

15 基于Python的自动化测试

<https://seleniumhq.github.io/selenium/docs/api/py/#>

<https://seleniumhq.github.io/selenium/docs/api/py/api.html>

本章大纲

15.1 环境搭建

15.2 Selenium常用API

15.3 Unittest单元测试框架

15.4 Page Object设计模式

安装Python

- Python可应用于多平台包括windows、 Linux 和 Mac OS X。
- Python官网: <http://www.python.org/>
- 可以在以下链接中下载Python 的文档, 你可以下载 HTML、PDF 和 PostScript 等格式的文档。
- Python文档下载地址: www.python.org/doc/

安装Python

<https://www.python.org/downloads/windows/>

Stable Releases

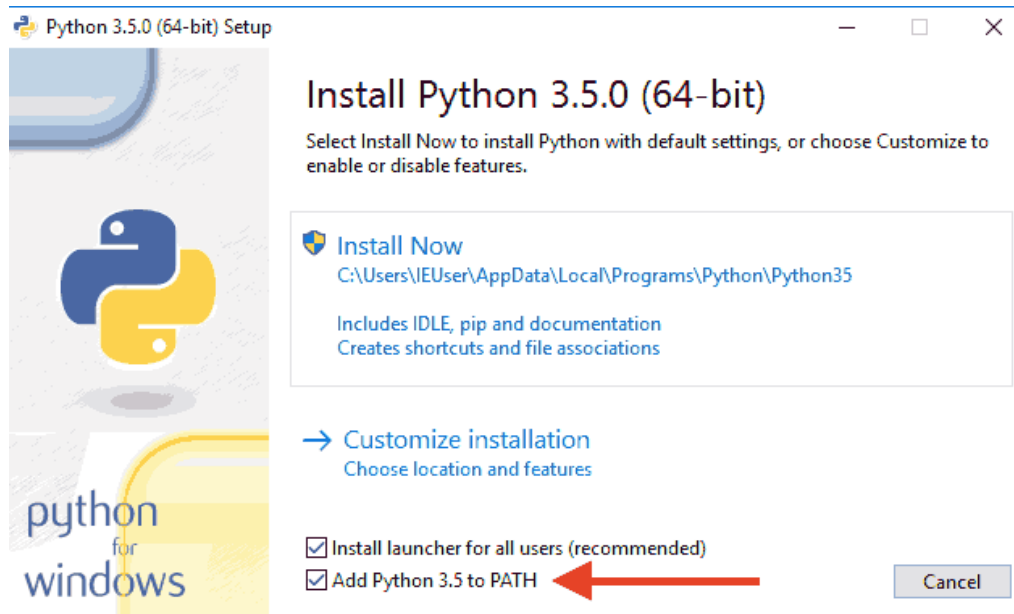
- [Python 3.7.3 - March 25, 2019](#)

Note that Python 3.7.3 cannot be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

由于**3.x**版越来越普及，以最新的**Python 3.7**版本为基础。**Python**版本是最新的**3.7x**

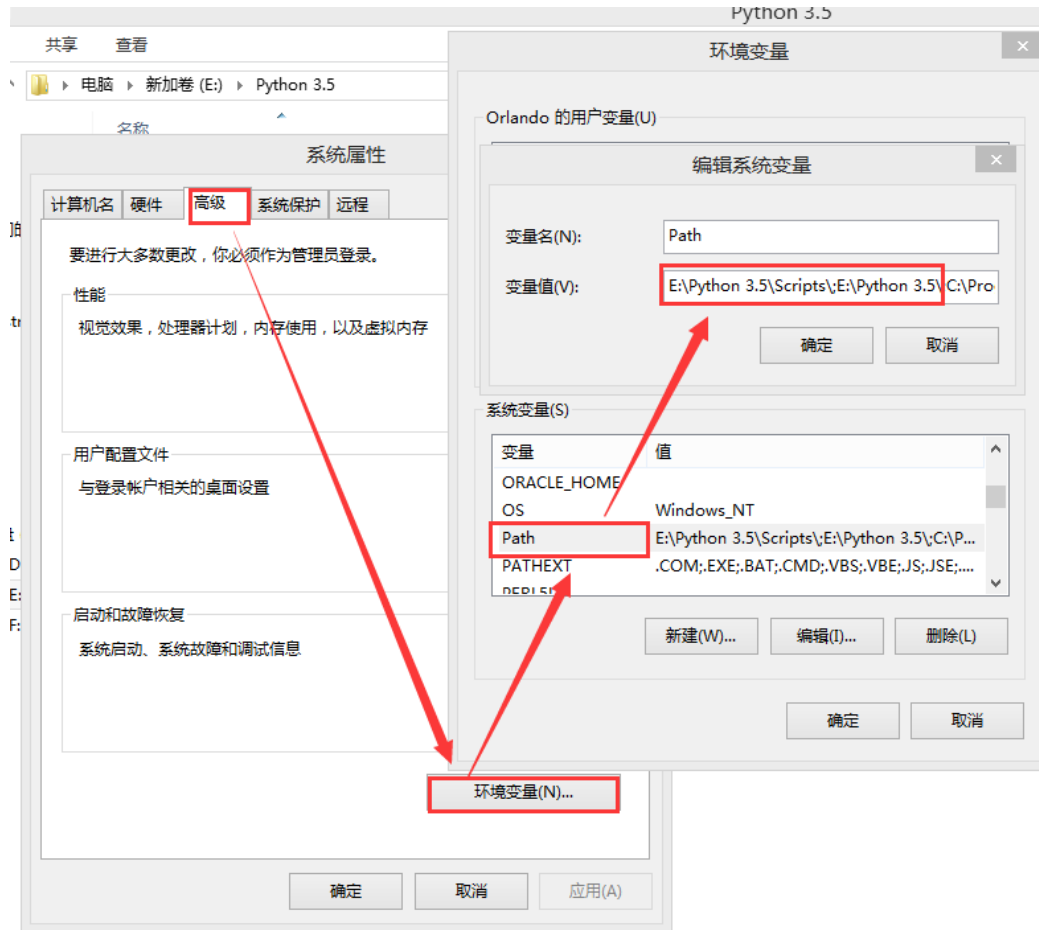
在Windows上安装Python



1.勾选『Add Python 3.7 to PATH』 目的是在环境变量设定的路径中去查找『python.exe』。

2.如果没有勾选，就需要手动添加

在Windows上安装Python



在环境变量中手动添加
python

pip

pip 是一个Python包管理工具，通过pip安装软件包变的十分简单，可以替代 （setuptools）

easy_install 工具。【python3.7已经集成了pip】

下载地址：<https://pypi.python.org/pypi/pip>

```
C:\Users\Leo>pip
Usage:
  pip <command> [options]

Commands:
  install          Install packages.
  uninstall        Uninstall packages.
  freeze           Output installed packages in requirements format.
  list             List installed packages.
  show             Show information about installed packages.
  search           Search PyPI for packages.
  wheel            Build wheels from your requirements.
  help            Show help for commands.
```

在线安装selenium

```
C:\Users\think>pip install selenium
```

<https://pypi.org/project/selenium/>

```
C:\Users\think>pip show selenium
-----
Metadata-Version: 2.0
Name: selenium
Version: 3.10.0
Summary: Python bindings for Selenium
Home-page: https://github.com/SeleniumHQ/selenium/
Author: UNKNOWN
Author-email: UNKNOWN
Installer: pip
License: Apache 2.0
Location: d:\programs\python\python35\lib\site-packages
Requires:
Classifiers:
    Development Status :: 5 - Production/Stable
    Intended Audience :: Developers
    License :: OSI Approved :: Apache Software License
    Operating System :: POSIX
    Operating System :: Microsoft :: Windows
    Operating System :: MacOS :: MacOS X
    Topic :: Software Development :: Testing
```


离线安装selenium

方法一：单击 .tar.gz结尾的文件，并且对文件进行解压，进入到解压目录中，通过python命令进行安装。

进入到目录名中> `python setup.py install`

方法二：.whl 文件本质上面是一个 .zip 包格式。

进入到目录名中> `pip install selenium-3.14.1-py2.py3-none-any.whl`

安装pycharm

<http://www.jetbrains.com/pycharm/>
Community版



本章大纲

15.1 环境搭建

15.2 Selenium常用API

15.3 Unittest单元测试框架

15.4 Page Object设计模式

浏览器驱动的部署

下载相应的浏览器驱动，geckodriver.exe放在D:\Programs\Python\Python35目录下

<https://sites.google.com/a/chromium.org/chromedriver/downloads>

<https://github.com/mozilla/geckodriver/releases>

- 注意：Selenium, Firefox与geckodriver.exe的版本

```
from selenium import webdriver
```

```
# from time import sleep
```

```
browser = webdriver.Firefox()
```

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

```
browser.find_element_by_id("kw").send_keys("selenium")
```

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

对浏览器的操作

- `driver.set_window_size(480, 600)`//设置浏览器尺寸
 - `driver = webdriver.Firefox()`
 - `driver = webdriver.Chrome()`
 - `driver.get(second_url)`
 - `driver.back()`
 - `driver.forward()`
 - `driver.refresh()`
 - `driver.quit()`
- #设置浏览器全屏
- `driver.maximize_window()`

定位元素的8种方法

Python 语言中对应 WebDriver 的 8 种元素定位方法

id	name
class name	Tag Name
linkText	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_linktext ()
find_element_by_xpath()	find_element_by_css_selector ()

定位方法举例

```
driver.find_element_by_id("kw").send_keys("taobao")
driver.find_element_by_name("wd").send_keys("taobao")
driver.find_element_by_class_name("s_ip").send_keys("taobao")
driver.find_element_by_css_selector("#kw").send_keys("taobao")
driver.find_element_by_xpath("//*[@id='kw']").send_keys("taobao")
driver.find_element_by_tag_name("input").send_keys("taobao")
driver.find_element_by_link_text("新闻").click()
driver.find_element_by_partial_link_text("新").click()
```

或者使用另外的定位方式

from selenium.webdriver.common.by import By

```
driver.find_element(By.ID,"kw").send_keys("taobao")
```

定位方法举例 (Xpath)

- 复合属性

`driver.find_element_by_xpath`

`("//input[@id='kw' and @class='su']/span/input")`

- 使用索引号进行定位

`driver.find_element_by_xpath("//input[2]")`

- 模糊的属性值

`driver.find_element_by_xpath("//img[starts-with(@alt,' div1')]")`

`driver.find_element_by_xpath("//img[contains(@alt,' div1')]")`

元素的操作

- 文本框输入：send_keys()
- 按钮/链接/复选框/单选框/下拉框点击：click()
- 复选框

选择页面上所有的 tag name 为 input 的元素

```
inputs = driver.find_elements_by_tag_name('input')
```

然后从中过滤出 type 为 checkbox 的元素，单击勾选

```
for i in inputs:
```

```
    if i.get_attribute('type') == 'checkbox':
```

```
        i.click()
```

元素的操作

- 表单提交：`submit()`
- 返回元素的尺寸：`size`
- 返回元素的文本：`text`
- 获得属性值：`get_attribute("")`
- 设置该属性是否用户可见：`is_displayed()`

下拉框 (select)

from selenium.webdriver.support.select import Select

Select称为选择类，主要使用场景在下拉菜单或者列表中
操作方法有两种

方法一：driver.find_element_by_id("31").click()

方法二：Select模块

select=Select(driver.find_element_by_name("**fruit**"))

select_by_index() :通过索引定位

select_by_value() :通过value值定位

select_by_visible_text() :通过文本值定位

ActionChains类

- **from selenium.webdriver.common.action_chains import ActionChains**
- ActionChains类中封装了鼠标操作的方法。
 - ✓ perform : 执行所有ActionChains中存储的行为
 - ✓ move_to_element : 悬停
 - ✓ context_click : 右击
 - ✓ double_click : 双击
 - ✓ drag_and_drop : 拖动

例：setting=driver.find_element_by_link_text(“**设置**”)(百度的例子)

ActionChains(driver).move_to_element(setting).perform()

键盘事件

模拟键盘的操作需要导入键盘模块：

```
from selenium.webdriver.common.keys import Keys
```

模拟Enter键：send_keys(Keys.ENTER)

键盘F1到F12：send_keys(Keys.F1)把F1改成对应的快捷键

复制Ctrl+C：send_keys(Keys.CONTROL,'c')

粘贴Ctrl+V：send_keys(Keys.CONTROL,'v')

全选Ctrl+A：send_keys(Keys.CONTROL,'a')

剪切Ctrl+X：send_keys(Keys.CONTROL,'x')

向下键：send_keys(Keys.DOWN)

制表键Tab：send_keys(Keys.TAB)

元素等待

- 现实：当浏览器加载页面时，页面上的元素并不会同时被加载，因此增加了定位元素的困难（`ElementNotVisibleException`错误）。
- WebDriver提供了2种类型的等待
 - ✓ 显示等待(`WebDriverWait`类)：使WebDriver等待某个条件成立时继续执行，否则在达到最大时间长时抛出超出时间异常(`TimeoutException`)
 - ✓ 隐式等待(`implicitly_wait`)：通过一定时长的等待页面上元素加载完成。如果超出了设置的时间长，元素还未被加载，则抛出异常`NoSuchElementException`

显示等待

- WebDriverWait类提供等待方法;
- expected_conditions类提供预制条件判断的方法。

参考表: expected_conditions

```
from selenium.webdriver.support.ui import  
WebDriverWait
```

```
from selenium.webdriver.support import  
expected_conditions as EC
```

expected_conditions类提供的判断

方法	说明
title_is	判断标题是否等于预期
title_contains	判断标题是否包含预期字符串
presence_of_element_located	判断元素是否在DOM，不一定可见
visibility_of_element_located	判断元素是否可见
visibility_of	判断元素是否可见，参数是定位后的元素
presence_of_all_elements_located	判断是否至少有一个元素存在
text_to_be_present_in_element	判断某个元素的text是否包含了预期的字符串
text_to_be_present_in_element_value	判断某个元素的value是否包含了预期的字符串
frame_to_be_available_and_switch_to_it	判断表单是否切进去，如果true，并且switch进去
invisibility_of_element_located	判断元素是否不可见
element_to_be_clickable	判断元素是否可点击
element_to_be_selected	判断元素是否被选择，Select
alert_is_present	判断页面是否存在alert

WebDriverWait类

```
element =WebDriverWait(driver,5,0.5).until  
(EC.presence_of_element_located((By.NAME,"username "  
)))
```

- Driver : 浏览器驱动
- Timeout : 最长超时，默认以秒为单位
- Poll_frequency:检测间隔的（步长）时间，默认为0.5s
- Ignored_exceptions:超时后的异常信息，默认情况下抛出NoSuchElementException
- WebDriverWait一般与until()与until_not()方法配合使用。

隐式等待

设置隐式等待为 10 秒

```
driver.implicitly_wait(10)
```

还可使用sleep方法达到等待的效果

```
from time import sleep
```

```
sleep(5)
```

定位一组元素

➤ 多用于以下场景：

- 批量操作元素，勾选所有的复选框
- 先获取一组元素，再从这组对象过滤出需要操作的元素

```
fruit = driver.find_elements_by_xpath("//input[@type='checkbox']")
```

```
for i in fruit:
```

```
    i.click()
```

```
    sleep(1)
```

Python 语言中对应 WebDriver 的定位一组元素	
<code>find_elements_by_id()</code>	<code>find_elements_by_name()</code>
<code>find_elements_by_class_name()</code>	<code>find_elements_by_tag_name()</code>
<code>find_elements_by_link_text()</code>	<code>find_elements_by_partial_linktext ()</code>
<code>find_elements_by_xpath()</code>	<code>find_elements_by_css_selector ()</code>

`pop()` 或 `pop(-1)` 获取最后一个

`pop(0)` 获取一组元素的第一个

`pop(1)` 获取一组元素的第二个

多表单切换

webdriver只能在一个页面上对元素进行定位，对于frame/iframe嵌套页面上的元素无法直接定位，需要通过switch_to.frame()方法进行切换。

1、driver.switch_to.frame("leftframe") 默认可以取表单的id或者name属性

2、如果没有id或者name，可以先通过xpath定位到iframe，再将定位对象传给driver.switch_to.frame

```
xf =driver.find_element_by_xpath("//*[@src='left.html']")
```

```
driver.switch_to.frame(xf)
```

driver.switch_to.parent_frame() 跳到上一层表单

driver.switch_to.default_content()回到最外层的表单

多窗口切换

`current_window_handle` : 获得当前窗口句柄

`window_handles` : 返回所有窗口的句柄到当前会话

`driver.switch_to.window()` : 用于切换到相应的窗口

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

```
baidu_window = driver.current_window_handle
```

```
driver.execute_script('window.open('http://localhost:8032/Mymovie/'))')
```

#获得当前所有打开的窗口

```
all_handles = driver.window_handles
```

```
for handle in all_handles:
```

```
    if handle==baidu_window:
```

```
        driver.switch_to.window(handle)
```

```
        print(driver.title)
```

警告框Alert

- 在WebDriver中处理javascript所生成的alert、confirm、prompt十分简单，具体方法是使用switch_to.alert方法。
 - ✓ a.accept() # 相当于点击确定
 - ✓ a.dismiss() # 相当于点击取消
 - ✓ a.text # 获取弹出框里的文字

```
a=driver.switch_to.alert
```

```
sleep(2)
```

```
print(a.text)
```

```
a.accept()
```

```
a.dismiss()
```

```
a.send_keys("hello")
```

上传文件

- 文件上传操作也较常见，上传功能操作webdriver并没有提供对应的方法。
 - ✓ 上传过程一般要打开一个系统的window 窗口，从窗口选择本地文件添加（问题：如何操作本地window 窗口）。
 - ✓ 上传本地文件；只要定位上传按钮，通send_keys添加本地文件路径就可以了。绝对路径和相对路径都可。

cookie的操作

- 需要验证浏览器中是否存在某个cookie。
- webdriver 操作cookie 的方法有：
 - ✓ `get_cookies()`: 获得所有cookie 信息
 - ✓ `get_cookie(name)`: 返回特定name 有cookie 信息
 - ✓ `add_cookie(cookie_dict)`: 添加cookie, 必须有name 和 value 值
 - ✓ `delete_cookie(name)`: 删除特定(部分)的cookie 信息
 - ✓ `delete_all_cookies()`: 删除所有cookie 信息

```
driver.add_cookie({'name': 'foo', 'value': 'bar'})  
driver.add_cookie({'name': 'foo', 'value': 'bar',  
'path': '/'})
```


调用JavaScript

- 当WebDriver遇到没法完成的操作时，可以考虑借用JavaScript 来完成。
- WebDriver 提供了execute_script() 接口用来调用js 代码。
- 案例：浏览器的滚动条
- `js = "window.scrollTo(100,300)"`
`driver.execute_script(js)`
- `document.getElementById("myHeader").click()`
- `document.getElementById("train_date").removeAttribute('readonly');`

Html5的介绍

- HTML5的设计目的是为了在移动设备上支持多媒体。
- HTML5 中的一些有趣的新特性：<http://videojs.com/>
 - ✓ 用于绘画的 canvas 元素
 - ✓ 用于媒介回放的 video 和 audio 元素
 - ✓ 对本地离线存储的更好的支持
 - ✓ 新的特殊内容元素，比如 article、footer、header、nav、section
 - ✓ 新的表单控件，比如 calendar、date、time、email、url、search

使用javascript可完成HTML5元素的测试

窗口截图

```
driver.get_screenshot_as_file("d:\\demo\\20180711.png")
```

关闭窗口

`close()` 关闭当前窗口

`quit()` 退出相关的驱动程序和关闭所有窗口

logging模块

- **import** logging
logging.basicConfig(level=logging.DEBUG)

可以捕捉到客户端向服务器发送的POST请求

本章大纲

15.1 环境搭建

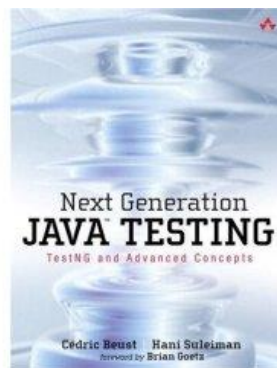
15.2 Selenium常用API

15.3 Unittest单元测试框架

15.4 Page Object设计模式

单元测试

- 单元测试框架 Xunit



如果不使用unittest

```
]class Calculator():
```

```
    # 实现两个数的加、减、乘、除
```

```
]    def __init__(self, a, b):
```

```
        self.a = int(a)
```

```
        self.b = int(b)
```

```
    # 加法
```

```
]    def add(self):
```

```
        return self.a + self.b
```

```
    # 减法
```

```
]    def sub(self):
```

```
        return self.a - self.b
```

```
    # 乘法
```

```
]    def mul(self):
```

```
        return self.a * self.b
```

```
    # 除法
```

```
]    def div(self):
```

```
        return self.a / self.b
```

```
from unitTestDemo.count import Calculator
```

```
# 测试两个整数相加
```

```
]class TestCount:
```

```
]    def test_add(self):
```

```
        try:
```

```
            c = Calculator(1,4)
```

```
            result = c.add()
```

```
            assert(result == 6), 'Integer addition result error!'
```

```
        except AssertionError as msg:
```

```
            print(msg)
```

```
        else:
```

```
            print('test pass!')
```

```
]if __name__ == '__main__':
```

```
    # 执行测试类的测试方法
```

```
    test = TestCount()
```

```
    test.test_add()
```


if __name__ 模块的内置属性

- if __name__ == '__main__': 当.py文件被直接运行时，if __name__ == '__main__'之下的代码块将被运行；
- 当.py文件以模块形式被导入时，if __name__ == '__main__'之下的代码块不被运行。

使用unittest

#使用unittest单元测试框架编写单元测试用例。

```
import unittest
```

```
from unitTestDemo.count import Calculator
```

测试两个整数相加

```
class TestCount(unittest.TestCase):
```

```
    def test_add(self):
```

```
        c = Calculator(2,4)
```

```
        result = c.add()
```

```
        print(result)
```

```
        self.assertEqual(result, 6)
```

```
if __name__ == '__main__':
```

```
    unittest.main()
```

unittest.main()使用TestLoader类搜索所有包含在该模块以

“test” 命名开头的测试方法，

并自动执行它们，执行方法的默认顺序是：根据ASCII码的顺序

加载测试用例，数字与字母的顺序为：0-9，A-Z，a-z。所以以

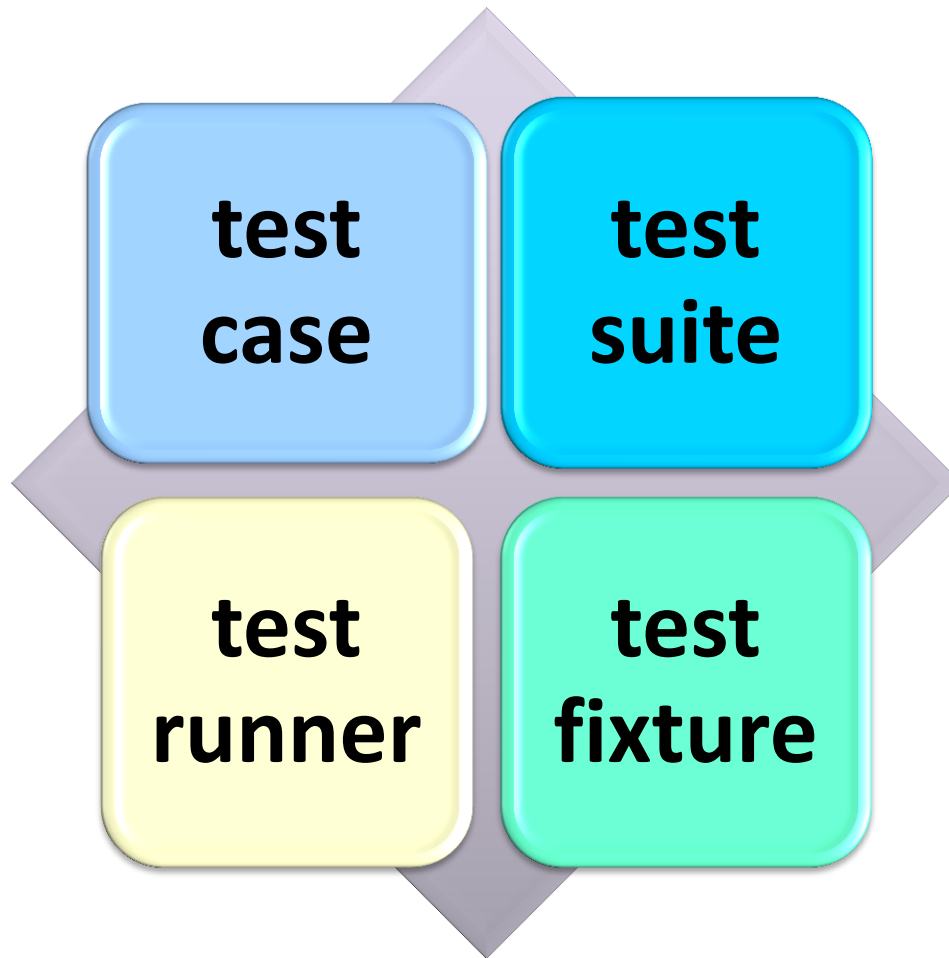
A开头的测试用例方法会优先执行，以a开头会后执行。

使用unittest步骤

1. 测试类必须继承unittest.TestCase
2. 测试方法必须以test开头
3. 使用断言 `self.assertEqual(result, 5)`
4. 运行

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

unittest四个重要的概念



unittest四个重要的概念

- TestCase类，所有测试用例类继承的基本类。

```
class TestCount(unittest.TestCase):
```

- TestSuite类，多个测试用例集合在一起执行，顺序为先加入先执行

```
suite = unittest.TestSuite()
```

```
suite.addTest(TestCount('test_add'))
```

```
suite.addTest(TestCount('test_sub'))
```

unittest四个重要的概念

- TextTestRunner类，通过该类下面的run()方法来运行suite所组装的测试用例，入参为suite测试套件。

```
runner = unittest.TextTestRunner()
```

```
runner.run(suite)
```

- Text Fixture类，通过覆盖TestCase的setUp()和tearDown()来实现环境的创建与销毁

```
def setUp(self):
```

```
    print("setUp")
```

```
def tearDown(self):
```

```
    print("tearDown")
```

setUp():setUp()方法用于每个测试用例执行前的初始化工作。如测试用例中需要访问浏览器，可以在setUp中实例化浏览器驱动。

tearDown():tearDown()方法用于每个测试用例执行之后的善后工作。如：关闭浏览器

unittest中的断言

断言方法：在执行测试用例的过程中，最终用例是否执行通过，是通过判断测试得到的实际结果和预期结果是否相等决定的。

- `assertEqual(a,b , [msg='测试失败时打印的信息'])`:断言a和b是否相等，相等则测试用例通过。
- `assertNotEqual(a,b , [msg='测试失败时打印的信息'])`:断言a和b是否相等，不相等则测试用例通过。
- `assertTrue(x , [msg='测试失败时打印的信息'])`：断言x是否True，是True则测试用例通过

跳过测试和预期测试

- `unittest.skip()`:装饰器，当运行用例时，有些用例可能不想执行，可用装饰器暂时屏蔽该条测试用例。常见的用法如想调试某一测试用例，可先屏蔽其他用例就。
- `@unittest.skip(reason)`: `skip(reason)`装饰器：无条件跳过装饰的测试，并说明跳过测试的原因。
- `@unittest.skipIf(condition,reason)`: `skipIf(condition,reason)`装饰器：条件为真时，跳过装饰的测试，并说明跳过测试的原因。
- `@unittest.skipUnless(condition,reason)`: `skipUnless(condition,reason)`装饰器：条件为假时，跳过装饰的测试，并说明跳过测试的原因。

Fixtures

- `def setUpModule(): /def tearDownModule():`

整个模块的开始和结束执行

- `def setUpClass(cls):/def tearDownClass(cls):`

整个测试类开始和结束时执行，必须用

`@classmethod`修饰

批量执行多个测试类

```
'''
~~~~~
discover() 匹配某目录下的所有测试用例。
'''
~~~~~
import unittest

suite = unittest.defaultTestLoader.discover("./unitTestDemo", pattern='test_*.py')

if __name__ == '__main__':
    runner = unittest.TextTestRunner()
    ~~~~~
    runner.run(suite)
    ~~~~~
```

HTMLTestRunner

- HTMLTestRunner是Python的标准库unittest单元测试框架的一个扩展，用于生成HTML测试报告。
- 下载地址：

<http://tungwaiyip.info/software/HTMLTestRunner.html>

将 HTMLTestRunner.py放在..\Python35\Lib路径下

[home](#)
[about me](#)
[links](#)

HTMLTestRunner

HTMLTestRunner is an extension to the Python standard library's unittest module. It generates easy to use HTML test reports. See a [sample report](#) here. HTMLTestRunner is released under a BSD style license.

[14 comments](#)

Download

[HTMLTestRunner.py](#) (0.8.2)

[test_HTMLTestRunner.py](#) test and demo of HTMLTestRunner.py

Return to [my software](#).

HTMLTestRunner

```
from HTMLTestRunner import HTMLTestRunner

if __name__ == "__main__":
    testunit = unittest.TestSuite()
    testunit.addTest(Demo("test_a"))
    # 定义测试报告存放路径
    fp = open("./result.html", "wb")
    # 定义测试报告
    runner = HTMLTestRunner(stream=fp, title='测试报告', description='用例执行情况：')
    # 运行测试用例
    runner.run(testunit)
    # 关闭报告文件
    fp.close()
```

HTMLTestRunner

使用步骤：

- 1、导入HTMLTestRunner
- 2、通过open()方法以二进制写方式打开当前目录的result.html，如果没有，则自行创建该文件
- 3、调用HTMLTestRunner下的HTMLTestRunner类，stream指定测试报告文件，title定义测试报告的标题，description用于定义测试报告的副标题
- 4、通过HTMLTestRunner的run()方法来运行测试套件中所组装的测试用例
- 5、通过close()关闭测试报告文件

实现邮件发送的功能

Python对SMTP支持有smtplib和email两个模块，email负责构造邮件，smtplib负责发送邮件。

1. 发送HTML格式的邮件 `send_mail1.py`
2. 发送带附件的邮件 `send_mail2.py`
3. 查找最新的测试报告 `find_file.py`
4. 整合自动发送邮件的功能 `run_all_test_sendmail.py`

实现邮件发送的功能

```
receiver=[' 626231936@qq. com' ]
```

```
#主题
```

```
subject = ' 测试1213'
```

```
#邮件内容
```

```
msg=MIMEText(' 你好!', ' html', ' utf-8')
```

```
msg[' Subject' ]=Header(subject, ' utf-8')
```

```
msg[' from' ] = ' test2014123@126. com'
```

```
msg[' to' ] = ' 626231936@qq. com'
```

```
smtp =smtplib.SMTP()
```

```
smtp.connect(smtpserver)
```

```
smtp.login(sender,password)
```

```
smtp.sendmail(sender,receiver,msg.as_string())
```

```
smtp.quit()
```

实现邮件发送的功能（附件）

```
subject = 'Python email test'
sendfile=open('D:\\pythonCode\\Demo2\\unitTestDemo\\result.html','rb').read()
att = MIMEText(sendfile, 'base64', 'utf-8')
att["Content-Type"] = 'application/octet-stream'
# 这里的filename可以任意写, 写什么名字, 邮件中显示什么名字
att["Content-Disposition"] = 'attachment; filename="result.html"'
msg=MIMEMultipart('related')
msg['Subject']=Header(subject,'utf-8')
msg.attach(att)
msg.attach(MIMEText('<html><h1>请查收测试报告! </h1></html>', 'html', 'utf-8'))
msg['from'] = 'test2014123@126.com'
msg['to'] = '626231936@qq.com'
```


查找最新的测试报告 find_file.py

```
import os
# 定义文件目录
result_dir = 'D:\\pythonCode\\Demo2\\unitTestDemo'
lists=os.listdir(result_dir)
lists.sort(key=lambda fn: os.path.getmtime(result_dir+"\\")+fn))
print('最新的文件为'+lists[-1])
file=os.path.join(result_dir,lists[-1])
print(file)
```

整合自动发送邮件的功能

```
if __name__ == '__main__':
    test_dir='./test_case'
    test_report='./eport'
    discover = unittest.defaultTestLoader.discover(test_dir, pattern='*_case.py')
    now_time = time.strftime("%Y%m%d%H%M%S")
    file_name=test_report+'\\'+now_time+'result.html'
    fp = open(file_name,'wb')
    runner = HTMLTestRunner(stream=fp,title="Mymovie测试报告",description="运行环境:  firefox")
    runner.run(discover)
    fp.close()
    new_report=new_report(test_report)
    send_mail(new_report)
```

本章大纲

15.1 环境搭建

15.2 Selenium常用API

15.3 Unittest单元测试框架

15.4 Page Object设计模式

PO模式

代码在随着进一步新增测试用例的情况会有以下几个问题：

1. 易读性，一连串的find element会显得杂乱无章
2. 可扩展不好：用例孤立，难以扩展
3. 可复用性：无公共方法，很那复用
4. 可维护性：一旦页面元素变化，需要维护修改大量的用例

```
#读取外部文件,格式为xml
from selenium import webdriver
from time import sleep
from public import Par_Login
import csv
from xml.dom import minidom

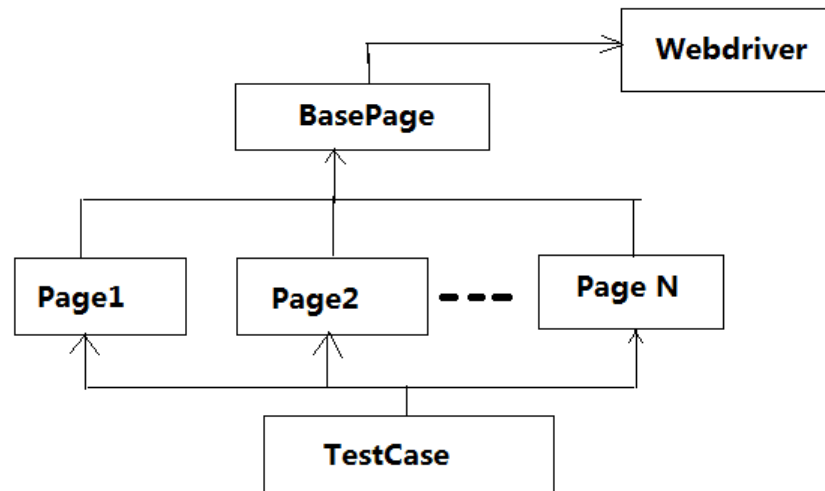
driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost:8032/ms/admin.php/Index/index")

# 读取外部info.xml
dom = minidom.parse("d:\\info.xml")
root = dom.documentElement #得到文档元素对象

name = dom.getElementsByTagName("name")
pwd = dom.getElementsByTagName("pwd")
# 获得第1个name标签对的值
name_value = name[0].firstChild.data
print(name_value)
pwd_value = pwd[0].firstChild.data
print(pwd_value)
```

Page Object

Page Object模式是一种自动化测试设计模式，将页面定位和业务操作分开，分离测试对象（元素对象）和测试脚本（用例脚本），提高用例的可维护性。



Page Object

```
class Base(object):
```

```
    def __init__(self, driver, base_url):
        self.driver = driver
        self.base_url = base_url
        self.timeout = 30
```

```
    def _open(self, url):
```



```
        url_ = self.base_url + url
        self.driver.maximize_window()
        self.driver.get(url_)
```

```
    def open(self):
```

```
        self._open(self.url)
```

*# *loc参数个数不是固定 *loc = (By.ID, "kw")*

```
    def find_element(self, *loc):
```

```
        return self.driver.find_element(*loc)
```

补充

```
def foo(*a):  
    print(a)
```

```
foo([1,2,3])
```

"*" 表示这个参数是一个元组参数

"**" 表示这个字典参数

Page Object

页面对象 (PO) 登录页面

```
class LoginPage(Base):
```

```
    login_username_text_loc = (By.NAME, "username")
```

```
    login_password_text_loc = (By.NAME, "password")
```

```
    login_button_loc = (By.CLASS_NAME, "sub")
```

#把每一个元素封装成一下方法

```
def login_username(self, text):
```

```
    self.find_element(*self.login_username_text_loc).send_keys(text)
```

```
def login_password(self, text):
```

```
    self.find_element(*self.login_password_text_loc).send_keys(text)
```

```
def login_submit(self):
```

```
    self.find_element(*self.login_button_loc).click()
```

```
def login_action(driver, username, password):
```

```
    login_page =LoginPage(driver)
```

```
    login_page.open()
```

```
    login_page.login_username(username)
```

```
    login_page.login_password(password)
```

```
    login_page.login_submit()
```


数据驱动模式

```
import xlrd
```

```
class Test(unittest.TestCase):
```

```
    def readexcel(self, rowvalue, colvalue):  
        workbook = xlrd.open_workbook('D:\\demo\\loginuser.xlsx', 'r')  
        table = workbook.sheet_by_index(0)  
        return table.cell_value(rowvalue, colvalue)
```

```
import csv
```

```
data = csv.reader(open('D:\\demo\\userdata.csv', 'r'))
```

```
for user in data:
```

```
    print(user[0])
```