



Python 操作 Excel 学习笔记

完美 Excel 出品
微信公众号: [excelperfect](#)



内容提要

随着大数据、人工智能的兴起，Python 日益火爆。在 Python 中，我们可以利用第三方库来方便地操作 Excel 文件。

本电子书《Python 操作 Excel 学习笔记》详细介绍了利用 openpyxl 库来操作 Excel 的一些基础代码，感兴趣的朋友可以跟着学习，通过 Excel 来学习和了解 Python，为自己开辟一片新天地。



目录

1. 安装用于操作 Excel 文件的 Python 库.....	1
安装.....	1
pip 命令简介	3
2. 初识操作代码	5
代码.....	5
参考：如何安装 Python IDE	7
3. 在内存中操控工作簿.....	11
创建工作簿.....	11
操作数据.....	13
保存文件.....	15
装载文件.....	16
4. 基础代码之读取 Excel 文件中的数据.....	17
关于 openpyxl.....	17
基础代码：读取 Excel 文件中的数据	18
5. 基础代码之向 Excel 文件中写入数据.....	21
基础代码：创建和保存文件.....	21
基础代码：装载已有 Excel 文件并保存修改	22
基础代码：添加工作表.....	23



基础代码：移除工作表.....	23
基础代码：在单元格中写入值.....	23
基础代码：设置字体.....	24
基础代码：输入公式.....	25
基础代码：调整行高列宽.....	25
基础代码：合并单元格和取消合并单元格.....	26
6. 基础代码之图表操作.....	29
引用对象说明.....	29
示例代码：使用 Python 创建图表.....	30
7. 简单的应用示例.....	31
在工作簿中写入数据.....	31
读取现有工作簿中的数据.....	34
8. 简单的应用示例（续）.....	35
使用数字格式.....	35
使用公式.....	36
合并/取消合并单元格.....	37
插入图像.....	37
折叠列（大纲）.....	38
9. 图表操作.....	39
图表类型.....	39
创建图表.....	39
10. 图表坐标轴.....	43
图表坐标轴.....	43
添加次要坐标轴.....	47



11. 图表布局	49
图表布局.....	49
图例布局.....	50
示例.....	50
12. 面积图	53
二维面积图.....	53
三维面积图.....	55
13. 条形图和柱状图	57
垂直、水平和堆积条形图.....	57
三维条形图.....	60
14. 气泡图	63
15. 折线图	65
三维折线图.....	69
16. 散点图	71
17. 饼图	73
复合饼图.....	76
三维饼图.....	77
渐变饼图.....	78
18. 圆环图	79
19. 雷达图	83
20. 股价图	87
21. 曲面图	93
22. 添加样式	97
23. 绘制仪表盘	99



24.图表工作表	103
25.给单元格添加批注	105
26.处理样式	109
单元格样式.....	110
复制样式.....	111
基本的字体颜色.....	112
应用样式.....	112
样式化合并单元格.....	113
编辑页面设置.....	115
命名样式.....	116
使用内置样式.....	116
27.一些工作表属性	119
工作表的可用属性.....	119
页面设置可用字段.....	120
大纲可用字段.....	120
28.条件格式	123
内置格式.....	123
标准条件格式.....	126
示例.....	127
29.打印设置	131
编辑打印选项.....	131
页眉和页脚.....	131
添加打印标题.....	132
添加打印区域.....	133



30.筛选和排序	135
31.数据有效性	137
32.表	143
33.保护工作簿和工作表	145
保护工作簿.....	145
保护工作表.....	146
关于完美 Excel	147
完美 Excel 微信公众号使用指南	148



温馨提示：

在完美 Excel 微信公众号中发送消息：Python 和 Excel，即可获得本电子书文档下载链接和密码。



如果您对本书有什么建议或者还有什么好的示例,欢迎前往完美 Excel 微信公众号沟通交流。

欢迎分享本电子书，让更多的人方便地得到所需要的知识。



本章内容 2018 年 7 月 2 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记(1): 安装
用于操作 Excel 文件的 Python 库

1. 安装用于操作 Excel 文件的 Python 库

在 Python 中, 我们可以利用第三方库来方便地操作 Excel 文件。这里使用 `python-excel.org` 中提供的 `openpyxl` 库, 用于读写 Excel 2007 及以上版本的 Excel 文件。

可以通过 `http://www.python-excel.org/` 页面进入 `openpyxl` 库的下载页面, 也可以进入 `openpyxl` 库的使用介绍页面。

安装

使用 `pip` 安装工具来安装 `openpyxl`, 安装需要打开系统命令行。

如果是 Windows 操作系统, 则单击左下角的图标, 在底部的搜索框中输入命令:

`cmd`

如图 1.1 所示。

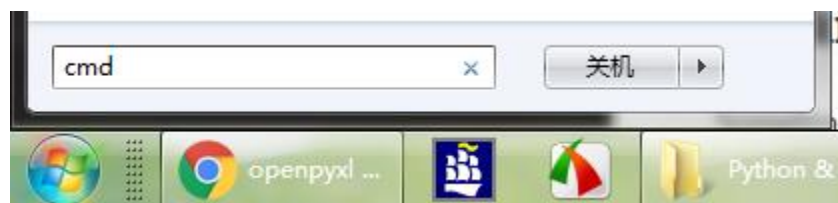


图 1.1

打开 Windows 命令行, 如图 1.2 所示。





图 1.2

输入:

```
pip install openpyxl
```

会出现连接、下载及安装 openpyxl 的一系列信息, 结果如图 1.3 所示。

```
C:\Users\Administrator>pip install openpyxl
Collecting openpyxl
  Downloading https://files.pythonhosted.org/packages/f6/13/3c1263b852377738eaa60f99602fb58cc8ad2fd1badb0b724b0d5b532727/openpyxl-2.5.4.tar.gz (170kB)
    41% |#####| 71kB 101kB/s eta 0:00:0
    47% |#####| 81kB 116kB/s eta 0:00
    53% |#####| 92kB 105kB/s eta 0:
    59% |#####| 102kB 105kB/s eta
    65% |#####| 112kB 109kB/s et
    71% |#####| 122kB 119kB/s
    77% |#####| 133kB 192kB/
    83% |#####| 143kB 192k
    89% |#####| 153kB 15
    95% |#####| 163kB
   100% |#####| 174k
B 152kB/s
Collecting jdcal (from openpyxl)
  Downloading https://files.pythonhosted.org/packages/a0/38/dcf83532480f25284f3ef13f8ed63e03c58a65c9d3ba2a6a894ed9497207/jdcal-1.4-py2.py3-none-any.whl
Collecting et_xmlfile (from openpyxl)
  Downloading https://files.pythonhosted.org/packages/22/28/a99c42aea746e18382ad9fb36f64c1c1f04216f41797f2f0fa567da11388/et_xmlfile-1.0.1.tar.gz
Installing collected packages: jdcal, et-xmlfile, openpyxl
  Running setup.py install for et-xmlfile ... done
  Running setup.py install for openpyxl ... done
Successfully installed et-xmlfile-1.0.1 jdcal-1.4 openpyxl-2.5.4
```

图 1.3

在图 1.3 的最后一行, 我们可以看到安装成功的信息。如果没有安装成功, 可能是网络问题, 多试几次; 也可能是其他原因, 可以使用集成或文件方法来安装。



pip 命令简介

pip 方法是最主要且简便的安装 python 第三方库的方法，因此要安装 Python 第三方库，首选 pip 方法。下面列出了常用的 pip 命令。

安装指定的第三方库：

```
pip install <第三方库名>
```

使用 -U 参数更新已安装的第三方库：

```
pip install -U <第三方库名>
```

卸载指定的第三方库：

```
pip uninstall <第三方库名>
```

下载但不安装指定的第三方库：

```
pip download <第三方库名>
```

列出指定的第三方库的详细信息：

```
pip show <第三方库名>
```

根据关键词在名称和介绍中搜索第三方库：

```
pip search <关键词>
```

列出当前系统已经安装的第三方库：

```
pip list
```





本章内容 2018 年 7 月 4 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (2): 初识
操作代码

2.初识操作代码

在上一章中，我们介绍并安装了用于操作 Excel 的 openpyxl 库。下面，通过一个简单的示例来初步认识一下用于操作 Excel 的 Python 代码。

代码

下面的代码用于创建 Excel 工作簿并在工作表中写入数据：

```
from openpyxl import Workbook
wb=Workbook()

# 获取活动工作表
ws=wb.active

# 直接将数据输入到指定单元格
ws['A1']="完美 Excel"

# 在最后数据行后一次添加多个数据
ws.append([1,2,3])

# 在指定单元格中输入当前时间
import datetime
ws['A3']=datetime.datetime.now()
```



```
# 在当前文件夹中保存文件
wb.save("sample.xlsx")
```

在代码中，我们已经使用注释解释了大部分语句的作用。其中，`import` 来完成库引用，如

```
import datetime
```

其通用形式为：

```
import <库名>
```

然后，使用：

```
<库名>.<函数名>(<函数参数>)
```

来使用库。如代码：

```
ws['A3']=datetime.datetime.now()
```

也可以使用 `from` 和 `import` 来共同完成库引用，如代码开头的

```
from openpyxl import Workbook
```

其通用形式为：

```
from <库名> import <函数名>
```

然后，在之后的代码中，可以使用：

```
<函数名>(<函数参数>)
```

来使用库。如代码：

```
wb=Workbook()
```

运行代码后，将在当前目录下创建一个名为 `sample.xlsx` 的工作簿，其内容如下图所示。

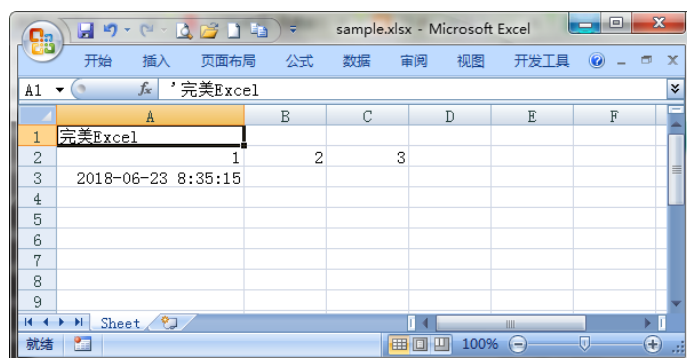


图 2.1



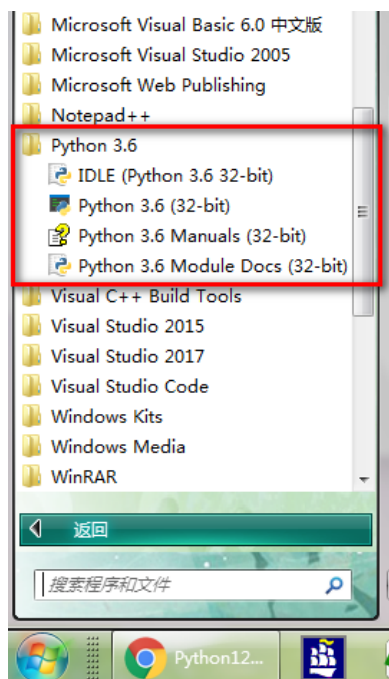


图 2.4

单击图 2.4 中的 Python 3.6 (32-bit) 进入命名行模式，如图 2.5 所示。

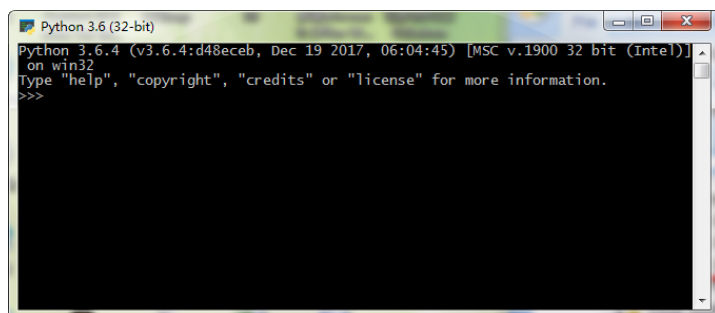


图 2.5

单击图 2.4 中的 IDLE (Python 3.6 32-bit)，进入 Python 自带的简洁的集成开发环境，如图 2.6 所示。

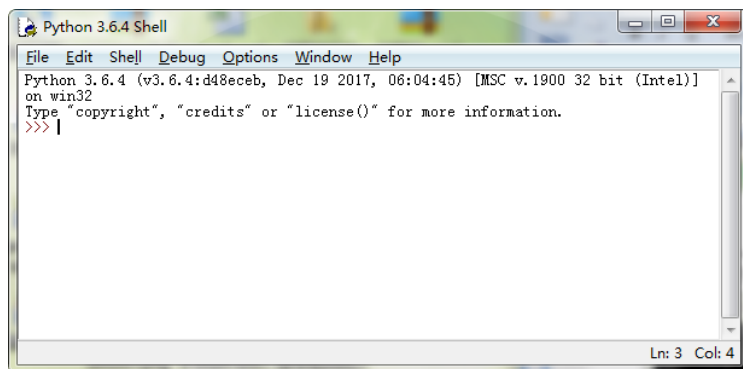


图 2.6



单击图 2.6 中的 File——New File，则可以创建新的代码文件。也可以单击 File——Open，打开已有的文件，如图 2.7 所示。

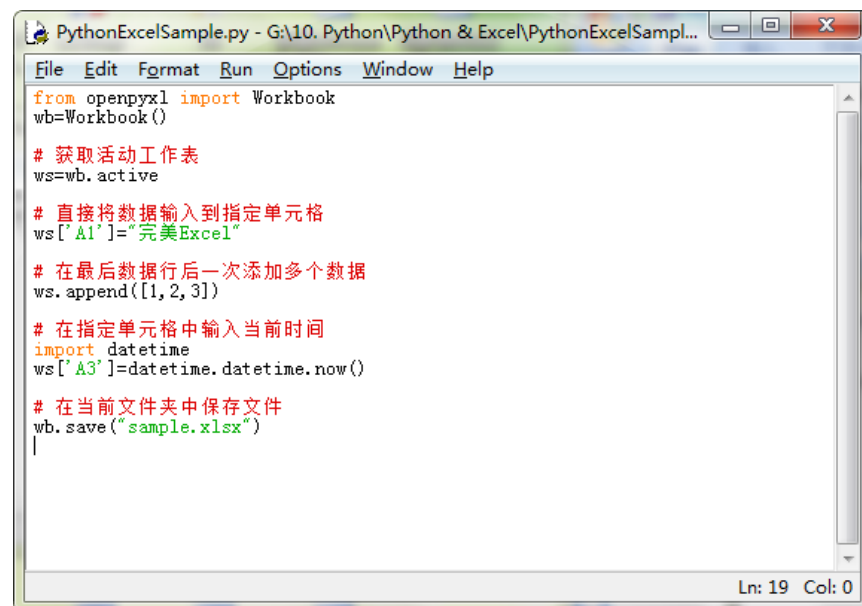


图 2.7

其中的 Run 菜单下，可以运行已输入的代码。





本章内容 2018 年 7 月 9 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (3): 在内存中
操控工作簿

3. 在内存中操控工作簿

创建工作簿

没有必要在文件系统上创建一个文件来开始使用 `openpyxl`，只需要导入 `Workbook` 类并开始使用它：

```
from openpyxl import Workbook  
wb=Workbook()
```

创建的工作簿总是至少含有一个工作表，可以使用：

```
openpyxl.workbook.Workbook.active()
```

属性来获取工作表。

相应的语句：

```
ws=wb.active
```

也可以使用：

```
openpyxl.workbook.Workbook.create_sheet()
```

方法创建新工作表。

相应的语句：

```
ws1=wb.create_sheet("Mysheet") #插入到末尾（缺省）  
#或者
```



```
ws2=wb.create_sheet("Mysheet",0) #插入到第 1 个位置
```

在创建时，自动给工作表提供一个名字，按顺序编号（Sheet, Sheet1, Sheet2, ...），可以在任何时候使用 title 属性修改名字：

```
ws.title = "新标题"
```

缺省情况下包含标题名的选项卡的背景色是白色，通过提供 RRGGBB 颜色代码给 sheet_properties.tabColor 属性来修改背景色。

```
ws.sheet_properties.tabColor = "1072BA"
```

一旦提供给工作表名字，就可以将其作为工作簿的键：

```
ws3 = wb["新标题"]
```

可以使用：

```
openpyxl.workbook.Workbook.sheetnames()
```

属性查看工作簿的所有工作表的名称。

```
print(wb.sheetname)
['Sheet2', '新标题', 'Sheet1']
```

遍历工作表：

```
for sheet in wb:
    print(sheet.title)
```

可以创建单个工作簿里工作表的副本：

```
openpyxl.workbook.Workbook.copy_worksheet()
```

相应的代码：

```
source = wb.active
target = wb.copy_worksheet(source)
```

注意，仅仅单元格（包括值、样式、链接和批注）和某些工作表属性（包括尺寸、



格式和属性)被复制,所有其他工作簿/工作表属性不被复制,例如图片、图表。
不能在工作簿间复制工作表,如果工作簿以只读或只写模式打开,也不能复制工作表。

操作数据

访问单元格

上文中,我们已经知道了如何访问工作表,下面我们开始修改单元格内容。

单元格可以作为工作表的键来直接访问:

```
c = ws['A4']
```

返回 A4 处的单元格,或者如果不存在则创建一个。值可以直接赋值。

```
ws['A4'] = 4
```

也可以使用:

```
openpyxl.worksheet.Worksheet.cell()
```

使用行和列表示来访问单元格:

```
d = ws.cell(row=4, column=2, value=10)
```

注意,在内存中创建工作表时,不包含单元格,它们在首次访问时才被创建。因此,滚动浏览单元格而不是直接访问会在内存中创建它们,即使没有给它们赋值。
例如:

```
for i in range(1,101):  
    for j in range(1,101):  
        ws.cell(row=i, column=j)
```

将在内存中创建 100×100 个单元格。

访问多个单元格

使用:

```
cell_range = ws['A1':'C2']
```

同样可以获得行区域或列区域:



```
colC = ws['C']
col_range = ws['C:D']
row10 = ws[10]
row_range = ws[5:10]
```

也可以使用：

```
openpyxl.worksheet.Worksheet.iter_rows()
```

相应的语句：

```
for row in ws.iter_rows(min_row=1, max_col=3, max_row=2):
    for cell in row:
        print(cell)
```

同样地，方法：

```
openpyxl.worksheet.Worksheet.iter_cols()
```

返回列。

相应的语句：

```
for col in ws.iter_cols(min_row=1, max_col=3, max_row=2):
    for cell in col:
        print(cell)
```

如果需要遍历文件的所有行或列，可以使用：

```
openpyxl.worksheet.Worksheet.rows()
```

相应的语句：

```
ws = wb.active
ws['C9'] = 'Hello World'
tuple(ws.rows)
```

或者使用：

```
openpyxl.worksheet.Worksheet.columns()
```



相应的语句：

```
tuple(ws.columns)
```

数据存储

一旦有 `openpyxl.cell.Cell`，则可以给它赋值：

```
c.value = 'Hello, world'
print(c.value)
```

```
d.value = 3.14
print(d.value)
```

还可以启用类型和格式：

```
wb = Workbook(guess_types=True)
c.value = '12%'
print(c.value)
```

```
import datetime
d.value = datetime.datetime.now()
print d.value
```

```
c.value = '31.50'
print(c.value)
```

保存文件

保存工作簿最简单和最安全的办法是使用 `openpyxl.workbook.Workbook` 对象的

`openpyxl.workbook.Workbook.save()`

方法。



相应的语句：

```
wb = Workbook()  
wb.save('balances.xlsx')
```

注意，该操作将覆盖已有文件而不会给出警告。

可以指定属性 *template=True*，将工作簿保存为模板：

```
wb = load_workbook('document.xlsx')  
wb.template = True  
wb.save('document_template.xltx')
```

或者设置该属性为 *False* (默认)，保存为文档：

```
wb = load_workbook('document_template.xltx')  
wb.template = False  
wb.save('document.xlsx', as_template=False)
```

装载文件

与写入相同，可以导入 `openpyxl.load_workbook()` 来打开已有的工作簿：

```
from openpyxl import load_workbook  
wb2 = load_workbook('test.xlsx')  
print wb2.get_sheet_names()
```



本章内容 2018 年 7 月 12 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记\(4\): 基础代码
之读取 Excel 文件中的数据](#)

4. 基础代码之读取 Excel 文件中的数据

在 Microsoft Excel 中做的所有事情都可以使用 Python 自动完成，因此，为什么不使用 Python 强大的功能让你更轻松呢？你可以制作智能化的电子表格，将 Python 的逻辑和思考能力带到通常是静态的 Excel，从而为 Excel 带来灵活性和大量机会。

通过前面的学习，我们初步了解了 Python 操作 Excel 的方法。下面，我们先介绍用于操作 Excel 的 Python 库 openpyxl，然后列出一些基本代码并进行讲解。

关于 openpyxl

openpyxl 是一个 Python 库，由 Eric Gazoni 和 Charlie Clark 开发，用来读写 Excel 的 xlsx/xlsm/xltx/xltm 文件而无须使用 Excel 软件。它是最广泛使用的用于 python-excel 操作的库。

openpyxl 是 Python Pandas 的默认读取器。

Excel 是电子表格中非常强大且最受欢迎的软件，openpyxl 帮助通过 python 编程来读取和修改 Excel 电子表格文件。有时，你必须将某些数据从一个电子表格复制到另一个电子表格。你可能必须读取十几个 Excel 文件，对特定的数据块排序，然后以特定的格式写入。如果你通过读取每个文件、排序并选择、复制特定的数据来执行这样的操作，那么可能需要几个小时。如果有成百上千的文件需要处理，则可能需要几天甚至几周。但是，在 Python 的帮助下，你只需花上几分钟编写一个简单的程序，这些枯燥无聊的工作会很快被计算机完成。这个程序可以读取成百上千个文件，搜索特定的数据块，以特定的顺序排列，然后以所需格式写入新的 Excel 文件。所有这一切都可以通过 openpyxl 来完成。



基础代码：读取 Excel 文件中的数据

下面的代码用于导入 openpyxl:

```
import openpyxl
```

装载 Excel 文件。下面的代码用于装载名为 test.xlsx 的文件:

```
wb=openpyxl.load_workbook('test.xlsx')
```

openpyxl.load_workbook() 是一个函数，接受一个文件名作为参数，返回 workbook 数据类型。输入:

```
type(wb)
```

在 Python Shell 中(下同)返回的信息如下:

```
<class 'openpyxl.workbook.workbook.Workbook'>
```

要想获取工作簿中工作表的数量及其名称的信息，可以使用函数:

```
wb.get_sheet_names()
```

或者:

```
wb.sheetnames
```

返回:

```
['Sheet1','Sheet2','Sheet3']
```

知道工作表的名字后，可以同时访问任何工作表。假设想要访问工作表 Sheet3，代码如下:

```
sheet=wb['Sheet3']
```

这种方法用于访问指定的工作表并将其存储在变量中，本例中为 sheet。

代码:

```
sheet.title
```

返回变量代表的工作表名，本例中为 'Sheet3'。

下面的代码:

```
wb.active
```

指向当前活动工作表。



下面的代码：

```
sheet['A2'].value
```

从指定的工作表返回指定的单元格中的数据。

另一种访问指定工作表中单元格数据的方法：

```
e=sheet['A2']  
e.value
```

此时，代码：

```
e.row
```

返回行 2。

```
e.column
```

返回列字母 A。

下面的代码：

```
sheet.cell(row=2,column=4)
```

返回指定工作表中的单元格 D2。

```
sheet.cell(row=2,column=4).value
```

返回单元格 D2 中的数据值。

也可以打印出整个列中的数据，代码如下：

```
for x in range(1,5):  
    print(x,sheet.cell(row=x,column=4).value)
```

例如示例工作表如图 4.1 所示。

	A	B	C	D	E	F	G	
1	时间	区域	名字	物品	数量	单价	合价	
2	2018/7/6 5:50 AM	A区	赵鲁	铅笔	15	2.99	44.85	
3	2018/5/23 2:25 PM	B区	钱三多	活页夹	20	21.98	439.6	
4	2018/6/9 5:45 AM	C区	马跑跑	铅笔	25	5.02	125.5	
5								

图 4.1



运行上述代码后的结果如图 4.2 所示。

```
>>> for x in range(1,5):  
...     print(x, sheet.cell(row=x, column=4).value)  
...  
1 物品  
2 铅笔  
3 活页夹  
4 铅笔
```

图 4.2

同样，可以打印多个列的值：

```
for y in range(1,5,1):  
    print(sheet.cell(row=y, column=1).value, sheet.cell(row=  
y, column=2).value, sheet.cell(row=y, column=3).value, sheet.  
cell(row=y, column=3).value, sheet.cell(row=y, column=4).val  
ue, sheet.cell(row=y, column=5).value,  
sheet.cell(row=y, column=6).value,  
sheet.cell(row=y, column=7).value)
```

在上面的代码中，我们使用了 for 循环来遍历行，使用了 range 函数生成列表序列。



本章内容 2018 年 7 月 16 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(5\): 基础代码
之向 Excel 文件中写入数据](#)

5. 基础代码之向 Excel 文件中写入数据

在 openpyxl 模块的帮助下, 我们也可以使用 Python 向 Excel 文件写入数据, 其过程与使用 Python 读取 Excel 电子表格相似。在本文中, 我们使用 Python Excel writer, 创建 Excel 工作表, 并在单元格中写入文本、数字和公式。在修改完成后, 保存工作簿。在 Excel 工作簿中添加和删除工作表, 应用设置、字体和样式, 设置单元格和区域的宽和高, 合并和取消合并单元格。我们可以创建任何类型的 Excel 文件, 带有成百上千的数据行和列。

基础代码: 创建和保存文件

首先要导入 openpyxl 模块:

```
import openpyxl
```

接着, 创建 Excel 文件或工作簿:

```
mywb=openpyxl.Workbook()
```

代码将创建含有一个工作表的工作簿。

然后, 可以使用代码来核查工作表的数量, 看看哪个工作表是当前工作表, 当前工作表的标题是什么。

```
mywb.get_sheet_names()
```

或者:

```
mywb.sheetnames
```

本例中, 由于只有一个工作表, 因此返回的结果是:

```
['Sheet']
```



将当前工作表赋给变量 sheet:

```
sheet=mywb.active
```

获取工作表的名字:

```
sheet.title
```

自定义工作表的名字:

```
sheet.title='MySheet'
```

保存工作簿:

```
mywb.save('MyExcelFile.xlsx')
```

此时, 如果打开根目录或当前目录, 将会发现生成了一个名为 MyExcelFile 的新 Excel 文件, 包含一个名为 MySheet 的工作表。

基础代码：装载已有 Excel 文件并保存修改

无论何时装载已有的 Excel 文件, 修改其内容, 无论是工作表还是单元格, 或者创建整个工作表, 或者删除工作表, 或者对其进行任何修改, 都必须调用 save() 方法来保存文件。否则, 任何修改都会丢失。

```
import openpyxl
mywb=openpyxl.load_workbook('MyExcelFile.xlsx')
sheet=mywb.active
sheet.title='ModifySheet'
mywb.save('MyExcelFileCopy.xlsx')
```

上面的代码中, 先装载已有的文件, 修改其工作表名并使用不同的名字来保存该 Excel 文件。此时, 在工作目录中, 除原来的 MyExcelFile 文件外, 将多一个新的名为 MyExcelFileCopy 的工作簿文件。

在使用 Python 操作 Excel 文件时, 建议最好以不同的文件名保存所要操作的文件, 这样即使发生误操作, 也不会影响原文件。



基础代码：添加工作表

要在工作簿中添加新的工作表，使用 `create_sheet()` 方法：

```
mywb.create_sheet()
```

此时，将添加一个默认名为 `Sheet1` 的工作表。

在添加新工作表时，默认将其添加到最后一个位置。然而，我们可以使用索引值指定新添加的工作表位置，也可以给新添加的工作表命名。代码：

```
wb.create_sheet(index=0,title='1st Sheet')
```

添加一个名为 `1st Sheet` 的工作表并放置在第 1 个位置。注意，第 1 个工作表的索引值为 0，第 2 个工作表的索引值为 1，依此类推。

基础代码：移除工作表

要想从工作簿中移除指定的工作表，使用 `remove()` 方法。

```
mywb.remove(mywb['Sheet1'])
```

删除工作表 `Sheet1`。

还可以使用 `Del` 语句：

```
Del mywb['1st Sheet']
```

删除工作表 `1st Sheet`。

基础代码：在单元格中写入值

下面介绍如何写入值到工作表特定的单元格中。下面的代码在工作表 `Sheet` 中的单元格 `A1` 中输入值：

```
mysheet=mywb['Sheet']  
mysheet['A1']='Hello, World!'
```



还可以指定行列号来确定单元格，然后写入值：

```
mycell=mysheet.cell(row=1,column=1)
mycell='Hello, World!'
```

使用循环来在一系列单元格中输入值：

```
for i in range(5,15):
    mycell=mysheet.cell(row=i,column=2)
    mycell.value=mylist[i]
```

在 B 列的单元格 B5:B14 中输入列表 mylist 中的值。

基础代码：设置字体

通常，我们会在工作表中应用不同的样式，来强调或突出特定的行或列。在工作表中应用统一的样式是非常重要的，这样可以清晰地读取数据。如果有成百上千行数据，应用样式可能是一项繁琐的工作，然而使用 Python 代码，可以编写几行代码并立即将样式应用于数百成行数据。但是要小心，首先应该使用绝对样式，然后总是要用不同的名称保存工作簿。

下面的代码首先导入 openpyxl，然后导入字体供我们使用。

```
>>> import openpyxl
>>> from openpyxl.styles import Font
>>> mywb = openpyxl.Workbook()
>>> mysheet = mywb['Sheet']
>>> italic32Font = Font(size = 32, italic = True)
>>> mysheet['A1'].font = italic32Font
>>> mysheet['A1'] = 'Hello, world!'
>>> mywb.save('styletest.xlsx')
```

代码设置单元格 A1 中的字体大小为 36 磅、斜体。结果如下图 5.1 所示。

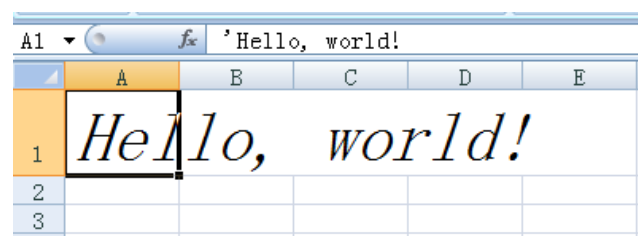


图 5.1



Font 对象有 4 个参数，分别为：

- Name: 字符串值，设置字体名称，例如 “Arial”
- Size: 整数值，设置字体大小。
- Bold: 布尔值，设置是否加粗字体，为 True 则加粗。
- Italic: 布尔值，设置是否为斜体，为 True 则为斜体。

基础代码：输入公式

公式是 Excel 中一个非常重要的功能。openpyxl 提供了在任何特定单元格中编写公式的能力，事实上，非常容易，输入一个等号和所需的公式。

下面的代码：

```
>>> import openpyxl
>>> mywb = openpyxl.Workbook()
>>> mysheet = mywb.active
>>> mysheet['A1'] = 1
>>> mysheet['A2'] = 2
>>> mysheet['A3'] = '=SUM(A1:A2)'
>>> mywb.save('formulatest.xlsx')
```

在单元格 A1 中输入 1，在单元格 A2 中输入 2，在单元格 A3 中放置一个公式来求单元格 A1 和 A2 的和。结果如下图 5.2 所示。

A3		fx =SUM(A1:A2)			
	A	B	C	D	
1	1				
2	2				
3	3				
4					

图 5.2

基础代码：调整行高列宽

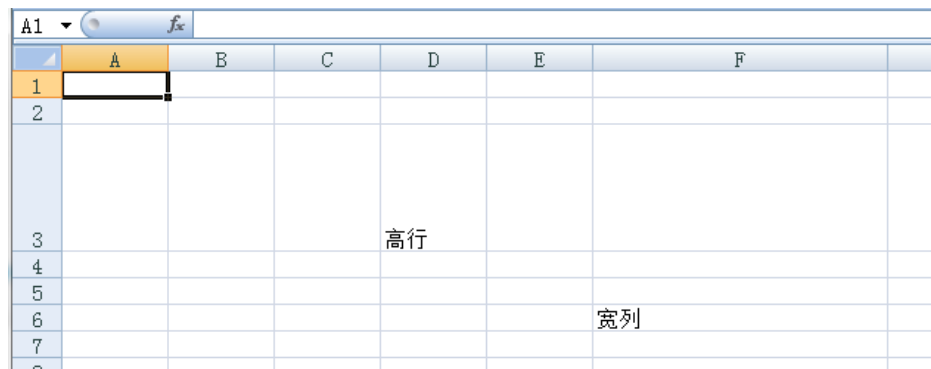
使用 openpyxl 可以设置工作表的行高和列宽，也可以冻结窗格以便一些行或



列总是可见，还可以隐藏行或列。代码：

```
>>> import openpyxl
>>> mywb = openpyxl.Workbook()
>>> mysheet = mywb.active
>>> mysheet['D3'] = '高行'
>>> mysheet['F6'] = '宽列'
>>> mysheet.row_dimensions[3].height = 65
>>> mysheet.column_dimensions['F'].width = 25
>>> mywb.save('RowHeightColWidth.xlsx')
```

调整第 3 行的行高和 F 列的列宽。结果如下图 5.3 所示。



	A	B	C	D	E	F
1						
2						
3				高行		
4						
5						
6						宽列
7						

图 5.3

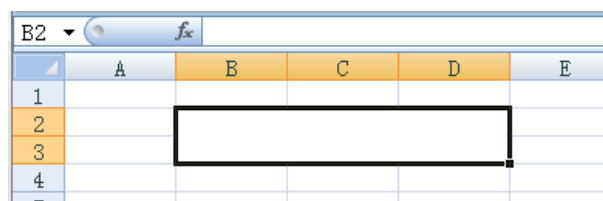
在 Excel 中默认的行高是 12.75 磅。这里，1 磅等于七十二分之一英寸，可以设置 0 至 409 之间的值。列宽可以设置为 0 至 255 之间的值，可以是整数或者浮点数。如果将列宽设置为 0，或者将行高设置为 0，则隐藏列或行。

基础代码：合并单元格和取消合并单元格

openpyxl 允许合并和取消合并单元格，代码：

```
>>> import openpyxl
>>> mywb = openpyxl.Workbook()
>>> mysheet = mywb.active
>>> mysheet.merge_cells('B2:D3')
>>> mywb.save('MergeCells.xlsx')
```

将工作表中的单元格区域 B2:D3 合并，结果如下图 5.4 所示。



	A	B	C	D	E
1					
2					
3					
4					

图 5.4



`merge_cells` 方法接受两个单元格地址作为其参数，第 1 个单元格是要合并区域的左上角单元格，第 2 个是右下角单元格。如果想要获取或设置合并后的单元格区域的值，那么使用整个合并区域左上角单元格的地址。

使用 `unmerge_cells` 方法取消合并单元格，代码如下：

```
>>> import openpyxl
>>> mywb = openpyxl.load_workbook('MergeCells.xlsx')
>>> mysheet = mywb.active
>>> mysheet.unmerge_cells('B2:D3')
>>> mywb.save('unmergecells.xlsx')
```





本章内容 2018 年 7 月 22 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记\(6\): 基础代码之图表操作](#)

6. 基础代码之图表操作

在 openpyxl 模块的帮助下，我们可以使用 Python 创建 Excel 图表。openpyxl 模块支持创建所有主要类型的图表，诸如条形图、折线图、散点图、饼图。

使用 openpyxl 创建 Excel 图表的步骤如下：

1. 使用矩形单元格区域，必须创建一个引用对象。
2. 传递这个引用对象，创建一个 Series 对象。
3. 创建图表对象。
4. 将创建的 Series 对象附加到所创建的图表对象中。
5. 设置图表对象的顶、左、宽、高。
6. 将图表对象添加到 Worksheet 对象。

引用对象说明

在使用 openpyxl 创建引用对象时，必须调用指定的函数：

```
openpyxl.chart.Reference()
```

该函数需要 5 个参数：

- 包含图表数据的 Worksheet 对象
- 后面 4 个参数分别指定数据区域左上角和右下角单元格所在位置。



示例代码：使用 Python 创建图表

下面的代码：

```
>>> import openpyxl
>>> chartwb = openpyxl.Workbook()
>>> mysheet = chartwb.active
>>> for x in range(1,10):
...     mysheet['A' + str(x)] = x
...
>>> refobj = openpyxl.chart.Reference(mysheet,min_row=1,min_col=1,max_row=10,max_col=1)
>>> seriesobj = openpyxl.chart.Series(refobj,title='First Series')
>>> chartobj = openpyxl.chart.BarChart()
>>> chartobj.title = 'My Chart'
>>> chartobj.append(seriesobj)
>>> mysheet.add_chart(chartobj)
>>> chartwb.save('examplechart.xlsx')
```

创建一个柱状图，如下图 6.1 所示。

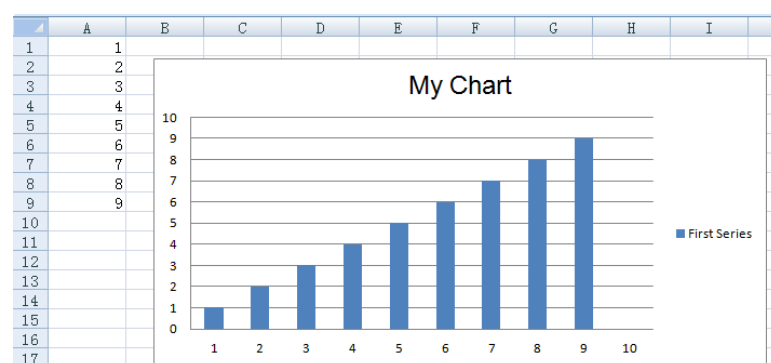


图 6.1

同样，代码：

```
openpyxl.chart.LineChart()
```

创建折线图。

代码：

```
openpyxl.chart.ScatterChart()
```

创建散点图。

代码：

```
openpyxl.chart.PieChart()
```

创建饼图。



本章内容 2018 年 7 月 27 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(7\): 简单的应用示例](#)

7. 简单的应用示例

下面让我们结合示例代码来理解 Python 对 Excel 的一些常用操作。

在工作簿中写入数据

下面的代码创建工作簿并添加工作表，在相应的工作表中写入数据：

```
from openpyxl import Workbook
from openpyxl.compat import range
from openpyxl.utils import get_column_letter

wb=Workbook()

dest_filename='empty_book.xlsx'

ws1=wb.active
ws1.title="range names"

for row in range(1,40):
    ws1.append(range(600))

ws2=wb.create_sheet(title="Pi")

ws2['F5']=3.14
```



```

ws3=wb.create_sheet(title="Data")

for row in range(10,20):
    for col in range(27,54):

        _=ws3.cell(column=col,row=row,value="{0}".format(get_colu
mn_letter(col)))
print(ws3['AA10'].value)

wb.save(filename=dest_filename)

```

代码中：

- 开始的 3 句从 openpyxl 库中导入相关的类，以便于后面的代码能够使用它们。
- 然后创建工作簿并将其命名为 empty_book.xlsx。
- 接着获取其中的工作表并将其命名为“range names”。
- 使用循环语句在工作表的第 1 行至第 39 行写入数字 0 至 599。其中，range 函数生成指定范围的一组数字序列。结果如图 7.1 所示。

	A	B	C	D	E	F	G
1	0	1	2	3	4	5	6
2	0	1	2	3	4	5	6
3	0	1	2	3	4	5	6
4	0	1	2	3	4	5	6
5	0	1	2	3	4	5	6
6	0	1	2	3	4	5	6
7	0	1	2	3	4	5	6
8	0	1	2	3	4	5	6
9	0	1	2	3	4	5	6
10	0	1	2	3	4	5	6
11	0	1	2	3	4	5	6
12	0	1	2	3	4	5	6
13	0	1	2	3	4	5	6

图 7.1



- 接着,添加一个名为“Pi”的工作表,在工作表的单元格 F5 中写入数字 3.14,如图 7.2 所示。

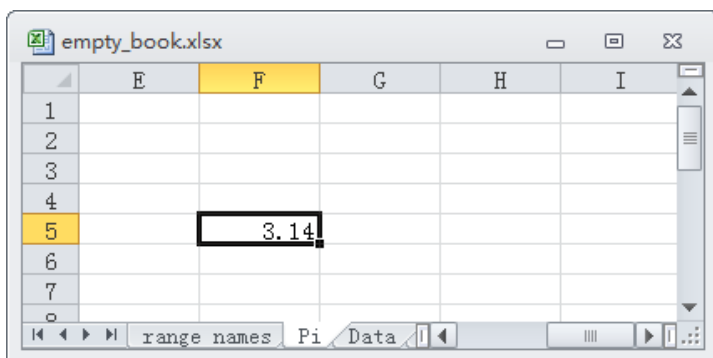


图 7.2

- 然后,再添加一个名为“Data”的工作表,在该工作表的第 10 行至第 19 行,第 27 列(即 AA 列)至 53 列(即 BA 列)的范围内,输入单元格所在的列字母,如图 7.3 所示。

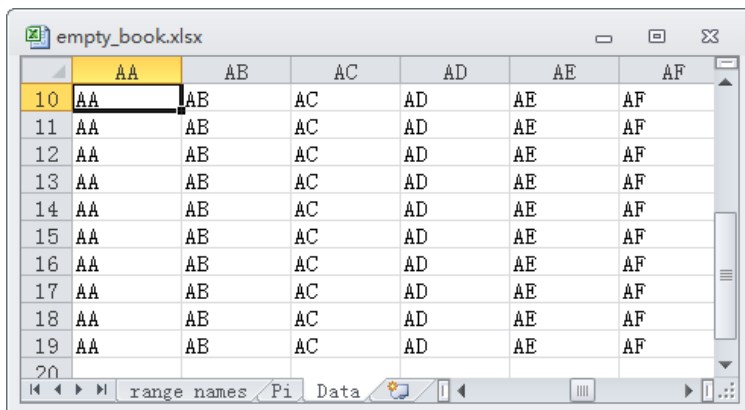


图 7.3

- 打印工作表 Data 中单元格 AA10 中的数据,即列字母 AA。
- 最后,以指定的名称保存工作簿。



读取现有工作簿中的数据

下面的代码读取上文创建的工作簿 `empty_book.xlsx` 中工作表 “range names” 的单元格 D18 的数据：

```
from openpyxl import load_workbook
wb=load_workbook(filename='empty_book.xlsx')
sheet_ranges=wb['range names']
print(sheet_ranges['D18'].value)
```

代码从 `openpyxl` 库中导入 `load_workbook` 类，装载名为 “empty_book.xlsx” 的工作簿，从该工作簿的工作表 range names 中取单元格 D18 的值。

注意，在 `load_workbook` 中可以使用几个标志：

- 当读取单元格时，`guess_types` 启用或禁用（缺省）类型接口。
- `data_only` 控制具有公式的单元格是公式（缺省）还是上次 Excel 读取工作表时存储的值。
- `keep_vba` 控制是否保留任何 Visual Basic 元素（缺省）。如果保留，则它们仍然是不可编辑的。

注意，`openpyxl` 当前没有读取 Excel 文件中所有可能的项目，因此如果打开并使用相同的名字保存的话，图像和图表将会从现有文件中丢失。



本章内容 2018 年 7 月 31 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (8): 简单的
应用示例 (续)

8. 简单的应用示例 (续)

下面让我们继续结合示例代码来理解 Python 对 Excel 的一些常用操作。

使用数字格式

下面的代码创建工作簿，在工作表中写入指定日期和数字，并输出数据的格式：

```
import datetime
from openpyxl import Workbook
wb=Workbook()
ws=wb.active
# 使用 Python 的 datetime 来设置日期
ws['A1']=datetime.datetime(2018,7,3)
print(ws['A1'].number_format)
#可以根据具体情况启用类型接口
wb.guess_types=True
#在结尾添加百分号来设置百分数
ws['B1']='3.14%'
wb.guess_types=False
ws['B1'].value
print(ws['B1'].number_format)
wb.save("numformat.xlsx")
```

代码中：

- `print(ws['A1'].number_format)` 将输出 `yyyy-mm-dd h:mm:ss`。



- `print(ws['B1'].number_format)` 将输出 0%。

使用公式

下面的代码演示在工作簿中输入公式：

```
from openpyxl import Workbook
wb=Workbook()
ws=wb.active
# 添加一个简单的公式
ws["A1"]="=SUM(1,1)"
wb.save("formula.xlsx")
```

代码运行后的结果如下图 8.1 所示。

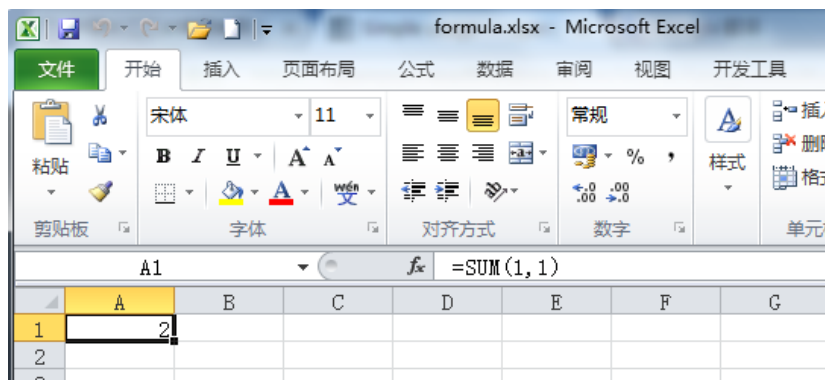


图 8.1

注意，必须使用英文作为函数的名字，且函数的参数必须由逗号分隔而不是其他的标点符号如分号。

openpyxl 决不会评估公式，但可能会检查公式的名字：

```
from openpyxl.utils import FORMULAE
"HEX2DEC" in FORMULAE
```

会得到结果：True



如果试图使用的公式未知,那么可能是因为所使用的公式没有包括在初始规范中。这些公式必须以_xlfn.为前缀来正常运转。

合并/取消合并单元格

在合并单元格时,除了左上角的单元格外,其他单元格都将从工作表中移除。下面的代码演示了合并和取消合并单元格的方法:

```
from openpyxl.workbook import Workbook
wb=Workbook()
ws=wb.active
ws.merge_cells('A2:D2')
ws.unmerge_cells('A2:D2')

# 或者等价地
ws.merge_cells(start_row=2,start_column=1,end_row=4,end_c
column=4)
ws.unmerge_cells(start_row=2,start_column=1,end_row=4,end
_column=4)
```

插入图像

下面的代码在工作表中添加图像:

```
from openpyxl import Workbook
from openpyxl.drawing.image import Image
wb=Workbook()
ws=wb.active
ws['A1']='You should see three logos below'

# 创建图像
img=Image('logo.png')
```



添加到工作表并锚定到旁边的单元格

```
ws.add_image(img, 'A1')
```

```
wb.save('logo.xlsx')
```

折叠列（大纲）

下面的代码在工作簿中添加一个新工作表，然后折叠列 A 至列 D 并隐藏：

```
import openpyxl
```

```
wb=openpyxl.Workbook()
```

```
ws=wb.create_sheet()
```

```
ws.column_dimensions.group('A','D',hidden=True)
```

```
wb.save('group.xlsx')
```

代码运行后的结果如图 8.2 所示。

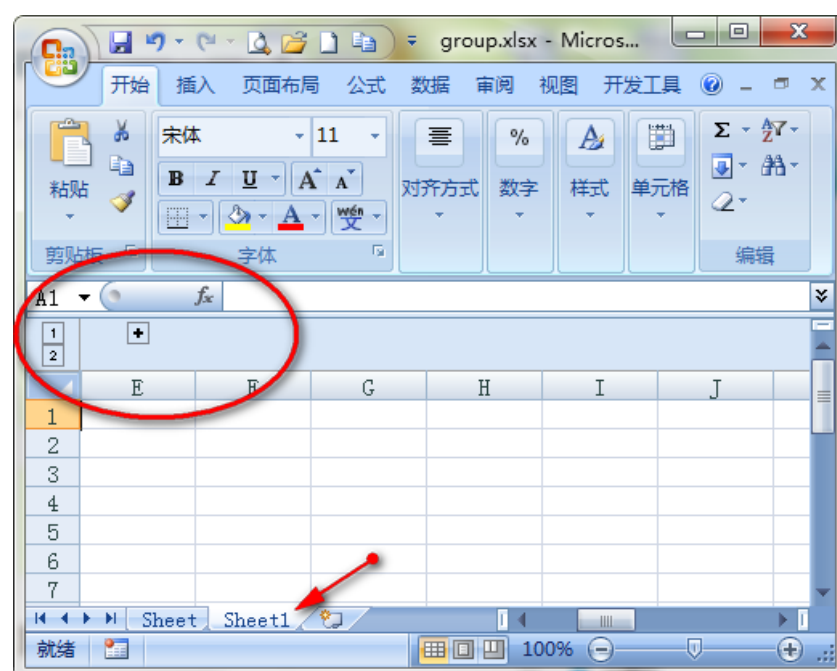


图 8.2



本章内容 2018 年 8 月 2 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (9): 图表操作

9. 图表操作

图表类型

在 Python 中，下列图表是可用的：

- 面积图：包括二维面积图、三维面积图
- 条形和柱形图：包括垂直、水平和堆积条形图，三维条形图
- 气泡图
- 折线图
- 散点图
- 饼图
- 圆环图
- 雷达图
- 股价图
- 曲面图

创建图表

图表由至少一个系列的一个或多个数据点组成。系列本身由单元格区域的引用组成。

下面的代码：



```

from openpyxl import Workbook
wb = Workbook()
ws = wb.active
for i in range(10):
    ws.append([i])

from openpyxl.chart import BarChart, Reference, Series
values = Reference(ws, min_col=1, min_row=1, max_col=1, max_row=10)
chart = BarChart()
chart.add_data(values)
ws.add_chart(chart, "E15")
wb.save("SampleChart.xlsx")

from openpyxl import Workbook
wb = Workbook()
ws = wb.active
for i in range(10):
    ws.append([i])

from openpyxl.chart import BarChart, Reference, Series
values = Reference(ws, min_col=1, min_row=1, max_col=1,
max_row=10)
chart = BarChart()
chart.add_data(values)
ws.add_chart(chart, "B2")
wb.save("SampleChart.xlsx")

```

首先创建一个工作簿并在工作表中输入数字 0-9, 然后使用这些数据生成柱状图并保存工作簿, 如下图 9.1 所示。

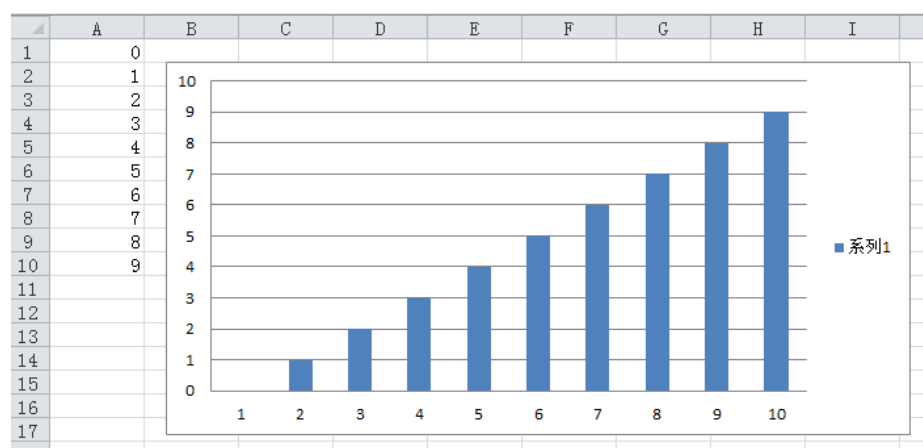


图 9.1



注意，默认情况下，图表左上角锚定在单元格 B2 中并且大小为 15×7.5cm，大约 14 行 5 列，实际的尺寸取决于操作系统和设备。当然，你可以改变图表的大小和锚定点。





本章内容 2018 年 8 月 5 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(10\): 图表坐标轴](#)

10. 图表坐标轴

下面介绍使用 Python 代码来设置 Excel 图表坐标轴。

图表坐标轴

可以手动设置坐标轴的最小值和最大值，以显示图表中的特定区域。如下所示的代码：

```
from openpyxl import Workbook
from openpyxl.chart import (
    ScatterChart,
    Reference,
    Series,
)

wb = Workbook()
ws = wb.active

ws.append(['X', '1/X'])
for x in range(-10, 11):
    if x:
        ws.append([x, 1.0/x])

chart1 = ScatterChart()
chart1.title = "Full Axes"
chart1.x_axis.title = 'x'
chart1.y_axis.title = '1/x'
chart1.legend = None

chart2 = ScatterChart()
chart2.title = "Clipped Axes"
chart2.x_axis.title = 'x'
chart2.y_axis.title = '1/x'
chart2.legend = None

chart2.x_axis.scaling.min = 0
chart2.y_axis.scaling.min = 0
chart2.x_axis.scaling.max = 11
chart2.y_axis.scaling.max = 1.5

x = Reference(ws, min_col=1, min_row=2, max_row=22)
y = Reference(ws, min_col=2, min_row=2, max_row=22)
s = Series(y, xvalues=x)
chart1.append(s)
chart2.append(s)

ws.add_chart(chart1, "C1")
ws.add_chart(chart2, "C15")

wb.save("minmax.xlsx")
```



结果如下图 10.1 所示。

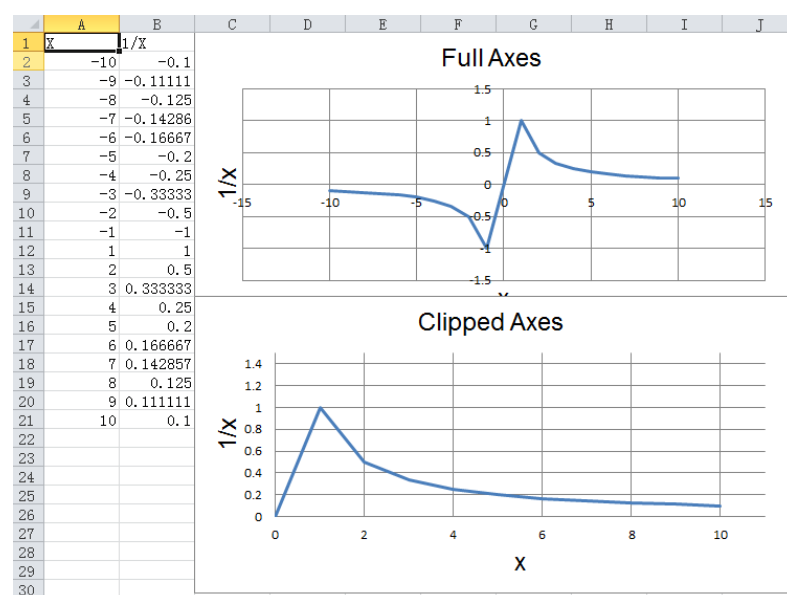


图 10.1

注意，在某些情况下，如上图 10.1 所示，设置坐标轴的限制值实际上相当于显示数据的子区域。对于大型数据集，使用数据子集而不是坐标轴限制值时，散点图的渲染速度会快得多。

x 轴和 y 轴都可以以对数比例来缩放。对数的基准可以设置为任何有效的浮点数。如果 x 轴以对数比例缩放，那么数据域中的负值将被丢弃。如下面的代码所示：

```
from openpyxl import Workbook
from openpyxl.chart import (
    ScatterChart,
    Reference,
    Series,
)
import math

wb = Workbook()
ws = wb.active

ws.append(['X', 'Gaussian'])
for i, x in enumerate(range(-10, 11)):
    ws.append([x, "=EXP(-(A$A{row}/6)^2)".format(row = i + 2)])

chart1 = ScatterChart()
chart1.title = "No Scaling"
chart1.x_axis.title = 'x'
chart1.y_axis.title = 'y'
chart1.lengend = None

chart2 = ScatterChart()
chart2.title = "X Log Scale"
chart2.x_axis.title = 'x (log10)'
chart2.y_axis.title = 'y'
chart2.legend = None
chart2.x_axis.scaling.logBase = 10
```



```

chart3 = ScatterChart()
chart3.title = "Y Log Scale"
chart3.x_axis.title = 'x'
chart3.y_axis.title = 'y (log10)'
chart3.legend = None
chart3.y_axis.scaling.logBase = 10

chart4 = ScatterChart()
chart4.title = "Both Log Scale"
chart4.x_axis.title = 'x (log10)'
chart4.y_axis.title = 'y (log10)'
chart4.legend = None
chart4.x_axis.scaling.logBase = 10
chart4.y_axis.scaling.logBase = 10

chart5 = ScatterChart()
chart5.title = "Log Scale Base e"
chart5.x_axis.title = 'x (ln)'
chart5.y_axis.title = 'y (ln)'
chart5.legend = None
chart5.x_axis.scaling.logBase = math.e
chart5.y_axis.scaling.logBase = math.e

x = Reference(ws, min_col=1, min_row=2, max_row=22)
y = Reference(ws, min_col=2, min_row=2, max_row=22)
s = Series(y, xvalues=x)
chart1.append(s)
chart2.append(s)
chart3.append(s)
chart4.append(s)
chart5.append(s)

ws.add_chart(chart1,"C1")
ws.add_chart(chart2,"I1")
ws.add_chart(chart3,"C15")
ws.add_chart(chart4,"I15")
ws.add_chart(chart5,"F30")

wb.save("log.xlsx")

```

上面的代码将在工作簿中生成 5 个图表，如下图 10.2 所示。

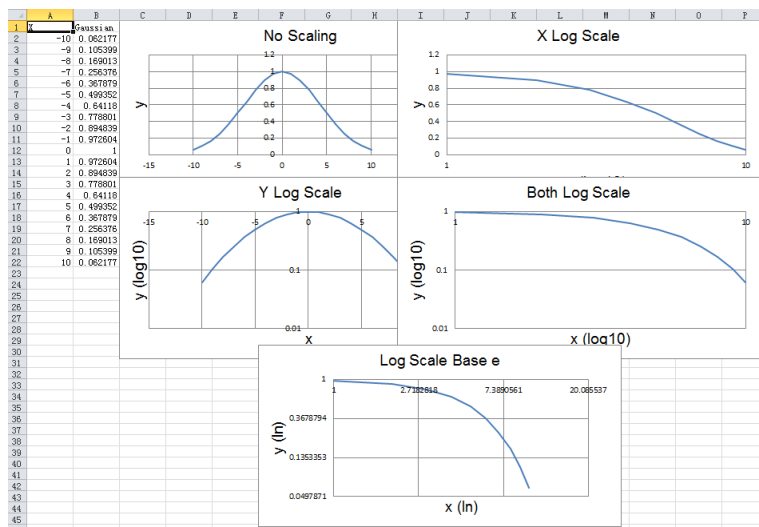


图 10.2



图 10.2 中，前 4 个图表在每个轴和两个轴上以对数比例缩放图表，对数的基数设置为 10。最后 1 个图表以 e 为基数的对数来缩放图表。

坐标轴可以正向或反向显示。坐标轴的方向由 orientation 属性来控制，其值为 minMax 时显示正向，其值为 maxMin 时显示反向。如下面的示例代码：

```
from openpyxl import Workbook
from openpyxl.chart import (
    ScatterChart,
    Reference,
    Series,
)

wb = Workbook()
ws = wb.active

ws["A1"] = "Archimedean Spiral"
ws.append(["T", "X", "Y"])
for i,t in enumerate(range(100)):
    ws.append([t/16.0, "={$A${row}*COS($A${row})}".format(row = i + 3),
               "={$A${row}*SIN($A${row})}".format(row = i + 3)])

chart1 = ScatterChart()
chart1.title = "Default Orientation"
chart1.x_axis.title = 'x'
chart1.y_axis.title = 'y'
chart1.legend = None

chart2 = ScatterChart()
chart2.title = "Flip X"
chart2.x_axis.title = 'x'
chart2.y_axis.title = 'y'
chart2.legend = None
chart2.x_axis.scaling.orientation = "maxMin"
chart2.y_axis.scaling.orientation = "minMax"

chart3 = ScatterChart()
chart3.title = "Flip Y"
chart3.x_axis.title = 'x'
chart3.y_axis.title = 'y'
chart3.legend = None
chart3.x_axis.scaling.orientation = "minMax"
chart3.y_axis.scaling.orientation = "maxMin"

chart4 = ScatterChart()
chart4.title = "Flip Both"
chart4.x_axis.title = 'x'
chart4.y_axis.title = 'y'
chart4.legend = None
chart4.x_axis.scaling.orientation = "maxMin"
chart4.y_axis.scaling.orientation = "maxMin"

x = Reference(ws, min_col=2, min_row=2, max_row=102)
y = Reference(ws, min_col=3, min_row=2, max_row=102)
s = Series(y, xvalues=x)
chart1.append(s)
chart2.append(s)
chart3.append(s)
chart4.append(s)

ws.add_chart(chart1, "D1")
ws.add_chart(chart2, "J1")
ws.add_chart(chart3, "D15")
ws.add_chart(chart4, "J15")

wb.save("orientation.xlsx")
```



代码将生成 4 张图表，坐标轴每种可能的方向组合看起来如下图 10.3 所示。

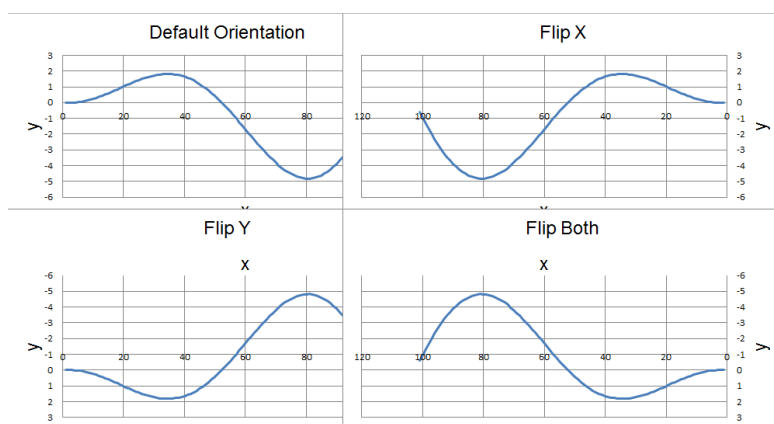


图 10.3

添加次要坐标轴

添加次要坐标轴实际上涉及到创建第二个图表，该图表与第一个图表共享一个公共的 x 轴但具有单独的 y 轴。下面是示例代码：



```

from openpyxl import Workbook
from openpyxl.chart import (
    LineChart,
    BarChart,
    Reference,
    Series,
)

wb = Workbook()
ws = wb.active

rows = [
    ['Aliens', 2, 3, 4, 5, 6, 7],
    ['Humans', 10, 40, 50, 20, 10, 50],
]

for row in rows:
    ws.append(row)

c1 = BarChart()
v1 = Reference(ws, min_col=1, min_row=1, max_col=7)
c1.add_data(v1, titles_from_data=True, from_rows=True)

c1.x_axis.title = 'Days'
c1.y_axis.title = 'Aliens'
c1.y_axis.majorGridlines = None
c1.title = 'Survey results'

# 创建第二个图表
c2 = LineChart()
v2 = Reference(ws, min_col=1, min_row=2, max_col=7)
c2.add_data(v2, titles_from_data=True, from_rows=True)
c2.y_axis.axId = 200
c2.y_axis.title = "Humans"

# 在右侧显示第二个图表的y轴
c1.y_axis.crosses = "max"
c1 += c2

ws.add_chart(c1, "D4")

wb.save("secondary.xlsx")

```

运行代码后，将生成一个折线和柱状的组合图表，如下图 10.4 所示。

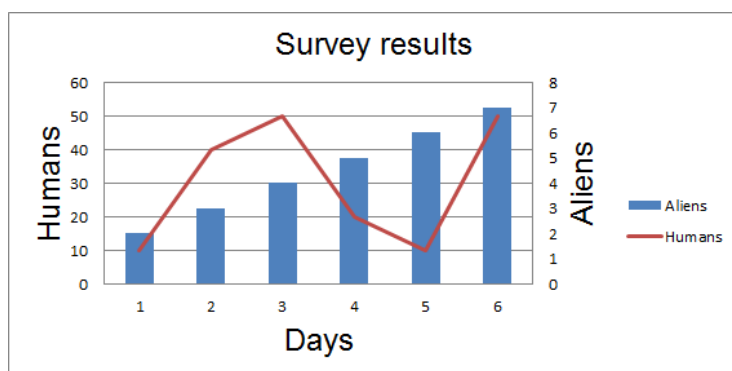


图 10.4



本章内容 2018 年 8 月 8 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (11): 图表布局

11. 图表布局

通过使用 `layout` 类实例的 `layout` 属性，可以设置图表的布局。

图表布局

大小和位置

图表能够在其容器中进行定位，`x` 和 `y` 调整位置，`w` 和 `h` 调整大小，单位是容器的比例。图表不能够被定位到其容器之外，宽度和高度限制规则：如果 $x+w>1$ ，那么 $x=1-w$ 。

`x` 是距离左侧的水平位置，`y` 是距离顶部的垂直位置，`h` 是图表相对于其容器的高度，`w` 是框的宽度。

模式

除了大小和位置外，相关属性的模式也可以设置为 `factor` 或 `edge`，默认为 `factor`。

```
layout.xMode = edge
```

目标

可以设置 `layoutTarget` 为 `outer` 或 `inner`，默认为 `outer`。

```
layout.layoutTarget = inner
```



图例布局

通过设置其位置：r、l、t、b 和 tr，分别对应右、左、顶、底和右上角，可以控制图例的位置，默认值是 r。

```
legend.position = 'tr'
```

或者应用手工布局：

```
legend.layout = ManualLayout()
```

示例

下面是一段示例代码：

```
from openpyxl import Workbook, load_workbook
from openpyxl.chart import ScatterChart, Series, Reference
from openpyxl.chart.layout import Layout, ManualLayout

wb = Workbook()
ws = wb.active

rows = [
    ['Size', 'Batch 1', 'Batch 2'],
    [2, 40, 30],
    [3, 40, 25],
    [4, 50, 30],
    [5, 30, 25],
    [6, 25, 35],
    [7, 20, 40],
]

for row in rows:
    ws.append(row)
```



```

ch1 = ScatterChart()
xvalues = Reference(ws, min_col=1, min_row=2, max_row=7)
for i in range(2,4):
    values = Reference(ws, min_col=i, min_row=1, max_row=7)
    series = Series(values, xvalues, title_from_data=True)
    ch1.series.append(series)

ch1.title = "Default layout"
ch1.style = 13
ch1.x_axis.title = 'Size'
ch1.y_axis.title = 'Percentage'
ch1.legend.position = 'r'

ws.add_chart(ch1, "B10")

from copy import deepcopy

# 图表变小 1/2, 右下角
ch2 = deepcopy(ch1)
ch2.title = "Manual chart layout"
ch2.legend.position = "tr"
ch2.layout = Layout(
    manualLayout = ManualLayout(
        x=0.25, y=0.25,
        h=0.5, w=0.5,
    )
)
ws.add_chart(ch2, "H10")

# 图表变小 1/2, 居中
ch3 = deepcopy(ch1)

```



```

ch3.layout = Layout(
    ManualLayout(
        x=0.25,y=0.25,
        h=0.5,w=0.5,
        xMode="edge",
        yMode="edge",
    )
)
ch3.title = "Manual chart layout, edge mode"
ws.add_chart(ch3,"B27")

# 手动放置图例在左下角
ch4 = deepcopy(ch1)
ch4.title = "Manual legend layout"
ch4.legend.layout = Layout(
    manualLayout=ManualLayout(
        yMode='edge',
        xMode='edge',
        x=0,y=0.9,
        h=0.1,w=0.5
    )
)
ws.add_chart(ch4,"H27")

wb.save("chart_layout.xlsx")

```

代码运行后生成的 4 张图表演示了各种可能性，如下图 11.1 所示。

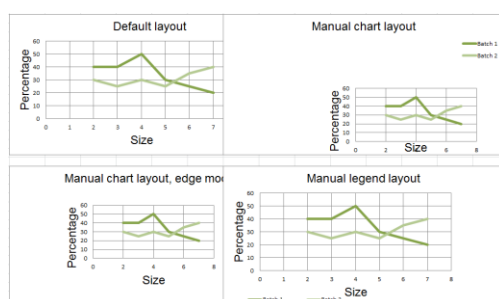


图 11.1



本章内容 2018 年 8 月 12 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
Python 操作 Excel 学习笔记 (12): 面积图

12. 面积图

面积图类似于折线图，并且填充了绘图线下面的区域。可以设置不同的面积图类型，包括标准型、堆积型、百分比堆积型，其中默认为标准型。

二维面积图

下面的代码用于创建 Excel 工作簿，在工作表中写入数据，并使用这些数据绘制面积图：

```
from openpyxl import Workbook
from openpyxl.chart import (
    AreaChart,
    Reference,
    Series,
)

wb=Workbook()
ws=wb.active

rows=[
    ['Number', 'Batch 1', 'Batch 2'],
    [2, 40, 30],
    [3, 40, 25],
    [4, 50, 30],
    [5, 30, 10],
    [6, 25, 5],
```



```

        [7,50,10],
    ]

    for row in rows:
        ws.append(row)

    chart=AreaChart()
    chart.title="面积图"
    chart.style=13
    chart.x_axis.title='测试'
    chart.y_axis.title='百分比'

    cats=Reference(ws,min_col=1,min_row=1,max_row=7)
    data=Reference(ws,min_col=2,min_row=1,max_col=3,max_row=7)
    chart.add_data(data,titles_from_data=True)
    chart.set_categories(cats)

    ws.add_chart(chart,"A10")

    wb.save("area.xlsx")

```

运行代码后，将在当前目录下创建一个名为 area.xlsx 的工作簿，其内容如下图所示 12.1 所示。

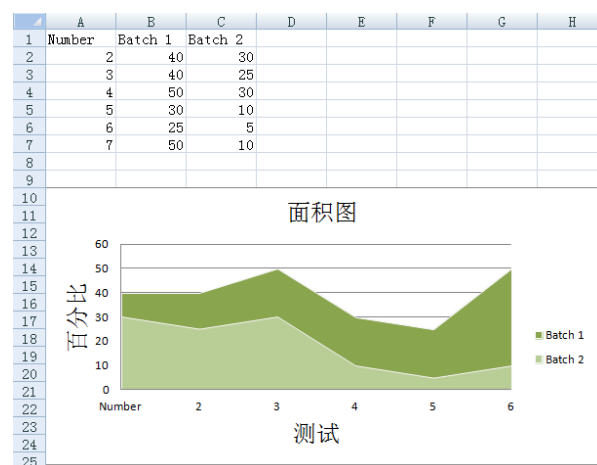


图 12.1



三维面积图

也可以创建三维面积图，代码如下：

```
from openpyxl import Workbook
from openpyxl.chart import (
    AreaChart3D,
    Reference,
    Series,
)

wb=Workbook()
ws=wb.active

rows=[
    ['Number', 'Batch 1', 'Batch 2'],
    [2, 30, 40],
    [3, 25, 40],
    [4, 30, 50],
    [5, 10, 30],
    [6, 5, 25],
    [7, 10, 50],
]

for row in rows:
    ws.append(row)

chart=AreaChart3D()
chart.title="面积图"
chart.style=13
chart.x_axis.title='测试'
chart.y_axis.title='百分比'
chart.legend=None
```



```

cats=Reference(ws,min_col=1,min_row=1,max_row=7)
data=Reference(ws,min_col=2,min_row=1,max_col=3,max_row=7)
chart.add_data(data,titles_from_data=True)
chart.set_categories(cats)

ws.add_chart(chart,"A10")
wb.save("area3D.xlsx")

```

代码将生成一个简单的三维面积图，第三坐标轴用于代替图例，如下图 12.2 所示。

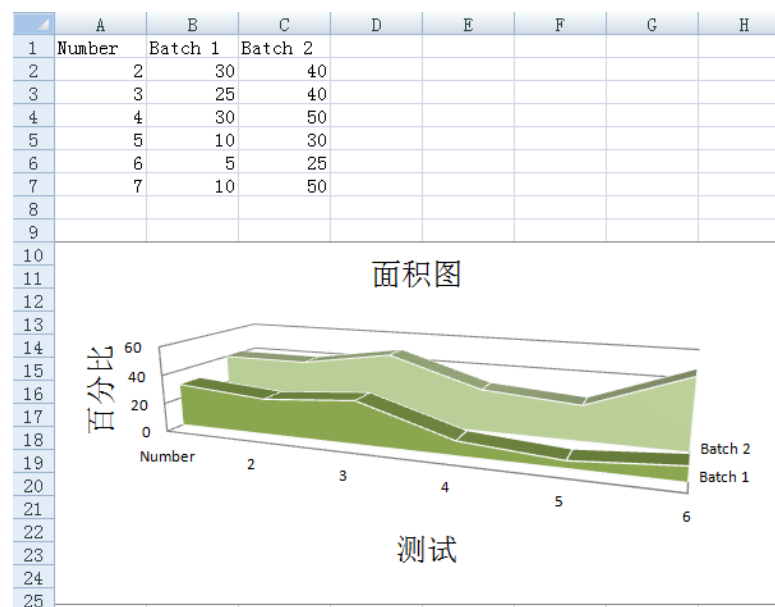


图 12.2



本章内容 2018 年 8 月 19 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
**Python 操作 Excel 学习笔记 (13): 条形图
和柱状图**

13. 条形图和柱状图

垂直、水平和堆积条形图

注意，下列设置影响不同的图表类型：

- 通过设置 `type` 为 `col` 或 `bar` 在垂直和水平条形图之间切换
- 当使用堆积图时，`overlap` 需要被设置成 100
- 如果条形是水平的，那么 `x` 和 `y` 轴相反

下面的代码用于创建不同型式的条形图：

```
from openpyxl import Workbook
from openpyxl.chart import BarChart, Series, Reference

wb = Workbook(write_only=True)
ws = wb.create_sheet()

rows = [
    ('Number', 'Batch 1', 'Batch 2'),
```



```

        (2, 10, 30),
        (3, 40, 60),
        (4, 50, 70),
        (5, 20, 10),
        (6, 10, 40),
        (7, 50, 30),
    ]

    for row in rows:
        ws.append(row)

    chart1 = BarChart()
    chart1.type = "col"
    chart1.style = 10
    chart1.title = "Bar Chart"
    chart1.y_axis.title = 'Test number'
    chart1.x_axis.title = 'Sample length (mm)'

    data = Reference(ws, min_col=2, min_row=1, max_row=7,
max_col=3)
    cats = Reference(ws, min_col=1, min_row=2, max_row=7)
    chart1.add_data(data, titles_from_data=True)
    chart1.set_categories(cats)
    chart1.shape = 4
    ws.add_chart(chart1, "A10")

    from copy import deepcopy

    chart2 = deepcopy(chart1)
    chart2.style = 11
    chart2.type = "bar"
    chart2.title = "Horizontal Bar Chart"

```



```
ws.add_chart(chart2, "G10")

chart3 = deepcopy(chart1)
chart3.type = "col"
chart3.style = 12
chart3.grouping = "stacked"
chart3.overlap = 100
chart3.title = 'Stacked Chart'

ws.add_chart(chart3, "A27")

chart4 = deepcopy(chart1)
chart4.type = "bar"
chart4.style = 13
chart4.grouping = "percentStacked"
chart4.overlap = 100
chart4.title = 'Percent Stacked Chart'

ws.add_chart(chart4, "G27")

wb.save("bar.xlsx")
```

运行代码后，将在当前目录下创建一个名为 `bar.xlsx` 的工作簿，其内容如下图所示 13.1 所示。



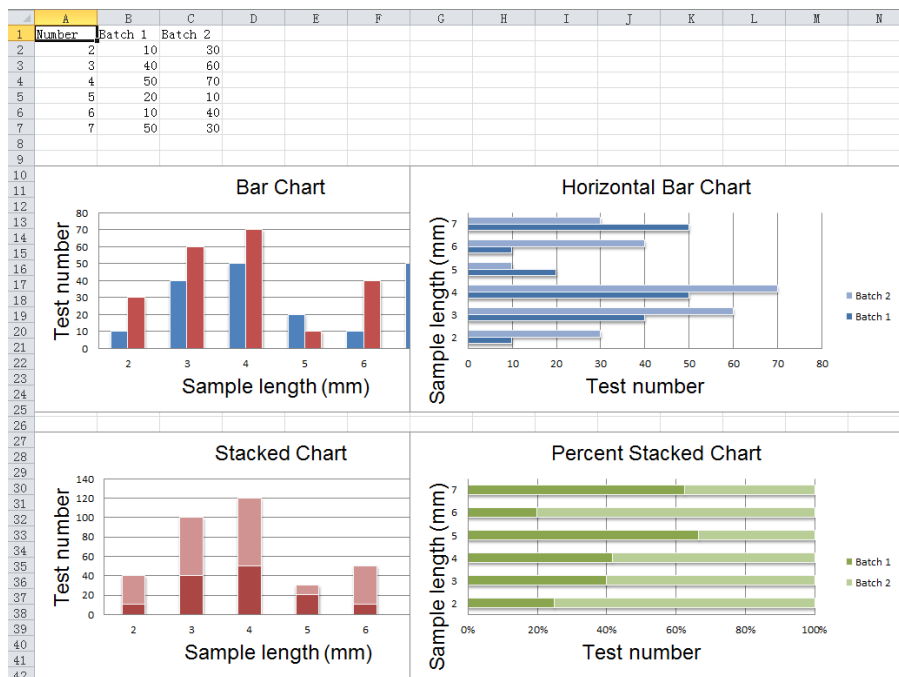


图 13.1

三维条形图

也可以创建三维条形图，代码如下：

```
from openpyxl import Workbook
from openpyxl.chart import (
    Reference,
    Series,
    BarChart3D,
)

wb = Workbook()
ws = wb.active

rows = [
    (None, 2013, 2014),
    ("Apples", 5, 4),
    ("Oranges", 6, 2),

```

60



```

        ("Pears", 8, 3)
    ]

    for row in rows:
        ws.append(row)

    data = Reference(ws, min_col=2, min_row=1, max_col=3,
max_row=4)
    titles = Reference(ws, min_col=1, min_row=2, max_row=4)
    chart = BarChart3D()
    chart.title = "3D Bar Chart"
    chart.add_data(data=data, titles_from_data=True)
    chart.set_categories(titles)

    ws.add_chart(chart, "A6")
    wb.save("bar3d.xlsx")

```

代码将生成一个简单的三维条形图，如下图 13.2 所示。

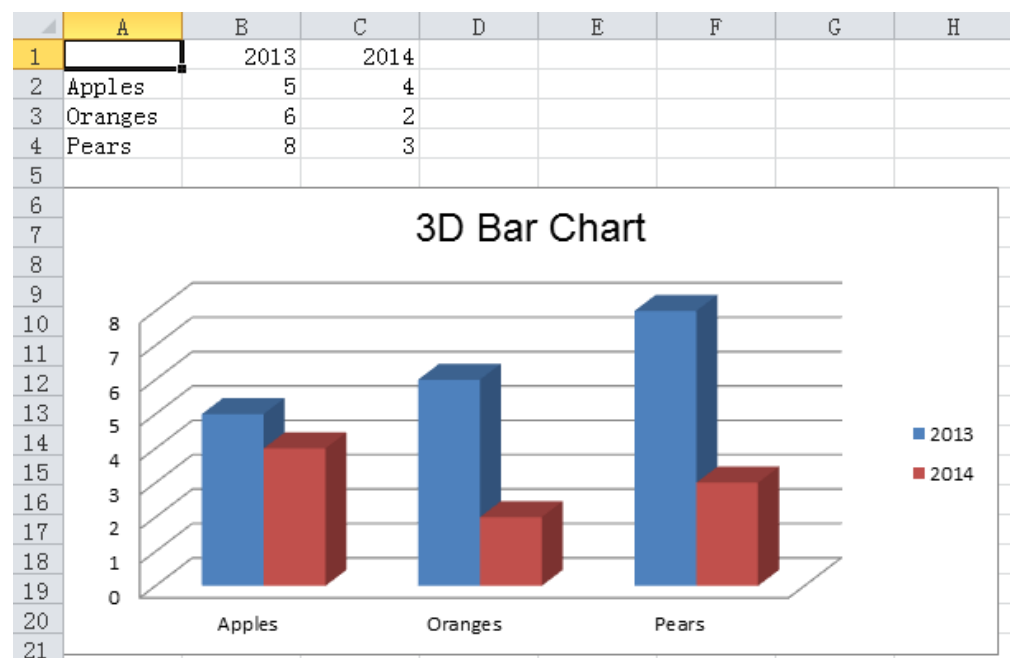


图 13.2





本章内容 2018 年 8 月 26 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(14\): 气泡图](#)

14. 气泡图

气泡图与散点图相似，但使用第三维来决定气泡的大小，可以包括多个系列。

下面的代码创建气泡图：

```
from openpyxl import Workbook
from openpyxl.chart import Series, Reference, BubbleChart

wb = Workbook()
ws = wb.active

rows = [
    ("Number of Products", "Sales in USD", "Market share"),
    (14, 12200, 15),
    (20, 60000, 33),
    (18, 24400, 10),
    (22, 32000, 42),
    (),
    (12, 8200, 18),
    (15, 50000, 30),
    (19, 22400, 15),
    (25, 25000, 50),
]

for row in rows:
    ws.append(row)
```



```

chart = BubbleChart()

chart.style = 18 #使用预设的样式

# 添加第一个数据系列

xvalues = Reference(ws, min_col=1, min_row=2, max_row=5)
yvalues = Reference(ws, min_col=2, min_row=2, max_row=5)
size = Reference(ws, min_col=3, min_row=2, max_row=5)
series = Series(values=yvalues, xvalues=xvalues,
zvalues=size, title="2013")
chart.series.append(series)

# 添加第二个数据系列

xvalues = Reference(ws, min_col=1, min_row=7, max_row=10)
yvalues = Reference(ws, min_col=2, min_row=7, max_row=10)
size = Reference(ws, min_col=3, min_row=7, max_row=10)
series = Series(values=yvalues, xvalues=xvalues,
zvalues=size, title="2014")
chart.series.append(series)

# 在以单元格 A12 开始的单元格中放置图表

ws.add_chart(chart, "A12")
wb.save("bubble.xlsx")

```

运行代码后，将在当前目录下创建一个名为 bubble.xlsx 的工作簿，其内容如下图所示 14.1 所示。

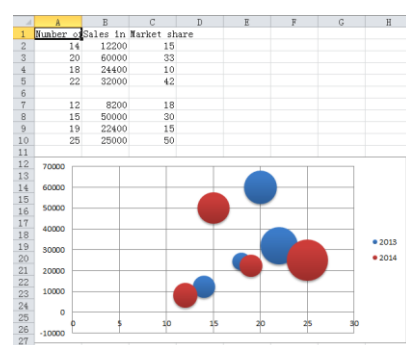


图 14.1



本章内容 2018 年 9 月 2 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(15\): 折线图](#)

15.折线图

折线图允许数据针对固定轴来绘制，与散点图相似，主要的区别是折线图的每个数据系统根据相同值绘制。不同种类的轴被用于次要坐标轴。

与条形图相似，折线图有 3 种：标准的、堆积的和百分比堆积的。

```
from datetime import date

from openpyxl import Workbook
from openpyxl.chart import (
    LineChart,
    Reference,
)
from openpyxl.chart.axis import DateAxis

wb = Workbook()
ws = wb.active

rows = [
    ['date', 'Batch 1', 'Batch 2', 'Batch 3'],
    [date(2018,8,1), 40, 30, 25],
    [date(2018,8,2), 40, 25, 30],
    [date(2018,8,3), 50, 30, 45],
    [date(2018,8,4), 30, 25, 40],
    [date(2018,8,5), 25, 35, 30],
    [date(2018,8,6), 20, 40, 35],
```



```

]

for row in rows:
    ws.append(row)

c1 = LineChart()
c1.title = "Line Chart"
c1.style = 13
c1.y_axis.title = 'Size'
c1.x_axis.title = 'Test Number'

data = Reference(ws, min_col=2, min_row=1, max_col=4,
max_row=7)
c1.add_data(data, titles_from_data=True)

# 线条样式
s1 = c1.series[0]
s1.marker.symbol = "triangle"
s1.marker.graphicalProperties.solidFill = "FF0000"
s1.marker.graphicalProperties.line.solidFill = "FF0000"

s1.graphicalProperties.line.noFill = True

s2 = c1.series[1]
s2.graphicalProperties.line.solidFill = "00AAAA"
s2.graphicalProperties.line.dashStyle = "sysDot"
s2.graphicalProperties.line.width = 100050

s2 = c1.series[2]
s2.smooth = True

ws.add_chart(c1, "A10")

```



```

from copy import deepcopy
stacked = deepcopy(c1)
stacked.grouping = "stacked"
stacked.title = "Stacked Line Chart"
ws.add_chart(stacked, "A27")

percent_stacked = deepcopy(c1)
percent_stacked.grouping = "percentStacked"
percent_stacked.title = "Percent Stacked Line Chart"
ws.add_chart(percent_stacked, "A44")

c2 = LineChart()
c2.title = "Data Axis"
c2.style = 12
c2.y_axis.title = "Size"
c2.y_axis.crossAx = 500
c2.x_axis = DateAxis(crossAx=100)
c2.x_axis.number_format = 'd-mmm'
c2.x_axis.majorTimeUnit = "days"
c2.x_axis.title = "Date"

c2.add_data(data, titles_from_data=True)
dates = Reference(ws, min_col=1, min_row=2, max_row=7)
c2.set_categories(dates)

ws.add_chart(c2, "A61")

wb.save("line.xlsx")

```

运行代码后，将在当前目录下创建一个名为 line.xlsx 的工作簿，其内容如下图 15.1 所示。



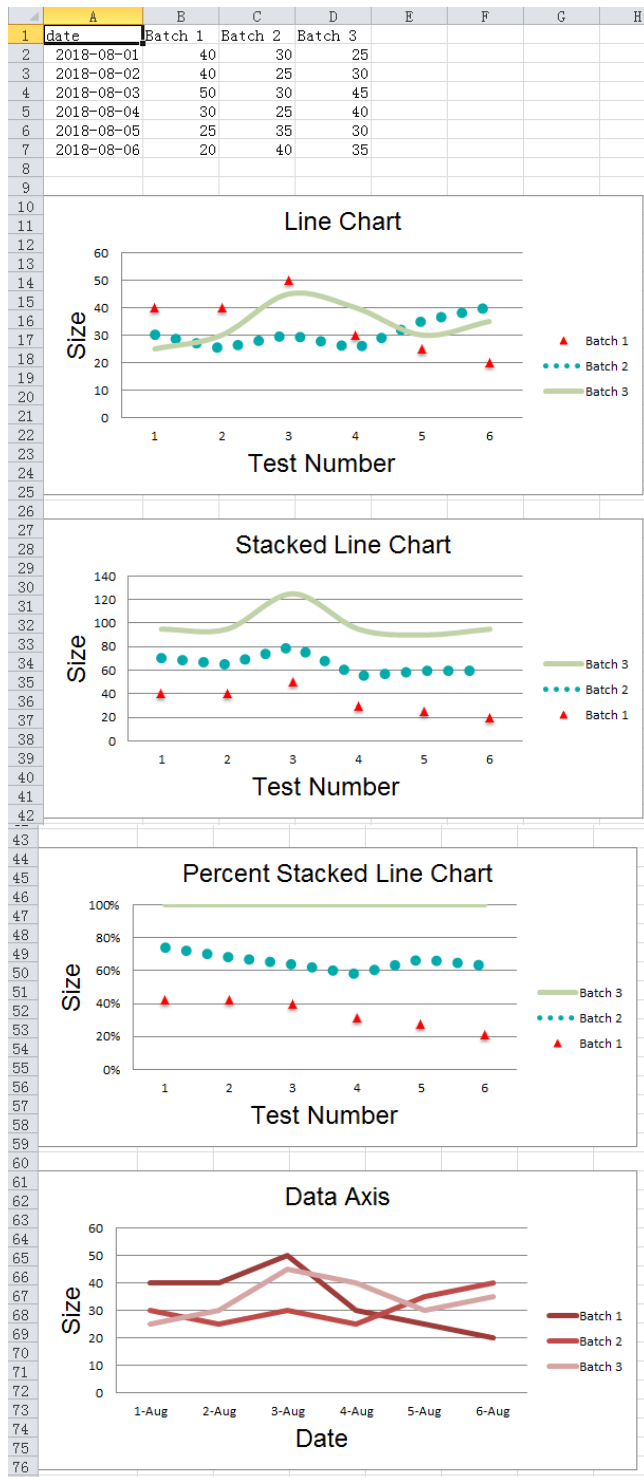


图 15.1



三维折线图

在三维折线图中，第三轴与系列的图例相同。

```
from datetime import date

from openpyxl import Workbook
from openpyxl.chart import (
    LineChart3D,
    Reference,
)

from openpyxl.chart.axis import DateAxis

wb = Workbook()
ws = wb.active

rows = [
    ['Date', 'Batch 1', 'Batch 2', 'Batch 3'],
    [date(2018,8,1), 40, 30, 25],
    [date(2018,8,2), 40, 25, 30],
    [date(2018,8,3), 50, 30, 45],
    [date(2018,8,4), 30, 25, 40],
    [date(2018,8,5), 25, 35, 30],
    [date(2018,8,6), 20, 40, 35],
]

for row in rows:
    ws.append(row)

c1 = LineChart3D()
c1.title = "3D Line Chart"
c1.legend = None
```



```

c1.style = 15
c1.y_axis.title = 'Size'
c1.x_axis.title = 'Test Number'

data = Reference(ws, min_col=2, min_row=1, max_col=4,
max_row=7)
c1.add_data(data, titles_from_data=True)

ws.add_chart(c1, "A10")

wb.save("line3D.xlsx")

```

运行代码后,将在当前目录下创建一个名为 line3D.xlsx 的工作簿,其内容如下图 15.2 所示。

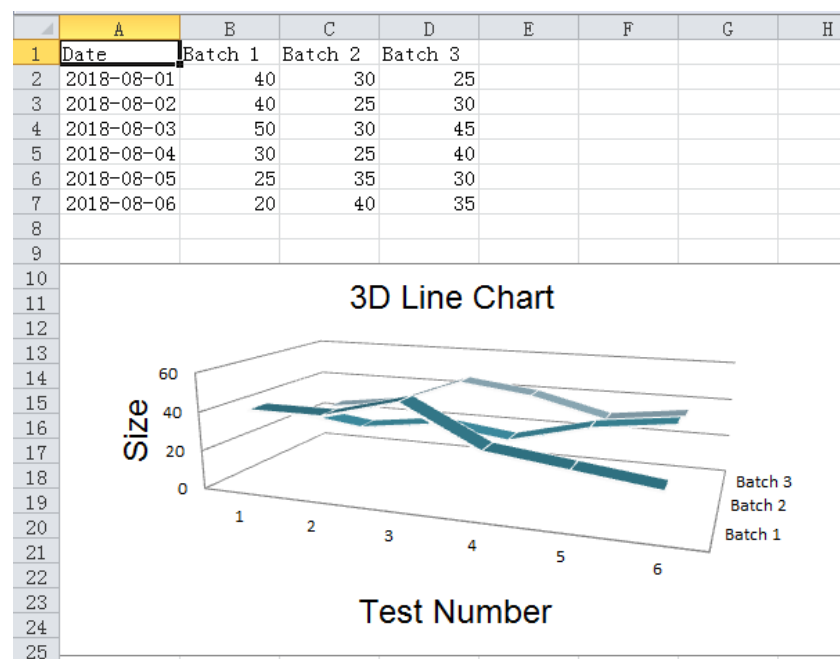


图 15.2



本章内容 2018 年 9 月 9 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(16\): 散点图](#)

16.散点图

散点图或 xy 图类似于某些折线图，主要区别在于一个系列的值相对于另一个系列来绘制。在数值无序的情况下，这种图表很有用。

下面的代码根据设定的数据在工作表中绘制散点图：

```
from openpyxl import Workbook
from openpyxl.chart import (
    ScatterChart,
    Reference,
    Series,
)

wb = Workbook()
ws = wb.active

rows = [
    ['Size', 'Batch 1', 'Batch 2'],
    [2, 40, 30],
    [3, 40, 25],
    [4, 50, 30],
    [5, 30, 25],
    [6, 25, 35],
    [7, 20, 40],
]

for row in rows:
```



```

ws.append(row)

chart = ScatterChart()
chart.title = "Scatter Chart"
chart.style = 13
chart.x_axis.title = 'Size'
chart.y_axis.title = 'Percentage'

xvalues = Reference(ws, min_col=1, min_row=2, max_row=7)
for i in range(2, 4):
    values = Reference(ws, min_col=i, min_row=1, max_row=7)
    series = Series(values, xvalues, title_from_data=True)
    chart.series.append(series)

ws.add_chart(chart, "A10")
wb.save("scatter.xlsx")

```

运行代码后，将在当前目录下创建一个名为 `scatter.xlsx` 的工作簿，其内容如下图 16.1 所示。

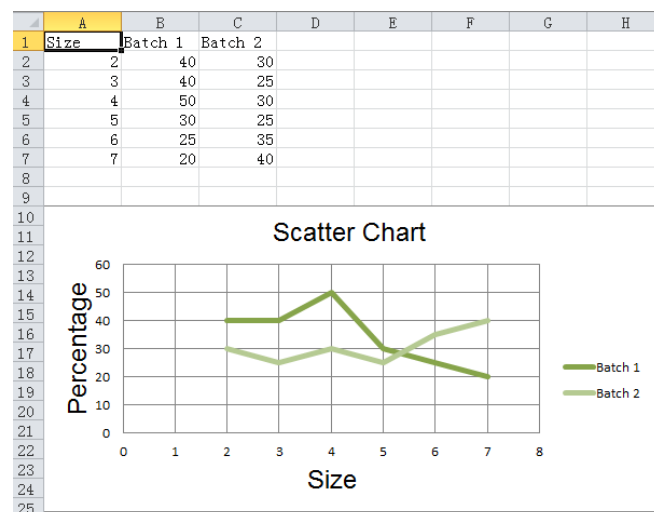


图 16.1

注：规范说明有下列类型的散点图：直线、直线标记、标记、平滑、平滑标记。然而，至少在 Microsoft Excel 中，这只是其他设置的快捷方式。为了与折线图保持一致，应手动设置每个系列的样式。



本章内容 2018 年 9 月 16 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(17\): 饼图](#)

17. 饼图

饼图将数据绘制为圆的切片，每个切片代表整体的百分比。切片以顺时针方向绘制，零度位于圆的顶部。饼图只能接受一个单独的数据系列，图表标题默认为系列标题。

使用 Python 代码在 Excel 中绘制饼图的示例代码如下：

```
from openpyxl import Workbook

from openpyxl.chart import (
    PieChart,
    ProjectedPieChart,
    Reference
)

from openpyxl.chart.series import DataPoint

data = [
    ['Pie', 'Sold'],
    ['Apple', 50],
    ['Cherry', 30],
    ['Pumpkin', 10],
    ['Chocolate', 40],
]

wb = Workbook()
```



```

ws = wb.active

for row in data:
    ws.append(row)

pie = PieChart()
labels = Reference(ws, min_col=1, min_row=2, max_row=5)
data = Reference(ws, min_col=2, min_row=1, max_row=5)
pie.add_data(data, titles_from_data=True)
pie.set_categories(labels)
pie.title = "Pies sold by category"

# 从饼图中切出一片
slice = DataPoint(idx=0, explosion=20)
pie.series[0].data_points = [slice]

ws.add_chart(pie, "D1")

ws = wb.create_sheet(title="Projection")

data = [
    ['Page', 'Views'],
    ['Search', 95],
    ['Products', 4],
    ['Offers', 0.5],
    ['Sales', 0.5],
]

for row in data:
    ws.append(row)

projected_pie = ProjectedPieChart()

```



```

projected_pie.type = "pie"
projected_pie.splitType = "val" #按值拆分
labels = Reference(ws, min_col=1, min_row=2, max_row=5)
data = Reference(ws, min_col=2, min_row=1, max_row=5)
projected_pie.add_data(data, titles_from_data=True)
projected_pie.set_categories(labels)

ws.add_chart(projected_pie, "A10")

from copy import deepcopy
projected_bar = deepcopy(projected_pie)
projected_bar.type = "bar"
projected_bar.splitType = 'pos' #按位置拆分

ws.add_chart(projected_bar, "A27")
wb.save("pie.xlsx")

```

代码运行后的结果如下图 17.1 所示。

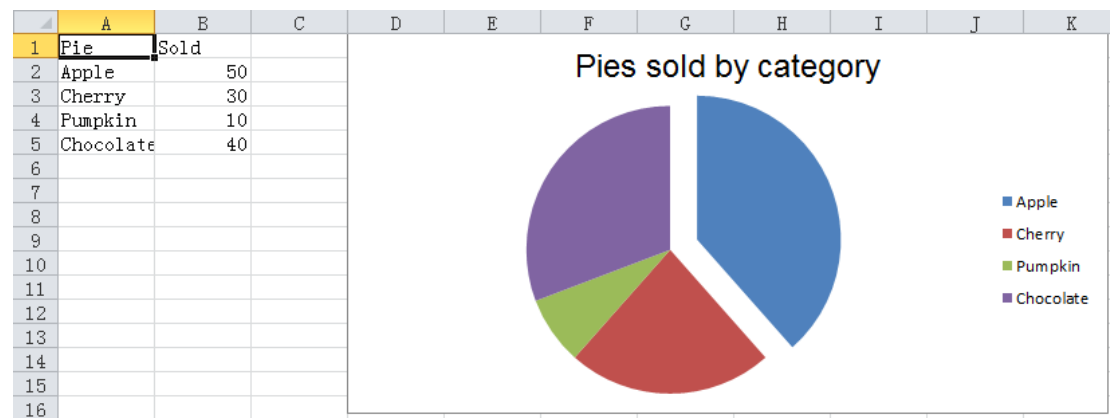


图 17.1



复合饼图

复合饼图从饼图中提取一些切片并将其投影到第二个饼图或条形图中。当数据系列中有多个较小的项目时，非常有用。这样的饼图可以按百分比、值或者位置进行拆分，如果未进行设置，则由应用程序确定使用的样式。此外，还可以自定义拆分样式。

上述代码创建的复合饼图如下图 17.2 所示。

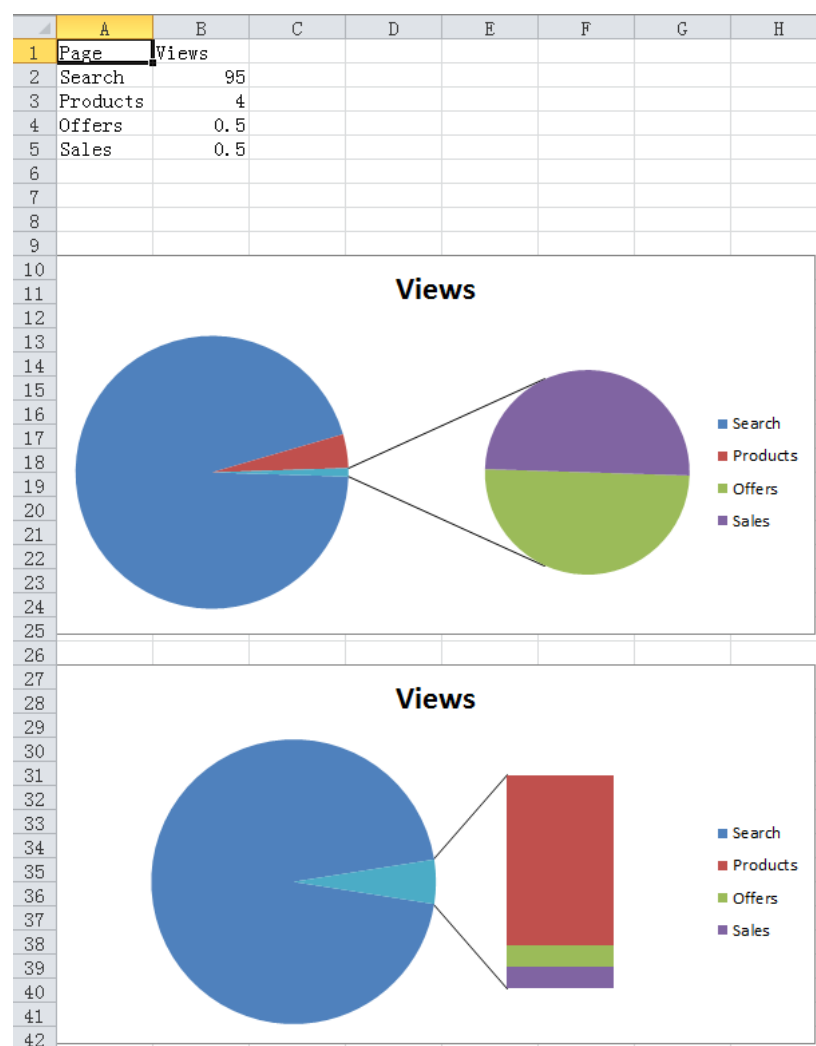


图 17.2



三维饼图

也能够创建三维效果的饼图。

```
from openpyxl import Workbook

from openpyxl.chart import (
    PieChart3D,
    Reference
)

data = [
    ['Pie', 'Sold'],
    ['Apple', 50],
    ['Cherry', 30],
    ['Pumpkin', 10],
    ['Chocolate', 40],
]

wb = Workbook()
ws = wb.active

for row in data:
    ws.append(row)

pie = PieChart3D()
labels = Reference(ws, min_col=1, min_row=2, max_row=5)
data = Reference(ws, min_col=2, min_row=1, max_row=5)
pie.add_data(data, titles_from_data=True)
pie.set_categories(labels)
pie.title = "Pies sold by category"

ws.add_chart(pie, "D1")
```



```
wb.save("pie3D.xlsx")
```

代码运行后的结果如下图 17.3 所示。

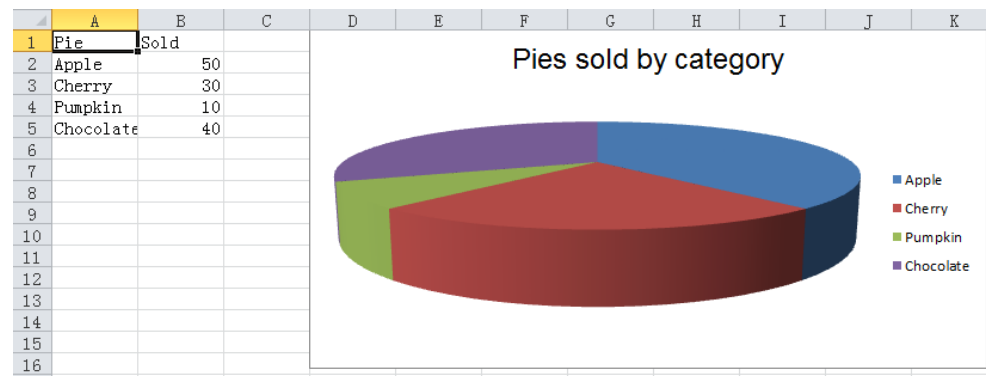


图 17.3

渐变饼图

也能够使用渐变系列创建饼图。



本章内容 2018 年 9 月 23 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(18\): 圆环图](#)

18.圆环图

圆环图与饼图相似，只是圆环图使用的是环形而饼图使用的是圆形。圆环图还可以将几个系列的数据绘制成同心环。

绘制圆环图的基本代码如下：

```
from openpyxl import Workbook

from openpyxl.chart import (
    DoughnutChart,
    Reference,
    Series,
)

from openpyxl.chart.series import DataPoint

data = [
    ['Pie', 2017, 2018],
    ['Plain', 40, 50],
    ['Jam', 2, 10],
    ['Lime', 20, 30],
    ['Chocolate', 30, 40],
]

wb = Workbook()
ws = wb.active
```



```

for row in data:
    ws.append(row)

chart = DoughnutChart()
labels = Reference(ws, min_col=1, min_row=2, max_row=5)
data = Reference(ws, min_col=2, min_row=1, max_row=5)
chart.add_data(data, titles_from_data=True)
chart.set_categories(labels)
chart.title = "Doughnuts sold by category"
chart.style = 26

# 从圆环图中切出第 1 片
slices = [DataPoint(idx=i) for i in range(4)]
plain, jam, lime, chocolate = slices
chart.series[0].data_points = slices
plain.graphicalProperties.solidFill = "FAE1D0"
jam.graphicalProperties.solidFill = "BB2244"
lime.graphicalProperties.solidFill = "22DD22"
chocolate.graphicalProperties.solidFill = "61210B"
chocolate.exlosion = 10

ws.add_chart(chart, "E1")

from copy import deepcopy

chart2 = deepcopy(chart)
chart2.title = None
data = Reference(ws, min_col=3, min_row=1, max_row=5)
series2 = Series(data, title_from_data=True)
series2.data_points = slices
chart2.series.append(series2)

```



```
ws.add_chart(chart2, "E17")
```

```
wb.save("doughnut.xlsx")
```

运行上述代码后的结果如下图 18.1 所示。

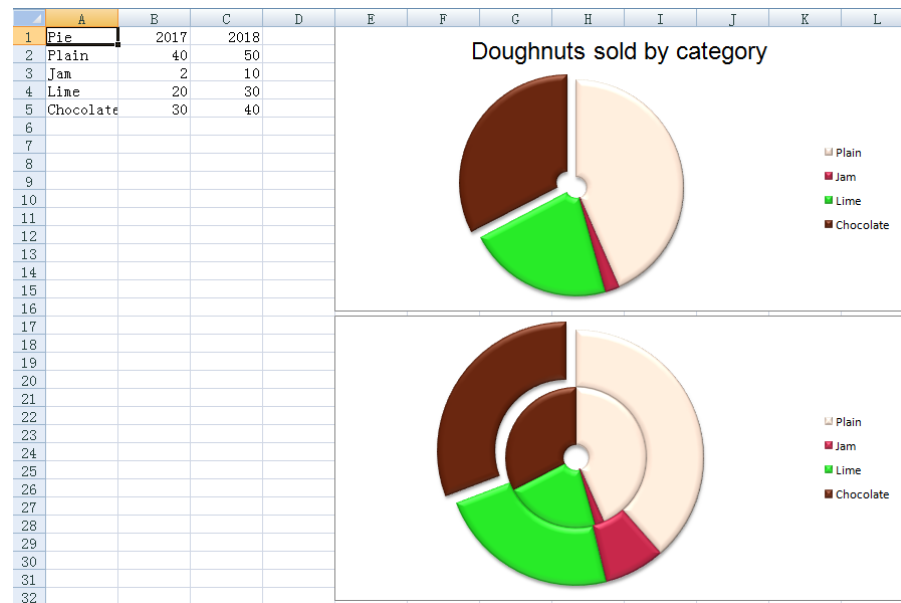


图 18.1





本章内容 2018 年 9 月 30 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(19\): 雷达图](#)

19. 雷达图

在雷达图中可以绘制工作表列或行中的数据。雷达图比较多个数据系列的集合值，实际上是面积图在圆形 x 轴上的投影。

有两种类型的雷达图：标准型，其区域带有直线标记；以及填充整个区域的填充位置。额外的类型“标记”无效。如果需要标记，可以为相关系列设置。

创建雷达图的基本代码如下：

```
from openpyxl import Workbook
from openpyxl.chart import (
    RadarChart,
    Reference,
)

wb = Workbook()
ws = wb.active

rows = [
    ['Month', 'Bulbs', 'Seeds', 'Flowers', 'Trees & shrubs'],
    ['Jan', 0, 2500, 500, 0],
    ['Feb', 0, 5500, 750, 1500],
    ['Mar', 0, 9000, 1500, 2500],
    ['Apr', 0, 6500, 2000, 4000],
    ['May', 0, 3500, 5500, 3500],
    ['Jun', 0, 0, 7500, 1500],
```



```

    ['Jul', 0, 0, 8500, 800],
    ['Aug', 1500, 0, 7000, 550],
    ['Sep', 5000, 0, 3500, 2500],
    ['Oct', 8500, 0, 2500, 6000],
    ['Nov', 3500, 0, 500, 5500],
    ['Dec', 500, 0, 100, 3000],
]

for row in rows:
    ws.append(row)

chart = RadarChart()
chart.type = "filled"
labels = Reference(ws, min_col=1, min_row=2, max_row=13)
data = Reference(ws, min_col=2, max_col=5, min_row=1,
max_row=13)
chart.add_data(data, titles_from_data=True)
chart.set_categories(labels)
chart.style = 26
chart.title = "Garden Centre Sales"
chart.y_axis.delete = True

ws.add_chart(chart, "A17")
wb.save("radar.xlsx")

```

运行上述代码后的结果如下图 19.1 所示。



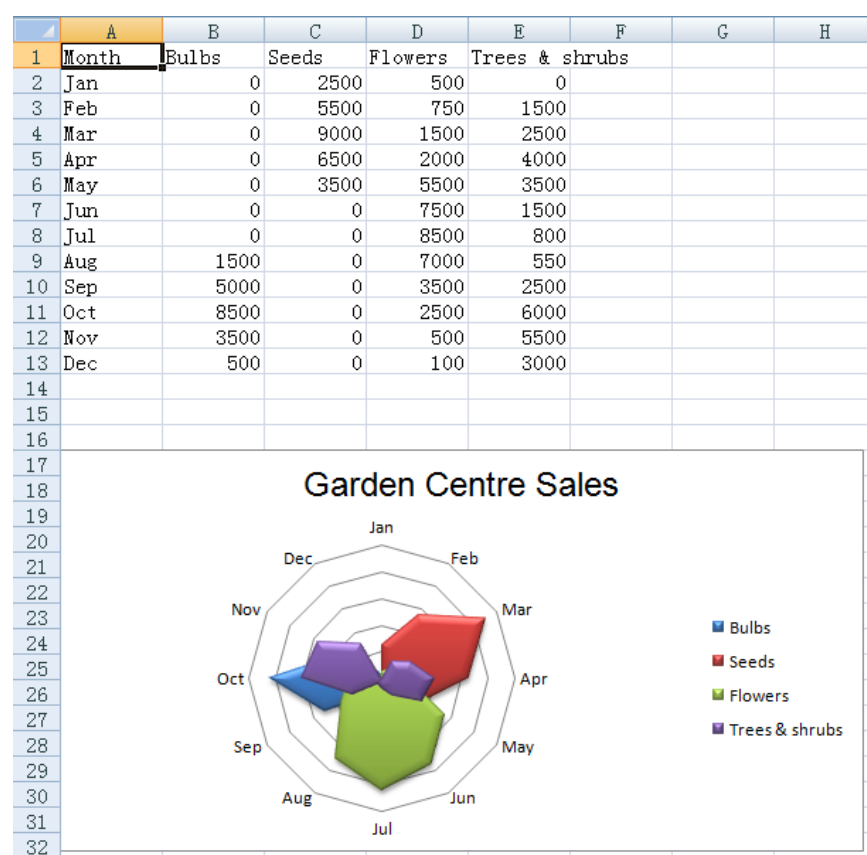


图 19.1





本章内容 2018 年 10 月 7 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(20\): 股价图](#)

20.股价图

以特定顺序排列在工作表列或行中的数据能够绘制股价图。正如其名字所暗示的，股价图经常用于表示股价的波动。然而，这类图表也能用于科学数据。例如，可以使用股价图来指示每日或每年温度的波动。

必须按正确的顺序组织数据来创建股价图。在工作表中组织股价图数据的方式非常重要，例如，要创建简单的高-低-收盘股价图，应该按照该顺序整理数据并输入相应的列标题。

不同形式股价图的特定格式选项：

- 高-低-收盘形式股价图实际上是一个不带线条的折线图，其标记设置为 XYZ，并将 hiLoLines 设置为 True。
- 开盘-高-低-收盘形式股价图就像一个带有标记的高-低-收盘股价图，每个数据点设置为 XYZ 和 upDownLines。

可以通过组合股价图和条形图来添加表示成交量的条形。

基本的代码如下：

```
from datetime import date
from openpyxl import Workbook

from openpyxl.chart import (
    BarChart,
    StockChart,
```



```

        Reference,
        Series,
    )

from openpyxl.chart.axis import DateAxis, ChartLines
from openpyxl.chart.updown_bars import UpDownBars

wb = Workbook()
ws = wb.active

rows = [
    ['日期', '成交量', '开盘', '高', '低', '收盘'],
    ['2018-01-01', 20000, 26.20, 27.20, 23.49, 25.45, ],
    ['2018-01-02', 10000, 25.45, 25.03, 19.55, 23.05, ],
    ['2018-01-03', 15000, 23.05, 24.46, 20.03, 22.42, ],
    ['2018-01-04', 2000, 22.42, 23.97, 20.07, 21.90, ],
    ['2018-01-05', 12000, 21.9, 23.65, 19.50, 21.51, ],
]

for row in rows:
    ws.append(row)

# 高-低-收盘
c1 = StockChart()
labels = Reference(ws, min_col=1, min_row=2, max_row=6)
data = Reference(ws, min_col=4, max_col=6, min_row=1, max_row=6)
c1.add_data(data, titles_from_data=True)
c1.set_categories(labels)
for s in c1.series:
    s.graphicalProperties.line.noFill = True

# 标记收盘
s.marker.symbol = "dot"

```



```

s.marker.size = 5
c1.title = "高-低-收盘"
c1.hiLowLines = ChartLines()

from openpyxl.chart.data_source import NumData, NumVal
pts = [NumVal(idx=i) for i in range(len(data) - 1)]
cache = NumData(pt=pts)
c1.series[-1].val.numRef.numCache = cache

ws.add_chart(c1, "A10")

# 开盘-高-低-收盘
c2 = StockChart()
data = Reference(ws, min_col=3, max_col=6, min_row=1,
max_row=6)
c2.add_data(data, titles_from_data=True)
c2.set_categories(labels)
for s in c2.series:
    s.graphicalProperties.line.noFill = True
c2.hiLowLines = ChartLines()
c2.upDownBars = UpDownBars()
c2.title = "开盘-高-低-收盘"

c2.series[-1].val.numRef.numCache = cache

ws.add_chart(c2, "G10")

# 创建条形图代表成交量
bar = BarChart()
data = Reference(ws, min_col=2, min_row=1, max_row=6)
bar.add_data(data, titles_from_data=True)
bar.set_categories(labels)

```



```

from copy import deepcopy

# 成交量-高-低-收盘价
b1 = deepcopy(bar)
c3 = deepcopy(c1)
c3.y_axis.majorGridlines = None
c3.y_axis.title = "价格"
b1.y_axis.axId = 20
b1.z_axis = c3.y_axis
b1.y_axis.crosses = "max"
b1 += c3

c3.title = "高 低 收盘 成交量"

ws.add_chart(b1, "A27")

# 成交量-开盘-高-低-收盘
b2 = deepcopy(bar)
c4 = deepcopy(c2)
c4.y_axis.majorGridlines = None
c4.y_axis.title = "价格"
b2.y_axis.axId = 20
b2.z_axis = c4.y_axis
b2.y_axis.crosses = "max"
b2 += c4

ws.add_chart(b2, "G27")
wb.save("stock.xlsx")

```

运行上述代码后的结果如下图 20.1 所示。



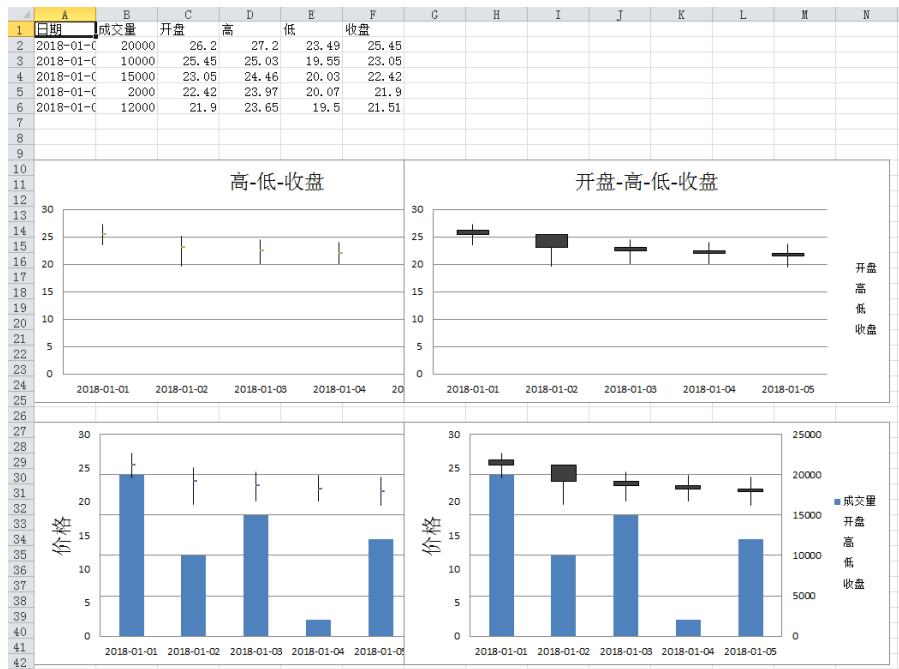


图 20.1





本章内容 2018 年 10 月 14 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(21\): 曲面图](#)

21. 曲面图

整理在工作表行或列中的数据可以绘制成曲面图。当你想在两组数据间找到最佳组合时，曲面图就非常有用。与地形图一样，颜色和图案表示相同范围值的区域。

默认情形下，所有曲面图都是三维图表。通过设置旋转和透视，创建二维线框和等高线图。

创建曲面图的 Python 代码如下：

```
from openpyxl import Workbook
from openpyxl.chart import (
    SurfaceChart,
    SurfaceChart3D,
    Reference,
    Series,
)
from openpyxl.chart.axis import SeriesAxis

wb = Workbook()
ws = wb.active

data = [
    [None, 10, 20, 30, 40, 50,],
    [0.1, 15, 65, 105, 65, 15,],
    [0.2, 35, 105, 170, 105, 35,],
    [0.3, 55, 135, 215, 135, 55,],
```



```

    [0.4, 75, 155, 240, 155, 75,],
    [0.5, 80, 190, 245, 190, 80,],
    [0.6, 75, 155, 240, 155, 75,],
    [0.7, 55, 135, 215, 135, 55,],
    [0.8, 35, 105, 170, 105, 35,],
    [0.9, 15, 65, 105, 65, 15],
]

for row in data:
    ws.append(row)

c1 = SurfaceChart()
ref = Reference(ws, min_col=2, max_col=6, min_row=1,
max_row=10)
labels = Reference(ws, min_col=1, min_row=2, max_row=10)
c1.add_data(ref, titles_from_data=True)
c1.set_categories(labels)
c1.title = "轮廓"

ws.add_chart(c1, "A12")

from copy import deepcopy

# 线框
c2 = deepcopy(c1)
c2.wireframe = True
c2.title = "二维线框"

ws.add_chart(c2, "G12")

# 三维表面
c3 = SurfaceChart3D()

```




```

c3.add_data(ref, titles_from_data=True)
c3.set_categories(labels)
c3.title = "曲面"

ws.add_chart(c3, "A29")

c4 = deepcopy(c3)
c4.wireframe = True
c4.title = "三维线框"

ws.add_chart(c4, "G29")
wb.save("surface.xlsx")

```

运行上述代码后的结果如下图 21.1 所示。

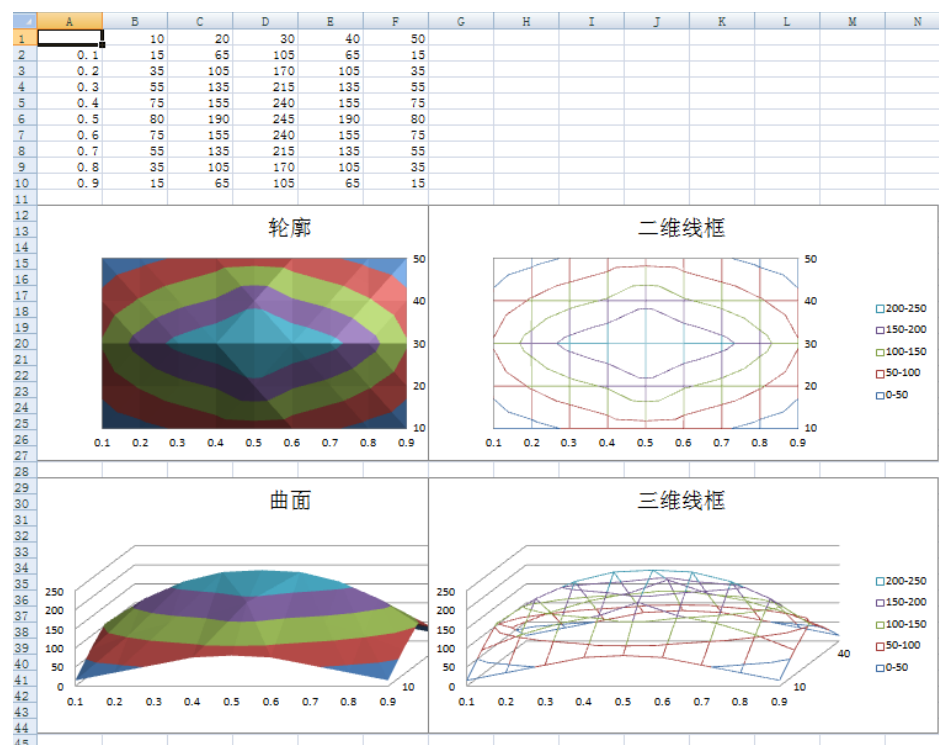


图 21.1





本章内容 2018 年 10 月 21 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(22\): 添加
样式](#)

22.添加样式

图表的整个数据系列和单个的数据点可以通过 `graphicalProperties` 来进行扩展的样式设置，当然，要做到恰到好处可能需要一些时间。

基本代码如下：

```
from openpyxl import Workbook
from openpyxl.chart import BarChart, Reference
from openpyxl.chart.marker import DataPoint

from openpyxl.drawing.fill import PatternFillProperties,
ColorChoice

wb = Workbook()
ws = wb.active

rows = [
    ("示例",),
    (1,),
    (2,),
    (3,),
    (2,),
    (3,),
    (3,),
    (1,),
    (2,)
```



```

]

for r in rows:
    ws.append(r)

c = BarChart()
data = Reference(ws, min_col=1, min_row=1, max_row=8)
c.add_data(data, titles_from_data=True)
c.title = "带有样式的图表"

# 对整个系列设置样式
series = c.series[0]
fill = PatternFillProperties(prst="pct5")
fill.foreground = ColorChoice(prstClr="red")
fill.background = ColorChoice(prstClr="blue")
series.graphicalProperties.pattFill = fill

# 对第 6 个数据点设置样式
pt = DataPoint(idx=5)
pt.graphicalProperties.pattFill
PatternFillProperties(prst="ltHorz")
series.dPt.append(pt)

ws.add_chart(c, "C1")
wb.save("pattern.xlsx")

```

运行代码后，生成的工作簿中的图表如下图 22.1 所示。

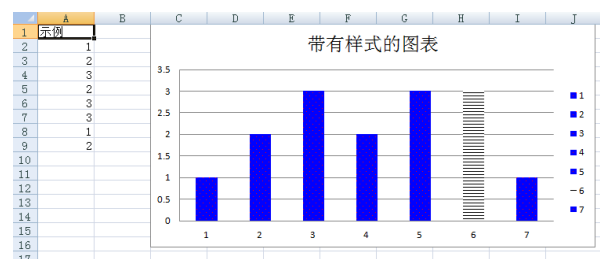


图 22.1



本章内容 2018 年 10 月 28 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(23\): 绘制仪表盘](#)

23.绘制仪表盘

可以组合了饼图和圆环图来创建仪表盘图表。

第一个图是带有 4 个切片的圆环图，前 3 个切片与仪表盘的颜色对应，第 4 个切片是圆环的另一半，设置为不可见。

然后，添加含有 3 个切片的饼图，将第 1 个和第 3 个切片设置为不可见，将第 2 个切片当作仪表盘的指针。

使用数据系列中各个数据点的图形属性来实现效果。代码如下：

```
from openpyxl import Workbook

from openpyxl.chart import PieChart, DoughnutChart, Series,
Reference

from openpyxl.chart.series import DataPoint

data = [
    ["Donut", "Pie"],
    [25, 75],
    [50, 1],
    [25, 124],
    [100],
]

wb = Workbook()
```



```

ws = wb.active
for row in data:
    ws.append(row)

# 第一个图是圆环图
c1 = DoughnutChart(firstSliceAng=270, holeSize=50)
c1.title = "Code coverage"
c1.legend = None

ref = Reference(ws, min_col=1, min_row=2, max_row=5)
s1 = Series(ref, title_from_data=False)

slices = [DataPoint(idx=i) for i in range(4)]
slices[0].graphicalProperties.solidFill = "FF3300" #红色
slices[1].graphicalProperties.solidFill = "FCF305" #黄色
slices[2].graphicalProperties.solidFill = "1FB174" #绿色
slices[3].graphicalProperties.noFill = True #不可见

s1.data_points = slices
c1.series = [s1]

# 第二个图是饼图
c2 = PieChart(firstSliceAng=270)
c2.legend = None

ref = Reference(ws, min_col=2, min_row=2, max_col=2,
max_row=4)
s2 = Series(ref, title_from_data=False)

slices = [DataPoint(idx=i) for i in range(3)]
slices[0].graphicalProperties.noFill = True #不可见
slices[1].graphicalProperties.solidFill = "000000" #黑色针

```



状

```
slices[2].graphicalProperties.noFill = True #不可见
```

```
s2.data_points = slices
```

```
c2.series = [s2]
```

```
c1 += c2 # 组合图表
```

```
ws.add_chart(c1, "D1")
```

```
wb.save("gauge.xlsx")
```

运行代码后，生成的工作簿中的图表如下图 23.1 所示。

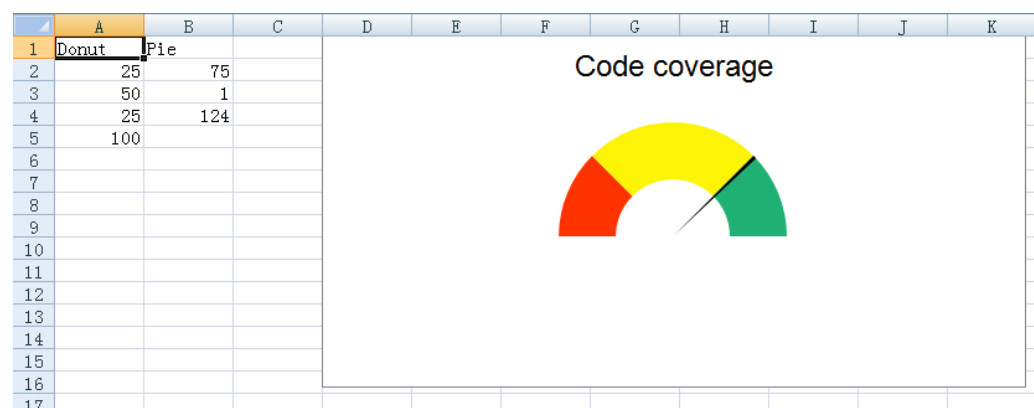


图 23.1





本章内容 2018 年 11 月 5 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(24\): 图表工作表](#)

24. 图表工作表

在前面的一系列章节中，我们花了大量的篇章介绍使用 Python 进行 Excel 图表操作的基础代码和应用示例，并分别详细讲解了在 Excel 中绘制面积图、条形图和柱状图、气泡图、折线图、散点图、饼图、圆环图、雷达图、股价图、曲面图等 Python 基本代码，也介绍了如何使用代码来调整图表坐标轴、图表布局、添加样式，并给出了一个绘制仪表盘的示例代码。

图表工作表是一种特殊的工作表，它仅包含图表，绘制图表的数据在另外的工作表中。下面是使用 Python 绘制 Excel 图表工作表的基础代码示例：

```
from openpyxl import Workbook
from openpyxl.chart import PieChart, Reference, Series

wb = Workbook()
ws = wb.active
cs = wb.create_chartsheet()

rows = [
    ["苹果", 30],
    ["香蕉", 20],
    ["脐橙", 40],
]

for row in rows:
    ws.append(row)
```



```
chart = PieChart()  
labels = Reference(ws, min_col=1, min_row=1, max_row=3)  
data = Reference(ws, min_col=2, min_row=1, max_row=3)  
chart.series = (Series(data),)  
chart.title = "饼图"  
  
cs.add_chart(chart)  
  
wb.save("demochartsheet.xlsx")
```

运行上述代码后，结果如下图 24.1 所示。

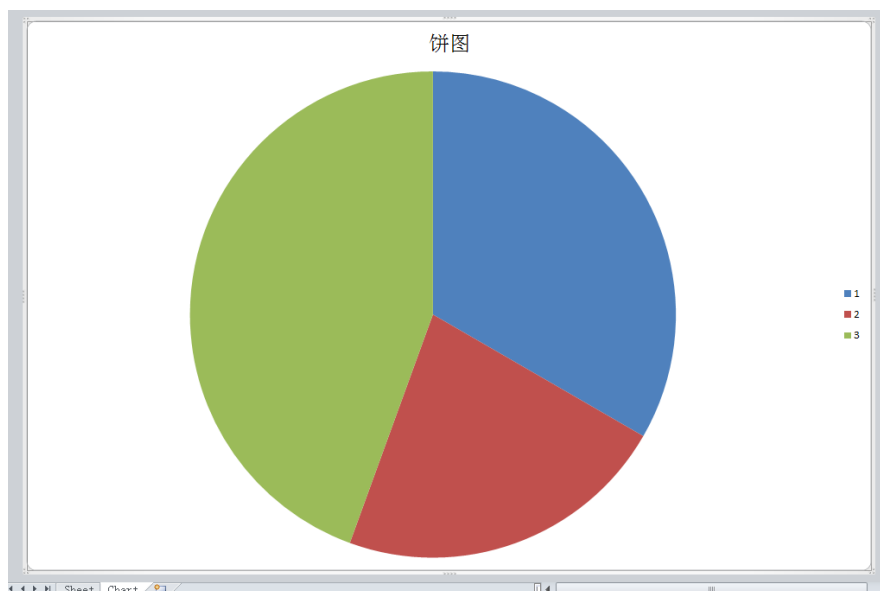


图 24.1



本章内容 2018 年 11 月 11 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(25\): 给单元格添加批注](#)

25.给单元格添加批注

在 Excel 中，我们可以给单元格添加批注，以说明该单元格的作用、数据的用途、或者存放一些信息。同样，可以使用 Python 来给工作表单元格添加批注。

下面的代码给工作表中的单元格 A1 添加批注，同时设置了批注的 text 属性和 author 属性：

```
>>> from openpyxl import Workbook
>>> from openpyxl.comments import Comment
>>> wb = Workbook()
>>> ws = wb.active
>>> comment = ws["A1"].comment
>>> comment = Comment('这是批注文本', '完美 Excel')
>>> comment.text
'这是批注文本'
>>> comment.author
'完美 Excel'
```

如果给多个单元格添加相同的批注，那么 openpyxl 将自动创建批注的副本，如下代码所示：

```
>>> from openpyxl import Workbook
>>> from openpyxl.comments import Comment
>>> wb = Workbook()
>>> ws = wb.active
>>> comment = Comment("批注文本", "作者")
>>> ws["A1"].comment = comment
```



```
>>> ws["B2"].comment = comment
>>> ws["A1"].comment is comment
True
>>> ws["B2"].comment is comment
False
```

在装载工作簿时，批注会自动存储在各自单元格的 `comment` 属性中，其格式化信息，例如字体大小、粗体和斜体将丢失，批注框的原始大小和位置也会丢失。

在保存工作簿时，工作簿中的批注会自动保存到工作簿文件中。

可以为批注指定大小，单位为像素。

下面的代码为工作表单元格 A1 创建批注，并指定批注框的大小。

```
from openpyxl import Workbook
from openpyxl.comments import Comment
from openpyxl.utils import units

wb = Workbook()
ws = wb.active

comment = Comment("批注文本", "完美 Excel")
comment.width = 300
comment.height = 50

ws["A1"].comment = comment

wb.save("commentssample.xlsx")
```

运行代码后的结果如下图 25.1 所示。



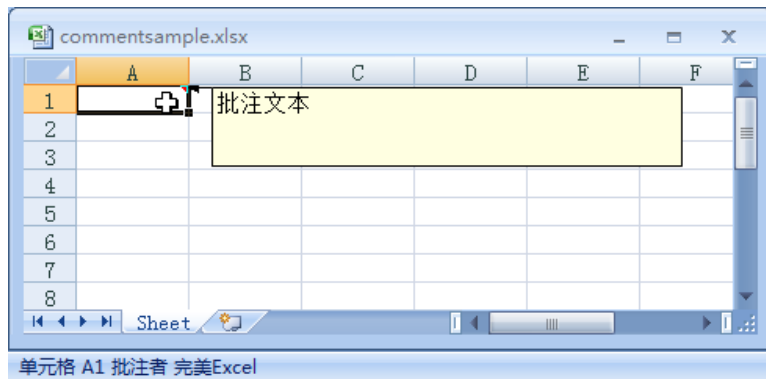


图 25.1

如果需要，`openpyxl.utils.units` 包含的辅助函数可以实现将其它的度量如毫米或磅转换为像素，如下代码所示：

```
>>> from openpyxl import Workbook
>>> from openpyxl.comments import Comment
>>> from openpyxl.utils import units
>>>
>>> wb = Workbook()
>>> ws = wb.active
>>>
>>> comment = Comment("批注文本", "完美 Excel")
>>> comment.width = units.points_to_pixels(300)
>>> comment.height = units.points_to_pixels(50)
>>>
>>> ws["A1"].comment = comment
```

说明：openpyxl 当前仅支持批注文本读写，格式化信息将会丢失。批注框尺寸在读取时会丢失，但可以写入。如果设置 `read_only=True`，那么当前不支持批注。





本章内容 2018 年 11 月 18 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(26\): 处理
样式](#)

26.处理样式

样式用于改变数据在屏幕上显示的外观，也用于确定数字的格式。通常，样式应用于：

- 设置字体大小、颜色、下划线等
- 设置填充的样式或颜色渐变
- 设置单元格边框
- 单元格对齐
- 保护

下面是应用样式的默认值：

```
>>> from openpyxl.styles import PatternFill, Border, Side,
Alignment, Protection, Font
>>> font = Font(name='Calibri',
                size=11,
                bold=False,
                italic=False,
                vertAlign=None,
                underline='none',
                strike=False,
                color='FF000000')
>>> fill = PatternFill(fill_type=None,
                       start_color='FFFFFFFF',
                       end_color='FF000000')
>>> border = Border(left=Side(border_style=None,
                               color='FF000000'),
```



```

        right=Side(border_style=None,
                    color='FF000000'),
        top=Side(border_style=None,
                  color='FF000000'),
        bottom=Side(border_style=None,
                     color='FF000000'),
        diagonal=Side(border_style=None,
                       color='FF000000'),
        diagonal_direction=0,
        outline=Side(border_style=None,
                      color='FF000000'),
        vertical=Side(border_style=None,
                       color='FF000000'),
        horizontal=Side(border_style=None,
                         color='FF000000')
    )

>>> alignment = Alignment(horizontal='general',
                            vertical='bottom',
                            text_rotation=0,
                            wrap_text=False,
                            shrink_to_fit=False,
                            indent=0)

>>> number_format = 'General'

>>> protection = Protection(locked=True,
                              hidden=False)

```

单元格样式

单元格样式在对象之间共享，并且一旦被赋予该样式，就无法更改。这样可以阻止不必要的副作用，例如当仅替换其中一个时而更改大量单元格的样式。

```
>>> from openpyxl.styles import colors
```




```

>>> from openpyxl.styles import Font, Color
>>> from openpyxl import Workbook
>>> wb = Workbook()
>>> ws = wb.active
>>> a1 = ws['A1']
>>> d4 = ws['D4']
>>> ft = Font(color=colors.RED)
>>> a1.font = ft
>>> d4.font = ft
>>> a1.font.italic = True #不被允许
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a1.font.italic = True #不被允许
  File
"C:\Users\Administrator\AppData\Local\Programs\Python\Python37\lib\site-packages\openpyxl\styles\proxy.py", line 30,
in __setattr__
    raise AttributeError("Style objects are immutable and
cannot be changed.")
AttributeError: Style objects are immutable and cannot be
changed.Reassign the style with a copy
>>> #如果想要修改字体颜色，需要重新赋值
>>> a1.font = Font(color=colors.RED, italic=True) #修改仅作
用于 A1

```

复制样式

可以复制样式：

```

>>> from openpyxl.styles import Font
>>> from copy import copy
>>> ft1 = Font(name='Arial', size=14)
>>> ft2 = copy(ft1)

```



```
>>> ft2.name = "Tahoma"
>>> ft1.name
'Arial'
>>> ft2.name
'Tahoma'
>>> ft2.size
14.0
```

基本的字体颜色

颜色通常是 RGB 或 aRGB 十六进制值，colors 模块包含一些方便的常量。

```
>>> from openpyxl.styles import Font
>>> from openpyxl.styles.colors import RED
>>> font = Font(color=RED)
>>> font = Font(color="FFBB00")
```

此外，也支持合法的索引颜色以及主题和色调：

```
>>> from openpyxl.styles.colors import Color
>>> c = Color(indexed=32)
>>> c = Color(theme=6, tint=0.5)
```

应用样式

直接将样式应用到单元格中：

```
>>> from openpyxl.workbook import Workbook
>>> from openpyxl.styles import Font, Fill
>>> wb = Workbook()
>>> ws = wb.active
```



```
>>> c = ws['A1']
>>> c.font = Font(size=12)
```

样式也可应用于列和行，但注意这仅适用于文件关闭后创建的单元格。如果要将样式应用至整行和整列，则必须自己将样式应用于每个单元格，这是文件格式的限制。

```
>>> col = ws.column_dimensions['A']
>>> col.font = Font(bold=True)
>>> row = ws.row_dimensions[1]
>>> row.font = Font(underline="single")
```

样式化合并单元格

有时想要格式化单元格区域，就好像它们是单个对象一样。Excel 假装这是可能的，通过合并单元格（删除除单元格区域左上角单元格之外的所有单元格）然后重新创建它们以便应用伪样式。

```
from openpyxl.styles import Border, Side, PatternFill, Font,
GradientFill, Alignment
```

```
from openpyxl import Workbook
```

```
def style_range(ws, cell_range, border=Border(), fill=None,
font=None, alignment=None):
```

```
    """
```

```
    应用样式到单元格区域,就好像它们是单个单元格
```

```
    :参数 ws: Excel 工作表实例
```

```
    :参数 range: 应用样式的 Excel 单元格区域 (例如 A1:F20)
```

```
    :参数 border: openpyxl 边框
```

```
    :参数 fill: openpyxl 样式填充或渐变填充
```

```
    :参数 font: openpyxl 字体对象
```

```
    """
```



```

top = Border(top=border.top)
left = Border(left=border.left)
right = Border(right=border.right)
bottom = Border(bottom=border.bottom)

first_cell = ws[cell_range.split(":")[0]]
if alignment:
    ws.merge_cells(cell_range)
    first_cell.alignment = alignment

rows = ws[cell_range]
if font:
    first_cell.font = font

for cell in rows[0]:
    cell.border = cell.border + top
for cell in rows[-1]:
    cell.border = cell.border + bottom

for row in rows:
    l = row[0]
    r = row[-1]
    l.border = l.border + left
    r.border = r.border + right
    if fill:
        for c in row:
            c.fill = fill

wb = Workbook()
ws = wb.active
my_cell = ws['B2']
my_cell.value = "My Cell"

```



```

thin = Side(border_style="thin", color="000000")
double = Side(border_style="double", color="ff0000")

border = Border(top=double, left=thin, right=thin,
bottom=double)

fill = PatternFill("solid", fgColor="DDDDDD")
fill = GradientFill(stop=("000000", "FFFFFF"))
font = Font(b=True, color="FF0000")
a1 = Alignment(horizontal="center", vertical="center")

style_range(ws, 'B2:F4', border=border, fill=fill,
font=font, alignment=a1)
wb.save("styled.xlsx")

```

运行上述代码后的结果如下图 26.1 所示。

	A	B	C	D	E	F	G
1							
2		My Cell					
3							
4							
5							
6							

图 26.1

编辑页面设置

```

>>> from openpyxl.workbook import Workbook
>>> wb = Workbook()
>>> ws = wb.active
>>> ws.page_setup.orientation = ws.ORIENTATION_LANDSCAPE
>>> ws.page_setup.paperSize = ws.PAPERSIZE_TABLOID
>>> ws.page_setup.fitToHeight = 0
>>> ws.page_setup.fitToWidth = 1

```



命名样式

与单元格样式相比，命名样式是可变的。它们特别适用于想要一次将格式应用于多个不同单元格时。一旦为某单元格赋予了命名样式，对样式的其他更改将不会影响该单元格。

一旦将命名样式注册到工作簿，就可以简单地通过名称来引用它。

```
>>> from openpyxl.styles import NamedStyle, Font, Border, Side
>>> highlight = NamedStyle(name="highlight")
>>> highlight.font = Font(bold=True, size=20)
>>> bd = Side(style='thick', color="000000")
>>> highlight.border = Border(left=bd, top=bd, right=bd, bottom=bd)
```

一旦创建了命名样式，就可将它注册到工作簿：

```
>>> wb.add_name_style(highlight)
```

然而，当它们首次赋给单元格时，命名样式也会自动注册：

```
>>> ws['A1'].style = highlight
```

注册后，仅使用名称赋予来指定样式：

```
>>> ws['D5'].style = 'highlight'
```

使用内置样式

规范包括一些可以使用的内置样式，但是这些样式名称以其本地化形式存储。openpyxl 只能识别英文名并且只能这样写。如下：

- 'Normal'

数字格式



- 'Comma'
- 'Comma[0]'
- 'Currency'
- 'Currency[0]'
- 'Percent'

信息

- 'Calculation'
- 'Total'
- 'Note'
- 'Warning Text'
- 'Explanatory Text'

文本样式

- 'Title'
- 'Headline 1'
- 'Headline 2'
- 'Headline 3'
- 'Headline 4'
- 'Hyperlink'
- 'Followed Hyperlink'
- 'Linked Cell'

比较

- 'Input'
- 'Output'
- 'Check Cell'
- 'Good'
- 'Bad'
- 'Neutral'



高亮

- 'Accent1'
- '20%-Accent1'
- '40%-Accent1'
- '60%-Accent1'
- 'Accent2'
- '20%-Accent2'
- '40%-Accent2'
- '60%-Accent2'
- 'Accent3'
- '20%-Accent3'
- '40%-Accent3'
- '60%-Accent3'
- 'Accent4'
- '20%-Accent4'
- '40%-Accent4'
- '60%-Accent4'
- 'Accent5'
- '20%-Accent5'
- '40%-Accent5'
- '60%-Accent5'
- 'Accent6'
- '20%-Accent6'
- '40%-Accent6'
- '60%-Accent6'
- 'Pandas'



本章内容 2018 年 11 月 25 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(27\): 一些工
作表属性](#)

27. 一些工作表属性

有一些高级属性对应于特定的工作表操作，最常使用的是“fitToPage”页面设置属性和“tabColor”来定义工作表标签的背景颜色。

工作表的可用属性

- “enableFormatConditionsCalculation”——启动条件格式计算
- “filterMode”——筛选模式
- “published”——发布
- “syncHorizontal”——水平同步
- “syncRef”——同步引用
- “syncVertical”——垂直同步
- “transitionEvaluation”——转换为 Lotus1-2-3 公式评估值
- “transitionEntry”——转换成 Lotus1-2-3 公式
- “tabColor”——工作表标签颜色



页面设置可用字段

- "autoPageBreaks"——自动分页
- "fitToPage"——缩放到页

大纲可用字段

- "applyStyles"——应用样式
- "summaryBelow"——下方汇总
- "summaryRight"——右侧汇总
- "showOutlineSymbols"——显示大纲符号

说明：缺省情况下，大纲属性被初始化以便你可以直接修改其 4 个属性中的任一个，而页面设置属性则不能。如果你想修改页面设置属性，则应该首先使用所需参数初始化 `openpyxl.worksheet.properties.PageSetupProperties` 对象。初始化后，如果需要则可在以后通过程序直接修改。

下面是设置工作表属性的一些 Python 示例代码：

```
from openpyxl.workbook import Workbook

from openpyxl.worksheet.properties import
WorksheetProperties, PageSetupProperties

wb = Workbook()
ws = wb.active

wsprops = ws.sheet_properties
wsprops.tabColor = "1072BA"
```



```
wsprops.filterMode = False
wsprops.pageSetUpPr = PageSetupProperties(fitToPage=True,
autoPageBreaks=False)
wsprops.outlinePr.summaryBelow = False
wsprops.outlinePr.applyStyles = True
wsprops.pageSetUpPr.autoPageBreaks = True
wb.save("properties.xlsx")
```





本章内容 2018 年 12 月 2 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(28\): 条件格式](#)

28. 条件格式

Excel 支持 3 种不同类型的条件格式：内置、标准和自定义。内置条件格式将特定规则与预定义样式相组合。标准条件格式将特定规则与自定义格式相组合。此外，可以定义自定义公式来应用使用不同样式的自定义格式。

创建条件格式的基本语法为：

```
>>> from openpyxl.formatting import Rule
>>> from openpyxl.styles import Font, PatternFill, Border
>>> from openpyxl.styles.differential import DifferentialStyle
>>> dxf = DifferentialStyle(font=Font(bold=True),
    fill=PatternFill(start_color='EE1111',
    end_color='EE1111'))
>>> rule = Rule(type='cellIs', dxf=dxf, formula=["10"])
```

内置格式

内置条件格式有：

- ColorScale（色阶）
- IconSet（图标集）
- DataBar（数据条）



内置格式包含一系列格式设置，这些设置将类型与整数组合以进行比较。可能的类型有：“数字”、“百分比”、“最大值”、“最小值”、“公式”、“百分点值”。

ColorScale (色阶)

可以使用 2 种或 3 种颜色的色阶。2 个色阶产生从一种颜色到另一种颜色的渐变，3 个色阶使用额外颜色产生 2 种颜色渐变。

创建色阶规则的完整语法是：

```
>>> from openpyxl.formatting.rule import ColorScale, FormatObject
>>> from openpyxl.styles import Color
>>> first = FormatObject(type='min')
>>> last = FormatObject(type='max')
>>> # 颜色与格式对象匹配
>>> colors = [Color('AA0000'), Color('00AA00')]
>>> cs2 = ColorScale(cfvo=[first, last], color=colors)
>>> # 三色阶将扩展序列
>>> mid = FormatObject(type='num', val=40)
>>> colors.insert(1, Color('00AA00'))
>>> cs3 = ColorScale(cfvo=[first, mid, last], color=colors)
>>> # 创建使用色阶的规则
>>> from openpyxl.formatting.rule import Rule
>>> rule = Rule(type='colorScale', colorScale=cs3)
```

下面是创建色阶规则的便利的方法：

```
>>> from openpyxl.formatting.rule import ColorScaleRule
>>> rule = ColorScaleRule(start_type='percentile',
                          start_value=10,
                          start_color='FFAA0000',
                          mid_type='percentile',
                          mid_value=50,
                          mid_color='FF0000AA',
```



```
end_type='percentile',
end_value=90,
end_color='FF00AA00')
```

IconSet (图标)

从下列图标集中选择：“3Arrows 三向箭头”、“3ArrowsGray 三向箭头（灰色）”、“3Flags 三色旗”、“3TrafficLights1 三色交通灯 1”、“3TrafficLights2 三色交通灯 2”、“3Signs 三标志”、“3Symbols 三个符号”、“3Symbols2 三个符号 2”、“4Arrows 四向箭头”、“4ArrowsGray 四向箭头（灰色）”、“4RedToBlack 红黑渐变”、“4Rating 四等级”、“4TrafficLights 四色交通灯”、“5Arrows 五向箭头”、“5ArrowsGray 五向箭头（灰色）”、“5Rating 五等级”、“5Quarters 五象限图”。

创建图标规则的完整语法是：

```
>>> from openpyxl.formatting.rule import IconSet,
FormatObject
>>> first = FormatObject(type='percent', val=0)
>>> second = FormatObject(type='percent', val=33)
>>> third = FormatObject(type='percent', val=67)
>>> iconset = IconSet(iconSet='3TrafficLights1',
cfvo=[first, second, third], showValue=None, percent=None,
reverse=None)
>>> # 将图标集赋给规则
>>> from openpyxl.formatting.rule import Rule
>>> rule = Rule(type='iconSet', iconSet=iconset)
```

下面是创建图标集规则的便利方法：

```
>>> from openpyxl.formatting.rule import IconSetRule
>>> rule = IconSetRule('5Arrows', 'percent', [10, 20, 30,
40, 50], showValue=None, percent=None, reverse=None)
```

DataBar (数据条)

当前，openpyxl 支持原始规范中定义的数据条，在以后的扩展中将会添加边框和方向。

创建数据条规则的完整语法是：



```

>>> from openpyxl.formatting.rule import DataBar,
FormatObject
>>> first = FormatObject(type='min')
>>> second = FormatObject(type='max')
>>> data_bar = DataBar(cfvo=[first, second],
color="638EC6", showValue=None, minLength=None,
maxLength=None)
>>> # 将数据条赋给规则
>>> from openpyxl.formatting.rule import Rule
>>> rule = Rule(type='dataBar', dataBar=data_bar)

```

下面是创建数据条规则的便利方法：

```

>>> from openpyxl.formatting.rule import DataBarRule
>>> rule = DataBarRule(start_type='percentile',
                        start_value=10,
                        end_type='percentile',
                        end_value='90',
                        color="FF638EC6",
                        showValue="None",
                        minLength=None,
                        maxLength=None)

```

标准条件格式

标准条件格式有：

- 平均值
- 百分比
- 唯一值或重复值
- 值
- 等级



示例

下面是在单元格区域中应用条件格式的示例代码：

```
from openpyxl import Workbook

from openpyxl.styles import Color, PatternFill, Font, Border

from openpyxl.styles.differential import DifferentialStyle

from openpyxl.formatting.rule import ColorScaleRule, CellIsRule, FormulaRule

from openpyxl.formatting import Rule

wb = Workbook()
ws = wb.active

# 创建填充
redFill = PatternFill(start_color='EE1111',
                      end_color='EE1111',
                      fill_type='solid')

# 添加二色色阶
# 采用 Excel 'RRGGBB' 样式的颜色
ws.conditional_formatting.add('A1:A10',
                              ColorScaleRule(start_type='min',
                                              start_color='AA0000',
                                              end_type='max',
                                              end_color='00AA00')
                              )

# 添加三色色阶
ws.conditional_formatting.add('B1:B10',
                              ColorScaleRule(start_type='percentile',
                                              start_value=10,
```



```

start_color='AA0000',

mid_type='percentile',

mid_value=50,
mid_color='0000AA',

end_type='percentile',

end_value=90,
end_color='00AA00')

)

# 基于单元格比较添加条件格式
# addCellIs(range_string, operator, formula, stopIfTrue,
wb, font, border, fill)
# 如果单元格小于'公式'则应用格式
ws.conditional_formatting.add('C2:C10',
                                CellIsRule(operator='lessThan',
                                              formula=['C$1'],
                                              stopIfTrue=True,
                                              fill=redFill))

# 如果单元格在'公式'之间则应用格式
ws.conditional_formatting.add('D2:D10',
                                CellIsRule(operator='between',
                                              formula=['1', '5'],
                                              stopIfTrue=True,
                                              fill=redFill))

# 使用公式的格式
ws.conditional_formatting.add('E1:E10',
                                FormulaRule(formula=['ISBLANK(E1)'],
                                              stopIfTrue=True,
                                              fill=redFill))

```



```
# 除了 2 个色阶和 3 个色阶外,格式规则还采用字体、边填和填充样式
myFont = Font()
myBorder = Border()
ws.conditional_formatting.add('E1:E10',
                                FormulaRule(formula=['E1=0'],
                                                font=myFont,
                                                border=myBorder,
                                                fill=redFill))

# 使用指定公式突出显示包含特定文本的单元格
red_text = Font(color="9C0006")
red_fill = PatternFill(bgColor="FFC7CE")
dxf = DifferentialStyle(font=red_text, fill=red_fill)
rule = Rule(type="containsText", operator="containsText",
            text="highlight", dxf=dxf)
rule.formula = ['NOT(ISERROR(SEARCH("完美 Excel",A1)))']
ws.conditional_formatting.add('A1:F40', rule)
wb.save("cftest.xlsx")
```

运行代码后的效果如下图 28.1 所示。

	A	B	C	D	E	F	G
1	完美Excel	1	5	1			
2	2	2	3	2			
3	3	3	6	3	完美Excel		
4	4	4	7	4			
5	5	5	8	5	excelperfect	完美Excel	
6	6	6	2	6			
7	7	7	1	7	excelperfect	完美Excel	
8	8	8	9	8			
9	9	9		9			
10	10	10	2	0			
11							
12							
13		完美Excel		完美Excel		完美Excel	
14	完美Excel		完美Excel		完美Excel		
15							
16							

图 28.1





本章内容 2018 年 12 月 9 日首发于
[完美 Excel]微信公众号 *excelperfect*
原标题为
[Python 操作 Excel 学习笔记 \(29\): 打印设置](#)

29.打印设置

openpyxl 为打印设置提供了非常完整的支持。

编辑打印选项

下面是设置水平居中和垂直居中打印的代码：

```
>>> from openpyxl.workbook import Workbook
>>> wb = Workbook()
>>> ws = wb.active
>>> ws.print_options.horizontalCentered = True
>>> ws.print_options.verticalCentered = True
```

页眉和页脚

支持左侧、居中或右侧元素的字体、大小和颜色。突出显示个别词语需要手动应用控制代码。

下面的代码设置工作表页眉：

```
from openpyxl.workbook import Workbook

wb = Workbook()
ws = wb.active
```



```
ws.oddHeader.left.text = "第 &[amp;Page] 页,共 &N 页"
ws.oddHeader.left.size = 14
ws.oddHeader.left.font = "Tahoma,Bold"
ws.oddHeader.left.color = "CC3366"

wb.save("header.xlsx")
```

运行代码后的结果如下图 29.1 所示。



图 29.1

注意，同时支持 evenHeader（偶数页眉）和 evenFooter（偶数页脚）以及 firstHeader（首页页眉）和 firstFooter（首页页脚）。

添加打印标题

可以在每页上打印标题，以确保正确地标记数据。

```
from openpyxl.workbook import Workbook

wb = Workbook()
ws = wb.active
```



```
ws.print_title_cols = 'A:B' # 最左侧的两列
ws.print_title_rows = '1:1' # 顶端行

wb.save("printtitle.xlsx")
```

添加打印区域

可以选择工作表的一部分作为想要打印的唯一部分。

```
from openpyxl.workbook import Workbook

wb = Workbook()
ws = wb.active

ws.print_area = 'A1:F6'

wb.save("printarea.xlsx")
```





本章内容 2018 年 12 月 16 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(30\): 筛选和排序](#)

30. 筛选和排序

在 Excel 中，筛选和排序是一项很重要的功能，它们能够让我们方便地找到并分析数据，并且进一步利用它们更强大的功能，能够更好地处理数据。

可以使用 Python 代码在工作表中添加筛选。但是请注意，筛选和排序只能通过 openpyxl 进行配置，但需要在 Excel 等应用程序中具体操作应用，这是因为它们实际上重新排列或格式化单元格区域中的单元格或行。

要添加筛选，先定义单元格区域，然后添加列和排序条件：

```
from openpyxl import Workbook

wb = Workbook()
ws = wb.active

data = [
    ["水果", "数量"],
    ["奇异果", 3],
    ["葡萄", 15],
    ["苹果", 3],
    ["桃子", 3],
    ["石榴", 3],
    ["梨", 3],
    ["脐橙", 3],
    ["蓝莓", 3],
```



```

    ["芒果", 3],
    ["西瓜", 3],
    ["黑莓", 3],
    ["桔子", 3],
    ["木莓", 3],
    ["香蕉", 3]
]

for r in data:
    ws.append(r)

ws.auto_filter.ref = "A1:B15"
ws.auto_filter.add_filter_column(0, ["奇异果", "苹果", "芒果"])
ws.auto_filter.add_sort_condition("B2:B15")

wb.save("filtered.xlsx")

```

这将在文件中添加相关的命令,但是实际上既不会筛选也不会排序,如下图 30.1 所示。

	A	B	C
1	水果	数量	
2	奇异果	3	
3	葡萄	15	
4	苹果	3	
5	桃子	3	
6	石榴	3	
7	梨	3	
8	脐橙	3	
9	蓝莓	3	
10	芒果	3	
11	西瓜	3	
12	黑莓	3	
13	桔子	3	
14	木莓	3	
15	香蕉	3	
16			

图 30.1



本章内容 2018 年 12 月 23 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(31\): 数据有效性](#)

31.数据有效性

Excel 的“数据有效性”功能，在 Excel 2010 及以上版本中又称作“数据验证”，是一项非常方便实用且强大的功能，有兴趣深入了解的朋友可以参考[完美 Excel](#) 微信公众号中的系列文章《[Excel 技术 | 带你玩转数据有效性](#)》。

数据有效性可以应用于单元格区域，且单元格区域可以是非连续区域，例如单元格区域“A1,B2:B5”包含单元格 A1 和单元格区域 B2:B5。

使用 Python 在 Excel 工作表中添加数据有效性的基本代码如下：

```
from openpyxl import Workbook

from openpyxl.worksheet.datavalidation import
DataValidation

# 创建要操作的工作簿和工作表
wb = Workbook()
ws = wb.active

# 创建序列有效性的数据有效性对象
dv = DataValidation(type="list", formula1='"苹果, 香蕉, 脐橙"', allow_blank=True)

# 设置自定义错误消息, 可选
dv.error = '在列表中没有你输入的条目'
dv.errorTitle = '无效的输入'
```



```

# 设置自定义提示消息, 可选
dv.prompt = '请从列表中选择'
dv.promptTitle = '列表选取'

# 在工作表中添加数据有效性对象
ws.add_data_validation(dv)

# 在工作表单元格区域中添加数据有效性
dv.add('A1:A5')

# 保存工作簿
wb.save("dv.xlsx")

```

运行后的效果如下图 31.1 所示。

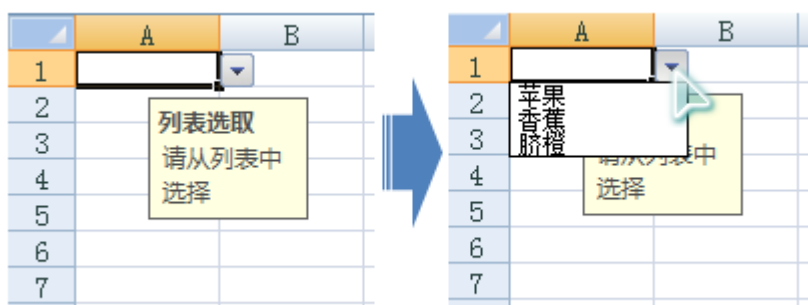


图 31.1

也可以选创建一些单元格，然后将其添加到数据有效性对象中，如下代码所示：

```

from openpyxl import Workbook

from openpyxl.worksheet.datavalidation import DataValidation

# 创建要操作的工作簿和工作表
wb = Workbook()
ws = wb.active

```



```

# 创建序列有效性的数据有效性对象
dv = DataValidation(type="list", formula1='"苹果, 香蕉, 脐橙"', allow_blank=True)

# 设置自定义错误消息, 可选
dv.error = '在列表中没有你输入的条目'
dv.errorTitle = '无效的输入'

# 设置自定义提示消息, 可选
dv.prompt = '请从列表中选择'
dv.promptTitle = '列表选取'

# 在工作表中添加数据有效性对象
ws.add_data_validation(dv)

# 在工作表单元格区域中添加数据有效性
cell1 = ws["B2"]
dv.add(cell1)

# 保存工作簿
wb.save("dv.xlsx")

```

运行后的效果如下图 31.2 所示。

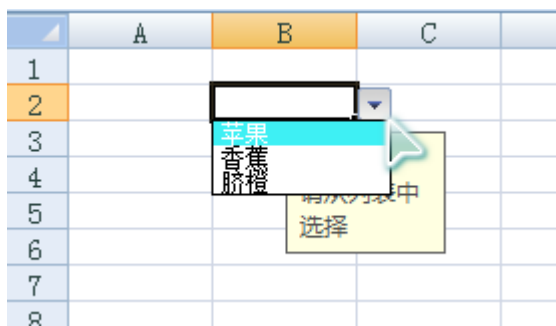


图 31.2

如果单元格中的值不在数据有效性序列值中，运行时会出现：



```
cell12 = ws["C3"]  
c3.value = "无效值"  
dv.add(cell12)
```

可用下列代码判断单元格中是否设置了数据有效性：

```
"B2" in dv
```

下面是一些设置数据有效性的代码示例：

1. 允许任何整数：

```
dv = DataValidation(type="whole")
```

2. 允许任何大于 100 的整数：

```
dv = DataValidation(type="whole", operator="greaterThan",  
formula=100)
```

3. 允许任何小数：

```
dv = DataValidation(type="decimal")
```

4. 允许任何 0 至 1 之间的小数：

```
dv = DataValidation(type="decimal", operator="between",  
formula1=0, formula2=1)
```

5. 允许日期：

```
dv = DataValidation(type="date")
```

6. 允许时间：

```
dv = DataValidation(type="time")
```

7. 允许最多 15 个字符的字符串：

```
dv = DataValidation(type="textLength",  
operator="lessThanOrEqual", formula1=15)
```

8. 单元格区域数据有效性：



```
from openpyxl.utils import quote_sheetname  
dv = DataValidation(type="list",  
formula1="{0}!$B$1:$B$10".format(quote_sheetname(sheetname)))
```

9. 自定义规则:

```
dv = DataValidation(type="custom", formula1="=公式")
```





本章内容 2018 年 12 月 30 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
Python 操作 Excel 学习笔记 (32): 表

32.表

表引用的是成组的单元格，提供了规范地确定数据结构的方法，Excel 提供了工具和智能来帮助管理表数据及其格式，这使得某些操作，诸如在表中给单元格应用样式变得更容易。

在 Excel 中，如果数据是结构化的，那么很容易转换成表。Excel 的“表工具”选项卡提供了很多操控表的工具命令，如下图 32.1 所示。



图 32.1

下面的 Python 代码在工作簿中创建表：

```
from openpyxl import Workbook
from openpyxl.worksheet.table import Table, TableStyleInfo

wb = Workbook()
ws = wb.active

data = [
    ['苹果', 10000, 5000, 8000, 6000],
    ['梨', 2000, 3000, 4000, 5000],
    ['香蕉', 6000, 6000, 6500, 6000],
    ['桔子', 500, 300, 200, 700],
]
```



```

# 添加列标题.必须是字符串

ws.append(["水果", "2014", "2015", "2016", "2017"])

for row in data:
    ws.append(row)

tab = Table(displayName="表 1", ref="A1:E5")

# 添加具有条带行和带状列的默认样式

style = TableStyleInfo(name="TableStyleMedium9",
showFirstColumn=False, showLastColumn=False,
showRowStripes=True, showColumnStripes=True)

tab.tableStyleInfo = style

ws.add_table(tab)

wb.save("table.xlsx")

```

运行上述代码后的结果如下图 32.2 所示。

	A	B	C	D	E
1	水果	2014	2015	2016	2017
2	苹果	10000	5000	8000	6000
3	梨	2000	3000	4000	5000
4	香蕉	6000	6000	6500	6000
5	桔子	500	300	200	700
6					
7					
8					

图 32.2

正如图 32.2 所示，在默认情况下，创建的表使用第一行作为标题行，所有列都带有筛选。

使用 TableStyleInfo 对象管理样式，这允许你对行或列进行条带化并应用不同的颜色方案。

说明：表名在工作簿中必须是唯一的，表头和筛选单元格区域必须始终包含字符串。如果不是这种情况，那么 Excel 可能会认为该文件无效并删除该表。



本章内容 2019 年 1 月 6 日首发于
[完美 Excel]微信公众号 [excelperfect](#)
原标题为
[Python 操作 Excel 学习笔记 \(33\): 保护
工作簿和工作表](#)

33.保护工作簿和工作表

openpyxl 支持保护工作簿和工作表不被修改，除非明确配置了其他算法，否则开放的 XML“传统密码散列算法”用于生成散列密码值。

注意，保护工作簿或工作表的密码仅提供非常基本的安全级别。数据没有加密，因此可以通过任意的免费工具进行修改。事实上，规范规定：“不应将工作表或工作簿元素保护与文件安全性混淆。它旨在使你的工作簿免于无意的修改，而无法保护它免受恶意修改。”

保护工作簿

为了阻止其他用户查看隐藏的工作表、添加、移动、删除或者隐藏工作表、重命名工作表，你可以使用密码保护工作簿结构。可以使用：

```
Openpyxl.workbook.protection.WorkbookProtection.workbookP  
assword()
```

属性来设置此密码。

```
wb.security.workbookPassword = '...'  
wb.security.lockStructure = True
```

同样，可以通过设置另一个密码来防止从共享工作簿中删除更改跟踪和更改历史记录。可以使用：

```
Openpyxl.workbook.protection.WorkbookProtection.revisions  
Password()
```

属性来设置此密码。



```
wb.security.revisionsPassword = '...'
```

`openpyxl.workbook.protection.WorkbookProtection` 对象的其他属性恰当地控制了特定的限制,但只有在设置了合适的密码后才会强制执行这些限制。

如果需要在不使用默认散列算法的情况下设置原始密码值,就使用特定的 setter 函数,例如:

```
Hashed_password =  
wb.security.set_workbook_password(hashed_password,already  
_hashed=True)
```

保护工作表

通过设置 `openpyxl.worksheet.protection.SheetProtection` 对象的属性来锁定工作表元素。与保护工作簿不同,可以使用或不使用密码启用工作表保护。`enable()` 和 `disable()` 方法用于启用或禁用工作表保护。`SheetProtection.sheet` 属性或调用 `enable()` 或 `disable()` 来启用工作表保护:

```
ws = wb.active  
wb.protection.sheet = True  
wb.protection.enable()  
wb.protection.disable()
```

如果没有指定密码,那么用户能够在不指定密码的情况下禁用已配置的工作表保护。否则,他们必须提供密码才能更改已配置的工作表保护。使用:

```
openpyxl.worksheet.protection.SheetProtection.password()
```

属性来设置该密码。

```
ws = wb.active  
ws.protection.password = '...'
```



关于完美 Excel

完美 Excel 是一个坚持分享 Excel 与 VBA 技术知识的微信公众号，自 2017 年 5 月 15 日开始，每天推送一篇 Excel 与 VBA 技术和应用方面的文章。目前，共有 670 余篇实用文章可供广大 Excel 爱好者和使用者学习交流。这本电子书就是根据完美 Excel 上发表的 Python 操作 Excel 学习笔记系列文章整理而成的。

每天清晨，完美 Excel 微信公众号：**excelperfect** 都会推送一篇关于 Excel 与 VBA 的相关文章。如果你有兴趣学习 Excel 和 VBA 的相关知识和实战技巧，可以关注完美 Excel 微信公众号，绝对不会让你失望！

可以通过下列方法关注[完美 Excel]微信公众号：

方法 1—在通讯录中搜索“完美 Excel”或者“**excelperfect**”后点击关注。

方法 2—扫一扫下面的二维码



完美 Excel 微信公众号使用指南

下图 1 是完美 Excel 微信公众号的界面。公众号名称显示在屏幕正上方，屏幕底部显示有“菜单栏”，目前设置的菜单为“技术精粹”、“VBA 精选”、“联系 me”。在底部左侧的小键盘图标为消息框入口，单击可进入消息框界面给完美 Excel 公众号发送消息。



图 1

下图 2、图 3、图 4 分别为底部 3 个菜单的子菜单。目前，菜单“技术精粹”中设置有“2018 年文章合集”、“480 篇文章合集”、“又一波 20 个函数”、“快速学会 30 个函数”、“玩转数据验证”等 5 个子菜单；菜单“VBA 精选”中设置有“Excel VBA 编程基础”、“最最基础入门篇”、“VBA 学习经验”等 3 个子菜单；菜单“联系 me”中设置有“投稿须知”、“网站集粹”、“个人声明”、“坚持的美好”、“2019 年目标”等 5 个子菜单。





图 2



图 3





图 4

单击这些子菜单会进入详细的文章页面或者文章整理的入口页面，方便读者浏览或查阅本公众号的文章。同时，这些子菜单会随着完美 Excel 微信公众号内容的增加而适时调整。

可以单击底部左侧的小键盘图标，进入发送消息界面，如图 5 所示。在文本框中输入想要发送的文字，单击底部的“发送”按钮，就可以将消息发送给完美 Excel 微信公众号。

大家应留意完美 Excel 微信公众号推送的文章中的一些信息，例如，我会在百度网盘中存放一些文档资料或者示例工作簿文件，并在文章中给出进入百度网盘下载的文本信息，你只需在发送消息框中输入我给出的文本，单击发送后，就会收到一条关于下载链接和密码的信息。单击链接并按提示输入密码后，即可获得



相关的文档资料或示例工作簿文件了。



图 5

例如，在图 5 所示的界面中输入“Excel 动画图 2”后，会自动收到图 6 所示的信息，根据信息即可获取这个 Excel 动画图表文件。



图 6



希望大家在完美 Excel 微信公众号中能够学习到所需要的知识，获取到所需要的 Excel 应用技巧，提高自己的水平。但愿在今后的日子里，完美 Excel 微信公众号能够真正帮助大家发挥 Excel 的威力，为大家解决问题，提高工作效率。

