RECHERCHE DOCUMENTAIRE

Rémi Forax, Sylvain Cherrier, Stéphane Lohier
forax@univ-mlv.fr, sylvain.cherrier@univ-mlv.fr, lohier@univ-mlv.fr

Il s'agit d'écrire, de façon modulaire :
- un outil d'indexation des fichiers d'une arborescence donnée
- un outil d'export de cet index dans un format normalisé
- un outil de statistiques sur cet index
- un outil de recherche de document

Un des buts importants d'un outil de gestion de connaissances est de permettre d'accéder de manière uniforme à des ressources variées.

En particulier, la possibilité d'effectuer des recherches sur des documents de formats différents est essentielle. Pour cela, une phase d'indexation permet d'associer des informations standards (liste de mots-clés, langue, ...) à chaque document, quelque soit son format (html, xml, texte simple, pdf, ...) et son origine géographique (fichier local, réseau local, web)

INDEXATION

La classe contenant le main s'appellera fr.umlv.indexer.Indexer

Pour indexer, on lancera simplement :

```
$ java -jar indexer.jar --build <racine de l'arborescence>
```

L'index sera stocké dans le sous-répertoire *.indexer* du répertoire indiqué par la variable d'environnement INDEXER_HOME (par défaut, HOME de l'utilisateur si la variable d'environement n'est pas définie).
Cet index pourra être constitué de un ou plusieurs fichiers (à vous de choisir).

Cette première phase va permettre d'extraire les informations suivantes de chaque document:
- liste de mots-clés avec le nombre d'occurrences de chacun de ces mots dans le document
- la langue
- taille en nombre de mots et nombre de caractères

Les documents correspondent à l'ensemble des fichiers dans la sous-arborescence spécifier par la racine de l'arborescence. Cette racine peut être indiquer sous la forme d'un chemin absolu ou relatif.
L'index devra toujours stocké les chemins vers les fichiers de façon absolue.

Les documents supportés seront reconnus par leur extension. Il vous est demandé de savoir traiter les :
- document TEXTE (supposé en ASCII ou en ISO-8859-1) : extension *.txt*
- document XML : extension *.xml*
- document ZIP : extension . *zip* ces fichiers seront traités comme des répertoires
  ils peuvent donc inclure eux même des document texte, xml et zip.

Optionnellement, vous pouvez supporter tout autre type de document.

Dans tous les cas, vous devez prévoir qu'on puisse ajouter de nouveaux types de documents et vous devrez décrire dans la doc développeur comment faire.

Optionnellement, si le parsing du document en fonction de l'extension échoue, vous devez essayer de trouver une solution de replie par exmple en considérant que c'est un texte simple.

Les mots-clés

- pour les fichiers XML, on prendra tous les textes inter-balise
- pour les fichiers ASCII divers.
- pour les fichiers inclus dans les ZIP, on incluera simplement le nom du zip dans le path complet du document. Attention aux ZIP dans les ZIP.

On prendra tous les mots (composés uniquement de lettres [a-zA-Z]) de 3 lettres ou plus. En particulier, les ' et – ne seront pas pris en compte (pas de mots-composés ni de l') ni les chiffres.

Appauvrissement typographique

Dans tous les cas, les lettres suivantes avec accent [é è ê à â ô û ù] seront assimilées à la même lettre sans accent. Idem pour le [ç] assimilé au [c].
Cette liste n'est pas exhaustive, vous pouvez optionellement la compléter.
Dans tous les cas, on se rendra insensible a la casse en sauvant tous ces mots-clés en minuscules.

La langue

- la détermination sera basique : on considérera le texte en français si un des caractères suivants est présent : é è ê à â ô û ù

sinon ça sera de l'anglais.

Les tailles

on comptabilise uniquement (pour chaque fichier) :

- le nombre de mots clés différents et le nombre total d'occurrences
- la taille (en octets) du fichier

RECHERCHE

Ce module permet d'effectuer des recherches sur un index précédemment créé.

Attention, le format décrit ici devra être scrupuleusement respecté.

Pour lancer le programme :

```
$ java -jar indexer.jar --search [chemin vers une arborescence]
--request "<requête>"
```

Le chemin vers une arborescence est optionnel.

Note : la requête doit être donné comme un argument unique (avec des " s'il y a plusieurs mots):

```
$ java -jar indexer.jar --search /var/spool/mail --request
"reseau informatique"
```

Ce module va effectuer une recherche basique.
Il va chercher dans l'index les fichiers :

- qui si un chemin vers une arborescence a été spécifié, sont préfixes de celle-ci et sinon tous les fichiers de l'index.
- qui *match* avec la requête.

1/ matching
les fichiers trouvés seront ceux contenant tous les mots de la requête
(penser bien sûr à l'appauvrissement typographique)

2/ score (nécessaire pour l'ordre d'affichage)
Chaque fichier trouvé aura un score correspondant à la somme du nombre d'occurrences de chaque mot de la requête présent dans le fichier.

3/ affichage
On affichera le nombre total de fichiers trouvés et la liste des fichiers (avec leur chemin relatif depuis le chemin de l'arborescence spécifiée ou absolue si aucun chemin n'est spécifié) avec leur score (séparés par un ; sans aucun espace ni CR superflu).
Les fichiers seront affichés dans l'ordre décroissant du score. Pour les fichiers de même score, on utilisera l'ordre ASCII du chemin complet affiché.

TOTAL:<nombre de fichiers>
chemin_relatif_ou_absolue_fichier1;<nombre total d'occurrences des mots de la requête dans ce fichier>
chemin_relatif_ou_absolue_fichier2;<nombre total d'occurrences des mots de la requête dans ce fichier>
...

Les '<' et '>', ne doivent pas apparaître dans le format de sortie.

AFFICHAGE DE L'INDEX

Attention, le format décrit ici devra être scrupuleusement respecté.

Pour affichage de l'index, on lancera :

```
$ java -jar indexer.jar --print [racine de l'arborescence]
```

Le format obtenu sera le suivant :

- 1 ligne par mot-clé (en minuscule)
- aucun espace superflu
- un = pour séparer le mot-clé des fichiers
- des , comme séparateur de fichiers
- un retour à la ligne à la fin de ligne ('\n' sous Unix ou '\r'\n' sous Windows)
- l'ordre d'affichage des mots-clés est l'ordre ASCII, l'ordre d'affichage des fichiers est aussi l'ordre ASCII
- les fichiers affichés avec le chemin relatif par rapport à la racine de l'arborescence, commençant tous par './'

soit :
```
motcle1=fichier1(type1,langue1),fichier2(type2,langue2),fichier3(type3,
langue3)
motcle2;fichier2(type2,langue2),fichier4(type4,langue4),fichier5(type5,
language5)
...
```

avec :
- type devant être (en minuscule) xml ou text ou les autres types que vous supportez.
- langue devant être le code iso369-1 du langage. soit fr pour le Français, et en pour l'anglais (en minuscule).

MODULE STATISTIQUE

Ce module permet d'obtenir des informations générales sur l'index:

Attention, le format décrit ici devra être scrupuleusement respecté.

3 options de statistiques sont demandées :

- des statistiques par type de fichier (--type),
- par language (--lang) ou
- par mot (--word). Si aucune option

De plus, si aucune option n'est demandée, ce module devra afficher toute les statistiques.

1/ obtention de statistiques par type de fichiers

```
$ java -jar indexer.jar --stat --type
```

affiche exactement (sans aucun espace ni CR superflu):
```
total:<nombre total de fichiers indexés+taille total en octet>
text:<nombre de fichier de ce type+taille en octet>
xml:<nombre de fichier de ce type+taille en octet>
...
```

La liste doit être ordonnée par nombre décroissant de fichiers.

2/ obtention de statistiques par langue

```
$ java -jar indexer.jar --stat --lang
```

affiche exactement (sans aucun espace ni CR superflu):
```
TOTAL:<nombre total de fichiers indexés>
en:<nombre de fichier de cette langue>
fr:<nombre de fichier de cette langue>
```

la liste doit être ordonnée par nombre décroissant de fichiers

3/ obtention de statistiques par mot-clé

Pour chaque mot-clé, on affichera le nombre total d'occurrences et le nombre de fichiers dans lesquels ce mot-clé a été trouvé.
Les mots-clés seront affichés dans l'ordre ASCII (rappel: tout en majuscule sans accent).

```
$ java -jar indexer.jar --stat --word
```

affiche exactement (sans aucun espace ni CR superflu):

```
TOTAL:<nombre total de fichiers indexés>/<nombre total
d'occurrences>
mots-clé:<nombre total d'occurrences>/<nombre de fichiers>
mots-clé:<nombre total d'occurrences>/<nombre de fichiers>
....
```

Aide

Une option --help doit permettre d'obtenir l'aide pour toutes les options supportées par votre programme.

Quelques conseils pour démarrer ...

- Posez vous la question sur structure de données à utilisées pour l'index, plusieurs peuvent convenir mais le choix d'un algortithme ayant une complexité de recherche trop grande sera sanctionné.
- Réfléchissez bien à la manière dont vous allez stocker votre index, aussi bien en mémoire que sur disque
- Commencer par traiter les fichiers textes simples. Quand vous traiterez les fichiers XML, seule la génération de l'index changera, mais l'affichage de l'index, les stats et la recherche sont indépendantes du type de fichier.
- Ne vous occupez pas immédiatement de la détection de la langue.

- Pensez que 50 000 fichiers et 100 000 mots ce n'est pas beaucoup.
- Pensez que vous n'avez pas forcément les droits en lecture et/ou écriture.

Format du Rendu

Le projet est à faire par binome.
Le document est à envoyer par mail aux adresses (notez le pluriel) indiquées au début de ce document au plus tard le dimanche 8 juin à 23h59 sous forme d'une archive au format ZIP (pas de rar, ni de tar, ni de tgz, ni de lhz etc.)

Le second rendu devra impérativement être effectué par les mêmes binôme que le précédent rendu.
Voici le nom des répertoires et fichiers qui doivent être contenus dans l'archive zip.

1. un fichier **readme.txt** indiquant comment compiler et exécuter le programme, où se trouve la doc, *etc.*
2. un fichier Ant **build.xml** permettant de compiler les sources du programme, et de créer le jar *exécutable* **indexer.jar**.
3. un répertoire **src** contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages. Les interfaces et classes contenues dans fr.umlv.solidvision ne doivent pas être modifiées.
4. un répertoire **classes** contenant l'ensemble des classes (.class) correspondant aux

sources sous forme de plusieurs paquetages.

5. un répertoire **bin** contenant deux fichiers, **indexer.sh**, script shell démarrant le logiciel sous Linux (en fait Unix) et **indexer.bat** démarrant le logiciel sous Windows.
6. un répertoire **test** contenant des sous-répertoires contenant des tests de fichiers à indexés.
7. un répertoire **docs** contenant deux documents au format PDF :
   - La documentation utilisateur **user.pdf** contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'application, et comment l'utiliser ainsi qu'une liste des bugs connus.
   - La documentation développeur **dev.pdf** contenant
     - une description de l'algorithme/structure de donnée que vous avez employé
     - une description de ce que fait grossièrement chaque classe lorsque l'on créé un index, effectue une recherche, affiche des stats, etc.
     - une description détaillée et au niveau de chaque classe que vous avez implantée.

En plus des deux documents, le répertoire **docs** devra contenir un sous-répertoire **api** contenant la documentation complète du logiciel au format javadoc (en anglais).

☞ ATTENTION : Le programme sera testé à l'aide d'un jeu unique de scripts de tests. Aucune liberté avec la syntaxe définie n'est donc possible.

Références

- pour récupérer la liste des fichiers voir la classe *java.io.File*
- pour parser les fichiers XML, vous utiliserez le parser SAX *javax.xml.parsers.SAXParser*
- pour manipuler les fichiers ZIP, regarder la classe *java.util.zip.ZipFile*
- pour les codes ISO639-1, voir http://www.loc.gov/standards/iso639-2/php/English_list.php
- pour les types mime : http://filext.com/