# 一、HTML课件（1-5~1-11）

**Html**

什么是 HTML？ HTML 是用来描述网页的一种语言。 HTML 指的是超文本标记语言 (Hyper Text Markup Language) HTML 不是一种编程语言，而是一种标记语言 (markup language) Html包含静态html和动态html

**学习推荐网站：**

* <http://www.w3school.com.cn/html/html_headings.asp>
* 自学网HTML教程

<html>

<head>

<title>我要自学网</title>

</head>

<body>

<h2>自学网口号</h2>

<p>自学成才，快速成功！！增加收入，改变命运</p>

<p>这是一个神奇的网站，你值得拥有！</p>

</body>

</html>

**HTML 标签**

* HTML 标记标签通常被称为 HTML 标签 (HTML tag)。
* HTML 标签是由尖括号包围的关键词，比如 <html>
* HTML 标签通常是成对！出现的，比如 <b> 和 </b>
* 标签对中的第一个标签是开始标签，第二个标签是结束标签
* 开始和结束标签也被称为开放标签和闭合标签

**案例解析**

* <html> 与 </html> 之间的文本描述网页
* <head>与</head>之间标签用于定义文档的头部，它是所有头部元素的容器。
* <body> 与 </body> 之间的文本是可见的页面内容
* <h1> 与 </h1> 之间的文本被显示为标题
* <p> 与 </p> 之间的文本被显示为段落

**HTML编辑器**

1. Adobe Dreamweaver
2. notepad++
3. 记事本

**基本的 HTML 标签**

**HTML 标题**

HTML 标题（Heading）是通过 <h1> - <h6> 等标签进行定义的。

<h1>This is a heading</h1>

<h2>This is a heading</h2>

<h3>This is a heading</h3>

**head定义和用法**

<head> 标签用于定义文档的头部，它是所有头部元素的容器。<head> 中的元素可以引用脚本、指示浏览器在哪里找到样式表、提供元信息等等。

<html>

<head>

<title>文档的标题</title>

</head>

<body>

文档的内容... ...

</body>

</html>

**HTML 段落**

HTML 段落是通过 <p> 标签进行定义的。

<p>This **is** a paragraph.</p>

<p>This **is** another paragraph.</p>

**HTML 链接**

* HTML 链接是通过 <a> 标签进行定义的。
* <a> 标签定义超链接，用于从一张页面链接到另一张页面。
* <a> 元素最重要的属性是 href 属性，它指示链接的目标。

<a href="http://www.51zxw.net">This is 51zxw link</a>

**HTML 图像**

HTML 图像是通过 <img> 标签进行定义的。

<img src="w3school.jpg" width="104" height="142" />

**HTML 注释**

可以将注释插入 HTML 代码中，这样可以提高其可读性，使代码更易被人理解。浏览器会忽略注释，也不会显示它们。 注释是这样写的：

*<!-- This is a comment -->*

**HTML 换行**

如果您希望在不产生一个新段落的情况下进行换行（新行），请使用 <br /> 标签：

<p>This is<br />a para<br />graph with line breaks</p>

**HTML Div**

<div> 可定义文档中的分区或节（division/section）。 <div> 标签可以把文档分割为独立的、不同的部分。它可以用作严格的组织工具，并且不使用任何格式与其关联。 如果用 id 或 class 来标记 <div>，那么该标签的作用会变得更加有效。

<html>

<head>

<title>我要自学网</title>

</head>

<body>

<h1>自学网口号</h1>

<p>自学成才，快速成功！！增加收入，改变命运</p>

<p>这是一个神奇的网站，你值得拥有！</p>

*<!--DIV --->*

<div style="background-color:red;">

<h3>This is a div block </h3>

<p>hello 51zxw!</p>

</div>

</body>

</html>

更多标签 请见：<http://www.w3school.com.cn/tags/index.asp>

#### name 属性

name 属性用于指定标签元素的名称。 <a> 标签内必须提供 href 或 name 属性。

<a name="value">

#### id 属性

1. id 属性规定 HTML 元素的唯一的 id。
2. id 在 HTML 文档中必须是唯一的。
3. id 属性可通过 JavaScript（HTML DOM）或通过 CSS 为带有指定 id 的元素改变或添加样式。

<html>

<head>

*<!--以下定义一个js方法，主要是根据id来改变内容-->*

<script type="text/javascript">

**function** **change\_header**()

{

document.getElementById("myHeader\_1").innerHTML="You have clicked!";

}

</script>

</head>

<body>

<h1 id="myHeader">Hello 51zxw!</h1>

<h1 id="myHeader\_1">Hello zxw!</h1>

<button onclick="change\_header()">Change text</button>

</body>

</html>

注意：

id 或 name 属性的值可以是引号引起来的任何字符串。字符串必须是唯一的标记，不能在同一文档中的其他 name 或 id 属性中重复使用，但是可以在不同的文档中再次使用。

#### style 属性

样式是 HTML 4 引入的，它是一种新的首选的改变 HTML 元素样式的方式。通过 HTML 样式，能够通过使用 style 属性直接将样式添加到 HTML 元素，或者间接地在独立的样式表中（CSS 文件）进行定义。

##### HTML 样式实例 - 背景颜色

background-color 属性为元素定义了背景颜色：

<html>

<body style="background-color:yellow">

<h2 style="background-color:red">This is a heading</h2>

<p style="background-color:green">This is a paragraph.</p>

</body>

</html>

##### HTML 样式实例 - 字体、颜色和尺寸

font-family、color 以及 font-size 属性分别定义元素中文本的字体系列、颜色和字体尺寸： <html>

<body>

<h1 style="font-family:verdana">A heading</h1>

<p style="font-family:arial;color:red;font-size:20px;">A paragraph.</p>

</body>

</html>

##### HTML 样式实例 - 文本对齐

text-align 属性规定了元素中文本的水平对齐方式：

<html>

<body>

<h1 style="text-align:center">This is a heading</h1>

<p>The heading above is aligned to the center of this page.</p>

</body>

</html>

##### 更多HTML属性请见

[http://www.w3school.com.cn/tags/html\_ref\_standardattributes.asp](http://www.w3school.com.cn/tags/html_ref_standardattributes.asp" \t "_blank)

表 (Cascading Style Sheets)

##### CSS产生缘由

HTML 标签原本被设计为用于定义文档内容。通过使用 <h1>、<p>、<table> 这样的标签，HTML 的初衷是表达“这是标题”、“这是段落”、“这是表格”之类的信息。同时文档布局由浏览器来完成，而不使用任何的格式化标签。 由于两种主要的浏览器（Netscape 和 Internet Explorer）不断地将新的 HTML 标签和属性（比如字体标签和颜色属性）添加到 HTML 规范中，创建文档内容清晰地独立于文档表现层的站点变得越来越困难。

##### CSS作用

样式通常保存在外部的 .css 文件中。通过仅仅编辑一个简单的 CSS 文档，外部样式表使你有能力同时改变站点中所有页面的布局和外观。

##### CSS语法

CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明。

**select**or {declaration1; declaration2; ... declarationN }

* 选择器通常是您需要改变样式的 HTML 元素。
* 每条声明由一个属性和一个值组成。
* 属性（property）是您希望设置的样式属性（style attribute）。每个属性有一个值。属性和值被冒号分开。

selector {property: value}

例子：

将 h1 元素内的文字颜色定义为红色，同时将字体大小设置为 14 像素。

h1 {color:red; font-size:14px;}

#### Css外部样式引用

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css">

</head>

##### 更多css内容请参考

[http://www.w3school.com.cn/css/index.asp](http://www.w3school.com.cn/css/index.asp" \t "_blank)

本文档参考来源：http://www.w3school.com.cn/index.html

# 二、第二章教案完整版

走进Selenium新世界

**浏览器**

* Firefox Setup 35.0.1
* 安装完成后设置菜单栏
* 关闭浏览器自动更新

**插件配置（必备武器）**

* FireBug

Firebug是firefox下的一个扩展,能够调试所有网站语言，如Html,Css等，但FireBug最吸引人的就是javascript调试功能，使用起来非常方便，而且在各种浏览器下都能使用（IE,Firefox,Opera, Safari）。

* FirePath
* HTML元素定位神器，谁用谁知道！

**Selenium IDE简介**

Selenium IDE是一个Firefox插件，用于记录和播放用户与浏览器的交互。 使用它来创建简单的脚本或协助进行探索性测试。

**Selenium IDE安装**

* 1.官网下载插件后本地安装 <http://www.seleniumhq.org/download/>
* 2.浏览器搜索插件安装

**Selenium打开运行**

* 工具栏——>Selenium IDE
* 直接点击菜单栏Selenium 图标

Selenium IDE 自动化实战

##### 任务1：

* 自动在百度搜索“我要自学网” 然后在搜索结果页面点击进入自学网主页

##### 任务2

实现自学网自动登录个人账号

* Test2017
* 123456test

##### 步骤

1. 输入测试 Base URL
2. 打开录制按钮
3. 在浏览器界面进行相关操作
4. 回放录制的操作（注意：回放时浏览器一定要处于打开状态）
5. 保存测试脚本

##### 导入已保存的脚本

文件——> Open——> 选择要导入的脚本

Selenium IDE脚本编辑与操作

### 1. 编辑一行命令

在Table标签下选中某一行命令，命令由command、Target、value三部分组成。可以对这三部分内容那进行编辑。

### 2. 插入命令。

在某一条命令上右击，选择“insert new command”命令，就可以插入一个空白，然后对空白行进程编辑。

### 3. 插入注释

鼠标右击选择“insert new comment”命令插入注解空白行，本行内容不被执行，可以帮助我们更好的理解脚本，插入的内容以紫色字体显示。

### 4. 移动命令或注解

有时我们需要移动某行命令的顺序，我们只需要左击鼠标拖动到相应的位置即可。

### 5.删除命令

选择单个或多个命令，然后点击鼠标右键选择“Delete”

### 6.命令执行

选定要执行的命令点击单个执行按钮即可，注意：有一些命令必须依赖于前面命令的运行结果才能成功执行，否则会导致执行失败。

Selenium IDE常用命令

### open(url)命令

（1）作用：打开指定的URL，URL可以为相对或是绝对URL；

（2）Target：要打开的URL；value值为空

* 当Target为空，将打开Base URL中填写的页面；
* 当Target不为空且值为相对路径，将打开Base URL + Target页面。如，假设Base URL为http://www.51zxw.net，而Target为/list.aspx?cid=451，则执行open命令时，将打开http://www.51zxw.net/list.aspx?cid=451
* 当Target以http://开头时，将忽略Base URL，直接打开Target的网址；

### pause(waitTime)

（1）作用：暂停脚本运行

（2）waitTime：等待时间，单位为ms；

### goBack()

（1）作用：模拟单击浏览器的后退按钮； （2）由于没有参数，所以Target和Value可不填；

### refresh()

（1）作用：刷新当前页；

（2）由于没有参数，所以Target和Value可不填；

### windowMaximize()

（1）作用：将当前的窗口最大化，即设置为全屏显示； （2）由于没有参数，所以Target和Value可不填；

### click（locator）

（1）作用：单击一个链接、按钮、复选框或单选按钮；

（2）如果该单击事件导致新的页面加载，命令将会加上后缀“AndWait”，即“clickAnd Wait”，或“waitForPageToLoad”命令；

### type(locator,value)

（1）作用：向指定输入域中输入指定值；也可为下拉框、复选框和单选框按钮赋值.

（2）Target：元素的定位表达式；

（3）Value：要输入的值；

### select(selectLocator,optionLocator)

（1）作用：模拟人工单击下拉列表框；

selectLocator：指向指定选择元素的元素定位器；

optionLocator：选项的选择器（默认为标签）；

（2）选项的选择方式两种 label 和Value

* label=文本值，基于选项的文本进行匹配（默认方式），如label=three；
* value=真实值，基于选项的真实值进行匹配，如value=3；

### close()

（1）作用：模拟用户单击窗口上的关闭按钮；

（2）由于没有参数，所以Target和Value可不填；

断言与验证

### 断言

验证应用程序的状态是否同所期望的一致。 常见的断言包括:验证页面内容，如标题是否为X或当前位置是否正确等等。

### 断言被用于4种模式+5种手段:

Assert

Assert 断言失败时，该测试将终止。

verify

Verify 断言失败时，该测试将继续执行，并将错误记入日显示屏。也就是说允许此单个验证通过。确保应用程序在正确的页面上。提高脚本的伸缩性。

waitfor

Waitfor用于等待某些条件变为真。可用于AJAX应用程序的测试。 如果该条件为真，他们将立即成功执行。如果该条件不为真，则将失败并暂停测试。直到超过当前所设定的超时时间。 一般跟setTimeout时间一起用

store

store 定义变量，可以获取页面的相关元素进行判断。

#### 5种手段：

* Title 获取页面的标题
* Value 获取元素的值
* Text 获取元素文本信息
* Table 获得元素标签
* ElementPresnt 获得当前元素。

### 断言常用的有：

assertTitle（检查当前页面的title是否正确）

assertValue（检查输入框的值， 单选或复选框的值）

VerifyValue() 验证元素的值。

### 断言设置方法

1.浏览器页面点击鼠标右键->Show All Available commands->选择具体的断言方式，脚本回自动加载选定的断言命令

2.直接在脚本页面编辑

第二章课程小结



# 三、1第三章教案（3-1~3-10）

人生苦短，我用Python

" Life is short, you need Python" ——Bruce Eckel "



**Python是什么**

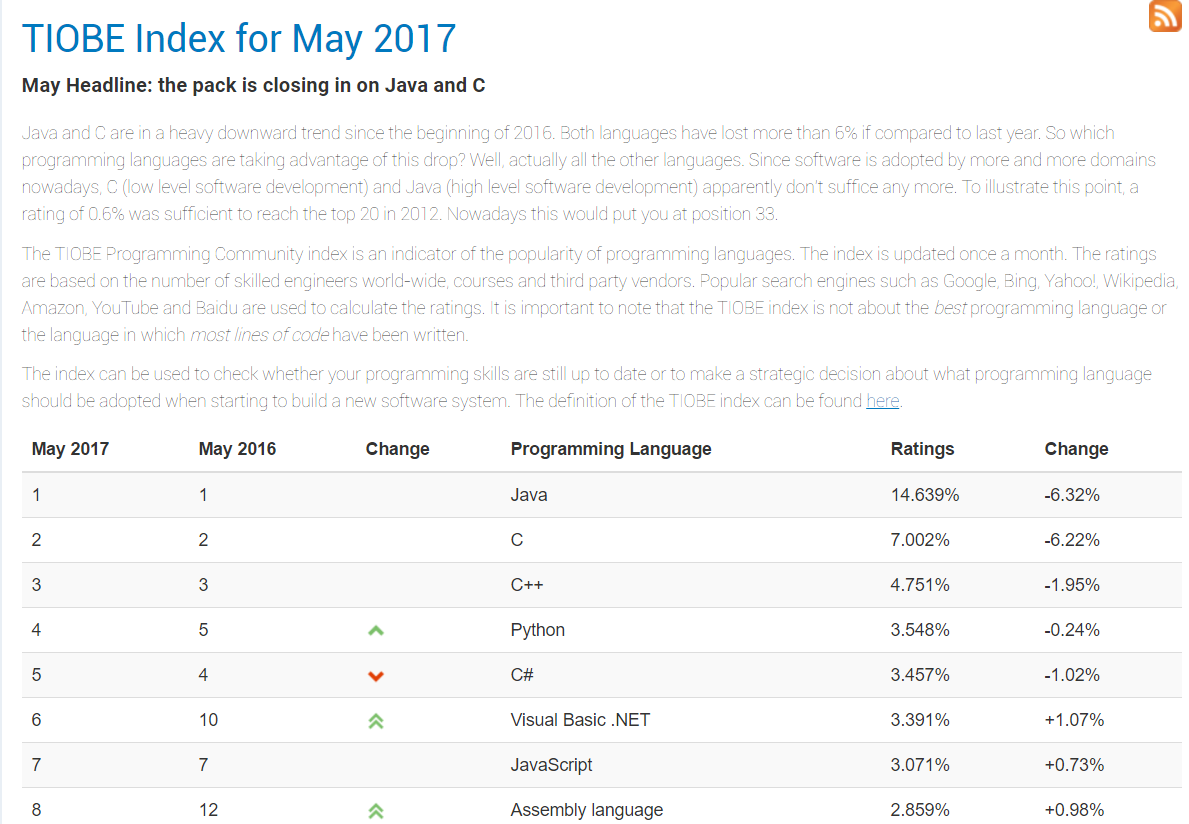
是一种面向对象、解释型计算机程序设计语言，由“龟叔”（Guido van Rossum）于1989年圣诞节为打发无聊时间，而开发的一个新的脚本解释程序，至于为什么选中Python作为语言名字，是因为他是一叫Monty Python的喜剧团体的爱好者，第一个公开发行版发行于1991年。

**设计哲学**

* 优雅
* 明确
* 简单

**Python 为什么流行**

* 代码量小
* 维护成本低
* 编程效率高



编程语言排名：<https://www.tiobe.com/tiobe-index/>

**Python 可以用来做什么好玩的事情？**

Python 最常用的应该就是写爬虫了吧，比较简单的应用就是爬取web网站的资源（图片，文字、链接等）

有些资深股民都是用 Python 抓取财经网站数据、并进行处理，然后输出可视化图表来帮助做决策。

人工智能机器学习方面应用，python有很多库很方便做人工智能，比如numpy, scipy做数值计算的，sklearn做机器学习的，pybrain做神经网络的。

软件测试领域，自动化测试（Web端(python+selenium)和移动客户端python+appium）

**Python2.X 与Python3.X学哪个？**

个人推荐学习Python3.X,因为Python的2.x分支版本在2020年将停止更新，同时，Python 3将被定为该语言的未来发展方向。

Python3针对Python2做了许多优化，特别是尤其是处理字符等。

Python安装配置

### Python下载

* 官网下载地址：<https://www.python.org/downloads/windows/>
* 下载安装包：

1. python-3.5.0-amd64（64位）.exe
2. python-3.5.0（32bit）.exe

### Python安装

直接安装下载的安装包即可（建议安装在C盘根目录）

### 安装完成后

* 菜单栏查看目录
* 磁盘路径查看
* 调试运行-IDLE

### Pip安装Selenium

pip 是一个现代的，通用的 Python 包管理工具。提供了对 Python 包的查找、下载、安装、卸载的功能。

##### 安装指定版本的Selenium

pip install selenium==2.48.0

##### 查看当前包的版本信息

pip show selenium

##### 卸载当前安装包

pip uninstall selenium

##### 安装完成后路径

...\Lib\site-packages

### PyCharm 安装

PyCharm是一种Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发。

官网下载地址（选择社区版）：

[http://www.jetbrains.com/pycharm/download/#section=windows](http://www.jetbrains.com/pycharm/download/" \l "section=windows)

第一个Python程序

print语句

#打印字符串

print ("Hello,51zxw")

**print**("are","you","OK?")

#打印整数

**print**(500)

**print**(300+200)

#打印变量

name="51zxw"

**print**("hello,%s" %name)

width=30

**print**("width is %d " %width)

input 语句

con=**input**("please input Content")

**print**("the input Content is %r" %con)

### 常用数据类型

#### 整数

x=5

y=5

z=x+y

print (z)

#### 浮点数

f=5.20

l=5.30

a=f+l

print(a)

#### 字符串

str='hello 51zxw'

print(str)

#### 转义字符

**print**("hello \n51zxw")

**print**('c:\\python35')

# my name is ‘Jack’ and “you”

**print**('My name is \'Jack\' and \”you\”')

#### 布尔值

t=True

f=False

print(t and f)

### 数组（List）

数组是一种有序的集合，可以随时添加和删除其中的元素。

##### 数组定义

student=['Jack','Bob','Harry','Micle']

**print**(student)

##### 访问数组元素

用索引来访问list中每一个位置的元素，记得索引是从0开始的：

student=['Jack','Bob','Harry','Micle']

**print**(student[0])

**print**(student[1])

**print**(student[-1]) #访问最后一个元素

注意： 当索引超出了范围时，Python会报一个IndexError错误，所以，要确保索引不要越界。

##### 添加元素

#末尾追加元素

student.**append**('51zxw')

**print**(student)

#指定位置添加元素

student.**insert**(0,'hello')

**print**(student)

##### 修改元素

student[0]='No.1'

**print**(student)

##### 删除元素

#删除末尾元素

student.**pop**()

**print**(student)

#删除指定位置元素

student.**pop**(1)

**print**(student)

### 元组（Tuple）

Python的元组与列表类似，不同之处在于元组的元素一旦定义就不能修改。 元组使用小括号，列表使用方括号。 元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

course=('Chinese','Math','English','computer')

**print**(course)

**print**(course[0])

**print**(course[1:3])

**print**(len(course))

要定义一个只有1个元素的元组，则需要在元素后面加逗号，用来消除数学歧义

t = (1,)

返回元组最大的值

score=(78,90,88,67,78)

print(max(score))

### 字典(Dictionary)

字典是另一种可变容器模型，且可存储任意类型对象。 字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号{}中 ,格式如下所示：

d = {key1 : **value**1, key2 : **value**2 }

键必须是唯一的，但值则不必。 值可以取任何数据类型，但键必须是不可变的。

#### 定义访问字典

student={1:'Jack',2:'Bob',3:'Marry',4:'Micle'}

print(student[3])

#### 添加元素

#增加新的键值对

student[5]='51zxw'

**print**(student)

#### 修改元素

#修改字典

student[2]='Harry'

**print**(student)

#### 删除字典

#删除某一个键值对

del student[1]

**print**(student)

#清空字典全部内容

student.**clear**()

**print**(student)

#删除字典

del student

条件判断

Python 编程中 if 语句用于控制程序的执行，基本形式为：

**if** 判断条件：

执行语句……

else 为可选语句，当需要在条件不成立时执行内容则可以执行相关语句

**if** 判断条件：

执行语句……

**else**：

执行语句……

案例1：根据分数来判断学生成绩是否为优秀，80分及以上为优秀，评级为A。

score=80

**if** score>=80:

**print**("Score is A")

**else**:

**print**("score is not A")

注意：print 语句要注意缩进，不要Tab和空格混用，否则回编译报错。

案例2：成绩80分以上为评级A，60~79分为B, 60分以下为C

score=90

**if** score>=80:

**print**("Score is A")

elif score>=60:

**print**("Score is B")

**else**:

**print**("Score is C")

Tips：elif是else if的缩写，完全可以有多个elif

循环语句

循环语句允许我们执行一个语句或语句组多次，Python提供了for循环和while循环（在Python中没有do..while循环）

### for循环

案例1：将Student 数组值全部打印出来

student=['Jack','Bob','Marry','Micle']

**for** stu **in** student:

**print**(stu)

案例2：计算1+2+3+...10的值

Python提供一个range()函数，可以生成一个整数序列，再通过list()函数可以转换为list。比如range(10)生成的序列是从0开始小于10的整数。

sum=0

**for** i **in** **range**(11):

sum=sum+i

**print**(sum)

### While循环

while循环，只要条件满足，就不断循环，条件不满足时退出循环。

**n**=10

**while** **n**>0:

**n**=**n**-1

**print**(**n**)

**print**('Game over!')

# 三、2第三章教案3-11~3-33(1015优化版)

**Python猜数小游戏**

【游戏规则】生成一个指定范围的随机数（如：1-100），然后玩家输入数值猜答案，屏幕会根据玩家输入的数字给出大小提示，一直到玩家猜出准确答案则游戏胜利并结束。

import random

answer=random.**randint**(1,100)

n=**int**(input("Please input num: "))

while n!=answer :

**if** n>answer:

n = **int**(input("Num is Big! Please Continue input: "))

elif n<answer:

n = **int**(input("Num is small! Please Continue input: "))

**print**("You Win the game!")

**Python 函数**

**函数概念**

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。 函数能提高应用的模块性，和代码的重复利用率。如print() range()函数，但你也可以自己创建函数，这被叫做用户自定义函数。

**函数定义**

案例：定义一个函数Max\_num()，用来比较两个数字的大小，然后将数值大的数字返回。

* 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
* 圆括号之间可以用于定义参数。
* 函数内容以冒号起始，并且缩进。
* return [表达式] 选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

def **Max\_num** (a,b):

**if** a>b:

**return** a

**elif** a<b:

**return** b

**else**:

**return** a

result=**Max\_num**(15,10)

print(result)

**Python 面向对象**

### 场景案例：

在自学网班级要求两名新同学Jack和Harry分别介绍自己的名字和来自哪座城市，然后分别说一句班级口号：“Hello,51zxw” 最终控制台打印效果如下：

My name **is** Jack **and** come **from** Beijing

Hello 51zxw!

My name **is** Harry **and** come **from** Shanghai

Hello 51zxw!

仅结合前面所学基础知识来实现：

name='Jack'

city='Beijing'

**print**("My name is %s and come from %s" %(name,city))

**print**("Hello 51zxw!")

name='Harry'

city='Shanghai'

**print**("My name is %s and come from %s" %(name,city))

**print**("Hello 51zxw!")

**？思考几个问题？**

1. 如果老师要求全班50个同学依次以上面形式自我介绍，该怎么办？
2. 每个同学介绍自己姓名和来自城市之后，再顺便介绍自己的年龄？
3. 每个同学自我介绍的代码块有何相同特征？

### 类与对象

现实世界中，随处可见的一种事物就是对象，对象是事物存在的实体，如人类、汽车、动物、水果这些都是一个抽象的类别，我们所见到的实物都是这些类的具体存在，因此**类是对象的抽象集合，对象是类的具体表现**。现实世界是万物皆对象！

人

* 属性：地域、肤色、国家.....
* 功能：走路、思考、饮食、繁衍、....
* 具体对象：中国人，非洲人

学生

* 属性：姓名，学号、城市、年龄....
* 功能：听，说、读、写...
* 具体对象：Jack同学，Harry同学

#### 面向对象关键要素

##### 类(Class):

用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。

##### 数据成员：

类的不同属性数据。

##### 对象：

对象是类的实例

##### 方法：

类中定义的函数，实现相关的功能。

#### 面向对象编程

简称OOP（Object Oriented Programming），是一种程序设计思想。OOP把对象作为程序的基本单元，一个对象包含了数据和操作数据的函数（方法）。

面向对象的设计思想是从自然界中来的，因为在自然界中，类（Class）和实例（Instance）的概念是很自然的。Class是一种抽象概念，比如我们定义的 Student类，是指学生这个概念，而实例（Instance）则是一个个体的Student对象。

Python是一门面向对象的语言，在Python中创建一个类和对象是很容易的。

#### 三大特性

* 封装
* 继承
* 多态

### Python 类与对象

#### 定义类

**class** **Student**(**object**):

类体

Class是类的定义的关键词，class后面紧接着是类名，即Student，类名通常是大写开头的单词，紧接着是(object)，表示该类是从哪个类继承下来的。通常如果没有明确的继承类，就使用object类，括号内一般为空默认就是继承Obejct类。这是所有类最终都会继承的类，也就是基类。

#### 属性初始化

由于类可以起到模板的作用，因此，可以在创建实例对象的时候，把一些我们认为必须绑定的属性强制填写进去。通过定义一个特殊的\_\_init\_\_方法，如：在创建Student实例的时候，就把name，city等属性绑上去：

**class** **Student**():

**def** **\_\_init\_\_**(self,name,city):

self.name=name

self.city=city

print("My name is %s and come from %s" % (name, city))

\_\_init\_\_方法的第一个参数永远是self，表示创建的实例本身，因此，在\_\_init\_\_方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量传进去。

#### 定义方法

类的方法除了第一个参数是self外，其他和普通函数一样。要调用一个方法，只需要在实例变量上直接调用，

**class** **Student**():

**def** **\_\_init\_\_**(self,name,city):

self.name=name

self.city=city

print("My name is %s and come from %s" % (name, city))

**def** **talk**(self):

print("Hello,51zxw")

#### 生成实例对象

stu1=**Student**('Jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

## 模 块

*为何要使用模块？*

随着项目功能和需求增多，代码量也会增大，把全部代码放在一个文件会显得冗余，因此需要使用模块进行分区管理。

Python模块是什么？

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

*使用模块有什么好处？*

最大的好处是大大提高了代码的可维护性。其次，编写代码不必从零开始。当一个模块编写完毕，就可以被其他地方引用。如：随机数模块，时间模块。

#### import语句

导入时间模块显示当前系统时间

*#模块 显示当前时间*

import **time**

**print**(**time**.ctime()) *#调用获取当前时间的方法*

导入随机数模块显示随机整数

import random

num=random.**randint**(1,10)

**print**(num)

### from ...import ....

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中。

from time import sleep

from Student import Student

stu1=**Student**('jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

**跨目录调用模块**

案例：调用School目录下的Student模块

from School.Student import Student

stu1=**Student**('jack','Beijing')

stu1.**talk**()

stu2=**Student**('Harry','Shanghai')

stu2.**talk**()

**import 搜索路径**

当你导入一个模块，Python 解析器对模块位置的搜索顺序是：

* 1、当前目录
* 2、如果不在当前目录，Python 则搜索 PYTHONPATH 下的每个目录。
* 3、如果都找不到，Python会察看安装默认路径。

Python异常

## 什么是异常？

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。 一般情况下，在Python无法正常处理程序时就会发生一个异常。 异常是Python对象，表示一个错误。 当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

## 常见异常类型

| **异常名称** | **描述** |
| --- | --- |
| FileNotFoundError | 找不到指定文件的异常 |
| NameError | 未声明/初始化对象 (没有属性) |
| BaseException | 所有异常的基类 |

## 异常处理语句

* try...except...
* try...except...finally
* raise

## 1.try....except

#### FileNotFoundError

try:

fileName=**input**("Please input fileName:")

**open**("%s.txt" %fileName)

except FileNotFoundError:

**print**("%s file not found " %fileName)

#### NameError

try:

**print**(stu)

except NameError:

**print**("stu not define !")

#### BaseException

try:

**print**(stu)

except BaseException:

**print**("stu not define !")

#### try…except…as…

try:

#stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

### try... except... else 使用

try:

stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

**else**:

**print**("stu is defined！")

### try..except...finally 输出

try:

#stu='Jack'

**print**(stu)

except BaseException as msg:

**print**(msg)

finally:

**print**("The end !")

### raise 抛出异常

### 

### 前面try语句是执行过程中捕获代码块的异常，而raise是通过事先定义一个条件，一旦符合异常条件就抛出异常。

**def** **division**(x,y):

**if** y==0:

**raise** ZeroDivisionError("Zero is not allow!")

**return** x/y

**try**:

division(8,0)

**except** BaseException **as** msg:

print(msg)



注意：raise只能用于Python标准异常类！

|  |  |
| --- | --- |
| **异常名称** | **描述** |
|  |  |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| **ZeroDivisionError** | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

文件处理

### 打开文件

使用Python内置的方法 open（）可以打开文件

file object = open(file\_name [, access\_mode][, buffering])

* file\_name：file\_name变量是一个包含了你要访问的文件名称的字符串值。
* access\_mode：access\_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。
* buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

f=**open**('stu\_info.txt','r')

f=**open**('E:\\test\\stu\_info.txt','r')

#### 常用文件打开模式

| **模式** | **描述** |
| --- | --- |
| r | 以只读方式打开文件。 |
| rb | 以二进制格式打开一个文件用于只读。 |
| w | 打开一个文件只用于写入。 |
| a | 打开一个文件用于追加。新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |

### 文件读取

line=f.**read**()

line1=f.**readline**()

line2=f.**readlines**()

* read() 每次读取整个文件，它通常用于将文件内容放到一个字符串变量中。
* readline() 每次只读取一行
* readlines()一次性读取文件所有行 自动将文件内容分析成一个行的列表，该列表可以由 Python 的 for ... in ... 结构进行处理。

### 关闭文件

f.close（）

#### 读取txt文件

案例：读取stu\_info.txt文件内容，并将所有文件中学生名称显示出来

f=**open**('stu\_info.txt','r')

lines=f.**readlines**()

**print**(lines)

**for** line **in** lines:

**print**(line.split(',')[0])

#### split()方法语法：

**str**.split(**str**="", num=string.**count**(**str**)).

参数

* str -- 分隔符，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。
* num -- 分割次数

### 读写csv文件

csv即为逗号分隔值（Comma-Separated Values，CSV），有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。

#### csv文件读取

案例：读取Stu\_info.csv文件里所有学生信息。

import csv

csv\_file=csv.**reader**(open('Stu\_info.csv','r'))

**for** stu **in** csv\_file:

**print**(stu)

#### csv文件写入

对Stu\_info.csv文件追加写入两个学生信息

stu=['Marry',28,'Changsha']

stu1=['Rom',23,'Chengdu']

out=**open**('Stu\_info.csv','a',newline='')

csv\_write=csv.**writer**(out,dialect='excel')

csv\_write.**writerow**(stu)

csv\_write.**writerow**(stu1)

**print**("Write File Over!")

Ctrl+d 复制当前行

xml文件处理

**什么是xml文件？**

xml即**可扩展标记语言**，它可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。

从结构上，很像HTML超文本标记语言。但他们被设计的目的是不同的，具体如下：

* XML 被设计用来传输和存储数据。
* HTML 被设计用来显示数据。

**<?xml version="1.0" encoding="utf-8"?>**

<note>

<to id='001'>George</to>

<from>John</from>

<heading>Reminder</heading>

<body>Don't forget the meeting!</body>

</note>

**xml特征：**

* 它是有标签对组成，<aa></aa>
* 标签可以有属性：<aa id='123'></aa>
* 标签对可以嵌入数据：<aa>abc</aa>
* 标签可以嵌入子标签（具有层级关系）

**XMl文件结构**

* XML 文档形成了一种树结构，它从“根部”开始，然后扩展到“枝叶”。
* 第一行是 XML 声明。它定义 XML 的版本 (1.0) 和所使用的编码
* <note>是根元素,也称为根节点。
* <to><from><heading><body>是子元素（子节点）
* XML 文档必须包含根元素。该元素是所有其他元素的父元素

**DOM文档对象模型**

**文档对象模型**（Document Object Model，简称DOM），DOM 就是针对 HTML 和 XML 提供的一个API。什么意思？就是说为了能以编程的方法操作这个 HTML 的内容（比如添加某些元素、修改元素的内容、删除某些元素），我们把这个 HTML或xml 看做一个对象树（DOM树），它本身和里面的所有东西比如 <div></div> 这些标签都看做一个对象，每个对象都叫做一个节点（node）。

**DOM 有什么用？**

就是为了操作 HTML或xml 中的元素，比如说我们要通过 JS 把这个网页的标题改了，直接这样就可以了：

document.title = '51zxw';

### 创建xml文件

创建一个xml文件Class\_info.xml 用来存储班级学生（姓名，年龄，城市），老师（姓名，年龄，城市）、教务账号（学生和老师的账号，密码）等信息。

**<?xml version="1.0" encoding="UTF-8" ?>**

<Class>

<student>

<name >Jack</name>

<age>28</age>

<city>Beijing</city>

</student>

<student>

<name >Bob</name>

<age>25</age>

<city>Shanghai</city>

</student>

<student>

<name>Harry</name>

<age>23</age>

<city>ShenZhen</city>

</student>

<teacher>

<name>Marry</name>

<age>23</age>

<city>Changsha</city>

</teacher>

<account>

<login username="student" password="123456"/>

<login username="teacher" password="888888"/>

</account>

</Class>

## xml节点

xml文件节点一般包含3类：

1. 元素节点
2. 文本节点
3. 属性节点

每个节点都拥有包含着关于节点某些信息的属性。这些属性是：

* nodeName（节点名称）
* nodeValue（节点值）
* nodeType（节点类型）

### 读取元素节点

### 案例：查看Class\_info.xml文件里Class节点的属性（结点名称，节点的值、节点类型）

from xml.dom import minidom

#加载xml文件

dom=minidom.**parse**('Class\_info.xml')

#加载dom对象元素

root=dom.documentElement

#打印节点信息

**print**(root.nodeName)

**print**(root.nodeValue)

**print**(root.nodeType)

* nodeName 节点名称
* nodeValue 返回文本节点的值
* nodeType 属性返回以数字值返回指定节点的节点类型。

1. 如果节点是元素节点，则 nodeType 属性将返回 1。
2. 如果节点是属性节点，则 nodeType 属性将返回 2。

### 读取文本节点的值

案例：分别打印出Class\_info.xml里的学生和老师的详细信息（姓名，年龄、城市）

from xml.dom import minidom

#获取标签对的值

#打开文件

dom=minidom.**parse**('Class\_info.xml')

#获取文档对象元素

root=dom.documentElement

#根据标签名称获取标签对象

names=root.**getElementsByTagName**('name')

ages=root.**getElementsByTagName**('age')

citys=root.**getElementsByTagName**('city')

#分别打印显示xml文档标签对里面的内容

**for** i **in** **range**(4):

**print**(names[i].firstChild.data)

**print**(ages[i].firstChild.data)

**print**(citys[i].firstChild.data)

### 读取属性节点的值

案例：分别读取打印老师和学生的账号密码。

from xml.dom import minidom

dom=minidom.**parse**('Class\_info.xml')

root=dom.documentElement

logins=root.**getElementsByTagName**('login')

#获取login标签的username属性

**for** i **in** **range**(2):

username=logins[i].**getAttribute**('username')

**print**(username)

password=logins[i].**getAttribute**('password')

**print**(password)

### 读取子节点信息

读取子节点<student>相关属性

* nodeName（节点名称）
* nodeValue（节点值）
* nodeType（节点类型）

from xml.dom import minidom

#加载xml文件

dom=minidom.**parse**('Class\_info.xml')

root=dom.documentElement

tags=root.**getElementsByTagName**('student')

**print**(tags[0].nodeName)

**print**(tags[0].tagName)

**print**(tags[0].nodeType)

**print**(tags[0].nodeValue)

多线程与多进程

#### 进程（Process）

是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

#### 线程（Thread）

有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。 线程是进程中的一个实体，是被系统独立调度和分派的基本单位，**一个进程可以包含多个线程，但是线程不能包含多个进程。**线程自己不拥有系统资源 ，在单个程序中同时运行多个线程完成不同的工作，称为多线程。

#### 线程与进程的区别

线程和进程的区别在于，子进程和父进程有不同的代码和数据空间，而多个线程则共享数据空间，每个线程有自己的执行堆栈和程序计数器为其执行上下文。

#### Tips：

LoadRunner和Jmeter性能测试工具也利用了**多线程和多进程**来构造多个并发用户来执行性能测试。

线程与进程图文解释

<http://www.ruanyifeng.com/blog/2013/04/processes_and_threads.html>

#### 单线程

单线程在程序执行时，所走的程序路径按照连续顺序排下来，前面的必须处理好，后面的才会执行。

案例：一个学生先用2秒说话，接着用3秒写字，最后结束。

**from** time **import** ctime,sleep

**def** **talk**():

print("Start talk %r" %ctime())

sleep(2)

**def** **write**():

print("Start Write! %r" %ctime())

sleep(3)

**if** \_\_name\_\_=='\_\_main\_\_'

:

talk()

write()

print("All end %r" %ctime())

if \_\_**name\_\_**=="\_\_**main\_\_**": 表示如果当前模块是被直接运行的，则该语句之后代码块被运行，如果模块是被导入的，则代码块不被运行。

#### 多线程

多线程（MultiThreading）是指从软件或者硬件上实现多个线程并发执行的技术。

案例：让学生同时进行说和写操作

from time import ctime,sleep

import threading

def **talk**(content,loop):

**for** i **in** **range**(loop):

**print**("Start Talk %s %s" %(content,ctime()))

**sleep**(3)

def **write**(content,loop):

**for** i **in** **range**(loop):

**print**("Start Write %s %s" %(content,ctime()))

**sleep**(5)

threads=[]

t1=threading.**Thread**(target=talk,args=('Speak: Hello,51zxw',2))

threads.**append**(t1)

t2=threading.**Thread**(target=write,args=('Write: Life is Short You need Python!',2))

threads.**append**(t2)

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

**for** t **in** threads:

t.**start**()

**for** t **in** threads:

t.join() # join()做为一个线程的守护，保证所有线程都执行完毕

**print**("All the End %r" %ctime())

### 多进程

与多线程相比，多进程就是import multiprocessing 然后替换相应的方法multiprocessing.Process（）

from time import sleep,ctime

import multiprocessing

def **talk**(content,loop):

**for** i **in** **range**(loop):

**print**("Talk: %s %s" %(content,ctime()))

**sleep**(2)

def **write**(content,loop):

**for** i **in** **range**(loop):

**print**("Write: %s %s"%(content,ctime()))

**sleep**(3)

process=[]

p1=multiprocessing.**Process**(target=talk,args=('hello 51zxw',2))

process.**append**(p1)

p2=multiprocessing.**Process**(target=write,args=('Python',2))

process.**append**(p2)

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

**for** p **in** process:

p.**start**()

**for** p **in** process:

p.**join**()

**print**("All process is Run %s" %ctime())

#### 相关延伸：

* 进程间通信IPC(Interprocess communication)
* 线程锁，进程锁
* 生命周期
* 进程调度

Python 爬虫——爬取Web页面图片

从网页页面上批量下载jpg格式图片，并按照数字递增命名保存到指定的文件夹。

Web地址：[http://p.weather.com.cn/2017/06/2720826.shtml#p=1](http://p.weather.com.cn/2017/06/2720826.shtml" \l "p=1)

**import** urllib

**import** urllib.request

**import** re *#正则表达式*

*#解析页面*

**def** **load\_page**(url):

request=urllib.request.Request(url) *#发送网络请求*

response=urllib.request.urlopen(request)*#根据url打开页面*

data=response.read() *#获取页面响应数据*

**return** data

*#下载图片*

**def** **get\_image**(html):

regx=r'http://[\S]\*jpg' *#定义正则表达式，匹配页面图片元素*

pattern=re.compile(regx) *#编译表达式构造匹配模式*

get\_image=re.findall(pattern,repr(html)) *#进行正则匹配并返回结果*

num = 1

*#遍历获取的图片*

**for** img **in** get\_image:

image=load\_page(img)

*#将图片存入到指定文件夹*

**with** open('E:\\Photo\\%s.jpg' %num,'wb') **as** fb:

fb.write(image)

print("正在下载第 %s张图片" %num)

num = num + 1

print("下载完成！")

url='http://p.weather.com.cn/2017/06/2720826.shtml#p=1'

html=load\_page(url)

get\_image(html)

正则表达式相关知识：

<https://deerchao.net/tutorials/regex/regex.htm>

# 四、1 Webdriver课件4-1~4-12( (1015优化版))

**Webdriver概述**

Webdriver (Selenium2）是一种用于Web应用程序的自动测试工具，它提供了一套友好的API，与Selenium 1（Selenium-RC）相比，Webdriver 的API更容易理解和使用，其可读性和可维护性也大大提高。Webdriver完全就是一套类库，不依赖于任何测试框架，除了必要的浏览器驱动，不需要启动其他进程或安装其他程序，也不必像Selenium 1那样需要先启动服务。

**支持浏览器**

* Firefox （FirefoxDriver）
* IE（InternetExplorerDriver）
* Opera（OperaDriver）
* Chrome （ChromeDriver）
* safari（SafariDriver）

**支持语言**

* Java
* C#
* PHP
* Python
* Perl
* Ruby

**安装，卸载、查看**

安装：pip install selenium==XXXX

卸载 pip uninstall selenium

查看版本号：

pip show selenium

**多浏览器运行**

**启动Firefox**

* 1.FireFox 48以上版本
* Selenium 3.X +FireFox驱动——geckodriver
* 2.Firefox 48 以下版本
* Selenium2.X 内置驱动
* 驱动下载地址<https://github.com/mozilla/geckodriver/releases> d

**启动IE浏览器**

* IE 9以上版本
* Selenium3.X +IE驱动
* IE 9以下版本
* Selenium 2.X +IE驱动

**启动Chrome浏览器**

selenium2.x/3.x +Chrome驱动

**注意！**

各个驱动下载地址： <http://www.seleniumhq.org/download/>

浏览器位数的版本和驱动版本要一致！ 如果是32bit浏览器而Driver是64bit则会导致脚本运行失败！

第一个自动化测试脚本

案例：

* 启动火狐浏览器，
* 首先打开我要自学网页面，打印网页标题，等待3秒
* 打开百度首页，打印网页标题，再等待2秒
* 关闭浏览器。

from selenium import webdriver

from time import sleep

#加载浏览器驱动

driver=webdriver.**Firefox**()

#打开自学网页面

driver.**get**("http://www.51zxw.net")

**print**(driver.title)

**sleep**(3)

#打开百度首页

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

**print**(driver.title)

**sleep**(3)

#关闭浏览器

driver.**quit**()

### 浏览器操作

* 浏览器窗口大小设置
* 页面前进后退
* 页面刷新

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net")

driver.**maximize\_window**()//窗口最大化

**sleep**(2)

driver.**get**("http://www.51zxw.net/list.aspx?cid=615")

driver.**set\_window\_size**(400,800)//自定义窗口

driver.**refresh**()//刷新窗口

**sleep**(2)

driver.**back**()//返回上也一个页面

**sleep**(2)

driver.**forward**()//前进上一个页面

**sleep**(2)

driver.**quit**()//退出浏览器

**元素定位**

元素的定位应该是自动化测试的核心，要想操作一个元素，首先应该识别这个元素。

webdriver提供了一系列的元素定位方法，常用的有以下几种

* id
* name
* class name
* link text
* partial link text
* tag name
* xpath
* css selector

案例：打开百度首页，在搜索框自动输入“Selenium我要自学网”关键词，然后点击搜索按钮，查看搜索页面。

### id与name 定位

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

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

driver.**find\_element\_by\_id**("kw").**send\_keys**("Selenium我要自学网")

driver.**find\_element\_by\_name**("wd").**send\_keys**("Selenium我要自学网")

**sleep**(2)

driver.**find\_element\_by\_id**("su").**click**()

### tag\_name定位

案例：打开我要自学网页面，在用户名输入框输入用户名“selenium”

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.com")

#定位标签名为input的元素

driver.**find\_element\_by\_tag\_name**("input").**send\_keys**("selenium")

#获取页面所有标签名称为“input”的标签。

driver.**find\_elements\_by\_tag\_name**("input")[0].**send\_keys**("selenium")

**sleep**(3)

driver.**quit**()

### class\_name定位

根据标签中属性class来进行定位的一种方法

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

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

driver.**find\_element\_by\_class\_name**("s\_ipt").**send\_keys**("Selenium 我要自学网")

**sleep**(2)

driver.**find\_element\_by\_id**("su").**click**()

**sleep**(3)

driver.**quit**()

### link\_text定位

link\_text定位就是根据超链接文字进行定位。

**from** selenium import webdriver

**from** time import sleep

driver=webdriver.Firefox()

driver.get("http://www.51zxw.net/")

driver.find\_element\_by\_link\_text('程序开发').click()

sleep(3)

driver.find\_element\_by\_partial\_link\_text('神秘面纱').click()

### XPath定位

XPath即为XML路径语言，它是一种用来确定XML文档中某部分位置的语言。XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。

##### xpath绝对与相对定位

from selenium import webdriver

from **time** import **sleep**

driver=webdriver.Firefox()

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

*# 绝对路径定位*

driver.find\_element\_by\_xpath("/html/body/div[2]/div[1]/div/div[1]/div/form/span[1]/input").send\_keys("51zxw")

*# 利用元素熟悉定位--定位到input标签中为kw的元素*

driver.find\_element\_by\_xpath("//input[@id='kw']").send\_keys("Selenium")

*# 定位input标签中name属性为wd的元素*

driver.find\_element\_by\_xpath("//input[@name='wd']").send\_keys("Selenium")

*# 定位所有标签元素中，class属性为s\_ipt的元素*

driver.find\_element\_by\_xpath("//\*[@class='s\_ipt']").send\_keys("Python3")

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

**sleep**(3)

driver.quit()

#### Xpath层级与逻辑定位

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net/")

#层级和属性结合定位--自学网首页输入用户和名密码

driver.**find\_element\_by\_xpath**("//form[@id='loginForm']/ul/input[1]").**send\_keys**("51zxw")

driver.**find\_element\_by\_xpath**("//form[@id='loginForm']/ul/input[2]").**send\_keys**("66666")

#逻辑运算组合定位

driver.**find\_element\_by\_xpath**("//input[@class='loinp' and @name='username']").**send\_keys**("51zxw")

**sleep**(3)

driver.**quit**()

**Css定位**

Selenium极力推荐使用CSS 定位，而不是XPath来定位元素，原因是CSS 定位比XPath 定速度快，语法也更加简洁。

**CSS常用定位方法**

1. find\_element\_by\_css\_selector（）
2. #id id选择器根据id属性来定位元素
3. .class class选择器，根据class属性值来定位元素
4. [attribute='value'] 根据属性来定位元素
5. element>element 根据元素层级来定位 父元素>子元素

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

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

#根据id来定位

driver.**find\_element\_by\_css\_selector**('#kw').**send\_keys**("Selenium 我要自学网")

#根据class定位

driver.**find\_element\_by\_css\_selector**('.s\_ipt').**send\_keys**('python')

#通过属性来定位

driver.**find\_element\_by\_css\_selector**("[autocomplete='off']").**send\_keys**("selenium")

**sleep**(2)

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

driver.**get**("http://www.51zxw.net")

#通过元素层级来定位

driver.**find\_element\_by\_css\_selector**("form#loginForm>ul>input").**send\_keys**("51zxw")

**sleep**(2)

driver.**quit**()

# 四、2 Webdriver4-13~4-21

### 下拉菜单元素定位

案例：在我要自学网登录页面选择指定的保留时间。

##### 1.根据选项元素标签定位

from selenium import webdriver

from time import sleep

from selenium.webdriver.support.ui import Select

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net")

**sleep**(2)

#根据option标签来定位

driver.**find\_elements\_by\_tag\_name**('option')[1].**click**()

driver.**find\_element\_by\_css\_selector**("[value='2']").**click**()

**sleep**(2)

driver.**quit**()

##### 2.使用Select类定位

from selenium import webdriver

from time import sleep

from selenium.webdriver.support.ui import Select

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net")

**sleep**(2)

#利用Select类来进行定位

select = **Select**(driver.find\_element\_by\_css\_selector("[name='CookieDate']"))

select.**select\_by\_index**(2)

select.**select\_by\_visible\_text**("留一年")

select.**select\_by\_value**("1")

**sleep**(2)

driver.**quit**()

鼠标操作

#### 实现思路

* 需要引入ActionChains类
* 然后定位相关元素
* 在ActionChains().调用相关鼠标操作方法

**from** selenium import webdriver

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

**from** time import sleep

driver=webdriver.Firefox()

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

driver.maximize\_window()

driver.find\_element\_by\_css\_selector("#kw").send\_keys("Python")

**# 获取搜索框元素对象**

element=driver.find\_element\_by\_css\_selector("#kw")

sleep(3)

**#双击操作**

ActionChains(driver).double\_click(element).perform()

sleep(2)

**#右击操作**

ActionChains(driver).context\_click(element).perform()

sleep(3)

**#鼠标悬停**

above=driver.find\_element\_by\_css\_selector(".pf")

ActionChains(driver).move\_to\_element(above).perform()

sleep(4)

driver.quit()

### 键盘操作

案例： 在百度搜索关键词“Python” 然后将关键词复制或剪切到搜狗搜索框进行搜索

**from** selenium **import** webdriver

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

**from** time **import** sleep

driver=webdriver.Firefox()

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

driver.find\_element\_by\_css\_selector("#kw").send\_keys("Python")

sleep(2)

*#键盘全选操作 Ctrl+A*

driver.find\_element\_by\_css\_selector("#kw").send\_keys(Keys.CONTROL,'a')

*#键盘选择复制或剪切操作 Ctrl+C*

driver.find\_element\_by\_css\_selector("#kw").send\_keys(Keys.CONTROL,'c')

driver.find\_element\_by\_css\_selector("#kw").send\_keys(Keys.CONTROL,'x')

*#打开搜狗页面*

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

sleep(2)

*#粘贴复制内容*

driver.find\_element\_by\_css\_selector(".sec-input").send\_keys(Keys.CONTROL,'v')

sleep(2)

*#点击搜索按钮*

*# driver.find\_element\_by\_xpath("//input[@id='stb']").click()*

driver.find\_element\_by\_css\_selector("#stb").click()

sleep(3)

driver.quit()

元素等待

#### 概念

* 显示等待是针对某一个元素进行相关等待判定；
* 隐式等待不针对某一个元素进行等待，全局元素等待。

##### a.相关模块

* WebDriverWait 显示等待针对元素必用
* expected\_conditions 预期条件类（里面包含方法可以调用，用于显示等待）
* NoSuchElementException 用于隐式等待抛出异常
* By 用于元素定位

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait #注意字母大写

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

from selenium.common.exceptions import NoSuchElementException

#### 显示等待

案例：检测百度页面搜索按钮是否存在，存在就输入关键词“自学网 Selenium” 然后点击搜索

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

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

from selenium.webdriver.common.by import By

from time import sleep

driver=webdriver.**Firefox**()

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

driver.**find\_element\_by\_css\_selector**("#kw").**send\_keys**("自学网 Selenium")

**sleep**(2)

#显示等待--判断搜索按钮是否存在

element=**WebDriverWait**(driver,5,0.5).**until**(EC.presence\_of\_element\_located((By.ID,"su")))

element.**click**()

**sleep**(3)

driver.**quit**()

#### 隐式等待

from selenium import webdriver

from selenium.common.exceptions import NoSuchElementException

from time import sleep,ctime

driver=webdriver.**Firefox**()

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

**sleep**(2)

driver.**implicitly\_wait**(5) #隐式等待时间设定 5秒

#检测搜索框是否存在

try:

**print**(ctime())

driver.**find\_element\_by\_css\_selector**("#kw").**send\_keys**("Python")

driver. **find\_element\_by\_css\_selector**("#su").click

except NoSuchElementException as msg:

**print**(msg)

finally:

**print**(ctime())

**sleep**(3)

driver.**quit**()

# 四、3 webdriver 4-22~4-32课件

frame嵌套页面元素定位

案例：在Frame.html文件种定位搜狗搜索页面，进行搜索操作。

**from** selenium **import** webdriver

**from** time **import** sleep

driver=webdriver.Firefox()

*#设置网页文件路径，r代表路径转义*

file\_path=r'E:\Python\_script\Webdriver\Frame.html'

*#路径转义另一种写法*

*# file\_path='E:\\Python\_script\\Webdriver\\Frame.html'*

driver.get(file\_path)

*#切换到frame页面内*

driver.switch\_to.frame("search")

*#定位到搜索框按钮输入关键词*

driver.find\_element\_by\_css\_selector("#query").send\_keys("Python")

sleep(3)

driver.find\_element\_by\_css\_selector("#stb").click()

sleep(3)

driver.quit()

自己的写法：

**from** selenium **import** webdriver  
**from** time **import** sleep  
  
driver = webdriver.Firefox()  
*# 读取本地地址的三种方式  
# 一  
# driver.get('file:///F:/sa/Frame.html)  
# 二  
# file\_path = 'F:\\sa\\Frame.html'  
# driver.get(file\_path)  
# 三*file\_path = **r'file:///F:\sa\Frame.html'***# Firefox使用r'F:\sa\Frame.html'打开报错，需要在前面加上file:///*driver.get(file\_path)  
  
driver.switch\_to.frame(**'search'**)  
driver.find\_element\_by\_css\_selector(**'#query'**).send\_keys(**'Selenium'**)  
driver.find\_element\_by\_css\_selector(**'#stb'**).click()  
  
sleep(2)  
driver.quit()

多窗口切换操作

**案例：打开我要自学网Selenium课程主页，然后打开2-1课程详情页面，再回到课程主页打开3-1课程详情页面**

**from** selenium import webdriver

**from** time import sleep

driver=webdriver.Firefox()

**#打开Selenium课程页面**

driver.get("http://www.51zxw.net/list.aspx?cid=615")

**# 获取课程主页的窗口句柄**

selenium\_index=driver.current\_window\_handle

sleep(2)

**#点击2-1课程链接。进入课程详情页面**

driver.find\_element\_by\_partial\_link\_text('2-1').click()

sleep(4)

**#跳转到课程主页窗口，点击3-1课程**

driver.switch\_to.window(selenium\_index)

sleep(3)

driver.find\_element\_by\_partial\_link\_text('3-1').click()

sleep(3)

driver.quit()

警告弹窗处理

#### 案例：点击百度首页设置按钮，然后进入搜索设置页面，点击“保存设置”或“恢复默认”按钮，处理警告弹窗窗口

from selenium import webdriver

from **time** import **sleep**

driver=webdriver.Firefox()

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

driver.find\_element\_by\_link\_text('设置').click()

**sleep**(2)

driver.find\_element\_by\_link\_text('搜索设置').click()

**sleep**(3)

driver.find\_element\_by\_link\_text('保存设置').click()

*# driver.find\_element\_by\_link\_text('恢复默认').click()*

**sleep**(3)

*#处理警告窗口*

alert=driver.switch\_to\_alert()

alert.**accept**()

**sleep**(2)

**sleep**(2)

driver.quit()

上传文件

案例：在百度搜索上传本地图片进行搜索。

**from** selenium **import** webdriver

**from** time **import** sleep

driver=webdriver.Firefox()

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

driver.find\_element\_by\_css\_selector(".soutu-btn").click()

sleep(3)

driver.find\_element\_by\_css\_selector(".upload-pic").send\_keys(r"E:\Python\_script\Webdriver\shuiyin.png")

sleep(3)

*# driver.quit()*

滚动条控制操作

案例：打开我要自学网页面，然后将滚动条拖到最底部，然后再拖到顶部

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net/")

**sleep**(2)

#将滚动调拖到最底部

js="var action=document.documentElement.scrollTop=10000"

driver.**execute\_script**(js)

**sleep**(2)

#将滚动条拖到最顶部

js="var action=document.documentElement.scrollTop=0"

driver.**execute\_script**(js)

**sleep**(3)

driver.**quit**()

网页截图操作

案例：分别打开我要自学网页面和百度页面，然后进行截图

**from** selenium **import** webdriver

**from** time **import** sleep

*#加载浏览器驱动*

driver=webdriver.Firefox()

*#打开自学网页面并截图*

driver.get("http://www.51zxw.net")

driver.get\_screenshot\_as\_file(r"E:\Python\_script\51zxw.jpg")

*#打开百度页面并截图*

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

driver.get\_screenshot\_as\_file(r"E:\Python\_script\baidu.png")

sleep(2)

driver.quit()

Cookie处理

### 什么是Cookie

Cookie是储存在用户本地终端上的数据，实际上是一小段的文本信息。

### Cookie作用

帮助 Web 站点保存有关访问者的信息，方便用户的访问。如记住用户名密码实现自动登录。

案例：查看访问我要自学网时的Cookie内容

from selenium import webdriver

driver=webdriver.**Firefox**()

driver.**get**("http://www.51zxw.net/")

#获取cookie全部内容

cookie=driver.**get\_cookies**()

#打印全部cookile信息

**print**(cookie)

#打印cookie第一组信息

**print**(cookie[0])

#添加cookie

driver.**add\_cookie**({'name':'51zxw','value':'www.51zxw.net'})

**for** cookie **in** driver.**get\_cookies**():

**print**("%s --- %s" %(cookie['name'],cookie['value']))

driver.**quit**()

自动化测试验证码问题

### 验证码作用

不少网站在用户登录、用户提交信息等登录和输入的页面上使用了验证码技术。验证码技术可以有效防止恶意用户对网站的滥用，使得网站可以有效避免用户信息失窃、保证网站稳定安全性。

但是验证码给自动化测试带来一些不便，使脚本无法正常运行覆盖功能模块。

### 如何解决

#### 1.去掉验证码

这是最简单的方法，对于开发人员来说，只是把验证码的相关代码注释掉即可，如果是在测试环境，这样做可省去了测试人员不少麻烦，如果自动化脚本是要在正式环境跑，这样就给系统带来了一定的风险。

#### 2.设置万能码

去掉验证码的主要是安全问题，为了应对在线系统的安全性威胁，可以在修改程序时不取消验证码，而是程序中留一个“后门”---设置一个“万能验证码”，只要用户输入这个“万能验证码”，程序就认为验证通过，否则按照原先的验证方式进行验证。

#### 3.验证码识别技术（OCR）

例如可以通过Python-tesseract 来识别图片验证码，Python-tesseract是光学字符识别Tesseract OCR引擎的Python封装类。能够读取任何常规的图片文件(JPG, GIF ,PNG , TIFF等)。不过，目前市面上的验证码形式繁多，目前任何一种验证码识别技术，识别率都不是100% 。

#### 4.记录cookie

通过向浏览器中添加cookie 可以绕过登录的验证码。

### 基于Cookie绕过验证码自动登录

案例：使用Cookie绕过百度验证码自动登录账户。

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

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

#手动添加cookie

driver.**add\_cookie**({'name':'BAIDUID','value':'9E4BF1D44014…(根据实际获取值填写)'})

driver.**add\_cookie**({'name':'BDUSS','value':'根据实际抓包获取值填写'})

**sleep**(2)

driver.**refresh**()

**sleep**(3)

driver.**quit**()

参考文档：

http://www.cnblogs.com/fnng/

# 五、第五章课件

自动化测试模型

**概念**

### 自动化测试模型可以看作自动化测试框架与工具设计的思想。自动化测试不仅仅是单纯写写脚本运行就可以了，还需要考虑到如何使脚本运行效率提高，代码复用、参数化等问题。自动化测试模型分为四大类：线性模型，模块化驱动测试、数据驱动、关键词驱动。

本地Web测试站点搭建：

* 工具：帝国CMS
* 下载地址：<http://www.phome.net/download/>

**- 安装步骤**

* 1、解压安装包；
* 2、将安装包的“EmpireServer”目录复制到D盘根目录；(D:\EmpireServer目录名不可更改)
* 3、双击“D:\EmpireServer\一键安装.bat”；(用vista或windows7以上则要鼠标右键以管理员身份运行)
* 4、至此，运行环境及帝国CMS全部安装完毕；
* 5、在浏览器打入：<http://localhost/e/admin> 后回车，进入帝国CMS后台登陆界面。
* 前台地址：<http://localhost>
* 后台地址：<http://localhost/e/admin> (登录用户名、密码与认证码均为admin)
* 搭建完成之后注册一个测试账号 51zxw 密码：123456

### 线性模型

线性脚本中每个脚本都相互独立，且不会产生其他依赖与调用，其实就是简单模拟用户某个操作流程的脚本。

案例：在帝国软件主页自动登录和退出操作

from selenium import webdriver

from time import sleep

driver=webdriver.**Firefox**()

driver.**get**("http://localhost")

#输入用户名

driver.**find\_element\_by\_name**('username').**clear**()

driver.**find\_element\_by\_name**('username').**send\_keys**('51zxw')

#输入密码

driver.**find\_element\_by\_name**('password').**clear**()

driver.**find\_element\_by\_name**('password').**send\_keys**('123456')

#点击登陆

driver.**find\_element\_by\_name**('Submit').**click**()

**sleep**(3)

#退出

driver.**find\_element\_by\_link\_text**('退出').**click**()

**sleep**(2)

driver.**switch\_to\_alert**().**accept**()

**sleep**(3)

driver.**quit**()

### 模块化驱动测试

线性模型虽然每个用例都可以拿出来独立运行，但是用例之间重复代码很多，开发、维护成本高。其实把重复的操作代码封装为独立的公共模块，当用例执行时需要用到这部分，直接调用即可，这就是模块驱动的方式。比如登录系统、退出登录、截图函数等等。

from selenium import webdriver

from time import sleep

class **Login**():

def **user\_login**(self,driver):

driver.**find\_element\_by\_name**('username').**clear**()

driver.**find\_element\_by\_name**('username').**send\_keys**('51zxw')

driver.**find\_element\_by\_name**('password').**clear**()

driver.**find\_element\_by\_name**('password').**send\_keys**('123456')

driver.**find\_element\_by\_name**('Submit').**click**()

def **user\_logout**(self,driver):

driver.**find\_element\_by\_link\_text**('退出').**click**()

**sleep**(2)

driver.**switch\_to\_alert**().**accept**()

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

driver = webdriver.**Firefox**()

driver.**get**("http://localhost/")

driver.**implicitly\_wait**(10)

**Login**().**user\_login**(driver)

**Login**().**user\_logout**(driver)

调用登录模块

**from** LoginClass import \*

driver = webdriver.Firefox()

driver.get("http://localhost/")

driver.implicitly\_wait(10)

Login().user\_login(driver)

Login().user\_logout(driver)

### 数据驱动测试

模块驱动的模型虽然解决了脚本的重复问题，但是需要测试不同数据的用例时，模块驱动的方式就不很适合了。 数据驱动就是数据的改变从而驱动自动化测试的执行，最终引起测试结果的改变。 装载数据的方式可以是列表、字典或是外部文件（txt、csv、xml、excel），目的就是实现数据和脚本的分离。

from selenium import webdriver

from time import sleep

class **Login**():

def **user\_login**(self,driver,username,password):

driver.**find\_element\_by\_name**('username').**clear**()

driver.**find\_element\_by\_name**('username').**send\_keys**(username)

driver.**find\_element\_by\_name**('password').**clear**()

driver.**find\_element\_by\_name**('password').**send\_keys**(password)

driver.**find\_element\_by\_name**('Submit').**click**()

def **user\_logout**(self,driver):

driver.**find\_element\_by\_link\_text**('退出').**click**()

**sleep**(2)

driver.**switch\_to\_alert**().**accept**()

数据驱动调用——实现多个账户登录

**from** LoginClass\_Para import \*

**from** selenium import webdriver

**from** time import sleep

driver=webdriver.Firefox()

driver.get("http://localhost")

driver.implicitly\_wait(10)

Login().user\_login(driver,"51zxw",'123456')

sleep(3)

Login().user\_logout(driver)

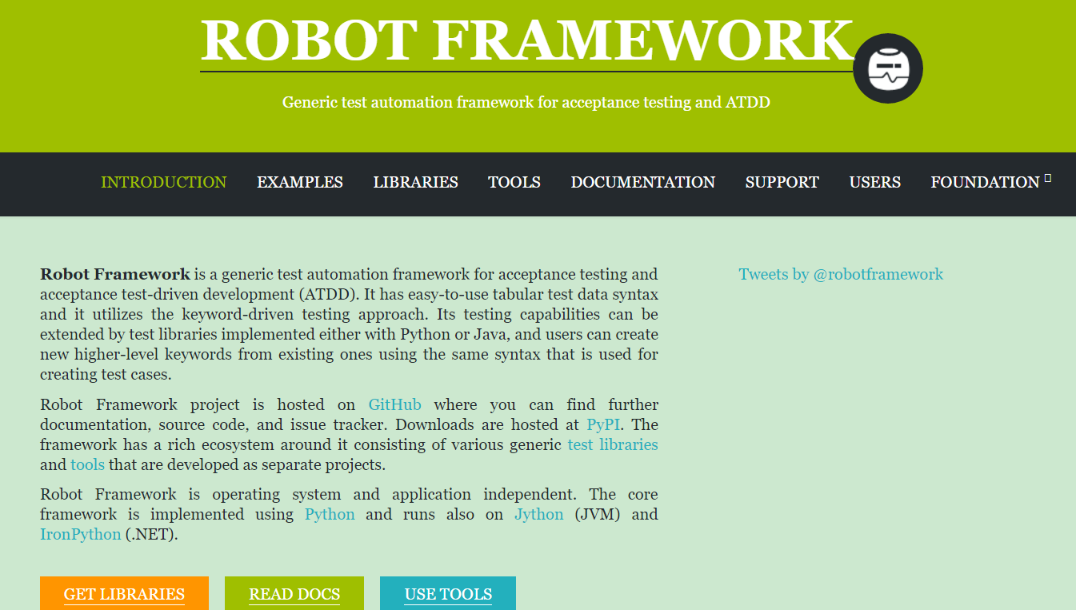
Login().user\_login(driver,"51zxwPro",'123456')

sleep(5)

Login().user\_logout(driver)

### 关键字驱动测试

通过关键字的改变引起测试结果的改变叫关键字驱动测试。 selenium IDE也是一种传统的关键字驱动的自动化工具，Robot Framework 是一个功能更强大的关键字驱动测试框架



# 六、第六章 unittest (1015优化版)

单元测试框架unittest

**单元测试**

单元测试（unit testing）是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。

**单元测试框架**

在单元测试框架出现之前，开发人员在创建可执行测试时饱受折磨。最初的做法是在应用程序中创建一个窗口，配有"测试控制工具(harness)"。它只是一个窗口，每个测试对应一个按钮。这些测试的结果要么是一个消息框，要么是直接在窗体本身给出某种显示结果。由于每个测试都需要一个按钮，所以这些窗口很快就会变得拥挤、不可管理。

单元测试框架提供了一种统一的编程模型，可以将测试定义为一些简单的类，这些类中的方法可以调用希望测试的应用程序代码。开发人员不需要编写自己的测试控制工具；单元测试框架提供了测试运行程序(runner)，只需要单击按钮就可以执行所有测试。利用单元测试框架，可以很轻松地插入、设置和分解有关测试的功能。测试失败时，测试运行程序可以提供有关失败的信息，包含任何可供利用的异常信息和堆栈跟踪。 不同编程语言有不同的单元测试框架，如Java 的Junit, TestNg, c#的 Nunit,Python的unittest,Pyunit,testtools, subunit....

**单元测试框架作用**

* 提供用例组织与执行
* 提供丰富的断言方法
* 提供丰富的日志与测试结果

**Python单元测试框架——unittest**

unittest官方文档 <https://docs.python.org/2.7/library/unittest.html>

unittest单元测试框架不仅可以适用于单元测试，还可以适用WEB自动化测试用例的开发与执行，该测试框架可组织执行测试用例，并且提供了丰富的断言方法，判断测试用例是否通过，最终生成测试结果。

**unittest核心要素**

1.TestCase

一个TestCase的实例就是一个测试用例。什么是测试用例呢？就是一个完整的测试流程，包括测试前准备环境的搭建(setUp)，执行测试代码(run)，以及测试后环境的还原(tearDown)。元测试(unit test)的本质也就在这里，一个测试用例是一个完整的测试单元，通过运行这个测试单元，可以对某一个问题进行验证。

2.TestSuite

而多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。 TestLoader是用来加载TestCase到TestSuite中的，其中有几个loadTestsFrom\_\_()方法，就是从各个地方寻找TestCase，创建它们的实例，然后add到TestSuite中，再返回一个TestSuite实例。

3.TextTestRunner

TextTestRunner是来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的run(result)方法。 测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

4.Fixture

而对一个测试用例环境的搭建和销毁，是一个fixture。

**unittest案例**

构造一个类Math 包含整数加法运算

**class** **Math**:

**def** **\_\_init\_\_**(self,a,b):

self.a=int(a)

self.b=int(b)

**def** **add**(self):

**return** self.a+self.b

对Math类进行单元测试

**from** caculator **import** Math

**import** unittest

**class** **TestMath**(unittest.TestCase):

**def** **setUp**(self):

print("Start test")

**def** **test\_add**(self):

j=Math(5,10)

self.assertEqual(j.add(),15)

*#用例失败场景*

*# self.assertEqual(j.add(),12)*

**def** **tearDown**(self):

print("test end")

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

*#构造测试集*

suite=unittest.TestSuite()

suite.addTest(TestMath("test\_add"))

*#执行测试*

runner=unittest.TextTestRunner()

runner.run(suite)

断言

断言内容是自动化脚本的重要内容，正确设置断言以后才能帮助我们判断测试用例执行结果。

断言方法

* assertEqual(a, b) 判断a==b
* assertNotEqual(a, b) 判断a！=b
* assertTrue(x) bool(x) is True
* assertFalse(x) bool(x) is False
* assertIs(a, b) a is b
* assertIsNot(a, b) a is not b
* assertIsNone(x) x is None
* assertIsNotNone(x) x is not None
* assertIn(a, b) a in b
* assertNotIn(a, b) a not in b
* assertIsInstance(a, b) isinstance(a, b)
* assertNotIsInstance(a, b) not isinstance(a, b)

**from** calculator **import** Math  
**import** unittest  
  
  
**class** Test1Math(unittest.TestCase):  
 **def** setUp(self):  
 print(**'开始'**)  
  
 **def** assertEqual\_test(self): *# 判断两个值是否相等* j = Math(18, 25)  
 self.assertEqual(j.add(), 35)  
  
 **def** assertNotEqual\_test(self): *# 判断两个值是否不相等* j = Math(18, 25)  
 self.assertNotEqual(j.add(), 35)  
  
 **def** assertTrue\_test(self): *# 判断是否为真* j = Math(18, 25)  
 self.assertTrue(j.add() > 35)  
  
 **def** assertFalse\_test(self): *# 判断是否为假* j = Math(18, 25)  
 self.assertNotEqual(j.add() == 35)  
  
 **def** assertIs\_test(self): *# 判断两个值是否相同* self.assertIs(**'selenium'**, **'selenium'**)  
  
 **def** assertIsNot\_test(self): *# 判断两个值是否不相同* self.assertIs(**'selenium'**, **'selenium'**)  
  
 **def** assertIsNone\_test(self): *# 判断值是否为空* self.assertIsNone(**'selenium'**)  
  
 **def** assertIsNotNone\_test(self): *# 判断值是否不为空* self.assertIsNotNone(**'selenium'**)  
  
 **def** assertIn\_test(self): *# 判断a是否被b包含* self.assertIn(**'selenium'**, **'selenium,python,java,android,php'**)  
  
 **def** assertNotIn\_test(self): *# 判断a是否不被b包含* self.assertNotIn(**'selenium'**, **'selenium,python,java,android,php'**)  
  
 *# self.assertIsInstance(a, b) # 判断a的数据类型是否为b，isinstance(a,b) 成立则通过，否则失败  
 # self.assertNotIsInstance # 判断同上相反* **def** tearDown(self):  
 print(**'结束'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 suite = unittest.TestSuite()  
 suite.addTest(Test1Math(**'assertNotEqual\_test'**))  
  
 runner = unittest.TextTestRunner()  
 runner.run(suite)

新增用例管理

前面是针对单个add方法来进行单元测试，如果需要多个方法来进行测试，该如何处理？如新增一个Sub方法来进行单元测试验证。

**class** **Math**:

**def** **\_\_init\_\_**(self,a,b):

self.a=int(a)

self.b=int(b)

**def** **add**(self):

**return** self.a+self.b

**def** **sub**(self):

**return** self.a-self.b

同时对add和Sub方法进行单元测试验证

**from** calculator **import** \*

**import** unittest

**class** **Test\_add**(unittest.TestCase):

**def** **setUp**(self):

print("Test is start")

**def** **test\_add**(self):

j=Math(5,5)

self.assertEqual(j.add(),10)

**def** **test\_add1**(self):

j=Math(10,20)

self.assertEqual(j.add(),30)

**def** **tearDown**(self):

print("test is end!")

**class** **Test\_sub**(unittest.TestCase):

**def** **setUp**(self):

print("Test is start")

**def** **test\_sub**(self):

i=Math(8,8)

self.assertEqual(i.sub(),0)

**def** **test\_sub1**(self):

i=Math(5,3)

self.assertEqual(i.sub(),2)

**def** **tearDown**(self):

print("test is end!")

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

suite=unittest.TestSuite()

suite.addTest(Test\_add("test\_add"))

suite.addTest(Test\_add("test\_add1"))

suite.addTest(Test\_sub("test\_sub"))

suite.addTest(Test\_sub("test\_sub1"))

runner=unittest.TextTestRunner()

runner.run(suite)

用例公共部分合并

上一节课程内容中，每个测试类都有SetUp()和tearDown()方法，而且两个方法内容都是一样的，用于打印开始与结束提示语句，是否可以合并在一起呢？

**from** calculator **import** \*

**import** unittest

**class** **Test\_StarEnd**(unittest.TestCase):

**def** **setUp**(self):

print("test start")

**def** **tearDown**(self):

print("test end")

**class** **Testadd**(Test\_StarEnd):

**def** **test\_add**(self):

j=Math(5,5)

self.assertEqual(j.add(),10)

**class** **TestSub**(Test\_StarEnd):

**def** **test\_sub**(self):

i=Math(3,2)

self.assertEqual(i.sub(),1)

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

unittest.main()

用例执行顺序

观察如下测试脚本，思考测试用例执行顺序。

**import** unittest

**class** **Test1**(unittest.TestCase):

**def** **setUp**(self):

print("Test1 start")

**def** **test\_c**(self):

print("test\_c")

**def** **test\_b**(self):

print("test\_b")

**def** **tearDown**(self):

print("test end")

**class** **Test2**(unittest.TestCase):

**def** **setUp**(self):

print("Test2 start")

**def** **test\_d**(self):

print("test\_d")

**def** **test\_a**(self):

print("test\_a")

**def** **tearDown**(self):

print("Test2 end!")

执行顺序规则——测试类或测试方法的数字与字母顺序 0~9，A-Z

### 用例综合框架管理

前面测试用例与执行都是写在一个文件，当用例数量不断增加的时候，用例的执行与管理变得非常麻烦，因此需要对用例根据具体的功能模块来使用单独的模块来管理。就像一所学校要根据不同年级进行分班管理，也是同样道理。

案例：Test\_Project 文件目录下包含4个python文件：

* l--StartEnd.py—— SetUp与TearDown管理
* l--calculatory.py——加减法运算方法的实现
* l--test\_add.py——加法测试用例
* l--test\_sub.py——减法测试用例
* l--runtest.py——用例执行管理

##### [calculatory.py](http://calculatory.py)

*#两个数相加或者相减*

**class** **Math**:

**def** **\_\_init\_\_**(self,a,b):

self.a=int(a)

self.b=int(b)

**def** **add**(self):

**return** self.a+self.b

**def** **sub**(self):

**return** self.a-self.b

##### [StartEnd.py](http://StartEnd.py)

**import** unittest

**class** **SetUp\_TearDown**(unittest.TestCase):

**def** **setUp**(self):

print("test start")

**def** **tearDown**(self):

print("test end")

##### test\_add.py

**from** calculator **import** \*

**from** StartEnd **import** \*

**class** **TestAdd**(SetUp\_TearDown):

**def** **test\_add**(self):

i=Math(5,5)

self.assertEqual(i.add(),10)

**def** **test\_add1**(self):

i=Math(8,8)

self.assertEqual(i.add(),16)

##### test\_sub.py

**from** calculator **import** \*

**from** StartEnd **import** \*

**class** **TestSub**(SetUp\_TearDown):

**def** **test\_sub**(self):

i=Math(5,5)

self.assertEqual(i.sub(),0)

**def** **test\_sub1**(self):

i=Math(8,5)

self.assertEqual(i.sub(),3)

##### [runtest.py](http://runtest.py)

* 使用discover 可以一次调用多个脚本
* test\_dir 被测试脚本的路径
* pattern 脚本名称匹配规则

import unittest

test\_dir='./'

discover=unittest.defaultTestLoader.discover(test\_dir,pattern='test\*.py')

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

runner=unittest.**TextTestRunner**()

runner.**run**(discover)

跳过测试和预期失败

#### 概要

* unittest.skip() 直接跳过测试
* unittest.skipIf() 条件为真，跳过测试
* unittest.skipUnless 条件为假，跳过测试
* unittest.expectedFailure 预期设置失败

### skip规则设定案例

**import** unittest

**class** **Test1**(unittest.TestCase):

**def** **setUp**(self):

print("Test1 start")

@unittest.skipIf(4 > 3, "skip Test\_d")

**def** **test\_c**(self):

print("test\_c")

*# @unittest.skipUnless(1<0, "skip test\_b")*

**def** **test\_b**(self):

print("test\_b")

**def** **tearDown**(self):

print("Test1 end")

@unittest.skip(" skip Test\_2")

**class** **Test2**(unittest.TestCase):

**def** **setUp**(self):

print("Test2 start")

**def** **test\_d**(self):

print("test\_d")

*# @unittest.expectedFailure*

**def** **test\_a**(self):

print("test\_a")

**def** **tearDown**(self):

print("Test2 end!")

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

unittest.main()

编写Web测试用例

案例：百度搜索关键词：“Selenium自学网” 并打开课程页面。

from selenium import webdriver

import unittest

from time import sleep

class **TestBaidu**(unittest.TestCase):

def **setUp**(self):

self.driver=webdriver.**Firefox**()

self.driver.implicitly\_wait(10)

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

def **test\_baidu**(self):

driver=self.driver

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

driver.**find\_element\_by\_id**("kw").**send\_keys**("Selenium自学网")

driver.**find\_element\_by\_id**("su").**click**()

**sleep**(3)

title=driver.title

self.**assertEqual**(title,"Selenium自学网\_百度搜索")

driver.**find\_element\_by\_partial\_link\_text**("Selenium自动化").**click**()

**sleep**(5)

def **tearDown**(self):

self.driver.quit()

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

unittest.**main**()

执行测试用例

import unittest

test\_dir='./test\_case'

discover=unittest.defaultTestLoader.discover(test\_dir,pattern="test\*.py")

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

runer=unittest.**TextTestRunner**()

runer.**run**(discover)

测试报告生成

自动化测试执行完成之后，我们需要生成测试报告来查看测试结果，使用HTMLTestRunner模块可以直接生成Html格式的报告。

#### 下载地址：

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

#### 下载后的修改:

* 94行引入的名称要改，从 import StringIO 改成import io。
* 539行 self.outputBuffer = StringIO.StringIO() 要改成self.outputBuffer=io.StringIO()
* 631行 print >>sys.stderr, '\nTime Elapsed: %s' % (self.stopTime-self.startTime)

修改为：print (sys.stderr, '\nTime Elapsed: %s' %(self.stopTime-self.startTime))

* 642行，if not rmap.has\_key(cls): 需要换成 if not cls in rmap:
* 766行的uo = o.decode('latin-1')，改成 uo=o
* 772行，把 ue = e.decode('latin-1') 直接改成 ue = e

#### 存放路径

将修改完成的模块存放在Python路径下Lib目录里面即可

import unittest

from HTMLTestRunner import HTMLTestRunner

import time

#定义测试用例路径

test\_dir='./test\_case'

discover=unittest.defaultTestLoader.discover(test\_dir,pattern="test\*.py")

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

#存放报告的文件夹

report\_dir='./test\_report'

#报告命名时间格式化

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

#报告文件完整路径

report\_name=report\_dir+'/'+now+'result.html'

#打开文件在报告文件写入测试结果

with **open**(report\_name,'wb')**as** f:

runer=**HTMLTestRunner**(stream=f,title="Test Report",description='Test case result')

runer.**run**(discover)

f.**close**()

测试报告美化

下载地址： <https://github.com/easonhan007/HTMLTestRunner>

#### 注意：

下载后也需要和前面的内容一样进行修改，然后放置在Python安装路径的Lib文件夹里

import unittest

from BSTestRunner import BSTestRunner

import time

test\_dir='./test\_case'

discover=unittest.defaultTestLoader.discover(test\_dir,pattern="test\*.py")

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

report\_dir='./test\_report'

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

report\_name=report\_dir+'/'+now+'result.html'

with **open**(report\_name,'wb')**as** f:

runer=**BSTestRunner**(stream=f,title="Test Report",description='Test case result')

runer.**run**(discover)

f.**close**()

参考资料来源：

<http://blog.csdn.net/huilan_same/article/details/52944782>

<http://blog.csdn.net/qq1124794084/article/details/51668672>

<http://www.cnblogs.com/nbkhic/p/5914400.html>

《Selenium2自动化实战》——虫师

# 七、第七章教案

### Python邮件发送

### SMTP（Simple Mail Transfer Protocol）

* 即简单邮件传输协议。它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。SMTP 协议属于TCP/IP协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。SMTP 服务器就是遵循 SMTP 协议的发送邮件服务器。

### SMTP 认证

* SMTP 认证，简单地说就是要求必须在提供了账户名和密码之后才可以登录 SMTP 服务器，这就使得那些垃圾邮件的散播者无可乘之机。
* 增加 SMTP 认证的目的是为了使用户避免受到垃圾邮件的侵扰。

更多资料： [http://help.163.com/09/1223/14/5R7P6CJ600753VB8.html](http://help.163.com/09/1223/14/5R7P6CJ600753VB8.html" \t "_blank)

### smtplib模块

Python内置对SMTP的支持，可以发送纯文本邮件、HTML邮件以及带附件的邮件。

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。

注意：使用前需要开启SMTP服务

案例：使用163邮箱来结合smtp模块发送邮件 准备工作：

import smtplib #发送邮件模块

from email.mime.text import MIMEText #定义邮件内容

from email.header import Header #定义邮件标题

#发送邮箱服务器

smtpserver='smtp.163.com'

#发送邮箱用户名密码

user='yuexiaolu2015@163.com'

password='…'

#发送和接收邮箱

sender='yuexiaolu2015@163.com'

receive='yuexiaolu2015@126.com'

#发送邮件主题和内容

subject='Web Selenium 自动化测试报告'

content='<html><h1 style="color:red">我要自学网，自学成才!</h1></html>'

#HTML邮件正文

msg=**MIMEText**(content,'html','utf-8')

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

msg['From']='yuexiaolu2015@163.com'

msg['To'] = 'yuexiaolu2015@126.com'

#SSL协议端口号要使用465

smtp = smtplib.**SMTP\_SSL**(smtpserver, 465)

**#HELO** 向服务器标识用户身份

smtp.**helo**(smtpserver)

#服务器返回结果确认

smtp.**ehlo**(smtpserver)

#登录邮箱服务器用户名和密码

smtp.**login**(user,password)

**print**("开始发送邮件...")

smtp.**sendmail**(sender,receive,msg.as\_string())

smtp.**quit**()

**print**("邮件发送完成！")

### 发送带附件的邮件

### 案例：发送E:\Python\_script\目录下 logo.png图片文件到指定的邮箱

import smtplib #发送邮件模块

from email.mime.text import MIMEText #定义邮件内容

from email.mime.multipart import MIMEMultipart #用于传送附件

#发送邮箱服务器

smtpserver='smtp.163.com'

#发送邮箱用户名密码

user='yuexiaolu2015@163.com'

password='070337shu'

#发送和接收邮箱

sender='yuexiaolu2015@163.com'

receives=['yuexiaolu2015@126.com','yuexiaolu2015@sina.com']

#发送邮件主题和内容

subject='Web Selenium 附件发送测试'

content='<html><h1 style="color:red">我要自学网，自学成才!</h1></html>'

#构造附件内容

send\_file=**open**(r"E:\Python\_script\logo.png",'rb').**read**()

att=**MIMEText**(send\_file,'base64','utf-8')

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

att["Content-Disposition"]='attachment;filename="logo.png"'

#构建发送与接收信息

msgRoot=**MIMEMultipart**()

msgRoot.**attach**(MIMEText(content, 'html', 'utf-8'))

msgRoot['subject']=subject

msgRoot['From']=sender

msgRoot['To'] = ','.**join**(receives)

msgRoot.**attach**(att)

#SSL协议端口号要使用465

smtp = smtplib.**SMTP\_SSL**(smtpserver, 465)

**#HELO** 向服务器标识用户身份

smtp.**helo**(smtpserver)

#服务器返回结果确认

smtp.**ehlo**(smtpserver)

#登录邮箱服务器用户名和密码

smtp.**login**(user,password)

**print**("Start send email...")

smtp.**sendmail**(sender,receives,msgRoot.as\_string())

smtp.**quit**()

**print**("Send End！")

### 整合测试报告发送

### 案例：获取…\Test\_Baidu\test\_report目录下最新的测试报告

import os #用于访问操作系统功能的模块

#报告存放位置

report\_dir='./test\_report'

**#os**.**listdir**() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表

lists=os.**listdir**(report\_dir)

#按时间顺序对该目录文件夹下面的文件进行排序

lists.**sort**(key=lambda fn:os.path.getatime(report\_dir+'\\'+fn))

**print**(lists)

**print**("latest report is :"+lists[-1])

#输出最新报告的路径

file=os.path.join(report\_dir,lists[-1])

**print**(file)

Python os模块相关知识： <http://www.cnblogs.com/MnCu8261/p/5483657.html>

### lambda 介绍 <http://www.cnblogs.com/evening/archive/2012/03/29/2423554.html>

发送测试报告

案例：将E:\Python\_script\unittest\Test\_Baidu生成的最新测试报告发送到指定邮箱。

**import** unittest

**from** BSTestRunner **import** BSTestRunner

**import** time

**import** smtplib *#发送邮件模块*

**from** email.mime.text **import** MIMEText *#定义邮件内容*

**from** email.header **import** Header *#定义邮件标题*

**import** os

**def** **send\_mail**(latest\_report):

f=open(latest\_report,'rb')

mail\_content=f.read()

f.close()

smtpserver='smtp.163.com'

*# 发送邮箱用户名密码*

user = 'yuexiaolu2015@163.com'

password = '…'

*# 发送和接收邮箱*

sender = 'yuexiaolu2015@163.com'

receives = ['yuexiaolu2015@126.com', 'yuexiaolu2015@sina.com']

*# 发送邮件主题和内容*

subject = 'Web Selenium 自动化测试报告'

*# HTML邮件正文*

msg = MIMEText(mail\_content, 'html', 'utf-8')

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

msg['From'] = sender

msg['To'] = ','.join(receives)

smtp = smtplib.SMTP\_SSL(smtpserver, 465)

*# HELO 向服务器标识用户身份*

smtp.helo(smtpserver)

*# 服务器返回结果确认*

smtp.ehlo(smtpserver)

*# 登录邮箱服务器用户名和密码*

smtp.login(user, password)

print("Start send Email...")

smtp.sendmail(sender, receives, msg.as\_string())

smtp.quit()

print("Send Email end!")

**def** **latest\_report**(report\_dir):

lists = os.listdir(report\_dir)

*# 按时间顺序对该目录文件夹下面的文件进行排序*

lists.sort(key=**lambda** fn: os.path.getatime(report\_dir + '\\' + fn))

print(("new report is :" + lists[-1]))

file = os.path.join(report\_dir, lists[-1])

print(file)

**return** file

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

test\_dir='./test\_case'

report\_dir='./test\_report'

discover = unittest.defaultTestLoader.discover(test\_dir, pattern="test\*.py")

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

report\_name = report\_dir + '/' + now + 'result.html'

**with** open(report\_name, 'wb') **as** f:

runner = BSTestRunner(stream=f, title="Test Report", description="baidu search")

runner.run(discover)

f.close()

*#h获取最新测试报告*

latest\_report=latest\_report(report\_dir)

*#发送邮件报告*

send\_mail(latest\_report)

### 163邮箱发生失败的常见问题

<http://help.163.com/09/1224/17/5RAJ4LMH00753VB8.html>

补充知识点——By 元素定位

from selenium import webdriver

from selenium.webdriver.common.by import By

from time import sleep

driver=webdriver.**Firefox**()

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

driver.**implicitly\_wait**(5)

driver.**find\_element**(By.ID,'kw').**clear**()

driver.**find\_element**(By.NAME,'wd').**send\_keys**("Selenium ")

driver.**find\_element**(By.CLASS\_NAME,'s\_ipt').**send\_keys**("自学网 ")

driver.**find\_element**(By.CSS\_SELECTOR,"#kw").**send\_keys**("自动化测试")

**sleep**(3)

driver.**find\_element**(By.ID,'su').**click**()

**sleep**(3)

driver.**quit**()

* find\_element([By.ID](http://By.ID),"loginName")
* find\_element([By.NAME](http://By.NAME),"SubjectName")
* find\_element(By.CLASS\_NAME,"u-btn-levred")
* find\_element(By.TAG\_NAME,"input")
* find\_element(By.LINK\_TEXT,"退出")
* find\_element(By.PARTIAL\_LINK\_TEXT,"退")
* find\_element(By.XPATH,".//\*[@id='Title")
* find\_element(By.CSS\_SELECTOR,"[type=submit]")

补充知识点——方法的参数个数

**def** **fun\_args1**(args):

print("args is %s" %args)

**def** **fun\_args2**(args1,args2):

print("args is %s and %s" %(args1,args2))

**def** **fun\_var\_args**(\*args):

**for** value **in** args:

print("args:", value)

*# fun\_args1('51zxw')*

*# fun\_args1()*

*# fun\_args2('51zxw','Python')*

*# fun\_args2('51zxw')*

fun\_var\_args("Python")

fun\_var\_args("hello","51zxw")

fun\_var\_args("Selenium",'Python','51zxw')

fun\_var\_args()

Page Object

Page Object是Selenium自动化测试项目开发实践的最佳设计模式之一，通过对界面元素和功能模块的封装减少冗余代码，同时在后期维护中，若元素定位或功能模块发生变化，只需要调整页面元素或功能模块封装的代码，提高测试用例的可维护性。

[BasePage.py](http://BasePage.py)

**from** time **import** sleep

**class** **Page**():

'''页面基础类'''

*#初始化*

**def** **\_\_init\_\_**(self, dirver):

self.base\_url = 'http://localhost'

self.driver = dirver

self.timeout = 10

*#打开不同的子页面*

**def** **\_open**(self, url):

url\_ = self.base\_url + url

print("Test page is： %s" %url\_)

self.driver.maximize\_window()

self.driver.get(url\_)

sleep(2)

**assert** self.driver.current\_url == url\_, 'Did ont land on %s' % url\_

**def** **open**(self):

self.\_open(self.url)

*#元素定位方法封装*

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

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

[LoginPage.py](http://LoginPage.py)

**from** BasePage **import** \*

**from** selenium.webdriver.common.by **import** By

**class** **LoginPage**(Page):

'''首页登录页面'''

url='/'

*#定位器*

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

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

submit\_loc=(By.NAME,'Submit')

*#用户名输入框元素*

**def** **type\_username**(self,username):

self.find\_element(\*self.username\_loc).clear()

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

*#密码输入框元素*

**def** **type\_password**(self,password):

self.find\_element(\*self.password\_loc).clear()

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

*#登录按钮元素*

**def** **type\_submit**(self):

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

*#登录功能模块封装*

**def** **test\_user\_login**(driver,username,password):

'''测试用户名密码是否可以登录'''

login\_page=LoginPage(driver)

login\_page.open()

login\_page.type\_username(username)

login\_page.type\_password(password)

login\_page.type\_submit()

loin\_test.py

**from** LoginPage **import** \*

**from** selenium **import** webdriver

driver=webdriver.Firefox()

username = '51zxw'

password = '123456'

test\_user\_login(driver, username, password)

sleep(3)

driver.quit()

参考文档：

<http://www.liaoxuefeng.com/>

<http://blog.csdn.net/menglei8625/article/details/7721746>

<http://blog.csdn.net/bravezhe/article/details/7659198>

<http://blog.csdn.net/spritzdance/article/details/5362220>

# 八、第八章教案

自动化测试综合实战

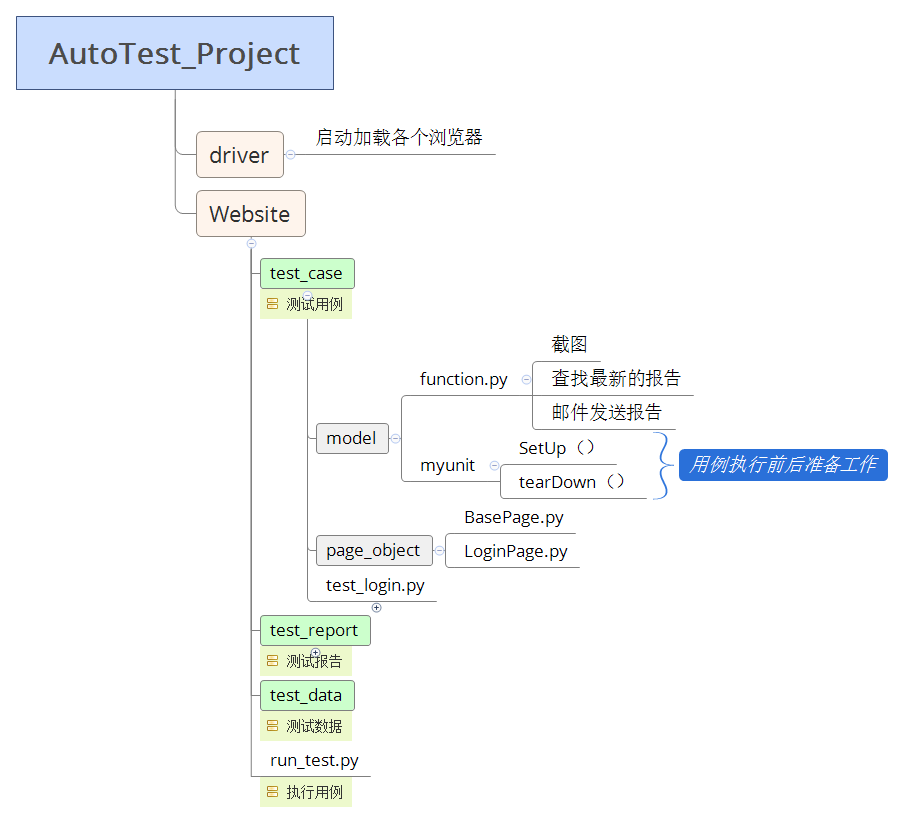
**项目背景**

在 <http://localhost/news/> 新闻子页面进行登录测试。

**功能实现**

* 自动运行用例
* 自动生成测试报告
* 自动断言与截图
* 自动将最新测试报告发送到指定邮箱
* PageObject+Unittest

**项目架构**

****

浏览器driver定义

**from** selenium **import** webdriver

*#启动浏览器驱动*

**def** **browser**():

driver = webdriver.Firefox()

*# driver = webdriver.Chrome()*

*# driver = webdriver.Ie()*

*# driver=webdriver.PhantomJS()*

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

**return** driver

*#调试运行*

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

browser()

用例运行前后的环境准备工作

**import** unittest

**from** driver **import** \*

**class** **StartEnd**(unittest.TestCase):

**def** **setUp**(self):

self.driver=browser()

self.driver.implicitly\_wait(10)

self.driver.maximize\_window()

**def** **tearDown**(self):

self.driver.quit()

工具方法模块（截图，查找最新报告、邮件发送）

**from** selenium **import** webdriver

**import** os

**import** smtplib

**from** email.mime.text **import** MIMEText

**from** email.header **import** Header

*#截图方法*

**def** **inser\_img**(driver,filename):

*#获取当前模块所在路径*

func\_path=os.path.dirname(\_\_file\_\_)

*# print("func\_path is %s" %func\_path)*

*#获取test\_case目录*

base\_dir=os.path.dirname(func\_path)

*# print("base\_dir is %s" %base\_dir)*

*#将路径转化为字符串*

base\_dir=str(base\_dir)

*#对路径的字符串进行替换*

base\_dir=base\_dir.replace('\\','/')

*# print(base\_dir)*

*#获取项目文件的根目录路径*

base=base\_dir.split('/Website')[0]

*# print(base)*

*#指定截图存放路径*

filepath=base+'/Website/test\_report/screenshot/'+filename

*# print(filepath)*

driver.get\_screenshot\_as\_file(filepath)

*#查找最新的测试报告*

**def** **latest\_report**(report\_dir):

lists = os.listdir(report\_dir)

*# print(lists)*

lists.sort(key=**lambda** fn: os.path.getatime(report\_dir + '\\' + fn))

*# print("the latest report is " + lists[-1])*

file = os.path.join(report\_dir, lists[-1])

*# print(file)*

**return** file

*#将测试报告发送到邮件*

**def** **send\_mail**(latest\_report):

f=open(latest\_report,'rb')

mail\_content=f.read()

f.close()

smtpserver = 'smtp.163.com'

user = 'yuexiaolu2015@163.com'

password = '...'

sender = 'yuexiaolu2015@163.com'

receives = ['yuexiaolu2015@126.com', 'yuexiaolu2015@sina.com']

subject = 'Web Selenium 自动化测试报告'

msg = MIMEText(mail\_content, 'html', 'utf-8')

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

msg['From'] = sender

msg['To'] = ','.join(receives)

smtp = smtplib.SMTP\_SSL(smtpserver, 465)

smtp.helo(smtpserver)

smtp.ehlo(smtpserver)

smtp.login(user, password)

print("Start send email...")

smtp.sendmail(sender, receives, msg.as\_string())

smtp.quit()

print("Send email end!")

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

driver=webdriver.Firefox()

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

inser\_img(driver,"sogou.png")

driver.quit()

#### Pageobject页面对象封装

[BasePage.py](http://BasePage.py) —— 基础页面类

**from** time **import** sleep

**class** **Page**():

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

self.driver=driver

self.base\_url="http://localhost"

self.timeout=20

**def** **\_open**(self,url):

url\_=self.base\_url+url

print('Test page is:%s' %url\_)

self.driver.maximize\_window()

self.driver.get(url\_)

sleep(2)

**assert** self.driver.current\_url == url\_, 'Did ont land on %s' % url\_

**def** **open**(self):

self.\_open(self.url)

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

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

[LoginPage.py](http://LoginPage.py) —— 新闻登录页面

**from** PageBase **import** \*

**from** selenium **import** webdriver

**from** selenium.webdriver.common.by **import** By

**class** **LoginPage**(Page):

'''新闻登录页面'''

url = '/news/'

*# 定位器——对相关元素进行定位*

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

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

submit\_loc = (By.NAME, 'Submit')

**def** **type\_username**(self, username):

self.find\_element(\*self.username\_loc).clear()

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

**def** **type\_password**(self, password):

self.find\_element(\*self.password\_loc).clear()

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

**def** **type\_submit**(self):

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

**def** **Login\_action**(self,username,password):

self.open()

self.type\_username(username)

self.type\_password(password)

self.type\_submit()

LoginPass\_loc = (By.LINK\_TEXT, '我的空间')

loginFail\_loc = (By.NAME, 'username')

**def** **type\_loginPass\_hint**(self):

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

**def** **type\_loginFail\_hint**(self):

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

test\_login.py ——unittest组织测试用例

* 用户名密码正确点击登录
* 用户名正确，密码错误点击登录
* 用户名和密码为空点击登录

**import** unittest

**from** model **import** function,myunit

**from** page\_object.LoginPage **import** \*

**from** time **import** sleep

**class** **LoginTest**(myunit.StartEnd):

*# @unittest.skip('skip this case')*

**def** **test\_login1\_normal**(self):

'''username password is normal'''

print("test\_login1\_normal is start run...")

po=LoginPage(self.driver)

po.Login\_action('51zxw',123456)

sleep(3)

*#断言与截屏*

self.assertEqual(po.type\_loginPass\_hint(),'我的空间')

function.insert\_img(self.driver,"51zxw\_login1\_normal.jpg")

print("test\_login1\_normal is test end!")

**def** **test\_login2\_PasswdError**(self):

'''username is ok,passwd is error!'''

print("test\_login2\_PasswdError is start run...")

po=LoginPage(self.driver)

po.Login\_action("51zxw",12342)

sleep(2)

self.assertEqual(po.type\_loginFail\_hint(),'')

function.insert\_img(self.driver,"51zxw\_login2\_fail.jpg")

print("test\_login2\_PasswdError is test end!")

*#*

*# @unittest.skip*

**def** **test\_login3\_empty**(self):

'''username password is empty'''

print("test\_login3\_empty is start run...")

po=LoginPage(self.driver)

po.Login\_action('','')

sleep(2)

*#断言与截屏*

self.assertEqual(po.type\_loginFail\_hint(),'')

function.insert\_img(self.driver,"51zxw\_login3\_empty.jpg")

print("test\_login3\_empty is test end!")

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

unittest.main()

run\_test.py——执行测试用例

import unittest

from function import \*

from BSTestRunner import BSTestRunner

import **time**

report\_dir = './test\_report'

test\_dir = './test\_case'

**print**("start run testcase...")

discover = unittest.defaultTestLoader.discover(test\_dir, pattern="test\_login.py")

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

report\_name = report\_dir + '/' + now + 'result.html'

**print**("start write report...")

*# 运行前记得把BSTestRunner.py 120行‘unicode’ 换成‘str’*

with **open**(report\_name, 'wb') as f:

runner = BSTestRunner(stream=f, title="Test Report", description="localhost login test")

runner.run(discover)

f.**close**()

**print**("find latest report...")

*# 查找最新的测试报告*

latest\_report = latest\_report(report\_dir)

*# 邮件发送报告*

**print**("send email report...")

send\_mail(latest\_report)

**print**("test end!")

### 浏览器内核

Webkit：目前最主流的浏览器内核，webkit是苹果公司开源的浏览器内核，其前身是KHTML。基于Webkit的浏览器很多，比如Safari，Chrome，Opera

Gecko：是Firefox浏览器的内核

Trident：是IE浏览器的内核

Blink：是webkit的一个分支版本，由google开发

### 无头浏览器

无头浏览器即headless browser，是一种没有界面的浏览器。既然是浏览器那么浏览器该有的东西它都应该有，只是看不到界面而已。

### PhantomJS

PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG.

PhantomJS可以说是目前使用最为广泛，也是最被认可的无头浏览器。由于采用的是Webkit内核，因此其和目前的Safari，Chrome等浏览器兼容性十分好。

### 为什么要使用PhantomJS？

PhantomJS 是一个无界面, 基于Webkit 的javascript 引擎. 一般来说我们的自动化脚本是须要运行在服务器上的, 往往这个时候系统并没有图形界面(如liunx服务器), 或者配置太低跑个浏览器实在是浪费. 而不需要图形界面的 PhantomJS 可谓是我们的不二之选.

### PhantomJS安装配置

下载地址：<http://phantomjs.org/download.html>

下载完成后将phantomjs.exe文件放置Python安装目录即可。

运行使用

driver=webdriver.**PhantomJS**()

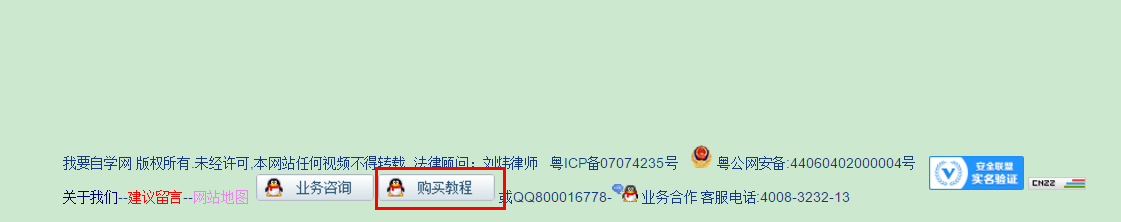
自学网App下载：

<http://www.51zxw.net/zt/aboutapp/downapp.html>



光盘购买：

点击网站底部“购买教程”按钮即可联系工作人员



参考文档

<http://www.cnblogs.com/diaosicai/p/6370321.html>

<http://blog.csdn.net/xtuhcy/article/details/52814210?locationNum=5>