

16 基于Python的自动化测试

本章大纲

16.1 环境搭建

16.2 Selenium常用API

16.3 Unittest单元测试框架

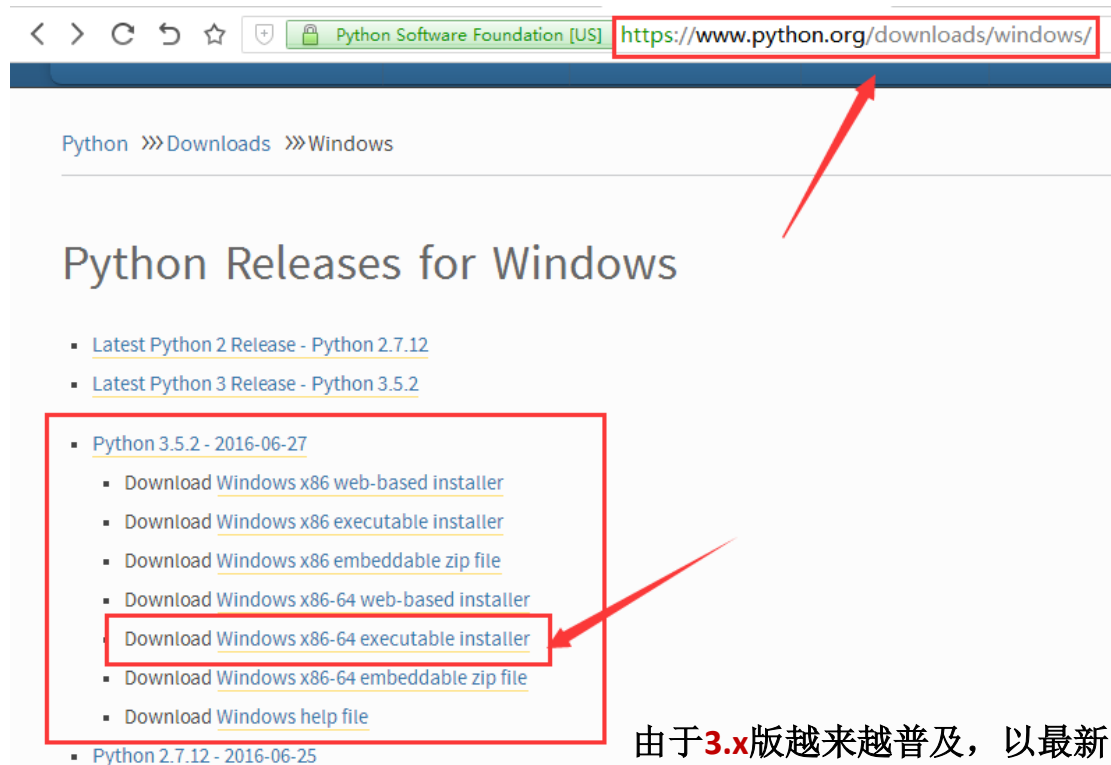
16.4 Page Object设计模式

安装Python

- Python可应用于多平台包括windows、 Linux 和 Mac OS X。
- Python最新源码，二进制文档，新闻资讯等可以在Python的官网查看到：
- 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/

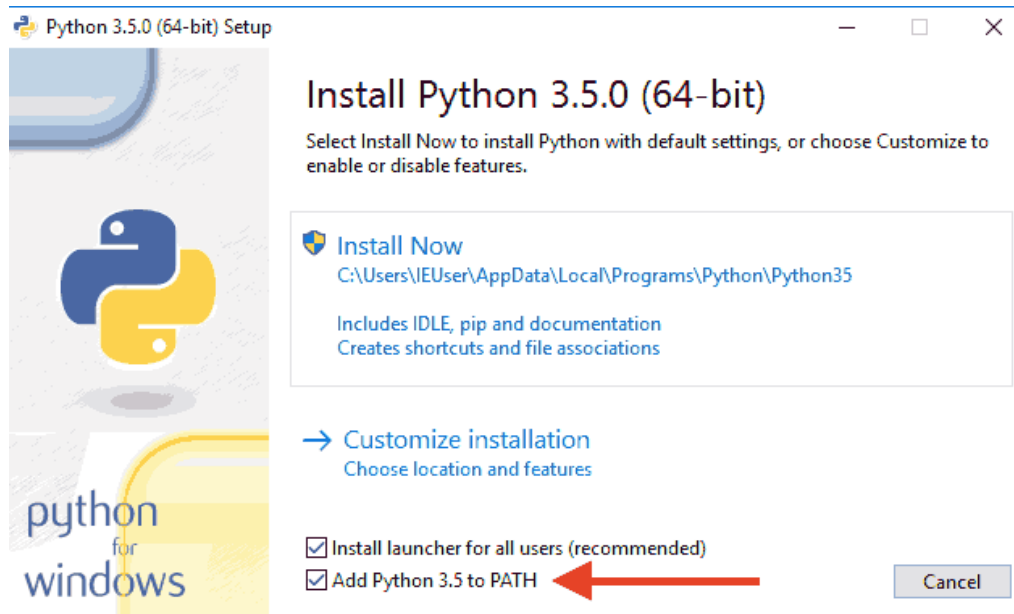
安装Python

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



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**

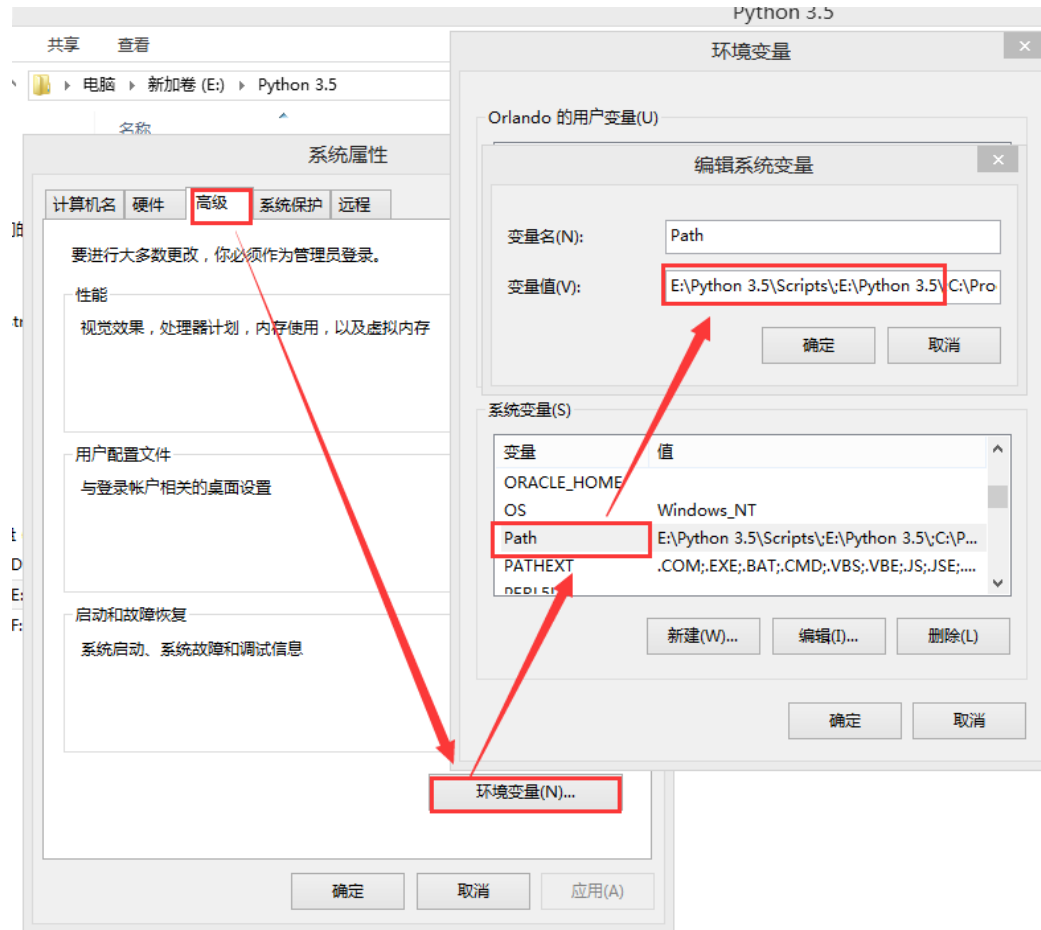
在Windows上安装Python



1.勾选『Add Python 3.5 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.
```

安装selenium

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

Selenium版本地址：<http://selenium-release.storage.googleapis.com/index.html>
最新：3.10

```
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
```


安装pycharm

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



本章大纲

16.1 环境搭建

16.2 Selenium常用API

16.3 Unittest单元测试框架

16.4 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放在英文目录下，添加环境变量
<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 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。

- 绝对路径定位（不建议使用）

```
driver.find_element_by_xpath( "/html/body/div[1]/input" )
```

例如div[1]表示当前层级下的第一个div标签。

```
driver.find_element_by_xpath
```

```
( "/html/body/div[1]/html/body/div[1]/input[@value= '查询' ]" )
```

- 相对路径定位（结合属性值）

```
driver.find_element_by_xpath( "//input[@value= '查询' ]" )
```

定位方法举例

- 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("//input[ends-with(@id,'很漂亮')]")`

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

定位方法举例

- 使用绝对路径来定位元素（不推荐）

```
driver.find_element_by_css_selector("html body div div form  
input"));
```

- 也可以以父子关系的方式“>”来描述这个选择器

```
("html >body > div > div > form > input"));
```

```
("html>body>div[1]>input[type= 'button' ]"));
```

定位方法举例

- 相对路径

可以用这样的方法来定位用户输入字段，假设它在 DOM 中是第一个 `<input>` 元素：

```
driver.find_element_by_xpath( "input" )
```

- 相对路径（结合属性定位）（`"input[type= 'button']"`）
- class属性 （`"input.spread"`）
- id属性 （`"input#divinput"`）

定位方法举例

- 复合属性 ("img[alt= 'aaa'][href= '****']")

- 模糊匹配

开头包含 a[href^= 'http://www.baidu']

结尾包含 a[href\$= 'baidu.com']

包含 a[href*= 'baidu']

- 子页面元素的查找

("div#div1>input#divinput1")

XPath与CSS的类似功能的对比

定位方式	XPath	CSS
标签	<code>//div</code>	<code>div</code>
By id	<code>//div[@id='eleid']</code>	<code>div#eleid</code>
By class	<code>//div[@class='eleclass']</code>	<code>div.eleid</code>
By 属性	<code>//div[@title='Move mouse here']</code>	<code>div[title=Move mouse here]</code> <code>div[title^=Move]</code> <code>div[title\$=here]</code> <code>div[title*=mouse]</code>
定位子元素	<code>//div[@id='eleid']/*</code> <code>//div/h1</code>	<code>div#eleid>*</code> <code>div >h1</code>

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)

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

- 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")`

`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')

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`

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

定位一组元素：批量操作一组元素

```
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

find_elements_by_id()	find_elements_by_name()
find_elements_by_class_name()	find_elements_by_tag_name()
find_elements_by_link_text()	find_elements_by_partial_linktext ()
find_elements_by_xpath()	find_elements_by_css_selector ()

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

frame/iframe嵌套，webdriver只能在一个页面上对元素进行定位
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)
```

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

```
search_windows = driver.current_window_handle
driver.find_element_by_link_text("新闻").click()
all_handles = driver.window_handles
for handle in all_handles:
    if handle == search_windows:
        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

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

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

本章大纲

16.1 环境搭建

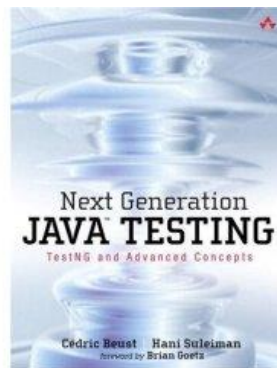
16.2 Selenium常用API

16.3 Unittest单元测试框架

16.4 Page Object设计模式

单元测试

- 单元测试框架 Xunit



Unittest



CPPUnit

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()
```

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):

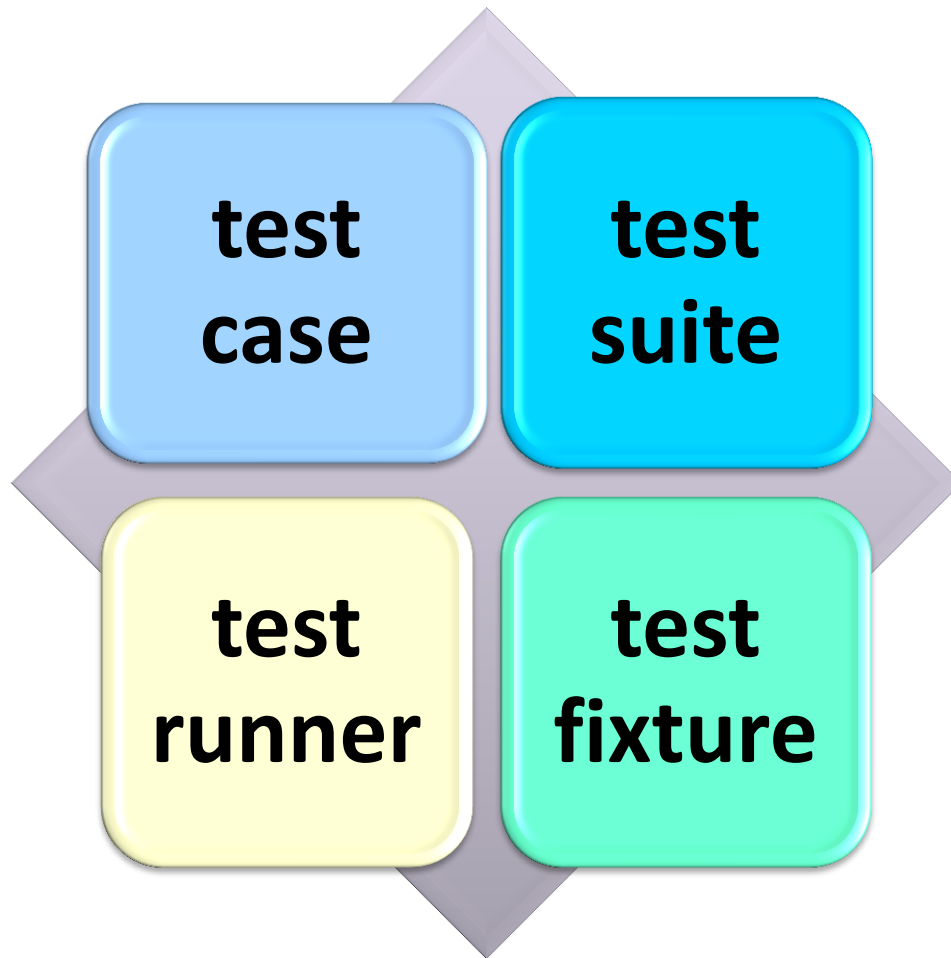
    def test_add(self):
        c = Calculator(2,4)
        result = c.add()
        print(result)
        self.assertEqual(result, 6)

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。

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

```
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()方法用于测试用例执行之后的善后工作。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

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](#).

实现邮件发送的功能

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`

本章大纲

16.1 环境搭建

16.2 Selenium常用API

16.3 Unittest单元测试框架

16.4 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

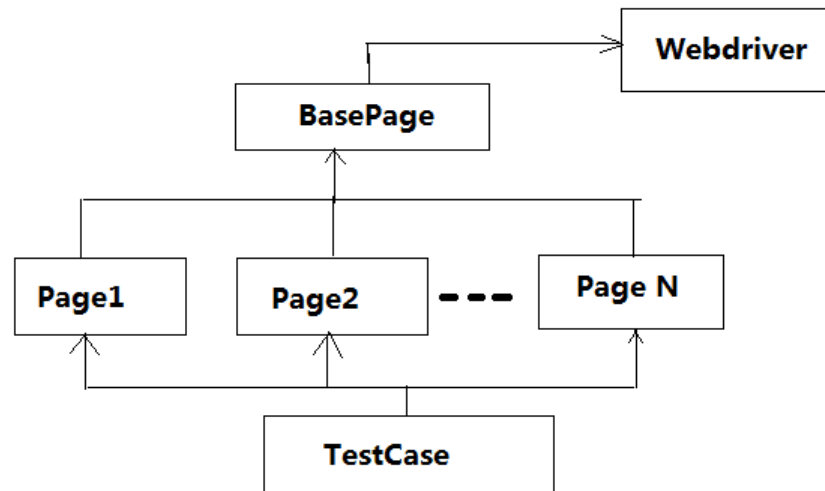
driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost:8032/ms/admin.php/Index/index")

# 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)
```

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()
```