# 前言

开发工具：eclipse

Lucene版本：5.5.1

# 1.索引

## 建立索引

准备工作：

**private** String[] ids = {"1","2","3","4","5","6"};

**private** String[] emails = {

"aa@itat.org","bb@itat.org",

"cc@cc.org","dd@sina.org",

"ee@zttc.edu",[ff@itat.org](mailto:ff@itat.org)

};

**private** String[] contents =

{

"welcome to visited the space,I like book",

"hello boy, I like pingpeng ball",

"my name is cc I like game",

"I like football",

"I like football and I like basketball too",

"I like movie and swim"

};

**private** String[] names = {"zhangsan","lisi","john","jetty","mike","jake"};

**private** Date[] dates = **null**;

**private** Map<String,Float> scores = **new** HashMap<String,Float>();

**private** Directory directory = **null** ;

**public** IndexUtil()

{

**try**

{

//设置索引文件保存路径

directory = FSDirectory.*open*(Paths.*get*("E:\\save"));

scores.put("itat.org",2.0f);

scores.put("zttc.edu", 1.5f);

setDates();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **void** setDates()

{

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");

**try** {

dates = **new** Date[ids.length];

dates[0] = sdf.parse("2010-02-19");

dates[1] = sdf.parse("2012-01-11");

dates[2] = sdf.parse("2011-09-19");

dates[3] = sdf.parse("2010-12-22");

dates[4] = sdf.parse("2012-01-01");

dates[5] = sdf.parse("2011-05-19");

} **catch** (ParseException e) {

e.printStackTrace();

}

}

**索引保存方式主要有两种：**

第一种是：保存到硬盘中（FSDirectory）。该类是Directory接口的基本实现类。FSDirectory有三个子类：

SimpleFSDirectory：该类对并发处理能力比较低，所以对多线程处理需要同步。

NIOFSDirectory：使用了NIO中的FileChannel实现FADirectory，对多个线程对同一个文件进行读操作，并不需要进行同步。

MMapDirectory：

Lucene会更具你的系统运行环境进行自动选择最合适的FSDirectory实现类。

第二种是：保存到内存中（RAMDirectory）。

建立索引：

**public** **void** index()

{

**try**

{

IndexWriter writer = **new** IndexWriter(directory, **new** IndexWriterConfig(**new** StandardAnalyzer()));

Document doc ;

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

{

doc = **new** Document();

TextField idField = **new** TextField("id", ids[i],Store.***YES***);

TextField contentField = **new** TextField("content",contents[i],Store.***NO***);

TextField emailField = **new** TextField("email",emails[i],Store.***YES***);

TextField nameField = **new** TextField("name",names[i],Store.***YES***);

//索引日期类型

StoredField dateField = **new** StoredField("date",DateTools.*dateToString*(dates[i], DateTools.Resolution.***DAY***));

doc.add(idField);doc.add(contentField);doc.add(emailField); doc.add(nameField);

doc.add(dateField);

String ek = emails[i].substring(emails[i].lastIndexOf("@")+1);

**if**(scores.containsKey(ek))

{

//设置权重

emailField.setBoost(scores.get(ek));

}

**else**

{

//加权

emailField.setBoost(0.5f);

}

//保存Docuemnt

writer.addDocument(doc);

}

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

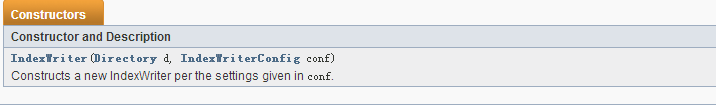
}

}

**IndexWriter:**用于创建和维护索引，可以用来更新，删除和添加索引。

在下面的章节，我们会详细介绍IndexWriter.hahahh

构造方法：



**IndexWriterConfig：**持有用来创建IndexWriter实例的所有配置，通过该对象创建IndexWriter之后，改变该对象并不能影响IndexWriter实例。

setOpenMode(IndexWriterConfig.OpenMode openMode)指定索引的打开模式。仅在IndexWriter创建时有效。

’

## 1.2 查看索引信息

在索引建立成功后，我们可以根据indexReader类，来获取索引信息：

**IndexReader：**一个抽象类，提供了一个接口，用于访问索引中某一个时间点的视图，在打开一个新的indexReader之前，通过IndexWriter对索引的修改都是不可见的。推荐使用DirecotroyReader.open(indexWriter)获取一个IndexReader，如果你的的IndexWriter正在处理中，当你需要重新打开以查看索引的更改时，最好使用DirectoryReader.openifchanged(DirectoryReader)，因为新的reader尽可能的与前一个reader共享资源。索引的搜索完全是通过这个抽象的接口完成的。

**public** **void** query()

{

**try**

{

//从reader对象可以有效获取文档数量

IndexReader reader = DirectoryReader.*open*(directory);

//获取最大Document数目

**int** maxDoc = reader.maxDoc();

//获取删除的Document数目

**int** numDocs = reader.numDocs();

**int** numDelete = reader.numDeletedDocs();

System.***out***.println("maxDoc:"+maxDoc);

System.***out***.println("numDocs:"+numDocs);

System.***out***.println("numDeleter:"+numDelete);

reader.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

## 1.3 删除指定的索引

**public** **void** delete()

{

IndexWriter writer;

**try** {

writer = **new** IndexWriter(directory, **new** IndexWriterConfig(**new** StandardAnalyzer()));

writer.deleteDocuments(**new** Term("id","1"));

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

## 1.4 索引更新

调用IndexWriter中的updateDocument方法来更新索引。

**public** **void** update()

{

IndexWriter writer;

**try**

{

writer = **new** IndexWriter(directory, **new** IndexWriterConfig(**new** StandardAnalyzer()));

Document doc = **new** Document();

TextField idField = **new** TextField("id", "11",Store.***YES***);

TextField contentField = **new** TextField("content",contents[0],Store.***NO***);

TextField emailField = **new** TextField("email",emails[0],Store.***YES***);

doc.add(idField);doc.add(contentField);doc.add(emailField);

writer.updateDocument(**new** Term("id","1"),doc);

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

# 2.搜索功能

## 2.1 索引建立

## 2.2 搜索

IndexReader是全局唯一的，无需关闭，我们通常是创建IndexSearch，在查询结束后关闭IndexSearch。

**private** IndexReader reader = **null** ;

**public** IndexSearcher getSearch()

{

**try**

{

**if**(reader == **null**)

reader = DirectoryReader.*open*(directory);

**else**

{

IndexReader newReader = DirectoryReader.*openIfChanged*((DirectoryReader)reader);

**if**(newReader != **null**)

{

reader.close();

reader = newReader ;

}

}

**return** **new** IndexSearcher(reader);

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null** ;

}

### 2.2.1 指定范围的搜索

### 2.2.2 数值搜索

### 2.2.3 多查询语句搜索

### 2.2.4 通配符搜索

## 2.3 分页

### 2.3.1 索引建立

# 4. 分词

## 4.1分词的核心类

* Analyzer
* TokenStream
* Tokenizer
* TokenFilter
* Attribute

## 4.2中文分词器

## 4.3 自定义分词器