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 `lucene/contrib/demo/`. This is the root for the Lucene demo. Under this directory is `src/java/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 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 `main()` method parses the command-line parameters, then in preparation for instantiating IndexWriter, opens a Directory and instantiates StandardAnalyzer and IndexWriterConfig.

The value of the `-index` command-line parameter is the name of the filesystem directory where all index information should be stored. If `IndexFiles` is invoked with a relative path given in the `-index` command-line parameter, or if the `-index` command-line parameter is not given, causing the default relative index path "`index`" to be used, the index path will be created as a subdirectory of the current working directory (if it does not already exist). On some platforms, the index path may be created in a different directory (such as the user's home directory).

The `-docs` command-line parameter value is the location of the directory containing files to be indexed.

The `-update` command-line parameter tells `IndexFiles` not to delete the index if it already exists. When `-update` is not given, `IndexFiles` will first wipe the slate clean before indexing any documents.

Lucene Directorys are used by the `IndexWriter` to store information in the index. In addition to the FSDirectory implementation we are using, there are several other

`Directory` subclasses that can write to RAM, to databases, etc.

Lucene Analyzers are processing pipelines that break up text into indexed tokens, a.k.a. terms, and optionally perform other operations on these tokens, e.g. downcasing, synonym insertion, filtering out unwanted tokens, etc. The `Analyzer` we are using is `StandardAnalyzer`, which creates tokens using the Word Break rules from the Unicode Text Segmentation algorithm specified in Unicode Standard Annex #29; converts tokens to lowercase; and then filters out stopwords. Stopwords are common language words such as articles (a, an, the, etc.) and other tokens that may have less value for searching. 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 modules/analysis/common/src/java/org/apache/lucene/analysis).

The `IndexWriterConfig` instance holds all configuration for `IndexWriter`. For example, we set the `OpenMode` to use here based on the value of the `-update` command-line parameter.

Looking further down in the file, after `IndexWriter` is instantiated, you should see the `indexDocs()` code. This recursive function crawls the directories and creates Document objects. The `Document` is simply a data object to represent the text content from the file as well as its creation time and location. These instances are added to the `IndexWriter`. If the `-update` command-line parameter is given, the `IndexWriter` `OpenMode` will be set to `OpenMode.CREATE_OR_APPEND`, and rather than adding documents to the index, the `IndexWriter` will **update** them in the index by attempting to find an already-indexed document with the same identifier (in our case, the file path serves as the identifier); deleting it from the index if it exists; and then adding the new document to the index.

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 word boundaries, downcasing, 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 the `IndexSearcher.search(query,n)` method that returns TopDocs with max `n` hits, the search results are printed in pages, sorted by score (i.e. relevance).