (flushingDWPT != null) {
    hasEvents |= doFlush(flushingDWPT);
  } else {
    final DocumentsWriterPerThread nextPendingFlush = flushControl.nextPendingFlush();
    if (nextPendingFlush != null) {
      hasEvents |= doFlush(nextPendingFlush);
    }
  }

  return hasEvents;
}
```

DocumentWriter.doFlush( )

- doFlush方法将DocumentWriterPerThread对象传给DocumentsWriterFlushControl，flushControl会将该线程写入的缓存信息进行累加。

```
        putEvent(ForcedPurgeEvent.INSTANCE);
        break;
      }
    } finally {
      flushControl.doAfterFlush(flushingDWPT);
      flushingDWPT.checkAndResetHasAborted();
    }

    flushingDWPT = flushControl.nextPendingFlush();
  }
  if (hasEvents) {
    putEvent(MergePendingEvent.INSTANCE);
  }
  // If deletes alone are consuming > 1/2 our RAM
  // buffer, force them all to apply now. This is to
  // prevent too-frequent flushing of a long tail of
  // tiny segments:
  final double ramBufferSizeMB = config.getRAMBufferSizeMB();
  if (ramBufferSizeMB != IndexWriterConfig.DISABLE_AUTO_FLUSH &&
      flushControl.getDeleteBytesUsed() > (1024*1024*ramBufferSizeMB/2)) {
    if (infoStream.isEnabled("DW")) {
      infoStream.message("DW", "force apply deletes bytesUsed=" + flushControl.
    }
    hasEvents = true;
```

DocumentsWriterFlushControl

- DocumentsWriterFlushControl用于控制缓存的刷新。

```
synchronized DocumentsWriterPerThread doAfterDocument(ThreadState perThread,
    boolean isUpdate) {
  try {
    commitPerThreadBytes(perThread);
    if (!perThread.flushPending) {
      if (isUpdate) {
        flushPolicy.onUpdate(this, perThread);
      } else {
        flushPolicy.onInsert(this, perThread);
      }
      if (!perThread.flushPending && perThread.bytesUsed > hardMaxBytesPerDWPT) {
        // Safety check to prevent a single DWPT exceeding its RAM limit. This
        // is super important since we can not address more than 2048 MB per DWPT
        setFlushPending(perThread);
      }
    }
```

```
synchronized void doOnDelete() {
  // pass null this is a global delete no update
  flushPolicy.onDelete(this, null);
}
```

什么时候将缓存索引写入磁盘？（续）

- FlushByRamOrCountsPolicy是Lucene的缓存刷新策略类，里面定义了文档添加时的刷新策略(onInsert)和文档删除时的刷新策略(onDelete)

```
@Override
public void onInsert(DocumentsWriterFlushControl control, ThreadState state) {
  if (flushOnDocCount()
      && state.dwpt.getNumDocsInRAM() >= indexWriterConfig
          .getMaxBufferedDocs()) {
    // Flush this state by num docs
    control.setFlushPending(state);
  } else if (flushOnRAM()) {// flush by RAM
    final long limit = (long) (indexWriterConfig.getRAMBufferSizeMB() * 1024.d * 1024.d);
    final long totalRam = control.activeBytes() + control.getDeleteBytesUsed();
    if (totalRam >= limit) {
      if (infoStream.isEnabled("FP")) {
        infoStream.message("FP", "flush: activeBytes=" + control.activeBytes() + " deleteB
      }
      markLargestWriterPending(control, state, totalRam);
    }
  }
}
```

什么时候将缓存索引写入磁盘？（续）

- FlushByRamOrCountsPolicy共有三种刷新机制：
  - flushOnDocCount：根据添加的文档数刷新
  - flushOnDeleteTerms：根据删除的文档书刷新
  - flushOnRAM：根据内存用量刷新

```
final long limit = (long) (indexWriterConfig.getRAMBufferSizeMB() * 1024.d * 1024.d);
final long totalRam = control.activeBytes() + control.getDeleteBytesUsed();
if (totalRam >= limit) {
  if (infoStream.isEnabled("FP")) {
    infoStream.message("FP", "flush: activeBytes=" + control.activeBytes() + " deleteBy
  }
  markLargestWriterPending(control, state, totalRam);
}
```

LucenWriter的关闭过程

关闭IndexWriter

- IndexWriter.close() =>
- IndexWriter.close(true) =>
- IndexWriter.closeInternal(boolean) =>
  - //将缓存中剩余的索引信息写入磁盘
  - flush(waitFormerges, true)
  - //进行段合并
  - mergeScheduler.merge(this)

```
public void createIndex(Pro
    Analyzer analyzer = new
    Directory dire = FSDire
    IndexWriterConfig iwc =
    IndexWriter iw = new In
    System.out.println(">>>
    addFiles(iw, prop);

    iw.close();
}
```

IndexWriter.close( )

- 目标：保证线程安全，将缓存中的索引信息写入磁盘
- 在这个过程中，如果内存溢出标志 hitOOM 为真，那么就需要终止关闭过程，进入异常处理函数rollbackInternal( )

```
public void close(boolean waitForMerges) throws IOException {

  // Ensure that only one thread actually gets to do the
  // closing, and make sure no commit is also in progress:
  synchronized(commitLock) {
    if (shouldClose()) {
      // If any methods have hit OutOfMemoryError, then abort
      // on close, in case the internal state of IndexWriter
      // or DocumentsWriter is corrupt
      if (hitOOM) {
        rollbackInternal();
      } else {
        closeInternal(waitForMerges, true);
      }
    }
    assert assertEventQueueAfterClose();
  }
}
```

谢谢！