作者：深圳51TESTING

# WebDriver基础

欢迎阅读WebDriver基础讲义。本篇讲义将会重点介绍Selenium WebDriver的环境搭建和基本使用方法。

## WebDriver环境搭建

> Selenium WebDriver 又称为 Selenium2。
>
> Selenium 1 + WebDriver = Selenium 2

WebDriver是主流Web应用自动化测试框架，具有清晰面向对象 API，能以最佳的方式与浏览器进行交互。

支持的浏览器：

- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer
- Opera
- Safari
- Apple iPhone
- Android browsers

### 环境搭建步骤

在上一篇中，我们已经确认使用Python来进行WebDriver的编码和操作。事实上Python+Selenium WebDriver环境的搭建分为两个部分：

1. 安装python
2. 安装Selenium

标准的安装步骤

1. 选择Python的版本。

   Python主流的有两个大的版本，2.7和3.5（请注意，从Python的3.5版本开始，不再支持Windows XP操作系统，Windows XP用户请安装3.4版本）。我们的例子将会选用面向未来的3.5版本。

2. 安装Python。

   在Python的官网（https://www.python.org/）下载最新的安装包，进行界面安装。

   安装的时候，推荐选择“Add exe to path”，将会自动添加Python的程序到环境变量中。然后可以在命令行输入`python -V`检测安装的Python版本。

   当前的版本安装中将会默认已经安装了`setuptools`和`pip`这两个Python的基本工具。如果使用了比较旧的Python版本的话，需要自行安装这两个工具。

   - setuptools：Python的基础工具包，用来构建、安装卸载Python程序
   - pip：Python软件包的安装和管理工具。通过pip可以简单的安装Python的任意类库

3. 安装Selenium2.0版本。

   在Windows安装Selenium2.0，有两种途径。使用pip命令行或者源码安装。以下两种方法，使用任何一个均可。推荐pip的方式。

   1. 方法一：pip命令行安装，运行 | cmd，打开命令行，`-U`其实就是`--upgrade`，升级安装。

      ```
      pip install -U selenium
      ```

   2. 方法二：源码解压安装，前往https://pypi.python.org/pypi/selenium下载最新版的PyPI版本的Selenium，解压后执行

      ```
      python setup.py install
      ```

### Ubuntu下的环境搭建

在Ubuntu下的Python的解释器一般情况下已经存在了。我们需要打开终端，输入`python -V`命令进行验证。如果没有安装Python，那么需要去

Python官网上下载指定版本的源文件，进行源码安装。安装完了以后并进行环境变量的设置。

安装Selenium WebDriver的方法与上述在Windows环境下安装部署的方法一致。依旧推荐使用pip命令行进行安装。

## 使用IDE编写Python

在上述环境搭建好以后，我们便可使用Python来编写自动化脚本程序，执行Selenium自动化测试。在此之前，我们依旧需要解决一个问题，那就是IDE的选择。

IDE，Integrated Development Environment，集成开发环境。一个好的编辑器或者好的IDE将会极大的提高生产力，帮我们做很多事情，使得编码工作更加简单，编码的体验更加容易。一般情况下，我们有以下几种工具可以选择：

- IDLE：Python自带的IDE，功能简单，使用方便
- Notepad++：一个强大的开源编辑器
- Vim：Linux系统中最好用的编辑器之一
- Sublime Text：一个非常轻便好用的现代化的编辑器，推荐。
- PyCharm：JetBrains公司提供的现代化的跨平台的Python IDE。

#### 使用Sublime Text 3搭建Python环境

1. ****通过快捷键`Ctrl` + 或者View > Show Console打开控制台，输入以下代码并回车

```
import urllib.request,os; pf = 'Package Control.sublime-package';
ipp = sublime.installed_packages_path();
urllib.request.install_opener( urllib.request.build_opener( urllib.request.ProxyHandler()) );
open(os.path.join(ipp, pf), 'wb').write(urllib.request.urlopen( 'http://sublime.wbond.net/' + pf.replace(' ','%20')).read(
```

2. 安装完以后，重启Sublime Text 3
3. 如果在Perferences->package settings中看到package control这一项，则安装成功
4. 按下Ctrl+Shift+P调出命令面板
5. 输入install package选项并回车
6. 输入i，匹配到以后，按回车，安装。Anaconda是一个终极的Python插件。安装完了以后便可用Sublime Text 3编写Python代码，并且使用`Ctrl` + `B`来编译执行。

一般来说，一个好的IDE提供一下功能，使你的编码开发工作变得更有效率：

1. 一个图形化的智能代码提示和补全功能
2. 可以轻松查看方法和类
3. 语法高亮
4. 提供单元测试和调试的工具
5. 源代码版本管理工具的支持

我们可以尝试用上述的编辑器或者IDE来进行Python代码编写工作。

## 开始使用WebDriver

接下来我们尝试几个简单的例子来体会一下Selenium WebDriver的使用。

示例1

```
## 引入WebDriver的包
from selenium import webdriver

## 创建浏览器对象
browser = webdriver.Firefox()

## 打开百度网站
browser.get('https://www.baidu.com/')
```

示例2

```
## 引入WebDriver包
from selenium import webdriver
```

```
## 引入WebDriver Keys包
from selenium.webdriver.common.keys import Keys

## 创建浏览器对象
browser = webdriver.Firefox()

## 导航到百度主页
browser.get('https://www.baidu.com')

## 检查标题是否为 ‘百度一下，你就知道’
assert '百度一下，你就知道' in browser.title

## 找到名字为wd的元素，赋值给elem
elem = browser.find_element_by_name('wd')  # 找到搜索框
elem.send_keys('seleniumhq' + Keys.RETURN)  # 搜索seleniumhq

## 关闭浏览器
browser.quit()
```

# 使用unittest编写测试脚本

通过上面的例子，我们可以简单的写出Selenium WebDriver的脚本，但是对于测试工作来说，上述的脚本还远远不够。因为上述的脚本没有“检查”。

接下来我们将会使用Python语言的`unittest`框架展开“检查”。

## unittest基础

`unittest`框架的原本的名字是PyUnit。是从JUnit这样一个被广泛使用的Java应用开发的单元测试框架创造而来。类似的框架还有NUnit（.Net开发的单元测试框架）等。

我们可以使用unittest框架为任意Python项目编写可理解的单元测试集合。现在这个unittest已经作为Python的标准库模块发布。我们安装完Python以后，便可以直接使用unittest。

unittest框架提供了编写`test cases`，`test suites`和`test fixtures`的基本功能。我们首先关注`Test cases`的编写与执行。

使用unittest需要以下简单的三步：

- 引入unittest模组
- 继承unittest.TestCase基类
- 测试方法以`test`开头

unittest示例

```
## 引入unittest模组
import unittest

## 定义测试类，名字为DemoTests
## 该类必须继承unittest.TestCase基类
class DemoTests(unittest.TestCase):

    ## 使用'@'修饰符，注明该方法是类的方法
    ## setUpClass方法是在执行测试之前需要先调用的方法
    ## 是开始测试前的初始化工作
    @classmethod
    def setUpClass(cls):
        pass

    ## 测试一（务必以test开头）
    def test_01(self):
        pass

    ## 测试三（务必以test开头）
    def test_02(self):
        pass

    ## 测试三（务必以test开头）
    def test_03(self):
        pass
```

```
    ## tearDownClass方法是执行完所有测试后调用的方法
    ## 是测试结束后的清除工作
    @classmethod
    def tearDownClass(cls):
        pass

# 执行测试主函数
if __name__ == '__main__':
    ## 执行main全局方法，将会执行上述所有以test开头的测试方法
    unittest.main(verbosity=2)
```

Python知识的补充：

1. Python文件的后缀名是`.py`
2. `.py`文件可以用来直接执行。也可以被用来作为模块导入。在cmd命令行中执行的命令为`python demo.py`
3. 在Python中导入模块（模组）一般使用的是`import`

###使用unittest框架编写Selenium WebDriver测试

接下来我们查看一个完整的Selenium WebDriver自动化测试实例

实例

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

class SearchTests(unittest.TestCase):

    @classmethod
    def setUp(self):
        # 创建一个新的浏览器对象
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.maximize_window()
        # 导航到京东主页
        self.driver.get("https://www.jd.com/")

    def test_search_by_category(self):
        # 获取到搜索框
        self.search_field = self.driver.find_element_by_id("key")
        self.search_field.clear()
        # 输入iphone 6s plus并按下回车进行搜索
        self.search_field.send_keys("iphone 6s plus"  + Keys.RETURN)
        # 获取所有的查询结果
        products = self.driver.find_elements_by_css_selector("li[class='gl-item']")
        # 断言：检查查询出来的个数是否为24个
        self.assertEqual(24, len(products))

    @classmethod
    def tearDown(self):
        # 关闭浏览器对象
        self.driver.quit()

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

###理解unittest框架提供的各种断言方法

| 方法 Method | 检查条件 |
| --- | --- |
| `assertEqual(a, b [, msg])` | a == b，msg可选，用来解释失败的原因 |
| `assertNotEqual(a, b [, msg]` | a != b，msg可选，用来解释失败的原因 |
| `assertTrue(x [, msg])` | x 是真，msg可选，用来解释失败的原因 |
| `assertFalse(x [, msg])` | x 是假，msg可选，用来解释失败的原因 |

| | |
|---|---|
| `assertIsNot(a, b [, msg])` | a 不是 b，msg可选，用来解释失败的原因 |

##WebDriver API 基本操作
这里我们将会开始WebDriver API的基本操作，从元素定位以及浏览器的基本操作开始。

## 定位符

元素的定位和操作是自动化测试的核心部分，其操作是建立在定位的基础上的。因此我们首选要开始定位元素。

在html里面，元素具有各种各样的属性。我们可以通过这样唯一区别其他元素的属性来定位到这个元素。WebDriver提供了一系列的元素定位方法。常见的有以下几种：

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

## 查找简单元素

我们从简单的一个元素开始定位。最基本的方法是`id`和`name`。大多数元素有这两个属性，在对控件的`id`和`name`命名是一般也会使其有意义，而取不同的名字。

例如我们看下面这段html

```
<input id="search" type="text" name="q" value="" class="input-text" maxlength="128" />
```

我们可以使用对应的方法来定位这个input

```
find_element_by_id('search')
find_element_by_name('q')
find_element_by_class_name('input-text')
```

这里我们开始用最简单的方式来尝试定位

查看下面一个例子

```
self.driver.get('http://pro.demo.zentao.net')
# 用name定位用户文本输入框
self.account_field = self.driver.find_element_by_name('account')
# 用name定位密码文本输入框
self.password_field = self.driver.find_element_by_name('password')
self.account_field.clear()
self.password_field.clear()
self.driver.implicitly_wait(30)

# 输入用户名demo
self.account_field.send_keys('demo')
# 输入密码123456
self.password_field.send_keys('123456')
self.driver.find_element_by_id('submit').click()
self.driver.implicitly_wait(30)

companyname = self.driver.find_element_by_id('companyname')
self.assertEqual('demo项目管理系统', companyname.text)
```

示例2

```
self.driver.find_element_by_id('menuproduct').click()
self.driver.implicitly_wait(30)

self.driver.find_element_by_id('menuproject').click()
self.driver.implicitly_wait(30)
```

```
self.driver.find_element_by_id('menuqa').click()
self.driver.implicitly_wait(30)

self.driver.find_element_by_id('menudoc').click()
self.driver.implicitly_wait(30)

self.driver.find_element_by_id('menureport').click()
self.driver.implicitly_wait(30)

self.driver.find_element_by_id('menucompany').click()
self.driver.implicitly_wait(30)

self.driver.find_element_by_link_text('退出').click()
self.driver.implicitly_wait(30)
```

此外`XPath`定位和`CSS Selector`定位，和定位一组元素这样的内容案例，将在后续的讲义继续探讨和讲解。

## 控制浏览器

浏览器的控制也是自动化测试的一个基本组成部分，我们可以将浏览器最大化，设置浏览器的高度和宽度以及对浏览器进行导航操作等。

```
## 浏览器最大化
driver.maximize_window()

## 设置浏览器的高度为800像素，宽度为480像素
driver.set_window_size(480, 800)

## 浏览器后退
driver.back()

## 浏览器前进
driver.forward()
```

这里我们做一个综合性的练习实例

```
# coding=utf-8
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
# import sys
# reload(sys)
# sys.setdefaultencoding('utf8')


class WebDriverTests(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        # create a new Firefox session
        cls.driver = webdriver.Firefox()
        cls.driver.get('about:blank')
        cls.driver.implicitly_wait(30)
        print(" -- set up finished -- ")
        print()

    def test_01_navigate(self):
        pass
        url_baidu = 'https://www.baidu.com/'
        url_zentao = 'http://pro.demo.zentao.net/user-login-Lw==.html'
        # 导航到百度
        self.driver.get(url_baidu)
        self.driver.maximize_window()
        self.driver.implicitly_wait(30)

        # 导航到禅道
        self.driver.get(url_zentao)
        self.driver.maximize_window()
        self.driver.implicitly_wait(30)
```

```
        # 后退
        self.driver.back()
        self.assertEqual(url_baidu, self.driver.current_url)
        self.driver.implicitly_wait(30)

        # 前进
        self.driver.forward()
        self.assertEqual(url_zentao, self.driver.current_url)
        self.driver.implicitly_wait(30)

        print("-- test 01 finished -- ")
        print()

    def test_02_element_interaction(self):
        self.driver.get('http://pro.demo.zentao.net')
        ## 找到用户名和密码的输入框
        self.account_field = self.driver.find_element_by_name('account')
        self.password_field = self.driver.find_element_by_name('password')
        ## 清除当前的输入
        self.account_field.clear()
        self.password_field.clear()
        self.driver.implicitly_wait(30)

        ## 输入用户名和密码，进行登录
        self.account_field.send_keys('demo')
        self.password_field.send_keys('123456')
        self.driver.find_element_by_id('submit').click()
        self.driver.implicitly_wait(30)

        companyname = self.driver.find_element_by_id('companyname')
        self.assertEqual('demo项目管理系统', companyname.text)
        print(companyname.get_attribute('type'))
        print()
        self.driver.implicitly_wait(30)

        print('-- test 02 finished -- ')
        print()

    def test_03_element_interation2(self):
        ## 这里执行了一段JavaScript代码
        js = 'selectTheme("green")'
        self.driver.execute_script(js)
        self.driver.implicitly_wait(30)

        js = 'selectTheme("red")'
        self.driver.execute_script(js)
        self.driver.implicitly_wait(30)

        js = 'selectTheme("lightblue")'
        self.driver.execute_script(js)
        self.driver.implicitly_wait(30)

        js = 'selectTheme("blackberry")'
        self.driver.execute_script(js)
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_id('menuproduct').click()
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_id('menuproject').click()
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_id('menuqa').click()
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_id('menudoc').click()
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_id('menureport').click()
        self.driver.implicitly_wait(30)
```

```
        self.driver.find_element_by_id('menucompany').click()
        self.driver.implicitly_wait(30)

        self.driver.find_element_by_link_text('退出').click()
        self.driver.implicitly_wait(30)

        WebDriverWait(self.driver, 10).until(
            expected_conditions.element_to_be_clickable((By.ID, "submit")))

        self.driver.implicitly_wait(30)

        print('-- test 03 finished -- ')
        print()

    def test_04_cookies(self):

        self.driver.add_cookie(
            {'name': 'key-neeeeew', 'value': 'value-neeeewwwww'})

        # 遍历cookies 中的name 和value 信息打印，当然还有上面添加的信息
        for cookie in self.driver.get_cookies():
            print("%s -> %s" % (cookie['name'], cookie['value']))
            print()

        self.driver.delete_all_cookies()

        cookies = self.driver.get_cookies()
        print(cookies)
        print()

        print('-- test 04 finished -- ')
        print()

    @classmethod
    def tearDownClass(cls):
        # close the browser window
        cls.driver.quit()
        pass
        print('-- tear down finished -- ')
        print()

if __name__ == '__main__':
    unittest.main(verbosity=2)
```