Apache Lucene - Basic Demo Sources Walk-through

Andrew C. Oliver

Table of contents

1. 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 wishing to understand how to use Lucene in their applications.

2. 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 directory called `demo`. This is the root for all of the Lucene demos. Under this directory is `org/apache/lucene/demo`. This is where all the Java sources for the demos live.

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

3. 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 `main` function does is instantiate IndexWriter. It passes the string "`index`" and a new instance of a class called StandardAnalyzer. The "`index`" string is the name of the filesystem 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 indices. To use it you must instantiate it with a path that it can write the index into. If this path does not exist it will first create it. Otherwise it will refresh the index at that path. You can also create an index using one of the subclasses of Directory. In any case, you must also pass an instance of org.apache.lucene.analysis.Analyzer.

The particular Analyzer we are using, StandardAnalyzer, is little more than a standard Java Tokenizer, converting all strings to lowercase and filtering out stop words and characters from the index. By stop words and characters I mean common language words such as articles (a, an, the, etc.) and other strings that may have less value for searching (e.g. 's) . It should be noted that there are different rules for 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

contrib/analyzers/src/java/org/apache/lucene/analysis).

Looking further down in the file, you should see the `indexDocs()` code. This recursive function simply crawls the directories and uses FileDocument to create Document objects. The Document is simply a data object to represent the content in the file as well as its creation time and location. These instances are added to the `indexWriter`. Take a look inside FileDocument. It's not particularly complicated. It just adds fields to the Document.

As you can see there isn't much to creating an index. The devil is in the details. You may also wish to examine the other samples in this directory, particularly the IndexHTML class. It is a bit more complex but builds upon this example.

4. Searching Files

The SearchFiles class is quite simple. It primarily collaborates with an IndexSearcher, StandardAnalyzer (which is used in the IndexFiles class as well) and a QueryParser. The query parser is constructed with an analyzer used to interpret your query text in the same way the documents are interpreted: finding the end of words and removing useless words like 'a', 'an' and 'the'. The Query object contains the results from the QueryParser which is passed to the searcher. Note that it's also possible to programmatically construct a rich Query object without using the query parser. The query parser just enables decoding the Lucene query syntax into the corresponding Query object. Search can be executed in two different ways:

- Streaming: A Collector subclass simply prints out the document ID and score for each matching document.
- Paging: Using a TopScoreDocCollector the search results are printed in pages, sorted by score (i. e. relevance).

5. The Web example...

read on>>>