Python 使用xlrd处理 excel

下载地址：<http://pypi.python.org/pypi/xlrd> 要下载exe版本

官网地址：http://www.python-excel.org/

文档地址:https://secure.simplistix.co.uk/svn/xlrd/trunk/xlrd/doc/xlrd.html

文档pdf下载：http://www.simplistix.co.uk/presentations/python-excel.pdf

## 安装

2.Windows系统下，你可以下载运行xlrd-0.7.1.win32.exe安装。

## 读取Excel文件

下面展示的所有的例子都是基于xlrd目录的教程。

#### 打开Workbooks

Workbooks能从一个文件、一个mmap.mmap对象或一个字符串加载：

from mmap import mmap,ACCESS\_READ

from xlrd import open\_workbook

print open\_workbook('simple.xls')

with open('simple.xls','rb') as f:

print open\_workbook(

file\_contents=mmap(f.fileno(),0,acc

)

aString = open('simple.xls','rb').read()

print open\_workbook(file\_contents=aString)

#### 操作Workbook

这是一个简单操作workbook的例子：

from xlrd import open\_workbook

wb = open\_workbook('simple.xls')

for s in wb.sheets():

print 'Sheet:',s.name

for row in range(s.nrows):

values = []

for col in range(s.ncols):

values.append(s.cell(row,col).value)

print ','.join(values)

print

下面几乎没有小节涉及到操作workbook的更多细节。

#### 揭秘Book

通过open\_workbook返回的xlrd.Book对象包含了所有对工作簿要的事情，能被用于在工作簿中取得独立的sheet。

这个nsheets属性是一个整数，包含工作簿sheet的数量。这个属性与sheet\_by\_index方法结合起来是获取独立sheet最常用的方法。

sheet\_names方法返回包含工作簿中所有sheet名字的unicode列表。单独的sheet可以通过sheet\_by\_name方法使用这些名字获取。

sheets方法的结果是迭代获取工作簿中的每个sheet。

下面是这些方法和属性的例子示范：

from xlrd import open\_workbook

book = open\_workbook('simple.xls')

print book.nsheets

for sheet\_index in range(book.nsheets):

print book.sheet\_by\_index(sheet\_index)

print book.sheet\_names()

for sheet\_name in book.sheet\_names():

print book.sheet\_by\_name(sheet\_name)

for sheet in book.sheets():

print sheet

xlrd.Book对象有与工作簿内容相关的其它属性，但很少用到：

* codepage
* countries
* user\_name

如果你可能需要运用这些属性，请查看xlrd文档。

#### 揭秘Sheet

通过上面介绍的方法返回的xlrd.sheet.Sheet对象包含了所有对worksheet和它的内容操作的信息。

name属性是worksheet名字的unicode表示。

nrows和ncols属性分别包含了worksheet中的行数和列数。

下面例子展示了如何使用迭代来显示一个worksheet的内容：

from xlrd import open\_workbook,cellname

book = open\_workbook('odd.xls')

sheet = book.sheet\_by\_index(0)

print sheet.name

print sheet.nrows

print sheet.ncols

for row\_index in range(sheet.nrows):

for col\_index in range(sheet.ncols):

print cellname(row\_index,col\_index),'-',

print sheet.cell(row\_index,col\_index).value

xlrd.sheet.Sheet对象有其他一些与worksheet内容相关的属性，但很少使用：

* col\_label\_ranges
* row\_label\_ranges
* visibility

如果你认为你可能需要运用这些属性，请参看xlrd文档。

#### 获得特定的单元格

正如你在前面例子中看到的，Sheet对象的cell方法能用来返回特定单元格的内容。

cell方法返回一个xlrd.sheet.Cell对象。除了value包含了单元格的真实值，ctype包含了单元格的类型，Cell对象几乎没有其他属性。

另外，Sheet对象有两个方法返回这两种数据类型。cell\_value方法返回特定单元格的值，而cell\_type方法返回特定单元格的类型。这两个方法执行时比获取Cell对象更快。

后面会讲述更多Cell类型的细节。下面示范了这些方法，属性和起作用的类：

from xlrd import open\_workbook,XL\_CELL\_TEXT

book = open\_workbook('odd.xls')

sheet = book.sheet\_by\_index(1)

cell = sheet.cell(0,0)

print cell

print cell.value

print cell.ctype==XL\_CELL\_TEXT

for i in range(sheet.ncols):

print sheet.cell\_type(1,i),sheet.cell\_value(1,i)

#### 迭代Sheet的内容

我们已经见过怎么迭代worksheet的内容，获取产生的单独的单元格。然而，有更容易的方法来获取单元格组。有一套对称的方法来通过行或列获取单元格组的信息。

row和col方法分别返回一整行（列）的Cell对象。

row\_slice和col\_slice方法分别返回一行（列）中以开始索引和一个可选的结束索引为边界的Cell对象列表。

row\_types和col\_types方法分别返回一行（列）中以开始索引和一个可选的结束索引为边界的表示单元格类型的整数列表。

row\_values和col\_values方法分别返回一行（列）中以开始索引和一个可选的结束索引为边界的表示单元格值的对象列表。

下面是所有sheet迭代方法的示例：

from xlrd import open\_workbook

book = open\_workbook('odd.xls')

sheet0 = book.sheet\_by\_index(0)

sheet1 = book.sheet\_by\_index(1)

print sheet0.row(0)

print sheet0.col(0)

print

print sheet0.row\_slice(0,1)

print sheet0.row\_slice(0,1,2)

print sheet0.row\_values(0,1)

print sheet0.row\_values(0,1,2)

print sheet0.row\_types(0,1)

print sheet0.row\_types(0,1,2)

print

print sheet1.col\_slice(0,1)

print sheet0.col\_slice(0,1,2)

print sheet1.col\_values(0,1)

print sheet0.col\_values(0,1,2)

print sheet1.col\_types(0,1)

print sheet0.col\_types(0,1,2)

**实用方法**

当围绕workbook进行操作的时候，把行和列转换成用户习惯看到的Excel单元格引用（如：(0,0)转换成A1），这是很有用的。下面提供的方法帮助我们实现它：

cellname方法把一对行和列索引转换为一个对应的Excel单元格引用。

cellnameabs方法把一对行和列索引转换为一个绝对的Excel单元格引用（如：$A$1）。

colname方法把一个列索引转换为Excel列名。

下面是这三个方法的示例：

from xlrd import cellname, cellnameabs, colname

print cellname(0,0),cellname(10,10),cellname(100,100)

print cellnameabs(3,1),cellnameabs(41,59),cellnameabs(265,358)

print colname(0),colname(10),colname(100)

**Unicode**

由xlrd产生的所有文本属性不是unidecode对象，就是ascii字符串（很少）。

由Microsoft Excel输入的每个文本都是下列编码之一：

* Latin1,如果匹配
* UTF\_16\_LE，如果不匹配Latin1
* 在更老的文件中，是按MS字符集规范编码的。他们由xlrd映射到Python编码，结果仍是unicode对象。

其他知名软件用错误字符集或不用字符集写入Excel文件的情况是很少的。这种情况下，可能需要在open\_workbook方法中指定正确的字符集。

from xlrd import open\_workbook

book = open\_workbook('dodgy.xls',encoding='cp1252')

**单元格的类型**

我们已经看过单元格类型用一个整数表示。这个整数相当于xlrd识别单元格类型的一组常数。可能的单元格类型在下面部分全部被列出来了。

**Text 文本**

这是由xlrd.XL\_CELL\_TEXT常数表示的。   
这种类型的单元格的值是unicode对象。

**Number 数字**

这是由xlrd.XL\_CELL\_NUMBER常数表示的。   
这种类型的单元格的值是float对象。

**Date 日期**

这是由xlrd.XL\_CELL\_DATE常数表示的。

注意：日期在Excel文件中实际上是不存在的，它们只不过是特别格式化后的数字。

如果数字格式字符串看起来像日期，xlrd将会返回xlrd.XL\_CELL\_DATE作为单元格类型。

提供的xldate\_as\_tuple方法把日期单元格中的float数转化为适合实例化各种日期或时间对象的元组。这个例子展示了怎么使用它：

from datetime import date,datetime,time

from xlrd import open\_workbook,xldate\_as\_tuple

book = open\_workbook('types.xls')

sheet = book.sheet\_by\_index(0)

date\_value =

xldate\_as\_tuple(sheet.cell(3,2).value,book.datemode)

print datetime(\*date\_value),date(\*date\_value[:3])

datetime\_value =

xldate\_as\_tuple(sheet.cell(3,3).value,book.datemode)

print datetime(\*datetime\_value)

time\_value =

xldate\_as\_tuple(sheet.cell(3,4).value,book.datemode)

print time(\*time\_value[3:])

print datetime(\*time\_value)

说明：

* Excel文件有两种可能的日期模式，一种用于最初由Windows创建的文件，一种用于最初由苹果电脑创建的文件。这个日期模式被表示成xlrd.Book对象的datemode属性，且必须传值给xldate\_as\_tuple方法。
* Excel文件格式对1904年一月3日以前的日期有各种问题，引起日期混乱，从而抛出XLDateError错误。
* Excel公式方法DATE()在某些情况下会返回出乎意料的日期。

**Boolean 布尔值**

这是由xlrd.XL\_CELL\_BOOLEAN常数表示的。   
这种单元格的值是bool对象。

**Error 错误**

这是由xlrd.XL\_CELL\_ERROR常数表示的。   
这种单元格的值是表示特定错误代码的整数。   
error\_text\_from\_code方法用来把错误代码转换为错误信息：

from xlrd import open\_workbook,error\_text\_from\_code

book = open\_workbook('types.xls')

sheet = book.sheet\_by\_index(0)

print error\_text\_from\_code[sheet.cell(5,2).value]

print error\_text\_from\_code[sheet.cell(5,3).value]

对一种明显显示所有单元格类型的简单方法，参看xlutils.display。

**Empty/Blank 空值或空白**

Excel只是在单元格中存储信息，或者对单元格格式化。而xlrd是作为单元格的矩形网格表示。

Excel文件中没有任何信息的单元格由xlrd.XL\_CELL\_EMPTY常数表示。另外，只要有一个空值，用于xlrd后整个值是空串，所以空值单元格应该使用一种Python标识检查。

Excel文件中只有格式信息的单元格由xlrd.XL\_CELL\_BLANK常数表示，它的值总是一个空字符串。

from xlrd import open\_workbook,empty\_cell

print empty\_cell.value

book = open\_workbook('types.xls')

sheet = book.sheet\_by\_index(0)

empty = sheet.cell(6,2)

blank = sheet.cell(7,2)

print empty is blank, empty is empty\_cell, blank is empty\_cell

book = open\_workbook('types.xls',formatting\_info=True)

sheet = book.sheet\_by\_index(0)

empty = sheet.cell(6,2)

blank =sheet.cell(7,2) print empty.ctype,repr(empty.value) print blank.ctype,repr(blank.value)

下面例子展示了以上所有单元格类型一起的使用：

from xlrd import open\_workbook

def cell\_contents(sheet,row\_x):

result = []

for col\_x in range(2,sheet.ncols):

cell = sheet.cell(row\_x,col\_x)

result.append((cell.ctype,cell,cell.value))

return result

sheet = open\_workbook('types.xls').sheet\_by\_index(0)

print 'XL\_CELL\_TEXT',cell\_contents(sheet,1)

print 'XL\_CELL\_NUMBER',cell\_contents(sheet,2)

print 'XL\_CELL\_DATE',cell\_contents(sheet,3)

print 'XL\_CELL\_BOOLEAN',cell\_contents(sheet,4)

print 'XL\_CELL\_ERROR',cell\_contents(sheet,5)

print 'XL\_CELL\_BLANK',cell\_contents(sheet,6)

print 'XL\_CELL\_EMPTY',cell\_contents(sheet,7)

print

sheet = open\_workbook(

'types.xls',formatting\_info=True

).sheet\_by\_index(0)

print 'XL\_CELL\_TEXT',cell\_contents(sheet,1)

print 'XL\_CELL\_NUMBER',cell\_contents(sheet,2)

print 'XL\_CELL\_DATE',cell\_contents(sheet,3)

print 'XL\_CELL\_BOOLEAN',cell\_contents(sheet,4)

print 'XL\_CELL\_ERROR',cell\_contents(sheet,5)

print 'XL\_CELL\_BLANK',cell\_contents(sheet,6)

print 'XL\_CELL\_EMPTY',cell\_contents(sheet,7)

**Names**

这些是很少使用但很强大的抽象方法，常用于查找Excel文件的内部信息。

它们有很多用途，xlrd能从它们之中获取信息。一个值得注意的例外是与sheet和宏命令相关的信息将会被忽略。

Names在Excel中是通过Insert > Name > Define操作创建的。如果你想使用xlrd来从Names中获取信息，在你选择的电子表格应用程序中精通names的定义和运用是一个不错的想法。

**Types 类型**

一个Name可以涉及到：

* 一个常数

CurrentInterestRate = 0.015   
NameOfPHB = “Attila T. Hun”

* 一个单元格的绝对引用

CurrentInterestRate = Sheet1!$B$4

* 一个单元格的1D，2D或3D块的绝对引用

MonthlySalesByRegion = Sheet2:Sheet5!$A$2:$M$100

* 一个绝对引用的列表

Print\_Titles = [row\_header\_ref, col\_header\_ref])

常数可以被获取。

绝对引用的坐标可以被获取，以便你稍后获取相关sheet的对应数据。

相对引用只有当你很熟悉被作为起源使用的单元格时是有用的。Excel文件中包含函数调用在内的公式和多重引用并不是有用的，也太难而无法评估。

xlrd中没有包含全部的计算引擎。

**Scope 范围**

一个Name的Score可以是全局的，或者它只针对特定的sheet。一个Name的标识符在不同Scope内可以被重用。但有多个相同标识符的Name，根据scope使用最合适的一个。一个好例子是内置名为Print\_Area；每个worksheet都可能有它们中的一个。

例：   
name=rate, scope=Sheet1, formula=0.015   
name=rate, scope=Sheet2, formula=0.023   
name=rate, scope=global, formula=0.040

一个单元格公式(1+rate)^20出现在Sheet1等价于1.015^20，出现在Sheet2等价于1.023^20，出现在其他Sheet等价于1.040^20。

**惯例**

使用names的一般原因包括：

* 一个workbook中可能多个地方出现的值设定文本名称。如：RATE = 0.015
* 容易被错误复制的复杂公式设定文本名称。如：SALES\_RESULTS = $A$10:$M$999

这里有个真实世界的案例：向总部报告。一个公司的总部制作了一个模版workbook。每个部门复制一份并填充内容。所有被提供的日期范围都定义了Names。当这些文件传回时，一个脚本用于验证这个部门是否损坏了这个workbook，这个Names用于获取数据来做进一步处理。使用names可以将这些范围解耦和，不管是总部设计模版的用户还是往模版里填充内容的部门用户 从这个脚本都只知道这些范围的names，而不知道具体的范围值。

在xlrd发布的examples目录中你会找到namesdemo.xls，有许多例子，大部分都是针对非苹果系统定义的names。也有个xlrdnamesAPIdemo.py文件展示了如何使用name查找字典，如何获取常数、引用和引用指向的数据。

**格式化**

我们已经看到open\_workbook方法有个参数从Excel文件加载信息。当这步完成，所有格式化信息都是可获得的，但是它是怎么实现的细节不再本书的范围内。

如果你想要复制格式化后的数据到一个新Excel文件中，参看xlutils.copy和xlutils.filter。

如果你想要检测格式化信息，你需要参考下面类的属性：

**xlrd.Book**

colour\_map         font\_list                format\_list        format\_map   
palette\_record   style\_name\_map   xf\_list

**xlrd.sheet.Sheet**

cell\_xf\_index       rowinfo\_map        colinfo\_map       computed\_column\_width   
default\_additional\_space\_above                              default\_additional\_space\_below   
default\_row\_height                                                  default\_row\_height\_mismatch   
default\_row\_hidden                                                 defcolwidth   
gcw                     &

nbsp;                                                     merged\_cells

standard\_width

**xlrd.sheet.Cell**

xf\_index

**Other Classes**

另外，下面类是只用于表示格式化的信息：   
xlrd.sheet.Rowinfo   
xlrd.sheet.Colinfo   
xlrd.formatting.Font   
xlrd.formatting.Format   
xlrd.formatting.XF   
xlrd.formatting.XFAlignment   
xlrd.formatting.XFBackground   
xlrd.formatting.XFBorder   
xlrd.formatting.XFProtection

**操作大的Excel文件**

如果你在操作特别大的Excel文件，那么有两个你应该注意的xlrd特性：

* open\_workbook方法的on\_demand参数为True，被访问时会导致只往内存里加载worksheet。
* xlrd.Book对象有一个unload\_sheet方法能通过指定sheet索引或sheet名称从内存中卸载worksheet。

下面的例子展示了一个大的workbook怎么去迭代被检查只匹配某一模式的sheet，并在内存中某个时间被卸载。

from xlrd import open\_workbook

book = open\_workbook('simple.xls',on\_demand=True)

for name in book.sheet\_names():

if name.endswith('2'):

sheet = book.sheet\_by\_name(name)

print sheet.cell\_value(0,0)

book.unload\_sheet(name)

**用runxlrd.py揭秘Excel文件**

xlrd源码发布包括runxlrd.py脚本，它是非常有用的，不用写单行Python就能揭秘Excel文件。

推荐运行教材提供的各种命令操作Excel文件。

下面是从runxlrd获得的一个预览，使用python runxlrd.py --help能得到：

runxlrd.py [options] command [input-file-patterns]

Commands:

2rows           Print the contents of first and last row in each sheet   
3rows           Print the contents of first, second and last row in each sheet   
bench           Same as "show", but doesn't print -- for profiling   
biff\_count[1]   Print a count of each type of BIFF record in the file   
biff\_dump[1]    Print a dump (char and hex) of the BIFF records in the file   
fonts           hdr + print a dump of all font objects   
hdr             Mini-overview of file (no per-sheet information)   
hotshot         Do a hotshot profile run e.g. ... -f1 hotshot bench bigfile\*.xls   
labels          Dump of sheet.col\_label\_ranges and ...row... for each sheet   
name\_dump       Dump of each object in book.name\_obj\_list   
names           Print brief information for each NAME record   
ov              Overview of file   
profile         Like "hotshot", but uses cProfile   
show            Print the contents of all rows in each sheet   
version[0]      Print versions of xlrd and Python and exit   
xfc             Print "XF counts" and cell-type counts -- see code for details

[0] means no file arg   
[1] means only one file arg i.e. no glob.glob pattern

Options:   
  -h, --help            show this help message and exit   
  -l LOGFILENAME, --logfilename=LOGFILENAME   
                        contains error messages   
  -v VERBOSITY, --verbosity=VERBOSITY   
                        level of information and diagnostics provided   
  -p PICKLEABLE, --pickleable=PICKLEABLE   
                        1: ensure Book object is pickleable (default); 0:   
                        don't bother   
  -m MMAP, --mmap=MMAP  1: use mmap; 0: don't use mmap; -1: accept heuristic   
  -e ENCODING, --encoding=ENCODING   
                        encoding override

sp; -f FORMATTING, --formatting=FORMATTING   
                        0 (default): no fmt info 1: fmt info (all cells)   
  -g GC, --gc=GC        0: auto gc enabled; 1: auto gc disabled, manual   
                        collect after each file; 2: no gc   
  -s ONESHEET, --onesheet=ONESHEET   
                        restrict output to this sheet (name or index)   
  -u, --unnumbered      omit line numbers or offsets in biff\_dump