Jakarta Lucene keresőmotor

Viczián István (vicziani.istvan a gmail-en)

Ez a cikk azon gyakorlott Java programozóknak szól, akiknek szükségük van szövegben való keresési lehetőségre, akár különálló programban, akár egy teljes webes alkalmazásban. A cikk röviden bemutatja a Lucene könyvtárat, mely egy nagyteljesítményű, minden alkalmazási területet lefedő, Java nyelven implementált ingyenes, nyílt forráskódú *keresőmotor* (Apache Software Licence alatt).

Az eredeti cikk megírásának időpontja 2003. május 10., de azóta több minden változott, ezért 2009. január 28-án frissítésre került sor. A második cikk megírása pillanatában az aktuálisan elérhető legújabb verzió a Lucene 2.4.0, mely 2008. október 8-án jelent meg.

A Lucene egy Top-Level projekt az Apache keretein belül (2001-ben csatlakozott a Jakarta projekthez, majd kivált, és Top-Level projekt lett), ennek csak egyik alprojektje a Java-ban implementált Lucene Java keresőmotor, melynek magját Doug Cutting írta, de mára többen csatlakoztak a fejlesztéshez. A Top-Level projekt része a Nutch, mely a Lucene-re épülő web kereső, mely tartalmaz egy crawler-t, mely letölti, indexeli az oldalakat, és követi linkeket, egy link adatbázist, valamint képes HTML és egyéb formátumok elemzésére. Része a Solr, mely egy Lucene-re épülő kereső szerver, különböző API-kkal, cache-eléssel, replikálással és webes adminisztrációs felülettel. A Tika különböző dokumentum formátumú állományoknak próbálja kitalálni a típusát, kinyerni különböző meta-adatokat és magát a szöveges tartalmat. A Mahout gépi tanulást biztosító algoritmusok implementációja. A gépi tanulás a mesterséges intelligencia egyik területe, és célja az adatok elemzésével, tapasztalati úton teljesítménynövelést elérni. Főbb alkalmazási területei a keresés, mintafelismerés, statisztika, adatbányászat, stb. Ezen kívül léteznek más programozási nyelven implementált megvalósítások is, mint a C nyelvű Lucy, melynek létezik Perl és Ruby kapcsolata is, a Python-ban használható PyLucence, valamint a Lucene.NET .NET keretrendszerre, C#-ban implementálva.

1. ábra Lucene logó

A Lucene keresőmotor alapvetően *dokumentumokat* kezel (`Document` osztály), melyek egy részhalmazát kell tudni kiválasztani egy keresési feltétel alapján. A dokumentumot *mezőkre* (`Field` osztály) bontja, melyek értékei származhatnak `String`-ből, vagy bármilyen `Reader` objektumból. A keresésként kapott dokumentumokat rangsorolja (ranking), mely megadja, hogy a keresési feltételnek mennyire felel meg az adott dokumentum. A keresést az index gyorsítja, ami tulajdonképpen egy adatbázis (fájlban, memóriában, vagy akár relációs adatbázisban tárolva), melyben az előfeldolgozott információk találhatók a dokumentumokra vonatkozóan. Az index formátuma teljesen nyílt, és bit szinten dokumentált, ezért akár bármilyen programozási nyelven fel lehet dolgozni. A dokumentum mezőire külön megadható, hogy indexelődjenek-e vagy sem. Az indexet az `IndexWriter` osztállyal lehet írni (dokumentumokat létrehozni és az indexhez adni), valamint `IndexReader` osztállyal olvasni. Az indexelést *feldolgozók* (Analyzer osztály) végzik, melyből több, eltérő funkcionalitású is rendelkezésre áll. Ezek `Token`-ek halmazát állítják elő az indexelendő

Reader-ből, méghozzá Tokenizer kimenetére illesztett TokenFilter-ekkel megszűrve. A Term a Lucene indexelés alapegysége, nyugati nyelveknél ez a fogalom általában megegyezik a szó fogalmával. Termek sorozata a TermEnum, egy term előfordulását egy dokumentumban a TermDocs interfész ír le, és a dokumentumban található összes termet a TermFreqVector. A *keresési feltétel* lehet egy szó (TermQuery), kifejezés (PhraseQuery) vagy akár ezekkel megadott bonyolult logikai feltétel. A szövegként megadott keresési kifejezést egy JavaCC-vel generált parser (QueryParser) képes értelmezni. A keresést Searcher osztály leszármazottjának példánya végzi, melynek search metódusa kapja meg a keresési feltételt, és sok egyéb paramétert. Az IndexSearcher osztály a Searcher osztály leszármazottja, mely az IndexReader-re épül. Megjegyzendő, hogyha nyitva tartunk egy IndexReader-t, akkor az indexhez felvett új dokumentumok, valamint törlési műveletek eredménye addig nem látszik, amíg az IndexReader-t újra meg nem nyitjuk. A keresésnek meg lehet adni Filter leszármazottat, mely a találatokat szűrheti (ebből több implementáció is rendelkezésre áll – sőt láncba is rendezhetőek), valamint Sort objektumot a találatok sorba rendezéséhez. Az indexben tárolt adatok perzisztálására a Directory leszármazottak valók, melyeket az IndexOutput osztály ír, és az IndexInput osztály olvas. A „Core” csomagban az FSDirectory található, mely az indexet fájlrendszerben tárolja (, egy NIO alapú megvalósítás, a NIOFSDirectory), valamint a RAMDirectory, mely a memóriában. A Lucene keresés régebben Hits osztály példányát adta vissza, de ez az osztály a 2.4.0-ás verzióban deprecate-d lett, jelezve, hogy a 3-as verzióban el fog tűnni. Helyette a TopDocs osztály használható, melynek scoreDocs attribútuma adja vissza a talált dokumentumok tömbjét (ScoreDoc[]), amelyben minden ScoreDoc elem tartalmazza a dokumentum sorszámát, és a pontszámot. A keresésnek át lehet adni, egy HitCollector implementációt is, mely a találatokat szűri. Pl. a TopDocCollector csak az első paraméterben meghatározott számú, legmagasabb pontszámú találatokat adja vissza, a TimeLimitedCollector pedig egy olyan megvalósítás, mely kivételt vált ki, ha a keresés nem tér vissza a paraméterként megadott időn belül. A talált dokumentumot aztán a Searcher osztálytól kell elkérni sorszám alapján, a ScoreDoc.doc attribútum alapján. Természetesen ezen fogalmak később forráskódon lesznek bemutatva.

A Lucene UNICODE kompatibilis könyvtár, azaz lehetőség van bármely nyelven írt dokumentum megfelelő indexelésére. A Lucene-nek magyar vonatkozása is van, a QueryParser osztály setDefaultOperator metódusának dokumentációjában szerepel a Hungary szó.

A Lucene index nem egy relációs adatbázis, tehát a dokumentumokban szereplő számok, dátumok, intervallumok kezelése is eltérő az ott megismerttől, de itt is megadható, melyet a Lucene wiki oldalai részletesen leírnak. Lehetőség van pl. per jelekkel elválasztott hierarchia szerinti keresésre is (pl. dmoz.org a Java programozási nyelvet a következő kategóriába sorolja: Top/Computers/Programming/Languages/Java).

A Lucene lehetőséget biztosít éles rendszer futása közben is az index mentésére, párhuzamos indexelésre és keresésre, valamint több index alapján történő keresésre, és a találatok összefésülésére.

A Lucene semmilyen más könyvtárakra nem tartalmaz hivatkozást, elegendő a JAR állományt (jelenlegi verzióban ennek neve lucene-core-2.4.0.jar) a CLASSPATH környezeti változóban elhelyezni.

A példában e-mail üzenetek indexelését mutatom be, ahol az indexelő metódus egy `javax.mail.Message` tömböt kap paraméterül. A `Message` osztály a JavaMail API specifikáció része, melyet a JSR 919 specifikál. A cikk írásának pillanatában a legutolsó verzió az 1.4.1-es, mely megköveteli a JavaBeans Activation Framework 1.1.1 meglétét, ami már a Java SE 6 része.

A linkek között található egy példa, hogy hogyan lehet Java forráskódot indexelni.

Ahhoz, hogy a Lucene-t használjuk, a következő importhivatkozásokat kell alkalmazni (a fejlesztőeszköz Organize imports funkciójával ezek konkrét interfészek és osztályok importjaira cserélhetőek):

```
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.document.*;
import org.apache.lucene.index.*;
import org.apache.lucene.queryParser.*;
import org.apache.lucene.search.*;
import org.apache.lucene.store.*;
```

A következő metódus indexeli a `Message` tömböt. A metódus hiányos, a kivételkezelést az olvasó fantáziájára bízom.

```
public void indexMessages(Message[] messages) {
...
Directory directory = new RAMDirectory();
// Directory directory = FSDirectory.getDirectory("/tmp/testindex");
Analyzer analyzer = new StandardAnalyzer();
IndexWriter writer = new IndexWriter(directory, analyzer,
MaxFieldLength.UNLIMITED);
for (Message message : messages) {
    Document document = new Document();
        document.add(new Field("fromAddress", ((InternetAddress)
message.getFrom()[0]).getAddress(), Field.Store.YES,
Field.Index.NOT_ANALYZED));
        document.add(new Field("fromPersonal", ((InternetAddress)
message.getFrom()[0]).getPersonal(), Field.Store.YES,
Field.Index.NOT_ANALYZED));
        document.add(new Field("sentDate",
DateTools.dateToString(message.getSentDate(), DateTools.Resolution.MINUTE),
Field.Store.YES, Field.Index.NOT_ANALYZED));
        document.add(new Field("subject", message.getSubject(),
Field.Store.YES, Field.Index.ANALYZED));
        document.add(new Field("size", Integer.toString(message.getSize()),
Field.Store.YES, Field.Index.NO));
        if (message.getContent()instanceof String) {
            document.add(new Field("content", (String)
message.getContent(), Field.Store.NO, Field.Index.ANALYZED));
        }
        writer.addDocument(document);
}
writer.close();
...
}
```

A kódrészlet először létrehoz egy `Directory`-t, mely lehet `RAMDirectory`, ami az indexet a memóriában tárolja (remekül használható teszt esetek futtatásához), vagy használható az `FSDirectory`, mely az indexet fájlrendszerbe tárolja. Majd példányosítja a beépített szabványos feldolgozót (`Analyzer`), létrehoz egy `IndexWriter` objektumot, melynek paraméterül átadja a `Directory` példányt, az `Analyzer`-t, és definiálja, hogy

nincs korlát az indexelendő adat méretére (, ide bájtban megadhatunk egy méretkorlátot is). Majd végigiterál az üzeneteken, és mindegyik üzenethez létrehoz egy üres dokumentumot (`Document`). A dokumentumhoz hozzáadja a következő mezőket. A tárolt mezőt kereséskor le lehet kérni az eredeti formában a találati listából. Az indexelt mezőre lehet keresni.

Mező neve	**Leírása**	**Tárolt**	**Indexelt**
fromAddress	Az első feladó e-mail címe (nem foglalkozunk a többi címmel)	Igen	Nem tokenizált
fromPersonal	Az első feladó viselt neve (nem foglalkozunk a többi névvel)	Igen	Nem tokenizált
sentDate	Küldés ideje, szöveggé konvertálva	Igen	Nem tokenizált
subject	Tárgy	Igen	Tokenizált
size	Az üzenet mérete	Igen	Nem indexelt
content	Az üzenet tartalma (csak akkor dolgozzuk fel, ha szimpla szöveges)	Nem	Tokenizált
A többi `Message` property-t nem dolgozzuk fel			

1. táblázat Mező tulajdonságok

A dokumentumot az `IndexWriter addDocument` metódusának kell átadni, mely feldolgozza a dokumentumot.

Jelenleg egy beépített elemzőt használtunk `StandardAnalyzer` névvel, de válogathatunk a több közül (pl. `Analyzer` leszármazottak különböző nyelvekhez, `SimpleAnalyzer`, `StopAnalyzer`, `WhitespaceAnalyzer`, `KeywordAnalyzer`, `PerFieldAnalyzerWrapper`), vagy akár sajátot is implementálhatunk, melynek az `Analyzer` osztályból kell származnia.

A `Field` osztálynak konstruktorban a következő paramétereket kell megadni. Első paraméter a mező neve, mely egy `String`. A második paraméter a feldolgozandó szöveg, mely lehet `String`, `Reader` vagy `byte[]` típusú is. Harmadik paraméter határozza meg, hogy kell-e tárolni a mező értékét az indexben (, lehetséges értékei: `Field.Store.COMRESS`, `Field.Store.NO`, `Field.Store.YES`). A negyedik paraméter határozza meg, hogy kell-e indexelni az adott értéket. Értékei a `Field.Index.ANALYZED`, ami azt jelenti, hogy tokenizálva kell indexelni, a `Field.Index.NO`, ami nem indexeli, a `Field.Index.NOT_ANALYZED`, ami indexeli, de nem tokenizálva. Ezen kívül lehet `Field.Index.ANALYZED_NO_NORMS` és `Field.Index.NOT_ANALYZED_NO_NORMS`, mely annyi különbség az előbbiekhez képest, hogy nem normalizál tároláskor, ez kereséskor kevesebb memóriafoglalást eredményez.

Látható a példában, hogy a feladó e-mail címe és neve ugyan indexelt, de nem kell tovább bontani, a tárgyat analizálni kell, és úgy indexelni, és tárolni is. A méretet el kell ugyan tárolni, de nem kell indexelni, hiszen arra nincs értelme keresni. Az üzenet tartalmát csak akkor dolgozzuk fel, ha szöveges, és ekkor analizálva indexeljük ugyan, de nem tároljuk el. A dátum típusú mezőt szöveggé kell konvertálni a `DateTools` osztály statikus `dateToString` metódusával, melynek paraméterként meg kell adni, hogy milyen részletességgel kell a dátumot tárolni.

Gyakran szükséges, hogy a keresés után visszakeressük az eredeti objektumot az adatbázisból, és nem az indexből, ilyenkor érdemes eltárolnunk az objektum egyedi azonosítóját is a dokumentumban, ami szintén nem kereshető.

```
aDocument.add(new Field("id", Long.toString(id), Field.Store.YES, Field.Index.NO));
```

A keresésre a következő metódus szolgál, mely hiányos, a kivételkezelés itt sem látható.

```
public void searchMessages(Directory dir, String query) {

    ...
    IndexSearcher s = new IndexSearcher(dir);
    Query q = new QueryParser("content", new
StandardAnalyzer()).parse(query);
    TopFieldDocs docs = s.search(q, null, 100, new Sort("sentDate", true));
    for (ScoreDoc scoreDoc : docs.scoreDocs) {
        Document doc = s.doc(scoreDoc.doc);
        System.out.println("Subject: " + doc.get("subject"));
    }
    ...
}
```

Először egy `IndexSearcher` objektumot kell példányosítani, melyek konstruktorban egy `Directory`-t kell átadni. Majd egy `Query`-t kell létrehozni a szöveges keresési feltétel elemzésével (ez dobhat `ParseException` kivételt), a `QueryParser` osztállyal lehetséges. Ennek meg kell adni a keresési feltétel szöveges ábrázolását, az alapértelmezett mezőt és egy feldolgozót. Vigyázni kell, ha nem ugyanabba az osztályba tartozó feldolgozót használjuk indexelésnél és keresésnél, ugyanis akkor a keresés sikertelen lehet. Megjegyzendő, hogy a `QueryParser` osztály nem szál biztos.

A `Query`-t összeállíthatjuk a `Query` osztály leszármazottjaiból, mint a `BooleanQuery`, `WildcardQuery`, `TermQuery`, `PhraseQuery`, `RangeQuery`, stb.

A keresésnek paraméternek meg lehet adni a `Query`-t, `Filter`-t, mely a találatokat képes szűrni, maximális találati számot, valamint mezőket, mely szerint rendezni kell (, akár fordított sorrendben) a találatokat. A mezőket, melyek szerint rendezni lehet, indexelni kell, de nem szabad tokenizálni.

A keresést futtatva egy `TopFieldDocs` objektumot kapunk vissza, amitől le lehet kérdezni a találatokat reprezentáló `ScoreDoc` objektumokat a dokumentum azonosítójával és pontszámával.

A dokumentumot az `IndexSearcher`-től lehet elkérni annak azonosítója alapján, és a dokumentum egy mezőjét pedig a `Document.get(String fieldName)` metódussal.

Abban az esetben, ha egy dokumentumot törölni akarunk, akkor használjuk az `IndexReader.delete(int docNum)` metódust, melynek a dokumentum sorszámát kell átadni. Törölni term alapján is lehet. Dokumentumot módosítani nem lehet, törölni kell, majd újra hozzáadni.

A keresési feltétel megadásakor megkülönböztetünk termeket (általában szó, pl. spam) és kifejezéseket (szókapcsolatok, pl. „Java lista”). A keresési feltételben megadhatjuk, hogy mely mezőben szeretnénk keresni, ekkor kettősponttal kell minősíteni (pl. subject:spam). Megengedettek a joker-karakterek is, melyekkel egy vagy több karaktert lehet helyettesíteni („?” és „*”). Fuzzy keresésre is lehetőség van a tilde („~”) karakter használatával. Meg lehet adni, hogy több szó esetén, a szavak maximum milyen távolságban helyezkedhetnek el egymástól („Java lista”~10). Lehetőség van intervallum megadására is, ekkor a rendezési elv

a névsorrend (pl. fromAddress:[a@gmail.com TO c@gmail.com]. A szavakat és szókapcsolatokat súlyokkal láthatjuk el (pl. Java^4 lista). Logikai operátorként használható az OR, AND NOT, illetve a „+” jel jelenti, hogy a szónak mindenképp szerepelnie kell egy mezőben, a „-„ jel, hogy nem szerepelhet. Lehetőség van csoportosításra is a kerek zárójelekkel, és escape szekvenciák is alkalmazhatóak speciális karakterekre.

Az `IndexWriter.optimize()` metódushívás optimalizálja az adatbázist. A Lucene tartalmaz egy egyszerű cache megoldást is `Filter`-ek alkalmazásával (`CachingWrapperFilter`, `QueryFilter`).

A Lucene weboldalán több benchmark-ot is találunk, különböző szoftver és hardver környezetben mért sebesség eredményekkel, melyek alapján a teljesítményét meg lehet állapítani, és esetleg hardvert is tervezhetünk.

Mint ahogy a legtöbb Jakarta project keretében fejlesztett projektről elmondható, a Lucene esetén is elérhető a teljes forráskód, részletes dokumentáció (JavaDoc), példaprogramok és tesztesetek is.

A Lucene áll az un. „Core” interfészekből és osztályokból, a „Demo” interfészekből és osztályokból, melyben több példa is található, és a „Contrib” csomagokból, melyek kiegészítő interfészeket és osztályokat tartalmaznak a következő témakörökben:

- Különböző (német, francia, orosz, kínai, japán, stb.) elemző osztályok, magyar nem található köztük
- Ant taszk
- Berkeley DB-ben tárolt index megvalósítás
- A benchmark-hoz szükséges interfészek és osztályok
- Highlighter a keresett szavaknak a talált szövegben való kiemelésére
- Parancssoros interfész
- Speciális keresési lehetőségek (pl. ehhez hasonló találatok – `MoreLikeThis`, Regexp, stb.)
- Snowball stemmer: szótövesítést végző modul (ebből viszont van magyar)
- Helyesírás ellenőrző
- Keresést támogató Swing-es komponensek (`JTable` és `JList` modell)
- Wikipedia tokenizer
- Wordnet által definiált szinonimák használata
- XML formátumú keresések kezelése
- stb.

A Lucene-nel kapcsolatban mindenképpen érdemes megemlíteni a Luke eszközt, mely egy Swing-es alkalmazás Lucene indexek kezelésére, megtekintésére. Lehet vele az indexelt dokumentumokat kezelni, keresést futtatni, dokumentumokat törölni, optimalizálni, stb. Plugin-elhető és script-ezhető, Java Web Start-tal indítható.

A Lucene web oldalán több eszköz is található, mely valamilyen kapcsolatban áll vele.

Budapest, 2008. január 28.

Hivatkozások

The Apache Software Foundation http://www.apache.org
Lucene http://lucene.apache.org/
Luke – Lucene Index Toolbox http://www.getopt.org/luke/
JavaMail - http://java.sun.com/products/javamail/
Using Lucene to Search Java Source Code - http://www.onjava.com/pub/a/onjava/2006/01/18/using-lucene-to-search-java-source.html

Szerzőről

Viczián István a Debreceni Egyetem programtervező matematikus szakán végzett 2001 nyarán. Jelenleg szoftver fejlesztőként dolgozik Budapesten, melynek keretében Java SE-vel, webes technológiákkal, Java EE-vel, alkalmazásintegrációval és SOA-val foglalkozik. Szabadidejében túrázik, geocache-el és sportol, valamint Java blog-ot ír (JTechLog).
E-mail: viczian.istvan a gmail-en
Honlap: http://delfin.unideb.hu/~vicziani/
Blog: http://jtechlog.blogspot.com/