# Python笔记

Python2每个脚本必须要开头加上：#coding=utf-8

python下载地址：<https://www.python.org>

安装python后配置path环境变量：例 C:\Python27，安装目录

要使用第三包时需要安装pip，安装后可以直接这样使用

pip install selenium == 2.48.0#任何程序都这样用，在后面跟包名，如果不指定版本号，则默认安装最新版

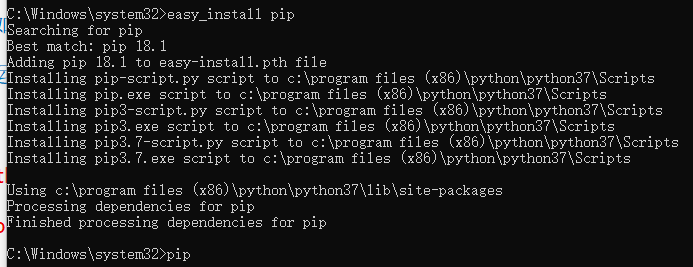
安装pip前需要先安装setuptools

setuptools下载地址：http://pypi.Python.org/pypi/ setuptools

pip下载地址：http://pypi.Python.org/pypi/ pip

pip安装成功验证：在cmd模式下输入pip

注意：如果输入pip报【pip.\_internal.cli.main】错误，直接管理员身份运行cmd，输入【easy\_install pip】问题解决，截图如下



注意python3中已经集成了pip包所有不需要再装了

* Pip安装模块包名

pip install 包名

* Pip升级包命令

python -m pip install --upgrade 包名

* python安装模块

进入模块目录执行python setup.py install

写python代码一般用pycham工具

1. pycham代码行显示设置：

file-settings-editor-General-Appearance-show line numbers

1. pycham代码字体大小设置：

file-settings-editor-colors&Fonts-Font-接下来的页面不能改字体大小，所以点击 Save As随便另存一个名字，就可以改了

1. PyCharm快捷键——搜索/替换快捷键

| **序号** | **快捷键** | **作用** |
| --- | --- | --- |
| **1** | **CTRL + F** | **当前文件下查找** |
| **2** | **F3** | **查找下一个** |
| **3** | **SHIFT + F3** | **查找上一个** |
| **4** | **CTRL + R** | **替换** |
| **5** | **CTRL + SHIFT + F** | **整个工程下查找** |
| **6** | **CTRL + SHIFT + R** | **指定路径下替换** |

快捷键设置：

File-settings-keymap 里面搜索想设置的快捷操作，比如查询整个工程下的函数或者变量，搜索find

1. PyCharm中修改新增文件为utf-8格式

file-settings- editor-file and code templates-python script

里写上#coding=utf-8

1. PyCharm中缩进快捷键
2. 选中多行，按tab进行缩进，按tab+shift去除缩进
3. Ctrl+r :全部替换
4. Ctrl+f：查找
5. 点击2下shift：查找py文件
6. Ctrl+shift+r：工程下查找字符串
7. Alt+左右方向键：跳转到上一个或下一个编辑文件
8. Ctrl+alt+方向左右键：快速切换到上一步的操作
9. pycharm切换python版本

设置-Project-project interpreter

## 输出：print

在python2中：print “hello”

在python3中：print (“hello”)

## 输出中的字符串连接

1. +（加号）：直接连接输出为一个字符串 print “hello” + “wold” 输出hellowold
2. ,（逗号）：分割字符串 print “hello” , “wold” 输出hello wold（为2个字符串）
3. \*（星号）：字符串显示多个 print “hello” \* 10 输出10个hello

## 引号的作用

1. “ ”,’ ’（双引号和单引号）：都代表一个字符串，在python中没有区别
2. ‘’’ 字符串 ‘’’（三引号）：一个长注释，脚本运行时会编译，所以要遵循python格式

可以用作输出，例：print ‘’’aaaa

Bbbb

Ccccc

‘’’

这样可以原格式输出

1. #（井号）：单行文本注释，不会编译

## 变量

直接进行赋值，不需要定义，例：

a = 10, b = ‘abc’

## 格式化字符串

把表达式右边的值赋值到字符串中格式化字符串的位置

%r：可以为任意类型的值

%s：代表字符串

%d：代表整数

整个表达式用 % 号连接，一条语句出现多个格式化字符串时，后面参数的方式为 （ ,）括号括起来用逗号隔开，例：

print "you are %d old,live %s,like %r" % (30,"chengdu","song")

## 转义字符\

在字符串中进行转义的作用

\\：能输出一个\

\:一般对单引号或者双引号用\",这样能输出"

\n:换行

\t:水平制表符，也是换行的意思，进行了一定的缩进

## 数据类型

* 1. int型：整数
  2. str型：字符串
  3. float：浮点型
  4. double：双精度型
  5. boolean：布尔型（返回true和false）

类型转换：int() 这个方式转换，转换的内容放在括号中

int，float，double可转str，不可反转

int，float，double三个可以互转

算术运算：必须要相同类型才能，不是同类型的就转为同类型

## 参数、解包、变量:argv

使用该功能时需要导入sys库,在代码开头写上：

from sys import argv

一旦使用了该功能，执行代码时就必须把参数传进去,例：

from sys import argv

script,a1, a2, a3 = argv

解释：script是脚本名字，必须这样写，a1等等是参数变量，把输入的参数分别传给a1，a2，a3，

执行该段脚本：python 脚本名.py lb1 lb2 lb3

lb1 lb2 lb3这3个参数分别传给了a1，a2，a3

## 接收用户输入

在python2中用：raw\_input(‘ ‘)

在python3中用：input(‘ ‘)

例：

age = raw\_input("输入一个整数") #括号内的参数可有可无，有且只能传一个字符串

注意：接收用户输入的值都会被看作为字符串，想要把数字转为整型这样用：

age = int(raw\_input("输入一个整数"))

## 文件操作

进行文件操作前必须先指定文件对象,即打开文件赋值给一个变量

open(‘文件名带后缀’,’ ’) open(‘test.txt’,’r’)

如果跟路径：open(**'d:/text.txt'**,**'r'**)

open()有2个参数，第一个文件名，第2个参数有w，r，a

r：不写第二个参数默认就是r，只是打开

w：以写入的状态打开，可以对文件内容进行编辑，要编辑文件内容时，必须先用 'w'打开后才行，该方式写内容会覆盖掉以前的内容

a：追加，也是可对文件进行编辑，在后面追加输入

1. **读写文件**

close – 关闭文件。跟你编辑器的 文件 -> 保存 .. 一个意思。

read – 读取文件内容。你可以把结果赋给一个变量。

readline – 读取文本文件中的一行。

truncate – 清空文件，请小心使用该命令。

write(“ ”) – 将引号里的内容写入文件

seek(0) – 将光标置于文件开头

例：

a1 = open(‘test.txt’,’r’) #获取文件对象

xx1 = a1.read()#把test文件的内容读取后存入变量xx1

xx2 = a1.readline() #读取文本的一行内容存入变量xx2

a1.close() #关闭文件

要执行文件写入操作，必须在打开文件时，加入写入权限，这里可以是w，a

a1 = open(‘test.txt’,’w’, encoding="utf-8")

a1.write(“写入一段话”) # 覆盖原有文件内容

注意：不加encoding="utf-8"的话，写入中文会是乱码

a1 = open(‘test.txt’,’a’)

a1.write(“写入一段话”) # 在文件内容后面追加“写入一段话”

a1.truncate() # 清空文件内容

## 函数

定义一个函数：def 变量名(): #注意后面要跟冒号

代码层级以4个字符区分，用tab键，可以传参数也可以无，如果传参数了，调用函数时必须传参

例:

def hs1(a,b):

print “a + b = %d” % (a + b)

return a + b #返回a+b的值，供其他函数调用并使用，也可以没有返回值

hs1(2,3)

执行后的结果：a + b =5

## 类

格式：class lei(object): #object可以没有

类中的函数必须加self，初始化函数相当于本脚本中的全局变量，可以供所有函数使用，初始化变量的写法：self.变量名 = 参数名

把函数放入类中，进行封装，方便其它脚本调用,如下一个脚本v1

**class lei1(object):**

**def \_\_init\_\_(self,a,b):**

**self.a = a**

**self.b = b**

def hs1(self):

a = self.a \* self.b

print **self.a + self.b**

**def hs2(self):**

**self.hs1()**

**a1 = lei1(2,3) #实例化lei1**

**a1.hs1() #调用类中函数 hs1，必须要进行了实例化类后才能调用类中的函数**

**知识点：类中的函数相互调用时，只需要加上在被调函数前加上self就行**

## 继承

从一个类中继承它的所有函数方法，继承时必须把继承的类写入本脚本类中，2个例子如下

脚本V1代码如下：

#coding=utf-8  
**class** lei1(object):  
 **def** \_\_init\_\_(self,a,b):  
 self.a = a  
 self.b = b  
 **def** hs1(self):  
 **print** self.a + self.b

脚本V2调用v1的脚本如下：

**from** v1 **import** \*  
**class** lei2(lei1):  
 **def** \_\_init\_\_(self,c,d):  
 self.c = c  
 self.d = d  
 **def** hs2(self):  
 **print '111'  
 print** self.c \* self.d  
 lei1(3,3).hs1()   
lei2(3,3).hs2()

例2：

**from** v1 **import** \*  
**class** lei2(lei1):  
 **def** hs2(self):  
 **print '111'**  
 lei1(4,3).hs1()   
lei2(3,3).hs2() # 由于继承了lei1，所以实例化本类必须传lei1的2个参数，执行hs2时，调用了lei1中的函数hs1(),所以这里也必须要传参数，lei2(3,3).hs2()传参只是运行hs2

lei2(3,3).hs1() # 可以直接调用lei1中的函数

## if elif else

a = 10

b = 8

if a > b:

print ‘yes’

elif a == b:

print ‘en’

else:

print “xiao”

## for循环

1. 循环遍历数组：数组a = [1,2,3,4,5]

for i in a:

print i

这段代码会输出

1

2

3

4

5

1. 循环次数：for i in range(5): 表示循环5次

可以这样写 for I in range(1,5):只会循环4次，最后一个数取值是小于

1. 跳出循环：break

## while循环

i = 0

while i <6:

print i

一般情况下不会使用while循环，while循环条件必须有假的时候才会跳出循环，

跳出循环 break

## 字符串截取

a = ‘1009988堂课’

print(a[0:3]) #输出1到3位：100

print (a[-3:]) #截取倒数第三位到结尾:988

直接用下标进行字符串分割

adb = [fwfeghtrt@163.com](mailto:fwfeghtrt@163.com)

取出@以前的方法：b = adb[0:adb.find(‘@’)] #用到了函数find

## split() 字符串分割

2个参数，第一个以什么来分割，第2个分割次数(该参数不写默认全部分割)，例：

a = ‘abc, def, ghi’

a.split(‘,’) 以逗号分割

该函数常用在读取txt文件时分割出字段

## 数组又叫列表list

类似字符串，列表一个序列的值。在字符串中，每个值是字符；在一个列表中可以是任何

数据类型。列表中的数值称为元素

数组用 [ ] 中括号括起来,计算列表元素个数用 len(数组)

a = [1,2,3,4]

b = [‘a’,’b’,’c’]

c = ['crunchy frog', 'ram bladder', 'lark vomit']

d = ['spam', 2.0, 5, [10, 20]] #在一个列表中的列表称为嵌套，嵌套数组作一个元素

取出数组中的某一个值用下标，下标从0开始

print a[0] 输出1

print b[2] 输出c

没有任何元素的列表称为空列表。你可以使用空的括号[] 创建一个空列表

正如你可能期望的，列表的值可以被赋值给变量：

>>> cheeses = ['Cheddar', 'Edam', 'Gouda']

>>> numbers = [17, 123]

>>> empty = []

>>> print cheeses, numbers, empty

['Cheddar', 'Edam', 'Gouda'] [17, 123] []

列表是可以改变的。当括号运算符出现在赋值语句的左边，它指向列表中

将被赋值的元素。

>>> numbers = [17, 123]

>>> numbers[1] = 5 #列表赋值

>>> print numbers

[17, 5]

numbers 中的第一个元素，原来是 123，现在是 5

### 列表操作

运算符 + 连接列表：

>>> a = [1, 2, 3]

>>> b = [4, 5, 6]

>>> c = a + b

>>> print c

[1, 2, 3, 4, 5, 6]

类似的，运算符 \* 给定次数地重复列表：

>>> [0] \* 4

[0, 0, 0, 0]

>>> [1, 2, 3] \* 3

[1, 2, 3, 1, 2, 3, 1, 2, 3]

第一个例子重复 [0]4 次。第二个例子重复列表 [1, 2, 3]3 次

### 列表切片

>>> t = ['a', 'b', 'c', 'd', 'e', 'f']

>>> t[1:3]

['b', 'c']

>>> t[:4]

['a', 'b', 'c', 'd']

>>> t[3:]

['d', 'e', 'f']

如果你忽略第一个下标，切片从列表头开始。如果你忽略第二个，切片到列表尾部结束。

因此如果你忽略两个，切片为整个列表的拷贝。

>>> t[:]

['a', 'b', 'c', 'd', 'e', 'f']

由于列表是可改变的，有必要在折叠、旋转或切断操作前复制列表。

赋值语句左边的切片运算符可以更新多个元素：

>>> t = ['a', 'b', 'c', 'd', 'e', 'f']

>>> t[1:3] = ['x', 'y']

>>> print t

['a', 'x', 'y', 'd', 'e', 'f']

### 列表方法

Python 提供了一个列表的方法。例如，append 方法将新的元素添加到列表尾部：

>>> t = ['a', 'b', 'c']

>>> t.append('d')

>>> print t

['a', 'b', 'c', 'd']

extend 方法读取一个列表作为参数，并附加其中所有的元素：

>>> t1 = ['a', 'b', 'c']

>>> t2 = ['d', 'e']

>>> t1.extend(t2)

>>> print t1

['a', 'b', 'c', 'd', 'e']

这个例子中 t2 没有改变。

sort 方法从小到大对列表中的元素进行排序：

>>> t = ['d', 'c', 'e', 'b', 'a']

>>> t.sort()

>>> print t

['a', 'b', 'c', 'd', 'e']

列表的方法都是空的，它们对列

### 移除列表元素

a = [1,2,3,4,5,6,7,8,9]  
a.remove(a[1]) #去掉数组下标为1的元素  
print(a)  
a = [1,2,3,4,5,6,7,8,9]  
a.remove(1) #去掉数组第一个元素  
print(a)  
a = [1,2,3,4,5,6,7,8,9]  
a.pop() #去掉数组最后一个  
print(a)

输入结果：

[1, 3, 4, 5, 6, 7, 8, 9]

[2, 3, 4, 5, 6, 7, 8, 9]

[1, 2, 3, 4, 5, 6, 7, 8]

### 获取列表最后一个元素

a = [1,2,3,4,5,6,7,8,9]

print(a[-1])

输出：9

## 字典

可以把字典当作是索引集合（关键字）和值集合之间的映射。每一个关键字对应一个值。

关键字和对应的值称为键 -值对，或者项。

字典用{ }大括号括起来，计算字典元素个数用 len(字典)

a = {‘name’:’lb’,’sex’:’man’,’old’:32}

冒号前面的值看着为 “键或者关键字”

a[“name”] 输出lb #只能以关键字作为字典的引用

1. 字典赋值：例

>>> a={'user':'lb'}

>>> print a

{'user': 'lb'}

>>> a['sex']='man' #赋值语句，直接在后面追加

>>> print a

{'user': 'lb', 'sex': 'man'}

print a[‘user’] 输出lb

1. in 运算符对字典也同样使用。它显示某个键是否在字典中作为关键字

>>> 'name' in a

True

>>> 'lb' in a

False

1. 如果想查看某个值是否在字典中，可以使用方法 values, 返回包含关键字值的列表，然后

使用 in 运算符：

>>> vals =a.values()

>>> 'lb' in vals

True

1. Python 字典(Dictionary) items() 函数以列表返回可遍历的(键, 值) 元组数组

items()方法语法： 字典.items()

例：

dict={'Google':'www.google.com','Runoob':'www.runoob.com', 'taobao': 'www.taobao.com'}

print "字典值 : %s" % dict.items() # 遍历字典列表

for key,values in dict.items(): #定义key，values来接收字典的返回值（对应键：值）

print key,values

以上实例输出结果为：

字典值 : [('Google', 'www.google.com'), ('taobao', 'www.taobao.com'), ('Runoob', 'www.runoob.com')]

Google www.google.com

taobao [www.taobao.com](http://www.taobao.com)；

Runoob www.runoob.com

## 元组

元组是一组序列的值。元组中的值可以是任何数据类型，使用整数作为下标，在这个方面

元组很想列表。但是一个主要的区别是元组是不可改变的。

从语法构成上来看，元组是用逗号隔开的值的序列：

b = ('a','b','c')

#只含一个元素的元组有2个方法

b = 'set', #不要忘记最后的逗号

b = tuple(‘set’)

创建一个空数组：

在python2中： b = ()

在python3中： b = tuple()

如果参数是一个序列（字符串、列表或元组） ，返回结果是使用序列中的元素构成的元组：

>>> t = tuple('lupins')

>>> print t

('l', 'u', 'p', 'i', 'n', 's')

### 取元组中的值

b = ('a','b','c') # 创建一个元组b

print b[0] 输出a，所以和数组取值是一样的，切片也一样

## 断言

直接用**assert**

后面就直接输入表达式，例判断内容在其字符串中

**assert ‘a’ in** ‘abc’

## 去掉字符串中的空格

1. 利用strip()方法

a = **" hello word "  
print** a.lstrip() *#去掉字符串中左边的空格***print** a.rstrip() *#去掉字符串中右边的空格***print** a.strip() *#去掉字符串中两边的空格*

1. 利用join()方法

该方法拼接字符串、列表、字典中的元素

语法格式：’seq’.join(sep)

seq：以什么方式来分割

sep：可以是字符串、列表、字典

去掉字符串中所有空格的方法是先将字符串用splip()方法分割成数组

**print ''**.join(a.split()) *#去掉字符串中所有的空格*

## mysql数据库连接

先安装pymysql，安装命令： pip install PyMysql

import pymysql  
  
# 打开数据库连接  
connect = pymysql.Connect(  
 host='gz-cdbxxx.sql.xxx.com', #连接地址  
 port=62276, #端口  
 user='xxx', #用户名  
 passwd='xxx', #密码  
 db=xxx, #数据库名  
 charset='utf8' #连接编码  
)  
# 使用 cursor() 方法创建一个游标对象 cursor  
cursor = connect.cursor()  
  
# 使用 execute() 方法执行 SQL 查询语句  
sql = 'select username from sys\_user where cid = "%s"'  
data = ('306b249d62114ddab7aa3112f3af1440')  
cursor.execute(sql % data)  
  
#sql删除  
sql = 'delete from dev\_group where did like "%sbdid%";delete from item\_group where item\_id like "%项目id%";delete from sys\_user where cid ="821b4afdc95548c1952b02cba59a1ef6";' #多条sql可用分号拼接  
cursor.execute(sql)  
connect.commit()  
# cursor.fetchall()获得返回所有行的数据  
results = cursor.fetchall()  
print(results) #输出的结果：(('xxx',),('xxx',))元祖格式  
print(results[0][0]) #输出字符串  
  
#connect对象的支持  
connect.commit() #提交当前事务  
connect.rollback() #回滚当前事务  
connect.close() # 关闭数据库连接  
connect.cursor() #创建游标  
  
#cursor对象的支持  
cursor.execute() # 执行一个数据库的查询命令  
cursor.fetchone() # 取得结果集的下一行  
cursor.fetchmany() # 获取结果集的下几行  
cursor.fetchall() # 获取结果集中的所有行  
cursor.rowcount # 返回数据条数或影响行数  
cursor.close() # 关闭游标对象

## 配置文件运用

建一个ini的文件，直接右键选择新建文件，文件名写XX.ini

**[mysql]**host = XXX.com  
port=XXX  
user=test  
passwd=XXX  
db=XXX  
  
**[api\_service\_url]**url = http://XXX.com

配置文件读取出来是元祖数据，配置文件注释用 【;】分号

读取配置文件的方法：

import configparser #导入配置文件模块  
conf = configparser.ConfigParser() #创建管理对象  
curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
rootPath = os.path.split(curPath)[0] + '\\' + 'config\configure.ini' #打开配置文件  
conf.read(rootPath,encoding='utf-8') #读取ini配置文件

1. Conf.sections() 读取所有的sections,返回list
2. Conf.items(‘mysql’) 读取mysql里面的内容，返回list，list里面的对象是元祖
3. Conf.get(‘mysql’,’host’) 获取host的值

## 字符串大小写转换

s=**'What is Your Name?'**s2=s.lower()  
**print**(s2) *#返回小写字符串  
 # what iss your name?***print**(s.upper()) *#返回大写字符串  
 # WHAT IS YOUR NAME?***print**(s.capitalize()) *#字符串首字符大写  
 # What is your name?***print**(s.title()) *#每个单词的首字母大写  
# What Is Your Name?***print**(s.swapcase()) *#大小写互换  
# wHAT IS yOUR nAME?*

## 获取目录下的文件名和后缀

导入包：import os.path

import os.path  
pathDir = os.listdir('F:\makeba') #获取目录下的文件名字含后缀  
print(pathDir) #获取的结果是个数组

## 删除本地文件

方法一：

import os  
path = 'F:\makeba'  
files = os.listdir(path)#遍历路径下的所有文件  
for f in files:  
 print(f)  
 # 找到老的文件所在位置  
 old\_file = os.path.join(path, f)  
 if f == 4.jpg:

os.remove(old\_file)

方法二：

if os.path.exists('F:\makeba\4.jpg'): # 判断文件是否存在  
 print('文件存在')

os.remove('F:\makeba\4.jpg')  
else:  
 print('文件不存在')

## 重命名本地文件

import os  
path = 'F:\makeba'  
files = os.listdir(path)#遍历路径下的所有文件  
i = 1  
for f in files:  
 print(f)  
 # 找到老的文件所在位置  
 old\_file = os.path.join(path, f)  
 #输出老的文件所在的位置  
 print("old\_file is {}".format(old\_file))  
 # 指定新文件的位置，如果没有使用这个方法，则新文件名生成在本项目的目录中  
 new\_file = os.path.join(path, 'test' + str(i)+'.jpg')  
 #重命名  
 os.rename(old\_file,new\_file)  
 i = i+1

## 将两个列表合并成一个字典 dict(zip())方法

keys = ['a', 'b', 'c']  
values = [1, 2, 3]  
dictionary = dict(zip(keys, values))  
print(dictionary)

输出结果：{'a': 1, 'b': 2, 'c': 3}

## Python爬虫-获取图片url

import re  
import urllib.request  
  
# ------ 获取网页源代码的方法 ---  
def getHtml(url):  
 page = urllib.request.urlopen(url)  
 htmlSourceCode = page.read()  
 return htmlSourceCode  
  
# ------ getHtml()内输入任意帖子的URL ------  
html = getHtml("https://tieba.baidu.com/p/5352556650")  
# ------ 修改html对象内的字符编码为UTF-8 ------  
html = html.decode('UTF-8')  
# ------ 获取帖子内所有图片地址的方法 ------  
def getImg(html):  
 # ------ 利用正则表达式匹配网页内容找到图片地址 ------  
 # reg = r'src="([.\*\S]\*\.jpg)"'  
 reg = r'<img class="BDE\_Image" src="(.\*?)"'  
 imgre = re.compile(reg)  
 imglist = re.findall(imgre, html)  
 return imglist  
  
imgList = getImg(html)  
print(imgList)  
  
for i in imgList:  
 print(i)

## 利用url-图片下载

import csv  
#获取所有url  
book = csv.reader(open('D:/imges.csv','r'))  
#定义空数组，存储url  
listmage=[]  
listUsername=[]  
for i in book:  
 # print(i)  
 # print(i[0])  
 if '\u4e00' <= i[0] <= '\u9fa5': #判断是否为中文  
 # print(i[0])  
 listUsername.append(i[0])  
 listmage.append(i[1])  
print(listmage)  
print(listUsername)  
print(len(listUsername))  
print(len(listmage))  
#用urllib里的urlretrieve方法下载图片  
from urllib.request import urlretrieve  
import os  
num=0  
for url in listmage:  
 print(num)  
 try:  
 if os.path.exists('D:\downmage/'+listUsername[num] +'.jpg'): # 判断文件是否存在  
 print('名字重复')  
 os.rename('D:\downmage/'+listUsername[num] +'.jpg','D:\downmage/'+listUsername[num] + str(num) +'.jpg') #重命名  
  
 urlretrieve(url,'D:\downmage/'+ listUsername[num] +'.jpg')#下载图片，url、存储路径+文件名  
 except:  
 print('第'+str(num)+'次发生错误'+listUsername[num])  
 continue  
 num = num+1

## 将图片转换为base64

import base64  
url='D:\lb\mage2\焦珂.jpg'  
img=open(url,'rb') #打开图片  
base64\_data = base64.b64encode(img.read()) #读出图片并转换为base64  
s = base64\_data.decode() #转换为decode  
print('data:image/jpeg;base64,%s' % s)

## 将base64还原为图片

import base64

base64str = "/9jxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
if not base64str.startswith("/9j"):  
 print("base64字符串必须以/9j开头")  
 return  
imageData = base64.b64decode(base64str)  
with open("temp.png", "wb") as picf:  
 picf.write(imageData)

## 判断字符串是否为字母、数字、中文

严格解析：有除了数字或者字母外的符号（空格，分号,etc.）都会False  
isalnum()必须是数字和字母的混合  
isalpha()不区分大小写

[复制代码](javascript:void(0);)

str\_1 = "123"

str\_2 = "Abc"

str\_3 = "123Abc"

#用isdigit函数判断是否数字

print(str\_1.isdigit())

Ture

print(str\_2.isdigit())

False

print(str\_3.isdigit())

False

#用isalpha判断是否字母

print(str\_1.isalpha())

False

print(str\_2.isalpha())

Ture

print(str\_3.isalpha())

False

#isalnum判断是否数字和字母的组合

print(str\_1.isalnum())

Ture

print(str\_2.isalnum())

Ture

print(str\_1.isalnum())

Ture  
注意：如果字符串中含有除了字母或者数字之外的字符，比如空格，也会返回False

#判断是否为中文

unicode编码的范围来判断中文，中文的范围是4e00~9fa5

print ('\u4e00' <= ‘中国’ <= '\u9fa5'): #判断是否为中文

输出：true

## 修改图片分辨率

from PIL import Image  
import os.path  
pathDir = os.listdir('D:\downmage') #获取目录下的文件名字含后缀  
print(pathDir) #获取的结果是个数组  
print(len(pathDir)) #获取的结果是个数组  
j=1  
a = 1  
for i in pathDir:  
 path = 'D:\downmage/'+i  
 img = Image.open(path)#打开图片  
 w,h = img.size #获取图片的像素大小  
 print(i,w,h)  
 if w > 1920 and h > 1080:  
 print('像素宽度超过1920,高度超过1080---%s' % i)  
 mage = img.resize((1920,1080), Image.ANTIALIAS)#设置分辨率大小  
 a = 2  
 elif h > 1080:  
 print('像素高度超过1080---%s' % i)  
 mage = img.resize((w,1080), Image.ANTIALIAS)  
 a = 2  
 elif w > 1920:  
 print('像素宽度超过1920---%s' % i)  
 mage = img.resize((1920, h), Image.ANTIALIAS)  
 a = 2  
 if a == 2:  
 os.remove('D:\downmage/'+i) #删除原文件  
 mage.save('D:\downmage/'+i) #保存新文件  
 a = 1  
 j=j+1

## 移动文件

import os.path  
import shutil  
filePath = "D:\downmage/"  
pathdir = os.listdir(filePath)  
image\_path = "D:/lbb/" #定义保存文件的路径  
for fileName in pathdir:  
 # shutil.copy(filePath+fileName,image\_path) #复制文件到指定路径  
 shutil.move(filePath+fileName,image\_path) #移动文件到指定路径

## 连接mqtt、订阅、发布

import paho.mqtt.client as mqtt  
import json  
import random  
HOST = "IP地址"  
PORT = 1883  
client\_id = f'python-mqtt-{random.randint(100000000, 999999999)}' #客户端的唯一标识，全局唯一  
client = mqtt.Client(client\_id) #定义client，可不用指定client\_id，系统自动分配  
  
def on\_connect(client, userdata, flags, rc):  
 *'''连接成功后回调，0表示成功'''* if rc != 0:  
 # print("Connected with result code "+str(rc))  
 # else:  
 print("Unexpected disconnection %s" % rc)  
 client.subscribe("abcde",qos=1) # 订阅消息，abcde为消息主题，必须写在这里  
  
def on\_message(client, userdata, msg):  
 *'''订阅消息，接收到消息后回调'''* print(client\_id+" 主题:"+msg.topic+" 消息:"+str(msg.payload.decode('utf-8')))  
 print('\t')  
  
def on\_subscribe(client, userdata, mid, granted\_qos):  
 *'''订阅消息成功后回调'''* print("On Subscribed: qos = %d" % granted\_qos)  
  
data = {  
 "cmd":"set\_client\_id",  
 "apikey":"Publish",  
 "messageId":"10010"  
}  
  
param = json.dumps(data)  
client.username\_pw\_set(username="XXX", password="XXX")#定义连接mqtt服务器的用户名和密码  
client.on\_connect = on\_connect #回调连接  
client.on\_message = on\_message #回调订阅消息  
client.on\_subscribe = on\_subscribe  
print(client.\_client\_id) #打印出clintid  
client.connect(HOST, PORT, 60) #连接mqtt  
client.publish("face/0107004758/request", payload=param, qos=0) # 发送消息  
client.loop\_forever() #保持连接，阻塞程序，实时监听订阅消息

## 写接口

import json  
from wsgiref.simple\_server import make\_server  
import urllib.parse  
import re  
# 定义函数，参数是函数的两个参数，都是python本身定义的，默认就行了。  
def application(environ, start\_response):  
 # 定义文件请求的类型和当前请求成功的code  
 start\_response('200 OK', [('Content-Type', 'application/json')])  
 # environ是当前请求的所有数据，包括Header和URL，body  
 request\_body = environ["wsgi.input"].read(int(environ.get("CONTENT\_LENGTH", 0))) #设备连接时用post，接收数据改为get  
 json\_str = request\_body.decode(encoding='utf-8')  
 # json\_str = request\_body.decode(encoding='gbk').encode(encoding='utf-8') # byte 转 str  
 json\_dict = json.loads(json\_str) # （注意：key值必须双引号）  
 json\_data = json.dumps(json\_dict,ensure\_ascii=False)  
 print(json\_data)  
 return json\_data  
if \_\_name\_\_ == "\_\_main\_\_":  
 port = 6008  
 httpd = make\_server("0.0.0.0", port, application)  
 print("serving http on port {0}...".format(str(port)))  
 httpd.serve\_forever()

## 获取时间戳

import time  
t = time.time()  
print(t) # 原始时间数据  
print(int(t)) # 秒级时间戳  
print(int(round(t \* 1000))) # 毫秒级时间戳  
print(int(round(t \* 1000000))) # 微秒级时间戳

## 字符串转md5

import hashlib

signature\_str = '334445xk91h123456781637144702250e71528f9445454b7e18ab10cc'

m2 = hashlib.md5()  
a = signature\_str.encode("UTF-8") #进行编码，不然报错  
m2.update(a)  
signature = m2.hexdigest()  
print(signature)

## 字符串编码

text = ‘kioreorieotirurouort’

text.encode('utf-8') 转utf-8

## AES加解密

python 在 **Windows**下使用AES时要安装的是pycryptodome 模块

 pip install pycryptodomex

python 在 **Linux**下使用AES时要安装的是pycrypto模块   pip install pycrypto

例：

* ECB模式、16位密钥、utf-8字符集、base64和hex编码示列如下
* CBC模式、16位密钥、iv16位偏移量、base64和hex编码示列如下

from Cryptodome.Cipher import AES  
from binascii import b2a\_hex, a2b\_hex  
import base64  
class ECBandCBC\_AES\_jiami\_AS\_jiemi():  
 *'''ECB和CBC模式hex和base64加解密'''* def add\_to\_16(self,text):  
 *'''长度补全，必须要使用该方法,密钥和加密的文本内容都必须是固定的16位倍数'''* if len(text.encode('utf-8')) % 16: #%求模运算，相当于mod，也就是计算除法的余数，比如5%2就得到1，不是16的倍数  
 # print(len(text.encode('utf-8')))  
 add = 16 - (len(text.encode('utf-8')) % 16)  
 # print(add)  
 else:  
 #求模运算等于0的时候  
 add = 0  
 text = text + ('\0' \* add) #位数必须要补全  
 # return text  
 return text.encode('utf-8')  
 def ecb\_hex\_jiami(self,text,keys):  
 *'''EXB HEX加密函数'''* key = self.add\_to\_16(keys)  
 # key = 'qdwphltib5kyfd3n'.encode('utf-8')  
 mode = AES.MODE\_ECB  
 text = self.add\_to\_16(text)  
 cryptos = AES.new(key, mode)  
 cipher\_text = cryptos.encrypt(text) #加密，text必须为字节bytes  
 print(bytes.decode(b2a\_hex(cipher\_text)))  
 return bytes.decode(b2a\_hex(cipher\_text)) #HEX 16进制输出  
 def ecb\_hex\_jiemi(self,text,keys):  
 *'''ECB HEX解密函数'''* key = self.add\_to\_16(keys)  
 # key = 'qdwphltib5kyfd3n'.encode('utf-8') #进行utf-8编码，转为bytes  
 text = self.add\_to\_16(text)  
 mode = AES.MODE\_ECB #定义ECB模式  
 cryptor = AES.new(key, mode) #创建实例  
 try:  
 plain\_text = cryptor.decrypt(a2b\_hex(text))#解密  
 print(bytes.decode(plain\_text).rstrip('\0'))  
 return bytes.decode(plain\_text).rstrip('\0') #转换为python的字符串  
 except:  
 return '请检查解密数据或者密钥是否正确！'  
 def ecb\_base64\_jiami(self, text,keys):  
 *'''ECB base64加密函数'''* key = self.add\_to\_16(keys)  
 mode = AES.MODE\_ECB  
 cryptos = AES.new(key, mode)  
 text = self.add\_to\_16(text)  
 t = cryptos.encrypt(text) #加密明文，text必须为字节bytes  
 t = base64.encodebytes(t) #将返回的字节数据进行base64编码  
 t = t.decode('utf-8') #将字节型数据转换成python中的字符串类型  
 try:  
 t = t.split()[0] + t.split()[1] #去除换行符  
 print(t)  
 return t  
 except:  
 return t  
  
 def ecb\_base64\_jiemi(self, text,keys):  
 *'''ECB base64解密函数'''* key = self.add\_to\_16(keys)  
 # key = 'qdwphltib5kyfd3n'.encode('utf-8')  
 text = base64.b64decode(text) # 转换为base64格式，为bytes  
 mode = AES.MODE\_ECB  
 cryptor = AES.new(key, mode)  
 try:  
 plain\_text = cryptor.decrypt(text) #解密  
 print(bytes.decode(plain\_text).rstrip('\r'))  
 return bytes.decode(plain\_text).rstrip('\r')  
 except:  
 return '请检查解密数据或者密钥是否正确！'  
 def cbc\_hex\_jiami(self,text,keys,pyls):  
 *'''CBC加密：HEX加密函数'''* key = self.add\_to\_16(keys)  
 iv = self.add\_to\_16(pyls) #定义偏移量  
 mode = AES.MODE\_CBC #设置CBC模式  
 text = self.add\_to\_16(text)  
 cryptos = AES.new(key, mode,iv) #创建实例  
 cipher\_text = cryptos.encrypt(text) #加密，text必须为字节bytes  
 print(bytes.decode(b2a\_hex(cipher\_text)))  
 return bytes.decode(b2a\_hex(cipher\_text)) #HEX 16进制输出  
  
 def cbc\_hex\_jiemi(self,text,keys,plys):  
 *'''CBC解密：HEX解密函数'''* key = self.add\_to\_16(keys)  
 iv = self.add\_to\_16(plys)  
 text = self.add\_to\_16(text)  
 mode = AES.MODE\_CBC  
 cryptor = AES.new(key, mode,iv) #创建实例  
 try:  
 plain\_text = cryptor.decrypt(a2b\_hex(text))#解密  
 print(bytes.decode(plain\_text).rstrip('\0'))  
 return bytes.decode(plain\_text).rstrip('\0') #转换为python的字符串  
 except:  
 return '请检查解密数据或者密钥是否正确！'  
 def cbc\_base64\_jiami(self, text,keys,plys):  
 *'''CBC base64加密函数'''* key = self.add\_to\_16(keys)  
 # key = 'qdwphltib5kyfd3n'.encode('utf-8')  
 iv = self.add\_to\_16(plys)  
 mode = AES.MODE\_CBC  
 cryptos = AES.new(key, mode,iv)  
 text = self.add\_to\_16(text)  
 t = cryptos.encrypt(text) #加密明文，text必须为字节bytes  
 t = base64.encodebytes(t) #将返回的字节数据进行base64编码  
 t = t.decode('utf-8') #将字节型数据转换成python中的字符串类型  
 try:  
 t = t.split()[0] + t.split()[1] # 去除换行符  
 print(t)  
 return t  
 except:  
 return t  
 def cbc\_base64\_jiemi(self, text,keys,plys):  
 *'''CBC base64解密函数'''* key = self.add\_to\_16(keys)  
 iv = self.add\_to\_16(plys)  
 # key = 'qdwphltib5kyfd3n'.encode('utf-8')  
 text = base64.b64decode(text) # 转换为base64格式，为bytes  
 mode = AES.MODE\_CBC  
 cryptor = AES.new(key, mode,iv)  
 try:  
 plain\_text = cryptor.decrypt(text) # 解密  
 print(bytes.decode(plain\_text).rstrip('\r'))  
 return bytes.decode(plain\_text).rstrip('\r')  
 except:  
 return '请检查解密数据或者密钥是否正确！'

# Selenium笔记

***本笔记中的变量wd为自定义的浏览器驱动***

***wd = webdriver.Firefox()***

***selenium2集成了火狐浏览器驱动,selenium3没有集成浏览器驱动***

***谷歌浏览器驱动下载地址：chromedriver.storage.googleapis.com/index.html***

***谷歌驱动下载减压（只有一个.EXE文件）放在pyhton根目标***

***selenium2只能用46及以下的火狐浏览器***

***selenium3支持46以上版本***

## 各浏览器版本驱动下载

**Chrome**

点击下载chrome的webdriver： <http://chromedriver.storage.googleapis.com/index.html>

不同的Chrome的版本对应的chromedriver.exe 版本也不一样，下载时不要搞错了。如果是最新的Chrome, 下载最新的chromedriver.exe 就可以了。

把chromedriver的路径也加到环境变量里。

**Firefox**

Firefox驱动下载地址为：<https://github.com/mozilla/geckodriver/releases/>

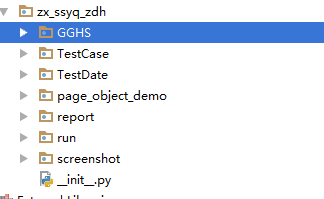
根据自己的操作系统下载对应的驱动即可，使用的话，需要把驱动的路径和火狐浏览器的路径加入到环境变量里面才可以

**IE**

IE浏览器驱动下载地址为：<http://selenium-release.storage.googleapis.com/index.html>

根据自己selenium版本下载对应版本的驱动即可，python的话，下载里面的IEDriverServerxxx.zip即可，这个是区分32和64位系统的，根据自己的系统下载即可，需要注意的是，如果要打开IE浏览器的话，需要在浏览器的Internet选项中的安全页里有4个安全选项，Internet、本地Internet、受信任的站点、受限制的站点，这4个里面都有一个启用保护模式，都需要勾选上才可以，还得把驱动的路径加入到环境变量中。

## 自动化测试框架



依次为：公共函数区、测试用例区、测试数据区、页面对象区、测试报告区、脚本运行区、截图存放区

## Unittest讲解

Unittest为python自带的自动化测试框架，一般在做自动化的时候用此框架来执行脚本，该脚本有2个函数setUp()和tearDown()与之前后对应

setUp()：框架的初始化操作，一般用于初始化脚本中的变量，也可以为空

tearDown()：一个函数（用例）执行完后，执行此函数

注意：

1. unittest脚本中必须有一个可执行函数，函数名称必须以test开头，否则找不到函数，报函数为空的错误
2. unittest执行是每执行一个用例函数就会执行一次setUp()和tearDown()函数
3. 在pycharm中，默认运行是用unittest框架机制运行的，所以不会去执行**if** \_\_name\_\_ == **'\_\_main\_\_'**:函数，这样就不会生成HTML报告了，如何区分是以什么方式运行的呢？

右键运行带有unittest的就不会运行**if** \_\_name\_\_ == **'\_\_main\_\_'**:，如果处理不了这个问题，可以在cmd中运行脚本，就会生成html报告了

1. 用例函数执行方式是以字符ascii码顺序执行的，所以用例有可能不是从上往下依次执行,如果要让用例顺序执行可以这样设计：test\_case1\_xxxx, test\_case2\_xxx
2. 在unittest框架中，用例函数之间可以相互调用，常用在接口测试时，下一个接口需要用到上一个接口的返回值作为参数传递，调用方法 lei. test\_case1\_xxxx(self)，注意了：此时类不需要再实例化，所有类后面不再跟括号()，脚本例子

*#coding=utf-8***import** unittest  
**class** lei(unittest.TestCase):  
 **def** setUp(self):  
 self.a =10  
 **def** test\_case1\_a(self):  
 self.b = 20**return** self.a + self.b  
 **def** test\_case2\_b(self):  
 self.c = lei.test\_case1\_a(self)  
 **print** self.c + 120  
 **def** tearDown(self):  
 **print 'end'**

## Unittest常用的两种结构

### setUp()结构

该框架结构在执行用例时，每次都会去执行setUp和tearDown里的东西

**class** lei(unittest.TestCase):  
 **def** setUp(self):  
   
 **def** tearDown(self):

使用场景：用例关系都是独立的时候，需要每次进行初始化操作

### setUpClass()结构

该结构，在执行测试用例时，只需要去执行一次setUpClass里的东西，该类里面的用例全部执行完后才会去执行tearDownClass

**class** lei(unittest.TestCase):

**@classmethod**  
 **def** setUpClass(self):

**@classmethod**  
 **def** tearDownClass(self):

使用场景：用例关系是一个流程依赖关系时，只需要进行一次初始化操作

## Unittest常用断言

1. assertEqual(a,b) 判断a和b是否相等
2. assertNotEqual(a,b) 判断a和b是否不相等
3. assertIsNone(x) 判断x元素是空
4. assertIsNotNone(x) 判断x元素不为空
5. assertIn(a,b) 判断a在b里面
6. assertNotIn(a,b) 判断a不在b里面

用断言时这样使用：self. assertEqual(a,b)

## unittest跳过测试用例

执行测试用例时，有时候有些用例是不需要执行的，那我们怎么办呢？难道删除这些用例？那下次执行时如果又需要执行这些用例时，又把它补回来？这样操作就太麻烦了。

unittest提供了一些跳过指定用例的方法

* @unittest.skip(reason)：强制跳转。reason是跳转原因
* @unittest.skipIf(condition, reason)：condition为True的时候跳转
* @unittest.skipUnless(condition, reason)：condition为False的时候跳转
* @unittest.expectedFailure：如果test失败了，这个test不计入失败的case数目

[复制代码](javascript:void(0);)

# coding = utf-8

import unittest

import warnings

from selenium import webdriver

from time import sleep

# 驱动文件路径

driverfile\_path = r'D:\coship\Test\_Framework\drivers\IEDriverServer.exe'

class CmsLoginTest(unittest.TestCase):

def setUp(self):

# 这行代码的作用是忽略一些告警打印

warnings.simplefilter("ignore", ResourceWarning)

self.driver = webdriver.Ie(executable\_path=driverfile\_path)

self.driver.get("http://172.21.13.83:28080/")

def tearDown(self):

self.driver.quit()

@unittest.skip("用户名密码都为空用例不执行")

def test\_login1(self):

'''用户名、密码为空'''

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message1 = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

error\_message2 = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message1, '用户名不能为空')

self.assertEqual(error\_message2, '密码不能为空')

@unittest.skipIf(3 > 2, "3大于2，此用例不执行")

def test\_login3(self):

'''用户名、密码正确'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("111111")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

sleep(1)

self.driver.switch\_to.frame("topFrame")

username = self.driver.find\_element\_by\_css\_selector("#nav\_top>ul>li>a").text

self.assertEqual(username,"autotest")

@unittest.skipUnless(3 < 2,"2没有大于3，此用例不执行")

def test\_login2(self):

'''用户名正确，密码错误'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '密码错误,请重新输入!')

## Webdriver 脚本

### 元素定位

css定位速度比普通定位速度快

1. 普通定位
2. id定位 find\_element\_by\_id(“”)
3. name定位 find\_element\_by\_name(‘’)
4. class定位 find\_element\_by\_class\_name(‘’)
5. tag定位find\_element\_by\_tag\_name(‘’) 每一个元素都是一个tag，所以该定位法不是唯一的
6. link定位 find\_element\_by\_link\_text(‘’)定位一个文本数据。一般是超链接a标签
7. xpath定位find\_element\_by\_xpath('’) xpath路径的获取：右键复制xpath路径
8. 绝对路径定位:一个完整的路径
9. 元素属性定位 ：例 //input[@id=’kw’]
10. 层级与属性结合：例 //form[@id=’form’]/span/input
11. 逻辑运算定位：当一个属性不能唯一区分时，可以用多个属性组合定位，用and连接

注意：如果找报不到元素的错误，一般是需要设置等待时间 time.sleep(3)

1. css定位

class定位：find\_element\_by\_css\_selector(‘.class值’)，用点号

id定位：find\_element\_by\_css\_selector(‘#id值’)，用#号

元素属性定位，一个元素的任意属性都可以，只要是惟一的

find\_element\_by\_css\_selector(‘[id=”id值”]’)，用大括号，里面填上属性及值

### 控制浏览器

1. 设置浏览器窗口大小
2. set.window.size() 例设置窗口为400,800： set.window.size(400,800)
3. maximize\_window() 直接设置浏览器全屏
4. 控制浏览器前进、后退
5. back() 后退
6. forward() 前进
7. 浏览器刷新：refresh()

函数的使用：wd.back()

### 元素操作

1. clean() 清除文本
2. send\_keys(‘’) 模拟键盘输入
3. click() 鼠标点击
4. submit() 用于提交表单，比如在搜索框输入内容后的回车操作，就可以通过该函数模拟
5. size 返回元素的尺寸，直接元素后面跟 .size
6. text 返回元素的文本类型为unicode，在脚本断言文本的时候需要在字符串前加上 u，转化为unicode计算机语言，例如要断言‘百度一下’，需要使用u‘百度一下’
7. get\_attribute(‘’) 获得元素属性的值

比如要想获得元素中class的值，使用方法为：

元素. get\_attribute(‘class’)

1. is\_displayed() 设置元素是否用户可见，返回的是一个布尔类型

例：wd.find\_element\_by\_id(“fw”).clean() 清除百度输入框中的文本信息，所有函数都是这个用法

### 鼠标事件

1. perform() 执行所有的行为，事件后面要跟这个函数
2. context\_click() 右键
3. double\_click() 双击
4. move\_to\_element() 鼠标悬停

上面几种事件的使用方法如下：

首先导入ActionChains类

from selenium.webdriver.common.action\_chains import ActionChains

shijian = wd.find\_element\_by\_id(‘kw’) #wd为浏览器驱动，wd=webdriver.Firefox()

ActionChains(wd).context\_click(shijian).perform()

### 键盘操作

#### 0.导入键盘类Keys()

* selenium中的Keys()类提供了大部分的键盘操作方法；通过send\_keys()方法来模拟键盘上的按键。

# 导入键盘类 Keys()

from selenium.webdriver.common.keys import Keys

#### 1.常用的键盘操作：

* send\_keys(Keys.BACK\_SPACE)：删除键(BackSpace)
* send\_keys(Keys.SPACE)：空格键(Space)
* send\_keys(Keys.TAB)：制表键(TAB)
* send\_keys(Keys.ESCAPE)：回退键(ESCAPE)
* send\_keys(Keys.ENTER)：回车键(ENTER)
* send\_keys(Keys.CONTROL,'a')：全选(Ctrl+A)
* send\_keys(Keys.CONTROL,'c')：复制(Ctrl+C)
* send\_keys(Keys.CONTROL,'x')：剪切(Ctrl+X)
* send\_keys(Keys.CONTROL,'v')：粘贴(Ctrl+V)
* send\_keys(Keys.F1)：键盘F1
* .....
* send\_keys(Keys.F12)：键盘F12

#### 2.实例

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

driver=webdriver.Chrome()

driver.get("http://www.baidu.com")

#输入关键词内容

driver.find\_element\_by\_id("kw").send\_keys("selenium")

#删除键

driver.find\_element\_by\_id("kw").send\_keys(Keys.BACK\_SPACE)

#空格键

driver.find\_element\_by\_id("kw").send\_keys(Keys.SPACE)

#输入内容

driver.find\_element\_by\_id("kw").send\_keys("教程")

#全选(Ctrl+A)

driver.find\_element\_by\_id("kw").send\_keys(Keys.CONTROL,'a')

#剪切(Ctrl+X)

driver.find\_element\_by\_id("kw").send\_keys(Keys.CONTROL,'x')

#粘贴(Ctrl+V)

driver.find\_element\_by\_id("kw").send\_keys(Keys.CONTROL,'v')

#回车键

driver.find\_element\_by\_id("kw").send\_keys(Keys.ENTER)

### windows控件文件上传下载

#### 用到工具autoit

Autoit下载地址：<https://www.autoitscript.com/site/autoit/downloads/>

进页面后往下拉，点击入下图所示



安装autoit，一路下一步完成

#### 使用autoit

安装完成后有多个应用程序可使用，具体功能如下：

1.AutoIt Window Info用来识别Windows控件，根据识别的控件信息编写脚本；x86为32位

2.SciTE Script Editor用来写脚本，并保存为au3格式，按F5可以调试代码，但需要是操作弹框（上传/下载/另存为弹框）开启的情况下

3.Complie Script to .exe， 将刚编写的脚本，编译成可执行文件，勾选x64；

4.编译后在Python代码中调用

#### 实例讲解（浏览器右键下载图片）

实现网页图片右键菜单图片另存为操作，具体思路如下：

1. python+selenium编写脚本，在图片上弹出右键菜单
2. 使用autoit工具，实现点击另存为，保存图片操作
3. 到python代码中调用

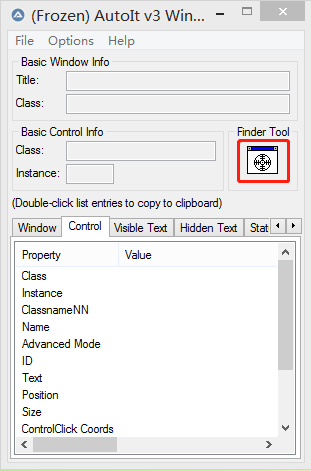
具体步骤详细描述：

##### 第一步：写代码，弹出右键菜单

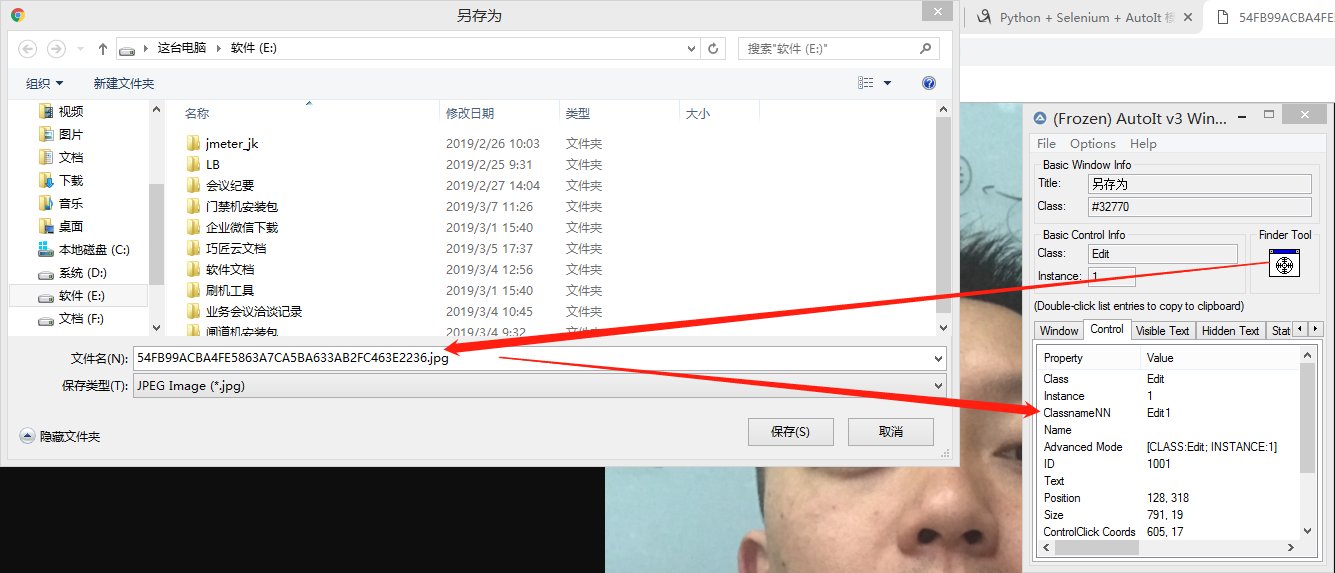
**from** selenium **import** webdriver  
**from** selenium.webdriver.common.action\_chains **import** ActionChains  
**import** time  
wd = webdriver.Chrome()  
wd.maximize\_window()  
wd.get(j)  
time.sleep(0.5)  
ys = wd.find\_element\_by\_xpath(**'/html/body/img'**)  
action = ActionChains(wd).move\_to\_element(ys)  
action.context\_click(ys).perform()

##### 第二部：使用autoit工具

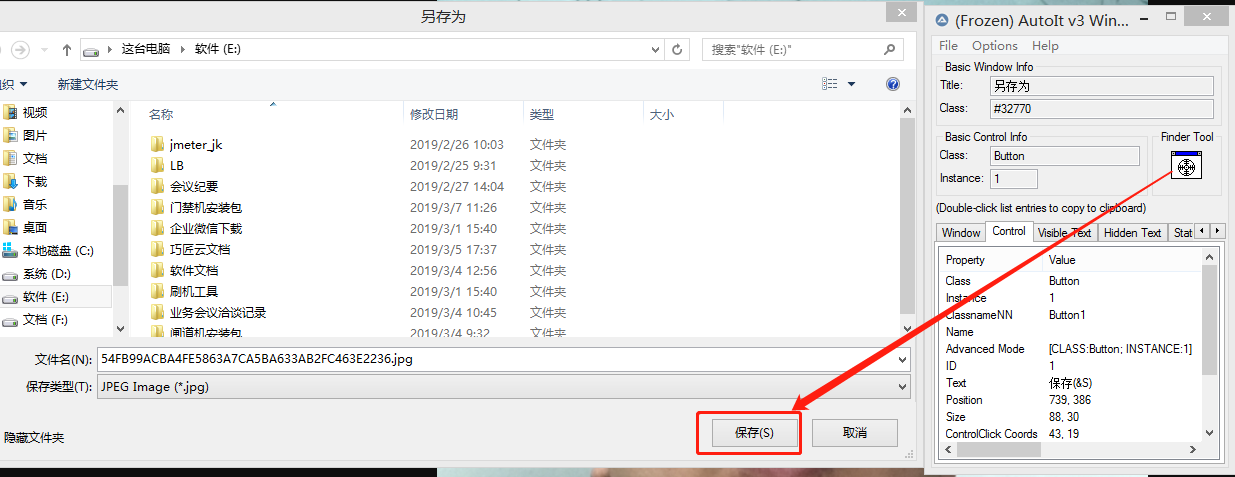
###### 打开AutoIt Window Info程序



点击红框处按钮，按住左键不放拖动到windows控件上，如图



图中拖动到文件名栏，工具上显示出了文件名栏的属性



拖动到保存按钮上，显示出了保存按钮的属性

###### 打开SciTE Script Editor程序来编写代码，保存为au3格式，代码如下

send("v")

Sleep(1000)

ControlFocus("另存为","","Edit1")

WinWait("[CLASS:#32770]","",10)

ControlSetText("另存为","", "Edit1", " baidu1.png")

Sleep(2000)

ControlClick("另存为","","Button1")

代码解释：

第一行：send("v") 模拟键盘输入V，既点击图片另存为

第二行：Sleep ( 延迟 ) 使脚本暂停指定时间段；

第三行：ControlFocus ( "title", "窗口文本", controlID) 设置输入焦点到指定窗口的某个控件上，

窗口文本可以为空，controlid为ClassnameNN的值；

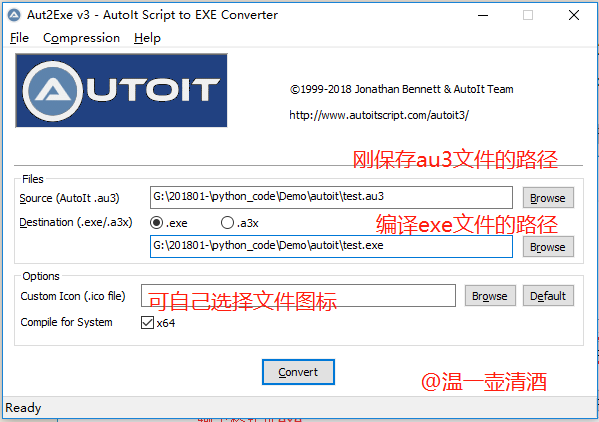
第四行：WinWait ( "title" , "窗口文本" , 超时时间 ) 暂停脚本的执行直至指定窗口存在（出现） 为止，窗口文本用[CLASS:#32770]；

第五行：ControlSetText ( "title", "窗口文本", controlID, "文件名" ) 修改指定控件的文件名；

第六行：Sleep ( 延迟 ) 使脚本暂停指定时间段；

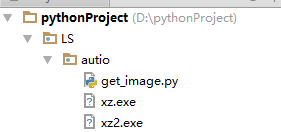
第七行：ControlClick ( "title", "窗口文本", controlID ) 向指定控件发送鼠标点击命令；其中， title即AutoIt Window Info识别出的Title字段， controlID即AutoItWindow Info识别出的Class和Instance的拼接， 如上图拼接后的结果应为：Button1，也就是ClassnameNN的值。

###### 打开Complie Script to .exe工具把刚编写的脚本编译成可执行文件，操作如下：



编辑后生成一个test.exe的可执行文件，拷贝到python的工程下

##### 第三部：python脚本调用



操作代码如下：

import os

os.system(os.getcwd() + "\\test.exe")

注释：os.system(os.getcwd()) 获得当前脚本所在的路径，

*D:\pythonProject\LS\autio*

后面再拼接可执行文件，完成

##### 整体代码实现如下：

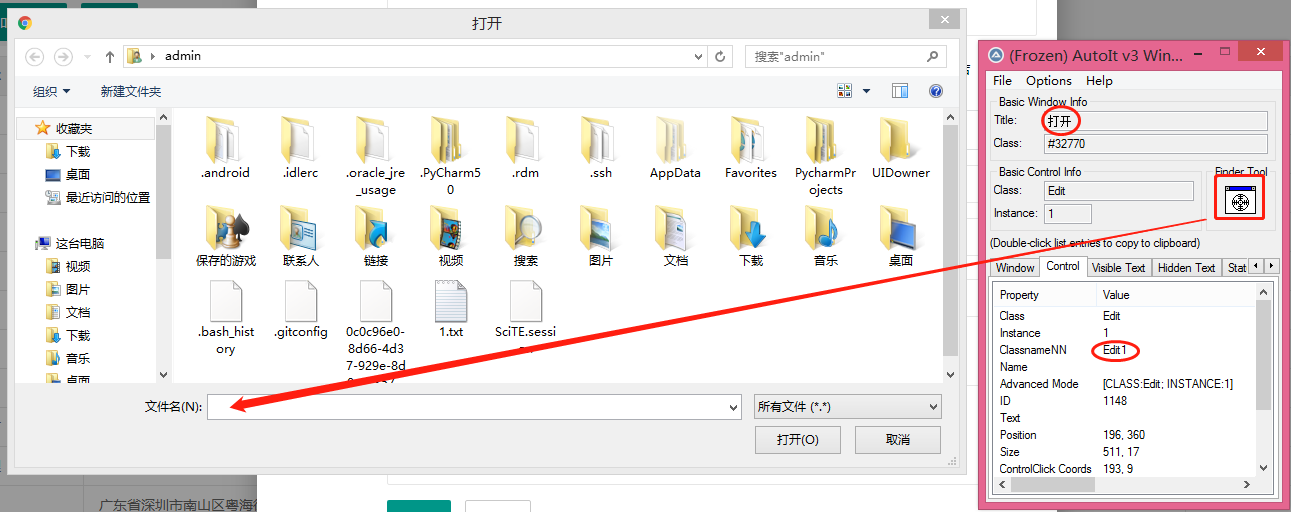
**from** selenium **import** webdriver  
**from** selenium.webdriver.common.action\_chains **import** ActionChains  
**import** time  
**import** os  
**import** csv  
book = csv.reader(open(**'e://face.csv'**,**'r'**))  
wd = webdriver.Chrome()  
wd.maximize\_window()  
**for** i **in** book:  
 **for** j **in** i:  
 wd.get(j) #获取url地址  
 time.sleep(0.5)  
 ys = wd.find\_element\_by\_xpath(**'/html/body/img'**)  
 action = ActionChains(wd).move\_to\_element(ys)  
 action.context\_click(ys).perform()  
 os.system(os.getcwd() + **"\\xz2.exe"**) *# 调用xz2.exe文件  
 # print(os.getcwd()) #获取脚本所在路径 D:\pythonProject\LS\autio*wd.quit()

#### 实例讲解（文件上传）

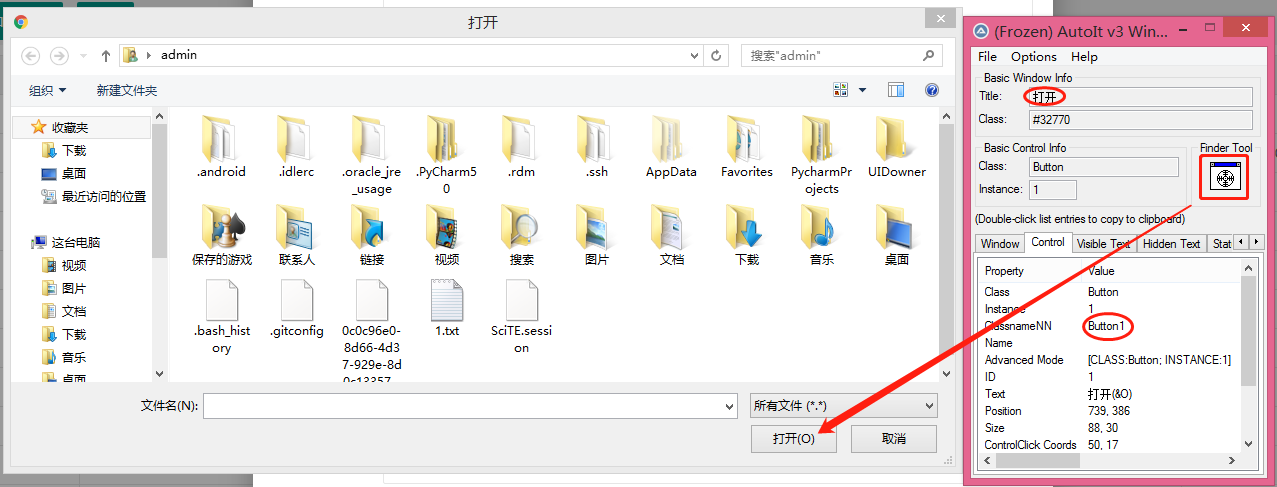
##### 第一步：写代码，点击浏览按钮，弹出windows上传框

##### 第二步：使用autoit工具

###### 打开AutoIt Window Info程序



按住方框按钮，拖动到文件名栏，显示出了文件名栏的属性



按住方框按钮，拖动到打开按钮上，显示出了打开按钮的属性

###### 打开SciTE Script Editor程序来编写代码，保存为au3格式，代码如下

Sleep(500)

ControlFocus("打开","","Edit1")

WinWait("[CLASS:#32770]","",10)

ControlSetText("打开","", "Edit1", "1.txt")

Sleep(500)

ControlClick("打开","","Button1")

代码解释：

第一行：Sleep ( 延迟 ) 使脚本暂停指定时间段；

第二行：ControlFocus ( "title", "窗口文本", controlID) 设置输入焦点到指定窗口的某个控件上，

窗口文本可以为空，controlid为ClassnameNN的值；

第三行：WinWait ( "title" , "窗口文本" , 超时时间 ) 暂停脚本的执行直至指定窗口存在（出现） 为止，窗口文本用[CLASS:#32770]；

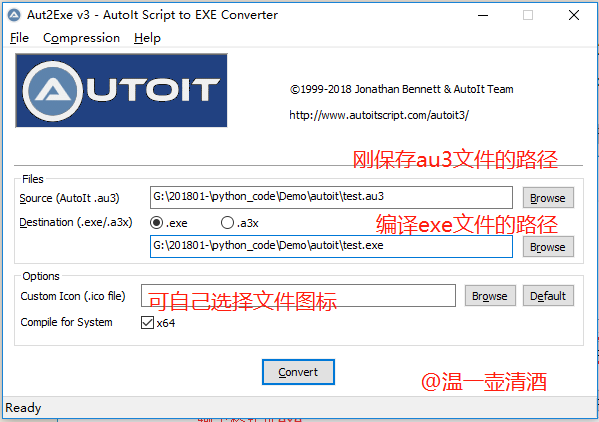
第四行：ControlSetText ( "title", "窗口文本", controlID, "文件名" ) 修改指定控件的文件名；

第五行：Sleep ( 延迟 ) 使脚本暂停指定时间段；

第六行：ControlClick ( "title", "窗口文本", controlID ) 向指定控件发送鼠标点击命令；其中， title即AutoIt Window Info识别出的Title字段， controlID即AutoItWindow Info识别出的Class和Instance的拼接， 如上图拼接后的结果应为：Button1，也就是ClassnameNN的值。

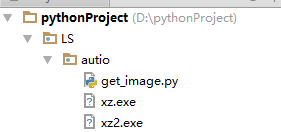
注意：第四行输入的是要上传的文件名，上传的文件要放在打开的默认窗口下

###### 打开Complie Script to .exe工具把刚编写的脚本编译成可执行文件，操作如下：



编辑后生成一个test.exe的可执行文件，拷贝到python的工程下

##### 第三部：python脚本调用



操作代码如下：

import os

os.system(os.getcwd() + "\\test.exe")

注释：os.system(os.getcwd()) 获得当前脚本所在的路径，

*D:\pythonProject\LS\autio*

后面再拼接可执行文件，完成

脚本调用拼接exe文件代码如下

wd.find\_element\_by\_xpath(**'/html/body/div/div[7]/div/button[1]'**).click() #点击上传按钮  
os.system(os.getcwd() + **"\\sc2.exe"**) #调用exe程序

### Cookie操作

wd = webdriver.Chrome()

##### 获取cookie

wd.get\_cookies() #获取当前页面的cookie

##### 添加cookie

# 向cookie中name和value中添加回话信息

wd.add\_cookie ({'name': 'key-aaaaaaa'}, {'value': 'value-bbbbb'})

##### 清除cookie

1. 清除指定cookie

Wd.delete\_cookie(name=’ key-aaaaaaa’) #清楚指定name的cookie

1. 清除所有cookie

Wd.delete\_all\_cookie()

##### 跳过登录验证

方法：删除原有cookie，添加登录后的cookie

先正常登录一次系统，用wd.get\_cookies()获取登录后的所有cookies

脚本编写：

1. 脚本打开web网页链接
2. 删除当前cookies
3. 添加前面获取到的cookie，注意cookie是字典
4. 再次打开web网页链接，这是就会直接进入系统了

### 等待时间

1. **显示等待**

WebDriverWait(wd,5,0.5) 每0.5秒刷新一次，5秒后结束，一般与unitl()套用

例： a = WebDriverWait(wd,5,0.5).until(EC.presence\_of\_element\_located((By.ID,’kw’)))

使用该函数需要先导入：

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver. support import expected\_conditions as EC

该等待方式如果在最大时间内找到了该元素就提前运行下一脚本内容，不会影响脚本运行时间

1. **隐式等待**：wd.implicitly\_wait(10)

隐式等待写在浏览器对象后，该方式不是指对某一元素的等待，当页面对象任意元素没有找到时，会去执行该隐式等待，超过等待时间后抛出异常

wd = webdriver.Firebox()

wd.implicitly\_wait(10) 等待10秒

1. **sleep() 休眠方式**，一般这个使用的最多

使用该方法需要先导入时间模块 import time,使用：time.sleep(3) 等待3秒

### 定位一组元素

把元素定位里的函数element改成elements就行了

经常用的是：find\_elements\_by\_tag\_name，定位一组标签，再取其中的一个属性

例：一组勾选框，其标签为input，其中type的属性值都是checkbox，想勾选全部

a = wd.find\_elements\_by\_tag\_name(‘input’) #定位一组元素

for I in a:

if i.get\_attribute(‘type’) == ‘checkbox’

i.click() # 进行单击勾选操作

### 多表单切换

在web应用中经常会遇到frame/iframe表单嵌套页面的应用，这时候直接定位表单里面的元素是定位不到的，必须先定位到表单

函数：switch\_to.frame() 切换到表单页面，该函数默认可以取元素的id和name属性

例：iframe的id = ‘if’ 可以这样使用switch\_to.frame(‘if’)，这个时候就可以定位表单下面的元素了

如果表单iframe没有id和name属性，则可以先用xpath定位到iframe并赋值给变量，

switch\_to.frame(变量)

switch\_to.parent\_frame() 方法可以跳出当前表单到最近的表单

switch\_to.default\_content() 方法跳回到最外层的页面,也是最顶层

### 多窗口切换

#定位初始窗口

csck = wd.current\_window\_handle

#定为所有打开的窗口

all\_ck = wd.window\_handles

#写一个循环遍历所有窗口，判断遍历的窗口不等于初始窗口时，就定位到当前窗口

for ck in all\_ck:

if ck != csck:

wd.switch\_to.window(ck) #定位到当前窗口

#当前窗口下面可以定位元素了

#回到初始窗口需要使用wd.switch\_to.window(csck)

脚本例子：

*#coding=utf-8***from** selenium **import** webdriver  
**import** unittest  
**import** time  
**class** huohushouye(unittest.TestCase):  
 **def** setUp(self):  
 self.wd = webdriver.Firefox()  
 self.url = **'http://home.firefoxchina.cn/?from=worldindex'** self.wd.maximize\_window()  
  
 **def** test\_weibo\_case1(self):  
 *''' 微博链接测试 '''* wd = self.wd  
 wd.get(self.url)  
 time.sleep(3)  
 csck = wd.current\_window\_handle  
 wd.find\_element\_by\_link\_text(**'微　博'**).click()  
 time.sleep(10)  
 all\_ck = wd.window\_handles  
 **for** ck **in** all\_ck:  
 **if** ck != dqck:  
 wd.switch\_to.window(ck)  
 a = wd.find\_element\_by\_link\_text(**'注册'**).text  
 **print** a  
 self.assertEqual(a,**u'注册'**,msg = **'预期结果错误'**)  
 **def** test\_wangyi\_case2(self):  
 *''' 百度链接测试 '''* wd = self.wd  
 wd.get(self.url)  
 time.sleep(3)  
 csck = wd.current\_window\_handle  
 wd.find\_element\_by\_link\_text(**'百　度'**).click()  
 time.sleep(10)  
 all\_ck = wd.window\_handles  
 **for** ck **in** all\_ck:  
 **if** ck != csck:  
 wd.switch\_to.window(ck)  
 a = wd.find\_element\_by\_id(**'su'**).get\_attribute(**'value'**)  
 **print** a  
 self.assertEqual(a,**u'百度一下'**,msg = **'预期结果错误'**)  
 **def** tearDown(self):  
 **print 'end'** self.wd.quit()  
**if** \_\_name\_\_ == **'\_\_main\_-'**:  
 unittest.main()

### 关闭浏览器窗口

wd.quit() 关闭浏览器

wd.close() 关闭当前浏览器打开窗口

### 时间获取

time.time() 获取当前时间戳

time.ctime() 当前时间的字符串形式

time.strftime('%Y-%m-%d %H:%M:%S')

获得当前时间，并指定格式字符串

### 下拉框选择

1. select下拉框选择方法如下：可以直接使用xptah定位
2. 其它下拉框或者是上面下拉框选择不到的情况下使用下面方法定位
3. 先用xpath定位到下拉框

self.select = self.wd.find\_element\_by\_xpath('')

把定位到的select对象赋值给变量self.select

1. 再用xpath定位到下拉框中的值，并点击它

self.select.find\_element\_by\_xpath('’).click()

注意了：这里直接用self.select对象来定位xpath

1. 鼠标悬停和右键出现的下拉框选择方法：

直接用find\_element\_by\_link\_text()方法定位

### 时间选择框处理

1. 每个前端写的获取日期的方法不一致，获取时间时间参数就不一样，前端有2种获取日期的方式
2. 直接取时间输入框里的值
3. 不取时间框里的值，取的是隐藏属性的值
4. 针对直接取时间框里的值作为时间参数传递的处理方法：

我们不去搞时间日期控件，我们把它当成一个普通的input框处理就好了！

可以输入的时间选择框

直接定位到输入框用send\_keys()进行时间输入

1. input时间框框禁止输入

很简单，用js把禁止输入的readonly属性干掉就好了。

下面4种方法可解除禁止输入，用send\_keys()直接输入时间

# js = "document.getElementById('txtBeginDate').removeAttribute('readonly')" # 1.原生js，移除'readonly'属性，'readonly'顾名思义就是【只读】

# js = "$('input[id=txtBeginDate]').attr('readonly',false)" # 3.jQuery，设置为false

#js = "$('input[id=txtBeginDate]').attr('readonly','')" # 4.jQuery，设置为空（同3）

js = "$('input[id=txtBeginDate]').removeAttr('readonly')" # 2.jQuery，移除只读属性

# input[id=txtBeginDate]时间框的id值，这里可以是name

driver.execute\_script(js)*#执行js脚本*

*移除属性后有2种方法输入时间*

1. driver.find\_element\_by\_id('txtBeginDate').send\_keys('2016-08-24') #直接输入
2. js\_value = 'document.getElementById("txtBeginDate ").value="2017-12-10"'

driver.execute\_script(js\_value) # js添加时间

注意：上面4种方法总有一种会行的，一个不行就换一个使用

如果遇到的日期时间框没有id、name，这里提醒一下，这就是普通的input框，定位方法太多了，class、tag name、xpath、css，总有一种适合你。同时友情提醒一下：注意时刻提防iframe！

1. 不取时间框里的值，取的是隐藏属性的值的方法：

此时需要联系写前端的同事，获取隐藏的日期属性值，获取到后用下面的代码

document.getElementById('createTime').removeAttribute('type')  
这一句可以把隐藏的元素弄出来

driver.execute\_script(js)*#执行js脚本*

driver.find\_element\_by\_id('createTime').send\_keys('2016-08-24') #直接输入

### 产生随机数

用到python包 random

* 产生1到100的随机整数

random.randint(1,100) #

* 产生随机字符串

random.sample(‘abcdefg12345678’,6) #在字符串里产生6个随机数，生成的值如下

[‘a’, ’c’,’ 1’, ’4’, ’d’, ’8’],需要对改list进行拼接成字符串，方法如下

Lists = [‘a’, ’c’,’ 1’, ’4’, ’d’, ’8’]

Str = ‘’.join(lists)

* 产生区间随机小数

random.uniform(10, 90)

### 操作屏幕显示之外的元素

为什么要用这种方法？

因为如果你操作屏幕显示以外的元素的话会报错，所以必须要滑动屏幕直到要使用的元素出现在当前屏幕上才行

注意：python3可直接定位屏幕之外的元素了

二种方法

1. 用js方法来实现

将页面滚动条移动到页面任意位置，改变等于号后的数值即可，目前火狐和谷歌浏览器是可以跑通的，注意：在使用滑动前最好加上time.sleep()等待时间，不然会没反应, execute\_script(js)执行js语句

1. 滑动到浏览器底部

time.sleep(2)

js="var q=document.documentElement.scrollTop=10000"

driver.execute\_script(js)

1. 滑动到浏览器顶部

js="var q=document.documentElement.scrollTop=0"

driver.execute\_script(js)

1. 设置浏览器的水平和垂直位置

js = “window.scrollTo(100,450);”

wd.execute\_script(js)

1. 利用鼠标悬停的方法来实现

from selenium.webdriver.common.action\_chains import ActionChains#导入鼠标操作模组

shijian = wd.find\_element\_by\_id(‘kw’) #wd为浏览器驱动，wd=webdriver.Firefox()

ActionChains(wd). move\_to\_element(shijian).perform()

shijian.click()

### JavaScript文本输入框

这种输入框虽然我们可以进行定位，但是无法使用send\_keys()向里面输入东西，我们需要借助JavaScript代码来完成输入

例

text = “shuruneirong

js = “var sum=document.getElementByID(‘id’); sum.value=’” + text +”’;”

wd.execute\_script(js)

### 按钮是否可点击的判断（按钮是否置灰）

用元素的disabled属性，如果不可点击，该属性值为true，否则为None

比如 a = wd.find\_element\_by\_id(‘kw’).disabled

print a

输出为true或者None

### 文件上传

如果上传元素框为input标签并且type类型为file，可直接使用send\_keys(‘文件路径’)上传文件

比如：a = wd.find\_element\_by\_id(‘kw’). send\_keys(‘d://test.txt’)

### web弹窗处理

在web中有三种弹窗方式alert、div、js三种方式弹窗

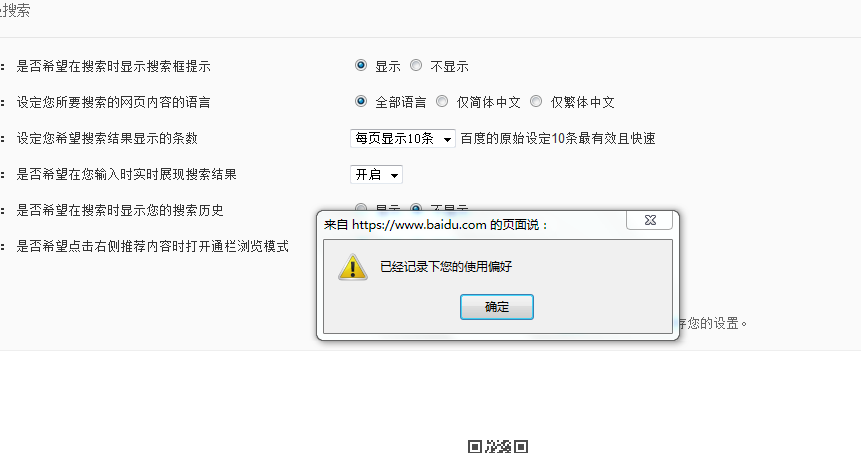
1. alert弹窗，该弹窗无法进行定位

在webdriber中处理javascript所生成的alert、confirm、prompt弹窗十分简单，使用

switch\_to\_alert()方法定位到alert、confirm、prompt弹窗，然后使用text,accept(),dismiss(),send\_keys方法进行操作

1. text：获取弹窗文本信息
2. accept()：接受弹窗
3. dismiss()：解散弹窗
4. send\_keys(“”) 将文本发送至弹窗中，如果有输入框的话

以下面百度首页设置为例，在点击保存设置后会出现如下弹窗：



处理这个弹窗的脚本如下：

*#coding=utf-8***from** selenium **import** webdriver  
**import** time  
**from** selenium.webdriver.common.action\_chains **import** ActionChains  
wd = webdriver.Firefox()  
wd.maximize\_window()  
wd.get(**'http://www.baidu.com'**)  
time.sleep(2)  
wd1 = wd.find\_element\_by\_link\_text(**'设置'**)  
ActionChains(wd).move\_to\_element(wd1).perform()  
wd.find\_element\_by\_link\_text(**'搜索设置'**).click()  
wd.find\_element\_by\_link\_text(**'保存设置'**).click()  
*#点击保存设置后会有一个弹窗*time.sleep(2)  
a = wd.switch\_to\_alert() *#接受这个弹窗***print** a.text *#获取弹窗文本*a.dismiss() *#解散弹出，相当于取消  
#a.accept() #接受这个弹窗，相当于确定*

1. div弹窗

该弹窗能进行元素定位，直接用普通的元素定位方式即可

注意：该弹窗是否是iframe或者是新窗口，如果是都要进行转换过去

如果以上两种都不是，直接定位又报错，那就是说明该div弹窗与操作页面元素层属于平级关系，需要使用：switch\_to.default\_content() 方法跳回到最外层的页面,也是最顶层，才能对元素进行定位。这种方式曾经困扰了笔者几个小时，用尽所有方式都不行

### 窗口截图

wd.get\_screenshot\_as\_file(**"d:/weibo.jpg"**)

### 浏览器对象转移

a = wd.find\_element\_by\_xpath(‘’)

a. find\_elements\_by\_tag\_name('tr')#定位a对象下的一组tr标签

利用该方法可以指定定位某一对象下的某一组标签

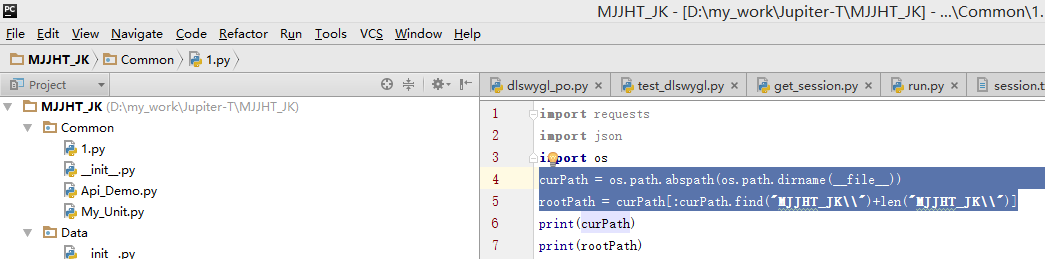
脚本实例：

**def** test\_case1\_ckbjg(self):  
 self.wd.find\_element\_by\_link\_text(**'查看表结构'**).click()  
 time.sleep(1)  
 self.select = self.wd.find\_element\_by\_xpath(**'/html/body/div[3]/div[2]/div[1]/div[1]/div[1]/div/div/select'**)  
 *#定位到select下拉框* self.select.find\_element\_by\_xpath(**'/html/body/div[3]/div[2]/div[1]/div[1]/div[1]/div/div/select/option[2]'**).click()  
 *#选择Demo1* self.a1 = self.wd.find\_element\_by\_xpath(**'/html/body/div[3]/div[2]/div[1]/div[2]/div[2]/table/tbody'**)  
 time.sleep(2)  
 *#定位到tbody，并把对象赋值给self.a1* trsl = self.a1.find\_elements\_by\_tag\_name(**'tr'**)  
 *#利用self.a1对象来查找该对象下的tr标签* self.assertEqual(len(trsl),int(12))*#断言tr标签有12个*

### 文件路径获取

首先导入包：import os

1. os.path.abspath(‘文件名含后缀’) #获得当前文件的完整路径
2. os.path.realpath(\_\_file\_\_) #获得当前文件的完整路径
3. os.path.dirname(os.path.realpath(\_\_file\_\_)) # #获得文件的所在目录路径
4. os.path.dirname(os.path.abspath(‘文件名含后缀’)) #获得文件的所在目录路径
5. 获取项目根路径，如图



项目名为MJJHT\_JK，获取根目录的代码（二种方法）：

第一种：

curPath = os.path.abspath(os.path.dirname(\_\_file\_\_)) #获取脚本所在路径  
rootPath = curPath[:curPath.find(**"MJJHT\_JK\\"**)+len(**"MJJHT\_JK\\"**)] #获取MJJHT\_JK根目录

输出结果：D:\my\_work\Jupiter-T\MJJHT\_JK\

第二种：

curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
rootPath = os.path.split(curPath)[0] + '\\' #直接获取项目根路径，不用指定项目名

输出结果：D:\my\_work\Jupiter-T\MJJHT\_JK\

### 文件读取

1. **读取txt文件**

read() 读取整个文件

readline() 读取文件的一行

readlines() 读取文件所有行数据

例：有一用户文本文件user.txt其文件内容如下：

user1,123

user2,123

user3,123

测试脚本test.py内容如下(取出不同的用户名和密码)：

file = open(‘user.txt’,’r’)

lines = file.readlines() #lines输入结果：[‘user1,123\n’, ‘user2,123\n’, ‘user3,123’]

file.close()

for line in lines:

username = line.split(‘,’)[0] #用到了数组切片的思想

password = line.split(‘,’)[1]

print username,password

1. **读取csv文件**

文件内容：

user1,123

user2,123

user3,123

创建csv文件的方法：excel表输入内容后直接另存为csv文件就可以了，当参数很多的时候一般选用这个方法，txt就显得不方便了。使用该方法前先导入csv包，脚本如下：

import csv

data = csv.reader(open(‘user.csv’,’r’)) #读取本地csv文件，通过reader()方法读取csv文件

#循环输出每一行信息

for user in data:

print user

输出结果： **[‘user1’,’123’]**

**[‘user2’,’123’]**

**[‘user3’,’123’]**

**例1**

**取出所有行的值放入一个list中的方法，完整代码如下：**

import csv

data = csv.reader(open(‘user.csv’,’r’))

a = []

for i in data:

a.append(i) #把每一行的值存入数组a中

print a

输出结果为**[[‘user1’,’123’], [‘user2’,’123’], [‘user3’,’123’]]**

**例2**

**取出每一列的值放入list**

import csv

data = csv.reader(open(‘user.csv’,’r’))

a = []

for i in data:

a.append(i[0]) #把第一列的值存入数组a中

print a

输出结果为**[‘user1’, ‘user2’, ‘user3’]**

1. **读取excel文件**

需要安装2个excel插件包

读取包：<http://pypi.python.org/pypi/xlrd> 路径下载

写入包：<http://pypi.python.org/pypi/xlwt> 路径下载，这个包只能是新增表写数据

也可以直接使用 pip install xlrd 安装

例1

注意：只能操作xls文件

**import** xlrd #导入执行excel的模块  
**class** urld:  
 **def** d\_xlrd(self):  
 book = xlrd.open\_workbook(**r'd:/test.xls'**)#打开excel的方法  
 sheet = book.sheets()[0] *#打开第一张表* li = sheet.col\_values(2)[1:]#取excel表中列的下标第2的值，即第3列，1是指行，从下标0开始，所以这里是第2行

#sheet.row\_values(i)  获得行的值  
 **print** li

**print** sheet.nrows #获得行数  
**print** sheet.ncols #获得列数

**return** li #返回的li是一个数组/列表，其它脚本调用取值  
a = urld()  
a.d\_xlrd()

例2

**import** xlrd  
book = xlrd.open\_workbook(**r'd:/test.xlsx'**)

worksheets = workbook.sheet\_names()#获取文件中所有sheet列sheet = book.sheet\_by\_name(**u'Sheet2'**) #获取工作表sheet名称为sheet2的表  
li = sheet.row\_values(2)[0:] *#取出行的值*num\_cols = sheet.ncols  
num\_cols2 = sheet.nrows  
**print** num\_cols  
**for** i **in** range(num\_cols2):  
 a = sheet.row\_values(i)  
 **print**(a)

1. **新建工作表写入**

**import** xlwt  
**class** urld:  
 **def** d\_xlwt(self):  
 workbook = xlwt.Workbook(encoding = **'ascii'**)  
 worksheet = workbook.add\_sheet(**'myexcel'**) # 新建表sheet名称  
 worksheet.write(1, 5, label = **'Row 0, Column 0 Value'**)#行、列、值  
 workbook.save(**'Excel\_Workbook1.xls'**) #保存为  
a = urld()  
a.d\_xlwt()

1. **追加工作表写入**

用复制一张表的思想，需要用到xlutils类，用pit install xlutils 命令安装

**import** xlrd  
**from** xlutils.copy **import** copy  
**class** urld:  
 **def** d\_xlrd(self):  
 book = xlrd.open\_workbook(**r'd:/test.xlsx',** formatting\_info=True)

#formatting\_info=True：保留Excel的原格式  
 cp1 = copy(book) #复制工作表  
 shtc = cp1.get\_sheet(0) # 获得工作表对象  
 shtc.write(1,5,**'pass'**) #行、列、值  
 cp1.save(**r'd:/copytest1.xls'**) #保存为  
a = urld()  
a.d\_xlrd()

网上实例：

!/usr/bin/env python

# -\*- coding: utf-8 -\*-

# 读取excel数据

# 小罗的需求，取第二行以下的数据，然后取每行前13列的数据

import xlrd

data = xlrd.open\_workbook('test.xls') # 打开xls文件

table = data.sheets()[0] # 打开第一张表

nrows = table.nrows # 获取表的行数

for i in range(nrows): # 循环逐行打印

if i == 0: # 跳过第一行

continue

print table.row\_values(i)[:13] # 取前十三列

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

excel的写操作等后面用到的时候在做记录

* **方法二**   
  使用xlrd读取文件，使用xlwt生成Excel文件（可以控制Excel中单元格的格式）。但是用xlrd读取excel是不能对其进行操作的；而xlwt生成excel文件是不能在已有的excel文件基础上进行修改的，如需要修改文件就要使用xluntils模块。pyExcelerator模块与xlwt类似，也可以用来生成excel文件。

**1.[代码]test\_xlrd.py**

#coding=utf-8

#######################################################

#filename:test\_xlrd.py

#author:defias

#date:xxxx-xx-xx

#function：读excel文件中的数据

#######################################################

import xlrd

#打开一个workbook

workbook = xlrd.open\_workbook('E:\\Code\\Python\\testdata.xls')

#抓取所有sheet页的名称

worksheets = workbook.sheet\_names()

print('worksheets is %s' %worksheets)

#定位到sheet1

worksheet1 = workbook.sheet\_by\_name(u'Sheet1')

"""

#通过索引顺序获取

worksheet1 = workbook.sheets()[0]

#或

worksheet1 = workbook.sheet\_by\_index(0)

"""

"""

#遍历所有sheet对象

for worksheet\_name in worksheets:

worksheet = workbook.sheet\_by\_name(worksheet\_name)

"""

#遍历sheet1中所有行row

num\_rows = worksheet1.nrows

for curr\_row in range(num\_rows):

row = worksheet1.row\_values(curr\_row)

print('row%s is %s' %(curr\_row,row))

#遍历sheet1中所有列col

num\_cols = worksheet1.ncols

for curr\_col in range(num\_cols):

col = worksheet1.col\_values(curr\_col)

print('col%s is %s' %(curr\_col,col))

#遍历sheet1中所有单元格cell

for rown in range(num\_rows):

for coln in range(num\_cols):

cell = worksheet1.cell\_value(rown,coln)

print cell

"""

#其他写法：

cell = worksheet1.cell(rown,coln).value

print cell

#或

cell = worksheet1.row(rown)[coln].value

print cell

#或

cell = worksheet1.col(coln)[rown].value

print cell

#获取单元格中值的类型，类型 0 empty,1 string, 2 number, 3 date, 4 boolean, 5 error

cell\_type = worksheet1.cell\_type(rown,coln)

print cell\_type

"""

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55

**2. [代码]test\_xlwt.py**

#coding=utf-8

#######################################################

#filename:test\_xlwt.py

#author:defias

#date:xxxx-xx-xx

#function：新建excel文件并写入数据

#######################################################

import xlwt

#创建workbook和sheet对象

workbook = xlwt.Workbook() #注意Workbook的开头W要大写

sheet1 = workbook.add\_sheet('sheet1',cell\_overwrite\_ok=True)

sheet2 = workbook.add\_sheet('sheet2',cell\_overwrite\_ok=True)

#向sheet页中写入数据

sheet1.write(0,0,'this should overwrite1')

sheet1.write(0,1,'aaaaaaaaaaaa')

sheet2.write(0,0,'this should overwrite2')

sheet2.write(1,2,'bbbbbbbbbbbbb')

"""

#-----------使用样式-----------------------------------

#初始化样式

style = xlwt.XFStyle()

#为样式创建字体

font = xlwt.Font()

font.name = 'Times New Roman'

font.bold = True

#设置样式的字体

style.font = font

#使用样式

sheet.write(0,1,'some bold Times text',style)

"""

#保存该excel文件,有同名文件时直接覆盖

workbook.save('E:\\Code\\Python\\test2.xls')

print '创建excel文件完成！'

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34

**3. [代码]test\_xlutils.py**

#coding=utf-8

#######################################################

#filename:test\_xlutils.py

#author:defias

#date:xxxx-xx-xx

#function：向excel文件中写入数据

#######################################################

import xlrd

import xlutils.copy

#打开一个workbook

rb = xlrd.open\_workbook('E:\\Code\\Python\\test1.xls')

wb = xlutils.copy.copy(rb)

#获取sheet对象，通过sheet\_by\_index()获取的sheet对象没有write()方法

ws = wb.get\_sheet(0)

#写入数据

ws.write(1, 1, 'changed!')

#添加sheet页

wb.add\_sheet('sheetnnn2',cell\_overwrite\_ok=True)

#利用保存时同名覆盖达到修改excel文件的目的,注意未被修改的内容保持不变

wb.save('E:\\Code\\Python\\test1.xls')

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

**4. [代码]test\_pyExcelerator\_read.py**

#coding=utf-8

#######################################################

#filename:test\_pyExcelerator\_read.py

#author:defias

#date:xxxx-xx-xx

#function：读excel文件中的数据

#######################################################

import pyExcelerator

#parse\_xls返回一个列表，每项都是一个sheet页的数据。

#每项是一个二元组(表名,单元格数据)。其中单元格数据为一个字典，键值就是单元格的索引(i,j)。如果某个单元格无数据，那么就不存在这个值

sheets = pyExcelerator.parse\_xls('E:\\Code\\Python\\testdata.xls')

print sheets

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

**5. [代码]test\_pyExcelerator.py**

#coding=utf-8

#######################################################

#filename:test\_pyExcelerator.py

#author:defias

#date:xxxx-xx-xx

#function：新建excel文件并写入数据

#######################################################

import pyExcelerator

#创建workbook和sheet对象

wb = pyExcelerator.Workbook()

ws = wb.add\_sheet(u'第一页')

#设置样式

myfont = pyExcelerator.Font()

myfont.name = u'Times New Roman'

myfont.bold = True

mystyle = pyExcelerator.XFStyle()

mystyle.font = myfont

#写入数据，使用样式

ws.write(0,0,u'ni hao 帕索！',mystyle)

#保存该excel文件,有同名文件时直接覆盖

wb.save('E:\\Code\\Python\\mini.xls')

print '创建excel文件完成！'

## 自动化测试模型

1. **线性测试**：每个测试脚本独立，开发成本高，每个脚本中存在大量相同的操作，所以现在一般不会使用，当然可以做流程测试
2. **模块驱动测试**：由于线性测试的缺点，所以有了这个测试，将脚本模块化，比如登录和退出脚本就是每个脚本都要用的，我们把登录和退出封装成一个脚本类，提供给后期脚本调用，这样只要修改一个脚本就所有脚本都行了

优点：提高了开发效率，不用重复写相同代码的操作

简化了脚本的维护复杂性

1. **数据驱动测试：**在模块驱动测试基础上将输入数据参数化

## 类的调用

被调用脚本a1的代码如下：

*#coding=utf-8***class** lei():  
 **def** hs1(self,a,b):  
 **print** a + b  
 **def** hs2(self,a,b):  
 **print** a - b  
**class** l1():  
 **def** hs3(self,a,b):  
 **print** a\*b

演示一下四种调用类的方法：

#coding=utf-8  
# 第一种：直接导入文件调用，该方法调用类中的函数时需要指定文件名  
# import a1  
# class jca1():  
# def hs(self):  
# print '111'  
# a1.lei().hs2(6,3)  
# def hsq(self):  
# print '111'  
# a1.lei().hs1(6,3)  
# a1.l1().hs3(6,3)  
# jca1().hs()  
# jca1().hsq()  
# 第二种：导入文件类，该方法调用类中函数时，直接：类().函数名()  
# from a1 import lei,l1  
# class jca1():  
# def hs(self):  
# print '111'  
# lei().hs2(6,3)  
# def hsq(self):  
# print '111'  
# lei().hs1(6,3)  
# l1().hs3(6,3)  
# jca1().hs()  
# jca1().hsq()  
# 第三种：from 文件名 import \*，导入该文件中的所有类，该方法调用类中函数时，直接：类().函数名()，同2一样  
# from a1 import \*  
# class jca1():  
# def hs(self):  
# print '111'  
# lei().hs2(6,3)  
# def hsq(self):  
# print '111'  
# lei().hs1(6,3)  
# l1().hs3(6,3)  
# jca1().hs()  
# jca1().hsq()  
# 第四种：from 文件名 import \*，导入该文件中的所有类，在本脚本中再继承文件中的类，继承后，该脚本的类也就继承了导入文件的类，如果被调用脚本有初始化函数，那么在执行本脚本实例化时必须传入参数，参见 [继承](#_继承)  
# 文件类中的所有函数，可以直接 "脚本类名().函数名()"  
from a1 import \*  
class jca1(lei,l1):  
 def hs(self):  
 print '111'  
 jca1().hs2(6,3)  
 def hsq(self):  
 print '111'  
 jca1().hs1(6,3)  
 jca1().hs3(6,3)  
jca1().hs()  
jca1().hsq()

## 测试集合

有2个脚本[csunittest，csunittest3](#_单个脚本生成报告)（脚本内容参见单个脚本生成报告），下面写一个测试集合脚本

*#coding=utf-8  
# 测试集合跑多个脚本，有以下二种方法  
# 方法一，构造测试集  
# import csunittest  
# import csunittest3*

*# import unittest  
#上面导入了2个测试脚本  
# # 构造测试集合  
# suite = unittest.TestSuite()  
# 指定测试用例  
# suite.addTest(csunittest.baidushouye("test\_sousuo\_case1"))  
# suite.addTest(csunittest.baidushouye("test\_xinwen\_case2"))  
#  
# suite.addTest(csunittest3.baidushouye("test\_xueshu\_case1"))  
# suite.addTest(csunittest3.baidushouye("test\_tieba\_case2"))  
#  
# if \_\_name\_\_ == "\_\_main\_\_":  
# runner = unittest.TextTestRunner()  
# runner.run(suite)  
  
# 方法二：模糊匹配所有测试文件，执行文件里的所有函数，这个方法很方便，不用添加测试用例  
# 用到函数discover(),里面有3个参数，第一个参数为脚本路径，第二个参数为匹配文件名，第三个参数可以不填写*test\_dir = **'./' # 定义测试脚本目录为当前目录，如果定义下级目录，这样定义’./test1/’**

**#上层目录用 ‘../’ 2个小点表示**discover = unittest.defaultTestLoader.discover(test\_dir,pattern = **'csunittest\*.py'**)  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 runner = unittest.TextTestRunner()   
 runner.run(discover)

## HTML测试报告

下载 HTMLTestRunner.py文件，放在python安装目录里里的lib目录下，如：C:\Python37-32\Lib

注意：生成测试报告的脚本最好不要有print输出语句，不然会报错（get ‘str’错误）

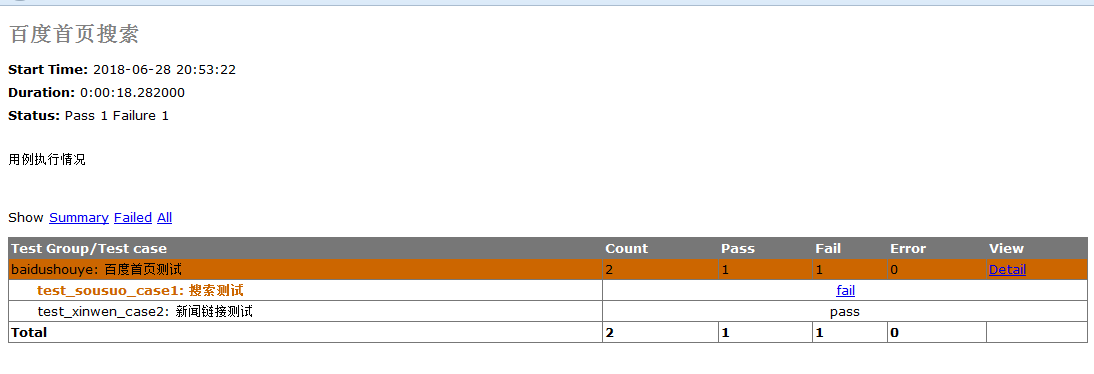
以脚本例子做详细讲解：

### 单个脚本生成报告

脚本一csunittest1：

*#coding=utf-8***from** selenium **import** webdriver  
**import** unittest  
**import** time #导入时间模块，用于脚本等待  
**from** HTMLTestRunner **import** HTMLTestRunner #导入HTMLTestRunner报告模块  
**import** sys  
reload(sys)  
sys.setdefaultencoding(**'utf-8'**)  
**class** baidushouye(unittest.TestCase):  
 *''' 百度首页测试 '''* **def** setUp(self):  
 self.wd = webdriver.Firefox()  
 self.url = **'http://www.baidu.com'  
 def** test\_sousuo\_case1(self):  
 *''' 搜索测试 '''* wd = self.wd  
 wd.get(self.url)  
 wd.find\_element\_by\_id(**'kw'**).send\_keys(**'selenium'**)  
 wd.find\_element\_by\_id(**'su'**).click()  
 time.sleep(3)  
 a = wd.find\_element\_by\_class\_name(**'nums\_text'**).text  
 *# print a  
 # print type(a)  
 # 断言页面出现的内容* self.assertIn(**u'百度为您找到相关结果'**,a,msg = **'搜索期结果错误'**)  
 *# 断言元素是否存在* self.assertIsNotNone(wd.find\_element\_by\_class\_name(**'nums\_text'**),msg = **'元素不存在'**)  
 **def** test\_xinwen\_case2(self):  
 *''' 新闻链接测试 '''* wd = self.wd  
 wd.get(self.url)  
 wd.find\_element\_by\_link\_text(**'新闻'**).click()  
 time.sleep(2)  
 a = wd.find\_element\_by\_class\_name(**'checked'**).text  
 *# print a  
 # print type(a)* self.assertEqual(a,**u'新闻全文'**,msg = **'新闻超链接失败'**)  
  
 **def** tearDown(self):  
 *#print 'end'* self.wd.quit()  
*# if \_\_name\_\_ == '\_\_main\_-':  
# unittest.main()***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 testunit = unittest.TestSuite()   
 testunit.addTest(baidushouye(**'test\_sousuo\_case1'**))  
 testunit.addTest(baidushouye(**'test\_xinwen\_case2'**))  
 now = time.strftime(**'%Y-%m-%d %H\_%M\_%S'**) #定义一个当前时间格式化字符串  
 filename = **'./'**+ now + **'result.html'** fp = open(filename,**'wb'**) #open()方法以二进制的形式打开**result.html文件**  
 runner = HTMLTestRunner(stream=fp, #指定测试报告文件  
 title=**'百度首页搜索'**, #指定报告标题  
 description=**'用例执行情况'**)#测试报告副标题  
 runner.run(testunit)

生成后的报告如下图：



脚本二csunittest3：

*#coding=utf-8***from** selenium **import** webdriver  
**import** unittest  
**import** time  
**from** HTMLTestRunner **import** HTMLTestRunner  
**import** sys  
reload(sys)  
sys.setdefaultencoding(**'utf-8'**)  
**class** baidushouye2(unittest.TestCase):  
 *''' 百度首页测试 '''* **def** setUp(self):  
 self.wd = webdriver.Firefox()  
 self.url = **'http://www.baidu.com'  
 def** test\_xueshu\_case1(self):  
 *''' 学术链接测试 '''* wd = self.wd  
 wd.get(self.url)  
 wd.find\_element\_by\_link\_text(**'学术'**).click()  
 time.sleep(2)  
 a = wd.find\_element\_by\_link\_text(**'注册'**).text  
 *# print a  
 # print type(a)* self.assertEqual(a,**u'注册1'**,msg = **'注册超链接失败'**)  
  
 **def** test\_tieba\_case2(self):  
 *''' 贴吧链接测试 '''* wd = self.wd  
 wd.get(self.url)  
 wd.find\_element\_by\_link\_text(**'贴吧'**).click()  
 time.sleep(2)  
 a = wd.find\_element\_by\_link\_text(**'进入贴吧'**).text  
 *# print a  
 # print type(a)* self.assertEqual(a,**u'进入贴吧'**,msg = **'贴吧超链接失败'**)  
 **def** tearDown(self):  
  
 self.wd.quit()  
*# if \_\_name\_\_ == '\_\_main\_-':  
# unittest.main()***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 testunit = unittest.TestSuite()  
 testunit.addTest(baidushouye2(**'test\_xueshu\_case1'**))  
 testunit.addTest(baidushouye2(**'test\_tieba\_case2'**))  
 now = time.strftime(**'%Y-%m-%d %H\_%M\_%S'**)  
 filename = **'./'**+ now + **'result.html'** fp = open(filename,**'wb'**)  
 runner = HTMLTestRunner(stream=fp,  
 title=**'百度首页搜索'**,  
 description=**'用例执行情况'**)  
 runner.run(testunit)

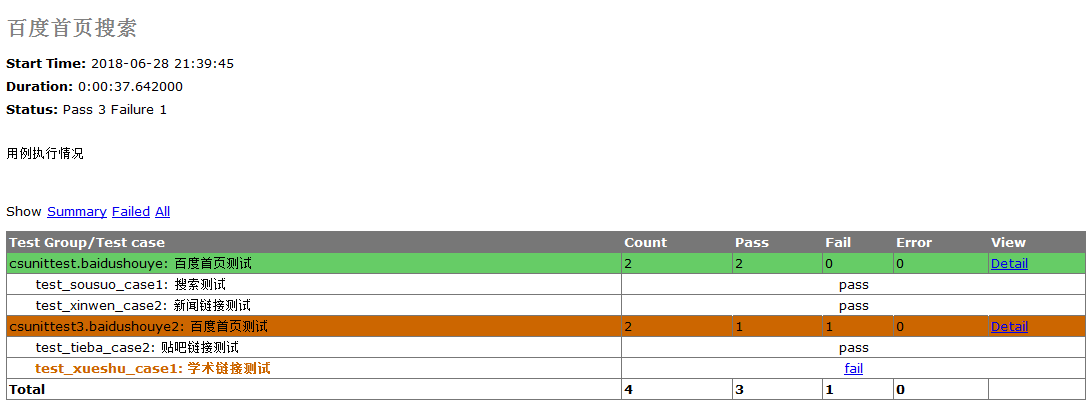
### 项目集成报告（多个脚本）

由于实际项目中运行多个脚本后是需要生成一个完整的测试报告方便查看，用到的方法是[测试集合](#_测试集合)的思想，下面代码就不做一一讲解

根据上面2个脚本写一个运行脚本 如下：

*#coding=utf-8***from** selenium **import** webdriver  
**import** unittest  
**import** time  
**from** HTMLTestRunner **import** HTMLTestRunner  
**import** sys  
reload(sys)  
sys.setdefaultencoding(**'utf-8'**)  
test\_dir = **'./HTMLBG' #**discover = unittest.defaultTestLoader.discover(test\_dir,pattern= **'csunittest\*.py'**)  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 now = time.strftime(**'%Y-%m-%d %H\_%M\_%S'**)  
 filename = **'./testBG/'**+ now + **'result.html'** fp = open(filename,**'wb'**)  
 runner = HTMLTestRunner(stream=fp,  
 title=**'百度首页搜索'**,  
 description=**'用例执行情况'**)  
 runner.run(discover)

生成报告如下：



### HTMLTestRunner.py报告文件美化修改

修改图中函数如下

def \_generate\_report\_test(self, rows, cid, tid, n, t, o, e): #lb修改函数  
 # e.g. 'pt1.1', 'ft1.1', etc  
 has\_output = bool(o or e)  
 tid = (n == 0 and 'p' or 'f') + 't%s.%s' % (cid+1,tid+1)  
 name = t.id().split('.')[-1]  
 doc = t.shortDescription() or ""  
 desc = doc and ('%s: %s' % (name, doc)) or name  
 tmpl = has\_output and self.REPORT\_TEST\_WITH\_OUTPUT\_TMPL or self.REPORT\_TEST\_NO\_OUTPUT\_TMPL  
 uo = o #输出脚本里的print  
 script = self.REPORT\_TEST\_OUTPUT\_TMPL % dict(  
 id = tid,  
 output = saxutils.escape(str(uo)+e),  
 )  
  
 row = tmpl % dict(  
 tid = tid,  
 Class = (n == 0 and 'hiddenRow' or 'none'),  
 style = n == 2 and 'errorCase' or (n == 1 and 'failCase' or 'none'),  
 desc = desc,  
 script = script,  
 status = self.STATUS[n],  
 )  
 rows.append(row)  
 if not has\_output:  
 return

## 多线程

多线程可以并发执行一件事情，或多件事情，具体看下面代码实例

*#coding=utf-8***import** threading *#导入线程模组***import** time  
**def** longin(t1,t2): *#定义线程执行的函数* **print "longin run time:%s"** % time.ctime()  
 *#print t1 \* t2***def** longin2(): *#定义线程执行的函数* **print "longin2 run time:%s"** % time.ctime()  
 *#print t1 + t2*threads = [] *#定义一个线程数组  
# t = threading.Thread(target=longin)  
# threads.append(t)  
# t1 = threading.Thread(target=longin)  
# threads.append(t1)  
# t2 = threading.Thread(target=longin)  
# threads.append(t2)  
# t3 = threading.Thread(target=longin)  
# threads.append(t3)  
# t4 = threading.Thread(target=longin)  
# threads.append(t4)  
# t5 = threading.Thread(target=longin)  
#threads.append(t5)***'''上面注释的代码是一种创建线程的方式，分别添加,线程过多时显得不得力'''  
for** x **in** range(100): *#利用for循环创建多个线程* a = threading.Thread(target=longin,args=(5,6)) *#创建线程执行函数longin，如果函数没有传参，不需要args* b = threading.Thread(target=longin2) *#创建线程执行函数longin2* **print** a  
 **print** b  
 threads.append(a) *#添加到线程数组* threads.append(b) *#添加到线程数组***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **for** i **in** threads: *#遍历线程数组中的所有线程* i.start() *#启动线程* **for** i **in** threads:  
 i.join() *#等待线程终止，必须使用* **print 'shuchu'**

例2 利用多线程跑系统稳定性测试连续运行时间

*#coding=utf-8***import** threading *#导入线程模组***import** time  
**from** selenium **import** webdriver  
**def** longin(t1,t2): *#定义线程执行的函数  
 ''' 利用while循环+sleep方法执行脚本的运行时间'''* i=0  
 **while** time.ctime() < **'Thu Jul 5 18:06:00 2018'**: *#利用返回时间字符串作对比，控制循环时间* **try**: #加入异常处理是防止页面出错的情况下能继续循环运行  
 **print** time.ctime()  
 i = i+1  
 wd = webdriver.Firefox()  
 wd.get(**'http://192.168.23.129:8000/agileone/index.php'**)  
 wd.find\_element\_by\_id(**'username'**).send\_keys(**'admin'**)  
 wd.find\_element\_by\_id(**'password'**).send\_keys(**'admin'**)  
 wd.find\_element\_by\_id(**'login'**).click()  
 time.sleep(10) *# 等待10秒* **except**:  
 **print u"服务未连接"  
 finally**:  
 wd.quit()  
 **print** i *#输出循环了多少次  
 #print t1 \* t2*threads = [] *#定义一个线程数组***for** x **in** range(2): *#利用for循环创建多个线程* a = threading.Thread(target=longin,args=(5,6)) *#创建线程执行函数longin，如果函数没有传参，不需要args* **print** a  
 threads.append(a) *#添加到线程数组***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **for** i **in** threads: *#遍历线程数组中的所有线程* i.start() *#启动线程,线程时并发启动的* **for** i **in** threads:  
 i.join() *#等待线程终止，必须使用* **print 'shuchu'**

**列3 多线程执行不同的函数，其它函数用到另一个线程的值**

import time  
import random  
# c =100  
def hh():  
 ran =random.randint(1,99999999)  
 a = open('1.txt','w')  
 a.write(str(ran))  
def bb():  
 while 1<2:  
 time.sleep(2)  
 b = open('1.txt')  
 c = b.read()  
 print(c)  
def dd():  
 while 1< 2:  
 time.sleep(10)  
 hh()  
import threading  
  
threads = []  
t = threading.Thread(target=hh)  
t1 = threading.Thread(target=bb)  
t2 = threading.Thread(target=dd)  
threads.append(t)  
threads.append(t1)  
threads.append(t2)  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in threads: #遍历线程数组中的所有线程  
 time.sleep(0.5)  
 i.start() #启动线程  
 for i in threads:  
 i.join() #等待线程终止，必须使用

## 异常处理

异常处理是处理在脚本发生错误或者某一元素不能找到时的一种处理方式，其目的是让脚本在发生错误后能够正常的运行，如果不作异常处理，脚本异常后就会终止脚本运行

try：

….. #有可能发生异常的脚本放这里

except:

……#发生异常后，进入except分支

finally:

……#异不异常都要走的分支

# requests接口自动化

web常用接口get、post，传参数一般是字典形式和json格式，python测试接口时，如果参数是json格式，那么需要将字典数据序列化成json字符串，方法 json.dumps(data)

下载requests： pin install requests

## 例get请求：get请求可以有多个参数添加头信息用

**headers=头信息**

**get请求url传参格式：url后跟？参数名=参数值&参数名=参数值**

**例子：**

**https://www.baidu.com/?wd=abc&rsv=111**

脚本实例1：

import requests

import unittest

class qwjs(unittest.TestCase):

def setUp(self):

self.user = {'username':'admin','password':'123456'}

login = requests.post('http://localhost:10093/rest/v1/account/login',data=self.user)

self.a = login.json()

#print self.a['tokenId']

self.tokenid = self.a['tokenId']

def test\_case1\_dkjssj(self):

'''遍历数据库表不获取字段信息'''

url = 'http://127.0.0.1:10093/rest/v1/business/fullTextSearch/databaseErgodic'

pram = {'isGetColumn':'false','tokenid':self.tokenid}

qq = requests.get(url,params=pram)

text = qq.text

self.assertIn('Demo1',text)

self.assertIn('Demo2',text)

self.assertIn('Demo3',text)

self.assertIn('test1',text)

实例2

list = {**'src'**:**'pctop'**}  
a = requests.get(**"http://news.baidu.com/app//"**,params=list)  
**print** a.headers,type(a.headers) *#获得消息头，class类型***print**(a.status\_code),type(a.status\_code) *#获得状态码，int型***print**(a.text),type(a.text) *#获得响应文本，'unicode'型***print**(a.url),type(a.url) *#获得连接地址，'unicode'型*

**print(a.elapsed.microseconds/1000) #获得接口响应时间ms**

## post请求，post传参用data

list1 = {**'red\_tag'**:**'w2589011504'**}  
url = **"http://tieba.baidu.com/f/search/adv"**a1 = requests.post(url,data=list1)  
**print** a1.headers,type(a1.headers) *#获得消息头，class类型***print**(a1.status\_code),type(a1.status\_code) *#获得状态码，int型***print**(a1.text),type(a1.text) *#获得响应文本，'unicode'型***print**(a1.url),type(a1.url) *#获得连接地址，'unicode'型*

## 头信息添加实例

requests.get(url,params=data,headers=self.token,verify=False)

## 文件上传

1、参数类型为file的方法上传

file = {'file': open("F:\makeba/1.jpg", 'rb')} #定义文件格式方法，以二进制读取

imagFile = {'imagFile': ('imagFile',open("F:\makeba/1.jpg", 'rb'))}#可定义文件名  
url = '/cloud/face/faceMatch'  
cs = {"did":"0101000006"}  
jg = requests.post(fwurl+url,data=cs,files = file)

2、参数类型为binary二进制方式上传

file = open("F:\makeba/1.jpg", 'rb')#直接读取文件二进制流

jg = requests.post(fwurl+url,data=file)#直接以二进制提交参数

## json序列化

需要导入包 json

a = {**'name'**:**'lb'**,**'sex'**:**'man'**} #dict（字典）类型  
**print** a,type(a)

1. Json格式传参数，2种方法
2. a1 = json.dumps(a) #转化为json格式,字符串形式  
   **print** a1,type(a1)

注意：序列化json格式传参时，必须定义头信息，如下

headers = {'Content-Type': 'application/json'}

1. requests.post(url,json=a)，这种方法直接将字典转化为json格式，不需要定义头信息
2. a2 = json.loads(a1) #反序列化，将json格式字符串转化为dict dict（字典）类型

**print** a2,type(a2)

脚本例子：

a1 = '{"name":"lb"}'  
a2 = '[{"name":"lb"}]'  
a3 = "[1,2]"  
print(json.loads(a1),type(json.loads(a1)))  
print(json.loads(a2),type(json.loads(a2)))  
print(json.loads(a3),type(json.loads(a3)))  
  
a4 = {"name":"lb"}  
a5 = [1,2]  
a6 = 123  
print(json.dumps(a4),type(json.dumps(a4)))  
print(json.dumps(a5),type(json.dumps(a5)))  
print(json.dumps(a6),type(json.dumps(a6)))

输出结果如下：

{'name': 'lb'} <class 'dict'>

[{'name': 'lb'}] <class 'list'>

[1, 2] <class 'list'>

{"name": "lb"} <class 'str'>

[1, 2] <class 'str'>

123 <class 'str'>

## 获取接口响应时间

result = requests.get(url)

gettime = result.elapsed.total\_seconds() #单位是秒

## 接口断言

获取返回数据，可以用下面2中方式断言

1. .text：获取返回数据的文本内容，用包含关系判断assertIn(a,b)，因为.text的返回数据类型是unicode型，所以在判断字符串时，前面需加上 u，例：u‘百度一下’
2. .json()或者json.loads()：将返回结果转换成dict字典型数据，这样就用等于assertEqual(a,b)

## 接口脚本实例

*#coding=utf-8***import** requests  
**import** unittest  
**import** time  
**from** HTMLTestRunner **import** HTMLTestRunner  
**import** sys  
reload(sys)  
sys.setdefaultencoding(**'utf-8'**)  
**class** qwjs(unittest.TestCase):  
 *'''全文检索接口测试'''* **def** setUp(self):  
 self.user = {**'username'**:**'admin'**,**'password'**:**'123456'**}  
 login = requests.post(**'http://localhost:10093/rest/v1/account/login'**,data=self.user)  
 self.a = login.json()  
 *#print self.a['tokenId']* self.tokenid = self.a[**'tokenId'**]  
 **def** test\_case1\_dkjssj(self):  
 *''' 单库检索数据 '''* jk = **'http://localhost:10093/rest/v1/business/fullTextSearch/searchData'** sj = {**'dataNames'**:**'Demo1'**,**'strWhere'**:**'自带火香烟'**,**'tokenid'**:self.tokenid}  
 dkjs = requests.post(jk,data=sj)  
 fhsj = dkjs.json()  
 *#print fhsj* self.assertEqual(fhsj[**'end'**],1)  
 **def** test\_case2\_dkjs(self):  
 *''' 多库检索数据 '''* jk = **'http://localhost:10093/rest/v1/business/fullTextSearch/searchDataAuthMulti'** sj = {**'dataNames'**:**'Demo1'**,**'strWhere'**:**'自带火香烟'**,**'tokenid'**:self.tokenid}  
 dkjs = requests.post(jk,data=sj)  
 fhsj = dkjs.json()  
 fh2 = fhsj[**'Demo1'**]  
 *#print type(fhsj['Demo1'])* self.assertEqual(fh2[**'end'**],1)  
 **def** tearDown(self):  
 a=**'end'  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 testunit = unittest.TestSuite()  
 testunit.addTest(qwjs(**'test\_case1\_dkjssj'**))  
 testunit.addTest(qwjs(**'test\_case2\_dkjs'**))  
 now = time.strftime(**'%Y-%m-%d %H\_%M\_%S'**)  
 filename = **'../TestBG/'**+ now + **'result.html'** fp = open(filename,**'wb'**)  
 runner = HTMLTestRunner(stream=fp,  
 title=**'全文检索接口测试'**,  
 description=**'用例执行情况'**)  
 runner.run(testunit)

## DDT数据驱动

概念：unittest框架里一个test用例，运用不同的测试数据来进行测试

运用ddt的步骤如下

1. 导入包：import ddt
2. [在测试类前加上@ddt.ddt](mailto:在测试类前加上@ddt.ddt)
3. [在测试用例前加上@ddt.data](mailto:在测试用例前加上@ddt.data)

### 具体网上实例如下：

ddt(data driven test)数据驱动测试  
由外部数据集合来驱动测试用例  
核心思想：数据和测试代码分离  
应用场景：一组数据来执行相同的操作  
安装cmd：pip install ddt  
  
**1、一组数据中，每个数据为单个值**用法：在测试类前标记ddt，在单个测试用例前标记data、unpack、file\_data

[复制代码](javascript:void(0);)

import ddt

import unittest

@ddt.ddt

class test\_ddt(unittest.TestCase):

#1）：一组数据中，每个数据为单个值

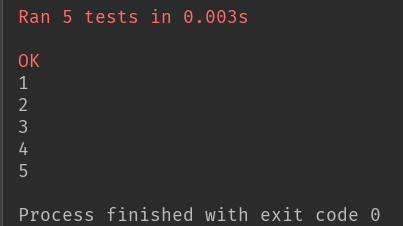
@ddt.data(1,2,3,4,5)

def test\_print(self,a):

print(a)

[复制代码](javascript:void(0);)

运行结果：



**2、对传入的数据分解用unpack，不加unpack说明是两组数据**

[复制代码](javascript:void(0);)

@ddt.ddt

class test\_ddt(unittest.TestCase):

@ddt.data([1,2,3],[10,11,12])

def test\_add(self,a):

print(a)  
　　　　 #print(a[0]+a[1]+a[2])

#运行结果

Ran 2 tests in 0.001s

OK

[1, 2, 3]

[10, 11, 12]  
  
ok  
6  
33

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

@ddt.ddt

class test\_ddt(unittest.TestCase):

@ddt.data([1,2,3],[10,11,12])

@ddt.unpack #对[1,2]分解数据,有多少个值传对应参数，data数据的值数据要对应

def test\_add(self,a,b,c):

print(a+b+c)  
  
#运行结果OK

6  
33

[复制代码](javascript:void(0);)

**3、从变量(一组数据)当中取值，(表达式：\*变量名)**

[复制代码](javascript:void(0);)

listA = [1,2,3,4,5]

@ddt.ddt

class test\_ddt(unittest.TestCase):

# 2）：一组数据中，每个数据为一个列表或一个字典

# 从变量(一组数据)当中取值--变量是字典/列表(表达式：\*变量名)

@ddt.data(\*listA)

def test\_a(self,a):

print(a)

#运行结果  
OK  
1

2

3

4

5

[复制代码](javascript:void(0);)

**4、从文件读取数据file\_data**

[复制代码](javascript:void(0);)

#前提：Mydict.json数据为：{"name":"jiao","sex":"female","age":"18"}

@ddt.ddt

class test\_ddt(unittest.TestCase):

# 3）：文件对象：json、yaml

# 变量是字典：键将用作测试用例名称的后缀，而值将作为测试数据提供。

# 列表中的每个值将对应于一个测试用例，并且该值将连接到测试方法名称。

@ddt.file\_data("D:\\PyCharm\\project\\python\\Mydict.json")

def test\_readData\_fromJsonFile(self,a):

print("从文件读取")

print(a)

#运行结果

从文件读取

jiao

从文件读取

female

从文件读取

18

## DDT用例名修改

1. 如果传入ddt的值是数组，可用如下方法

修改ddt.py文件里的函数mk\_test\_name

把value = str(value)修改成如下代码

value = value[1] +'：'+ value[4]

这里的value为读取excel函数后，用例里每一行的值

1. 如果传入ddt的值是字典，可用如下方法（兼容数组的情况）

def mk\_test\_name(name,value, index=0):  
 *"""  
 Generate a new name for a test case.  
  
 It will take the original test name and append an ordinal index and a  
 string representation of the value, and convert the result into a valid  
 python identifier by replacing extraneous characters with ``\_``.  
  
 We avoid doing str(value) if dealing with non-trivial values.  
 The problem is possible different names with different runs, e.g.  
 different order of dictionary keys (see PYTHONHASHSEED) or dealing  
 with mock objects.  
 Trivial scalar values are passed as is.  
  
 A "trivial" value is a plain scalar, or a tuple or list consisting  
 only of trivial values.  
 """* # Add zeros before index to keep order  
 index = "{0:0{1}}".format(index + 1, index\_len)  
 #注释原方法  
 # if not is\_trivial(value):  
 # return "{0}\_{1}".format(name, index)  
  
 if not is\_trivial(value) and type(value) is not dict:  
 return "{0}\_{1}".format(name, index)  
 # 如果数据是字典，则获取字典当中的api\_name对应的值，加到测试用例名称中。  
 if type(value) is dict:  
 try:  
 value = value['用例编号'] + '：' + value['接口名称'] +'：'+ value['用例描述'] # case\_name作为value值  
 except:  
 return "{0}\_{1}".format(name, index)  
 else:  
 try:  
 value = value[0] + '：' + value[1] +'：'+ value[4] # case\_name作为value值  
 except:  
 return "{0}\_{1}".format(name, index)  
 try:  
 value = str(value)  
 # value = value[0] + '：' + value[1] +'：'+ value[4]  
 # value = value['用例编号'] + '：' + value['接口名称'] +'：'+ value['用例描述']  
 except UnicodeEncodeError:  
 # fallback for python2  
 value = value.encode('ascii', 'backslashreplace')  
 test\_name = "{0}\_{1}\_{2}".format(name, index,value)  
 return re.sub(r'\W|^(?=\d)', '\_', test\_name)

## 接口自动化封装实例（输出excel测试报告）

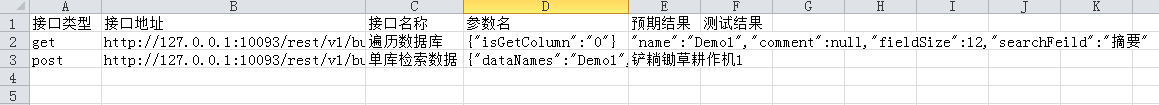
### 接口demo封装

import requests  
# from config.readconfig import \* #使用配置文件，已废弃  
from config.configReadExcel import \*  
import time  
class apiPack():  
 def \_\_init\_\_(self,file=None):  
 # self.jkfwurl = api\_service\_url() #使用配置文件，已废弃  
 self.jkfwurl = getApiService(1)  
 self.file = file  
  
 def sendGet(self,url,cookies,data=None):  
 *'''普通字典参数上传get请求'''* print(time.strftime('%Y-%m-%d %H:%M:%S'))  
 get\_jg = requests.get(url,params=data,headers= cookies,verify=False)  
 print(time.strftime('%Y-%m-%d %H:%M:%S'))  
 return get\_jg  
  
 def sendPost(self,url,cookies,data=None):  
 *'''普通字典参数上传post请求'''* print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 post\_jg = requests.post(url,data=data,headers=cookies,verify=False)  
 print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 return post\_jg  
  
 def sendPostJson(self,url,cookies,data=None):  
 *'''post json传参'''* print('接口执行开始时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 json\_jg = requests.post(url,json=data,headers=cookies,verify=False)  
 print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 return json\_jg  
  
 def sendPostFile(self,url,cookies,data=None):  
 *'''post 文件上传'''* jg = requests.post(url,files=self.file, headers=cookies, data=data)  
 return jg  
  
 def runMethod(self,jklx,url,cookies,data=None):  
 *'''自动化和手工第一次测试接口调用函数'''* if jklx == 'get':  
 res = self.sendGet(self.jkfwurl+url,cookies,data)  
 elif jklx == 'post':  
 res = self.sendPost(self.jkfwurl+url,cookies,data)  
 elif jklx == 'json\_post':  
 res = self.sendPostJson(self.jkfwurl+url,cookies,data)  
 else:  
 res = self.sendPostFile(self.jkfwurl+url,cookies,data)  
 return res  
  
 def runDebugMethod(self,jklx,url,cookies,data=None):  
 *'''接口调试、联调调用函数'''* urls = getUrlAndKey()  
 jkfwurl = urls[0]  
 if jklx == 'get':  
 res = self.sendGet(jkfwurl+url,cookies,data)  
 elif jklx == 'post':  
 res = self.sendPost(jkfwurl+url,cookies,data)  
 elif jklx == 'json\_post':  
 res = self.sendPostJson(jkfwurl+url,cookies,data)  
 else :  
 res = self.sendPostFile(jkfwurl+url,cookies,data)  
 return res

#### 使用excle做接口自动化（使用用例表输出测试报告）

##### 先获取xlsx表的数据

*有数据***jk\_data2.xlsx**

**import** xlrd  
**import** json  
**import** xlutils  
**class** read\_excle\_data():  
 **def** \_\_init\_\_(self):  
 book = xlrd.open\_workbook(**'d:/jk\_data2.xlsx'**,**'r'**)  
 self.sheet = book.sheets()[0]  
 self.row = self.sheet.nrows  
 **def** rown(self):  
 **return** self.row #返回数据行数  
 **def** get\_line(self,bs):  
 **return** self.sheet.row\_values(bs) #获取某一行的值  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 a = read\_excle\_data().get\_line(1)  
 b = json.loads(a[3])  
 c = read\_excle\_data().rown()  
 print (b,type(b),c,type(c))

##### 运行excle脚本

*#coding=utf-8***from** ssyq\_jk\_zdh.jk\_demo.jk\_demo **import** JK\_FZ  
**from** ssyq\_jk\_zdh.jk\_data.read\_xlsx2 **import** read\_excle\_data  
**from** ssyq\_jk\_zdh.gghs.tokenid **import** tokenid  
**import** xlrd  
**import** json  
**from** xlutils.copy **import** copy #复制工作表写入  
**def** run():  
 token = tokenid().token()  
 rown = read\_excle\_data().rown()  
 book = xlrd.open\_workbook(**'d:/jk\_data.xlsx'**,**'r'**)  
 cp = copy(book) #复制表对象  
 shtc = cp.get\_sheet(0) #复制表sheet  
 **for** i **in** range(rown-1):  
 i = i+1  
 cs = read\_excle\_data().get\_line(i) #获取某一行的数据  
 csz = json.loads(cs[3]) #序列化字符串参数为字典类型  
 csz[**"tokenid"**]=token  
 jg = JK\_FZ(cs[1],cs[0],csz).run\_method() #传参调用接口封装脚本  
 **if** cs[4] **in** jg.text:  
 print (**'yes'**)  
 shtc.write(i,6,**'pass'**) #在表的第i行，6列，写上pass  
 cp.save(**r'E:\PycharmProjects\ssyq\_jk\_zdh\test\_bg/copytest2.xls'**)

# 保存为**copytest2.xls文件，注意：只能保存为xls文件**  
 **else**:  
 print (**'no'**)  
 shtc.write(i,6,**'faile'**)  
 cp.save(**r'E:\PycharmProjects\ssyq\_jk\_zdh\test\_bg/copytest2.xls'**)  
  
run()

#### 使用csv文件做接口自动化

##### 先获取csv表的数据

*#coding=utf-8***import** csv  
**class** read\_csvfile():  
 **def** \_\_init\_\_(self):  
 read = csv.reader(open(**'d:/jk\_data2.csv'**,**'r'**))  
 self.a = [] #创建一个空的list  
 self.j = 0  
 **for** i **in** read:  
 self.j = self.j+1  
 self.a.append(i) #利用循环把csv每一行的数据添加到数组a中

#存入的格式:[[…],[…],[…]]  
 **def** rown(self):  
 **return** self.j #获取行数  
 **def** get\_line(self,bs):  
 **return** self.a[bs] #获取数组a中某一个参数值  
*# print(read\_csvfile().get\_line(1))*print(read\_csvfile().rown())

##### 运行csv脚本

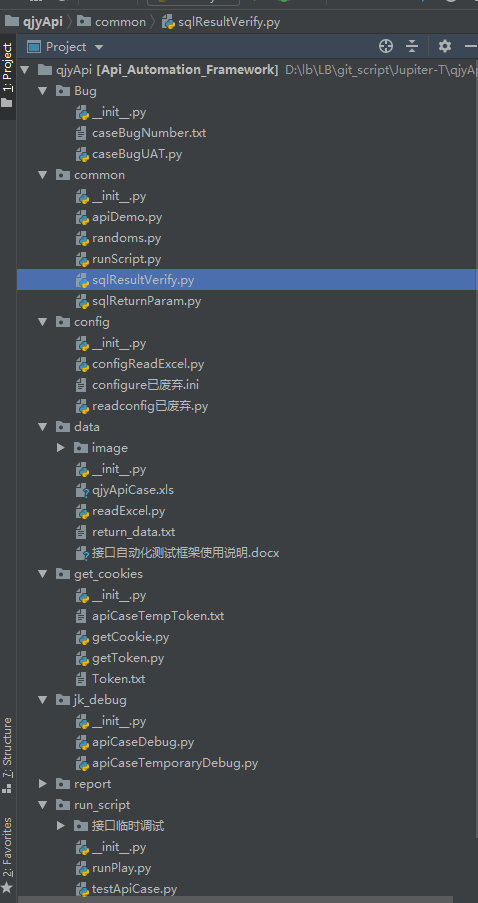
*#coding=utf-8***from** ssyq\_jk\_zdh.jk\_demo.jk\_demo **import** JK\_FZ  
**from** ssyq\_jk\_zdh.jk\_data.read\_csv **import** read\_csvfile  
**from** ssyq\_jk\_zdh.gghs.tokenid **import** tokenid  
**import** xlrd  
**import** json  
**from** xlutils.copy **import** copy  
**def** run():  
 token = tokenid().token()  
 rown = read\_csvfile().rown()  
 book = xlrd.open\_workbook(**'d:/jk\_data.xlsx'**,**'r'**)  
 cp = copy(book)  
 shtc = cp.get\_sheet(0)  
 **for** i **in** range(rown-1):  
 i = i+1  
 cs = read\_csvfile().get\_line(i)  
 csz = json.loads(cs[3])  
 csz[**"tokenid"**]=token  
 jg = JK\_FZ(cs[1],cs[0],csz).run\_method()  
 **if** cs[4] **in** jg.text:  
 print (**'yes'**)  
 shtc.write(i,6,**'pass'**)  
 cp.save(**r'E:\PycharmProjects\ssyq\_jk\_zdh\test\_bg/copytest3.xls'**)  
 **else**:  
 print (**'no'**)  
 shtc.write(i,6,**'faile'**)  
 cp.save(**r'E:\PycharmProjects\ssyq\_jk\_zdh\test\_bg/copytest3.xls'**)  
run()

## 最新接口自动化测试框架（20190417）

### Unittest+excel+html报告输出

框架结构同上，这种方式需要用ddt模块（数据驱动概念），run\_script文件下测试文件需要分解成2个py文件，一个是测试业务文件，一个是生成报告文件

#### 整体框架图



#### common公共模块封装

##### ApiDemo.py接口封装源码：

import requests  
# from config.readconfig import \* #使用配置文件，已废弃  
from config.configReadExcel import \*  
import time  
class apiPack():  
 def \_\_init\_\_(self,file=None):  
 # self.jkfwurl = api\_service\_url() #使用配置文件，已废弃  
 self.jkfwurl = getApiService(1)  
 self.file = file  
  
 def sendGet(self,url,cookies,data=None):  
 *'''普通字典参数上传get请求'''* print(time.strftime('%Y-%m-%d %H:%M:%S'))  
 get\_jg = requests.get(url,params=data,headers= cookies,verify=False)  
 print(time.strftime('%Y-%m-%d %H:%M:%S'))  
 return get\_jg  
  
 def sendPost(self,url,cookies,data=None):  
 *'''普通字典参数上传post请求'''* print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 post\_jg = requests.post(url,data=data,headers=cookies,verify=False)  
 print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 return post\_jg  
  
 def sendPostJson(self,url,cookies,data=None):  
 *'''post json传参'''* print('接口执行开始时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 json\_jg = requests.post(url,json=data,headers=cookies,verify=False)  
 print('接口执行结束时间：%s' % time.strftime('%Y-%m-%d %H:%M:%S'))  
 return json\_jg  
  
 def sendPostFile(self,url,cookies,data=None):  
 *'''post 文件上传'''* jg = requests.post(url,files=self.file, headers=cookies, data=data)  
 return jg  
  
 def runMethod(self,jklx,url,cookies,data=None):  
 *'''自动化和手工第一次测试接口调用函数'''* if jklx == 'get':  
 res = self.sendGet(self.jkfwurl+url,cookies,data)  
 elif jklx == 'post':  
 res = self.sendPost(self.jkfwurl+url,cookies,data)  
 elif jklx == 'json\_post':  
 res = self.sendPostJson(self.jkfwurl+url,cookies,data)  
 else:  
 res = self.sendPostFile(self.jkfwurl+url,cookies,data)  
 return res  
  
 def runDebugMethod(self,jklx,url,cookies,data=None):  
 *'''接口调试、联调调用函数'''* urls = getUrlAndKey()  
 jkfwurl = urls[0]  
 if jklx == 'get':  
 res = self.sendGet(jkfwurl+url,cookies,data)  
 elif jklx == 'post':  
 res = self.sendPost(jkfwurl+url,cookies,data)  
 elif jklx == 'json\_post':  
 res = self.sendPostJson(jkfwurl+url,cookies,data)  
 else :  
 res = self.sendPostFile(jkfwurl+url,cookies,data)  
 return res

##### sqlResultVerify.py数据库通用验证

from config.configReadExcel import getMysql  
class resultVerify():  
 *'''数据库通用验证'''* def \_\_init\_\_(self,sqlCondition,sqls):  
 self.sqlCondition = sqlCondition #sql查询的条件参数  
 self.sql = sqls #sql语句  
 # 连接数据库  
 try:  
 # dbcon = mysql\_config() #使用配置文件，已废弃  
 dbcon = getMysql()  
 self.cursor = dbcon[0]  
 self.connect = dbcon[1]  
 sql = self.sql  
 if self.sqlCondition == 0:  
 self.cursor.execute(sql)  
 else:  
 data = (self.sqlCondition)  
 self.cursor.execute(sql % data)  
 except:  
 self.connect.close()  
 def rowcount(self):  
 *'''查询出影响的行数'''* try:  
 return self.cursor.rowcount  
 finally:  
 self.connect.close()  
 def fetchall(self):  
 *'''判断sql返回多列数据与预期对比（比如修改后的值与预期结果对比）'''* try:  
 sqlResult = self.cursor.fetchall()  
 print(sqlResult)  
 listSqlResult = []  
 for i in sqlResult[0]:  
 listSqlResult.append(i)  
 return str(listSqlResult)  
 finally:  
 self.connect.close()  
 def closeConnect(self):  
 *'''关闭数据库连接'''* self.connect.close()  
 def runMethod(self,faction):  
 if faction == 1.0 or faction == 0.0:  
 sqlReturnResult = self.rowcount()  
 else :  
 sqlReturnResult = self.fetchall()  
 return sqlReturnResult  
if \_\_name\_\_ == '\_\_main\_\_':  
 ''''''  
 # result\_verify('项目id：52422514').dy\_all('ryxmbd')  
 # result\_verify('项目id：52422514').dy\_all('ryxmjb')

##### sqlReturnParam.py数据库通用返回参数

from config.configReadExcel import \*  
#连接数据库  
bs = 0 #初始化标识变量，此变量用来判断是否返回一个sql里的多个值  
class refer():  
 *'''返回数据获取公共调用'''* def parameter(self,sqls,bs=None):  
 # 连接数据库  
 try:  
 # dbcon = mysql\_config() #使用配置文件，已废弃  
 dbcon = getMysql()  
 cursor = dbcon[0]  
 connect = dbcon[1]  
 cursor.execute(sqls)  
 sqlResult = cursor.fetchall() #输出为元祖，格式：(('123',),)  
 # print(sqlResult)  
 # print(sqlResult[0])  
 # print(len(sqlResult))  
 if len(sqlResult) > 1:  
 if bs == 1:  
 return sqlResult[0][0],sqlResult[0][1] #返回一个元祖，返回一个元素的2个值  
 elif bs == 2:  
 return sqlResult #返回所有元素  
 else:  
 return sqlResult[0][0],sqlResult[1][0] #返回的是一个元祖，返回2个元素  
 else:  
 return sqlResult[0] #返回元祖  
 except:  
 ''''''  
 finally:  
 connect.close()  
if \_\_name\_\_ == '\_\_main\_\_':  
 ''''''  
 # cookies = {"Cookie":"JSESSIONID=E780B9F78EA6BB8FA6BF2E1A8EC5DB00"}  
 # refer().get\_dlsCid(cookies)  
 # refer()

##### randoms.py获得随机数

import random  
def getRandomParam(bs):  
 *'''获得随机参数'''* randomInt = random.randint(1000000000000000000000000000000000000000,9999999999999999999999999999999999999999)  
 randomStr = 'qazwsxedcrfvtgbyhnujmikolp'  
 if bs == 'str': #获取字符串  
 res = random.sample(randomStr,8)  
 getStr = ''.join(res)  
 return getStr  
 else:  
 res = random.sample(str(randomInt),bs - 3)  
 getInt = ''.join(res)  
 return int('177' + getInt) #获取整数

##### runScript.py 业务执行主体

from common.apiDemo import apiPack  
from common.sqlResultVerify import resultVerify  
from common.sqlReturnParam import refer  
from config.configReadExcel import \*  
# from config.readconfig import \* #使用配置文件，已废弃  
import json  
from common.randoms import getRandomParam  
  
def runApiCase(apiCaseData, headers, sgin, debug, data=None, shtc=None):  
 *"""脚本执行主体：传入接口参数、头信息、是否输出excel报告标志、是否为调式程序、完整用例数据、工作表对象  
 sgin=1 ：单次测试接口，执行apiCaseDebug.py文件  
 sgin=2 ：自动化执行接口并输出html和excel测试报告，执行testApiCase.py文件  
 sgin=3 ：自动化回归bug用例，不输出任何报告，执行caseBugUAT.py文件  
 debug=1 ：为执行调试代码，执行apiCaseTemporaryDebug.py文件  
 """* file = 0 # 初始化文件路径file  
 apiParameters = 0 # 初始化参数  
 ran = 5 # 初始化随机数  
  
# 判断是否为文件上传接口  
 if apiCaseData['是否文件'] == 1.0:  
 if len(apiCaseData['接口参数']) > 0: # 判断是否存在参数，有些文件上传请求是不带参数的  
 apiParameters = json.loads(apiCaseData['接口参数']) #参数序列化，execl读取的数据是字符串  
 # file = {'file': open(rootPath + 'data/image/' + apiCaseData['文件路径'], 'rb')} # 定义文件 #使用配置文件，已废弃  
 file = {'file': open(rootPath() + 'data/image/' + apiCaseData['文件路径'], 'rb')} # 定义文件  
  
 else:  
 if len(apiCaseData['接口参数']) > 0: # 判断是否存在参数，因为有些get请求是不带参数的  
 print(apiCaseData['接口参数'])  
 apiParameters = json.loads(apiCaseData['接口参数']) #参数序列化  
 print(apiParameters)  
 print(type(apiParameters))  
  
  
# 参数传递是否需要参数化  
 prameter = [] #初始化参数数组，用于保存参数化后的参数  
 if type(apiParameters) == list:  
 apiParameterList = apiParameters #如果接口参数是数组，就直接赋值给apiParameterList  
 else:  
 apiParameterList = [apiParameters] # 如果不是数组格式，就转换为数组  
  
 for apiParameter in apiParameterList:  
 '''遍历接口参数list'''  
 # 获得随机数  
 if apiCaseData['status'] == 1.0 or apiCaseData['status'] == 6.0:  
 '''标志位1或者6时需要用到随机数，获取随机数'''  
 rando = []  
 listsgin = apiCaseData['randoSgin'].split(',')  
 # print(listsgin)  
 for i in listsgin: # 遍历随机数标志位  
 if i == 'str':  
 rando.append(getRandomParam(i)) # 调用随机字符串函数  
 else:  
 res = getRandomParam(int(i)) # 调用随机整数函数  
 rando.append(res) # 把随机整数传给rando数组  
 # print(rando)  
  
 if apiCaseData['status'] == 1.0:  
 '''变量用随机数赋值'''  
 j = 0 #初始化获取随机数组的下标  
 for i in apiParameter: # 遍历接口参数，遍历的值是key  
 if apiParameter[i] == 'variable':  
 if len(rando) > 1: # 如果随机数组大于1  
 apiParameter[i] = rando[j] # 指定随机元素赋值给变量  
 print(apiParameter[i])  
 j = j + 1  
 else:  
 apiParameter[i] = rando[0]  
  
 elif apiCaseData['status'] == 2.0:  
 '''sql返回值代替变量'''  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 vari = refer().parameter(apiCaseData['参数传递sql'])  
 print(type(vari))  
 vari = vari[0] # 把调用函数得到的list的值赋值给对应的key  
 apiParameter[i] = vari  
  
 elif apiCaseData['status'] == 3.0:  
 '''变量用上一接口的返回参数代替'''  
 # getApiReturnData = open(rootPath + 'data/return\_data.txt', 'r') # 打开上一接口返回参数文件 #使用配置文件，已废弃  
 getApiReturnData = open(rootPath() + 'data/return\_data.txt', 'r') # 打开上一接口返回参数文件  
 readApiData = getApiReturnData.read().split(',') # 读取参数值  
 readApiData.pop() #去掉数组中最后一个逗号  
 print(readApiData)  
 j = 0  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 apiParameter[i] = readApiData[j] # 将返回参数赋值给当前变量  
 if len(readApiData) > 1:  
 j = j + 1  
 # getApiReturnData = open(rootPath + 'data/return\_data.txt', 'r') # 打开上一接口返回参数文件  
 # readApiData = getApiReturnData.read() # 读取参数值  
 # for i in apiParameter:  
 # if apiParameter[i] == 'variable':  
 # apiParameter[i] = readApiData # 将返回参数赋值给当前变量  
  
 elif apiCaseData['status'] == 4.0:  
 '''参数值是数组的情况'''  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 vari = refer().parameter(apiCaseData['参数传递sql'])  
 vari = [vari[0], vari[1]]  
 apiParameter[i] = vari # 把调用函数得到的list的值赋值给对应的key  
  
 elif apiCaseData['status'] == 5.0:  
 '''sql返回的多个值依次代替参数中的多个变量'''  
 j = 0  
 # listSql = apiCaseData['参数传递sql'].split(';')  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 vari = refer().parameter(apiCaseData['参数传递sql'], 1)  
 print(type(vari))  
 # print(vari)  
 vari = vari[j] # 把调用函数得到的list的值赋值给对应的key  
 apiParameter[i] = vari  
 j = j + 1  
  
 elif apiCaseData['status'] == 6.0:  
 '''变量既有sql返回值又有随机数'''  
 listSqlSgin = apiCaseData['选择sql变量'].split(',')  
 j = 0 # 定义变量个数判断条件  
 s = 0 # 定义sql数组取值下标  
 w = 0 # 定义随机数取值下标  
 listSql = apiCaseData['参数传递sql'].split(';') # 分割sql  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 for value in listSqlSgin: # 遍历需要使用sql传参的位置  
 if int(value) == j + 1: # 判读sql传参的位置计数=变量出现的次数  
 vari = refer().parameter(listSql[s])  
 print(type(vari))  
 vari = vari[0] # 把调用函数得到的list的值赋值给对应的key  
 apiParameter[i] = vari  
 s = s + 1  
 break  
 j = j + 1  
  
 for i in apiParameter: # 遍历接口参数，遍历的值是key，sql传参替换完后进行随机数传参  
 if apiParameter[i] == 'variable': #  
 if len(rando) > 1:  
 apiParameter[i] = rando[w] #  
 print(apiParameter[i])  
 w = w + 1  
 else:  
 apiParameter[i] = rando[0]  
  
 elif apiCaseData['status'] == 7.0:  
 '''调试程序需要循环执行接口时使用'''  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 apiParameter[i] = shtc[data] # shtc：listData数组数据，data：循环次数的下标  
 # print(apiParameter)  
  
 elif apiCaseData['status'] == 8.0:  
 '''一组数据中，随机选择一个'''  
 import random  
 listRno = apiCaseData['randoSgin'].split(',')  
 for i in apiParameter:  
 if apiParameter[i] == 'variable':  
 apiParameter[i] = listRno[random.randint[len(listRno - 1)]]  
  
 prameter.append(apiParameter) #把字典参数加入到数组中  
 if type(apiParameters) == list: #如果原参数是数组，就把当前接口参数赋值给数组  
 apiParameter = prameter  
  
# 判断是否需要进行算术运算  
 if apiCaseData['算术计算'] == 1.0:  
 '''判断是否进行算术运行，为1时代表执行接口前查出影响的行数（比如批量删除操作）'''  
 initialNum = resultVerify(0, apiCaseData['算术计算sql']).rowcount()  
 elif apiCaseData['算术计算'] == 2.0:  
 '''判断为2时查询出sql查询出的结果（比如入库出库操作）'''  
 initialNum = refer().parameter(apiCaseData['算术计算sql'])  
  
  
  
# 执行接口  
 '''创建文本存放不通过的测试用例编号'''  
 # openFile = open(rootPath + 'Bug/caseBugNumber.txt', 'a') #使用配置文件，已废弃  
 openFile = open(rootPath() + 'Bug/caseBugNumber.txt', 'a')  
 if debug == 1:  
 '''判断是否为调试程序'''  
 apiReturnResults = apiPack(file).runDebugMethod(apiCaseData['接口类型'], apiCaseData['接口地址'], headers,apiParameter) # 传参调用接口封装脚本  
 else:  
 apiReturnResults = apiPack(file).runMethod(apiCaseData['接口类型'], apiCaseData['接口地址'], headers,apiParameter) # 传参调用接口封装脚本  
 print('接口用例：%s' % apiCaseData)  
 print('\t') #水平制表符  
 print('接口传参：%s' % json.dumps(apiParameter)) #转化为json格式输出  
 print('\t')  
 print('接口响应结果：%s' % apiReturnResults.json())  
 print('\t')  
 print('接口响应结果：%s' % apiReturnResults.text)  
 print('\t')  
 # a = apiReturnResults.text  
 # a1 = open('test.txt','w', encoding="utf-8")  
 # a1.write(a)  
 # print('接口响应条数：%s' % len(apiReturnResults.json()['result']))  
 # print('\t')  
 print('预期结果：%s' % apiCaseData['接口返回预期结果'])  
 print('\t')  
 print('接口url：%s' % apiReturnResults.url)  
 print('\t')  
 # print(len(apiReturnResults.json()["result"]))  
  
  
 # 找出sql查询出的userid与接口返回出的userid不一致的问题  
 # listid=[]  
 # for id in apiReturnResults.json()['result']:  
 # listid.append(id['id'])  
 # # print(listid)  
 # sqlr = refer().parameter(apiCaseData['验证结果sql语句'],2)  
 #  
 # listidsql=[]  
 # for userid in sqlr:  
 # listidsql.append(userid[0])  
 # # print(len(listidsql))  
 # for i in listidsql:  
 # if i not in listid:  
 # print(i)  
  
 ExResultsList = apiCaseData['接口返回预期结果'].split(',') # 取出预期结果加入到数组ExResultsList中  
 for i in ExResultsList: # 循环判断预期结果是否在接口返回结果中  
 try:  
 assert i in apiReturnResults.text  
 if sgin == 2:  
 for j in range(len(data)):  
 if data[j]['用例编号'] == apiCaseData['用例编号']:  
 shtc.write(j + 1, 8, '通过') # 行、列、值  
 except:  
 if sgin == 1:  
 assert i in apiReturnResults.text  
 else:  
 openFile.write(apiCaseData['用例编号'])  
 openFile.write(',')  
 if sgin == 2:  
  
 for j in range(len(data)):  
 if data[j]['用例编号'] == apiCaseData['用例编号']:  
 shtc.write(j + 1, 8, '不通过') # 行、列、值  
 shtc.write(j + 1, 7, apiReturnResults.text) # 行、列、值  
 break  
 assert i in apiReturnResults.text  
  
  
 if apiCaseData['是否参数传递'] == 1.0:  
 '''判断是否需要把接口的返回数据作为下一个接口的参数传递'''  
 # openFile = open(rootPath + 'data/return\_data.txt', 'w') #使用配置文件，已废弃  
 openFile = open(rootPath() + 'data/return\_data.txt', 'w')  
 openFile.truncate()  
 listReturnParam = apiCaseData['参数名称'].split(',') #分割需要返回的参数名称  
 for paramName in listReturnParam:  
 returnData = apiReturnResults.json()[paramName] # 不同项目这里需要更改  
 openFile.write(returnData)  
 openFile.write(',')  
 # returnData = apiReturnResults.json()[apiCaseData['参数名称']] # 不同项目这里需要更改  
 # openFile = open(rootPath + 'data/return\_data.txt', 'w')  
 # openFile.write(returnData)  
  
  
# 执行算术运算为前提：执行接口后再来查询出影响的行数或者sql查询的数值  
 if apiCaseData['算术计算'] == 1.0:  
 nowNum = resultVerify(0, apiCaseData['算术计算sql']).rowcount()  
 elif apiCaseData['算术计算'] == 2.0:  
 nowNum = refer().parameter(apiCaseData['算术计算sql'])  
 if apiCaseData['算术标识'] == 1.0:  
 '''为1时代表做减法操作'''  
 assert int(initialNum) - int(apiCaseData['算术值']) == int(nowNum)  
 elif apiCaseData['算术标识'] == 2.0:  
 '''为2时代表做加法操作'''  
 assert int(initialNum) + int(apiCaseData['算术值']) == int(nowNum)  
  
  
# 数据库校验  
 verifySql = apiCaseData['验证结果sql语句'].split(';')  
 for i in range(len(verifySql)):  
 if apiCaseData['数据库验证预期识别码'] == 2.0:  
 '''sql传参用sql返回的变量'''  
 resert = resultVerify(vari[0], verifySql[i]).runMethod(apiCaseData['调用验证函数']) # 分别传入sql的条件参数、sql语句、调用sql验证标识  
 try:  
 # self.assertEqual(resert, apiCaseData['sql预期结果'])  
 assert resert == apiCaseData['sql预期结果']  
 except:  
 if sgin == 1: #如果标志位为1，就直接输出结果和断言，用于第一次执行接口和调试  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 assert resert == apiCaseData['sql预期结果']  
 else:  
 openFile.write(apiCaseData['用例编号'])#把用例编号写入bug记录文本  
 openFile.write(',')  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 if sgin == 2: #如果等于2，就输出excel测试报告，用于自动化测试  
 for j in range(len(data)):  
 if data[j]['用例编号'] == apiCaseData['用例编号']:  
 shtc.write(j + 1, 8, 'sql不通过') # 行、列、值  
 shtc.write(j + 1, 7, 'sql返回结果：' + str(resert))  
 break  
 # self.assertEqual(resert, apiCaseData['sql预期结果'])  
 assert resert == apiCaseData['sql预期结果']  
  
  
 elif apiCaseData['数据库验证预期识别码'] == 1.0: #为1时，sql传参变量用随机数  
 resert = resultVerify(rando[0], verifySql[i]).runMethod(apiCaseData['调用验证函数'])  
 try:  
 assert resert == apiCaseData['sql预期结果']  
 except:  
 if sgin == 1:  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 assert resert == apiCaseData['sql预期结果']  
 else:  
 openFile.write(apiCaseData['用例编号'])  
 openFile.write(',')  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 if sgin == 2:  
 for j in range(len(data)):  
 if data[j]['用例编号'] == apiCaseData['用例编号']:  
 shtc.write(j + 1, 8, 'sql不通过') # 行、列、值  
 shtc.write(j + 1, 7, 'sql返回结果：' + str(resert))  
 break  
 # self.assertEqual(resert, apiCaseData['sql预期结果'])  
 assert resert == apiCaseData['sql预期结果']  
  
  
 elif apiCaseData['数据库验证预期识别码'] == 3.0: #为3时  
 resert = resultVerify(0, verifySql[i]).runMethod(apiCaseData['调用验证函数'])  
 print('sql查询结果条数 = %d' % int(resert))  
 try:  
 if apiCaseData['调用验证函数'] == 0.0:  
 '''为0时作用：sql查询出的结果与接口返回的结果自动对比'''  
 param = apiCaseData['响应参数'].split(',')  
 #取多层返回数据  
 print(param)  
 if len(param) > 1:  
 a = apiReturnResults.json()[param[0]]  
 print(type(a))  
 b = json.loads(a)  
 # print(b["totle"])  
 # leng = len(apiReturnResults.json()[apiCaseData['响应参数']]) # 取出接口返回的数据条数，换项目后这里需要修改  
 leng = int(b["totle"]) # 取出接口返回的数据条数，换项目后这里需要修改  
 print("接口返回的条数=%d" % leng)  
 else:  
 leng = len(apiReturnResults.json()[apiCaseData['响应参数']])  
 # assert leng == int(resert)  
 # self.assertEqual(leng, int(resert))  
 assert leng == int(resert)  
 else:  
 # self.assertEqual(resert, apiCaseData['sql预期结果'])  
 assert resert == apiCaseData['sql预期结果']  
 except:  
 if sgin == 1:  
 if apiCaseData['调用验证函数'] == 0.0:  
 '''为0时作用：sql查询出的结果与接口返回的结果自动对比'''  
 leng = len(apiReturnResults.json()[apiCaseData['响应参数']]) # 取出接口返回的数据条数，换项目后这里需要修改  
 # assert leng == int(resert)  
 # self.assertEqual(leng, int(resert))  
 print('接口返回参数个数:%s' % leng)  
 print('sql返回结果:%s' % resert)  
 assert leng == int(resert)  
 else:  
 # self.assertEqual(resert, apiCaseData['sql预期结果'])  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 assert resert == apiCaseData['sql预期结果']  
 else:  
 openFile.write(apiCaseData['用例编号'])  
 openFile.write(',')  
 print('sql预期结果:%s' % apiCaseData['sql预期结果'])  
 print('sql返回结果:%s' % resert)  
 if sgin == 2:  
 for j in range(len(data)):  
 if data[j]['用例编号'] == apiCaseData['用例编号']:  
 shtc.write(j + 1, 8, 'sql不通过') # 行、列、值  
 shtc.write(j + 1, 7, 'sql返回结果：' + str(resert))  
 break  
 assert 1 == 2

#### config配置模块

##### （废弃）configure.ini配置文件

**[mysql]**;定义数据库-测试库  
host = gzxxxxdb.com  
port=62xxx  
user=xxx  
passwd=@xxx8  
db=txxxas  
  
**[api\_service\_url]**;定义接口服务地址  
;测试服务器  
url = http://xxxint.com  
#张耀文本地  
;url = http://1xxxx47:9091  
#正式服务器  
;url = https://xxxcom  
**[fileName]**#定义用例excel名字  
filename = qjy\_ht\_api.xls

##### （废弃）readconfig.py读取配置文件

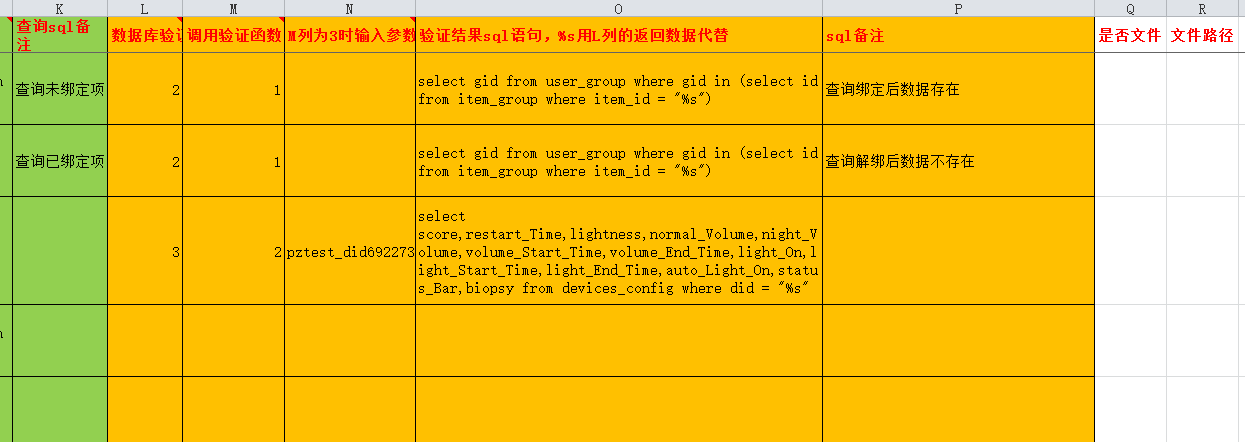
import configparser  
import pymysql  
import os  
conf = configparser.ConfigParser() #创建管理对象  
curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
rootPath = os.path.split(curPath)[0] + '\\' + 'config\configure.ini'  
conf.read(rootPath,encoding='utf-8')  
class sql():  
 def host(self):  
 *''''''*def mysql\_config():  
 *'''创建数据库连接'''* connect = pymysql.Connect(  
 host=conf.get('mysql','host'), # 连接地址  
 port=int(conf.get('mysql','port')), # 端口  
 user=conf.get('mysql','user'), # 用户名  
 passwd=conf.get('mysql','passwd'), # 密码  
 db=conf.get('mysql','db'), # 数据库名  
 charset='utf8' # 连接编码  
 )  
 cursor = connect.cursor()  
 return cursor,connect  
def api\_service\_url():  
 *'''获取接口服务url'''* return conf.get('api\_service\_url','url')  
def filenames():  
 *'''获取用例excel名'''* return conf.get('fileName','filename')  
def rootpath():  
 *'''获取项目根目录'''* curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 root\_Path = os.path.split(curPath)[0] + '\\'  
 return root\_Path  
if \_\_name\_\_ == '\_\_main\_\_':  
 mysql\_config()  
 # ss1()

##### configReadExcel.py配置文件

import xlrd  
import os  
import pymysql  
def filesName():  
 *'''接口用例表名'''* return 'qjyApiCase.xls'  
  
def rootPath():  
 *'''获取项目跟路径'''* curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 root\_Path = os.path.split(curPath)[0] + '\\'  
 return root\_Path  
  
def getApiService(bs):  
 *'''获取服务器地址'''* #测试服务器  
 testurl = 'http://test-paas.mustint.com'  
 # 开发本地  
 localurl = 'http://192.168.166.99:9091'  
 # 正式服务器  
 releaseurl = 'https://paas-api.mustint.com'  
 if bs == 1:  
 return testurl  
 elif bs == 2:  
 return releaseurl  
 else:  
 return localurl  
  
def getMysql():  
 *'''获得数据库信息'''* connect = pymysql.Connect(  
 host='XXX.107', # 连接地址  
 port=7777, # 端口  
 user=XXXX, # 用户名  
 passwd='XXXX', # 密码  
 db='XXX', # 数据库名  
 charset='utf8' # 连接编码  
 )  
 # 使用 self.cursor() 方法创建一个游标对象 self.cursor  
 cursor = connect.cursor()  
 return cursor,connect  
  
def getUrlAndKey():  
 *'''获得接口地址与appkey'''* curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
 root\_Path = os.path.split(curPath)[0] + '\\'  
 filename = filesName()  
 path = root\_Path + 'data' + '\\'  
 sheet = 'config'  
 book = xlrd.open\_workbook(path + filename, 'r')  
 sheets = book.sheet\_by\_name(sheet)  
 formalDb = sheets.row\_values(2) #获取正式库的信息  
 testDb = sheets.row\_values(4) #获取测试库的信息  
 devUrl = sheets.row\_values(9) #开发本地服务  
 choiceDb = sheets.row\_values(5) #获取选择数据库的值  
 keys = sheets.row\_values(6) #获取选择公司的值  
 key = keys[1]  
 if choiceDb[1] == 1.0: #代表正式库  
 if key == 1.0: #智必选  
 apiParameter = {"appid":formalDb[1],"appkey":formalDb[2]}  
 elif key == 2.0: #马丁洛克  
 apiParameter = {"appid": formalDb[3], "appkey": formalDb[4]}  
 elif key == 3.0: #臻识科技  
 apiParameter = {"appid": formalDb[5], "appkey": formalDb[6]}  
 elif key == 4.0:  
 apiParameter = {"appid": formalDb[7], "appkey": formalDb[8]}  
 else:  
 apiParameter = {"appid": formalDb[9], "appkey": formalDb[10]}  
 apiSeversUrl = formalDb[0]  
 elif choiceDb[1] == 2.0:  
 if key == 1.0: #智必选  
 apiParameter = {"appid": testDb[1], "appkey": testDb[2]}  
 elif key == 2.0: #马丁洛克  
 apiParameter = {"appid": testDb[3], "appkey": testDb[4]}  
 elif key == 3.0: #臻识科技  
 apiParameter = {"appid": testDb[5], "appkey": testDb[6]}  
 elif key == 4.0: #大益科技  
 apiParameter = {"appid": testDb[7], "appkey": testDb[8]}  
 else: #汉博德  
 apiParameter = {"appid": formalDb[9], "appkey": formalDb[10]}  
 apiSeversUrl = testDb[0]  
 else:  
 if key == 1.0: #智必选  
 apiParameter = {"appid": testDb[1], "appkey": testDb[2]}  
 elif key == 2.0: #马丁洛克  
 apiParameter = {"appid": testDb[3], "appkey": testDb[4]}  
 elif key == 3.0: #臻识科技  
 apiParameter = {"appid": testDb[5], "appkey": testDb[6]}  
 elif key == 4.0: #大益科技  
 apiParameter = {"appid": testDb[7], "appkey": testDb[8]}  
 else: #汉博德  
 apiParameter = {"appid": formalDb[9], "appkey": formalDb[10]}  
 apiSeversUrl = devUrl[0]  
 return apiSeversUrl,apiParameter  
  
  
  
if \_\_name\_\_=='\_\_main\_\_':  
 getUrlAndKey()

#### excel表数据结构





#### 读取excel代码如下：

import xlrd  
import os  
class read\_excel\_data():  
 def \_\_init\_\_(self,filepath,filename,sheet):  
   
 book = xlrd.open\_workbook(filepath+'\\'+filename,'r')  
   
 self.sheet = book.sheet\_by\_name(sheet) #指定获取表里的sheet列  
 self.row = self.sheet.nrows  
 def rown(self):  
 return self.row #返回数据行数  
 def get\_line(self,bs):  
 # print(self.sheet.row\_values(bs))  
 return self.sheet.row\_values(bs) #获取某一行的值  
 def get\_all(self): #获取excel所有用例  
 a = self.rown()  
 b = [] #新建一个数组存放excel数据  
 for i in range(1,a): #遍历excel表  
 c = self.get\_line(i) #获取每一行的数据  
 b.append(c) #把每一行的数据加入到数组中，每行数据是一个list  
 return b  
if \_\_name\_\_ == '\_\_main\_\_':  
 filepath = './'  
 filename = 'qjy\_open\_api.xls'  
 sheet = "Sheet1"  
 a = read\_excel\_data(filepath,filename,sheet).get\_line(1)  
 print(a)  
 print(type(a))

# Page Object设计思想

实现元素、业务操作、用例三者分离，方便维护和减少代码的重复量

## 关于断言部分

如果写在demo里，那么如果预期结果与实际结果不相等时，输出报告日志上不会输出实际脚本的运行结果

如果写在unittest框架里，那么如果预期结果与实际结果不相等时，输出报告日志上会输出实际脚本的运行结果

所以建议把断言写在unittest框架里

## Page\_object设计实例

### 公共封装commom模块

脚本源码：

from selenium import webdriver  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver. support import expected\_conditions as EC  
import unittest  
from selenium.webdriver.common.by import By  
from common.sql\_result\_verify import result\_verify  
import time  
class commons():  
 *'''封装公共调用'''*#公共元素层  
 cx = By.ID, 'queryBtn' #查询按钮  
 csjghs = By.CSS\_SELECTOR, '[class="pagination-info"]' #列表底部查询结果行数  
 sjcz = By.CSS\_SELECTOR, '[type="reset"]' #搜素框数据重置按钮  
 refresh = By.CSS\_SELECTOR, '[onclick="refreshList();"]' #列表刷新按钮  
 pl\_delete = By.ID, 'batchRemove' #批量删除按钮  
 mod = By.CSS\_SELECTOR,'[class="queryDetails"]' #编辑按钮  
 delete = By.CSS\_SELECTOR,'[class="remove"]' #删除按钮  
 delete\_tsk = By.CSS\_SELECTOR,'[class="layui-layer layui-layer-dialog layer-anim"]' #删除提示框  
 delete\_tsk\_qd = By.CSS\_SELECTOR,'[class="layui-layer-btn0"]' #删除提示框里的 【确定】按钮  
 delete\_tsk\_qx = By.CSS\_SELECTOR,'[class="layui-layer-btn1"]' #删除提示框里的 【取消】按钮  
 list\_table = By.ID,'dataGrid' #数据列表table  
 wsjzs = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr/td' #列表无数据展示：没有找到匹配的记录  
 rqsrk = By.ID,'test6' #时间输入框  
 username = By.XPATH,'/html/body/div/header/nav/div[2]/ul/li' #右上角用户名  
 loginout\_button = By.XPATH,'/html/body/div/header/nav/div[2]/ul/li/ul/li[2]' #退出按钮  
  
#公共函数调用层  
 def get\_cx\_rown(self,driver):  
 *'''获得查询结果的总数量'''* data = self.get\_text(driver,self.csjghs)  
 b = data[data.find('共') + 1:] #从【显示第 1 到第 10 条记录，总共 93 条记录】取出【 93 条记录】  
 a = b[0:b.find('条')] #从【 93 条记录】取出【 93 】  
 hsz = a.strip() #去掉字符串2边的空格得到一个整数  
 return int(hsz)  
  
 def cx\_alldata(self,driver,sql):  
 *'''默认全查'''* self.Click(driver,self.cx)  
 hsz = self.get\_cx\_rown(driver)  
 rown = result\_verify(sql).rowcount()  
 return hsz,rown  
 # assert hsz == rown  
  
 def select\_sjk(self,driver):  
 *'''选择勾选数据框'''* a = self.ysdw(driver,self.list\_table) #定位数据列表标签，并把浏览器对象driver转移到变量a  
 b = a.find\_element\_by\_tag\_name('tbody') #定位tbody标签  
 c = b.find\_elements\_by\_tag\_name('tr') #定位一组元素标签tr  
 print(len(c))  
 c[0].find\_element\_by\_css\_selector('[class="bs-checkbox bella-checkbox"]').click() #点击第一条数据的勾选框  
 c[1].find\_element\_by\_css\_selector('[class="bs-checkbox bella-checkbox"]').click() #点击第二条数据的勾选框  
  
 def default\_content(self,driver):  
 *'''返回到顶层目录结构'''* driver.switch\_to.default\_content()  
  
 def delete\_pl(self,driver):  
 *'''批量删除'''* self.Click(driver,self.cx)  
 before\_hsz = self.get\_cx\_rown(driver) #查询出删除前的数据总数  
 self.select\_sjk(driver) #调用选择数据框  
 self.Click(driver, self.pl\_delete) #点击批量删除按钮  
 time.sleep(1)  
 self.default\_content(driver) #切换到元素区对外层  
 driver.switch\_to\_frame('mainFrame') #切换到frame标签  
 self.ysdw(driver,self.delete\_tsk) #定位删除提示框  
 self.Click(driver,self.delete\_tsk\_qd)  
 time.sleep(2)  
 now\_hsz = self.get\_cx\_rown(driver) #查询出删除后的数据总数  
 return before\_hsz,now\_hsz  
 # assert before\_hsz - 2 == now\_hsz #删除2条数据后比对  
  
 def delete\_dt(self,driver):  
 *'''单条删除'''* before\_hsz = self.get\_cx\_rown(driver) # 查询出删除前的数据总数  
 self.Click(driver,self.delete)  
 time.sleep(1)  
 self.default\_content(driver) # 切换到元素区对外层  
 driver.switch\_to\_frame('mainFrame') # 切换到frame标签  
 self.ysdw(driver, self.delete\_tsk) # 定位删除提示框  
 self.Click(driver, self.delete\_tsk\_qd)  
 time.sleep(2)  
 now\_hsz = self.get\_cx\_rown(driver) # 查询出删除后的数据总数  
 return before\_hsz, now\_hsz  
 # assert before\_hsz - 1 == now\_hsz #删除一条数据后比对  
  
 def datatime\_cl(self,driver,sj):  
 *'''时间输入处理:创建时间'''* # js = "$('input[id=test6]').removeAttr('readonly')" # 2.jQuery，移除属性  
 # self.ysdw(driver,self.sj)  
 # js = "document.getElementById('test6').removeAttribute('readonly')"  
 js = "document.getElementById('createTime').removeAttribute('type')" #获得隐藏属性  
 driver.execute\_script(js) # 执行js脚本  
 driver.find\_element\_by\_id('createTime').send\_keys(sj)  
  
 def datatime\_cx(self,driver,sj,sql=None):  
 *'''时间查询'''* self.datatime\_cl(driver,sj)  
 self.Click(driver, self.cx)  
 time.sleep(3)  
 try:  
 hsz = self.get\_cx\_rown(driver)  
 assert hsz > 0  
 except:  
 assert self.get\_text(driver, self.wsjzs) == '没有找到匹配的记录'  
 else:  
 sql\_hsz = result\_verify(sql).rowcount()  
 assert hsz == sql\_hsz  
 # assert hsz == sql\_hsz  
  
 def login(self):  
 *'''登录'''* wd = webdriver.Chrome()  
 wd.maximize\_window()  
 wd.get('http://test-pass-admin.mustint.com/console/login')  
 wd.implicitly\_wait(3)  
 wd.find\_element\_by\_css\_selector('[name="username"]').send\_keys('admin')  
 wd.find\_element\_by\_css\_selector('[name="password"]').send\_keys('000000')  
 wd.find\_element\_by\_css\_selector('[placeholder="验证码"]').send\_keys('h$\*+1@+3^&5#%prs)rs\_a') #万能验证码  
 wd.find\_element\_by\_css\_selector('[class="btn btn-login bg\_color mb-4"]').click()  
 time.sleep(2)  
 return wd  
 def loginout(self,driver):  
 *'''退出登录'''* self.Click(driver,self.username)  
 time.sleep(1)  
 self.Click(driver,self.loginout\_button)  
 def ysdw(self,driver,cs):  
 *'''元素定位'''* return driver.find\_element(\*cs)  
 def yz\_ysdw(self,driver,cs):  
 *'''定位一组元素定位'''* return driver.find\_elements(\*cs)  
 def Click(self,driver,cs):  
 *'''元素点击'''* return self.ysdw(driver,cs).click()  
 def get\_text(self,driver,cs):  
 *'''获取元素文本'''* return self.ysdw(driver,cs).text  
 def send\_text(self,driver,cs,wb):  
 *'''输入文本'''* return self.ysdw(driver,cs).send\_keys(wb)  
 def get\_sx(self,driver,cs,ys):  
 *'''获取元素的某一个属性'''* return self.ysdw(driver,cs).get\_attribute(ys)  
 def xsdd(self,driver,cs):  
 *'''显示等待'''* return WebDriverWait(driver,10,0.5).until(EC.presence\_of\_element\_located(cs))  
 def ys\_is\_displayed(self,driver,ys):  
 return self.ysdw(driver,ys).is\_displayed()  
  
if \_\_name\_\_=='\_\_main\_\_':  
 commons().login()

### config配置模块

该模块下有2个文件

1. Configure.ini脚本源码：

**[mysql]**;定义数据库-测试库  
host = gz-xxx.sql.tencentcdb.com  
port=6xx6  
user=xx  
passwd=@xxx8  
db=texxxas  
  
**[api\_service\_url]**;定义接口服务地址  
;测试服务器  
url = http://xxxx.com

1. Readconfig.py脚本源码
2. import configparser  
   import pymysql  
   import os  
   conf = configparser.ConfigParser() #创建管理对象  
   curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
   rootPath = os.path.split(curPath)[0] + '\\' + 'config\configure.ini'  
   conf.read(rootPath,encoding='utf-8')  
   class sql():  
    def host(self):  
    *''''''*def mysql\_config():  
    *'''创建数据库连接'''* connect = pymysql.Connect(  
    host=conf.get('mysql','host'), # 连接地址  
    port=int(conf.get('mysql','port')), # 端口  
    user=conf.get('mysql','user'), # 用户名  
    passwd=conf.get('mysql','passwd'), # 密码  
    db=conf.get('mysql','db'), # 数据库名  
    charset='utf8' # 连接编码  
    )  
    cursor = connect.cursor()  
    return cursor,connect  
   def api\_service\_url():  
    *'''获取接口服务url'''* return conf.get('api\_service\_url','url')  
   def filenames():  
    *'''获取用例excel名'''* return conf.get('fileName','filename')  
   def rootpath():  
    *'''获取项目根目录'''* curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
    root\_Path = os.path.split(curPath)[0] + '\\'  
    return root\_Path  
   if \_\_name\_\_ == '\_\_main\_\_':  
    mysql\_config()  
    # ss1()

### page\_object模块

脚本源码：

from selenium.webdriver.common.by import By  
from common.common import commons  
from common.sql\_result\_verify import result\_verify  
import time  
class dsfgs\_object(commons):  
 *'''第三方公司页面对象与业务逻辑封装'''*#首页对象层  
 gsmc = By.ID,"companyName"  
 rq = By.XPATH,'//\*[@id="test6"]'  
 # cz = By.XPATH,'/html/body/div[1]/div[1]/form/div[3]/button/i'  
 dsfgs = By.XPATH,'/html/body/div/div[1]/aside/section/ul/li[2]/a/b'  
 gslb = By.XPATH,'//html/body/div/div[1]/aside/section/ul/li[2]/ul'  
 # pl\_delete = By.ID,'batchRemove'  
 add\_dsfgs = By.XPATH,'/html/body/div[1]/div[2]/a[3]'  
 show\_appid = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr[1]/td[3]/a/span'  
 show\_appkey = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr[1]/td[4]/a/span'  
 show\_secretkey = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr[1]/td[5]/a/span'  
 mod = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr[1]/td[9]/a[1]'  
 # delete = By.XPATH,'//\*[@id="dataGrid"]/tbody/tr[1]/td[9]/a[2]'  
 # csjghs = By.XPATH,'/html/body/div[1]/section/div[1]/div[2]/div[4]/div[1]/span[1]'  
 # cx = By.ID, 'queryBtn'  
  
#新建第三方公司页面对象层  
  
#业务层  
 def gsmc\_cx(self,driver,gsmc\_txt,sql=None):  
 *'''公司名称查询'''* self.send\_text(driver,self.gsmc,gsmc\_txt)  
 self.Click(driver,self.cx)  
 time.sleep(1)  
 try:  
 hsz = self.get\_cx\_rown(driver)  
 assert hsz > 0  
 except:  
 assert self.get\_text(driver, self.wsjzs) == '没有找到匹配的记录'  
 else:  
 sql\_hsz = result\_verify(sql).rowcount()  
 assert hsz == sql\_hsz  
  
#第方公司管理unittest封装  
import unittest  
class dsfgs\_unit(unittest.TestCase,dsfgs\_object):  
 @classmethod  
 def setUpClass(self):  
 wd = commons().login()  
 commons().Click(wd, self.dsfgs)  
 time.sleep(1)  
 commons().Click(wd, self.gslb)  
 time.sleep(1)  
 wd.switch\_to\_frame('mainFrame') # 切换到iframe标签  
 self.driver = wd  
 @classmethod  
 def tearDownClass(self):  
 self.driver.quit()

### report模块

用于存放测试用例执行后的html报告

### run模块生成html测试报告

用来执行批量测试用例并生成html测试报告，脚本源码：

import unittest  
import time  
from HTMLTestRunner import HTMLTestRunner  
import os  
curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
rootPath = os.path.split(curPath)[0] + '\\'  
test\_dir = rootPath+'test\_case' #  
discover = unittest.defaultTestLoader.discover(test\_dir,pattern= 'test\*.py')  
if \_\_name\_\_ == '\_\_main\_\_':  
 now = time.strftime('%Y-%m-%d %H\_%M\_%S')  
 filename = rootPath+'report/'+ now + '巧匠云后台功能自动化测试报告.html'  
 fp = open(filename,'wb')  
 runner = HTMLTestRunner(stream=fp,  
 title='巧匠云后台功能自动化测试',  
 description='用例执行情况')  
 runner.run(discover)

### test\_case用例模块

每个功能模块建一个文件夹存放该模块下的所有接口

某一个用例文件的代码：

from page\_object.dsfgs\_po import dsfgs\_object,dsfgs\_unit  
import time  
# from config.readconfig import \*  
from common.sql\_result\_verify import result\_verify  
class cx\_dsfgs(dsfgs\_unit,dsfgs\_object):  
 *'''查询第三方公司'''* def test01(self):  
 *'''默认全查'''* jg = self.cx\_alldata(self.driver,'select \* from companies')  
 self.assertEqual(jg[0],jg[1])  
 def test02(self):  
 *'''公司名称查询:查询存在的公司'''* self.gsmc\_cx(self.driver,'133',"select \* from companies where company\_name like '%133%'")  
  
 def test03(self):  
 *'''公司名称查询:查询不存在的公司'''* self.gsmc\_cx(self.driver,'金融界合同日进内部')  
  
 def test04(self):  
 *'''验证重置按钮'''* self.send\_text(self.driver,self.gsmc,'123')  
 self.datatime\_cl(self.driver,'2019-05-06')  
 self.Click(self.driver,self.sjcz) #点击重置按钮  
 gsmc = self.get\_text(self.driver,self.gsmc)  
 rq = self.driver.find\_element\_by\_id('createTime')  
 print(gsmc,rq)  
 self.assertNotEqual(gsmc,'123')  
 self.assertNotEqual(rq,'2019-05-06')  
  
 def test05(self):  
 *'''时间查询'''* self.datatime\_cx(self.driver,'2019-05-10',"select \* from companies where create\_time like '%2019-05-10%'")  
  
 def test06(self):  
 *'''时间查询不出数据'''* self.Click(self.driver,self.sjcz)  
 self.datatime\_cx(self.driver,'2022-02-03')  
 def test07(self):  
 *'''时间和公司名称组合查询'''* self.Click(self.driver, self.sjcz)  
 self.send\_text(self.driver,self.gsmc,'lb')  
 self.datatime\_cl(self.driver,'2019-03-28')  
 self.Click(self.driver,self.cx)  
 time.sleep(2)  
 rown = self.get\_cx\_rown(self.driver)  
 sql\_rown = result\_verify("select \* from companies where company\_name like '%lb%' and create\_time like '%2019-03-28%'").rowcount()  
 self.assertEqual(rown,sql\_rown)

# 项目实战注意的问题

## 命令行与自动化部署

用到路径的地方必须用项目绝对路径，这样在命令行和自动化部署时才能正常运行项目，不然会报XXX模块找不到

绝对路径代码：

curPath = os.path.abspath(os.path.dirname(\_\_file\_\_))  
rootPath = os.path.split(curPath)[0] + '\\' #直接获取项目根路径，不用指定项目名

需要在电脑环境变量中添加PYTHONPATH环境变量，变量值为项目绝对路径

如：E:\api\_auto\Jupiter-T\Api\_Automation\_Framework\;

E:\LB\Api\_Automation\Jupiter-T\QJY\_OPEN\_API\;

C:\Python37-32;

上面我添加了2个项目，注意【\;】这2个东西必须有，就算只有一个项目也必须加分号，后面再加上python安装跟目录路径，不然只能在python安装跟目录下才能执行程序

## 命令行运行项目里的某一个py文件

首先先按照命令行与自动化部署完成本地环境的配置，其次在项目里的脚本文件后面必须加上，入下图所示，这样在命令行才能运行脚本文件

if \_\_name\_\_=='\_\_main\_\_':  
 suit = unittest.TestSuite()  
 suit.addTest(runtest('test\_api'))  
 runner = unittest.TextTestRunner()  
 runner.run(suit)

## bat文件执行脚本

新建bat文件，代码如下

chcp 65001

python D:\git\_script\Jupiter-T\qjyApi\run\_script\接口临时调试\NewPub健康码接口.py

pause

注释：

chcp 65001 让中文不乱码

python 执行文件主题

pause 让窗口停留，不然会一闪而过

# pyqt5

ui界面开发工具，使用designer.exe

路径位于：\Python37\Lib\site-packages\qt5\_applications\Qt\bin

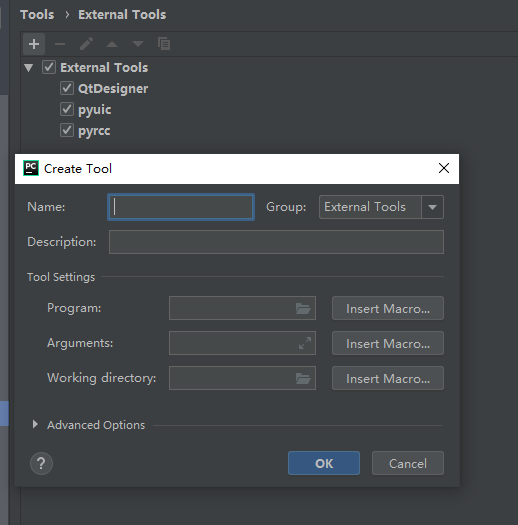
需要下载2个工具包

1. pip install pyqt5
2. pip install pyqt5-tools

安装pyqt5-tools时会自动安装qt designer

## Pycharm配置外部工具执行qt

打开PyCharm，选择Settings -> Tools -> External Tools，点击左上角的绿色加号。



* QtDesigner添加

Name填入QtDesigner（方便后续使用，名称无所谓）。Program选择我们安装的PyQt5-tools下面的designer.exe。Working directory则选择我们的工作目录。然后点击OK，则添加了QtDesigner作为PyCharm的外置工具。

* PyUIC添加

然后添加PyUIC（UI转换工具），PyUIC的Program为Python.exe，Working directory同理设为我们的工作目录，Arguments则填入如下代码：

-m PyQt5.uic.pyuic $FileName$ -o $FileNameWithoutExtension$.py

* Pyrcc添加

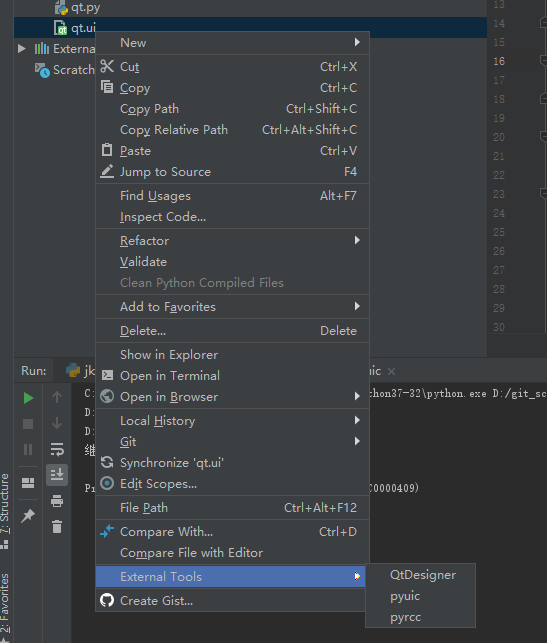
用于PyQt5的资源文件转码。具体配置与PyUIC添加内容相同，Arguments填入

$FileName$ -o $FileNameWithoutExtension$\_rc.py

### 外部工具执行方法

如下图：右键ui文件-External Tools，里面有三个工具，即上面添加的三个工具

1. QtDesigner：启动QtDesigner
2. pyuic：将ui文件转换为.py文件，py文件的名称转换后和ui文件名称一致，转换后py文件保持在当前路径下
3. pyrcc：使用中没用到



## Qt designer常用函数

* setEnabled(False)：将控件对象置为不可编辑
* setPlaceholderText()：设置控件提示文字
* text()：获取line edit控件对象文本输入值
* toPlainText()：获取text edit控件对象文本的值
* clear()：清除lable控件数据
* currentText()：获取下拉框的值
* isChecked()：获取控件对象是否被选中，值为boole
* setText('接口执行中,请等待')： setText()函数设置界面显示，一般应用于text Browser控件对象，line edit对象也适用
* setPlainText(‘设置显示文本’)：设置text edit控件的值
* QApplication.processEvents()：在程序中展示其它状态时立即执行函数，比如点击登录按钮后马上变灰，在设置操作之后加上这个函数即可，如果不加这个函数，程序将会在运行完后才会去执行

## Qt designer常用控件

* Label：文本标签，显示文本，可以用来标记控件
* Text Browser：显示文本控件，用于后台命令执行结果显示
* Line Edit：单行文本框，输入单行字符串。控件对象常用函数为Text() 返回文本框内容，用于获取输入。setText() 用于设置文本框显示。
* Push Button：命令按钮。常见的确认、取消、关闭等按钮就是这个控件。clicked信号一定要记住。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作。

Push Button.clicked.connect(self.方法)：提交按钮监听触发事件

* Radio Button：单选按钮
* Check Box：多选按钮
* Combo Box：下拉框列表

Combo Box .currentIndexChanged.connect(self.方法)：下拉框监听触发事件

## Qt designer常用操作

### 改变按钮颜色

右键控件-改变样式表-添加颜色-color为字体颜色，background为背景色

### 上传本地照片

1. 先放一个Push Button按钮，取名图片上传
2. 再放一个label，拉到2寸图片大小，用来展示上传的图片，不要值
3. 写代码，照着下面写即可实现

导入包：

#PyQt5中使用的基本控件都在PyQt5.QtWidgets模块中  
from PyQt5.QtWidgets import QApplication, QMainWindow,QFileDialog  
from PyQt5 import QtGui #上传图片用

import sys

self.imgupdata.clicked.connect(self.openimage)

#添加信号和槽。注意要执行的函数不加小括号()，必须在这里写了事件才会触发页面流程，  
#clicked信号一定要记住。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作

def openimage(self):  
 imgName, imgType = QFileDialog.getOpenFileName(self, "打开图片", "", "\*.jpg;;\*.png;;All Files(\*)") #打开选择图片窗口  
 jpg = QtGui.QPixmap(imgName).scaled(self.photo\_label.width(), self.photo\_label.height()) #让图片适应lable控件大小  
 self.photo\_label.setPixmap(jpg) #展示到控件上  
 print('imgname',imgName)  
 import base64  
 url = imgName #图片路径  
 img = open(url, 'rb') # 打开图片  
 base64\_data = base64.b64encode(img.read()) # 读出图片并转换为base64  
 s = base64\_data.decode() # 转换为decode  
 print('one:',s)  
 self.basimg = s # self.basimg为变量，在创建类的时候定义了初始值为basimg=’’,执行了该函数后给basimg赋值

### 多窗口，从一窗口跳转另一窗口

1. 先画2个窗口，保存2个ui文件并转py
2. 正常写引用两个py文件的业务代码
3. 从A跳转到B窗口的示列代码实现
4. import sys  
   from PyQt5.QtWidgets import QApplication, QMainWindow,QFileDialog #工具常用的包  
   from dck import Ui\_MainWindow #导入主窗口  
   from dck2 import Ui\_Form #导入待跳转窗口  
   class qq(QMainWindow,Ui\_Form):  
    *'''待跳转窗口'''* def \_\_init\_\_(self, parent=None):  
    super(qq, self).\_\_init\_\_(parent)  
    self.setupUi(self)  
    #添加信号和槽。注意要执行的函数不加小括号()，必须在这里写了事件才会触发页面流程，  
    #clicked信号一定要记住。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作  
   class MyMainForm(QMainWindow,Ui\_MainWindow):  
    *'''主窗口'''* def \_\_init\_\_(self, parent=None):  
    super(MyMainForm, self).\_\_init\_\_(parent)  
    self.setupUi(self)  
    #添加信号和槽。注意要执行的函数不加小括号()，必须在这里写了事件才会触发页面流程，  
    #clicked信号一定要记住。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作  
    self.tz.clicked.connect(self.tt) #触发跳转操作  
    def tt(self):  
    mains.show() #将待跳转的窗口显示到页面上  
    myWin.close() #关闭主窗口  
   if \_\_name\_\_ == "\_\_main\_\_":  
    #固定的，PyQt5程序都需要QApplication对象。sys.argv是命令行参数列表，确保程序可以双击运行  
    app = QApplication(sys.argv)  
    #初始化  
    myWin = MyMainForm()  
    mains = qq() #实例化跳转后的窗口，必须在这里实例化，不然窗口会闪一下关闭  
    #将窗口控件显示在屏幕上  
    myWin.show()  
    #程序运行，sys.exit方法确保程序完整退出。  
    sys.exit(app.exec\_())

* 错误弹框-警示框

from PyQt5.QtWidgets import QApplication, QMainWindow,QFileDialog #工具常用的包

from PyQt5 import QtGui, QtWidgets

QtWidgets.QMessageBox.warning(self, "警告", "图片Base64必须以/9j开头")

## 写的工具代码



## Qt designer运行实例

照着下面框架写，我们只需要改变点击按钮要执行的函数即可，执行函数为我们自己的业务函数

import sys  
#PyQt5中使用的基本控件都在PyQt5.QtWidgets模块中  
from PyQt5.QtWidgets import QApplication, QMainWindow,QFileDialog #工具常用的包  
from PyQt5 import QtGui, QtWidgets #上传图片用  
from qt import Ui\_Form #qt为自己画的页面，转换成的qt.py文件，自定义的名字，需要把类导入进来  
  
class MyMainForm(QMainWindow, Ui\_Form):  
 def \_\_init\_\_(self, parent=None):  
 super(MyMainForm, self).\_\_init\_\_(parent)  
 self.setupUi(self)  
 #添加信号和槽。注意要执行的函数不加小括号()，必须在这里写了事件才会触发页面流程，  
 #clicked信号一定要记住。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作  
 self.imgupdata.clicked.connect(self.openimage) #上传图片按钮  
 self.gbck.clicked.connect(self.gbzck) #关闭窗口按钮  
  
 def gbzck(self):  
 self.close() #关闭当前窗口  
 # sys.exit(0) #关闭所有窗口  
  
 def openimage(self):  
 *'''上传图片并转为base64'''* imgName, imgType = QFileDialog.getOpenFileName(self, "打开图片", "", "\*.jpg;;\*.png;;All Files(\*)") #打开选择图片窗口  
 try:  
 jpg = QtGui.QPixmap(imgName).scaled(self.photo\_label.width(), self.photo\_label.height()) #让图片适应lable控件大小  
 self.photo\_label.setPixmap(jpg) #展示到控件上  
 print('imgname',imgName)  
 import base64  
 url = imgName #图片路径  
 img = open(url, 'rb') # 打开图片  
 base64\_data = base64.b64encode(img.read()) # 读出图片并转换为base64  
 s = base64\_data.decode() # 转换为decode  
 # print('one:',s)  
 self.basimg = s  
 except:  
 return  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 #固定的，PyQt5程序都需要QApplication对象。sys.argv是命令行参数列表，确保程序可以双击运行  
 app = QApplication(sys.argv)  
 #初始化  
 myWin = MyMainForm()  
 #将窗口控件显示在屏幕上  
 myWin.show()  
 #程序运行，sys.exit方法确保程序完整退出。  
 sys.exit(app.exec\_())

## 打包成exe文件

* 安装PyInstalle

pip install pyinstaller

* 安装wheel

pip install wheel

注意：打包出问题都是python环境问题或者site-packages下有对应包导致安装不完整，常见的问题出现找不到模块、提示pefile不存在等

解决方法：

1. 从新安装python，使用默认路径，把原来的python文件夹全部改名，避免后续安装到其它位置
2. 删除site-packages下的包，从新使用pip安装

### PyInstalle使用



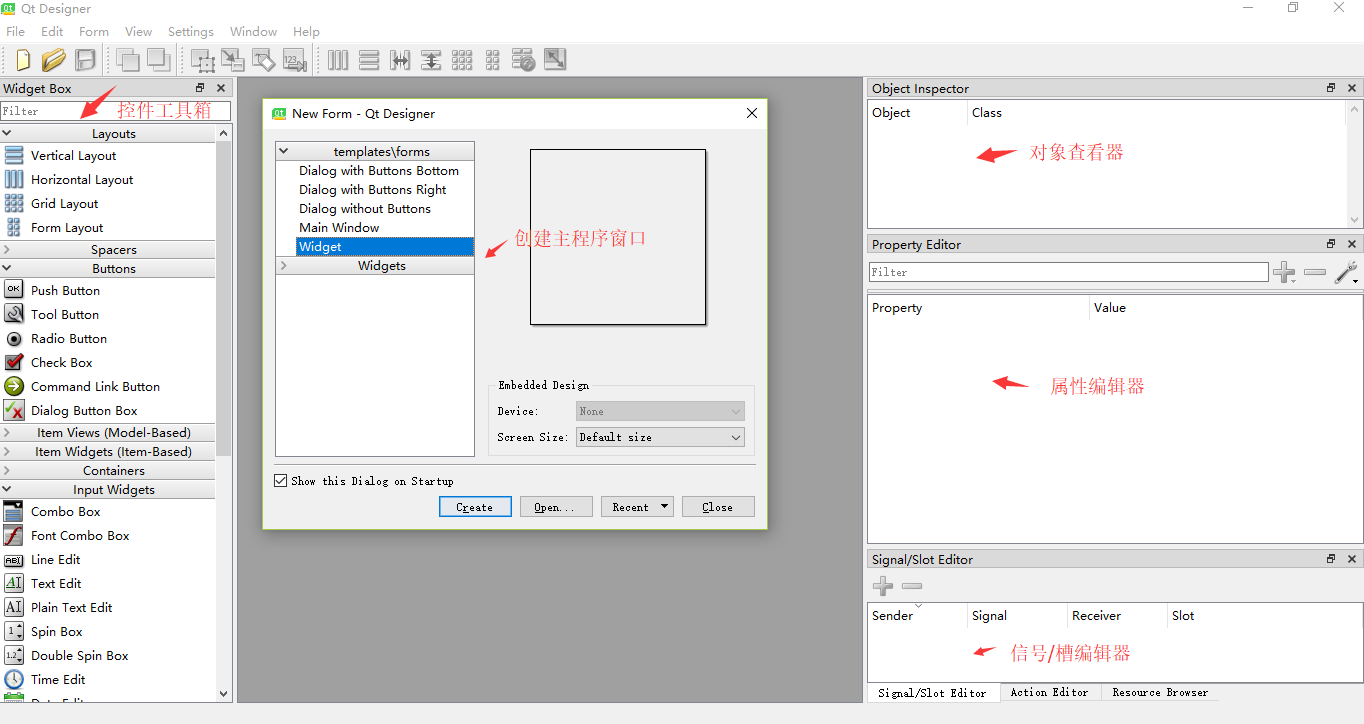
* 使用方法

pyinstaller -F XXX.py

## Designer网上教程

### Qt Designer工具主界面

  打开路径：${python安装目录}/Lib/site-packages/pyqt5\_tools/designer.exe。主界面如下：



 主界面不同区域介绍：

**工具箱 区域**：提供GUI界面开发使用的各种基本控件，如单选框、文本框等。可以拖动到新创建的主程序界面。

**主界面区域**：用户放置各种从工具箱拖过来的各种控件。模板选项中最常用的就是Widget（通用窗口）和MainWindow（主窗口）。二者区别主要是Widget窗口不包含菜单栏、工具栏等。可以分别创建对比看看。

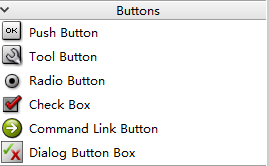
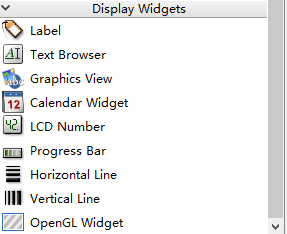
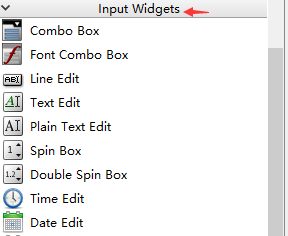
**对象查看器 区域**：查看主窗口放置的对象列表。

**属性编辑器 区域**： 提供对窗口、控件、布局的属性编辑功能。比如修改控件的显示文本、对象名、大小等。

**信号/槽编辑器 区域**：编辑控件的信号和槽函数，也可以添加自定义的信号和槽函数。

### Qt Designer基本控件介绍

  Widget Box控件工具箱是按照控件作用类别进行划分的。这里作为实现入门级界面实现，主要介绍最常使用的控件及控件对象相关函数。函数方法知道怎么获取控件输入内容以及如何将后台操作结果输出到界面控件显示的主要函数就可以了。



  （1）显示控件。

**Lable**：文本标签，显示文本，可以用来标记控件。

**Text Browser**：显示文本控件。用于后台命令执行结果显示。

  （2）输入控件，提供与用户输入交互

**Line Edit**：单行文本框，输入单行字符串。控件对象常用函数为Text() 返回文本框内容，用于获取输入。setText() 用于设置文本框显示。

**Text Edit**：多行文本框，输入多行字符串。控件 对象常用函数同Line Edit控件。

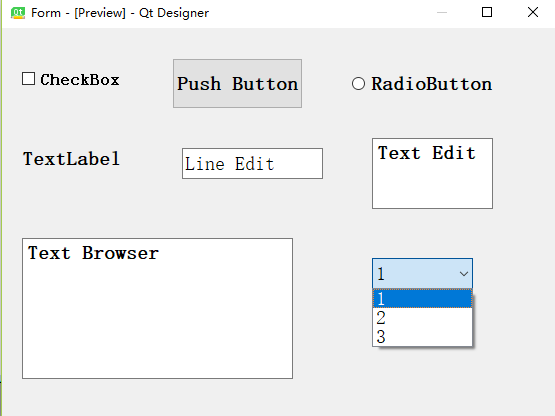
**Combo Box**：下拉框列表。用于输入指定枚举值。

  （3）控件按钮，供用户选择与执行

**Push Button**：命令按钮。常见的确认、取消、关闭等按钮就是这个控件。**clicked信号一定要记住**。clicked信号就是指鼠标左键按下然后释放时会发送信号，从而触发相应操作。

**Radio Button**：单选框按钮。

**Check Box**：多选框按钮。

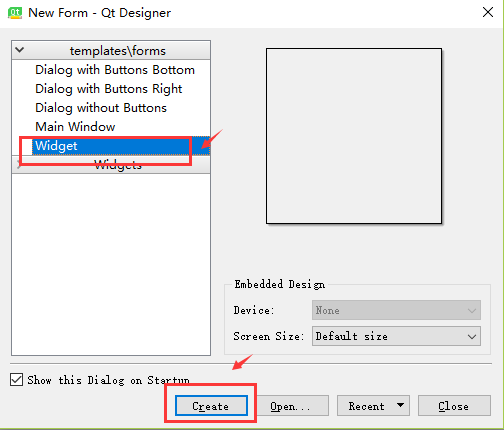


### Qt Designer工具实现

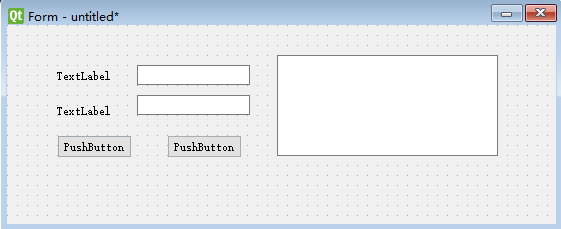
  了解基本控件及作用和获取输入/显示方法后，就可以开始动手实现小需求了。。比如登录界面。获取用户名和密码并显示。。

  打开Qt Designer，开始拖拽控件实现吧。。

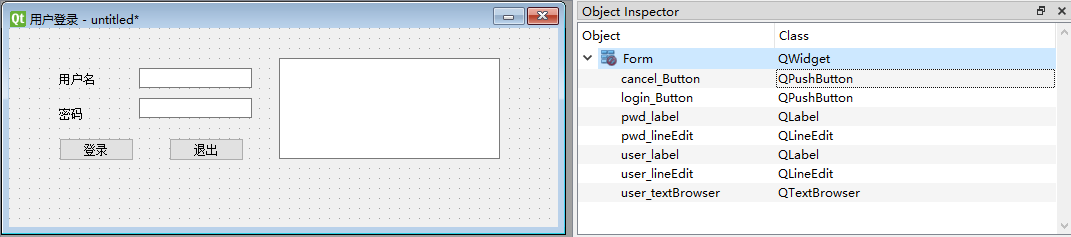
**Step1**：打开主界面，选择Widget模板



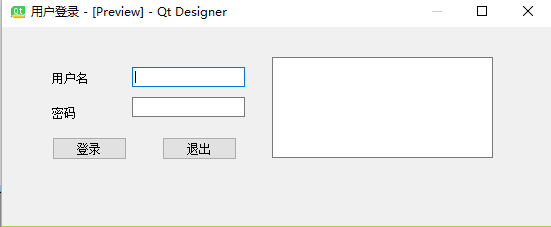
**Step2**：从Widget Box工具箱中拖拽2个label、2个line Edit、2个Push Button以及1个Text Browser。拖完后如下：



**Step3**：双击各个控件，修改控件名称（对应属性编辑区中的text，可直接双击控件修改）以及对象名称（对应属性编辑区中的objectName）。对象名称一定记得修改。默认生成的label\_1、label\_2这种名称无法直接判断到底是对应哪个控件。。



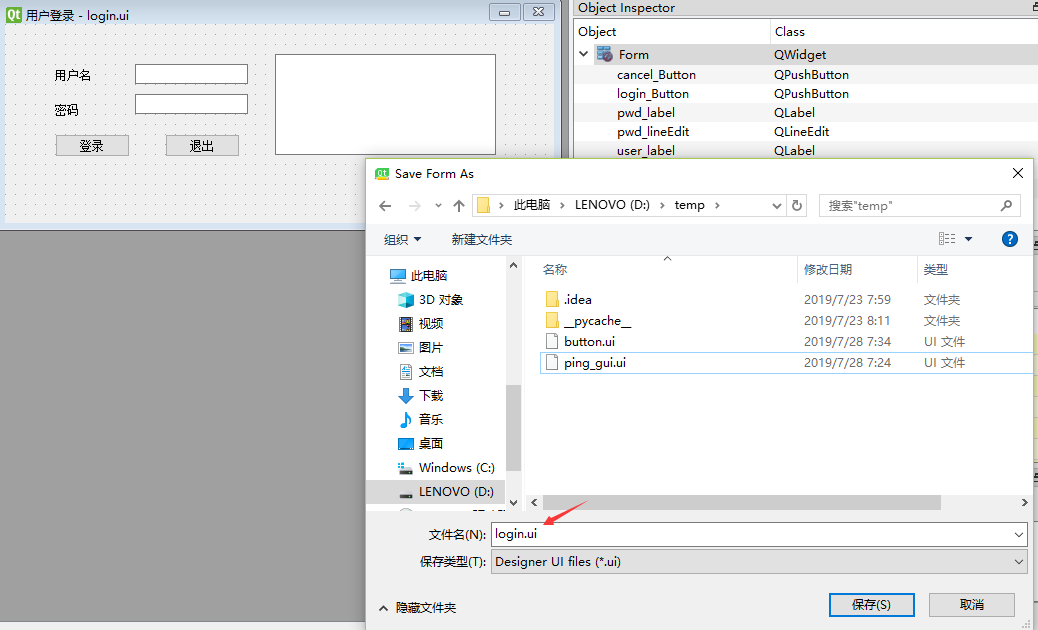
  点击菜单栏Form - Prview。预览界面实现效果



 login.ui的程序代码如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

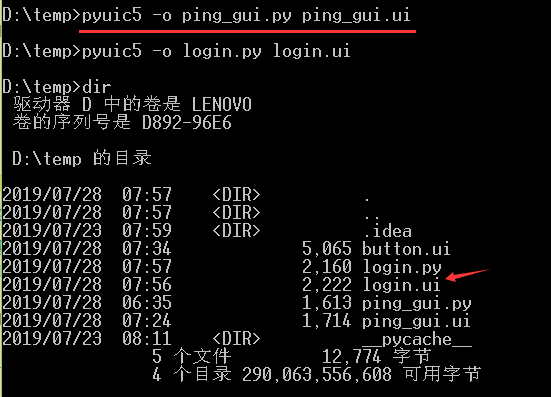
**Step4**：点击File -Save保存实现结果。保存文件名为login.ui。



**Step5：**界面开发完成。

### 将.ui文件转换为.py文件

  使用命令行**pyuic5 -o login.py login.ui**转换成.py文件。调用格式为**pyuic5 -o {输出文件名} {输入designer设计好的.ui后缀界面文件}**。执行结果如下



转换后的.py文件内容如下：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif View Code

**界面与业务逻辑分离实现**

  这一步主要实现业务逻辑，也就是点击登录和退出按钮后程序要执行的操作。为了后续维护方便，采用界面与业务逻辑相分离来实现。也就是通过创建主程序调用界面文件方式实现。这有2个好处。第1就是实现逻辑清晰。第2就是后续如果界面或者逻辑需要变更，好维护。新建call\_login.py文件程序，调用login.py文件。

[复制代码](javascript:void(0);)

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'connect\_me.ui'

#

# Created by: PyQt5 UI code generator 5.11.3

#

# WARNING! All changes made in this file will be lost!

#导入程序运行必须模块

import sys

#PyQt5中使用的基本控件都在PyQt5.QtWidgets模块中

from PyQt5.QtWidgets import QApplication, QMainWindow

#导入designer工具生成的login模块

from login import Ui\_Form

class MyMainForm(QMainWindow, Ui\_Form):

def \_\_init\_\_(self, parent=None):

super(MyMainForm, self).\_\_init\_\_(parent)

self.setupUi(self)

if \_\_name\_\_ == "\_\_main\_\_":

#固定的，PyQt5程序都需要QApplication对象。sys.argv是命令行参数列表，确保程序可以双击运行

app = QApplication(sys.argv)

#初始化

myWin = MyMainForm()

#将窗口控件显示在屏幕上

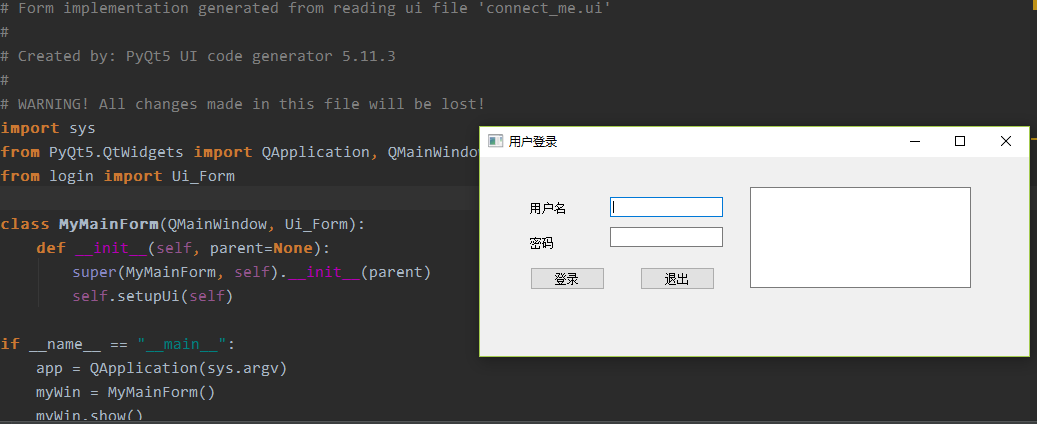
myWin.show()

#程序运行，sys.exit方法确保程序完整退出。

sys.exit(app.exec\_())

[复制代码](javascript:void(0);)

运行call\_login.py程序，结果如下：



  到这里，界面实现和业务主程序已经写好了。但是现在具体业务功能逻辑还未实现。需要对登录和退出的按钮点击执行相对应的操作。

### 添加信号和槽，实现业务逻辑

  实现部分见代码注释。这里主要添加如下两行命令配置信号和槽的关系。信号和槽的创建和原理下文描述。这里可以参照添加即可。

   登录按钮：self.login\_Button.clicked.connect(self.display)

   退出按钮：self.cancel\_Button.clicked.connect(self.close)

  详细代码如下：

[复制代码](javascript:void(0);)

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'connect\_me.ui'

#

# Created by: PyQt5 UI code generator 5.11.3

#

# WARNING! All changes made in this file will be lost!

#导入程序运行必须模块

import sys

#PyQt5中使用的基本控件都在PyQt5.QtWidgets模块中

from PyQt5.QtWidgets import QApplication, QMainWindow

#导入designer工具生成的login模块

from login import Ui\_Form

class MyMainForm(QMainWindow, Ui\_Form):

def \_\_init\_\_(self, parent=None):

super(MyMainForm, self).\_\_init\_\_(parent)

self.setupUi(self)

#添加登录按钮信号和槽。注意display函数不加小括号()

self.login\_Button.clicked.connect(self.display)

#添加退出按钮信号和槽。调用close函数

self.cancel\_Button.clicked.connect(self.close)

def display(self):

#利用line Edit控件对象text()函数获取界面输入

username = self.user\_lineEdit.text()

password = self.pwd\_lineEdit.text()

#利用text Browser控件对象setText()函数设置界面显示

self.user\_textBrowser.setText("登录成功!\n" + "用户名是: "+ username+ ",密码是： "+ password)

if \_\_name\_\_ == "\_\_main\_\_":

#固定的，PyQt5程序都需要QApplication对象。sys.argv是命令行参数列表，确保程序可以双击运行

app = QApplication(sys.argv)

#初始化

myWin = MyMainForm()

#将窗口控件显示在屏幕上

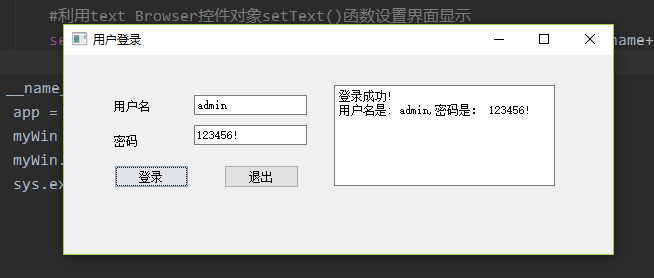
myWin.show()

#程序运行，sys.exit方法确保程序完整退出。

sys.exit(app.exec\_())

[复制代码](javascript:void(0);)

运行结果如下：



# 常用的包site-packages

* requests:接口库
* appium：app自动化库
* certifi：证书
* pycryptodomex：AES加解密
* ddt：数据驱动
* pyInstaller：文件打包成exe
* pymysql：mysql数据库
* PyQt5：界面开发
* wheel：打包关联模块
* xlrd：excel读文件
* xlutils：复制excel文件
* xlwt：写exce文件