# 全文检索基础

## 我们身边的搜索功能

* + - 1. Windows系统中的有搜索功能：打开“我的电脑”，按“F3”就可以使用查找的功能，查找指定的文件或文件夹。搜索的范围是整个电脑中的文件资源。
      2. Eclipse中的帮助子系统：点击Help🡪Help Contents，可以查找出相关的帮助信息。搜索的范围是Eclipse的所有帮助文件。
      3. 在BBS、BLOG、新闻等系统中提供的搜索文章的功能，如这里的贴吧的例子。搜索的范围是系统内的文章数据（都在数据库中）。
      4. 搜索引擎，如Baidu或Google等，可以查询到互联网中的网页、PDF、DOC、PPT、图片、音乐、视频等。下图是使用百度搜索的效果：



以上的查询功能都类似。都是查询的文本内容，都是相同的查询方式，即找出含有指定字符串的资源，不同的只是查询范围（分别为硬盘、所有帮助文件、数据库、互联网）。

## 什么是全文检索

全文检索是计算机程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置。当用户查询时根据建立的索引查找，类似于通过字典的检索字表查字的过程。

对于搜索，按被搜索的资源类型，分为两种：可以转为文本的、多媒体类型。我们上一节提到的搜索功能都是搜索的可以转为文本的资源（第一种）。注意，百度或谷歌提供的音乐或视频搜索不是多媒体搜索，他们是按文件名搜索。在智能手机上有一款音乐搜索的软件，可以让他听10秒钟的音乐，然后他就能上网找出这段音乐的名称、演奏者等信息。这是多媒体搜索。

按搜索的方式，上一节提到的搜索功能都是不处理语义，只是**找出包含指定词的所有资源**（只对词进行匹配）。下图就是显示“中国的首都是哪里”这个搜索要求对应的结果，可以看到，是没有“北京”这个结果的，结果页面都是出现了这些词的网页：



全文检索（Full-Text Retrieval）是指以文本作为检索对象，找出含有指定词汇的文本。全面、准确和快速是衡量全文检索系统的关键指标。

关于全文检索，我们要知道：1，只处理文本。2，不处理语义。3，搜索时英文不区分大小写。4，结果列表有相关度排序。

在信息检索工具中，全文检索是最具通用性和实用性的。

## 全文检索的应用场景

我们使用Lucene，主要是做站内搜索，即对一个系统内的资源进行搜索。如BBS、BLOG中的文章搜索，网上商店中的商品搜索等。使用Lucene的项目有Eclipse、Jira等。一般不做互联网中资源的搜索，因为不易获取与管理海量资源（专业搜索方向的公司除外）。

所以，学完Lucene后我们就可以为自已的系统增加全文检索的功能。跟这个学习内容相关的练习为：为“传智手播客贴吧”增加搜索其中的文章的功能。

## 全文检索不同于数据库搜索

全文检索不同于数据库的SQL查询。（他们所解决的问题不一样，解决的方案也不一样，所以不应进行对比）。在数据库中的搜索就是使用SQL，如：SELECT \* FROM t WHERE content like ‘%ant%’。这样会有如下问题：

1. 匹配效果：如搜索ant会搜索出planting。这样就会搜出很多无关的信息。
2. 相关度排序：查出的结果没有相关度排序，不知道我想要的结果在哪一页。我们在使用百度搜索时，一般不需要翻页，为什么？因为百度做了相关度排序：为每一条结果打一个分数，这条结果越符合搜索条件，得分就越高，叫做相关度得分，结果列表会按照这个分数由高到低排列，所以第1页的结果就是我们最想要的结果。
3. 全文检索的速度大大快于SQL的like搜索的速度。这是因为查询方式不同造成的，以查字典举例：数据库的like就是一页一页的翻，一行一行的找，而全文检索是先查目录，得到结果所在的页码，再直接翻到这一页。

所以数据库搜索不能替代全文检索。

## Lucene简介

全文检索就如同ORM，是一个概念。ORM的框架有很多种：Hibernate、TopLink、iBatis等，我们之前学习的是Hibernate。同样的，全文检索领域中也有多种框架，Lucene就是其中的一个用开源的全文检索框架。

Lucene的主页为：**http://lucene.apache.org/**。本文档中所使用的Lucene为3.0.1的版本。

# 第一个Lucene程序

## 准备Lucene的开发环境

搭建Lucene的开发环境只需要加入Lucene的Jar包，要加入的jar包至少要有：

* + - lucene-core-3.0.1.jar（核心包）
    - contrib\analyzers\common\lucene-analyzers-3.0.1.jar（分词器）
    - contrib\highlighter\lucene-highlighter-3.0.1.jar（高亮）
    - contrib\memory\lucene-memory-3.0.1.jar（高亮）

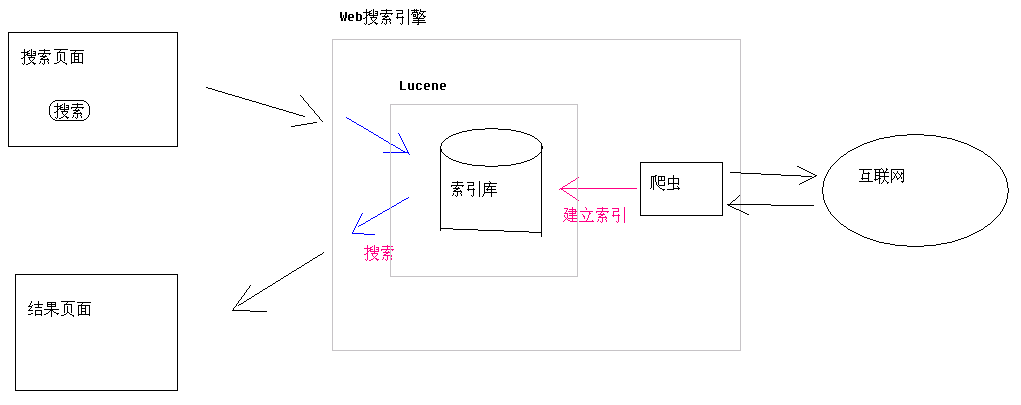
## 实现建立索引功能（Indexer类）

## 实现搜索功能（Searcher类）

# Lucene的核心概念

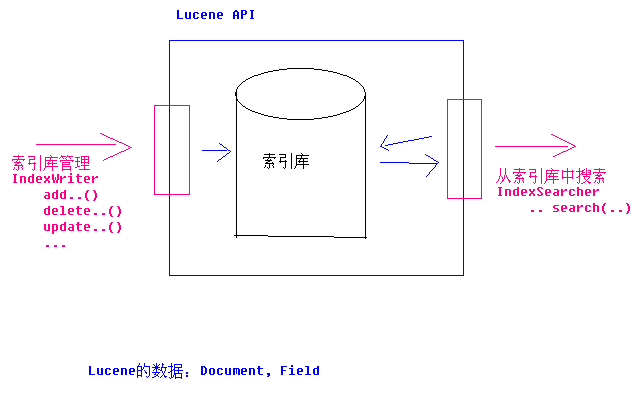
## 全文检索程序的工作流程

如果信息检索系统在用户发出了检索请求后再去互联网上找答案，根本无法在有限的时间内返回结果。所以要先把要检索的资源集合放到本地，并使用某种特定的结构存储，称为索引，这个索引的集合称为索引库。由于索引库的结构是按照专门为快速查询设计的，所以查询的速度非常快。我们每次搜索都是在本地的索引库中进行，如下图：



从图片上可以看出，我们不仅要搜索，还要保证数据集合与索引库的一致性。所以对于全文检索功能的开发，要做的有两个方面：索引库管理（维护索引库中的数据）、在索引库中进行搜索。而Lucene就是操作索引库的工具。

## 使用Lucene的API操作索引库



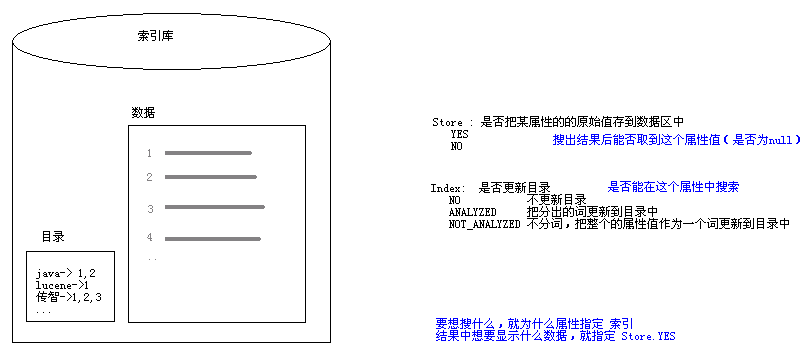
索引库是一个目录，里面是一些二进制文件，就如同数据库，所有的数据也是以文件的形式存在文件系统中的。我们不能直接操作这些二进制文件，而是使用Lucene提供的API完成相应的操作，就像操作数据库应使用SQL语句一样。

对索引库的操作可以分为两种：管理与查询。管理索引库使用IndexWriter，从索引库中查询使用IndexSearcher。Lucene的数据结构为Document与Field。Document代表一条数据，Field代表数据中的一个属性。一个Document中有多个Field，Field的值为String型，因为Lucene只处理文本。

我们只需要把在我们的程序中的对象转成Document，就可以交给Lucene管理了，搜索的结果中的数据列表也是Document的集合。

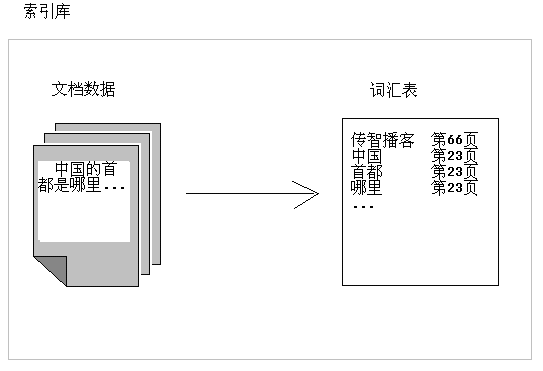
有了这些概念，可以写HelloWorld了，其他的概念可以在写完HelloWorld后再进行说明。

## 索引库结构——倒排序索引



我们需要对文档进行预处理，建立一种便于检索的数据结构，以此来提高信息检索的速度，这种数据结构就是索引。目前广泛使用的一种索引方式是倒排序索引。

倒排序索引的原理就如同查字典。要先查目录，得到数据对应的页码，在直接翻到指定的页码。不是在文章中找词，而是从目录中找词所在的文章。这需要在索引库中生成一个词汇表（目录），在词汇表中的每一个条记录都是类似于“词🡪所在文档的编号列表”的结构，记录了每一个出现过的单词，和单词出现的地方（哪些文档）。查询时先查词汇表，得到文档的编号，再直接取出相应的文档。



把数据转成指定格式放到索引库中的操作叫做建立索引。**建立索引时**，在把数据存到索引库后，再更新词汇表。**进行搜索时**，先从检索词汇表开始，然后找到相对应的文档。如果查询中仅包含一个关键词，则在词汇表中找到该单词，并取出他对应的文档就可以了。如果查询中包含多个关键词，则需要将各个单词检索出的记录进行合并再取出相应的文档记录。

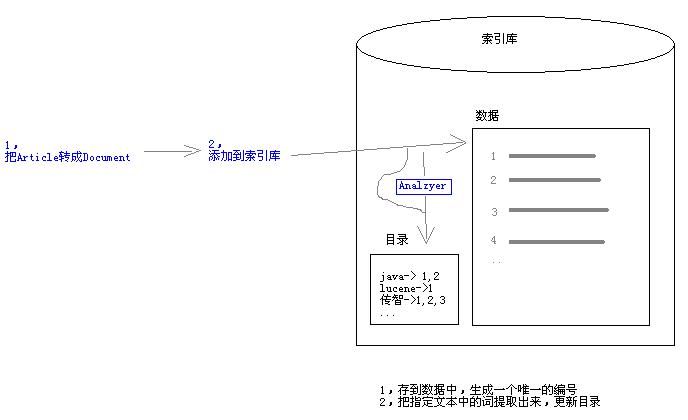
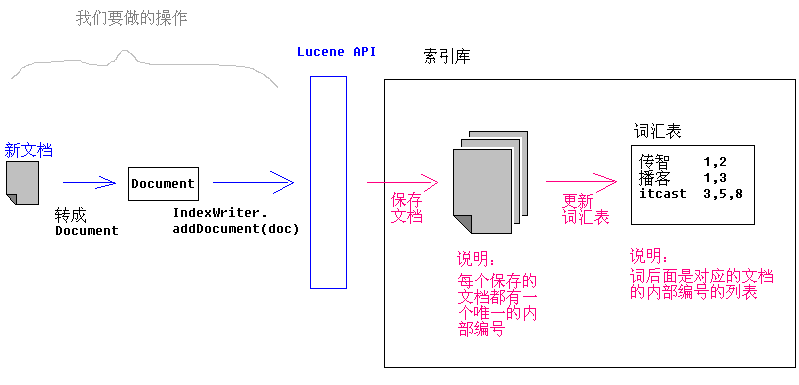
如果词汇表中有一个词“传智播客”对应的文档编号列表为“1”。现在又有添加了一个包含“传智播客”的文档，则词汇表中的“传智播客”词后对应的编号列表变成了“1,2”。因为关键词的数量受实际语言的限制，所以不用担心词汇表会变的很大。

## 索引文件的检索与维护，更新是先删除后创建

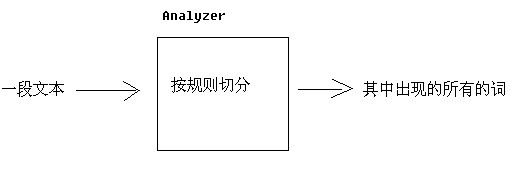
维护倒排索引有三个操作：添加、删除和更新文档。但是更新操作需要较高的代价。因为文档修改后（即使是很小的修改），就可能会造成文档中的很多的关键词的位置都发生了变化，这就需要频繁的读取和修改记录，这种代价是相当高的。因此，一般不进行真正的更新操作，而是使用“**先删除，再创建**”的方式代替更新操作。

## 建立索引的执行过程

在建立索引时，先要把文档存到索引库中，还要更新词汇表。如下图：



* + - 1. 我们做的操作：把数据对象转成相应的Document，其中的属性转为Field。
      2. 我们做的操作：调用工具IndexWriter的addDocument(doc)，把Document添加到索引库中。
      3. Lucene做的操作：**把文档存到索引库中，并自动指定一个内部编号，用来唯一标识这条数据。**内部编号类似于这条数据的地址，在索引库内部的数据进行调整后，这个编号就可能会改变，同时词汇表中引用的编号也会做相应改变，以保证正确。但我们如果在外面引用了这个编号，前后两次去取，得到的可能不是同一个文档！所以内部编号最好只在内部用。
      4. Lucene做的操作：**更新词汇表。把文本中的词找出并放到词汇表中，建立与文档的对应关系。**要把哪些词放到词汇表中呢，也就是文本中包含哪些词呢？这就用到了一个叫做Analyzer（分词器）的工具。他的作用是把一段文本中的词按规则取出所包含的所有词。对应的是Analyzer类，这是一个抽象类，切分词的具体规则是由子类实现的，所以对于不同的语言（规则），要用不同的分词器。如下图：



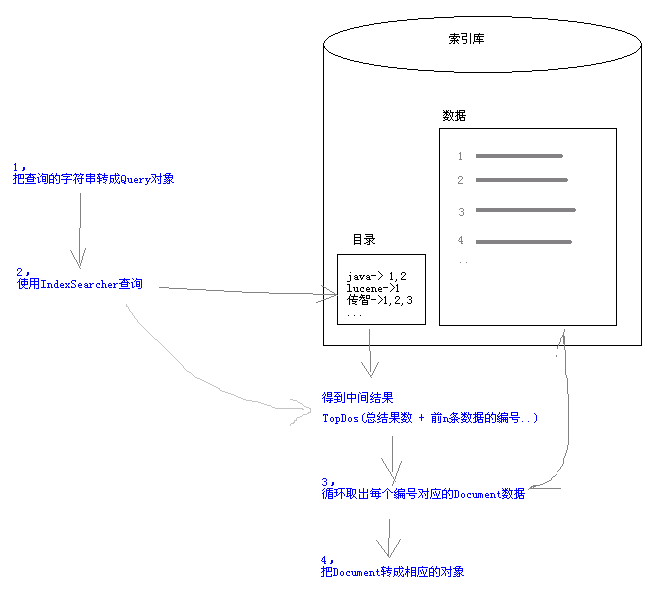
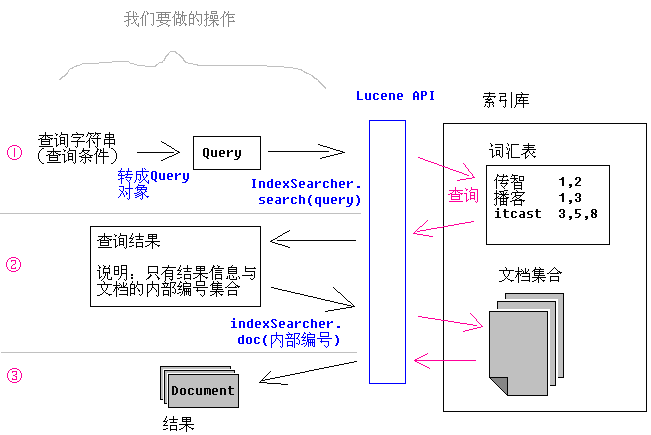
在把对象的属性转为Field时，相关代码为：doc.add(**new** Field("title", article.getTitle(), Store.*YES*, Index.*ANALYZED*))。第三与第四个参数的意思为：

|  |  |  |
| --- | --- | --- |
| 枚举类型 | 枚举常量 | 说明 |
| Store | NO | 不存储属性的值 |
| YES | 存储属性的值 |
| Index | NO | 不建立索引 |
| ANALYZED | 分词后建立索引 |
| NOT\_ANALYZED | 不分词，把整个内容作为一个词建立索引 |

说明：Store是影响搜索出的结果中是否有指定属性的原始内容。Index是影响是否可以从这个属性中查询（No），或是查询时可以查其中的某些词（ANALYZED），还是要把整个内容作为一个词进行查询（NOT\_ANALYZED）。

## 从索引库中搜索的执行过程

在进行搜索时，先在词汇表中查找，得到符合条件的文档编号列表。再根据文档编号真正的去取出数据（Document）。如下图：



1. 把要查询字符串转为Query对象。这就像在Hibernate中使用HQL查询时，也要先调用Session.createQuery(hql)转成Hibernate的Query对象一样。把查询字符串转换成Query是使用QueryParser，或使用MultiFieldQueryParser。查询字符串也要先经过Analyzer（分词器）。要求搜索时使用的Analyzer要与建立索引时使用的Analzyer要一致，否则可能搜不出正确的结果。
2. 调用IndexSearcher.search()，进行查询，得到结果。此方法返回值为TopDocs，是包含结果的多个信息的一个对象。其中有totalHits 代表决记录数，ScoreDoc的数组。ScoreDoc是代表一个结果的相关度得分与文档编号等信息的对象。
3. 取出要用到的数据列表。调用IndexSearcher.doc(scoreDoc.doc)以取出指定编号对应的Document数据。在分页时要用到：一次只取一页的数据。

# Lucene的核心API介绍

## 与建立索引有关的API

### IndexWriter

构造方法1：IndexWriter(Directory d, Analyzer a, MaxFieldLength mfl)

构造方法2：IndexWriter(Directory d, Analyzer a, boolean create, MaxFieldLength mfl)，第三个参数指定，true表示建立新的索引库或覆盖现有的索引库（删除后重建）；false表示使用已有的索引库，如果不存在，就报错。

commit()

rollback()

close()

### Directory

### Analyzer

### Document

### Field

## 与搜索有关的API

### IndexSearcher

在索引库中进行搜索是使用类IndexSearcher。创建其实例的构造方法为：IndexSearcher (Directory path)。用完后要调用IndexSearcher.close()方法释放资源。

### Term

### Query类及其子类

Query：抽象类，必须通过一系列子类来表述检索的具体需求。

### QueryParser与MultiFieldQueryParser

1. QueryParser与MultiFieldQueryParser

查询分析器，处理用户输入的查询条件。把用户输入的非格式化检索词转化成后台检索可以理解的Query对象。使用的构造方法为：QueryParser(Version matchVersion, String f, Analyzer a)

1. MultiFieldQueryParser

是QueryParser的子类。与父类相比，MultiFieldQueryParser可以在多个属性中搜索。使用的构造方法为：MultiFieldQueryParser(Version matchVersion, String[] fields, Analyzer analyzer)

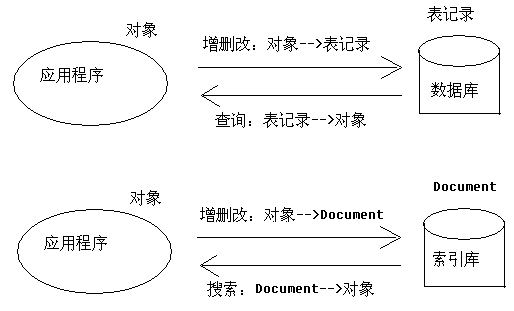
### TopDocs

### ScoreDoc

# 维护索引库

## 数据与Document、Field的转换

我们在应用程序中使用对象表示数据。在数据库中使用的是表记录，所以存在来回转换的问题。同样，要索引库中使用的是Document，也存在来回转换的问题。如下图：



对于一个要进行搜索的实体对象，我们会写一个对应的工具类，其中有两个方法：

Document Object2Document(Object object); // 对象🡪Document

Object Document2Object(Document doc); // Document🡪对象

在转换时，对象中的属性对应Document中的Field。由于Lucene只处理文本，所有所有的属性值在存储前都要先转成字符串。使用构造方法：Field(String name, String value, Store store, Index index)。

Store与Index都是枚举类型。**Store**：指定是否把当前属性值的原始内容存储到索引库中。如果存储（YES），在搜索出相应数据时这个属性就有原始的值；如果不存储（NO），得到的数据的这个属性的值为null。**Index**：指定是否建立索引（词汇表）。建立索引才能被搜索到。不可以不存储也不建立索引（没有意义）。

// Store 指定当前字段的数据要不要存到索引库中

// Index 指定当前字段的数据是否可以被搜索（是否更新词汇表）

**索引设置的一些建议：**

1) 尽量减少不必要的存储

2) 不需要检索的内容不要建立索引

3) 非文本格式需要提前转化

4）需要整体存放的内容不要分词

NumericUtils与DateTools

如果属性的类型不是字符串，则要先进转换：如果是数字类型，使用NumericUtils。如果是日期类型，则使用DataTools。

## 创建索引

## 删除索引

## 更新索引

## 索引库文件的优化

### 合并索引库文件

IndexWriter.optimize()

indexWriter.setMergeFactor(int)

### 使用RAMDirectory

Lucene的API接口设计的比较通用，输入输出结构都很像数据库的表==>记录==>字段，所以很多传统的应用的文件、数据库等都可以比较方便的映射到Lucene的存储结构/接口中。总体上看：可以先把Lucene当成一个支持全文索引的数据库系统。

Lucene的索引存储位置使用的是一个接口（抽象类），也就可以实现各种各样的实际存储方式（实现类、子类），比如存到文件系统中，存在内存中、存在数据库中等等。Lucene提供了两个子类：FSDirectory与RAMDirectory。

1. FSDirectory：在文件系统中，是真实的文件夹与文件。
2. RAMDirectory：在内存中，是模拟的文件夹与文件。与FSDirectory相比：1因为没有IO操作，所以速度快。2，因为在内存中，所以在程序退出后索引库数据就不存在了。

**索引库的相关操作：**

1. 合并索引库：Directory.addIndexes()
2. 索引的优化：IndexWriter.optimize()

# 从索引库中搜索

## 两种查询方式说明

## 使用查询字符串＋QueryParser的查询方式

QueryParser：只在一个字段中查询

MultiFieldQueryParser：可以在多个字段查询

## 使用构建Query子类的查询方式

### MatchAllDocsQuery

查询所有

### TermQuery

关键词查询

### NumericRangeQuery

范围查询。使用静态方法构造实例：

newIntRange(**final** String field,

Integer min, Integer max,

**final** **boolean** minInclusive, **final** **boolean** maxInclusive)

newLongRange(**final** String field,

Long min, Long max,

**final** **boolean** minInclusive, **final** **boolean** maxInclusive)

newFloatRange(**final** String field,

Float min, Float max,

**final** **boolean** minInclusive, **final** **boolean** maxInclusive)

newDoubleRange(**final** String field,

Double min, Double max,

**final** **boolean** minInclusive, **final** **boolean** maxInclusive)

### WildcardQuery

通配符查询

### FuzzyQuery

模糊查询

### PhraseQuery

短语查询

public void add(Term term, int position)

public void setSlop(int s)

例：add( new Term(“name”, “lucene”, 1);

add(new Term(“name”, “教程”, 3);

代表搜索的是“Lucene ？ 教程”，？表示中间隔一个词。

setSlop(2);

代表这两个词中间可以最多隔2个词

### BooleanQuery

public void add(Query query, Occur occur)

Occur 用于表示布尔查询子句关系的类，包括：

Occur.MUST，Occur.MUST\_NOT，Occur.SHOULD。

1. MUST和MUST：取得连个查询子句的交集。
2. MUST和MUST\_NOT：包含MUST并且查询结果中不包含MUST\_NOT的检索结果。
3. SHOULD与SHOULD，表示“或”关系，最终检索结果为所有检索子句的并集。

一般不单独使用，因为单独就不应使用BooleanQuery了。

使用时注意：

1. 单独使用MUST\_NOT：无意义，检索无结果。（也不报错）
2. 单独使用SHOULD：结果相当于MUST。
3. MUST\_NOT和MUST\_NOT：无意义，检索无结果。（也不报错）
4. SHOULD和MUST\_NOT： 此时SHOULD相当于MUST，结果同MUST和MUST\_NOT。
5. MUST和SHOULD：此时SHOULD无意义，结果为MUST子句的检索结果。

### 测试各种查询的代码模板

// 范围查询

@Test

**public** **void** testNumericRangeQuery() { }

// 关键词查询

@Test

**public** **void** testTermQuery() { }

// 通配符查询

@Test

**public** **void** testWildcardQuery() { }

// 查询所有

@Test

**public** **void** testMatchAllDocsQuery () { }

// 模糊查询

@Test

**public** **void** testFuzzyQuery(){ }

// 布尔查询

@Test

**public** **void** testBooleanQuery() { }

## 排序

通过改变文档Boost值来改变排序结果。Boost是指索引建立过程中，给整篇文档或者文档的某一特定属性设定的权值因子，在检索时，优先返回分数高的。通过Document对象的setBoost()方法和Field对象的setBoost()方法，可以分别为Document和Field指定Boost参数。不同在于前者对文档中每一个域都修改了参数，而后者只针对指定域进行修改。默认情值为1F，一般不做修改。

使用Sort对象定制排序。Sort支持的排序功能以文档当中的域为单位，通过这种方法，可以实现一个或者多个不同域的多形式的值排序。时间类型的属性采用STRING常量。

### 按相关度排序

1，相关度得分是在查询时根据查询条件实进计算出来的

2，如果索引库据不变，查询条件不变，查出的文档得分也不变

### 按指定的字段排序

If you want to be able to sort results by a Field value, you must add it as a Field that is indexed but not analyzed, using Field.Index.NOT\_ANALYZED.

## 过滤

使用Filter可以对搜索结果进行过滤以获得更小范围的结果。使用Filter对性能的影响很大（有可能会使查询慢上百倍）。

## 高亮

### 高亮所需的jar包

contrib\highlighter\lucene-highlighter-3.0.1.jar

contrib\memory\lucene-memory-3.0.1.jar

### 实现高亮效果的代码

// 生成高亮器

Formatter formatter = new SimpleHTMLFormatter("<span class='kw'>", "</span>");

Scorer scorer = new QueryScorer(query);

Highlighter highlighter = new Highlighter(formatter, scorer);

highlighter.setTextFragmenter(new SimpleFragmenter(20));

// 使用高亮器：对content属性值进行摘要并高亮

String text = highlighter.getBestFragment(LuceneUtils.getAnalyzer(), "content", doc.get("content"));

// 如果进行高亮的属性值中没有要搜索的关键字，则返回null

if (text != null) {

doc.getField("content").setValue(text);

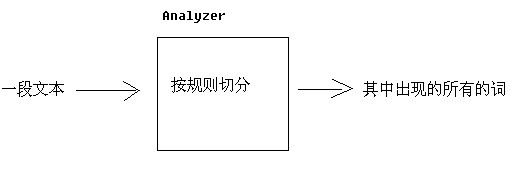
}

# 分词器

## 分词器的作用

在创建索引时会用到分词器，在使用字符串搜索时也会用到分词器，这两个地方要使用同一个分词器，否则可能会搜索不出结果。

Analyzer（分词器）的作用是把一段文本中的词按规则取出所包含的所有词。对应的是Analyzer类，这是一个抽象类，切分词的具体规则是由子类实现的，所以对于不同的语言（规则），要用不同的分词器。如下图：



## 分词器的工作流程

（英文）分词器的一般工作流程：

1. 切分关键词
2. 去除停用词

3，对于英文单词，把所有字母转为小写（搜索时不区分大小写）

说明：有的分词器还对英文进行形态还原，就是去除单词词尾的形态变化，将其还原为词的原形。这样做可以搜索出更多有意义的结果。如搜索sutdent时，也可以搜索出students，这是很有用的。

## 停用词

有些词在文本中**出现的频率非常高**，但是**对文本所携带的信息基本不产生影响**，例如英文的“a、an、the、of”，或中文的“的、了、着、是”，以及各种标点符号等。文本经过分词之后，**停用词通常被过滤掉，不会被进行索引，**这样的词称为**停用词**（stop word）。在检索的时候，用户的查询中如果含有停用词，检索系统也会将其过滤掉（因为用户输入的查询字符串也要进行分词处理）。排除停用词可以加快建立索引的速度，减小索引库文件的大小。

## 常用的中文分词器

中文的分词比较复杂，因为不是一个字就是一个词，而且一个词在另外一个地方就可能不是一个词，如在“帽子和服装”中，“和服”就不是一个词。对于中文分词，通常有三种方式：单字分词、二分法分词、词典分词。

* 单字分词：就是按照中文一个字一个字地进行分词。如：“我们是中国人”，  
  效果：“我”、“们”、“是”、“中”、“国”、“人”。（StandardAnalyzer就是这样）。
* 二分法分词：按两个字进行切分。如：“我们是中国人”，效果：“我们”、“们是”、“是中”、“中国”、“国人”。（CJKAnalyzer就是这样）。
* 词库分词：按某种算法构造词，然后去匹配已建好的词库集合，如果匹配到就切分出来成为词语。通常词库分词被认为是最理想的中文分词算法。如：“我们是中国人”，效果为：“我们”、“中国人”。（使用极易分词的MMAnalyzer。可以使用“极易分词”，或者是“庖丁分词”分词器、IKAnalyzer）。

其他的中文分词器有：

1. 极易分词：MMAnalyzer，最后版本是1.5.3，更新时间是2007-12-05，不支持Lucene3.0
2. 庖丁分词：PaodingAnalzyer，最后版本是2.0.4，更新时间是2008-06-03，不支持Lucene3.0

中文分词器使用IKAnalyzer，主页：<http://www.oschina.net/p/ikanalyzer>。

实现了以词典为基础的正反向全切分，以及正反向最大匹配切分两种方法。IKAnalyzer是第三方实现的分词器，继承自Lucene的Analyzer类，针对中文文本进行处理。具体的使用方式参见其文档。

注意：**扩展的词库与停止词文件要是UTF-8的编码**，并且在要文件头部加一空行。

## 测试分词器的代码

/\*\*

\* 使用指定的分词器对指定的文本进行分词，并打印出分出的词

\* **@param** analyzer

\* **@param** text

\* **@throws** Exception

\*/

**private** **void** testAnalyzer(Analyzer analyzer, String text) **throws** Exception {

System.*out*.println("当前使用的分词器：" + analyzer.getClass().getSimpleName());

TokenStream tokenStream = analyzer.tokenStream("content", **new** StringReader(text));

tokenStream.addAttribute(TermAttribute.**class**);

**while** (tokenStream.incrementToken()) {

TermAttribute termAttribute = tokenStream.getAttribute(TermAttribute.**class**);

System.*out*.println(termAttribute.term());

}

System.out.println();

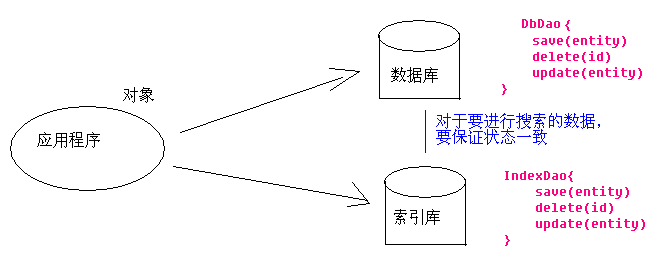
}

# Lucene应用

## 使用IndexDao封装对索引库的增删改查

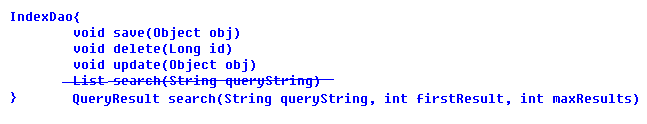
### IndexDao说明

提出问题：所有的数据（对象），我们都要存到数据库中。对于要进行搜索的数据，还要存到索引库中，以供搜索。一份数据同时存到数据库与索引库中（格式不同），就要想办法保证他们的状态一致。否则，就会影响搜索结果。



解决思路：对于上一段提出的问题：保证索引库中与数据库中的数据一致（只要针对要进行搜索的数据）。我们采用的方法是，在数据库中做了相应的操作后，在索引库中也做相应的操作。具体的索引库操作，是通过调用相应的IndexDao方法完成的。IndexDao类似于数据库层的Dao。

我们目前只关注IndexDao中的每个方法的作用（怎么用起来方便就怎么设计）。现在不需要关心IndexDao的每个方法怎么实现，因为那是下一步的事情。设计IndexDao如下：



**IndexDao的使用：**

PublishAction.execute(){ // 发表文章

actionForm 🡪 article对象

articleDao.save( article ); // 保存到数据库

articleIndexDao.save( article ); // 保存到索引库

}

DeleteAction.execute(){ // 删除文章

articleDao.delete( id ); // 从数据库中删除

articleIndex.delete( id ); // 从索引库中删除

}

UpdateAction.execute(){ // 更新文章

actionForm 🡪 article对象

articleDao.update( article ); // 更新数据库中的相应数据

articleIndexDao.update( article ); // 更新索引库中的相应数据

}

### 实现IndexDao中的方法：建立、删除、更新索引

索引库的管理操作操作是通过类IndexWriter完成的。创建实例是使用构造方法：IndexWriter(Directory d, Analyzer a, MaxFieldLength mfl)。用完后要调用IndexWriter.close()方法释放资源。

1. 建立索引：保存文档到索引库中。
   1. 把数据转成Document对象的形式。
   2. 调用方法IndexWriter.addDocument(Document doc)
2. 删除索引：删除所有包含指定Term的文档。
   1. 生成用于确定要删除的文档的Term
   2. 调用方法IndexWriter.deleteDocuments(Term term)

说明：在生成Term时，一般。如果有多个文档含有指定的Term，则都会被删掉。

1. 更新索引：实际执行的是先删除，后创建的操作。（参见前面的 [索引文件的检索与维护](#_索引文件的检索与维护)）
   1. 把要更新后的对象转为Document对象
   2. 生成用于确定要更新的文档的Term
   3. 调用方法IndexWriter.updateDocument(Term term, Document doc)

说明：如果有多个文档含有指定的Term，更新后就只有一条记录（删掉所有，再创建一个）。如果没有文档含有指定的记录，不会报错，更新后有一条（新创建的）记录。