# Selenium介绍

## 一、安装、基本介绍

在cmd输入 python -m pip install selenium

Selenium`是开源的自动化测试工具，它主要是用于Web 应用程序的自动化测试，不只局限于此，同时支持所有基于web 的管理任务自动化

Selenium 是用于测试 Web 应用程序用户界面 (UI) 的常用框架。它是一款用于运行端到端功能测试的超强工具。您可以使用多个编程语言编写测试，并且 Selenium 能够在一个或多个浏览器中执行这些测试。

Selenium 经历了三个版本：Selenium 1，Selenium 2 和 Selenium 3。Selenium 也不是简单一个工具，而是由几个工具组成，每个工具都有其特点和应用场景。

Selenium 工具集

Selenium IDE

Selenium RC

Selenium Grid

Selenium WebDriver

**浏览器驱动driver的安装，**推荐放置在scripts目录中

简单的脚本：

# coding=utf-8

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-11-25

@author: 北京-宏哥 QQ交流群：705269076

Project: python+ selenium-打开和关闭浏览器

'''

# 3.导入模块

from selenium import webdriver # 导入webdriver包

driver = webdriver.Firefox() # 初始化一个火狐浏览器实例：driver

driver.maximize\_window() # 最大化浏览器

driver.get("https://www.baidu.com") # 通过get()方法，打开一个url站点

driver.quit() #关闭并退出浏览器

## 二、基础篇-常用操作方法

### 浏览器操作：

1.窗口尺寸设置方法

driver.get\_window\_size() # 获取窗口尺寸

driver.set\_window\_size(800,600) # 设置窗口尺寸为800\*600

driver.minimize\_window() # 窗口最小化，窗口尺寸未发生变化

driver.maximize\_window() # 窗口最大化

2.网页截图

driver.get\_screenshot\_as\_file("D:\\screenshot.png") # 截图

3.刷新、前进和后退

driver.back() # 后退

driver.forward() # 前进

time.sleep(3)

driver.refresh()# 对网页进行刷新

4.元素定位

webdriver 提供了八种元素定位方法：定位元素后输入send\_keys() 清除 clear()

id name class name tag name link text partial link text

xpath css selector

find\_element\_by\_id()

find\_element\_by\_name()

find\_element\_by\_class\_name()

find\_element\_by\_tag\_name()

find\_element\_by\_link\_text()

find\_element\_by\_partial\_link\_text()

find\_element\_by\_xpath()

find\_element\_by\_css\_selector()

也可以使用find\_element 和  find\_elements需要导入by类， 如

from selenium.webdriver.common.by import By

driver.find\_element(By.ID,XXXX) XPATH CLASS\_NAME…..

查找多个元素的方法

eles = driver.find\_elements\_by\_class\_name("Feed")

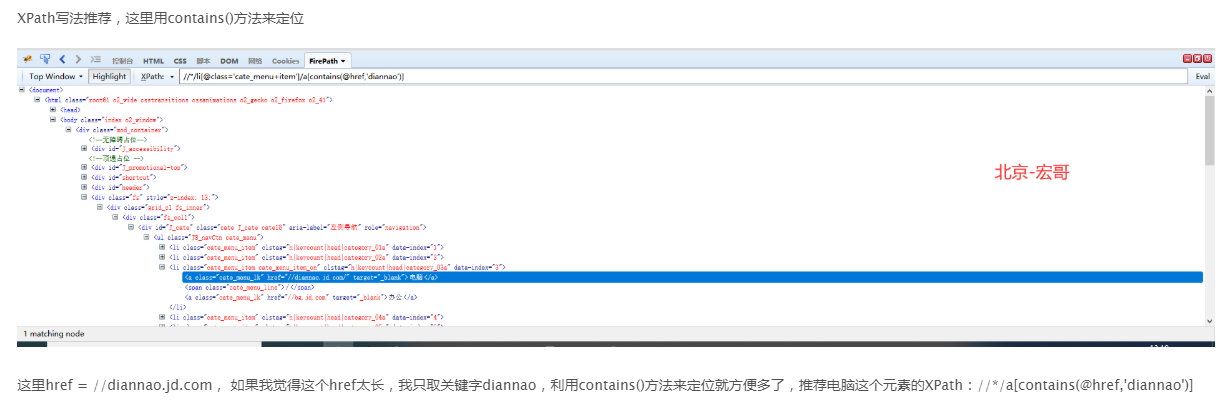
eles = driver.find\_elements(By.CLASS\_NAME, 'Feed')

这里说一下xpath定位：

#### XPath定位技巧之text()方法



#### XPath定位技巧之contains()方法。有时候，我们不喜欢写很长的XPath表达式，而且节点信息里面，有些信息是动态的，每次都获取都不一样，这个时候contains()方法就很好用



#### 相对XPath路径写法

find\_element\_by\_xpath("//input[@id='wd']") 利用元素属性

find\_element\_by\_xpath("//span[@class='bg s\_ipt\_wr']/input") 利用层级

#### find\_element\_by\_xpath("//input[@id='kw' and @class='su']/span/input")使用逻辑运算符

#### 元素操作方法

Send\_keys() clear() click() double\_click()双击 context\_click()右键

Drag\_and\_drop() 拖动

## 三、练习篇

### 练习场景：例如：在某一个网页上有些字段或者关键字等信息是我们感兴趣的，我们希望将其摘取出来，进行其他操作。但是这些字段可能在一个网页的不同地方。例如，我们需要在关于百度页面-联系我们，摘取全部的邮箱

### 思路拆分：

（1）需要得到当前页面的source内容，就像，打开一个页面，右键-查看页面源代码。

（2）找出规律，通过正则表达式去摘取匹配的字段，存储到一个字典或者列表。

（3）循环打印字典或列表中内容，Python中用 for 语句实现。

### 技术角度实现相关方法：

(1)查看页面的源代码，在Selenium中有driver.page\_source 这个方法得到

(2)Python中利用正则，需要导入re模块

(3)将字段通过下列代码打印出来

for email in emails :

print email

# coding=utf-8🔥

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-11-28

@author: 北京-宏哥 QQ交流群：705269076

Project: python+ selenium自动化测试练习篇

'''

# 3.导入模块

from selenium import webdriver

import re

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

driver.get("http://home.baidu.com/contact.html")

# 得到页面源代码

doc = driver.page\_source

emails = re.findall(r'[\w]+@[\w\.-]+', doc) # 利用正则，找出 xxx@xxx.xxx 的字段，保存到emails列表

# 循环打印匹配的邮箱

for email in emails:

print(email)

在python正则表达式语法中，Python中字符串前面加上 r 表示原生字符串，用\w表示匹配字母数字及下划线。re模块下findall方法返回的是一个匹配子字符串的列表。

### 利用ID定位元素

# coding=utf-8🔥

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-11-28

@author: 北京-宏哥 QQ交流群：705269076

Project: python+ selenium自动化测试练习篇

'''

# 3.导入模块

from selenium import webdriver

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

try:

driver.find\_element\_by\_id("kw")

print ('test pass: ID found')

except Exception as e:

print ("Exception found", format(e))

driver.quit()

这里，我们通过try except语句块来进行测试断言，这个在实际自动化测试脚本开发中，经常要用到处理异常。本文，我们学习了可以利用find\_element\_by\_id（）方法来定位网页元素对象

driver.implicitly\_wait(6) 隐式等待

driver.capabilities['version'] # 打印浏览器version的值

### webdriver方法获取当前测试页面的URL

获取当前URL有什么用处呢，一般URL可以帮助我们判断跳转的页面是否正确，或者URL中部分字段可以作为我们自动化测试脚本期待结果的一部分

driver.current\_url # current\_url 方法可以得到当前页面的URL

### 获取当前页面的title

driver.title # title方法可以获取当前页面的标题显示的字段

### selenium方法打开一个新的tab

from selenium.webdriver.common.keys import Keys

keys模块下相关方法，可以通过这个方法，输入任何一个键盘上支持的字符或者快捷键

ele = driver.find\_element\_by\_tag\_name('body').send\_keys(Keys.CONTROL + 't') # 触发ctrl + t

### selenium中的方法去操作单选按钮

driver.find\_elements\_by\_xpath(".//\*[@id='se-settting-1']/input[@type='radio']")

### 操作复选框-Checkbox

### title断言方法

# 方法一

try:

assert u"百度一下" in driver.title

print ('Assertion test pass.')

except Exception as e:

print ('Assertion test fail.', format(e))

# 方法二

if u"百度一下，你就知道" == driver.title :

print ('Assertion test pass.')

else:

print ('Assertion test fail.')

print(driver.title)

### 获取某一个元素的text属性值

# 断言方法一

try :

error\_message= driver.find\_element\_by\_xpath("//\*[@id='TANGRAM\_\_PSP\_10\_\_error' and text()='请您输入手机/邮箱/用户名']").is\_displayed() 判断元素是否显示在页面（is\_displayed()方法）

print ("Test pass. the error message is display.")

except Exception as e:

print ("Test fail.", format(e))

# 断言方法二，本文重点介绍方法

error\_mes= driver.find\_element\_by\_xpath(".//\*[@id='TANGRAM\_\_PSP\_10\_\_error']").text //获取text

try:

assert error\_mes == u'请您输入手机/邮箱/用户名'

print ('Test pass.')

except Exception as e:

print ("Test fail.", format(e))

### 验证控件是否被选中

try:

driver.find\_element\_by\_xpath(".//\*[@id='TANGRAM\_\_PSP\_10\_\_memberPass']").is\_selected()

print("Test Pass.")

except Exception as e:

print('Test fail', format(e))

元素方法is\_selected()返回是是布尔值，用来判断单选或者多选控件是否被选中，或者下拉选择菜单是否选择一个默认的option，都可以通过这个方法去判断。

### 获取页面元素大小

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

time.sleep(1)

search\_btn = driver.find\_element\_by\_id('su')

print (search\_btn.size)

### 组合键-全选文字

driver.get('https://www.baidu.com')

element = driver.find\_element\_by\_tag\_name('body')

element.send\_keys(Keys.CONTROL + 'a')

### 组合键-退格键删除文字

driver.get('https://www.baidu.com')

element = driver.find\_element\_by\_id('kw')

element.send\_keys("Selenium automation")

time.sleep(1)

element.send\_keys(Keys.CONTROL+'a')

element.send\_keys(Keys.BACKSPACE)

### 鼠标右键

在Selenium中，有一个ActionChains模块支持，右键，鼠标悬停，拖拽，双击等动作。我们可以通过键盘向下箭头来选择查看图像这个菜单，然后点击就可以达到目的。

from selenium.webdriver import ActionChains

driver.get('https://www.baidu.com')

time.sleep(2)

element = driver.find\_element\_by\_xpath("//\*[@id='lg']/img")

actionChains = ActionChains(driver)

actionChains.context\_click(element).send\_keys(Keys.ARROW\_DOWN).send\_keys(Keys.ENTER).perform()

### 键盘事件

driver.find\_element(By.ID, 'kw').send\_keys(Keys.CONTROL, 'x') # 剪切搜索框中的内容，也可以使用c进行复制

"""

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（Esc）

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

"""

### 鼠标事件

ActionChains是一种自动执行低级别交互的方法，例如鼠标移动，鼠标按钮操作，按键和上下文菜单交互。这对于执行更复杂的操作非常有用，例如悬停和拖放。

用户操作。  
 在ActionChains对象上调用操作方法时，操作将存储在ActionChains对象的队列中。  
  当调用perform()时，事件将按它们排队的顺序触发。

 ActionChains可以用于链式或者可以将操作逐个排队，然后执行

# 定位元素

ele\_1 = driver.find\_element(By.XPATH, '//\*[@id="u1"]/a[9]')

ele\_2 = driver.find\_element(By.XPATH, '//\*[@id="u1"]/a[8]')

# 将鼠标悬停在元素1上3秒后再次悬停到元素2上

ActionChains(driver).move\_to\_element(ele\_1).perform()

time.sleep(3)

ActionChains(driver).move\_to\_element(ele\_2).perform()

time.sleep(3)

driver.quit()

"""

click(on\_element=None) ——单击鼠标左键

click\_and\_hold(on\_element=None) ——点击鼠标左键，不松开

context\_click(on\_element=None) ——点击鼠标右键

double\_click(on\_element=None) ——双击鼠标左键

drag\_and\_drop(source, target) ——拖拽到某个元素然后松开

drag\_and\_drop\_by\_offset(source, xoffset, yoffset) ——拖拽到某个坐标然后松开

key\_down(value, element=None) ——按下某个键盘上的键

key\_up(value, element=None) ——松开某个键

move\_by\_offset(xoffset, yoffset) ——鼠标从当前位置移动到某个坐标

move\_to\_element(to\_element) ——鼠标移动到某个元素

move\_to\_element\_with\_offset(to\_element, xoffset, yoffset) ——移动到距某个元素（左上角坐标）多少距离的位置

perform() ——执行链中的所有动作

release(on\_element=None) ——在某个元素位置松开鼠标左键

send\_keys(\*keys\_to\_send) ——发送某个键到当前焦点的元素

send\_keys\_to\_element(element, \*keys\_to\_send) ——发送某个键到指定元素

"""

### 执行JavaScript

前面文章介绍了ActionChains下的鼠标悬停和右键操作，其实ActionChains还有其他方法，你可以 action = ActionChains,通过这，先初始化一个ActionChains实例对象，然后在action后面输入一个点号，查看支持哪些action方法。本文开始介绍如何执行JS脚本，会用两个例子说明：示例一，执行js脚本触发一个alert弹出框

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

time.sleep(1)

driver.execute\_script("window.alert('这是一个alert弹框。');") # 注意这里的分号是英文输入法的分号，不能用中文

示例二，执行js脚本来控制浏览器竖向滚动条。打开百度贴吧，然后拖动滚动条到左侧 “地区"

driver.get("https://tieba.baidu.com/index.html")

time.sleep(1)

target\_elem = driver.find\_element\_by\_link\_text("地区")

driver.execute\_script("return arguments[0].scrollIntoView();",target\_elem) # 用目标元素参考去拖动

### 多窗口之间切换 ？

本文来介绍如何处理driver在多窗口之间切换，想一下这样的场景，在页面A点击一个连接，会触发在新Tab或者新窗口打开页面B，由于之前的driver实例对象在页面A，但是你接下来的脚本是操作页面B的元素，这样就造成了找不到元素的报错。本来介绍selenium中switch\_to.window()方法来处理这个问题。

测试场景：打开百度新闻（页面A），点击热点新闻中第一个新闻链接（一般是国家领导人的新闻），会在第二个窗口打开这个新闻的具体详情页（页面B），测试需要去判断你点击这个这个新闻，在打开的详情页是否正确。

问题拆分：

1. 我们已经知道switch\_to.window()方法可以处理窗口切换的问题

2. 在页面A跳转到页面B之前，我们需要用一个变量保存这个新闻的标题

3. 切换到页面B后，我们获取这个新闻标题，然后和前面这个变量保存的值去对比，如果相等，那么就测试通过。

我们分两个步骤去解答这个测试需求：

1. 先实现页面A切换到页面B

2. 页面A和页面B两个新闻标题进行对比

driver = webdriver.Chrome()

driver.maximize\_window()

driver.get('http://news.baidu.com')

time.sleep(1)

driver.find\_element\_by\_xpath("//\*[@id='pane-news']/div/ul/li[1]/strong/a").click()

print (driver.current\_window\_handle) # 输出当前窗口句柄

handles = driver.window\_handles # 获取当前全部窗口句柄集合

print (handles) # 输出句柄集合

for handle in handles:# 切换窗口

if handle != driver.current\_window\_handle:

print ('switch to second window',handle)

driver.close() # 关闭第一个窗口

driver.switch\_to.window(handle) #切换到第二个窗口

#### 断言切换的是不是你刚刚点击的新闻

driver = webdriver.Chrome()

driver.maximize\_window()

driver.get('http://news.baidu.com')

time.sleep(1)

news\_link = driver.find\_element\_by\_xpath("//\*[@id='pane-news']/div/ul/li[1]/strong/a")

page1\_title\_string = news\_link.text #得到页面A新闻标题

print('page1\_title\_string'+ page1\_title\_string)

news\_link.click() # 点击新闻链接

time.sleep(1)

handles = driver.window\_handles

for handle in handles:# 切换窗口（切换到搜狗）

if handle != driver.current\_window\_handle:

print ('switch to second window',handle)

driver.close() # 关闭第一个窗口

driver.switch\_to.window(handle) #切换到第二个窗口

page2\_title\_string = driver.find\_element\_by\_xpath(".//\*[@id='title\_area']/h1").text # 详情页有一个原标题

print('page2\_title\_string'+ page2\_title\_string)

try :

assert page1\_title\_string in page2\_title\_string # 判断页面B标题是否包含页面A标题

print ('Test Pass.')

except Exception as e:

print ('Test Fail')

### 处理iframe切换 ？

本文介绍如何利用Selenium中方法，在不同iframe中切换。

问题：有时候我们发现元素定位没问题，在测试回放的过程，发现就是找不到元素报错。

方法介绍：

driver.switch\_to.frame("iframe1")  
# 操作目标元素，这个目标元素在 iframe1里面，这里就是百度文本输入框输入文字  
driver.switch\_to.default\_content()   
步骤：

1.先确认你要操作的元素，是否存在于iframe中，如果元素在iframe中就需要切换

2.找到该iframe

3.切换到该iframe （两种方式）

   第一种方式：有三种方法，只是单纯的切换  
   driver.switch\_to.frame(1) #通过index（下标）  
   driver.switch\_to.frame('login\_frame\_qq') #通过name  
  driver.switch\_to.frame(driver.find\_element\_by\_xpath('//iframe[@name="login\_frame\_qq"]')) #通过webelement

   第二种方式：既等待元素可见又进行了iframe切换

   WebDriverWait(driver,20).until(EC.frame\_to\_be\_available\_and\_switch\_to\_it('login\_frame\_qq'))

4.切换完后，就以这个iframe中的html为主html

5.返回默认的html页面，无论现在在那一层的iframe，执行一次即可。  
   driver.switch\_to.default\_content()

import time

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

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

from selenium.webdriver.common.by import By

driver = webdriver.Chrome()

driver.maximize\_window()

# 登录腾讯课堂页面

driver.get('https://ke.qq.com/course/list')

# 点击首页登录

WebDriverWait(driver,20).until(EC.visibility\_of\_element\_located((By.XPATH,'//a[@id="js\_login"]')))

driver.find\_element\_by\_id("js\_login").click()

# 登录弹窗点击QQ登录

WebDriverWait(driver,20).until(EC.visibility\_of\_element\_located((By.XPATH,'//a[contains(@class,"btns-enter-qq")]')))

driver.find\_element\_by\_xpath('//a[contains(@class,"btns-enter-qq")]').click()

# iframe切换

# driver.switch\_to.frame('login\_frame\_qq') #通过name

driver.switch\_to.frame(driver.find\_element\_by\_xpath('//iframe[@name="login\_frame\_qq"]')) #通过webelement

# 弹窗点击账户密码登录switcher\_plogin

#WebDriverWait(driver,20).until(EC.visibility\_of\_element\_located((By.XPATH,'//a[@id="switcher\_plogin"]')))

time.sleep(10)

driver.find\_element\_by\_xpath('//a[@id="switcher\_plogin"]').click()

小结：

#### 多个iframe的切换问题

# 如：有两个iframe：f1、f2

# 1.定位到f1

driver.switch\_to\_frame("f1")

# 2.操作元素

# ······

# 3.退出iframe

driver.switch\_to\_default\_content()

# 4.定位到f2

driver.switch\_to\_frame("f2")

# ······

#### 嵌套：f1中嵌套着f2

driver.switch\_to\_frame("f1")

driver.switch\_to\_frame("f2")

# 操作元素

#### 退出iframe：

# 第一种方式：跳出所有iframe，回到主界面

driver.switch\_to\_default\_content()

# 第二种方式：回到f1（返回上一级）

driver.switch\_to.parent\_frame()

### 处理Alert弹窗 ？

本文来介绍如何通过Selenium方法去处理网页Alert弹窗，和处理iframe类似，都是通过switch\_to方法。这里还是没有找到合适的alert弹窗网站，我们就自己创建一个吧，前面文章介绍了如何通过执行JS来增加一个弹窗。

（1）使用switch\_to方法先切换到浏览器弹出框  
         driver.switch\_to.alert  
（2）Alert类提供了一系列的操作方法  
        dismiss(): 否  
        accept(): 是  
        text(): 获取弹出框里的文本内容  
        send\_keys(): 在弹出框里输入文本

# coding=utf-8🔥

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-12-09

@author: 北京-宏哥 QQ交流群：705269076

Project: python+ selenium自动化测试练习篇8

'''

# 3.导入模块

import time

from selenium import webdriver

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

time.sleep(1)

driver.execute\_script("window.alert('这是一个测试Alert弹窗');")

time.sleep(2)

driver.switch\_to\_alert().accept() # 点击弹出里面的确定按钮

#driver.switch\_to\_alert().dismiss() # 点击弹出上面的X按 优先使用这个方法

### 获取当前页面全部图片信息

本文来介绍下如何获取当前页面全部的图片信息，图片信息可能包括，图片名称，图片大小等。

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

time.sleep(1)

for image in driver.find\_elements\_by\_tag\_name("img"):

print (image.text)

print (image.size)

print (image.tag\_name)

### 获取页面元素的href属性

本文来介绍如何通过Selenium获取页面元素的某一个属性。一个元素可能有多个属性，例如 class, id, name, text, href, vale等等。这里我们举例一个爬虫中经常需要处理的链接问题：找出当前页面所有的超链接。

已百度首页为例，打印所有包含href的元素的链接。

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

time.sleep(1)

for link in driver.find\_elements\_by\_xpath("//\*[@href]"):

print (link.get\_attribute('href'))

driver.quit()

### 如何截图并保存

本文介绍如何利用Selenium的方法进行截图，在测试过程中，是有必要截图，特别是遇到错误的时候进行截图。在selenium for python中主要有三个截图方法，我们挑选其中最常用的一种。**get\_screenshot\_as\_file()**

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(6)

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

time.sleep(1)

driver.get\_screenshot\_as\_file("C:\\Users\\你的账户名\\Desktop\\baidu.png")

driver.quit()

### python-web自动化-三种等待方式

当有元素定位不到时，比如下拉框，弹出框等各种定位不到时；

一般是两种问题：1 、有frame ；2、没有加等待

下面学习三种等待方式：

#### 强制等待 sleep（xx）或者Time.sleep

这种方法简单粗暴，不管浏览器是否加载完，程序都要等待规定的xx时间，时间到了才继续执行下面的代码。

不建议总是用这种等待方式，会严重影响程序的执行速度。通过time模块中sleep进行代码的暂停，但是实际使用过程中，如果都以sleep进行控制严重影响了程序的运行。

from selenium import webdriver

from time import sleep

driver = webdriver.Chrome()

driver.get('https://www.baidu.com/')

sleep(3)#强制性等待3s再执行以下代码

print(driver.current\_url)

driver.quit()#退出驱动，关闭所有窗口

#### 隐性等待 implicitly\_wait(xx)

这种方法是设置一个最长的等待时间，如果在规定时间内网页全部元素加载完成，则执行下一步，否则一直等待时间截止才执行下一步。比强制性等待智能些

！隐性等待对整个driver的周期都起作用，所以只要设置一次即可

from selenium import webdriver

from selenium.common.exceptions import NoSuchElementException

from time import ctime

driver = webdriver.Firefox()

# 设置隐式等待为10秒

driver.implicitly\_wait(10)

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

try:

print(ctime())

driver.find\_element\_by\_id("kw22").send\_keys('selenium') #找不到

except NoSuchElementException as e:

print(e)

finally:

print(ctime())

driver.quit()

这里可以看到在10秒内没有找到想要找到的元素，但是依旧执行了10秒，然后报错，如果修改代码为可以找到，代码执行非常迅速。

implicitly\_wait()默认参数的单位为秒，本例中设置等待时长为10秒。首先这10秒并非一个固定的等待时间，它并不影响脚本的执行速度。其次，它并不针对页面上的某一元素进行等待。当脚本执行到某个元素定位时，如果元素可以定位，则继续执行；如果元素定位

不到，则它将以轮询的方式不断地判断元素是否被定位到。假设在第6秒定位到了元素则继续执行，若直到超出设置时长（10秒）还没有定位到元素，则抛出异常。

#### 显性等待 WebDriverWait

#A. 使用前，先引用相关库

from selenium.webdriver.support.wait import WebDriverWait

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

WebDriverWait配合该类的until()和until\_not()方法，根据条件灵活的等待

程序每隔xx秒看一眼，如果条件成立了，则执行下一步，否则继续等待，直到超过设置的最长时间，然后抛出TimeoutException。

显式等待是你在代码中定义等待一定条件发生后再进一步执行你的代码。

A. 使用前，先引用相关库

B. 确定元素的定位表达式

C.  使用expected\_conditions对应的方法来生成判断条件

WebDriverWait(driver,10,1).until(EC.visibility\_of\_element\_located((By.ID,ele\_locator)))

WebDriverWait(driver,10,1).until(EC.visibility\_of\_element\_located((By.XPATH,ele\_locator)))

D.  调用WebDriverWait类设置等待总时长、轮询周期

# coding=utf-8🔥

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-12-10

@author: 北京-宏哥 QQ交流群：705269076

Project: 《手把手教你》系列练习篇之9-python+ selenium自动化测试

'''

# 3.导入模块

#A. 使用前，先引用相关库

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

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

from selenium.webdriver.common.by import By

driver = webdriver.Chrome()#打开Chrome浏览器

driver.get('https://www.baidu.com/')#打开百度

driver.find\_element\_by\_xpath('//div[@id="u1"]//a[@name="tj\_login"]').click()#点击【登录】；click() 方法，可模拟在按钮上的一次鼠标单击。

# B. 确定元素的定位表达式

ele\_locator = "TANGRAM\_\_PSP\_10\_\_footerULoginBtn"#通过id,确定‘用户名登录’元素

# C. 使用expected\_conditions对应的方法来生成判断条件

# EC.方法名(定位方式,定位表达式)

# EC.visibility\_of\_element\_located(By.ID,ele\_locator)#元素可见

# D. 调用WebDriverWait类设置等待总时长、轮询周期

# WebDriverWait(driver, 超时时长, 调用频率（默认0.5s）).until(可执行方法, 超时时返回的信息)

# 等待10秒钟，每隔1秒去查看对应的元素是否可见；如果可见，继续下一步操作；如果不可见，则继续等待，直到10s结束，如果元素还是不可见，则抛出超时异常

WebDriverWait(driver,10,1).until(EC.visibility\_of\_element\_located((By.ID,ele\_locator)))

driver.find\_element\_by\_id('TANGRAM\_\_PSP\_10\_\_footerULoginBtn').click()#点击【用户名登录】

driver.close()#关闭当前窗口

代码2：

from time import ctime

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

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

dr = webdriver.Chrome()

dr.get('https://www.baidu.com')

try:

print(ctime())

element = WebDriverWait(dr, 10).until(

EC.presence\_of\_element\_located((By.ID, "kw"))

)

# WebDriverWait(driver=self.driver, timeout=300, poll\_frequency=0.5, ignored\_exceptions=None)

# driver：浏览器驱动

# timeout：最长超时等待时间

# poll\_frequency：检测的时间间隔，默认为500ms

# ignore\_exception：超时后抛出的异常信息，默认情况下抛 NoSuchElementException 异常

print("我已找到")

finally:

print(ctime())

dr.quit()

以上代码执行后就发现，整段代码执行速度非常快，即使我在WebDriverWait中设置10秒，也不会等待10秒的情况，因为在不到一秒内，已经完成了加载并定位id为“kw”的元素。

通过WebDriverWait 和 ExpectedCondition 组合使用，让我们的代码执行只需要等待需要的时长，而不是固定的时长，这样最大限度的节省时间。

 此外ExpectedCondition类中提供了很多预期条件判断方法，省去了再创建包的功夫：

"""

title\_is：判断当前页面的title是否等于预期

title\_contains：判断当前页面的title是否包含预期字符串

presence\_of\_element\_located：判断某个元素是否被加到了dom树里，并不代表该元素一定可见

visibility\_of\_element\_located：判断某个元素是否可见. 可见代表元素非隐藏，并且元素的宽和高都不等于0

visibility\_of：跟上面的方法做一样的事情，只是上面的方法要传入locator，这个方法直接传定位到的element就好了

presence\_of\_all\_elements\_located：判断是否至少有1个元素存在于dom树中。举个例子，如果页面上有n个元素的class都是'column-md-3'，那么只要有1个元素存在，这个方法就返回True

text\_to\_be\_present\_in\_element：判断某个元素中的text是否 包含 了预期的字符串

text\_to\_be\_present\_in\_element\_value：判断某个元素中的value属性是否包含了预期的字符串

frame\_to\_be\_available\_and\_switch\_to\_it：判断该frame是否可以switch进去，如果可以的话，返回True并且switch进去，否则返回False

invisibility\_of\_element\_located：判断某个元素中是否不存在于dom树或不可见

element\_to\_be\_clickable - it is Displayed and Enabled：判断某个元素中是否可见并且是enable的，这样的话才叫clickable

staleness\_of：等某个元素从dom树中移除，注意，这个方法也是返回True或False

element\_to\_be\_selected：判断某个元素是否被选中了,一般用在下拉列表

element\_located\_to\_be\_selected

element\_selection\_state\_to\_be：判断某个元素的选中状态是否符合预期

element\_located\_selection\_state\_to\_be：跟上面的方法作用一样，只是上面的方法传入定位到的element，而这个方法传入locator

alert\_is\_present：判断页面上是否存在alert

"""

另外这里使用了until()函数也可以使用until\_not()

　　until\_not(method, message='')

　　调用该方法体提供的回调函数作为一个参数，直到返回值为False

　　until(method, message='')

　　调用该方法体提供的回调函数作为一个参数，直到返回值为True

skipIf()条件为真执行

skipUnless()条件为假时执行

### 下拉框or弹框

import time

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

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

from selenium.webdriver.support.select import Select

driver = webdriver.Chrome()

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

time.sleep(5)

# 折腾半天才发现定位出来的不是下拉框……

driver.find\_element(By.LINK\_TEXT, '设置').click()

driver.find\_element(By.LINK\_TEXT, '搜索设置').click()

time.sleep(2)

sel = driver.find\_element(By.XPATH, '//\*[@id="nr"]') # 定位下拉框

Select(sel).select\_by\_value('50') # 通过value的值进行选定条目

driver.find\_element(By.CLASS\_NAME, 'prefpanelgo').click()

time.sleep(2)

driver.switch\_to.alert.accept() # 此处同意警告框提示内容

time.sleep(2)

driver.quit()

### 自动化测试demo

使用selenium进行测试，当然是需要进行验证的，此时结合python的断言函数就变得非常有用了。测试中设定好预期，当实际情况与预期有差别，给出错误信息，好像正是所希望的。

来一段简单的代码看下断言的基本使用:

dr = webdriver.Chrome()

dr.maximize\_window()

dr.get("https://www.baidu.com")

print(dr.title)

try:

assert (dr.title == "百度，你就知道")

except:

print('标题错误，请查看错误信息')

time.sleep(5)

dr.quit()

本文就基于python+selenium结合unittest单元测试框架来进行一次简单但比较完整的cnblog自动化登录测试，可提供点参考！下面就包括测试代码和每种测试情况的截图。

coding=utf-8🔥

# 1.先设置编码，utf-8可支持中英文，如上，一般放在第一行

# 2.注释：包括记录创建时间，创建人，项目名称。

'''

Created on 2019-12-10

@author: 北京-宏哥 QQ交流群：705269076

Project: python+ selenium自动化测试练习篇9

'''

# 3.导入模块

import unittest

from selenium import webdriver

from time import sleep

'''

cnblog的登录测试，分下面几种情况：

(1)用户名、密码正确

(2)用户名正确、密码不正确

(3)用户名正确、密码为空

(4)用户名错误、密码正确

(5)用户名为空、密码正确（还有用户名和密码均为空时与此情况是一样的，这里就不单独测试了）

'''

class LoginCase(unittest.TestCase):

def setUp(self):

self.dr = webdriver.Chrome()

self.dr.maximize\_window()

# 定义登录方法

def login(self, username, password):

self.dr.get('https://passport.cnblogs.com/user/signin') # cnblog登录页面

self.dr.find\_element\_by\_id('LoginName').send\_keys(username)

self.dr.find\_element\_by\_id('Password').send\_keys(password)

self.dr.find\_element\_by\_id('submitBtn').click()

def test\_login\_success(self):

'''用户名、密码正确'''

self.login('北京-宏哥', '!qaz2wsx') # 正确用户名和密码

sleep(3)

link = self.dr.find\_element\_by\_id('lnk\_current\_user')

self.assertTrue('北京-宏哥' in link.text) # 用assertTrue(x)方法来断言 bool(x) is True 登录成功后用户昵称在lnk\_current\_user里

self.dr.get\_screenshot\_as\_file("D:\\cnblogtest\\login\_success.jpg") # 截图 可自定义截图后的保存位置和图片命名

def test\_login\_pwd\_error(self):

'''用户名正确、密码不正确'''

self.login('北京-宏哥', 'kemi') # 正确用户名，错误密码

sleep(2)

error\_message = self.dr.find\_element\_by\_class\_name('ajax-error-box').text

self.assertIn('用户名或密码错误', error\_message) # 用assertIn(a,b)方法来断言 a in b '用户名或密码错误'在error\_message里

self.dr.get\_screenshot\_as\_file("D:\\cnblogtest\\login\_pwd\_error.jpg")

def test\_login\_pwd\_null(self):

'''用户名正确、密码为空'''

self.login('北京-宏哥', '') # 密码为空

error\_message = self.dr.find\_element\_by\_id('Password-error').text

self.assertEqual(error\_message, '请输入密码') # 用assertEqual(a,b)方法来断言 a == b 请输入密码等于error\_message

self.dr.get\_screenshot\_as\_file("D:\\cnblogtest\\login\_pwd\_null.jpg")

def test\_login\_user\_error(self):

'''用户名错误、密码正确'''

self.login('北京-宏哥1', '!qaz2wsx') # 密码正确，用户名错误

sleep(2)

error\_message = self.dr.find\_element\_by\_id('ajax-error-box').text

self.assertIn('用户名或密码错误', error\_message) # 用assertIn(a,b)方法来断言 a in b

self.dr.get\_screenshot\_as\_file("D:\\cnblogtest\\login\_user\_error.jpg")

def test\_login\_user\_null(self):

'''用户名为空、密码正确'''

self.login('', '!qaz2wsx') # 用户名为空，密码正确

error\_message = self.dr.find\_element\_by\_id('LoginName-error').text

self.assertEqual(error\_message, '请输入登录用户名') # 用assertEqual(a,b)方法来断言 a == b

self.dr.get\_screenshot\_as\_file("D:\\cnblogtest\\login\_user\_null.jpg")

def tearDown(self):

sleep(2)

print('自动测试完毕！')

self.dr.quit()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

## 四、进阶篇

自动化测试大概可以分三个层次：

（1）手工测试用例转换成自动化测试脚本的过程

（2）能设计自动化测试框架，至少能够维护自动化测试框架。

（3）流程自动化方案设计，例如，一键打包，自动开始测试，自动发送测试报告，自动运维部署上线等。

在开始介绍设计简单的自动化测试框架之前，我想在这个节点，插入一个新章节，叫Python+Selenium进阶篇，主要是介绍一些设计框架的前提技能和基本Python编写代码的能力。这些能力包括但不限于以下内容：

1. 主流Python开发IDE工具的基本使用，例如Pycharm
2. Python中模块，类和对象的具体代码讲解。
3. Selenium 常见方法的二次封装。
4. 自定义方法的封装和方法的调用-浏览器引擎类。
5. Python读写配置文件介绍
6. Python如何获取系统时间和时间的格式化处理。
7. Python中常见字符串切割处理。
8. Python自定义一个日志生成方法封装。
9. Selenium中一个截图方法的封装。
10. Python中继承的使用。

通过介绍以上进阶技能学习后，我们才可以，或者有能力去思考和动手去设计一个简单的自动化测试框架。

### Python中类/函数/模块的简单介绍和方法调用

class ClassA(object):

string1 = "这是一个字符串。"

def instancefunc(self):

print ('这是一个实例方法。')

print (self)

@classmethod

def classfunc(cls):

print ('这是一个类方法。')

print (cls)

@staticmethod

def staticfun():

print ('这是一个静态方法。')

test = ClassA() # 初始化一个ClasssA的对象，test是类ClassA的实例对象

test.instancefunc() # 对象调用实例方法

test.staticfun() # 对象调用静态方法

test.classfunc() # 对象调用类方法

print (test.string1) # 对象调用类变量

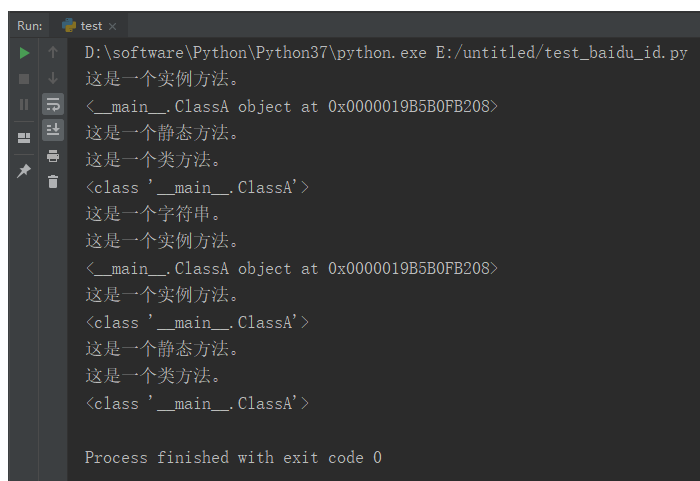
ClassA.instancefunc(test) # 类调用实例方法，需要带参数，这里的test是一个对象参数

ClassA.instancefunc(ClassA) # 类调用实例方法，需要带参数，这里的ClassA是一个类参数

ClassA.staticfun() # 类调用静态方法

ClassA.classfunc() # 类调用类方法

执行结果：



### 二次封装Selenium中几个方法

class BasePage(object):

"""

主要是把常用的几个Selenium方法封装到BasePage这个类，我们这里演示以下几个方法

back()

forward()

get()

quit()

"""

def \_\_init\_\_(self, driver):

"""

写一个构造函数，有一个参数driver

:param driver:

"""

self.driver = driver

def back(self):

"""

浏览器后退按钮

:param none:

"""

self.driver.back()

def forward(self):

"""

浏览器前进按钮

:param none:

"""

self.driver.forward()

def open\_url(self, url):

"""

打开url站点

:param url:

"""

self.driver.get(url)

def quit\_browser(self):

"""

关闭并停止浏览器服务

:param none:

"""

self.driver.quit()

#### 调用自己封装过的方法

import time

from selenium import webdriver

from blog.basepage import BasePage

class BaiduSearch(object):

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(10)

basepage = BasePage(driver)

def open\_baidu(self):

self.basepage.open\_url("https://www.baidu.com")

time.sleep(1)

def test\_search(self):

self.driver.find\_element\_by\_id('kw').send\_keys("Selenium")

time.sleep(1)

self.basepage.back()

self.basepage.forward()

self.basepage.quit\_browser()

baidu = BaiduSearch()

baidu.open\_baidu()

baidu.test\_search()

上面self.basepage的几行代码就是调用我们自己封装的方法去执行相关webdriver操作。这个只是一个简单的封装介绍，等后面，我们介绍了字符串切割，我们会再次介绍二次封装Selenium方法，例如将会把八大find\_element方法封装到一个方法里去

#### 封装一个自己的类-浏览器引擎类

from selenium import webdriver

class BrowserEngine(object):

"""

定义一个浏览器引擎类，根据browser\_type的值去，控制启动不同的浏览器，这里主要是IE，Firefox, Chrome

"""

def \_\_init\_\_(self, driver):

self.driver = driver

browser\_type = "IE" # maybe Firefox, Chrome, IE

def get\_browser(self):

"""

通过if语句，来控制初始化不同浏览器的启动，默认是启动Chrome

:return: driver

"""

if self.browser\_type == 'Firefox':

driver = webdriver.Firefox()

elif self.browser\_type == 'Chrome':

driver = webdriver.Chrome()

elif self.browser\_type == 'IE':

driver = webdriver.Ie()

else: driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(10)

return driver

再看看test.py代码，进行测试，更改browser\_engine.py中browser\_type的值，去测试三大浏览器是否启动正常

# 3.导入模块

from test1.browser\_engine import BrowserEngine

class TestBrowserEngine(object):

def open\_browser(self):

browserengine = BrowserEngine(self)

driver = browserengine.get\_browser()

tbe = TestBrowserEngine()

tbe.open\_browser()

### python几种骚操作

python几种骚操作：读取配置文件、获取根目录的相对路径、获取系统时间和格式化时间显示、字符串切割等等操作。为后边的自动化框架打下一个结实的基础。

#### Python读取配置文件内容

本文来介绍下Python中如何读取配置文件。任何一个项目，都涉及到了配置文件和管理和读写，Python支持很多配置文件的读写，这里我们就介绍一种配置文件格式的读取数据，叫ini文件。Python中有一个类ConfigParser支持读ini文件。

首先我们紧接着前边的内容，在项目下，新建一个文件夹，叫config，然后在这个文件夹下新建一个file类型的文件:config.ini

# this is config file, only store browser type and server URL

[browserType]

#browserName = Firefox

browserName = Chrome

#browserName = IE

[testServer]

URL = https://www.baidu.com

#URL = http://www.google.com

#### 获取当前项目的根目录的相对路径

os.path.abspath('.') 返回当前目录

os.path.dirname(os.path.abspath('.')) 返回当前目录的上级目录名

os.path.abspath(\_\_file\_\_) 返回当前文件的路径，包含文件自身

在另外一个包下新建一个测试类，用来测试读取配置文件是否正常

import configparser

import os

class TestReadConfigFile(object):

def get\_value(self):

root\_dir = os.path.dirname(os.path.abspath('.')) # 获取项目根目录的相对路径

print (root\_dir)

config = configparser.ConfigParser()

file\_path = os.path.dirname(os.path.abspath('.')) + '/config/config.ini'

config.read(file\_path)

browser = config.get("browserType", "browserName")

url = config.get("testServer", "URL")

return(browser,url) # 返回的是一个元组

trcf = TestReadConfigFile()

print (trcf.get\_value())

#### Python获取系统时间和格式化时间显示

在PyCharm下的一个包，右键，新建一个get\_time.py文件，输入一下代码

import time

class GetTime(object):

def get\_system\_time(self):

print (time.time()) # time.time()获取的是从1970年到现在的间隔，单位是秒

print (time.localtime())

new\_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime()) # 格式化时间，按照 2019-12-13 13:46:32的格式打印出来

print (new\_time)

gettime = GetTime()

gettime.get\_system\_time()

#### Python中字符串切割操作

本文来介绍Python中字符串切割操作，在Python中自带的一个切割方法split(),这个方法不带参数，就默认按照空格去切割字段，如果带参数，就按照参数去切割

class GetSubString(object):

def get\_search\_result(self):

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(8)

driver.get('https://www.baidu.com')

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

time.sleep(1)

driver.find\_element\_by\_id('su').click()

time.sleep(1)

search\_result\_string = driver.find\_element\_by\_xpath("//\*/div[@class='nums']").text

print (search\_result\_string)

new\_string = search\_result\_string.split('约')[1] # 第一次切割得到 xxxx个，[1]代表切割右边部分

print (new\_string)

last\_result = new\_string.split('个')[0] # 第二次切割，得到我们想要的数字 [0]代表切割参照参数的左边部分

print (last\_result)

getstring = GetSubString()

getstring.get\_search\_result()

### python超神操作

今天分享和讲解的超神操作，对于菜鸟来说是超神的操作，对于大佬来说也就是几个简单方法的封装和调用。这里讲解和分享这部分主要是为了培养小伙伴们和童鞋们的面向对象的开发思维，对比这样做的好处让你自己身临其境的感受一番。

#### 自定义封装一个简单的Log类

　本文介绍如何写一个Python日志类，用来输出不同级别的日志信息到本地文件夹下的日志文件里。为什么需要日志输出呢，我们需要记录我们测试脚本到底做了什么事情，最好的办法是写事件监听。这个事件监听，对我们现在来说，还是有点复杂去理解，所以我这里，选择封装一个简单的日志类，同样达到这个效果。

问题分析：

我们需要封装一个简单的日志类，主要有以下内容：

1. 生成的日志文件格式是 年月日时分秒.log
2. 生成的xxx.log文件存储在项目根目录下Logs文件夹下
3. 这个日志类，支持INFO,ERROR两种日志级别
4. 日志里，每行日志输出，如上图，时间日期+执行类名称+日志级别+日志描述

解决问题思路：

1. 在根目录下新建一个Logs的文件夹，如何获取这个Log的相对路径，前面介绍过。
2. 日志的保存命名，需要系统时间，前面也介绍过时间格式化输出
3. Python中有一个logging模块来支持我们自定义封装一个新日志类。
4. 在脚本里，初始化一个日志类的实例对象，然后去控制输出INFO还是ERROR日志信息。

自定义日志类封装如下：logger.py，新建在test包下

# 3.导入模块

import logging

import os.path

import time

class Logger(object):

def \_\_init\_\_(self, logger):

"""

指定保存日志的文件路径，日志级别，以及调用文件

将日志存入到指定的文件中

:param logger:

"""

# 创建一个logger

self.logger = logging.getLogger(logger)

self.logger.setLevel(logging.DEBUG)

# 创建一个handler，用于写入日志文件

rq = time.strftime('%Y%m%d%H%M', time.localtime(time.time()))

print(os.getcwd())

log\_path = (os.path.dirname(os.getcwd() + '\\Logs\\'))

print(log\_path)

log\_name = log\_path + rq + '.log'

fh = logging.FileHandler(log\_name)

fh.setLevel(logging.INFO)

# 再创建一个handler，用于输出到控制台

ch = logging.StreamHandler()

ch.setLevel(logging.INFO)

# 定义handler的输出格式

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

fh.setFormatter(formatter)

ch.setFormatter(formatter)

# 给logger添加handler

self.logger.addHandler(fh)

self.logger.addHandler(ch)

def getlog(self):

return self.logger

新建日志测试类

# 3.导入模块

import time

from selenium import webdriver

from test1.logger import Logger

mylogger = Logger(logger='test\_log').getlog()

class TestMyLog(object):

def print\_log(self):

driver = webdriver.Chrome()

mylogger.info("打开浏览器")

driver.maximize\_window()

mylogger.info("最大化浏览器窗口。")

driver.implicitly\_wait(8)

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

mylogger.info("打开百度首页。")

time.sleep(1)

mylogger.info("暂停一秒。")

driver.quit()

mylogger.info("关闭并退出浏览器。")

testlog = TestMyLog()

testlog.print\_log()

运行代码后，在Logs文件夹下可以看到日志文件，如下图的结果

#### 把截图类方法封装到前面的BasePage.py

本文介绍把截图类方法封装到BasePage.py文件里，这个文件是在前面Selenium方法二次封装文章里创建的，具体代码请到前面这篇里找。我们截图类写死了把截图图片保存到根目录下的Screenshots文件夹里，图片名称是当前系统时间，图片后缀名是png。

import os

import time

from test1.logger import Logger

mylog = Logger(logger='BasePage').getlog()

class BasePage(object):

"""

主要是把常用的几个Selenium方法封装到BasePage这个类，我们这里演示以下几个方法

back()

forward()

get()

quit()

"""

def \_\_init\_\_(self, driver):

"""

写一个构造函数，有一个参数driver

:param driver:

"""

self.driver = driver

def back(self):

"""

浏览器后退按钮

:param none:

"""

self.driver.back()

def forward(self):

"""

浏览器前进按钮

:param none:

"""

self.driver.forward()

def open\_url(self, url):

"""

打开url站点

:param url:

"""

self.driver.get(url)

def quit\_browser(self):

"""

关闭并停止浏览器服务

:param none:

"""

self.driver.quit()

def take\_screenshot(self):

"""

截图并保存在根目录下的Screenshots文件夹下

:param none:

"""

file\_path = os.path.dirname(os.getcwd()) + '/Screenshots/'

rq = time.strftime('%Y%m%d%H%M%S',time.localtime(time.time()))

screen\_name = file\_path + rq + '.png'

try :

self.driver.get\_screenshot\_as\_file(screen\_name)

mylog.info("开始截图并保存")

except Exception as e:

mylog.error("出现异常",format(e))

主要看最后一个截图类方法的封装。

测试类相关代码如下：

# 3.导入模块

import time

from selenium import webdriver

from blog.basepage import BasePage

class TestScreenshot(object):

driver = webdriver.Chrome()

driver.maximize\_window()

driver.implicitly\_wait(10)

basepage = BasePage(driver)

def test\_take\_screen(self):

self.basepage.open\_url("https://www.baidu.com")

time.sleep(1)

self.basepage.take\_screenshot()

self.basepage.quit\_browser()

test = TestScreenshot()

test.test\_take\_screen()

运行代码后，在Screenshots文件夹下可以看到截图文件

#### Python中的继承的使用

在test1包名下新建一个classA.py，这个就是我们的父类，里面有一个打开chrome浏览器和打开百度首页的方法。

# 3.导入模块

from selenium import webdriver

import time

class ClassA(object):

def open\_baidu(self):

driver = webdriver.Chrome()

driver.maximize\_window()

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

time.sleep(1)

driver.quit()

在test2包下新建一个classB.py文件，这个继承classA.py里的ClassA类。

# 3.导入模块

from test1.classA import ClassA

class ClassB(ClassA):

def test\_inherit(self):

self.open\_baidu()

test = ClassB()

test.test\_inherit()

中文乱码

解决办法：在FileHandler 此处要设置 encoding 格式，

fh = logging.FileHandler(log\_name,encoding='utf-8')

## 五、python+selenium框架设计篇

### 什么是自动化测试框架

框架基本组件：

1）需要配置文件管理

2）业务逻辑代码和测试脚本分离

3）报告和日志文件输出

4）自定义的库的封装

5）管理、执行脚本方式

6）第三方插件引入

7）持续集成



### unittest单元测试框架

unittest是一个单元测试框架，是Python编程的单元测试框架。unittest支持测试自动化，共享测试用例中的初始化和关闭退出代码，在unittest中最小单元是test，也就是一个测试用例。要了解unittest单元测试框架，先来了解以下几个重要的概念

#### 测试固件（test fixture）

一个测试固件包括两部分，执行测试代码之前的准备部分和测试结束之后的清扫代码。这两部分一般用函数setUp()和tearDown()表示。这里举例以下，例如要测试百度搜索selenium这个场景，我们的测试固件可以这样写，setUp()里写打开浏览器，浏览器最大

化，和打开百度首页等脚本代码；在tearDown（）里写结束搜索后，退出并关闭浏览器的代码。

#### 测试用例（test case）

 unittest中管理的最小单元是测试用例，一个测试用例，包括测试固件，和具体测试业务的函数或者方法。一个测试用例中，测试固件可以不写，但是至少有一个已test开头的函数。unittest会自动化识别test开头的函数是测试代码，如果你写的函数不是test开头，unittest是不会执行这个函数里面的脚本的，这个千万要记住，所有的测试函数都要test开头，记住是小写的哦。

#### 测试套件 （test suite）

很简单，就是很多测试用例的集合，叫测试套件，一个测试套件可以随意管理多个测试用例。如果测试用例比作单个学生，测试套件就是好像是班级的概念。

#### 测试执行器 （test runner）

test runner是一个用来执行加载测试用例，并执行用例，且提供测试输出的一个组建。test runner可以加载test case或者test suite进行执行测试任务。

我们举例来，练习一下test fixture和test case的使用，学习unittest的简单用法：

#### 设计思路

1)新建一个testbaidu.py的文件

2)导入unittest模块

3)当前测试类继承unittest.TestCase，相当于当前利用unittest创建了一个test case，这个test case是能够被unittest直接识别。

4)写setUP(),主要是打开浏览器和打开站点

5)写一个test\_search（）用例写搜索的代码

6)写tearDown(),主要是浏览器退出操作

相关脚本代码如下：

# 3.导入模块

import time

import unittest

from selenium import webdriver

class BaiduSearch(unittest.TestCase):

def setUp(self):

"""

测试固件的setUp()的代码，主要是测试的前提准备工作

:return:

"""

self.driver = webdriver.Chrome()

self.driver.maximize\_window()

self.driver.implicitly\_wait(8)

self.driver.get("https://www.baidu.com")

def tearDown(self):

"""

测试结束后的操作，这里基本上都是关闭浏览器

:return:

"""

self.driver.implicitly\_wait(8)

self.driver.quit()

def test\_baidu\_search(self):

"""

这里一定要test开头，把测试逻辑代码封装到一个test开头的方法里。

:return:

"""

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

time.sleep(1)

self.driver.find\_element\_by\_id('su').click()

time.sleep(3)

try:

assert 'selenium' in self.driver.title

print ('Test Pass.')

except Exception as e:

print ('Test Fail.', format(e))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

解释：  
        最后结尾处的unittest.main(),添加这个是支持在cmd，里面，cd到这个脚本文件所在的目录，然后python 脚本名.py执行，如果不添加这一段，是无法执行cmd里面运行脚本的，在PyCharm中，不添加最后一段，也可以通过，右键 Run "unittest xxx"，来达到执行效果。

### 什么是POM（Page Object Model）

前面我们介绍了Python中的单元测试框架unittest，以后我们所有的测试类文件，都采用unittest来辅助我们进行debug和脚本开发。搞定了debug机制和确定了unittest来进行创建和管理我们的自动化测试脚本，接下来我们来考虑下，框架设计中一种很普遍的设计思想-POM（Page Object Model）

#### POM是什么

Page Object Model (POM) 直译为“页面对象模型”，这种设计模式旨在为每个待测试的页面创建一个页面对象(class)，将那些繁琐的定位操作封装到这个页面对象中，只对外提供必要的操作接口。

#### POM 有什么好处

POM 将页面定位和业务操作分开，分离了测试对象和测试脚本，如果UI更改页面，测试脚本不需要更改，只需要更改页面对象中的某些代码就可以，提高了可维护性.

比如测试一个登陆页面：[新浪微博](https://links.jianshu.com/go?to=https%3A%2F%2Fweibo.com%2F) ，执行测试的人员传递不同的数据到帐号、密码框就可以了，而不应该去顾虑：页面是否已经加载完成？怎样定位到帐号输入框？怎样定位到登陆按钮等等问题。

# 3.导入模块

from selenium import webdriver

from selenium.webdriver.support.wait import WebDriverWait

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

class BasePage(object):

"""

基础页面，提供给其他页面继承

"""

def \_\_init\_\_(self, driver, base\_url, title):

"""

初始化

"""

self.driver = driver

self.base\_url = base\_url

self.title = title

def \_open(self, url):

"""

私有方法，打开url参数指定的页面,

并检查打开是否正确

"""

self.driver.get(url)

# 显式等待10秒，如果打开页title与预期不符或者超时，抛出异常

WebDriverWait(self.driver, 10).until(EC.title\_is(self.title))

def open(self):

"""

公共方法，调用私有方法\_open()打开链接

"""

self.\_open(self.base\_url)

def find\_element(self, \*loc):

"""

定位指定元素

"""

# 显式等待元素，超过10秒未找到则抛出超时异常(TimeoutException)

# presence\_of\_element\_located： 不关心元素是否可见，只关心元素是否存在在页面中

# visibility\_of\_element\_located： 不仅找到元素，并且该元素必须可见

WebDriverWait(self.driver, 15).until(EC.visibility\_of\_element\_located(loc))

return self.driver.find\_element(\*loc)

if \_\_name\_\_ == '\_\_main\_\_':

driver = webdriver.Chrome()

driver.maximize\_window()

page = BasePage(driver, 'https://www.baidu.com/','百度一下，你就知道')

page.open()

driver.quit()

再定义一个 LoginPage类 继承 BasePage类 ，向外提供测登陆方法。文件命名为 xl\_login.py

# 3.导入模块

from selenium import webdriver

from selenium.webdriver.common.by import By

from basePage import BasePage

class LoginPage(BasePage):

"""

新浪微薄登陆页面

继承自基础页面BasePage

"""

# 定位帐号输入框

username\_loc = (By.ID, 'loginname')

# 定位密码输入框

password\_loc = (By.NAME, 'password')

# 定位登陆按钮

submit\_loc = (By.XPATH, './/\*[@id="pl\_login\_form"]/div/div[3]/div[6]/a')

# 定位提示信息，如：请输入验证码

# 不要迷信开发者工具提供的Xpath，

# 比如这里提供的Xpath：//\*[@id="layer\_15582553868501"]/div/p/span[2]，

# id是动态的，无法使用，需自行推到Xpath

message\_loc = (By.XPATH, '//div[@class="content layer\_mini\_info"]/p/span[2]')

# 输入用户名操作

def type\_username(self, username):

self.find\_element(\*self.username\_loc).send\_keys(username)

# 输入密码操作

def type\_password(self, password):

self.find\_element(\*self.password\_loc).send\_keys(password)

# 点击登陆按钮操作

def submit(self):

self.find\_element(\*self.submit\_loc).click()

# 获取提示信息

def get\_message(self):

return self.find\_element(\*self.message\_loc).text

if \_\_name\_\_ == '\_\_main\_\_': # 测试登陆

# 预打开页面

base\_url = 'https://weibo.com/'

# 页面title

title = '微博-随时随地发现新鲜事'

# 准备好待输入的用户名和密码

username = 'haha'

password = 'hehe'

# 打开Chrome浏览器

driver = webdriver.Chrome()

driver.maximize\_window()

# 登陆页面初始化

login = LoginPage(driver, base\_url, title)

# 打开新浪微博页

login.open()

# 输入用户名

login.type\_username(username)

# 输入密码

login.type\_password(password)

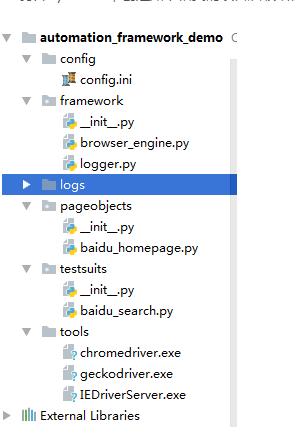
# 点击登陆

login.submit()

# 打印提示信息

print(login.get\_message())

### 创建项目层级结构



### 相关文件代码

#### config.ini 配置文件

# this is config file, only store browser type and server URL

[browserType]

#browserName = Firefox

browserName = Chrome

#browserName = IE

[testServer]

URL = https://www.baidu.com

#URL = www.google.com

#### 封装浏览器驱动（引擎）类

##### browser\_engine.py 文件

主要目前常用的Chrome、Firefox和IE三大浏览器引擎的封装

浏览器引擎类browser\_engine.py

# 3.导入模块

import configparser

import os.path

from selenium import webdriver

from automation\_framework\_demo.framework.logger import Logger

logger = Logger(logger="BrowserEngine").getlog()

class BrowserEngine(object):

dir = os.path.dirname(os.path.abspath('.')) # 注意相对路径获取方法

chrome\_driver\_path = dir + '/tools/chromedriver.exe'

ie\_driver\_path = dir + '/tools/IEDriverServer.exe'

def \_\_init\_\_(self, driver):

self.driver = driver

# read the browser type from config.ini file, return the driver

def open\_browser(self, driver):

config = configparser.ConfigParser()

# file\_path = os.path.dirname(os.getcwd()) + '/config/config.ini'

file\_path = os.path.dirname(os.path.abspath('.')) + '/config/config.ini'

config.read(file\_path)

browser = config.get("browserType", "browserName")#获取浏览器类型、名字

logger.info("You had select %s browser." % browser) #日志打印你选择的浏览器

url = config.get("testServer", "URL") #获取测试的URL地址

logger.info("The test server url is: %s" % url) #日志打印测试的URL地址

#判断你所选择的浏览器

if browser == "Firefox":

driver = webdriver.Firefox()

logger.info("Starting firefox browser.")

elif browser == "Chrome":

driver = webdriver.Chrome(self.chrome\_driver\_path)#初始化一个实例

logger.info("Starting Chrome browser.")

elif browser == "IE":

driver = webdriver.Ie(self.ie\_driver\_path)

logger.info("Starting IE browser.")

driver.get(url)#访问URL

logger.info("Open url: %s" % url)

driver.maximize\_window() #将窗口放大

logger.info("Maximize the current window.")

driver.implicitly\_wait(10)

logger.info("Set implicitly wait 10 seconds.")

print(driver)

return driver

#关闭浏览器

def quit\_browser(self):

logger.info("Now, Close and quit the browser.")

self.driver.quit()

#### 封装日志类

##### logger.py文件

# 3.导入模块

import logging

import logging.handlers

import os.path

import time

class Logger(object):

def \_\_init\_\_(self, logger):

'''''

指定保存日志的文件路径，日志级别，以及调用文件

将日志存入到指定的文件中

'''

# 创建一个日志器logger，并设置其日志级别为DEBUG

self.logger = logging.getLogger(logger)

self.logger.setLevel(logging.DEBUG)

# 创建一个handler，用于写入日志文件

rq = time.strftime('%Y%m%d%H%M', time.localtime(time.time()))

# log\_path = os.path.dirname(os.getcwd()) + '/Logs/' # 项目根目录下/Logs 保存日志

log\_path = os.path.dirname(os.path.abspath('.')) + '/logs/'

# 如果case组织结构式 /testsuit/featuremodel/xxx.py ， 那么得到的相对路径的父路径就是项目根目录

log\_name = log\_path + rq + '.log'

# 创建一个文件处理器handler并设置其日志级别为INFO

#fh = logging.FileHandler(log\_name, maxBytes=1024 \* 1024, backupCount=5,

# encoding='utf-8')

fh = logging.handlers.RotatingFileHandler(log\_name, maxBytes=1024 \* 1024, backupCount=5,

encoding='utf-8') # 实例化handler

#fh = logging.FileHandler(log\_name)

fh.setLevel(logging.INFO)

# 再创建一个handler，用于输出到控制台

'''

创建一个流处理器handler并设置其日志级别为INFO

'''

ch = logging.StreamHandler()

ch.setLevel(logging.INFO)

# 定义handler的输出格式

#handler = logging.handlers.RotatingFileHandler(fh, maxBytes=1024 \* 1024, backupCount=5,

# encoding='utf-8') # 实例化handler

'''

创建一个格式器formatter并将

'''

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

fh.setFormatter(formatter)

ch.setFormatter(formatter)

# 给日志处理器logger添加上面创建的handler

self.logger.addHandler(fh)

self.logger.addHandler(ch)

def getlog(self):

return self.logger

### 新建测试脚本

#### 测试脚本baidu\_search.py

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

class BaiduSearch(unittest.TestCase):

def setUp(self):

"""

测试固件的setUp()的代码，主要是测试的前提准备工作

:return:

"""

browse = BrowserEngine(self)

self.driver = browse.open\_browser(self)

def tearDown(self):

"""

测试结束后的操作，这里基本上都是关闭浏览器

:return:

"""

self.driver.quit()

def test\_baidu\_search(self):

"""

这里一定要test开头，把测试逻辑代码封装到一个test开头的方法里。

:return:

"""

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

time.sleep(1)

self.driver.find\_element\_by\_id('su').click()

time.sleep(5)

try:

assert 'selenium' in self.driver.title

print ('Test Pass.')

except Exception as e:

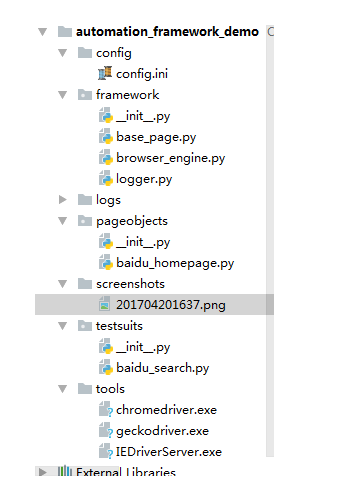
print ('Test Fail.', format(e))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

前面文章，我们实现了框架的一部分功能，包括日志类和浏览器引擎类的封装，今天我们继续封装一个基类和介绍如何实现POM。关于基类，是这样定义的：把一些常见的页面操作的selenium封装到base\_page.py这个类文件，以后每个POM中的页面类，都继承这个基类，这样每个页面类都有基类的方法，这个我们会在这篇文章由宏哥实现。

### 项目层级结构



### 定位和截图类封装

1. 在实现封装基类里，我们实现了元素八大方式的定位和截图类封装。

2. 基类base\_page.py的具体实现代码，这里就封装了几个常用方法，其他方法，你自己去练习封装下。

# 3.导入模块

import time

from selenium.common.exceptions import NoSuchElementException

import os.path

from automation\_framework\_demo.framework.logger import Logger

# create a logger instance

logger = Logger(logger="BasePage").getlog()

class BasePage(object):

"""

定义一个页面基类，让所有页面都继承这个类，封装一些常用的页面操作方法到这个类

"""

def \_\_init\_\_(self, driver):

self.driver = driver

# quit browser and end testing

def quit\_browser(self):

self.driver.quit()

# 浏览器前进操作

def forward(self):

self.driver.forward()

logger.info("Click forward on current page.")

# 浏览器后退操作

def back(self):

self.driver.back()

logger.info("Click back on current page.")

# 隐式等待

def wait(self, seconds):

self.driver.implicitly\_wait(seconds)

logger.info("wait for %d seconds." % seconds)

# 点击关闭当前窗口

def close(self):

try:

self.driver.close()

logger.info("Closing and quit the browser.")

except NameError as e:

logger.error("Failed to quit the browser with %s" % e)

# 保存图片

def get\_windows\_img(self):

"""

在这里我们把file\_path这个参数写死，直接保存到我们项目根目录的一个文件夹.\Screenshots下

"""

file\_path = os.path.dirname(os.path.abspath('.')) + '/screenshots/'

rq = time.strftime('%Y%m%d%H%M', time.localtime(time.time()))

screen\_name = file\_path + rq + '.png'

try:

self.driver.get\_screenshot\_as\_file(screen\_name)

logger.info("Had take screenshot and save to folder : /screenshots")

except NameError as e:

logger.error("Failed to take screenshot! %s" % e)

self.get\_windows\_img()

# 定位元素方法

def find\_element(self, selector):

"""

这个地方为什么是根据=>来切割字符串，请看页面里定位元素的方法

submit\_btn = "id=>su"

login\_lnk = "xpath => //\*[@id='u1']/a[7]" # 百度首页登录链接定位

如果采用等号，结果很多xpath表达式中包含一个=，这样会造成切割不准确，影响元素定位

:param selector:

:return: element

"""

element = ''

if '=>' not in selector:

return self.driver.find\_element\_by\_id(selector)

selector\_by = selector.split('=>')[0]

selector\_value = selector.split('=>')[1]

print(selector\_value)

if selector\_by == "i" or selector\_by == 'id':

try:

element = self.driver.find\_element\_by\_id(selector\_value)

logger.info("Had find the element \' %s \' successful "

"by %s via value: %s " % (element.text, selector\_by, selector\_value))

except NoSuchElementException as e:

logger.error("NoSuchElementException: %s" % e)

self.get\_windows\_img() # take screenshot

elif selector\_by == "n" or selector\_by == 'name':

element = self.driver.find\_element\_by\_name(selector\_value)

elif selector\_by == "c" or selector\_by == 'class\_name':

element = self.driver.find\_element\_by\_class\_name(selector\_value)

elif selector\_by == "l" or selector\_by == 'link\_text':

element = self.driver.find\_element\_by\_link\_text(selector\_value)

elif selector\_by == "p" or selector\_by == 'partial\_link\_text':

element = self.driver.find\_element\_by\_partial\_link\_text(selector\_value)

elif selector\_by == "t" or selector\_by == 'tag\_name':

element = self.driver.find\_element\_by\_tag\_name(selector\_value)

elif selector\_by == "x" or selector\_by == 'xpath':

try:

element = self.driver.find\_element\_by\_xpath(selector\_value)

logger.info("Had find the element \' %s \' successful "

"by %s via value: %s " % (element.text, selector\_by, selector\_value))

except NoSuchElementException as e:

logger.error("NoSuchElementException: %s" % e)

self.get\_windows\_img()

elif selector\_by == "s" or selector\_by == 'selector\_selector':

element = self.driver.find\_element\_by\_css\_selector(selector\_value)

else:

raise NameError("Please enter a valid type of targeting elements.")

print(element)

return element

# 输入

def type(self, selector, text):

el = self.find\_element(selector)

el.clear()

try:

el.send\_keys(text)

logger.info("Had type \' %s \' in inputBox" % text)

except NameError as e:

logger.error("Failed to type in input box with %s" % e)

self.get\_windows\_img()

# 清除文本框

def clear(self, selector):

el = self.find\_element(selector)

try:

el.clear()

logger.info("Clear text in input box before typing.")

except NameError as e:

logger.error("Failed to clear in input box with %s" % e)

self.get\_windows\_img()

# 点击元素

def click(self, selector):

el = self.find\_element(selector)

try:

el.click()

logger.info("The element \' %s \' was clicked." % el)

except NameError as e:

logger.error("Failed to click the element with %s" % e)

# 或者网页标题

def get\_page\_title(self):

logger.info("Current page title is %s" % self.driver.title)

return self.driver.title

@staticmethod

def sleep(seconds):

time.sleep(seconds)

logger.info("Sleep for %d seconds" % seconds)

### pageObjects文件夹下相关代码

页面对象中，百度主页的元素定位和简单的操作函数，页面类主要是元素定位和页面操作写成函数，供测试类调用。baidu\_homepage.py

from automation\_framework\_demo.framework.base\_page import BasePage

class HomePage(BasePage):

input\_box = "id=>kw"

search\_submit\_btn = "xpath=>//\*[@id='su']"

#百度新闻入口

#news\_link = "xpath=>//\*[@id='u1']/a[@name='tj\_trnews']"

news\_link = "xpath=>//\*[@id='u1']/a[@name='tj\_trnews']"

def type\_search(self, text):

self.type(self.input\_box, text)

def send\_submit\_btn(self):

self.click(self.search\_submit\_btn)

def click\_news(self,):

self.click(self.news\_link)

self.sleep(2)

### 测试类的写法举例

新建一个测试类baidu\_search1.py

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

class BaiduSearch(unittest.TestCase):

def setUpClass(cls):

"""

测试固件的setUp()的代码，主要是测试的前提准备工作

:return:

"""

browse = BrowserEngine(cls)

cls.driver = browse.open\_browser(cls)

def tearDownClass(cls):

"""

测试结束后的操作，这里基本上都是关闭浏览器

:return:

"""

cls.driver.quit()

def test\_baidu\_search(self):

"""

这里一定要test开头，把测试逻辑代码封装到一个test开头的方法里。

:return:

"""

# self.driver.find\_element\_by\_id('kw').send\_keys('selenium')

# time.sleep(1)

homepage = HomePage(self.driver)

homepage.type\_search('selenium') # 调用页面对象中的方法

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(2)

homepage.get\_windows\_img() # 调用基类截图方法

print(self.driver.title)

try:

assert('selenium' in HomePage.get\_page\_title(self))

print('Test Pass.')

except Exception as e:

print('Test Fail.', format(e))

def test\_search2(self):

homepage = HomePage(self.driver)

homepage.type\_search('python') # 调用页面对象中的方法

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(2)

homepage.get\_windows\_img() # 调用基类截图方法

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

#### 代码说明

homepage = HomePage(self.driver)

 上面这行代码要注意，意思是：到一个页面，第一件事情是初始化这个页面的一个页面对象实例。注意，一定要带self.driver，不然会报错，这个self.driver，可以这样理解：在当前测试类里面，self.driver是来自浏览器引擎类中方法得到的，在初始化一个页面对象

的时候，也把这个来自浏览器引擎类的driver给赋值给当前的页面对象，这样，才能执行页面对象或者基类里面的相关driver方法。写多了selenium的自动化脚本，你会明白，最重要的是保持前后driver的唯一性。

#### 生成图片

测试结果：会在logs文件夹生成一个日志文件，也会在screenshots文件夹生成一个png图片。日志看过了，这里我们看一下图片。

接下来的主要是介绍，unittest管理脚本，如何如何加载执行脚本，再就是采用第三方插件，实现输出html的测试报告。本文来介绍下，在同一个类中，多个测试函数时候，测试固件如何写和进一步实现POM和可能遇到问题解决办法。

### 一个类文件多个测试方法情况下测试固件的写法

为了说明这个问题，我们在之前的测试类基础上，再写一个test\_search2()的测试用例，看看会发生什么

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

class BaiduSearch(unittest.TestCase):

def setUp(self):

"""

测试固件的setUp()的代码，主要是测试的前提准备工作

:return:

"""

browse = BrowserEngine(self)

self.driver = browse.open\_browser(self)

def tearDown(self):

"""

测试结束后的操作，这里基本上都是关闭浏览器

:return:

"""

self.driver.quit()

def test\_baidu\_search(self):

"""

这里一定要test开头，把测试逻辑代码封装到一个test开头的方法里。

:return:

"""

homepage = HomePage(self.driver)

homepage.type\_search('selenium') # 调用页面对象中的方法

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(2)

homepage.get\_windows\_img() # 调用基类截图方法

try:

assert 'selenium' in homepage.get\_page\_title() # 调用页面对象继承基类中的获取页面标题方法

print('Test Pass.')

except Exception as e:

print('Test Fail.', format(e))

def test\_search2(self):

homepage = HomePage(self.driver)

homepage.type\_search('python') # 调用页面对象中的方法

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(2)

homepage.get\_windows\_img() # 调用基类截图方法

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

 问题发现了没，我们的浏览器启动和关闭了两次，是不是这个问题？其实细心地小伙伴或者童鞋们在上一篇文章里就可能发现这个问题了，由于时间的关系宏哥在这里把它单独拿出来分享讲解一下，希望可以加深小伙伴或者童鞋们的印象。问题是原来每执行一次

test开头的函数，都要执行一次测试固件，也就是说执行setUp()和()一次，如果有N个test开头的函数，测试固件就执行N次，我们到底有没有，只需要执行一次测试固件，支持执行多次测试函数。

     我们测试中，肯定需要，打开一个页面，然后测试这个页面的多个用例，才关闭这个页面，去测试其他页面，在unittest是有相关测试固件方法去支持这种行为。请看下面调整，自己对比下，能不能找出不同。

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

class BaiduSearch(unittest.TestCase):

@classmethod

def setUpClass(cls):

"""

测试固件的setUp()的代码，主要是测试的前提准备工作

:return:

"""

browse = BrowserEngine(cls)

cls.driver = browse.open\_browser(cls)

@classmethod

def tearDownClass(cls):

"""

测试结束后的操作，这里基本上都是关闭浏览器

:return:

"""

cls.driver.quit()

def test\_baidu\_search(self):

"""

这里一定要test开头，把测试逻辑代码封装到一个test开头的方法里。

:return:

"""

# self.driver.find\_element\_by\_id('kw').send\_keys('selenium')

# time.sleep(1)

homepage = HomePage(self.driver)

homepage.type\_search('selenium') # 调用页面对象中的方法

time.sleep(12)

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(12)

homepage.get\_windows\_img() # 调用基类截图方法

print(self.driver.title)

try:

assert('selenium' in HomePage.get\_page\_title(self))

print('Test Pass.')

except Exception as e:

print('Test Fail.', format(e))

def test\_search2(self):

homepage = HomePage(self.driver)

homepage.type\_search('python') # 调用页面对象中的方法

homepage.send\_submit\_btn() # 调用页面对象类中的点击搜索按钮方法

time.sleep(2)

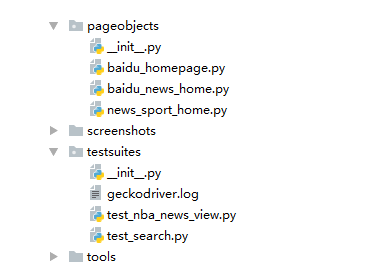
homepage.get\_windows\_img() # 调用基类截图方法

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

### 进一步实现POM

本小节宏哥将会进一步演示POM的具体实现，前面POM只是一个页面，一个测试脚本，现在我们要实现三个页面，两个测试脚本。在pageobjects包下，我新建了2个页面对象：百度新闻首页，百度体育新闻首页，具体文件结构如下图，其他和之前项目层级结构保持不变



 百度首页页面类代码（baidu\_homepage.py），定义了百度新闻的入口

# 3.导入模块

from automation\_framework\_demo.framework.base\_page import BasePage

class HomePage(BasePage):

input\_box = "id=>kw"

search\_submit\_btn = "xpath=>//\*[@id='su']"

#百度新闻入口

#news\_link = "xpath=>//\*[@id='u1']/a[@name='tj\_trnews']"

news\_link = "xpath=>//\*[@id='u1']/a[@name='tj\_trnews']"

def type\_search(self, text):

self.type(self.input\_box, text)

def send\_submit\_btn(self):

self.click(self.search\_submit\_btn)

def click\_news(self,):

self.click(self.news\_link)

self.sleep(2)

百度新闻首页的页面类代码（baidu\_news\_home.py）,定义了体育新闻入口

# 3.导入模块

from automation\_framework\_demo.framework.base\_page import BasePage

class NewsHomePage(BasePage):

#点击体育新闻入口

sports\_link = "xpath=>//\*[@id='channle-all']/div/ul/li[7]/a"

def click\_sports(self):

self.click(self.sports\_link)

self.sleep(2)

百度体育新闻页面类代码（news\_sports\_home.py）

# 3.导入模块

from automation\_framework\_demo.framework.base\_page import BasePage

class SportsNewsHomePage(BasePage):

#NBA入口

nba\_link = "xpath=>.//\*[@id='col\_focus']/div[1]/div[2]/div/div[2]/div/ul/li[1]/a"

def click\_nba\_link(self):

self.click(self.nba\_link)

self.sleep(2)

### 测试类代码（test\_nba\_news\_view.py）

测试步骤大概是：百度首页点击新闻链接-进入新闻主页，点击体育-进入体育新闻主页，点击NBA-进入NBA页面-其他后续脚本操作。为什么要采用这样的步骤呢，干嘛不直接driver.get('nba的链接')？因为我们就是要利用POM的思想去写我们测试脚本，才有上面的测试步骤

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

from automation\_framework\_demo.pageobjects.baidu\_news\_home import NewsHomePage

from automation\_framework\_demo.pageobjects.news\_sport\_home import SportsNewsHomePage

class ViewNBANews(unittest.TestCase):

def setUp(self):

browse = BrowserEngine(self)

self.driver = browse.open\_browser(self)

def tearDown(self):

self.driver.quit()

def test\_view\_nba\_views(self):

# 初始化百度首页，并点击新闻链接

baiduhome = HomePage(self.driver)

baiduhome.click\_news()

# 初始化一个百度新闻主页对象，点击体育

newshome = NewsHomePage(self.driver)

newshome.click\_sports()

# 初始化一个体育新闻主页，点击NBA

sportnewhome = SportsNewsHomePage(self.driver)

sportnewhome.click\_nba\_link()

sportnewhome.get\_windows\_img()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

我们调整下我们测试类代码，添加find\_element（）语句

# 3.导入模块

import time

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

from automation\_framework\_demo.pageobjects.baidu\_news\_home import NewsHomePage

from automation\_framework\_demo.pageobjects.news\_sport\_home import SportsNewsHomePage

class ViewNBANews(unittest.TestCase):

def setUp(self):

browse = BrowserEngine(self)

self.driver = browse.open\_browser(self)

def tearDown(self):

self.driver.quit()

def test\_view\_nba\_views(self):

# 初始化百度首页，并点击新闻链接

baiduhome = HomePage(self.driver)

# baiduhome.click\_news()

self.driver.find\_element\_by\_xpath("//\*[@id='u1']/a[@name='tj\_trnews']").click()

# 初始化一个百度新闻主页对象，点击体育

newshome = NewsHomePage(self.driver)

# newshome.click\_sports()

self.driver.find\_element\_by\_xpath("//\*[@id='channel-all']/div/ul/li[7]/a").click()

# 初始化一个体育新闻主页，点击NBA

sportnewhome = SportsNewsHomePage(self.driver)

# sportnewhome.click\_nba\_link()

self.driver.find\_element\_by\_xpath(".//\*[@id='col\_focus']/div[1]/div[2]/div/div[2]/div/ul/li[1]/a").click()

sportnewhome.get\_windows\_img()

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

本文开始介绍如何通过unittest来管理和执行测试用例，这一篇主要是介绍unittest下addTest()方法来加载测试用例到测试套件中去、用addTest()方法来加载我们测试用例到suite中去和利用discover（）方法去加载一个路径下所有的测试用例

### addTest()方法

这里首先介绍unittest下addTest()方法来加载测试用例到测试套件中去。为了演示效果，我在前面文章的脚本基础上，新建了一个测试脚本，这个测试脚本有一个测试用例，加上前面的测试脚本，一共就是三个测试用例，前面那个测试类里面包括两个测试用例。

新建的测试脚本，test\_get\_page\_title.py代码如下：

# 3.导入模块

import unittest

from automation\_framework\_demo.framework.browser\_engine import BrowserEngine

from automation\_framework\_demo.pageobjects.baidu\_homepage import HomePage

class GetPageTitle(unittest.TestCase):

@classmethod

def setUpClass(cls):

browser = BrowserEngine(cls)

cls.driver = browser.open\_browser(cls)

@classmethod

def tearDownClass(cls):

cls.driver.quit()

def test\_get\_title(self):

homepage = HomePage(self.driver)

print(homepage.get\_page\_title())

今天来学习下，如果加载这两个测试类文件里面的3个测试脚本。在unittest下有一个管理测试套件的叫TestSuit(),我们要使用这个测试套件，需要先初始化一个suite实例，然后这个实例有一个addTest()的方法，可以加载不同类里面的不同测试函数，格式这样的 addTest(测试类的类名（‘测试函数名称，就是test开头的函数’）)，我们在testsuites这个包下新建一个TestRunner.py文件用来管理我们的用例启动方式。  
来看看这个TestRunner.py的内容

# 3.导入模块

import unittest

#from automation\_framework\_demo.testsuits import testsuits

from automation\_framework\_demo.testsuits.baidu\_search2 import BaiduSearch

from automation\_framework\_demo.testsuits.test\_get\_page\_title import GetPageTitle

suite = unittest.TestSuite()

suite.addTest(BaiduSearch('test\_baidu\_search'))

suite.addTest(BaiduSearch('test\_search2'))

suite.addTest(GetPageTitle('test\_get\_title'))

if \_\_name\_\_ == '\_\_main\_\_':

# 执行用例

runner = unittest.TextTestRunner()

runner.run(suite)

这个方法有没有局限性呢？当然有，加入你有几百个测试类，你是不是也一直这样手动去添加？有没有其他更好的方法去加载我们的测试脚本，关于这个问题，请紧跟宏哥的脚步继续往下看、往下走。

前面文章介绍了如何用addTest()方法来加载我们测试用例到suite中去，本文介绍利用makeSuite()方法，一次性加载一个类文件下所有测试用例到suite中去。还是利用前面一篇文章中的测试脚本，来看看TestRunner.py如何写。

# 3.导入模块

import unittest

#from automation\_framework\_demo.testsuits import testsuits

from automation\_framework\_demo.testsuits.baidu\_search2 import BaiduSearch

from automation\_framework\_demo.testsuits.test\_get\_page\_title import GetPageTitle

'''

suite = unittest.TestSuite()

suite.addTest(BaiduSearch('test\_baidu\_search'))

suite.addTest(BaiduSearch('test\_search2'))

suite.addTest(GetPageTitle('test\_get\_title'))

'''

suite =unittest.TestSuite(unittest.makeSuite(BaiduSearch))

if \_\_name\_\_ == '\_\_main\_\_':

# 执行用例

runner = unittest.TextTestRunner()

runner.run(suite)

缺陷: makeSuite()方法虽然比前面文章的addTest()方法有了一定的效率提升，在添加测试用例到测试套件过程。但是这个方法也有很多缺点，我不可能把所有项目脚本都放一个测试类文件吧。在unittest中还是有很多方法可以用来添加测试用例到suite中，makeSuite（类名称），只是一种，你可以多看看unittest的文档介绍。接下来我们介绍如何一次性加载一个文件目录下所有测试用例。

### discover()方法

继续接着介绍，如何利用unittest管理和执行测试用例的问题，这里我们还是利用之前已经有的三条测试用例，如果你跳过了前面文章，请回到框架设计篇的第八篇和第七篇，里面有相关测试类的文件。本文来介绍，如果利用discover（）方法去加载一个路径下所有的测试用例。这个方法是不是太适合我们现在的需求，例如我只需要跑testsuite包下所有测试用例，不管在里面有多少个包和多少个测试类，只需要用discover()搞定，来看看这个场景下TestRunner.py如何写。

# 3.导入模块

import unittest

suite = unittest.TestLoader().discover("testsuits")

if \_\_name\_\_ == '\_\_main\_\_':

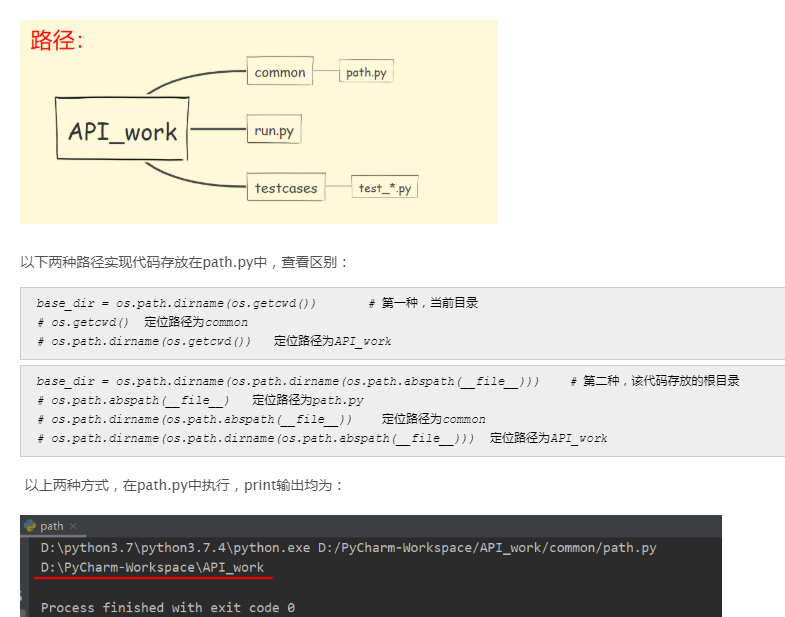
# 执行用例

runner = unittest.TextTestRunner()

runner.run(suite)

上面的testsuits是可以包名，也可以是一个文件夹名称，在实际脚本开发过程中，最后都采用这个方法来批量管理和执行几百上千的测试用例。下一篇内容，大概会介绍如何利用HTMLTestRunner.py这个开源的插件，生成测试报告，这个测试报告是一个HTML的格式。介绍完了生成报告文章，框架设计系列教程就会结束，不打算过多介绍其他技能点。

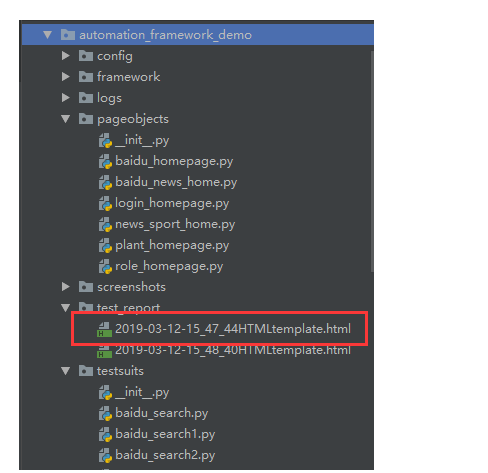
介绍一下路径：



**其他模块会调用固定路径时，不要使用 os.getcwd() 当前定位方式来实现；使用 os.path.abspath(\_\_file\_\_) 实现**

### 测试报告效果

#### 测试报告文件



#### 浏览器预览效果

#### 实现

这里我们报告输出代码写到了，负责执行测试套件TestRunner.py这个文件。相关代码如下：

# 3.导入模块

from automation\_framework\_demo.testsuits import HTMLTestReportCN

import os

import unittest

import time

from automation\_framework\_demo.framework.SendEmail import SendMail

# 设置报告文件保存路径

report\_path = os.path.dirname(os.path.abspath('.')) + '/test\_report/'

print('report\_path'+ report\_path)

# 获取系统当前时间

now = time.strftime("%Y-%m-%d-%H\_%M\_%S", time.localtime(time.time()))

# 设置报告名称格式

HtmlFile = report\_path + now + "HTMLtemplate.html"

#fp = file(HtmlFile, "wb")

fp = open(HtmlFile, "wb")

#用例路径

case\_path = os.path.join(os.getcwd(), '')

print('case\_path',case\_path)

# 构建suite

# suite = unittest.TestLoader().discover(case\_path, "test\_baidu\_search.py", top\_level\_dir=None)

suite = unittest.TestLoader().discover(case\_path, "baidu\_search1.py", top\_level\_dir=None)

if \_\_name\_\_ == '\_\_main\_\_':

#初始化一个HTMLTestRunner实例对象，用来生成报告

runner = HTMLTestReportCN.HTMLTestRunner(stream=fp, title=u"Python+Selenium自动化测试框架实战篇7项目演示测试报告", description=u"用例测试情况")

#开始执行测试套件

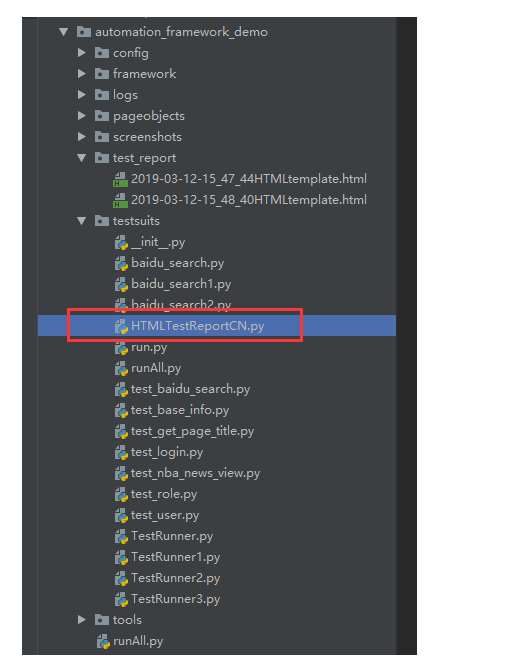
runner.run(suite)

# runner = unittest.TextTestRunner()

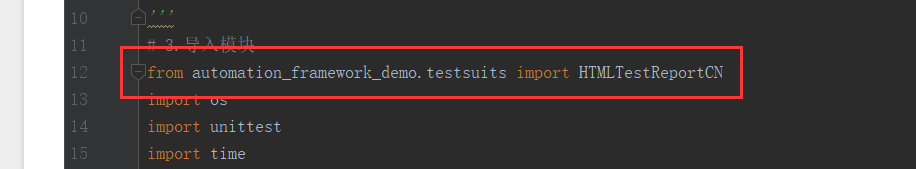
# runner.run(suite)

fp.close()

### 测试报告模板插件



### 导入测试报告模板插件



### 执行所有测试用例

#### 命名规则：

所有测试用例的名字都以test开头，例如：test\_login.py。

#### 正则表达式：

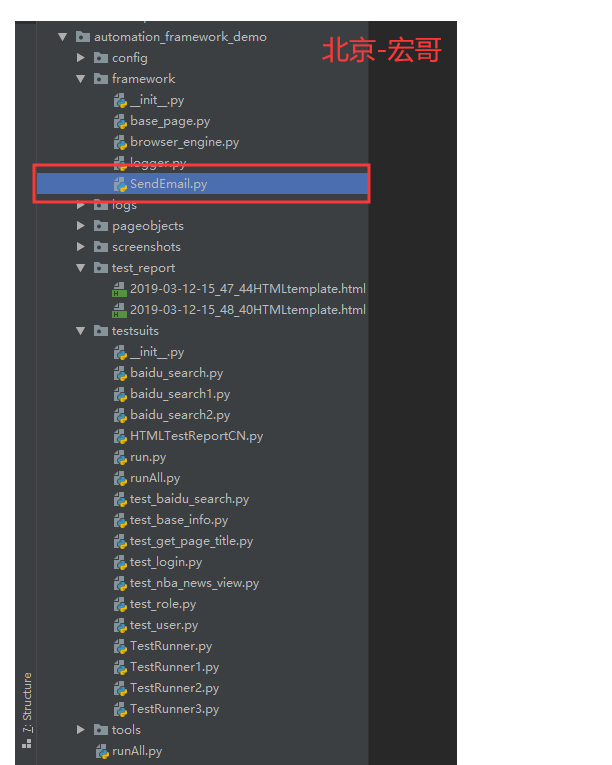
正则表达式：test\*.py

# 构建suite

# suite = unittest.TestLoader().discover(case\_path, "test\_baidu\_search.py", top\_level\_dir=None)

suite = unittest.TestLoader().discover(case\_path, "test\*.py", top\_level\_dir=None)

### 框架结构



### 邮件模块封装

# 3.导入模块

import os, sys

import smtplib

import time

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

#reportPath = os.path.join(os.getcwd(), 'test\_report') # 测试报告的路径

reportPath = os.path.dirname(os.path.abspath('.')) + '/test\_report/'

# reportPath = "E:\\untitled\\automation\_framework\_demo\\test\_report" # 测试报告的路径

print("打印路径：")

print(reportPath)

class SendMail(object):

def get\_report(self): # 该函数的作用是为了在测试报告的路径下找到最新的测试报告

dirs = os.listdir(reportPath)

dirs.sort()

newreportname = dirs[-1]

print('The new report name: {0}'.format(newreportname))

return newreportname # 返回的是测试报告的名字

def take\_messages(self): # 该函数的目的是为了 准备发送邮件的的消息内容

newreport = self.get\_report()

self.msg = MIMEMultipart()

self.msg['Subject'] = 'BBA JIT自动化测试报告' # 邮件的标题

self.msg['date'] = time.strftime('%a, %d %b %Y %H:%M:%S %z')

with open(os.path.join(reportPath, newreport), 'rb') as f:

mailbody = f.read() # 读取测试报告的内容

html = MIMEText(mailbody, \_subtype='html', \_charset='utf-8') # 将测试报告的内容放在 邮件的正文当中

self.msg.attach(html) # 将html附加在msg里

# html附件 下面是将测试报告放在附件中发送

att1 = MIMEText(mailbody, 'base64', 'utf-8')

att1["Content-Type"] = 'application/octet-stream'

att1["Content-Disposition"] = 'attachment; filename="TestReport.html"' # 这里的filename可以任意写，写什么名字，附件的名字就是什么

self.msg.attach(att1)

def send(self):

# recipients = ['xxxx@xxxx.com', 'xxxx@qq.com', 'xxx@xxxxx.com'] # 发送给多个人

recipients = ['hgxxx@nnnx.cn'] #发送给一个人

self.take\_messages()

self.msg['from'] = 'hg@xxx.cn' # 发送邮件的人，这种是公司邮箱转发

#self.msg['to'] = recipients # 收件人和发送人必须这里定义一下，执行才不会报错。

toaddrs = recipients

smtp = smtplib.SMTP()

smtp.connect('smtp.mxhichina.com')

smtp.ehlo()

smtp.login('hg@xxx.cn', '\*\*\*\*\*\*\*xxxx')

smtp.sendmail(self.msg['from'], toaddrs, self.msg.as\_string()) # 发送邮件

smtp.close()

print('sendmail success')

if \_\_name\_\_ == '\_\_main\_\_':

sendMail = SendMail()

sendMail.send()

### 查看收件箱