博客地址：[http://www.moonxy.com](http://www.moonxy.com/)

**一、前言**

Lucene 是 apache 软件基金会的一个子项目，由 Doug Cutting 开发，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库，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。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源工具。就其本身而言，Lucene 是当前以及最近几年最受欢迎的免费 Java 信息检索程序库。人们经常提到信息检索程序库，虽然与搜索引擎有关，但不应该将信息检索程序库与搜索引擎相混淆。

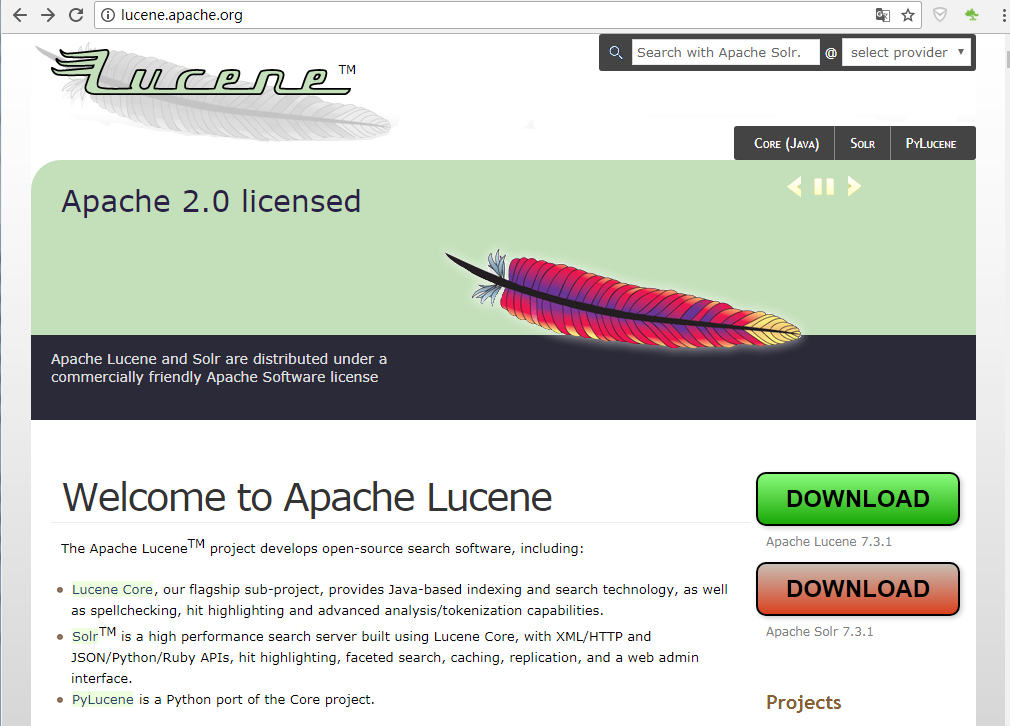
最开始 Lucene 只由 java 开发，供 java 程序调用，随着 python 越来越火，Lucene 官网也提供了 python 版本的 lucene 库，供 python 程序调用，即 [PyLucene](http://lucene.apache.org/pylucene/)。



**二、下载 Lucene**

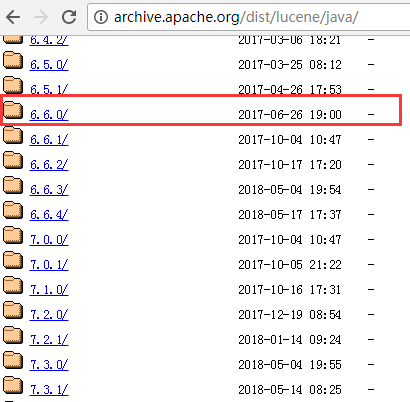
**2.1 下载**

访问 Lucene 官网 <http://lucene.apache.org/>，可以看到 绿色和红色两个下载按钮，分别提供 Lucene 和 Solr 的下载。



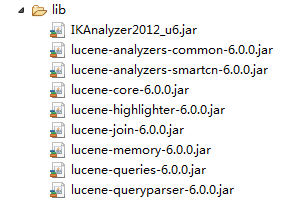
这里简要说明一下 Lucene 和 Solr，Lucene 是一个做全文检索的库，开发者可以按照自己的实际业务需求来使用，而 Solr 是一个基于 Lucene 的全文检索服务器。Solr 是在 Lucene 的基础上进行扩展，并且提供了更加丰富的查询语句，可扩展性和可配置性比 Lucene 更高。除此之外 Solr 还提供了一个完善的管理页面，是一个产品级的全文搜索引擎。

官网首页提供了最新版本的下载链接，如果需要下载使用历史版本，可以访问 <http://archive.apache.org/dist/lucene/java/>，可以下载 Lucene 所有的发行版本。此处下载 6.6.0 版本。



**2.2 添加依赖**

将下载的 Lucene 包解压之后，找到如下的 jar 包，新建自己的工程，此处不使用 Maven，所以手动添加 jar 包到工程的 lib 目录下，如下：



IKAnalyzer2012\_u6.jar，此 jar 包在 IK 分词器项目中，是单独的一个工具包，需要额外在网上下载：IKAnalyzer2012\_u6。**IK 分词器采用了特有的 "正向迭代最细粒度切分算法"，即从左到右的 正向最大（最长）和最小（最短）匹配，支持细粒度和智能分词两种切分模式，可将分词器扩展配置文件 IKAnalyzer.cfg.xml 放在项目的 class 根目录，并在其中配置扩展词典路径。**当 IKAnalyzer6x() 构造方法参数为空或者 false 时，是最细粒度分词，为 true 时是智能分词。

lucene-analyzers-common-6.6.0.jar：lucene-6.6.0/common/

lucene-analyzers-smartcn-6.6.0.jar：lucene-6.6.0/smartcn/

lucene-core-6.6.0.jar：lucene-6.6.0/core/

lucene-highlighter-6.6.0.jar：lucene-6.6.0/highlighter/

lucene-memory-6.6.0.jar：lucene-6.6.0/memory/

lucene-queries-6.6.0.jar：lucene-6.6.0/queries/

lucene-queryparser-6.6.0.jar：lucene-6.6.0/queryparser/

**2.3 Lucene 架构**

首先是信息采集的过程，文件系统、数据库、万维网以及手工输入的文件都可以作为信息采集的对象，也是要搜索的文档的来源，采集万维网上的信息一般使用网络爬虫。完成信息采集之后到 Lucene 层面主要有两个任务：**索引文档**和**搜索文档**。

**索引文档**的过程完成由原始文档到倒排索引的构建过程；

**搜索文档**用以处理用户查询。然后当用户输入查询关键词，Lucene 完成文档搜索任务，经过**分词、匹配、评分、排序**等一系列过程之后返回用户想要的文档。

**倒排索引（Inverted index）**，也常被称为反向索引，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射，它是文档检索系统中最常用的数据结构，包括词项所在的文章号、词项频率、词项位置等。

**三、Lucene 索引详解**

**3.1 创建实体模型**

创建新闻实体类模型

[复制代码](javascript:void(0);)

package tup.lucene.index;

/\*\*

\* 新闻实体类

\* @author moonxy

\*

\*/

public class News {

private int id;//新闻id

private String title;//新闻标题

private String content;//新闻内容

private int reply;//评论数

public News() {

}

public News(int id, String title, String content, int reply) {

super();

this.id = id;

this.title = title;

this.content = content;

this.reply = reply;

}

// 省略 setter 和 getter 方法

}

[复制代码](javascript:void(0);)

**3.2 创建索引**

Lucene 索引文档需要依靠 IndexWriter 对象，创建 IndexWriter 需要两个参数：一个是 IndexWriterConfig 对象，该对象可以设置创建索引使用哪种分词器，另一个是索引的保存路径。IndexWriter 对象的 addDocument() 方法用于添加文档，该方法的参数为 Document 对象，IndexWriter 对象一次可以添加多个文档，最后调用 commit() 方法生成索引。

[复制代码](javascript:void(0);)

package tup.lucene.index;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.Date;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.document.Document;

import org.apache.lucene.document.Field;

import org.apache.lucene.document.FieldType;

import org.apache.lucene.document.IntPoint;

import org.apache.lucene.document.StoredField;

import org.apache.lucene.index.IndexOptions;

import org.apache.lucene.index.IndexWriter;

import org.apache.lucene.index.IndexWriterConfig;

import org.apache.lucene.index.IndexWriterConfig.OpenMode;

import org.apache.lucene.store.Directory;

import org.apache.lucene.store.FSDirectory;

import tup.lucene.ik.IKAnalyzer6x;

/\*\*

\* Lucene 创建索引

\* @author moonxy

\*

\*/

public class CreateIndex {

public static void main(String[] args) {

// 创建3个News对象

News news1 = new News();

news1.setId(1);

news1.setTitle("安倍晋三本周会晤特朗普 将强调日本对美国益处");

news1.setContent("日本首相安倍晋三计划2月10日在华盛顿与美国总统特朗普举行会晤时提出加大日本在美国投资的设想");

news1.setReply(672);

News news2 = new News();

news2.setId(2);

news2.setTitle("北大迎4380名新生 农村学生700多人近年最多");

news2.setContent("昨天，北京大学迎来4380名来自全国各地及数十个国家的本科新生。其中，农村学生共700余名，为近年最多...");

news2.setReply(995);

News news3 = new News();

news3.setId(3);

news3.setTitle("特朗普宣誓(Donald Trump)就任美国第45任总统");

news3.setContent("当地时间1月20日，唐纳德·特朗普在美国国会宣誓就职，正式成为美国第45任总统。");

news3.setReply(1872);

// 开始时间

Date start = new Date();

System.out.println("\*\*\*\*\*\*\*\*\*\*开始创建索引\*\*\*\*\*\*\*\*\*\*");

// 创建IK分词器

Analyzer analyzer = new IKAnalyzer6x();//使用IK最细粒度分词

IndexWriterConfig icw = new IndexWriterConfig(analyzer);  
　　　　　// CREATE 表示先清空索引再重新创建

icw.setOpenMode(OpenMode.CREATE);

Directory dir = null;

IndexWriter inWriter = null;

// 存储索引的目录

Path indexPath = Paths.get("indexdir");

try {

if (!Files.isReadable(indexPath)) {

System.out.println("索引目录 '" + indexPath.toAbsolutePath() + "' 不存在或者不可读,请检查");

System.exit(1);

}

dir = FSDirectory.open(indexPath);

inWriter = new IndexWriter(dir, icw);

// 设置新闻ID索引并存储

FieldType idType = new FieldType();

idType.setIndexOptions(IndexOptions.DOCS);

idType.setStored(true);

// 设置新闻标题索引文档、词项频率、位移信息和偏移量，存储并词条化

FieldType titleType = new FieldType();

titleType.setIndexOptions(IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS);

titleType.setStored(true);

titleType.setTokenized(true);

FieldType contentType = new FieldType();

contentType.setIndexOptions(IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS);

contentType.setStored(true);

contentType.setTokenized(true);

contentType.setStoreTermVectors(true);

contentType.setStoreTermVectorPositions(true);

contentType.setStoreTermVectorOffsets(true);

Document doc1 = new Document();

doc1.add(new Field("id", String.valueOf(news1.getId()), idType));

doc1.add(new Field("title", news1.getTitle(), titleType));

doc1.add(new Field("content", news1.getContent(), contentType));

doc1.add(new IntPoint("reply", news1.getReply()));

doc1.add(new StoredField("reply\_display", news1.getReply()));

Document doc2 = new Document();

doc2.add(new Field("id", String.valueOf(news2.getId()), idType));

doc2.add(new Field("title", news2.getTitle(), titleType));

doc2.add(new Field("content", news2.getContent(), contentType));

doc2.add(new IntPoint("reply", news2.getReply()));

doc2.add(new StoredField("reply\_display", news2.getReply()));

Document doc3 = new Document();

doc3.add(new Field("id", String.valueOf(news3.getId()), idType));

doc3.add(new Field("title", news3.getTitle(), titleType));

doc3.add(new Field("content", news3.getContent(), contentType));

doc3.add(new IntPoint("reply", news3.getReply()));

doc3.add(new StoredField("reply\_display", news3.getReply()));

inWriter.addDocument(doc1);

inWriter.addDocument(doc2);

inWriter.addDocument(doc3);

inWriter.commit();

inWriter.close();

dir.close();

} catch (IOException e) {

e.printStackTrace();

}

Date end = new Date();

System.out.println("索引文档用时:" + (end.getTime() - start.getTime()) + " milliseconds");

System.out.println("\*\*\*\*\*\*\*\*\*\*索引创建完成\*\*\*\*\*\*\*\*\*\*");

}

}

[复制代码](javascript:void(0);)

执行之后，在控制台输出如下：

\*\*\*\*\*\*\*\*\*\*开始创建索引\*\*\*\*\*\*\*\*\*\*

加载扩展词典：dict/ext.dic

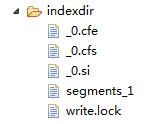
加载扩展停止词典：dict/stopword.dic

加载扩展停止词典：dict/ext\_stopword.dic

索引文档用时:1064 milliseconds

\*\*\*\*\*\*\*\*\*\*索引创建完成\*\*\*\*\*\*\*\*\*\*

并且在项目中生成如下索引文件：

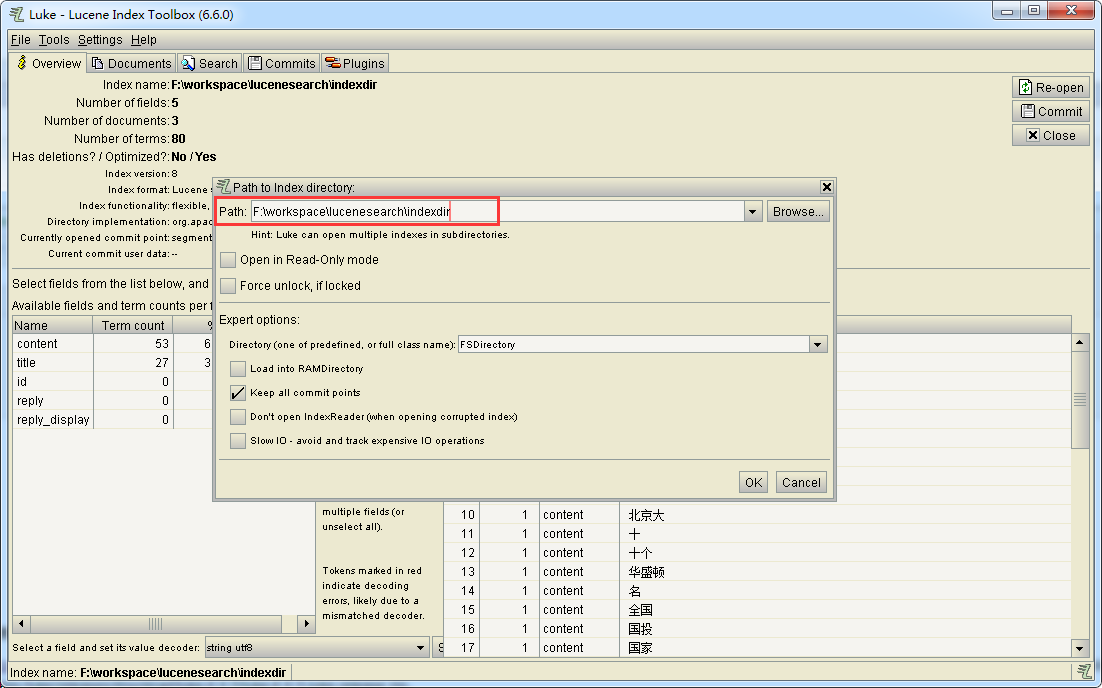


**3.3 Luke 查看索引**

索引创建完成以后生成了如上的一批特殊格式的文件，如果直接用工具打开，会显示的都是乱码。可以使用索引查看工具 Luke 来查看。

Luke 是开源工具，代码托管在 GitHub 上，项目地址：<https://github.com/DmitryKey/luke/releases>，此处下载 luke 6.6.0，地址为 https://github.com/DmitryKey/luke/releases/download/luke-6.6.0/luke-6.6.0-luke-release.zip，如果在 Windows 中无法下载，可以在 Linux 中使用 wget 下载，命令为：wget https://github.com/DmitryKey/luke/releases/download/luke-6.6.0/luke-6.6.0-luke-release.zip。

下载后解压，进入 luke 目录，如果是在 Linux 平台，运行 luke.bat 即可启动软件，并在 Path 中输入 index 存储的目录，即可打开索引文件，显示出索引的具体内容。



注意：对于不同版本的 Lucene，需要选择对应版本的 Luke，否则可能会出现不能正常解析的错误。

**3.4 Lucene 查询详解**

在 Lucene 中，处理用户输入的查询关键词其实就是构建 Query 对象的过程。Lucene 搜索文档需要先读入索引文件，实例化一个 IndexReader 对象，然后实例化出 IndexSearch 对象，IndexSearch 对象的 search() 方法完成搜索过程，Query 对象作为 search() 方法的对象。搜索结果保存在一个 TopDocs 类型的文档集合中，遍历 TopDocs 集合输出文档信息。

**QueryParser 可以搜索单个字段，而 MultiFieldQueryParser 则可以查询多个字段，并且多个字段之间是或的关系，所以在开发中，MultiFieldQueryParser 使用的较多。**

[复制代码](javascript:void(0);)

package tup.lucene.queries;

import java.io.IOException;

import java.nio.file.Path;

import java.nio.file.Paths;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.document.Document;

import org.apache.lucene.index.DirectoryReader;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.queryparser.classic.MultiFieldQueryParser;

import org.apache.lucene.queryparser.classic.ParseException;

import org.apache.lucene.queryparser.classic.QueryParser;

import org.apache.lucene.queryparser.classic.QueryParser.Operator;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreDoc;

import org.apache.lucene.search.TopDocs;

import org.apache.lucene.store.Directory;

import org.apache.lucene.store.FSDirectory;

import tup.lucene.ik.IKAnalyzer6x;

/\*\*

\* 单域搜索

\* @author moonxy

\*

\*/

public class QueryParseTest {

public static void main(String[] args) throws ParseException, IOException {

// 搜索单个字段

String field = "title";

// 搜索多个字段时使用数组

//String[] fields = { "title", "content" };

Path indexPath = Paths.get("indexdir");

Directory dir = FSDirectory.open(indexPath);

IndexReader reader = DirectoryReader.open(dir);

IndexSearcher searcher = new IndexSearcher(reader);

Analyzer analyzer = new IKAnalyzer6x(false);//最细粒度分词

QueryParser parser = new QueryParser(field, analyzer);

// 多域搜索

//MultiFieldQueryParser multiParser = new MultiFieldQueryParser(fields, analyzer);

// 关键字同时成立使用 AND, 默认是 OR

parser.setDefaultOperator(Operator.AND);

// 查询语句

Query query = parser.parse("农村学生");//查询关键词

System.out.println("Query:" + query.toString());

// 返回前10条

TopDocs tds = searcher.search(query, 10);

for (ScoreDoc sd : tds.scoreDocs) {

// Explanation explanation = searcher.explain(query, sd.doc);

// System.out.println("explain:" + explanation + "\n");

Document doc = searcher.doc(sd.doc);

System.out.println("DocID:" + sd.doc);

System.out.println("id:" + doc.get("id"));

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

System.out.println("content:" + doc.get("content"));

System.out.println("文档评分:" + sd.score);

}

dir.close();

reader.close();

}

}

[复制代码](javascript:void(0);)

控制台输出如下：

[复制代码](javascript:void(0);)

加载扩展词典：dict/ext.dic

加载扩展停止词典：dict/stopword.dic

加载扩展停止词典：dict/ext\_stopword.dic

Query:+title:农村 +title:村学 +title:学生

**DocID:1**

id:2

title:北大迎4380名新生 农村学生700多人近年最多

content:昨天，北京大学迎来4380名来自全国各地及数十个国家的本科新生。其中，农村学生共700余名，为近年最多...

文档评分:2.320528

[复制代码](javascript:void(0);)

注意，在结果中打印了 DocID 和 id，前者是文档的 ID，是 Lucene 为索引的每个文档标记，后者是文档自定义的 id 字段。

**3.5 Lucene 查询高亮**

高亮功能一直都是全文检索的一项非常优秀的模块，在一个标准的搜索引擎中，高亮的返回命中结果，几乎是必不可少的一项需求，因为通过高亮，可以在搜索界面上快速标记出用户的搜索关键字，从而减少了用户自己寻找想要的结果的时间，在一定程度上大大提高了用户的体验性和友好度。

Highlight 包含3个主要部分：

1）段划分器：Fragmenter

2）计分器：Score

3）格式化器：Formatter

[复制代码](javascript:void(0);)

package tup.lucene.highlfighter;

import java.io.IOException;

import java.nio.file.Path;

import java.nio.file.Paths;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.document.Document;

import org.apache.lucene.index.DirectoryReader;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.queryparser.classic.ParseException;

import org.apache.lucene.queryparser.classic.QueryParser;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreDoc;

import org.apache.lucene.search.TopDocs;

import org.apache.lucene.search.highlight.Fragmenter;

import org.apache.lucene.search.highlight.Highlighter;

import org.apache.lucene.search.highlight.InvalidTokenOffsetsException;

import org.apache.lucene.search.highlight.QueryScorer;

import org.apache.lucene.search.highlight.SimpleHTMLFormatter;

import org.apache.lucene.search.highlight.SimpleSpanFragmenter;

import org.apache.lucene.search.highlight.TokenSources;

import org.apache.lucene.store.Directory;

import org.apache.lucene.store.FSDirectory;

import tup.lucene.ik.IKAnalyzer6x;

/\*\*

\* Lucene查询高亮

\* @author moonxy

\*

\*/

public class HighlighterTest {

public static void main(String[] args) throws IOException, InvalidTokenOffsetsException, ParseException {

String field = "title";

Path indexPath = Paths.get("indexdir");

Directory dir = FSDirectory.open(indexPath);

IndexReader reader = DirectoryReader.open(dir);

IndexSearcher searcher = new IndexSearcher(reader);

Analyzer analyzer = new IKAnalyzer6x();

QueryParser parser = new QueryParser(field, analyzer);

Query query = parser.parse("北大学生");

System.out.println("Query:" + query);

// 查询高亮

QueryScorer score = new QueryScorer(query, field);

SimpleHTMLFormatter fors = new SimpleHTMLFormatter("<span style=\"color:red;\">", "</span>");// 定制高亮标签

Highlighter highlighter = new Highlighter(fors, score);// 高亮分析器

// 返回前10条

TopDocs tds = searcher.search(query, 10);

for (ScoreDoc sd : tds.scoreDocs) {

// Explanation explanation = searcher.explain(query, sd.doc);

// System.out.println("explain:" + explanation + "\n");

Document doc = searcher.doc(sd.doc);

System.out.println("id:" + doc.get("id"));

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

Fragmenter fragment = new SimpleSpanFragmenter(score);

highlighter.setTextFragmenter(fragment);

// TokenStream tokenStream = TokenSources.getAnyTokenStream(searcher.getIndexReader(), sd.doc, field, analyzer);// 获取tokenstream

// String str = highlighter.getBestFragment(tokenStream, doc.get(field));// 获取高亮的片段

String str = highlighter.getBestFragment(analyzer, field, doc.get(field));// 获取高亮的片段

System.out.println("高亮的片段:" + str);

}

dir.close();

reader.close();

}

}

[复制代码](javascript:void(0);)

控制台输出如下：

[复制代码](javascript:void(0);)

加载扩展词典：dict/ext.dic

加载扩展停止词典：dict/stopword.dic

加载扩展停止词典：dict/ext\_stopword.dic

Query:title:北大学生 title:北大 title:大学生 title:大学 title:学生

id:2

title:北大迎4380名新生 农村学生700多人近年最多

高亮的片段:<span style="color:red;">北大</span>迎4380名新生 农村<span style="color:red;">学生</span>700多人近年最多

[复制代码](javascript:void(0);)

**四、Tika 文件内容提取**

Apache Tika 是一个用于文件类型检测和文件内容提取的库，是 Apache 软件基金会的项目。Tika 可以检测操作 1000 种不同类型的文档，比如 PPT、PDF、DOC、XLS 等，所有的文本类型都可以通过一个简单的接口被解析，Tika 广泛应用于搜索引擎、内容分析、文本翻译等领域。

Tika 下载的官网地址：<http://tika.apache.org/download.html>，其历史版本下载地址为：<http://archive.apache.org/dist/tika/>，此处现在当前最新版 tika-app-1.18.jar

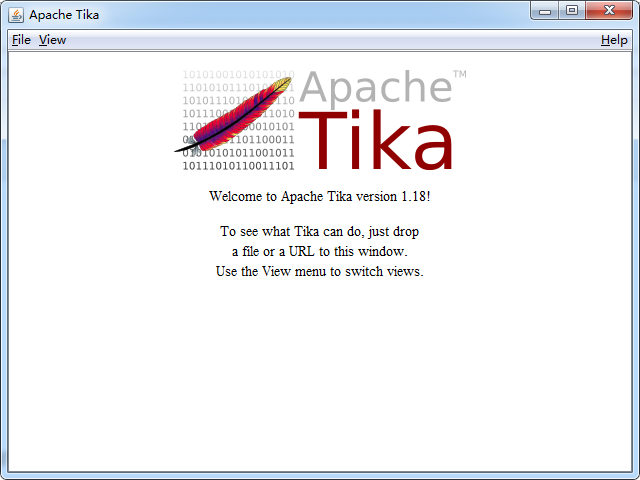


下载下来之后为一个 jar 包，但 Tika 可作为 GUI 工具使用，在 CMD 中先进入下载目录，然后使用如下命令启动 Tika GUI：

java -jar tika-app-1.18.jar -g

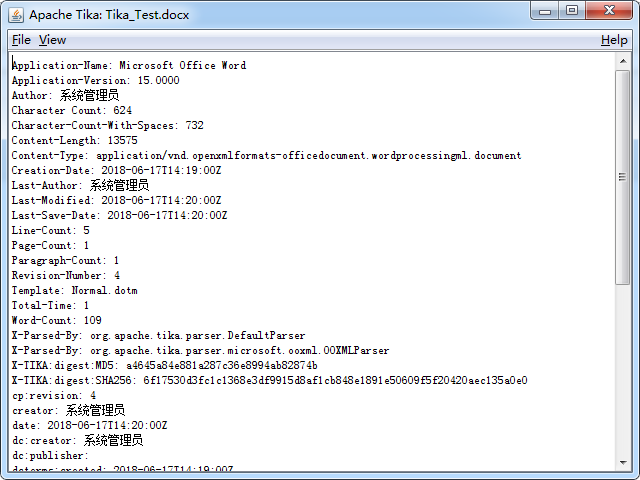
java -jar 表示启动 jar 包，后面跟 jar 包的名字，**-g（--gui） 参数表示以 GUI 的方式启动 Tika（Start the Apache Tika GUI）**，当然前提是已经配置好了 java 环境变量。

界面如下：

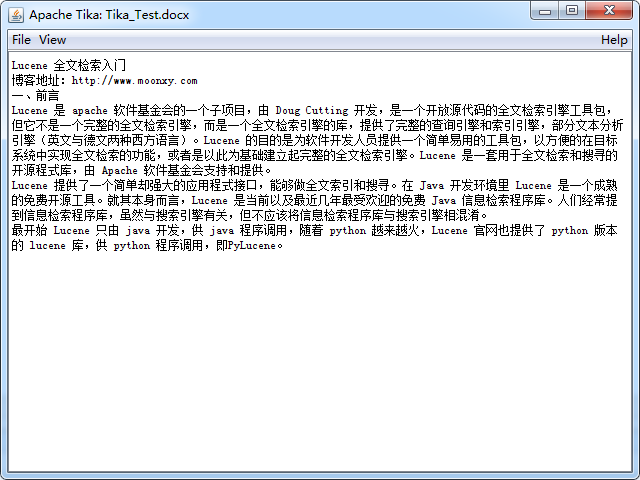


可以点击 File 菜单中的 Open...来打开一个本地文件或者输入 URL 来打开远程文件，也可直接将本地文件拖入到 Tika，Tika 将会自动识别文件类型并显示文件信息。

默认显示文件的元数据信息，如下：



点击 View 菜单，里面可以选择查看文件的具体内容，如选择其中的 Plain Text，显示如下：



将该 jar 包放入项目 lib 中，调用相应的接口就可以提取不同文件的内容，主要分为如下提取方法。

**方法一：使用 Tika 对象提取文档内容**

[复制代码](javascript:void(0);)

package tup.tika.demo;

import java.io.File;

import java.io.IOException;

import org.apache.tika.Tika;

import org.apache.tika.exception.TikaException;

/\*\*

\* 使用Tika对象自动解析文档，提取文档内容

\* @author moonxy

\*

\*/

public class TikaExtraction {

public static void main(String[] args) throws IOException, TikaException {

Tika tika = new Tika();

// 新建存放各种文件的files文件夹

File fileDir = new File("files");

// 如果文件夹路径错误，退出程序

if (!fileDir.exists()) {

System.out.println("文件夹不存在, 请检查!");

System.exit(0);

}

// 获取文件夹下的所有文件，存放在File数组中

File[] fileArr = fileDir.listFiles();

String filecontent;

for (File f : fileArr) {

// 获取文件名

System.out.println("File Name: " + f.getName());

filecontent = tika.parseToString(f);// 自动解析

// 获取文件内容

System.out.println("Extracted Content: " + filecontent);

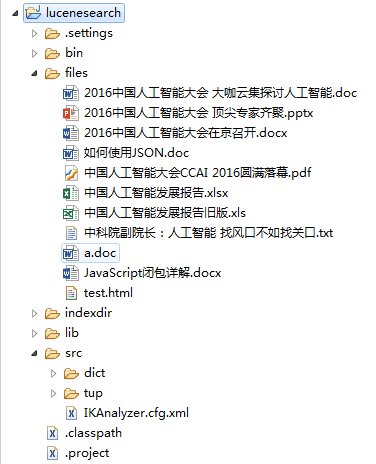
}

}

}

[复制代码](javascript:void(0);)

在工程中新建 Files 目录，放入需要解析的文件：



上述代码中首先新建一个 File 对象指向存放各种文档的文件夹 Files，通过 File 对象的 exists() 方法判断目录路径是否存在，如果路径错误则退出程序，打印提示信息。接下来，通过 listFiles() 方法获取 files 目录下所有的文件，存放在文件数组中。最后新建一个 Tika 对象，调用 parseToString() 方法获取文档内容，该方法的传入参数为 File 对象。

**方法二：使用 Parser 接口提取文档内容**

[复制代码](javascript:void(0);)

package tup.tika.demo;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import org.apache.tika.exception.TikaException;

import org.apache.tika.metadata.Metadata;

import org.apache.tika.parser.AutoDetectParser;

import org.apache.tika.parser.ParseContext;

import org.apache.tika.parser.Parser;

import org.apache.tika.sax.BodyContentHandler;

import org.xml.sax.SAXException;

/\*\*

\* 使用Parser接口自动解析文档，提取文档内容

\* @author moonxy

\*

\*/

public class ParserExtraction {

public static void main(String[] args) throws IOException, SAXException, TikaException {

// 新建存放各种文件的files文件夹

File fileDir = new File("files");

// 如果文件夹路径错误，退出程序

if (!fileDir.exists()) {

System.out.println("文件夹不存在, 请检查!");

System.exit(0);

}

// 获取文件夹下的所有文件，存放在File数组中

File[] fileArr = fileDir.listFiles();

// 创建内容处理器对象

BodyContentHandler handler = new BodyContentHandler();

// 创建元数据对象

Metadata metadata = new Metadata();

FileInputStream inputStream = null;

Parser parser = new AutoDetectParser();

// 自动检测分析器

ParseContext context = new ParseContext();

for (File f : fileArr) {

// 获取文件名

System.out.println("File Name: " + f.getName());

inputStream = new FileInputStream(f);

parser.parse(inputStream, handler, metadata, context);

// 获取文件内容

System.out.println(f.getName() + ":\n" + handler.toString());

}

}

}

[复制代码](javascript:void(0);)

使用 Parse 接口自动提取内容和单一的提取一种文档的区别在于实例化对象不一样，AutoDetectParser 是 CompositeParser 的子类，它能够自动检测文件类型，并使用相应的方法把接收到的文档自动发送给最接近的解析器类。

可以参考官方文档，将文档解析为不同的格式，如上面都是解析为纯文本格式（Plain text），也可解析为 html 格式（Structured text）等，如：

<http://tika.apache.org/1.18/examples.html#Parsing_to_XHTML>

