Selenium 高级自动化(图文教程)

作者：上海-悠悠

个人微信公众号：yoyoketang (扫二维码关注)

个人博客地址：www.cnblogs.com/yoyoketang/

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

目录

第 1 章 环境搭建

1.1 环境搭建

前言

目前 selenium 版本已经升级到 3.0 了，网上的大部分教程是基于 2.0 写的，所以在学习前先要弄清楚版本号,这点非常重要。本系列依然以 selenium2 为基础，目前 selenium3 坑比较多，暂时没精力去研究，后续会出相关教程。

1.1.1 selenium 简介

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

Selenium 的发展经历了三个阶段，第一个阶段，也就是 selenium1 的时代，在运行 selenium1.0 程序之前,我们得先启动 selenium server 端(selenium remote control)，我们简称 RC。RC 主要包括三个部

分：launcher，http proxy，selenium core。其中 selenium core 是被 selenium server 嵌入到浏览器页面中的，selenium core 内部是一堆 javascript 函数构成，通过调用这些函数来实现对浏览器的各种操作。

很显然比较繁琐，这并不是最佳自动化解决方案，于是后来有了 webdriver。

selenium2 的时代合并了 webdriver，也就是我们通常说的 selenium，selenium2 是默认支持 Firefox 浏览器的，这点非常方便。当然也支持其他更多浏览器，Ie 和 chrome 浏览器需要下载驱动包，并添加到环境变量下

selenium3 是 2016 年十月份左右出来的，并且现在默认安装都是 selenium3 了，selenium3 在 selenium2 的基础上做了一些调整，最明显的区别 就是 selenium2 对 Firefox 的支持最高只支持 46 及以下版本。selenium3 可以支持 47 以上版本，但是需要下载 geckodriver.exe 驱动，并添加到环境变量 path 下。

接下来的内容以 selenium2 为主

**********环境组合**********

初学者最佳环境：python2.7+selenium2+Firefox46 以下版本

喜欢尝新的环境：python3.6+selenium3+Firefox47 以上版本

****

小编的环境：

windows10 64 位

python 2.7.12

selenium 2.53.6

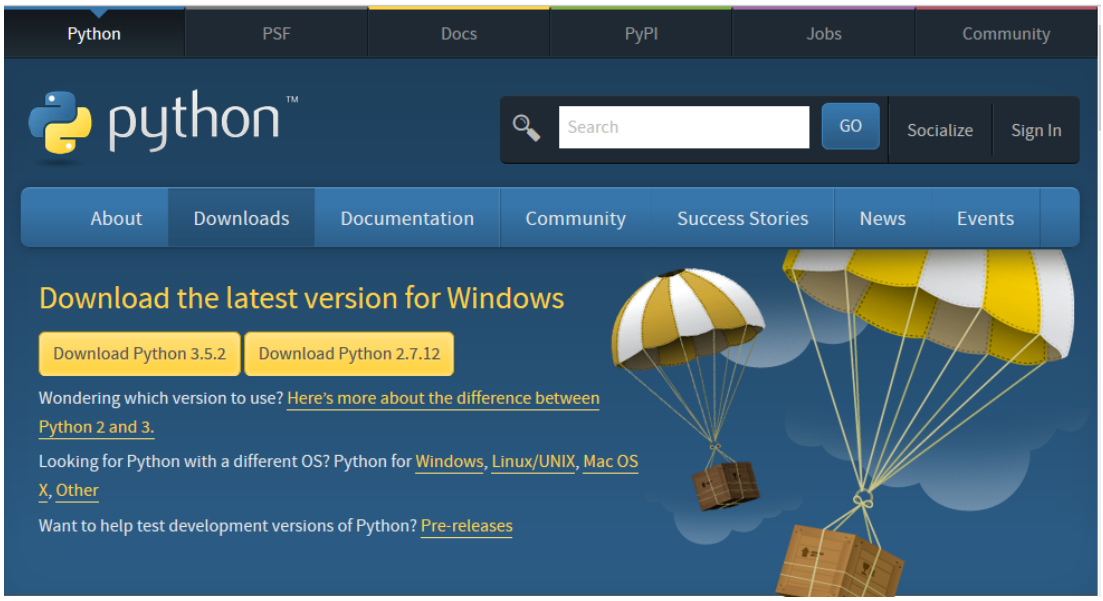
1.1.2 python 安装

1.小编的电脑操作系统：win10 64 位系统

2.下载 Python 安装包，选择 2.7 版本和 3.6 版本都可以

（下面的教程，两个版本会一起讲，所以不用担心版本问题）

官网下载地址：https://www.python.org/57

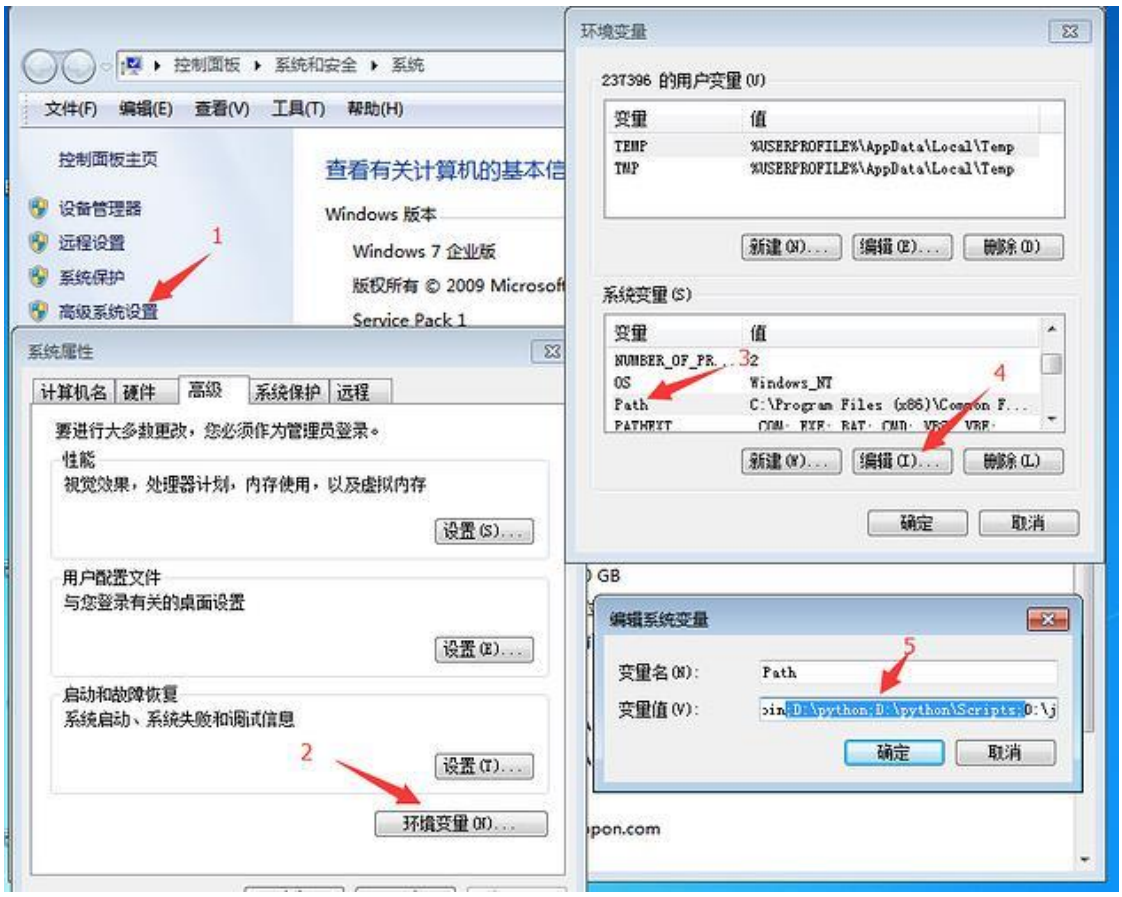


3. Python 安装，双击傻瓜式安装（别安装在 c 盘）

1.1.3 环境变量

1. 安装完成后，看下这个目录 D:\python\Scripts，有没 pip.exe 和 easy_install.exe（一般都有，没有的话得重新安装一次了）

2. 将 D:\python 和 D:\python\Scripts，添加到环境变量 path 下

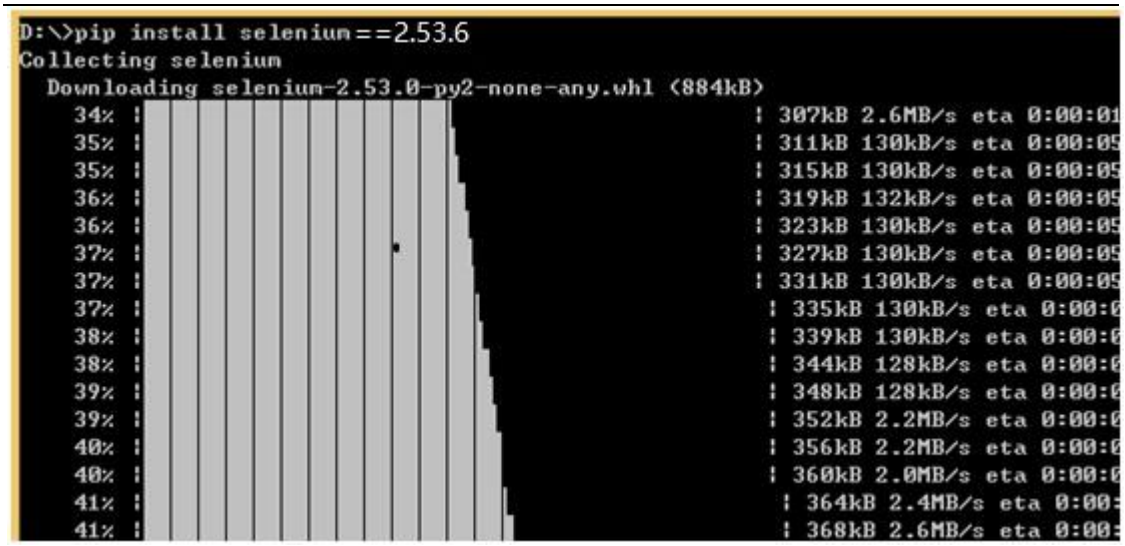


1.1.4 安装 selenium

1. 打开 cmd 窗口输入：pip(如果有内容显示，说明正常)

2. cmd 输入指令安装 selenium：pip install selenium==2.53.6

(注意：首次装一定要看到进度 100%完成，如果中途失败了，重新输入指令安装，直到看到 100%完成为止)



1.1.5 验证 selenium

如何才能知道 selenium 正确安装好了呢？

1.确保电脑上安装了 Firefox 浏览器

2.cmd 窗口输入如下指令

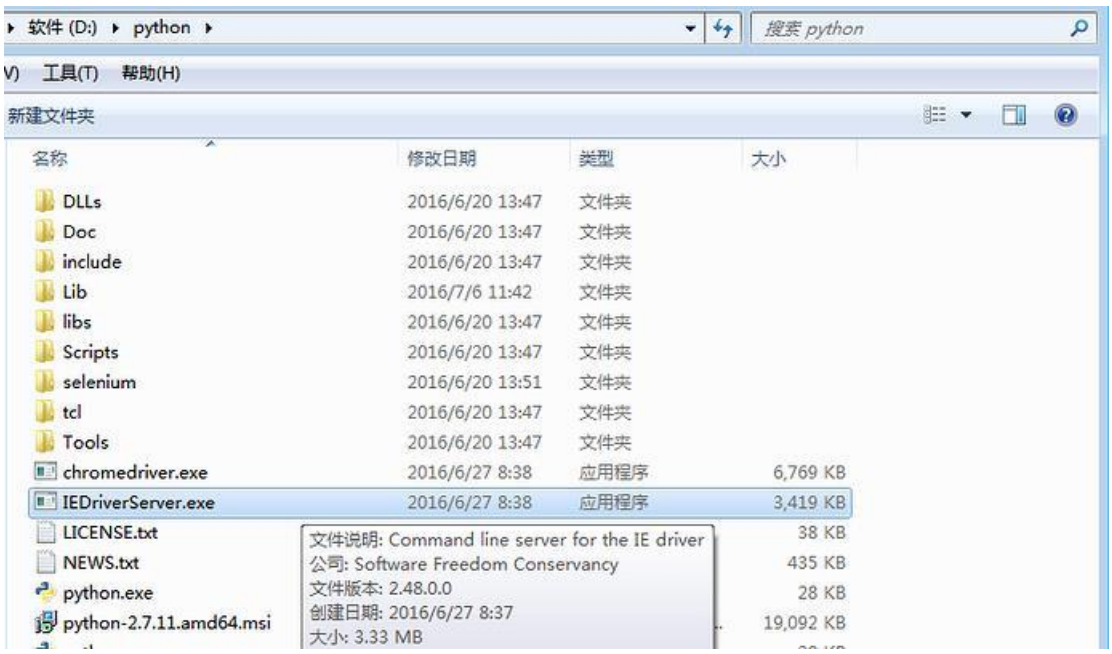
```
>python
>from selenium import webdriver
>webdriver.Firefox()
```

3.如果能启动浏览器，说明环境安装 OK。

1.1.6 浏览器

1.如果你打算用 Firefox 浏览器，那么千万别安装 47 以上版本（selenium2 不兼容 47 以上）

2.如果你打算用 Ie 或 Chrome 浏览器，需要先下载浏览器驱动，将驱动文件放到 python 根目录。



如果有的已经安装过 3.0 的版本，启动 firefox 时候会报错，下一章讲如何使用 pip 降级 selenium 版本

1.2 pip 降级 selenium3.0

selenium 版本安装后启动 Firefox 出现异常：'geckodriver' executable needs to be in PATH

selenium 默默的升级到了 3.0，然而网上的教程都是基于 selenium2 的，最近有不少小伙伴踩坑了，决定有必要出这一篇，帮助刚入门的小伙伴们解决好环境问题。

selenium+python 环境搭配：

selenium2+firefox46 以下版本（无需驱动包，firefox 喜欢偷偷升级，你懂的）

selenium3+firefix47 以上版本（必须下载驱动：geckodriver.exe，且添加到环境变量）

1.2.1 遇到问题

1. 安装完 selenium 后，再 cmd 进入 python 环境

2.从 selenium 导入 webdriver

3.启动 Firefox 浏览器

>>python

>>from selenium import webdriver

>>webdriver.Firefox()

然后出现以下异常：'geckodriver' executable needs to be in PATH

1.2.2 解决方案

1.'geckodriver' executable needs to be in PATH，这句话意思就是说，geckodriver.exe 的驱动文件需要添加到环境变量下，

selenium2 是默认支持 firefox 的，不需要驱动包，但是，selenium3 需要驱动包的支持了，于是就有了上面的问题

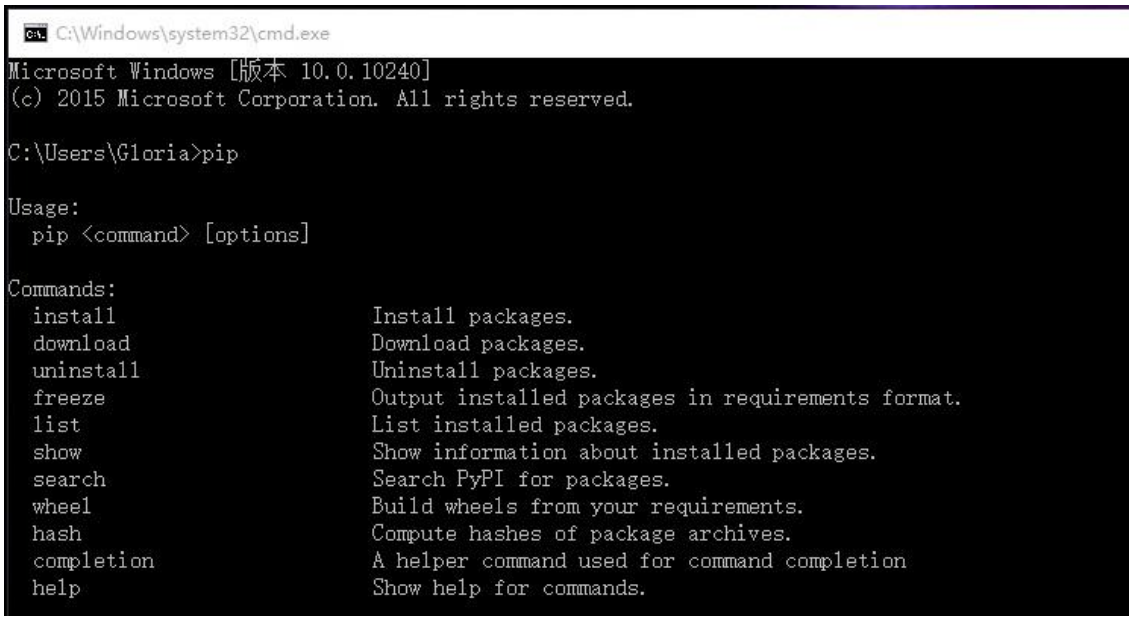
2.解决办法一：继续使用 selenium3,去下载驱动包，然后加到环境变量下（不推荐此办法，因为解决完这个问题后，后面还会接着有其它问题）

3.解决办法二：selenium3 降级到 selenium2(接下来会介绍)

1.2.3 检查 pip 环境

1.打开 cmd，输入 pip，检查 pip 环境是否正常

>>pip



2.如果输入 pip 出现提示:Did not provide a command 说明 pip 环境有问题，临时解决办法，输入 pip 时候加上后缀 pip.exe 就可以了，具体原因看下一篇解决办法。

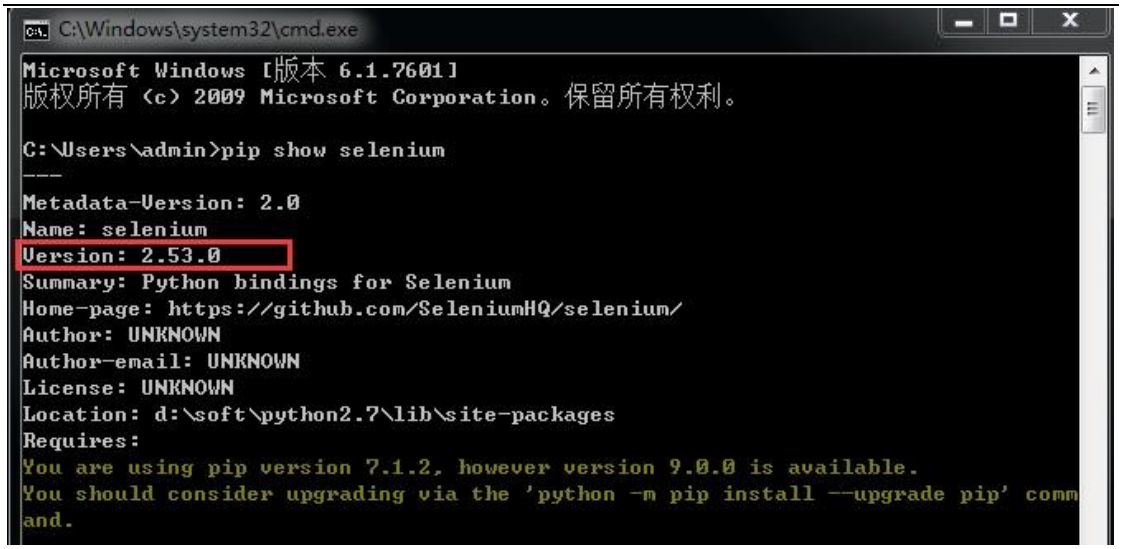
1.2.4 pip 查看 selenium 版本号

1.打开 cmd，输入 pip show selenium

>>pip show selenium

2.看红色区域位置版本号显示：2.53.0，显示的就是当前使用的版本号

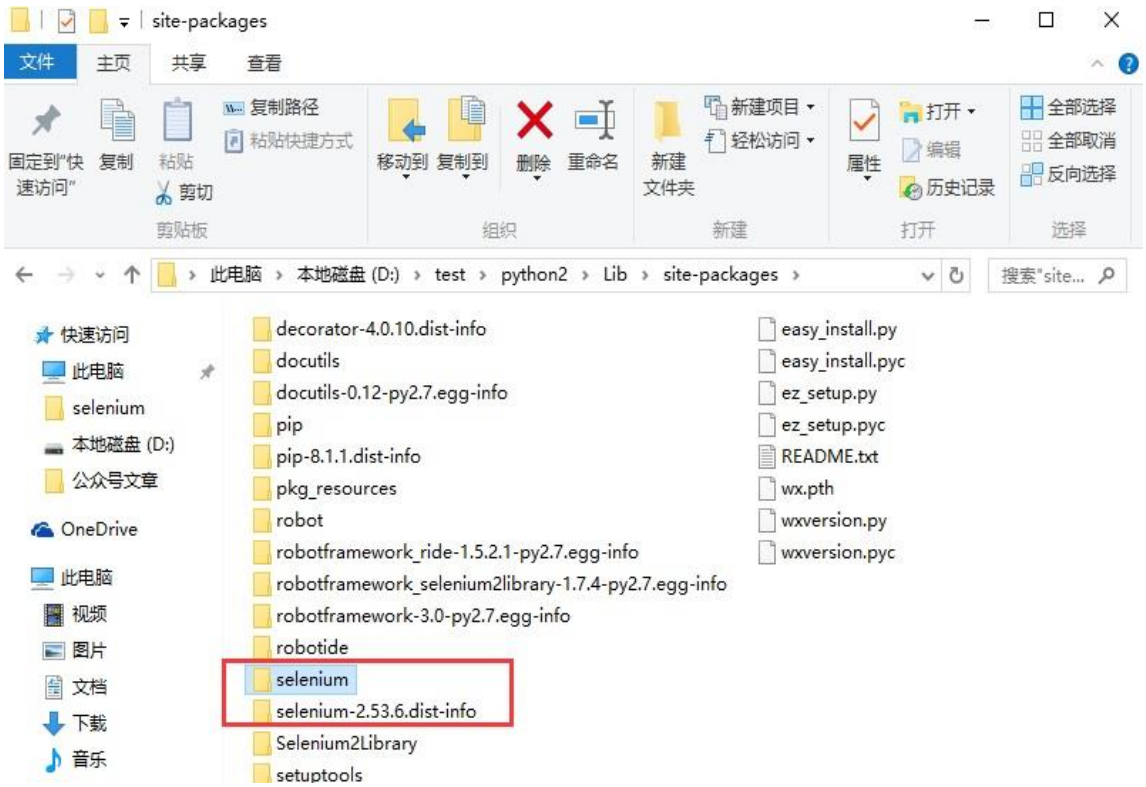
（如果你这里显示的是 3.0 开头，就需要接下来步骤了）



1.2.5 pip 降级 selenium

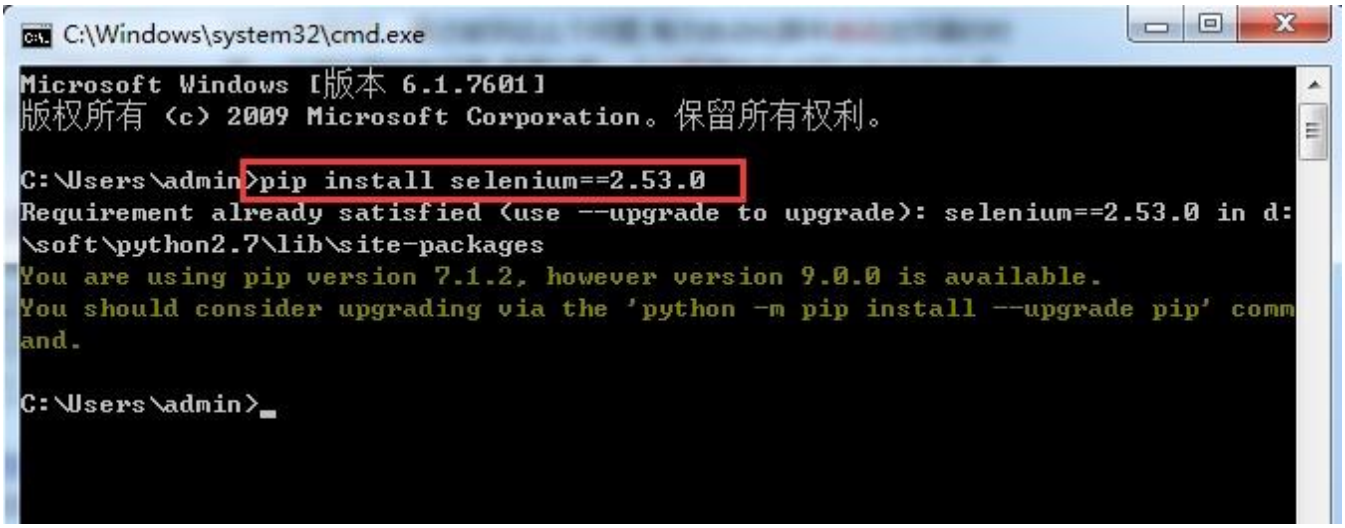
1.为了避免与之前安装的selenium版本冲突,先找到selenium3.0目录:python\Lib\site-packages 目录

把里面 selenium 开头的文件全部删除就可以了。python 所有的第三方包都在这个目录下面。



2.打开 cmd，输入 pip install selenium==2.53.6（注意是两个==，中间不要留空格，这里推荐 2.53.6 的版本）

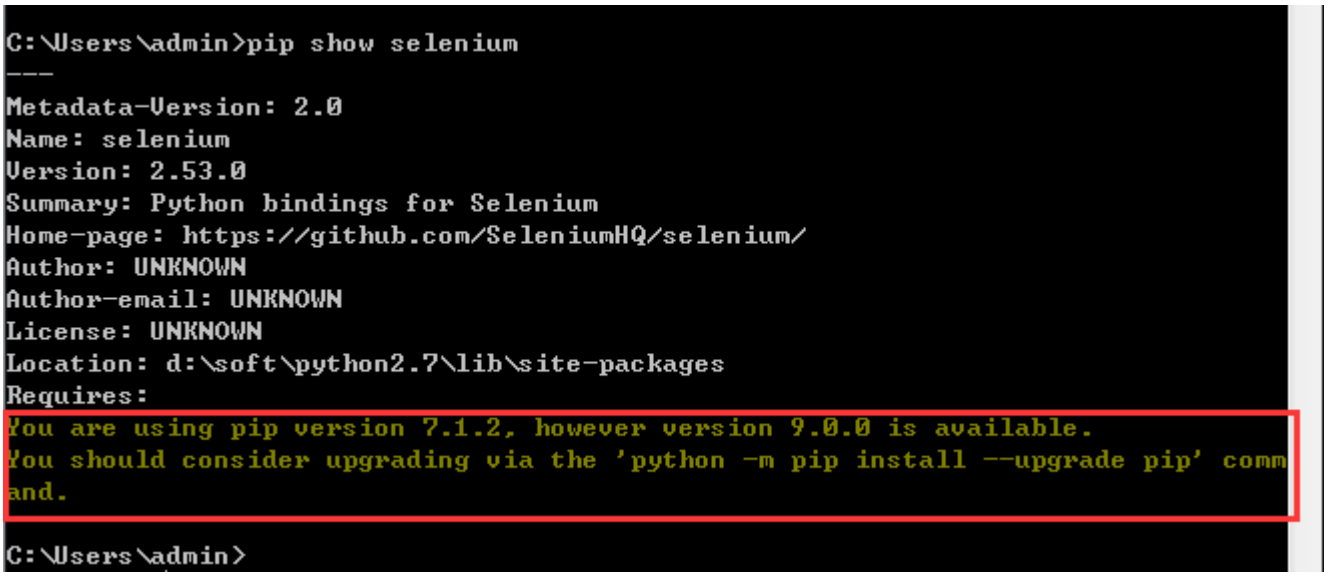
>>pip install selenium==2.53.6



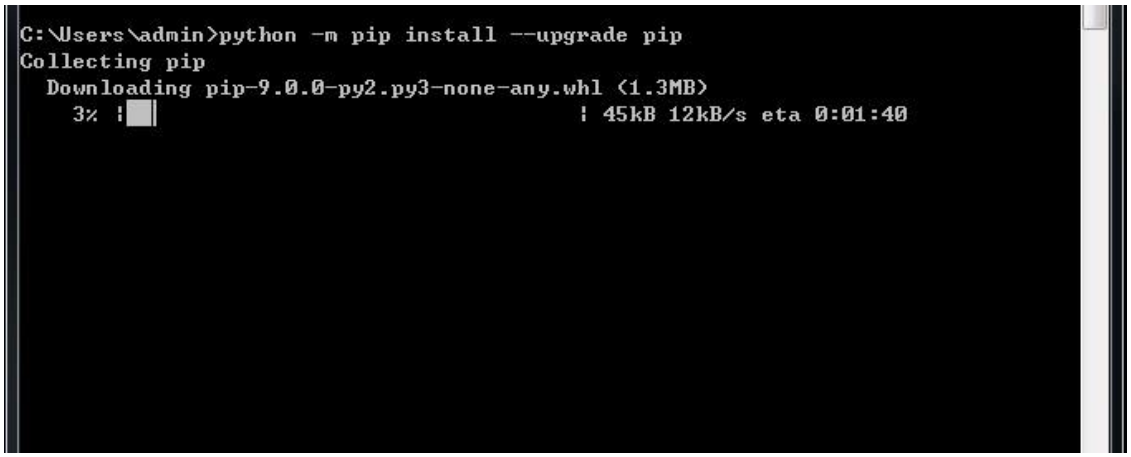
1.2.6 升级 pip 版本

1.在使用pip过程中如果出现下方红色区域字样,就是说pip版本过低了,建议升级

2.如何升级pip呢？看最后一句话:python -m pip install --upgrade pip



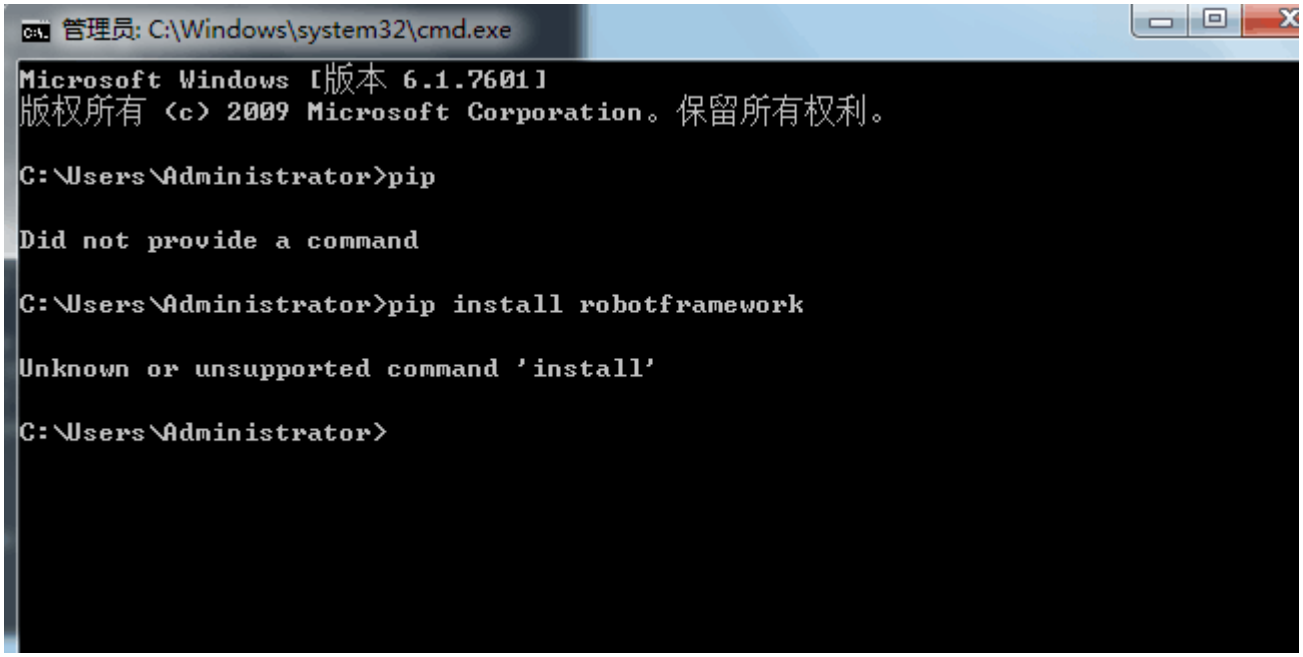
3.把上面对应的提示照着敲一遍就可以了



1.3 解决 pip 使用异常

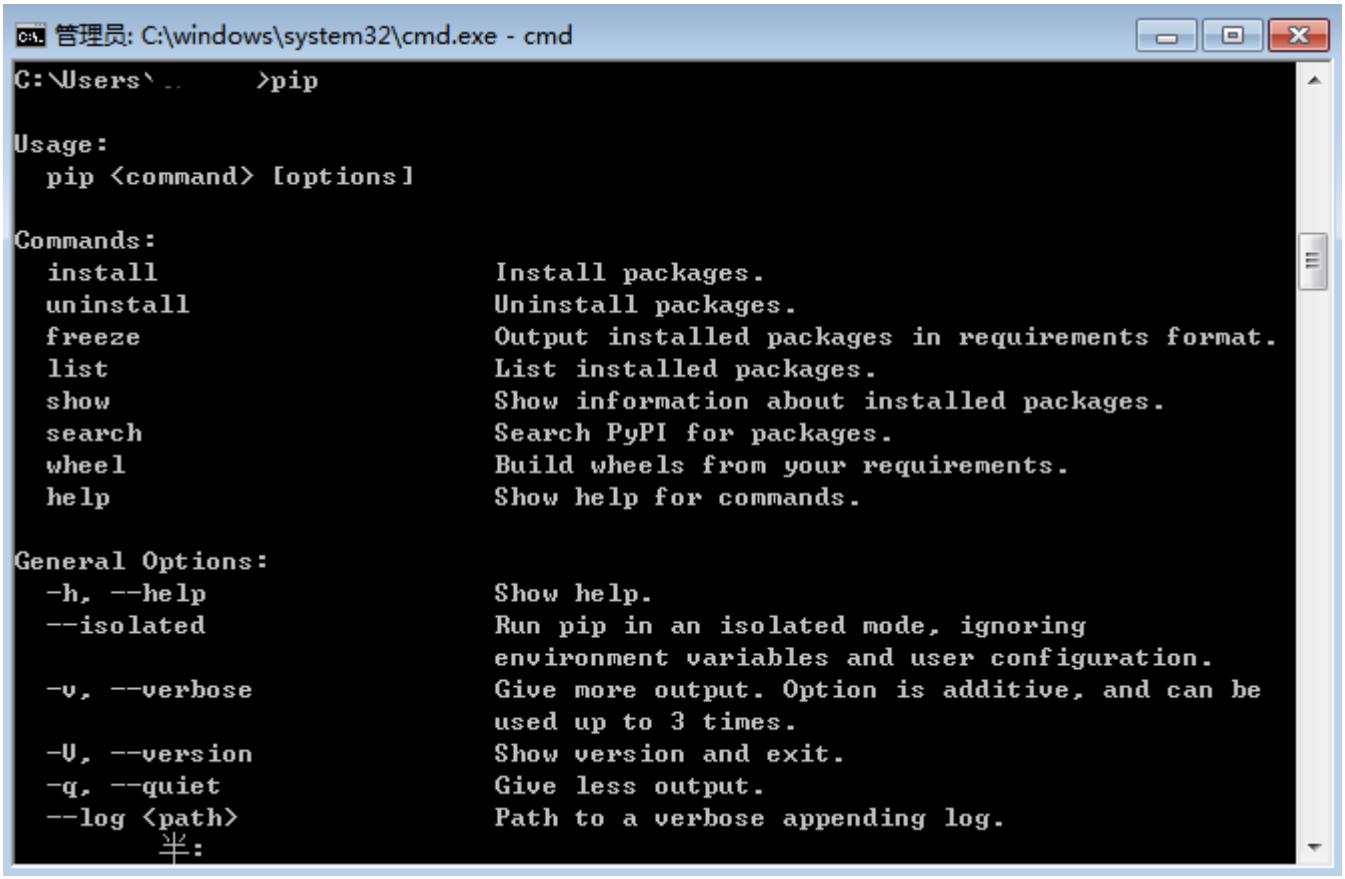
1.3.1 pip 出现异常

有一小部分童鞋在打开 cmd 输入 pip 后出现下面情况:Did not provide a command



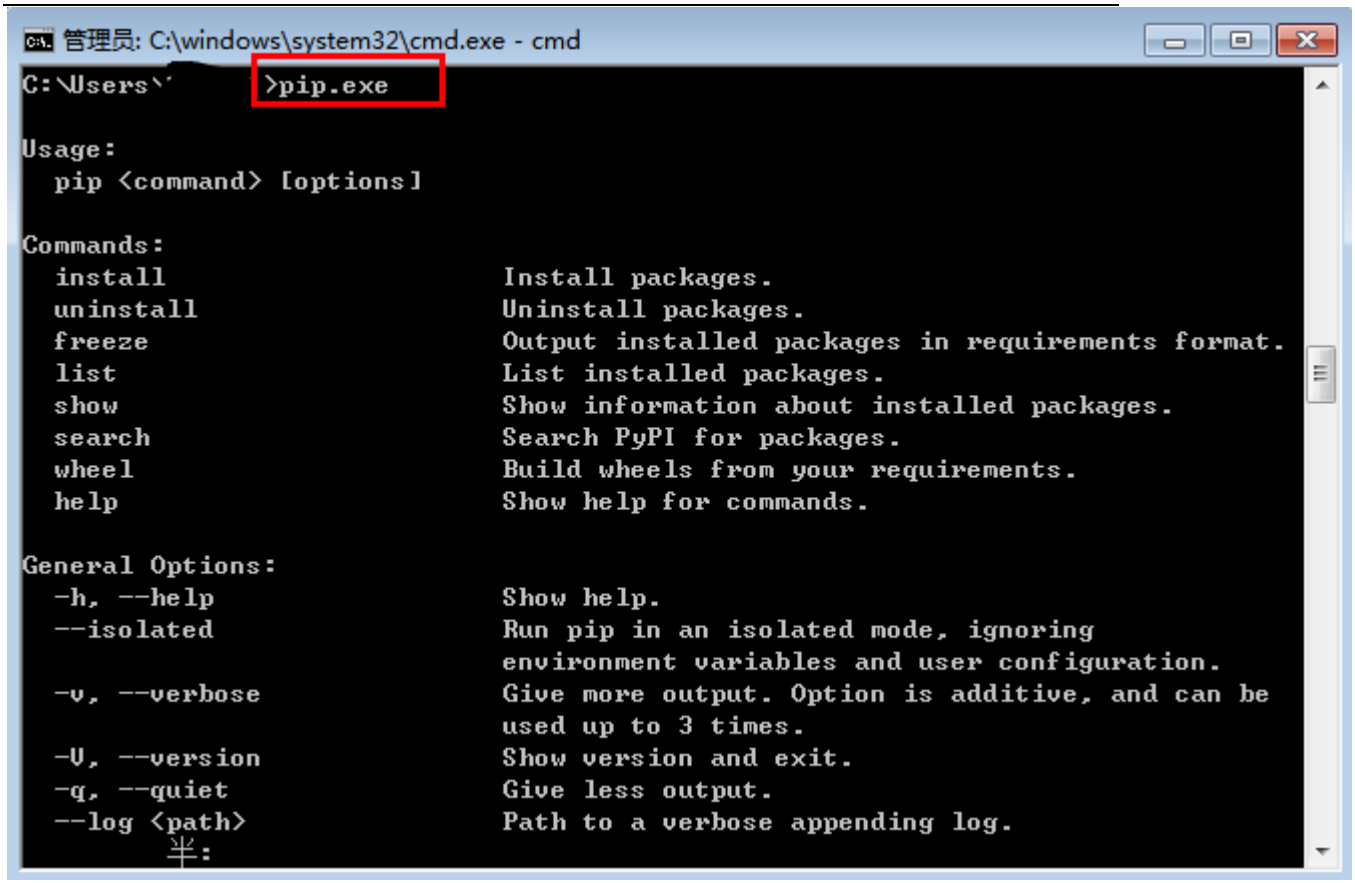
Did not provide a command?这是什么鬼?

正常情况应该是酱紫



1.3.2 解决办法

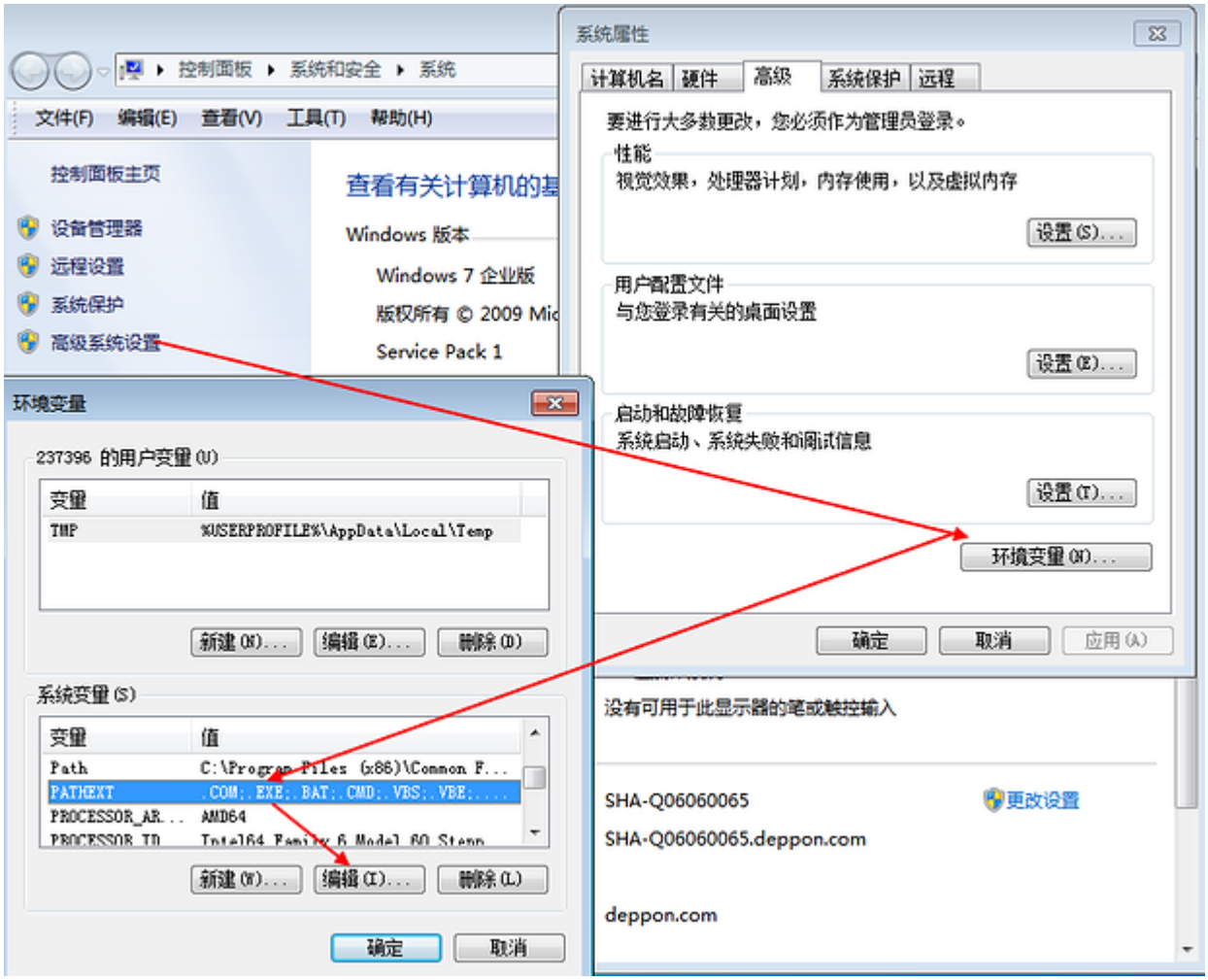
1.pip 是一个.exe 的可执行文件,在 cmd 输入 pip.exe 就可以解决了



2.所以在后面的安装指令中都需要带上后缀，那么问题来了，为什么会出现这种情况，如何彻底解决？

1.3.3 配置环境变量

1.主要原因是环境变量的 PATHEXT 里面缺少.EXE 的文件名
2.在 PATHEXT 下编辑后面加上;.EXE(注意分号是英文的)



1.4 Chrome 浏览器（chromedriver）

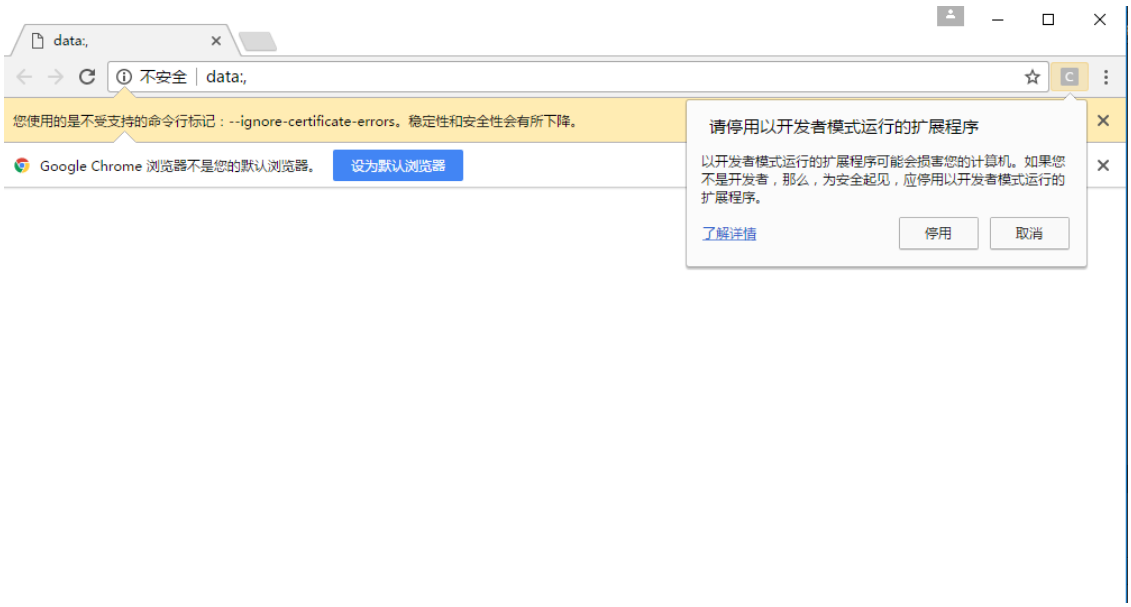
前言

selenium2 启动 Chrome 浏览器是需要安装驱动包的，但是不同的 Chrome 浏览器版本号，对应的驱动文件版本号又不一样，如果版本号不匹配，是没法启动起来的。

1.4.1 Chrome 遇到问题

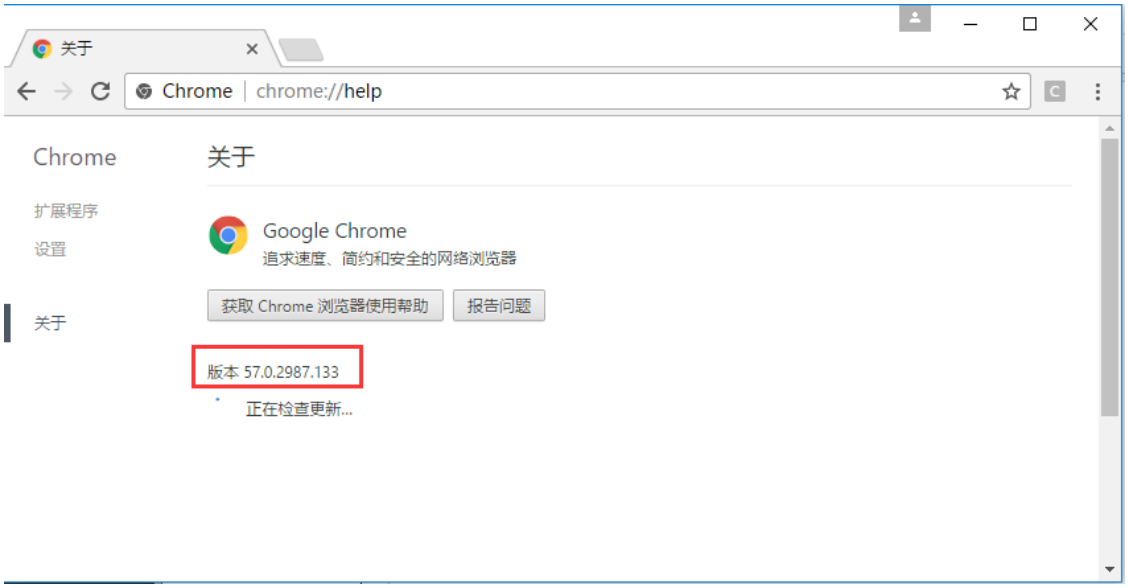
1. 如果在启动 chrome 浏览器时候，出现如下界面，无法打开网址，那么首先恭喜你，踩到了坑，接下来的内容或许对你有所帮助

```
>># coding:utf-8
>>from selenium import webdriver

>>driver = webdriver.Chrome()
>>driver.get("http://www.cnblogs.com/yoyoketang/")
```

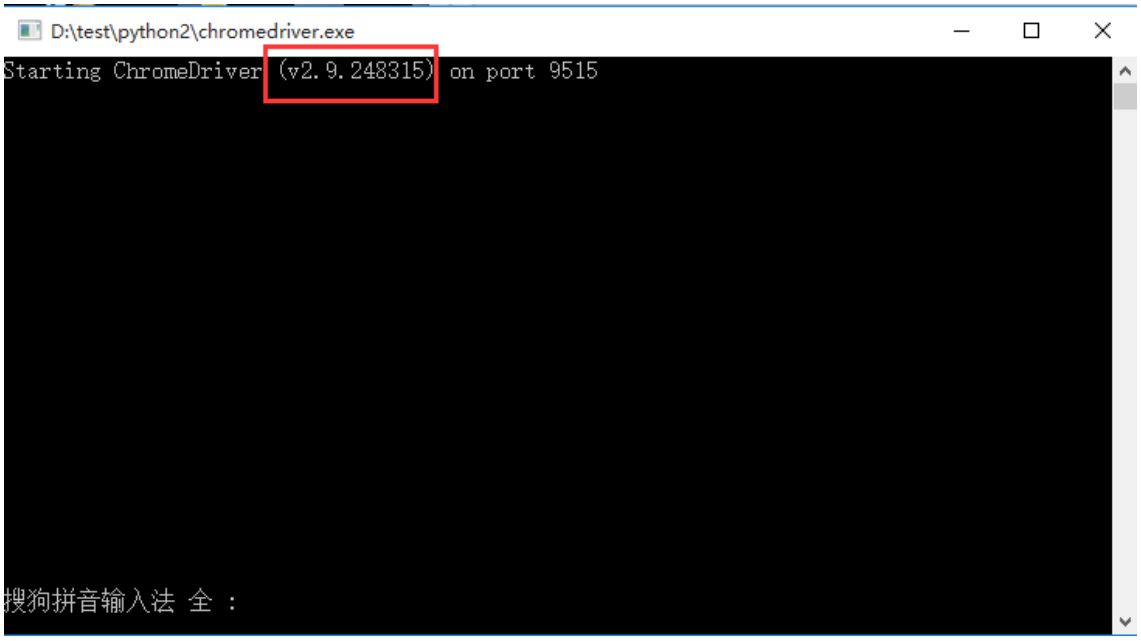


1.4.2 查看版本号

1.查看 Chrome 版本号，设置>关于，查出来版本号是 57.0



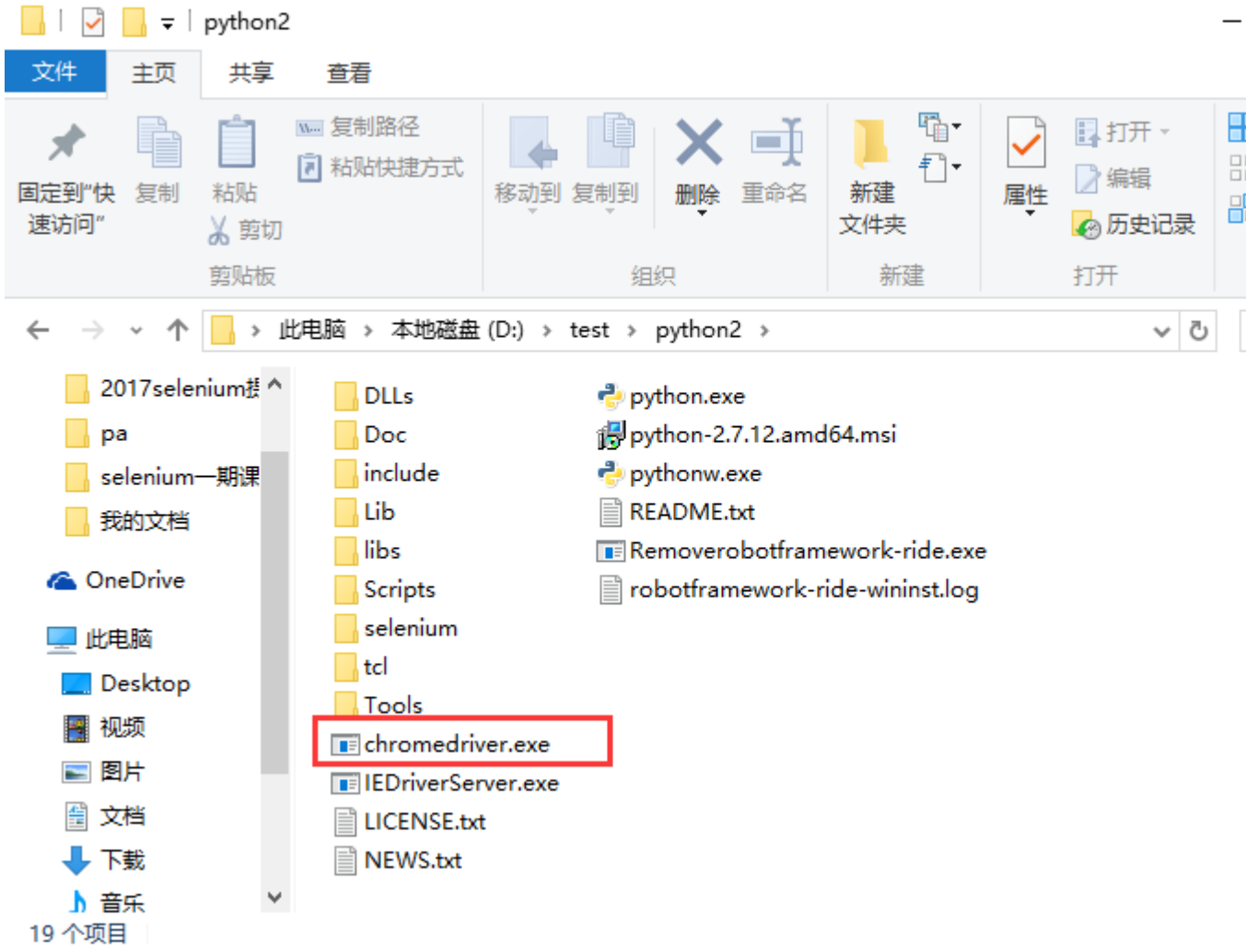
2. 查看 chromedriver.exe 版本号，双击这个文件就可以了，查出来版本号是 V2.9



3. 很显然是 chromedriver 的版本号过低了，于是可以找个更高级的版本：V2.24

1.4.3 chromedriver

1. 确保 chromedriver.exe 文件在 path 路径下，这里我放到 Python 的根目录了（python 根目录已配置到 path 了）



2.确保驱动文件名称是 chromedriver.exe，如果名称后面带版本号的，改下文件名称就行。

3.Chrome 版本 V57.0+chromedriverv2.24

1.4.4 各版本匹配表

chromedriver 版本	支持的 Chrome 版本
v2.24	v52-57
v2.23	v51-53
v2.22	v49-52
v2.21	v46-50
v2.20	v43-48
v2.19	v43-47
v2.18	v43-46

v2.17	v42-43
v2.13	v42-45
v2.15	v40-43
v2.14	v39-42
v2.13	v38-41
v2.12	v36-40
v2.11	v36-40
v2.10	v33-36
v2.9	v31-34
v2.8	v30-33
v2.7	v30-33
v2.6	v29-32
v2.5	v29-32
v2.4	v29-32

chromedriver 版本下载大全：
http://chromedriver.storage.googleapis.com/index.html

1.5 Pycharm 使用

前言

在写脚本之前，先要找个顺手的写脚本工具。python 是一门解释性编程语言，所以一般把写 python 的工具叫解释器。写 python 脚本的工具很多，小编这里就不一一列举的，只要自己用着顺手就可以的，如果你还没有选好解释器，小编这里推荐 pycharm。

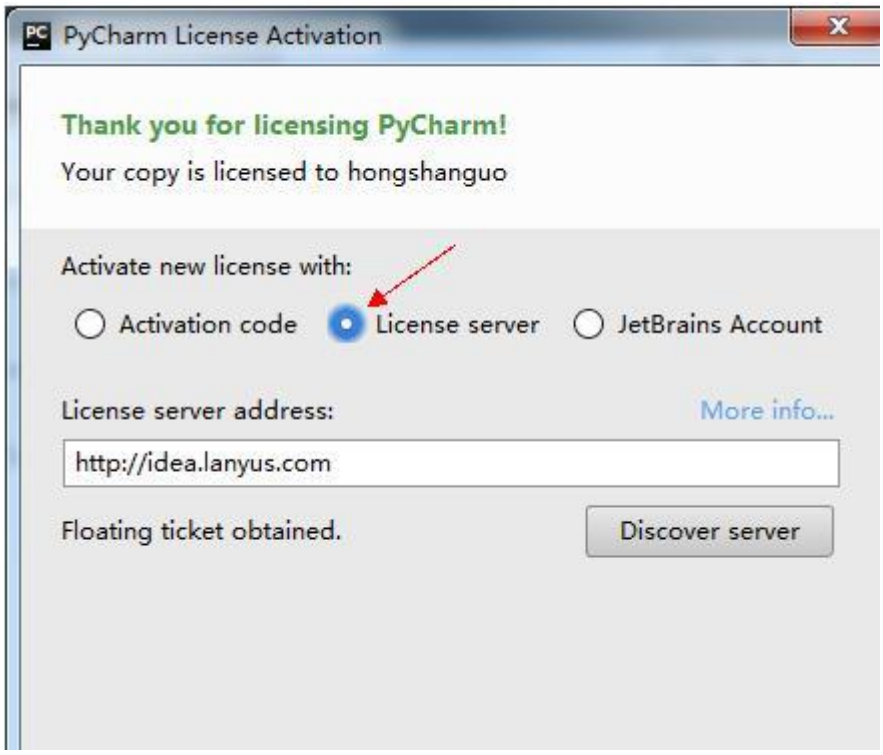
在安装 pycharm 后，有一些小伙伴不会破解，这里小编还是推荐大家买正版的。当然，如果你不想付费，想破解 pycharm，也是很容易的事情，这里小 编列 举几种破解办法。前提是你要先下载 pycharm 安装包，安装包可以去官网 http://www.jetbrains.com/pycharm/下载最新版。

1.5.1 pycharm 安装

方法一：

1. 在注册界面，选择 License serve。填入 http://idea.lanyus.com/71

2. 点击 ok



方法二：

1.注册界面选择：Activation coede

2.打开网址：http://idea.lanyus.com/71，点击“获取注册码”按钮

3.复制弹出框的注册码

4.copy 到注册界面 Activation coede 位置

方法三：

1. 安装 pycharm 在注册界面先别动
2. 调整电脑系统时间到 2036 年(20 年应该够用了)。
3. 注册界面选择申请 30 天试用
4. 退出 pycharm
5. 电脑时间调整回来。

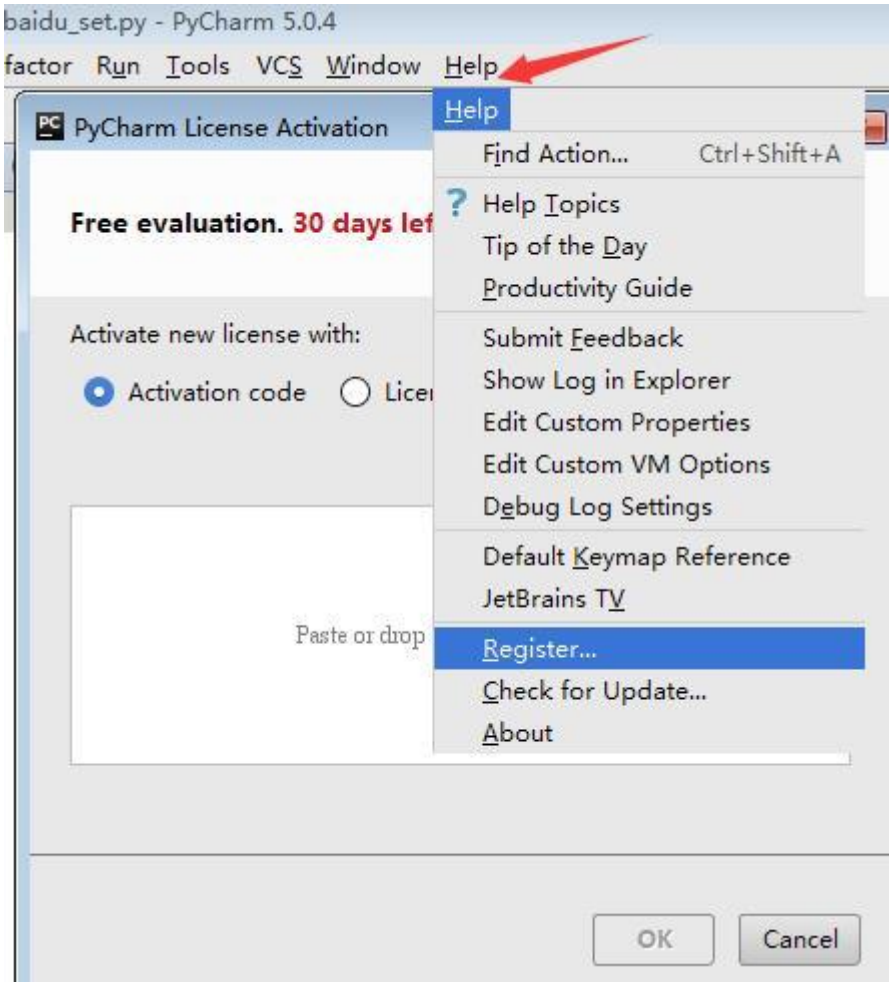
方法四：

1. 安装 pycharm 在注册界面，选择使用 30 天

2. 打开 pycharm 菜单里 Help>Register

3. 打开网址：http://idea.lanyus.com/71，点击“获取注册码”按钮

3. copy 到注册界面 Activation coede 位置



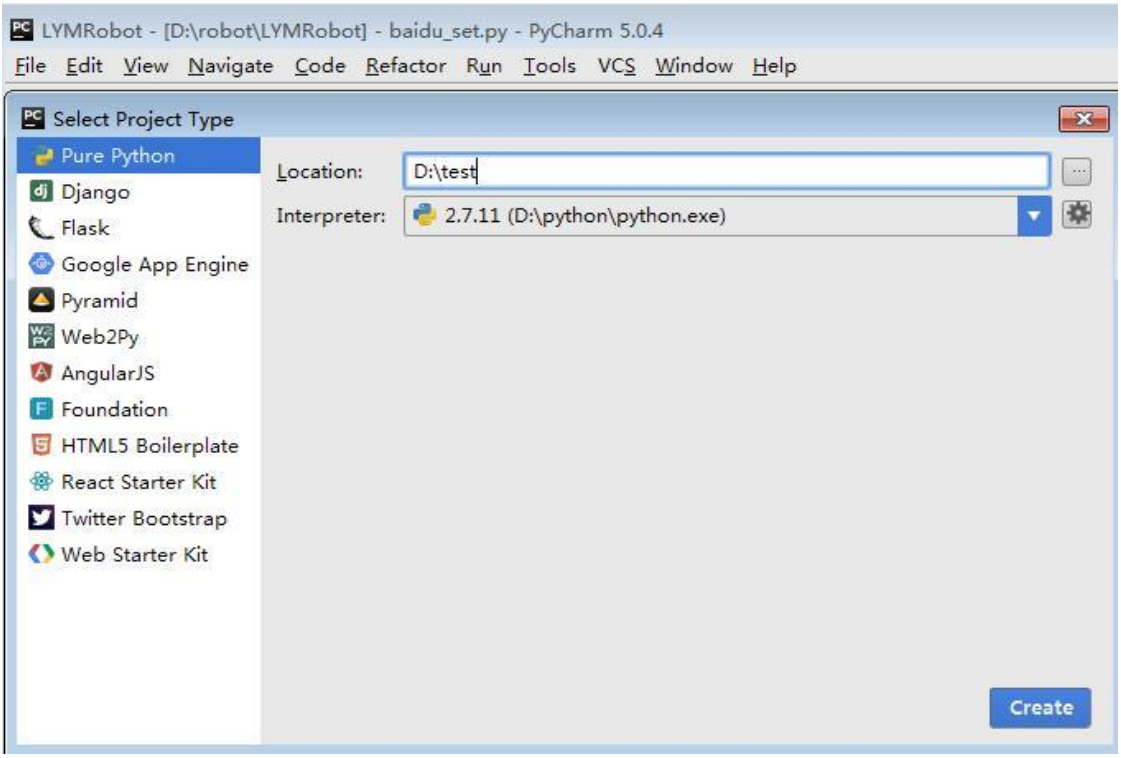
接下来开始 pycharm 之旅吧~

1.5.2 新建工程

1. 在 d 盘新建一个 test 文件夹

2. 打开 pycharm 左上角 File 按钮

3. 点 New Project 新建一个工程

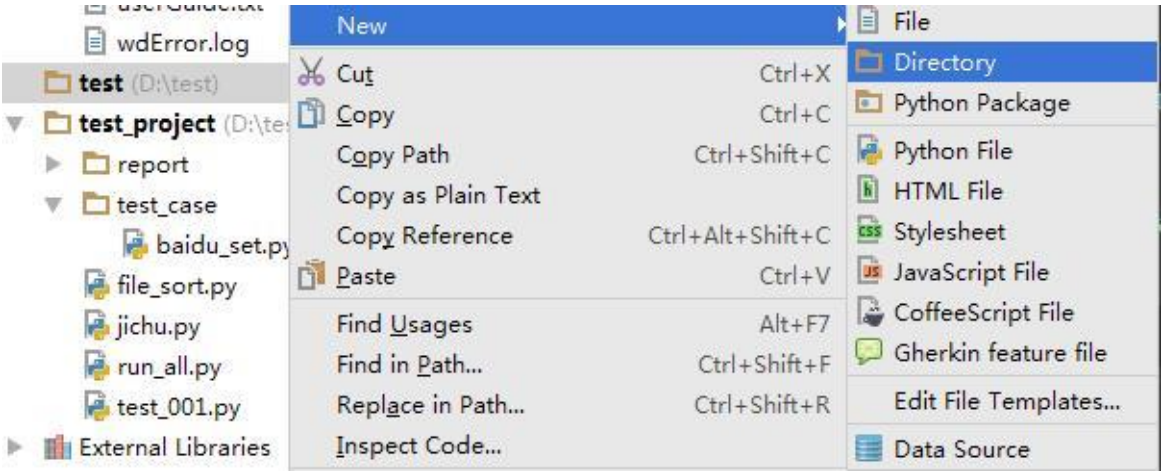


1.5.3 新建脚本

1. 在 pycharm 左侧菜单树右键，新建一个 Directory(文件夹)。

2. 选择对应文件夹，在文件夹中新建 Python File(脚本文件)。

3. 脚本名称自己命名，后缀.py 会自动带出

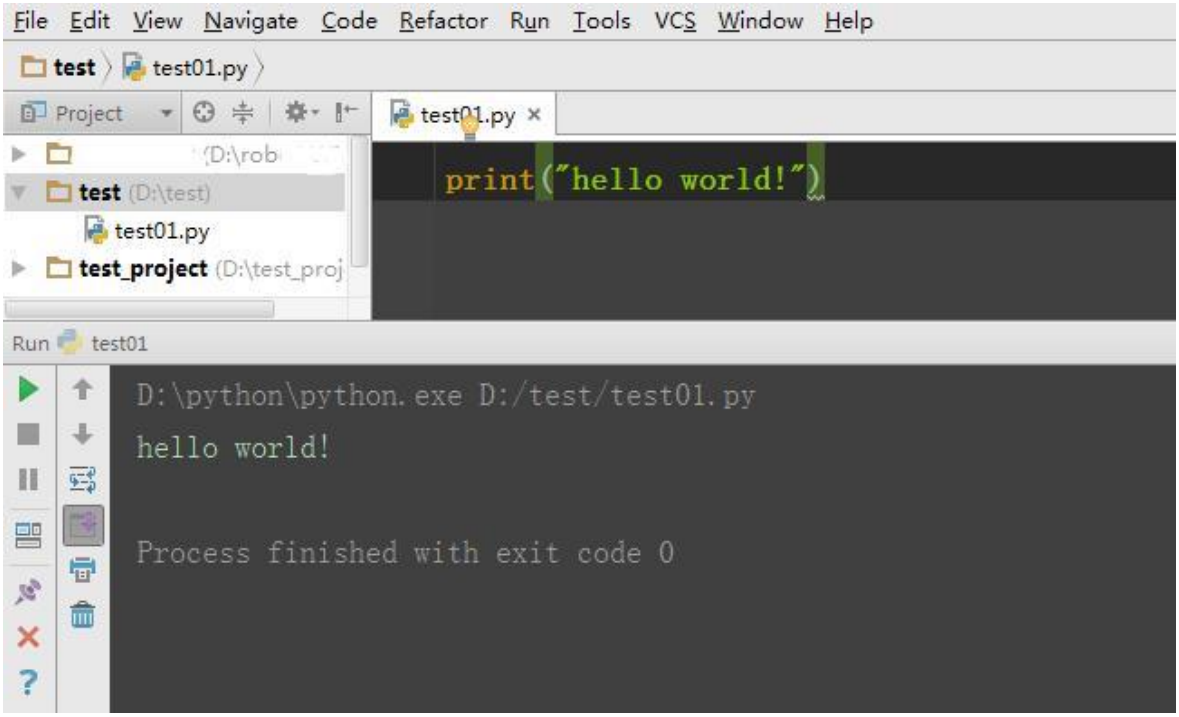


1.5.4 开始编程

1. 双击打开需要编写的脚本

2. 在右侧编辑框输入：print("hello world!")

3. 点脚本的 title，右击后选择 Run“test01”,运行结果如下



第 2 章 webdriver

2.1 操作浏览器基本方法

前言
前面已经把环境搭建好了，这从这篇开始，正式学习 selenium 的 webdriver 框架。我们平常说的 selenium 自动化，其实它并不是类似于 QTP 之类的有 GUI 界面的可视化工具，我们要学的是 webdriver 框架的 API。

本篇主要讲如何用 Python 调用 webdriver 框架的 API，对浏览器做一些常规的操作，如打开、前进、后退、刷新、设置窗口大小、截屏、退出等操作。

2.1.1 打开网页

1.第一步：从 selenium 里面导入 webdriver 模块
2.打开 Firefox 浏览器（Ie 和 Chrome 对应下面的）
3.打开百度网址

```
# coding:utf-8
# 第一步导入webdriver模块
from selenium import webdriver
# 第二步打开浏览器
driver = webdriver.Firefox()
# driver = webdriver.Ie() # Ie浏览器用这个
# driver = webdriver.Chrome() #谷歌浏览器用这个
# 第三部打开百度
driver.get("https://www.baidu.com")
```

2.1.2 设置休眠

1.由于打开百度网址后，页面加载需要几秒钟，所以最好等到页面加载完成后再继续下一步操作

2.导入 time 模块，time 模块是 Python 自带的，所以无需下载

3.设置等待时间，单位是秒（s）,时间值可以是小数也可以是整数

```
# coding:utf-8
from selenium import webdriver
# 导入time模块
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 设置休眠时间3秒,也可以是小数，单位是秒
time.sleep(3)
```

2.1.3 页面刷新

1.有时候页面操作后，数据可能没及时同步，需要重新刷新
2.这里可以模拟刷新页面操作，相当于浏览器输入框后面的刷新按钮

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
time.sleep(3)
# 等待3秒后刷新页面
driver.refresh()
```

2.1.4 页面切换

1.当在一个浏览器打开两个页面后，想返回上一页面，相当于浏览器左上角的左箭头按钮
2.返回到上一页面后，也可以切换到下一页，相当于浏览器左上角的右箭头按钮

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
time.sleep(3)
driver.get("http://www.hordehome.com")
time.sleep(5)
# 返回上一页
driver.back()
time.sleep(3)
# 切换到下一页
driver.forward()
```

2.1.5 设置窗口大小

1.可以设置浏览器窗口大小，如设置窗口大小为手机分辨率 540*960

2.也可以最大化窗口

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
time.sleep(3)
# 设置窗口大小为540*960
driver.set_window_size(540,960)
time.sleep(2)
# 将浏览器窗口最大化
driver.maximize_window()
```

2.1.6 截屏

1. 打开网站之后，也可以对屏幕截屏

2.截屏后设置制定的保存路径+文件名称+后缀

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
time.sleep(3)
driver.get_screenshot_as_file("D:\\test\\b1.jpg")
```

2.1.7 退出

1.退出有两种方式，一种是 close;另外一种是 quit

2.close 用于关闭当前窗口，当打开的窗口较多时，就可以用 close 关闭部分窗口

3.quit 用于结束进程，关闭所有的窗口

4.最后结束测试，要用 quit。quit 可以回收 c 盘的临时文件

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
time.sleep(3)
# driver.close() 关闭当前窗口
# quit用于退出浏览器进程
driver.quit()
```

掌握了浏览器的基本操作后，接下来就可以开始学习元素定位了，元素定位需要有一定的 html 基础。没有基础的可以按下浏览器的 F12 快捷键先看下 html 的布局，先了解一些就可以了。

2.2 常用 8 种元素定位（Firebug 和 firepath）

前言

元素定位在 firefox 上可以安装 Firebug 和 firepath 辅助工具进行元素定位。

2.2.1 环境准备

1.浏览器选择：Firefox

2.安装插件：Firebug 和 FirePath(设置》附加组件》搜索：输入插件名称》下载安装后重启浏览器)

3.安装完成后，页面右上角有个小爬虫图标

4.快速查看 xpath 插件：XPath Checker 这个可下载，也可以不用下载

5.插件安装完成后，点开附加组件》扩展，如下图所示

2.2.2 查看页面元素：

以百度搜索框为例，先打开百度网页

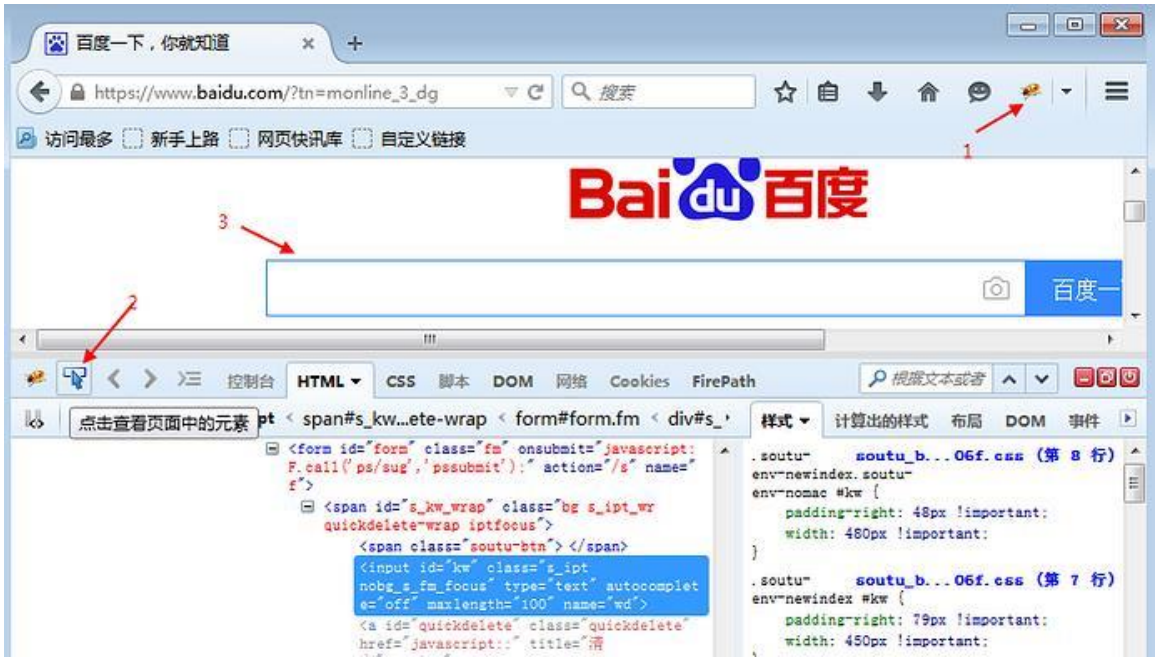
1. 点右上角爬虫按钮

2. 点左下角箭头

3. 讲箭头移动到百度搜索输入框上，输入框高亮状态

4. 下方红色区域就是单位到输入框的属性：

<input id="kw" class="s_ipt" type="text" autocomplete="off" maxlength="100" name="wd">



2.2.3 find_element_by_id()

1.从上面定位到的元素属性中，可以看到有个 id 属性：id="search-key"，这里可以通过它的 id 属性单位到这个元素。

2.定位到搜索框后，用 send_keys()方法

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过id定位百度搜索框，并输入“python”
driver.find_element_by_id("kw").send_keys("python")
```

2.2.4 find_element_by_name()

1.从上面定位到的元素属性中，可以看到有个 name 属性：name="wd"，这里可以通过它的 name 属性单位到这个元素。

说明：这里运行后会报错，说明这个搜索框的 name 属性不是唯一的，无法通过 name 属性直接定位到输入框

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过name定位百度搜索框，并输入“python”
driver.find_element_by_name("wd").send_keys("python")
```

2.2.5 find_element_by_class_name()

1.从上面定位到的元素属性中，可以看到有个 class 属性：class="s_ipt"，这里可以通过它的 class 属性单位到这个元素。

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过class定位百度搜索框，并输入“python”
driver.find_element_by_class_name("s_ipt").send_keys("python")
```

2.2.6 find_element_by_tag_name()

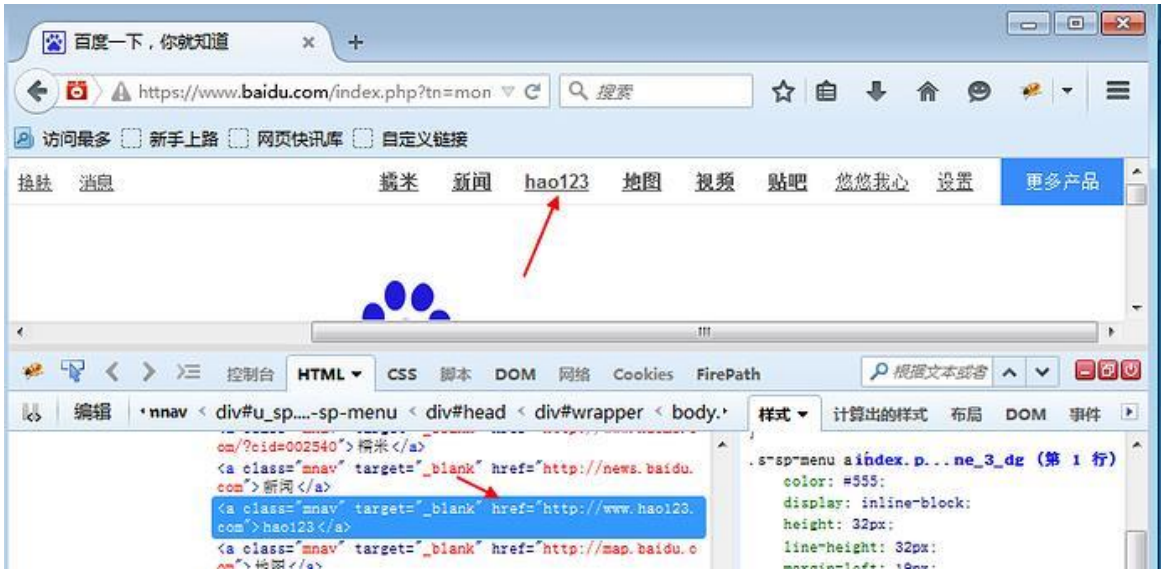
1.从上面定位到的元素属性中，可以看到每个元素都有 tag（标签）属性，如搜索框的标签属性，就是最前面的 input

2.很明显，在一个页面中，相同的标签有很多，所以一般不用标签来定位。以下例子，仅供参考和理解，运行肯定报错

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过tag(标签)定位百度搜索框，并输入“python”
driver.find_element_by_tag_name("input").send_keys("python")
```

2.2.7 find_element_by_link_text()

1.定位百度页面上"hao123"这个按钮



查看页面元素：

hao123

2.从元素属性可以分析出，有个 href = "http://www.hao123.com

说明它是个超链接，对于这种元素，可以用以下方法

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过link(超链接)属性定位到hao123按钮，并点击
driver.find_element_by_link_text("hao123").click()
```

2.2.8 find_element_by_partial_link_text()

1.有时候一个超链接它的字符串可能比较长，如果输入全称的话，会显示很长，这时候可以用一模糊匹配方式，截取其中一部分字符串就可以了

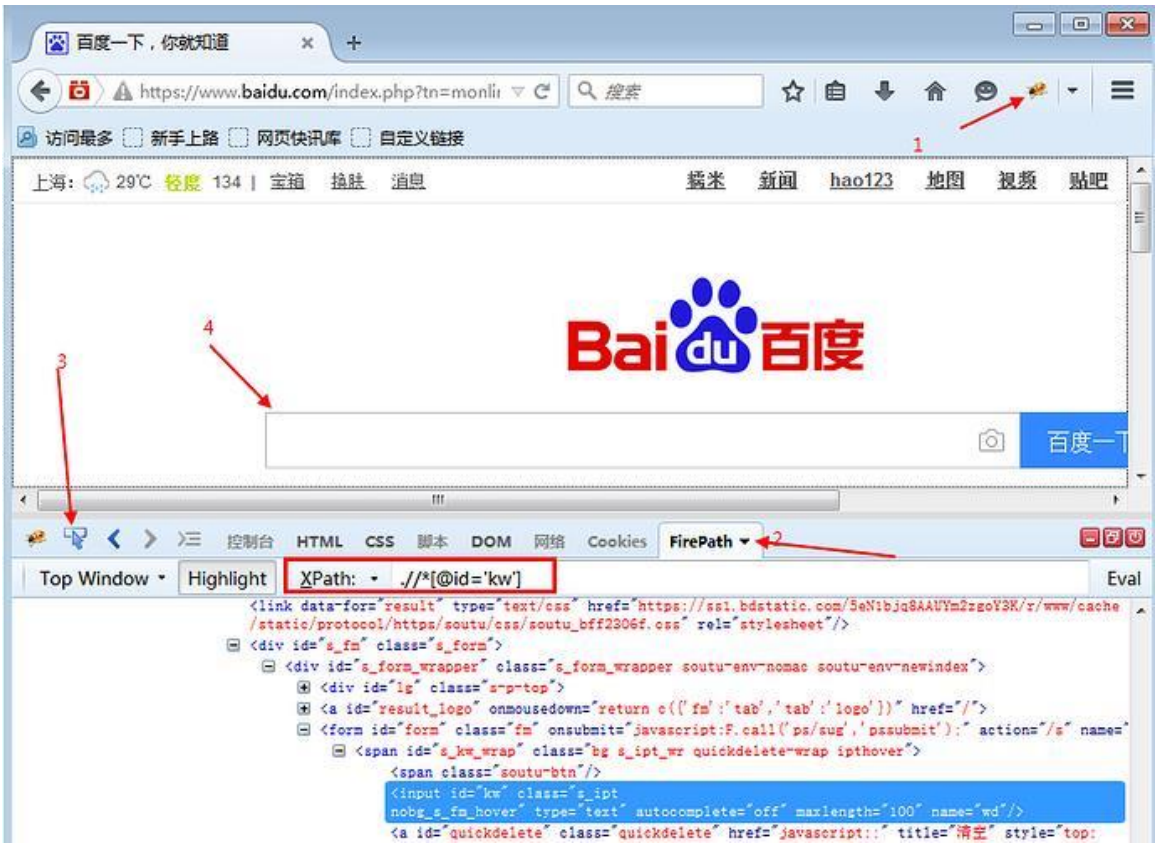
2.如“hao123”,只需输入“ao123”也可以定位到

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# partial_link是一种模糊匹配的方式，对于超长字符串截取其中一部分
driver.find_element_by_partial_link_text("ao123").click()
```

2.2.9 find_element_by_xpath()

1.以上定位方式都是通过元素的某个属性来定位的，如果一个元素它既没有 id、name、class 属性也不是超链接，这么办呢？或者说它的属性很多重复的。这个时候就可以用 xpath 解决

2.xpath 是一种路径语言，跟上面的定位原理不太一样，首先第一步要先学会用工具查看一个元素的 xpath



3.安装上图的步骤，在 FriePath 插件里 copy 对应的 xpath 地址

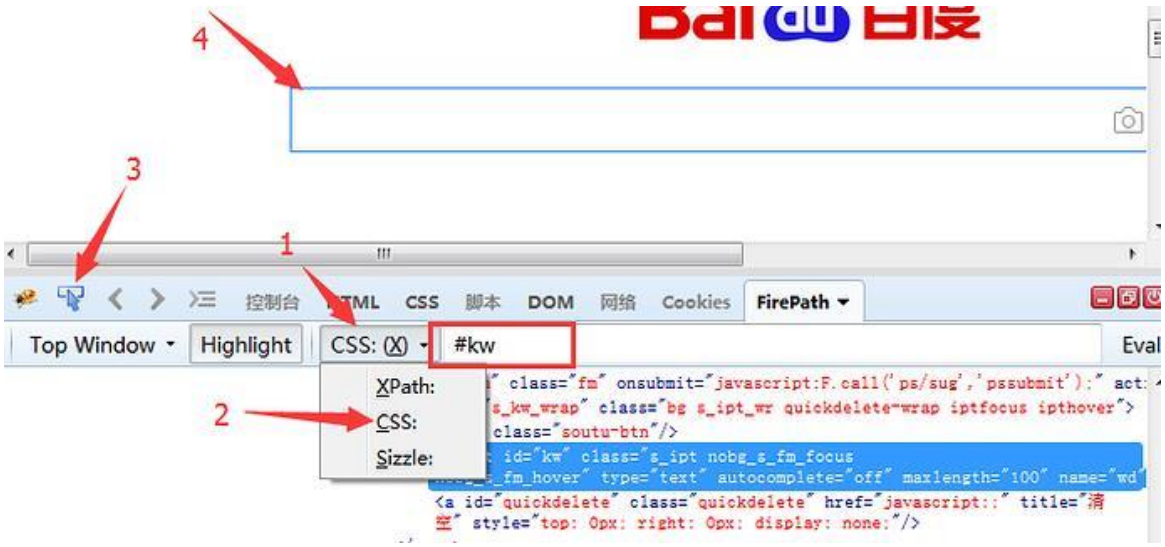
```python
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 在FirePath里copy出xpath地址
driver.find_element_by_xpath(".//*[@id='kw']").send_keys("python")
```

2.2.10 find_element_by_css_selector()

1.css 是另外一种语法，比 xpath 更为简洁，但是不太好理解。这里先学会如何用工具查看，后续的教程再深入讲解

2.打开 FirePath 插件选择 css

3.定位到后如下图红色区域显示



总结：

selenium 的 webdriver 提供了八种基本的元素定位方法，前面六种是通过元素的属性来直接定位的，后面的 xpath 和 css 定位更加灵活，需要重点掌握其中一个。

1.通过 id 定位：find_element_by_id()

2. 通过 name 定位：find_element_by_name()

3. 通过 class 定位：find_element_by_class_name()

4. 通过 tag 定位：find_element_by_tag_name()

5. 通过 link 定位：find_element_by_link_text()

6. 通过 partial_link 定位：find_element_by_partial_link_text()

7. 通过 xpath 定位：find_element_by_xpath()

8. 通过 css 定位：find_element_by_css_selector()

2.3 xpath 定位

前言

在上一篇简单的介绍了用工具查看目标元素的 xpath 地址，工具查看比较死板，不够灵活，有时候直接复制粘贴会定位不到。这个时候就需要自己手动的去写 xpath 了，这一篇详细讲解 xpath 的一些语法。

什么是 xpath 呢？

官方介绍：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。反正小编看这个介绍是云里雾里的，通俗一点讲就是通过元素的路径来查找到这个元素的，相当于通过定位一个对象的坐标，来找到这个对象。

2.3.1 xpath:属性定位

1. xptah 也可以通过元素的 id、name、class 这些属性定位，如下图

2.于是可以用以下 xpath 方法定位

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 用xpath通过id属性定位
driver.find_element_by_xpath("//*[@id='kw']").send_keys("python")
# 用xpath通过name属性定位
driver.find_element_by_xpath("//*[@name='wd']").send_keys("python")
# 用xpath通过class属性定位
driver.find_element_by_xpath("//*[@class='s_ipt']").send_keys("python")
```

2.3.2 xpath:其它属性

1.如果一个元素 id、name、class 属性都没有，这时候也可以通过其它属性定位到

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 用xpath通过其它属性定位
driver.find_element_by_xpath("//*[@autocomplete='off']").send_keys("python")
```

2.3.3 xpath:标签

1.有时候同一个属性，同名的比较多，这时候可以通过标签筛选下，定位更准一点

2.如果不想制定标签名称，可以用*号表示任意标签

3.如果想制定具体某个标签，就可以直接写标签名称

```python
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 用xpath通过其它属性定位
driver.find_element_by_xpath("//input[@autocomplete='off']").send_keys("python")
# 用xpath通过id属性定位
driver.find_element_by_xpath("//input[@id='kw']").send_keys("python")
# 用xpath通过name属性定位
driver.find_element_by_xpath("//input[@name='wd']").send_keys("python")
```

2.3.4 xpath:层级

1.如果一个元素，它的属性不是很明显，无法直接定位到，这时候我们可以先找它老爸（父元素）

2.找到它老爸后，再找下个层级就能定位到了

3.如上图所示，要定位的是 input 这个标签，它的老爸的 id=s_kw_wrap.

4.要是它老爸的属性也不是很明显，就找它爷爷 id=form

5.于是就可以通过层级关系定位到

```python
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# 通过定位它老爸来定位input输入框
driver.find_element_by_xpath("//span[@id='s_kw_wrap']/input").send_keys("python")
# 通过定位它爷爷来定位input输入框
driver.find_element_by_xpath("//form[@id='form]/span/input").send_keys("python")
```

2.3.5 xpath:索引

1.如果一个元素它的兄弟元素跟它的标签一样，这时候无法通过层级定位到。因为都是一个父亲生的，多胞胎兄弟。

2.虽然双胞胎兄弟很难识别，但是出生是有先后的，于是可以通过它在家里的排行老几定位到。

3.如下图三胞胎兄弟

```
<tr>
<tr>
    <th>搜索结果显示条数：</th>
    <td>设定您希望搜索结果显示的条数</td>
    <td id="se-setting-3">
        <select id="nr" name="NR">
            <option selected="" value="10">每页显示10条</option>
            <option value="20">每页显示20条</option>
            <option value="50">每页显示50条</option>
        </select>
        百度的原始设定10条最有效且快速
    </td>
```

4.用 xpath 定位老大、老二和老三（这里索引是从 1 开始算起的，跟 Python 的索引不一样）

```
# 用xpath定位老大
driver.find_element_by_xpath("//select[@id='nr']/option[1]").click()
# 用xpath定位老二
driver.find_element_by_xpath("//select[@id='nr']/option[2]").click()
# 用xpath定位老三
driver.find_element_by_xpath("//select[@id='nr']/option[3]").click()
```

2.3.6 xpath:逻辑运算

1.xpath 还有一个比较强的功能，是可以多个属性逻辑运算的，可以支持与（and）、或（or）、非（not）

2.一般用的比较多的是 and 运算，同时满足两个属性

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# xpath逻辑运算
driver.find_element_by_xpath("//*[@id='kw' and @autocomplete='off']")
```

2.3.7 xpath:模糊匹配

1.xpath 还有一个非常强大的功能，模糊匹配

2.掌握了模糊匹配功能，基本上没有到位不到的

3.比如我要定位百度页面的超链接“hao123”，在上一篇中讲过可以通过 by_link，也可以通过 by_partial_link，模糊匹配定位到。当然 xpath 也可以有同样的功能，并且更为强大。

```python
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# xpath模糊匹配功能
driver.find_element_by_xpath("//*[contains(text(),'hao123')]").click()
# xpath也可以模糊匹配某个属性
driver.find_element_by_xpath("//*[contains(@id,'kw']").click()
# xpath可以模糊匹配以什么开头
driver.find_element_by_xpath("//*[starts-with(@id,'s_kw_')]").click()
# xpath可以模糊匹配以什么结尾
driver.find_element_by_xpath("//*[ends-with(@id,'kw_wrap')]").click()
# xpath还支持最强的正则表达式
driver.find_element_by_xpath("//*[matchs(text(),'hao13')]").click()
```

可以把 xpath 看成是元素定位界的屠龙刀。武林至尊，宝刀 xpath，css 不出，谁与争锋？下节课将亮出倚天剑 css 定位。Selenium2+python 自动化 6

2.4 CSS 定位语法

前言

大部分人在使用 selenium 定位元素时，用的是 xpath 定位，因为 xpath 基本能解决定位的需求。css 定位往往被忽略掉了，其实 css 定位也有它的价值，css 定位更快，语法更简洁。

这一篇 css 的定位方法，主要是对比上一篇的 xpath 来的，基本上 xpath 能完成的，css 也可以做到。两篇对比学习，更容易理解。

2.4.1 css:属性定位

1.css 可以通过元素的 id、class、标签这三个常规属性直接定位到

2.如下是百度输入框的的 html 代码：

<input id="kw" class="s_ipt" type="text" autocomplete="off" maxlength="100" name="wd"/>

3.css 用#号表示 id 属性,如：#kw

4.css 用.表示 class 属性，如：.s_ipt

5.css 直接用标签名称，无任何标示符，如：input

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# css通过id属性定位
driver.find_element_by_css_selector("#kw").send_keys("python")
# css 通过class属性定位
driver.find_element_by_css_selector(".s_ipt").send_keys("python")
# css通过标签属性定位，这里运行会报错，主要是了解这个写法
driver.find_element_by_css_selector("input").send_keys("python")
```

2.4.2 css:其它属性

1.css 除了可以通过标签、class、id 这三个常规属性定位外，也可以通过其它属性定位

2.以下是定位其它属性的格式

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# css通过name属性定位
driver.find_element_by_css_selector("[name='wd']").send_keys("python")
# css通过autocomplete属性定位
driver.find_element_by_css_selector("[autocomplete='of']").send_keys("python")
# css通过type属性定位
driver.find_element_by_css_selector("[type='text']").send_keys("python")
```

2.4.3 css:标签

1. css 页可以通过标签与属性的组合来定位元素

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.find_element_by_css_selector("input:contains('kw')")
# css通过标签与class属性的组合定位
driver.find_element_by_css_selector("input.s_ipt").send_keys("python")
# css通过标签与id属性的组合定位
driver.find_element_by_css_selector("input#kw").send_keys("python")
# css通过标签与其它属性组合定位
driver.find_element_by_css_selector("input[id='kw']").send_keys("python")
```

2.4.4 css:层级关系

1. 在前面一篇 xpath 中讲到层级关系定位，这里 css 也可以达到同样的效果

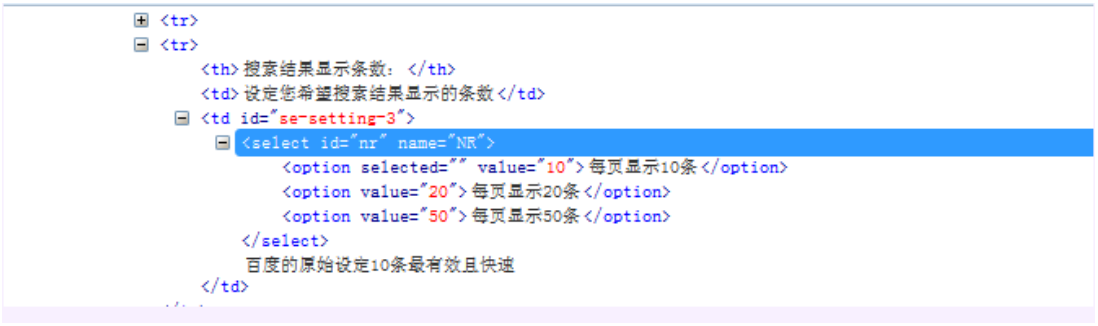
2. 如 xpath：//form[@id='form']/span/input 和

//form[@class='fm']/span/input 也可以用 css 实现

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
# css通过层级关系定位
driver.find_element_by_css_selector("form#form>span>input").send_keys("python")
# css通过层级关系定位
driver.find_element_by_css_selector("form.fm>span>input").send_keys("python")
```

2.4.5 css:索引

1. 以下图为例，跟上一篇一样



2.css 也可以通过索引 option：nth-child(1)来定位子元素，这点与 xpath 写法用很大差异，其实很好理解，直接翻译过来就是第几个小孩

```
# 选择第1个option
driver.find_element_by_css_selector("select#nr>option:nth-child(1)").click()
# 选择第2个option
driver.find_element_by_css_selector("select#nr>option:nth-child(2)").click()
# 选择第3个option
driver.find_element_by_css_selector("select#nr>option:nth-child(3)").click()
```

2.4.6 css:逻辑运算

1.css 同样也可以实现逻辑运算，同时匹配两个属性，这里跟 xpath 不一样，无需写 and 关键字

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.find_element_by_css_selector("input[id='kw'][name='wd']").send_keys("python")
```

2.4.7 css:模糊匹配

1.css 的模糊匹配 contains('xxx'),网上虽然用各种资料显示能用，但是小编亲自试验了下，一直报错。

2.在各种百度后找到了答案：you can't do this with CSS selectors, because there is no such thing as:contains() in CSS. It was a proposal that was abandoned years ago.

非常遗憾，这个语法已经被抛弃了，所以这里就不用管这个语法了。

css 语法远远不止上面提到的，还有更多更强大定位策略，有兴趣的可以继续深入研究。官方说法，css 定位更快，语法更简洁，但是 xpath 更直观，更好理解一些。

2.5 SeleniumBuilder 辅助定位元素

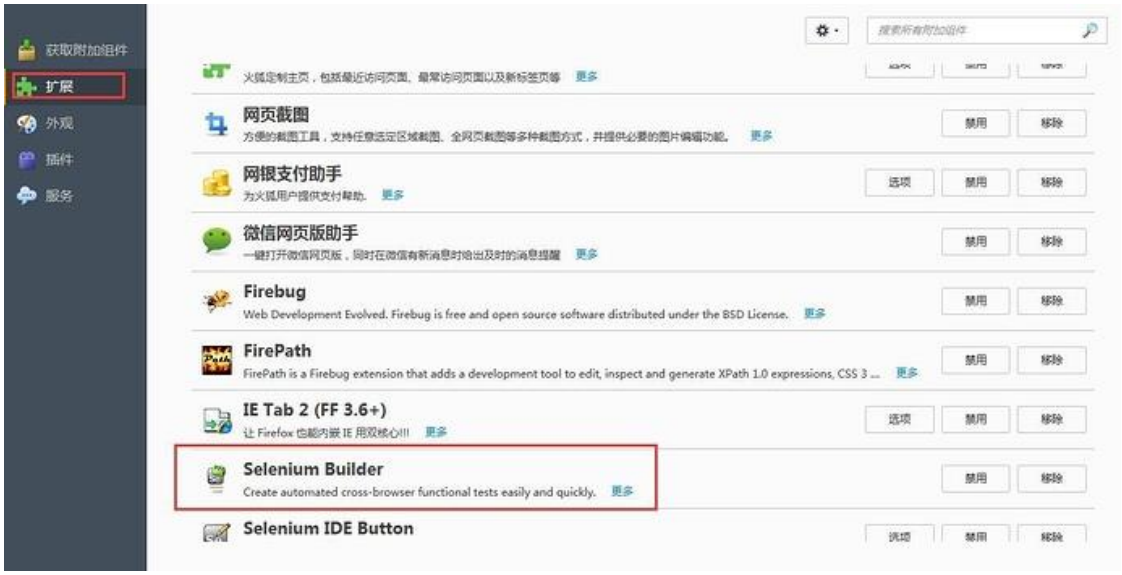
前言

对于用火狐浏览器的小伙伴们，你还在为定位元素而烦恼嘛？

上古神器 Selenium Builder 来啦，哪里不会点哪里，妈妈再也不用担心我的定位元素问题啦！（但是也不是万能，基本上都能覆盖到）

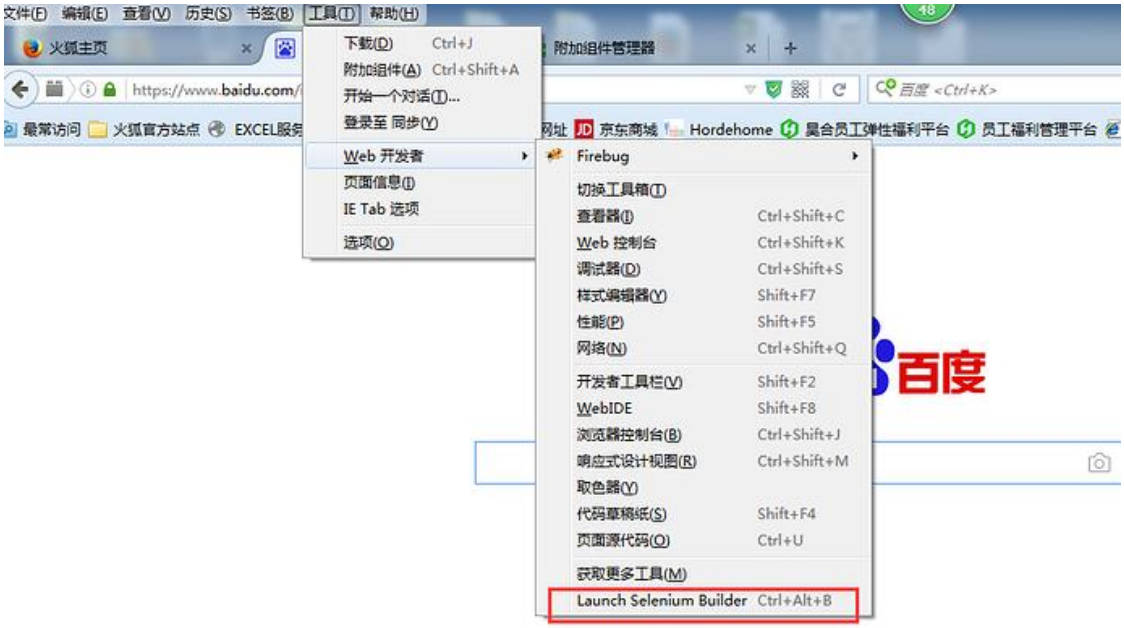
2.5.1 安装 Selenium Builder

在火狐浏览器的附加组件中搜索添加 Selenium Builder 即可。安装好后如下图所示：

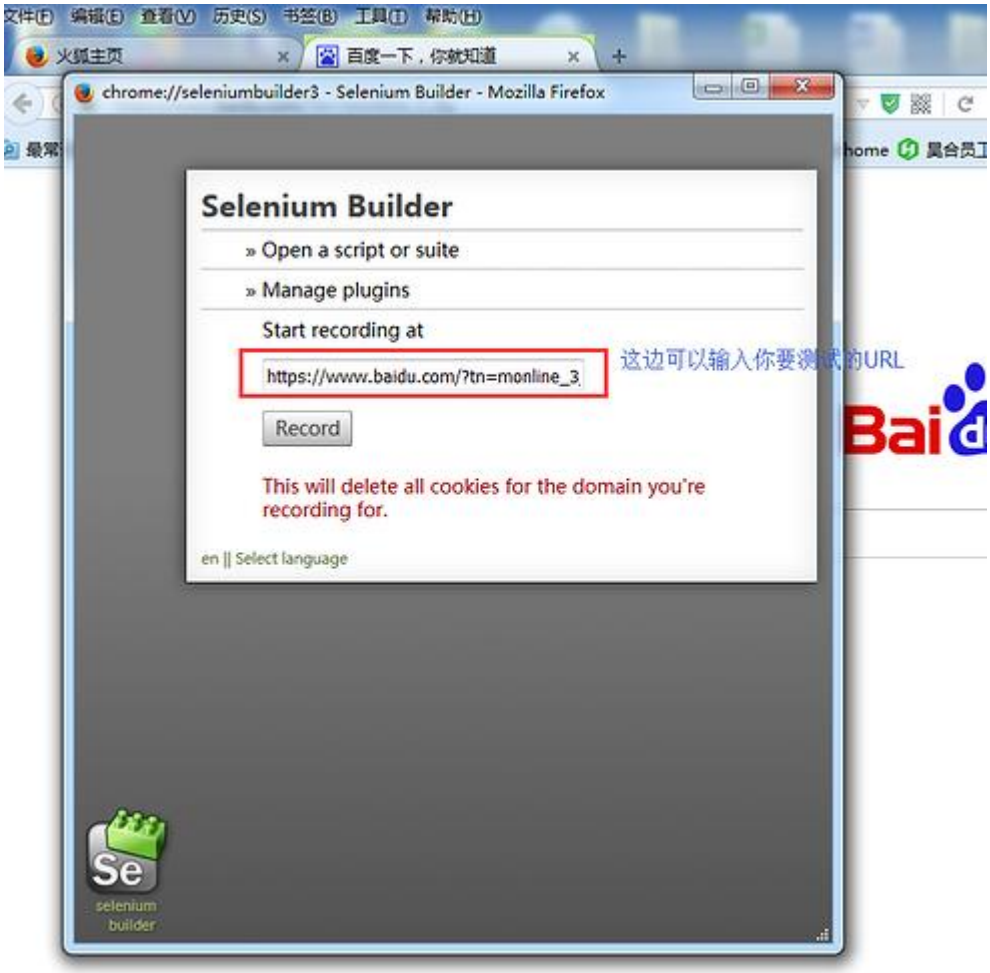


2.5.2 直接运用

1.打开你要测试的 URL 或者打开插件后输入你要测试的 URL，如下图

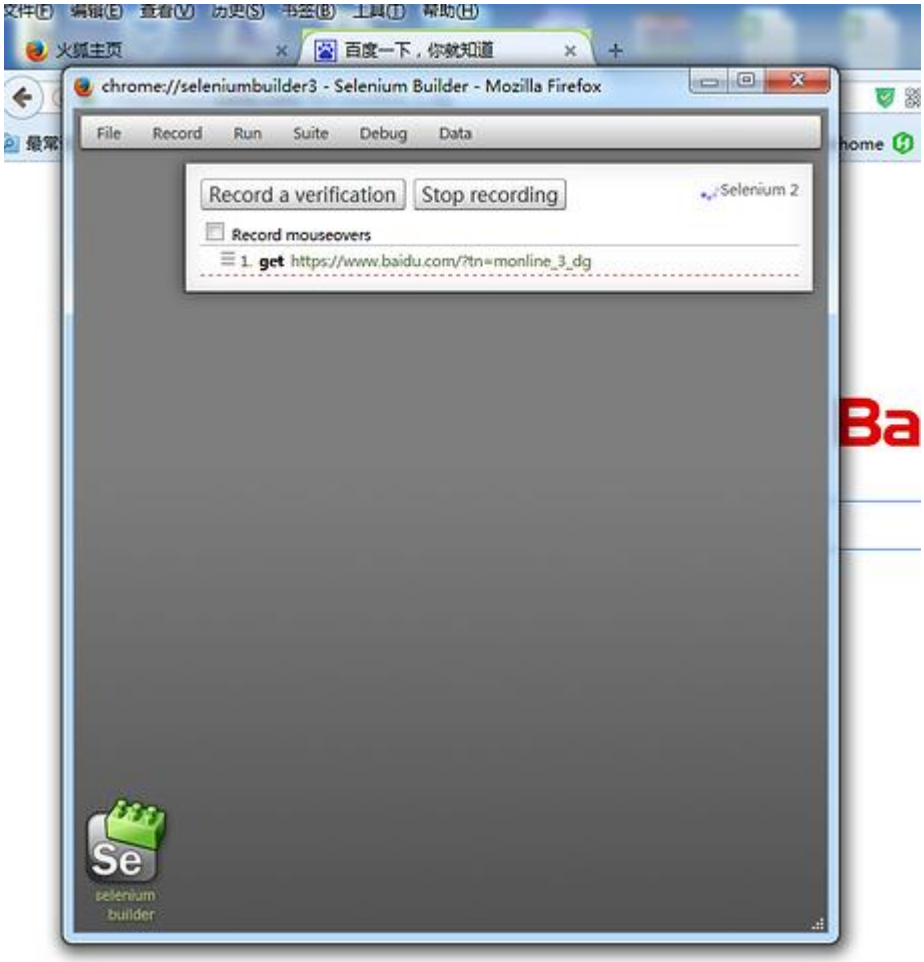


2.点击后弹出一个弹窗，如下图：



注：如果你是直接在你要测的网页页面打开这个插件时，selenium builder 会直接获取你要测的 URL

3. 点击 record：



然后你就可以哪里不会点哪里了。这里举个例子：

2.5.3 实践案例

1.百度首页，点击百度一下，然后点击登录，再一次点击账号和密码输入框，让我们来看看结果

2.这里没有展开，点击展开后可以发现定位该元素的多种方法

直接选择你想要的方法复制粘贴即可,不用的话直接关掉弹窗即可。

2.6 操作元素（键盘和鼠标事件）

前言

在前面的几篇中重点介绍了一些元素的到位方法，到位到元素后，接下来就是需要操作元素了。本篇总结了 web 页面常用的一些操作元素方法，可以统称为行为事件

有些 web 界面的选项菜单需要鼠标悬停在某个元素上才能显示出来（如百度页面的设置按钮）。

2.6.1 简单操作

1.点击（鼠标左键）页面按钮：click()

2.请空输入框：clear()

3.输入字符串：send_keys()

4.打开测试部落论坛后，点击放大镜搜索图标，一般为了保证输入的正确性，可以先清空下输入框，然后输入搜索关键字

s_hordehome.py ×

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("http://www.hordehome.com")
driver.implicitly_wait(10)
driver.find_element_by_id("search-button").click()
driver.find_element_by_id("search-term").clear()
driver.find_element_by_id("search-term").send_keys("selenium")
```

2.6.2 submit 提交表单

1.在前面百度搜索案例中，输入关键字后，可以直接按回车键搜索，也可以点搜索按钮搜索。

2.submit()一般用于模拟回车键

baidu_search.py ×

```
# coding:utf-8
from selenium import webdriver
import random
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.implicitly_wait(10)
driver.find_element_by_id("kw").send_keys(u"测试部落")
# submit()模拟enter键提交表单
driver.find_element_by_id("kw").submit()
```

3.但是论坛的搜索，如果用 submit 的话，会报错，可是又没有搜索点击按钮，怎么办呢？

2.6.3 键盘操作

1.selenium 提供了一整套的模拟键盘操作事件，前面 submit()方法如果不行的话，可以试试模拟键盘事件

2.模拟键盘的操作需要先导入键盘模块：from selenium.webdriver.common.keys import Keys

3.模拟 enter 键，可以用 send_keys(Keys.ENTER)

s_hordehome.py ×

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
driver = webdriver.Firefox()
driver.get("http://www.hordehome.com")
driver.implicitly_wait(10)
driver.find_element_by_id("search-button").click()
driver.find_element_by_id("search-term").clear()
driver.find_element_by_id("search-term").send_keys("selenium")
# 模拟enter键操作回车按钮
driver.find_element_by_id("search-term").send_keys(Keys.ENTER)
```

4.其它常见的键盘操作：

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)

这里只是列了一些常用的，当然除了键盘事件，也有鼠标事件

2.6.4 鼠标悬停事件

1.鼠标不仅仅可以点击(click),鼠标还有其它的操作，如：鼠标悬停在某个元素上，鼠标右击，鼠标按住某个按钮拖到

2.鼠标事件需要先导入模块：from selenium.webdriver.common.action_chains import ActionChains

perform() 执行所有 ActionChains 中的行为

move_to_element() 鼠标悬停

3.这里以百度页面设置按钮为例

baidu_set.py ×

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.implicitly_wait(10)
# 鼠标悬停在搜索设置按钮上
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
```

4.除了常用的鼠标悬停事件外，还有

右击鼠标：context_click()

双击鼠标：double_click()

依葫芦画瓢，替换上面案例中对应的鼠标事件就可以了

selenium 提供了一整套完整的鼠标和键盘行为事件，功能还是蛮强大滴。下一篇介绍多窗口的情况下如何处理。

2.7 多窗口、句柄（handle）

前言

有些页面的链接打开后，会重新打开一个窗口，对于这种情况，想在新页面上操作，就得先切换窗口了。获取窗口的唯一标识用句柄表示，所以只需要切换句柄，我们就能在多个页面上灵活自如的操作了。

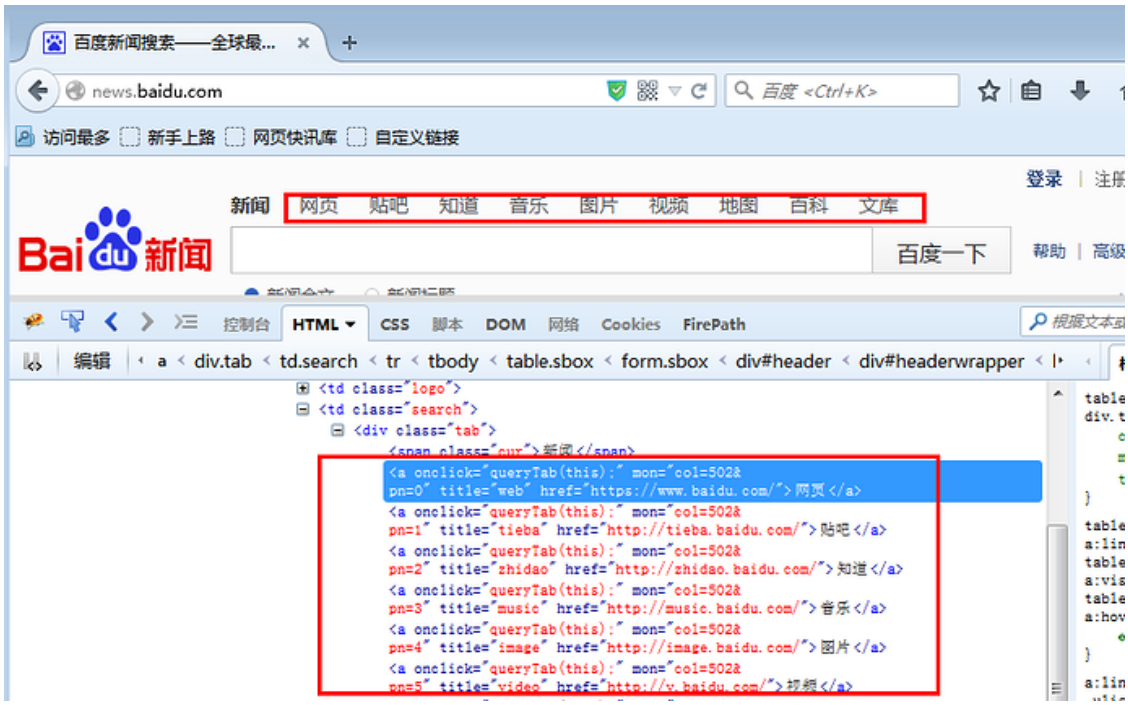
本篇以打开百度新闻页面搜索按钮上的链接页面为例，依次打开每个按钮，并检验测试结果。用脚本批量操作，可以减少重复劳动，重复的事情让脚本去执行吧！

2.7.1 定位一组元素

1. 打开百度新闻页面：17http://news.baidu.com/17

2. 定位搜索框上一排按钮网页、贴吧，知道等

3. 定位一组元素方法在第八篇已经讲过，这里就不多讲，通过 firebug 可以查看他们有共同属性标签为 a，且有个老爸为：



2.7.2 获取当前窗口句柄

1. 当点击百度新闻页面上“网页”按钮时，会打开一个新的窗口

2. 人为操作的话，可以通过点击窗口切换到不同的窗口上，但是脚本它不知道你要操作哪个窗口，这时候只能获取窗口唯一的标识：句柄

3. 获取当前页面的句柄：driver.current_window_handle

2.7.3 获取所有句柄

1. 先通过 css 定位到所有按钮：".tab>a"

2. 从定位的一组中随便取一个点击打开

3. 然后获取当前页面的所有句柄，发现此时有两个了

```
# 定位网页、贴吧等链接
s = driver.find_elements_by_css_selector(".tab>a")
# 点击第一个按钮
s[0].click()
all_h = driver.window_handles
print(all_h)
```

```
idu_news
D:\python\python.exe D:/test_project/test_case/baidu_news.py
{1d46b69b-962e-4819-b16e-d704056d9881}
[u'{1d46b69b-962e-4819-b16e-d704056d9881}', u'{4129929c-e73f-487b-8b03-d0e042d8eb0f}']

Process finished with exit code 0
```

2.7.4 切换句柄

1. 循环判断是否与首页句柄相等

2. 如果不等，说明是新页面的句柄

3. 获取的新页面句柄后，可以切换到新打开的页面上

4. 打印新页面的 title,看是否切换成功

```
    # 循环判断是否与首页句柄相等
    for i in all_h:
        if i != h:
            # 如果不等于首页句柄则切换
            driver.switch_to.window(i)
            print driver.title
```

```
idu_news
D:\python\python.exe D:/test_project/test_case/baidu_news.py
{e508c9d1-d454-4f57-b326-0136c12be087}
[u'{e508c9d1-d454-4f57-b326-0136c12be087}', u'{37558df7-bdd4-4971-b358-f1ab8638951f}']
百度一下，你就知道

Process finished with exit code 0
```

2.7.5 关闭新窗口，切回主页

1. 打开新页面后，其实只想验证新页面跳转对不对，这里可以做个简单的验证，获取当前页面的 title 验证

2. 验证完后切关闭新窗口

3. 切回句柄到首页

4. 打印当前页面的句柄，看是否切换到首页了

```
    for i in all_h:
        if i != h:
            driver.switch_to.window(i)
        print driver.title
        if u"百度一下" in driver.title:
            print("页面打开正常")
        else:
            print ("测试失败")
        driver.close()                          # 关闭当前页面
        driver.switch_to.window(h)              # 切回句柄到首页
        print driver.current_window_handle      # 打印当前句柄
```

```
idu_news
D:\python\python.exe D:/test_project/test_case/baidu_news.py
{5fc67161-8048-4b4d-857b-59699b73f548}
[u'{5fc67161-8048-4b4d-857b-59699b73f548}', u'{9d7f1edc-08b6-442f-ab89-46bef9b8f0b9}']
百度一下，你就知道
页面打开正常
{5fc67161-8048-4b4d-857b-59699b73f548}

Process finished with exit code 0
```

2.7.6 批量操作

1. 把需要校验的结果放在 list 队列 r 里

2. for 循环遍历执行每个按钮的点击操作

3. 判断结果对应 list 里的每个结果

```
s = driver.find_elements_by_css_selector(".tab>a")
r = [u"百度一下", u"贴吧", u"知道",  u"音乐",
     u"图片", u"视频", u"地图", u"百科", u"文库"]
for a, b in zip(s, r):
    a.click()
    text = a.text
    time.sleep(2)
    all_h = driver.window_handles
    # 循环判断是否与首页句柄相等
    for i in all_h:
        if i != h:
            driver.switch_to.window(i)
            time.sleep(1)
    print driver.title
    if b in driver.title:
        print(text+u"页面打开正常")
    else:
        print(text+u"页面测试失败")
    driver.close()                    # 关闭当前页面
    driver.switch_to.window(h)        # 切回句柄到首页
```

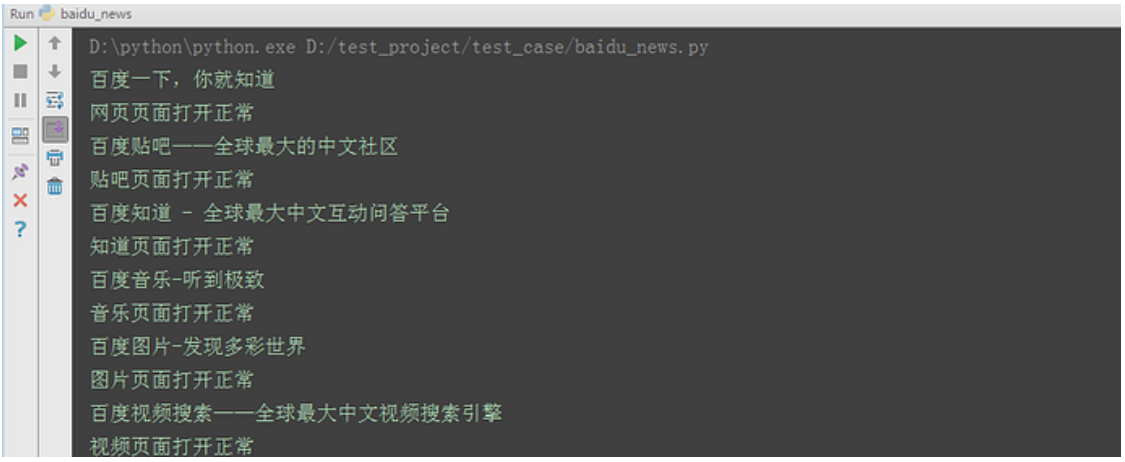
2.7.7 最终脚本

1. 整理后最终脚本如下，结果如图所示

```
# coding:utf-8

from selenium import webdriver

import time

driver = webdriver.Firefox()

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

```
driver.implicitly_wait(10)
# 获取当前窗口句柄
h = driver.current_window_handle
# 定位网页、贴吧等链接
s = driver.find_elements_by_css_selector(".tab>a")
r = [u"百度一下", u"贴吧", u"知道",  u"音乐",
      u"图片", u"视频", u"地图", u"百科", u"文库"]
for a, b in zip(s, r):
    a.click()
    text = a.text
    time.sleep(2)
    all_h = driver.window_handles
    # 循环判断是否与首页句柄相等
    for i in all_h:
        if i != h:
            driver.switch_to.window(i)
            time.sleep(1)
    print driver.title
    if b in driver.title:
        print(text+u"页面打开正常")
    else:
        print(text+u"页面测试失败")
    driver.close()                                    # 关闭当前页面
```

```
        driver.switch_to.window(h)                    # 切回句柄到首页

driver.quit()
```



（备注：此网站已更新过，可以换成赶集网试试）

2.8 定位一组元素 find_elements

前言

前面的几篇都是讲如何定位一个元素，有时候一个页面上有多个对象需要操作，如果一个个去定位的话，比较繁琐，这时候就可以定位一组对象。

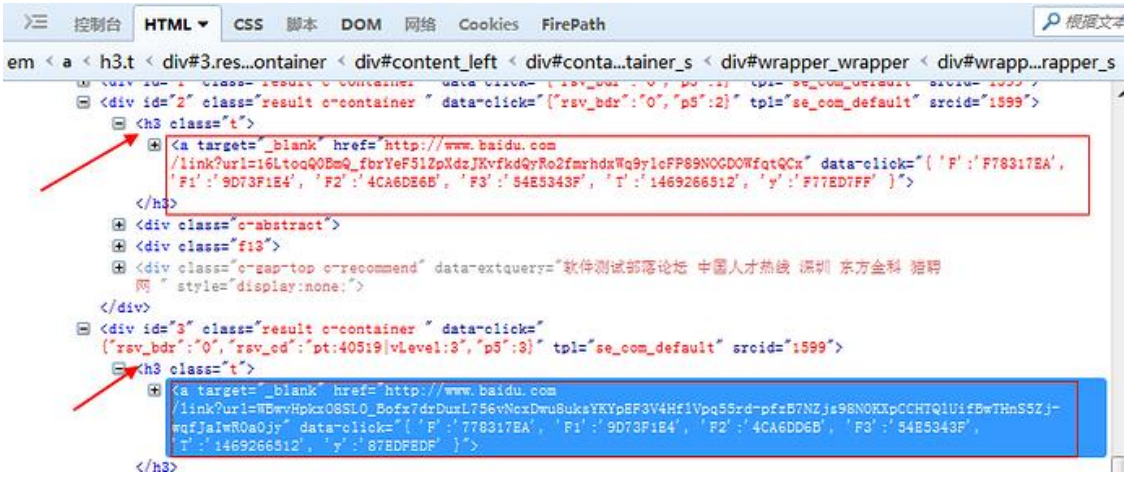
webdriver 提供了定位一组元素的方法，跟前面八种定位方式其实一样，只是前面是单数，这里是复数形式：find_elements

本篇拿百度搜索作为案例，从搜索结果中随机选择一条搜索结果，然后点击查看。

2.8.1 定位搜索结果

Selenium100 例（上海-悠悠）

1.在百度搜索框输入关键字“上海-悠悠”后，用 firebug 查看页面元素，可以看到这些搜索结果有共同的属性。



2.从搜索的结果可以看到，他们的父元素一样：<h3 class="t">

3.标签都一样，且 target 属性也一样：<a target="_blank"

4.于是这里可以用 css 定位（当然用 xpath 也是可以的）

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.implicitly_wait(10)
driver.find_element_by_id("kw").send_keys(u"测试部落")
driver.find_element_by_id("kw").submit()
s = driver.find_elements_by_css_selector("h3.t>a")
```

2.8.2 确认定位结果

1.前面的定位策略只是一种猜想，并不一定真正获取到自己想要的对象的，也行会定位到一些不想要的对象。

2.于是可以获取对象的属性，来验证下是不是定位准确了。这里可以获取 href 属性，打印出 url 地址

2.8.3 随机函数

1.搜索结果有 10 条，从这 10 条中随机取一个就 ok 了

2.先导入随机函数：import random

3.设置随机值范围为 0~9：a=random.randint(0~9)

```
import random
t = random.randint(0,9)
print(t)
```

```
idu_search
D:\python\python.exe D:/test_project/test_case/baidu_search.py
6

Process finished with exit code 0
```

2.8.4 随机打开 url

1.从返回结果中随机取一个 url 地址

2.通过 get 方法打卡 url

3.其实这种方式是接口测试了，不属于 UI 自动化，这里只是开阔下思维，不建议用这种方法

```
# coding:utf-8
from selenium import webdriver
import random
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
driver.implicitly_wait(10)
driver.find_element_by_id("kw").send_keys(u"测试部落")
driver.find_element_by_id("kw").submit()
s = driver.find_elements_by_css_selector("h3.t>a")
# 设置随机值
t = random.randint(0, 9)
# 随机取一个结果获取url地址
a = s[t].get_attribute("href")
print a
driver.get(a)
```

2.8.5 通过 click 点击打开

1.前面那种方法 ,是直接访问 url 地址 ,算是接口测试的范畴了 ,真正模拟用户点击行为，得用 click 的方法

```
# coding:utf-8

from selenium import webdriver

import random

driver = webdriver.Firefox()

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

driver.implicitly_wait(10)

driver.find_element_by_id("kw").send_keys(u"yoyoketang")

driver.find_element_by_id("kw").submit()

s = driver.find_elements_by_css_selector("h3.t>a")

# 设置随机值

t = random.randint(0, 9)

# 随机取一个结果点击鼠标

s[t].click()
```

不知道有小伙伴有没注意一个细节 ,前面在搜索框输入关键字后 ,我并没有去点击搜索按钮，而是用的 submit 的方法，submit 相当于回车键。

具体的操作对象方法，下篇详细介绍。本篇主要学会定位一组对象，然后随机操作其中的一个。

2.9 iframe

前言

有很多小伙伴在拿 163 作为登录案例的时候，发现不管怎么定位都无法定位到，到底是什么鬼呢，本篇详细介绍 iframe 相关的切换

以 http://mail.163.com/登录页面 10 为案例，详细介绍 switch_to_frame 使用方法.

2.9.1 frame 和 iframe 区别

frame 与 iframe 两者可以实现的功能基本相同，不过 iframe 比 frame 具有更多的灵活性。 frame 是整个页面的框架，iframe 是内嵌的网页元素，也可以说是内嵌的框架

iframe 标记又叫浮动帧标记，可以用它将一个 HTML 文档嵌入在一个 HTML 中显示。它和 Frame 标记的最大区别是在网页中嵌入 的<iframe></iframe>所包含的内容与整个页面是一个整体，而<frame>< /frame>所包含的内容是一个独立的个体，是可以独立显示的。另外，应用 iframe 还可以在同一个页面中多次显示同一内容，而不必重复这段内容的代码。

2.9.2 163 登录界面

1. 打开 http://mail.163.com/登录页面 10

2. 用 firebug 定位登录框

3. 鼠标停留在左下角（定位到 iframe 位置）时，右上角整个登录框显示灰色，说明 iframe 区域是整个登录框区域

4. 左下角箭头位置显示 iframe 属性
<iframe id="x-URS-iframe" frameborder="0" name=""

2.9.3 切换 iframe

1. 由于登录按钮是在 iframe 上，所以第一步需要把定位器切换到 iframe 上

2. 用 switch_to_frame 方法切换，此处有 id 属性，可以直接用 id 定位切换

```python
# coding:utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://mail.163.com/")
driver.implicitly_wait(30)
# 切换iframe
driver.switch_to_frame("x-URS-iframe")
driver.find_element_by_name("email").send_keys("123")
driver.find_element_by_name("password").send_keys("456")
```

2.9.4 如果 iframe 没有 id 怎么办？

1. 这里 iframe 的切换是默认支持 id 和 name 的方法的，当然实际情况中会遇到没有 id 属性和 name 属性为空的情况，这时候就需要先定位 iframe

2. 定位元素还是之前的八种方法同样适用，这里我可以通过 tag 先定位到，也能达到同样效果

```python
# coding:utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://mail.163.com/")
driver.implicitly_wait(30)
# 切换iframe
iframe = driver.find_element_by_tag_name("iframe")
driver.switch_to_frame(iframe)
# driver.switch_to_frame("x-URS-iframe")
driver.find_element_by_name("email").send_keys("123")
driver.find_element_by_name("password").send_keys("456")
```

2.9.5 释放 iframe

1. 当 iframe 上的操作完后，想重新回到主页面上操作元素，这时候，就可以用 switch_to_default_content()方法返回到主页面

```
# coding:utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://mail.163.com/")
driver.implicitly_wait(30)
# 切换iframe
# iframe = driver.find_element_by_tag_name("iframe")
# driver.switch_to_frame(iframe)
# driver.switch_to_frame("x-URS-iframe")
driver.switch_to.frame("x-URS-iframe")
driver.find_element_by_name("email").send_keys("123")
driver.find_element_by_name("password").send_keys("456")
# 释放iframe，重新回到主页面上
driver.switch_to.default_content()
```

2.9.6 如何判断元素是否在 iframe 上？

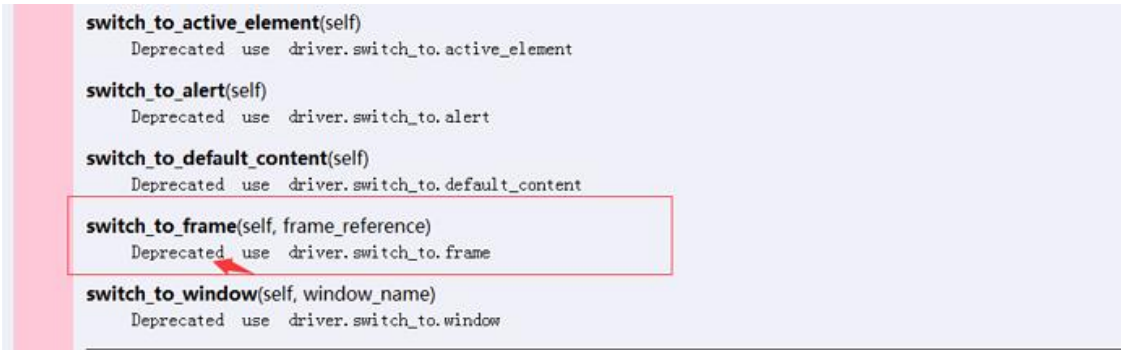
1. 定位到元素后，切换到 firepath 界面

2. 看 firebug 工具左上角，如果显示 Top Window 说明没有 iframe

3. 如果显示 iframe#xxx 这样的，说明在 iframe 上，#后面就是它的 id

2.9.7 如何解决 switch_to_frame 上的横线呢？

1.先找到官放的文档介绍



2.官方已经不推荐上面的写法了，用这个写法就好了
driver.switch_to.frame()

八、参考代码如下

```
# coding:utf-8

from selenium import webdriver



driver = webdriver.Firefox()

driver.get("http://mail.163.com/")

driver.implicitly_wait(30)

# 切换 iframe

# iframe = driver.find_element_by_tag_name("iframe")

# driver.switch_to_frame(iframe)

# driver.switch_to_frame("x-URS-iframe")

driver.switch_to.frame("x-URS-iframe")

driver.find_element_by_name("email").send_keys("123")

driver.find_element_by_name("password").send_keys("456")

# 释放 iframe，重新回到主页面上
```

```
driver.switch_to.default_content()
```

2.10 select 下拉框

前言
最近由于工作原因，更新慢了一点，今天终于抽出一点时间给大家继续更新 selenium 系列，学习的脚本不能停止，希望小伙伴能多多支持。

本篇以百度设置下拉选项框为案例，详细介绍 select 下拉框相关的操作方法。

2.10.1 认识 select

1.打开百度-设置-搜索设置界面，如下图所示

2.箭头所指位置，就是 select 选项框，打开页面元素定位，下方红色框框区域，可以看到 select 标签属性：

```
<select  id="nr"  name="NR">
```

3.选项有三个

```
<option  selected=""  value="10">每页显示 10 条</option>
<option  value="20">每页显示 20 条</option>
<option  value="50">每页显示 50 条</option>
```

2.10.2 二次定位

1.定位 select 里的选项有多种方式，这里先介绍一种简单的方法：二次定位

2.基本思路，先定位 select 框，再定位 select 里的选项

3.代码如下

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到"设置"按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 分两步：先定位下拉框，再点击选项
s = driver.find_element_by_id("nr")
s.find_element_by_xpath("//option[@value='50']").click()
```

4.还有另外一种写法也是可以的，把最下面两步合并成为一步：

```
driver.find_element_by_id("nr").find_element_by_xpath("//option[@value='50']").click()
```

2.10.3 直接定位

1.有很多小伙伴说 firebug 只能定位到 select 框，不能定位到里面的选项，其实是工具掌握的不太熟练。小编接下来教大家如何定位里面的选项。

2.用 direbug 定位到 select 后，下方查看元素属性地方，点 select 标签前面的+号，就可以展开里面的选项内容了。

3. 然后自己写 xpath 定位或者 css, 一次性直接定位到 option 上的内容。（不会自己手写的，回头看前面的元素定位内容）

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到"设置"按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 直接通过xpath定位
driver.find_element_by_xpath(".//*[@id='nr']/option[2]").click()
```

2.10.4 Select 模块(index)

1. 除了上面介绍的两种简单的方法定位到 select 选项，selenium 还提供了更高级的玩法，导入 Select 模块。直接根据属性或索引定位。

2. 先要导入 select 方法：

```
from selenium.webdriver.support.select import Select
```

3. 然后通过 select 选项的索引来定位选择对应选项（从 0 开始计数），如选择第三个选项:select_by_index(2)

```python
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.select import Select
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到"设置"按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 通过索引：select_by_index()
s = driver.find_element_by_id("nr")
Select(s).select_by_index(2)
```

2.10.5 Select 模块(value)

1.Select 模块里面除了 index 的方法，还有一个方法，通过选项的 value 值来定位。每个选项，都有对应的 value 值，如

<select id="nr" name="NR">
<option selected="" value="10">每页显示 10 条</option>

<option value="20">每页显示 20 条</option>

<option value="50">每页显示 50 条</option>

2.第二个选项对应的 value 值就是“20”：select_by_value(2)

```python
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.select import Select
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到"设置"按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 通过value：select_by_value()
s = driver.find_element_by_id("nr")
Select(s).select_by_value("20")
```

2.10.6 Select 模块(text)

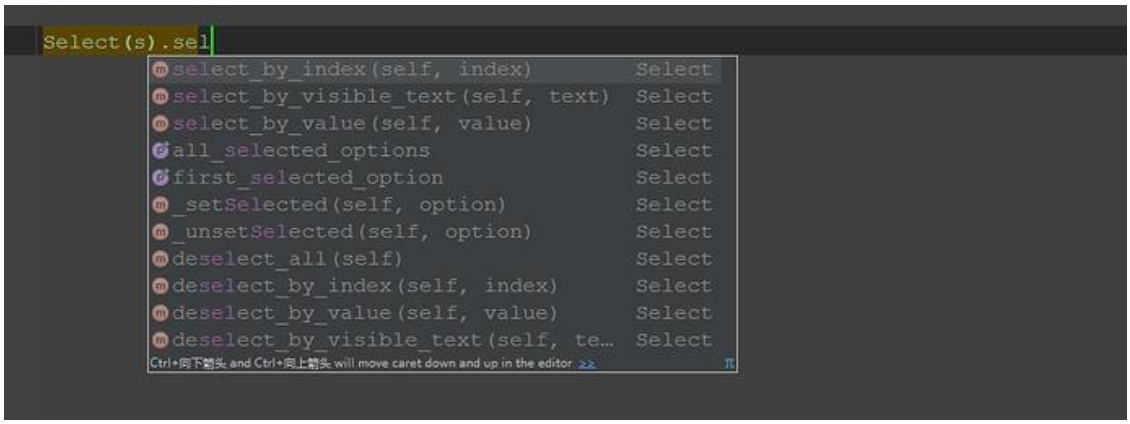
1.Select 模块里面还有一个更加高级的功能，可以直接通过选项的文本内容来定位。

2.定位“每页显示 50 条”：select_by_visible_text("每页显示 50 条")

```
from selenium.webdriver.support.select import Select
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到"设置"按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 通过text:select_by_visible_text()
s = driver.find_element_by_id("nr")
Select(s).select_by_visible_text("每页显示50条")
```

2.10.7 Select 模块其它方法

1.select 里面方法除了上面介绍的三种，还有更多的功能如下



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
deselect_all()　　　　　　：取消所有选项
deselect_by_index()　　：取消对应 index 选项
deselect_by_value()　　：取消对应 value 选项
deselect_by_visible_text()：取消对应文本选项

first_selected_option()　：返回第一个选项
all_selected_options()　　：返回所有的选项

2.10.8 参考代码：

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.select import Select
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
driver.implicitly_wait(20)
# 鼠标移动到“设置”按钮
mouse = driver.find_element_by_link_text("设置")
ActionChains(driver).move_to_element(mouse).perform()
driver.find_element_by_link_text("搜索设置").click()
# 通过 text:select_by_visible_text()
s = driver.find_element_by_id("nr")
Select(s).select_by_visible_text("每页显示 50 条")

# # 分两步：先定位下拉框，再点击选项
# s = driver.find_element_by_id("nr")
# s.find_element_by_xpath("//option[@value='50']").click()

# # 另外一种写法
#
driver.find_element_by_id("nr").find_element_by_xpath("//option[@valu
e='50']").click()

# # 直接通过 xpath 定位
# driver.find_element_by_xpath(".//*[@id='nr']/option[2]").click()

# # 通过索引：select_by_index()
# s = driver.find_element_by_id("nr")
# Select(s).select_by_index(2)

# # 通过 value：select_by_value()
# s = driver.find_element_by_id("nr")
# Select(s).select_by_value("20")
```

2.11 alert\confirm\prompt

前言

不是所有的弹出框都叫 alert，在使用 alert 方法前，先要识别出到底是不是 alert。先认清楚 alert 长什么样子，下次碰到了，就可以用对应方法解决。

alert\confirm\prompt 弹出框操作主要方法有：

text：获取文本值

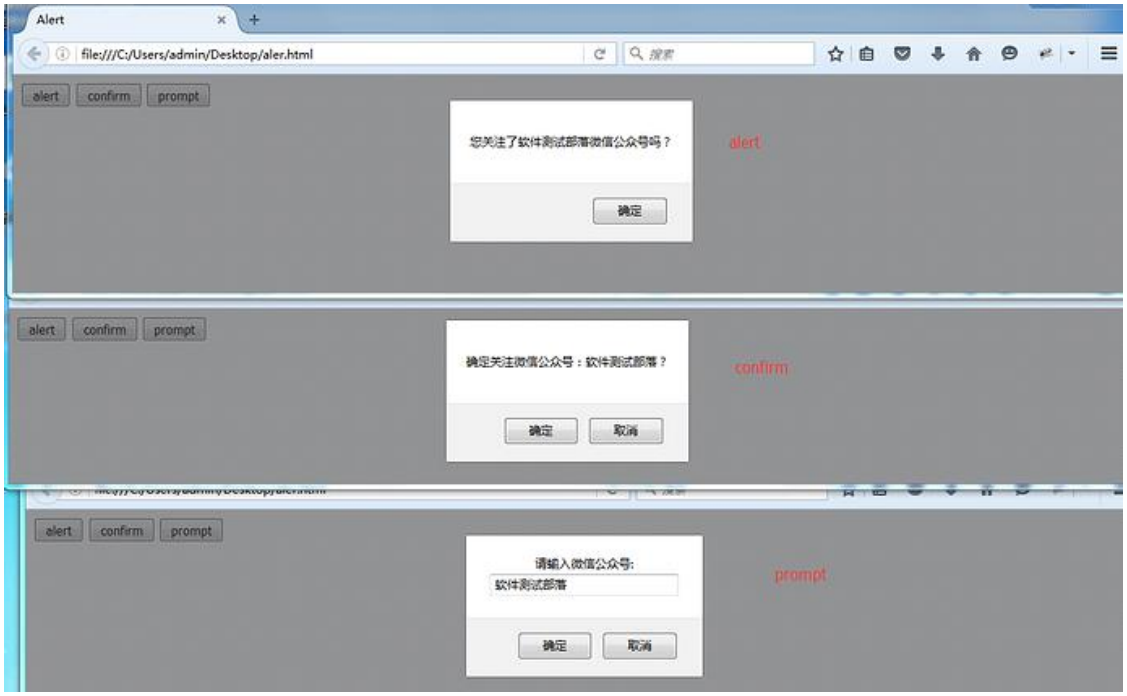
accept() ：点击"确认"

dismiss() ：点击"取消"或者叉掉对话框

send_keys() ：输入文本值 --仅限于 prompt,在 alert 和 confirm 上没有输入框

2.11.1 认识 alert\confirm\prompt

1.如下图，从上到下依次为 alert\confirm\prompt，先认清楚长什么样子，以后遇到了就知道如何操作了。



2.html 源码如下（有兴趣的可以 copy 出来，复制到 txt 文本里，后缀改成 html 就可以了，然后用浏览器打开）

```
<html>

<head>

     <title>Alert</title>

      </head>

<body>

<input id = "alert" value = "alert" type = "button" onclick = "alert('您关注了 yoyoketang 吗？');"/>

<input id = "confirm" value = "confirm" type = "button" onclick = "confirm('确定关注微信公众号：yoyoketang？');"/>

<input id = "prompt" value = "prompt" type = "button" onclick = "var name = prompt('请输入微信公众号:','yoyoketang'); document.write(name) "/>

</body>

</html>
```

2.11.2 alert 操作

1.先用 switch_to_alert()方法切换到 alert 弹出框上

2.可以用 text 方法获取弹出的文本 信息

3.accept()点击确认按钮

4.dismiss()相当于点右上角 x，取消弹出框

（url 的路径，直接复制浏览器打开的路径）

```python
# coding:utf-8
from selenium import webdriver
import time
url = "file:///C:/Users/admin/Desktop/testalert.html"
driver = webdriver.Firefox()
driver.get(url)
time.sleep(4)
driver.find_element_by_id("alert").click()
time.sleep(3)
t = driver.switch_to_alert()
# 打印警告框文本内容
print t.text
# 点警告框确认按钮
t.accept()
# t.dismiss()相当于点x按钮，取消
```

```
lert  testalert
D:\soft\python2.7\python.exe D:/web_project/debug/testalert.py
您关注了软件测试部落微信公众号吗？

Process finished with exit code 0
```

2.11.3 confirm 操作

1. 先用 switch_to_alert()方法切换到 alert 弹出框上

2. 可以用 text 方法获取弹出的文本 信息

3. accept()点击确认按钮

4. dismiss()相当于点取消按钮或点右上角 x，取消弹出框

（url 的路径，直接复制浏览器打开的路径）

```python
# coding:utf-8
from selenium import webdriver
import time
url = "file:///C:/Users/admin/Desktop/testalert.html"
driver = webdriver.Firefox()
driver.get(url)
time.sleep(4)
driver.find_element_by_id("confirm").click()
time.sleep(3)
t = driver.switch_to_alert()
# 打印警告框文本内容
print t.text
# 点警告框确认按钮
t.accept()
# t.dismiss()相当于点x按钮，取消
```

```
ert  testalert
D:\soft\python2.7\python.exe D:/web_project/debug/testalert.py
确定关注微信公众号：软件测试部落？
```

2.11.4 prompt 操作

1. 先用 switch_to_alert()方法切换到 alert 弹出框上

2. 可以用 text 方法获取弹出的文本 信息

3. accept()点击确认按钮

4. dismiss()相当于点右上角 x，取消弹出框

5. send_keys()这里多个输入框，可以用 send_keys()方法输入文本内容（url 的路径，直接复制浏览器打开的路径）

```
# coding:utf-8
from selenium import webdriver
import time
url = "file:///C:/Users/admin/Desktop/testalert.html"
driver = webdriver.Firefox()
driver.get(url)
time.sleep(4)
driver.find_element_by_id("prompt").click()
time.sleep(3)
t = driver.switch_to_alert()
# 打印警告框文本内容
print t.text
t.send_keys("hello selenium2")
# 点警告框确认按钮
# t.accept()
# t.dismiss()相当于点x按钮，取消
```

```
alert  testalert
D:\soft\python2.7\python.exe D:/web_project/debug/testalert.py
请输入微信公众号:
```

2.11.5 select 遇到的坑

1.在操作百度设置里面，点击“保存设置”按钮时，alert 弹出框没有弹出来。（Ie 浏览器是可以的）

2.分析原因：经过慢慢调试后发现，在点击"保存设置"按钮时，由于前面的 select 操作后，失去了焦点

3.解决办法：在 select 操作后，做个 click()点击操作


```
s = driver.find_element_by_id("nr")
```

```
Select(s).select_by_visible_text("每页显示 20 条")

time.sleep(3)

s.click()
```

2.11.6 最终代码

```
# coding:utf-8

from selenium import webdriver

from selenium.webdriver.common.action_chains import ActionChains

from selenium.webdriver.support.select import Select

import time

driver = webdriver.Firefox()

url = "https://www.baidu.com"

driver.get(url)

driver.implicitly_wait(20)

# 鼠标移动到“设置”按钮

mouse = driver.find_element_by_link_text("设置")

ActionChains(driver).move_to_element(mouse).perform()

driver.find_element_by_link_text("搜索设置").click()

# 通过 text:select_by_visible_text()

s = driver.find_element_by_id("nr")

Select(s).select_by_visible_text("每页显示 20 条")

time.sleep(3)

s.click()

driver.find_element_by_link_text("保存设置").click()
```

```
time.sleep(5)

# 获取 alert 弹框

t = driver.switch_to_alert()

print t.text

t.accept()
```

这一篇应该比较简单，Alert 相关的内容比较少，虽然有一些页面也有弹窗，但不是所有的弹窗都叫 alert。alert 的弹出 框界面比较简洁，是调用的系统弹窗警告框，没花里胡哨的东西，还是很容易区分的。

2.12 单选框和复选框（radiobox、checkbox）

本篇主要介绍单选框和复选框的操作

2.12.1 认识单选框和复选框

1.先认清楚单选框和复选框长什么样

2.各位小伙伴看清楚哦，上面的单选框是圆的；下图复选框是方的，这个是业界的标准，要是开发小伙伴把图标弄错了，可以先抽他了。

2.12.2 radio 和 checkbox 源码

1.上图的 html 源码如下，把下面这段复杂下来，写到文本里，后缀改成.html 就可以了。

```
  <html>
       <head>
       <meta http-equiv="content-type"
content="text/html;charset=utf-8" />
       <title>单选和复选</title>
       </head>
       <body>

       </form>
       <h4>单选：性别</h4>
       <form>
       <label value="radio">男</label>
       <input name="sex" value="male" id="boy" type="radio"><br>
       <label value="radio1">女</label>
       <input name="sex" value="female" id="girl" type="radio">
       </form>

       <h4>微信公众号：从零开始学自动化测试</h4>
       <form>
       <!-- <label for="c1">checkbox1</label> -->
       <input id="c1" type="checkbox">selenium<br>
       <!-- <label for="c2">checkbox2</label> -->
       <input id="c2" type="checkbox">python<br>
       <!-- <label for="c3">checkbox3</label> -->
       <input id="c3" type="checkbox">appium<br>

       <!-- <form>
       <input type="radio" name="sex" value="male" /> Male
       <br />
       <input type="radio" name="sex" value="female" /> Female
       </form> -->

       </body>
       </html>
```

2.12.3 单选：radio

1. 首先是定位选择框的位置

2. 定位 id，点击图标就可以了，代码如下（获取 url 地址方法：把上面源码粘贴到文本保存为.html 后缀后用浏览器打开，在浏览器 url 地址栏复制出地址就可以了）

3. 先点击 boy 后，等十秒再点击 girl，观察页面变化

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/checkbox.html")
driver.find_element_by_id("c1").click()
```

2.12.4 复选框：checkbox

1.勾选单个框，比如勾选 selenium 这个，可以根据它的 id=c1 直接定位到点击就可以了

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/checkbox.html")
driver.find_element_by_id("c1").click()
```

2.那么问题来了:如果想全部勾选上呢？

2.12.5 全部勾选：

1.全部勾选，可以用到定位一组元素，从上面源码可以看出，复选框的 type=checkbox,这里可以用 xpath 语法：.//*[@type='checkbox']

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/checkbox.html")
checkboxs = driver.find_elements_by_xpath(".//*[@type='checkbox']")
for i in checkboxs:
    i.click()
```

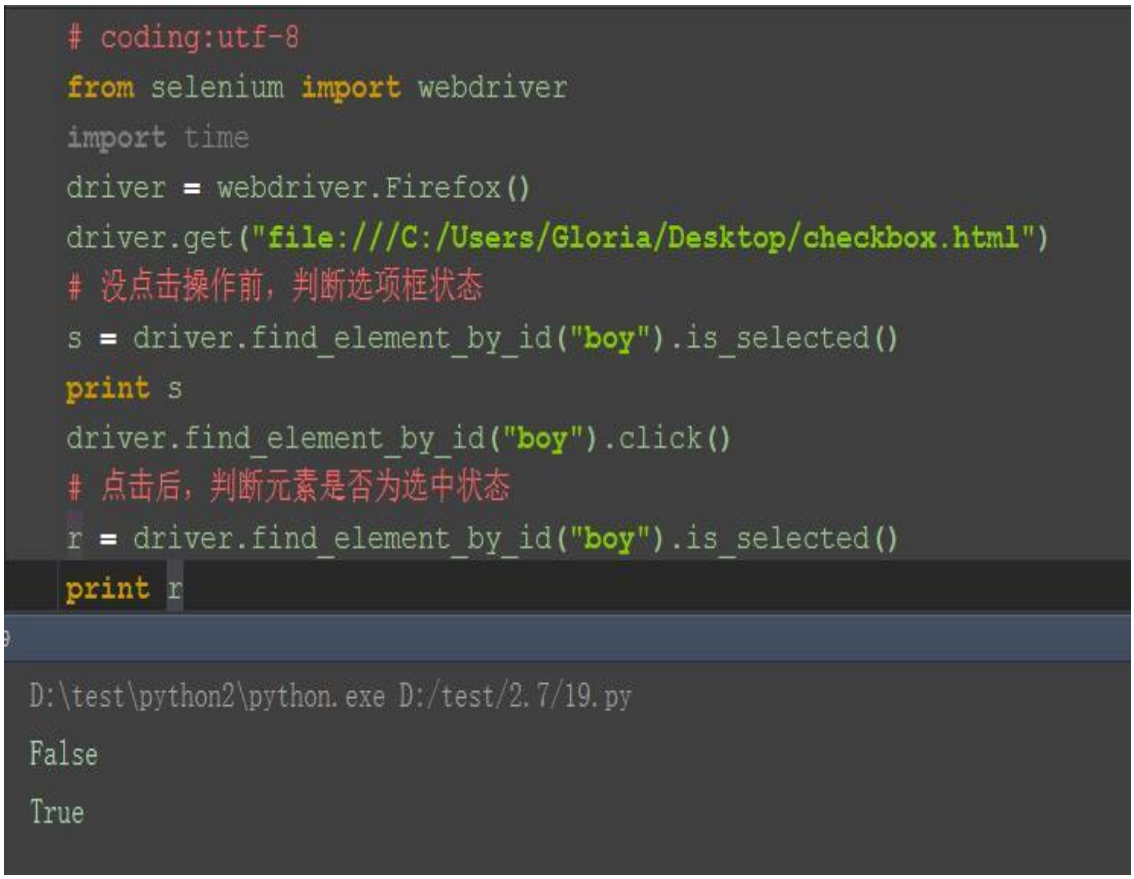
2.这里注意，敲黑板做笔记了：find_elements 是不能直接点击的，它是复数的，所以只能先获取到所有的 checkbox 对象，然后通过 for 循环去一个个点击操作

2.12.6 判断是否选中：is_selected()

1.有时候这个选项框，本身就是选中状态，如果我再点击一下，它就反选了,这可不是我期望的结果,那么可不可以当它是没选中的时候,我去点击下；当它已经是选中状态，我就不点击呢？那么问题来了：如何判断选项框是选中状态？

2.判断元素是否选中这一步才是本文的核心内容，点击选项框对于大家来说没什么难度。获取元素是否为选中状态，打印结果如下图。

3.返回结果为 bool 类型，没点击时候返回 False,点击后返回 True，接下来就很容易判断了，既可以作为操作前的判断，也可以作为测试结果的判断



2.12.7 参考代码：

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/checkbox.html")
# 没点击操作前，判断选项框状态
s = driver.find_element_by_id("boy").is_selected()
print s
driver.find_element_by_id("boy").click()
# 点击后，判断元素是否为选中状态
r = driver.find_element_by_id("boy").is_selected()
print r

# 复选框单选
driver.find_element_by_id("c1").click()
# 复选框全选
checkboxs = driver.find_elements_by_xpath(".//*[@type='checkbox']")
for i in checkboxs:
        i.click()
```

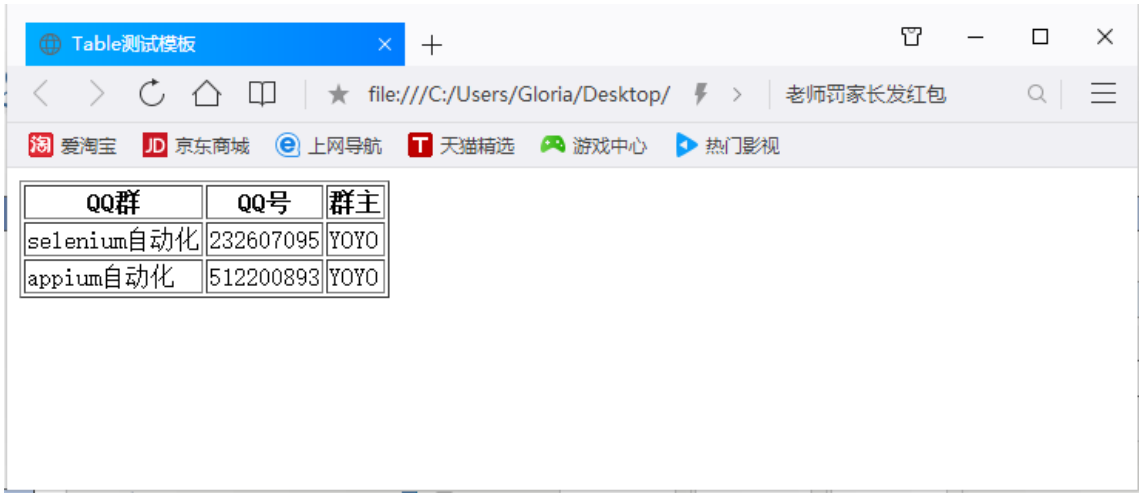
2.13 table 定位

前言

在 web 页面中经常会遇到 table 表格，特别是后台操作页面比较常见。本篇详细讲解 table 表格如何定位。

2.13.1 认识 table

1. 首先看下 table 长什么样，如下图，这种网状表格的都是 table



2. 源码如下：（用 txt 文本保存，后缀改成 html）

```
<!DOCTYPE html>
<meta charset="UTF-8"> <!-- for HTML5 -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<html>
		<head>
			<title>Table 测试模板</title>

		</head>
		<body>
			<table border="1" id="myTable">
				<tr>
					<th>QQ 群</th>
					<th>QQ 号</th>
					<th>群主</th>
				</tr>
				<tr>
					<td>selenium 自动化</td>
```

```
                                        <td>232607095</td>
                                        <td>YOYO</td>
                                    </tr>
                                    <tr>
                                        <td>appium 自动化</td>
                                        <td>512200893</td>
                                        <td>YOYO</td>
                                    </tr>
                                </table>
                            </body>
</html>
```

2.13.2 table 特征

1.table 页面查看源码一般有这几个明显的标签：table、tr、th、td

2.<table>标示一个表格

3.<tr>标示这个表格中间的一个行

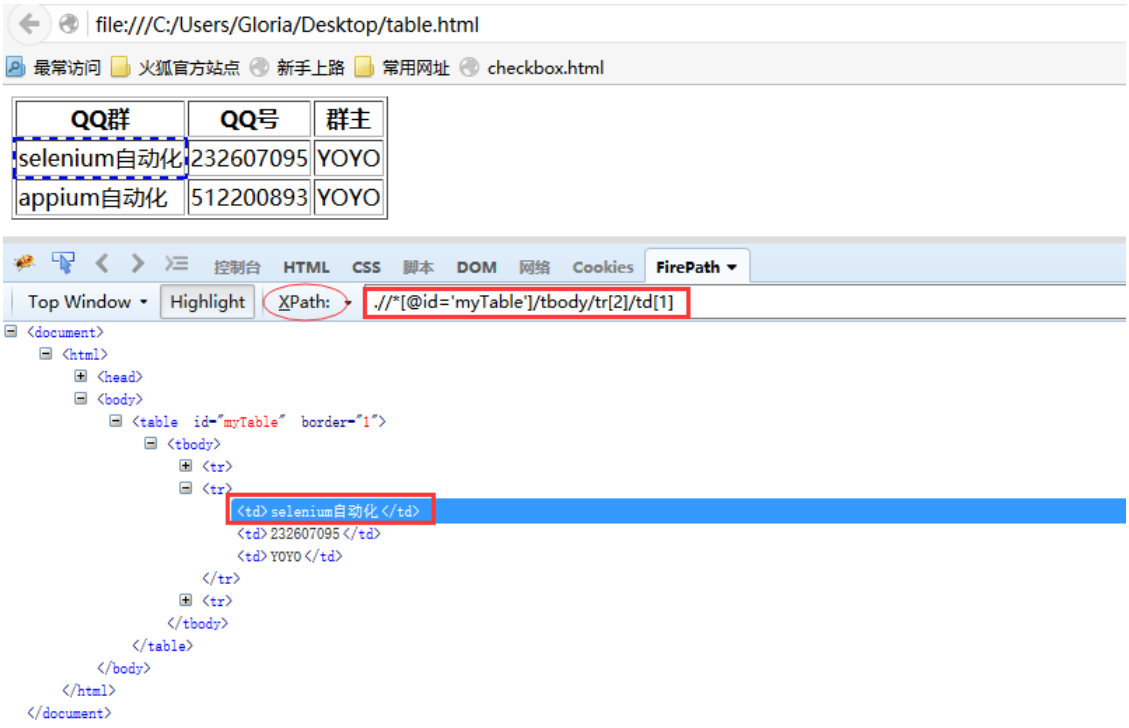
4.</th> 定义表头单元格

5.</td> 定义单元格标签，一组<td>标签将将建立一个单元格，<td>标签必须放在<tr>标签内

2.13.3 xpath 定位 table

1.举个例子：我想定位表格里面的“selenium 自动化”元素，这里可以用 xpath 定位：

```
.//*[@id='myTable']/tbody/tr[2]/td[1]
```



2. 这里定位的格式是固定的，只需改 tr 和 td 后面的数字就可以了. 如第二行第一列 tr[2]td[1].

对 xpath 语法不熟悉的可以看这篇 Selenium2+python 自动化 7-xpath 定位

2.13.4 打印表格内容

1. 定位到表格内文本值，打印出来，脚本如下

```
from selenium import webdriver

url = 'file:///C:/Users/Gloria/Desktop/table.html'
driver = webdriver.Firefox()
driver.get(url)

t = driver.find_element_by_xpath(".//*[@id='myTable']/tbody/tr[2]/td[1]")
print t.text
```

```
D:\test\python2\python.exe D:/test/web-project/table.py
selenium自动化

Process finished with exit code 0
```

2.13.5 参考代码：

```
# coding:utf-8
from selenium import webdriver
import time
url = 'file:///C:/Users/Gloria/Desktop/table.html'
driver = webdriver.Firefox()
driver.get(url)
time.sleep(3)
t =
driver.find_element_by_xpath(".//*[@id='myTable']/tbody/tr[2]/td[1]")
print t.text
```

补充说明：有些小伙伴可能会遇到 table 在 ifame 上的情况，这时候就需要先切换 iframe 了

2.14 加载 Firefox 配置

前言
有小伙伴在用脚本启动浏览器时候发现原来下载的插件不见了，无法用 firebug 在打开的页面上继续定位页面元素，调试起来不方便 。

加载浏览器配置，需要用 FirefoxProfile(profile_directory)这个类来加载，

profile_directory 既为浏览器配置文件的路径地址

2.14.1 遇到问题

1.在使用脚本打开浏览器时候，发现右上角原来下载的插件 firebug 不见了，到底去哪了呢？

2.用脚本去打开浏览器时候，其实是重新打开了一个进程，跟手动打开浏览器不是一个进程。

所以没主动加载插件，不过 selenium 里面其实提供了对应的方法去打开，只是很少有人用到。

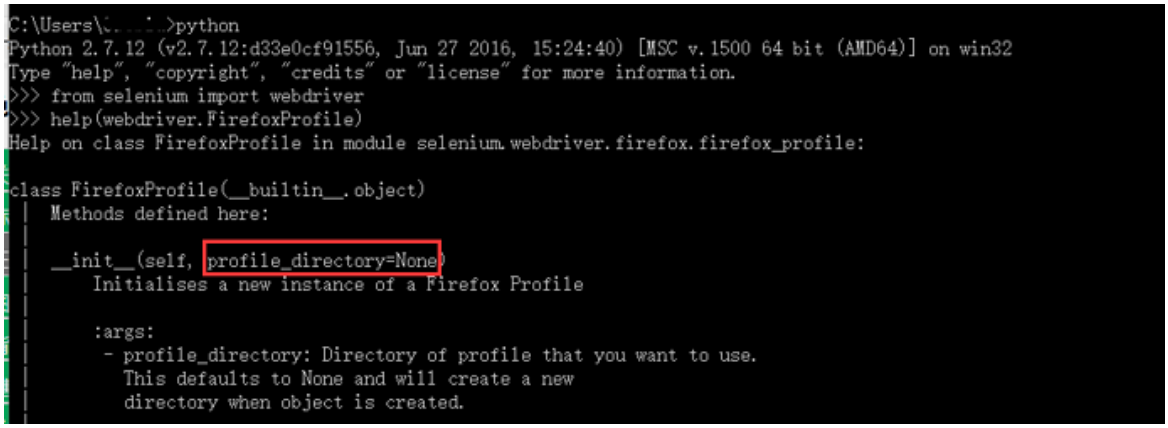
2.14.2 FirefoxProfile

1.要想了解 selenium 里面 API 的用法，最好先看下相关的帮助文档打开 cmd 窗口，

输入如下信息：
-》python

-》from selenium import webdriver

-》help(webdriver.FirefoxProfile)



Help on class FirefoxProfile in module

```
selenium.webdriver.firefox.firefox_profile:

class FirefoxProfile(builtin.object)
 | Methods defined here:
 |
 | init(self, profile_directory=None)
 | Initialises a new instance of a Firefox Profile
 |
 | :args:
 | - profile_directory: Directory of profile that you want to use.
 | This defaults to None and will create a new
 | directory when object is created.
```

2.翻译过来大概意思是说，这里需要 profile_directory 这个配置文件路径的参数

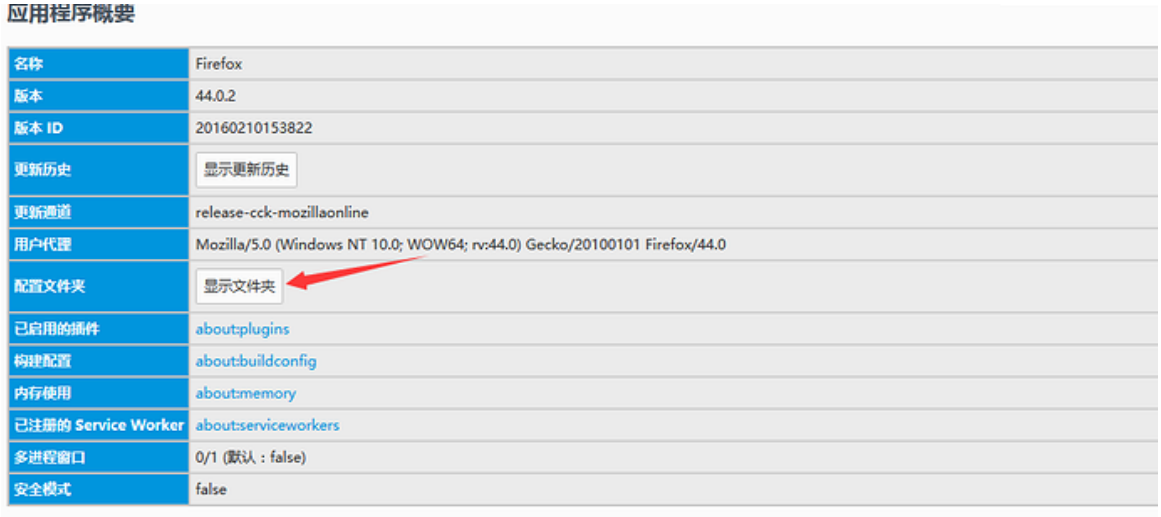
3.profile_directory=None，如果没有路径，默认为 None，启动的是一个新的，

有的话就加载指定的路径。

2.14.3 profile_directory

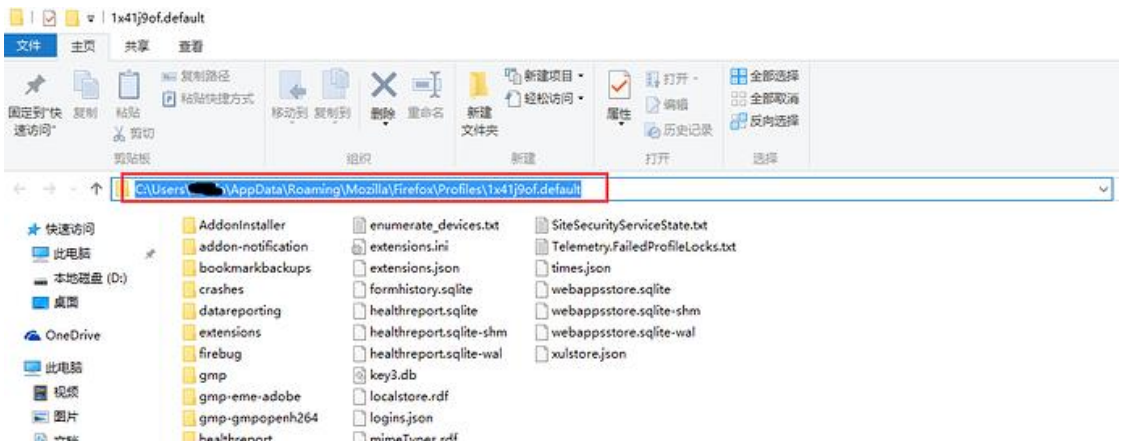
1.问题来了：Firefox 的配置文件地址如何找到呢？

2.打开 Firefox 点右上角设置>？（帮助）>故障排除信息>显示文件夹



3.打开后把路径复制下来就可以了：

C:\Users\xxx\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.default



2.14.4 启动配置文件

1.由于文件路径存在字符：\ ，反斜杠在代码里是转义字符，这个有点代码基础的应该都知道。

不懂什么叫转义字符的，自己翻书补下基础吧！

2.遇到转义字符，为了不让转义，有两种处理方式：

第一种：\\（前面再加一个反斜杠）

第二种:r”\”（字符串前面加 r，使用字符串原型）

```
# coding=utf-8
from selenium import webdriver
# 配置文件地址
profile_directory = r'C:\Users\[redacted]\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.default'
# 加载配置配置
profile = webdriver.FirefoxProfile(profile_directory)
# 启动浏览器配置
driver = webdriver.Firefox(profile)
```

2.14.5 参考代码：

```
# coding=utf-8
from selenium import webdriver
# 配置文件地址
```

```
profile_directory =
r'C:\Users\xxx\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.default'
# 加载配置配置
profile = webdriver.FirefoxProfile(profile_directory)
# 启动浏览器配置
driver = webdriver.Firefox(profile)
```

2.15 富文本

前言

富文本编辑框是做 web 自动化最常见的场景，有很多小伙伴遇到了不知道无从下手，本篇以博客园的编辑器为例，解决如何定位富文本，输入文本内容

2.15.1 加载配置

1.打开博客园写随笔，首先需要登录，这里为了避免透露个人账户信息，我直接加载配置文件，免登录了。

不懂如何加载配置文件的，看这篇 Selenium2+python 自动化 18-加载 Firefox 配置

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

profileDir = r'C:\xxx...\Firefox\Profiles\1x41j9of.default'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)
```

2.15.2 打开编辑界面

1.博客首页地址：bolgurl = "http://www.cnblogs.com/"

2.我的博客园地址：yoyobolg = bolgurl + "yoyoketang"

3. 点击“新随笔”按钮，id=blog_nav_newpost

```
bolgurl = "http://www.cnblogs.com/"
yoyobolg = bolgurl + "yoyoketang"
driver.get(yoyobolg)
driver.find_element_by_id("blog_nav_newpost").click()
```

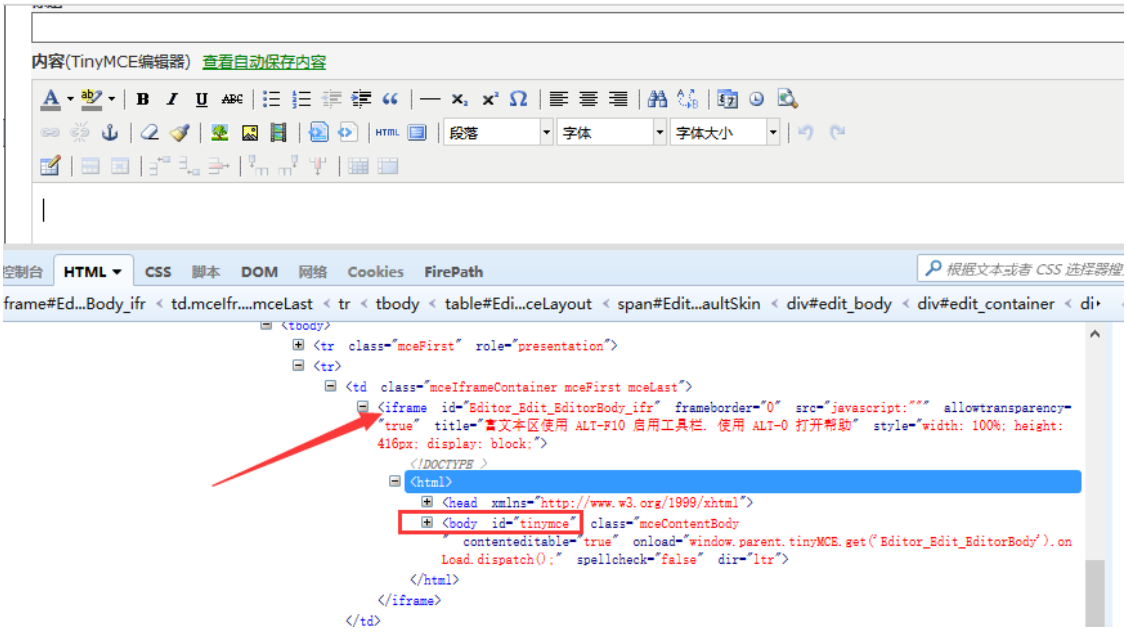
2.15.3 iframe 切换

1. 打开编辑界面后先不要急着输入内容，先 sleep 几秒钟

2. 输入标题，这里直接通过 id 就可以定位到，没什么难点

3. 接下来就是重点要讲的富文本的编辑，这里编辑框有个 iframe，所以需要先切换

（关于 iframe 不懂的可以看前面这篇：Selenium2+python 自动化 14-iframe）



2.15.4 输入正文

1.这里定位编辑正文是定位上图的红色框框位置 body 部分，也就是 id=tinymce

2.定位到之后，直接 send_keys()方法就可以输入内容了

3.有些小伙伴可能输入不成功，可以在输入之前先按个 table 键，send_keys(Keys.TAB)

```
time.sleep(5)
edittile = u"Selenium2+python自动化23-富文本"
editbody = u"这里是发帖的正文"
driver.find_element_by_id("Editor_Edit_txbTitle").send_keys(edittile)
driver.switch_to.frame("Editor_Edit_EditorBody_ifr")
driver.find_element_by_id("tinymce").send_keys(Keys.TAB)
driver.find_element_by_id("tinymce").send_keys(editbody)
```

2.15.5 参考代码：

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

profileDir =
r'C:\Users\Gloria\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.d
efault'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)

bolgurl = "http://www.cnblogs.com/"
yoyobolg = bolgurl + "yoyoketang"
driver.get(yoyobolg)
driver.find_element_by_id("blog_nav_newpost").click()

time.sleep(5)
edittile = u"Selenium2+python 自动化-富文本"
editbody = u"这里是发帖的正文"
driver.find_element_by_id("Editor_Edit_txbTitle").send_keys(edittile)
```

```
driver.switch_to.frame("Editor_Edit_EditorBody_ifr")
driver.find_element_by_id("tinymce").send_keys(Keys.TAB)
driver.find_element_by_id("tinymce").send_keys(editbody)
```

2.16 文件上传（send_keys）

前言

文件上传是 web 页面上很常见的一个功能，自动化成功中操作起来却不是那么简单。

一般分两个场景：一种是 input 标签，这种可以用 selenium 提供的 send_keys()方法轻松解决；

另外一种非 input 标签实现起来比较困难，可以借助 autoit 工具或者 SendKeys 第三方库。

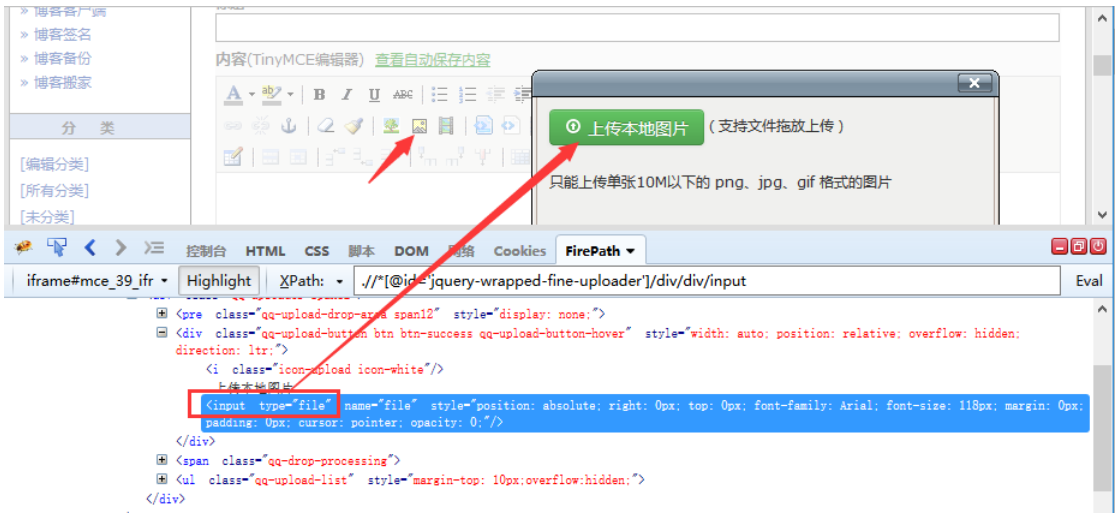
本篇以博客园的上传图片为案例，通过 send_keys()方法解决文件上传问题

2.16.1 识别上传按钮

1. 点开博客园编辑器里的图片上传按钮，弹出”上传本地图片”框。

2. 用 firebug 查看按钮属性，这种上传图片按钮有个很明显的标识，它是一个 input 标签，并且 type 属性的值为 file。

只要找到这两个标识，我们就可以直接用 send_keys()方法上传文件了。



2.16.2 定位 iframe

1. 这里定位图片上传按钮情况有点复杂，首先它是在 iframe 上（不懂 iframe 的看这篇:Selenium2+python 自动化 14-iframe）

2. 这个 iframe 的 id 是动态的，且没有 name 属性，其它属性也不是很明显

3. 通过搜索发现，这个页面上有两个 iframe，需要定位的这个 iframe 是处于第二个位置

4. 可以通过标签定位所有的 iframe 标签，然后取对应的第几个就可以了。

```
driver.get("http://www.cnblogs.com/yoyoketang/")
driver.find_element_by_link_text("新随笔").click()
time.sleep(3)
# 点开编辑器图片
driver.find_element_by_css_selector("img.mceIcon").click()
time.sleep(3)
# 定位所有iframe,取第二个
iframe = driver.find_elements_by_tag_name('iframe')[1]
# 切换到iframe上
driver.switch_to_frame(iframe)
```

2.16.3 文件上传

1. 先定位到文件上传按钮，直接调用 send_keys()方法就可以实现啦

```
# coding:utf-8
from selenium import webdriver
import time
```

```
profileDir =
r'C:\Users\Gloria\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.d
efault'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)
driver.implicitly_wait(30)
driver.get("http://www.cnblogs.com/yoyoketang/")
driver.find_element_by_link_text("新随笔").click()
time.sleep(3)
# 点开编辑器图片
driver.find_element_by_css_selector("img.mceIcon").click()
time.sleep(3)
# 定位所有 iframe,取第二个
iframe = driver.find_elements_by_tag_name('iframe')[1]
# 切换到 iframe 上
driver.switch_to_frame(iframe)

# 文件路径
driver.find_element_by_name('file').send_keys(r"D:\test\xuexi\test\14
.png")
```

备注：非 input 标签的文件上传，就不适用于此方法了，需要借助 autoit 工具或者 SendKeys 第三方库。

2.17 获取元素属性

前言

通常在做断言之前，都要先获取界面上元素的属性，然后与期望结果对比。本篇介绍几种常见的获取元素属性方法。

2.17.1 获取页面 title

1.有很多小伙伴都不知道 title 长在哪里，看下图左上角。

2. 获取 title 方法很简单，直接 driver.title 就能获取到

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://www.baidu.com")
time.sleep(2)
title = driver.title
print title
```

```
keyuan
D:\test\python2\python.exe D:/test/xuexi/element/bokeyuan.py
百度一下，你就知道

Process finished with exit code 0
```

2.17.2 获取元素的文本

1. 如下图这种显示在页面上的文本信息，可以直接获取到

2. 查看元素属性：<a id="setf" target="_blank" onmousedown="return ns_c({'fm':'behs','tab':'favorites','pos':0})

" href="//www.baidu.com/cache/sethelp/help.html">**把百度设为主页**

3. 通过 driver.text 获取到文本

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://www.baidu.com")
time.sleep(2)
title = driver.title
print title
text = driver.find_element_by_id("setf").text
print text
```

```
bokeyuan
D:\test\python2\python.exe D:/test/xuexi/element/bokeyuan.py
百度一下，你就知道
把百度设为主页
```

2.17.3 获取元素的标签

1. 获取百度输入框的标签属性

```
    # 获取元素的标签
    tag = driver.find_element_by_id("kw").tag_name
    print tag
```

```
okeyuan
D:\test\python2\python.exe D:/test/xuexi/element/bokeyuan.py
百度一下，你就知道
把百度设为主页
input
```

2.17.4 获取元素的其它属性

1.获取其它属性方法:get_attribute("属性")，这里的参数可以是 class、name 等任意属性

2.如获取百度输入框的 class 属性

```
    # 获取元素的其它属性
    name = driver.find_element_by_id("kw").get_attribute("class")
    print name
```

```
okeyuan
D:\test\python2\python.exe D:/test/xuexi/element/bokeyuan.py
百度一下，你就知道
把百度设为主页
input
s_ipt
```

2.17.5 获取输入框内的文本值

1、如果在百度输入框输入了内容，这里输入框的内容也是可以获取到的

```
    # 获取输入框的内容
    driver.find_element_by_id("kw").send_keys("yoyoketang")
    value = driver.find_element_by_id("kw").get_attribute("value")
    print value
```

```
keyuan
D:\test\python2\python.exe D:/test/xuexi/element/bokeyuan.py
yoyoketang

Process finished with exit code 0
```

2.17.6 获取浏览器名称

1.获取浏览器名称很简单，用 driver.name 就能获取到

```
# 获取浏览器名称

driver.name
```

2.17.7 参考代码

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://www.baidu.com")
time.sleep(2)
title = driver.title
print title
text = driver.find_element_by_id("setf").text
print text
# 获取元素的标签
tag = driver.find_element_by_id("kw").tag_name
print tag
# 获取元素的其它属性
name = driver.find_element_by_id("kw").get_attribute("class")
print name
# 获取输入框的内容
driver.find_element_by_id("kw").send_keys("yoyoketang")
value = driver.find_element_by_id("kw").get_attribute("value")
print value
# 获取浏览器名称
print driver.name
```

2.18 爬页面源码（page_source）

前言

有时候通过元素的属性的查找页面上的某个元素，可能不太好找，这时候可以从源码中爬出想要的信息。selenium 的 page_source 方法可以获取到页面源码。

2.18.1 page_source

1. selenium 的 page_source 方法可以直接返回页面源码

2. 重新赋值后打印出来

```
# coding:utf-8
from selenium import webdriver
import re
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang/")
page = driver.page_source
print page
```

```
D:\test\python2\python.exe D:/test/xuexi/pa.py
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="zh-cn"><head>
<meta charset="utf-8" />
<meta content="width=device-width, initial-scale=1" name="viewport" />
<title>上海-悠悠 - 博客园</title>
```

2.18.2 re 非贪婪模式

1. 这里需导入 re 模块

2. 用 re 的正则匹配：非贪婪模式

3. findall 方法返回的是一个 list 集合

4. 匹配出来之后发现有一些不是 url 链接，可以删选下

```
# "非贪婪匹配,re.S('.'匹配字符,包括换行符)"
url_list = re.findall('href=\"(.*?)\"', page, re.S)
for url in url_list:
    print url
```

```
D:\test\python2\python.exe D:/test/xuexi/pa.py
/bundles/blog-common.css?v=03KQeESEmpLfzDqUoONiWUg5Zd5aRam3JHBZTKO5Wug1
/skins/coffee/bundle-coffee.css?v=H7P3zU3_M529NtLVh9kKULJaZLDGv9Qz_L_VdgyN1P41
/blog/customcss/319138.css?v=UX8%2bVoXBiSDs5YSRKz2E3bJcjLO%3d
/skins/coffee/bundle-coffee-mobile.css?v=SRuzVEnpvmJLI16sZ57txiTZOm1_h-eefz9mvfcpj381
http://www.cnblogs.com/yoyoketang/rss
http://www.cnblogs.com/yoyoketang/rsd.xml
http://www.cnblogs.com/yoyoketang/wlwmanifest.xml
```

2.18.3 删选 url 地址出来

1.加个 if 语句判断，‘http’在 url 里面说明是正常的 url 地址了

2.把所有的 url 地址放到一个集合，就是我们想要的结果啦

2.18.4 参考代码

```
# coding:utf-8
from selenium import webdriver
import re
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang/")
page = driver.page_source
# print page
# "非贪婪匹配,re.S('.'匹配字符,包括换行符)"
url_list = re.findall('href=\"(.*?)\"', page, re.S)
url_all = []
for url in url_list:
        if "http" in url:
                print url
                url_all.append(url)
# 最终的 url 集合
print url_all
```

2.19 cookie 相关操作

前言

虽然 cookie 相关操作在平常 ui 自动化中用得少，偶尔也会用到，比如登录有图形验证码，可以通过绕过验证码方式，添加 cookie 方法登录。

登录后换账号登录时候，也可作为后置条件去删除 cookie 然后下个账号登录

2.19.1 获取 cookies：get_cookies()

1. 获取 cookies 方法直接用：get_cookies()

2. 先启动浏览器，获取 cookies，打印出来发现是空：[]

3. 打开博客首页后，重新获取 cookies,打印出来，就有值了

```
# coding:utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()
# 启动浏览器后获取cookies
print driver.get_cookies()
driver.get("http://www.cnblogs.com/yoyoketang/")
# 打开主页后获取cookies
print driver.get_cookies()
```

```
coo
D:\test\python2\python.exe D:/test/xuexi/coo.py
[]
[{u'domain': u'.cnblogs.com', u'name': u'_ga', u'value': u'GA1.2.969916148.1489217903', u'expiry'
```

2.19.2 登录后的 cookies

1. 先登录博客园（这里登录用自己的账号和密码吧）

2. 重新获取 cookies，发现跟之前获取的不一样了

3. 主要是找到这一个 cookie,发现它的 name 和 value 发生了变化，这就是未登录和已登录的区别了（对比上下两张图）

{u'name': u'.CNBlogsCookie', u'value':
u'B7813EBA142142CE88CC8C0B33B239F566xxxx'}

```
# 登录后获取cookies
url = "https://passport.cnblogs.com/user/signin"
driver.get(url)
driver.implicitly_wait(30)
driver.find_element_by_id("input1").send_keys(u"上海-悠悠")
driver.find_element_by_id("input2").send_keys(u"xxxxx")
driver.find_element_by_id("signin").click()
time.sleep(3)
print driver.get_cookies()
```

```
oo

domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489219806, u'path': u'/',
, u'name': u'.CNBlogsCookie', u'value': u'B7813EBA142142CE88CC8C0B33B239F56651566E5ACF2133EE27EF7
```

2.19.3 获取指定 name 的 cookie:driver.get_cookie（name）

1.获取 cookies 发现里面有多个 cookie,有时候我们只需要其中的一个，把重要的提出来，比如登录的 cookie

2.这里用 get_cookie（name），指定对应的 cookie 的 name 值就行了，比如博客园的：.CNBlogsCookie

```
# 获取指定name的cookie
print driver.get_cookie(name=".CNBlogsCookie")
```

```
D:\test\python2\python.exe D:/test/xuexi/coo.py
[]
[{u'domain': u'.cnblogs.com', u'name': u'_ga', u'value': u'GA1.2.102829882.1489219746', u'expiry': 15
[{u'domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489219806, u'path': u'/',
{u'domain': u'.cnblogs.com', u'name': u'.CNBlogsCookie', u'value': u'B7813EBA142142CE88CC8C0B33B239F5
```

2.19.4 清除指定 cookie：delete_cookie()

1.为了进一步验证上一步获取到的就是登录的 cookie，可以删除它看看页面什么变化

2.删除这个 cookie 后刷新页面，发现刚才的登录已经失效了，变成未登录状态了

```
# 清除指定name的cookie
driver.delete_cookie(name=".CNBlogsCookie")
print driver.get_cookies()
# 为了验证此cookie是登录的，可以删除后刷新页面
driver.refresh()
```

```
coo
D:\test\python2\python.exe D:/test/xuexi/coo.py
[]
[{u'domain': u'.cnblogs.com', u'name': u'_ga', u'value': u'GA1.2.976505212.1489221099', u'expiry':
[{u'domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489221159, u'path': u'/
{u'domain': u'.cnblogs.com', u'name': u'.CNBlogsCookie', u'value': u'17B338D81ED8DEF181234858160FE6
[{u'domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489221159, u'path': u'/
```

2.19.5 清除所有 cookies：delete_all_cookies()

1.清除所有 cookies 后登录状态也失效了，cookies 为空[]

```
# 清除所有的cookie
driver.delete_all_cookies()
print driver.get_cookies()
```

```
coo
D:\test\python2\python.exe D:/test/xuexi/coo.py
[]
[{u'domain': u'.cnblogs.com', u'name': u'_ga', u'value': u'GA1.2.1631529019.1489221426', u'exp
[{u'domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489221485, u'path'
{u'domain': u'.cnblogs.com', u'name': u'.CNBlogsCookie', u'value': u'368DFDEE43B9D6B6D15D232E
[{u'domain': u'.cnblogs.com', u'name': u'_gat', u'value': u'1', u'expiry': 1489221485, u'path'
[]
```

2.19.6 cookie 操作的几个方法

1.get_cookies()：获取所有 cookies

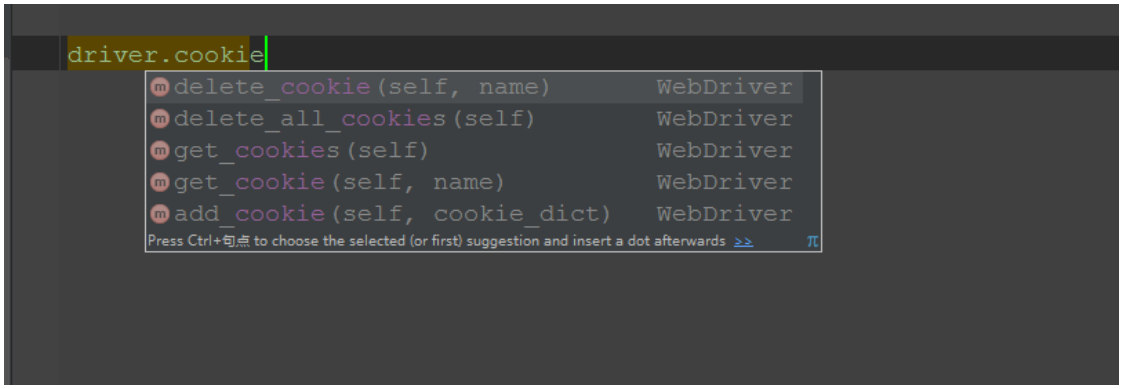
2.driver.get_cookie（name）：获取指定 name 的 cookie:

3.清除指定 cookie：delete_cookie()

4.delete_all_cookies()：清除所有 cookies

5.add_cookie(cookie_dict):添加 cookie 的值

（第五个方法可以用于绕过验证码登录，下篇详细介绍）



2.19.7 参考代码

```
# coding:utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()
# 启动浏览器后获取 cookies
print driver.get_cookies()
driver.get("http://www.cnblogs.com/yoyoketang/")
# 打开主页后获取 cookies
print driver.get_cookies()
# 登录后获取 cookies
url = "https://passport.cnblogs.com/user/signin"
driver.get(url)
driver.implicitly_wait(30)
driver.find_element_by_id("input1").send_keys(u"上海-悠悠")
driver.find_element_by_id("input2").send_keys(u"xxx")
driver.find_element_by_id("signin").click()
time.sleep(3)
print driver.get_cookies()

# 获取指定 name 的 cookie
print driver.get_cookie(name=".CNBlogsCookie")

# 清除指定 name 的 cookie
driver.delete_cookie(name=".CNBlogsCookie")
print driver.get_cookies()
# 为了验证此 cookie 是登录的，可以删除后刷新页面
driver.refresh()

# 清除所有的 cookie
```

```
driver.delete_all_cookies()
print driver.get_cookies()
```

2.20 绕过验证码（add_cookie）

前言

验证码这种问题是比较头疼的，对于验证码的处理，不要去想破解方法，这个验证码本来就是为了防止别人自动化登录的。如果你能破解，说明你们公司的验证码吗安全级别不高，那就需要提高级别了。

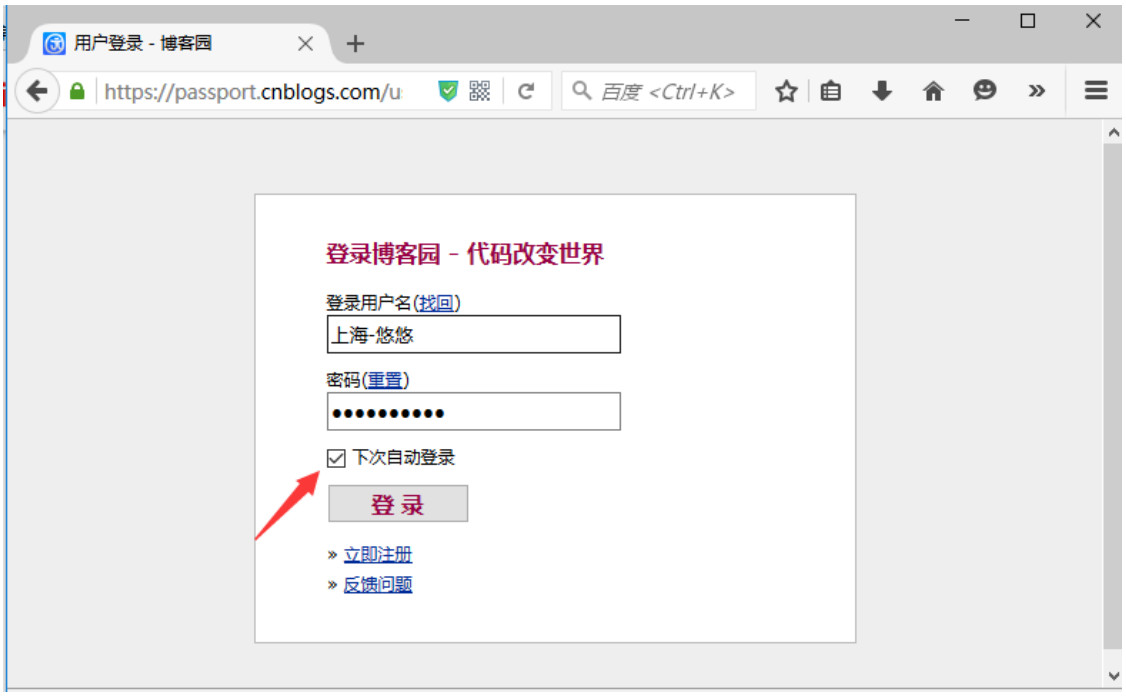
对于验证码，要么是让开发在测试环境弄个万能的验证码，如：1234，要么就是尽量绕过去，如本篇介绍的添加 cookie 的方法。

2.20.1 fiddler 抓包

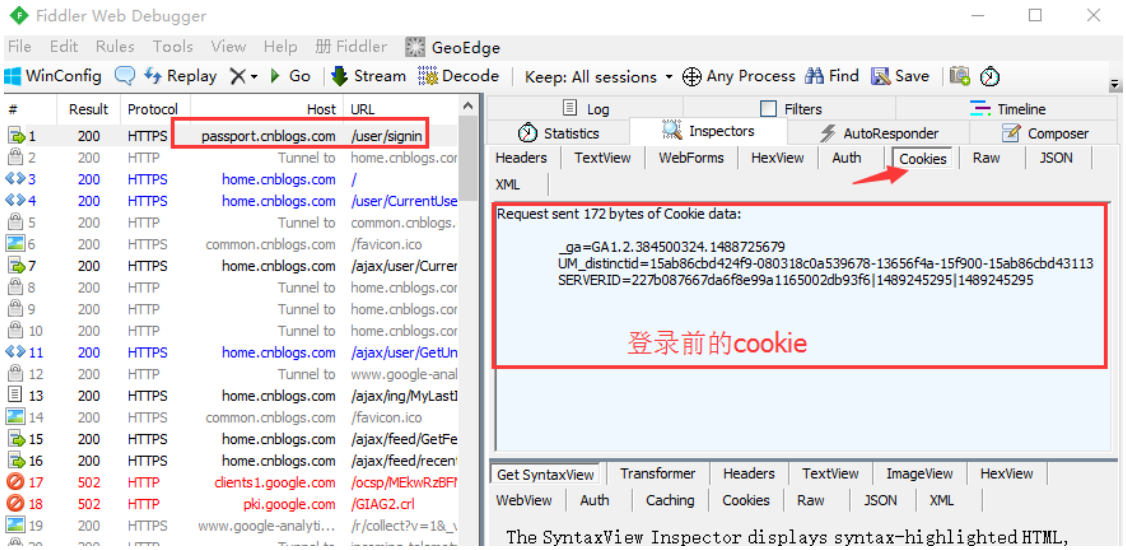
1. 前一篇讲到，登录后会生成一个已登录状态的 cookie，那么只需要直接把这个值添加到 cookies 里面就可以了。

2. 可以先手动登录一次，然后抓取这个 cookie，这里就需要用抓包工具 fiddler 了

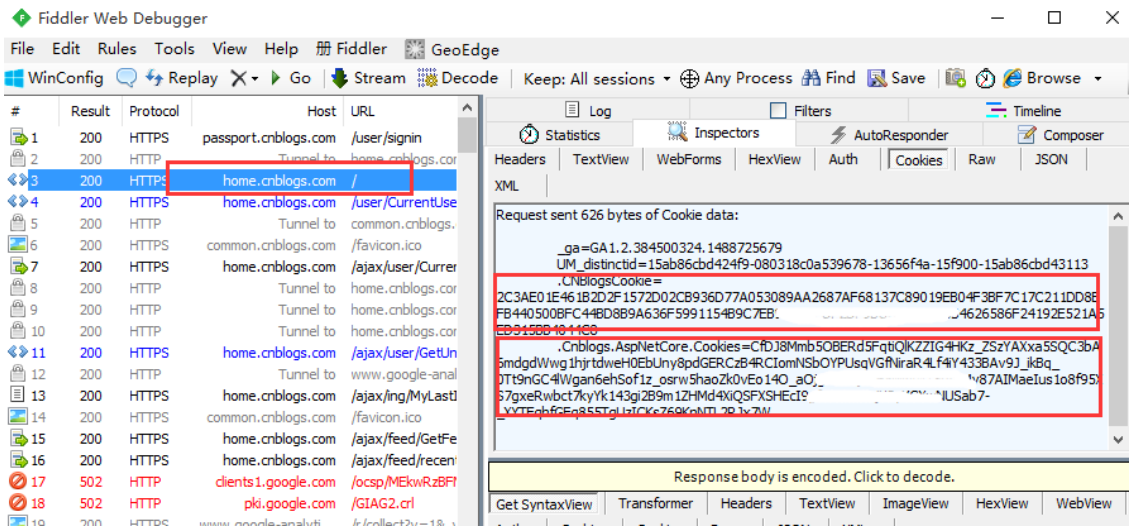
3. 先打开博客园登录界面，手动输入账号和密码（不要点登录按钮）



4. 打开 fiddler 抓包工具，此时再点博客园登录按钮



5. 登录成功后，再查看 cookie 变化，发现多了两组参数，多的这两组参数就是我们想要的，copy 出来，一会有用



2.20.2 添加 cookie 方法：driver.add_cookie（）

1. add_cookie(cookie_dict)方法里面参数是 cookie_dict，说明里面参数是字典类型。

2. 源码官方文档介绍：

```
add_cookie(self, cookie_dict)
Adds a cookie to your current session.

:Args:
- cookie_dict: A dictionary object, with required keys - "name" and
```

```
"value";
optional keys - "path", "domain", "secure", "expiry"

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

3.从官方的文档里面可以看出，添加 cookie 时候传入字典类型就可以了，等号左边的是 name，等号左边的是 value。

4.把前面抓到的两组数据（参数不仅仅只有 name 和 value），写成字典类型：

```
{'name':'.CNBlogsCookie','value':
'2C3AE01E461B2D2F1572D02CB936D77A053089AA2xxxx...'}

{'name':'.Cnblogs.AspNetCore.Cookies','value':'CfDJ8Mmb5OBERd5FqtiQlK
ZZIG4HKz_Zxxx...'}
```

2.20.3 cookie 组成结构

1.用抓包工具 fidller 只能看到 cookie 的 name 和 value 两个参数，实际上 cookie 还有其它参数

2.cookie 参数组成，以下参数是我通过 get_cookie（name）获取到的，

参考上一篇：Selenium2+python 自动化 40-cookie 相关操作

```
cookie ={u'domain': u'.cnblogs.com',
                      u'name': u'.CNBlogsCookie',
                    u'value': u'xxxx',
                     u'expiry': 1491887887,
                      u'path': u'/',
                    u'httpOnly': True,
                    u'secure': False}
```

name：cookie 的名称

value：cookie 对应的值，动态生成的

domain：服务器域名

expiry：Cookie 有效终止日期

path：Path 属性定义了 Web 服务器上哪些路径下的页面可获取服务器设置的 Cookie

httpOnly：防脚本攻击

secure:在 Cookie 中标记该变量，表明只有当浏览器和 Web Server 之间的通信协议为加密认证协议时，

浏览器才向服务器提交相应的 Cookie。当前这种协议只有一种，即为 HTTPS。

2.20.4 添加 cookie

1.这里需要添加两个 cookie，一个是.CNBlogsCookie，另外一个是.Cnblogs.AspNetCore.Cookies。

2.我这里打开的网页是博客的主页：http://www.cnblogs.com/yoyoketang，没进入登录页。

3.添加 cookie 后刷新页面,接下来就是见证奇迹的时刻了。

```
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang")
# # 添加cookie
c1 = {u'domain': u'.cnblogs.com',
      u'name': u'.CNBlogsCookie',
      u'value': u'DF31BF4099BF8F0FFA2E90D5B79A7E4E3C72CBE979C226900FCE56C615F20
      u'expiry': 1491887887,u'path': u'/',u'httpOnly': True,u'secure': False}
c2 = {u'domain': u'.cnblogs.com',
      u'name': u'.Cnblogs.AspNetCore.Cookies',
      u'value': u'CfDJ8Mmb5OBERd5FqtiQ1KZZIG65JBX85wOgESHGXD8yg0HBcNRyc_bdD9M0Q
      u'expiry': 1491887887,u'path': u'/',u'httpOnly': True, u'secure': False}
driver.add_cookie(c1)   # 添加2个值
driver.add_cookie(c2)
time.sleep(3)           # 交流QQ群：232607095
driver.refresh()    # 刷新下页面就见证奇迹了
```

2.20.5 参考代码：

```
# coding:utf-8
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang")

# # 添加 cookie
c1 = {u'domain': u'.cnblogs.com',
           u'name': u'.CNBlogsCookie',
           u'value': u'xxxx',
           u'expiry': 1491887887,
           u'path': u'/',
           u'httpOnly': True,
           u'secure': False}

c2 = {u'domain': u'.cnblogs.com',
           u'name': u'.Cnblogs.AspNetCore.Cookies',
           u'value': u'xxxx',
           u'expiry': 1491887887,
           u'path': u'/',
           u'httpOnly': True,
           u'secure': False}

driver.add_cookie(c1)   # 添加 2 个值
driver.add_cookie(c2)

time.sleep(3)                    # 交流 QQ 群：232607095

# 刷新下页面就见证奇迹了
driver.refresh()
```

有几点需要注意：

1.登录时候要勾选下次自动登录按钮。

2.add_cookie（）只添加 name 和 value，对于博客园的登录是不成功。

3.**本方法并不适合所有的网站，一般像博客园这种记住登录状态的才会适合**。

2.21 JS 处理滚动条

前言

selenium 并不是万能的，有时候页面上操作无法实现的，这时候就需要借助 JS 来完成了。

常见场景：

当页面上的元素超过一屏后，想操作屏幕下方的元素，是不能直接定位到，会报元素不可见的。

这时候需要借助滚动条来拖动屏幕，使被操作的元素显示在当前的屏幕上。

滚动条是无法直接用定位工具来定位的。selenium 里面也没有直接的方法去控制滚动条，

这时候只能借助 J 了，还好 selenium 提供了一个操作 js 的方法:

execute_script()，可以直接执行 js 的脚本。

2.21.1 JavaScript 简介

1.JavaScript 是世界上最流行的脚本语言，因为你在电脑、手机、平板上浏览的所有的网页，

以及无数基于 HTML5 的手机 App，交互逻辑都是由 JavaScript 驱动的。简单地说，

JavaScript 是一种运行在浏览器中的解释型的编程语言。

那么问题来了，为什么我们要学 JavaScript？
2.有些特殊的操作 selenium2+python 无法直接完成的,JS 刚好是这方面的强项，所以算是一个很

好的补充。对 js 不太熟悉的，可以网上找下教程，简单了解些即可。

http://www.w3school.com.cn/js/index.asp4

2.21.2 控制滚动条高度

1.滚动条回到顶部：

```
js="var q=document.getElementById('id').scrollTop=0"
driver.execute_script(js)
```

2.滚动条拉到底部

```
js="var q=document.documentElement.scrollTop=10000"
driver.execute_script(js)
```

2. 这里可以修改 scrollTop 的值,来定位右侧滚动条的位置,0 是最上面,10000 是最底部。

2.21.3 横向滚动条

1.有时候浏览器页面需要左右滚动（一般屏幕最大化后，左右滚动的情况已经很少见了）。
2.通过左边控制横向和纵向滚动条 scrollTo(x, y) js =
"window.scrollTo(100,400);"

```
driver.execute_script(js)
```

3. 第一个参数 x 是横向距离，第二个参数 y 是纵向距离

2.21.4 Chrome 浏览器

1. 以上方法在 Firefox 上是可以的，但是用 Chrome 浏览器，发现不管用。
谷歌浏览器就是这么任性，不听话，于是用以下方法解决谷歌浏览器滚动条的问题。

2. Chrome 浏览器解决办法：

```
js = "var q=document.body.scrollTop=0"
driver.execute_script(js)
```

2.21.5 元素聚焦

1. 虽然用上面的方法可以解决拖动滚动条的位置问题，但是有时候无法确定我需要操作的元素

在什么位置，有可能每次打开的页面不一样，元素所在的位置也不一样，怎么办呢？

2.这个时候我们可以先让页面直接跳到元素出现的位置，然后就可以操作了。同样需要借助 JS 去实现。
3.元素聚焦：

```
target = driver.find_element_by_xxxx()
driver.execute_script("arguments[0].scrollIntoView();", target)
```

2.21.6 获取浏览器名称:driver.name

1.为了解决不同浏览器操作方法不一样的问题，可以写个函数去做兼容。

2.先用 driver.name 获取浏览器名称，然后用 if 语句做个判断

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
print driver.name
```

```
D:\test\python2\python.exe D:/test/web-project/image/name.py
firefox

Process finished with exit code 0
```

2.21.7 兼容性

1.兼容谷歌和 firefox/IE

```
# 回到顶部
def scroll_top():
    if driver.name == "chrome":
        js = "var q=document.body.scrollTop=0"
    else:
        js = "var q=document.documentElement.scrollTop=0"
    return driver.execute_script(js)
# 拉到底部
def scroll_foot():
    if driver.name == "chrome":
        js = "var q=document.body.scrollTop=10000"
    else:
        js = "var q=document.documentElement.scrollTop=10000"
    return driver.execute_script(js)
```

2.21.8 scrollTo 函数

scrollTo 函数不存在兼容性问题，直接用这个函数就可以了

--scrollHeight 获取对象的滚动高度。
--scrollLeft 设置或获取位于对象左边界和窗口中目前可见内容的最左端之间的距离。
--scrollTop 设置或获取位于对象最顶端和窗口中可见内容的最顶端之间的距离。
--scrollWidth 获取对象的滚动宽度。

```
#滚动到底部
js = "window.scrollTo(0,document.body.scrollHeight)"
driver.execute_script(js)

#滚动到顶部
js = "window.scrollTo(0,0)"
driver.execute_script(js)
```

2.21.9 参考代码

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://www.baidu.com")
print driver.name
## 回到顶部
#def scroll_top():
#         if driver.name == "chrome":
#            js = "var q=document.body.scrollTop=0"
#         else:
#                  js = "var q=document.documentElement.scrollTop=0"
#         return driver.execute_script(js)
# 拉到底部
#def scroll_foot():
#      if driver.name == "chrome":
#                  js = "var q=document.body.scrollTop=10000"
#         else:
#                  js = "var
```

```
q=document.documentElement.scrollTop=10000"
#          return driver.execute_script(js)

#滚动到底部
js = "window.scrollTo(0,document.body.scrollHeight)"
driver.execute_script(js)

#滚动到顶部
js = "window.scrollTo(0,0)"
driver.execute_script(js)


# 聚焦元素
target = driver.find_element_by_xxxx()
driver.execute_script("arguments[0].scrollIntoView();", target)
```

2.22 处理富文本（带 iframe）

前言

上一篇 Selenium2+python 自动化 23-富文本（自动发帖）解决了富文本上 iframe 问题，其实没什么特别之处，主要是 iframe 的切换，本篇讲解通过 js 的方法处理富文本上 iframe 的问题

2.22.1 加载配置

1.打开博客园写随笔，首先需要登录，这里为了避免透露个人账户信息，我直接加载配置文件，免登录了。

不懂如何加载配置文件的，看这篇 Selenium2+python 自动化 18-加载 Firefox 配置

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

profileDir = r'C:\xxx...\Firefox\Profiles\1x41j9of.default'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)
```

2.22.2 打开编辑界面

1. 博客首页地址：bolgurl = "http://www.cnblogs.com/"

2. 我的博客园地址：yoyobolg = bolgurl + "yoyoketang"

3. 点击“新随笔”按钮，id=blog_nav_newpost

```
bolgurl = "http://www.cnblogs.com/"
yoyobolg = bolgurl + "yoyoketang"
driver.get(yoyobolg)
driver.find_element_by_id("blog_nav_newpost").click()
```

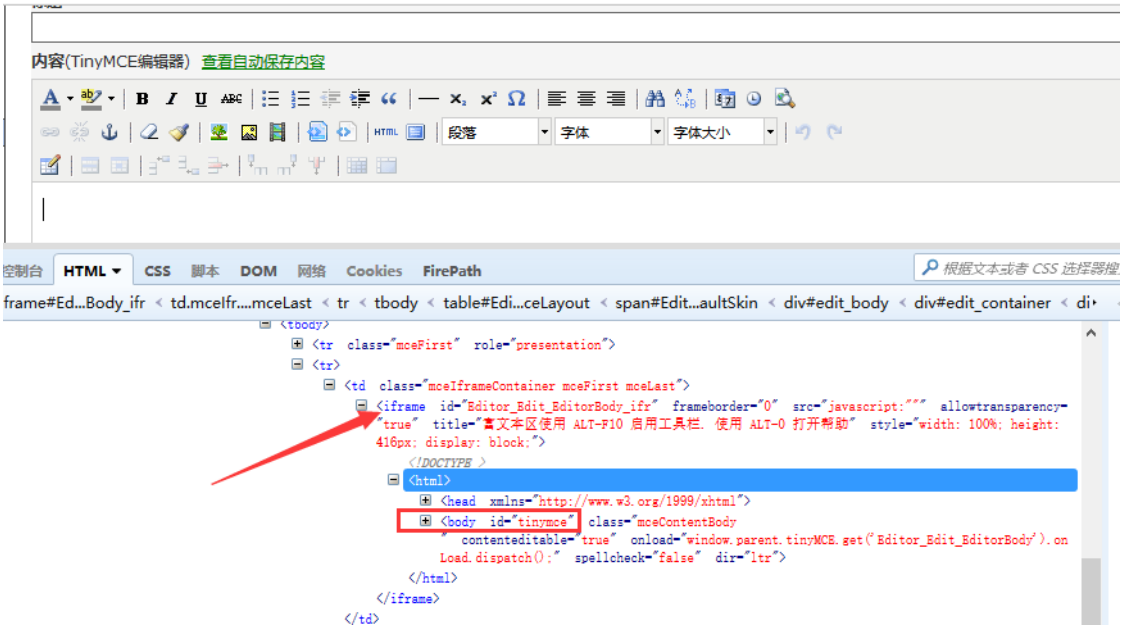
2.22.3 定位 iframe

1. 打开编辑界面后先不要急着输入内容，先 sleep 几秒钟

2. 输入标题，这里直接通过 id 就可以定位到，没什么难点

3. 接下来就是重点要讲的富文本的编辑，这里编辑框有个 iframe，所以需要先切换

（关于 iframe 不懂的可以看前面这篇：Selenium2+python 自动化 14-iframe）



2.22.4 js 输入正文

1.这里定位编辑正文是定位上图的红色框框位置 body 部分，也就是 id=tinymce

2.定位到之后，用 js 的方法直接输入，无需切换 iframe

3.直接点保存按钮，无需再切回来

```
body = "这里是通过js发的正文内容"

# js处理iframe问题(js代码太长了，我分成两行了)
js = 'document.getElementById("Editor_Edit_EditorBody_ifr")' \
     '.contentWindow.document.body.innerHTML="%s"' % body
driver.execute_script(js)
# 保存草稿
driver.find_element_by_id("Editor_Edit_lkbDraft").click()
```

2.22.5 参考代码：

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

# profileDir 路径对应直接电脑的配置路径
profileDir =
r'C:\xxx\xxx\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.defaul
t'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)

bolgurl = "http://www.cnblogs.com/"
yoyobolg = bolgurl + "yoyoketang"
driver.get(yoyobolg)
driver.find_element_by_id("blog_nav_newpost").click()

time.sleep(5)
edittile = u"Selenium2+python 自动化 23-富文本"
editbody = u"这里是发帖的正文"
```

```
driver.find_element_by_id("Editor_Edit_txbTitle").send_keys(edittile)


body = "这里是通过 js 发的正文内容"

# js 处理 iframe 问题(js 代码太长了，我分成两行了)
js = 'document.getElementById("Editor_Edit_EditorBody_ifr")' \
         '.contentWindow.document.body.innerHTML="%s"' % body
driver.execute_script(js)
# 保存草稿
driver.find_element_by_id("Editor_Edit_lkbDraft").click()
```

2.23 js 处理日历控件（修改 readonly 属性）

前言

日历控件是 web 网站上经常会遇到的一个场景，有些输入框是可以直接输入日期的，有些不能，以我们经常抢票的 12306 网站为例，详细讲解如何解决日历控件为 readonly 属性的问题。

基本思路：先用 js 去掉 readonly 属性，然后直接输入日期文本内容

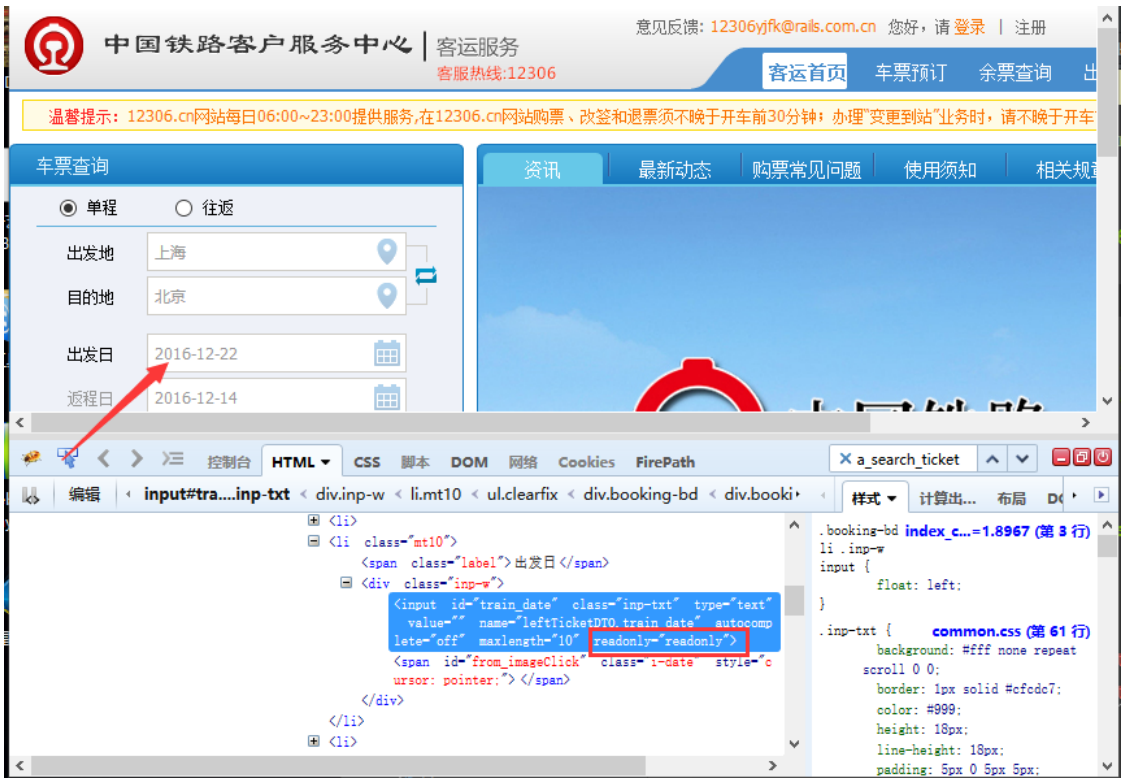
2.23.1 日历控件

1.打开 12306 的车票查询界面，在出发日期输入框无法直接输入时间

2.常规思路是点开日历控件弹出框，从日历控件上点日期，这样操作比较烦躁，并且我们测试的重点不在日历控件上，只是想输入个时间，做下一步的操作

3.用 firebug 查看输入框的属性：readonly="readonly"，如下：

```
<input id="train_date" class="inp-txt" type="text" value="" name
="leftTicketDTO.train_date" autocomplete="off" maxlength="10" read
only="readonly">
```

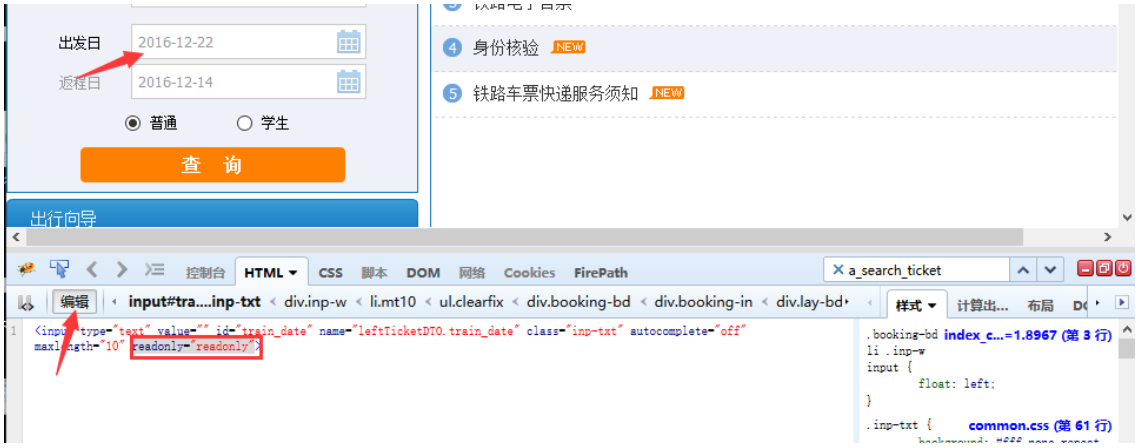


2.23.2 去掉 readonly 属性

1. 很明显这种元素的属性是 readonly，输入框是无法直接输入的，这时候需要先去掉元素的 readonly 属性，然后就可以输入啦。

2. 点左下角 firebug 的“编辑按钮”，找到对应元素，直接删除 readonly="readonly"，然后回车。

3. 在页面出发日位置输入：yoyoketang 试试，嘿嘿，有没发现可以输入成功。当然这里只是为了验证可以输入内容，测试时候还是输入测试的日期。



2.23.3 用 js 去掉 readonly 属性

1.用 js 去掉元素属性基本思路：先定位到元素，然后用 removeAttribute("readonly")方法删除属性。

2.出发日元素 id 为：train_date，对应 js 代码为：'document.getElementById("train_date").removeAttribute("readonly");'

```
# coding:utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("https://kyfw.12306.cn/otn/index/init")

# 去掉元素的readonly属性
js = 'document.getElementById("train_date").removeAttribute("readonly");'
driver.execute_script(js)
```

2.23.4 输入日期

1.输入日期前，一定要先清空文本，要不然无法输入成功的。

2.这里输入日期后，会自动弹出日历控件，随便点下其它位置就好了，接下来会用 js 方法传入日期，就不会弹啦！

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://kyfw.12306.cn/otn/index/init")
# 去掉元素的readonly属性
js = 'document.getElementById("train_date").removeAttribute("readonly");'
driver.execute_script(js)

# 清空文本后输入值
driver.find_element_by_id("train_date").clear()
driver.find_element_by_id("train_date").send_keys("2016-12-25")
```

2.23.5 js 方法输入日期

1.这里也可以用 js 方法输入日期，其实很简单，直接改掉输入框元素的 value 值就可以啦

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://kyfw.12306.cn/otn/index/init")
# 去掉元素的readonly属性
js = 'document.getElementById("train_date").removeAttribute("readonly");'
driver.execute_script(js)

# 用js方法输入日期
js_value = 'document.getElementById("train_date").value="2016-12-25"'
driver.execute_script(js_value)
```

2.23.6 参考代码如下：

```
# coding:utf-8
from selenium import webdriver
driver = webdriver.Firefox()
driver.get("https://kyfw.12306.cn/otn/index/init")
# 去掉元素的 readonly 属性
js =
'document.getElementById("train_date").removeAttribute("readonly");'
driver.execute_script(js)

# 用 js 方法输入日期
js_value = 'document.getElementById("train_date").value="2016-12-25"'
driver.execute_script(js_value)

# # 清空文本后输入值
# driver.find_element_by_id("train_date").clear()
# driver.find_element_by_id("train_date").send_keys("2016-12-25")
```

2.24 js 处理内嵌 div 滚动条

前言

前面有篇专门用 js 解决了浏览器滚动条的问题，生活总是多姿多彩，有的滚动条就在页面上，这时候又得仰仗 js 大哥来解决啦。

2.24.1 内嵌滚动条

1. 下面这张图就是内嵌 div 带有滚动条的样子，记住它的长相。

2.页面源码如下:（老规矩:copy 下来,用文本保存下来,后缀改成.html,用浏览器打开）

```
<!DOCTYPE html>
<meta charset="UTF-8"> <!-- for HTML5 -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<html>
<head>
<style type="text/css">

div.scroll
{
background-color:#afafaf;
width:500px;
height:100px;
overflow:auto;
}


</style>
</head>

<body>
<p>个人微信公众号：yoyoketang</p>
<p>这是一个内嵌的 div 滚动条</p>
<div id="yoyoketang" name="yoyo" class="scroll">这是一个内嵌 div:民国年间，九大家族镇守长沙，被称为“九门提督”。这九门势力庞大，外八行的无人不知，无人不 晓，几乎所有冥器，流出长沙必然经过其中一家。
1933 年秋，一辆神秘鬼车缓缓驶入长沙火车站，九门之首“张大佛爷”张启山身为布防官，奉命调查始末。张启山与八爷齐铁嘴一路探访，发现长沙城外有一座疑点重重的矿山，一直被日本人窥伺。
为 破解矿山之谜，张启山求助同为九门上三门的戏曲名伶二月红，无奈二月红虽出身考古世家，却心系重病的妻子丫头，早已金盆洗手。张启山为了国家大义和手足之 情，北上去往新月饭店为二月红爱妻求药。在北平，张启山邂逅了新月饭店的大小姐尹新月，并为尹新月连点三盏天灯，散尽家财。尹新月帮助张启山等人顺利返回 长沙，二人暗生情愫。二月红爱妻病入膏肓，服药后不见好转，最终故去。
二月红悲伤之余却意外发现家族祖辈与矿山亦有重大关联，于是振作精神，决定与张启山联手，解开矿山之谜
zhegedancihenchanghenchangchangchangchanchanchanchangchangchangchancg</div>
```

```
</body>
</html>
```

2.24.2 纵向滚动

1.这个是 div 的属性：

2.这里最简单的通过 id 来定位，通过控制 scrollTop 的值来控制滚动条高度

3.运行下面代码，观察页面是不是先滚动到底部，过五秒再回到顶部。（get 里面地址是浏览器打开该页面的地址）

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/div.html")
# 纵向底部
js1 = 'document.getElementById("yoyoketang").scrollTop=10000'
driver.execute_script(js1)

time.sleep(5)
# 纵向顶部
js1 = 'document.getElementById("yoyoketang").scrollTop=0'
driver.execute_script(js1)
```

2.24.3 横向滚动

1.先通过 id 来定位，通过控制 scrollLeft 的值来控制滚动条高度

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/div.html")
# 横向右侧
js3 = 'document.getElementById("yoyoketang").scrollLeft=10000'
driver.execute_script(js3)

time.sleep(5)
# 横向左侧
js4 = 'document.getElementById("yoyoketang").scrollLeft=0'
driver.execute_script(js4)
```

2.24.4 用 class 属性定位

1. js 用 class 属性定位，返回的是一个 list 对象，这里取第一个就可以了。

2. 这里要注意了，element 和 elements 有很多小伙伴傻傻分不清楚。

```
# coding:utf-8
from selenium import webdriver
import time
driver = webdriver.Firefox()
driver.get("file:///C:/Users/Gloria/Desktop/div.html")

# 获取的class返回的是list对象，取list的第一个
js5 = 'document.getElementsByClassName("scroll")[0].scrollTop=10000'
driver.execute_script(js5)

time.sleep(5)
# 控制横向滚动条位置
js6 = 'document.getElementsByClassName("scroll")[0].scrollLeft=10000'
driver.execute_script(js6)
```

有时候很多元素属性都一样时候，就可以用复数定位，取对应的第几个就可以了

2.25 js 处理多窗口

前言

在打开页面上链接的时候，经常会弹出另外一个窗口（多窗口情况前面这篇有讲解：Selenium2+python 自动化 13-多窗口、句柄（handle））， 这样在多个窗口之间来回切换比较复杂，那么有没有办法让新打开的链接在一个窗口打开呢？

要解决这个问题，得从 html 源码上找到原因，然后修改元素属性才能解决。很显然 js 在这方面是万能的，于是本篇得依靠万能的 js 大哥了。

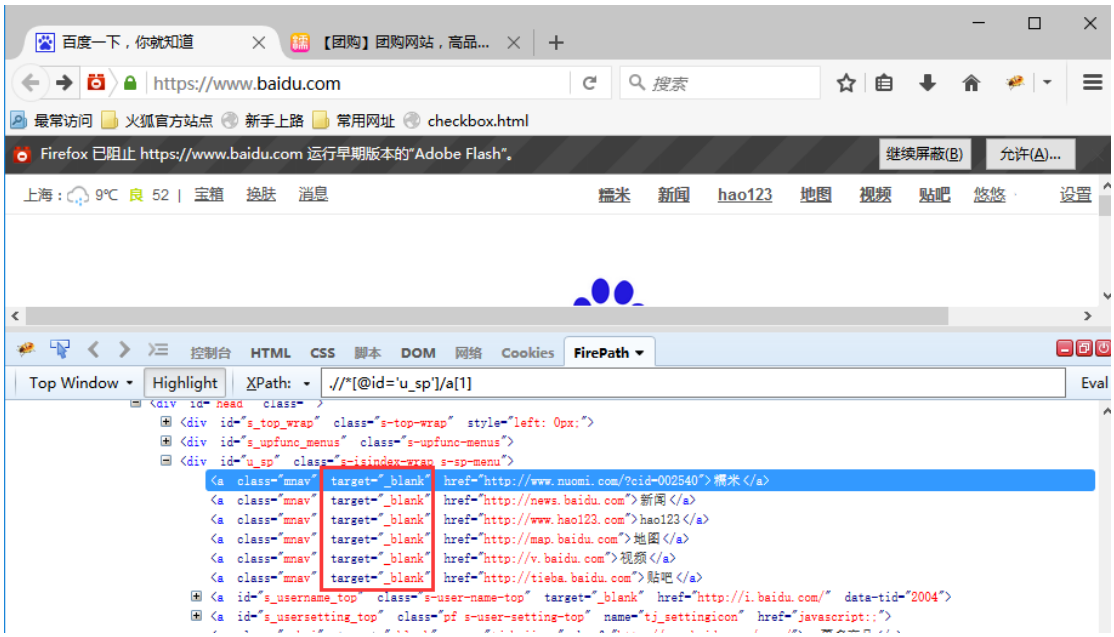
2.25.1 多窗口情况

1. 在打 baidu 的网站链接时，会重新打开一个窗口

（注意：我的百度页面是已登录状态，没登录时候是不会重新打开窗口的）

2.25.2 查看元素属性：target="_blank"

1. 查看元素属性，会发现这些链接有个共同属性：target="_blank"

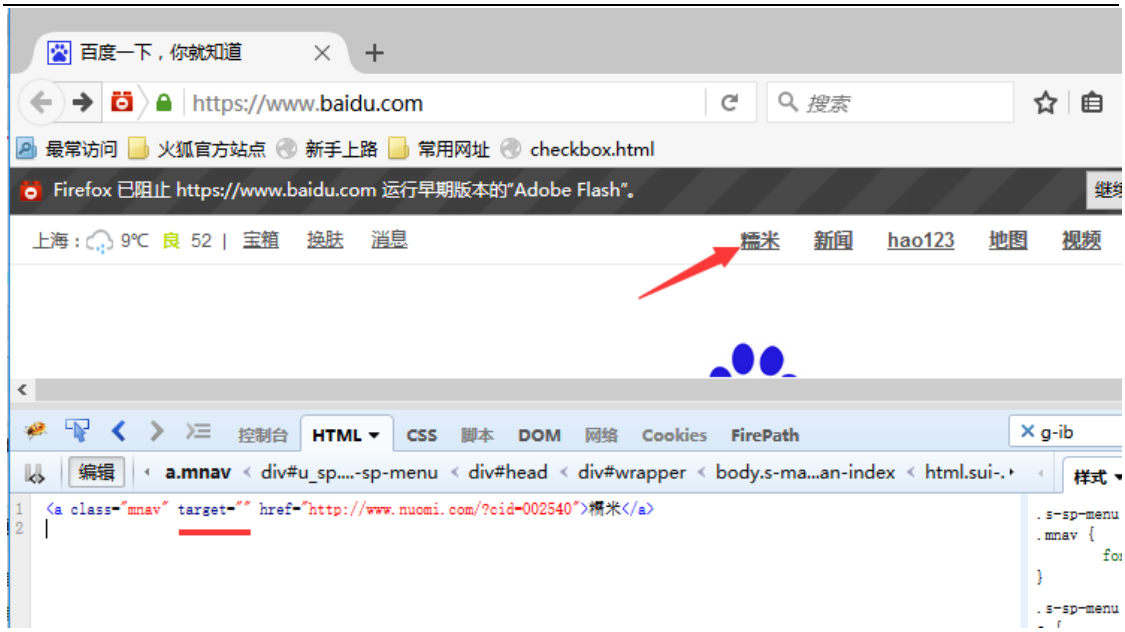


2.25.3 去掉 target="_blank"属性

1. 因为此链接元素 target="_blank"，所以打开链接的时候会重新打开一个标签页，那么解决这个问题，去掉该属性就可以了。

2. 为了验证这个问题，可以切换到 html 编辑界面，手动去掉“_blank”属性

3. 删除“_blank”属性后，重新打开链接，这时候会发现打开的新链接会在原标签页打开。



2.25.4 js 去掉 target="_blank"属性

1. 第一步为了先登录，我这里加载配置文件免登录了（不会的看这篇：Selenium2+python 自动化 18-加载 Firefox 配置）

2. 这里用到 js 的定位方法，定位该元素的 class 属性

3. 定位到该元素后直接修改 target 属性值为空

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

# 加载配置文件免登录
profileDir = r'C:\Users\Gloria\AppData\Roaming\Mozilla\Firefox\Profiles
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)

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

# 修改元素的target属性
js = 'document.getElementsByClassName("mnav")[0].target="";'
driver.execute_script(js)
driver.find_element_by_link_text("糯米").click()
```

2.25.5 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

# 加载配置文件免登录
profileDir =
r'C:\Users\Gloria\AppData\Roaming\Mozilla\Firefox\Profiles\1x41j9of.d
efault'
profile = webdriver.FirefoxProfile(profileDir)
driver = webdriver.Firefox(profile)

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

# 修改元素的 target 属性
js = 'document.getElementsByClassName("mnav")[0].target="";'
driver.execute_script(js)
driver.find_element_by_link_text("糯米").click()
```

注意：并不是所有的链接都适用于本方法，本篇只适用于有这个 target="_blank"属性链接情况

本篇仅提供解决问题的办法和思路，不要完全照搬代码！！！

2.26 js 解决 click 失效问题

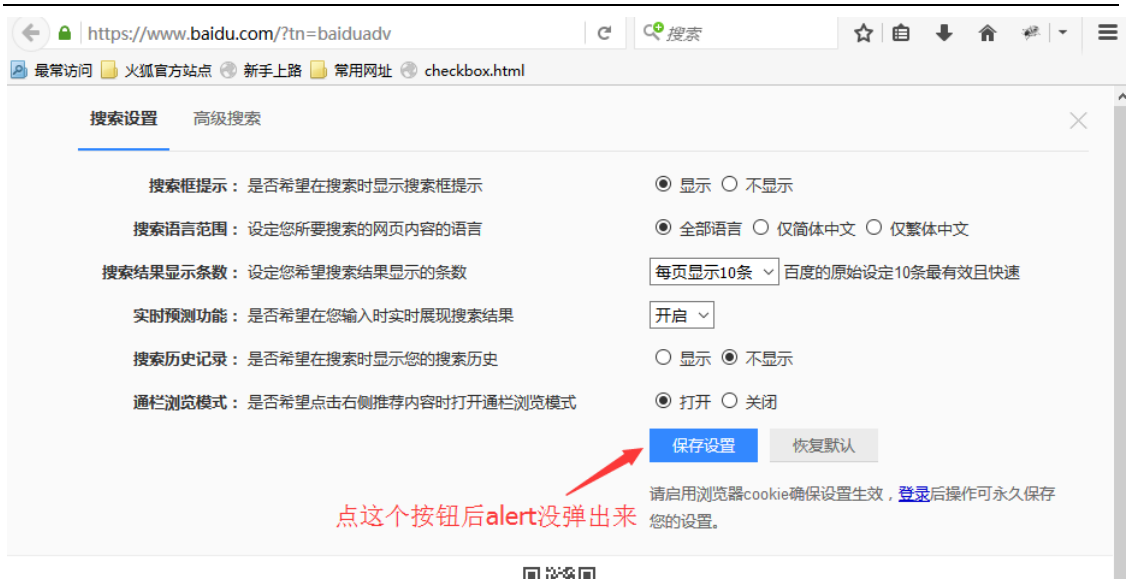
前言

有时候元素明明已经找到了，运行也没报错，点击后页面没任何反应。这种问题遇到了，是比较头疼的，因为没任何报错，只是 click 事件失效了。

本篇用 2 种方法解决这种诡异的点击事件失效问题

2.26.1 遇到的问题

1.在练习百度的搜索设置按钮时，点保存设置按钮，alert 弹出没弹出（代码没报错，只是获取 alert 失败），相信不只是我一个人遇到过。



2.26.2 点击父元素

1. 遇到这种问题，应该是前面操作 select 后导致的后遗症（因为我注释掉 select 那段是可以点击成功的）

2. 第一种解决办法，先点击它的父元素一次，然后再点击这个元素

3. 实现代码如下

```
mouse = driver.find_element("link text", "设置")
ActionChains(driver).move_to_element(mouse).perform()
time.sleep(3)
driver.find_element("link text", "搜索设置").click()
time.sleep(3)
s = driver.find_element("id", "nr")
Select(s).select_by_visible_text("每页显示50条")

# 先点父元素 交流QQ群：232607095
driver.find_element("id", "gxszButton").click()
driver.find_element("class name", "prefpanelgo").click()
```

2.26.3 js 直接点击

1.遇到这种诡异问题，是时候出绝招了：js 大法

2.用 js 直接执行点击事件

```
# 方法一：先点父元素 交流QQ群：232607095
# driver.find_element("id", "gxszButton").click()
# driver.find_element("class name", "prefpanelgo").click()

# 方法二：用js直接去点击 交流QQ群：232607095
js = 'document.getElementsByClassName("prefpanelgo")[0].click();'
driver.execute_script(js)
```

2.26.4 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.select import Select
import time
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
time.sleep(3)
mouse = driver.find_element("link text", "设置")
```

```
ActionChains(driver).move_to_element(mouse).perform()
time.sleep(3)
driver.find_element("link text", "搜索设置").click()
time.sleep(3)
s = driver.find_element("id", "nr")
Select(s).select_by_visible_text("每页显示 50 条")

# 方法一：先点父元素 交流 QQ 群：232607095
# driver.find_element("id", "gxszButton").click()
# driver.find_element("class name", "prefpanelgo").click()

# 方法二：用 js 直接去点击 交流 QQ 群：232607095
js = 'document.getElementsByClassName("prefpanelgo")[0].click();'
driver.execute_script(js)
```

2.27 查看 webdriver API

前言

前面都是点点滴滴的介绍 selenium 的一些 api 使用方法，那么 selenium 的 api 到底有多少呢？本篇就叫大家如何去查看 selenium api，不求人，无需伸手找人要，在自己电脑就有。

pydoc 是 Python 自带的模块，主要用于从 python 模块中自动生成文档，这些文档可以基于文本呈现的、也可以生成 WEB 页面的，还可以在服务器上以浏览器的方式呈现！

2.7.1 pydoc

1.到底什么是 pydoc?,这个是准确的解释：Documentation generator and online help system. pydoc 是 Python 自带的模块，主要用于从 python 模块中自动生成文档，这些文档可以基于文本呈现的、也可以生成 WEB 页面的，还可以在服务器上以浏览器的方式呈现！简而言之，就是帮你从代码和注释自动生成文档的工具。

2.举个栗子，我需要查看 python 里面 open 函数的功能和语法，打开 cmd,输入:python -m pydoc open

3.-m 参数：python 以脚本方法运行模块

```
>>python -m pydoc open
```

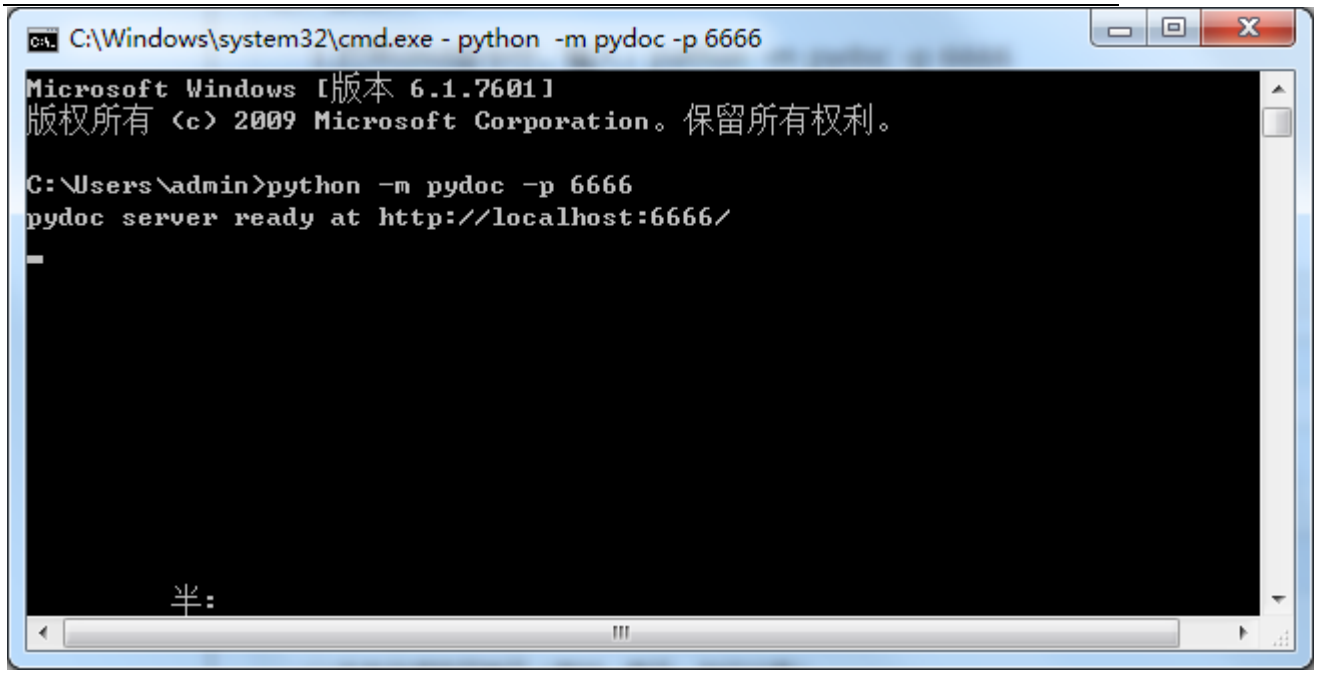
那么问题来了，这个是已经知道有这个函数，去查看它的功能，selenium 里面不知道到底有多少个函数或方法，那如何查看呢？

2.7.2 启动 server

1. 打开 cmd 命令行，输入：python -m pydoc -p 6666

2. -p 参数：这个表示在本机上启动服务

3. 6666 参数：这个是服务端口号，随意设置

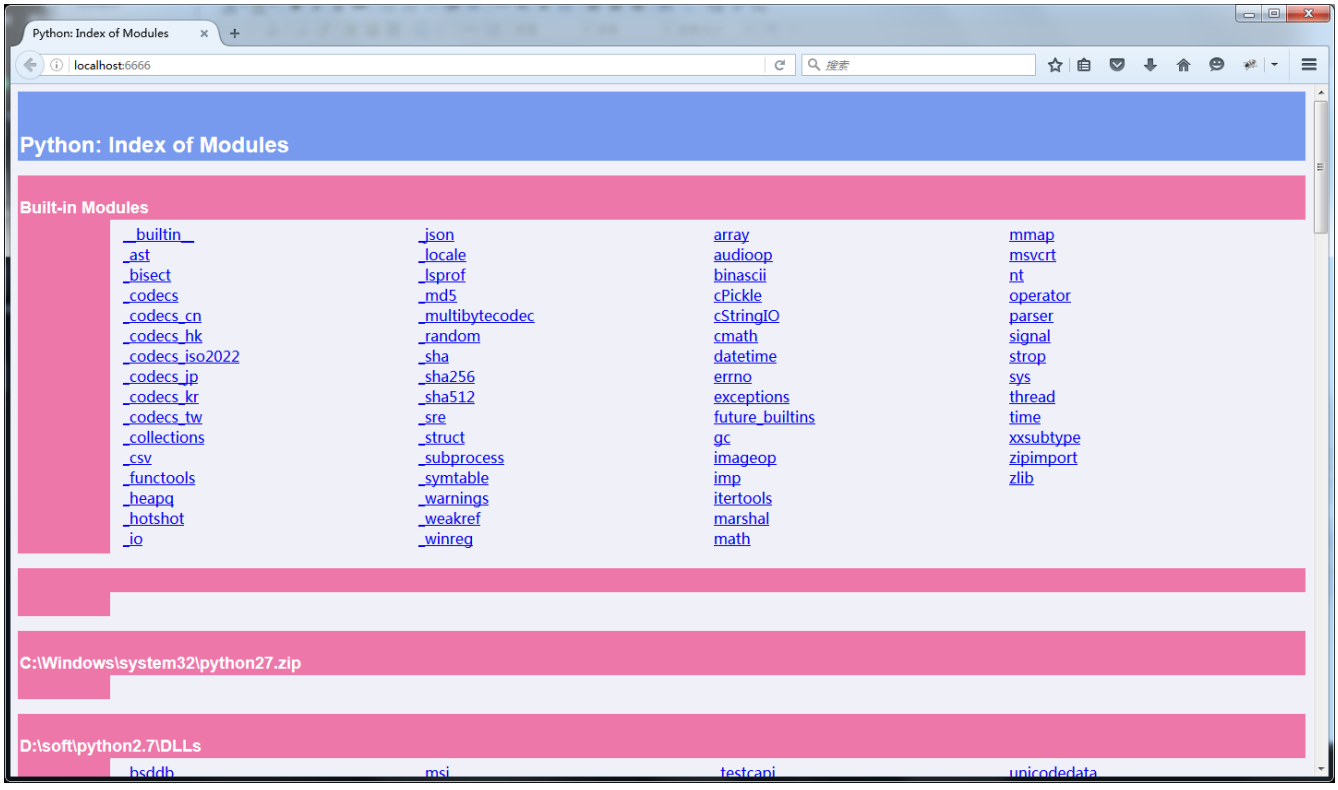


打开后，界面会出现一个地址：http://localhost:6666/,在浏览器直接打开

2.7.3 浏览器查看文档

1. 在浏览器输入：http://localhost:6666/

2. Built-in Moudles ：这个是 python 自带的模块

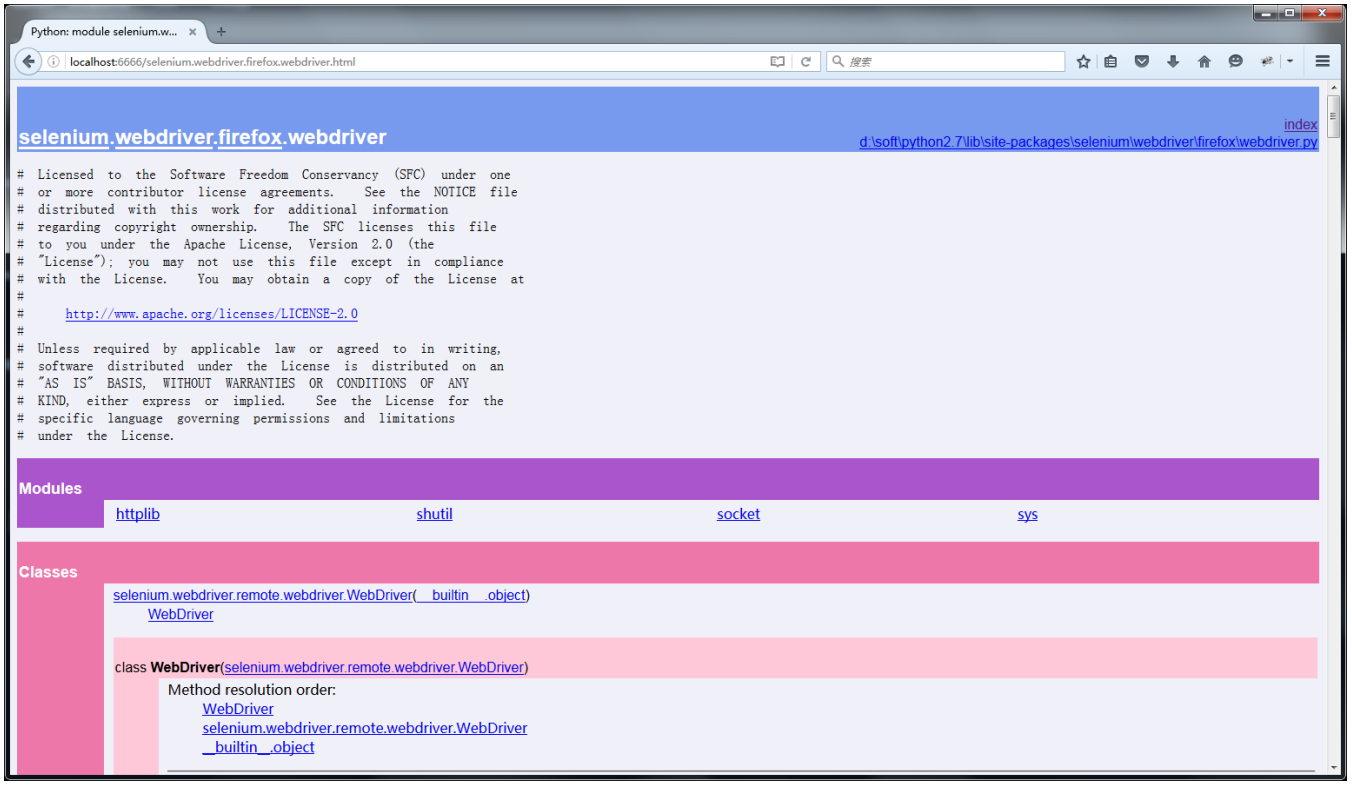


2.7.4 webdriver API（带翻译）

1. 找到这个路径：python2.7\lib\site-packages，点开 selenium

2. 打开的 selenium>webdriver>firefox>webdriver,最终路径：http://localhost:6666/selenium.webdriver.firefox.webdriver.html

3. 最终看到的这些就是 selenium 的 webdriver API 帮助文档啦



1.add_cookie(self, cookie_dict)

--翻译：添加 cookie,cookie 参数为字典数据类型
Adds a cookie to your current session.

:Args:
- cookie_dict: A dictionary object, with required keys - "name" and "value";
optional keys - "path", "domain", "secure", "expiry"

Usage:

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

2.back(self)

--浏览器返回
Goes one step backward in the browser history.

:Usage:
driver.back()

3.close(self)

--关闭浏览器
Closes the current window.

:Usage:
driver.close()

4.create_web_element(self, element_id)

--给元素分配一个 id
Creates a web element with the specified element_id.

5.delete_all_cookies(self)

--删除所有的 cookies
Delete all cookies in the scope of the session.

:Usage:
driver.delete_all_cookies()

6.delete_cookie(self, name)

--删除指定 name 的 cookie
Deletes a single cookie with the given name.

:Usage:
driver.delete_cookie('my_cookie')

7.execute(self, driver_command, params=None)

Sends a command to be executed by a command.CommandExecutor.

:Args:
- driver_command: The name of the command to execute as a string.
- params: A dictionary of named parameters to send with the command.

:Returns:
The command's JSON response loaded into a dictionary object.

8.execute_async_script(self, script, *args)
Asynchronously Executes JavaScript in the current window/frame.

:Args:
- script: The JavaScript to execute.
- *args: Any applicable arguments for your JavaScript.

:Usage:
driver.execute_async_script('document.title')

9.execute_script(self, script, *args)

--执行 JS
Synchronously Executes JavaScript in the current window/frame.

:Args:
- script: The JavaScript to execute.
- *args: Any applicable arguments for your JavaScript.

:Usage:
driver.execute_script('document.title')

10.file_detector_context(*args, **kwds)
Overrides the current file detector (if necessary) in limited context.
Ensures the original file detector is set afterwards.

Example:

with webdriver.file_detector_context(UselessFileDetector):
someinput.send_keys('/etc/hosts')

:Args:
- file_detector_class – Class of the desired file detector. If the class is different

from the current file_detector, then the class is instantiated with args and kwargs
and used as a file detector during the duration of the context manager.
- args - Optional arguments that get passed to the file detector class during
instantiation.
- kwargs - Keyword arguments, passed the same way as args.

11.find_element(self, by='id', value=None)

--定位元素，参数化的方法
'Private' method used by the find_element_by_* methods.

:Usage:
Use the corresponding find_element_by_* instead of this.

:rtype: WebElement

12.find_element_by_class_name(self, name)

--通过 class 属性定位元素
Finds an element by class name.

:Args:
- name: The class name of the element to find.

:Usage:
driver.find_element_by_class_name('foo')

13.find_element_by_css_selector(self, css_selector)

--通过 css 定位元素
Finds an element by css selector.

:Args:
- css_selector: The css selector to use when finding elements.

:Usage:
driver.find_element_by_css_selector('#foo')

14.find_element_by_id(self, id_)

--通过 id 定位元素
Finds an element by id.

:Args:
- id_ - The id of the element to be found.

:Usage:
driver.find_element_by_id('foo')

15.find_element_by_link_text(self, link_text)

--通过 link 链接定位
Finds an element by link text.

:Args:
- link_text: The text of the element to be found.

:Usage:
driver.find_element_by_link_text('Sign In')

16.find_element_by_name(self, name)

--通过 name 属性定位
Finds an element by name.

:Args:
- name: The name of the element to find.

:Usage:
driver.find_element_by_name('foo')

17.find_element_by_partial_link_text(self, link_text)

--通过部分 link 的模糊定位
Finds an element by a partial match of its link text.

:Args:
- link_text: The text of the element to partially match on.

:Usage:
driver.find_element_by_partial_link_text('Sign')

18.find_element_by_tag_name(self, name)

--通过标签定位
Finds an element by tag name.

:Args:

- name: The tag name of the element to find.

:Usage:
driver.find_element_by_tag_name('foo')

19.find_element_by_xpath(self, xpath)

--通过 xpath 语法定位
Finds an element by xpath.

:Args:
- xpath - The xpath locator of the element to find.

:Usage:
driver.find_element_by_xpath('//div/td[1]')

20.find_elements(self, by='id', value=None)

--定位一组元素
'Private' method used by the find_elements_by_* methods.

:Usage:
Use the corresponding find_elements_by_* instead of this.

:rtype: list of WebElement

21.find_elements_by_class_name(self, name)
Finds elements by class name.

:Args:
- name: The class name of the elements to find.

:Usage:
driver.find_elements_by_class_name('foo')

22.find_elements_by_css_selector(self, css_selector)
Finds elements by css selector.

:Args:
- css_selector: The css selector to use when finding elements.

:Usage:
driver.find_elements_by_css_selector('.foo')

23.find_elements_by_id(self, id_)
Finds multiple elements by id.

:Args:
- id_ - The id of the elements to be found.

:Usage:
driver.find_elements_by_id('foo')

24.find_elements_by_link_text(self, text)
Finds elements by link text.

:Args:
- link_text: The text of the elements to be found.

:Usage:
driver.find_elements_by_link_text('Sign In')

25.find_elements_by_name(self, name)
Finds elements by name.

:Args:
- name: The name of the elements to find.

:Usage:
driver.find_elements_by_name('foo')

26.find_elements_by_partial_link_text(self, link_text)
Finds elements by a partial match of their link text.

:Args:
- link_text: The text of the element to partial match on.

:Usage:
driver.find_element_by_partial_link_text('Sign')

27.find_elements_by_tag_name(self, name)
Finds elements by tag name.

:Args:
- name: The tag name the use when finding elements.

:Usage:
driver.find_elements_by_tag_name('foo')

28.find_elements_by_xpath(self, xpath)

Finds multiple elements by xpath.

:Args:
- xpath - The xpath locator of the elements to be found.

:Usage:
driver.find_elements_by_xpath("//div[contains(@class, 'foo')]")

29.forward(self)

--切换到下一页面
Goes one step forward in the browser history.

:Usage:
driver.forward()

30.get(self, url)

--打开 url 地址
Loads a web page in the current browser session.

31.get_cookie(self, name)

--获取指定名称的 cookie
Get a single cookie by name. Returns the cookie if found, None if not.

:Usage:
driver.get_cookie('my_cookie')

32.get_cookies(self)

--获取所有的 cookies
Returns a set of dictionaries, corresponding to cookies visible in the current session.

:Usage:
driver.get_cookies()

33.get_log(self, log_type)
Gets the log for a given log type

:Args:

- log_type: type of log that which will be returned

:Usage:
driver.get_log('browser')
driver.get_log('driver')
driver.get_log('client')
driver.get_log('server')

34.get_screenshot_as_base64(self)

--截图 base64 格式
Gets the screenshot of the current window as a base64 encoded string which is useful in embedded images in HTML.

:Usage:
driver.get_screenshot_as_base64()

35.get_screenshot_as_file(self, filename)

--截图保存为指定文件名称
Gets the screenshot of the current window. Returns False if there is any IOError, else returns True. Use full paths in your filename.

:Args:
- filename: The full path you wish to save your screenshot to.

:Usage:
driver.get_screenshot_as_file('/Screenshots/foo.png')

36.get_screenshot_as_png(self)

--截图为 png 格式二进制流
Gets the screenshot of the current window as a binary data.

:Usage:
driver.get_screenshot_as_png()

37.get_window_position(self, windowHandle='current')
Gets the x,y position of the current window.

:Usage:
driver.get_window_position()

38.get_window_size(self, windowHandle='current')

--获取窗口的宽高
Gets the width and height of the current window.

:Usage:
driver.get_window_size()

39.implicitly_wait(self, time_to_wait)

--隐式等待
Sets a sticky timeout to implicitly wait for an element to be found, or a command to complete. This method only needs to be called one time per session. To set the timeout for calls to execute_async_script, see set_script_timeout.

:Args:
- time_to_wait: Amount of time to wait (in seconds)

:Usage:
driver.implicitly_wait(30)

40.maximize_window(self)

--最大化窗口
Maximizes the current window that webdriver is using

41.refresh(self)

--刷新页面
Refreshes the current page.

:Usage:
driver.refresh()

save_screenshot = get_screenshot_as_file(self, filename)
Gets the screenshot of the current window. Returns False if there is any IOError, else returns True. Use full paths in your filename.

:Args:
- filename: The full path you wish to save your screenshot to.

:Usage:
driver.get_screenshot_as_file('/Screenshots/foo.png')

42.set_page_load_timeout(self, time_to_wait)

--设置页面加载超时时间
Set the amount of time to wait for a page load to complete
before throwing an error.

:Args:
- time_to_wait: The amount of time to wait

:Usage:
driver.set_page_load_timeout(30)

43.set_script_timeout(self, time_to_wait)
Set the amount of time that the script should wait during an
execute_async_script call before throwing an error.

:Args:
- time_to_wait: The amount of time to wait (in seconds)

:Usage:
driver.set_script_timeout(30)

44.set_window_position(self, x, y, windowHandle='current')
Sets the x,y position of the current window. (window.moveTo)

:Args:
- x: the x-coordinate in pixels to set the window position
- y: the y-coordinate in pixels to set the window position

:Usage:
driver.set_window_position(0,0)

45.set_window_size(self, width, height, windowHandle='current')

--设置窗口大小
Sets the width and height of the current window. (window.resizeTo)

:Args:
- width: the width in pixels to set the window to
- height: the height in pixels to set the window to

:Usage:
driver.set_window_size(800,600)

46.start_client(self)
Called before starting a new session. This method may be overridden
to define custom startup behavior.

start_session(self, desired_capabilities, browser_profile=None)
Creates a new session with the desired capabilities.

:Args:
- browser_name - The name of the browser to request.
- version - Which browser version to request.
- platform - Which platform to request the browser on.
- javascript_enabled - Whether the new session should support JavaScript.
- browser_profile - A
selenium.webdriver.firefox.firefox_profile.FirefoxProfile object. Only used if Firefox is requested.

47.stop_client(self)
Called after executing a quit command. This method may be overridden to define custom shutdown behavior.

48.switch_to_active_element(self)

--切换到活动的元素上，一般失去焦点时候会用到
Deprecated use driver.switch_to.active_element

49.switch_to_alert(self)

--切换到 alert 弹出框上
Deprecated use driver.switch_to.alert

50.switch_to_default_content(self)

--切换到默认的主页面上
Deprecated use driver.switch_to.default_content

51.switch_to_frame(self, frame_reference)

--切换 iframe
Deprecated use driver.switch_to.frame

52.switch_to_window(self, window_name)

--切换窗口
Deprecated use driver.switch_to.window

Data descriptors inherited from
selenium.webdriver.remote.webdriver.WebDriver:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

53.application_cache
Returns a ApplicationCache Object to interact with the browser app cache

54.current_url

--获取当前页面的 url 地址
Gets the URL of the current page.

:Usage:
driver.current_url

55.current_window_handle

--获取当前页面 handle
Returns the handle of the current window.

:Usage:
driver.current_window_handle

56.desired_capabilities
returns the drivers current desired capabilities being used

57.file_detector

58.log_types
Gets a list of the available log types

:Usage:
driver.log_types

59.mobile

60.name

--获取浏览器名称
Returns the name of the underlying browser for this instance.

:Usage:
- driver.name

61.orientation
Gets the current orientation of the device

:Usage:
orientation = driver.orientation

62.page_source

--获取页面源码
Gets the source of the current page.

:Usage:
driver.page_source

63.switch_to

--切换 iframe，handle 等方法

64.title

--获取页面 title
Returns the title of the current page.

:Usage:
driver.title

65.window_handles

--获取所有的 handle
Returns the handles of all windows within the current session.

:Usage:
driver.window_handles

小编后续有空再翻译下吧，英文水平有限。在学习过程中有遇到疑问的，可以加 selenium（python+java） QQ 群交流:232607095

第 3 章 unittest

3.1 unittest 简介

前言

熟悉 java 的应该都清楚常见的单元测试框架 Junit 和 TestNG，这个招聘的需求上也是经常见到的。python 里面也有单元测试框架-unittest,相当于是一个 python 版的 junit。

python 里面的单元测试框架除了 unittest,还有一个 pytest 框架，这个用的比较少，后面有空再继续分享。

3.1.1 unittest 简介

1）.先导入 unittest

2）.用 help 函数查看源码解析

3）.查看描述：

Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's Smalltalk testing framework.

翻译：python 的单元测试框架，是基于 java 的 junit 测试框架

```
import unittest
print help(unittest)
```

```
(1)

 Simple usage:

     import unittest

     class IntegerArithmeticTestCase(unittest.TestCase):
         def testAdd(self):  ## test method names begin 'test*'
             self.assertEqual((1 + 2), 3)
             self.assertEqual(0 + 1, 1)
         def testMultiply(self):
             self.assertEqual((0 * 10), 0)
             self.assertEqual((5 * 8), 40)

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

3.1.2 简单用法

1）.可以把上图的这段代码 copy 出来，单独运行，看看测试结果。

```
        Simple usage:

                import unittest

                class IntegerArithmeticTestCase(unittest.TestCase):
                        def testAdd(self):    ## test method names begin
'test*'
                                self.assertEqual((1 + 2), 3)
                                self.assertEqual(0 + 1, 1)
                        def testMultiply(self):
                                self.assertEqual((0 * 10), 0)
                                self.assertEqual((5 * 8), 40)

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

2）.第一行是导入 unittest 这个模块

3）.class 这一行是定义一个测试的类，并继承 unittest.TestCase 这个类

4）.接下来是定义了两个测试 case 名称:testAdd 和 testMultiply

5）.注释里面有句话很重要，这个要敲下黑板记笔记了：**## test method names begin 'test*'**

--**翻译：测试用例的名称要以 test 开头**

6）.然后是断言 assert，这里的断言方法是 assertEqual-判断两个是否相等，**这个断言可以是一个也可以是多个**

7）.if 下面的这个 unittest.main()是运行主函数，运行后会看到测试结果（跑了两个用例耗时 0.000 秒,两个用例都通过）：

```
..
----------------------------------------------------------------------
-
Ran 2 tests in 0.000s
```

OK

3.1.3 小试牛刀

1）.上面的两个案例是加法和乘法，我们可以写个 case 试下减法和除法。

2）.有很多小伙伴不知道断言怎么写，断言其实就是拿实际结果和期望结果去对比，对比的方法很多，这里只是举的最简单的一个判断相等的方法。

```
# coding:utf-8
import unittest

class Test(unittest.TestCase):

    def testMinus(self):    # test method names begin 'test*'
        u'''这里是测试减法'''
        result = 6-5  # 实际结果
        hope = 1      # 期望结果
        self.assertEqual(result, hope)

    def testDivide(self):
        u'''这里是测试除法'''
        result = 7/2    # 实际结果
        hope = 3.5      # 期望结果
        self.assertEqual(result, hope)

if __name__ == '__main__':
    unittest.main()  # 交流QQ群：232607095
```

3）.最后运行结果，第二个是失败的，失败原因：AssertionError: 3 != 3.5

```
F.
======================================================================
=
FAIL: testDivide (__main__.Test)
这里是测试除法
----------------------------------------------------------------------
-
Traceback (most recent call last):
   File "D:/test/web-project/p.py", line 14, in testDivide
       self.assertEqual(result, hope)
AssertionError: 3 != 3.5

----------------------------------------------------------------------
```

```
-
Ran 2 tests in 0.000s

FAILED (failures=1)
```

3.1.4 前置和后置

1）.setUp：在写测试用例的时候，每次操作其实都是基于打开浏览器输入对应网址这些操作，这个就是执行用例的前置条件。

2）.tearDown：执行完用例后，为了不影响下一次用例的执行，一般有个数据还原的过程，这就是执行用例的后置条件。

3）.很多人执行完用例，都不去做数据还原，以致于下一个用例执行失败，这就是不喜欢擦屁股的事情，习惯不好。

4）.前置和后置都是非必要的条件，如果没有也可以写 pass

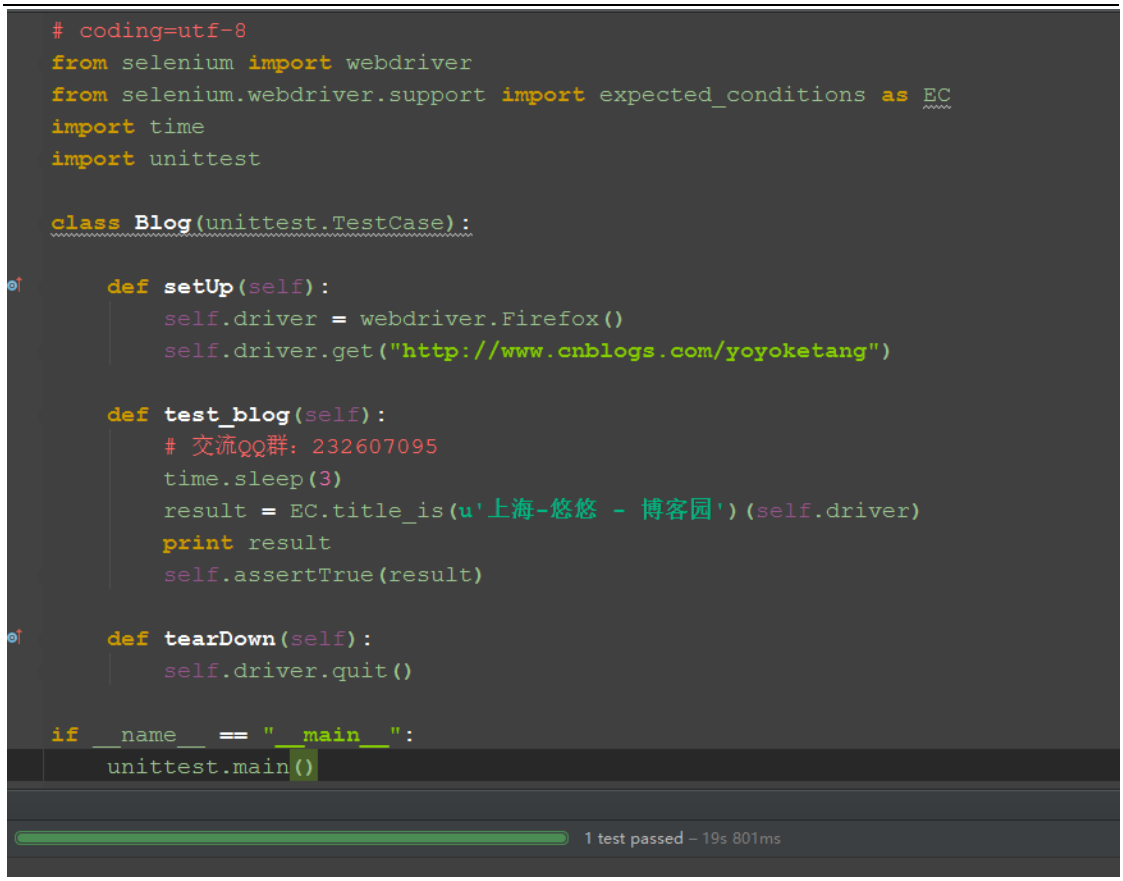
```python
# coding:utf-8
import unittest

class Test(unittest.TestCase):

    def setUp(self):
        pass  # 如果没有可以不写，或者pass代替

    def tearDown(self):
        pass

    def testMinus(self):    # test method names begin 'test*'
        u'''这里是测试减法'''
        result = 6-5  # 实际结果
        hope = 1      # 期望结果
        self.assertEqual(result, hope)
```

3.1.5 博客案例

1）.打开博客首页为例，写一个简单的 case

2）.判断 title 完全等于期望结果

3）.运行通过，下面会有一个绿条显示：1 test passed



3.1.6 参考代码

```
# coding=utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
import time
import unittest

class Blog(unittest.TestCase):

        def setUp(self):
                self.driver = webdriver.Firefox()
                self.driver.get("http://www.cnblogs.com/yoyoketang")

        def test_blog(self):
                # 交流 QQ 群：232607095
                time.sleep(3)
                result = EC.title_is(u'上海-悠悠 - 博客园')(self.driver)
                print result
```

```
                self.assertTrue(result)

        def tearDown(self):
                self.driver.quit()

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

3.2 unittest 执行顺序

前言

很多初学者在使用 unittest 框架时候，不清楚用例的执行顺序到底是怎样的。对测试类里面的类和方法分不清楚，不知道什么时候执行，什么时候不执行。

本篇通过最简单案例详细讲解 unittest 执行顺序。

3.2.1 案例分析

1）.先定义一个测试类，里面写几个简单的 case

```
# coding:utf-8
import unittest
import time
class Test(unittest.TestCase):
        def setUp(self):
                print "start!"

        def tearDown(self):
                time.sleep(1)
                print "end!"

        def test01(self):
                print "执行测试用例 01"

        def test03(self):
                print "执行测试用例 03"

        def test02(self):
                print "执行测试用例 02"

        def addtest(self):
                print "add 方法"
```

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

3.2.2 执行结果

```
D:\test\python2\python.exe D:/test/test01.py
start!
执行测试用例 01
.end!
start!
执行测试用例 02
end!
.start!
执行测试用例 03
end!
.
----------------------------------------------------------------------
-
Ran 3 tests in 3.001s

OK
```

3.2.3 结果分析

1）.执行顺序：

start!-执行测试用例 01-end!

start!-执行测试用例 02-end!

start!-执行测试用例 03-end!

2）.从执行结果可以看出几点

--先执行的前置 setUp，然后执行的用例(test*)，最后执行的后置 tearDown

--测试用例（test*）的执行顺序是根据 01-02-03 执行的，也就是说根据用例名称来顺序执行的

--addtest（self）这个方法没执行,说明只执行 test 开头的用例

3.2.4 selenium 实例

1）.具体实例参考这篇 Selenium2+python 自动化 48-登录方法（参数化）

```
# coding:utf-8
from selenium import webdriver
import unittest
import time
class Bolg(unittest.TestCase):
        u'''登录博客'''
        def setUp(self):
                self.driver = webdriver.Firefox()
                url = "https://passport.cnblogs.com/user/signin"
                self.driver.get(url)
                self.driver.implicitly_wait(30)

        def login(self, username, psw):
                u'''这里写了一个登录的方法,账号和密码参数化'''
                self.driver.find_element_by_id("input1").send_keys(use
rname)
                 self.driver.find_element_by_id("input2").send_keys(psw)
                self.driver.find_element_by_id("signin").click()
                time.sleep(3)

        def is_login_sucess(self):
                u'''判断是否获取到登录账户名称'''
                try:
                        text =
self.driver.find_element_by_id("lnk_current_user").text
                        print text
                        return True
                except:
                        return False

        def test_01(self):
                u'''登录案例参考:账号，密码自己设置'''
                self.login(u"上海-悠悠", u"xxxx")    # 调用登录方法
                # 判断结果
                result = self.is_login_sucess()
                self.assertTrue(result)

        def test_02(self):
                u'''登录案例参考:账号，密码自己设置'''
```

```
                self.login(u"上海-悠悠", u"xxxx")    # 调用登录方法
                # 判断结果       # 交流 QQ 群：232607095
                result = self.is_login_sucess()
                self.assertTrue(result)

        def tearDown(self):
                self.driver.quit()

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

3.3 unittest 批量执行

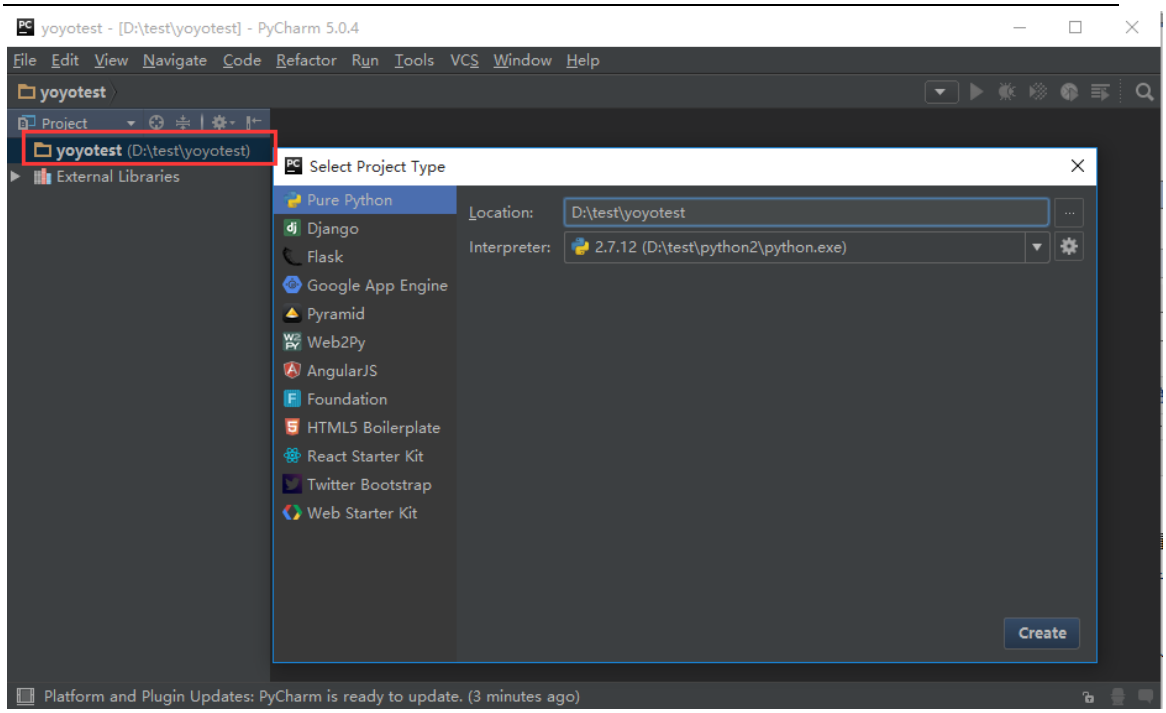
前言

我们在写用例的时候，单个脚本的用例好执行，那么多个脚本的时候，如何批量执行呢？这时候就需要用到 unittet 里面的 discover 方法来加载用例了。

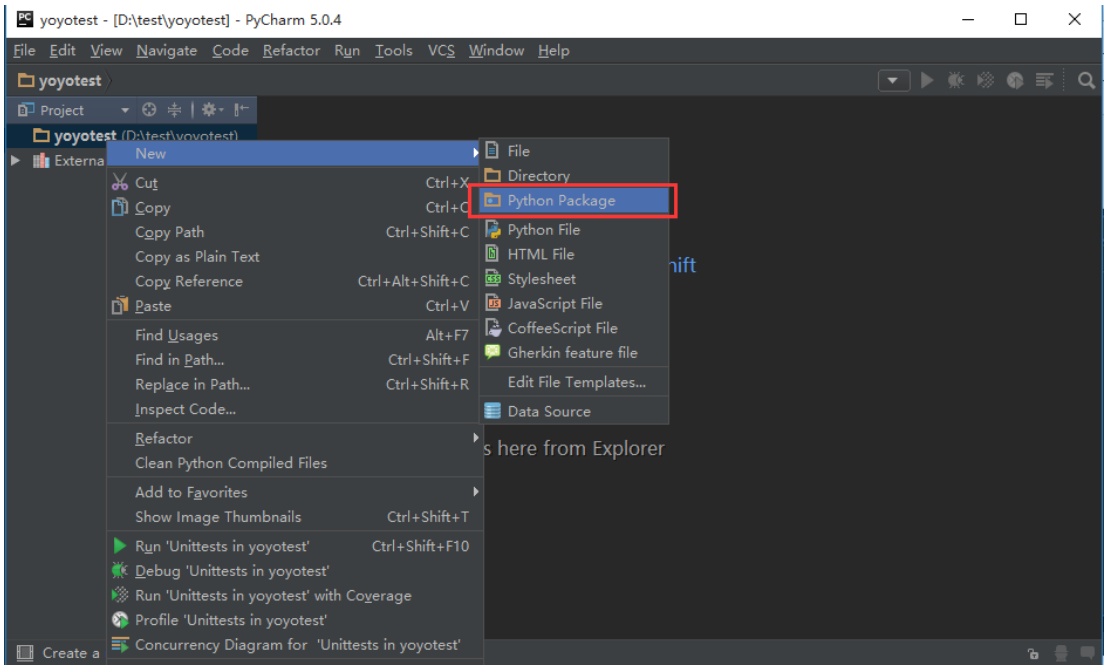
加载用例后，用 unittest 里面的 TextTestRunner 这里类的 run 方法去一次执行多个脚本的用例。

3.3.1 新建测试项目

1）.pycharm 左上角 File>New Projetc>Pure Python,在 location 位置命名一个测试工程的名称：yoyotest,然后保存

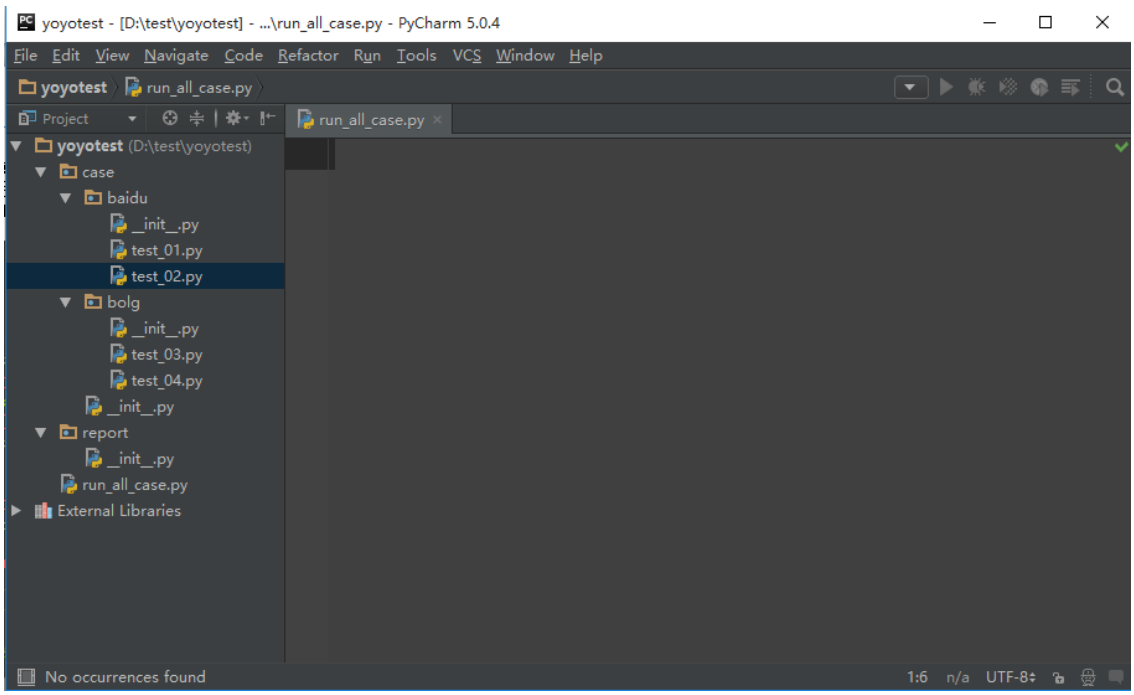


2）.选中刚才新建的工程右键>New>Python Package>新建一个 case 文件夹



3）.重复第 2 步的操作，新建一个 case 的文件夹，在里面添加一个 baidu 和一个 blog 的文件夹，里面分别有两个用例的脚本，如下图所示。

test_01, test_02, test_03, test_04 是我们写用例的脚本



4）.在 yoyotest 这个项目下面创建一个脚本 run_all_case.py，接下来用这个脚本去批量执行所有的用例。

3.3.2 diascover 加载测试用例

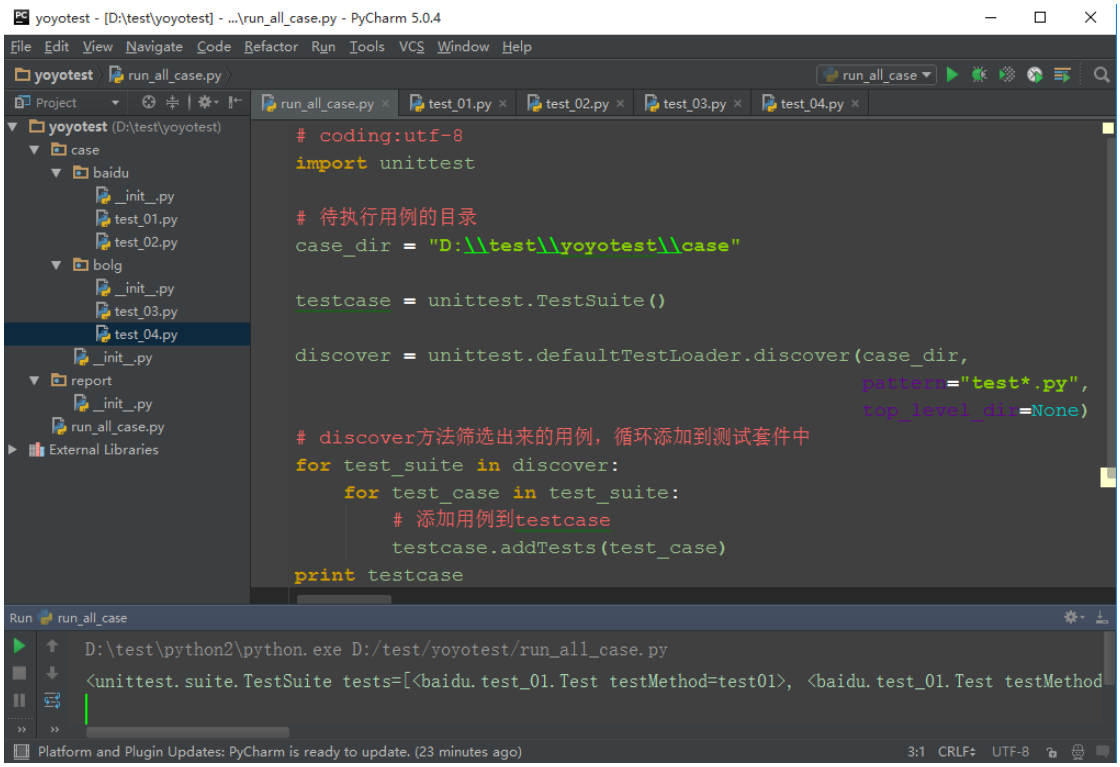
1）.discover 方法里面有三个参数：

-case_dir:这个是待执行用例的目录。

-pattern：这个是匹配脚本名称的规则，test*.py 意思是匹配 test 开头的所有脚本。

-top_level_dir：这个是顶层目录的名称，一般默认等于 None 就行了。

2）.discover 加载到的用例是一个 list 集合，需要重新写入到一个 list 对象 testcase 里，这样就可以用 unittest 里面的 TextTestRunner 这里类的 run 方法去执行。



3）.运行后结果入下，就是加载到的所有测试用例了：

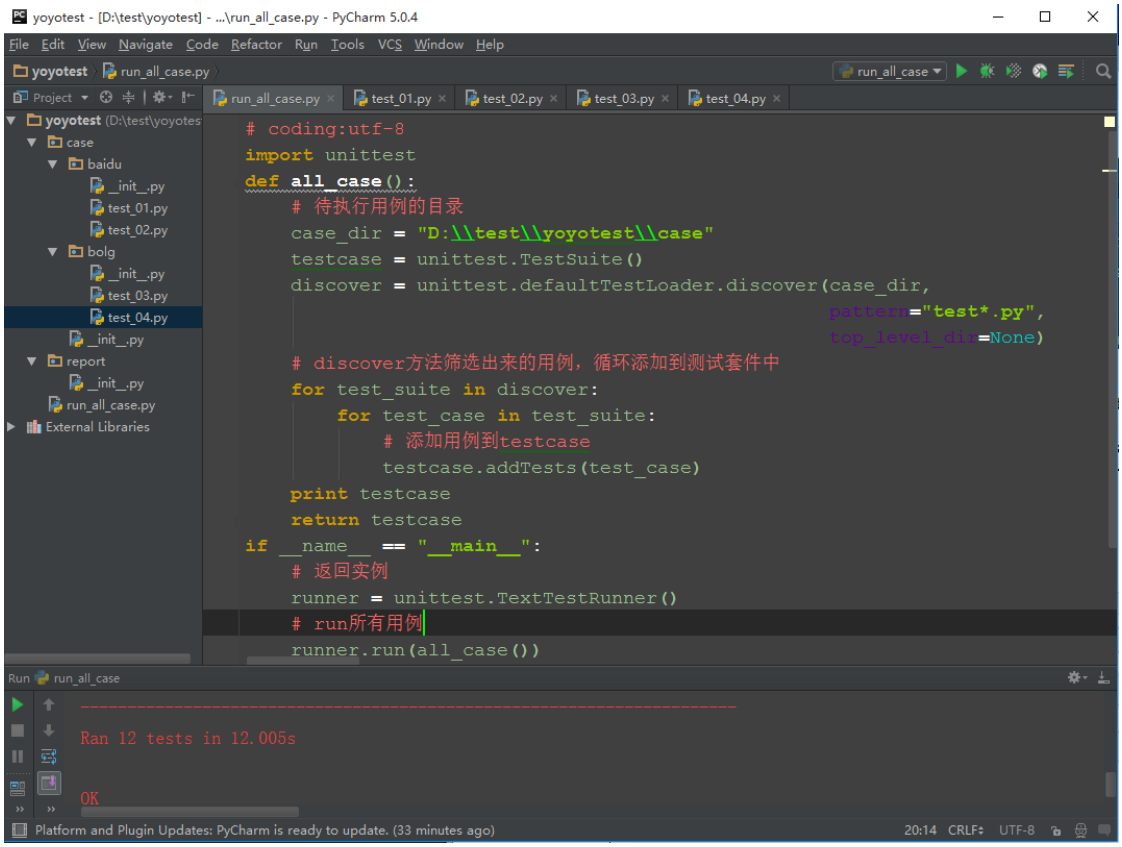
<unittest.suite.TestSuite tests=[<baidu.test_01.Test testMethod=test01>, <baidu.test_01.Test testMethod=test02>, <baidu.test_01.Test testMethod=test03>, <baidu.test_02.Test testMethod=test01>, <baidu.test_02.Test testMethod=test02>, <baidu.test_02.Test testMethod=test03>, <bolg.test_03.Test testMethod=test01>, <bolg.test_03.Test testMethod=test02>, <bolg.test_03.Test testMethod=test03>, <bolg.test_04.Test testMethod=test01>, <bolg.test_04.Test testMethod=test02>, <bolg.test_04.Test testMethod=test03>]>

3.3.3 run 测试用例

1）.为了更方便的理解，可以把上面 discover 加载用例的方法封装下，写成一个函数

2）.先返回 TextTestRunner()类的实例

3）.调用 run 方法去执行 all_case()这个函数



3.3.4 参考代码：

```
# coding:utf-8
import unittest


def all_case():
        # 待执行用例的目录
        case_dir = "D:\\test\\yoyotest\\case"


        testcase = unittest.TestSuite()
        discover = unittest.defaultTestLoader.discover(case_dir,

                                pattern="test*.py",

                                top_level_dir=None)
        # discover 方法筛选出来的用例，循环添加到测试套件中
        for test_suite in discover:
                for test_case in test_suite:
```

```
                    # 添加用例到 testcase
                    testcase.addTests(test_case)
        print testcase
        return testcase


if __name__ == "__main__":
        # 返回实例
        runner = unittest.TextTestRunner()
        # run 所有用例
        runner.run(all_case())
```

学习过程中有遇到疑问的，可以加 selenium（python+java） QQ 群交流:232607095

觉得对你有帮助，就在右下角点个赞吧，感谢支持！

3.4 生成 html 测试报告

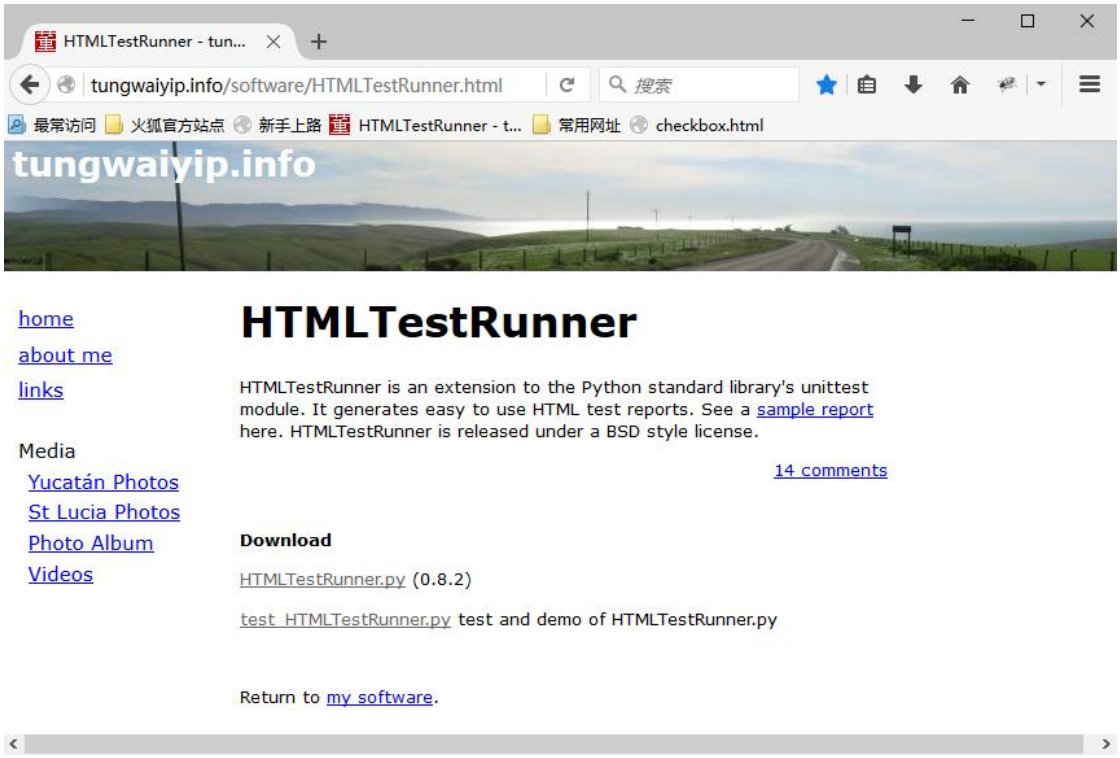
前言

批量执行完用例后，生成的测试报告是文本形式的，不够直观，为了更好的展示测试报告，最好是生成 HTML 格式的。

unittest 里面是不能生成 html 格式报告的，需要导入一个第三方的模块：HTMLTestRunner

3.4.1 导入 HTMLTestRunner

1）.这个模块下载不能通过 pip 安装了，只能下载后手动导入，下载地址：http://tungwaiyip.info/software/HTMLTestRunner.html



2）.Download 下 HTMLTestRunner.py 文件就是我们需要下载的包。

3）.下载后手动拖到 python 安装文件的 Lib 目录下

3.4.2 demo 解析

1）.下载 Download 下的第二个文件 test_HTMLTestRunner.py,这个就是官方给的一个测试 demo 了，从这个文件可以找到该模块的用法。

2）.找到下图这段，就是官方给的一个 demo 了，test_main()里上半部分就是加载测试 case,我们不需要搞这么复杂。

参考前面一篇内容就行了 Selenium2+python 自动化 53-unittest 批量执行（discover）

3）.最核心的代码是下面的红色区域，这个就是本篇的重点啦。

test_HTMLTestRunner.py

```
# This is the main test on HTMLTestRunner

class Test_HTMLTestRunner(unittest.TestCase):

    def test0(self):
        self.suite = unittest.TestSuite()
        buf = StringIO.StringIO()
        runner = HTMLTestRunner.HTMLTestRunner(buf)
        runner.run(self.suite)
        # didn't blow up? ok.
        self.assert_('</html>' in buf.getvalue())

    def test_main(self):
        # Run HTMLTestRunner. Verify the HTML report.

        # suite of TestCases
        self.suite = unittest.TestSuite()
        self.suite.addTests([
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTest0),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTest1),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestBasic),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestHTML),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestLatin1),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestUnicode),
            ])

        # Invoke TestRunner
        buf = StringIO.StringIO()
        #runner = unittest.TextTestRunner(buf)       #DEBUG: this is the unittest baseline
        runner = HTMLTestRunner.HTMLTestRunner(
                    stream=buf,
                    title='<Demo Test>',
                    description='This demonstrates the report output by HTMLTestRunner.'
                    )
        runner.run(self.suite)

        # Define the expected output sequence. This is imperfect but should
        # give a good sense of the well being of the test.
        EXPECTED = u"""
```

3.4.3 生成 html 报告

1）.我们只需把上面红色区域代码 copy 到上一篇的基础上稍做修改就可以了，这里主要有三个参数:

--stream:测试报告写入文件的存储区域

--title:测试报告的主题

--description：测试报告的描述

2）.report_path 是存放测试报告的地址

```
if __name__ == "__main__":
    # 返回实例
    # runner = unittest.TextTestRunner()
    import HTMLTestRunner
    report_path = "D:\\test\\yoyotest\\report\\result.html"

    fp = open(report_path, "wb")
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp,
                                           title=u'这是我的自动化测试报告',
                                           description=u'用例执行情况：')

    # run所有用例   交流QQ群：232607095
    runner.run(all_case())
    fp.close()
```

3.4.4 测试报告详情

1）.找到测试报告文件，用浏览器打开，点开 View 里的 Detail 可以查看详情描述。

这是我的自动化测试报告

Start Time: 2017-04-07 23:08:26
Duration: 0:00:12.042000
Status: Pass 12

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
baidu.test_01.Test	3	3	0	0	Detail
test01			pass		
test02			pass		
test03			pass		
baidu.test_02.Test	3	3	0	0	Detail
bolg.test_03.Test	3	3	0	0	Detail
bolg.test_04.Test	3	3	0	0	Detail
Total	**12**	**12**	**0**	**0**	

2）.为了生成带中文描述的测试用例，可以在 case 中添加注释，如在 test_01 的脚本添加如下注释：

```
class Test(unittest.TestCase):
        def setUp(self):
                print "start!"
```

```
    def tearDown(self):
        time.sleep(1)
        print "end!"

    def test01(self):
        u'''测试登录用例，账号：xx 密码 xx'''
        print "执行测试用例 01"

    def test03(self):
        u'''测试登搜索用例，关键词：xxx'''
        print "执行测试用例 03"
```

3）.重新运行后查看测试报告

这是我的自动化测试报告

Start Time: 2017-04-07 23:24:12
Duration: 0:00:12.013000
Status: Pass 12

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
baidu.test_01.Test	3	3	0	0	Detail
test01: 测试登录用例，账号：xx 密码xx			pass		
test02			pass		
test03: 测试登搜索用例，关键词：xxx			pass		
baidu.test_02.Test	3	3	0	0	Detail
bolg.test_03.Test	3	3	0	0	Detail
bolg.test_04.Test	3	3	0	0	Detail
Total	**12**	**12**	**0**	**0**	

3.4.5 参考代码：

```
# coding:utf-8
import unittest
import HTMLTestRunner

def all_case():
    # 待执行用例的目录
    case_dir = "D:\\test\\yoyotest\\case"
    testcase = unittest.TestSuite()
```

```
        discover = unittest.defaultTestLoader.discover(case_dir,

                                    pattern="test*.py",

                                    top_level_dir=None)
        # discover 方法筛选出来的用例，循环添加到测试套件中
        for test_suite in discover:
                for test_case in test_suite:
                        # 添加用例到 testcase
                        testcase.addTests(test_case)
        print testcase
        return testcase

if __name__ == "__main__":
        # 返回实例
        # runner = unittest.TextTestRunner()
        report_path = "D:\\test\\yoyotest\\report\\result.html"

        fp = open(report_path, "wb")
        runner = HTMLTestRunner.HTMLTestRunner(stream=fp,

                  title=u'这是我的自动化测试报告',

                  description=u'用例执行情况：')

        # run 所有用例    交流 QQ 群：232607095
        runner.run(all_case())
        fp.close()
```

学习过程中有遇到疑问的，可以加 selenium（python+java） QQ 群交流:232607095

觉得对你有帮助，就在右下角点个赞吧，感谢支持！

3.5 unitest 之装饰器

前言

前面讲到 unittest 里面 setUp 可以在每次执行用例前执行，这样有效的减少了代码量,但是有个弊端,比如打开浏览器操作,每次执行用例时候都会重新打开,这样就会浪费很多时间。

于是就想是不是可以只打开一次浏览器,执行完用例再关闭呢？这就需要用到装饰器（@classmethod）来解决了。

3.5.1 装饰器

1）.用 setUp 与 setUpClass 区别

setup():每个测试 case 运行前运行
teardown():每个测试 case 运行完后执行
setUpClass():必须使用@classmethod 装饰器,所有 case 运行前只运行一次
tearDownClass():必须使用@classmethod 装饰器,所有 case 运行完后只运行一次

2）.@是修饰符，classmethod 是 python 里的类方法

3.5.2 执行顺序

1）.用类方法写几个简单 case，可以对比这篇：Selenium2+python 自动化 52-unittest 执行顺序

```
# coding:utf-8
import unittest
import time
class Test(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
                print "start!"


        @classmethod
        def tearDownClass(cls):
                time.sleep(1)
                print "end!"

        def test01(self):
                print "执行测试用例 01"
```

```
        def test03(self):
                print "执行测试用例 03"

        def test02(self):
                print "执行测试用例 02"

        def addtest(self):
                print "add 方法"

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

2）.从执行结果可以看出，前置和后置在执行用例前只执行了一次。

```
start!
执行测试用例 01
执行测试用例 02
执行测试用例 03
...end!

----------------------------------------------------------------------
-
Ran 3 tests in 1.001s
```

3.5.3 selenium 实例

1.可以把打开浏览器操作放到前置 setUpClass(cls)里，这样就可以实现打开一次浏览器，执行多个 case 了

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
import unittest
class BolgHome(unittest.TestCase):
        u'''博客首页'''
        @classmethod
        def setUpClass(cls):
                cls.driver = webdriver.Firefox()
                url = "http://www.cnblogs.com/yoyoketang/"
                cls.driver.get(url)
                cls.driver.implicitly_wait(30)

        @classmethod
        def tearDownClass(cls):
```

```
                cls.driver.quit()

        def test_01(self):
                u'''验证元素存在：博客园'''
                locator = ("id", "blog_nav_sitehome")
                text = u"博客园"
                result = EC.text_to_be_present_in_element(locator,
text)(self.driver)
                self.assertTrue(result)

        def test_02(self):
                u'''验证元素存在：首页'''
                locator = ("id", "blog_nav_myhome")
                text = u"首页"
                result = EC.text_to_be_present_in_element(locator,
text)(self.driver)
                self.assertTrue(result)

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

学习过程中有遇到疑问的，可以加 selenium（python+java） QQ 群交流:232607095

觉得对你有帮助，就在右下角点个赞吧，感谢支持！

3.6 unittest 之断言

前言

在测试用例中,执行完测试用例后,最后一步是判断测试结果是 pass 还是 fail，自动化测试脚本里面一般把这种生成测试结果的方法称为断言（assert）。

用 unittest 组件测试用例的时候，断言的方法还是很多的，下面介绍几种常用的断言方法：assertEqual、assertIn、assertTrue

3.6.1 简单案例

1）.下面写了 4 个 case，其中第四个是执行失败的

```
# coding:utf-8
import unittest
class Test(unittest.TestCase):
	def test01(self):
		'''判断 a == b '''
		a = 1
		b = 1
		self.assertEqual(a, b)

	def test02(self):
		'''判断 a in b '''
		a = "hello"
		b = "hello world!"
		self.assertIn(a, b)

	def test03(self):
		'''判断 a is True '''
		a = True
		self.assertTrue(a)

	def test04(self):
		'''失败案例'''
		a = "上海-悠悠"
		b = "yoyo"
		self.assertEqual(a, b)

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

2）.执行结果如下

```
Failure
Expected :'\xe4\xb8\x8a\xe6\xb5\xb7-\xe6\x82\xa0\xe6\x82\xa0'
Actual     :'yoyo'
  <Click to see difference>

Traceback (most recent call last):
   File "D:\test\yoyotest\kecheng\test12.py", line 27, in test04
      self.assertEqual(a, b)
```

AssertionError: '\xe4\xb8\x8a\xe6\xb5\xb7-\xe6\x82\xa0\xe6\x82\xa0' != 'yoyo'

3.执行的结果，中文编码不对，没正常显示中文，遇到这种情况，可以自定义异常输出

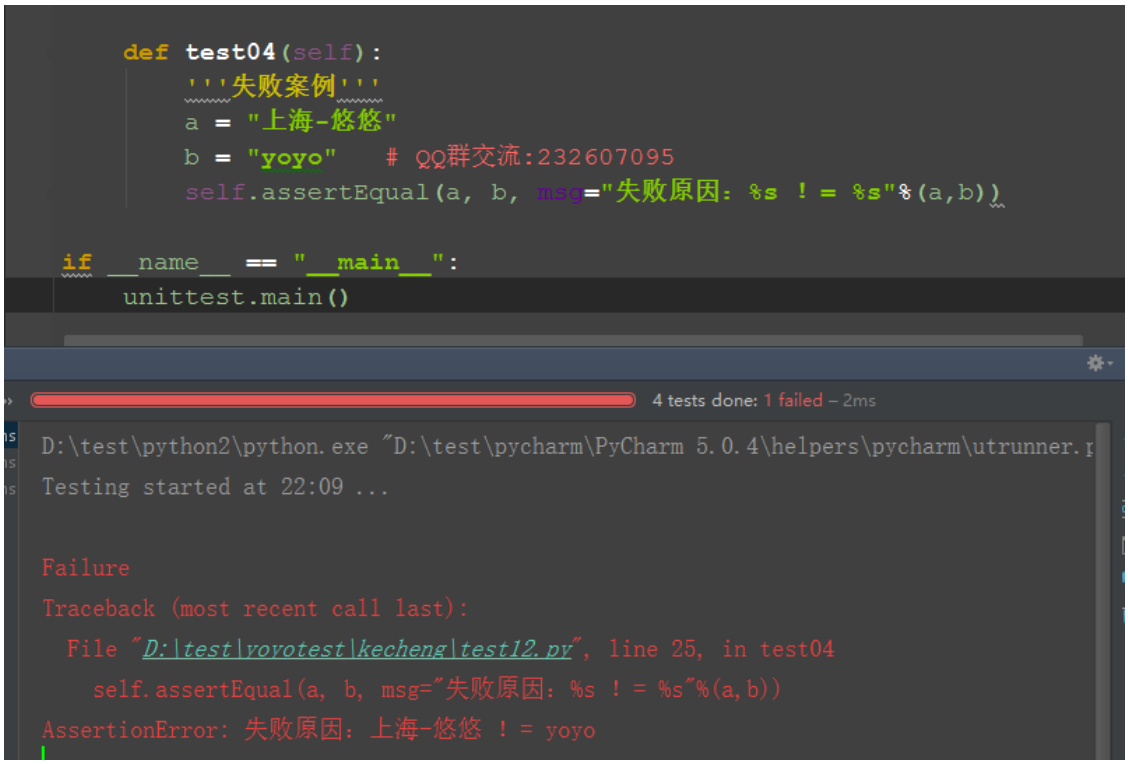
3.6.2 自定义异常

1）.以 assertEqual 为例分析：

assertEqual(self, first, second, msg=None)
Fail if the two objects are unequal as determined by the '==' operator.

2）.翻译：如果两个对象不能相等，就返回失败，相当于 return: first==second

3）.这里除了相比较的两个参数 first 和 second，还有第三个参数 msg=None，这个 msg 参数就是遇到异常后自定义输出信息



3.6.3 unittest 常用的断言方法

1）.assertEqual(self, first, second, msg=None)

--判断两个参数相等：first == second

2）.assertNotEqual(self, first, second, msg=None)

--判断两个参数不相等：first ！= second

3）.assertIn(self, member, container, msg=None)

--判断是字符串是否包含：member in container

4）.assertNotIn(self, member, container, msg=None)

--判断是字符串是否不包含：member not in container

5）.assertTrue(self, expr, msg=None)

--判断是否为真：expr is True

6）.assertFalse(self, expr, msg=None)

--判断是否为假：expr is False

7）.assertIsNone(self, obj, msg=None)

--判断是否为 None：obj is None

8）.assertIsNotNone(self, obj, msg=None)
--判断是否不为 None：obj is not None

3.6.4 unittest 所有断言方法

1）.下面是 unittest 框架支持的所有断言方法，有兴趣的同学可以慢慢看。

```
|   assertAlmostEqual(self, first, second, places=None, msg=None,
delta=None)
|           Fail if the two objects are unequal as determined by their
|           difference rounded to the given number of decimal places
```

```
|           (default 7) and comparing to zero, or by comparing that the
|           between the two objects is more than the given delta.
|
|           Note that decimal places (from zero) are usually not the same
|           as significant digits (measured from the most signficant
digit).
|
|           If the two objects compare equal then they will automatically
|           compare almost equal.
|
|   assertAlmostEquals = assertAlmostEqual(self, first, second,
places=None, msg=None, delta=None)
|
|   assertDictContainsSubset(self, expected, actual, msg=None)
|           Checks whether actual is a superset of expected.
|
|   assertDictEqual(self, d1, d2, msg=None)
|
|   assertEqual(self, first, second, msg=None)
|           Fail if the two objects are unequal as determined by the '=='
|           operator.
|
|   assertEquals = assertEqual(self, first, second, msg=None)
|
|   assertFalse(self, expr, msg=None)
|           Check that the expression is false.
|
|   assertGreater(self, a, b, msg=None)
|           Just like self.assertTrue(a > b), but with a nicer default
message.
|
|   assertGreaterEqual(self, a, b, msg=None)
|           Just like self.assertTrue(a >= b), but with a nicer default
message.
|
|   assertIn(self, member, container, msg=None)
|           Just like self.assertTrue(a in b), but with a nicer default
message.
|
|   assertIs(self, expr1, expr2, msg=None)
|           Just like self.assertTrue(a is b), but with a nicer default
message.
|
|   assertIsInstance(self, obj, cls, msg=None)
```

```
|           Same as self.assertTrue(isinstance(obj, cls)), with a nicer
|           default message.
|
|   assertIsNone(self, obj, msg=None)
|           Same as self.assertTrue(obj is None), with a nicer default
message.
|
|   assertIsNot(self, expr1, expr2, msg=None)
|           Just like self.assertTrue(a is not b), but with a nicer
default message.
|
|   assertIsNotNone(self, obj, msg=None)
|           Included for symmetry with assertIsNone.
|
|   assertItemsEqual(self, expected_seq, actual_seq, msg=None)
|           An unordered sequence specific comparison. It asserts that
|           actual_seq and expected_seq have the same element counts.
|           Equivalent to::
|
|                   self.assertEqual(Counter(iter(actual_seq)),
|                                                   Counter(iter(ex
pected_seq)))
|
|           Asserts that each element has the same count in both sequences.
|           Example:
|                   - [0, 1, 1] and [1, 0, 1] compare equal.
|                   - [0, 0, 1] and [0, 1] compare unequal.
|
|   assertLess(self, a, b, msg=None)
|           Just like self.assertTrue(a < b), but with a nicer default
message.
|
|   assertLessEqual(self, a, b, msg=None)
|           Just like self.assertTrue(a <= b), but with a nicer default
message.
|
|   assertListEqual(self, list1, list2, msg=None)
|           A list-specific equality assertion.
|
|           Args:
|                   list1: The first list to compare.
|                   list2: The second list to compare.
|                   msg: Optional message to use on failure instead of
a list of
```

```
|                                       differences.
|
|   assertMultiLineEqual(self, first, second, msg=None)
|           Assert that two multi-line strings are equal.
|
|   assertNotAlmostEqual(self, first, second, places=None, msg=None,
delta=None)
|           Fail if the two objects are equal as determined by their
|           difference rounded to the given number of decimal places
|           (default 7) and comparing to zero, or by comparing that the
|           between the two objects is less than the given delta.
|
|           Note that decimal places (from zero) are usually not the same
|           as significant digits (measured from the most signficant
digit).
|
|           Objects that are equal automatically fail.
|
|   assertNotAlmostEquals = assertNotAlmostEqual(self, first, second,
places=None, msg=None, delta=None)
|
|   assertNotEqual(self, first, second, msg=None)
|           Fail if the two objects are equal as determined by the '!='
|           operator.
|
|   assertNotEquals = assertNotEqual(self, first, second, msg=None)
|
|   assertNotIn(self, member, container, msg=None)
|           Just like self.assertTrue(a not in b), but with a nicer
default message.
|
|   assertNotIsInstance(self, obj, cls, msg=None)
|           Included for symmetry with assertIsInstance.
|
|   assertNotRegexpMatches(self, text, unexpected_regexp, msg=None)
|           Fail the test if the text matches the regular expression.
|
|   assertRaises(self, excClass, callableObj=None, *args, **kwargs)
|           Fail unless an exception of class excClass is raised
|           by callableObj when invoked with arguments args and keyword
|           arguments kwargs. If a different type of exception is
|           raised, it will not be caught, and the test case will be
|           deemed to have suffered an error, exactly as for an
|           unexpected exception.
```

```
|
|           If called with callableObj omitted or None, will return a
|           context object used like this::
|
|                    with self.assertRaises(SomeException):
|                            do_something()
|
|           The context manager keeps a reference to the exception as
|           the 'exception' attribute. This allows you to inspect the
|           exception after the assertion::
|
|                   with self.assertRaises(SomeException) as cm:
|                           do_something()
|                   the_exception = cm.exception
|                   self.assertEqual(the_exception.error_code, 3)
|
|   assertRaisesRegexp(self, expected_exception, expected_regexp,
callable_obj=None, *args, **kwargs)
|           Asserts that the message in a raised exception matches a
regexp.
|
|           Args:
|                   expected_exception: Exception class expected to be
raised.
|                   expected_regexp: Regexp (re pattern object or string)
expected
|                                    to be found in error message.
|                   callable_obj: Function to be called.
|                   args: Extra args.
|                   kwargs: Extra kwargs.
|
|   assertRegexpMatches(self, text, expected_regexp, msg=None)
|           Fail the test unless the text matches the regular expression.
|
|   assertSequenceEqual(self, seq1, seq2, msg=None, seq_type=None)
|           An equality assertion for ordered sequences (like lists and
tuples).
|
|           For the purposes of this function, a valid ordered sequence
type is one
|           which can be indexed, has a length, and has an equality
operator.
|
|           Args:
```

```
|                   seq1: The first sequence to compare.
|                   seq2: The second sequence to compare.
|                   seq_type: The expected datatype of the sequences, or
None if no
|                                   datatype should be enforced.
|                   msg: Optional message to use on failure instead of
a list of
|                                   differences.
|
|   assertSetEqual(self, set1, set2, msg=None)
|           A set-specific equality assertion.
|
|           Args:
|                   set1: The first set to compare.
|                   set2: The second set to compare.
|                   msg: Optional message to use on failure instead of
a list of
|                                   differences.
|
|           assertSetEqual uses ducktyping to support different types of
sets, and
|           is optimized for sets specifically (parameters must support
a
|           difference method).
|
|   assertTrue(self, expr, msg=None)
|           Check that the expression is true.
|
|   assertTupleEqual(self, tuple1, tuple2, msg=None)
|           A tuple-specific equality assertion.
|
|           Args:
|                   tuple1: The first tuple to compare.
|                   tuple2: The second tuple to compare.
|                   msg: Optional message to use on failure instead of
a list of
|                                   differences.
```

3.7 简单项目设计

3.7.1 获取最新测试报告

3.7.2 发送最新报告

3.7.3 最终执行用例代码

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

第 4 章 场景判断

4.1 显式等待（WebDriverWait）

前言：

在脚本中加入太多的 sleep 后会影响脚本的执行速度，虽然 implicitly_wait() 这种方法隐式等待方法一定程度上节省了很多时间。

但是一旦页面上某些 js 无法加载出来（其实界面元素经出来了），左上角那个图标一直转圈，这时候会一直等待的。

4.1.1 参数解释

1.这里主要有三个参数：

class WebDriverWait(object):driver, timeout, poll_frequency

2.driver:返回浏览器的一个实例，这个不用多说

3.timeout：超时的总时长

4.poll_frequency：循环去查询的间隙时间，默认 0.5 秒

以下是源码的解释文档（案例一个是元素出现，一个是元素消失）

```
       def __init__(self, driver, timeout,
poll_frequency=POLL_FREQUENCY, ignored_exceptions=None):
               """Constructor, takes a WebDriver instance and timeout in
seconds.

                   :Args:
                     - driver - Instance of WebDriver (Ie, Firefox,
Chrome or Remote)
                     - timeout - Number of seconds before timing out
                     - poll_frequency - sleep interval between calls
                         By default, it is 0.5 second.
                     - ignored_exceptions - iterable structure of
exception classes ignored during calls.
                         By default, it contains
NoSuchElementException only.

                   Example:
                     from selenium.webdriver.support.ui import
WebDriverWait \n
                     element = WebDriverWait(driver,
```

```
10).until(lambda x: x.find_element_by_id("someId")) \n
                        is_disappeared = WebDriverWait(driver, 30, 1,
(ElementNotVisibleException)).\ \n
                                                    until_not(lambda x:
x.find_element_by_id("someId").is_displayed())
                """
```

4.1.2 元素出现：until()

1.until 里面有个 lambda 函数，这个语法看 python 文档吧

2.以百度输入框为例

```python
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait

driver = webdriver.Firefox()
driver.get("http://www.baidu.com")
# 等待时长10秒，默认0.5秒询问一次
WebDriverWait(driver, 10).until(lambda x: x.find_element_by_id("kw")).send_keys("yoyo")
```

4.1.3 元素消失：until_not（）

1.判断元素是否消失

```python
driver = webdriver.Firefox()
driver.get("http://www.baidu.com")
# 等待时长10秒，默认0.5秒询问一次
WebDriverWait(driver, 10).until(lambda x: x.find_element_by_id("kw")).send_keys("yoyo")

# 判断id为kw元素是否消失

is_disappeared = WebDriverWait(driver, 10, 1).\
    until_not(lambda x: x.find_element_by_id("kw").is_displayed())
print is_disappeared
```

4.1.4 参考代码：

```
# coding:utf-8
from selenium import webdriver
```

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

driver = webdriver.Firefox()
driver.get("http://www.baidu.com")
# 等待时长 10 秒，默认 0.5 秒询问一次
WebDriverWait(driver, 10).until(lambda x:
x.find_element_by_id("kw")).send_keys("yoyo")

# 判断 id 为 kw 元素是否消失
is_disappeared = WebDriverWait(driver, 10, 1).\
       until_not(lambda x: x.find_element_by_id("kw").is_displayed())
print is_disappeared
```

4.1.5 WebDriverWait 源码

1.WebDriverWait 主要提供了两个方法,一个是 until(),另外一个是 until_not()

以下是源码的注释，有兴趣的小伙伴可以看下

```
# Licensed to the Software Freedom Conservancy (SFC) under one
# or more contributor license agreements.   See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.   The SFC licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.   You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.   See the License for the
# specific language governing permissions and limitations
# under the License.

import time
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import TimeoutException

POLL_FREQUENCY = 0.5   # How long to sleep inbetween calls to the method
IGNORED_EXCEPTIONS = (NoSuchElementException,)   # exceptions ignored
during calls to the method
```

```
class WebDriverWait(object):
       def __init__(self, driver, timeout,
poll_frequency=POLL_FREQUENCY, ignored_exceptions=None):
               """Constructor, takes a WebDriver instance and timeout in
seconds.

                    :Args:
                      - driver - Instance of WebDriver (Ie, Firefox,
Chrome or Remote)
                      - timeout - Number of seconds before timing out
                      - poll_frequency - sleep interval between calls
                          By default, it is 0.5 second.
                      - ignored_exceptions - iterable structure of
exception classes ignored during calls.
                          By default, it contains
NoSuchElementException only.

                    Example:
                      from selenium.webdriver.support.ui import
WebDriverWait \n
                      element = WebDriverWait(driver,
10).until(lambda x: x.find_element_by_id("someId")) \n
                      is_disappeared = WebDriverWait(driver, 30, 1,
(ElementNotVisibleException)).\ \n
                                                  until_not(lambda x:
x.find_element_by_id("someId").is_displayed())
               """
               self._driver = driver
               self._timeout = timeout
               self._poll = poll_frequency
               # avoid the divide by zero
               if self._poll == 0:
                       self._poll = POLL_FREQUENCY
               exceptions = list(IGNORED_EXCEPTIONS)
               if ignored_exceptions is not None:
                       try:
                               exceptions.extend(iter(ignored_excepti
ons))
                       except TypeError:   # ignored_exceptions is not
iterable
                               exceptions.append(ignored_exceptions)
               self._ignored_exceptions = tuple(exceptions)
```

```
        def __repr__(self):
                return '<{0.__module__}.{0.__name__}
(session="{1}")>'.format(
                        type(self), self._driver.session_id)

        def until(self, method, message=''):
                """Calls the method provided with the driver as an
argument until the \
                return value is not False."""
                screen = None
                stacktrace = None

                end_time = time.time() + self._timeout
                while True:
                        try:
                                value = method(self._driver)
                                if value:
                                        return value
                        except self._ignored_exceptions as exc:
                                screen = getattr(exc, 'screen', None)
                                stacktrace = getattr(exc, 'stacktrace',
None)
                        time.sleep(self._poll)
                        if time.time() > end_time:
                                break
                raise TimeoutException(message, screen, stacktrace)

        def until_not(self, method, message=''):
                """Calls the method provided with the driver as an
argument until the \
                return value is False."""
                end_time = time.time() + self._timeout
                while True:
                        try:
                                value = method(self._driver)
                                if not value:
                                        return value
                        except self._ignored_exceptions:
                                return True
                        time.sleep(self._poll)
                        if time.time() > end_time:
                                break
```

```
raise TimeoutException(message)
```

4.2 判断元素（expected_conditions）

前言

经常有小伙伴问，如何判断一个元素是否存在，如何判断 alert 弹窗出来了，如何判断动态的元素等等一系列的判断，在 selenium 的 expected_conditions 模块收集了一系列的场景判断方法，这些方法是逢面试必考的！！！

expected_conditions 一般也简称 EC，本篇先介绍下有哪些功能，后续更新中会单个去介绍。

4.2.1 功能介绍和翻译

title_is： 判断当前页面的 title 是否完全等于（==）预期字符串，返回布尔值

title_contains ： 判断当前页面的 title 是否包含预期字符串，返回布尔值

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

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

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

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

text_to_be_present_in_element ： 判断某个元素中的 text 是否 包含 了预期的字符串

text_to_be_present_in_element_value ： 判断某个元素中的 value 属性是否包含 了预期的字符串

frame_to_be_available_and_switch_to_it ： 判断该 frame 是否可以 switch

进去，如果可以的话，返回 True 并且 switch 进去，否则返回 False

invisibility_of_element_located： 判断某个元素中是否不存在于 dom 树或不可见

element_to_be_clickable ： 判断某个元素中是否可见并且是 enable 的，这样的话才叫 clickable

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

element_to_be_selected ： 判断某个元素是否被选中了,一般用在下拉列表

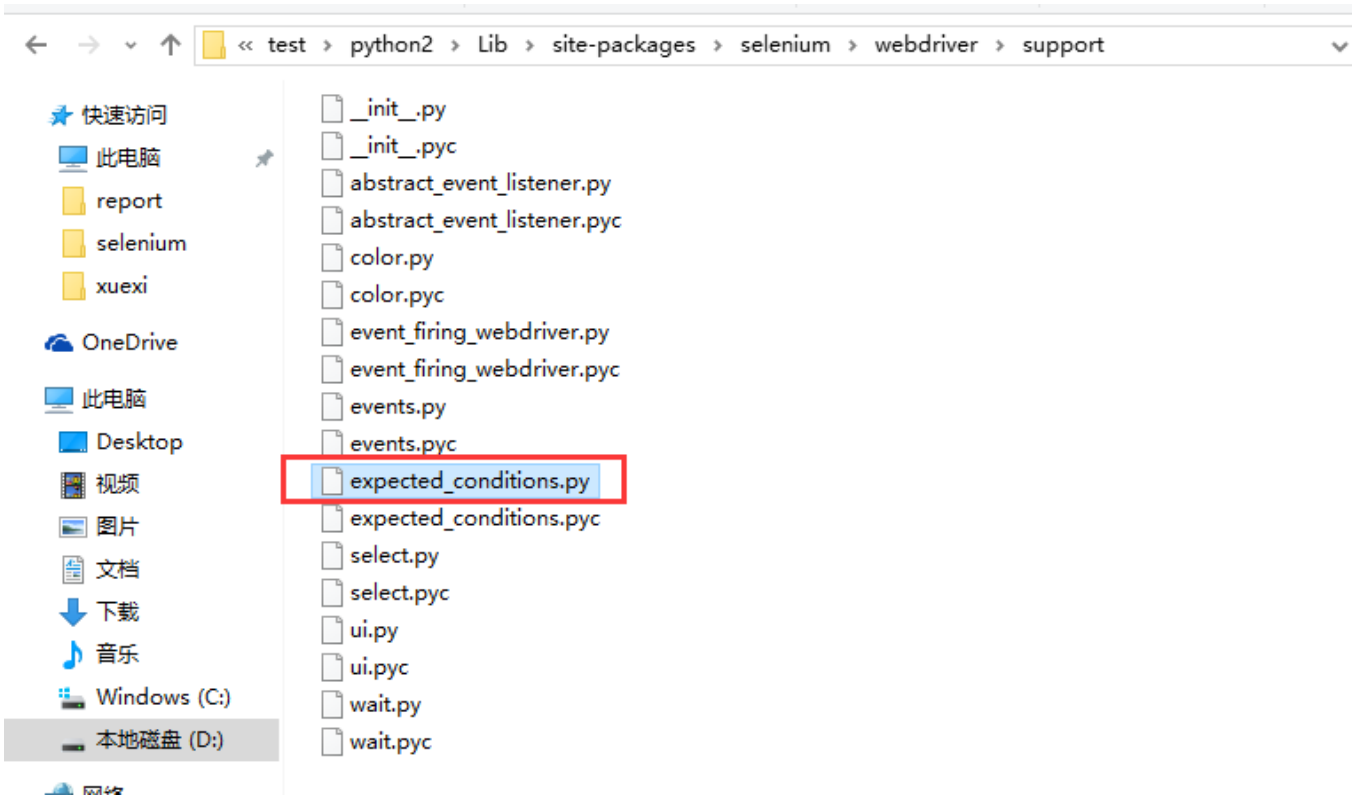
element_selection_state_to_be ： 判断某个元素的选中状态是否符合预期

element_located_selection_state_to_be： 跟上面的方法作用一样，只是上面的方法传入定位到的 element，而这个方法传入 locator

alert_is_present ： 判断页面上是否存在 alert

4.2.2 查看源码和注释

1.打开 python 里这个目录 1 可以找到：
Lib\site-packages\selenium\webdriver\support\expected_conditions.py




```
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoSuchFrameException
from selenium.common.exceptions import StaleElementReferenceException
from selenium.common.exceptions import WebDriverException
from selenium.common.exceptions import NoAlertPresentException

"""
  * Canned "Expected Conditions" which are generally useful within
webdriver
  * tests.
"""


class title_is(object):
        """An expectation for checking the title of a page.
        title is the expected title, which must be an exact match
        returns True if the title matches, false otherwise."""
        def __init__(self, title):
                self.title = title

        def __call__(self, driver):
                return self.title == driver.title
```

```
class title_contains(object):
        """ An expectation for checking that the title contains a
case-sensitive
        substring. title is the fragment of title expected
        returns True when the title matches, False otherwise
        """
        def __init__(self, title):
                self.title = title

        def __call__(self, driver):
                return self.title in driver.title



class presence_of_element_located(object):
        """ An expectation for checking that an element is present on the
DOM
        of a page. This does not necessarily mean that the element is
visible.
        locator - used to find the element
        returns the WebElement once it is located
        """
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                return _find_element(driver, self.locator)



class visibility_of_element_located(object):
        """ An expectation for checking that an element is present on the
DOM of a
        page and visible. Visibility means that the element is not only
displayed
        but also has a height and width that is greater than 0.
        locator - used to find the element
        returns the WebElement once it is located and visible
        """
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                try:
                        return
```

```
_element_if_visible(_find_element(driver, self.locator))
                except StaleElementReferenceException:
                        return False


class visibility_of(object):
        """ An expectation for checking that an element, known to be
present on the
        DOM of a page, is visible. Visibility means that the element is
not only
        displayed but also has a height and width that is greater than 0.
        element is the WebElement
        returns the (same) WebElement once it is visible
        """
        def __init__(self, element):
                self.element = element

        def __call__(self, ignored):
                return _element_if_visible(self.element)


def _element_if_visible(element, visibility=True):
        return element if element.is_displayed() == visibility else False


class presence_of_all_elements_located(object):
        """ An expectation for checking that there is at least one element
present
        on a web page.
        locator is used to find the element
        returns the list of WebElements once they are located
        """
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                return _find_elements(driver, self.locator)


class visibility_of_any_elements_located(object):
        """ An expectation for checking that there is at least one element
visible
        on a web page.
        locator is used to find the element
```

```
        returns the list of WebElements once they are located
        """
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                return [element for element in _find_elements(driver,
self.locator) if _element_if_visible(element)]


class text_to_be_present_in_element(object):
        """ An expectation for checking if the given text is present in
the
        specified element.
        locator, text
        """
        def __init__(self, locator, text_):
                self.locator = locator
                self.text = text_

        def __call__(self, driver):
                try:
                        element_text = _find_element(driver,
self.locator).text
                        return self.text in element_text
                except StaleElementReferenceException:
                        return False


class text_to_be_present_in_element_value(object):
        """
        An expectation for checking if the given text is present in the
element's
        locator, text
        """
        def __init__(self, locator, text_):
                self.locator = locator
                self.text = text_

        def __call__(self, driver):
                try:
                        element_text = _find_element(driver,

              self.locator).get_attribute("value")
```

```
                        if element_text:
                                return self.text in element_text
                        else:
                                return False
                except StaleElementReferenceException:
                                return False


class frame_to_be_available_and_switch_to_it(object):
        """ An expectation for checking whether the given frame is
available to
        switch to.    If the frame is available it switches the given driver
to the
        specified frame.
        """
        def __init__(self, locator):
                self.frame_locator = locator

        def __call__(self, driver):
                try:
                        if isinstance(self.frame_locator, tuple):
                                driver.switch_to.frame(_find_element(d
river,

                                                       self.frame_locator))
                        else:
                                driver.switch_to.frame(self.frame_loca
tor)
                        return True
                except NoSuchFrameException:
                        return False


class invisibility_of_element_located(object):
        """ An Expectation for checking that an element is either invisible
or not
        present on the DOM.

        locator used to find the element
        """
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
```

```
                try:
                        return
_element_if_visible(_find_element(driver, self.locator), False)
                except (NoSuchElementException,
StaleElementReferenceException):
                        # In the case of NoSuchElement, returns true
because the element is
                        # not present in DOM. The try block checks if the
element is present
                        # but is invisible.
                        # In the case of StaleElementReference, returns
true because stale
                        # element reference implies that element is no
longer visible.
                        return True


class element_to_be_clickable(object):
        """ An Expectation for checking an element is visible and enabled
such that
        you can click it."""
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                element =
visibility_of_element_located(self.locator)(driver)
                if element and element.is_enabled():
                        return element
                else:
                        return False


class staleness_of(object):
        """ Wait until an element is no longer attached to the DOM.
        element is the element to wait for.
        returns False if the element is still attached to the DOM, true
otherwise.
        """
        def __init__(self, element):
                self.element = element

        def __call__(self, ignored):
                try:
```

```
                        # Calling any method forces a staleness check
                        self.element.is_enabled()
                        return False
                except StaleElementReferenceException:
                        return True


class element_to_be_selected(object):
        """ An expectation for checking the selection is selected.
        element is WebElement object
        """
        def __init__(self, element):
                self.element = element

        def __call__(self, ignored):
                return self.element.is_selected()


class element_located_to_be_selected(object):
        """An expectation for the element to be located is selected.
        locator is a tuple of (by, path)"""
        def __init__(self, locator):
                self.locator = locator

        def __call__(self, driver):
                return _find_element(driver, self.locator).is_selected()


class element_selection_state_to_be(object):
        """ An expectation for checking if the given element is selected.
        element is WebElement object
        is_selected is a Boolean."
        """
        def __init__(self, element, is_selected):
                self.element = element
                self.is_selected = is_selected

        def __call__(self, ignored):
                return self.element.is_selected() == self.is_selected


class element_located_selection_state_to_be(object):
        """ An expectation to locate an element and check if the selection
state
```

```
        specified is in that state.
        locator is a tuple of (by, path)
        is_selected is a boolean
        """
        def __init__(self, locator, is_selected):
                self.locator = locator
                self.is_selected = is_selected

        def __call__(self, driver):
                try:
                        element = _find_element(driver, self.locator)
                        return element.is_selected() ==
self.is_selected
                except StaleElementReferenceException:
                        return False


class alert_is_present(object):
        """ Expect an alert to be present."""
        def __init__(self):
                pass

        def __call__(self, driver):
                try:
                        alert = driver.switch_to.alert
                        alert.text
                        return alert
                except NoAlertPresentException:
                        return False


def _find_element(driver, by):
        """Looks up an element. Logs and re-raises ``WebDriverException``
        if thrown."""
        try:
                return driver.find_element(*by)
        except NoSuchElementException as e:
                raise e
        except WebDriverException as e:
                raise e


def _find_elements(driver, by):
        try:
```

```
                return driver.find_elements(*by)
        except WebDriverException as e:
                raise e
```

4.3 判断 title（title_is）

前言

获取页面 title 的方法可以直接用 driver.title 获取到，然后也可以把获取到的结果用做断言。

本篇介绍另外一种方法去判断页面 title 是否与期望结果一种，用到上一篇 Selenium2+python 自动化 42-判断元素（expected_conditions）

提到的 expected_conditions 模块里的 title_is 和 title_contains 两种方法

4.3.1 源码分析

1.首先看下源码,如下

```
class title_is(object):
        """An expectation for checking the title of a page.
        title is the expected title, which must be an exact match
        returns True if the title matches, false otherwise."""

        '''翻译：检查页面的 title 与期望值是都完全一致，如果完全一致，返回 Ture,否则返回 Flase'''
        def __init__(self, title):
                self.title = title

        def __call__(self, driver):
                return self.title == driver.title
```

2.注释翻译：检查页面的 title 与期望值是都完全一致，如果完全一致，返回 True,否则返回 Flase

3.title_is()这个是一个 class 类型，里面有两个方法

4.__init__是初始化内容，参数是 title，必填项

5. __call__是把实例变成一个对象，参数是 driver，返回的是 self.title == driver.title，布尔值

4.3.2 判断 title:title_is()

1. 首先导入 expected_conditions 模块

2. 由于这个模块名称比较长，所以为了后续的调用方便，重新命名为 EC 了（有点像数据库里面多表查询时候重命名）

3. 打开博客首页后判断 title,返回结果是 True 或 False

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang")

title = EC.title_is(u'上海-悠悠 - 博客园')
print title(driver)
```

```
raise_case  boke (1)
D:\test\python2\python.exe D:/test/wait/ec/boke.py
True

Process finished with exit code 0
```

4.3.3 判断 title 包含:title_contains

1. 这个类跟上面那个类差不多，只是这个是部分匹配（类似于 xpath 里面的 contains 语法）

2. 判断 title 包含’上海-悠悠’字符串

```
# 判断title完全等于
title = EC.title_is(u'上海-悠悠 - 博客园')
print title(driver)

# 判断title包含
title1 = EC.title_contains(u'上海-悠悠')
print title1(driver)

# 另外一种写法,交流QQ群：232607095
r1 = EC.title_is(u'上海-悠悠 - 博客园')(driver)
r2 = EC.title_contains(u'上海-悠悠')(driver)
print r1
print r2
```

```
oke (1)
D:\test\python2\python.exe D:/test/wait/ec/boke.py
True
True
True
True
```

4.3.4 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang")
# 判断 title 完全等于
title = EC.title_is(u'上海-悠悠 - 博客园')
print title(driver)

# 判断 title 包含
title1 = EC.title_contains(u'上海-悠悠')
print title1(driver)

# 另外一种写法,交流 QQ 群：232607095
```

```
r1 = EC.title_is(u'上海-悠悠 - 博客园')(driver)
r2 = EC.title_contains(u'上海-悠悠')(driver)
print r1
print r2
```

4.4 判断文本（text_to_be_present_in_element）

前言

在做结果判断的时候，经常想判断某个元素中是否存在指定的文本，如登录后判断页面中是账号是否是该用户的用户名。

在前面的登录案例中，写了一个简单的方法，但不是公用的，在 EC 模块有个方法是可以专门用来判断元素中存在指定文本的：
text_to_be_present_in_element。

另外一个差不多复方法判断元素的 value 值：
text_to_be_present_in_element_value。

4.4.1 源码分析

```
class text_to_be_present_in_element(object):
        """ An expectation for checking if the given text is present in
the
        specified element.
        locator, text
        """

        '''翻译：判断元素中是否存在指定的文本，参数：locator, text'''
        def __init__(self, locator, text_):
                self.locator = locator
                self.text = text_

        def __call__(self, driver):
                try:
                        element_text = _find_element(driver,
self.locator).text
                        return self.text in element_text
```

```
except StaleElementReferenceException:
    return False
```

1. 翻译：判断元素中是否存在指定的文本，两个参数：locator, text

2. __call__里返回的是布尔值：Ture 和 False

4.4.2 判断文本

1. 判断百度首页上，“糯米”按钮这个元素中存在文本：糯米

2. locator 参数是定位的方法

3. text 参数是期望的值

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
locator = ("name", "tj_trnuomi")
text = u"糯米"
result = EC.text_to_be_present_in_element(locator, text)(driver)
print result
# 交流QQ群：232607095
```

```
ext_is
D:\test\python2\python.exe D:/test/wait/pa/text_is.py
True

Process finished with exit code 0
```

4.4.3 失败案例

1. 如果判断失败，就返回 False

```
locator = ("name", "tj_trnuomi")
text = u"糯米"
result = EC.text_to_be_present_in_element(locator, text)(driver)
print result
# 交流QQ群：232607095

# 下面是失败的案例
text1 = u"糯米网"
result1 = EC.text_to_be_present_in_element(locator, text1)(driver)
print result1
```

```
ext_is
D:\test\python2\python.exe D:/test/wait/pa/text_is.py
True
False
```

4.4.4 判断 value 的方法

```
class text_to_be_present_in_element_value(object):
    """
    An expectation for checking if the given text is present in the
element's
    locator, text
    """
    def __init__(self, locator, text_):
        self.locator = locator
        self.text = text_

    def __call__(self, driver):
        try:
            element_text = _find_element(driver,

        self.locator).get_attribute("value")
            if element_text:
                return self.text in element_text
            else:
```

```
                return False
        except StaleElementReferenceException:
                return False
```

1.这个方法跟上面的差不多，只是这个是判断的 value 的值

2.这里举个简单案例，判断百度搜索按钮的 value 值

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
locator2 = ("id", "su")
text2 = u"百度一下"
result2 = EC.text_to_be_present_in_element(locator2, text2)(driver)
print result2
```

```
text_is
D:\test\python2\python.exe D:/test/wait/pa/text_is.py
False

Process finished with exit code 0
```

4.4.5 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)

locator = ("name", "tj_trnuomi")
text = u"糯米"
result = EC.text_to_be_present_in_element(locator, text)(driver)
print result
# 交流 QQ 群：232607095

# 下面是失败的案例
text1 = u"糯米网"
result1 = EC.text_to_be_present_in_element(locator, text1)(driver)
print result1
```

```
locator2 = ("id", "su")
text2 = u"百度一下"
result2 = EC.text_to_be_present_in_element(locator2, text2)(driver)
print result2
```

4.5 判断弹出框存在(alert_is_present)

前言

系统弹窗这个是很常见的场景，有时候它不弹出来去操作的话，会抛异常。那么又不知道它啥时候会出来，那么久需要去判断弹窗是否弹出了。

本篇接着 Selenium2+python 自动化 42-判断元素（expected conditions）讲 expected_conditions 这个模块

4.5.1 判断 alert 源码分析

```
class alert_is_present(object):
        """ Expect an alert to be present."""

        """判断当前页面的 alert 弹窗"""
        def __init__(self):
                pass

        def __call__(self, driver):
                try:
                        alert = driver.switch_to.alert
                        alert.text
                        return alert
                except NoAlertPresentException:
                        return False
```

1.这个类比较简单，初始化里面无内容

2.__call__里面就是判断如果正常获取到弹出窗的 text 内容就返回 alert 这个对象（**注意这里不是返回 Ture**），没有获取到就返回 False

4.5.2 实例操作

1.前面的操作步骤优化了下，为了提高脚本的稳定性，确保元素出现后操作，

这里结合 WebDriverWait 里的方法：Selenium2+python 自动化 38-显式等待（WebDriverWait）

2.实现步骤如下，这里判断的结果返回有两种：没找到就返回 False；找到就返回 alert 对象

3.先判断 alert 是否弹出，如果弹出就点确定按钮 accept()

```
mouse = WebDriverWait(driver, 10).until(lambda x: x.find_element("link text", "设置"))
ActionChains(driver).move_to_element(mouse).perform()
WebDriverWait(driver, 10).until(lambda x: x.find_element("link text", "搜索设置")).click()
#  选择设置项
s = WebDriverWait(driver, 10).until(lambda x: x.find_element("id", "nr"))
Select(s).select_by_visible_text("每页显示50条")
# 点保存按钮
js = 'document.getElementsByClassName("prefpanelgo")[0].click();'
driver.execute_script(js)
# 判断弹窗结果 交流QQ群: 232607095
result = EC.alert_is_present()(driver)
if result:
    print result.text
    result.accept()
else:
    print "alert 未弹出！"
```

```
tan
D:\test\python2\python.exe D:/test/wait/pa/tan.py
已经记录下您的使用偏好
```

4.5.3 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.support.select import Select
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
url = "https://www.baidu.com"
driver.get(url)
mouse = WebDriverWait(driver, 10).until(lambda x: x.find_element("link text", "设置"))
ActionChains(driver).move_to_element(mouse).perform()
```

```
WebDriverWait(driver, 10).until(lambda x: x.find_element("link text", "
搜索设置")).click()
#  选择设置项
s = WebDriverWait(driver, 10).until(lambda x: x.find_element("id", "nr"))
Select(s).select_by_visible_text("每页显示 50 条")
# 点保存按钮
js = 'document.getElementsByClassName("prefpanelgo")[0].click();'
driver.execute_script(js)
# 判断弹窗结果 交流 QQ 群: 232607095
result = EC.alert_is_present()(driver)
if result:
       print result.text
       result.accept()
else:
       print "alert 未弹出！"
```

4.6 判断元素出现

presence_of_element_located

presence_of_all_elements_located

敬请期待！

4.7 判断元素可见

visibility_of_element_located

invisibility_of_element_located

visibility_of

敬请期待！

4.8 判断 iframe 可切入

frame_to_be_available_and_switch_to_it

敬请期待！

4.9 判断元素可点击

element_to_be_clickable

敬请期待！

4.10 判断元素被选中

element_to_be_selected

element_located_to_be_selected

element_selection_state_to_be

element_located_selection_state_to_be

敬请期待！

4.11 判断元素是否消失

判断一个元素是否仍在 DOM 中，传入 WebElement 对象，可以判断页面是否刷新了

staleness_of

敬请期待！

第 5 章 二次封装

5.1 元素定位参数化（find_element）

前言

元素定位有八种方法，这个能看到这一篇的小伙伴都知道了，那么有没有一种方法，可以把八种定位合为一种呢？也就是把定位的方式参数化，如 id,name.css 等设置为一个参数，这样只需维护定位方式的参数就行了。

小编曾经自己封装过这种方法，最后定位方法写成这样：
find_element("id=kw"),find_element("css=#kw"),这个思路是来源于 RF 框架里面的，等号前面是定位方法，等号后面是元素名称。

这两天闲着无事查看定位方法的源码，发现了新大陆，忍不住想分享给小伙伴。

5.1.1 find_element()

1.selenium 元素定位里面其实是有这个方法的，只是大部分时候都是结合 By 方法使用，如下图

5.1.2 查看 find_element 方法源码

1.find_element 跟 find_element_by_xxx 到底有什么区别呢？好奇害死猫啊，找到这个路径：Lib\site-packages\selenium\webdriver\remote\utils.py

2.打开文件夹后发现，其实定 find_element_by_xxx 的方法都是返回的 find_element 方法，也就是说那八个定位方法其实就是八个小分支。

```
def find_element(self, by=By.ID, value=None):
    return self._dispatch("find", (by, value, self._driver), "find_element", (by, value))

def find_elements(self, by=By.ID, value=None):
    return self._dispatch("find", (by, value, self._driver), "find_elements", (by, value))

def find_element_by_id(self, id_):
    return self.find_element(by=By.ID, value=id_)

def find_elements_by_id(self, id_):
    return self.find_elements(by=By.ID, value=id_)

def find_element_by_xpath(self, xpath):
    return self.find_element(by=By.XPATH, value=xpath)

def find_elements_by_xpath(self, xpath):
    return self.find_elements(by=By.XPATH, value=xpath)

def find_element_by_link_text(self, link_text):
    return self.find_element(by=By.LINK_TEXT, value=link_text)

def find_elements_by_link_text(self, text):
    return self.find_elements(by=By.LINK_TEXT, value=text)
```

5.1.3 By 定位方法

1.找到这个路径：Lib\site-packages\selenium\webdriver\common\by.py

2.打开 by 这个模块，其实里面很简单啊，就是几个字符串参数。

3.那么问题就简单了，其实压根可以不用绕这么大弯路去导入这个模块啊，说实话，我一点都不喜欢去导入这个 By，总觉得太繁琐。

```
"""
The By implementation.
"""


class By(object):
        """
        Set of supported locator strategies.
        """

        ID = "id"
        XPATH = "xpath"
        LINK_TEXT = "link text"
        PARTIAL_LINK_TEXT = "partial link text"
        NAME = "name"
        TAG_NAME = "tag name"
        CLASS_NAME = "class name"
        CSS_SELECTOR = "css selector"
```

5.1.4 定位参数化

1.小编一直追求简单粗暴的方式，接下来就用最简单的方法去定位

2.总结下几种定位方法(字符串中间是空格需注意)

```
by_id= "id"
by_xpath = "xpath"
by_link_text = "link text"
by_partial_text = "partial link text"
by_name = "name"
by_tag_name = "tag name"
by_class_name = "class name"
by_css_selector = "css selector"
```

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Firefox()
driver.get("https://www.baidu.com/")

driver.find_element("id", "kw").send_keys("yoyoketang")
driver.find_element('css selector', "#su").click()

# 其它定位参考 交流QQ群：232607095
# t1 = driver.find_element("link text", "糯米").text
# print t1
# t2 = driver.find_element("name", "tj_trnews").text
# print t2
# t3 = driver.find_element("class name", "bri").text
# print t3
```

5.1.5 参考代码

```
# coding:utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
driver = webdriver.Firefox()
driver.get("https://www.baidu.com/")
```

```
driver.find_element("id", "kw").send_keys("yoyoketang")
driver.find_element('css selector', "#su").click()

# 其它定位参考 交流 QQ 群：232607095
# t1 = driver.find_element("link text", "糯米").text
# print t1
# t2 = driver.find_element("name", "tj_trnews").text
# print t2
# t3 = driver.find_element("class name", "bri").text
# print t3
```

5.2 登录方法（参数化）

前言

登录这个场景在写用例的时候经常会有，我们可以把登录封装成一个方法，然后把账号和密码参数化，这样以后用的登录的时候，只需调用这个方法就行了

5.2.1 登录方法

1. 把输入账号、输入密码、点击登录按钮三个步骤写成一个方法

2. 然后把输入的账号和密码参数化

```python
# coding:utf-8
from selenium import webdriver
import unittest
import time
class Bolg(unittest.TestCase):
    u'''登录博客'''
    def setUp(self):
        self.driver = webdriver.Firefox()
        url = "https://passport.cnblogs.com/user/signin"
        self.driver.get(url)
        self.driver.implicitly_wait(30)

    def login(self, username, psw):
        u'''这里写了一个登录的方法,账号和密码参数化'''
        self.driver.find_element_by_id("input1").send_keys(username)
        self.driver.find_element_by_id("input2").send_keys(psw)
        self.driver.find_element_by_id("signin").click()
        time.sleep(3)
```

5.2.2 用例

1.下面的用例可以调用前面写的登录方法，这样不用每次都去走登录流程

2.判断是否登录成功，我这里是取的登录成功后的账户名

```
def test_01(self):
    u'''登录案例参考:账号，密码自己设置'''
    self.login(u"上海-悠悠", u"xxxx")  # 调用登录方法
    # 获取登录后的账号名称
    text = self.driver.find_element_by_id("lnk_current_user").text
    print text
    # 断言实际结果与期望结果一致
    self.assertEqual(text, u"上海-悠悠")

def test_02(self):
    u'''登录案例参考:账号，密码自己设置'''
    self.login(u"上海-悠悠", u"oooo")  # 调用登录方法
    # 获取登录后的账号名称
    text = self.driver.find_element_by_id("lnk_current_user").text
    print text        # 交流QQ群：232607095
    # 断言实际结果与期望结果一致
    self.assertEqual(text, u"上海-悠悠")
```

5.2.3 判断方法封装

1.如果用上面的方法去判断的话，有个缺陷，当登录不成功的时候，页面是不会跳转的，所以查找元素会报异常：

NoSuchElementException: Message: Unable to locate element: {"method":"id","selector":"lnk_current_user"}

2.这个时候就简单封装下判断方法：获取到账户名返回 Ture;没有获取到返回 False

```
def is_login_sucess(self):
    u'''判断是否获取到登录账户名称'''
    try:
        text = self.driver.find_element_by_id("lnk_current_user").text
        print text
        return True
    except:
        return False
```

5.2.4 优化后案例

1.优化后的登录案例如下，这样看起来更清楚了

```
def test_01(self):
    u'''登录案例参考:账号，密码自己设置'''
    self.login(u"上海-悠悠", u"xxxx")  # 调用登录方法
    # 判断结果
    result = self.is_login_sucess()
    self.assertTrue(result)

def test_02(self):
    u'''登录案例参考:账号，密码自己设置'''
    self.login(u"上海-悠悠", u"xxxx")  # 调用登录方法
    # 判断结果    # 交流QQ群：232607095
    result = self.is_login_sucess()
    self.assertTrue(result)
```

5.2.5 参考代码

```
# coding:utf-8
from selenium import webdriver
import unittest
import time
class Bolg(unittest.TestCase):
        u'''登录博客'''
        def setUp(self):
                self.driver = webdriver.Firefox()
                url = "https://passport.cnblogs.com/user/signin"
```

```
                self.driver.get(url)
                self.driver.implicitly_wait(30)

        def login(self, username, psw):
                u'''这里写了一个登录的方法,账号和密码参数化'''
                self.driver.find_element_by_id("input1").send_keys(use
rname)
                 self.driver.find_element_by_id("input2").send_keys(psw)
                self.driver.find_element_by_id("signin").click()
                time.sleep(3)

        def is_login_sucess(self):
                u'''判断是否获取到登录账户名称'''
                try:
                        text =
self.driver.find_element_by_id("lnk_current_user").text
                        print text
                        return True
                except:
                        return False

        def test_01(self):
                u'''登录案例参考:账号，密码自己设置'''
                self.login(u"上海-悠悠", u"xxxx")   # 调用登录方法
                # 判断结果
                result = self.is_login_sucess()
                self.assertTrue(result)

        def test_02(self):
                u'''登录案例参考:账号，密码自己设置'''
                self.login(u"上海-悠悠", u"xxxx")   # 调用登录方法
                # 判断结果     # 交流 QQ 群：232607095
                result = self.is_login_sucess()
                self.assertTrue(result)



        # def test_01(self):
        #         u'''登录案例参考:账号，密码自己设置'''
        #         self.login(u"上海-悠悠", u"xxxx")   # 调用登录方法
        #         # 获取登录后的账号名称
        #         text =
self.driver.find_element_by_id("lnk_current_user").text
        #         print text
        #         # 断言实际结果与期望结果一致
```

```
        #         self.assertEqual(text, u"上海-悠悠")
        #
        # def test_02(self):
        #         u'''登录案例参考:账号，密码自己设置'''
        #         self.login(u"上海-悠悠", u"oooo")    # 调用登录方法
        #         # 获取登录后的账号名称
        #         text =
self.driver.find_element_by_id("lnk_current_user").text
        #         print text              # 交流 QQ 群：232607095
        #         # 断言实际结果与期望结果一致
        #         self.assertEqual(text, u"上海-悠悠")

        def tearDown(self):
                self.driver.quit()

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

5.3 捕获异常（NoSuchElementException）

前言

在定位元素的时候，经常会遇到各种异常，为什么会发生这些异常，遇到异常又该如何处理呢？

本篇通过学习 selenium 的 exceptions 模块，了解异常发生的原因。

5.3.1 发生异常

1.打开博客首页，定位“新随笔”元素，此元素 id="blog_nav_newpost"

2.为了故意让它定位失败，我在元素属性后面加上 xx

3.运行失败后如下图所示，程序在查找元素的这一行发生了中断，不会继续执行 click 事件了

```
# coding:utf-8
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang/")
# 定位首页"新随笔"
element = driver.find_element("id", "blog_nav_newpostxx")
#点击该元素
element.click()
```

```
test06
D:\test\python2\python.exe D:/test/yoyotest/di2ke/test06.py
Traceback (most recent call last):
  File "D:/test/yoyotest/di2ke/test06.py", line 7, in <module>
    element = driver.find_element("id", "blog_nav_newpostxx")
  File "D:\test\python2\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 752, in find
    'value': value})['value']
  File "D:\test\python2\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 236, in exec
```

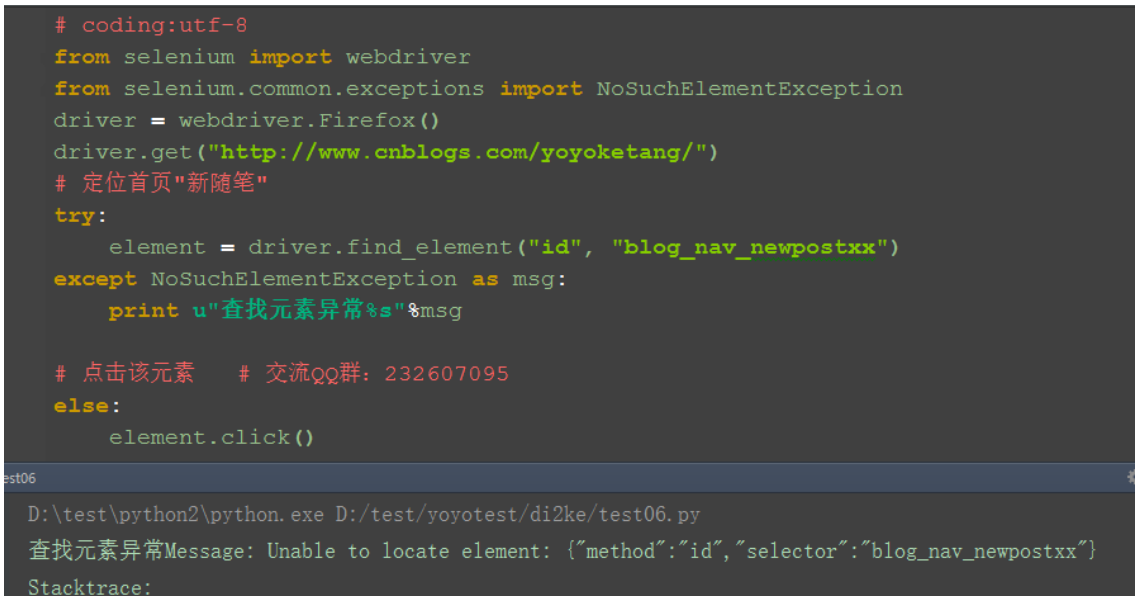
5.3.2 捕获异常

1.为了让程序继续执行，我们可以用 try...except...捕获异常。捕获异常后可以打印出异常原因，这样以便于分析异常原因

2.从如下异常内容可以看出，发生异常原因是：NoSuchElementException

selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: {"method":"id","selector":"blog_nav_newpostxx"}

3.从 selenium.common.exceptions 导入 NoSuchElementException 类



5.3.3 参考代码：

```
# coding:utf-8
from selenium import webdriver
from selenium.common.exceptions import NoSuchElementException
driver = webdriver.Firefox()
driver.get("http://www.cnblogs.com/yoyoketang/")
# 定位首页"新随笔"
try:
        element = driver.find_element("id", "blog_nav_newpostxx")
except NoSuchElementException as msg:
        print u"查找元素异常%s"%msg

# 点击该元素      # 交流 QQ 群：232607095
else:
        element.click()
```

5.3.4 selenium 常见异常

1.NoSuchElementException：没有找到元素

2.NoSuchFrameException：没有找到 iframe

3.NoSuchWindowException:没找到窗口句柄 handle

4.NoSuchAttributeException:属性错误

5.NoAlertPresentException：没找到 alert 弹出框

6.lementNotVisibleException：元素不可见

7.ElementNotSelectableException：元素没有被选中

8.TimeoutException：查找元素超时

5.3.5 其它异常与源码

1.在 Lib 目录下：selenium/common/exceptions 有兴趣的可以看看

```
# Licensed to the Software Freedom Conservancy (SFC) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The SFC licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.

"""
Exceptions that may happen in all the webdriver code.
"""


class WebDriverException(Exception):
       """
       Base webdriver exception.
       """
```

```
        def __init__(self, msg=None, screen=None, stacktrace=None):
                self.msg = msg
                self.screen = screen
                self.stacktrace = stacktrace

        def __str__(self):
                exception_msg = "Message: %s\n" % self.msg
                if self.screen is not None:
                        exception_msg += "Screenshot: available via
screen\n"
                if self.stacktrace is not None:
                        stacktrace = "\n".join(self.stacktrace)
                        exception_msg += "Stacktrace:\n%s" % stacktrace
                return exception_msg


class ErrorInResponseException(WebDriverException):
        """
        Thrown when an error has occurred on the server side.

        This may happen when communicating with the firefox extension
        or the remote driver server.
        """
        def __init__(self, response, msg):
                WebDriverException.__init__(self, msg)
                self.response = response


class InvalidSwitchToTargetException(WebDriverException):
        """
        Thrown when frame or window target to be switched doesn't exist.
        """
        pass


class NoSuchFrameException(InvalidSwitchToTargetException):
        """
        Thrown when frame target to be switched doesn't exist.
        """
        pass


class NoSuchWindowException(InvalidSwitchToTargetException):
```

```
        """
        Thrown when window target to be switched doesn't exist.

        To find the current set of active window handles, you can get a
list
        of the active window handles in the following way::

                print driver.window_handles

        """
        pass


class NoSuchElementException(WebDriverException):
        """
        Thrown when element could not be found.

        If you encounter this exception, you may want to check the
following:
                * Check your selector used in your find_by...
                * Element may not yet be on the screen at the time of the
find operation,
                 (webpage is still loading) see
selenium.webdriver.support.wait.WebDriverWait()
                 for how to write a wait wrapper to wait for an element to
appear.
        """
        pass


class NoSuchAttributeException(WebDriverException):
        """
        Thrown when the attribute of element could not be found.

        You may want to check if the attribute exists in the particular
browser you are
        testing against.   Some browsers may have different property
names for the same
        property.    (IE8's .innerText vs. Firefox .textContent)
        """
        pass


class StaleElementReferenceException(WebDriverException):
```

```
        """
        Thrown when a reference to an element is now "stale".

        Stale means the element no longer appears on the DOM of the page.


        Possible causes of StaleElementReferenceException include, but
not limited to:
                * You are no longer on the same page, or the page may have
refreshed since the element
                was located.
                * The element may have been removed and re-added to the
screen, since it was located.
                Such as an element being relocated.
                This can happen typically with a javascript framework when
values are updated and the
                node is rebuilt.
                * Element may have been inside an iframe or another context
which was refreshed.
        """
        pass


class InvalidElementStateException(WebDriverException):
        """
        """
        pass


class UnexpectedAlertPresentException(WebDriverException):
        """
        Thrown when an unexpected alert is appeared.

        Usually raised when when an expected modal is blocking webdriver
form executing any
        more commands.
        """
        def __init__(self, msg=None, screen=None, stacktrace=None,
alert_text=None):
                super(UnexpectedAlertPresentException,
self).__init__(msg, screen, stacktrace)
                self.alert_text = alert_text

        def __str__(self):
```

```
                return "Alert Text: %s\n%s" % (self.alert_text,
super(UnexpectedAlertPresentException, self).__str__())


class NoAlertPresentException(WebDriverException):
    """
    Thrown when switching to no presented alert.

    This can be caused by calling an operation on the Alert() class
when an alert is
    not yet on the screen.
    """
    pass


class ElementNotVisibleException(InvalidElementStateException):
    """
    Thrown when an element is present on the DOM, but
    it is not visible, and so is not able to be interacted with.

    Most commonly encountered when trying to click or read text
    of an element that is hidden from view.
    """
    pass


class ElementNotSelectableException(InvalidElementStateException):
    """
    Thrown when trying to select an unselectable element.

    For example, selecting a 'script' element.
    """
    pass


class InvalidCookieDomainException(WebDriverException):
    """
    Thrown when attempting to add a cookie under a different domain
    than the current URL.
    """
    pass


class UnableToSetCookieException(WebDriverException):
```

```
        """
        Thrown when a driver fails to set a cookie.
        """
        pass


class RemoteDriverServerException(WebDriverException):
        """
        """
        pass


class TimeoutException(WebDriverException):
        """
        Thrown when a command does not complete in enough time.
        """
        pass


class MoveTargetOutOfBoundsException(WebDriverException):
        """
        Thrown when the target provided to the `ActionsChains` move()
        method is invalid, i.e. out of document.
        """
        pass


class UnexpectedTagNameException(WebDriverException):
        """
        Thrown when a support class did not get an expected web element.
        """
        pass


class InvalidSelectorException(NoSuchElementException):
        """
        Thrown when the selector which is used to find an element does not
return
        a WebElement. Currently this only happens when the selector is an
xpath
        expression and it is either syntactically invalid (i.e. it is not
a
        xpath expression) or the expression does not select WebElements
        (e.g. "count(//input)").
```

```
        """
        pass


class ImeNotAvailableException(WebDriverException):
        """
        Thrown when IME support is not available. This exception is thrown
for every IME-related
        method call if IME support is not available on the machine.
        """
        pass


class ImeActivationFailedException(WebDriverException):
        """
        Thrown when activating an IME engine has failed.
        """
        pass
```

5.4 读取 Excel 数据（xlrd）

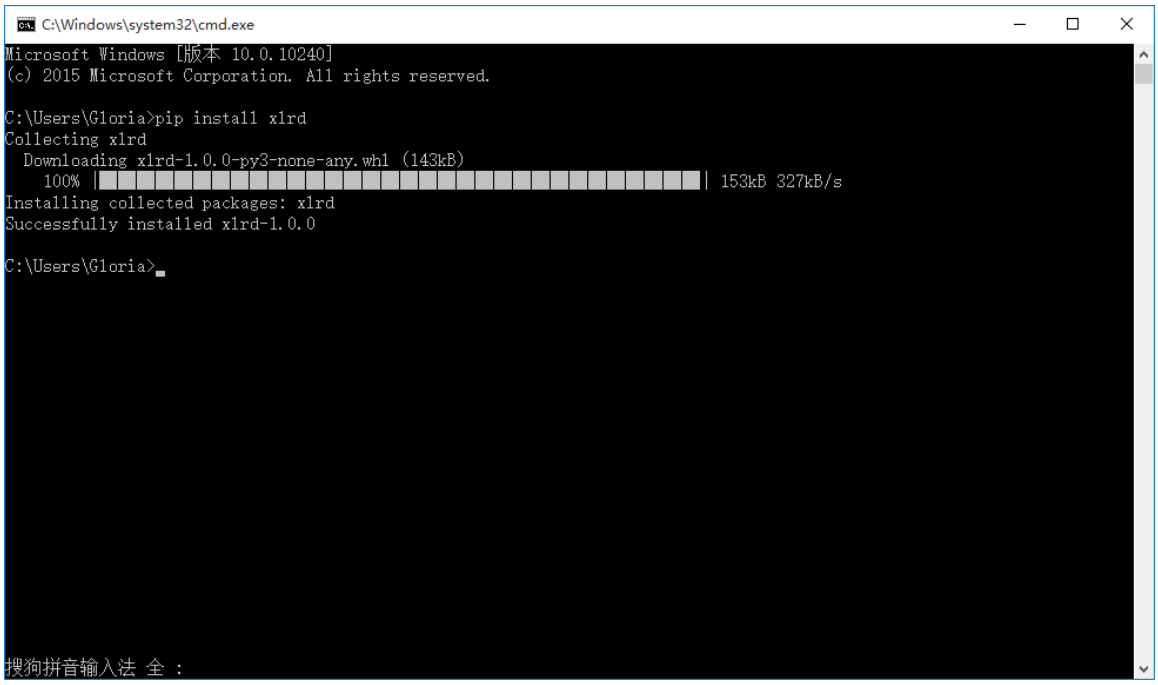
前言

当登录的账号有多个的时候，我们一般用 excel 存放测试数据，本节课介绍，python 读取 excel 方法，并保存为字典格式。

5.4.1 环境准备

1.先安装 xlrd 模块，打开 cmd，输入 pip install xlrd 在线安装

```
>>pip install xlrd
```



5.4.2 基本操作

1.exlce 基本操作方法如下

```
# 打开 exlce 表格，参数是文件路径
data = xlrd.open_workbook('test.xlsx')

# table = data.sheets()[0]                    #    通过索引顺序获取
# table = data.sheet_by_index(0)              #    通过索引顺序获取
table = data.sheet_by_name(u'Sheet1')    # 通过名称获取

nrows = table.nrows    # 获取总行数
ncols = table.ncols    # 获取总列数

#  获取一行或一列的值，参数是第几行
print table.row_values(0)    # 获取第一行值
print table.col_values(0)    # 获取第一列值
```

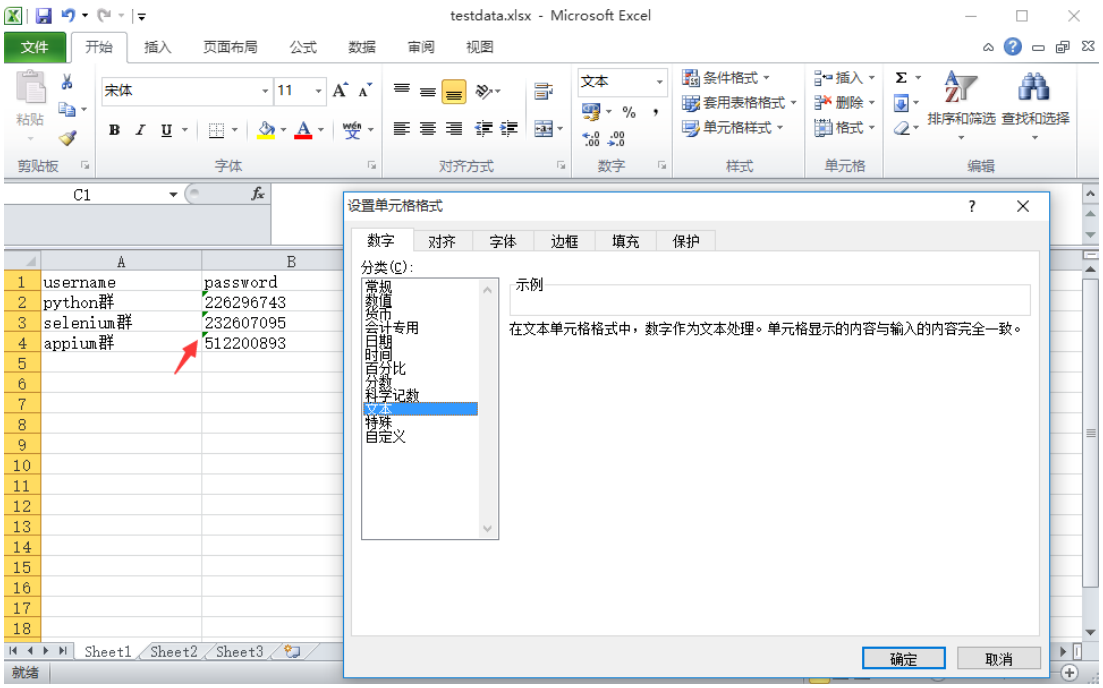
```python
# coding:utf-8
import xlrd

# 打开exlce表格
data = xlrd.open_workbook('testdata.xlsx')

# table = data.sheets()[0]            #  通过索引顺序获取
# table = data.sheet_by_index(0)      #  通过索引顺序获取
table = data.sheet_by_name(u'Sheet1')  # 通过名称获取

nrows = table.nrows  # 获取总行数
ncols = table.ncols  # 获取总列数

print table.row_values(0)  # 获取第一行值
print table.col_values(0)  # 获取第一列值
```

5.4.3 excel 存放数据

1. 在 excel 中存放数据，第一行为标题，也就是对应字典里面的 key 值，如：username，password

2. 如果 excel 数据中有纯数字的一定要右键》设置单元格格式》文本格式，要不然读取的数据是浮点数

（先设置单元格格式后编辑，编辑成功左上角有个小三角图标）



5.4.4 封装读取方法

1.最终读取的数据是多个字典的 list 类型数据，第一行数据就是字典里的 key 值，从第二行开始一一对应 value 值

2.封装好后的代码如下

```
# coding:utf-8
import xlrd
class ExcelUtil():
	def __init__(self, excelPath, sheetName):
		self.data = xlrd.open_workbook(excelPath)
		self.table = self.data.sheet_by_name(sheetName)
		# 获取第一行作为 key 值
		self.keys = self.table.row_values(0)
		# 获取总行数
		self.rowNum = self.table.nrows
		# 获取总列数
		self.colNum = self.table.ncols

	def dict_data(self):
		if self.rowNum <= 1:
			print("总行数小于 1")
		else:
			r = []
			j=1
			for i in range(self.rowNum-1):
				s = {}
				# 从第二行取对应 values 值
				values = self.table.row_values(j)
				for x in range(self.colNum):
					s[self.keys[x]] = values[x]
				r.append(s)
				j+=1
			return r

if __name__ == "__main__":
	filepath = "D:\\test\\web-project\\5ke\\testdata.xlsx"
	sheetName = "Sheet1"
	data = ExcelUtil(filepath, sheetName)
	print data.dict_data()
```

运行结果：

[{u'username': u'python\u7fa4', u'password': u'226296743'},

{u'username': u'selenium\u7fa4', u'password': u'232607095'},

{u'username': u'appium\u7fa4', u'password': u'512200893'}]

5.5 数据驱动（ddt）

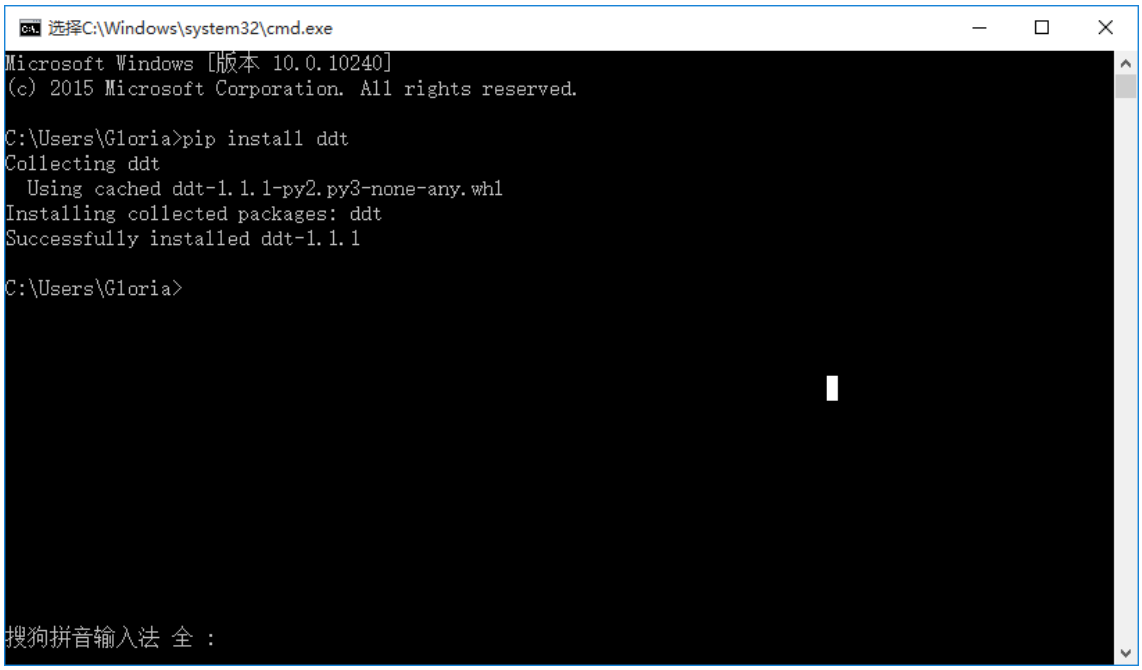
前言

在设计用例的时候，有些用例只是参数数据的输入不一样，比如登录这个功能，操作过程但是一样的。如果用例重复去写操作过程会增加代码量，对应这种多组数据的测试用例，可以用数据驱动设计模式，一组数据对应一个测试用例，用例自动加载生成。

5.5.1 环境准备

1.安装 ddt 模块，打开 cmd 输入 pip install ddt 在线安装

>>pip install ddt



5.5.2 数据驱动原理

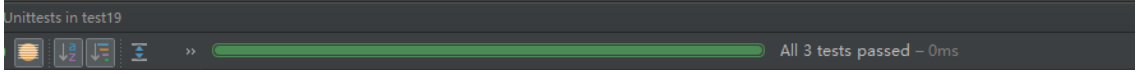
1. 测试数据为多个字典的 list 类型

2. 测试类前加修饰@ddt.ddt

3. case 前加修饰@ddt.data()

4. 运行后用例会自动加载成三个单独的用例

```
# coding:utf-8
import ddt
import unittest
# 测试数据
testData = [{"username": "selenium群", "psw": "232607095"},
            {"username": "python群", "psw": "226296743"},
            {"username": "appium群", "psw": "512200893"}]
@ddt.ddt
class Test(unittest.TestCase):
    def setUp(self):
        print "start!"

    def tearDown(self):
        print "end!"

    @ddt.data(*testData)
    def test_ddt(self, data):
        print data

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



5. 测试结果：

```
Testing started at 21:51 ...
start!
{'username': 'selenium\xe7\xbe\xa4', 'psw': '232607095'}
end!
start!
{'username': 'python\xe7\xbe\xa4', 'psw': '226296743'}
end!
start!
{'username': 'appium\xe7\xbe\xa4', 'psw': '512200893'}
end!
```

5.5.3 selenium 案例

1.从上一篇封装的 excel 方法里面读取数据，作为测试测试 Selenium2+python 自动化 58-读取 Excel 数据（xlrd）

2.在之前写的登录那篇基础上做点修改，测试参数读取 excel 里的数据 Selenium2+python 自动化 48-登录方法（参数化）

3.代码参考如下

```
# 测试数据
testData = data.dict_data()
print testData

@ddt.ddt
class Bolg(unittest.TestCase):
        u'''登录博客'''
        def setUp(self):
                self.driver = webdriver.Firefox()
                url = "https://passport.cnblogs.com/user/signin"
                self.driver.get(url)
                self.driver.implicitly_wait(30)

        def login(self, username, psw):
                u'''这里写了一个登录的方法,账号和密码参数化'''
                self.driver.find_element_by_id("input1").send_keys(use
rname)
                self.driver.find_element_by_id("input2").send_keys(psw)
                self.driver.find_element_by_id("signin").click()
                time.sleep(3)

        def is_login_sucess(self):
                u'''判断是否获取到登录账户名称'''
                try:
                        text =
self.driver.find_element_by_id("lnk_current_user").text
                        print text
                        return True
                except:
                        return False

        @ddt.data(*testData)
        def test_login(self, data):
```

```
                u'''登录案例参考'''
                print ("当前测试数据%s"%data)
                # 调用登录方法
                self.login(data["username"], data["password"])
                # 判断结果
                result = self.is_login_sucess()
                self.assertTrue(result)

        def tearDown(self):
                self.driver.quit()

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

第 6 章 Page Object

6.1 分层设计

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

6.2 底层封装常用操作

敬请期待！

6.3 page 页面

敬请期待！

6.4 用例设计

敬请期待！

6.5 封装判断方法

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

6.6 多页面交互

敬请期待！

6.7 项目源码

敬请期待！

第 8 章 持续集成 jenkins

8.1 tomcat+jenkins 环境搭建

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

8.2 配置 selenium 环境

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

8.3 命令行参数化

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3

8.4 git 仓库

请购买正版阅读

https://yuedu.baidu.com/ebook/0f6a093b7dd184254b35eefdc8d376eeaeaa17e3