Java Basic for Tester

Reading and Writing Excel file in Java

Agenda

- *What is Apache POI?*
- *Interfaces and Classes in Apache POI*
- *Apache POI library – Writing a Simple Excel*
- *Apache POI library – Reading an Excel file*

What is Apache POI?

Apache POI is well trusted library among many other open source libraries to handle such usecases involving excel files. Using POI, you can read and write MS Excel files using Java.

What is Apache POI?

If you are working on a maven project, you can include the POI dependency in pom.xml file using this:

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>4.1.2</version>
</dependency>
```

If you are not using maven, then you can download maven jar files from POI download page

dom4j-1.6.1.jar
poi-3.9-20121203.jar
poi-ooxml-3.9-20121203.jar
poi-ooxml-schemas-3.9-20121203.jar
xmlbeans-2.3.0.jar

Interfaces and Classes in Apache POI

Apache POI main classes usually start with either HSSF, XSSF or SXSSF.

HSSF – is the POI Project's pure Java implementation of the Excel '97(-2007) file format. e.g. HSSFWorkbook, HSSFSheet.

XSSF – is the POI Project's pure Java implementation of the Excel 2007 OOXML (.xlsx) file format. e.g. XSSFWorkbook, XSSFSheet.

SXSSF (since 3.8-beta3) – is an API-compatible streaming extension of XSSF to be used when very large spreadsheets have to be produced, and heap space is limited. e.g. SXSSFWorkbook, SXSSFSheet. SXSSF achieves its low memory footprint by limiting access to the rows that are within a sliding window, while XSSF gives access to all rows in the document.

Interfaces and Classes in Apache POI

Row and Cell

Apart from above classes, Row and Cell are used to interact with a particular row and a particular cell in excel sheet.

Style Classes

A wide range of classes like CellStyle, BuiltinFormats, ComparisonOperator, ConditionalFormattingRule, FontFormatting, IndexedColors, PatternFormatting, SheetConditionalFormatting etc. are used when you have to add formatting in a sheet, mostly based on some rules.

FormulaEvaluator

Another useful class FormulaEvaluator is used to evaluate the formula cells in excel sheet.

Apache POI library – Writing a Simple Excel

Writing excel using POI is very simple and involve following steps:

1. Create a workbook
2. Create a sheet in workbook
3. Create a row in sheet
4. Add cells in sheet
5. Repeat step 3 and 4 to write more data

Apache POI library – Writing a Simple Excel

```
package com.howtodoinjava.demo.poi;
//import statements
public class WriteExcelDemo
{
    public static void main(String[] args)
    {
        //Blank workbook
        XSSFWorkbook workbook = new XSSFWorkbook();

        //Create a blank sheet
        XSSFSheet sheet = workbook.createSheet("Employee Data");

        //This data needs to be written (Object[])
        Map<String, Object[]> data = new TreeMap<String, Object[]>();
        data.put("1", new Object[] {"ID", "NAME", "LASTNAME"});
        data.put("2", new Object[] {1, "Amit", "Shukla"});
        data.put("3", new Object[] {2, "Lokesh", "Gupta"});
        data.put("4", new Object[] {3, "John", "Adwards"});
        data.put("5", new Object[] {4, "Brian", "Schultz"});

        //Iterate over data and write to sheet
        Set<String> keyset = data.keySet();
        int rownum = 0;
        for (String key : keyset)
        {
            Row row = sheet.createRow(rownum++);
            Object [] objArr = data.get(key);
            int cellnum = 0;
            for (Object obj : objArr)
            {
               Cell cell = row.createCell(cellnum++);
               if(obj instanceof String)
                    cell.setCellValue((String)obj);
                else if(obj instanceof Integer)
                    cell.setCellValue((Integer)obj);
            }
        }
```

```
        try
        {
            //Write the workbook in file system
            FileOutputStream out = new FileOutputStream(new File("excel_demo.xlsx"));
            workbook.write(out);
            out.close();
            System.out.println(excel_demo.xlsx written successfully on disk.");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
```

	A	B	C	D
1	ID	NAME	LASTNAME	
2	1	Amit	Shukla	
3	2	Lokesh	Gupta	
4	3	John	Adwards	
5	4	Brian	Schultz	
6				

Apache POI library – Reading an Excel file

Apache POI – Read an excel file

```
//import statements
public class ReadExcelDemo
{
    public static void main(String[] args)
    {
        try
        {
            FileInputStream file = new FileInputStream(new File("excel_demo.xlsx"));

            //Create Workbook instance holding reference to .xlsx file
            XSSFWorkbook workbook = new XSSFWorkbook(file);

            //Get first/desired sheet from the workbook
            XSSFSheet sheet = workbook.getSheetAt(0);

            //Iterate through each rows one by one
            Iterator<Row> rowIterator = sheet.iterator();
            while (rowIterator.hasNext())
            {
                Row row = rowIterator.next();
                //For each row, iterate through all the columns
                Iterator<Cell> cellIterator = row.cellIterator();

                while (cellIterator.hasNext())
                {
                    Cell cell = cellIterator.next();
                    //Check the cell type and format accordingly
                    switch (cell.getCellType())
                    {
                        case Cell.CELL_TYPE_NUMERIC:
                            System.out.print(cell.getNumericCellValue() + "t");
                            break;
                        case Cell.CELL_TYPE_STRING:
                            System.out.print(cell.getStringCellValue() + "t");
                            break;
                    }
                }
                System.out.println("");
            }
            file.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Apache POI library – Reading an Excel file

Apache POI – Add and evaluate formula cells

When working on complex excel sheets, we encounter many cells which have formula to calculate their values. These are formula cells. Apache POI has excellent support for adding formula cells and evaluating already present formula cells also.

In this code, there are four cells in a row and fourth one in multiplication of all previous 3 rows. So the formula will be : A2*B2*C2 (in second row)

Apache POI library – Reading an Excel file

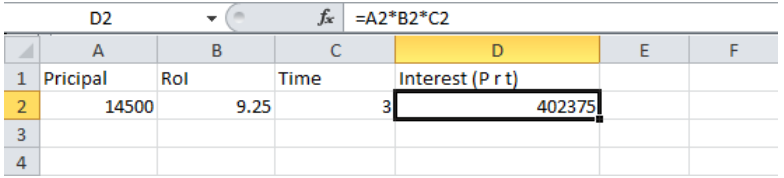
Apache POI – Formatting the cells

1) Cell value is in between a certain range:

This piece of code will color any cell in range whose value is between a configured range. [e.g. between 50 and 70]

Apache POI – Formatting the cells

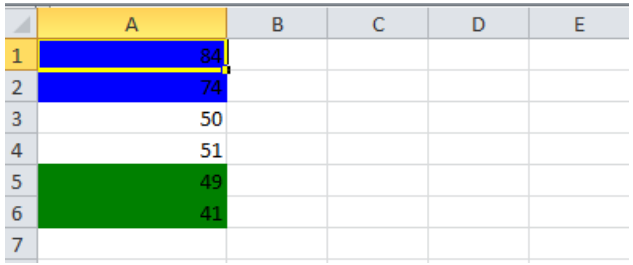
```
public static void main(String[] args)
{
    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet sheet = workbook.createSheet("Calculate Simple Interest");

    Row header = sheet.createRow(0);
    header.createCell(0).setCellValue("Pricipal");
    header.createCell(1).setCellValue("RoI");
    header.createCell(2).setCellValue("T");
    header.createCell(3).setCellValue("Interest (P r t)");

    Row dataRow = sheet.createRow(1);
    dataRow.createCell(0).setCellValue(14500d);
    dataRow.createCell(1).setCellValue(9.25);
    dataRow.createCell(2).setCellValue(3d);
    dataRow.createCell(3).setCellFormula("A2*B2*C2");

    try {
        FileOutputStream out =  new FileOutputStream(new File("formulaDemo.xlsx"));
        workbook.write(out);
        out.close();
        System.out.println("Excel with foumula cells written successfully");

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



Apache POI – Formatting the cells

2) Highlight duplicate values:
Highlight all cells which have duplicate values in observed cells.

```
static void basedOnValue(Sheet sheet)
{
    //Creating some random values
    sheet.createRow(0).createCell(0).setCellValue(84);
    sheet.createRow(1).createCell(0).setCellValue(74);
    sheet.createRow(2).createCell(0).setCellValue(50);
    sheet.createRow(3).createCell(0).setCellValue(51);
    sheet.createRow(4).createCell(0).setCellValue(49);
    sheet.createRow(5).createCell(0).setCellValue(41);

    SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();

    //Condition 1: Cell Value Is   greater than  70    (Blue Fill)
    ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule(ComparisonOperator.GT, "70");
    PatternFormatting fill1 = rule1.createPatternFormatting();
    fill1.setFillBackgroundColor(IndexedColors.BLUE.index);
    fill1.setFillPattern(PatternFormatting.SOLID_FOREGROUND);

    //Condition 2: Cell Value Is  less than      50    (Green Fill)
    ConditionalFormattingRule rule2 = sheetCF.createConditionalFormattingRule(ComparisonOperator.LT, "50");
    PatternFormatting fill2 = rule2.createPatternFormatting();
    fill2.setFillBackgroundColor(IndexedColors.GREEN.index);
    fill2.setFillPattern(PatternFormatting.SOLID_FOREGROUND);

    CellRangeAddress[] regions = {
            CellRangeAddress.valueOf("A1:A6")
    };

    sheetCF.addConditionalFormatting(regions, rule1, rule2);
}
```



	A	B	C	D	E
1	84				
2	74				
3	50				
4	51				
5	49				
6	41				
7					

Apache POI library – Reading an Excel file

Apache POI – Formatting the cells

3) Color alternate rows in different colors

```
static void expiryInNext30Days(Sheet sheet)
{
    CellStyle style = sheet.getWorkbook().createCellStyle();
    style.setDataFormat((short)BuiltinFormats.getBuiltinFormat("d-mmm"));

    sheet.createRow(0).createCell(0).setCellValue("Date");
    sheet.createRow(1).createCell(0).setCellFormula("TODAY()+29");
    sheet.createRow(2).createCell(0).setCellFormula("A2+1");
    sheet.createRow(3).createCell(0).setCellFormula("A3+1");

    for(int rownum = 1; rownum <= 3; rownum++) sheet.getRow(rownum).getCell(0).setCellStyle(style);

    SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();

    // Condition 1: Formula Is   =A2=A1    (White Font)
    ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule("AND(A2-TODAY()>=0,A2-TODAY()<=30)");
    FontFormatting font = rule1.createFontFormatting();
    font.setFontStyle(false, true);
    font.setFontColorIndex(IndexedColors.BLUE.index);

    CellRangeAddress[] regions = {
            CellRangeAddress.valueOf("A2:A4")
    };

    sheetCF.addConditionalFormatting(regions, rule1);

    sheet.getRow(0).createCell(1).setCellValue("Dates within the next 30 days are highlighted");
}
```

	A	B	C	D	E	F	G
1	Date	Dates within the next 30 days are highlighted					
2	18-Jul						
3	19-Jul						
4	20-Jul						
5							

Thank you