Rapport TP2

Willian Ver Valen Paiva Alan Guitard

September 18, 2016

Contents

1 the API

1.1 org.apache.hadoop.conf.Configuration

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Configuration
extends Object
implements Iterable<Map.Entry<String,String>>, Writable
```

Cette classe est chargé de fournir un accès à la configuration des paramètres depuis le répertoire de configuration.

1.2 org.apache.hadoop.conf.Configured

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Configured
```

extends Object
implements Configurable

Les objets pouvant être configurés par un objet de type Configuration étendront cette classe.

1.3 org.apache.hadoop.fs.FileSystem

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public abstract class FileSystem
extends Configured
implements Closeable
```

Cette classe est utilisé pour accéder un fichier distribué d'Hadoop.

1.4 org.apache.hadoop.fs.Path

```
@Stringable
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Path
extends Object
implements Comparable
```

Cette classe contient le nom et le chemin d'un fichier de classe FileSystem.

1.5 org.apache.hadoop.io.IOUtils

```
@InterfaceAudience.Public
@InterfaceStability.Evolving
public class IOUtils
extends Object
```

Cette classe est une boîte à outils qui contient des fonctions d'entrée/sortie.

1.6 org.apache.hadoop.util.Tool

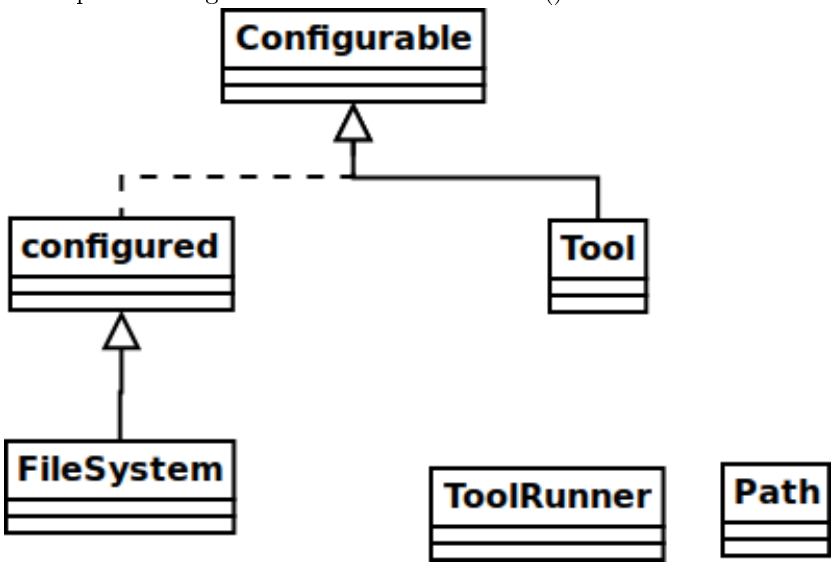
```
@InterfaceAudience.Public
@InterfaceStability.Stable
public interface Tool
extends Configurable
```

Cette interface est en charge des options en ligne de commande génériques.

1.7 org.apache.hadoop.util.ToolRunner

```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class ToolRunner
extends Object
```

Cette classe implémente l'interface `Tool` et lance donc les lignes de commandes qu'on lui passe en argument dans sa méthode run().



2 Deux premiers programmes d'exercices

Comme l'implémentation de ces programmes ont été donnés en classe, notre solution sera donc plutôt similaire. Nous allons donc expliquer comment ces programmes fonctionne pas à pas.

2.1 La base

La première étape est de créer un objet implémentant `Tool` qui implémentera donc la méthode de run(), laquelle sera appelée en conjonction avec la méthode run() de `ToolRunner`.

2.2 La méthode run()

C'est dans cette méthode que tout le programme se situe. Voici décrit ci-après les étapes suivies pour accomplir la fonctionnalité du premier programme:

1. Créer un objet URI avec le chemin de sortie (fichier sur le HDFS).
2. Normaliser l'URI, ce qui signifie supprimer les points ou double-points du chemin, par exemple.
3. Créer l'objet de classe `Path` à partir de l'URI normalisé.
4. Créer un object de classe `Configuration` et y charger l'actuel configuration via la fonction `getConf()`.
5. Créer un object de classe `FileSystem` spécifiant le chemin, la configuration et l'utilisateur.
6. Créer les objets utiles à l'écriture, `OutputStream`, sur le fichier nouvellement créé.

7. Créer les objets utiles à la lecture, `InputStream`, sur le fichier local.
8. Copier les bytes de l'`InputStream` vers `OutputStream`.
9. Fermer les flux.

Pour le second programme, les étapes sont plutôt, à l'exception de l'étape 7 et de celle qui suivent:

8. Créer une boucle pour lire chaque fichier.
9. Créer les objets utiles à la lecture, `InputStream`, sur le fichier local courant.
10. Copier les bytes de l'`InputStream` vers `OutputStream`.
11. Fermer les flux de lecture.

et, après la fin de ces actions sur tout les fichiers, fermer le flux d'écriture.

3 Troisième exercice: Génération de mots

L'implémentation de ce dernier exercice ne diffère pas énormément des deux précédents. Pour accomplir l'objectif, trois fonctions ont été implémentées:

```
public static char getRandom(char[] array) {
    int rnd = new Random().nextInt(array.length);
    return array[rnd];
}

public static String randomSyllable(){
    char[] alphabet = "abcdefghijklmnopqrstuvwxyz".toCharArray();
    return new StringBuilder().append(getRandom(alphabet)).append(
        getRandom(alphabet)).toString();
}

public static String randomWord(int size){
    String exampleString = "";
    for(int i=0 ; i < size; i++){
        exampleString += randomSyllable();
    }
    return exampleString + " ";
 }
```

Ensuite, le même code que le précédent exercice en changeant la boucle pour itérer un nombre de fois définis en argument. La façon dont le flux de lecture a été crée est aussi modifié:

```
InputStream is = new ByteArrayInputStream(randomWord(10).getBytes(
    StandardCharsets.UTF_8));
```

Cette modification est nécessaire pour lire les bytes non plus à partir d'un fichier mais d'une chaîne de caractères.

4 Exercice 7: Temps d'exécution

Dans cet exercice, nous pouvons la différence de temps d'exécution entre ce programme et le script écrit dans l'exercice 4. La version écrite en Java est bien plus rapide. Ceci s'explique par le fait que la version script ouvre et ferme un descripteur de fichier sur chaque fichier, ce qui est une opération coûteuse en temps. En l'occurence, sur la version de Java, le descripteur de fichier s'ouvre sur le HDFS jusqu'à ce que tout les fichiers soit concaténés à l'intérieur, puis se ferme, ce qui réduit considérablement le nombre d'opérations dans la version Java.