RIM - Recherche d'Information Multimedia

Labo 4

Implémenation de Apache Lucene

Librairie de recherche écrite en Java

Romain de Wolff et Simon Hintermann
IL2008

Professeurs Mme. Laura Elena Raileanu
et Mme Nastaran Fatemi

Lausanne, le 5 décembre 2007

1 Objectifs

Dans ce laboratoire nous allons découvrir l'outil d'indexation et de recherche Open-Source *Apache Lucene* et l'intégrer dans notre framework de travail, à savoir, **RIM**.

2 Démarches adoptées

La première étape effectuée pour commencer, est de lire la documentation d'introduction concernant ces librairies. *Apache Lucene* met à disposition tout une série de classes qui permettent d'indexer des document et d'effectuer des recherches. Lire la documentation est le meilleur moyen de se plonger de la structure.

Après s'être un peu mieux imprégné de la structure de Lucene, nous reprenons notre code du laboratoire précédant. Nous supprimons la grande majorité des éléments codés dans les laboratoire précédant. Les fichiers à modifier sont `CACMIndexer.java` et `CACAMRetriever.java`. Ce sont les document principaux dans lesquelles nous allons travailler.

Nous n'utilisons pas les "stop word" mis à disposition mais directement ceux disponibles dans la librairie de Lucene.

Une fois les classes modifiées nous effectuons des tests pour voir si les résultats semble correctes. Nous allons aussi comparer les résultats avec nos collègues afin de voir si nous avons des résultats similaire, voir identique.

3 Réponses aux questions

Voici les réponses aux questions posées durant le laboratoire.

3.1 Qu'est-ce que Lucene ? Est-ce destiné à Java uniquement ?

Apache Lucene est un moteur de recherche de texte écrit completement en Java. C'est une technologie qui est utilisable avec n'importe quelle application et sur n'importe quelle plateforme. C'est aussi un projet Open Source. Il est destiné pour Java mais d'autres versions existent, comme en .NET ou en C.

3.2 Dans Lucene, à quoi sert un Analyzer ? Peut-on écrire son propre Analyzer ?

Un "Analyzer" crée des "TokenStreams" en parcourant le texte. Il permet de découper un fichier source en token. C'est dans l'analyzer que l'on va spécifier les stop words. Ainsi, nous

pouvons tout à fait créer un analyzer en français, ou en fonction de nos données à indexer. A noter qu'il existe déjà beaucoup d'analyzer existant dans différentes langues disponibles, comme en anglais, français ou encore allemand.

3.3 Comment un document est-il représenté dans Lucene ?

Voyons comment créer un document. Pour ce faire, nous utilisons la classe `Document`. La méthode `add()` permet d'ajouter un document au corpus des documents. Nous pouvons ajouter autant de champs dans un document que désiré. Nous allons utiliser deux champs, à savoir `id` et `content`.

Il y a donc une classe qui permet de créer des documents personnalisés. Ces documents seront par la suite indexé par Lucene. Les fichiers d'index sont stocké dans des fichiers binaires ou dans la mémoire.

3.4 Quelles sont les possibilités offertes par Lucene pour la formulation d'une requête ?

Les possibilités offertes par Lucene sont nombreuses. Il est même possible de définir son propre système de requête.

La recherche booléenne est possible ainsi que des options sur les champs que nous désirons inclure dans la recherche.

La syntaxe de recherche est très complète. Citons les possibilités suivantes :

- termes : simple chaîne de caractères
- champs : spécifier les champs où l'on désire rechercher l'information
- caractères spéciaux : l'étoile (*) ou le point d'interrogation (?) permettre d'augmenter les critères de recherches
- similarité : en utilisant le caractère `~` on peut effectuer des recherches approximatives
- intervalle : on peut par exemple spécifier un intervalle de temps en utilisant deux dates
- booster un terme : on peut augmenter la pondération d'un terme a l'aide de `^`

La syntaxe complète est décrite à l'adresse suivante : `http://lucene.apache.org/java/docs/queryparsersyntax.html`.

3.5 Expliquez brièvement comment Lucene gère ses index (en termes de fichiers) et expliquez l'intérêt de cette façon de procéder

Lucene enregistre ses index dans un dossier ou dans la mémoire. Les deux commandes utilisées sont les suivantes :

```
// sauvegarde des index dans un répértoire
Directory dir = FSDirectory.getDirectory("chemin/de/sauvegarde");

// sauvegarde des index dans la mémoire
Directory directory = new RAMDirectory();
```

En sauvegardant les index sur le disque dur, on peut voir leur structure. Ces fichiers sont des fichiers binaires.

En fait, *Lucene* utilise la notion de `segment`. Les index peuvent comporter plusieurs segments qui sont en faîte des sous-index. Chaque segment est indépendant, ce qui permet d'augmenter les performances dans plusieurs cas, notamment lors de l'indexation (modification que de certains segments).

4 Analyse des résultats obtenus

Analysons une requête ensemble pour voir comment ce déroule les recherche avec *Lucene*. Ci-dessous une recherche sur les mots clé déjà utilisé dans le laboratoire précédant : `Automatic Data Processing`.

```
Entrer le(s) terme(s) recherche:
Automatic Data Processing
Documents trouve ainsi que similarite par cosinus:
Nombre de resultat(s): 150
Document no : 987 (cosinus = 0.9999999403953552)
Document no : 86 (cosinus = 0.8808058500289917)
Document no : 59 (cosinus = 0.8808057904243469)
Document no : 84 (cosinus = 0.7707051038742065)
Document no : 528 (cosinus = 0.656242311000824)
Document no : 696 (cosinus = 0.6354188323020935)
Document no : 1424 (cosinus = 0.6249999403953552)
Document no : 1457 (cosinus = 0.5992525219917297)
Document no : 617 (cosinus = 0.550503671169281)
Document no : 1252 (cosinus = 0.5208895802497864)
Document no : 597 (cosinus = 0.4921817183494568)
Document no : 1272 (cosinus = 0.4353649616241455)
Document no : 1769 (cosinus = 0.3749999701976776)
Document no : 675 (cosinus = 0.3520166575908661)
Document no : 2999 (cosinus = 0.33250659704208374)
Document no : 1012 (cosinus = 0.3255559504032135)
Document no : 1516 (cosinus = 0.3110170066356659)
Document no : 1588 (cosinus = 0.3088557720184326)
```

```
Document no : 2307 (cosinus = 0.3056703507900238)
Document no : 1359 (cosinus = 0.30548951029777527)
Document no : 414 (cosinus = 0.2900208830833435)
Document no : 2727 (cosinus = 0.2882053256034851)
Document no : 2718 (cosinus = 0.2658224105834961)
Document no : 855 (cosinus = 0.264616996049881)
Document no : 273 (cosinus = 0.25908273458480835)
Document no : 2217 (cosinus = 0.25715601444244385)
Document no : 2286 (cosinus = 0.2513396739959717)
Document no : 1719 (cosinus = 0.24864108860492706)
Document no : 1570 (cosinus = 0.23427890241146088)
Document no : 3031 (cosinus = 0.23259462416172028)
Document no : 1226 (cosinus = 0.23220109939575195)
Document no : 1465 (cosinus = 0.2320166677236557)
Document no : 1195 (cosinus = 0.22969068586826324)
[...]
```

La première chose intéressante a remarquer c'est que les résultats semble bien étalé entre 0.00 et 0.99. On ne voit pas ici pour éviter de mettre 4 pages de résultats, mais les valeurs sont bien étalées.

On remarque aussi que les résultats sont différents de ceux obtenus avec notre propre algorithme (cf. laboratoire précédant). Ainsi le document ayant pour ID le numéro 86 apparaissait plus tard.

On en déduit donc que *Lucene* a un algorithme implémenté de manière légèrement différente.

5 Conclusion personnelle

L'utilisation des classes définies par *Lucene* nous permet de très facilement indexer des documents et effectuer des recherches sur un corpus. Les méthodes mises à dispositions permettent une grande souplesse et il est intéressant de voir que l'implémentation d'un tel outil se fait sans difficultés majeur.

L'implémentation s'est déroulée sans problème.