

# 全文检索Lucene

---

@Author 健哥

## 课程计划

---

### 全文检索Lucene

#### 课程计划

#### 1. 搜索技术理论基础

- 1.1. 为什么要学习Lucene
- 1.2. 数据查询方法
  - 1.2.1. 顺序扫描法
  - 1.2.2. 倒排索引
- 1.3. 全文检索技术应用场景

#### 2. Lucene介绍

- 2.1. 什么是全文检索
- 2.2. 什么是Lucene
- 2.3. Lucene官网

#### 3. Lucene全文检索的流程

- 3.1. 索引和搜索流程图
- 3.2. 索引流程
  - 3.2.1. 原始内容
  - 3.2.2. 获得文档（采集数据）
  - 3.2.3. 创建文档
  - 3.2.4. 分析文档
  - 3.2.5. 索引文档
  - 3.2.6 Lucene底层存储结构

#### 4. Lucene入门

- 4.1. 开发环境
- 4.2. 创建Java工程
- 4.3. 索引流程
  - 4.3.1. 数据采集
    - 4.3.1.1. 创建pojo
    - 4.3.1.2. 创建DAO接口
    - 4.3.1.3. 创建DAO接口实现类
  - 4.3.2. 实现索引流程
- 4.4. 搜索流程
  - 4.4.1. 输入查询语句
    - 4.4.1.1. 搜索分词
  - 4.4.2. 代码实现

## 5. Field域类型

### 5.1. Field属性

### 5.2. Field常用类型

### 5.3. Field修改

#### 5.3.1. 修改分析

#### 5.3.2. 代码修改

## 6. 索引维护

### 6.1. 需求

### 6.2. 添加索引

### 6.3. 修改索引

### 6.4. 删除索引

#### 6.4.1. 删除指定索引

#### 6.4.2. 删除全部索引（慎用）

## 7. 分词器

### 7.1. 分词理解

### 7.2. Analyzer使用时机

#### 7.2.1. 索引时使用Analyzer

#### 7.2.2. 搜索时使用Analyzer

### 7.3. Lucene原生分词器

#### 7.3.1. StandardAnalyzer

#### 7.3.2. WhitespaceAnalyzer

#### 7.3.3. SimpleAnalyzer

#### 7.3.4. CJKAnalyzer

### 7.4. 第三方中文分词器

#### 7.4.1. 什么是中文分词器

#### 7.4.2. 第三方中文分词器简介

#### 7.4.3. 使用中文分词器IKAnalyzer

#### 7.4.4. 扩展中文词库

## 8. 搜索案例

### 8.1. 引入依赖

### 8.2. 项目加入页面和资源

### 8.3. 创建包和启动类

### 8.4. 配置文件

### 8.5. 业务代码:

#### 8.5.1. 封装pojo

#### 8.5.2. controller代码

#### 8.5.3. service代码

## 9. Lucene底层储存结构(高级)

### 9.1. 详细理解lucene存储结构

### 9.2. 索引库物理文件

### 9.3. 索引库文件扩展名对照表

### 9.4. 词典的构建

#### 9.4.1. 词典数据结构对比

9.4.2. 跳跃表原理

9.4.3. FST原理简析

## 10. Lucene相关度排序(高级)

10.1.什么是相关度排序

10.1.1. 如何打分

10.1.2. 什么是词的权重

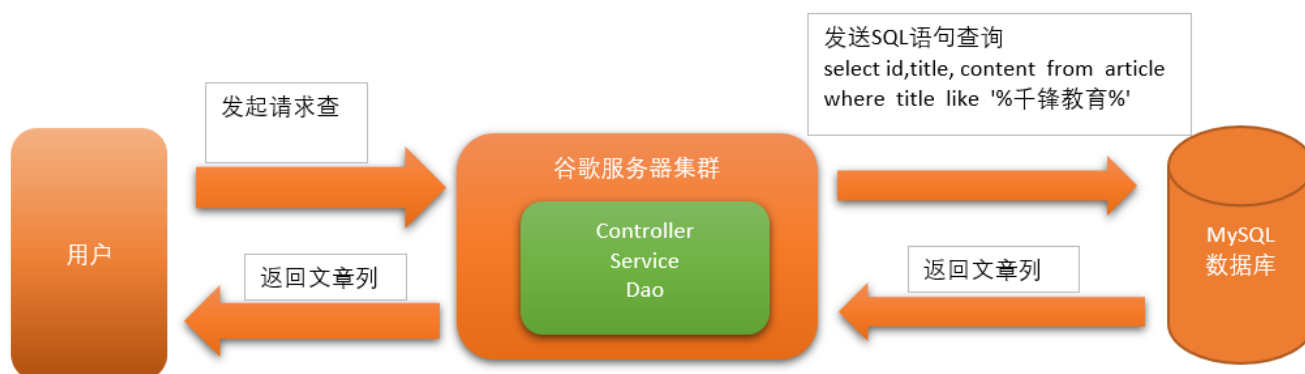
10.1.3. 怎样影响相关度排序

10.2.人为影响相关度排序

# 1. 搜索技术理论基础

## 1.1. 为什么要学习Lucene

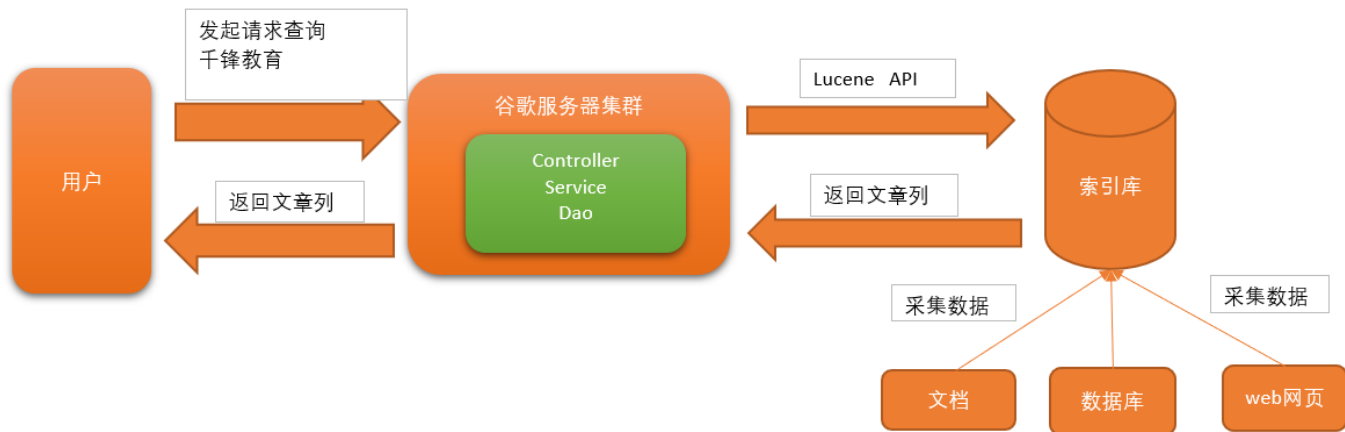
原来的方式实现搜索功能，我们的搜索流程如下图：



上图就是原始搜索引擎技术，如果用户比较少而且数据库的数据量比较小，那么这种方式实现搜索功能在企业中是比较常见的。

但是数据量过多时，数据库的压力就会变得很大，查询速度会变得非常慢。我们需要使用更好的解决方案来分担数据库的压力。

现在的方案（使用Lucene），如下图



为了解决数据库压力和速度的问题，我们的数据库就变成了索引库，我们使用Lucene的API的来操作服务器上的索引库。这样完全和数据库进行了隔离。

## 1.2. 数据查询方法

### 1.2.1. 顺序扫描法

#### 算法描述：

所谓顺序扫描，例如要找内容包含一个字符串的文件，就是一个文档一个文档的看，对于每一个文档，从头看到尾，如果此文档包含此字符串，则此文档为我们要找的文件，接着看下一个文件，直到扫描完所有的文件。

#### 优点：

查询准确率高

#### 缺点：

查询速度会随着查询数据量的增大，越来越慢

#### 使用场景：

- 数据库中的like关键字模糊查询
- 文本编辑器的Ctrl + F 查询功能

## 1.2.2. 倒排索引

先举一个栗子：

例如我们使用新华字典查询汉字，新华字典有偏旁部首的目录（索引），我们查字首先查这个目录，找到这个目录中对应的偏旁部首，就可以通过这个目录中的偏旁部首找到这个字所在的位置（文档）。

Lucene会对文档建立倒排索引

- 1、 提取资源中关键信息， 建立索引（目录）
- 2、 搜索时，根据关键字（目录），找到资源的位置

**算法描述：**

查询前会先将查询的内容提取出来组成文档(正文), 对文档进行切分词组成索引(目录), 索引和文档有关联关系, 查询的时候先查询索引, 通过索引找文档的这个过程叫做全文检索。

**切分词：**就是将一句一句话切分成一个一个的词, 去掉停用词(的, 地, 得, a, an, the等)。去掉空格, 去掉标点符号, 大写字母转成小写字母, 去掉重复的词。

**为什么倒排索引比顺序扫描快？**

**理解：**因为索引可以去掉重复的词, 汉语常用的字和词大概等于, 字典加词典, 常用的英文在牛津词典也有收录.如果用计算机的速度查询, 字典+词典+牛津词典这些内容是非常快的. 但是用这些字典, 词典组成的文章却是千千万万不计其数. 索引的大小最多也就是字典+词典. 所以通过查询索引, 再通过索引和文档的关联关系找到文档速度比较快. 顺序扫描法则是直接去逐个查询那些不计其数的文章就算是计算的速度也会很慢.

**优点：**

查询准确率高

查询速度快， 并且不会因为查询内容容量的增加， 而使查询速度逐渐变慢

**缺点：**

索引文件会占用额外的磁盘空间， 也就是占用磁盘量会增大。

**使用场景：**

海量数据查询, pb级数据查询

## 1.3. 全文检索技术应用场景

---

应用场景：

- 1、站内搜索（baidu贴吧、论坛、京东、taobao）
- 2、垂直领域的搜索（818工作网）
- 3、专业搜索引擎公司（google、baidu）

## 2. Lucene介绍

---

### 2.1. 什么是全文检索

---

计算机索引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，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式

### 2.2. 什么是Lucene

---



他是Lucene、Nutch、Hadoop等项目的发起人Doug Cutting

Lucene是apache软件基金会4 jakarta项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言）。

Lucene的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

目前已经有很多应用程序的搜索功能是基于 Lucene 的，比如 Eclipse 的帮助系统的搜索功能。Lucene 能够为文本类型的数据建立索引，所以你只要能把你要索引的数据格式转化的文本的，Lucene 就能对你的文档进行索引和搜索。比如你要对一些 HTML 文档，PDF 文档进行索引的话你就首先需要把 HTML 文档和 PDF 文档转化成文本格式的，然后将转化后的内容交给 Lucene 进行索引，然后把创建好的索引文件保存到磁盘或者内存中，最后根据用户输入的查询条件在索引文件上进行查询。不指定要索引的文档的格式也使 Lucene 能够几乎适用于所有的搜索应用程序。

- Lucene是一套用于全文检索和搜寻的开源程式库，由Apache软件基金会支持和提供
- Lucene提供了一个简单却强大的应用程式接口，能够做全文索引和搜寻，在Java开发环境里Lucene是一个成熟的免费开放源代码工具
- Lucene并不是现成的搜索引擎产品，但可以用来制作搜索引擎产品

## 2.3. Lucene官网

---

官网：<http://lucene.apache.org/>

Ultra-fast Search Library

APACHE  
**LUCENE**<sup>™</sup>

Apache Lucene sets the standard for search and indexing performance

## Lucene<sup>™</sup> Core News

Apache Lucene is a high-performance, full-featured search engine library written entirely in Java. It is a technology suitable for nearly any application that requires structured search, full-text search, faceting, nearest-neighbor search across high-dimensionality vectors, spell correction or query suggestions.

You may also read these news as an [ATOM feed](#).

### 16 December 2021 - Apache Lucene<sup>™</sup> 8.11.1 available

The Lucene PMC is pleased to announce the release of Apache Lucene 8.11.1.

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

This release contains one bug fix. The release is available for immediate download at:

**DOWNLOAD**

Apache Lucene 9.0.0

### Resources

[Mailing Lists](#)

[Developer](#)

[Features](#)

[Releases](#)

[System Requirements](#)

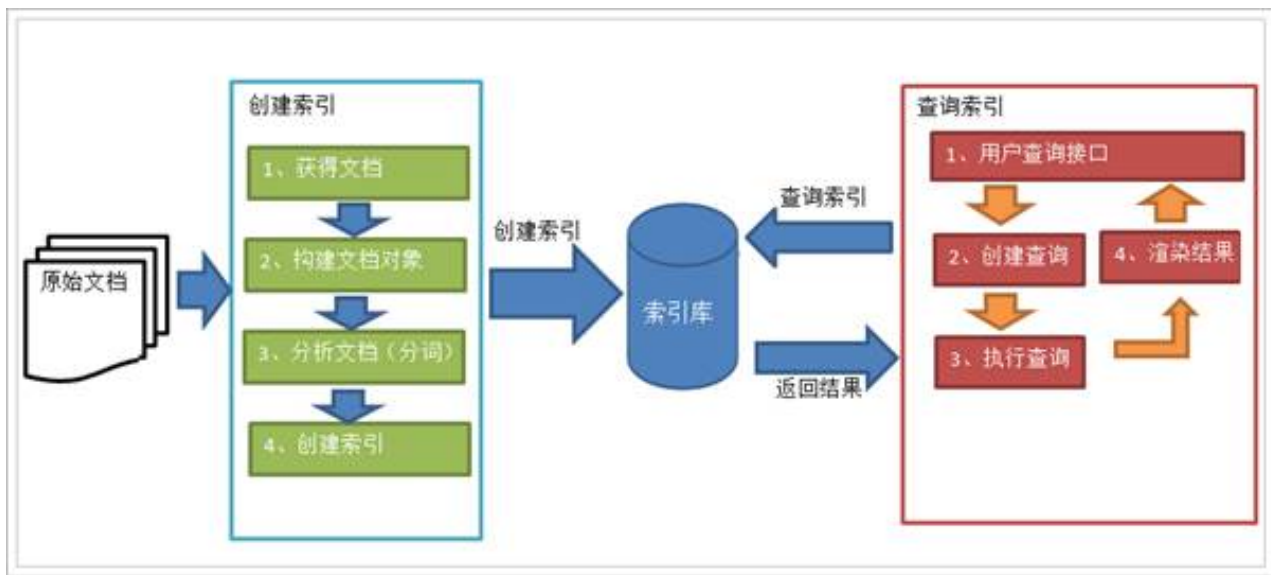
### Release Docs

[9.0.0](#)

### About

## 3. Lucene全文检索的流程

### 3.1. 索引和搜索流程图





1、绿色表示索引过程，对要搜索的原始内容进行索引构建一个索引库，索引过程包括：

确定原始内容即要搜索的内容

- 获得文档
- 创建文档
- 分析文档
- 索引文档

2、红色表示搜索过程，从索引库中搜索内容，搜索过程包括：

用户通过搜索界面

- 创建查询
- 执行搜索，从索引库搜索
- 渲染搜索结果

## 3.2. 索引流程

---

对文档索引的过程，将用户要搜索的文档内容进行索引，索引存储在索引库（index）中。

### 3.2.1. 原始内容

原始内容是指要索引和搜索的内容。

原始内容包括互联网上的网页、数据库中的数据、磁盘上的文件等。

### 3.2.2. 获得文档（采集数据）

从互联网上、数据库、文件系统中等获取需要搜索的原始信息，这个过程就是信息采集，采集数据的目的是为了对原始内容进行索引。

采集数据分类：

- 1、对于互联网上网页，可以使用工具将网页抓取到本地生成html文件。
- 2、数据库中的数据，可以直接连接数据库读取表中的数据。
- 3、文件系统中的某个文件，可以通过I/O操作读取文件的内容。

在互联网上采集信息的软件通常称为爬虫或蜘蛛，也称为网络机器人，爬虫访问互联网上的每一个网页，将获取到的网页内容存储起来。

### 3.2.3. 创建文档

获取原始内容的目的是为了索引，在索引前需要将原始内容创建成文档（Document），文档中包括一个一个的域（Field），域中存储内容。

这里我们可以将磁盘上的一个文件当成一个document，Document中包括一些Field，如下图：



注意：每个Document可以有多个Field，不同的Document可以有不同的Field，同一个Document可以有相同的Field（域名和域值都相同）

### 3.2.4. 分析文档

将原始内容创建为包含域（Field）的文档（document），需要再对域中的内容进行分析，分析成为一个一个的单词。

比如下边的文档经过分析如下：

原文档内容：

vivo Z3 6GB+64GB 极光蓝 性能实力派 全面屏游戏手机 移动联通电信全网通4G手机

华为 HUAWEI 畅享9 Plus 4GB+64GB 幻夜黑 全网通 四摄超清全面屏大电池 移动联通电信4G手机 双卡双待

分析后得到的词：

vivo, Z3, 6GB, 64GB, 极光, 极光蓝, 全网, 全网通, 网通, 4G, 手机, 华为, HUAWEI, 畅享9。。。。

### 3.2.5. 索引文档

对所有文档分析得出的语汇单元进行索引，索引的目的是为了搜索，最终要实现只搜索被索引的语汇单元从而找到Document（文档）。

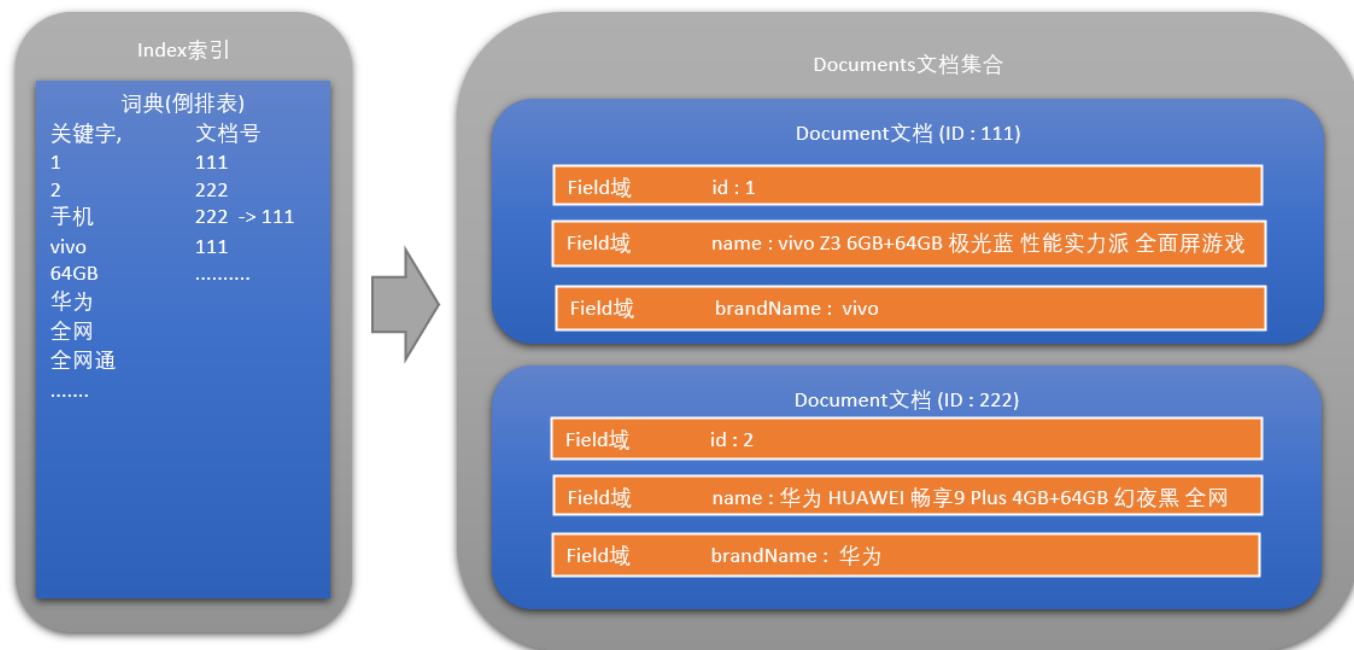
创建索引是对语汇单元索引，通过词语找文档，这种索引的结构叫倒排索引结构。

倒排索引结构是根据内容（词汇）找文档，如下图：



倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。

### 3.2.6 Lucene底层存储结构



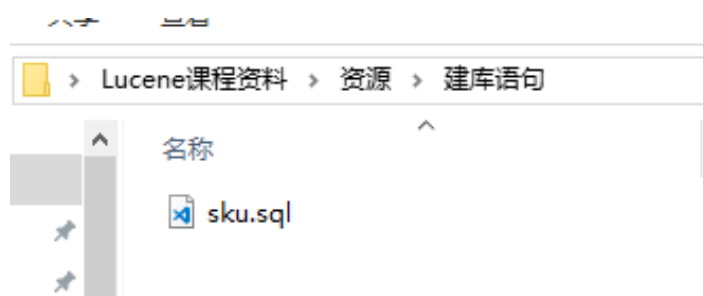
## 4. Lucene入门

### 4.1. 开发环境

JDK: 1.8 (Lucene8以上, 必须使用JDK1.8及以上版本)

数据库: MySQL

数据库脚本位置如下图:

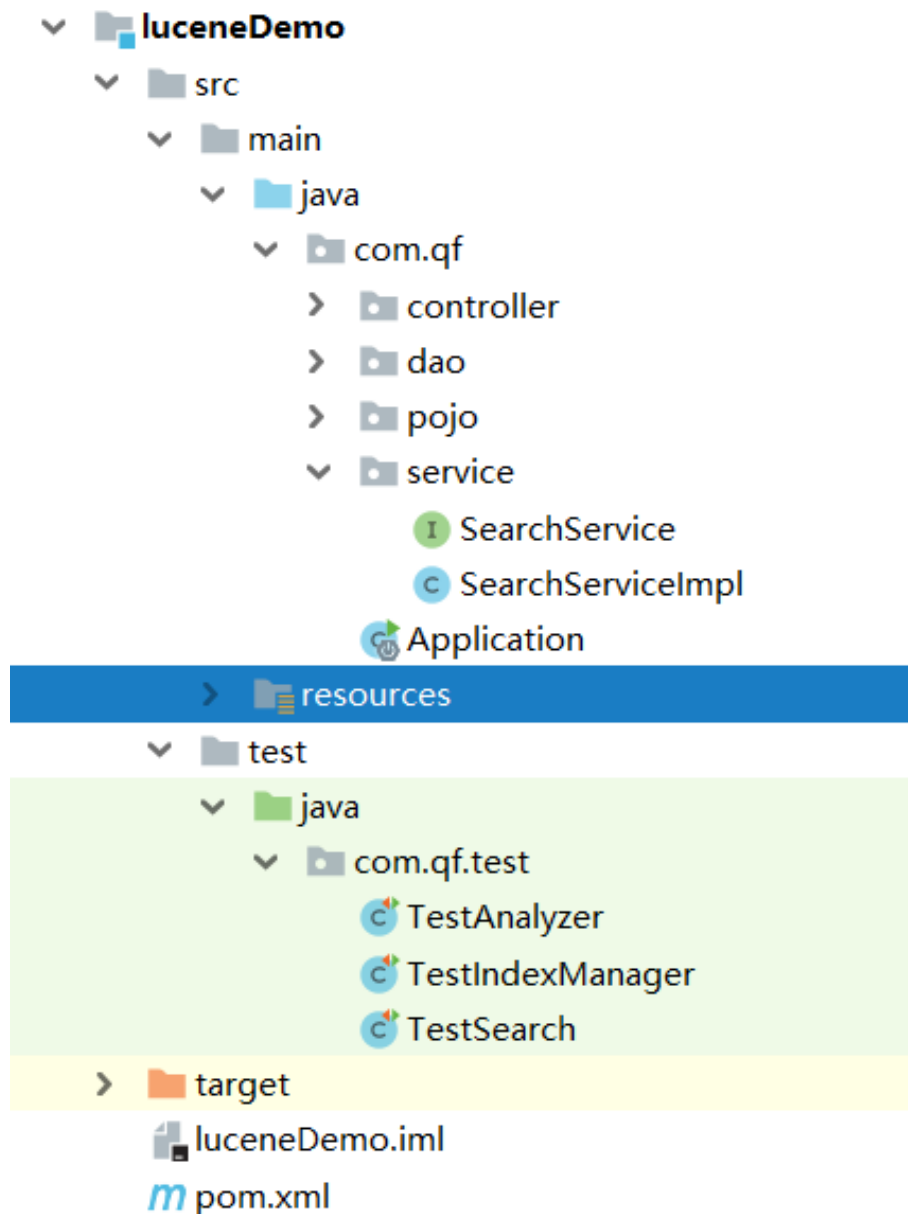


导入到MySQL效果如下图:

id	name	price	num	image	category_name
100001021787	稻草人(MEXICAN)行李箱男女 24英寸大容量拉杆箱 万向轮静音德国拜耳PC材质海出差旅行箱 星空黑2	49400	10000	https://m.360buyimg.cc	拉杆箱
100001021793	【京选尚品】地平线8号 (LEVEL8) 商务行李箱旅行箱 (标准版) 铝镁合金登机箱20英寸拉杆箱灰色 (穗科出品)	76200	10000	https://m.360buyimg.cc	拉杆箱
100001021803	卡拉羊 (Carany) 儿童拉杆箱18英寸旅行行李箱万向轮男女学生登机小箱子CX8632宝蓝	13300	10000	https://m.360buyimg.cc	拉杆箱
100001036214	努比亚 nubia X 双面屏 海光蓝 8GB+256GB 全网通 移动联通电信4G手机 双卡双待	97700	10000	https://m.360buyimg.cc	手机
100001036228	努比亚 nubia X 双面屏 海光蓝 8GB+128GB 全网通 移动联通电信4G手机 双卡双待	73900	10000	https://m.360buyimg.cc	手机
100001036244	努比亚 nubia X 双面屏 海光蓝 8GB+128GB 全网通 移动联通电信4G手机 双卡双待	39000	10000	https://m.360buyimg.cc	手机
100001036246	努比亚 nubia X 双面屏 海光蓝 8GB+256GB 全网通 移动联通电信4G手机 双卡双待	50300	10000	https://m.360buyimg.cc	手机
100001036248	努比亚 nubia X 双面屏 海光蓝 8GB+512GB 全网通 移动联通电信4G手机 双卡双待	30400	10000	https://m.360buyimg.cc	手机
100001045283	vivo Y73 3GB+64GB 香槟金 大屏大内存全面屏手机 移动联通电信全网通4G手机	94100	10000	https://m.360buyimg.cc	手机
100001053161	黑鲨游戏手机 Helo 双液冷 更能打 6GB+128GB 极夜黑 全网通4G 双卡双待	48400	10000	https://m.360buyimg.cc	手机
100001054331	OPPO R17新年版 2500万美颜拍照 6.4英寸水滴屏 光感屏幕指纹 6G+128G 全网通 移动联通电信4G 双卡双待手机	78400	10000	https://m.360buyimg.cc	手机
100001054333	OPPO R17新年版 2500万美颜拍照 6.4英寸水滴屏 光感屏幕指纹 6G+128G 全网通 移动联通电信4G 双卡双待手机	55000	10000	https://m.360buyimg.cc	手机
100001054621	vivo Y73 4GB+64GB 香槟金 大屏大内存全面屏手机 移动联通电信全网通4G手机	9800	10000	https://m.360buyimg.cc	手机
100001062900	新西兰原装进口牛奶 湿康WDOM 4.0全脂纯牛奶 800ml*6瓶/箱 3.7蛋白质高钙	94600	10000	https://m.360buyimg.cc	牛奶
100001069955	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 30	28400	10000	https://m.360buyimg.cc	牛仔裤
100001069957	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 30	20900	10000	https://m.360buyimg.cc	牛仔裤
100001069961	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 28	68000	10000	https://m.360buyimg.cc	牛仔裤
100001069967	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 30	9700	10000	https://m.360buyimg.cc	牛仔裤
100001069969	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 27	27000	10000	https://m.360buyimg.cc	牛仔裤
100001069971	俞兆林 2018冬季新款韩版百搭加绒加厚微喇叭牛仔裤女 YWKN181002-1 深蓝 27	83200	10000	https://m.360buyimg.cc	牛仔裤
100001069987	俞兆林 2018冬季新款高腰排扣牛仔裤女修身百搭铅笔小脚黄金加绒裤 YWKN181002-2 黑色 27	85600	10000	https://m.360buyimg.cc	牛仔裤
100001070121	俞兆林 2018冬季新款韩版加绒加厚牛仔裤女排扣高腰裤修身铅笔小脚长裤 YWKN181002-4 黑色 29	6300	10000	https://m.360buyimg.cc	牛仔裤

## 4.2. 创建Java工程

创建maven工程不依赖骨架, 测试即可, 效果如下:



## pom.xml依赖配置

```
1 <properties>
2     <maven.compiler.source>1.8</maven.compiler.source>
3     <maven.compiler.target>1.8</maven.compiler.target>
4     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
5     <project.reporting.outputEncoding>UTF-
6     <skipTests>true</skipTests>
7 </properties>
8
9 <parent>
10     <groupId>org.springframework.boot</groupId>
11     <artifactId>spring-boot-starter-parent</artifactId>
12     <version>2.1.4.RELEASE</version>
```

```
13 </parent>
14
15 <dependencies>
16     <dependency>
17         <groupId>commons-io</groupId>
18         <artifactId>commons-io</artifactId>
19         <version>2.6</version>
20     </dependency>
21
22     <dependency>
23         <groupId>org.apache.lucene</groupId>
24         <artifactId>lucene-core</artifactId>
25         <version>8.11.1</version>
26     </dependency>
27     <dependency>
28         <groupId>org.apache.lucene</groupId>
29         <artifactId>lucene-analyzers-common</artifactId>
30         <version>8.11.1</version>
31         <exclusions>
32             <exclusion>
33                 <groupId>org.apache.lucene</groupId>
34                 <artifactId>lucene-core</artifactId>
35             </exclusion>
36         </exclusions>
37     </dependency>
38     <dependency>
39         <groupId>org.apache.lucene</groupId>
40         <artifactId>lucene-queryparser</artifactId>
41         <version>8.11.1</version>
42         <exclusions>
43             <exclusion>
44                 <groupId>org.apache.lucene</groupId>
45                 <artifactId>lucene-core</artifactId>
46             </exclusion>
47         </exclusions>
48     </dependency>
49
50     <dependency>
51         <groupId>org.apache.lucene</groupId>
52         <artifactId>lucene-backward-codecs</artifactId>
53         <version>8.11.1</version>
54     </dependency>
55
```



```
56      <!-- 测试 -->
57      <dependency>
58          <groupId>junit</groupId>
59          <artifactId>junit</artifactId>
60          <version>4.12</version>
61          <scope>test</scope>
62      </dependency>
63      <!-- mysql数据库驱动 -->
64      <dependency>
65          <groupId>mysql</groupId>
66          <artifactId>mysql-connector-java</artifactId>
67          <version>5.1.48</version>
68      </dependency>
69
70      <!-- IK中文分词器 -->
71      <dependency>
72          <groupId>org.wltea.ik-analyzer</groupId>
73          <artifactId>ik-analyzer</artifactId>
74          <version>8.1.0</version>
75      </dependency>
76
77      <!--web起步依赖-->
78      <dependency>
79          <groupId>org.springframework.boot</groupId>
80          <artifactId>spring-boot-starter-web</artifactId>
81      </dependency>
82      <!-- 引入thymeleaf -->
83      <dependency>
84          <groupId>org.springframework.boot</groupId>
85          <artifactId>spring-boot-starter-thymeleaf</artifactId>
86      </dependency>
87      <!-- Json转换工具 -->
88      <dependency>
89          <groupId>com.alibaba</groupId>
90          <artifactId>fastjson</artifactId>
91          <version>1.2.51</version>
92      </dependency>
93  </dependencies>
```

## 4.3. 索引|流程

## 4.3.1. 数据采集

在电商网站中，全文检索的数据源在数据库中，需要通过jdbc访问数据库中 **sku** 表的内容。

### 4.3.1.1. 创建pojo

```
1  public class Sku {
2
3      //商品主键id
4      private String id;
5      //商品名称
6      private String name;
7      //价格
8      private Integer price;
9      //库存数量
10     private Integer num;
11     //图片
12     private String image;
13     //分类名称
14     private String categoryName;
15     //品牌名称
16     private String brandName;
17     //规格
18     private String spec;
19     //销量
20     private Integer saleNum;
21
22     get/set。。。
23
24 }
```

### 4.3.1.2. 创建DAO接口

```
1 public interface SkuDao {
2
3     /**
4      * 查询所有的Sku数据
5      * @return
6      */
7
8     public List<Sku> querySkuList();
9 }
```

### 4.3.1.3. 创建DAO接口实现类

使用jdbc实现

```
1 public class SkuDaoImpl implements SkuDao {
2
3     public List<Sku> querySkuList() {
4         // 数据库链接
5         Connection connection = null;
6         // 预编译statement
7         PreparedStatement preparedStatement = null;
8         // 结果集
9         ResultSet resultSet = null;
10        // 商品列表
11        List<Sku> list = new ArrayList<Sku>();
12
13        try {
14            // 加载数据库驱动
15            Class.forName("com.mysql.jdbc.Driver");
16            // 连接数据库
17            connection =
18                DriverManager.getConnection("jdbc:mysql://localhost:3306/lucenedemo",
19                    "root", "123456");
20
21            // SQL语句
22            String sql = "SELECT * FROM tb_sku";
23            // 创建preparedStatement
24            preparedStatement = connection.prepareStatement(sql);
25
26            // 获取结果集
27            resultSet = preparedStatement.executeQuery();
28        } catch (Exception e) {
29            e.printStackTrace();
30        }
31
32        return list;
33    }
34 }
```

```

25         // 结果集解析
26         while (resultSet.next()) {
27             Sku sku = new Sku();
28             sku.setId(resultSet.getString("id"));
29             sku.setName(resultSet.getString("name"));
30             sku.setSpec(resultSet.getString("spec"));
31             sku.setBrandName(resultSet.getString("brand_name"));
32
33             sku.setCategoryName(resultSet.getString("category_name"));
34             sku.setImage(resultSet.getString("image"));
35             sku.setNum(resultSet.getInt("num"));
36             sku.setPrice(resultSet.getInt("price"));
37             sku.setSaleNum(resultSet.getInt("sale_num"));
38             list.add(sku);
39         }
40     } catch (Exception e) {
41         e.printStackTrace();
42     }
43     return list;
44 }
45 }

```

## 4.3.2. 实现索引流程

1. 采集数据
2. 创建Document文档对象
3. 创建分析器（分词器）
4. 创建IndexWriterConfig配置信息类
5. 创建Directory对象，声明索引库存储位置
6. 创建IndexWriter写入对象
7. 把Document写入到索引库中
8. 释放资源

```

1 public class TestManager {
2
3     @Test
4     public void createIndexTest() throws Exception {
5         // 1. 采集数据

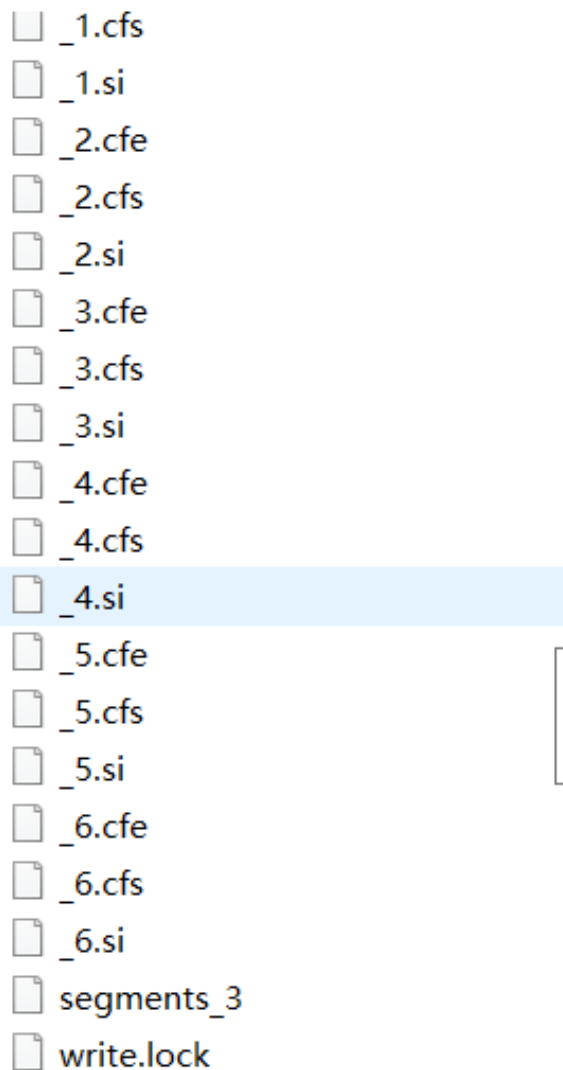
```

```
6      SkuDao skuDao = new SkuDaoImpl();
7      List<Sku> skuList = skuDao.querySkuList();
8
9      // 2. 创建Document文档对象
10     List<Document> documents = new ArrayList<Document>();
11     for (Sku sku : skuList) {
12         Document document = new Document();
13
14         // Document文档中添加Field域
15         // 商品Id
16         // Store.YES:表示存储到文档域中
17         document.add(new TextField("id", sku.getId(),
Field.Store.YES));
18         // 商品名称
19         document.add(new TextField("name", sku.getName(),
Field.Store.YES));
20         // 商品价格
21         document.add(new TextField("price",
sku.getPrice().toString(), Field.Store.YES));
22         // 品牌名称
23         document.add(new TextField("brandName", sku.getBrandName(),
Field.Store.YES));
24         // 分类名称
25         document.add(new TextField("categoryName",
sku.getCategoryName(), Field.Store.YES));
26         // 图片地址
27         document.add(new TextField("image", sku.getImage(),
Field.Store.YES));
28
29         // 把Document放到list中
30         documents.add(document);
31     }
32
33     // 3. 创建Analyzer分词器,分析文档,对文档进行分词
34     Analyzer analyzer = new StandardAnalyzer();
35
36     // 4. 创建Directory对象,声明索引库的位置
37     Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
38
39     // 5. 创建IndexWriterConfig对象,写入索引需要的配置
40     IndexWriterConfig config = new IndexWriterConfig(analyzer);
41
42     // 6. 创建IndexWriter写入对象
```

```
43     IndexWriter indexWriter = new IndexWriter(directory, config);
44
45     // 7.写入到索引库, 通过IndexWriter添加文档对象document
46     for (Document doc : documents) {
47         indexWriter.addDocument(doc);
48     }
49
50     // 8.释放资源
51     indexWriter.close();
52 }
53
54 }
```

执行效果:

在文件夹中出现了以下文件, 表示创建索引成功



- \_1.cfs
- \_1.si
- \_2.cfe
- \_2.cfs
- \_2.si
- \_3.cfe
- \_3.cfs
- \_3.si
- \_4.cfe
- \_4.cfs
- \_4.si
- \_5.cfe
- \_5.cfs
- \_5.si
- \_6.cfe
- \_6.cfs
- \_6.si
- segments\_3
- write.lock

## 4.4. 搜索流程

### 4.4.1. 输入查询语句

Lucene 可以通过 query 对象输入查询语句。同数据库的 sql 一样，lucene 也有固定的查询语法：

最基本的有比如：AND, OR, NOT 等（必须大写）

举个栗子：

用户想找一个 **name** 域中包括 **手** 或 **机** 关键字的文档。

它对应的查询语句：**name:手 OR name:机**

#### 4.4.1.1. 搜索分词

和索引过程的分词一样，这里要对用户输入的关键字进行分词，一般情况索引和搜索使用的分词器一致。

比如：输入搜索关键字“java 学习”，分词后为 java 和学习两个词，与 java 和学习有关的内容都搜索出来了，如下：



## 4.4.2. 代码实现

1. 创建Query搜索对象
2. 创建Directory流对象,声明索引库位置
3. 创建索引读取对象IndexReader
4. 创建索引搜索对象IndexSearcher
5. 使用索引搜索对象, 执行搜索, 返回结果集TopDocs
6. 解析结果集
7. 释放资源

IndexSearcher搜索方法如下：

方法	说明
<code>indexSearcher.search(query, n)</code>	根据Query搜索, 返回评分最高的n条记录
<code>indexSearcher.search(query, filter, n)</code>	根据Query搜索, 添加过滤策略, 返回评分最高的n条记录
<code>indexSearcher.search(query, n, sort)</code>	根据Query搜索, 添加排序策略, 返回评分最高的n条记录
<code>indexSearcher.search(booleanQuery, filter, n, sort)</code>	根据Query搜索, 添加过滤策略, 添加排序策略, 返回评分最高的n条记录

代码实现

```
1 public class TestSearch {
2
3     @Test
4     public void testIndexSearch() throws Exception {
5         // 1. 创建Query搜索对象
6         // 创建分词器
7         Analyzer analyzer = new StandardAnalyzer();
8         // 创建搜索解析器, 第一个参数: 默认Field域, 第二个参数: 分词器
9         QueryParser queryParser = new QueryParser("brandName",
10 analyzer);
```



```
11      // 创建搜索对象
12      Query query = queryParser.parse("name:手机 AND 华为");
13
14      // 2. 创建Directory流对象,声明索引库位置
15      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
16
17      // 3. 创建索引读取对象IndexReader
18      IndexReader reader = DirectoryReader.open(directory);
19
20      // 4. 创建索引搜索对象
21      IndexSearcher searcher = new IndexSearcher(reader);
22
23      // 5. 使用索引搜索对象,执行搜索,返回结果集TopDocs
24      // 第一个参数:搜索对象,第二个参数:返回的数据条数,指定查询结果最顶部的n
条数据返回
25      TopDocs topDocs = searcher.search(query, 10);
26      System.out.println("查询到的数据总条数是: " +
topDocs.totalHits.value);
27      // 获取查询结果集
28      ScoreDoc[] docs = topDocs.scoreDocs;
29
30      // 6. 解析结果集
31      for (ScoreDoc scoreDoc : docs) {
32          // 获取文档
33          int docID = scoreDoc.doc;
34          Document doc = searcher.doc(docID);
35
36          System.out.println("=====");
37          System.out.println("docID:" + docID);
38          System.out.println("id:" + doc.get("id"));
39          System.out.println("name:" + doc.get("name"));
40          System.out.println("price:" + doc.get("price"));
41          System.out.println("brandName:" + doc.get("brandName"));
42          System.out.println("image:" + doc.get("image"));
43      }
44      // 7. 释放资源
45      reader.close();
46  }
47 }
```

# 5. Field域类型

## 5.1. Field属性

Field是文档中的域，包括Field名和Field值两部分，一个文档可以包括多个Field，Document只是Field的一个承载体，Field值即为要索引的内容，也是要搜索的内容。

域名:域值

- 是否分词(tokenized)

是：作分词处理，即将Field值进行分词，分词的目的是为了索引。

比如：商品名称、商品描述等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分词后将语汇单元建立索引

否：不作分词处理

比如：商品id、订单号、身份证号等

- 是否索引(indexed)

是：进行索引。将Field分词后的词或整个Field值进行索引，存储到索引域，索引的目的是为了搜索。

比如：商品名称、商品描述分析后进行索引，订单号、身份证号不用分词但也要索引，这些将来都要作为查询条件。

否：不索引。

比如：图片路径、文件路径等，不用作为查询条件的不用索引。

- 是否存储(stored)

是：将Field值存储在文档域中，存储在文档域中的Field才可以从Document中获取。

比如：商品名称、订单号，凡是将来要从Document中获取的Field都要存储。

否：不存储Field值

比如：商品描述，内容较大不用存储。如果要向用户展示商品描述可以从系统的关系数据库中获取。

## 5.2. Field常用类型

下边列出了开发中常用 的Filed类型，注意Field的属性，根据需求选择：

Field类	数据类型	Analyzed 是否分词	Indexed 是否索引	Stored 是否存储	说明
StringField(FieldName, FieldValue, Store.YES))	字符串	N	Y	Y或N	这个Field用来构建一个字符串Field，但是不会进行分词，会将整个串存储在索引中，比如(订单号,身份证号等) 是否存储在文档中用Store.YES或Store.NO 决定
FloatPoint(FieldName, FieldValue)	Float型	Y	Y	N	这个Field用来构建一个Float数字型Field，进行分词和索引，不存储, 比如(价格) 存储在文档中
DoublePoint(FieldName, FieldValue)	Double型	Y	Y	N	这个Field用来构建一个Double数字型Field，进行分词和索引，不存储
LongPoint(FieldName, FieldValue)	Long型	Y	Y	N	这个Field用来构建一个Long数字型Field，进行分词和索引，不存储
IntPoint(FieldName, FieldValue)	Integer型	Y	Y	N	这个Field用来构建一个Integer数字型Field，进行分词和索引，不存储
StoredField(FieldName, FieldValue)	重载方法，支持多种类型	N	N	Y	这个Field用来构建不同类型Field 不分析，不索引，但要Field存储在文档中
TextField(FieldName, FieldValue, Store.NO) 或 TextField(FieldName, reader)	字符串或流	Y	Y	Y或N	如果是一个Reader, lucene猜测内容比较多,会采用Unstored的策略。
NumericDocValuesField(FieldName, FieldValue)	数值	-	-	-	配合其他域排序使用

## 5.3. Field修改

### 5.3.1. 修改分析

图书id：

是否分词：不用分词，因为不会根据商品id来搜索商品

是否索引：不索引，因为不需要根据图书ID进行搜索

是否存储：要存储，因为查询结果页面需要使用id这个值。

图书名称：

是否分词：要分词，因为要根据图书名称的关键词搜索。

是否索引：要索引。

是否存储：要存储。

图书价格：

是否分词：要分词，lucene对数字型的值只要有搜索需求的都要分词和索引，因为 lucene对数字型的内容要特殊分词处理，需要分词和索引。

是否索引：要索引

是否存储：要存储

图书图片地址：

是否分词：不分词

是否索引：不索引

是否存储：要存储

图书描述：

是否分词：要分词

是否索引：要索引

是否存储：因为图书描述内容量大，不在查询结果页面直接显示，不存储。

不存储是不在lucene的索引域中记录，节省lucene的索引文件空间。

如果要在详情页面显示描述，解决方案：

从lucene中取出图书的id，根据图书的id查询关系数据库（MySQL）中book表得到描述信息。

## 5.3.2. 代码修改

对之前编写的testCreateIndex()方法进行修改。

代码片段

```
1 Document document = new Document();
2
3 // Document文档中添加Field域
4 // 商品Id, 不分词,索引,存储
5 document.add(new StringField("id", sku.getId(), Field.Store.YES));
6 // 商品名称, 分词, 索引, 存储
7 document.add(new TextField("name", sku.getName(), Field.Store.YES));
8
9 // 商品价格, 分词,索引,不存储, 不排序
10 document.add(new FloatPoint("price", sku.getPrice()));
11 //添加价格存储支持
12 document.add(new StoredField("price", sku.getPrice()));
13 //添加价格排序支持
14 //document.add(new NumericDocValuesField("price",sku.getPrice()));
15
16 // 品牌名称, 不分词, 索引, 存储
17 document.add(new StringField("brandName", sku.getBrandName(),
18     Field.Store.YES));
19 // 分类名称, 不分词, 索引, 存储
20 document.add(new StringField("categoryName", sku.getCategoryName(),
21     Field.Store.YES));
22 // 图片地址, 不分词,不索引,存储
23 document.add(new StoredField("image", sku.getImage()));
24
25 // 把Document放到list中
26 documents.add(document);
```

## 6. 索引|维护

### 6.1. 需求

管理人员通过电商系统更改图书信息，这时更新的是关系数据库，如果使用lucene搜索图书信息，需要在数据库表book信息变化时及时更新lucene索引库。

### 6.2. 添加索引

调用 `indexWriter.addDocument (doc)` 添加索引。

参考入门程序的创建索引。

### 6.3. 修改索引

更新索引是先删除再添加，建议对更新需求采用此方法并且要保证对已存在的索引执行更新，可以先查询出来，确定更新记录存在执行更新操作。

如果更新索引的目标文档对象不存在，则执行添加。

代码

```
1  @Test
2  public void testIndexUpdate() throws Exception {
3      // 创建分词器
4      Analyzer analyzer = new StandardAnalyzer();
5      // 创建Directory流对象
6      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
7      // 创建IndexWriteConfig对象，写入索引需要的配置
8      IndexWriterConfig config = new IndexWriterConfig(analyzer);
9      // 创建写入对象
10     IndexWriter indexWriter = new IndexWriter(directory, config);
11
12     // 创建Document
13     Document document = new Document();
14     document.add(new TextField("id", "1202790956", Field.Store.YES));
15     document.add(new TextField("name", "lucene测试test 002",
16                               Field.Store.YES));
17
18     // 执行更新，会把所有符合条件的Document删除，再新增。
```

```
18     indexWriter.updateDocument(new Term("id", "1202790956"), document);
19
20     // 释放资源
21     indexWriter.close();
22 }
```

## 6.4. 删除索引

### 6.4.1. 删除指定索引

根据Term项删除索引，满足条件的将全部删除。

```
1  @Test
2  public void testIndexDelete() throws Exception {
3      // 创建分词器
4      Analyzer analyzer = new StandardAnalyzer();
5      // 创建Directory流对象
6      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
7      // 创建IndexWriterConfig对象，写入索引需要的配置
8      IndexWriterConfig config = new IndexWriterConfig(analyzer);
9      // 创建写入对象
10     IndexWriter indexWriter = new IndexWriter(directory, config);
11
12     // 根据Term删除索引库，name:java
13     indexWriter.deleteDocuments(new Term("id", "998188"));
14
15     // 释放资源
16     indexWriter.close();
17 }
```

### 6.4.2. 删除全部索引（慎用）

将索引目录的索引信息全部删除，直接彻底删除，无法恢复。

建议参照关系数据库基于主键删除方式，所以在创建索引时需要创建一个主键Field，删除时根据此主键Field删除。

索引删除后将放在Lucene的回收站中，Lucene3.X版本可以恢复删除的文档，3.X之后无法恢复。

代码：

```
1  @Test
2  public void testIndexDeleteAll() throws Exception {
3      // 创建分词器
4      Analyzer analyzer = new StandardAnalyzer();
5      // 创建Directory流对象
6      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
7      // 创建IndexWriterConfig对象，写入索引需要的配置
8      IndexWriterConfig config = new IndexWriterConfig(analyzer);
9      // 创建写入对象
10     IndexWriter indexWriter = new IndexWriter(directory, config);
11
12     // 全部删除
13     indexWriter.deleteAll();
14
15     // 释放资源
16     indexWriter.close();
17 }
```

## 7. 分词器

### 7.1. 分词理解

在对Document中的内容进行索引之前，需要使用分词器进行分词，分词的目的是为了搜索。分词的主要过程就是先分词后过滤。

- 分词：采集到的数据会存储到document对象的Field域中，分词就是将Document中Field的value值切分成一个一个的词。
- 过滤：包括去除标点符号过滤、去除停用词过滤（的、是、a、an、the等）、大写转小写、词的形还原（复数形式转成单数形参、过去式转成现在式。。。等）等。



什么是停用词？停用词是为节省存储空间和提高搜索效率，搜索引擎在索引页面或处理搜索请求时会自动忽略某些字或词，这些字或词即被称为Stop Words(停用词)。比如语气助词、副词、介词、连接词等，通常自身并无明确的意义，只有将其放入一个完整的句子中才有一定作用，如常见的“的”、“在”、“是”、“啊”等。

对于分词来说，不同的语言，分词规则不同。Lucene作为一个工具包提供不同国家的分词器

## 7.2. Analyzer使用时机

---

### 7.2.1. 索引时使用Analyzer

输入关键字进行搜索，当需要让该关键字与文档域内容所包含的词进行匹配时需要对文档域内容进行分析，需要经过Analyzer分析器处理生成语汇单元（Token）。分析器分析的对象是文档中的Field域。当Field的属性tokenized（是否分词）为true时会对Field值进行分析。

对于一些Field可以不用分析：

- 1、不作为查询条件的内容，比如文件路径
- 2、不是匹配内容中的词而匹配Field的整体内容，比如订单号、身份证号等。

### 7.2.2. 搜索时使用Analyzer

对搜索关键字进行分析和索引分析一样，使用Analyzer对搜索关键字进行分析、分词处理，使用分析后每个词语进行搜索。比如：搜索关键字：spring web，经过分析器进行分词，得出：spring web拿词去索引词典表查找，找到索引链接到Document，解析Document内容。

对于匹配整体Field域的查询可以在搜索时不分析，比如根据订单号、身份证号查询等。

**注意：**搜索使用的分析器要和索引使用的分析器一致。

## 7.3. Lucene原生分词器

---

以下是Lucene中自带的分词器

## 7.3.1. StandardAnalyzer

特点：

Lucene提供的标准分词器, 可以对用英文进行分词, 对中文是单字分词, 也就是一个字就认为是一个词.

如下是org.apache.lucene.analysis.standard.standardAnalyzer的部分源码：

```
1  protected TokenStreamComponents createComponents(String fieldName) {
2      final StandardTokenizer src = new StandardTokenizer();
3      src.setMaxTokenLength(this.maxTokenLength);
4      TokenStream tok = new LowerCaseFilter(src);
5      TokenStream tok = new StopFilter(tok, this.stopwords);
6      return new TokenStreamComponents(src, tok) {
7          protected void setReader(Reader reader) {
8
9              src.setMaxTokenLength(StandardAnalyzer.this.maxTokenLength);
10             super.setReader(reader);
11         }
12     };
13 }
```

Tokenizer就是分词器，负责将reader转换为语汇单元即进行分词处理，Lucene提供了很多的分词器，也可以使用第三方的分词，比如IKAnalyzer一个中文分词器。

TokenFilter是分词过滤器，负责对语汇单元进行过滤，TokenFilter可以是一个过滤器链儿，Lucene提供了很多的分词器过滤器，比如大小写转换、去除停用词等。

如下图是语汇单元的生成过程：



从一个Reader字符流开始, 创建一个基于Reader的Tokenizer分词器, 经过三个TokenFilter生成语汇单元Token。

比如下边的文档经过分析器分析如下:

原文档内容:

Lucene is a Java full-text search engine.

分析后得到的多个语汇单元:

lucene 、 java 、 full 、 text 、 search 、 engine

## 7.3.2. WhitespaceAnalyzer

特点:

仅仅是去掉了空格, 没有其他任何操作, 不支持中文。

测试代码:

```
1  @Test
2  public void TestWhitespaceAnalyzer() throws Exception{
3      // 1. 创建分词器,分析文档, 对文档进行分词
4      Analyzer analyzer = new WhitespaceAnalyzer();
5
6      // 2. 创建Directory对象,声明索引库的位置
7      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
8
9      // 3. 创建IndexWriterConfig对象, 写入索引需要的配置
10     IndexWriterConfig config = new IndexWriterConfig(analyzer);
11
12     // 4.创建IndexWriter写入对象
13     IndexWriter indexWriter = new IndexWriter(directory, config);
14
15     // 5.写入到索引库, 通过IndexWriter添加文档对象document
16     Document doc = new Document();
17     doc.add(new TextField("name", "vivo X23 8GB+128GB 幻夜蓝",
18         Field.Store.YES));
19     indexWriter.addDocument(doc);
20 }
```

```
20 // 6.释放资源
21 indexWriter.close();
22 }
```

### 7.3.3. SimpleAnalyzer

特点：

将除了字母以外的符号全部去除，并且将所有字母变为小写，需要注意的是这个分词器同样把数字也去除了，同样不支持中文。

测试：

```
1  @Test
2  public void TestSimpleAnalyzer() throws Exception{
3      // 1. 创建分词器,分析文档,对文档进行分词
4      Analyzer analyzer = new SimpleAnalyzer();
5
6      // 2. 创建Directory对象,声明索引库的位置
7      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
8
9      // 3. 创建IndexWriteConfig对象,写入索引需要的配置
10     IndexWriterConfig config = new IndexWriterConfig(analyzer);
11
12     // 4.创建IndexWriter写入对象
13     IndexWriter indexWriter = new IndexWriter(directory, config);
14
15     // 5.写入到索引库,通过IndexWriter添加文档对象document
16     Document doc = new Document();
17     doc.add(new TextField("name", "vivo, X23。 8GB+128GB; 幻夜蓝",
18         Field.Store.YES));
19     indexWriter.addDocument(doc);
20
21     // 6.释放资源
22     indexWriter.close();
23 }
```

## 7.3.4. CJKAnalyzer

特点：

这个支持中日韩文字，前三个字母也就是这三个国家的缩写。对中文是二分法分词，去掉空格，去掉标点符号。个人感觉对中文支持依旧很烂。

代码：

```
1  @Test
2  public void TestCJKAnalyzer() throws Exception{
3      // 1. 创建分词器,分析文档, 对文档进行分词
4      Analyzer analyzer = new CJKAnalyzer();
5
6      // 2. 创建Directory对象,声明索引库的位置
7      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
8
9      // 3. 创建IndexWriterConfig对象, 写入索引需要的配置
10     IndexWriterConfig config = new IndexWriterConfig(analyzer);
11
12     // 4.创建IndexWriter写入对象
13     IndexWriter indexWriter = new IndexWriter(directory, config);
14
15     // 5.写入到索引库, 通过IndexWriter添加文档对象document
16     Document doc = new Document();
17     doc.add(new TextField("name", "vivo, X23。 8GB+128GB; 幻夜蓝",
18         Field.Store.YES));
19     indexWriter.addDocument(doc);
20
21     // 6.释放资源
22     indexWriter.close();
23 }
```

## 7.4. 第三方中文分词器

## 7.4.1. 什么是中文分词器

学过英文的都知道，英文是以单词为单位的，单词与单词之间以空格或者逗号句号隔开。所以对于英文，我们可以简单以空格判断某个字符串是否为一个单词，比如I love China, love 和 China很容易被程序区分开来。

而中文则以字为单位，字又组成词，字和词再组成句子。中文“我爱中国”就不一样了，电脑不知道“中国”是一个词语还是“爱中”是一个词语。

把中文的句子切分成有意义的词，就是中文分词，也称切词。我爱中国，分词的结果是：我、爱、中国。

## 7.4.2. 第三方中文分词器简介

- paoding：庖丁解牛最新版在 <https://code.google.com/p/paoding/> 中最多支持Lucene 3.0，且最新提交的代码在 2008-06-03，在svn中最新也是2010年提交，已经过时，不予考虑。
- mmseg4j：最新版已从 <https://code.google.com/p/mmseg4j/> 移至 <https://github.com/chenlb/mmseg4j-solr>，支持Lucene 4.10，且在github中最新提交代码是2014年6月，从09年~14年一共有：18个版本，也就是一年几乎有3个大小版本，有较大的活跃度，用了mmseg算法。
- IK-analyzer：最新版在<https://code.google.com/p/ik-analyzer/>上，支持Lucene 4.10从2006年12月推出1.0版开始，IKAnalyzer已经推出了4个大版本。最初，它是以开源项目Luence为应用主体的，结合词典分词和文法分析算法的中文分词组件。从3.0版本开始，IK发展为面向Java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现。在2012版本中，IK实现了简单的分词歧义排除算法，标志着IK分词器从单纯的词典分词向模拟语义分词衍化。但是也就是2012年12月后没有在更新。
- ansj\_seg：最新版本在 [https://github.com/NLPchina/ansj\\_seg](https://github.com/NLPchina/ansj_seg) tags仅有1.1版本，从2012年到2014年更新了大小6次，但是作者本人在2014年10月10日说明：“可能我以后没有精力来维护ansj\_seg了”，现在由“nlp\_china”管理。2014年11月有更新。并未说明是否支持Lucene，是一个由CRF（条件随机场）算法所做的分词算法。
- imdict-chinese-analyzer：最新版在 <https://code.google.com/p/imdict-chinese-analyzer/>，最新更新也在2009年5月，下载源码，不支持Lucene 4.10。是利用HMM（隐马尔科夫链）算法。
- Jcseg：最新版本在[git.oschina.net/lionsoul/jcseg](http://git.oschina.net/lionsoul/jcseg)，支持Lucene 4.10，作者有较高的活

跃度。利用mmseg算法。

### 7.4.3. 使用中文分词器IKAnalyzer

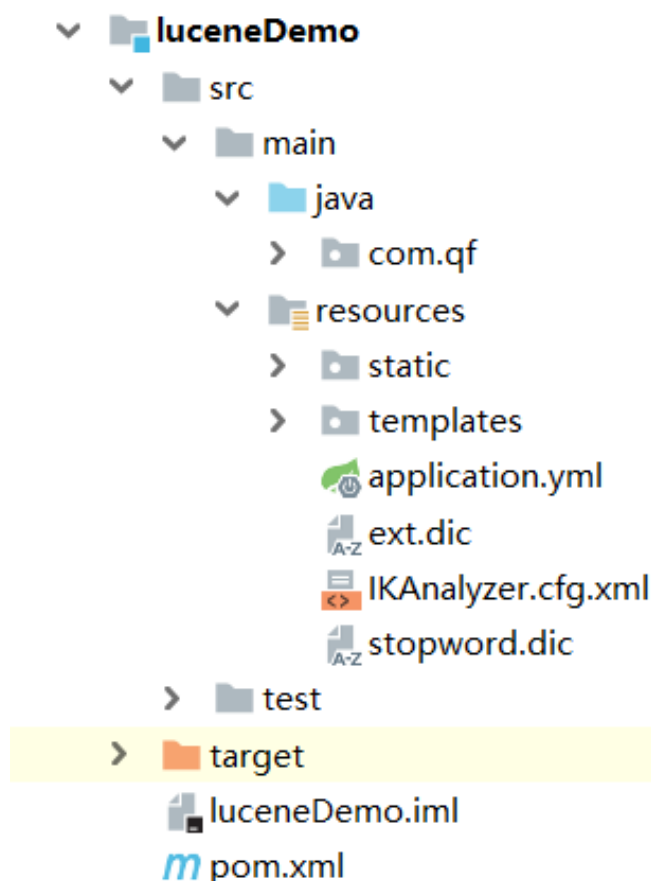
IKAnalyzer继承Lucene的Analyzer抽象类，使用IKAnalyzer和Lucene自带的分析器方法一样，将Analyzer测试代码改为IKAnalyzer测试中文分词效果。

如果使用中文分词器ik-analyzer，就需要在索引和搜索程序中使用一致的分词器：IK-analyzer。

1. 添加依赖, pom.xml中加入依赖

```
1 <dependency>
2     <groupId>org.wltea.ik-analyzer</groupId>
3     <artifactId>ik-analyzer</artifactId>
4     <version>8.1.0</version>
5 </dependency>
```

2. 加入配置文件:



### 3. 测试代码

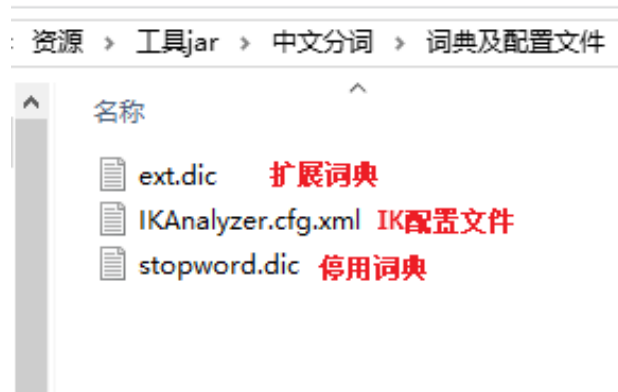
```
1  @Test
2  public void TestIKAnalyzer() throws Exception{
3      // 1. 创建分词器,分析文档,对文档进行分词
4      Analyzer analyzer = new IKAnalyzer();
5
6      // 2. 创建Directory对象,声明索引库的位置
7      Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
8
9      // 3. 创建IndexWriterConfig对象,写入索引需要的配置
10     IndexWriterConfig config = new IndexWriterConfig(analyzer);
11
12     // 4.创建IndexWriter写入对象
13     IndexWriter indexWriter = new IndexWriter(directory, config);
14
15     // 5.写入到索引库,通过IndexWriter添加文档对象document
16     Document doc = new Document();
17     doc.add(new TextField("name", "vivo X23 8GB+128GB 幻夜蓝,水滴屏全面屏,
游戏手机.移动联通电信全网通4G手机", Field.Store.YES));
18     indexWriter.addDocument(doc);
19
20     // 6.释放资源
21     indexWriter.close();
22
23 }
```

#### 7.4.4. 扩展中文词库

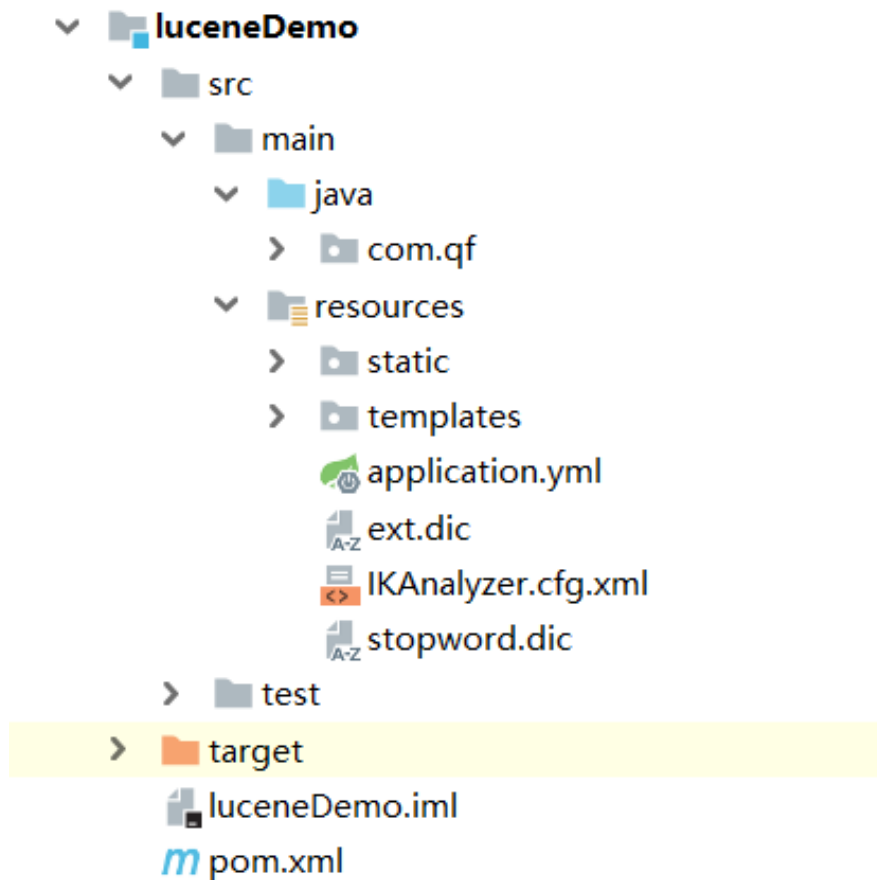
如果想配置扩展词和停用词,就创建扩展词的文件和停用词的文件。

从ikalyzer包中拷贝配置文件





拷贝到资源文件夹中



IKAnalyzer.cfg.xml配置文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3 <properties>
4     <comment>IK Analyzer 扩展配置</comment>
5     <!--用户可以在这里配置自己的扩展字典 -->
6     <entry key="ext_dict">ext.dic;</entry>
7
8     <!--用户可以在这里配置自己的扩展停止词字典-->
9     <entry key="ext_stopwords">stopword.dic;</entry>
10
11 </properties>
12
```

### 停用词典stopword.dic作用：

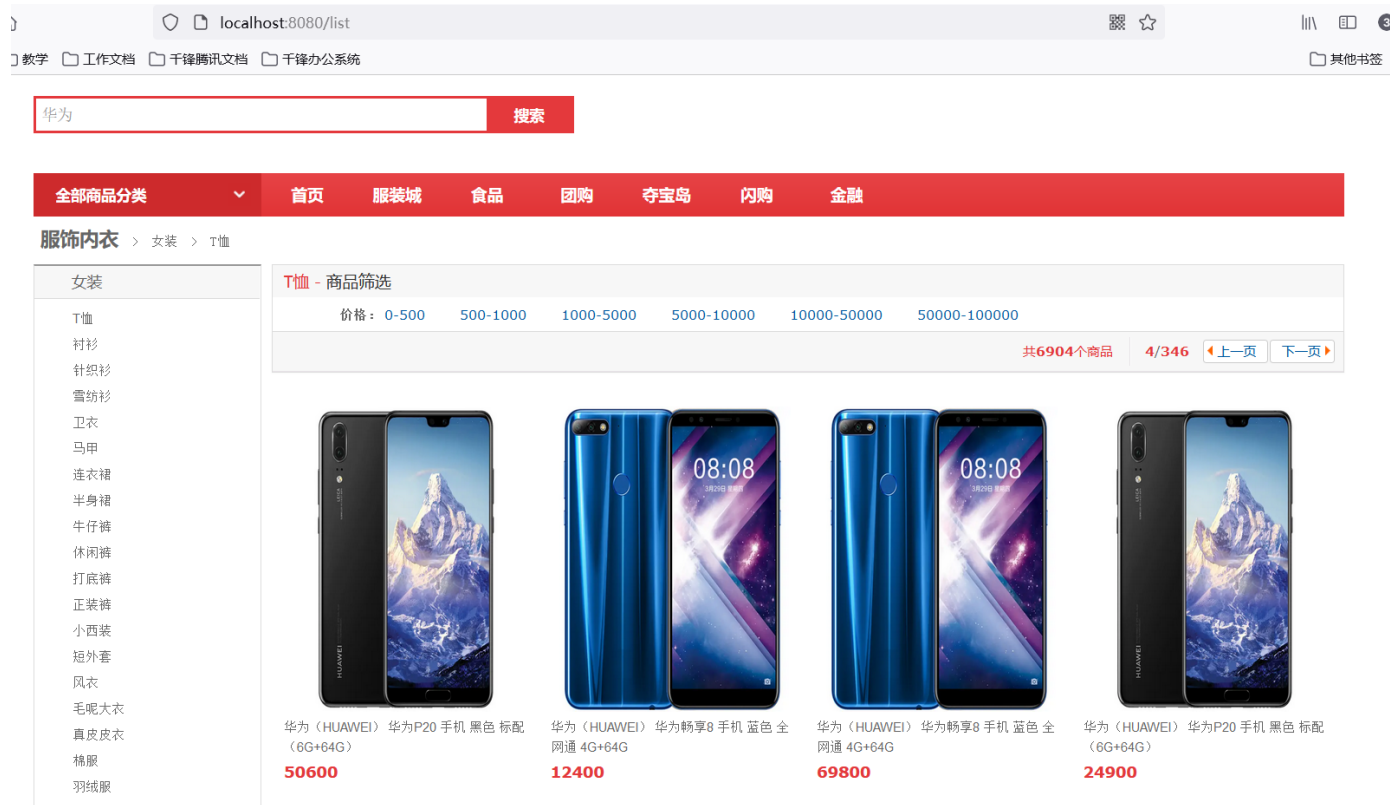
停用词典中的词例如：a, an, the, 的, 地, 得等词汇, 凡是出现在停用词典中的字或者词, 在切分词的时候会被过滤掉.

### 扩展词典ext.dic作用：

扩展词典中的词例如：千锋教育, 贵州茅台等专有名词, 在汉语中一些公司名称, 行业名称, 分类, 品牌等不是汉语中的词汇, 是专有名词. 这些分词器默认不识别, 所以需要放入扩展词典中, 效果是被强制分成一个词.

## 8. 搜索案例

成品效果：



## 8.1. 引入依赖

在项目的pom.xml中引入依赖:

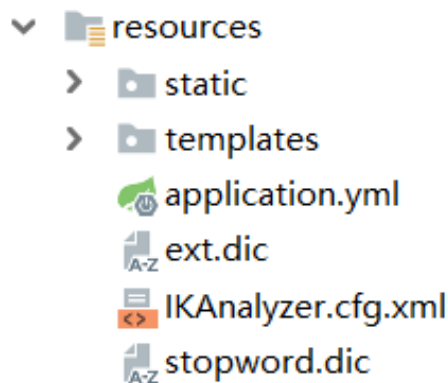
```
1 <properties>
2     <maven.compiler.source>1.8</maven.compiler.source>
3     <maven.compiler.target>1.8</maven.compiler.target>
4     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
5     <project.reporting.outputEncoding>UTF-
6     <skipTests>true</skipTests>
7 </properties>
8
9 <parent>
10     <groupId>org.springframework.boot</groupId>
11     <artifactId>spring-boot-starter-parent</artifactId>
12     <version>2.1.4.RELEASE</version>
13 </parent>
14
15 <dependencies>
16     <dependency>
17         <groupId>commons-io</groupId>
```

```
18         <artifactId>commons-io</artifactId>
19         <version>2.6</version>
20     </dependency>
21
22     <dependency>
23         <groupId>org.apache.lucene</groupId>
24         <artifactId>lucene-core</artifactId>
25         <version>8.11.1</version>
26     </dependency>
27     <dependency>
28         <groupId>org.apache.lucene</groupId>
29         <artifactId>lucene-analyzers-common</artifactId>
30         <version>8.11.1</version>
31         <exclusions>
32             <exclusion>
33                 <groupId>org.apache.lucene</groupId>
34                 <artifactId>lucene-core</artifactId>
35             </exclusion>
36         </exclusions>
37     </dependency>
38     <dependency>
39         <groupId>org.apache.lucene</groupId>
40         <artifactId>lucene-queryparser</artifactId>
41         <version>8.11.1</version>
42         <exclusions>
43             <exclusion>
44                 <groupId>org.apache.lucene</groupId>
45                 <artifactId>lucene-core</artifactId>
46             </exclusion>
47         </exclusions>
48     </dependency>
49
50     <dependency>
51         <groupId>org.apache.lucene</groupId>
52         <artifactId>lucene-backward-codecs</artifactId>
53         <version>8.11.1</version>
54     </dependency>
55
56     <!-- 测试 -->
57     <dependency>
58         <groupId>junit</groupId>
59         <artifactId>junit</artifactId>
60         <version>4.12</version>
```

```
61         <scope>test</scope>
62     </dependency>
63     <!-- mysql数据库驱动 -->
64     <dependency>
65         <groupId>mysql</groupId>
66         <artifactId>mysql-connector-java</artifactId>
67         <version>5.1.48</version>
68     </dependency>
69
70     <!-- IK中文分词器 -->
71     <dependency>
72         <groupId>org.wltea.ik-analyzer</groupId>
73         <artifactId>ik-analyzer</artifactId>
74         <version>8.1.0</version>
75     </dependency>
76
77     <!--web起步依赖-->
78     <dependency>
79         <groupId>org.springframework.boot</groupId>
80         <artifactId>spring-boot-starter-web</artifactId>
81     </dependency>
82     <!-- 引入thymeleaf -->
83     <dependency>
84         <groupId>org.springframework.boot</groupId>
85         <artifactId>spring-boot-starter-thymeleaf</artifactId>
86     </dependency>
87     <!-- Json转换工具 -->
88     <dependency>
89         <groupId>com.alibaba</groupId>
90         <artifactId>fastjson</artifactId>
91         <version>1.2.51</version>
92     </dependency>
93 </dependencies>
```

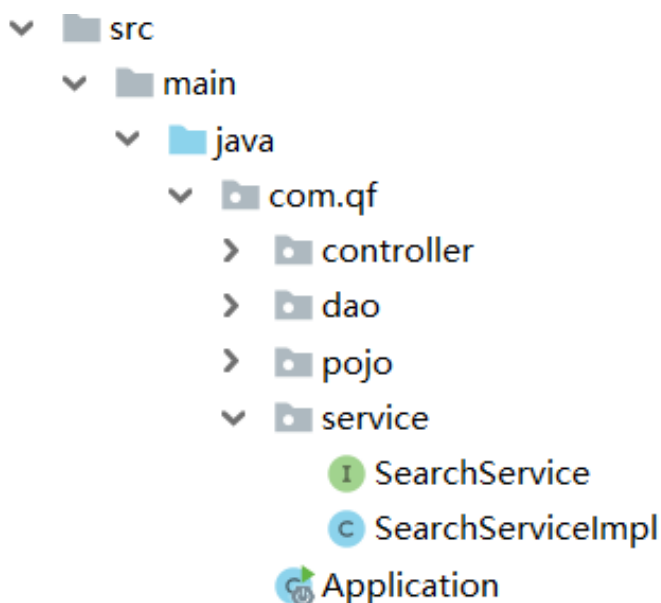
## 8.2. 项目加入页面和资源

将Lucene课程资料\资源\页面和静态资源, 下的页面和静态资源拷贝到项目的resources目录下



## 8.3. 创建包和启动类

创建目录, 并加入启动类:



启动类代码:

```
1 @SpringBootApplication
2 public class LuceneApplication {
3     public static void main(String[] args) {
4         SpringApplication.run(LuceneApplication.class, args);
5     }
6 }
```

## 8.4. 配置文件

项目的resources目录下创建application.yml内容如下:

```
1 spring:
2     thymeleaf:
3         cache: false
```

## 8.5. 业务代码:

### 8.5.1. 封装pojo

pojo包下加入ResultModel实体类

```
1 public class ResultModel {
2
3     // 商品列表
4     private List<Sku> skuList;
5     // 商品总数
6     private Long recordCount;
7     // 总页数
8     private Long pageCount;
9     // 当前页
10    private long curPage;
11
12    .....get和set方法.....略
```

### 8.5.2. controller代码

```
1 @Controller
2 @RequestMapping("/list")
3 public class SearchController {
4
5     @Autowired
6     private SearchService searchService;
7
8     @RequestMapping
9     public String list(String queryString, String price, Integer page,
10    Model model) throws Exception {
11         if (StringUtils.isEmpty(page)) {
12             page = 1;
13         }
14         if (page <= 0) {
```

```
14         page = 1;
15     }
16
17     ResultModel resultModel = searchService.search(queryString,
18 price, page);
19
20     model.addAttribute("result", resultModel);
21     model.addAttribute("queryString", queryString);
22     model.addAttribute("price", price);
23     model.addAttribute("page", page);
24     return "search";
25 }
```

### 8.5.3. service 代码

service 接口:

```
1 public interface SearchService {
2
3     /**
4      * 根据关键字全文检索
5      *
6      * @param queryString 查询关键字
7      * @param price 价格过滤条件
8      * @param page 当前页
9      */
10    public ResultModel search(String queryString, String price, Integer
11 page) throws Exception;
12 }
```

service 实现类:

```
1 package com.qf.service;
2
3 import com.qf.pojo.ResultModel;
4 import com.qf.pojo.Sku;
5 import org.apache.lucene.analysis.Analyzer;
```



```
6 import org.apache.lucene.document.Document;
7 import org.apache.lucene.document.IntPoint;
8 import org.apache.lucene.index.DirectoryReader;
9 import org.apache.lucene.index.IndexReader;
10 import org.apache.lucene.queryparser.classic.QueryParser;
11 import org.apache.lucene.search.*;
12 import org.apache.lucene.store.Directory;
13 import org.apache.lucene.store.FSDirectory;
14 import org.springframework.stereotype.Service;
15 import org.springframework.util.StringUtils;
16 import org.wltea.analyzer.lucene.IKAnalyzer;
17
18 import java.nio.file.Paths;
19 import java.util.ArrayList;
20 import java.util.List;
21
22 /**
23  *
24  */
25 @Service
26 public class SearchServiceImpl implements SearchService {
27
28     //每页查询20条数据
29     public final static Integer PAGE_SIZE = 20;
30
31     @Override
32     public ResultModel search(String queryString, String price,
33 Integer page) throws Exception {
34
35         //1. 需要使用的对象封装
36         ResultModel resultModel = new ResultModel();
37         //从第几条开始查询
38         int start = (page - 1) * PAGE_SIZE;
39         //查询到多少条为止
40         Integer end = page * PAGE_SIZE;
41         //创建分词器
42         Analyzer analyzer = new IKAnalyzer();
43         //创建组合查询对象
44         BooleanQuery.Builder builder = new BooleanQuery.Builder();
45
46         //2. 根据查询关键字封装查询对象
47         QueryParser queryParser = new QueryParser("name", analyzer);
```

```

48     Query query1 = null;
49     //判断传入的查询关键字是否为空，如果为空查询所有，如果不为空，则根据关键字查询
50     if (StringUtils.isEmpty(queryString)) {
51         query1 = queryParser.parse("*:");
52     } else {
53         query1 = queryParser.parse(queryString);
54     }
55     //将关键字查询对象，封装到组合查询对象中
56     builder.add(query1, BooleanClause.Occur.MUST);
57
58     //3. 根据价格范围封装查询对象
59     if (!StringUtils.isEmpty(price)) {
60         String[] split = price.split("-");
61         Query query2 = IntPoint.newRangeQuery("price", Integer.parseInt(split[0]), Integer.parseInt(split[1]));
62         //将价格查询对象，封装到组合查询对象中
63         builder.add(query2, BooleanClause.Occur.MUST);
64     }
65
66     //4. 创建Directory目录对象，指定索引库的位置
67     /**
68      * 使用MMapDirectory消耗的查询时间
69      * ====消耗时间为=====324ms
70      * ====消耗时间为=====18ms
71      */
72     Directory directory = FSDirectory.open(Paths.get("E:\\dir"));
73     //5. 创建输入流对象
74     IndexReader reader = DirectoryReader.open(directory);
75     //6. 创建搜索对象
76     IndexSearcher indexSearcher = new IndexSearcher(reader);
77     //7. 搜索并获取搜索结果
78     TopDocs topDocs = indexSearcher.search(builder.build(), end);
79     //8. 获取查询到的总条数
80     resultModel.setRecordCount(topDocs.totalHits.value);
81     //9. 获取查询到的结果集
82     ScoreDoc[] scoreDocs = topDocs.scoreDocs;
83
84     long endTime = System.currentTimeMillis();
85     System.out.println("====消耗时间为=====" + (endTime - startTime) + "ms");
86
87     //10. 遍历结果集封装返回的数据

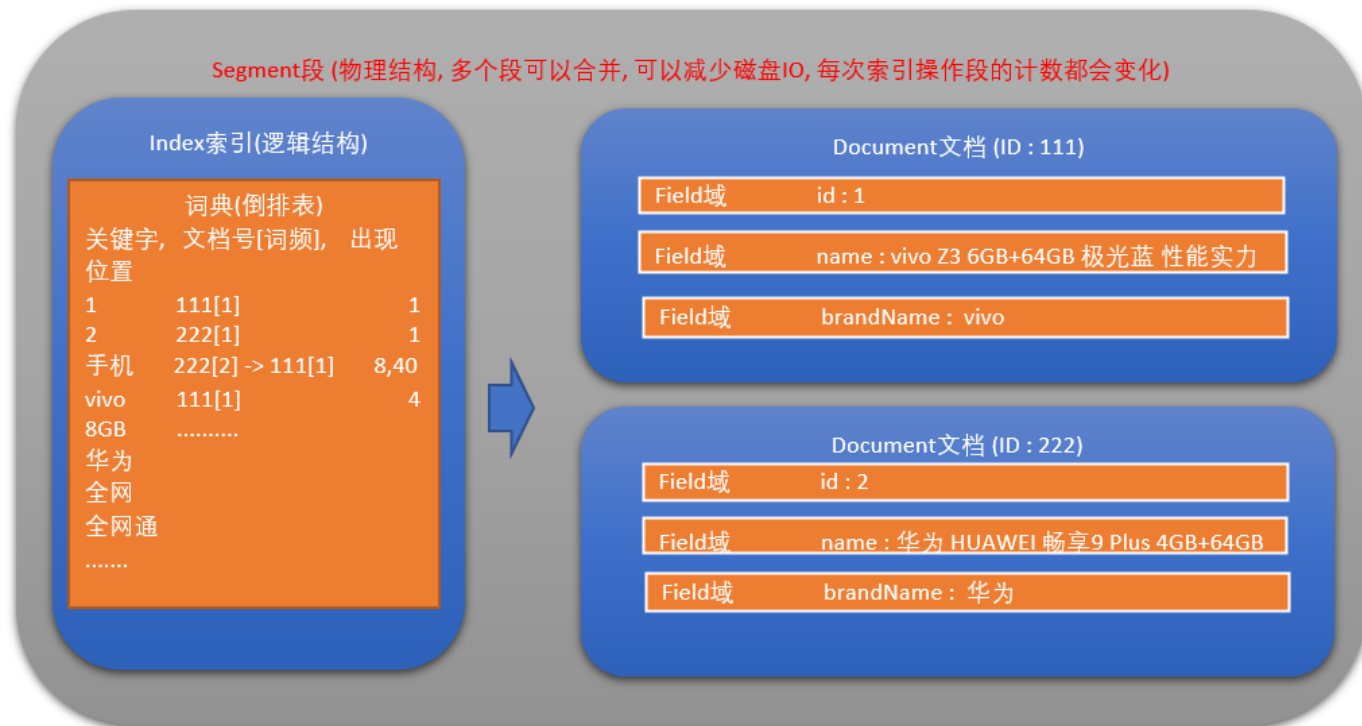
```

```
88     List<Sku> skuList = new ArrayList<>();
89     if (scoreDocs != null) {
90         for (int i = start; i < end; i++) {
91             //通过查询到的文档编号, 找到对应的文档对象
92             Document document = reader.document(scoreDocs[i].doc);
93             //封装Sku对象
94             Sku sku = new Sku();
95             sku.setId(document.get("id"));
96             sku.setPrice(Integer.parseInt(document.get("price")));
97             sku.setImage(document.get("image"));
98             sku.setName(document.get("name"));
99             sku.setBrandName(document.get("brandName"));
100            sku.setCategoryName(document.get("categoryName"));
101
102            skuList.add(sku);
103        }
104    }
105    //封装查询到的结果集
106    resultModel.setSkuList(skuList);
107    //封装当前页
108    resultModel.setCurPage(page);
109    //总页数
110    Long pageCount = topDocs.totalHits.value % PAGE_SIZE > 0 ?
    (topDocs.totalHits.value/PAGE_SIZE) + 1 :
    topDocs.totalHits.value/PAGE_SIZE;
111    resultModel.setPageCount(pageCount);
112    return resultModel;
113 }
114 }
```

## 9. Lucene底层储存结构(高级)

### 9.1. 详细理解lucene存储结构

存储结构：



## 索引库(Index) :

- 一个目录一个索引库, 在Lucene中一个索引是放在一个文件夹中的。

## 段(Segment) :

- 一个索引(逻辑索引)由多个段组成, 多个段可以合并, 以减少读取内容时候的磁盘IO。
- Lucene中的数据写入会先写内存的一个Buffer, 当Buffer内数据到一定量后会被flush成一个Segment, 每个Segment有自己独立的索引, 可独立被查询, 但数据永远不能被更改。这种模式避免了随机写, 数据写入都是批量追加, 能达到很高的吞吐量。Segment中写入的文档不可被修改, 但可被删除, 删除的方式也不是在文件内部原地更改, 而是会由另外一个文件保存需要被删除的文档的DocID, 保证数据文件不可被修改。Index的查询需要对多个Segment进行查询并对结果进行合并, 还需要处理被删除的文档, 为了对查询进行优化, Lucene会有策略对多个Segment进行合并。

## 文档(Document) :

- 文档是我们建索引的基本单位, 不同的文档是保存在不同的段中的, 一个段可以包含多篇文档。
- 新添加的文档是单独保存在一个新生成的段中, 随着段的合并, 不同的文档合并到同一个段中。

## 域(Field) :

- 一篇文档包含不同类型的信息, 可以分开索引, 比如标题, 时间, 正文, 描述等, 都可以保存在不同的域里。









- 不同域的索引方式可以不同。

词(Term)：

- 词是索引的最小单位，是经过词法分析和语言处理后的字符串。

## 9.2. 索引库物理文件

---

名称	
 _3.cfs	
 _4.cfs	
 _5.cfs	
 _6.cfe	
 _6.cfs	
 _6.si	
 segments_b	
 write.lock	

## 9.3. 索引库文件扩展名对照表

---

名称	文件扩展名	简短描述
Segments File	segments_N	保存了一个提交点（a commit point）的信息
Lock File	write.lock	防止多个IndexWriter同时写到一份索引文件中
Segment Info	.si	保存了索引段的元数据信息
Compound File	.cfs, .cfe	一个可选的虚拟文件，把所有索引信息都存储到复合索引文件中
Fields	.fnm	保存fields的相关信息
Field Index	.fdx	保存指向field data的指针
Field Data	.fdt	文档存储的字段的价值
Term Dictionary	.tim	term词典，存储term信息
Term Index	.tip	到Term Dictionary的索引
Frequencies	.doc	由包含每个term以及频率的docs列表组成
Positions	.pos	存储出现在索引中的term的位置信息
Payloads	.pay	存储额外的per-position元数据信息，例如字符偏移和用户payloads
Norms	.nvd, .nvm	.nvm文件保存索引字段加权因子的元数据，.nvd文件保存索引字段加权数据
Per-Document Values	.dvd, .dvm	.dvm文件保存索引文档评分因子的元数据，.dvd文件保存索引文档评分数据
Term Vector Index	.tvx	将偏移存储到文档数据文件中
Term Vector Documents	.tvd	包含有term vectors的每个文档信息
Term Vector Fields	.tvf	字段级别有关term vectors的信息
Live Documents	.liv	哪些是有效文件的信息
Point values	.dii, .dim	保留索引点，如果有的话

## 9.4. 词典的构建

为何Lucene大数据量搜索快, 要分两部分来看：

- 一点是因为底层的倒排索引存储结构.
- 另一点就是查询关键字的时候速度快, 因为词典的索引结构.

### 9.4.1. 词典数据结构对比

倒排索引中的词典位于内存，其结构尤为重要，有很多种词典结构，各有各的优缺点，最简单如排序数组，通过二分查找来检索数据，更快的有哈希表，磁盘查找有B树、B+树，但一个能支持TB级数据的倒排索引结构需要在时间和空间上有个平衡，下图列了一些常见词典的优缺点：

数据结构	优缺点
跳跃表	占用内存小，且可调，但是对模糊查询支持不好
排序列表Array/List	使用二分法查找，不平衡
字典树	查询效率跟字符串长度有关，但只适合英文词典
哈希表	性能高，内存消耗大，几乎是原始数据的三倍
双数组字典树	适合做中文词典，内存占用小，很多分词工具均采用此种算法
Finite State Transducers (FST)	一种有限状态转移机，Lucene 4有开源实现，并大量使用
B树	磁盘索引，更新方便，但检索速度慢，多用于数据库

Lucene3.0之前使用的也是跳跃表结构，后换成了FST，但跳跃表在Lucene其他地方还有应用如倒排表合并和文档号索引。

### 9.4.2. 跳跃表原理

Lucene3.0版本之前使用的跳跃表结构后换成了FST结构

**优点：**结构简单、跳跃间隔、级数可控，Lucene3.0之前使用的也是跳跃表结构，，但跳跃表在Lucene其他地方还有应用如倒排表合并和文档号索引。

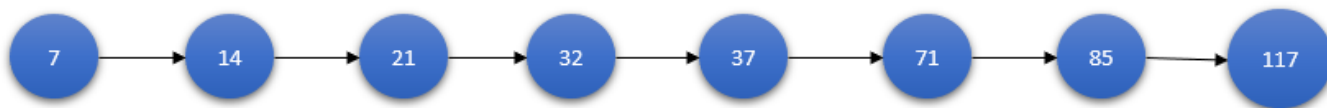
**缺点：**模糊查询支持不好.

## 单链表：

单链表中查询一个元素即使是有序的，我们也不能通过二分查找法的方式缩减查询时间。

通俗的讲也就是按照链表顺序一个一个找。

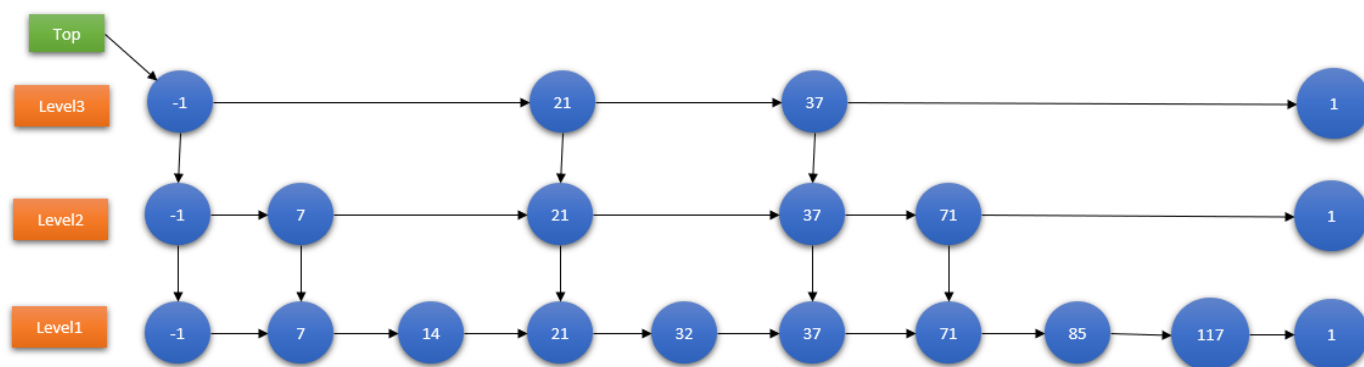
举例：查找85这个节点，需要查找7次。



## 跳跃表：

举例：查询85这个节点，一共需要查询6次。

1. 在level3层，查询3次，查询到1结尾，退回到37节点
2. 在level2层，从37节点开始查询，查询2次，查询到1结尾，退回到71节点
3. 在level1层，从71节点开始查询，查询1次，查询到85节点。



## 9.4.3. FST原理简析

FST, 全称Finite State Transducer, 中文翻译: 有限状态转换器或有限状态传感器。

FST最重要的功能是可以实现Key到Value的映射，相当于HashMap<Key,Value>。FST的内存消耗要比HashMap少很多，但FST的查询速度比HashMap要慢。

FST在Lucene中被大量使用，例如：倒排索引的存储，同义词词典的存储，搜索关键字建议等。

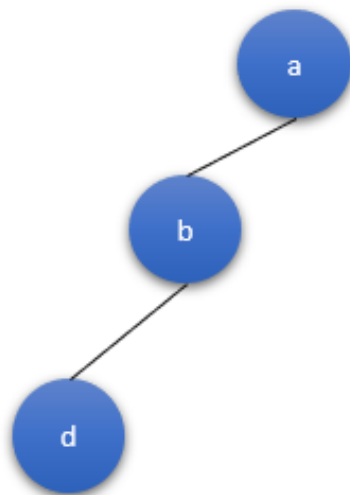
Lucene现在采用的数据结构为FST，它的特点就是：

**优点：**内存占用率低，压缩率一般在3倍~20倍之间、模糊查询支持好、查询快

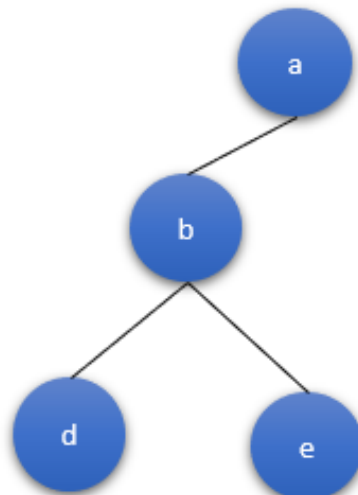
**缺点：**结构复杂、输入要求有序、更新不易



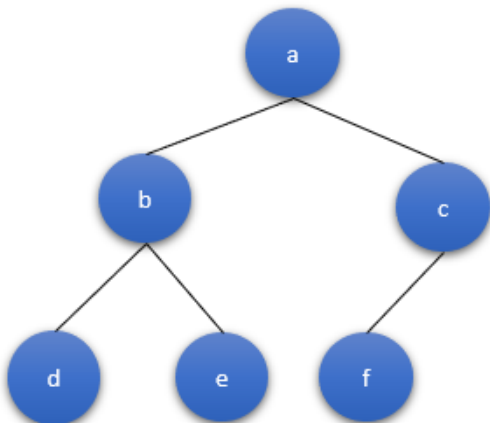
已知FST要求输入有序，所以Lucene会将解析出来的文档单词预先排序，然后构建FST，我们假设输入为abd,abe,acf,acg，那么整个构建过程如下：



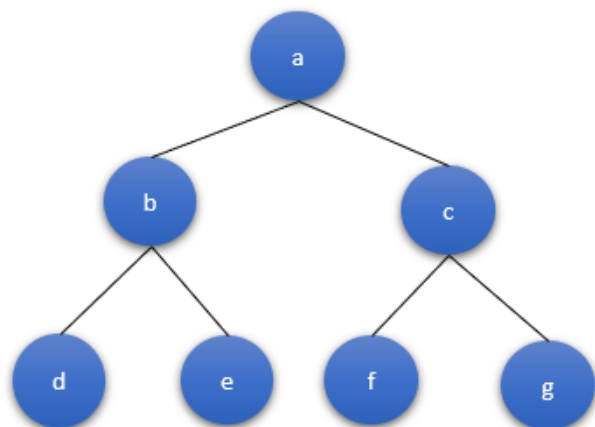
1.插入abd



2.插入abe



3.插入acf



4.插入acg

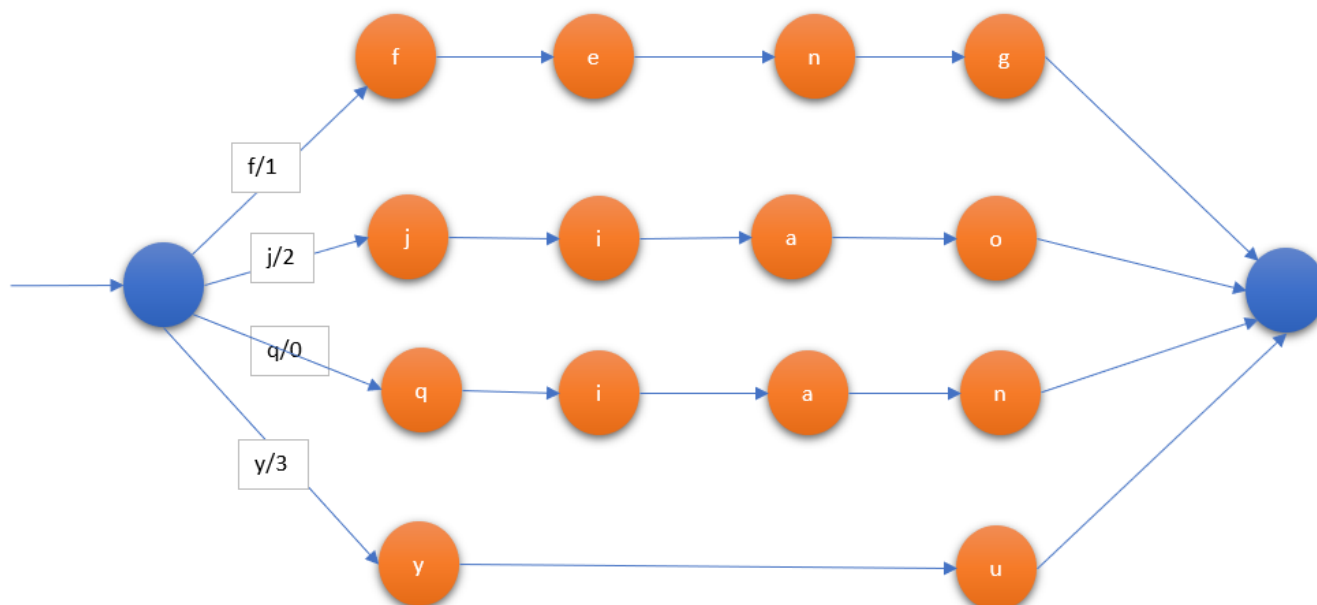
输入数据:

```
1 String inputValues[] = {"qian","feng","jiao","yu"};  
2 long outputValues[] = {0,1,2,3};
```

输入的数据如下:

qian/0  
feng/1  
jiao/2  
yu/3

存储结果如下:



## 10. Lucene相关度排序(高级)

### 10.1.什么是相关度排序

Lucene对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。

#### 10.1.1. 如何打分

Lucene是在用户进行检索时实时根据搜索的关键字计算出来的，分两步：

1. 计算出词（Term）的权重
2. 根据词的权重值，计算文档相关度得分。

## 10.1.2. 什么是词的权重

明确索引的最小单位是一个Term(索引词典中的一个词), 搜索也是要从Term中搜索, 再根据Term找到文档, Term对文档的重要性称为权重, 影响Term权重有两个因素:

- Term Frequency (tf):  
指此Term在此文档中出现了多少次。tf 越大说明越重要。词(Term)在文档中出现的次数越多, 说明此词(Term)对该文档越重要, 如“Lucene”这个词, 在文档中出现的次数很多, 说明该文档主要就是讲Lucene技术的。
- Document Frequency (df):  
指有多少文档包含此Term。df 越大说明越不重要。  
比如, 在一篇英语文档中, this出现的次数更多, 就说明越重要吗? 不是的, 有越多的文档包含此词(Term), 说明此词(Term)太普通, 不足以区分这些文档, 因而重要性越低。

## 10.1.3. 怎样影响相关度排序

boost是一个加权值(默认加权值为1.0f), 它可以影响权重的计算。

- 在索引时对某个文档中的field设置加权值高, 在搜索时匹配到这个文档就可能排在前边。
- 在搜索时对某个域进行加权, 在进行组合域查询时, 匹配到加权值高的域最后计算的相关度得分就高。

设置boost是给域(field) 或者Document设置的。

## 10.2.人为影响相关度排序

查询的时候, 通过设置查询域的权重, 可以人为影响查询结果.

```
1  @Test
2  public void testIndexSearch() throws Exception {
3
4      long startTime = System.currentTimeMillis();
5
6      // 1. 创建Query搜索对象
7      // 创建分词器
8      Analyzer analyzer = new IKAnalyzer();
9
10     //查询的域名
11     String[] fields = {"name", "brandName", "categoryName"};
```

```
12 //设置权重
13 Map<String, Float> boots = new HashMap<>();
14 boots.put("categoryName", 1000000f);
15 // 根据多个域进行搜索
16 MultiFieldQueryParser queryParser = new
MultiFieldQueryParser(fields, analyzer, boots);
17 // 创建搜索对象
18 Query query = queryParser.parse("手机");
19
20 // 2. 创建Directory流对象,声明索引库位置
21 Directory directory = MMapDirectory.open(Paths.get("E:\\dir"));
22
23 // 3. 创建索引读取对象IndexReader
24 IndexReader reader = DirectoryReader.open(directory);
25
26 // 4. 创建索引搜索对象
27 IndexSearcher searcher = new IndexSearcher(reader);
28
29 // 5. 使用索引搜索对象,执行搜索,返回结果集TopDocs
30 // 第一个参数: 搜索对象, 第二个参数: 返回的数据条数, 指定查询结果最顶部的n条数据
返回
31 TopDocs topDocs = searcher.search(query, 50);
32 System.out.println("查询到的数据总条数是: " + topDocs.totalHits.value);
33 // 获取查询结果集
34 ScoreDoc[] docs = topDocs.scoreDocs;
35
36 // 6. 解析结果集
37 for (ScoreDoc scoreDoc : docs) {
38     // 获取文档
39     int docID = scoreDoc.doc;
40     Document doc = searcher.doc(docID);
41
42     System.out.println("=====");
43     System.out.println("docID:" + docID);
44     System.out.println("id:" + doc.get("id"));
45     System.out.println("name:" + doc.get("name"));
46     System.out.println("price:" + doc.get("price"));
47     System.out.println("brandName:" + doc.get("brandName"));
48     System.out.println("image:" + doc.get("image"));
49 }
50 // 7. 释放资源
51 reader.close();
52
```

```
53     long endTime = System.currentTimeMillis();
54     System.out.println("=====消耗时间:======" + (startTime -
endTime) + "ms");
55 }
```

千锋教育Java教研院 关注公众号【Java架构栈】 下载所有课程代码课件及工具 让技术回归本该有的纯静!