# 模块化驱动脚本开发

```
业务代码:
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import MyseleniumTools
import time
# 场景:打开百度首页,获取百度热搜中第一个新闻标题,然后把新闻标题复制到点开新闻链接页面的搜索框中,\
# 然后点击百度一下,获取相关资讯数量,然后把这个数量填写到QQ邮箱登录页面的QQ号码输入框中

'''
1.打开浏览器进入指定网站模块
2.获取元素模块
3.操作元素模块
'''
tools=MyseleniumTools.Maker()
dr=tools.openHtml("https://www.baidu.com")
'''
By.ID
By.NAME
By.LINK_TEXT
By.PARTIAL_LINK_TEXT
By.TAG_NAME
By.CLASS_NAME
By.CSS_SELECTOR
By.XPATH
'''
e=tools.getELE(dr,'By.PARTIAL_LINK_TEXT',"凝聚团结共识 促进人类进步")
#获取元素标签对之间的内容
text=tools.operateELE(e,'text')
print(text)
#点击新闻链接
e=tools.getELE(dr,'By.LINK_TEXT','新闻')
tools.operateELE(e,'click')
time.sleep(3)
#切换窗口
tools.MySwitch(dr,1)

#获取元素
e=tools.getELE(dr,'By.ID',"ww")
#输入内容到新闻搜索框
tools.operateELE(e,'send_keys',text)
time.sleep(2)
#获取百度一下按键
```

```
e=tools.getELE(dr,"By.ID",'s_btn_wr')
tools.operateELE(e,'click')
time.sleep(2)

#获取头部资讯信息文字
e=tools.getELE(dr,'By.XPATH','//div[@id="header_top_bar"]/span[1]')
text=tools.operateELE(e,"text")
print(text)

mytext=text[-3:-1]
print(mytext)

#打开qq邮箱登录页面
dr.get("https://mail.qq.com/")
time.sleep(2)

#获取第一个内层的iframe元素
e=tools.getELE(dr,'By.CLASS_NAME','QQMailSdkTool_login_loginBox_qq_iframe')
#进入第一个内层
tools.MySwitchFrame(dr,e)
time.sleep(2)
#进入第二个内层
tools.MySwitchFrame(dr,'ptlogin_iframe')
#点击密码登录链接
e=tools.getELE(dr,'By.ID','switcher_plogin')
tools.operateELE(e,"click")#点击链接
time.sleep(3)
#获取输入框元素
e=tools.getELE(dr,'By.ID','u')
#操作元素
tools.operateELE(e,"send_keys",mytext)#点击链接

time.sleep(5)
dr.quit()


功能代码:
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
from selenium import webdriver
import time
from selenium.webdriver.common.by import By
class Maker():
    #打开浏览器并进入到指定网站
    def openHtml(self,url):
        dr=webdriver.Firefox()

        dr.get(url)
```

```
        time.sleep(2)
        return dr
    #获取元素
    def getELE(self,dr,type,msg):
        if type=="By.ID":
            e=dr.find_element(By.ID,msg)
        elif type=='By.LINK_TEXT':
            e = dr.find_element(By.LINK_TEXT, msg)
        elif type=='By.PARTIAL_LINK_TEXT':
            e = dr.find_element(By.PARTIAL_LINK_TEXT, msg)
        elif type=='By.XPATH':
            e=dr.find_element(By.XPATH,msg)
        elif type=='By.CLASS_NAME':
            e=dr.find_element(By.CLASS_NAME,msg)
        return e


    #操作元素
    def operateELE(self,e,type,msg=""):
        if type=="text":
            text=e.text
            return text
        elif type=="click":
            e.click()
        elif type=="send_keys":
            e.send_keys(msg)

    #切换窗口
    def MySwitch(self,dr,index):
        #获取浏览器所有的句柄
        mylist = dr.window_handles
        dr.switch_to.window(mylist[index])

    #切换到frame中,如果value是字符串,那么就表示是iframe的ID或name的值,
    #如果不是字符串,那么就是元素
    def MySwitchFrame(self,dr,value):
        dr.switch_to.frame(value)
```

# 数据驱动脚本开发

```
1 ) excel或csv数据读取
2 ) 数据驱动脚本原理
3) 数据驱动脚本开发

数据驱动测试：
由数据的改变,驱动测试的执行，最终改变测试的结果，这种方式就是数据驱动测试
数据与业务的分离，分离出来的数据应该怎么存储：
字典：
    数据量比较小，只有几个的时候，使用的频率还挺高
    在python定义一个字典类型的数据结构，把数据存进去
```

```
dictData =
[{"username":"","email":"nz1903_0000@163.com","password":"123456","repassword":"123456","expect":"请设置用户名"},


{"username":"nz19","email":"nz1903_0000@163.com","password":"123456","repassword":"123456","expect":"用户名不符合格式要求"},


{"username":"nz1903_01001001012","email":"nz1903_0000@163.com","password":"123456","repassword":"123456","expect":"用户名不符合格式要求"}]

csv文件（excel文件）：
    数据量比较大，几十~几万条数据，使用频率不算太高，常用测试数据的存储
    写在一个外部文件中，用的时候读取出来即可

数据库：
    数据非常大，几万条以上，使用频率较低，如基础类数据
    直接存储在数据库，通过数据库读写模块，直接获取想要的数据使用即可
配置文件：
    数据库连接，数据库主机地址、数据库用户名、密码、端口等
    用于更改数据库环境，比如直接在配置文件中更改数据库的用户名，你的脚本是不需要变化
数据的参数化：
    其实就是python中的知识，就是用变量替换常量的过程
    csv文件存储数据进行数据驱动（熟练使用）：
    CSV文件数据读取：
        1、新建一个excel文件，写入测试数据
        2、将文件另存为utf都逗号分隔符的csv文件
        3、将csv文件转码为utf-8格式
        4、在python代码中导入数据
            导包
            使用with+open打开文件
            使用data = csv.reader(f)
            使用for循环遍历该数据
        5、文件路径问题,两种相对路径的方式（不建议使用绝对路径）
            ./20200403/data_csv.csv
            获取当前编辑文件的目录：
                os.path.dirname(__file__)
                os.path.dirname(os.path.dirname(__file__))

案例:用xls存储数据的方式驱动脚本开发
excel存储数据进行数据驱动（熟练使用）：
    先安装xlrd模块：cmd  ->  pip3 install xlrd
从excel文件中读入数据：
    1、导包，import xlrd
    2、使用xlrd的方法打开excel文件（创建一个文件对象）
    3、获取excel文件的sheet页
    4、获取sheet页中的行数据、列数据、单元格数据
    5、需要遍历数据，你先要直到文件中有多少行、多少列数据
    6、使用for循环遍历
```

# 检查点

```
    严格意义上讲，之前所写的自动化测试脚本不能叫测试用例，真正的测试用例是需要做检查（断言和验证）的，一定要有预期结果与实际结果进行比较的过程。在功能测试用例执行过程中，检查是由测试人员通过眼睛来完成；在自动化测试脚本中可以通过判断语句以及工具或框架中提供的断言与验证方法来实现。
    在使用断言的时候，注意异常的处理，如果不处理，可能会导致pvm退出，后续的脚本或者代码不能执行
只有使用try--except--finally这种语法处理，保证后续代码或脚本的正常运行


```

# Unittest测试框架介绍(重点)

```
1.介绍
之前脚本存在的问题：
    在测试用例的执行的时候，发现挺麻烦（如果测试用例数量过大，要么挨个执行，要么导包执行）
    断言方式太low了，只能是在控制台打印测试用例是否通过，我们希望这个断言的状态是显示在报告中的，而且不应该通过if-else去判断。
    基本上看不到测试报告的效果，显示总共执行了多少条用例、通过了多少条、失败了多少条、失败的原因是什么？
怎么办呢？引入框架...
web自动化：python + selenium + unittest
unittest：单元测试框架，有很多好的特性，在自动化测试中是可以借鉴的
2.好处
    提供用例组织与执行
    提供丰富的断言方法
    提供丰富的日志和报告（HTML格式的更好一些）

3.原理
1）TestCase：
在unittest中的一个TestCase的实例就是一个测试用例，就是一个完整的测试流程，包括测试前资源初始化（setUp），执行测试代码（test），以及测试后环境的还原（tearDown）。
2）TestSuite
测试套件，可以理解为：多个独立的测试用例（test case）或者多个独立的测试套件（test suite，可以理解为子套件）可以构成一个测试套件，然后传递给TestRunner进行测试执行。，内容也有run函数可以执行测试
3）TestLoader
通过unittest.TestLoader类的loadTestsFromTestCase、loadTestsFromModule、LoadTestsFromName、discover方法，可以将测试用例添加一个测试套件中。
4）TestRunner
可以理解为测试集的运行器，可以在其基础上扩展子类TextTestRunner或者HTMLTestRunner，只不过生成的测试报告样式不同，此处讲解TextTestRunner，后续课程再扩展HTMLTestRunner。
5、TestResult
测试结果类，用来处理测试用例或测试集执行过程中的所有信息并最终输出,比如代码错误、异常、断言失败、skip等等。
```

# pythonIDLE执行代码

```
因为pycharm的缺陷,我们需要在pythonIDLE执行代码
看pythonIDLE执行代码文档
```

# Unittest测试框架运行说明(重点)

```
步骤:
1.导包，unittest是自带的框架，不需要安装
2.创建一个单元测试类（其实就是类，只不过他继承了单元测试框架单元测试用例的类）
3.执行

单元测试类中的方法说明:
1.setUpClass：给当前单元测试类的所有的用例进行初始化的,是类方法
2.tearDownClass：给当前单元测试类的所有的用例进行资源释放,是类方法
3.setUp()：主要是进行测试用例的资源初始化，测试用例的前提条件写在这
4.test_xxx()：测试用例，要把测试用例的步骤写在这个方法中,注意要test开头,是规定
5.tearDown()：主要是进行测试用例的资源释放的

执行顺序说明:
1.先执行setUpClass
2.setUp()、test_xxx()、tearDown(),不管你怎么调整为，执行顺序不变
3.最后执行tearDownClass
4.每执行一个测试用例,2都要执行一遍

区别说明:
1.setUpClass和setUp()的区别：
    setUp()不需要@classmethod注解；setUpClass方法需要@classmethod注解
    setUp()实例方法，就需要创建对象再调用；setUpClass类方法，不需要对象也可以调用
    setUp()再每一个测试用例执行之前运行一次；setUpClass方法在测试执行之前只执行一次
    setup()是对一条测试用例的初始化；setUpClass()给当前单元测试类的所有的用例进行初始化的
2.tearDownClass和tearDown的区别:
    tearDown()不需要@classmethod注解；tearDownClass方法需要@classmethod注解
    tearDown()实例方法，就需要创建对象再调用；tearDownClass类方法，不需要对象也可以调用
    tearDown()再每一个测试用例执行之后运行一次；tearDownClass方法在测试执行之后只执行一次
    tearDown()是对一条测试用例的资源释放；tearDownClass给当前单元测试类的所有的用例进行资源释放

执行方式:
1.执行main()方法执行的特点:unittest.main()
2.有选择的执行测试用例
    1.通过测试集合内部函数添加测试用例
    2.通过模块添加执行用例


```

# 常规单元测试代码

```
#定义一个类，实现加减乘除算法
#对该类中的加减乘除进行单元测试（加法实现对不对）


```

场景:从数据库中获取2条信息,一条信息粘贴到百度首页的搜索框,一条信息粘贴到淘宝首页搜索框,测试完成之后把数据库中的数据删除,恢复测试场景

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import unittest
import pymysql
from selenium import webdriver
import time
# 场景:从数据库中获取2条信息,一条信息粘贴到百度首页的搜索框,\
# 一条信息粘贴到淘宝首页搜索框,测试完成之后把数据库中的数据删除,恢复测试场景
#这里继承了TestCase,那么你的类就变为单元测试类,里面有一些方法可以直接使用
class Maker(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print("setUpClass...给所有用例进行初始化工作的..")
        globals()['data']=0
        db = pymysql.connect(host='127.0.0.1', port=3306, user="root", passwd='123456', db="mytest202305",
                             charset='utf8')
        # 创建游标
        cur = db.cursor()
        cur.execute("select name from user")
        # 获取查询的结果
        cls.data = cur.fetchall()
        print(cls.data)


    @classmethod
    def tearDownClass(cls):
        print("tearDownClass..给所有用例进行清空工作的..")
        db = pymysql.connect(host='127.0.0.1', port=3306, user="root", passwd='123456', db="mytest202305",
                              charset='utf8')
         # 创建游标
        cur = db.cursor()
        cur.execute("delete from user")

    def setUp(self):
        print("setUp....给单个测试用例初始化的")
        globals()['data']+=1
        self.dr=webdriver.Firefox()
        if globals()['data']==1:
            self.dr.get("https://www.baidu.com")
        elif globals()['data']==2:
            self.dr.get("https://www.taobao.com")
        time.sleep(2)
```

```
55
56      def tearDown(self):
57          print("tearDown....给单个用例做清理工作的")
58          time.sleep(3)
59          self.dr.quit()
60
61      #粘贴到百度首页
62      def test_01(self):
63          print("test_01测试用例")
64          print(self.data[0][0])
65          self.dr.find_element_by_id('kw').send_keys(self.data[0][0])
66
67
68      #粘贴到淘宝首页
69      def test_02(self):
70          print("test_02测试用例")
71          print(self.data[1][0])
72          self.dr.find_element_by_id('q').send_keys(self.data[1][0])
73
74
75  if __name__=='__main__':
76      unittest.main()
77
78
```