

python webdriver 项目实战

第 5 章 测试模型与测试脚本优化

第一节、测试模型介绍

线性测试

通过录制或编写脚本，一个脚本完成用户一套完整的操作，通过对脚本的回放来进行自动化测试。这是早期进行自动化测试的一种形式；我们在上一章中练习使用 webdriver API 所编写的脚本也是这种形式。

脚本一

```
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.get("http://www.xxx.com")

driver.find_element_by_id("tbUserName").send_keys("username")
driver.find_element_by_id("tbPassword").send_keys("123456")
driver.find_element_by_id("btnLogin").click()
# 执行具体用例操作
.....

driver.quit()
```

脚本二

```
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.get("http://www.xxx.com")

driver.find_element_by_id("tbUserName").send_keys("username")
driver.find_element_by_id("tbPassword").send_keys("123456")
driver.find_element_by_id("btnLogin").click()
# 执行具体用例操作
```

```
.....  
  
driver.quit ()
```

通过上面的两个脚本，我们很明显的发现它的问题：

一个用例对应一个脚本，假如界面发生变化，用户名的属性发生改变，不得不需要对每一个脚本进行修改，测试用例形成一种规模，我们可能将大量的工作用于脚本的维护，从而失去自动化的意义。

这种模式下数据和脚本是混在一起的，如果数据发生变也也需要对脚本进行修改。

这种模式下脚本的可重复使用率很低。

模块化与库

我们会清晰的发现在上面的脚本中，其实有不少内容是重复的；于是就有了下面的改进。

login.py

```
#登录模块  
  
def login():  
  
    driver.find_element_by_id(        "tbUserName"    ).send_keys(    "username"    )  
  
    driver.find_element_by_id(        "tbPassword"   ).send_keys(    "456123"    )  
  
    driver.find_element_by_id(        "btnLogin"     ).click()
```

quit.py

```
#退出模块  
  
def quit_():  
  
    .....
```

测试用例：

```
#coding=utf-8  
  
from selenium    import    webdriver  
  
import    login,quit_    #调用登录、退出模块
```

```
driver = webdriver.Firefox()

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

#调用登录模块

login.login()

#其它个性化操作

.....

#调用退出模块

quit.quit()
```

注意，上面代码并非完整代码，不能运行。

通过上面的代码发现，我们可以把脚本中相同的部分独立出来，形成模块或库；当脚本需要进行调用。这样做有两个好处：

一方面提高了开发效率，不用重复的编写相同的脚本； 另一方面提高了代码的复用。

数据驱动

数据驱动应该是自动化的一个进步；从它的本意来讲，数据的改变（更新）驱动自动化的执行，从而引起结果改变。这显然是一个非常高级的概念和想法。

其实，我们能做到的是下面的形式。

d:\abc\data.txt

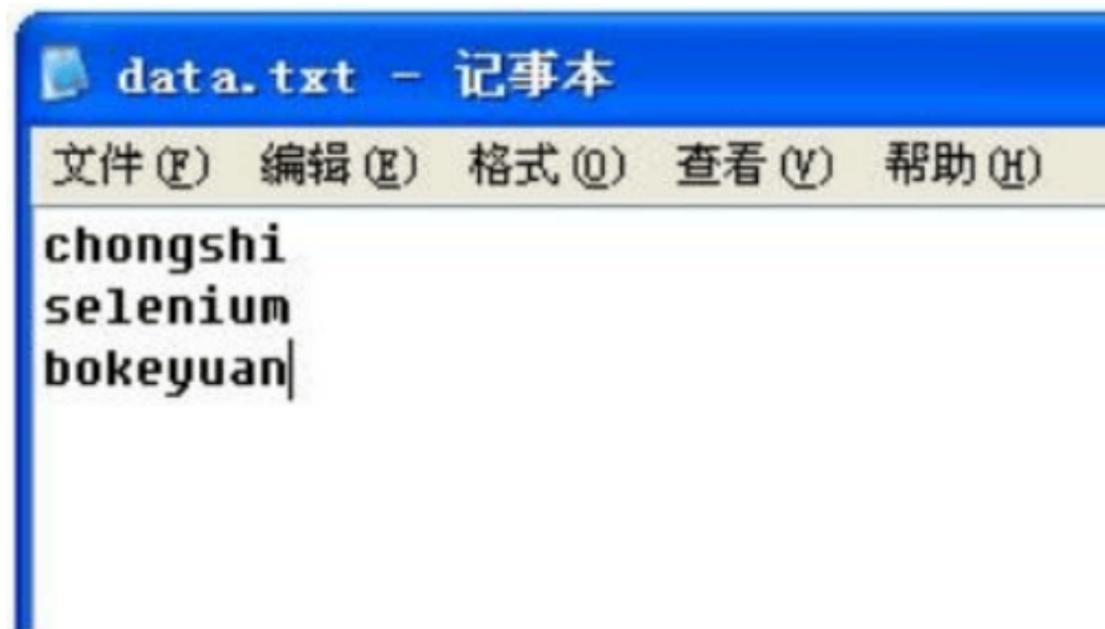


图 4.x

```
#coding=utf-8

from selenium import webdriver

import os,time

source = open( "D:\\abc\\data.txt" , "r" )

values = source.readlines()

source.close()

# 执行循环

for serch in values:

    driver = webdriver.Firefox()

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

    driver.find_element_by_id( "kw" ).send_keys(serch)

.....
```

不管我们读取的是 txt 文件，还是 csv、excel 文件的之类，又或者是数组、字典函数。我们实现了数据与脚本的分离，换句话说，我们实现了参数化。我们仍一千条数据，通过脚本的执行，可以返回一千条结果出来。

同样的脚本执行不同的数据从而得到了不同的结构。是不是增强的脚本的复用性呢！

其实，这对开发来说是完全没有什么技术含量的；对于当初 QTP 自动化工具来说确是一个买点，因为它面对的大多是不懂开发的测试。

关键字驱动

理解了数据驱动，无非是把“数据”换成“关键字”，关键字的改变引起测试结果的变化。

关键字驱动用编程方式就不太容易表现了。 QTP、 robot framework 等自动化工具都提供了关键字驱动（填表格）。

好吧！我能说 selenium IDE 也是关键字驱动么？

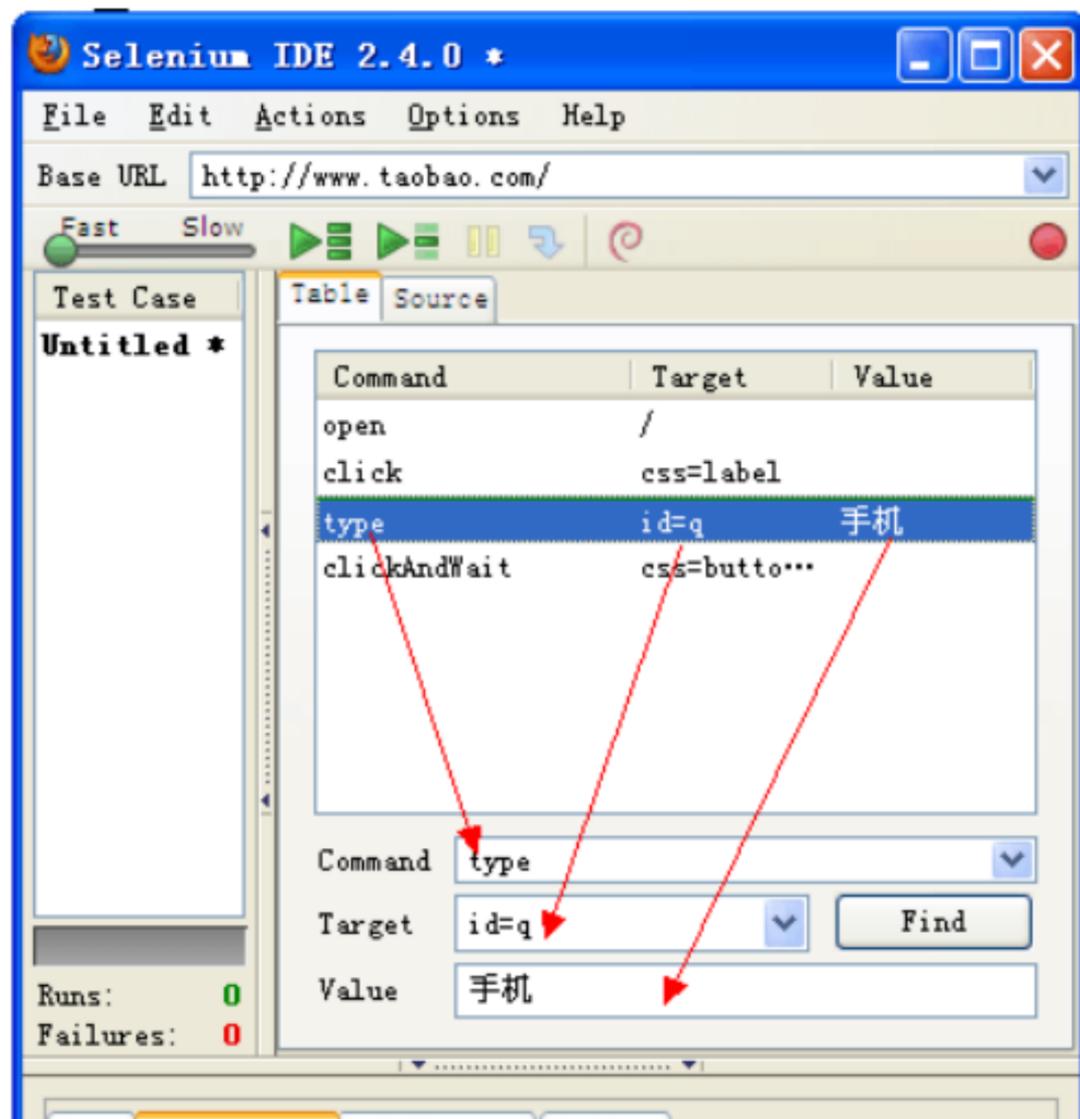


图5.x

转化成表格是这样的：

http://www.taobao.com/		
Command	Target	Value
open	/	
type	Id=q	手机
clickAndWait	Css=button.tsearch-submit	
verifyTextPresent	手机	

图 4.x

Selenium IDE 脚本分：命令 (command)、对象 (command)、值 (value)

格式就那里不偏不移，通过这样的格式去描述不同的对象，从而引起最终结果的改变。也就是说一切以对象为出发点。

当然，这样的脚本，显然对于不懂代码的同学非常直观！我要找谁（对象）？怎么做（命令）？做什么（值）？

更高级的关键字驱动，可以自己定义 keyword 然后“注册”到框架；从而实现更强大的功能和扩展性。关键字更详细的理解可以看我偶像的那篇文章。

这里简单介绍了自动化测试的几种不同的模型，虽然简单阐述了他们的优缺点，但他们并非后者淘汰前者的关系，在实施自动化更多的是以需求为出发点，混合的来使用以上模型去解决问题；使我们的脚本更易于开发与维护。

第二节、登录模块化

通过上一节对测试模型的学习可以看到，在我们的目前的脚本中还是有很多可以模块化的地方，比如登录模块。我们的每一个用例的执行都需要登录脚本，那可我们是否可以将登录脚本独立到单独的文件调用。

下面以快播私有云的登录退出测试用例为例：

webcloud.py

```
#coding=utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
```

```
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time

class Login(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://passport.kuaibo.com"
        self.verificationErrors = []
        self.accept_next_alert = True

        # 私有云登录用例
    def test_login(self):
        driver = self.driver

        driver.get(self.base_url
"/login/?referrer=http%3A%2F%2Fwebcloud.kuaibo.com%2F")

        driver.maximize_window()

        # 登陆
        driver.find_element_by_id("user_name").clear()
        driver.find_element_by_id("user_name").send_keys("username")
        driver.find_element_by_id("user_pwd").clear()
        driver.find_element_by_id("user_pwd").send_keys("123456")
        driver.find_element_by_id("dl_an_submit").click()
        time.sleep(3)

        # 新功能引导
        driver.find_element_by_class_name("guide-ok-btn").click()
        time.sleep(3)

        # 退出
        driver.find_element_by_class_name("Usertool").click()
        time.sleep(2)
```

+

```
driver.find_element_by_link_text("退出").click()

time.sleep(2)

def tearDown(self):
    self.driver.quit()
    self.assertEqual([], self verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

从业务流程及用例分析，每一个自动化测试用例的执行过程为：先执行登录操作，然后执行具体的操作（如文件 / 文件夹的创建、删除、移动、重命名等操作），最后执行退出操作。如上面的测试用例，登录与退出操作是相对固定的，那么我们可以把登录与退出操作模块化出去，然后调用，一方面不用写重复代码，另一方面可以使测试用例更关注具体的用例代码。

login.py

在与 webcloud.py 相同的文件夹下创建 login.py 文件：

```
#coding=utf-8
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
import unittest, time

def login(self):
    driver = self.driver
    driver.maximize_window()
    driver.find_element_by_id("user_name").clear()
    driver.find_element_by_id("user_name").send_keys("username")
    driver.find_element_by_id("user_pwd").clear()
    driver.find_element_by_id("user_pwd").send_keys("123456")
    driver.find_element_by_id("dl_an_submit").click()
    time.sleep(3)
```

webcloud.py

```
#coding=utf-8
from selenium import webdriver
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time
import login # 导入登录文件

class Login(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://passport.kuaibo.com"
        self.verificationErrors = []
        self.accept_next_alert = True

        # 私有云登录用例
    def test_login(self):
        driver = self.driver

        driver.get(self.base_url
"/login/?referrer=http%3A%2F%2Fwebcloud.kuaibo.com%2F")

        # 调用登录模块
        login.login(self)

        # 新功能引导
        driver.find_element_by_class_name("guide-ok-btn").click()
        time.sleep(3)

        # 退出
        driver.find_element_by_class_name("Usertool").click()
        time.sleep(2)
        driver.find_element_by_link_text("退出").click()
        time.sleep(2)
```

```
def tearDown(self):  
    self.driver.quit()  
    self.assertEqual([], self.verificationErrors)  
  
if __name__ == "__main__":  
    unittest.main()
```

进行到这里，我们有必要补充一下 python 语言中函数、类、方法的使用，这将有助于我们自动化测试脚本的开发。下面打开 python IDLE ：

函数的使用：

```
#例 1  
>>> def add(a,b):  
    c=a+b  
    print c  
  
>>> add(1,3)  
4  
#例 2  
>>> def add2(a=1,b=3):  
    c=a+b  
    return c  
  
>>> d=add2()  
>>> print d  
4
```

通过 def 关键字可创建函数，在例 1 中我们创建了 add() 函数，默认接收两个参数化 a、b，对 a、b 相加结果给 c，并将结果函数内打印。

例 2 中创建了 add2() 函数，这一次对 a、b 设置了默认值，同样对 a、b 做加法，并将结果用 return 返回；d 在接收 add2() 时用的是默认值，将后将 d 接收的结果打印。

类与方法的使用：

```
>>> class Counter:  
    def add(self,a,b):  
        c=a+b
```

```
print c
def subtract(self,a,b):
    c=a-b
    print c

>>> d=Counter()
>>> d.add(5,3)
8
>>> d.subtract(5,3)
2
```

通过 class 关键字我们创建了一个 Counter 类,定义了 add() 和 subtract() 两个方法分别来完成加法和减法运算,并将计算结果打印。

通过上面的例子我们明显的发现类的方法与函数有一个明显的区别,在类的方法中必须有个额外的第一个参数(self),但在调用类的方法时却不必为这个参数赋值。 self 参数所指的是对象本身,所以习惯性地命名为 self 。

为何 Python 给 self 赋值而你不必给 self 赋值?

创建了一个类 MyClass ,实例化 MyClass 得到了 MyObject 这个对象,然后调用这个方法 MyObject.method(a,b) ,在这个过程中, Python 会自动转为 Myclass.method(MyObject,a,b) ,这就是 Python 的 self 的原理。即使你的类的方法不需要任何参数,但还是得给这个方法定义一个 self 参数,虽然我们在实例化调用的时候不用理会这个参数。

下面回到用例本身来讨论如何模块化和调用的,在 login.py 文件中:

```
def login(self):
    driver = self.driver
```

这里用到的是方法, (driver = self.driver) driver 为对象身的 driver ,这一句很重要,否则我们无法在 login() 方法中使用 driver 操作浏览器。

在 webdriver.py 文件中:

```
#导入登录文件
import login
...
# 调用登录模块
login.login(self)
```

...

首先导入 login 文件，然后对文件中的 login () 方法进行调用。

下面笔者动手把退出的相关操作也模块化出去吧！

第三节、数据驱动（参数化）

在测试模型一节的数据驱动中我们已经介绍了如何通过 python 的 readlines() 函数对百度输入信息进行参数化设置，将其它循环的读取 data.txt 文件中每一行数据。这里再回顾一下实现参数化的方式。

baidu_read_data.py

```
#coding=utf-8
from selenium import webdriver
import os,time

source = open("D:\\abc\\data.txt" , "r" )
values = source.readlines()
source.close()

# 执行循环
for serch in values:

    browser = webdriver.Firefox()

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

    browser.find_element_by_id( "kw" ).send_keys(serch)

    browser.find_element_by_id( "su" ).click()

    browser.quit()
```

open 方法以只读方式 (r) 打开本地的 data.txt 文件，readlines 方法是逐行的读取文件内容。

通过 for 循环，serch 可以每次获取到文件中的一行数据，在定位到百度的输入框后，将数据传入 send_keys(serch) 。这样通过循环调用，直到文件的中的所有内容全被读取。

登录参数化（读取 txt 文件）

现在按照上面的思路，对自动化脚本中用户名、密码进行参数化，通过 python 文档我们发现 python 读取文件的方式有：整个文件读取、逐行读取、固定字节读取。并没有找到一次读取两条数据的好方法。

创建两个文件，分别存放用户名密码，如图 6.x：

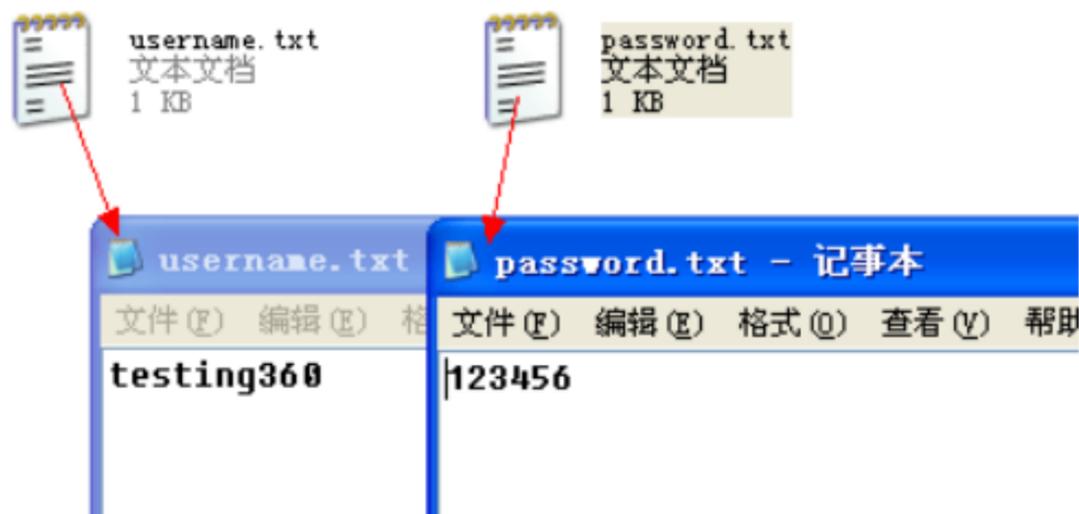


图 6.x

打开之前编写的 login.py 文件，做如下修改：

```
#coding=utf-8
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
import unittest, time, os

source = open("D:\\selenium_python\\data\\username.txt", "r") # 用户名文件
un = source.read() #读取用户名
source.close()

source2 = open("D:\\selenium_python\\data\\password.txt", "r") # 密码文件
pw = source2.read() #读取密码
source2.close()

def login(self):
    driver = self.driver
    driver.maximize_window()
    driver.find_element_by_id("user_name").clear()
    driver.find_element_by_id("user_name").send_keys(un)
    driver.find_element_by_id("user_pwd").clear()
    driver.find_element_by_id("user_pwd").send_keys(pw)
    driver.find_element_by_id("dl_an_submit").click()
    time.sleep(3)
```

分别打开两个 txt 文件，通过 un 和 pw 来接收用户名和密码信息，将接收的数据通过 send_key(xx) 转入到执行程序中。

运行我们前面创建的 `webcloud.py` 文件，程序可以正常的执行。虽然这样做比较丑，但是确实达到了数据与脚本分离的目的。

缺点：

虽然目的达到了这，但这样的实现有很多问题：

- 1、用户名密码分别在不同的文件里，修改用户名和密码比较麻烦。
- 2、`username.txt` 和 `password.txt` 文件中只能保存一个用户密码，无能很好的循环读取。

登录参数化（函数）

函数是我们前面刚介绍的 `python` 知识，函数可以预先给参数化赋值，借助这个特性，我们可以通过调用函数的方式对用户名密码进行参数化

`userinfo.py`

```
def fun(un='testing',pw=123456):  
  
    print "success reader username and password!!"  
  
    return un,pw
```

我们为两个参数 `un` 和 `pw` 赋了初值，赋值内容如果是字符串需要加引号，如果是数字可以不需要引号。再次打开 `login.py` 文件，做如下修改：

```
#coding=utf-8  
  
from selenium import webdriver  
  
from selenium.common.exceptions import NoSuchElementException  
  
import unittest, time  
  
import userinfo # 导入函数  
  
#通过两个变量，来接收调用函数获得用户名 & 密码  
us,pw = userinfo.fun()  
  
#打印两个变量  
print us,pw  
  
def login(self):  
  
    driver = self.driver  
  
    driver.maximize_window()
```

```
driver.find_element_by_id("user_name").clear()

driver.find_element_by_id("user_name").send_keys(un)

driver.find_element_by_id("user_pwd").clear()

driver.find_element_by_id("user_pwd").send_keys(pw)

driver.find_element_by_id("dl_an_submit").click()

time.sleep(3)
```

单独运行 login.py 文件：

```
>>> ===== RESTART =====
>>>
success reader username and password!!
testing 123456
```

说明我们对函数的参数调用是成功的，将 un、pw 两个变量的值传入用户名密码的 send_keys()方法中即可。

登录参数化（读取字典）

既然是固定的读取用户名和密码两个值，那么可以借助 python 字典的方式来完成这个需求。

字典由大括号内的多键值对组成；下面继续在 python IDLE 交互模式下演示字典的创建与使用。

```
>>> data = {'abc':'123','def':'456'}

>>> print data
{'abc': '123', 'def': '456'}

>>> data.keys()
['abc', 'def']

>>> data.values()
['123', '456']

>>> data.items()
[('abc', '123'), ('def', '456')]
```

创建字典用大括号，数据由 key/value 键值对组成，keys()方法返回字典中的键列表。values()返回字典

中的值列表， items()返回 (key , value) 元组。

下面创建一个存放字典的函数 文件 userinfo.py :

```
def zidian():  
  
    data={'username':'123456','testing360':'123123'}  
  
    print "success reader username and password!!"  
  
    return data
```

字典的可以方便的存放 k,v 键值对，一个键对应一个值；注意，如果密码中有非数字，需要加引号。

需要说明的是我们的需求并不适合循环的读取用户名密码，不过我们可以写一小程序单独验证这种循环读取的方式：

loop_reader.py

```
#coding=utf-8  
  
import userinfo # 导入函数  
  
#获取字典数据  
info = userinfo.zidian()  
  
#通过 items() 循环读取元组 ( 键 / 值对 )  
for us,pw in info.items():  
  
    print us  
  
    print pw
```

运行结果如下：

```
>>> ===== RESTART =====  
  
>>>  
success reader username and password!!  
  
username  
123456  
  
testing360  
123456
```

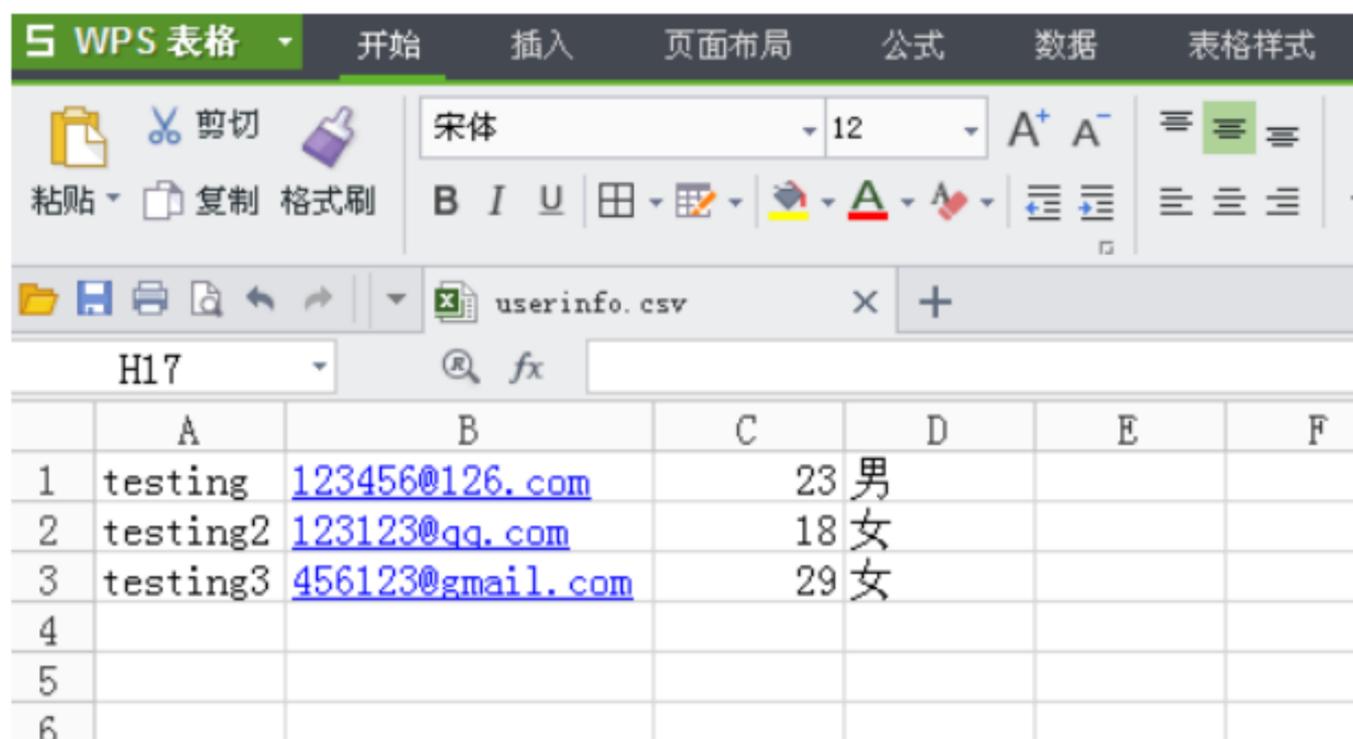
在实际使用中，可以将 un、 pw 两个变量循环获得的值传入相应的 send_keys()方法中。

目前方式解决了两个问题：一次传入两个值，以及循环读取。

表单参数化 (csv)

假如我有自动化脚本中要参数化一张表单，表单需要填写用户名、邮箱，年龄，性别等信息，使用上面的方法就很难来解决这个问题，下面通过读取 csv 文件的方法来解决这个问题。

创建 userinfo.csv 文件，如图 5.x



The screenshot shows the WPS Spreadsheets interface with a file named 'userinfo.csv' open. The spreadsheet contains the following data:

	A	B	C	D	E	F
1	testing	123456@126.com	23	男		
2	testing2	123123@qq.com	18	女		
3	testing3	456123@gmail.com	29	女		
4						
5						
6						

图 5.x

通过 WPS 或 excel 创建表格，文件另存为选择 CSV 格式，下面修改 loop_reader.py 文件进行循环读

取：

```
#coding=utf-8

import csv # 导入 csv 包

#读取本地 CSV 文件

my_file='D:\\selenium_python\\data\\userinfo.csv'

data=csv.reader(file(my_file,'rb'))

#循环输出每一行信息

for user in data:

    print user[0]

    print user[1]

    print user[2]
```

```
print user[3]
```

运行结果：

```
>>> ===== RESTART =====
>>>
testing
123456@126.com
23
男
testing2
123123@qq.com
18
女
testing3
456123@gmail.com
29
女
```

`csv.reader()`用于读取 CSV 文件，`user[0]` 表示表格中第一列的数据（用户名），`user[1]`表示表格中第二列的数据（邮箱），后面类推。

通过 CSV 读取文件比较灵活，可以循环读取每一条数据，从而又不局限每次所读取数据的个数。

我们这里举例了多种方式进行参数化，虽然最终看来 CSV 的方式最灵活，这里更多的是想告诉读者解决问题的方法是多样的，我们可以用 python 的各种技巧来选择最简单的方法来解决。从这个过程中我们也学到了不少 python 编程技术。

我这里可以说是用 python 最基础的知识点解决了问题，这里只算提供一个思路，温习一下 python，你一定可以找到更完美优雅的方法；解决问题的方法是多样的，使用最贴合需求的方法，简单解决问题。这一节写的比较多，对构建自动化框架来说，参数化是非常重要的一个知识点。

脚本的模块化与参数化是我们在自动化脚本开发中用到最多的两个技巧，希望读者能认真体会，灵活的运行到具体的项目中。

编号	测试目标	测试步骤概述
SSQ-1	投注倍数限制	<ol style="list-style-type: none"> 1. 机选1注，设置倍数10000倍，点击增加倍数 2. 机选1注，在倍数框输入10001，点击框外任意空白位置
SSQ-2	投注金额限制	<ol style="list-style-type: none"> 1. 机选12个红球和16个蓝球，设置投注倍数为100倍，点击提交并购买
SSQ-3	投注注数限制	<ol style="list-style-type: none"> 1. 机选16个红球和16个蓝球，默认倍数1倍，点击提交
SSQ-4	智能机选投注限制	<ol style="list-style-type: none"> 1. 智能机选时设置注数为1000以上，点击“开始机选”
SSQ-5	保存方案注数限制	<ol style="list-style-type: none"> 1. 机选12个红球和16个蓝球，默认倍数1倍，点击提交并免费保存
SSQ-6	普通单式投注方案	<ol style="list-style-type: none"> 1. 选择6个红球和1个蓝球，添加到号码篮 2. 调整投注倍数为5倍、100倍 3. 点击“提交”按钮
SSQ-7	普通复式投注方案	<ol style="list-style-type: none"> 1. 随机选择大于6个红球和大于1个蓝球，添加到号码篮 2. 调整投注倍数为5倍、100倍 3. 点击“提交”按钮
SSQ-8	普通单式投注方案购买（内网）	<ol style="list-style-type: none"> 1. 选择6个红球和1个蓝球，添加到号码篮 2. 调整投注倍数为5倍、100倍 3. 点击“提交”按钮，再点击“购买”按钮，输入支付密码，点击“下一步：付款”按钮，完
SSQ-9	普通复式投注方案购买（内网）	<ol style="list-style-type: none"> 1. 随机选择大于6个红球和大于1个蓝球，添加到好嘛篮 2. 调整投注倍数为5倍、100倍 3. 点击“提交”按钮，再点击“购买”按钮，输入支付密码，点击“下一步：付款”按钮，完
SSQ-10	定胆投注方案	<ol style="list-style-type: none"> 1. 随机选择大于6个红球和大于1个蓝球，随机选择y个胆，添加到号码篮 2. 调整投注倍数为5倍、100倍 3. 点击“提交”按钮

核心校验点	备注
<ol style="list-style-type: none"> 1. 页面弹出对话框提示框“倍数最大为10000倍” 2. 倍数框的数字自动变为10000 	未完成（倍数框取不出数据）
<ol style="list-style-type: none"> 1. 页面弹出提示信息“方案金额不能大于200W” 	已完成
<ol style="list-style-type: none"> 1. 页面弹出对话框提示框“最大注数为100000注” 	已完成
<ol style="list-style-type: none"> 1. 页面弹出对话框提示框“最大机选1000注，请返回重新设置” 	未完成（智能机选有bug）
<ol style="list-style-type: none"> 1. 页面弹出提示信息“您的保存方案注数超过1万注” 	本期不做
<ol style="list-style-type: none"> 1. 号码篮显示 1注、单式，号码，下方显示投注倍数1倍，投注金额为2元 2. 投注倍数正确显示，投注金额随着倍数的变化而正确变化 3. 在方案页面验证注数，倍数，金额，号码是否正确 	已完成
<ol style="list-style-type: none"> 1. 号码篮显示x注（通过公式计算）、复式，号码，下方显示投注倍数1倍，投注金额为2*x元 2. 投注倍数正确显示，投注金额随着倍数的变化而正确变化 3. 在方案页面验证注数，倍数，金额，号码是否正确 	已完成
<ol style="list-style-type: none"> 1. 号码篮显示 1注、单式，号码，下方显示投注倍数1倍，投注金额为2元 2. 投注倍数正确显示，投注金额随着倍数的变化而正确变化 3. 验证已支付的订单注数，倍数，金额，号码是否正确 	支付密码自动判定 本期不做
<ol style="list-style-type: none"> 1. 号码篮显示x注（通过公式计算）、复式，号码，下方显示投注倍数1倍，投注金额为2*x元 2. 投注倍数正确显示，投注金额随着倍数的变化而正确变化 3. 验证已支付的订单注数，倍数，金额，号码是否正确 	保证注数以及金额在边界值内 本期不做
<ol style="list-style-type: none"> 1. 号码篮显示x注（通过公式计算）、定胆，胆，号码，下方显示投注倍数1倍，投注金额为2*x元 2. 投注倍数正确显示，投注金额随着倍数的变化而正确变化 3. 在方案页面验证注数，胆，倍数，金额，号码是否正确 	定胆投注方案购买（内网），保证注数 已完成