Université de Bordeaux I

Compte rendu du TP
UE Programmation large echelle

Réalisé par :

GAMELIN Antoine
LASSOUANI Sofiane

Université Bordeaux 1 (2016–2017) – Master Génie Logiciel (UE PLE)

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

L'objectif de ce TP est de se familiariser avec les bases de l'API hadoop (HDFS). Les besoins énoncés sont :

— La réalisation d'un programme JAVA, qui permet de faire un équivalent de la commande hdfs dfs -CopyFromLocal <input ><output >

— L'implémentation de la commande MergeFromLocal qui permet concatener le programme en utilisant appendToFile. Cette commande prend en paramètres une liste de fichiers en entrée et le flux hdfs pour stocker la concatenation de ces fichiers.

— Implémentation d'un programme permettant de générer un mot à partir une liste de syllable que nous avons prédéfinis.

CHAPITRE 2

DIAGRAMME DE CLASSE

Réalisation du diagramme de Classe, des classes/interfaces suivantes :
— org.apache.hadoop.conf.Configuration

— org.apache.hadoop.conf.Configured
— org.apache.hadoop.fs.FileSystem
— org.apache.hadoop.fs.Path
— org.apache.hadoop.io.IOUtils
— org.apache.hadoop.util.Tool
— org.apache.hadoop.util.ToolRunner

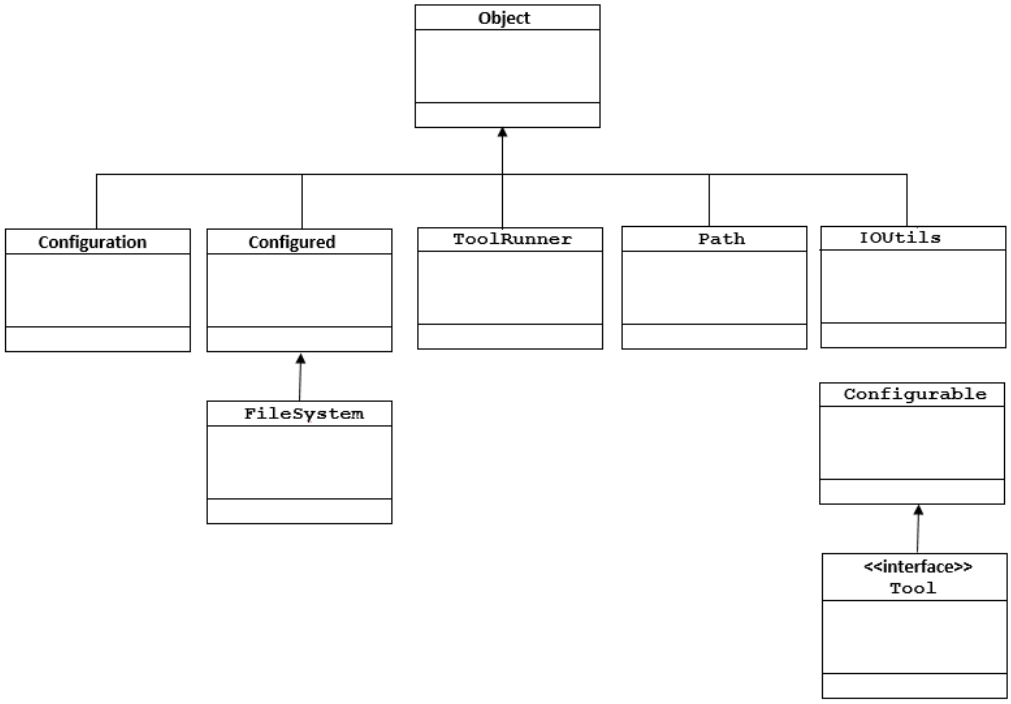

FIGURE 2.1 – Diagramme de class

CHAPITRE 3

PROJET

3.1 Présentation des classes utilisés

Pour réaliser ces différents programmes nous avons utilisés les classes suivantes

3.1.1 FileSystem

Une classe de base abstraite pour un système de fichiers assez générique . Il peut être mis en oeuvre comme un système de fichiers distribué ou comme "local" qui reflète le disque connecté localement.

3.1.2 URI

URI signifie Universal Resource Identifier, URI est le protocole qui sert à définir le chemin pour récupérer la ressource .

3.1.3 OutputStream

Cette classe abstraite est la superclasse de toutes les classes représentant un flux de sortie d'octets

3.1.4 InputStream

Cette classe abstraite est la superclasse de toutes les classes représentant un flux d'octets d'entrée

3.1.5 BufferedInputStream

BufferedInputStream permet de lire sur un flux de données en utilisant une zone mémoire tampon (buffer).
Cette classe se charge de lire sur le flux de données un grand nombre d'octets qu'elle garde dans un tampon.
Tous les appels à la méthode read() ultérieurs sur une instance de cette classe renvoient des données provenant de ce tampon tant qu'il y reste des octets à lire, le tampon est rechargé à partir du fichier quand on a épuisé toutes les informations qui s'y trouvent.

3.1.6 ByteArrayInputStream

Un ByteArrayInputStream contient une mémoire tampon interne qui contient les octets qui peuvent être lus à partir du flux. Un compteur interne conserve une trace de l'octet suivant devant être fourni par la méthode de lecture.

3.1.7 FileInputStream

Un FileInputStream obtient les octets d'entrée d'un fichier dans un système de fichiers, il est destiné à la lecture de flux d'octets bruts.

3.1.8 IOUtils.copyBytes

IOUtils est une classr qui propose des methodes d'entré/sortie, (input/output) . IOUtils.copyBytes nous permet de faire une copie d'un flux a un autre.

3.2 CopyFromLocal

La classe CopyFromLocal est un équivalent de la commande hdfs dfs -CopyFromLocal. Le principe de cette commande est de transférer un fichier local sur un serveur HDFS.

Utilisation de la commande : hadoop jar CopyFromLocal <namejar ><src file ><path output hdfs >

Nous avons implémenté le code vue en cours, nous avons juste rajouté un contrôle qui permet de s'assurer que nous passons bien deux paramètres.

Pour pouvoir écrire sur le HDFS, il faut s'assurer d'avoir créer un dossier avec des droits suffisant pour l'utilisateur hadoop, afin de pouvoir transférer le fichier sur le serveur

3.3 MergeFromLocal

La classe MergeFromLocal permet de concatener plusieurs fichiers en un seul fichier sur un serveur HDFS.
Ce programme va ouvrir fichier par fichier et les envoyer sur le HDFS en les concatenant. Le fichier sera envoyé sur le HDFS définit en dernier paramètre.

Utilisation de la commande : hadoop jar MergeFromLocal <namejar><file1>[<file n>]* <path output hdfs >

Nous avons implémenté le code vue en cours, nous avons juste rajouté un contrôle qui permet de s'assurer que nous passons au minimum deux paramètres.

3.4 GenerateWord

Le but de cette classe est de générer des mots à partir d'un ensemble de syllables tiré aléatoirement.

Utilisation de la commande : hadoop jar GenerateWord <namejar ><number syllabes ><path output hdfs >

Pour cela nous avons définit une liste de String (syllabes) qui contient la liste des syllabes prédéfinit.

Une fois la liste de syllabes prédéfinit, nous récupérons n syllabe (variable définit en paramètre) à l'aide d'une boucle for.

Une fois la chaine générer dans une variable nous utilisons la classe ByteArrayInputStream pour flux d'entrée afin de l'envoyer sur le hdfs.

Un exemple de test du programme :

```
$ hadoop jar ~/Documents/minimal-skeleton-0.0.1.jar GenerateWord 8 hdfs ://10.0.205.6 :9000/users/antga
$ hdfs dfs -copyToLocal hdfs ://10.0.206.6 :9000/users/antgamelin/resultWord /net/cremi/antgamelin/resu
$ more ~/result
dotpopopotatedeclap
```

3.5 Help

Nous avons créer une petite classe help, qui permet de lister la liste des commandes disponible.

3.6 Main

Pour executer les différentes méthodes nous avons fait en sorte de prendre le premier paramètre comme le nom de la fonction qu'on souhaite executé.

Au départ nous avons essayé d'utiliser la reflection java pour faire automatiquement cette appel.

Mais il y a eu une erreur, c'est pourquoi nous avons dû opter pour la solution d'utilisation d'un switch par manque de temps pour faire le bon appel en fonction de l'argument.

CHAPITRE 4

CONCLUSION

Ce TD nous a permi de découvrir les bases de l'API HDFS.
Aujourd'hui nous savons produire du code JAVA qui permet de :

— Transférer un fichier local sur un serveur hdfs (CopyFromLocal)
— Transferer un fichier local en le concaténant à un fichier du serveur HDFS (MergeFromLocal)
— Transférer une chaine de caractère (String) sur un serveur HDFS (GenerateWord)