Lucene.Net全文检索

目 录

[1 lucene简介 3](#_Toc31234685)

[1.1 什么是lucene 3](#_Toc31234686)

[1.2 lucene能做什么 3](#_Toc31234687)

[1.3 你该不该选择lucene 3](#_Toc31234688)

[1.4 Lucene.Net 版本及安装 4](#_Toc31234689)

[2 lucene的工作方式 4](#_Toc31234690)

[2.1写入流程 4](#_Toc31234691)

[2.2读出流程 4](#_Toc31234692)

[3 一些需要知道的概念 5](#_Toc31234693)

[3.1 analyzer 5](#_Toc31234694)

[3.2 document 6](#_Toc31234695)

[3.3 Field 6](#_Toc31234696)

[3.4 term 7](#_Toc31234697)

[3.5 Token 7](#_Toc31234698)

[3.6 Segment 7](#_Toc31234699)

[4 Lucene的结构 7](#_Toc31234700)

[4.1 analysis 7](#_Toc31234701)

[4.2 document 8](#_Toc31234702)

[4.3 index 8](#_Toc31234703)

[4.4 queryParser 8](#_Toc31234704)

[4.5 search 8](#_Toc31234705)

[4.6 store 9](#_Toc31234706)

[4.7 util 9](#_Toc31234707)

[5 分词 9](#_Toc31234708)

[5.1内置分词器 9](#_Toc31234709)

[5.2 中文分词——盘古分词 11](#_Toc31234710)

[6 索引 12](#_Toc31234711)

[6.1 索引的存储结构 12](#_Toc31234712)

[6.2 索引的写入目录Directory 13](#_Toc31234713)

[6.3索引文档Document 13](#_Toc31234714)

[7.索引维护 15](#_Toc31234715)

[6.1索引创建（重点） 15](#_Toc31234716)

[6.2索引删除 16](#_Toc31234717)

[6.2索引更新 17](#_Toc31234718)

[6.4索引优化 18](#_Toc31234719)

[8 查询语法 19](#_Toc31234720)

[8.1 TermQuery 20](#_Toc31234721)

[8.2 BooleanQuery 20](#_Toc31234722)

[8.3 WildcardQuery 21](#_Toc31234723)

[8.4 PhraseQuery 21](#_Toc31234724)

[8.5 PrefixQuery 21](#_Toc31234725)

[8.6 RangeQuery 21](#_Toc31234726)

[8.7 QueryParser 22](#_Toc31234727)

[9、Filter索引过滤 23](#_Toc31234728)

[10. Sort结果排序 24](#_Toc31234729)

[11 关键词高亮 25](#_Toc31234730)

[12 完整查询示例 26](#_Toc31234731)

## 1 lucene简介

### 1.1 什么是lucene

Lucene是一个全文搜索框架，而不是应用产品。因此它并不像www.baidu.com 或者google Desktop那样拿来就能用，它只是提供了一种工具让你能实现这些产品。

Lucene.Net是Lucene搜索引擎库的一个用C#语文开发的基于.net Frameword的全文搜索框架。

### 1.2 lucene能做什么

要回答这个问题，先要了解lucene的本质。实际上lucene的功能很单一，说到底，就是你给它若干个字符串，然后它为你提供一个全文搜索服务，告诉你 你要搜索的关键词出现在哪里。知道了这个本质，你就可以发挥想象做任何符合这个条件的事情了。你可以把站内新闻都索引了，做个资料库；你可以把一个数据库表的若干个字段索引起来，那就不用再担心因为“%like%”而锁表了；你也可以写个自己的搜索引擎……

### 1.3 你该不该选择lucene

下面给出一些测试数据，如果你觉得可以接受，那么可以选择。

测试一：250万记录，300M左右文本，生成索引380M左右，800线程下平均处理时间300ms。

测试二：37000记录，索引数据库中的两个varchar字段，索引文件2.6M，800线程下平均处理时间1.5ms。

### 1.4 Lucene.Net 版本及安装

Lucece.Net官网：<https://lucenenet.apache.org/index.html>

Lucene.Net安装

可以通过NuGet安装，目前Lucene.Net最新版本为4.8.0-beta00007，但需.NETStandard 2.1支持，通常选择下载量最大的稳定版本：3.0.3，可以通过以下NuGet命令安装：

Install-Package Lucene.Net -Version 3.0.3

## 2 lucene的工作方式

lucene提供的服务实际包含两部分：一入一出。所谓入是写入，即将你提供的源（本质是字符串）写入索引或者将其从索引中删除；所谓出是读出，即向用户提供全文搜索服务，让用户可以通过关键词定位源。

### 2.1写入流程

源字符串首先经过analyzer处理，包括：分词，分成一个个单词；去除stopword（可选）。

将源中需要的信息加入Document的各个Field中，并把需要索引的Field索引起来，把需要存储的Field存储起来。

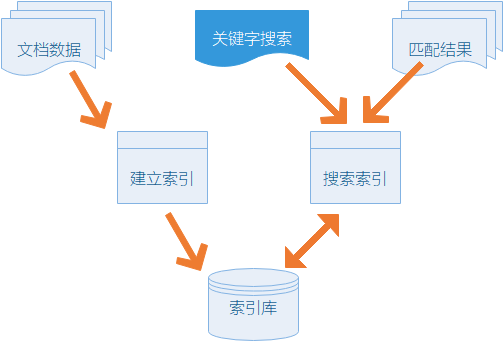
将索引写入存储器，存储器可以是内存或磁盘。

### 2.2读出流程

用户提供搜索关键词，经过analyzer处理。

对处理后的关键词搜索索引找出对应的Document。

用户根据需要从找到的Document中提取需要的Field。



## 3 一些需要知道的概念

lucene用到一些概念，了解它们的含义，有利于下面的讲解。

### 3.1 analyzer

Analyzer 是分析器，它的作用是把一个字符串按某种规则划分成一个个词语，并去除其中的无效词语，这里说的无效词语是指英文中的 “of”、 “the”，中文中的“的”、“地”等词语，这些词语在文章中大量出现，但是本身不包含什么关键信息，去掉有利于缩小索引文件、提高效率、提高命中率。

分词的规则千变万化，但目的只有一个：按语义划分。这点在英文中比较容易实现，因为英文本身就是以单词为单位的，已经用空格分开；而中文则必须以某种方法将连成一片的句子划分成一个个词语。具体划分方法下面再详细介绍，这里只需了解分析器的概念即可。

### 3.2 document

用户提供的源是一条条记录，它们可以是文本文件、字符串或者数据库表的一条记录等等。一条记录经过索引之后，就是以一个Document的形式存储在索引文件中的。用户进行搜索，也是以Document列表的形式返回。

### 3.3 Field

一个Document可以包含多个信息域，例如一篇文章可以包含“标题”、“正文”、“最后修改时间”等信息域，这些信息域就是通过Field在Document中存储的。

Field有两个属性可选：存储和索引。通过存储属性你可以控制是否对这个Field进行存储；通过索引属性你可以控制是否对该Field进行索引。这看起来似乎有些废话，事实上对这两个属性的正确组合很重要，下面举例说明：

还是以刚才的文章为例子，我们需要对标题和正文进行全文搜索，所以我们要把索引属性设置为真，同时我们希望能直接从搜索结果中提取文章标题，所以我们把标题域的存储属性设置为真，但是由于正文域太大了，我们为了缩小索引文件大小，将正文域的存储属性设置为假，当需要时再直接读取文件；我们只是希望能从搜索解 果中提取最后修改时间，不需要对它进行搜索，所以我们把最后修改时间域的存储属性设置为真，索引属性设置为假。上面的三个域涵盖了两个属性的三种组合，还 有一种全为假的没有用到，事实上Field不允许你那么设置，因为既不存储又不索引的域是没有意义的。

### 3.4 term

term是搜索的最小单位，它表示文档的一个词语，term由两部分组成：它表示的词语和这个词语所出现的field。

### 3.5 Token

token是term的一次出现，它包含trem文本和相应的起止偏移，以及一个类型字符串。一句话中可以出现多次相同的词语，它们都用同一个term表示，但是用不同的token，每个token标记该词语出现的地方。

### 3.6 Segment

添加索引时并不是每个document都马上添加到同一个索引文件，它们首先被写入到不同的小文件，然后再合并成一个大索引文件，这里每个小文件都是一个segment。

## 4 Lucene的结构

Lucene包括core和sandbox两部分，其中core是lucene稳定的核心部分，sandbox包含了一些附加功能，例如highlighter、各种分析器。

Lucene core有七个包：analysis，document，index，queryParser，search，store，util。

### 4.1 analysis

Analysis包含一些内建的分析器，例如按空白字符分词的WhitespaceAnalyzer，添加了stopwrod过滤的StopAnalyzer，最常用的StandardAnalyzer。

### 4.2 document

Document包含文档的数据结构，例如Document类定义了存储文档的数据结构，Field类定义了Document的一个域。

### 4.3 index

Index 包含了索引的读写类，例如对索引文件的segment进行写、合并、优化的IndexWriter类和对索引进行读取和删除操作的 IndexReader类，这里要注意的是不要被IndexReader这个名字误导，以为它是索引文件的读取类，实际上删除索引也是由它完成， IndexWriter只关心如何将索引写入一个个segment，并将它们合并优化；IndexReader则关注索引文件中各个文档的组织形式。

### 4.4 queryParser

QueryParser 包含了解析查询语句的类，lucene的查询语句和sql语句有点类似，有各种保留字，按照一定的语法可以组成各种查询。 Lucene有很多种Query类，它们都继承自Query，执行各种特殊的查询，QueryParser的作用就是解析查询语句，按顺序调用各种 Query类查找出结果。

### 4.5 search

Search包含了从索引中搜索结果的各种类，例如刚才说的各种Query类，包括TermQuery、BooleanQuery等就在这个包里。

### 4.6 store

Store包含了索引的存储类，例如Directory定义了索引文件的存储结构，FSDirectory为存储在文件中的索引，RAMDirectory为存储在内存中的索引，MmapDirectory为使用内存映射的索引。

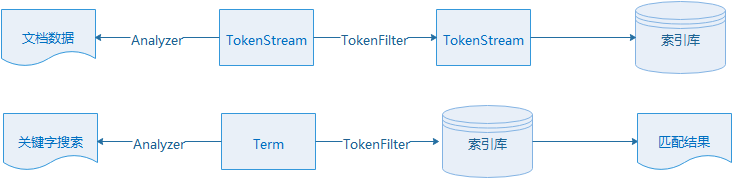
### 4.7 util

Util包含一些公共工具类，例如时间和字符串之间的转换工具。

## 5 分词

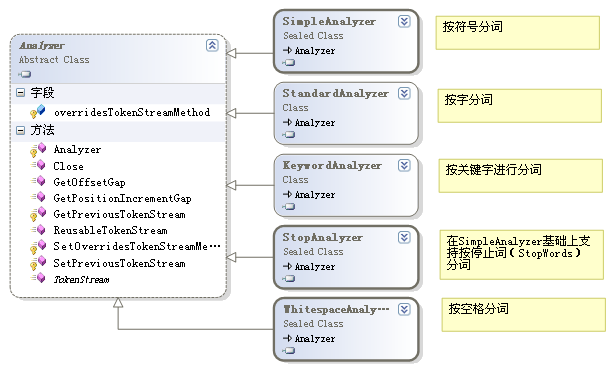
### 5.1内置分词器

分词（切词）是实现全文检索的基础，之所以我们能够让机器理解我们的自然语言，是因为有了分词的帮助。分词工作由Analyzer 类完成，它负责把文本切成Token序列，Token就是索引中的单词。Lucene.Net在两个地方用到分词：创建文档索引和分析搜索关键字。其过程示意如下：

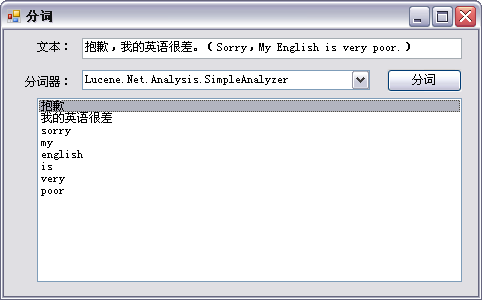
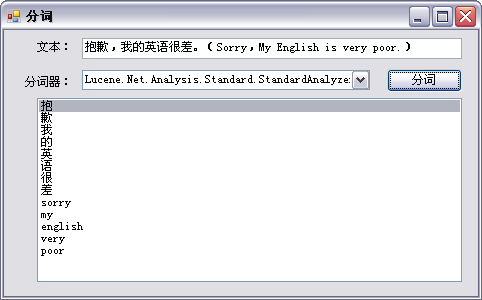


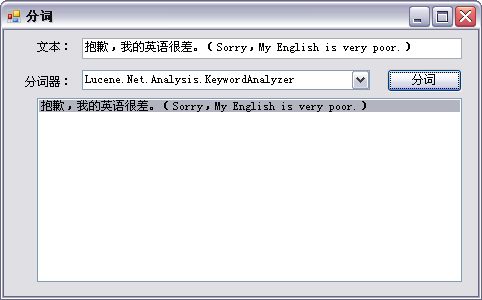
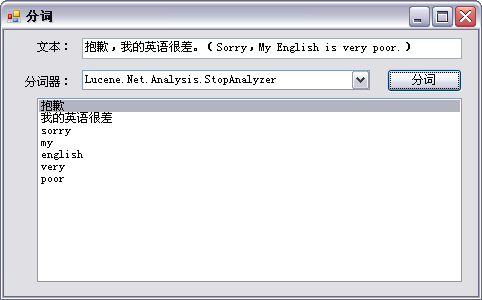
由此可知，在创建索引和搜索时，必须使用同样的分词器，保证其切出相同的Token才能检索到结果。

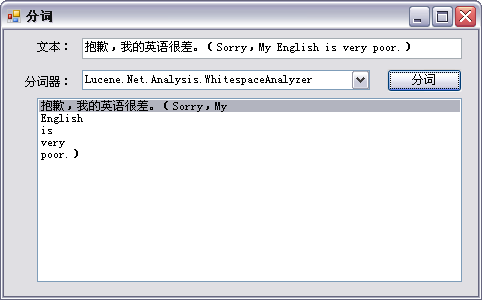
Lucene.Net实现了一些分词器，其对英文支持较好，但是对中文支持欠佳。



针对内置分词器测试结果如下：



### 5.2 中文分词——盘古分词

盘古分词是一个中英文分词组件。作者eaglet 曾经开发过KTDictSeg 中文分词组件，拥有大量用户。 作者基于之前分词组件的开发经验，结合最新的开发技术重新编写了盘古分词组件。

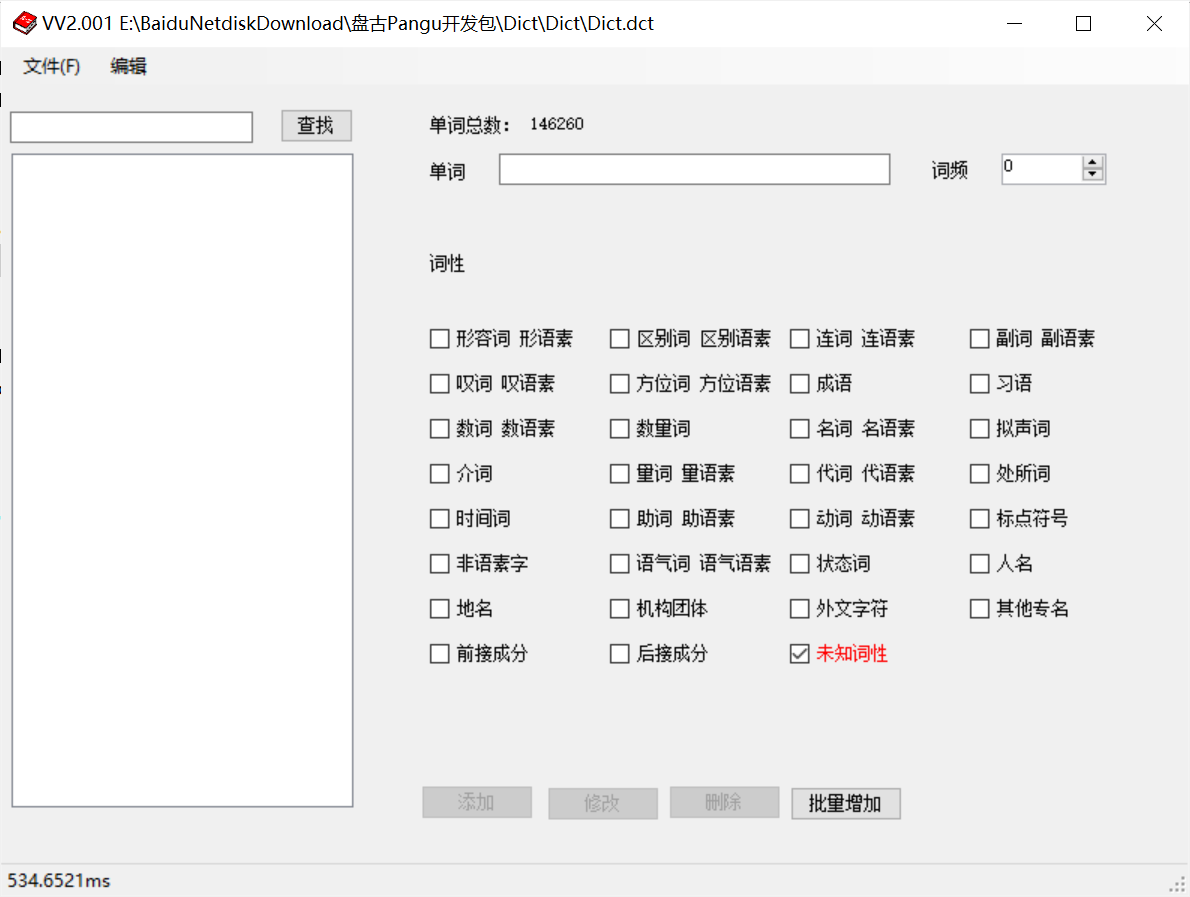
盘古分词(https://pangusegment.codeplex.com/ )，由于老版本不支持最新Lucene.Net 3.0.3，对其进行了升级，可以支持最新的Lucene.Net 3.0.3。可以直接NuGet安装。

盘古分词最新版本NuGet包安装命令：

Install-Package Lucene.Net.Analysis.PanGu -Version 2.4.1

盘古分词维护

目前网上流传资料较多的，是使用DictManage.exe管理和维护词库，词典文件为：Dict.Dct，如图所示：



使用NuGet安装最新版本的盘古分词后，分词词典打包到了Bin目录中，不能单独进行编辑维护，但最终版本的盘古分词在内部进行了判断处理，会先在项目的BIN目录中查找PanGu.xml和Dictionaries文件夹，其中PanGu.xml中配置了外部词典的路径，如果没有找到，则启用内部词典。

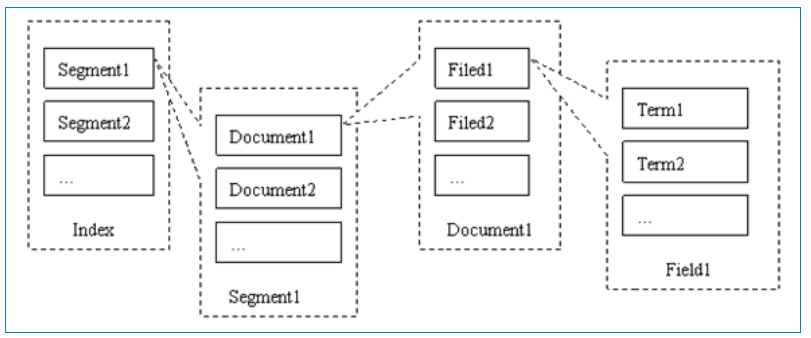
附：盘下分词开发包下载



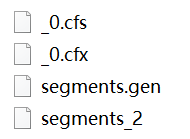
## 6 索引

### 6.1 索引的存储结构

为了方便索引大量文档，Lucene.Net中的一个索引包括多个子索引，叫做Segment（段）。每个Segment包括多个可搜索的文档，叫做Document；每个Document包括多个Field；每个Field又包括多个Term。综上所述，Lucene.Net的索 引文件的逻辑结构如下：

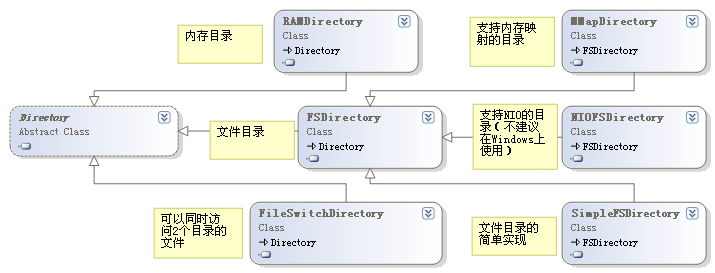


索引文件的物理表示如下：



其中后两个文件是段（Segment）的元数据文件。

### 6.2 索引的写入目录Directory



通常，可以选择将索引写入内存或磁盘上的某个文件目录，对应的类为：**Directory**

使用如下代码将索引写入文件目录：

Lucene.Net.Store.Directory d = FSDirectory.Open(new DirectoryInfo(path), new NativeFSLockFactory());

或使用如下代码将索引写入内存：

Lucene.Net.Store.Directory dir = new RAMDirectory();

### 6.3索引文档Document

包含了可索引文档的信息。每个Document都有一个编号，但该编号并非永远不变。

每个Document相当于数据库中的一条记录，将这条包含各个字段信息的记录写入内存或相应目录；

Field 类似实体的某个属性，就像数据库中的一个列，Field有两个属性可选：存储和索引。 通过存储属性你可以控制是否对这个Field进行存储； 通过索引属性你可以控制是否对该Field进行索引。

**Field.Store可选值**

Field.Store.YES:存储字段值（未分词前的字段值）

Field.Store.NO:不存储，存储与索引没有关系

Field.Store.COMPRESS:压缩存储，用于长文本或二进制，但性能受损

**Field.Index可选值**

Field.Index.ANALYZED:分词并建立索引

Field.Index.ANALYZED\_NO\_NORMS:分词并建立索引，但禁用标准化因子（Norms）。

Field.Index.NOT\_ANALYZE：在不使用分析器的情况下索引字段的值，以便可以对其进行搜索。由于没有使用分析器，因此该值将作为单个术语存储。这对于唯一的id（如产品编号）很有用。

Field.Index.NOT\_ANALYZED\_NO\_NORMS: 在不使用分析器的情况下索引字段值，并禁用存储标准化因子。因为标准化因子占用了一个byte的空间。

**TermVector**表示文档的条目（由一个Document和Field定位）和它们在当前文档中所出现的次数

Field.TermVector.YES:为每个文档（Document）存储该字段的TermVector

Field.TermVector.NO:不存储TermVector

Field.TermVector.WITH\_POSITIONS:存储位置

Field.TermVector.WITH\_OFFSETS:存储偏移量

Field.TermVector.WITH\_POSITIONS\_OFFSETS:存储位置和偏移量

标准化因子参考：<https://www.iteye.com/blog/wwty-700193>

使用举例：

我们那文章表为例.articleinfo.有ID,title(标题),sumary(摘要),content(内容),userName(用户名)

其中title(标题),sumary(摘要)属于第一种情况,既要索引也要分词,也要存储.

content(内容)要分词,索引,但不存储.由于他太大了,而且界面也不用显示整个内容.

ID要存储,不用索引.因为没人用他来查询.但拼URL却很需要他.索引要存储.

UserName(用户名)索引,但不分词.可用保存.为什么不分词?比如"成吉思汗",我不想被"成汉"搜索到.我希望要么"成吉思汗"或者"\*吉思\*"通配符搜到.

常用列选项组合及用法如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **Index** | **Store** | **TermVector** | **用法** |
| NOT\_ANSLYZED | YES | NO | 文件名、主键 |
| ANSLYZED | YES | WITH\_POSITUION\_OFFSETS | 标题、摘要 |
| ANSLYZED | NO | WITH\_POSITUION\_OFFSETS | 很长的全文 |
| NO | YES | NO | 文档类型 |
| NOT\_ANSLYZED | NO | NO | 隐藏的关键词 |

## 7.索引维护

### 6.1索引创建（重点）

private void createIndex(News Model)

{

//定义索引目录路径

string path = Server.MapPath("/Indexs");

//定义一个分词器

Analyzer analyzer = new PanGuAnalyzer();

//定义索引用到的目录

Lucene.Net.Store.Directory d = FSDirectory.Open(new DirectoryInfo(path), new NativeFSLockFactory());

//如果指定目录中存在索引，则返回true,否则返回假 6217 0001 4002 9603 964

bool isUpdate = IndexReader.IndexExists(d);

if (isUpdate)

{

//如果当前目录中的索引是锁定状态，则解锁当前目录

if (IndexWriter.IsLocked(d))

IndexWriter.Unlock(d);

}

//第三个参数：true：创建索引或覆盖现有索引，false：追加索引

using (IndexWriter iw = new IndexWriter(d, analyzer, !isUpdate, IndexWriter.MaxFieldLength.LIMITED))

{

Document doc = new Document();

doc.Add(new Field("Id", Model.Id.ToString(), Field.Store.YES, Field.Index.NOT\_ANALYZED, Field.TermVector.NO));

doc.Add(new Field("Title", Model.Title, Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

doc.Add(new Field("Content", Model.Content, Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

doc.Add(new Field("AddTime", Model.AddTime.ToString("yyyy-MM-dd HH:mm:ss"), Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

iw.AddDocument(doc);

iw.Optimize();

}

}

### 6.2索引删除

lucene提供了两种从索引中删除document的方法，一种是

void deleteDocument(int docNum)

这种方法是根据document在索引中的编号来删除，每个document加进索引后都会有个唯一编号，所以根据编号删除是一种精确删除，但是这个编号是索引的内部结构，一般我们不会知道某个文件的编号到底是几，所以用处不大。另一种是

void deleteDocuments(Term term)

这种方法实际上是首先根据参数term执行一个搜索操作，然后把搜索到的结果批量删除了。我们可以通过这个方法提供一个严格的查询条件，达到删除指定document的目的。

示例代码：

public string DeleteIndex(int id)

{

Lucene.Net.Store.Directory dict = Lucene.Net.Store.FSDirectory.Open(Server.MapPath("/Indexs"));

using (IndexReader reader = IndexReader.Open(dict, false))

{

reader.DeleteDocuments(new Term("Id", id.ToString()));

return "删除成功";

}

}

### 6.2索引更新

索引文件的维护：维护索引常使用三个操作：插入、删除和更新文档。但是更新操作需要较高的代价，因为文档修改后（即使是很小的修改），也可以造成文档中的很多的关键词的位置发生了变化，这时需要频繁的读取和修改记录，这种代价是相当高的。因此，一般不进行更新操作，而是使用“先删除，后创建”的方式代替更新操作。

第一种方式：手动删除、手动添加

Lucene.Net.Store.Directory dict = Lucene.Net.Store.FSDirectory.Open(Server.MapPath("/Indexs"));

using (IndexReader reader = IndexReader.Open(dict, false))

{

reader.DeleteDocuments(new Term("Id", id.ToString()));

return "删除成功";

}

第二种方式，利用IndexWrite的UpdateDocument方法实现文档更新，其内部也是先删后加； UpdateDocument方法参数说明如下：

public virtual void UpdateDocument(Term term, Document doc, Analyzer analyzer);

term: 用于标识要删除的文档的术语

doc: the document to be added

analyzer:添加的document的分词器

示例代码：

public void UpdateIndex(News Model)

{

Lucene.Net.Store.Directory dict = Lucene.Net.Store.FSDirectory.Open(Server.MapPath("/Indexs"));

using (IndexWriter iw = new IndexWriter(dict, new PanGuAnalyzer(), IndexWriter.MaxFieldLength.UNLIMITED))

{

Term term = new Term("Id", Model.Id.ToString());

Document doc = new Document();

doc.Add(new Field("Id", Model.Id.ToString(), Field.Store.YES, Field.Index.NOT\_ANALYZED, Field.TermVector.NO));

doc.Add(new Field("Title", Model.Title, Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

doc.Add(new Field("Content", Model.Content, Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

doc.Add(new Field("AddTime", Model.AddTime.ToString("yyyy-MM-dd HH:mm:ss"), Field.Store.YES, Field.Index.ANALYZED, Field.TermVector.WITH\_OFFSETS));

iw.UpdateDocument(term, doc, new PanGuAnalyzer());

}

}

### 6.4索引优化

通过IndexWriter的Optimize方法优化索引，以加快搜索的速度，该方法提供多个重载，其执行相当耗时，应谨慎使用。优化产生的垃圾文件，在执行Flush/Commit/Close方法后才会被物理删除。Optimize方法及其重载包括：

Optimize() 合并段，完成后返回。

Optimize(bool doWait) 与Optimize()相同，但立即返回。

Optimize(int maxNumSegments) 针对最多maxNumSegments个段进行优化，而并非全部索引。

Optimize(int maxNumSegments, bool doWait) 与Optimize(int maxNumSegments)相同，但立即返回。

（优化索引实际就是在压缩索引文件，需要大约2倍索引大小的临时空间，且特别耗时。一种好的做法是把内存中的索引合并到应硬盘中。）

## 8 查询语法

Lucene的查询方式很丰富，对于数值类型的数据，采取TermRangeQuery的方式，对于String类型的，就可以采取TermQuery等，查询方式了，可以通过采取合适的查询方式，检索到数据。Queryparser这个查询方式包含了其他几种查询方式。

| **查询方式** | **意义** |
| --- | --- |
| TermQuery | 精确查询 |
| TermRangeQuery | 查询一个范围 |
| PrefixQuery | 前缀匹配查询 |
| WildcardQuery | 通配符查询 |
| BooleanQuery | 多条件查询 |
| PhraseQuery | 短语查询 |
| FuzzyQuery | 模糊查询 |
| Queryparser | 万能查询（上面的都可以用这个来查询到） |

lucene 的搜索相当强大，它提供了很多辅助查询类，每个类都继承自Query类，各自完成一种特殊的查询，你可以像搭积木一样将它们任意组合使用，完成一些复杂操 作；另外lucene还提供了Sort类对结果进行排序，提供了Filter类对查询条件进行限制。你或许会不自觉地拿它跟SQL语句进行比 较：“lucene能执行and、or、order by、where、like ‘%xx%’操作吗？”回答是：“当然没问题！”

注意：Field名字是区分大小写的；

### 8.1 TermQuery

首先介绍最基本的查询，如果你想执行一个这样的查询：“在content域中包含‘lucene’的document”，那么你可以用TermQuery：

Term t = new Term("Content", " lucene");

Query query = new TermQuery(t);

### 8.2 BooleanQuery

如果你想这么查询：“在content域中包含java或perl的document”，那么你可以建立两个TermQuery并把它们用BooleanQuery连接起来：



示例代码：

TermQuery termQuery1 = new TermQuery(new Term("Content", "java");

TermQuery termQuery 2 = new TermQuery(new Term("Content", "perl");

BooleanQuery booleanQuery = new BooleanQuery();

booleanQuery.add(termQuery 1, BooleanClause.Occur.SHOULD);

booleanQuery.add(termQuery 2, BooleanClause.Occur.SHOULD);

### 8.3 WildcardQuery

如果你想对某单词进行通配符查询，你可以用WildcardQuery，通配符包括’?’匹配一个任意字符和’\*’匹配零个或多个任意字符，例如你搜索’use\*’，你可能找到’useful’或者’useless’：

Query query = new WildcardQuery(new Term("content", "use\*");

### 8.4 PhraseQuery

你可能对中日关系比较感兴趣，想查找‘中’和‘日’挨得比较近（5个字的距离内）的文章，超过这个距离的不予考虑，你可以：

PhraseQuery query = new PhraseQuery();

query.setSlop(5);

query.add(new Term("content ", “中”));

query.add(new Term(“content”, “日”));

那么它可能搜到“中日合作……”、“中方和日方……”，但是搜不到“中国某高层领导说日本欠扁”。

### 8.5 PrefixQuery

FuzzyQuery用来搜索相似的term，使用Levenshtein算法。假设你想搜索跟‘wuzza’相似的词语，你可以：

Query query = new FuzzyQuery(new Term("content", "wuzza");

你可能得到‘fuzzy’和‘wuzzy’。

### 8.6 RangeQuery

另一个常用的Query是RangeQuery，你也许想搜索时间域从20060101到20060130之间的document，你可以用RangeQuery：

RangeQuery query = new RangeQuery(new Term(“time”, “20060101”), new Term(“time”, “20060130”), true);

最后的true表示用闭合区间。

### 8.7 QueryParser

看 了这么多Query，你可能会问：“不会让我自己组合各种Query吧，太麻烦了！”当然不会，lucene提供了一种类似于SQL语句的查询语句，我们 姑且叫它lucene语句，通过它，你可以把各种查询一句话搞定，lucene会自动把它们查分成小块交给相应Query执行。下面我们对应每种 Query演示一下：

TermQuery可以用“field:key”方式，例如“content:lucene”。

BooleanQuery中‘与’用‘+’，‘或’用‘ ’，例如“content:java contenterl”。

WildcardQuery仍然用‘?’和‘\*’，例如“content:use\*”。

PhraseQuery用‘~’，例如“content:"中日"~5”。

PrefixQuery用‘\*’，例如“中\*”。

FuzzyQuery用‘~’，例如“content: wuzza ~”。

RangeQuery用‘[]’或‘{}’，前者表示闭区间，后者表示开区间，例如“time:[20060101 TO 20060130]”，注意TO区分大小写。

你可以任意组合query string，完成复杂操作，例如“标题或正文包括lucene，并且时间在20060101到20060130之间的文章” 可以表示为：“+ (title:lucene content:lucene) +time:[20060101 TO 20060130]”。

代码如下：

Directory dir = FSDirectory.getDirectory(PATH, false);

IndexSearcher is = new IndexSearcher(dir);

QueryParser parser = new QueryParser("content", new StandardAnalyzer());

Query query = parser.parse("+(title:lucene content:lucene) +time:[20060101 TO 20060130]";

Hits hits = is.search(query);

for (int i = 0; i < hits.length(); i++)

{

Document doc = hits.doc(i);

System.out.println(doc.get("title");

}

is.close();

## 9、Filter索引过滤

filter 的作用就是限制只查询索引的某个子集，它的作用有点像SQL语句里的 where，但又有区别，它不是正规查询的一部分，只是对数据源进行预处理，然后交给查询语句。注意它执行的是预处理，而不是对查询结果进行过滤，所以使用filter的代价是很大的，它可能会使一次查询耗时提高一百倍。

最常用的filter是RangeFilter和QueryFilter。RangeFilter是设定只搜索指定范围内的索引；QueryFilter是在上次查询的结果中搜索。

Filter的使用非常简单，你只需创建一个filter实例，然后把它传给searcher。继续上面的例子，查询“时间在20060101到20060130之间的文章”除了将限制写在query string中，你还可以写在RangeFilter中：

Directory dir = FSDirectory.getDirectory(PATH, false);

IndexSearcher is = new IndexSearcher(dir);

QueryParser parser = new QueryParser("content", new StandardAnalyzer());

Query query = parser.parse("title:lucene content:lucene";

RangeFilter filter = new RangeFilter("time", "20060101", "20060230", true, true);

Hits hits = is.search(query, filter);

for (int i i < hits.length(); i++)

{

Document doc = hits.doc(i);

System.out.println(doc.get("title");

}

is.close();

再举个例子，我们搜索上架时间在 2005-10-1 到 2005-10-30 之间的商品。

对于日期时间，我们需要转换一下才能添加到索引库，同时还必须是索引字段。

// index

document.Add(FieldDate, DateField.DateToString(date), Field.Store.YES, Field.Index.UN\_TOKENIZED);

//...

// search

Filter filter = new DateFilter(FieldDate, DateTime.Parse("2005-10-1"), DateTime.Parse("2005-10-30"));

Hits hits = searcher.Search(query, filter);

除了日期时间，还可以使用整数。比如搜索价格在 100 ~ 200 之间的商品。

Lucene.Net NumberTools 对于数字进行了补位处理，如果需要使用浮点数可以自己参考源码进行。

// index

document.Add(new Field(FieldNumber, NumberTools.LongToString((long)price), Field.Store.YES, Field.Index.UN\_TOKENIZED));

//...

// search

Filter filter = new RangeFilter(FieldNumber, NumberTools.LongToString(100L), NumberTools.LongToString(200L), true, true);

Hits hits = searcher.Search(query, filter);

使用 Query 作为过滤条件。

QueryFilter filter = new QueryFilter(QueryParser.Parse("name2", FieldValue, analyzer));

我们还可以使用 FilteredQuery 进行多条件过滤。

Filter filter = new DateFilter(FieldDate, DateTime.Parse("2005-10-10"), DateTime.Parse("2005-10-15"));

Filter filter2 = new RangeFilter(FieldNumber, NumberTools.LongToString(11L), NumberTools.LongToString(13L), true, true);

Query query = QueryParser.Parse("name\*", FieldName, analyzer);

query = new FilteredQuery(query, filter);

query = new FilteredQuery(query, filter2);

IndexSearcher searcher = new IndexSearcher(reader);

Hits hits = searcher.Search(query);

## 10. Sort结果排序

有时你想要一个排好序的结果集，就像SQL语句的“order by”，lucene能做到：通过Sort。

=========SortField类============

//field是排序字段type是排序类型

public SortField(String field, Type type);

//field是排序字段type是排序类型reverse是指定升序还是降序

//reverse 为true是降序 false为升序

public SortField(String field, Type type, boolean reverse)

=========Sort类============

public Sort();//Sort对象构造方法默认是按文档评分排序

public Sort(SortField field);//排序的一个SortField

public Sort(SortField... fields)//排序的多个SortField可以传入一个数组

下面是一个完整的例子：

Directory dir = FSDirectory.getDirectory(PATH, false);

IndexSearcher is = new IndexSearcher(dir);

QueryParser parser = new QueryParser("content", new StandardAnalyzer());

Query query = parser.parse("title:lucene content:lucene");

RangeFilter filter = new RangeFilter("time", "20060101", "20060230", true, true);

Sort sort = new Sort(“time”);

Hits hits = is.search(query, filter, sort);

for (int i = 0; i < hits.length(); i++)

{

Document doc = hits.doc(i);

System.out.println(doc.get("title");

}

is.close();

## 11 关键词高亮

网站搜索关键词，往往搜索的结果中，要把用户搜索的词突出显示出来，这就是高亮搜索词的含义。而lucene也恰恰支持这样的操作。

代码如下：

//构造一个简单的html格式化标准

PanGu.HighLight.SimpleHTMLFormatter formatter = new PanGu.HighLight.SimpleHTMLFormatter("<font color='red'>", "</font>");

//构造一个高亮对象，它将应用改革才创建的格式化标准

PanGu.HighLight.Highlighter highter = new PanGu.HighLight.Highlighter(formatter, new PanGu.Segment());

//设置片段的长度，应该是格式化搜索词后带html标签的长度

highter.FragmentSize = 120;

//调用方法，替换数据title中的关键词，也就是高亮此关键词

highter.GetBestFragment(this.txtKeywords.Text, doc.Get("title");

## 12 完整查询示例

public List<News> search(string keyWord)

{

//定义分词器

Analyzer analyzer = new PanGuAnalyzer();

//定义索引目录路径

string path = Server.MapPath("/Indexs");

//定义索引用到的目录

FSDirectory directory = FSDirectory.Open(new DirectoryInfo(path), new NoLockFactory());

//返回读取给定目录中索引的IndexReader。

//您应该传递readOnly =true，因为它提供了更好的并发性能，除非您打算对读取器执行写操作（删除文档或更改规范）。

IndexReader reader = IndexReader.Open(directory, true);

IndexSearcher searcher = new IndexSearcher(reader);

//设置查询

Query query = new TermQuery(new Term("Content", keyWord));

TopScoreDocCollector collector = TopScoreDocCollector.Create(1000, true);

// 使用query这个查询条件进行搜索，搜索结果放入collector

searcher.Search(query, null, collector);

// 从查询结果中取出第m条到第n条的数据

// collector.GetTotalHits()表示总的结果条数

ScoreDoc[] docs = collector.TopDocs(0, collector.TotalHits).ScoreDocs;

// 遍历查询结果

List<News> resultList = new List<News>();

for (int i = 0; i < docs.Length; i++)

{

// 拿到文档的id，因为Document可能非常占内存（DataSet和DataReader的区别）

int docId = docs[i].Doc;

//// 所以查询结果中只有id，具体内容需要二次查询

//// 根据id查询内容：放进去的是Document，查出来的还是Document

Document doc = searcher.Doc(docId);

News result = new News();

result.Id = Convert.ToInt32(doc.Get("id"));

result.Title = doc.Get("Title");

result.AddTime = Convert.ToDateTime(doc.Get("AddTime"));

SimpleHTMLFormatter formatter = new SimpleHTMLFormatter("<font color='red'>", "</font>");

//构造一个高亮对象，它将应用改革才创建的格式化

Highlighter highter = new Highlighter(formatter, new PanGu.Segment());

//设置片段的长度，应该是格式化搜索词后带html标签的长度

highter.FragmentSize = 120;

//调用方法，替换数据title中的关键词，也就是高亮此关键词

result.Content = highter.GetBestFragment(keyWord, doc.Get("Content"));

resultList.Add(result);

}

return resultList;

}

参考文档：

[Lucene 的存储结构概述](https://blog.csdn.net/ghj1976/article/details/5586329)

[lucene 标准化因子 norm](https://blog.csdn.net/ghj1976/article/details/5586329)