Task 02 Python Excel 自动化之 OpenPyXL

2.0 包的安装

操作难度：⭐

打开 CMD/Terminal 进入到自己环境后，执行下面语句安装 `openpyxl` 模块。

```
pip3 install openpyxl
```

注：openpyxl可以读/写 .xlsx /.xlsm /.xltx /.xltm 的格式文件，但是不支持去读 /.xls 格式；读取 xls 格式，可以安装 **xlrd** 模块， `pip3 install xlrd` ，本章节以 /.xlsx 格式为主。

2.1 Excel读取

项目难度：⭐

- Excel 全称为 Microsoft Office Excel，2003年版本的是 xls 格式，2007和2007年之后的版本是 xlsx 格式。
- xlsx 格式通过 `openpyxl` 模块打开； xls 格式通过 `xlwt` 模块写， `xlrd` 模块读取。
- 本文以 xlsx 模式为例

2.1.1 读取Excel中的工作表

关于路径：

文件应在当前工作目录才可直接用相对路径引用，可导入 `os` ，使用函数 `os.getcwd()` 弄清楚当前工作目录是什么，可使用 `os.chdir()` 改变当前工作目录，具体可参考第一章节。（此处显现为相对路径）

```
# 获取当前工作目录
import os
print(os.getcwd())

import warnings
warnings.filterwarnings('ignore')
root_path = './OpenPyXL_test/'
```

1. 读取Excel文件 `用户行为偏好.xlsx` ，查看返回值属性

```
# 导入模块，查看属性
import openpyxl

wb = openpyxl.load_workbook(root_path+'用户行为偏好.xlsx')
type(wb)
```

```
openpyxl.workbook.workbook.Workbook
```

【代码解释】

这里我们使用 openpyxl 中的 load_workbook 函数来加载指定的 xlsx 文件，。

- openpyxl.load_workbook(
  filename,
  read_only=False,
  keep_vba=False,
  data_only=False,
  keep_links=True,
)

load_workbook 函数有五个参数，除 filename 外，其他参数都有默认值，各参数含义如下：

- `filename`：str 类型，表示要打开的文件的相对/绝对路径；
- `read_only`：bool 类型，是否以只读模式打开文件，默认值为 False，可读写；
- `keep_vba`：bool 类型，是否保留文件中的 vba 内容（即使保留了也不一定在代码中能使用），默认值为 False，不保留；
- `data_only`：bool 类型，如果单元格中是 excel 公式，是以公式计算后的值的形式显示还是以公式内容形式显示，默认值为 False，以公式内容形式展示；
- `keep_links`：bool 类型，是否保留单元格中的外链，默认值为 True，保留外链；
- 返回值类型：`openpyxl.workbook.Workbook`

如无特殊要求，我们只需要指定 `filename` 参数即可。

【小知识】

import * 和from...import...

`import *` 和 `from...import...` 的区别

- `import` 导入一个模块，相当于导入的是一个文件夹，相对路径。
- `from...import...` 导入了一个模块中的一个函数，相当于文件夹中的文件，绝对路径。

2. 查看对应工作簿包含的 sheet(工作表) 的名称，读取活动表

```
# 导入模块中的函数，查询对应表的名称
print(wb.sheetnames)
```

```
['订单时长分布', 'Sheet3']
```

【代码解释】

这里我们使用 `openpyxl.workbook.Workbook` 类对象的 `sheetnames` 属性来获取读取的工作簿中包含的 sheet(工作表) 的名称。

通过上述代码输出内容，我们可以知道 `用户行为偏好.xlsx` 中包含两个 sheet(工作表)，分别是：订单时长分布、 Sheet3。

```
# 读取工作簿的活动表
# 活动表是工作簿在 Excel 中打开时出现的工作表，在取得 Worksheet 对象后，可通过 title 属性取得它的名称。
active_sheet = wb.active
print(f'active_sheet对象: {active_sheet}')
print(f'active_sheet 名称: {active_sheet.title}')
```

```
active_sheet对象: <Worksheet "订单时长分布">
active_sheet 名称: 订单时长分布
```

【小知识】

活动表是可以修改的，在我们正常打开excel，完成修改后，保存excel，在关闭 excel 前显示的 sheet 就是活动表。

3. 查看指定sheet信息

```
# 通过传递表名字符串读取表、类型和名称、内容占据的大小
sheet = wb.get_sheet_by_name('Sheet3')
print(f'sheet: {sheet}')
print(f'type(sheet): {type(sheet)}')
print(f'sheet.title: {sheet.title}')
print(f'sheet.dimensions: {sheet.dimensions}')
```

```
sheet: <Worksheet "Sheet3">
type(sheet): <class 'openpyxl.worksheet.worksheet.Worksheet'>
sheet.title: Sheet3
sheet.dimensions: A1:I17
```

【代码解释】

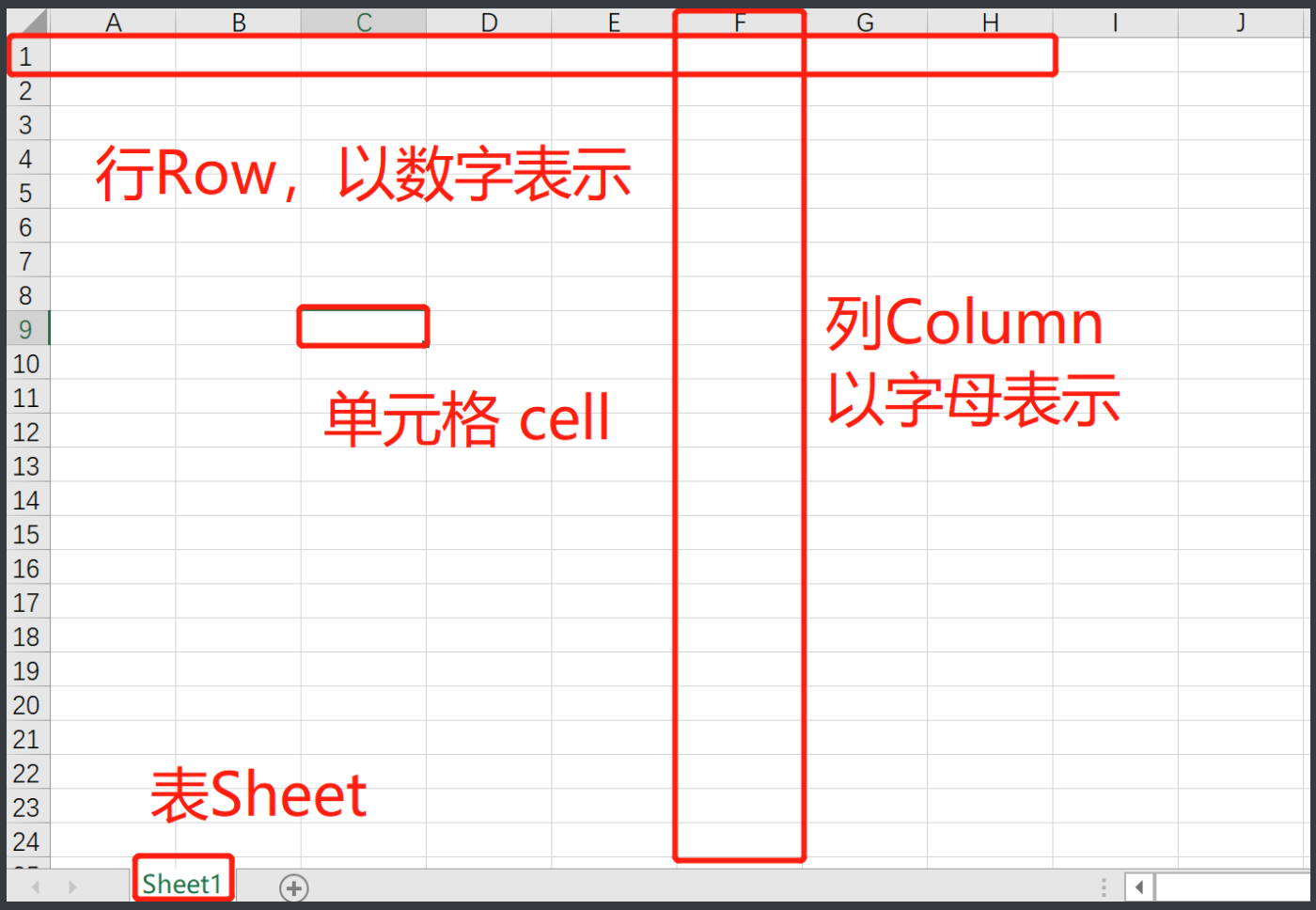
这里我们使用 `openpyxl.workbook.Workbook` 类对象的 `get_sheet_by_name` 方法，通过指定 sheetname 的方式来获取读取的工作簿中指定的 sheet(工作表) 对象。

并使用 `openpyxl.worksheet.worksheet.Worksheet` 类对象的一些属性来获取 sheet 的基本信息，比如 `Worksheet.title` 获取 sheet 名称， `Worksheet.dimensions` 获取 sheet 中值的范围。

Workbook.get_sheet_by_name(name) 函数只有一个参数，就是：sheetname(工作表名称)，功能是：通过 sheetname 获取到 Worksheet 对象，除了通过函数的方式获取到 Worksheet 对象，你还可以提过索引的方式，如：

```
wb['Sheet3']
```

2.1.2 读取工作表中的单元格



Cell(Excel单元格)

- Cell 对象有一个 value 属性，包含这个单元格中保存的值。
- Cell 对象也有 row 、column 和 coordinate 属性，提供该单元格的位置信息。
- Excel 用字母指定列，在Z列之后，列开始使用两个字母：AA、AB等，所以在调用的 cell() 方法时， 可传入整数作为 row 和 column 关键字参数，也可以得到一个单元格。

- 注：第一行或第一列的整数取1，而不是0.

```
# 从表中取得单元格 在 2.1.1 中我们已经读取过工作簿了 返回结果存储变量为 wb
## 获取表格名称
print(f'sheetnames: {wb.sheetnames}')
```

```
sheetnames: ['订单时长分布', 'Sheet3']
```

```
# 获取指定sheet
sheet = wb.get_sheet_by_name('订单时长分布')

# 通过单元格位置获取单元格对象，如：B1
a = sheet['B1']
print(f"sheet[B1']: {a}")

# 获取并打印 B1 单元格的文本内容
print(f"sheet[B1'].value: {a.value}")

# 获取并打印 B1 单元格所在行、列和数值
print(f'Row: {a.row}, Column: {a.column}')

# 获取并打印 B1 单元格坐标 和 值
print(f'Cell {a.coordinate} is {a.value}')
```

```
sheet[B1']: <Cell '订单时长分布'.B1>
sheet[B1'].value: 日期
Row: 1, Column: 2
Cell B1 is 日期
```

```
# 获取并打印出 B列 前8行的奇数行单元格的值
for i in range(1,8,2):
    print(i, sheet.cell(row=i,column=2).value)
```

```
1 日期
3 2020-07-24 00:00:00
5 2020-07-24 00:00:00
7 2020-07-24 00:00:00
```

```
# 确定表格的最大行数和最大列数，即表的大小
print(f'sheet.max_row: {sheet.max_row}')
print(f'sheet.max_column: {sheet.max_column}')
```

```
sheet.max_row: 14
sheet.max_column: 4
```

2.1.3 读取多个单元格的值

```
# 方法一：直接通过sheet索引，A1到C8区域的值
cells = sheet['A1:C8']
print(f'type(cells): {type(cells)} \n')

# 遍历元组 print每一个cell值
for rows in cells:
    for cell in rows:
        print(cell.value, end=" |")
    print("\n")
```

```
type(cells): <class 'tuple'>

编号 |日期 |行为时长 |
```

```
71401.30952380953 |2020-07-24 00:00:00 |a |

71401.30952380953 |2020-07-24 00:00:00 |b |

71401.30952380953 |2020-07-24 00:00:00 |c |

71401.30952380953 |2020-07-24 00:00:00 |d |

71401.30952380953 |2020-07-24 00:00:00 |e |

71401.30952380953 |2020-07-24 00:00:00 |f |

71401.30952380953 |2020-07-24 00:00:00 |g |
```

```python
# 方法二：sheet.iter_rows函数 按行获取数据
rows = sheet.iter_rows(min_row=1, max_row=8, min_col=1, max_col=3)
# 遍历元组 print每一个cell值
for row in rows:
    for cell in row:
        print(cell.value, end=" |")
    print("\n")
```

```
编号 |日期 |行为时长 |

71401.30952380953 |2020-07-24 00:00:00 |a |

71401.30952380953 |2020-07-24 00:00:00 |b |

71401.30952380953 |2020-07-24 00:00:00 |c |

71401.30952380953 |2020-07-24 00:00:00 |d |
```

```
71401.30952380953 |2020-07-24 00:00:00 |e |

71401.30952380953 |2020-07-24 00:00:00 |f |

71401.30952380953 |2020-07-24 00:00:00 |g |
```

```
# 方法三：sheet.iter_cols函数 按列获取数据
cols = sheet.iter_cols(min_row=1, max_row=4, min_col=1, max_col=3)
# 遍历元组 print每一个cell值
for col in cols:
    for cell in col:
        print(cell.value, end=" |")
    print("\n")
```

```
编号 |71401.30952380953 |71401.30952380953 |71401.30952380953 |

日期 |2020-07-24 00:00:00 |2020-07-24 00:00:00 |2020-07-24 00:00:00 |

行为时长 |a |b |c |
```

2.1.4 练习题

找出 用户行为偏好.xlsx 中 Sheet3 表中空着的格子，并输出这些格子的坐标

```
from openpyxl import load_workbook

exl = load_workbook(root_path+'用户行为偏好.xlsx')
sheet3 = exl.get_sheet_by_name('Sheet3')
```

```
sheet3.dimensions
```

```
'A1:I17'
```

```
# 直接通过sheet索引，sheet3.dimensions获取sheet数据区域
cells = sheet3[sheet3.dimensions]

# 遍历元组 判断每一个cell值是否为空
for rows in cells:
    for cell in rows:
        if not cell.value:
            print(f'{cell.coordinate} is None \n')
```

```
D3 is None

D8 is None

G10 is None
```

2.2 Excel写入

项目难度：⭐

2.2.1 写入数据并保存

1. 原有工作簿中修改数据并保存

```
# 1) 导入 openpyxl 中的 load_workbook 函数
from openpyxl import load_workbook

# 2) 获取指定 excel文件对象 Workbook
```

```
exl = load_workbook(filename=root_path+'用户行为偏好.xlsx')
# 3) 通过指定 sheetname 从 Workbook 中获取 sheet 对象 Worksheet
sheet = exl.get_sheet_by_name('Sheet3')
# 4) 通过索引方式获取指定 cell 值，并重新赋值
print(f"修改前 sheet['A1']: {sheet['A1'].value}")
sheet['A1'].value = 'hello world'
print(f"修改后 sheet['A1']: {sheet['A1'].value}")
# 5) 保存修改后的内容
# 如果 filename 和原文件同名，则是直接在原文件中修改；
# 否则会新建一个 excel 文件，并保存内容
exl.save(filename=root_path+'用户行为偏好_1.xlsx')  # 保存到一个新文件中 新文件名称为：用户行为偏好_1.xlsx
```

```
修改前 sheet['A1']: 1
修改后 sheet['A1']: hello world
```

```
# 验证保存修改内容是否成功
exl_1 = load_workbook(filename=root_path+'用户行为偏好_1.xlsx')
# 我们将原表中 Sheet3 中的 A1 值改为了 'hello world'
# 所以读取保存文件，查看对应值是否为 'hello world' 即可
a1 = exl_1['Sheet3']['A1'].value
if a1 == 'hello world':
    print(f"修改保存成功啦～，exl_1['Sheet3']['A1'].value = {a1}")
else:
    print(f"修改保存有问题，现在exl_1['Sheet3']['A1'].value = {a1}")
```

```
修改保存成功啦～，exl_1['Sheet3']['A1'].value = hello world
```

【代码解释】

从这里我们可以看到，我们只需要获取到 sheet 中的 cell 对象后，就可以通过改变 cell.value 的值来改变 对应单元格中的值，然后使用 Workbook 对象的 save 函数可以将修改后的工作簿内容保存起来。

2. 创建新的表格写入数据并保存

```
# 1) 导入 openpyxl 中的 Workbook 类
from openpyxl import Workbook

# 2) 初始化一个 Workbook 对象
wb = Workbook()
print(f'默认sheet: {wb.sheetnames}')

# 3) 通过 Workbook 对象的 create_sheet 函数创建一个 sheet
# title sheet 名称
# index sheet 位置，默认从0开始
sheet = wb.create_sheet(title='mysheet', index=0)
print(f'添加后sheet: {wb.sheetnames}')

# 4) 在新建的 sheet 中写入数据
# 比如 在 A1 单元格中写入 'this is test'
sheet['A1'].value = 'this is test'

print(f"sheet['A1'].value = {sheet['A1'].value}")

# 保存
wb.save(root_path+'creat_sheet_test.xlsx')
```

```
默认sheet: ['Sheet']
添加后sheet: ['mysheet', 'Sheet']
sheet['A1'].value = this is test
```

2.2.2 将公式写入单元格保存

```
# 1) 导入 openpyxl 中的 load_workbook 函数
from openpyxl import load_workbook

# 2) 获取指定 excel文件对象 Workbook
exl_1 = load_workbook(filename=root_path+'用户行为偏好_1.xlsx')
# 3) 通过指定 sheetname 从 Workbook 中获取 sheet 对象 Worksheet
sheet = exl_1['订单时长分布']

print(f'订单时长分布 值范围: {sheet.dimensions}')      #先查看原有表格的单元格范围，防止替代原有数据
```

```
订单时长分布 值范围: A1:D14
```

```
# 单元格 A15 中写入 合计
sheet['A15'].value = '合计'
```

```
# 单元格 D15 中写入求和公式: SUM(D2:D14)
sheet['D15'] = '=SUM(D2:D14)'
exl_1.save(filename='用户行为偏好_1.xlsx')
```

```
# 使用 xlwings 打开 excel 文件然后保存 使写入的 公式生效
import xlwings as xw
# 打开工作簿
app = xw.App(visible=False, add_book=False)
wb = app.books.open('用户行为偏好_1.xlsx')
wb.save()
# 关闭工作簿
wb.close()
app.quit()
```

```
# 验证写入是否成功
# 1) 获取指定 excel文件对象 Workbook,
#    并设置 data_only=True，表示读取的时候如果单元格内是公式的话，以公式计算后的值的形式显示
exl_2 = load_workbook(filename = '用户行为偏好_1.xlsx', data_only=True)
# 2) 打印相关信息
sheet = exl_2['订单时长分布']
print(f"sheet['A15']={sheet['A15'].value}，sheet['D15']={sheet['D15'].value}")
print(f"{sheet['D1'].value} 求和值为SUM(D2:D14)={sheet['D15'].value}")
```

```
sheet['A15']=合计，sheet['D15']=4004.7261561561563
次数 求和值为SUM(D2:D14)=4004.7261561561563
```

【注意】

即使设置了 data_only=True，也不能立即获取到刚刚添加的公式计算后的结果，需要自己 手动/添加代码 打开下 对应excel表格，然后 ctrl s保存下，再运行上面代码才能获取到对应公式计算后的值。

你可以使用下面代码自动打开指定 excel 文件然后保存使写入的公式生效，使用前你需要安装 xlwings，输入 `pip3 install xlwings` 即可，再后面我们也会学习这个模块。

```
# 使用 xlwings 打开 excel 文件然后保存 使写入的 公式生效
import xlwings as xw
# 打开工作簿
app = xw.App(visible=False, add_book=False)
wb = app.books.open('用户行为偏好_1.xlsx')
wb.save()
# 关闭工作簿
wb.close()
app.quit()
```

2.2.3 插入空列/行

```
# 获取指定 sheet
sheet = exl_1['Sheet3']

# 插入列数据 insert_cols(idx,amount=1)
# idx是插入位置，amount是插入列数，默认是1
# idx=2第2列，第2列前插入一列
sheet.insert_cols(idx=2)
# 第2列前插入5
# sheet.insert_cols(idx=2, amount=5)

# 插入行数据 insert_rows(idx,amount=1)
# idx是插入位置，amount是插入行数，默认是1
# 在第二行前插入一行
sheet.insert_rows(idx=2)
# 第2行前插入5行
# sheet.insert_rows(idx=2, amount=5)

exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.2.4 删除

```
# 删除多列
sheet.delete_cols(idx=5, amount=2)
# 删除多行
sheet.delete_rows(idx=2, amount=5)

exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.2.5 移动

当数字为正即向下或向右，为负即为向上或向左

```
# 移动
# 当数字为正即向下或向右，为负即为向上或向左
sheet.move_range('B3:E16',rows=1,cols=-1)
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.3 Excel 样式

项目难度：⭐⭐

2.3.1设置字体样式

1. 设置单个 cell(单元格) 字体样式

`Font(name字体名称,size大小,bold粗体,italic斜体,color颜色)`

```
# 1) 导入 openpyxl 中的 load_workbook 函数
#    导入 openpyxl 中的  styles 模块中的 Font 类
from openpyxl import load_workbook
from openpyxl.styles import Font

# 2) 获取指定 excel文件对象 Workbook
exl_1 = load_workbook(filename=root_path+'用户行为偏好_1.xlsx')
# 3) 通过指定 sheetname 从 Workbook 中获取 sheet 对象 Worksheet
sheet = exl_1['订单时长分布']
```

```
# 4) 获取到指定 cell 后，查看cell字体属性
cell = sheet['A1']
cell.font
```

```
<openpyxl.styles.fonts.Font object>
Parameters:
name='宋体', charset=134, family=3.0, b=True, i=False, strike=None,
outline=None, shadow=None, condense=None, color=
<openpyxl.styles.colors.Color object>
Parameters:
rgb=None, indexed=None, auto=None, theme=1, tint=0.0, type='theme',
extend=None, sz=11.0, u=None, vertAlign=None, scheme='minor'
```

```
# 5) 实例化一个 Font 对象，设置字体样式
#    字体改为：黑体  大小改为：20  设置为：加粗 斜体 红色
font = Font(name='黑体', size=20, bold=True, italic=True, color='FF0000')
cell.font = font
# 6) 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2. 设置多个 cell 的字体样式

```
# 上面我们已经获取到了 '用户行为偏好_1.xlsx' 中的 订单时长分布 工作表
# 我们处理了 单元格 A1 的字体样式，我们也可以通过遍历的形式，批量设置单元格字体样式

# 1) 获取要处理的单元格
#    通过 sheet 索引获取第二行 cell
#    获取列可以用 字母索引，如 sheet['A'] 获取第一列 cell
cells = sheet[2]
# 2) 实例化一个 Font 对象，设置字体样式
#    字体改为：黑体  大小改为：10  设置为：加粗 斜体 红色
font = Font(name='黑体', size=10, bold=True, italic=True, color='FF0000')
# 3) 遍历给每一个 cell 都设置上对应字体样式
for cell in cells:
    cell.font = font
# 4) 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.3.2 设置边框样式

1. 设置单元格边框样式

`Side`：边线样式设置类，边线颜色等

Side(style=None, color=None, border_style=None)

- style：边线的样式，有以下值可选：double, mediumDashDotDot, slantDashDot, dashDotDot, dotted, hair, mediumDashed, dashed, dashDot, thin, mediumDashDot, medium, thick
- color：边线颜色
- border_style：style 的别名，必须设置，一般直接设置 border_style 就行，不用设置 style

`Border`：边框定位类，左右上下边线

Border常用参数解释：

- top bottom left right diagonal：上下左右和对角线的边线样式，为 Side 对象
- diagonalDown：对角线从左上角向右下角方向，默认为 False
- diagonalUp：对角线从右上角向左下角方向，默认为 False

```
# 上面我们已经获取到了 '用户行为偏好_1.xlsx' 中的 订单时长分布 工作表 sheet
# 1) 导入 openpyxl 中的  styles 模块中的 Side, Border 类
from openpyxl.styles import Side, Border
# 2) 首先初始化一个边线对象（也可以设置多个）
side = Side(border_style='double', color='FF000000')
# 3) 通过 Border 去设置 整个单元格边框样式
border = Border(left=side, right=side, top=side, bottom=side,
diagonal=side, diagonalDown=True, diagonalUp=True)
```

```
# 4）查看目前单元格边框样式
# 获取第一行 cells
cells = sheet[1]
# 取出一个 cell 看边框样式
cells[0].border
```

```
<openpyxl.styles.borders.Border object>
Parameters:
outline=True, diagonalUp=False, diagonalDown=False, start=None,
end=None, left=<openpyxl.styles.borders.Side object>
Parameters:
style=None, color=None, right=<openpyxl.styles.borders.Side object>
Parameters:
style=None, color=None, top=<openpyxl.styles.borders.Side object>
Parameters:
style=None, color=None, bottom=<openpyxl.styles.borders.Side object>
Parameters:
style=None, color=None, diagonal=<openpyxl.styles.borders.Side object>
Parameters:
style=None, color=None, vertical=None, horizontal=None
```

```
# 5）修改边框样式，并保存修改
for cell in cells:
    cell.border = border
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.3.3 设置单元格其他样式

1. 设置单元格背景色

```
# 上面我们已经获取到了 '用户行为偏好_1.xlsx' 中的 订单时长分布 工作表 sheet
# 1) 从 openpyxl.styles 中导入 背景颜色设置类 PatternFill, GradientFill
from openpyxl.styles import PatternFill, GradientFill

# 2) 实例化 PatternFill 对象，fill_type 参数必须指定
pattern_fill = PatternFill(fill_type='solid',fgColor="DDDDDD")
# 3) 实例化 GradientFill 对象，填充类型 type 默认为 linear
gradient_fill = GradientFill(stop=('FFFFFF', '99ccff','000000'))
```

```
# 4) 获取指定 cells 遍历填充
# 对第三行 PatternFill 模式设置背景色
cells = sheet[3]
for cell in cells:
    cell.fill = pattern_fill

# 对第四行 GradientFill 模式设置背景色
cells = sheet[4]
for cell in cells:
    cell.fill = gradient_fill

# 5) 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.设置水平居中

openpyxl.styles 中的 Alignment 类常用参数介绍：

- horizontal：水平对齐，常见值 `distributed, justify, center, left, fill, centerContinuous, right, general`
- vertical：垂直对齐，常见值 `bottom, distributed, justify, center, top`
- textRotation：文字旋转角度，数值：0-180
- wrapText：是否自动换行，bool值，默认 False

```
# 上面我们已经获取到了 '用户行为偏好_1.xlsx' 中的 订单时长分布 工作表 sheet
# 1) 从 openpyxl.styles 中导入 对齐方式设置类 Alignment
from openpyxl.styles import Alignment

# 2) 实例化一个 Alignment 对象，设置水平、垂直居中
alignment = Alignment(horizontal='center', vertical='center')

# 3) 获取指定 cells 遍历填充
# 对第五行数据设置上面的对齐方式
cells = sheet[5]
for cell in cells:
    cell.alignment = alignment
# 4) 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

3. 设置行高与列宽

```
# 1) 设置行高，通过 row_dimensions 和 column_dimensions 来获取行和列对象
# 2) 设置第1行行高为 30
sheet.row_dimensions[1].height = 30
# 3) 设置第3列列款为 24
sheet.column_dimensions['C'].width = 24
# 4) 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.3.3 合并、取消合并单元格

```
# 注意：合并后的单元格只会显示合并区域中最右上角的单元格的值，会导致其他单元格内容丢失
# 上面我们已经获取到了 '用户行为偏好_1.xlsx' 对象 exl_1，我们可以通过 exl_1 来索引获取自己想要的 sheet
# 1）获取 Sheet3 这个工作表
sheet = exl_1['Sheet3']

# 合并指定区域单元格
sheet.merge_cells('A1:B2')

# sheet.merge_cells(start_row=1, start_column=3,
#                   end_row=2, end_column=4)

# 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

```
# 解除合并
sheet.unmerge_cells('A1:B2')

# sheet.unmerge_cells(start_row=1, start_column=3,
#                     end_row=2, end_column=4)

# 保存修改
exl_1.save(filename=root_path+'用户行为偏好_1.xlsx')
```

2.3.5 练习题

打开 test.xlsx 文件，找出文件中购买数量 `buy_mount` 超过5的单元格，并对其标红、加粗、加上红色边框。

```
# 1) 导入 openpyxl 相关函数和类
from openpyxl import load_workbook
from openpyxl.styles import Font, Side, Border

# 2) 读取 test.xlsx 文件，并筛选出 buy_mount 这一列
workbook = load_workbook(root_path+'test.xlsx')
sheet = workbook.active
buy_mount = sheet['B']
```

```
# 3) 设置边框 文字样式
side = Side(style='thin', color='FF0000')
border = Border(left=side, right=side, top=side, bottom=side)
font = Font(bold=True, color='FF0000')
```

```
# 4) 遍历判断 cell 值是否满足筛选条件
for cell in buy_mount:
    if isinstance(cell.value, float) and cell.value > 5:
        cell.font = font
        cell.border = border
# 5) 修改内容另存为 new_test.xlsx
workbook.save(root_path+'new_test.xlsx')
```

2.4 综合练习

2.4.1 将 业务联系表.xlsx 拆分成以下两个 excel：

- 客户信息表：客户名称 客户地址 客户方负责人 性别 联系电话 对接业务经理编号
- 业务经理信息表：业务经理编号 所在分区 所在区域 业务经理姓名

```python
# 1) 导入 openpyxl 相关函数和类
from openpyxl import load_workbook, Workbook

# 2) 读取原表数据
wb = load_workbook(root_path+'业务联系表.xlsx')
# 3) 获取工作表
sheet = wb.active
```

```python
# 草稿纸
# 我们知道我们表格的实际列名在第二行
# 获取每列第二行的坐标和值
for i in sheet[2]:
    print(i.coordinate, i.value)
```

```
A2 业务经理编号
B2 分区
C2 区域
D2 业务经理
E2 客户名称
F2 客户地址
G2 客户方负责人
H2 性别
I2 联系电话
J2 备注
```

```python
sheet.max_column, sheet.max_row
```

```
(10, 57)
```

```
# 4) 筛选出需要的列
# 4.1) 客户信息表：客户名称 客户地址 客户方负责人 性别 联系电话 备注 对接业务经理编号
cust_info = {'业务经理编号': 'A', '客户名称': 'B', '客户地址': 'C', '客户方负责人': 'D', '性别': 'E', '联系电话': 'F', '备注': 'G'}

# 4.2) 新建一个工作簿，并将默认sheet名称改成 客户信息
cust_info_excel = Workbook()
cust_info_sh = cust_info_excel.active
cust_info_sh.title = '客户信息'
```

```
# 4.3) 遍历筛选，如果是需要的表头，就将该列的值复制到新的工作簿中的 客户信息 工作表中
for i in sheet[2]:
    if i.value in cust_info:
        # 遍历将这一列中除了第一个cell外的所有cell值复制到新表
        for cell in sheet[i.coordinate[0]]:
            if cell.row == 1:
                continue
            cust_info_sh[f'{cust_info[i.value]}{cell.row-1}'].value = cell.value
```

```
# 5) 筛选出需要的列
# 5.1) 业务经理信息表：业务经理编号 所在分区 所在区域 业务经理姓名
manager_info = {'业务经理编号': 'A', '分区': 'B', '区域': 'C', '业务经理': 'D'}

# 5.2) 新建一个工作簿，并将默认sheet名称改成 客户信息
manager_info_excel = Workbook()
manager_info_sh = manager_info_excel.active
manager_info_sh.title = '业务经理信息'
```

```
    # 5.3) 遍历筛选，如果是需要的表头，就将该列的值复制到新的工作簿中的 业务经理信息 工作表中
    for i in sheet[2]:
        if i.value in manager_info:
            # 遍历将这一列中除了第一个cell外的所有cell值复制到新表
            for cell in sheet[i.coordinate[0]]:
                if cell.row == 1:
                    continue
                manager_info_sh[f'{manager_info[i.value]}{cell.row-1}'].value = cell.value
```

```
    # 6.1 ) 保存 客户信息表 工作簿内容
    cust_info_excel.save(root_path+'客户信息表_xl.xlsx')
    # 6.2) 保存 业务经理信息表 工作簿内容
    manager_info_excel.save(root_path+'业务经理信息表_xl.xlsx')
```

以上，虽然完成了数据拆分，但是对于进一步数据处理，继续使用 openpyxl 并不是很便捷，比如数据去重，筛选等，接下来我将给大家介绍如何使用 pandas 更便捷的处理 excel 数据。

```
    import pandas as pd

    # 1) 读取数据
    data = pd.read_excel(root_path+'业务联系表.xlsx', header=1)
```

```
    # 2) 数据筛选处理
    # 2.1) 客户信息表
    # 筛选出 客户信息表 需要的列
    cust_info_pd = data[['业务经理编号', '客户名称', '客户地址', '客户方负责人', '性别', '联系电话', '备注']]
    # 去除重复行
    cust_info_pd.drop_duplicates(inplace=True)
    # 打印出前三行
    cust_info_pd.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	业务经理编号	客户名称	客户地址	客户方负责人	性别	联系电话	备注
0	1	尹承望	*****-*****-****	孙康适	男	***-****-***	NaN
1	1	何茂材	*****-*****-****	孙康适	男	***-****-***	NaN
2	1	徐新霁	*****-*****-****	孙康适	男	***-****-***	NaN

```
# 2.2) 业务经理信息表
# 筛选出 业务经理信息表 需要的列，并打印出前三行
manager_info_pd = data[['业务经理编号', '分区', '区域', '业务经理']]
# 去除重复行
manager_info_pd.drop_duplicates(inplace=True)
# 打印出前三行
manager_info_pd.head(3)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	业务经理编号	分区	区域	业务经理
0	1	南区	贵州	占亮
5	2	南区	贵州	李朝华
11	3	北区	河北	王一磊

```
# 3) 数据保存
cust_info_pd.to_excel(root_path+'客户信息表_pd.xlsx', index=None)
manager_info_pd.to_excel(root_path+'业务经理信息表_pd.xlsx', index=None)
```

2.4.2 将 客户信息表.xlsx 和 客户关系表.xlsx 合并成一个excel

```
# 接上面的，将 客户信息表.xlsx 和 客户关系表.xlsx 合并成一个excel
# 这里我们依然用 pandas 来处理
business_contact = pd.merge(manager_info_pd, cust_info_pd, on='业务经理编
号')
# 查看合并后数据基本信息
business_contact.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55 entries, 0 to 54
Data columns (total 10 columns):
```

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   业务经理编号  55 non-null     int64
 1   分区      55 non-null     object
 2   区域      55 non-null     object
 3   业务经理    55 non-null     object
 4   客户名称    55 non-null     object
 5   客户地址    55 non-null     object
 6   客户方负责人  55 non-null     object
 7   性别      55 non-null     object
 8   联系电话    55 non-null     object
 9   备注      0 non-null      float64
dtypes: float64(1), int64(1), object(8)
memory usage: 4.7+ KB
```

```
# 查看前10条数据
business_contact.head(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	业务经理编号	分区	区域	业务经理	客户名称	客户地址	客户方负责人	性别	联系电话	备注
0	1	南区	贵州	占亮	尹承望	*****-*****_****	孙康适	男	***-****_***	NaN

1	1	南区	贵州	占亮	何茂材	*****-*****_****	孙康适	男	***-****_***	NaN
2	1	南区	贵州	占亮	徐新霁	*****-*****_****	孙康适	男	***-****_***	NaN
3	1	南区	贵州	占亮	郭承悦	*****-*****_****	邓翰翮	男	***-****_***	NaN
4	1	南区	贵州	占亮	梁浩思	*****-*****_****	邓翰翮	男	***-****_***	NaN
5	2	南区	贵州	李朝华	毛英朗	*****-*****_****	邓翰翮	男	***-****_***	NaN
6	2	南区	贵州	李朝华	侯俊美	*****-*****_****	任敏智	女	***-****_***	NaN
7	2	南区	贵州	李朝华	许高轩	*****-*****_****	任敏智	女	***-****_***	NaN
8	2	南区	贵州	李朝华	段英豪	*****-*****_****	任敏智	女	***-****_***	NaN
9	2	南区	贵州	李朝华	汤承福	*****-*****_****	任敏智	女	***-****_***	NaN

```
# 数据保存
manager_info_pd.to_excel(root_path+'业务联系表_pd.xlsx', index=None)
```

2.5 后记

- Python与Excel的自动化内容较多，此篇重在介绍基础，起到抛砖引玉的学习效果。
- 后面还给大家介绍了 pandas 处理excel的案例，比较简单，大家实际工作、学习中可以按自己需要使用不同框架

```
```