

Introduction to Lucene

Lecture 2b
CS 410/510
Information Retrieval on the Internet

Lucene

- An open source project, part of Apache
- A set of libraries, or a toolkit, for **building** a search engine
- Written in Java; ported to some other languages
 - Only the Java version will be supported in this course
- API available on the web

2

Lucene

- <http://lucene.apache.org/java/docs/index.html>
- <http://lucene.apache.org/java/docs/features.html>
- <http://wiki.apache.org/jakarta-lucene/LuceneFAQ>
- <http://lucene.apache.org/java/docs/gettingstarted.html>
- <http://lucene.apache.org/java/docs/demo.html>
- <http://lucene.apache.org/java/docs/api/>

3

Organization of Lucene

- Core libraries
 - Contain code for dealing with text processing, documents, indexing, parsing queries, and searching the index
 - <http://lucene.apache.org/java/docs/api/>
- External resources that others have contributed

4

Building a Search Engine

Need

1. A collection of documents to be indexed
2. A program to build an index
3. A user interface
4. A program to search the index for documents to match the query

Numbers 3 & 4 can be combined

5

Lucene Demo Programs

- Described in detail in the “Getting Started” pages of the Lucene website
- The demo programs and the source code come with the Lucene download
- IndexFiles.java
 - Basic code to index all the files in a given directory
- SearchFiles.java
 - Basic code to get a user query from the command line, search the indexed documents, and return results

6

```

try {
    IndexWriter writer = new IndexWriter(INDEX_DIR, new StandardAnalyzer(), true);
    System.out.println("Indexing to directory " + INDEX_DIR + "...");
    indexDocs(writer, docDir);
    System.out.println("Optimizing...");
    writer.optimize();
    writer.close();
} catch (IOException e) {}
}

static void indexDocs(IndexWriter writer, File file) throws IOException {
    // do not try to index files that cannot be read
    if (file.canRead()) {
        if (file.isDirectory()) {
            String[] files = file.list();
            // an IO error could occur
            if (files != null) {
                for (int i = 0; i < files.length; i++) {
                    indexDocs(writer, new File(file, files[i]));
                }
            }
        } else {
            System.out.println("adding " + file);
        }
    }
}

```

Excerpts from IndexFiles.java
in the demo

7

```

IndexReader reader = IndexReader.open(index);

Searcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
QueryParser parser = new QueryParser(field, analyzer);
while (true) {
    if (queries == null) // prompt the user
        System.out.print("Query: ");

    String line = in.readLine();

    Query query = parser.parse(line);

    Hits hits = searcher.search(query);

    final int HITS_PER_PAGE = 10;
    for (int start = 0; start < hits.length(); start += HITS_PER_PAGE) {
        int end = Math.min(hits.length(), start + HITS_PER_PAGE);
        for (int i = start; i < end; i++) {

            Document doc = hits.doc(i);
            String title = doc.get("title");
            if (title != null) {
                System.out.println(" Title: " + doc.get("title"));
            }
        }
    }
}

```

Excerpts from SearchFiles.java
in the demo

8

Assignment 3

- Preliminaries
 - Download Lucene 2.0 and the source code
 - Look at, run, the demo programs
 - Use the source code for the demos and the Lucene API to understand what's happening

9

Assignment 3

- Assignment
 - Create your own simple search engine
 - You can base your code on the demos and copy from the demo source code
 - Index the provided files
 - Answer some questions
 - Modify your search engine
 - Answer more questions
 - Submit your answers and your code

10

Backup screen shots

11



Apache Lucene - Features

- Features
 - High performance
 - Scalable, accurate and efficient search algorithms
 - Powerful, accurate and efficient search algorithms
 - Cross-Platform Solution

Lucene offers powerful features through a simple API:

Scalable, High-Performance Indexing

- over 200M records on a 100M file
- small RAM requirements - fits into RAM
- incremental indexing - add as documents
- Index size roughly 20-30% the size of raw indexed

Powerful, Accurate and Efficient Search Algorithms

- word searching - Best results returned first
- many powerful query types (phrase queries, wild-card queries, proximity queries, range queries and more)
- related searching (e.g., file authors, categories)
- date range searching
- sorting by any field
- document searching with merged results
- direct simultaneous update and searching

Cross-Platform Solution

- works on open source software under the **Apache License** which lets you use Lucene in both commercial and open source programs
- 100% Java code
- implementations in other programming languages available that are index-compatible

LuceneFAQ - Jakarta-lucene Wiki

This is the official Lucene FAQ

If you have a question about using Java Lucene, please do not add it directly to this FAQ. Join the [Java User mailing list](#) and email your question there. Questions should only be added to this Wiki page when they already have an answer that can be added at the same time.

1. Lucene FAQ

1. How do I get started with Lucene?
2. Are there any training tools available?
3. How does Lucene relate to other search engines?
4. How does Lucene work with the Java API?
5. How can I test the speed of my development code?
6. Where can I get the source code for the Java API?
7. Where does the name Lucene come from?
8. Are there any alternatives to Lucene?
9. Does Lucene have a public contest?
10. Why did you change the version number from 2.4 to 3.0?
11. What's the difference between Lucene 2.4 and Lucene 3.0?
12. How do I contribute to the project?
13. How do I report a bug?
14. How do I report a security vulnerability?
15. How do I get the source code for Lucene 2.4 and Lucene 3.0?

2. Searching

1. How does Lucene allow searching and indexing simultaneously?
2. Why can't I search on the fly (incremental)?
3. Why can't I search on the fly (incremental)?
4. How can I search over multiple fields?
5. What sort of search support is available from Lucene?
6. Is the QueryParser thread-safe?
7. How do I restrict searches to only return results from a limited subset of documents in the index for a particular request? What is the best way to approach this?

Apache Lucene - Getting Started Guide

This document is intended as a "getting started" guide. It has three audiences: first time users looking to install Apache Lucene in their application or web service, developers looking to build or tune the applications they develop on Lucene, and developers looking to become involved in and contribute to the development of Lucene. This document is written in tutorial and walk-through format. The goal is to help you "get started". It does not go into great depth on some of the constructs or inner details of Lucene.

Each section below builds on one another. More advanced users may wish to skip sections:

- [About the command-line Lucene demo and its usage](#). This section is intended for anyone who wants to use the command-line Lucene demo.
- [About the Lucene API and how to use it](#). This section walks through the implementation details (sources) of the API.
- [About indexing and searching the demo Lucene web application](#). While this walk-through assumes a familiarity with a container of choice, there is no prerequisite knowledge assumed and the information is your container. This section is intended for those responsible for the application that will be used to demonstrate the Lucene API.
- [About the Lucene API to construct the demo Lucene web application](#). Please note the template application is designed to highlight features of the Lucene API and is not intended to be a production-ready application. Use the information such as provided by Lucene API and regular file filtering you may use to that would not be beyond the scope of this guide. This section is intended for developers and those willing to customize the demo template web application to their needs.

Apache Lucene - Building and Installing the Basic Demo

This document is intended as a "getting started" guide to using and running the Lucene demo. It walks you through some basic installation and configuration.

About the Demo

The Lucene demo is a Java code sample that demonstrates various functionalities of Lucene and how one should go about adding Lucene to their applications.

Setting your CLASSPATH

First, you should get the latest Lucene distribution and then extract it to a working directory. Alternatively, you can [check out the sources from Subversion](#) and then run `ant -Dsrc=src` to generate the JARs and WADs.

You should use the Lucene JARs in the directory you created when you extracted the archive. It should be named something like `Lucene-core-xxxxxx.jar`. This should also use the JARs from `Lucene-demo-xxxxxx.jar`. If you checked out the sources from Subversion then the JARs are located under the build subdirectory (after running `ant` successfully). For both of these files in your `CLASSPATH`.

Indexing Files

Once you've gotten this far you're probably itching to go. Let's **build an index**. Assuming you've set your `CLASSPATH` correctly, just type:

```
java org.apache.lucene.demo.IndexFiles -I C:\path-to-lucene\src
```

This will produce a subdirectory called `Index` which will contain an index of all of the Lucene source code.

To **search the Index**, type:

```
java org.apache.lucene.demo.SearchIndex
```

You'll be prompted for a query. Type in a search word and press the enter key. You'll see that the Lucene developers are very well organized and get no results. Now try entering the word "vector". That should return a whole bunch of documents. The results will give at every path used and ask you whether

Apache Lucene - Basic Demo Sources Walk-through

About the Code

In this section, we walk through the sources behind the command-line Lucene demo, where to find them, their parts and their function. This section is intended for Java developers wanting to understand how to use Lucene in their applications.

Location of the source

Relative to the directory created when you extracted Lucene or retrieved it from Subversion, you should see a directory called `src` which in turn contains a subdirectory called `demo`. This is the root for all of the Lucene demo, under the directory `org.apache.lucene.demo`. This is where all the Java sources for the demo live.

Within this directory you should see the `IndexFiles.java` class we mentioned earlier. Bring it up to you or your editor of choice and let's take a look at it.

IndexFiles

As we discussed in the previous walk-through, the `IndexFiles` class creates a Lucene Index. Let's take a look at how it does this.

The first substantial thing the `IndexFiles` class does is instantiate `IndexWriter`. If passed the string "index" and a new instance of a class called `IndexWriter`. The "index" string is the name of the directory where all index information should be stored. Because we're not passing a full path, this will be created as a subdirectory of the current working directory (if it does not already exist). On some platforms, it may be created in other directories (such as the user's home directory).

The `IndexWriter` is the main class responsible for creating indexes. To use it you must instantiate it with a path that can write the index into. If this path does not exist it will first create it. Otherwise it will verify the index at that path. This can also create an index using one of the subclasses of `IndexWriter`. In any case, you must also pass an instance of `org.apache.lucene.analysis.Analyzer`.

The particular analyzer we are using, `StandardAnalyzer`, is more than a standard Java tokenizer, converting all strings to lowercase and filtering out common words and characters from the index. It is useful mostly and character's mean common language words such as articles (a, an, the, etc.) and other strings that would be useless for searching (e.g., " "). It should be noted that there are different index file every language, and you should use the proper analyzer for each. Lucene currently provides analyzers for a number of different languages (see the "Analyzer" Java sources under `src/org/apache/lucene/demo/Analyzer`).

Overview (Lucene nightly API)

Lucene nightly API

Apache Lucene is a high-performance, full featured text search engine library.

Core

org.apache.lucene	Top-level package
org.apache.lucene.analysis	API and code to convert text into searchable tokens
org.apache.lucene.analysis.standard	A general purpose tokenizer restricted with <code>TermAttribute</code>
org.apache.lucene.analysis.tokenattributes	The <code>TokenStream</code> interface
org.apache.lucene.analysis.tokenattributes.attributes	Code to manage and access attributes
org.apache.lucene.analysis.tokenattributes.attributes.attributes	A single query aware implementation with <code>TermAttribute</code>
org.apache.lucene.analysis.tokenattributes.attributes.attributes.attributes	Table <code>OK</code> columns
org.apache.lucene.analysis.tokenattributes.attributes.attributes.attributes.attributes	The table of open
org.apache.lucene.analysis.tokenattributes.attributes.attributes.attributes.attributes.attributes	Terms in <code>OK</code> will be all index data
org.apache.lucene.analysis.tokenattributes.attributes.attributes.attributes.attributes.attributes.attributes	Open table classes

Demos

org.apache.lucene.demo	Lucene demo
org.apache.lucene.demo.index	Indexing demo
org.apache.lucene.demo.search	Searching demo

contrib.analysis

Overview Package Class Use Tree Deprecated Index Help

Lucene nightly API

Apache Lucene is a high-performance, full-featured text search engine library

See: [Description](#)

Core

org.apache.lucene	Top-level package
org.apache.lucene.analysis	API and code to convert text into indexable tokens
org.apache.lucene.analysis.standard	A grammar-based tokenizer constructed with JavaCC
org.apache.lucene.document	The Document abstraction
org.apache.lucene.index	Code to maintain and access indices
org.apache.lucene.queryParser	A simple query parser implemented with JavaCC
org.apache.lucene.search	Table Of Contents
org.apache.lucene.search.spans	The calculus of spans
org.apache.lucene.store	Binary I/O API, used for all index data
org.apache.lucene.util	Some utility classes

Demo

[org.apache.lucene.demo](#)

Package org.apache.lucene.analysis

API and code to convert text into indexable tokens

See: [Description](#)

Class Summary

Analyzer	An Analyzer builds TokenStreams, which analyze text
CharTokenizer	An abstract base class for simple, character-oriented tokenizers
ISOLatinAccentFilter	A filter that replaces accented characters in the ISO Latin 1 character set (ISO-8859-1) by their unaccented
KeywordAnalyzer	"Tokenizes" the entire stream as a single token.
KeywordTokenizer	tokenizes the entire input as a single token.
LengthFilter	Removes words that are too long and too short from the stream
LetterTokenizer	A LetterTokenizer is a tokenizer that divides text at non-letters
LowerCaseFilter	Normalizes token text to lower case
LowerCaseTokenizer	LowerCaseTokenizer performs the functions of LetterTokenizer and LowerCaseFilter together
PorterAnalyzer	This analyzer is used to facilitate scenarios where different fields require different analysis techniques
PorterStemFilter	Transforms the token stream as per the Porter stemming algorithm
SimpleAnalyzer	An Analyzer that filters LetterTokenizer with LowerCaseFilter
StopAnalyzer	Filters LetterTokenizer with LowerCaseFilter and StopFilter
StopFilter	Removes stop words from a token stream
Token	A Token is an occurrence of a term from the text of a field
TokenFilter	A TokenFilter is a TokenStream whose input is another token stream
Tokenizer	A Tokenizer is a TokenStream whose input is a Reader
TokenStream	A TokenStream enumerates the sequence of tokens, either from fields of a document or from query text
WhitespaceAnalyzer	An Analyzer that uses WhitespaceTokenizer
WhitespaceTokenizer	A WhitespaceTokenizer is a tokenizer that divides text at whitespace

org.apache.lucene.document (Lucene nightly API) - Mozilla Firefox

Package org.apache.lucene.document

The Document abstraction

See: [Description](#)

Interface Summary

FieldSelector	Documents with fields
FieldSelector	Define a FieldSelector, the FieldSelector allows one to make decisions about what Fields are included in a Document, by

Class Summary

AbstractField	Deprecated. If you build a new index, use FieldSelector instead.
BooleanField	Provides support for converting dates to strings and vice versa
DateField	Provides for date granularity
DocValuesField	Documents are the unit of indexing and search
Field	A field is a section of a Document
FieldSelector	Specifies whether and how a field should be indexed
FieldSelector	Specifies whether and how a field should be stored
FieldSelector	Specifies whether and how a field should have term vectors
FieldSelector	Provides information about what should be done with the field
FieldSelector	Level for FieldSelector
FieldSelector	A FieldSelector based on a Map of field names to FieldSelector
FieldSelector	Provides support for converting long to strings, and back again
FieldSelector	Defines what fields to load normally and what fields to load lazily

org.apache.lucene.search (Lucene nightly API) - Mozilla Firefox

Package org.apache.lucene.search

Table Of Contents

See: [Description](#)

Interface Summary

FieldSelector	Request Maximum number of term values
FieldSelector	Defines to parse from the document fields
FieldSelector	Defines to parse from the document fields
FieldSelector	Request CompositeDoc objects for scoring
FieldSelector	The interface for search implementations
FieldSelector	Request return a comparator for sorting DocValues
FieldSelector	Request Calculate query weight and build query object

Class Summary

BooleanDocValues	A class in a BooleanDocValues
BooleanDocValues	Specifies how clauses are to score in matching documents
BooleanDocValues	A Query that matches documents matching boolean combinations of other queries, e.g.
BooleanDocValues	Wraps another filter's result and caches it
BooleanDocValues	Request Describe the score computation for document and query, including a match independent of a relative value
BooleanDocValues	A query that wraps a filter and simply returns a constant score equal to the query boost for every document in the filter
BooleanDocValues	A query query that returns a constant score equal to its boost for all documents in the range
BooleanDocValues	Request Default scoring implementation.
BooleanDocValues	A query that generates the scores of documents provided by its sub-queries, and that scores each document with the maximum score for that document as produced by any sub-query, plus a boosting term for any additional matching sub-queries
BooleanDocValues	Request Describe the score computation for document and query