Tutoriel pour extraire des données depuis une feuille Excel, les transformer et les sauvegarder : utilisation d'Apache POI

Par Marc AUTRAN

Date de publication : 27 juillet 2016

CONFIRMÉ

Durée : 2 heures

Cet article montre comment automatiser le chargement de données depuis un fichier Excel vers une collection Java. L'utilitaire développé s'appuie sur la célèbre bibliothèque Open Source **Apache POI**

N'hésitez pas à donner vos avis sur ce tutoriel sur le forum : **Commentez**

I - Introduction

I-A - Les prérequis

Ce tutoriel s'adresse à des lecteurs qui connaissent déjà le langage JAVA et sont familiarisés avec la pensée objet.

Les bibliothèques et versions des logiciels utilisées pour les besoins de l'article :

- JDK Oracle 8 ;
- Eclipse JEE Mars ;
- l'API Apache POI 3.14 ;
- un éditeur de fichiers Excel, OpenOffice sera utilisé pour ce tutoriel.

I-B - L'origine du besoin

Dans le cadre des Tests/Recette d'un projet, on doit fréquemment réaliser des chargements de données depuis des fichiers Excel vers des tableaux JAVA, mais également PHP ou JavaScript.
Ces chargements se révèlent être une perte de temps importante. Un outil pour réaliser cette tâche nous serait particulièrement utile. Ce besoin ne s'applique pas aux bases de données qui, quant à elles, se peuplent aisément avec des fichiers Excel.
Des listes de données ou jeux de tests, sont effectivement fournis par le groupe des utilisateurs, sous forme de fichiers Excel. Ce format bureautique universel est tout à fait légitime, dans la mesure ou l'outil bureautique afférent (Excel), est une sorte de prolongement de la main chez nos utilisateurs bureauticiens. Néanmoins, ce format n'est pas vraiment idéal pour la fourniture de données et encore moins pour des jeux de tests.
La problématique des données fournies sous ce format est triple :

- il faut sélectionner manuellement les données qui ne commencent pas à une ligne précise (ni une colonne d'ailleurs) dans la feuille. Les feuilles, dans le classeur, peuvent être nombreuses et leur dénomination répond parfois à une logique très floue ;
- le typage des données, est également une source de difficulté, car il n'est parfois même pas cohérent :
    - certaines données numériques sont codées comme du texte. Cette remarque vaut aussi pour les dates…
    - il arrive que dans une même colonne, on se retrouve avec des cellules de types différents (suite à des copier/coller malheureux) ;
- l'en-tête de tableau, donc l'organisation des colonnes, n'est pas toujours conforme à ce qui était demandé initialement.

Cette problématique étant particulièrement récurrente, on peut en déduire qu'elle ne changera pas. D'ailleurs, imposer plus de rigueur aux groupes des utilisateurs n'est pas forcement une bonne chose, dans la mesure ou cela risque de nuire à la bonne entente de l'équipe et que leur emploi du temps les contraint souvent à livrer aux équipes techniques un petit fichier copier/coller entre deux rendez-vous.
Disposer d'un outil permettant de lire vite et proprement les fichiers Excel semble donc être la solution idoine.

II - Le cahier des charges

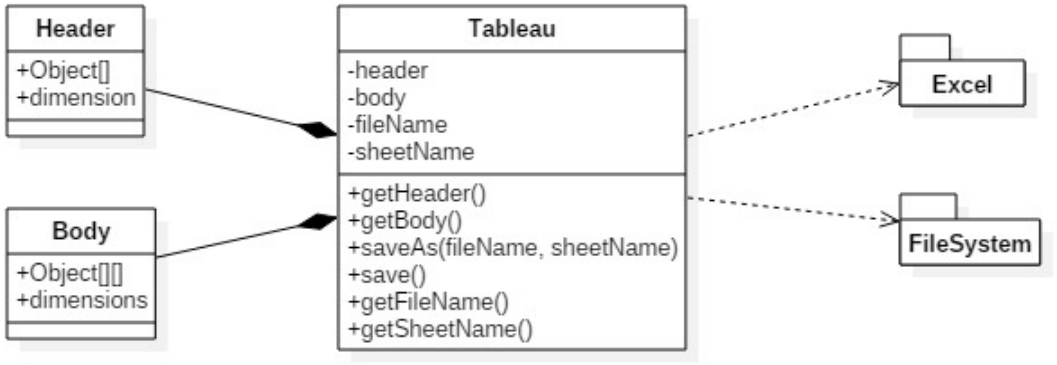
Le besoin se résume à extraire d'une feuille Excel l'en-tête d'un tableau ainsi que ses valeurs, puis à injecter ces données dans deux tableaux distincts que nous appellerons *header* et *body*. Le premier sera de dimension 1 et le second de dimension 2.

Bien que cette description soit extrêmement simple, les contraintes d'exploitation suivantes devront être respectées lors de l'implémentation de l'outil :

- on ne sait pas où commence le tableau dans la feuille ni quelles sont ses dimensions. Mais on sait en revanche que notre expert métier nous fournira des tableaux continus. C'est-à-dire qu'une feuille ne contiendra qu'un seul et unique tableau (pas de dessin, de formulaire…) ; on impose que le fichier Excel, une fois chargé, soit libéré pour d'autres utilisateurs dans l'instant. C'est-à-dire qu'on précisera à l'outil le nom du fichier et de la feuille dans le classeur, l'outil chargera les données et libérera le fichier pour un autre utilisateur ;
- on ne connaît pas le type des données dans les cellules (varchar, numeric, boolean, date) ;
- une fois les tableaux opérés par l'application, on doit pouvoir les sauvegarder dans un fichier Excel.

III - L'architecture logicielle

Voici le diagramme des classes qui modélise :

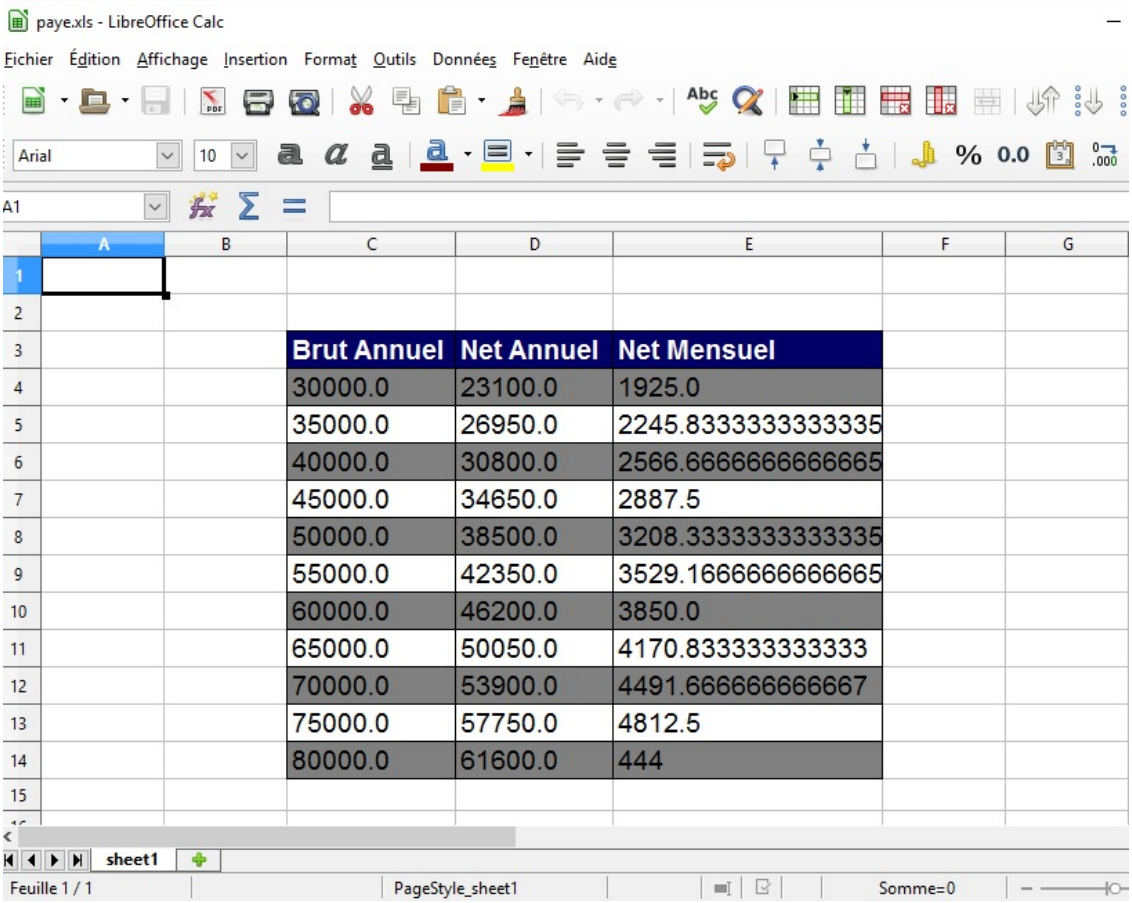


l'outil sera constitué d'une seule classe monolithique.

On y retrouve pour les propriétés, l'en-tête (header) et le corps (body) du tableau. Les méthodes quant à elles, sont le miroir du cahier des charges, puisqu'elles permettent d'extraire l'en-tête ou le corps du tableau, ou encore de sauvegarder les données dans un ficher Excel.

Nous utiliserons l'API Apache POI pour manipuler le fichier Excel. Cette bibliothèque, qui permet de manipuler les fichiers bureautiques en général (.doc - .xls…), est expliquée dans le cas particulier des fichiers Excel par Thierry-Leriche-Dessirier dans **ce tutoriel**.

Le fichier paye.xls que nous utiliserons sera le suivant :



Brut Annuel	Net Annuel	Net Mensuel
30000.0	23100.0	1925.0
35000.0	26950.0	2245.8333333333335
40000.0	30800.0	2566.6666666666665
45000.0	34650.0	2887.5
50000.0	38500.0	3208.3333333333335
55000.0	42350.0	3529.1666666666665
60000.0	46200.0	3850.0
65000.0	50050.0	4170.833333333333
70000.0	53900.0	4491.666666666667
75000.0	57750.0	4812.5
80000.0	61600.0	444

Pour tester notre classe Tableau, nous chargerons le fichier dans un tableau et changerons dans celui-ci la dernière valeur. Nous finirons le test en sauvegardant ce tableau dans le même fichier.

On remarquera que le tableau ne commence ni à la première ligne ni à la première colonne de la feuille.

Voici le code source du test final :

```
1. public class TestOutil
2. {
3.     public static void main(String[] args)
4.     {
5.         Tableau tab = new Tableau("paye.xls", "sheet1");
6.         Object corps[][] = tab.getBody();
7.         System.out.println(corps[10][2]);
8.         corps[10][2] = 5000;
9.         Object corps2[][] = tab.getBody();
10.         System.out.println(corps2[10][2]);
11.         tab.save();
12.     }
13. }
```

IV - La classe Tableau

Le constructeur Tableau(String fileName, String sheetName) prend en paramètre le nom du fichier et de la feuille qui contient le tableau.

La méthode saveAs(String fileName, String sheetName) prend en paramètre le nom du fichier et de la feuille sous lesquels sauvegarder le Tableau.

La méthode saveAs() appelle la méthode précédente avec les derniers paramètres sauvegardés (à défaut les paramètres du fichier initial).

Le code source de la classe Tableau :

```
1. import java.io.FileInputStream;
2. import java.io.FileOutputStream;
3. import java.io.IOException;
4. import org.apache.poi.hssf.usermodel.HSSFWorkbook;
5. import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
6. import org.apache.poi.ss.usermodel.Cell;
7. import org.apache.poi.ss.usermodel.Row;
8. import org.apache.poi.ss.usermodel.Sheet;
9. import org.apache.poi.ss.usermodel.Workbook;
10. import org.apache.poi.ss.usermodel.WorkbookFactory;
11. public class Tableau
12. {
13.     private String header[];
14.     private Object body[][];
15.     private String lastFileName = null;
16.     private String lastSheetName = null;
17.
18.     public Tableau(String fileName, String sheetName)
19.     {
20.         try
21.         {
22.             this.setLastFileName(fileName);
23.             this.setLastSheetName(sheetName);
24.             FileInputStream file = new FileInputStream(fileName);
25.             Workbook workbook = WorkbookFactory.create(file);
26.             final Sheet sheet = workbook.getSheet(sheetName);
27.             int top = sheet.getFirstRowNum();
28.             int bottom = sheet.getLastRowNum();
29.             Row line = sheet.getRow(top);
30.             int start = line.getFirstCellNum();
31.             int end = line.getLastCellNum();
32.             int length = end - start;
33.             while(length == 0)
34.             {
35.                 top++;
36.                 line = sheet.getRow(top);
37.                 start = line.getFirstCellNum();
38.                 end = line.getLastCellNum();
39.                 length = end - start;
40.             }
41.             int hight = bottom - top;
42.             this.header =  new String[length];
43.             this.body = new Object[hight][length];
44.             for (int i = 0; i < length; i++)
45.             {
46.                 header[i] = line.getCell(start + i).getStringCellValue();
47.             }
48.
49.             for (int index = 0; index < hight; index++)
50.             {
51.                 line = sheet.getRow(index + top + 1);
52.                 for (int i = 0; i < length; i++)
53.                 {
54.                     Cell cellule = line.getCell(start + i);
55.                     switch (cellule.getCellType())
56.                     {
57.                         case Cell.CELL_TYPE_STRING :
58.                             this.body[index][i] = cellule.getStringCellValue();
59.                             break;
60.                         case Cell.CELL_TYPE_BOOLEAN :
61.                             this.body[index][i] = cellule.getBooleanCellValue();
62.                             break;
63.                         default :
64.                             this.body[index][i] = cellule.getNumericCellValue();
65.                     }
```

```
                }
            }
            workbook.close();
            file.close();
        }
         catch (InvalidFormatException | IOException e)
        {
             e.printStackTrace();
        }
    }

    public void saveAs(String fileName, String sheetName)
    {
        try
        {
            if (this.getLastFileName().compareTo(fileName) != 0)
                this.setLastFileName(fileName);
            if (this.getLastSheetName() != sheetName)
                this.setLastSheetName(sheetName);
            Workbook workbook = new HSSFWorkbook();
            Sheet sheet = workbook.createSheet(sheetName);
            Row row = sheet.createRow(0);
            for(int i = 0; i < this.getHeader().length; i++)
            {
                row.createCell(i).setCellValue(this.getHeader()[i]);
            }

            for (int index = 0; index < this.getBody().length; index++)
            {
                row = sheet.createRow(index + 1);
                for (int i = 0; i < this.getBody()[index].length; i++)
                {
                    String valeur = String.valueOf(this.getBody()[index][i]);
                    row.createCell(i).setCellValue(valeur);
                 }
             }
             FileOutputStream fileOut = new FileOutputStream(fileName);
             workbook.write(fileOut);
             workbook.close();
             fileOut.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    public void save()
    {
        this.saveAs(this.getLastFileName(), this.getLastSheetName());
    }

    public String[] getHeader()
    {
        return this.header;
    }

    public Object[][] getBody()
    {
        return this.body;
    }

    public String getLastFileName() {
        return this.lastFileName;
    }

    private void setLastFileName(String lastFileName) {
        this.lastFileName = lastFileName;
    }

    public String getLastSheetName() {
        return this.lastSheetName;
```

```
138.     }
139.
140.     private void setLastSheetName(String lastSheetName) {
141.         this.lastSheetName = lastSheetName;
142.     }
143. }
```

V - Conclusions et remerciements

Cet outil de chargement de données depuis Excel satisfait aux tests fonctionnels. Ce mini ETL est utilisé couramment sur des fichiers Excel pour lesquels on manipule en Java des tableaux de plus d'un million de cellules. Pour des fichiers de ce volume (10 000 lignes et une centaine de colonnes), le temps de chargement et de sauvegarde est inférieur à une seconde sur un PC bureautique d'entrée de gamme.

Néanmoins, cet outil ne peut pas être considéré comme abouti, dans la mesure où ce n'est qu'une classe monolithique brute de décoffrage (pas de patron de conception…) et que tous les types de cellules ne sont pas testés…

Il est intéressant de noter que si ce tutoriel permet d'extraire des données depuis un fichier .xls en utilisant le package HSSF, il existe aussi le package XSSF qui permet, quant à lui, de manipuler de façon similaire des fichiers .xlsx.

Nous tenons à remercier **Mickael Baron** et **OButterlin** pour la relecture technique et **jacques_jean** pour la relecture orthographique de cet article.