SEEM 5680
Introduction to Lucene

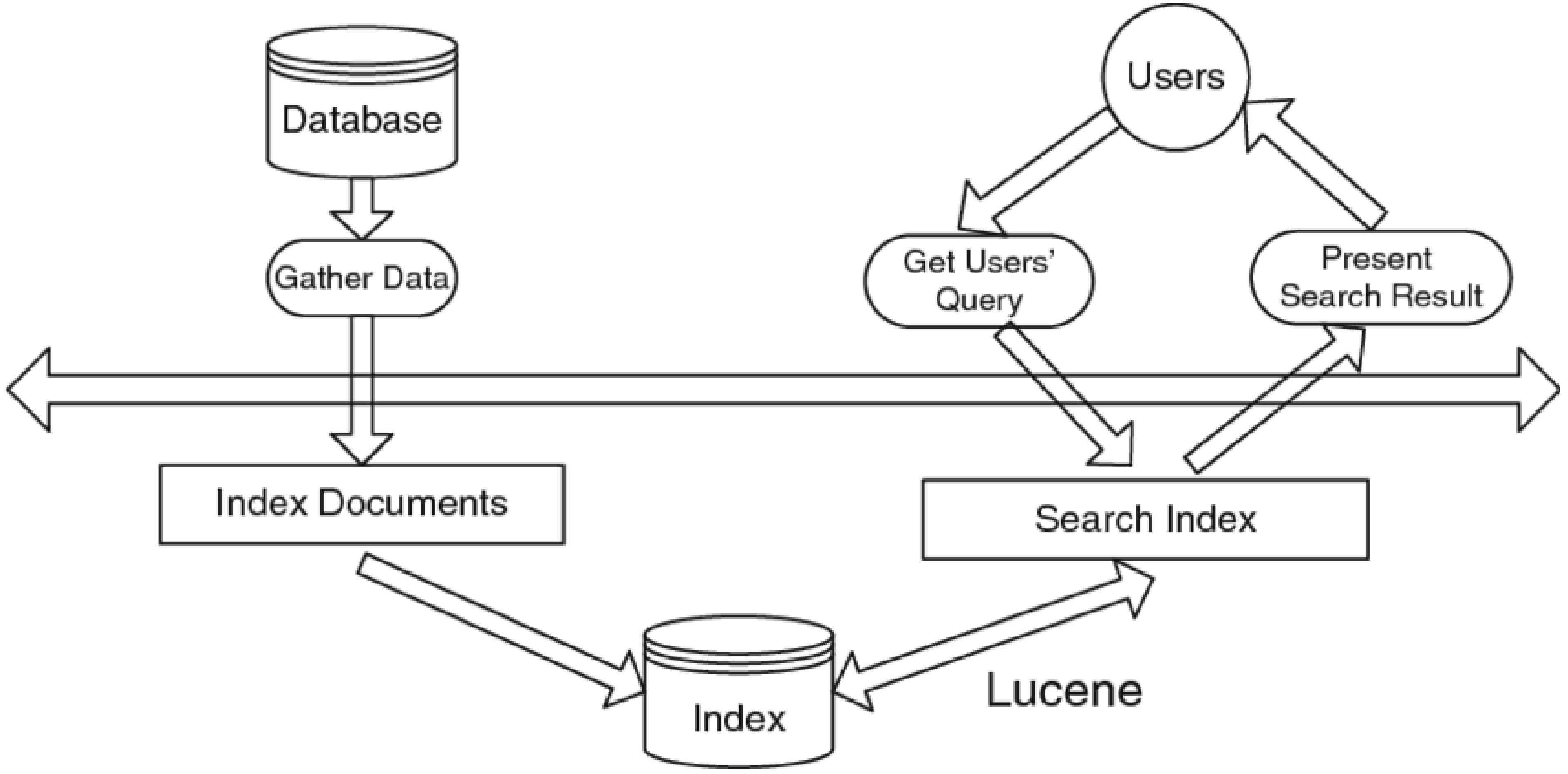
Haoran Yang

2022.02.16

Popular IR packages

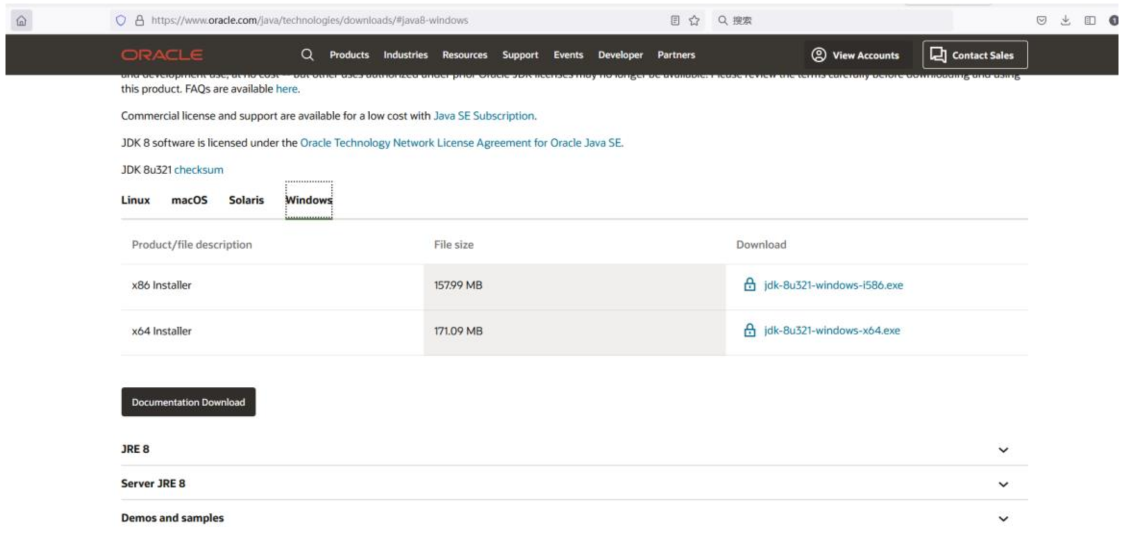
- For academic use:
  - Galago (developed by CIIR center, Umass)
  - Anserini (based on Lucene, focus on IR reproducibility)
  - trec_eval (an official IR evaluation tool developed by TREC)
  - ….
- For commercial use
  - Lucene (low-level full text search library)
  - Elasticsearch (based on Lucene, enhanced support for distributed environments)
  - Solr (based on Lucene, enhanced performance with more functionalities)
  - Sphinx (a SQL database-like full text search engine)
  - …

General Architecture of Search Engine



Use Lucene with Java Development Kits (JDK)

1. Download and install JDK (1.8.x or 1.9.x)



this product. FAQs are available here.

Commercial license and support are available for a low cost with Java SE Subscription.

JDK 8 software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE.

JDK 8u321 checksum

Linux macOS Solaris Windows

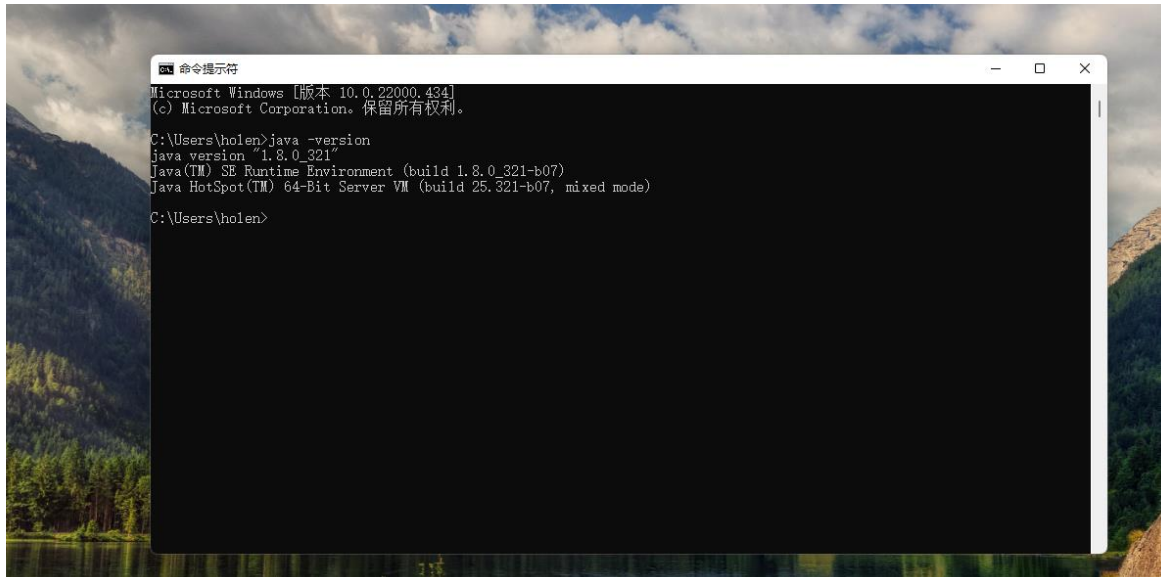
Product/file description	File size	Download
x86 Installer	157.99 MB	jdk-8u321-windows-i586.exe
x64 Installer	171.09 MB	jdk-8u321-windows-x64.exe

Documentation Download

JRE 8

Server JRE 8

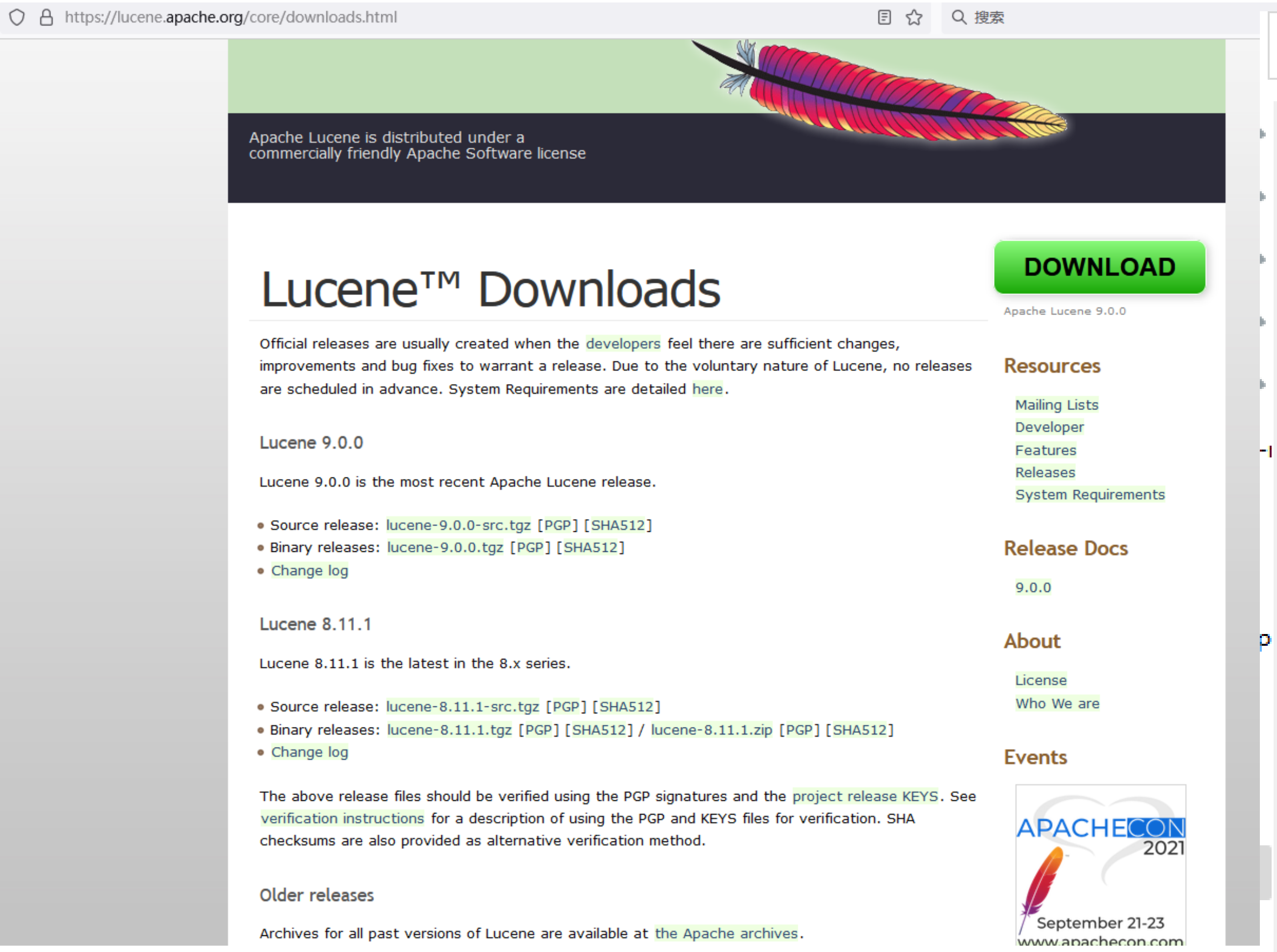
Demos and samples



```
Microsoft Windows [版本 10.0.22000.434]
(c) Microsoft Corporation。保留所有权利。

C:\Users\holen>java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.321-b07, mixed mode)

C:\Users\holen>
```

Use Lucene with Java Development Kits (JDK)

2. Visit https://lucene.apache.org/core/downloads.html to download lucene


https://lucene.apache.org/core/downloads.html

Apache Lucene is distributed under a commercially friendly Apache Software license

Lucene™ Downloads

Official releases are usually created when the developers feel there are sufficient changes, improvements and bug fixes to warrant a release. Due to the voluntary nature of Lucene, no releases are scheduled in advance. System Requirements are detailed here.

Lucene 9.0.0

Lucene 9.0.0 is the most recent Apache Lucene release.

- Source release: lucene-9.0.0-src.tgz [PGP] [SHA512]
- Binary releases: lucene-9.0.0.tgz [PGP] [SHA512]
- Change log

Lucene 8.11.1

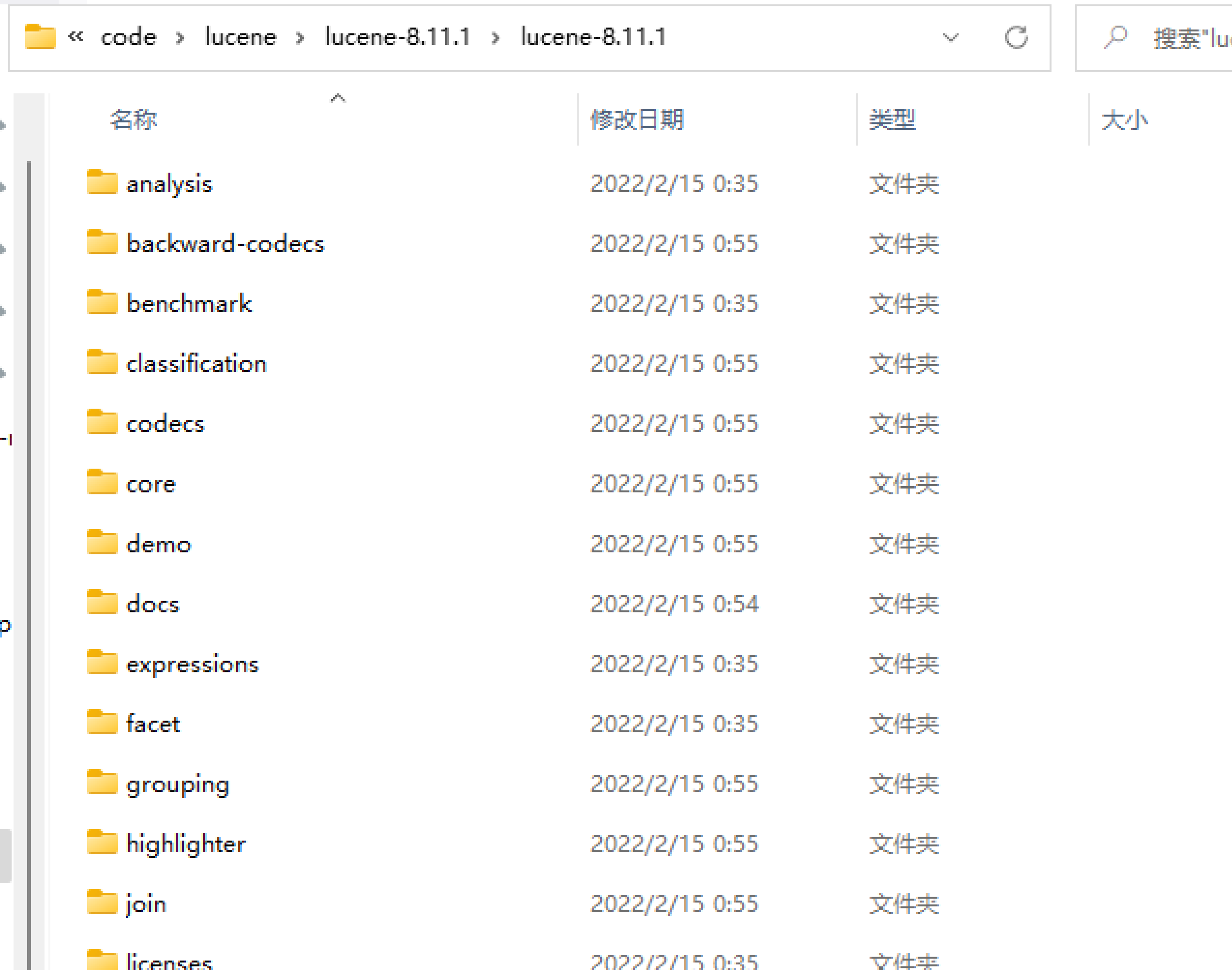
Lucene 8.11.1 is the latest in the 8.x series.

- Source release: lucene-8.11.1-src.tgz [PGP] [SHA512]
- Binary releases: lucene-8.11.1.tgz [PGP] [SHA512] / lucene-8.11.1.zip [PGP] [SHA512]
- Change log

The above release files should be verified using the PGP signatures and the project release KEYS. See verification instructions for a description of using the PGP and KEYS files for verification. SHA checksums are also provided as alternative verification method.

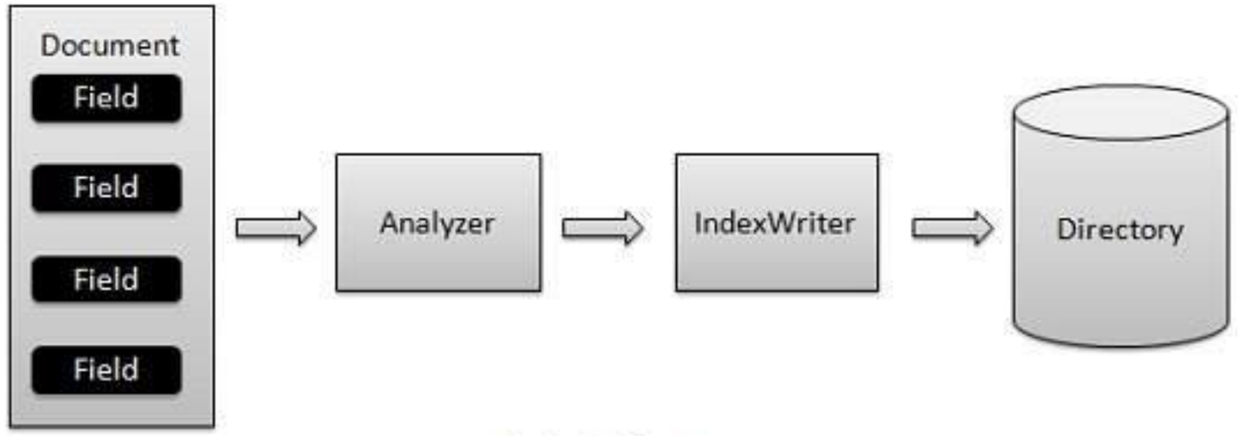
Older releases

Archives for all past versions of Lucene are available at the Apache archives.


« code › lucene › lucene-8.11.1 › lucene-8.11.1

名称	修改日期	类型	大小
analysis	2022/2/15 0:35	文件夹	
backward-codecs	2022/2/15 0:55	文件夹	
benchmark	2022/2/15 0:35	文件夹	
classification	2022/2/15 0:55	文件夹	
codecs	2022/2/15 0:55	文件夹	
core	2022/2/15 0:55	文件夹	
demo	2022/2/15 0:55	文件夹	
docs	2022/2/15 0:54	文件夹	
expressions	2022/2/15 0:35	文件夹	
facet	2022/2/15 0:35	文件夹	
grouping	2022/2/15 0:55	文件夹	
highlighter	2022/2/15 0:55	文件夹	
join	2022/2/15 0:55	文件夹	
licenses	2022/2/15 0:35	文件夹	

Building index in Lucene

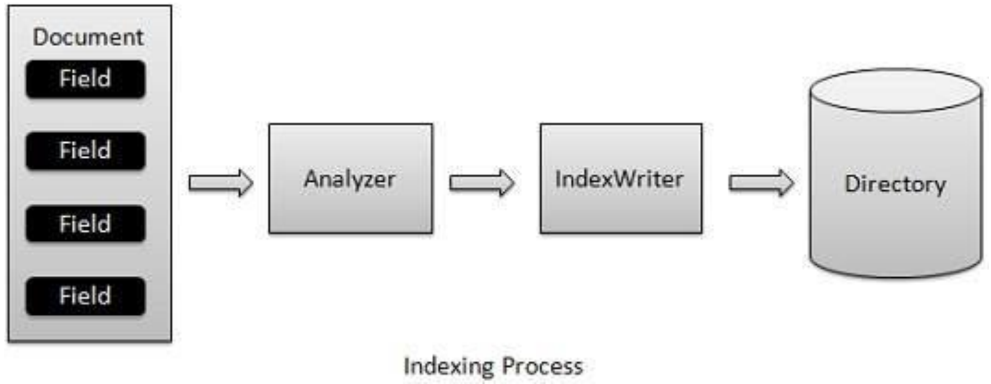


Indexing Process

Document are the unit of indexing and search. A Document is a set of fields. **Each field** has a name and a textual value. A field may be stored with the document, in which case it is returned with search hits on the document. Thus each document should typically contain one or more stored fields which uniquely identify it.
Term: Its expression is a “word” in the text.
Token: term+metadata of the term (term type, start and offset position of term in the text.)

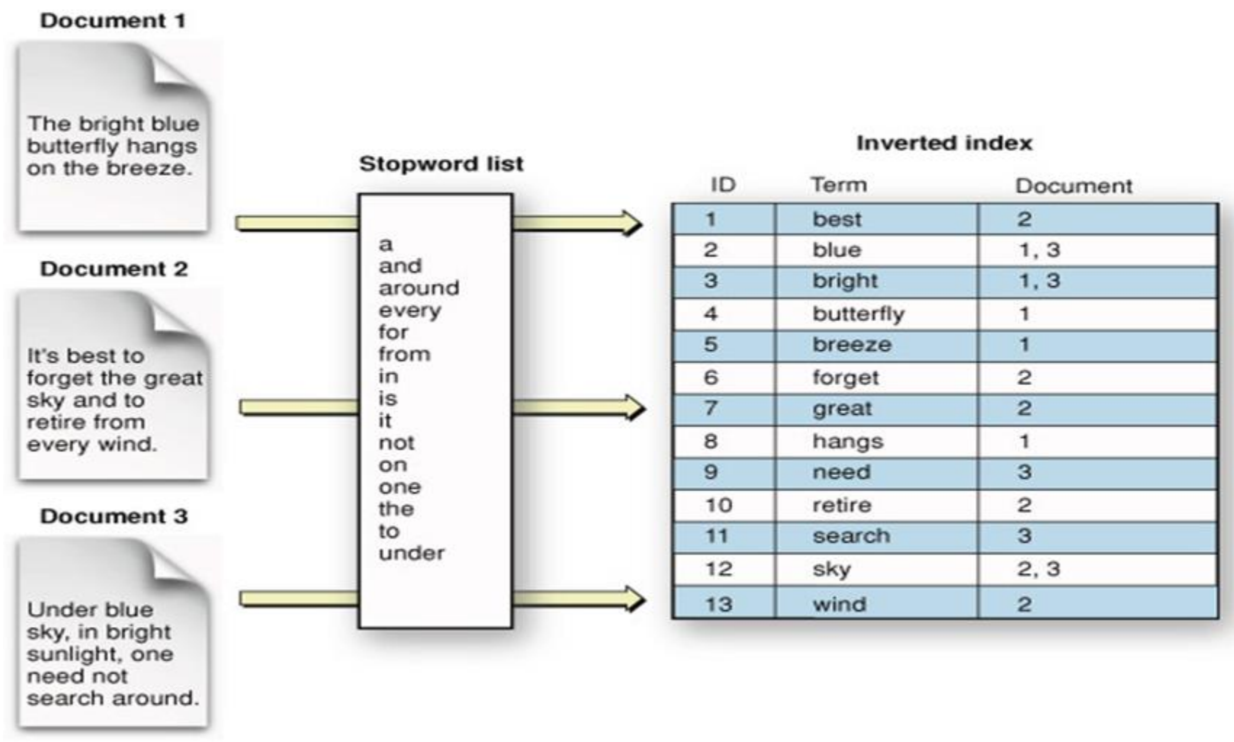
For example, A Chinese poem is a document. It contains fields like content, title, author, dynasty…

Building index in Lucene



Analyzer: tokenizer+filter(s)

Tokenizer: Split the text into tokens. The token contains a lot of information, such as the position of term in the text, the original text of term, and the length of term…
Filters: Lowercase filter, ASCII folding filter, Synonyms filter, Multiple language-stemming filter



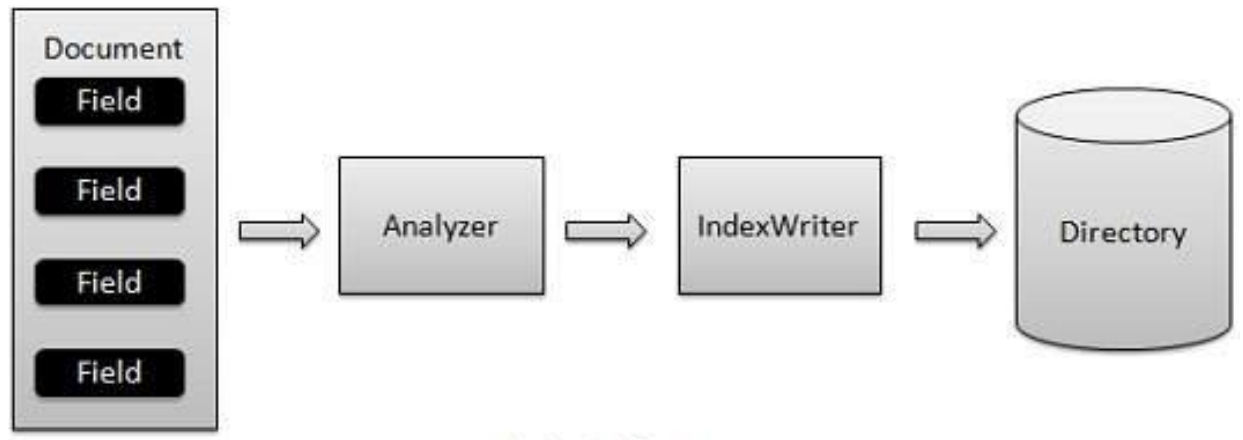
Document 1: The bright blue butterfly hangs on the breeze.

Document 2: It's best to forget the great sky and to retire from every wind.

Document 3: Under blue sky, in bright sunlight, one need not search around.

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Building index in Lucene



Indexing Process

Indexwriter is the core component of the indexing process. This class is responsible for creating a new index, opening an existing index, and adding, deleting or updating the information of the indexed document to the index, but it cannot read or search the index.
Indexwriter needs to open up a certain space to store indexes, which is completed by **directory**

Lucene demo

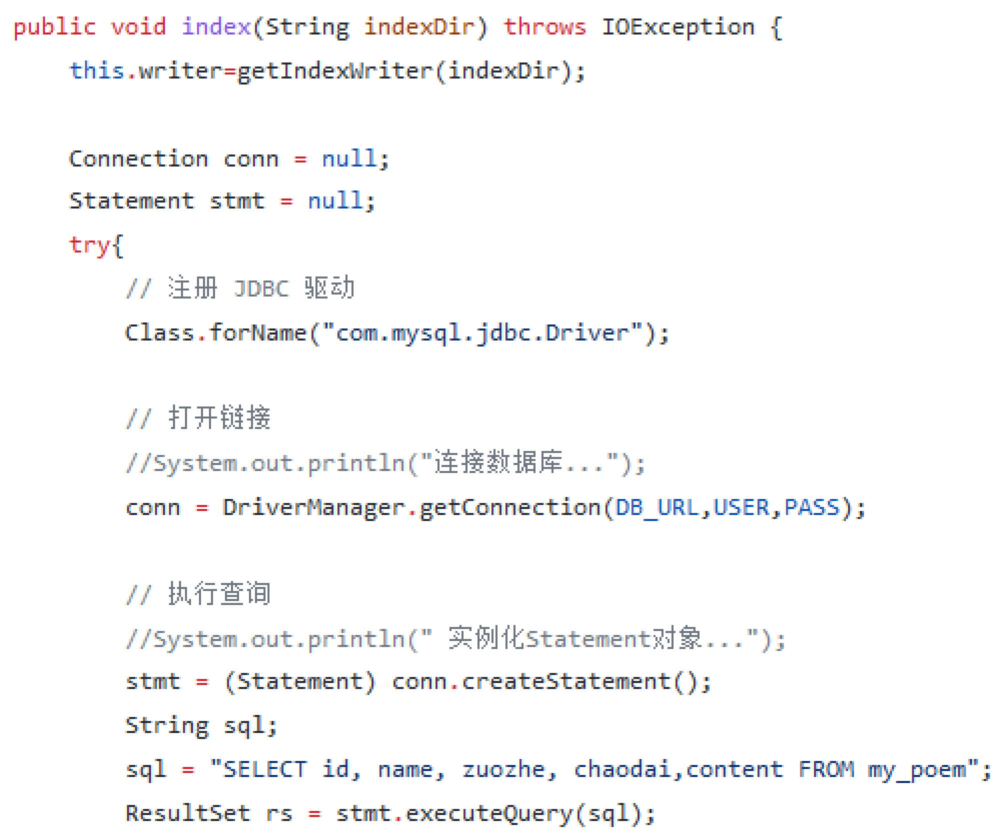
- Firstly I will show you the index building process (in BuildIndex.java)

Lucene demo

- Create an Analyser
  - Options
    - WhitespaceAnalyzer
      - divides text at whitespace
    - SimpleAnalyzer
      - divides text at non-letters
      - convert to lower case
    - StopAnalyzer
      - SimpleAnalyzer
      - removes stop words
    - StandardAnalyzer
      - good for most European Languages
      - removes stop words
      - convert to lower case

Lucene demo

- Firstly I will show you the index building process (in BuildIndex.java)

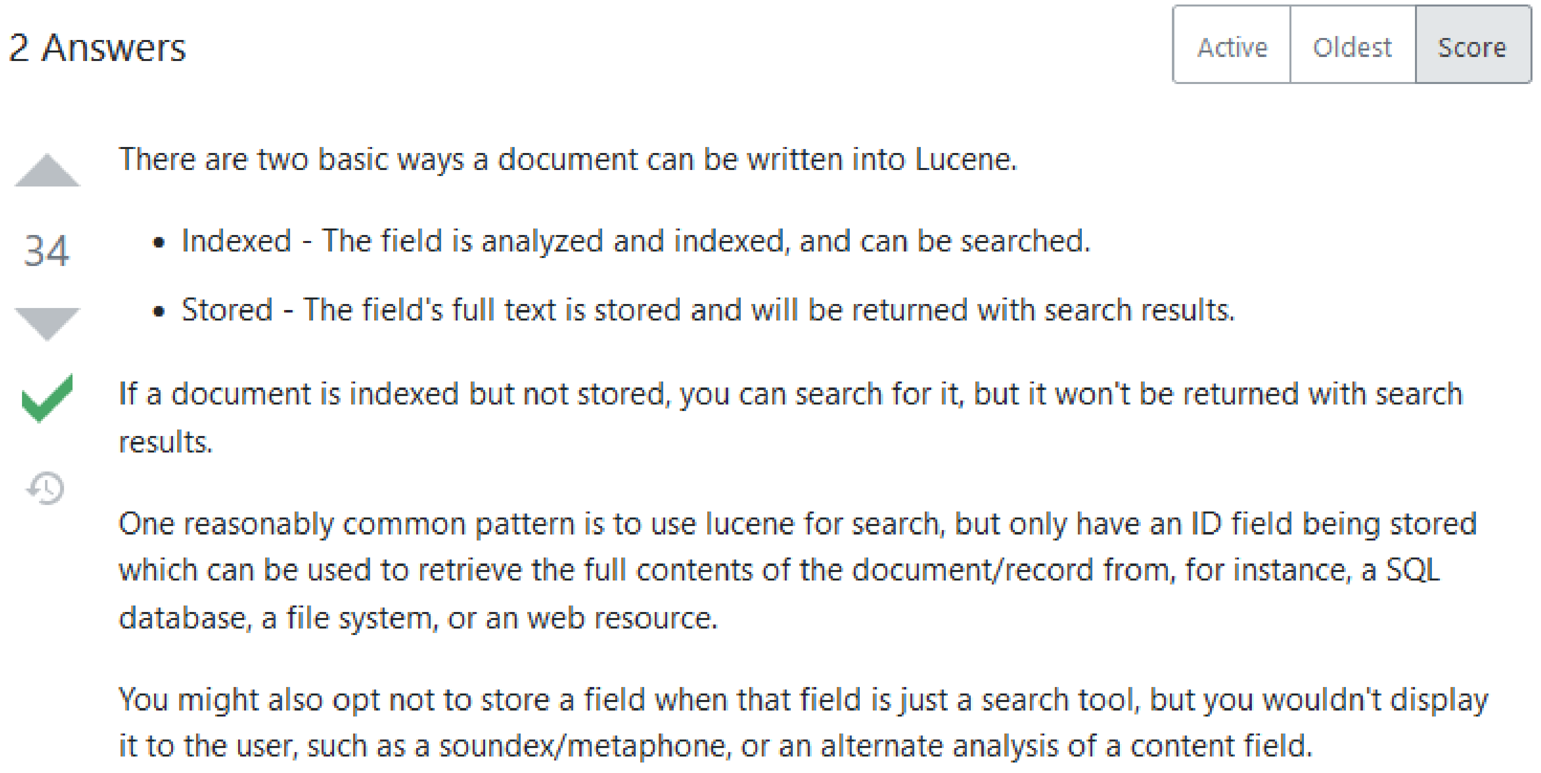


```
public void index(String indexDir) throws IOException {
    this.writer=getIndexWriter(indexDir);

    Connection conn = null;
    Statement stmt = null;
    try{
        // 注册 JDBC 驱动
        Class.forName("com.mysql.jdbc.Driver");

        // 打开链接
        //System.out.println("连接数据库...");
        conn = DriverManager.getConnection(DB_URL,USER,PASS);

        // 执行查询
        //System.out.println(" 实例化Statement对象...");
        stmt = (Statement) conn.createStatement();
        String sql;
        sql = "SELECT id, name, zuozhe, chaodai,content FROM my_poem";
        ResultSet rs = stmt.executeQuery(sql);
```

Load data from database
for index building

Lucene demo

- `Field.Store.`***YES/*** `Field.Store.`***No***



There are two basic ways a document can be written into Lucene.

- Indexed - The field is analyzed and indexed, and can be searched.
- Stored - The field's full text is stored and will be returned with search results.

If a document is indexed but not stored, you can search for it, but it won't be returned with search results.

One reasonably common pattern is to use lucene for search, but only have an ID field being stored which can be used to retrieve the full contents of the document/record from, for instance, a SQL database, a file system, or an web resource.

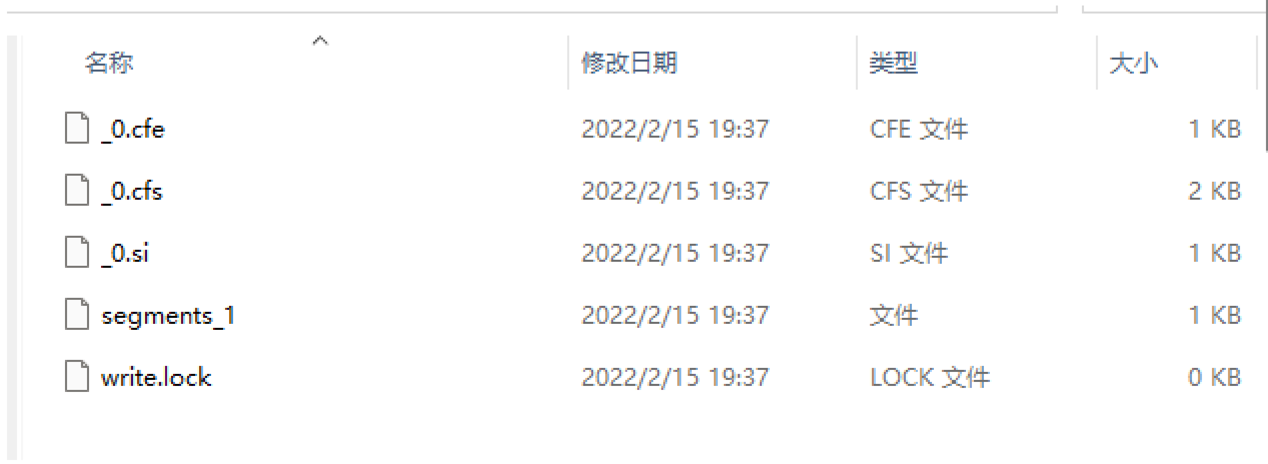
You might also opt not to store a field when that field is just a search tool, but you wouldn't display it to the user, such as a soundex/metaphone, or an alternate analysis of a content field.

https://stackoverflow.com/questions/32940650/lucene-field-store-yes-versus-field-store-no

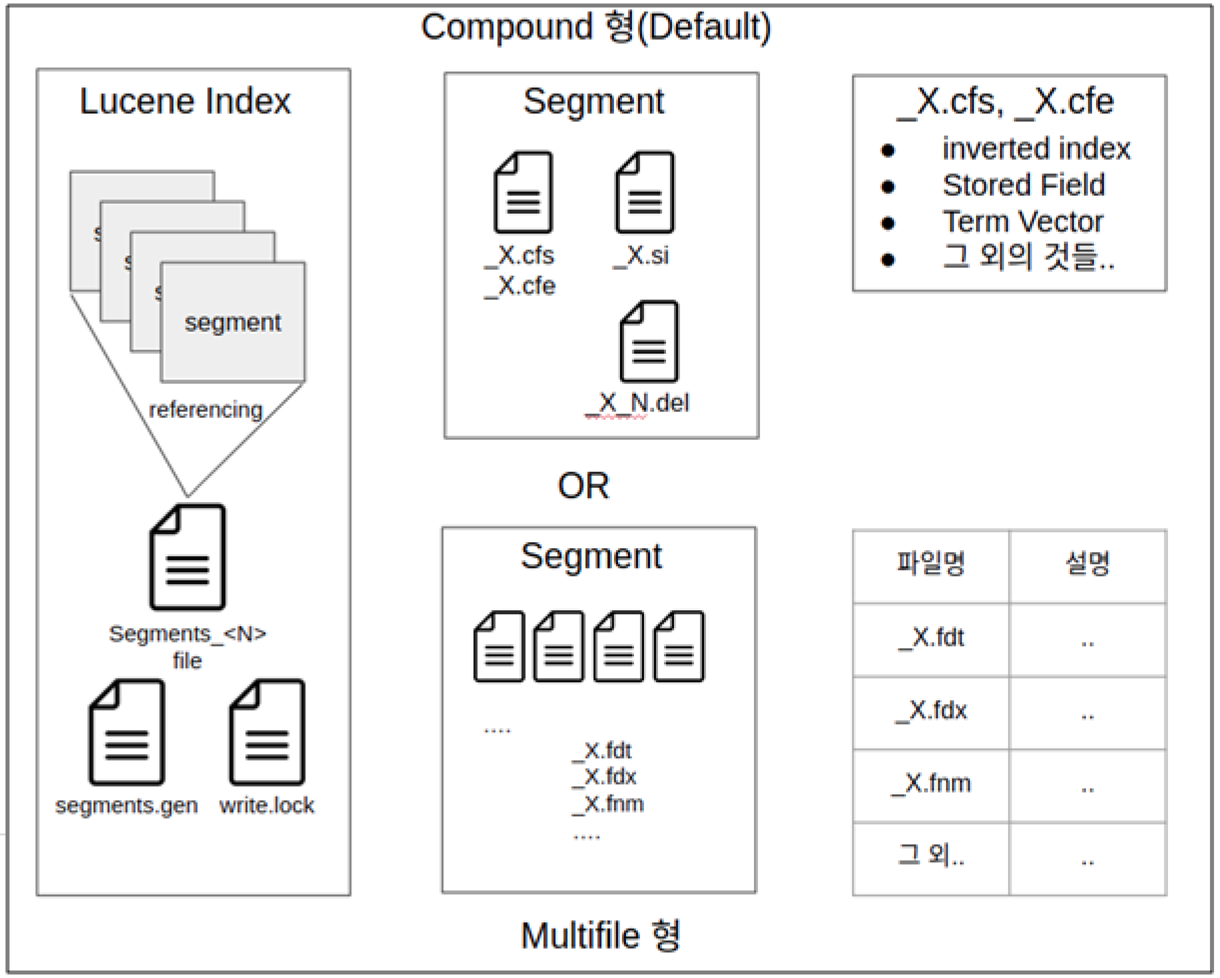
Luence Index – Behind the Scene

The **same prefix** in an index directory means that these files belong to the same **segment.**

Lucene can store segments in two ways:
Multifile format: .nvd, .fnm, .fdt .....
Compound format:CFS(Compound File System)



名称	修改日期	类型	大小
_0.cfe	2022/2/15 19:37	CFE 文件	1 KB
_0.cfs	2022/2/15 19:37	CFS 文件	2 KB
_0.si	2022/2/15 19:37	SI 文件	1 KB
segments_1	2022/2/15 19:37	文件	1 KB
write.lock	2022/2/15 19:37	LOCK 文件	0 KB



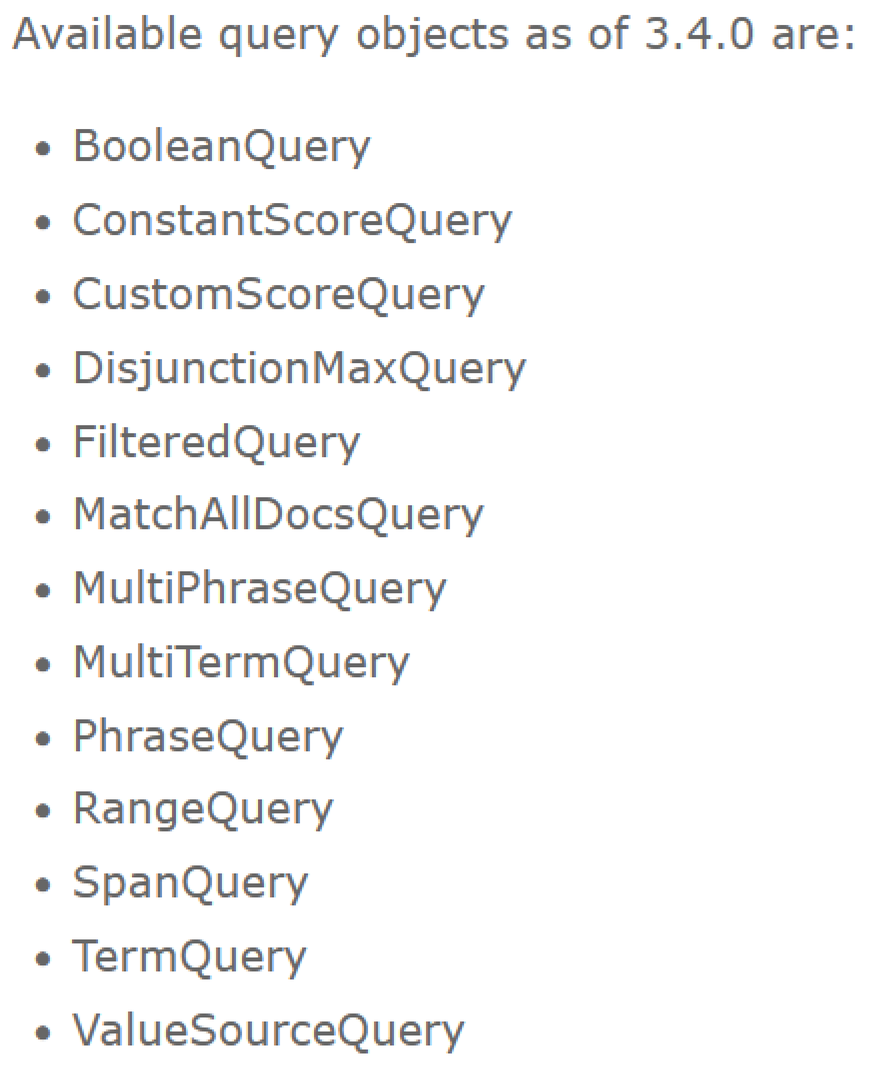
파일명	설명
_X.fdt	..
_X.fdx	..
_X.fnm	..
그 외..	..

Search

- Firstly I will show you the index building process (in BuildIndex.java)
- Secondly, after building the index, we can search given a query.

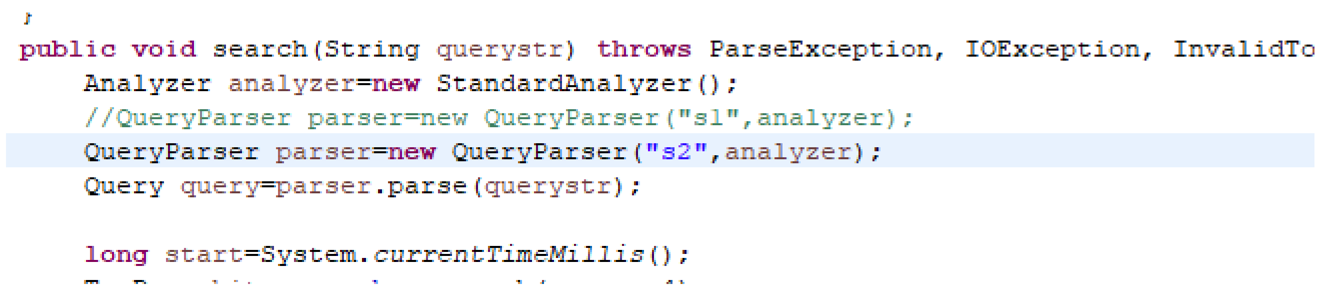
Search

- Types of query:
  - Boolean: [IST441 Giles] [IST441 OR Giles] [java AND NOT SUN]
  - wildcard: [nu?ch] [nutc*]
  - phrase: [“JAVA TOMCAT”]
  - proximity: [“lucene nutch” ~10]
  - fuzzy: [roam~] matches roams and foam
  - date range
  - …

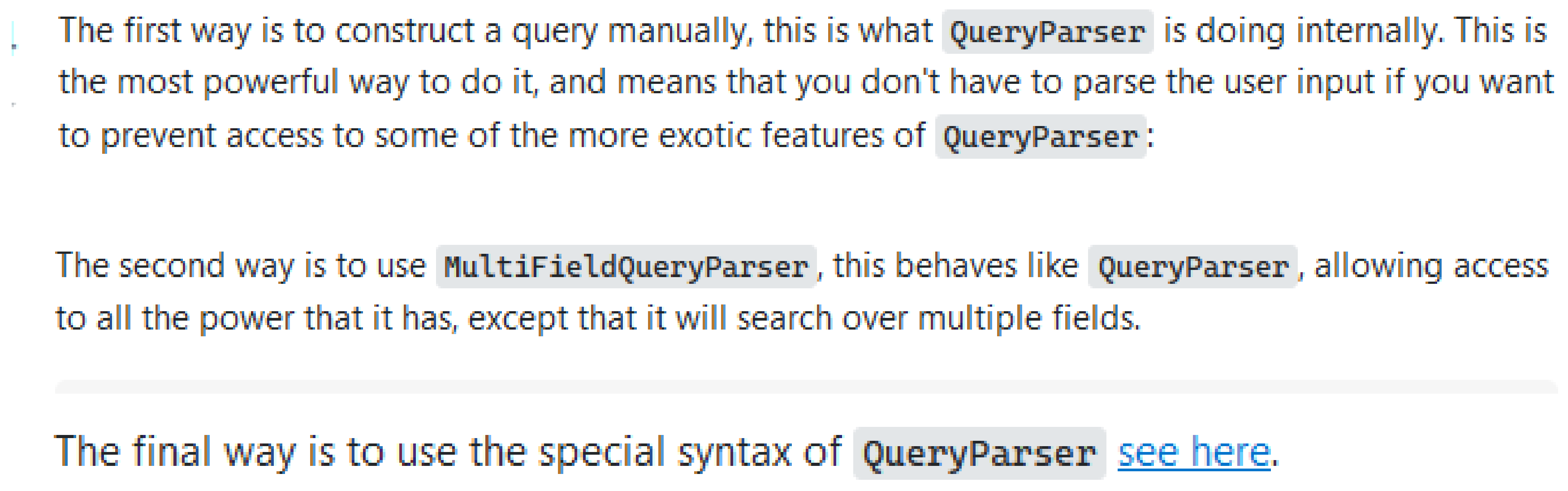

Available query objects as of 3.4.0 are:

- BooleanQuery
- ConstantScoreQuery
- CustomScoreQuery
- DisjunctionMaxQuery
- FilteredQuery
- MatchAllDocsQuery
- MultiPhraseQuery
- MultiTermQuery
- PhraseQuery
- RangeQuery
- SpanQuery
- TermQuery
- ValueSourceQuery

MultiField Search



```
public void search(String querystr) throws ParseException, IOException, InvalidTo
    Analyzer analyzer=new StandardAnalyzer();
    //QueryParser parser=new QueryParser("s1",analyzer);
    QueryParser parser=new QueryParser("s2",analyzer);
    Query query=parser.parse(querystr);

    long start=System.currentTimeMillis();
```



The first way is to construct a query manually, this is what `QueryParser` is doing internally. This is the most powerful way to do it, and means that you don't have to parse the user input if you want to prevent access to some of the more exotic features of `QueryParser`:

The second way is to use `MultiFieldQueryParser`, this behaves like `QueryParser`, allowing access to all the power that it has, except that it will search over multiple fields.

The final way is to use the special syntax of `QueryParser` see here.

https://stackoverflow.com/questions/2005084/how-to-specify-two-fields-in-lucene-queryparser

Default Scoring Function

https://lucene.apache.org/core/4_5_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

Factors need to be considered:

Document boost: The weight value set for a document when indexing.
Field boost: The weight value set for a field during query.
Query boost: At search time users can specify boosts to each query, sub-query, and each query term
Coord: The adjustment factor calculated based on the number of query keywords contained in the document.
Inerse document frequency: The scoring formula uses this factor to increase the weight of documents containing rare words.
Length norm: The longer the text of the field, the lower the weight of the factor.
Term frequency: you all know
Query norm*:* *is a normalizing factor used to make scores between queries comparable*.

$$\text{score}(q,d) \;=\; \text{coord-factor}(q,d) \cdot \text{query-boost}(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot \text{doc-len-norm}(d) \cdot \text{doc-boost}(d)$$

Lucene Conceptual Scoring Formula

Default Scoring Function

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d) \right)$$

Lucene Practical Scoring Function

$$\text{tf}(t \text{ in } d) = \text{frequency}^{1/2}$$

$$\text{idf}(t) = 1 + \log \left(\frac{\text{numDocs}}{\text{docFreq}+1} \right)$$

Document length norm *doc-len-norm(d)* and document boost *doc-boost(d)* are known at indexing time. They are computed in advance and their multiplication is saved as a single value in the index: *norm(d)*. (In the equations below, *norm(t in d)* means *norm(field(t) in doc d)* where *field(t)* is the field associated with term *t*.)

Python alternatives for Lucene

- PyLucene
  - PyLucene is not a Lucene port but a Python wrapper around Java Lucene. PyLucene embeds a Java VM with Lucene into a Python process. The PyLucene Python extension, a Python module called lucene is machine-generated by JCC.
  - https://lucene.apache.org/pylucene/
  - Requires manual compilation and installation (major disadvantage)
  - Works for both Linux and Windows

Appendix