4. Lucene

- PyLucene安装
- 创建索引
- 搜索索引
- 代码解析
- 中文分词

Lucene 安装 https://lucene.apache.org/pylucene/install.html

- **课程Docker镜像中已安装pylucene 8.3.0**

- 以下内容适用于手动安装—适合在本地环境或虚拟机中使用

安装java

- sudo apt-get install openjdk-8-jre-headless
- 运行 javac –version 验证是否安装成功

安装ant

- sudo apt-get install ant
- 运行 which ant 查看ant路径

下载pylucene并解压

- 下载pylucene-8.3.0-src.tar.gz，官方源下载较慢，建议使用清华源 https://mirrors.tuna.tsinghua.edu.cn/apache/lucene/pylucene/
- 使用tar –zxvf pylucene-8.3.0-src.tar.gz解压

Lucene 安装 https://lucene.apache.org/pylucene/install.html

安装jcc

- 进入pylucene-8.3.0/jcc路径
- vim setup.py 修改JDK路径（不同操作系统和java版本修改的路径不同）
- 依次运行
- python3 setup.py build（如果提示缺少setuptools，先运行pip3 install setuptools）
- python3 setup.py install（如果提示权限问题在前面加sudo）
- 运行python3 -m jcc 测试是否安装成功

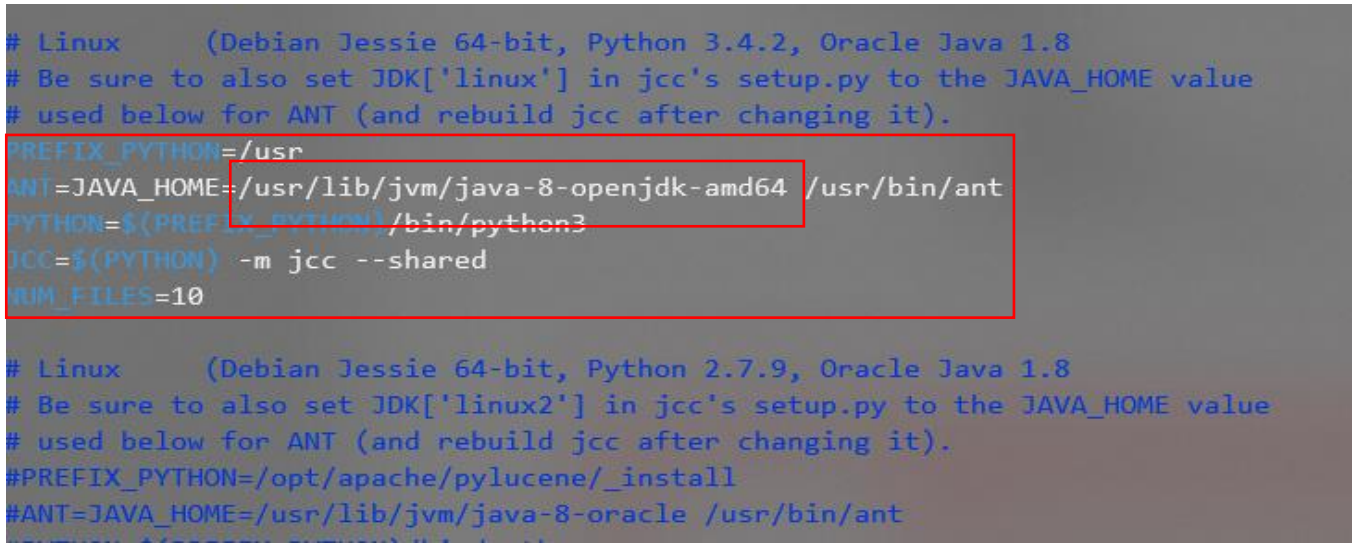
```
            from helpers2.darwin import JAVAHOME, JAVAFRAMEWORKS
        else:
            from helpers3.darwin import JAVAHOME, JAVAFRAMEWORKS
    except ImportError:
        JAVAHOME = None
        JAVAFRAMEWORKS = None
else:
    JAVAHOME = None
    JAVAFRAMEWORKS = None

JDK = {
    'darwin': JAVAHOME or JAVAFRAMEWORKS,
    'ipod': '/usr/include/gcc',
#    'linux': '/usr/lib/jvm/java-8-oracle',
    'linux': '/usr/lib/jvm/java-8-openjdk-amd64',
    'sunos5': '/usr/jdk/instances/jdk1.6.0',
    'win32': JAVAHOME,
    'mingw32': JAVAHOME,
    'freebsd7': '/usr/local/diablo-jdk1.6.0'
}
if 'JCC_JDK' in os.environ:
    JDK[platform] = os.environ['JCC_JDK']


if not JDK[platform]:
    raise RuntimeError('''

Can't determine where the Java JDK has been installed on this machine.

"setup.py" 451L, 17080C
```

Lucene 安装 https://lucene.apache.org/pylucene/install.html

安装PyLucene

- 进入pylucene-8.3.0，vim Makefile编辑Makefile
- 找到自己的系统版本和python版本对应的位置，解除注释，并将java路径改为自己的java路径，保存退出
- 依次运行

```
make

make test

sudo make install
```



- 验证安装（进入python ）

```
import lucene

import jcc

lucene.initVM()

lucene.VERSION
```

Lucene 简介

- Lucene是一个高效的，基于Java的全文检索库。

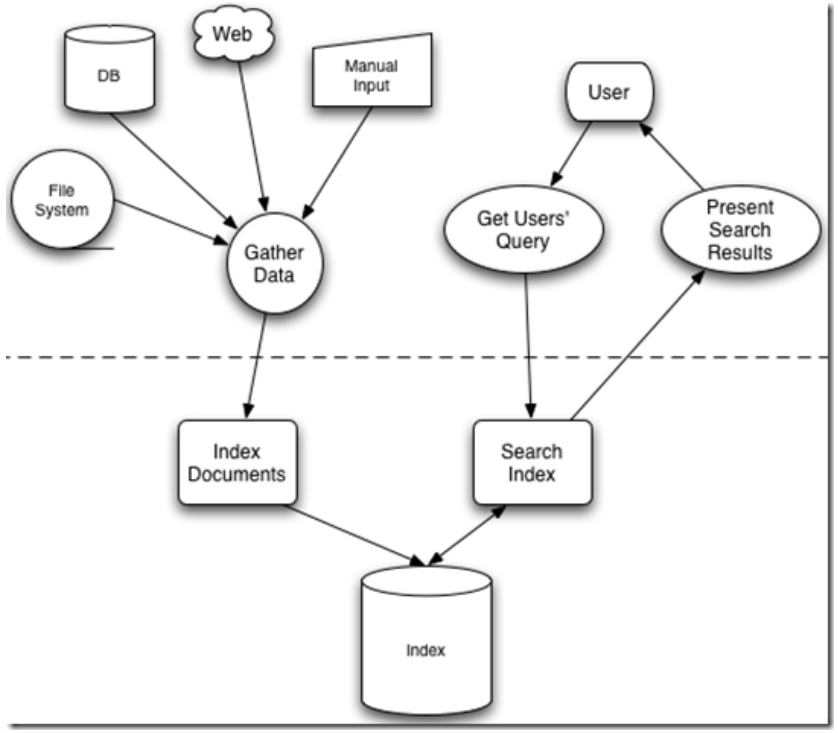
  先建立索引，再对索引进行搜索的过程叫全文检索(Full-text Search)。

- 全文检索大体分两个过程，索引创建(Indexing)和搜索索引(Search)。

  索引创建：将现实世界中所有的结构化和非结构化数据提取信息，创建索引的过程。

  搜索索引：就是得到用户的查询请求，搜索创建的索引，然后返回结果的过程。

- Lucene中全文检索的一般过程(《Lucene in Action》)



创建索引

- 创建索引一般有如下几步：

1.一些要索引的原文档(Document)

以两个文件为例

文件一：Students should be allowed to go out with their friends, but not allowed to drink beer.

文件二：My friend Jerry went to school to see his students but found them drunk which is not allowed.

2.将原文档传入分词组件(Tokenizer)

a)将文档分成一个一个单独的单词。

b)去除标点符号。

c)去除停词(Stop word)。

在例子中，得到以下词元(Token)：“Students”，“allowed”，“go”，“their”，“friends”，“allowed”，“drink”，“beer”，“My”，“friend”，“Jerry”，“went”，“school”，“see”，“his”，“students”，“found”，“them”，“drunk”，“allowed”。

3.将得到的词元(Token)传给语言处理组件(linguistic processor)

a)变为小写

b)Stemming：如“cars”到“car”等

c)Lemmatization：如“drove”到“drive”等

在例子中，得到如下词(Term)：“student”，“allow”，“go”，“their”，“friend”，“allow”，“drink”，“beer”，“my”，“friend”，“jerry”，“go”，“school”，“see”，“his”，“student”，“find”，“them”，“drink”，“allow”。

4.将得到的词(Term)传给索引组件(Indexer)

a)利用得到的词(Term)创建一个字典。

b)对字典按字母顺序进行排序。

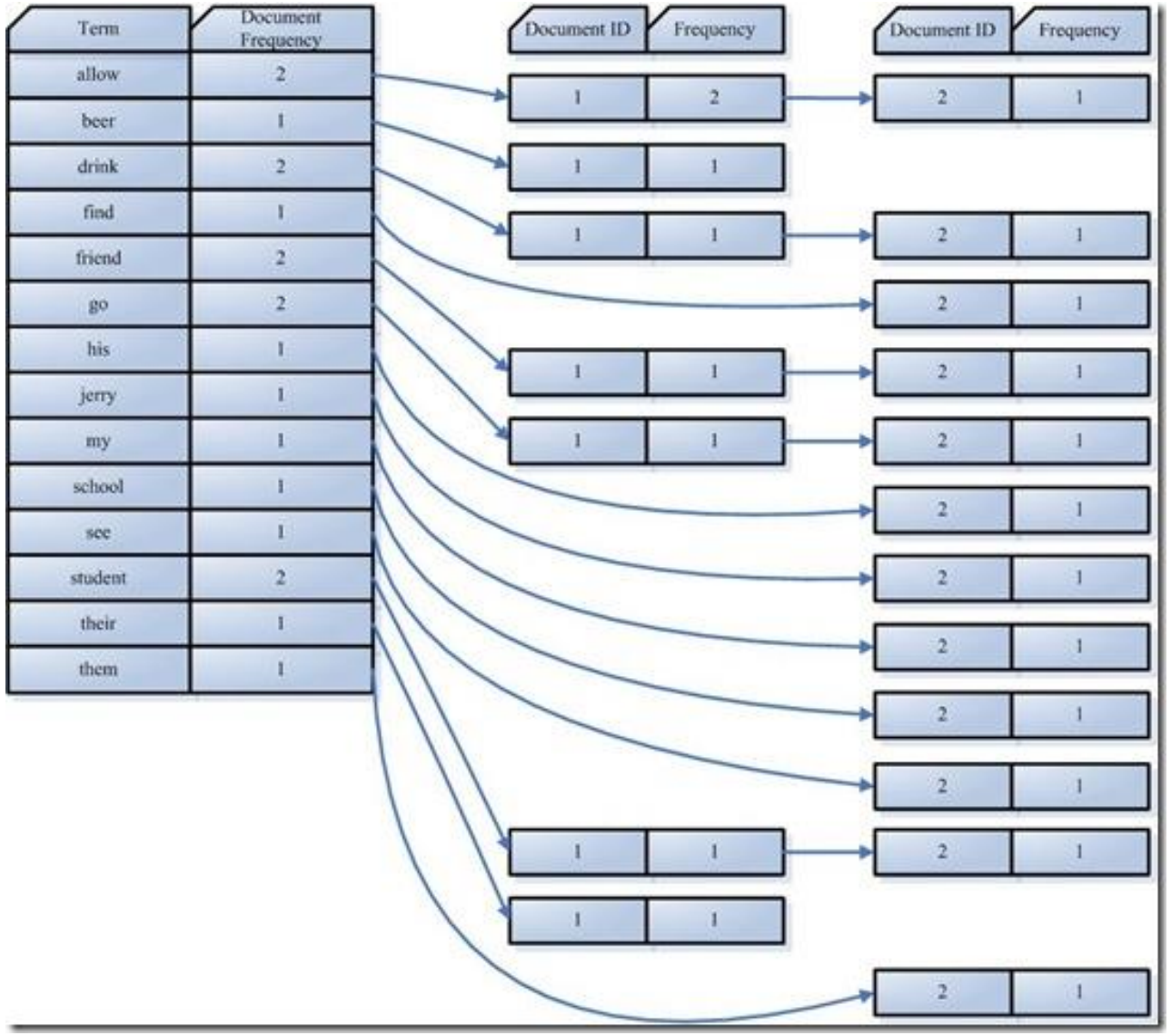
c)合并相同的词(Term)成为文档倒排(Posting List)链表。

创建索引

- **最后得到如图的倒排索引**

➢ Document Frequency 即文档频次，表示总共有多少文件包含此词(Term)。

➢ Frequency 即词频率，表示此文件中包含了几个此词(Term)。



搜索索引

1.用户输入查询语句

例如lucene AND learned NOT hadoop，说明用户想找一个包含lucene和learned然而不包括hadoop的文档。

2.对查询语句进行词法分析，语法分析，及语言处理。

a)词法分析

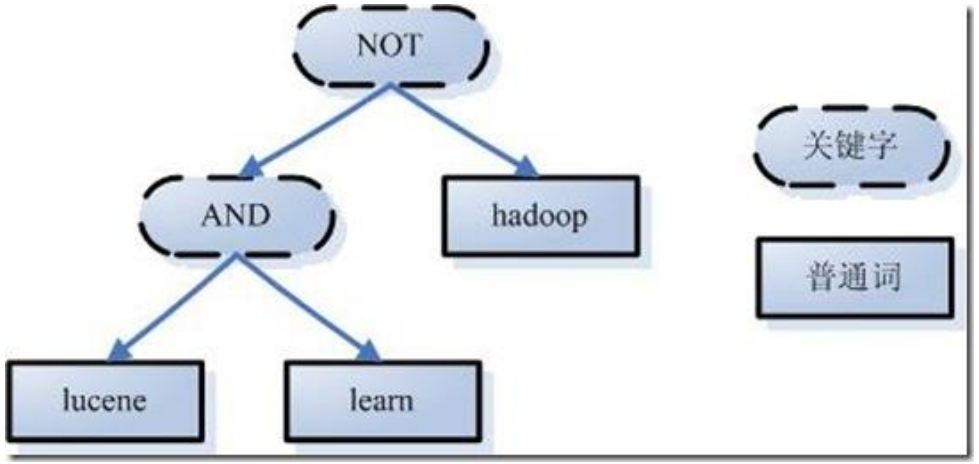
识别单词和关键字。例子中，单词有lucene，learned，hadoop，关键字有AND，NOT。

b)语法分析

根据查询语句的语法规则来形成一棵语法树。

c)语言处理

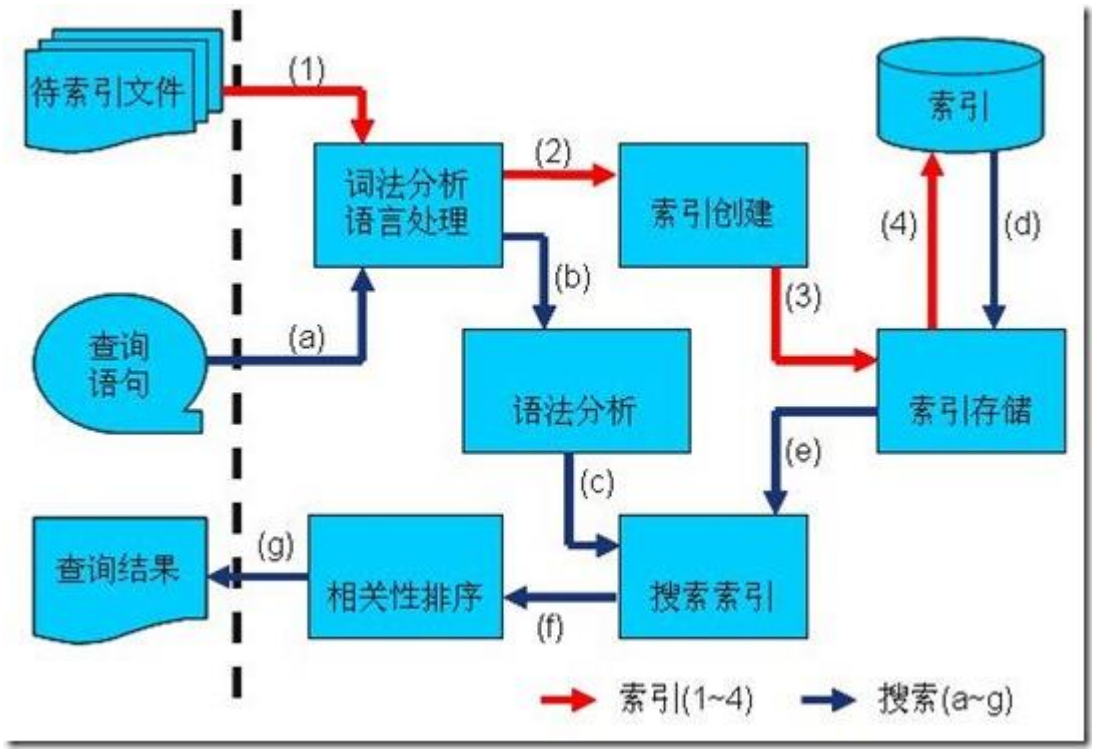
同索引中的语言处理。例子中将learned变为learn。



搜索索引

3.搜索索引，得到符合语法树的文档
4.根据得到的文档和查询语句的相关性，对结果进行排序

- 上述索引创建和搜索过程如图



Lucene 文档

- 文档主页

  https://lucene.apache.org/core/8_3_0/index.html

- 核心相关API

  https://lucene.apache.org/core/8_3_0/core/overview-summary.html

- analyzer相关

  https://lucene.apache.org/core/8_3_0/core/org/apache/lucene/analysis/package-summary.html#package.description

代码解析

- 运行samples4中的IndexFiles.py对samples4/testfolder下的txt文档创建索引。

```
root@43612dc2e594:/workspaces/hello_world/lab4_pylucene/samples4# python IndexFiles.py
lucene 8.3.0
adding pg1342.txt
adding pg16328.txt
adding pg1661.txt
adding pg19445.txt
adding pg2591.txt
adding pg27827.txt
adding pg30601.txt
adding pg5200.txt
adding pg76.txt
commit index
..done
0:00:02.530676
```

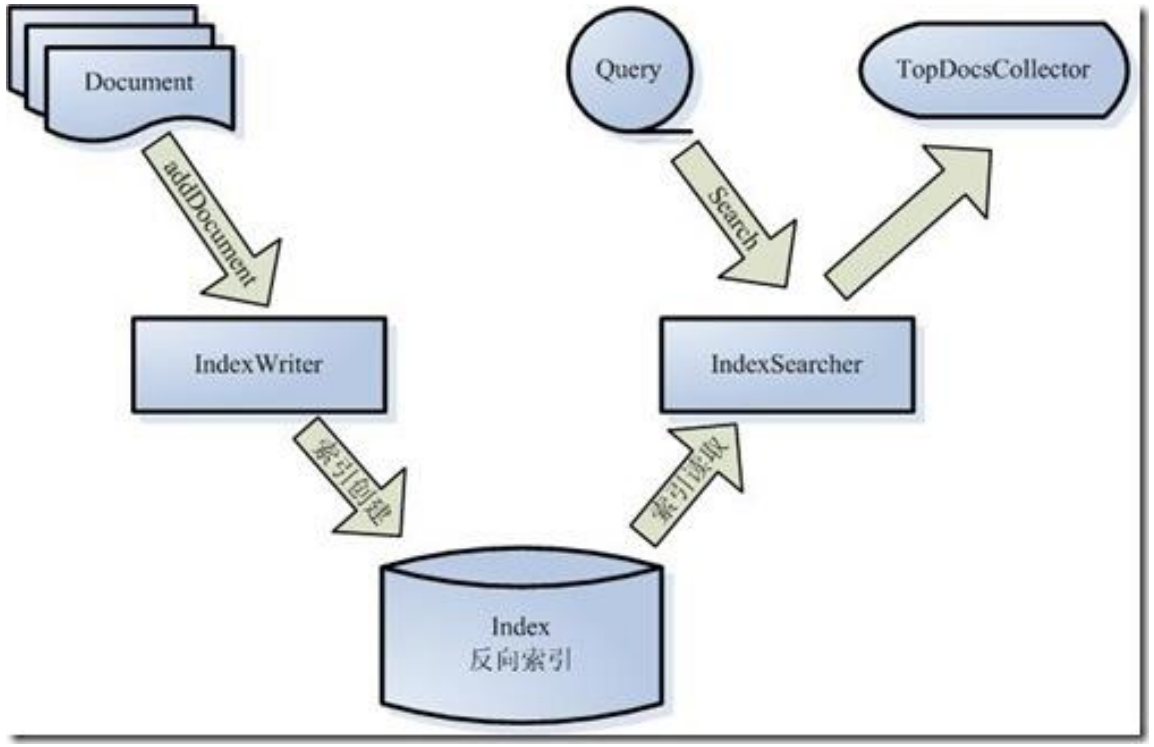
- 运行SearchFiles.py搜索索引。

```
root@43612dc2e594:/workspaces/hello_world/lab4_pylucene/samples4# python SearchFiles.py
lucene 8.3.0

Hit enter with no input to quit.

Searching for: EBook
9 total matching documents.
path: testfolder/pg19445.txt name: pg19445.txt score: 0.04896830767393112
path: testfolder/pg5200.txt name: pg5200.txt score: 0.04886522889137268
path: testfolder/pg16328.txt name: pg16328.txt score: 0.047642502933740616
path: testfolder/pg27827.txt name: pg27827.txt score: 0.04646031931042671
path: testfolder/pg30601.txt name: pg30601.txt score: 0.04646031931042671
path: testfolder/pg2591.txt name: pg2591.txt score: 0.044664204120635986
path: testfolder/pg76.txt name: pg76.txt score: 0.04449198395013809
path: testfolder/pg1661.txt name: pg1661.txt score: 0.04428377002477646
path: testfolder/pg1342.txt name: pg1342.txt score: 0.0441485196352005
```

代码解析

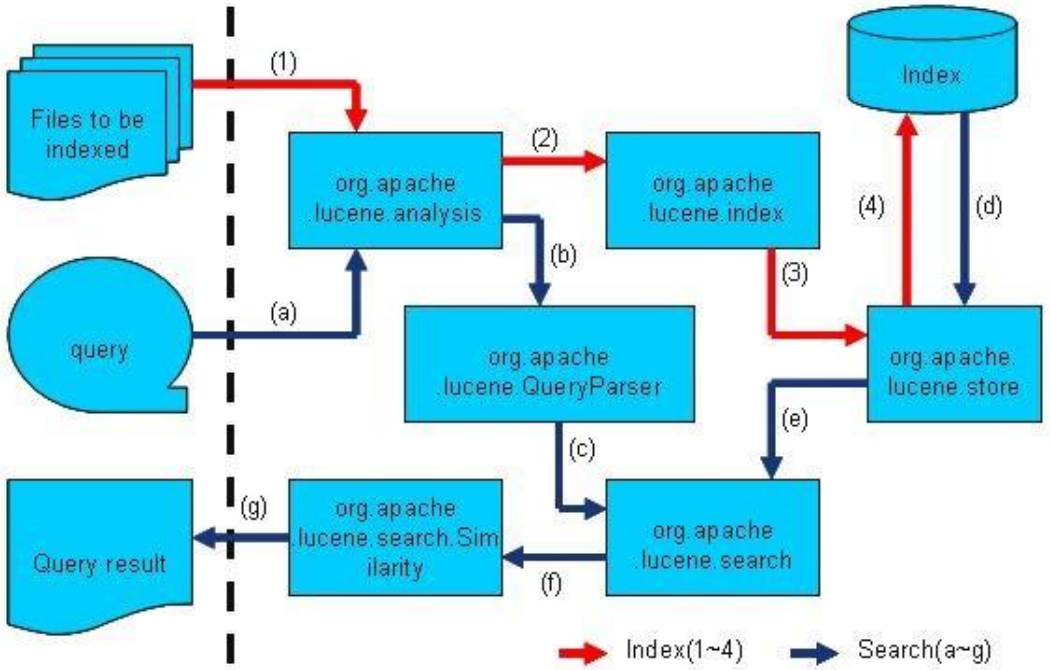
- **Lucene各组件如图**

➢ 被索引的文档用Document对象表示。
➢ IndexWriter通过函数addDocument将文档添加到索引中，实现创建索引的过程。
➢ 当用户有请求时，Query代表用户的查询语句。
➢ IndexSearcher通过函数search搜索Lucene Index。
➢ IndexSearcher计算term weight和score并且将结果返回给用户。



代码解析

- **全文检索的流程对应的Lucene实现的包结构如图**



- Lucene的analysis模块主要负责词法分析及语言处理而形成Term。
- Lucene的index模块主要负责索引的创建，里面有IndexWriter。
- Lucene的store模块主要负责索引的读写。
- Lucene的QueryParser主要负责语法分析。
- Lucene的search模块主要负责对索引的搜索。
- Lucene的similarity模块主要负责对相关性打分的实现。

代码解析

- **索引过程如下：**
  - 创建一个IndexWriter用来写索引文件，它有几个参数，store是索引文件所存放的位置，analyzer用来对文档进行词法分析和语言处理的。
  - 创建一个Document代表我们要索引的文档。
  - 将不同的Field加入到文档中。一篇文档有多种信息，如题目，作者，修改时间，内容等。不同类型的信息用不同的Field来表示，
  - IndexWriter调用函数addDocument将索引写到索引文件夹中。

- **搜索过程如下：**
  - 创建IndexSearcher准备进行搜索。
  - 创建analyer用来对查询语句进行词法分析和语言处理。
  - 创建QueryParser用来对查询语句进行语法分析。
  - QueryParser调用parser进行语法分析，形成查询语法树，放到Query中。
  - IndexSearcher调用search对查询语法树Query进行搜索，得到结果

代码解析

● Field

- 每个文档可包含多个Field，每个Field表示一种类型的相关信息
- 建立索引时，要对各个文档添加Field；检索时则要针对某个Field进行搜索
- 每个Field创建时都可利用FieldType类型来订制、优化Field的相关属性，例如进行倒排索引的方式，是否完全储存内容，是否进行分词，索引选项等等

- 例子中，content的FieldType为t2，
- t2.setStored(False)表示不需要完全储存内容
- t2.setTokenized(True)表示需要分词
- t2.setIndexOptions(FieldInfo.IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)表示需要建立词语到文档的（倒排）索引，还要记录词语在文档中出现的词频和位置

- 此外，还可以使用一些Field的派生类，如StringField，TextField，IntPoint, DoublePoint等，其中已预设了一些FieldType，方便使用

代码解析

● IndexFiles.py 关键代码解析

```
lucene.initVM()      #初始化Java虚拟机
…
store = lucene.SimpleFSDirectory(Path.get(storeDir))      #索引文件存放的位置
analyzer = lucene.StandardAnalyzer()
analyzer = LimitTokenCountAnalyzer(analyzer, 1048576)
#analyzer是用来对文档进行词法分析和语言处理的
config = IndexWriterConfig(analyzer)
config.setOpenMode(IndexWriterConfig.OpenMode.CREATE)
writer = IndexWriter(store, config)
#创建一个IndexWriter用来写索引文件


…

for root, dirnames, filenames in os.walk(root):
#遍历testfolder下的文件
…
file = open(path, encoding='gbk’)
contents = file.read()
#打开gbk编码的文件。
#文件内容存放在contents中
```

代码解析

● IndexFiles.py 关键代码解析

```
#文档的文件名及路径的FieldType
t1 = FieldType()
…
#文档内容相关的FieldType
t2 = FieldType()
…
doc = Document()
#创建一个Document代表我们要索引的文档
doc.add(Field("name", filename, t1))
doc.add(Field("path", path, t1))
doc.add(Field("contents", contents, t2))
#将不同的Field加入到文档中。一篇文档有多种信息，如题目，作者，修改时间，内容等。不同类型的信息用不同的Field来表示，在本例子中，一共有三类信息进行了索引，一个是文件路径，一个是文件名，一个是文件内容。
writer.addDocument(doc)
#IndexWriter调用函数addDocument将索引写到索引文件夹中
```

代码解析

● SearchFiles.py 关键代码解析

```
initVM()  #初始化Java虚拟机
directory = SimpleFSDirectory(File(STORE_DIR).toPath())    #索引文件存放的位置
searcher = IndexSearcher(DirectoryReader.open(directory))
#索引信息读入到内存，创建IndexSearcher准备进行搜索
analyzer = StandardAnalyzer()
#analyzer用来对查询语句进行词法分析和语言处理的，和IndexFiles.py中使用同样的analyzer。
...
query = QueryParser("contents", analyzer).parse(command)
#用analyzer来对查询语句进行词法分析和语言处理。
#QueryParser调用parser进行语法分析，形成查询语法树，放到Query中。
scoreDocs = searcher.search(query, 50).scoreDocs
#IndexSearcher调用search对查询语法树Query进行搜索，得到结果
```

Analyzer

- **Analyzer的介绍详见**

  https://lucene.apache.org/core/8_3_0/core/org/apache/lucene/analysis/package-summary.html

- **常用的Analyzer** https://lucene.apache.org/core/8_3_0/analyzers-common/index.html

1. **org.apache.lucene.analysis.core.StopAnalyzer**
   StopAnalyzer能过滤词汇中的特定字符串和词汇，并且完成大写转小写的功能。
2. **org.apache.lucene.analysis.standard.StandardAnalyzer**
   StandardAnalyzer 根据空格和符号来完成分词，还可以完成数字、字母、E-mail地址、IP地址以及中文字符的分析处理。对中文分词时，他将每个汉字作为一个词。
3. **org.apache.lucene.analysis.core.SimpleAnalyzer**
   SimpleAnalyzer具备基本西文字符词汇分析的分词器，处理词汇单元时，以非字母字符作为分割符号。分词器不能做词汇的过滤。输出的词汇单元完成小写字符转换，去掉标点符号等分割符。
4. **org.apache.lucene.analysis.core.WhitespaceAnalyzer**
   WhitespaceAnalyzer使用空格作为间隔符的词汇分割分词器。处理词汇单元的时候，以空格字符作为分割符号。分词器不做词汇过滤，也不进行小写字符转换。
5. **org.apache.lucene.analysis.cjk.CJKAnalyzer**
   CJKAnalyzer根据汉语中词条长度为2居多的特点，将每相邻2个字作为词汇单元。

中文分词

- 由于汉字词条长度主要在2~4之间，StandardAnalyzer的词汇单元与汉语中词相差甚远。CJKAnalyzer虽然在某种程度更符合汉语的习惯，但是这样分词使得每个汉字都在两个词语中，使得词语的效率只有50%左右。

- 让Lucene支持中文分词，有两种做法：一种是实现自己的Analyzer，一般需要实现自己的Analyzer，Filter，Tokenizer类；一种是用现有的分词库，将文本先以空格方式分好词后，再给WhitespaceAnalyzer或SimpleAnalyzer这些英文分词器处理（他们以空格做为分割分词）

- 除CJKAnalyzer外，Lucene还提供了SmartChineseAnalyzer，pylucene中未安装，感兴趣的同学可以尝试安装并使用它

中文分词

- Docker中提供了几个常用的中文分词
  - Jieba（https://github.com/fxsjy/jieba）

    支持GBK,UTF8,Unicode编码
  - Pynlpir（https://github.com/tsroten/pynlpir）

    支持GBK,UTF8,Unicode编码
  - THULAC（https://github.com/thunlp/THULAC-Python）

    支持GBK

网页预处理

- 网页源代码中包含HTML tag（例如<html>,<body>等），在加入lucene前，可以用BeautifulSoup等库过滤文档中的 HTML tag。

➢ 方法一：BeautifulSoup过滤tag

''.join(soup.findAll(text=True))

```
>>> content
'<html>\n <head>\n  <title>\n   Page title\n  </title>\n </head>\n <body>\n  <p
id="firstpara" align="center">\n   This is paragraph\n   <b>\n    one\n   </b>\n
   .\n  </p>\n  <p id="secondpara" align="blah">\n   This is paragraph\n   <b>\n
    two\n   </b>\n   .\n  </p>\n </body>\n</html>'
>>> soup = BeautifulSoup(content)
>>> ''.join(soup.findAll(text=True))
u'\n\n\n   Page title\n  \n\n\n\n   This is paragraph\n   \n    one\n   \n   .\n
  \n\n   This is paragraph\n   \n    two\n   \n   .\n  \n\n'
```

➢ 方法二：nltk库过滤tag（docker未安装，速度比BeautifulSoup快）

nltk.clean_html(content)

```
>>> import nltk
>>> nltk.clean_html(content)
'Page title\n \n \n \n \n This is paragraph\n \n one\n \n .\n \n \n This is para
graph\n \n two\n \n .'
```

练习

- **实现一个中文网页索引与搜索程序**

- 爬取一定数量（>5k）的中文网页（可利用之前实验爬取的网页），修改IndexFiles.py和SearchFiles.py，对这些中文网页建立索引并进行搜索，搜索时需要打印出检出文档的路径、网页标题、url。
- doc的Field中需要有name(文件名)，path(文件路径)，title(网页标题)，url(网页地址)，contents(索引的文件内容)
- 搜索时显示出相关信息

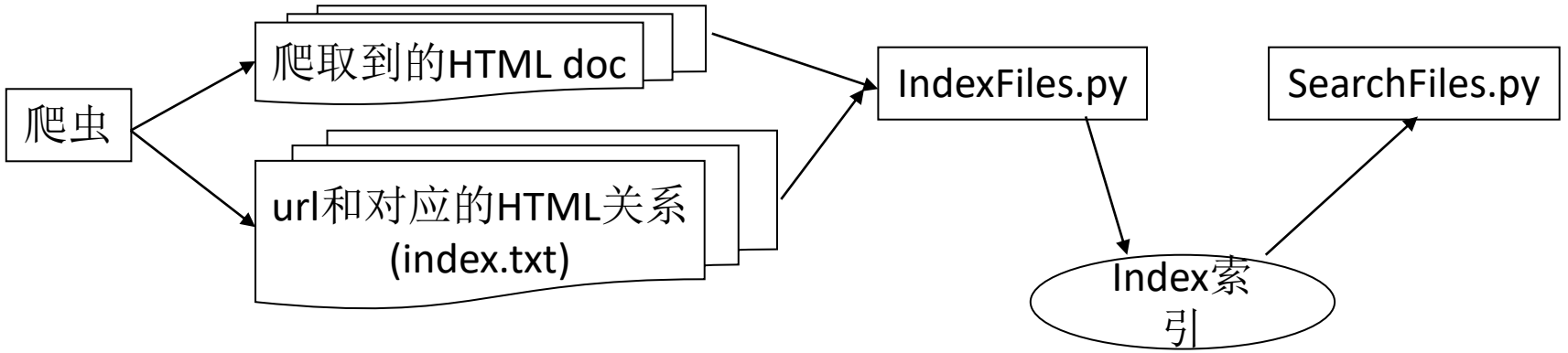
```
Hit enter with no input to quit.
Query:战争游戏

Searching for: 战争 游戏
10 total matching documents.
path: C:/Users/Alpha/Desktop/py/baidu_sim/html\httpwf.qq.com title: 战争前线-WarF
ace-官方网站-腾讯游戏-孤岛危机系列射击巨作 url: http://wf.qq.com/ name: httpwf.qq.com
path: C:/Users/Alpha/Desktop/py/baidu_sim/html\httpwww.pcgames.com.cnkzztpcgameG
OW title: 战争机器PC_战争机器_太平洋游戏网战争机器专题 url: http://www.pcgames.com.cn
/kzzt/pcgame/GOW/ name: httpwww.pcgames.com.cnkzztpcgameGOW
path: C:/Users/Alpha/Desktop/py/baidu_sim/html\httpwww.7k7k.comflash_fl491_1.htm
 title: 战争小游戏_战争小游戏大全_战争小游戏全集_7k7k战争小游戏 - 7k7k小游戏 url: http:
//www.7k7k.com/flash_fl/491_1.htm name: httpwww.7k7k.comflash_fl491_1.htm


Hit enter with no input to quit.
Query:战争 NOT 游戏

Searching for: 战争 NOT 游戏
10 total matching documents.
path: C:/Users/Alpha/Desktop/py/baidu_sim/html\httpbaike.baidu.comview14949.htm
title: 越南战争_百度百科 url: http://baike.baidu.com/view/14949.htm name: httpbaik
e.baidu.comview14949.htm
path: C:/Users/Alpha/Desktop/py/baidu_sim/html\httpbaike.baidu.comview67404.htm
title: 普法战争_百度百科 url: http://baike.baidu.com/view/67404.htm name: httpbaik
e.baidu.comview67404.htm
```

练习

- **实现一个中文网页索引与搜索程序**

提示：

- 在爬取网页时，可以把网页的网址和对应的文件名存放在文件中。这样IndexFiles.py可以从中提取出URL信息。例如实验二中，网址和文件名对应的文件为index.txt。



- 分词时注意网页的编码，网页的编码可以通过网页开头charset属性查看，可以通过decode和encode来转换GBK和UTF8编码。

```
<html>
  <head>
    <meta content="text/html; charset=gb2312" http-equiv="Content-Type">
```

- 可以使用任意分词库和Analyzer，但是在IndexFiles和SearchFiles中必须使用同一种分词库和Analyzer。

练习

- 提交作业时，需要提供代码和索引文件。

- 若使用初始docker环境中未安装的分词工具时，务必将该工具附上，并在README中说明代码测试方法。

参考资料

- https://lucene.apache.org/pylucene/
- https://www.php.cn/python-tutorials-372617.html