Selenium 自动化测试

Web 版

作者：木头人（WoodMan）

网络文章链接：https://zhuanlan.zhihu.com/automatedtest

欢迎关注，并点赞

1.1 自动化测试流程

系统评估 ==> 需求筛选、评审 ==> 用例设计 ==>脚本实现 ==>执行、报告分析 ==> 用例维护更新==>收益分析

1.环境搭建

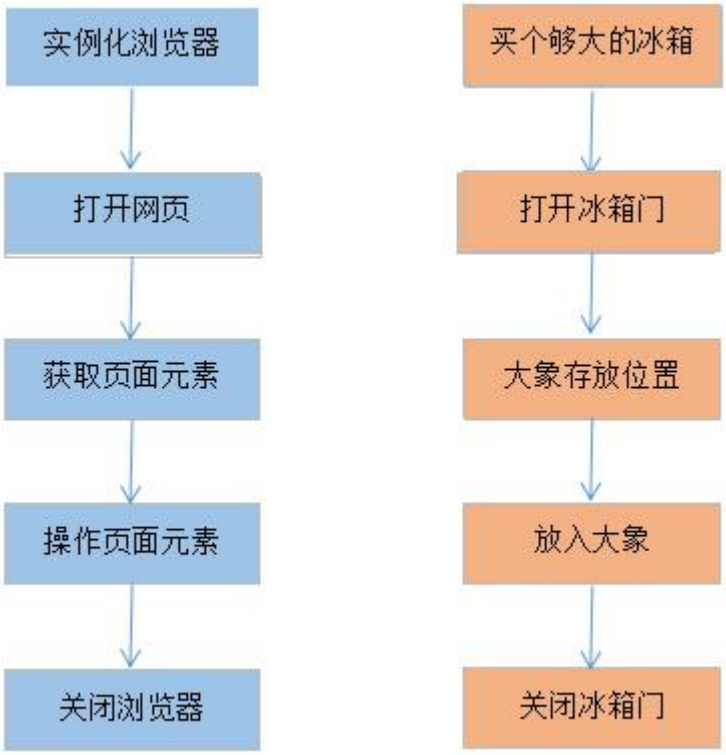
Python 安装 https://www.python.org/getit/
环境变量配置 path 添加两个 ;C:\Software\Python35;C:\Software\Python35\Scripts;
Selenium 安装 pip install selenium
WebDriver 驱动 将所需要的驱动放入 path 环境变量的文件夹中

1.1 一个简单的示例

Python+Selenium 环境搭建好，webDriver 驱动下载设置好后，我们可以 web UI 的自动化测试学习了。
从一个简单的实例开始，代码如下。

```
from selenium import webdriver  # 引入包
driver = webdriver.Chrome()  # 实例化浏览器
driver.get('https://www.zhihu.com/')  # 打开网页
element = driver.find_element_by_name('fullname')  # 获取页面元素对象
element.send_keys('woodman')  # 操作页面元素对象
driver.quit()  # 关闭浏览器
```

上面示例为 打开知乎首页，输入注册用户名，然后关闭浏览器。
整个过程如同我们将一只大象放进冰箱，如下图：



实例化浏览器：根据测试需求打开不同类型的浏览器，可以对浏览器进行各种个性化的配置
打开网页：打开需要测试 web 系统网页
获取元素对象：对那个页面进行操作，先获取该元素对象(链接、输入框、按钮、文本等)
操作页面元素：对页面元素进行各种操作（单击、输入文字、选择下拉框、获取页面文本等）
关闭浏览器：测试完成后关闭浏览器，释放资源

2.Selenium 常用操作

2.1 浏览器操作

2.1.1 实例化浏览器

启动 Chrome 浏览器（驱动以放入 path 环境变量下）

```
driver = webdriver.Chrome()
```

指定驱动路径驱动 Chrome 浏览器

```
# .\driver\chromedriver.exe 为驱动存放位置,可以是相对路径或者绝对路径
driver = webdriver.Chrome(executable_path=r'.\driver\chromedriver.exe')
```

启动 Firefox 浏览器

```
driver=webdriver.Firefox()
driver=webdriver.Firefox(executable_path="..\driver\geckodriver.exe") # 指定驱动路径
```

启动 IE 浏览器

```
driver=webdriver.Ie()
driver=webdriver.Ie(executable_path=r".\driver\IEDriverServer.exe") # 指定驱动路径启动
```

2.1.2 最大化浏览器

```
driver.maximize_window()
```

2.1.3 设置浏览器大小

```
driver.set_window_size(480, 800)
# 获取浏览器窗口大小
size=driver.get_window_size() # 返回浏览器窗口大小字典 {'width': 1050, 'height': 840}
```

2.1.4 控制浏览器前进、后退

```
driver.get('https://www.zhihu.com/')
driver.get('https://mail.163.com')  # 在同一窗口重新打开一个网页
driver.back()  # 后退 到知乎
driver.forward()  # 前进 换回到 163
```

2.1.5 打开网页

```
URL="https://www.zhihu.com"
driver.get(URL)
# 新开一个窗口，通过执行 js 来新开一个窗口
js='window.open("https://www.sogou.com");'
driver.execute_script(js)
```

2.1.6 关闭浏览器

```
driver.close()  # 关闭浏览器
关闭的是当前浏览器窗口的页签，存在多个窗口时关闭当前的活动窗口
driver.quit()  # 退出浏览器
关闭整个浏览器，包括 webdriver 的进程也会退出
```

2.1.7 获取网页标题

```
title =driver.title  # 获取网页的 title
```

获取到的 title 为页面源码 head 标签中 title 中的文本信息

2.1.8 获取网页的 URL

```
driver.current_url  # 获取网页的 URL
```

获取的 url 为当前浏览器地址栏中的 url

2.1.9、刷新页面

```
driver.refresh()
```

2.1.10、获取浏览器窗口大小

```
size=driver.get_window_size()
```

返回为字典型 如：{'width': 1050, 'height': 840}

2.2 元素定位，获取元素对象

八种属性定位页面元素：

```
By.ID
By.XPATH
```

```
By.LINK_TEXT
By.PARTIAL_LINK_TEXT
By.NAME
By.TAG_NAME
By.CLASS_NAME
By.CSS_SELECTOR
```

webdriver 元素定位方法：

```
driver.find_element(By.XXX,'元素属性')  # 定位单个元素
driver.find_elements(By.XXX,'元素属性')  # 定位一组元素，返回 list 列表
```

webdriver 中元素定位元素的简便方法：

```
driver.find_element_by_id('元素 id 属性')  #------ 最常用，简单
driver.find_element_by_name('元素 name 属性')  #------ 最常用，简单
driver.find_element_by_class_name('元素 class 属性')  #------ 易重复，看情况使用
driver.find_element_by_tag_name('元素标签名')  #------ 最不靠谱
driver.find_element_by_link_text('链接文本')  #------ 精确匹配链接 （<a> 标签中的文字）
driver.find_element_by_partial_link_text(‘部分链接文本’)  #------ 模糊匹配链接
driver.find_element_by_xpath()  #------ 最灵活，万能的灵药
driver.find_element_by_css_selector()  #------ 没 xpath 灵活
```

如果需要使用简便的方法点位一组元素，在 element 后加个 s，如，driver.find_elements_by_id() ；它返回的也是 list 列表。

怎么获取元素属性？

第一种：Chrome 或者 Firefox 打开网页，在需要获取属性的元素上[右键 ==> 检查（or 查看元素）]

第二种：Chrome 或者 Firefox 打开网页，按 F12（开发者选项），选择左上角的鼠标指针后单击页面元素

2.2.1 id 元素定位

<input **id="query"** class="sec-input" name="query" maxlength="100" autocomplete="off" type="text">

```
driver.find_element_by_id('query')
```

2.2.2 name 元素定位

<input id="query" class="sec-input" **name="query"** maxlength="100" autocomplete="off" type="text">

driver.find_element_by_name(**'query'**)

2.2.3 class name 元素定位

<input id="query" **class="sec-input"** name="query" maxlength="100" autocomplete="off" type="text">

driver.find_element_by_class_name(**'sec-input')**

2.2.4 tag name 元素定位(最不靠谱)

<**input** id="query" class="sec-input" name="query" maxlength="100" autocomplete="off" type="text">

driver.find_element_by_tag_name('**input** ')

2.2.5 link_text 元素定位

<a uigs-id="mid_pinyin" href="http://pinyin.sogou.com/" target="_blank">**搜狗输入法**

driver.find_element_by_link_text('搜狗输入法')

注意：连接文本是<a>标签对之间的文本

2.2.6 partial link text 元素定位

<<a uigs-id="mid_pinyin" href="http://pinyin.sogou.com/" target="_blank">搜狗**输入法**

driver.find_element_by_partial_link_text(‘输入法’)

2.2.7 XPath 元素定位(强大)

学习资料： http://www.w3school.com.cn/xpath/

xpath 可以根据元素的父节点或者哥哥弟弟节点定位到元素。

driver.find_element_by_xpath(‘//form[@id="sf"]//input[@type="text"]’)# 上级节点定位下级子节点

driver.find_element_by_xpath(‘//span[@class="enter-input"]/preceding-sibling::span/input’) # 通过节点的弟弟节点定位

注意：使用 xpath 最好不要使用工具获取，手写的可靠性更高

XPath 最初是用来在 XML 文档中定位 DOM 节点的语言，由于 HTML 也可以算作 XML 的一种实现，所以 Selenium 也可以利用 XPath 这一强大的语言来定位 Web 元素。XPath 在传统属性定位之外扩展了诸如“定位第三个多选框”等定位能力，以便应对没有 ID 或 name 属性的情况。**利用 Xpath 可以通过绝对路径，或者相对于一个可精确定位的元素的相对路径来定位。为了保证定位的健壮性，推荐使用相对路径和基于位置关系的定位。**

2.2.8 css 元素定位(强大，比 xpath 快)

css 定位元素比 xpath 块，id，name，class，tag name 都是转换为 css 后定位元素。具体请看 find_element 方法的代码。

<input id="**query**" class="**sec-input**" name="query" maxlength="100" autocomplete="off" type="text">

```
driver.find_elements_by_css_selector(".sec-input")  #  .表示 class
driver.find_elements_by_css_selector("#query")  #  #表示 id
```

CSS (Cascading Style Sheets) 是一种用于渲染 HTML 或者 XML 文档的语言，CSS 利用其选择器可以将样式属性绑定到文档中的指定元素，即前端开发人员可以利用 CSS 设定页面上每一个元素的样式。所以理论上说无论一个元素定位有多复杂，既然开发人员能够定位到并设置样式，那么测试人员同样应该也能定位继而操作该元素。这也正是 Selenium 官方极力推荐使用 CSS 定位，而不是 XPath 定位的主要原因。CSS 定位被推崇的另一个原因是不同的浏览器 XPath 引擎不同甚至没有自己的 Xpath 引擎，这就导致了 XPath 定位速度较慢，而采用 CSS 定位往往能用更简洁的语法快速定位到复杂的元素。

2.3 操作元素对象

2.3.1、元素的常用操作

element.click() # 单击元素；除隐藏元素外，所有元素都可单击
element.submit() # 提交表单；可通过 form 表单元素提交改表单
element.clear() # 清除元素的内容；如果可以的话
element.send_keys('需要输入的内容') # 模拟按键输入；只针对支持输入的元素
注意：send_keys() 输入的内容必须为字符串
搜狗查询实例：

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.maximize_window()  # 最大化
driver.get(r'https://www.sogou.com/')  # 打开网页
driver.find_element_by_id('query').send_keys('selenium')  # 搜索框输入 selenium
time.sleep(2)  # 等待 3 秒
driver.find_element_by_id('query').clear()  # 清除搜索框内容
time.sleep(2)
driver.find_element_by_id('query').send_keys('selenium')  # 重新输入内容
driver.find_element_by_id('sf').submit()  # 提交搜索框的表单
# driver.find_element_by_id('stb').submit()  #  标点提交也可以单击提交按钮
time.sleep(2)
driver.quit()  # 关闭浏览器
```

注意：submit() 提交表单，可以是提交按钮，也可以是表单元素，也可以是输入框元素

2.3.2 WebElement 常用方法

element.location 返回元素的坐标字典（相对于网页左上角 0,0 开始）
element.text 获取元素的文本，页面上看得到的文本
element.get_attribute('属性名称') 获得元素的属性 强调“有”
element.get_property('属性名称') 获得元素的固有属性值 强调“专”
element.is_displayed() 返回元素的结果是否可见，有些元素肉眼可见，但是他是隐藏的
示例：

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
```

```
driver.maximize_window()  # 最大化
driver.get(r'https://www.sogou.com/')  # 打开网页
driver.find_element_by_id('query').send_keys('selenium')  # 搜索框输入 selenium
element=driver.find_element_by_id('query')
print('搜索框的内容为：',element.get_attribute('value'))
print('搜索框的 class 属性：',element.get_attribute('class'))
print('搜索框的 type 属性：',element.get_attribute('type'))
print('搜索框的内容的位置：',element.location)
print('搜索框是否可操作：',element.is_displayed())
time.sleep(2)
text = driver.find_element_by_class_name('erwm-box').text  # 获取二位码的文本
print('底部二维码的文本为：',text)
time.sleep(2)
driver.quit()  # 关闭浏览器
```

2.3.2.a、element.location 获取元素的坐标位置

对于已加载到浏览器的底部元素，操作元素时现在 chrome 无法自动拖动滚动条，需要获取元素位置后，采用 js 拖动滚动条到相应位置采用操作元素。

2.3.2.b、element.text 获取元素的文本

```
<div class="erwm-box">
        <span class="ewm"></span>
        <div class="erwx">
            <p>搜狗搜索 APP</p>
            <p class="p2">搜你所想</p>
        </div>
    </div>
```

如上，我们定位 class="erwm-box"元素，获取到的文本是［搜狗搜索 APP 搜你所想］，也就是界面上能看到的文字内容。输入框除外（输入框的值是存储在 value 属性中），只要是界面上的文本内容都可以获取。
多用于校验点。

2.3.2.c、element.get_attribute('属性名称') 获取对应的属性值，强调“有”

```
<input type="text" class="sec-input" name="query" id="query" maxlength="100" autocomplete="off">
```

如上搜索输入框的属性有 type、class、name、id、maxlength、autocomplete；我都可以通过 get_attribute()获取到他的值，因为他’有’。
value 是特殊的属性，输入框，单项按钮，多选按钮多具有改属性。

2.3.2.d、element.get_property('属性名称') 获得元素的固有属性值，强调“专”

它与 get_attribute() 差别，get_property() 是获取元素的固有属性。
我们所有的元素都有特定固有属性，如 id、type、value 等。
当使用 get_attribute()无法获取到属性的值时，可使用 get_property()。

2.3.2.e、element.is_displayed() 判定改元素是否可见

当我们定位到了元素，但是无法操作时，可以看看他是否可见，不可见不一定就是在界面上消失了。

2.3.2.f、其他方法

element.size 元素的大小
element.is_enabled() 元素是否可用

element.is_selected()　元素是否被选中，用于检测复选框或单项按钮

2.4 鼠标事件（ActionChains）

click(element=None)　左击
context_click(element=None) 右击
double_click(element=None) 双击
move_to_element(element) 移动鼠标到元素中间（悬停）
drag_and_drop(source,target)　source 上按下左键拖动到 target 元素上
click_and_hold(element=None) 在元素上按下鼠标左键
release()　释放鼠标
perform()　执行 ActionChains 中存储的动作
element 有 None 默认值的表示不传入参数该动作在原地执行。

2.4.1 左击

示例 1：鼠标左键点击

```
action=ActionChains(driver)
action.click() # 在鼠标当前位置单击
action.perform() # 执行 action 存储的动作

# 鼠标在 '新闻' 元素位置单击
action.click(driver.find_element_by_link_text('新闻')).perform()
```

注意：action.click() 动作并未执行，它只是存储在 action 实例中，需要通过 action.perform()方法执行存储动作；鼠标键盘事件动作动作可以存储多个，然后一次性执行。如下执行 Cytl+C：

```
ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).perform()
```

2.4.2 右击

示例 2：鼠标右击

```
action=ActionChains(driver)
action.context_click().perform() # 在鼠标当前位置右击

# 鼠标在 '新闻' 元素位置右击
action.context_click(driver.find_element_by_link_text('新闻')).perform()
```

2.4.3 双击

示例 3：鼠标双击操作

```
action=ActionChains(driver)
action.double_click().perform() # 在鼠标当前位置双击

# 鼠标在 '新闻' 元素位置双击击
action.double_click(driver.find_element_by_link_text('新闻')).perform()
```

2.4.4 鼠标移动

示例 4：鼠标移动

```
action = ActionChains(driver)
element=driver.find_element_by_link_text('设置')
# 鼠标移动到 '新闻' 元素上的中心点
action.move_to_element(element).perform()

# 鼠标在原位置向 x 轴移动 xoffset、y 轴移动 yoffset；xoffset、yoffset 可为正负数
action.move_by_offset(-200,100).perform()

# 鼠标移动到 element 元素中心点偏移 xoffset、yoffset
action.move_to_element_with_offset(element,-500,600).perform()
```

action.move_by_offset(xoffset,yoffset) 这里需要注意，如果 xoffset 为负数，表示横坐标向左移动，yoffset 为负数表示纵坐标向上移动。而且如果这两个值大于当前屏幕的大小，鼠标只能移到屏幕最边界的位置。

鼠标移动操作在测试环境中比较常用到的场景是需要获取某元素的 flyover/tips，实际应用中很多 flyover 只有当鼠标移动到这个元素之后才出现，所以这个时候通过执行 move_to_element(to_element) 操作，就能达到预期的效果。

根据我个人的经验，这个方法对于某些特定产品的图标的 flyover/tips 也不起作用，虽然在手动操作的时移动鼠标到这些图标上面可出现 flyover，但当使用 WebDriver 来模拟这一操作时，虽然方法成功执行，但 flyover 却不出来。所以在实际应用中，还需要对具体的产品页面做相应的处理。

2.4.5 鼠标悬停

示例 5：鼠标悬停

```

action.click_and_hold().perform()  # 鼠标在当前位置按下并不释放
# 鼠标 在'设置' 上悬停
action.click_and_hold(driver.find_element_by_link_text('设置')).perform()
```

action.click_and_hold(element) 这个方法实际上是执行了两个动作，首先是鼠标移动到元素 element，然后再 click_and_hold, 所以这个方法也可以写成 action.move_to_element(element).click_and_hold()。

2.4.6 鼠标拖拽

示例 6：鼠标拖拽

```
source = driver.find_element_by_id("kw")  # 获取起始位置元素
target = driver.find_element_by_id("sk")  # 获取目标元素
# 将元素 source 拖动到 target 的位置
ActionChains(driver).drag_and_drop(source, target).perform()

# 鼠标拖拽动作，将 source 元素向 x、y 轴方向移动 (xoffset, yoffset) ，其中 xoffset 为横坐标，yoffset 为纵坐标。
ActionChains(driver).drag_and_drop_by_offset(source, -100, 100).perform()
```

在这个拖拽的过程中，已经使用到了鼠标的组合动作，首先是鼠标点击并按住 click_and_hold(source) 元素，然后执行鼠标移动动作 (move_to)，移动到 target 元素位置或者是 (xoffset, yoffset) 位置，再执行鼠标的释放动作 (release)。所以上面的方法也可以拆分成以下的几个执行动作来完成：

```
ActionChains(driver).click_and_hold(source).move_to_element(target).release().perform()
```

2.4.7 鼠标释放操

示例 7：鼠标释放操

```
action = ActionChains(driver)
action.release().perform()  # 释放按下的鼠标
```

2.5 键盘操作

对于键盘的模拟操作，ActionChains 类中有提供了按下 key_down(keys)、释放 key_up(keys)、按下并释放 send_keys(keys_to_send) 等方法。
键盘的操作有普通键盘和修饰键盘两种 。
普通键盘为常用字母数字等；修饰键盘为 Ctrl、Shift、Alt 等，修饰键盘一般和其他键组合使用的键。
使用键盘操作时需要引入 from selenium.webdriver.common.keys import Keys 包，Keys 包中含所有特殊用键。

2.5.1、普通键盘操作

键盘操作使用 send_keys(*keys_to_send)方法，该方法支持多个按键连续操作，如果需要对某个元素执行按键操作使用 send_keys_to_element(element, *keys_to_send)方法。具体使用如下示例：

```
from selenium.webdriver.common.keys import Keys
action = ActionChains(driver)
action.send_keys(Keys.SPACE).perform()  # 按下并释放空格键
action.send_keys(Keys.TAB).perform()  # 按下并释放 Tab 键
action.send_keys(Keys.BACKSPACE).perform()  # 按下并释放 Backspace 键
action.send_keys(Keys.BACKSPACE,Keys.SPACE).perform()  # 连续执行按键动作
action.send_keys(Keys.TAB).send_keys(Keys.TAB).perform() # 也可以这样组合

'''
针对某个元素发出某个键盘的按键操作，或者是输入操作
'''
element = driver.find_element_by_id('query')
# 对一元素使用键盘操作
action.send_keys_to_element(element, 'selenium').perform()
# 上面动作拆解为下面动作
action.click(element).send_keys('selenium').perform()
```

注意除了 ActionChains 类有 send_keys（*keys_to_send）方法外，WebElement 类也有一个 send_keys_to_element（*keys_to_send）方法，这两个方法对于一般的输入操作基本上相同，不同点在于以下几点：
第一：Actions 中的 send_keys（*keys_to_send）对修饰键操作后并不会释放，也就是说当调用 actions.send_keys(Keys.ALT)、 actions.send_keys(Keys.CONTROL)、 action.send_keys(Keys.SHIFT) 的时候，相当于调用 actions.key_down(keys_to_send)，而如果在现实的应用中想要模拟按下并且释放这些修饰键，应该先 action.reset_actions()重设 action，然后再调用 action.send_keys(keys.NULL).perform()取消按下的修饰键。
第三点，在 WebDriver 中，我们可以使用 WebElement 类的 send_keys() 来上传附件，比如 element.send_keys(“D:\\test\\uploadfile\\test.jpg”)上文件，但不能使用 ActionChains 来上传附件，因为 type=’file’ 的输入框并不支持键盘输入。

2.5.2、修饰键的使用

修饰键是键盘上的一个或者一组特别的键，当它与一般按键同时使用时，用来临时改变一般键盘的普通行为。
修饰键一般跟普通键组合使用，比如 Ctrl+A、Alt+F4 等。
我们电脑中的修饰键一般有以下几种修：Ctrl、Alt(Option)、Shift、AltGr、Windows logo、Command、FN(Function)。

一般使用的都是前三种。
对于修饰键的使用在 Python selenium 中一般使用按下 key_down(keys)、释放 key_up(keys)、按下并释放 send_keys(keys_to_send)组合实现。

```
action = ActionChains(driver)
action.key_down(Keys.CONTROL).perform() # 按下 ctrl 键
action.key_up(Keys.CONTROL).perform() # 释放 ctrl 键

action.key_down(Keys.SHIFT).perform() # 按下 shift 键
action.key_up(Keys.SHIFT).perform() # 释放 shift 键

action.key_down(Keys.ALT).perform() # 按下 alt 键
action.key_up(Keys.ALT).perform() # 释放 alt 键
```

示例：通过 ctrl+c 来复制文本

```
ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).perform()
```

2.5.3、WebElement.send_keys()键盘操作

WebElement 元素对象下的 send_keys 也支持组合键盘操作。
代码示例如下：

```
element = driver.find_element_by_id('query')
element.send_keys('selenium')
element.send_keys(Keys.BACK_SPACE)  # 按 BACKSPACE 删除一个字符
element.send_keys(Keys.SPACE)  # 空格键(Space)
element.send_keys(Keys.CONTROL, 'a')  # 全选（Ctrl+A）
element.send_keys(Keys.CONTROL, 'c')  # 复制（Ctrl+C）
element.send_keys(Keys.CONTROL, 'v')  # 粘贴（Ctrl+v）
element.send_keys(Keys.TAB)  # 制表键(Tab)
element.send_keys(Keys.ESCAPE)  # 回退键（Esc）
element.send_keys(Keys.ENTER)  # 回车键（Enter）
```

2.6 设置等待时间

time.sleep(3)　　固定等待 3 秒
driver.implicitly_wait(10)　　隐性的等待，对应全局
WebDriverWait(driver, timeout).until(‘有返回值的__call__()方法或函数’)　显性的等待，对应到元素

2.6.1 time.sleep(seconds) 固定等待

```
import time
time.sleep(3) #等待 3 秒
```

time.sleep(seconds)　seconds 参数为整数，单位（秒）。
它是 Python 的 time 提供的休眠方法。
常用于短时间的等待，为了自动测试用例的执行效率固定等待的时间需要控制在 3 秒内。在用例中尽量少用固定等待。

2.6.2 智能隐性的等待（回应超时等待）

driver.implicitly_wait(time_to_wait)

回应超时等待，隐性的，设置后对应的是全局，如查找元素。

```
driver.implicitly_wait(10)  # 设置全局隐性等待时间，单位秒
```

每次 driver 执行 找不到元素都会等待设置的时间，它的值设置的过长对用例执行效率有很大的影响，必须在执行完成之后还原回来。driver.implicitly_wait() 要慎之又慎的使用。
driver 对它的默认值为 0，driver.implicitly_wait(0)能还原隐性等待的设置时间。

2.6.3 智能显性等待 WebDriverWait

WebDriverWait(driver, timeout, poll_frequency=0.5, ignored_exceptions=None)
参数说明：

driver 为 webdriver 驱动
timeout 最长超时时间，单位(秒)
poll_frequency 循环查找元素每次间隔的时间，默认 0.5 秒
ignored_exceptions 超时后需要输出的异常信息

WebDriverWait()下面有两个方法可用 until()和 until_not()
WebDriverWait(driver, timeout).until(method, message='')
method 函数或者实例__call__()方法返回 True 时停止，否则超时后抛出异常。
参数说明：

method 在等待时间内调用的方法或者函数，该方法或函数需要有返回值，并且只接收一个参数 driver。
message 超时时抛出 TimeoutException，将 message 传入异常显示出来

WebDriverWait(driver, timeout).until_not(method, message='')
于上面的 until() 相反，until_not 中的 method 函数或者实例__call__()方法返回 False 结束，否则抛出异常。

until 方法使用的 method 的函数或者类__call()__方法详解：

函数我们一般采用匿名函数 lambda 。
lambda driver:driver.find_element(<定位元素>) # 当定位的元素时为 True，无元素时为 False。
如示例 1、2：

WebDriverWait 示例 1：

```
WebDriverWait(driver,5).until(lambda driver:driver.find_element_by_id('query'))
```

5 秒内等待元素(id='query')出现，lambda driver:driver.find_element_by_id('query') 为一个匿名函数，只有一个 driver 参数，返回的是查找的元素对象。

WebDriverWait 示例 2：

```
WebDriverWait(driver, 5).until_not(
lambda driver:driver.find_element_by_name('query'))
```

5 秒内等待元素消失，同示例 1 until_not 要求无元素返回即元素不存在于该页面。

定义类中的__call()__方法。

```
class wait_element(object):
    def __init__(self, locator):
        self.locator = locator

    def __call__(self, driver):
        return driver.find_element(self.locator)
```

```
WebDriverWait(driver, 5).until(wait_element((By.ID, 'query')))   # 等待元素出现
WebDriverWait(driver, 5).until_not(wait_element((By.ID, 'query')))   # 等待元素消失
```

wait_element 类中__init__()方法接收需要定位的元素，__call__()方法中只能有唯一变量 driver，并且返回元素对象。这样做是是不是很麻烦，其实 selenium 提供的一个库进行操作，expected_conditions 库。引入位置
from selenium.webdriver.support import expected_conditions as ec，它囊括了我们需要使用等待的大部分情况。

2.6.4 expected_conditions 类库

```
from selenium.webdriver.support import expected_conditions as ec   # 引入包
```

下面示例都是以搜狗搜索首页为例。
方法中参数说明 locator=(By.ID, 'id') 表示使用 By 方法定位元素的元组，element 表示获取的 webElement 元素对象。
ec.title_is('title') 判断页面标题等于 title
ec.title_contains('title'') 判断页面标题包含 title

```
WebDriverWait(driver, 10).until(ec.title_is('搜狗搜索引擎 - 上网从搜狗开始'))   # 等待 title 于参数相等
WebDriverWait(driver, 10).until(ec.title_contains('搜狗搜索引擎'))   # 等待 title 包含 参数的内容
```

ec.presence_of_element_located(locator) 等待 locator 元素是否出现
ec.presence_of_all_elements_located(locator) 等待所有 locator 元素是否出现

```
WebDriverWait(driver, 10).until(ec.presence_of_element_located((By.ID, 'query')))
WebDriverWait(driver, 10).until(ec.presence_of_all_elements_located((By.ID, 'query')))
```

ec.visibility_of_element_located(locator) 等待 locator 元素可见
ec.invisibility_of_element_located(locator) 等待 locator 元素隐藏
ec.visibility_of(element) 等待 element 元素可见

```
WebDriverWait(driver, 10).until(ec.visibility_of_element_located((By.ID, ''stb'')))   # 等待元素可见
WebDriverWait(driver, 10).until(ec.invisibility_of_element_located((By.ID, ''stb'')))   # 等待元素隐藏
WebDriverWait(driver, 10).until(ec.visibility_of(driver.find_element_by_link_text('高级搜索')))   # 等待元素可见
```

ec.text_to_be_present_in_element(locator,text) 等待 locator 的元素中包含 text 文本
ec.text_to_be_present_in_element_value(locator,value) 等待 locator 元素的 value 属性为 value

```
WebDriverWait(driver, 10).until(ec.text_to_be_present_in_element((By.ID, 'erwx'), '搜索 APP'))   # 等待元素中包含搜索 APP 文本
WebDriverWait(driver, 10).until(ec.text_to_be_present_in_element_value((By.ID, 'query'),'selenium'))   # 等待元素的值为 selenium，一般用于输入框
```

ec.frame_to_be_available_and_switch_to_it(locator) 等待 frame 可切入

```
WebDriverWait(driver, 10).until(ec.frame_to_be_available_and_switch_to_it((By.ID, 'frame')))
WebDriverWait(driver, 10).until(ec.frame_to_be_available_and_switch_to_it('frameid'))
```

ec.alert_is_present() 等待 alert 弹出窗口出现

```
WebDriverWait(driver, 10).until(ec.alert_is_present())
```

ec.element_to_be_clickable(locator) 等待 locator 元素可点击

```
WebDriverWait(driver, 10).until(ec.element_to_be_clickable((By.ID, 'kw')))
```

等待元素被选中，一般用于复选框,单选框
ec.element_to_be_selected(element) 等待 element 元素是被选中
ec.element_located_to_be_selected(locator) 等待 locator 元素是被选中 ec.element_selection_state_to_be(element, is_selected) 等待 element 元素的值被选中为 is_selected(布尔值)
ec.element_located_selection_state_to_be(locator,is_selected) 等待 locator 元素的值是否被选中 is_selected(布尔值)

```
from selenium import webdriver
import time
from selenium.webdriver.support.wait import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as ec   # 引入包
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'https://www.baidu.com/')
driver.find_element_by_link_text('设置').click()
WebDriverWait(driver, 3).until(ec.element_to_be_clickable((By.LINK_TEXT, '搜索设置')))   # 等待搜索可点击，不可缺少
driver.find_element_by_link_text('搜索设置').click()
element = driver.find_element_by_id('s1_1')
WebDriverWait(driver, 2).until(ec.element_to_be_selected(element))   # element 被选中
WebDriverWait(driver, 10).until(ec.element_located_to_be_selected((By.ID, 'SL_0'))) # id=’SL_0’ 被选中
WebDriverWait(driver, 10).until(ec.element_selection_state_to_be(element,True )) # element  被选中
WebDriverWait(driver, 10).until(ec.element_located_selection_state_to_be((By.ID, 'SL_1'), False)) #   id=’SL_1’不被选中
time.sleep(3)
driver.quit()
```

2.6.5 什么时候使用等待

固定等待 sleep 与隐性等待 implicitly_wait 尽量少用，它会对测试用例的执行效率有影响。
显性的等待 WebDriverWait 可以灵活运用，什么时候需要用到？
1、页面加载的时候，确认页面元素是否加载成功可以使用 WebDriverWait
2、页面跳转的时候，等待跳转页面的元素出现，需要选一个在跳转前的页面不存在的元素
3、下拉菜单的时候，如上百度搜索设置的下拉菜单，需要加上个时间断的等待元素可点击
4、页面刷新的时候
总之，页面存在改变的时候；页面上本来没的元素，然后再出现的元素

2.7 层级定位

在实际测试过程中，一个页面可能有多个属性基本相同的元素，如果要定位到其中的一个，这时候需要用到层级定位。先定位其父元素，然后再通过父元素定位该元素。

2.7.1 多次分开定位

示例：多次分开定位百度 设置 -->搜索设置 下拉菜单

```
from selenium import webdriver    # 引入包
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Chrome()    # 实例化浏览器
driver.maximize_window()    # 最大化浏览器
driver.get('http://www.baidu.com')
WebDriverWait(driver, 10).until(lambda driver: driver.find_element_by_link_text('设置'))
ele = driver.find_element_by_link_text('设置')    # 获取设置元素
ActionChains(driver).move_to_element(ele).perform()    # 鼠标移动到设置元素
# driver.find_element_by_link_text('设置').click() #单击设置元素
WebDriverWait(driver, 3).until(lambda driver: driver.find_element_by_link_text('搜索设置'))    # 等待搜索元素出现
driver.find_element_by_link_text('搜索设置').click()
```

2.7.2 二次定位

示例 2：二次定位百度登录用户名

```
#通过二次定位找到用户名输入框 tang-content 定位整个窗口
div=driver.find_element_by_class_name("tang-content").find_element_by_name("userName")
div.send_keys("username")
```

示例 3：通过层级定位搜狗输入框

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'https://www.sogou.com/')
form_element = driver.find_element_by_id('sf')  # 获取 form 表单元素
form_element.find_element_by_id('query').send_keys('selenium')  # 通过表单定位输入框
form_element.find_element_by_id('stb').click()  # 通过表单定位搜索按钮
time.sleep(3)
driver.quit()
```

如上代码，我们先定位到了 form 表单，然后通过表单定位下面的输入框与按钮。

注意：上面示例只通过两层定位到了元素，层级定位不一定定位两次，我们可以定位多次。

2.8 定位一组元素

2.8.1 定位一组元素

而当我们需要获取多个属性相同的对象，并且需要批量操作该对象时，就会使用 find_elements 定位一组元素。

创建以下 html 文件，文件名 checkbox.html

```
<form class="form-horizontal">
    <div class="control-group">
        <label class="controllabel" for="China">中国人</label>
        <div class="controls">
            <input type="checkbox" id="China"/>
        </div>
    </div>
    <div class="control-group">
        <label class="controllabel" for="American">美国人</label>
        <div class="controls">
```

```
            <input type="checkbox" id="American"/>
        </div>
    </div>
    <div class="control-group">
        <label class="controllabel" for="German">德国人</label>
        <div class="controls">
            <input type="checkbox" id="German"/>
        </div>
    </div>
    <div class="button">
        <input type="submit" id="submit"/>
    </div>
</form>
```

示例：全选上面的多选框

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'E:\xxx\checkbox.html') # 打开 checkbox.html 文件，使用绝对地址
checkboxs = driver.find_elements_by_xpath('//input[@type="checkbox"]')  # 获取批量的对象
for checkbox in checkboxs:  # 循环控制
    if not checkbox.is_selected():  # 判断多选框是否被选中
        checkbox.click()  # 单击
time.sleep(3)
driver.quit()
```

2.8.2 综合运用

当我们需要定位一组元素时，页面上相似的元素会很多，这时我们需要和层级一定一起使用。先定位到该组元素的父元素，然后通过父元素定位其子孙元素。

示例 1：获取搜狗微信页面搜索热词的内容

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'http://weixin.sogou.com/')
topele = driver.find_element_by_id('topwords')  # 搜索热词的父元素
tops = topele.find_elements_by_tag_name('a')  # 二次批量定位热词元素
for top in tops:  # 循环获取元素
    print(top.text)  # 打印文本内容
driver.quit()
```

UI 自动化测试中对于表格的定位是个难点，怎么样快速获取表格数据，请看下面几个示例。

示例 2：定位表格获取表头

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'http://www.w3school.com.cn/cssref/css_selectors.asp')
# //table[@class="dataintable"]//tr[1]//th  获取表头元素
table_header = driver.find_elements_by_xpath('//table[@class="dataintable"]//tr[1]//th')
for header in table_header:  # 循环获取元素
```

```
    print(header.text)  # 打印文本内容
driver.quit()
```

示例 3：定位表格第二列数据内容

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'http://www.w3school.com.cn/cssref/css_selectors.asp')
# //table[@class="dataintable"]//tr[y]//td[x]    y 第几条记录，x 第几列数据
# //table[@class="dataintable"]//tr//td[2]  获取第二列数据
table_header = driver.find_elements_by_xpath('//table[@class="dataintable"]//tr//td[2]')
for header in table_header:  # 循环获取元素
    print(header.text)  # 打印文本内容
driver.quit()
```

示例 4：获取表格中所有的数据

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(r'http://www.w3school.com.cn/cssref/css_selectors.asp')
# //table[@class="dataintable"]//tr  定位所有行
tables = driver.find_elements_by_xpath('//table[@class="dataintable"]//tr')
for tr in tables:  # 循环每行元素
    for td in tr.find_elements_by_tag_name('td'):  # 循环获取列
        print(td.text, end='\t\t')
    print('\n')
driver.quit()
```

定位表格，采用 find_elements 组定位，使用 xpath=//table//tr[y]//td[x]（y 第几条记录，x 第几列数据），当 y 或者 x 其中一个没有值时定位一行或一列。

2.9 定位 frame 中的对象

< frame> <iframe> 标签，浏览器会在标签中打开一个特定的页面窗口（框架），它在本窗口中嵌套进入一个网页，当用 selenium 定位页面元素的时候会遇到定位不到 frame 框架内元素的问题 frame 框架内的元素。

定位 frame 中的元素前我们需要 driver.switch_to.frame()切换到对应的 frame 中，执行操作后要操作 frame 框架外的元素，需要通过 driver.switch_to.default_content()切换回主文档页面。

driver.switch_to.frame(index/id/name/WebElement) 切入 frame 裤架中，参数可以为 id/name/index
driver.switch_to.parent_frame() 切换回当前 frame 的上一层，如果当前已是主文档，则无效果
driver.switch_to.default_content() 切换回主文档

创建如下两个 html 文件，两个文件放入同一个文件夹内

frame.html

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8"/>
```

```
    <title>frame</title>
</head>
<body>
<div class="row-fluid">
    <label>frame 外输入框</label>
    <input type='text' id="frameinput">
    <div class="span10 well">
        <h3>frame</h3>
        <iframe id="f1" name="frame2" src="inner_frame.html" width="800" height="600"></iframe>
    </div>
</div>
</body>
</html>
```

inner_frame.html

```
<html>
<head>
    <title>inner frame</title>
</head>
<body>
<label id="innerlable">frame1 内多选按钮 </label>
<input type="checkbox" id="innercheck" name="inner">
<div class="row-fluid">
    <h3>inner frame</h3>
    <iframe id="f2" name="frame2" src="http://m.baidu.com/" width="700" height="400">
    </iframe>
</div>
</body>
</html>
```

示例：操作主文档的元素 --> 切换到外层 frame 操作外层 frame 的元素 --> 切换到内层 frame 操作内层的元素 --> 切换回外层 frame 操作外层 frame 的元素 --> 再次切入内层 frame 操作元素 --> 切换回主文档操作文档元素

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.get(r'E:\frame.html')  # 打开 frame.html 页面，注意修改为你的位置
driver.find_element_by_id('frameinput').send_keys('操作 frame 外的元素')
driver.switch_to.frame(0)  # 根据 index 切换，从 0 开始
text = driver.find_element_by_id('innerlable').text
print(text)
driver.find_element_by_id('innercheck').click()
driver.switch_to.frame('f2')  # 根据 id 切入 内层 frame
driver.find_element_by_id('index-kw').send_keys('selenium frame')
driver.switch_to.parent_frame()  # 切换到上一层表单
driver.find_element_by_id('innercheck').click()
driver.switch_to.frame('frame2')  # 根据 name 再次切入内层 frame
driver.find_element_by_id('index-bn').click()
```

```
driver.switch_to.default_content() # 切换回主文档
driver.find_element_by_ta('frameinput').clear()
driver.switch_to.frame(driver.find_elements_by_tag_name('iframe'))  # 通过webelement切换
driver.find_element_by_id('innercheck').click()
time.sleep(3)
driver.quit()
```

2.9.1、 driver.switch_to.frame(frame_reference)切换进入 frame

switch_to_frame() 将淘汰使用，建议使用 switch_to.frame()。
switch_to.frame() 切换表单支持 4 参数形式切换，元素的 frame 的 index，frame 元素的 id 或 name 属性，frame 元素的 WebElement 对象。
通常采用 id 和 name 就能够解决绝大多数问题。但有时候 frame 并无这两项属性，则可以用 index 和 WebElement 来定位：

index 从 0 开始，整型参数，根据同层 frame 的顺序定位

WebElement 对象，即 find_element 方法所取得的对象，我们可以用 tag_name、xpath 等来定位 frame 对象。
如上示例的：driver.switch_to.frame(driver.find_elements_by_tag_name('iframe'))

2.9.2、 driver.switch_to.default_content() 切换回主文档

切到 frame 中之后，我们便不能继续操作主文档的元素，这时如果想操作主文档内容，则需切回主文档。

```
driver.switch_to.default_content() # 切换到主文档中。
```

注意：很多人都会忘记这步操作

2.9.3、driver.switch_to.parent_frame() 切换到上一层表单

```
<html>
    <iframe id="frame1">
        <iframe id="frame2" / >
    </iframe>
</html>
```

嵌套 frame 很少会遇到，如下 frame1 为外层，frame2 嵌套在 frame1 中。

a. 从主文档切到 frame2，一层层切进去

```
driver.switch_to.frame("frame1")
driver.switch_to.frame("frame2")
```

b. 从 frame2 再切回 frame1，selenium 提供了一个方法能够从子 frame 切回到父 frame，而不用我们切回主文档再切进来。

```
driver.switch_to.parent_frame()  # 如果当前已是主文档，则无效果
```

parent_frame()这个相当于后退的方法，我们可以随意切换不同的 frame，随意的跳来跳去了。

2.10 alert/confirm/prompt 弹出窗口处理

2.10.1 Alert/Confirm/Prompt 弹出窗口特征说明

Alert 弹出窗口：
提示用户信息只有确认按钮，无法通过页面元素定位，不关闭窗口无法在页面上做其他操作。

Confirm 弹出窗口：

有确认和取消按钮，该弹出窗口无法用页面元素定位，不关闭窗口无法在页面上做其他操作。

Prompt 弹出窗口：

有输入框、确认和取消按钮，该弹出窗口无法用页面元素定位，不关闭窗口无法在页面上做其他操作。

注意：3 种窗口为浏览器自带的窗口，该窗口无法定位到元素，能定位到元素需要使用 WebElement 操作。

2.10.2、Alert/Confirm/Prompt 弹出窗口操作

第一步：需要获取弹出窗口，两种方法 与 Alert(driver)

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

或

```
from selenium.webdriver.common.alert import Alert
alert=Alert(driver)
```

第二步：对获取到的窗口进行操作，常用方法如下：

```
alert.text()  # 获取窗口信息
alert.accept()  # 确认
alert.dismiss()  # 取消
alert.send_keys(keysToSend)  # 输入信息
alert.authenticate(username, password)  # 用户认证信息登录，已有确认操作
```

2.10.3、实例说明

创建下面 3 个 html 文件

alertTest.html

```
<html>
```

```
<head>
    <title>Alert Test</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
</head>
<script type="text/javascript">
function showAlert(){
    alert(document.from1.t1.value);
}
function showMultilineAlert(){
    alert("你必须纠正以下错误:\n 你必须输入 XXXX.\n 你必须做 XXXX.\n 你必须 XXXX");
}
</script>
<body>
<h2>Alert Test</h2>
<form name="from1">
    <input type="text" name="t1" value="可以输入  Alert  信息"><br><br>
    <input type="button" name="button1" value="点击 Alert 获取输入框信息" onclick="showAlert()"><br><br>
    <input type="button" name="button2" value="Alert 自带多行文本信息" onclick="showMultilineAlert()"><br>
</form>
</body>
</html>
```

confirmTest.html

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <title>Confirm Test</title>
</head>
<script type="text/javascript">
function showConfirm(){
    var t1 = document.from1.t1;
    if (confirm("请点击确认或取消")){
        t1.value = "确认";
    }else{
        t1.value = "取消";
    }
}
</script>
<body>
<h2>Confirm Test</h2>
<form name="from1">
    <input type="button" name="button1" value="点击 Confirm 按钮" onclick="showConfirm()"><br><br>
    <input type="text" name="t1">
</form>
</body>
</html>
```

promptTest.html

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <title>Prompt Test</title>
</head>
<script type="text/javascript">
    function showPrompt(){
        var t1 = document.from1.t1;
        t1.value = prompt("请输入信息，信息将填入页面输入框.");
    }
</script>
<body>
<h2>Prompt Test</h2>
<form name="from1">
    <input type="button" name="button1" value="点击 Prompt 按钮" onclick="showPrompt()"><br><br>
    <input type="text" name="t1">
</form>
</body>
</html>
```

示例 1：Alert 弹窗获取文本与确认操作

```
from selenium import webdriver
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support.expected_conditions import alert_is_present
from selenium.webdriver.common.alert import Alert

driver = webdriver.Chrome()
driver.get(r'E:\XXX\Html\alertTest.html')
driver.find_element_by_name('button1').click()  # 点击第一个按钮
WebDriverWait(driver, 5).until(alert_is_present())  # 等待弹出窗口出现
alert = driver.switch_to.alert  # 获取弹出窗口
text1 = alert.text  # 获取窗口文本信息
print(text1)  # 打印窗口文本信息
alert.accept()  # 确认
print('----------')
driver.find_element_by_name('button2').click()  # 点击第二个按钮
WebDriverWait(driver, 5).until(alert_is_present())  # 等待弹出窗口出现
alert = Alert(driver)  # 获取弹出窗口
text1 = alert.text  # 获取窗口文本信息
print(text1)  # 打印窗口文本信息
alert.accept()  # 确认
driver.quit()
```

注意：WebDriverWait(driver, 5).until(alert_is_present()) 加上这个可提高代码的可靠性

示例 2：Comfirm 弹窗获取文本、确认、取消操作

```
driver = webdriver.Chrome()
driver.get(r'E:\XXX\Html\confirmTest.html')
driver.find_element_by_name('button1').click()  # 点击按钮
```

```
WebDriverWait(driver, 5).until(alert_is_present())  # 等待弹出窗口出现
alert = driver.switch_to.alert  # 获取弹出窗口
print(alert.text)  # 打印窗口信息
alert.accept()  # 确认
time.sleep(2)

driver.find_element_by_name('button1').click()  # 点击按钮
WebDriverWait(driver, 5).until(alert_is_present())  # 等待弹出窗口出现
alert = driver.switch_to.alert  # 获取弹出窗口
alert.dismiss()  # 取消
time.sleep(2)
driver.quit()
```

示例 3：Prompt 弹窗获取文本、输入内容、确认操作

```
driver = webdriver.Chrome()
driver.get(r'E:\XXX\promptTest.html')
driver.find_element_by_name('button1').click()  # 点击按钮
WebDriverWait(driver, 5).until(alert_is_present())  # 等待弹出窗口出现
alert = Alert(driver)  # Alert 获取弹出窗口
alert.send_keys('selenium Alert 弹出窗口输入信息')  # 输入信息
alert.accept()  # 确认
time.sleep(2)
driver.quit()
```

2.11 浏览器多窗口处理

有时 web 应用会打开多个浏览器窗口，当我们要定位新窗口中的元素时，我们需要将 webDriver 的 handle（句柄）指定到新窗口。

什么意思？

假设我们打开 web 应用，在系统运行过程中重新打开一个新窗口（可以是页签，当前浏览器存在两个窗口），这时我们 webDriver 对浏览器的操作指针（句柄）还再原窗口，如果需要操作新窗口元素就要将 handle 句柄切换到新窗口。

常用方法：

driver.current_window_handle 获取当前窗口 handle

driver.window_handles 获取所有窗口的 handle，返回 list 列表

driver.switch_to.window(handle) 切换到对应的窗口

driver.close() 关闭当前窗口

示例：进入搜狗搜索 --> 打开搜狗输入法页面 --> 输入法页面查看皮肤 --> 关闭搜狗搜索页面 --> 输入法页面查看词库

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.get('https://www.sogou.com/')
driver.find_element_by_link_text('搜狗输入法').click()
nowhandle = driver.current_window_handle  # 获得当前窗口
```

```
print(driver.current_window_handle)
print('当前窗口为：', driver.title)
allhandles = driver.window_handles  # 获得所有窗口
for handle in allhandles:  # 循环判断窗口是否为当前窗口
    if handle != nowhandle:
        driver.switch_to.window(handle)  # 切换窗口
        time.sleep(2)
print('切换到窗口：', driver.title)
driver.find_element_by_link_text('皮肤').click()  # 操作新窗口元素
driver.switch_to.window(nowhandle)  # 回到原先的窗口
print('回到原来窗口：', driver.title)
driver.close()  # 关闭当前窗口,关闭的是当前的窗口
driver.switch_to.window(driver.window_handles[0])  # 再次切换窗口
driver.find_element_by_link_text('词库').click()
time.sleep(5)
driver.quit()
```

在自动化测试中切换窗口页面一般会封装为函数，如下示例：

```
def switch_to_window(driver, winB):
    """
    :param winB:
        1.切换窗口的标题
        2.切换窗口的序号
        3.切换页面的元素
    :return: True  切换成功
    :Usage:
    driver.switch_to_window('win_name')
    driver.switch_to_window(2) #  切换到第二个窗口
    located=(By.ID,'id') #  确定切换页面的元素
    driver.switch_to_window(located) #  切换到页面中存在 located 的元素窗口
    """
    result = False
    handles = driver.window_handles
    current_handle = driver.current_window_handle
    if isinstance(winB, tuple):
        for handle in handles:
            driver.switch_to.window(handle)
            time.sleep(2)
            try:
                driver.find_element(*winB)
            except NoSuchElementException:
                pass
            else:
                result = True
                break
        if not result:
            driver.switch_to.window(current_handle)
            time.sleep(2)
```

```
    elif isinstance(winB, str):
        for handle in handles:
            driver.switch_to.window(handle)
            time.sleep(2)
            if winB in driver.title:
                result = True
                break
        if not result:
            driver.switch_to.window(current_handle)
            time.sleep(2)
    elif isinstance(winB, int):
        if winB <= len(handles):
            driver.switch_to.window(winB - 1)
            time.sleep(2)
            result = True
    else:
        print('参数错误')
    return result
```

2.12 下拉框元素定位

选择获取反选下拉框元素首先要实例化 select 元素

```
from selenium.webdriver.support.ui import Select #  引入包
select_element=Select(element)    #  实例化 select
```

三种常用选择方法

select_element.select_by_index(index) 根据 index 定位，从 0 开始

select_element.select_by_value(value) 根据 value 属性定位

select_element.select_by_visible_text(text) 根据文本定位

反选的方法

select_element.deselect_by_index(index) 根据 index 定位，从 0 开始

select_element.deselect_by_value(value) 根据 value 属性定位

select_element.deselect_by_visible_text(text) 根据文本定位

select_element.deselect_all() 取消全部选择

获取选项的值

select_element.options 返回这个 select 元素所有的 options

select_element.all_selected_options 所有被选中的 options

select_element.first_selected_option 第一个被选中的 option

创建 select.html 文件，代码如下：

```
<form>
    <select id="s1Id">
        <option></option>
        <option value="o1" id="id1">o1</option>
        <option value="o2" id="id2">o2</option>
        <option value="o3" id="id3">o3</option>
    </select>
```

```
    <br/><br/>
    <select id="s2Id" multiple="multiple" size="6">
        <option></option>
        <option value="o1val">o1</option>
        <option value="o2val">o2</option>
        <option value="o3val">o3</option>
        <option value="o4val"> With spaces</option>
        <option value="o4val">    With nbsp</option>
    </select>
</form>
```

采用三种方法依次选择's1Id'的值

```
driver.get('http://sahitest.com/demo/selectTest.htm')
s1 = Select(driver.find_element_by_id('s1Id'))  # 实例化 Select
s1.select_by_index(1)  # 选择第二项选项：o1
s1.select_by_value("o2")  # 选择 value="o2"的项
s1.select_by_visible_text("o3")  # 选择 text="o3"的值，即在下拉时我们可以看到的文本
```

想查看一个 select 所有的选项

```
s1 = Select(driver.find_element_by_id('s1Id'))
for select in s1.options:
    print (select.text)
```

想查看我已选中的选项

```
s4 = Select(driver.find_element_by_id('s2Id'))
s4.select_by_index(1)  # 根据顺序选择第 2 个
s4.select_by_value("o2val")  # 根据 value 属性选择 o2val
s4.select_by_visible_text("With spaces")  # 根据展示文本选择 With spaces
s4.select_by_visible_text(u"       With nbsp") # 注意空格，只有为  是才用空格
for select in s4.all_selected_options:  # 循环获取所有的值
    print(select.text)
```

注意：只有只有为 的空格才是空格，在 html 中代码只有 才代表空格，代码中的空格不是空格。

想要查看选择框的默认值，或者我选中的值

```
s1 = Select(driver.find_element_by_id('s1Id'))
print(s1.first_selected_option.text)  # 查看选择默认值
s1.select_by_value("o2")
print (s2.first_selected_option.text)  # 查看选中的值
```

取消选择

```
s4 = Select(driver.find_element_by_id('s2Id'))
s4.select_by_index(1)
s4.select_by_value("o2val")
s4.select_by_visible_text("With spaces")
s4.select_by_visible_text(u"       With nbsp")
s4.deselect_by_index(1)  # 根据值顺序取消选择
s4.deselect_by_value("o2val")  # 根据 value 属性取消选择
s4.deselect_by_visible_text("With spaces")  # 根据文本取消选择
```

```
s4.select_by_value("o2val")
s4.select_by_index(1)
s4.deselect_all()  # 取消全部选择
```

2.13 上传文件

selenium 无法识别非 web 的控件，上传文件窗口为系统自带，无法识别窗口元素。
上传文件有两种场景：input 控制上传和非 input 控件上传。
大多数情况都是 input 控件上传文件，只有非常少数的使用自定义的非 input 上传文件。

2.13.1、input 控件上传文件

查看长传文件的页面元素标签，如果为 input 表明是通过 input 控件上传文件。我们可以直接采用 WebElement.send_keys('文件地址') 长传文件。
创建 html 文件，如下：
upload.html

```
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <title>upload file</title>
</head>
<body>
<h3>upload file</h3>
<input type="file" name="file"/>
</body>
</html>
```

示例：长传 C:\install.log 文件。

```
from selenium import webdriver
import time
driver = webdriver.Chrome()
driver.get(r'E:\XXXX\Html\upload.html')  # 文件的地址
driver.find_element_by_name('file').send_keys(r'C:\install.log')  # 上传文件
time.sleep(2)
driver.quit()
```

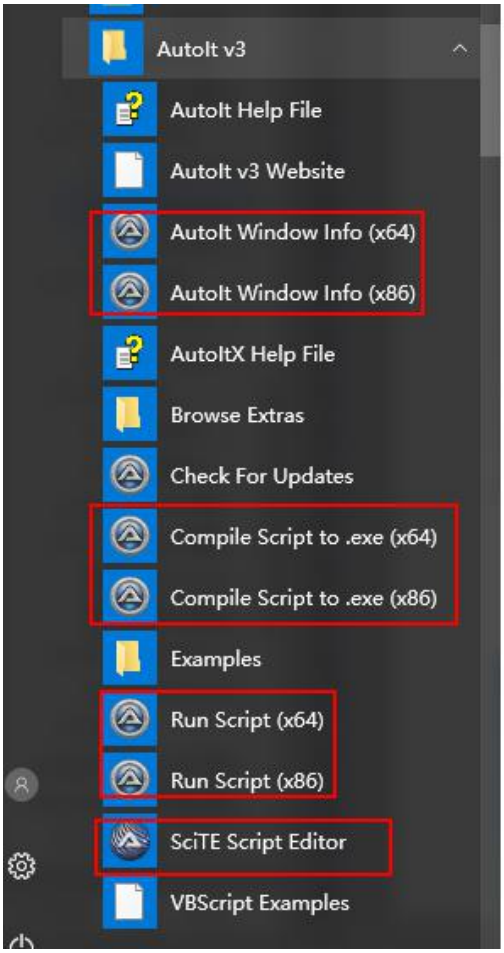
2.13.2、非 input 控件上传文件

非 input 控件上传文件，我们要引入外部插件上传。也有两种方法一种通过 pywin32 上传，另一种是通过 autoit 上传。这里我们只会讲到 autoit 上传文件。
下载安装 autoit，使用 autoit Windows info 识别控件，获取输入框和打开按钮的 Class，instance，Classname 3 个属性。
autoit 官网：https://www.autoitscript.com/site/

1、autoit 简介

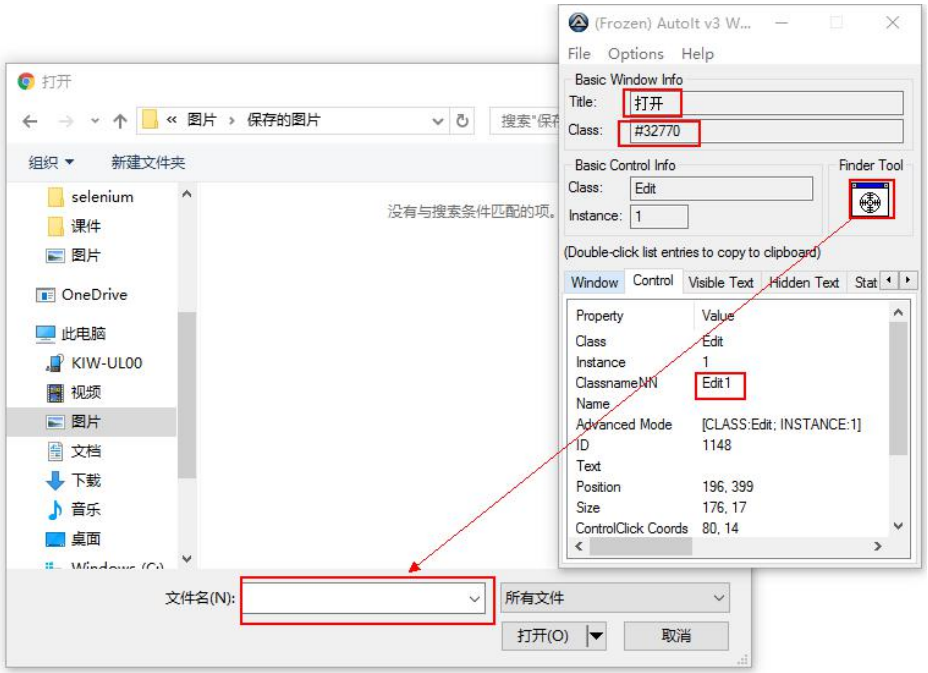
开始菜单如图：
AutoIt Windows Info　用于帮助我们识 Windows 控件信息。
Compile Script to.exe　用于将 AutoIt 生成 exe 执行文件。
Run Script　用于执行 AutoIt 脚本。

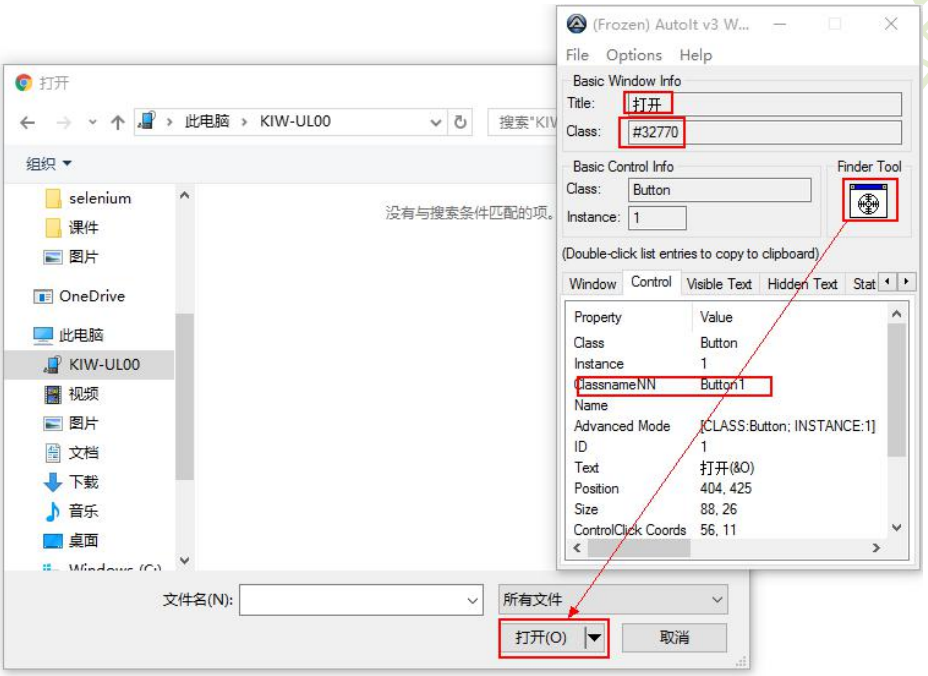
SciTE Script Editor　　用于编写 AutoIt 脚本。



2、autoit 使用

第一步：使用 AutoIT Window Info 获取窗口的 Title、class 属性， 获取控件的 ClassnameNN 属性。如下图所示：
窗口的 Title 为‘打开’、class 属性‘#32770’
文件名输入框的 ClassnameNN 属性为“Edit1”
打开按钮的 ClassnameNN 属性为“Button1”





第二步：打开 SciTE Script Editor 编辑器，编写脚本

;表示注释

$CmdLine[0] 获取的是命令行参数的总数，在例中$CmdLine[0]=2，参数化时使用

$CmdLine[1]~$CmdLine[63] 获取的是命令行参数第 1 到第 63 位，这个方式最多只能获取 63 个参数

WinWait(title,text，timeout)设置 timeout 秒钟用于等待窗口出现，其用法与 WebDriver 所提供的 implicitly_wait()类似。

ControlFocus("title","text",ClassnameNN) 方法用于识别 Window 窗口。3 个参数窗口 Title 属性

ControlSetText("title","text", ClassnameNN, param)用于向“文件名”输入框内输入本地文件的路径。

Sleep(2000)表示固定休眠 2000 毫秒。

ControlClick("title","text", ClassnameNN)用于点击上传窗口中的“打开”按钮。

参数说明： title 表示窗口的标题即窗口的 title 属性；

text 控件的文本属性可以使用""替代；

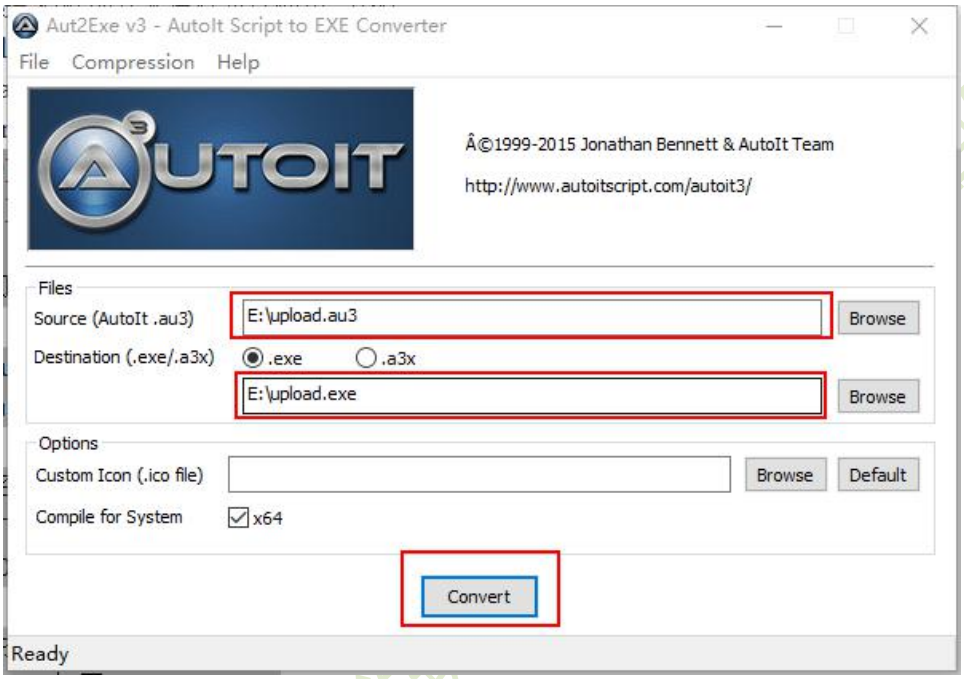
ClassnameNN 控件的 ClassnameNN 属性

param 需要传入的参数

timeout 时间秒

```
$CmdLine[0] ; = 2 参数的总数量，不虚赋值
$CmdLine[1] ; = 上传文件路径
;在 10 秒内等待打开窗口出现
WinWait("打开","",10)
;ControlFocus("title","text",ClassnameNN) 识别 Window 窗口
ControlFocus("打开", "","")
;向“文件名”输入框内输入本地文件的路径
ControlSetText("打开", "", "Edit1", $CmdLine[1])
Sleep(2000)
;单击打开按钮
ControlClick("打开", "","Button1");
```

第三步：通过 Aut2Exe（Compile Script to.exe）工具将脚本转成 exe 文件



生产 exe 文件后先通过命令行试试：

打开网页上传弹出窗口 --> cmd 中执行该脚本如下

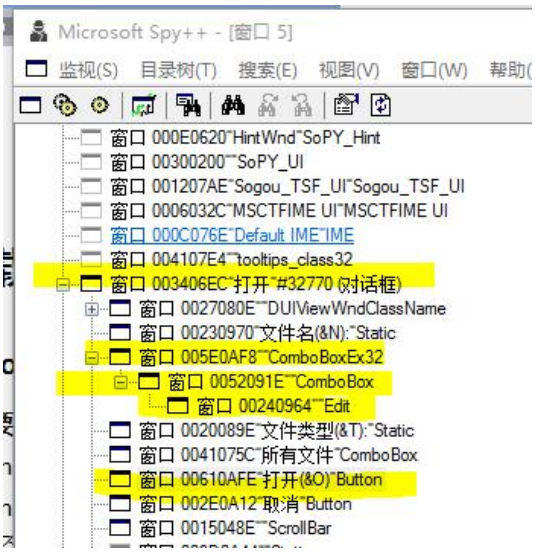
CMD>>>> upfile.exe "D:\1.html"

可以使用 python 的 os 模块来调用改文件了，代码如下：

```
from selenium import webdriver
import os
import time
driver = webdriver.Chrome()
driver.get(r'E:\code\Python\pythonDoc\自动化练习\Html\upload.html')
driver.find_element_by_name('file').click()
time.sleep(1)
# 这里可以对传参进行参数化
os.system(r'E:\upload.exe "C:\install.log"')
time.sleep(3)
driver.quit()
```

2.13.3 非 input 使用 pywin32 上传文件

使用 spy++ 查出对应窗口属性，输入的对话框，打开按钮



win32gui.FindWindow(lpClassName=None, lpWindowName=None):

自顶层窗口开始寻找匹配条件的窗口，并返回这个窗口的句柄。

```
dialog = win32gui.FindWindow('#32770', u'打开')  # 对话框
```

lpClassName：类名，在 Spy++里能够看到

lpWindowName：窗口名，标题栏上能看到的名字

代码示例里我们用来寻找上传窗口，你可以只用其中的一个，用 classname 定位容易被其他东西干扰，用 windowname 定位不稳定，不同的上传对话框可能 window_name 不同，怎么定位取决于你的情况。

win32gui.FindWindowEx(hwndParent=0, hwndChildAfter=0, lpszClass=None, lpszWindow=None)

搜索类名和窗体名匹配的窗体，并返回这个窗体的句柄。找不到就返回 0。

```
ComboBoxEx32 = win32gui.FindWindowEx(dialog, 0, 'ComboBoxEx32', None)
ComboBox = win32gui.FindWindowEx(ComboBoxEx32, 0, 'ComboBox', None)
```

hwndParent：若不为 0，则搜索句柄为 hwndParent 窗体的子窗体。

hwndChildAfter：若不为 0，则按照 z-index 的顺序从 hwndChildAfter 向后开始搜索子窗体，否则从第一个子窗体开始搜索。

lpClassName：字符型，是窗体的类名，这个可以在 Spy++里找到。

lpWindowName：字符型，是窗口名，也就是标题栏上你能看见的那个标题。

代码示例里我们用来层层寻找输入框和寻找确定按钮

win32gui.SendMessage(hWnd, Msg, wParam, lParam)

```
win32gui.SendMessage(Edit, win32con.WM_SETTEXT, None, r'C:\Users\Arrow.doc')  # 往输入框输入绝对地址
```

hWnd：整型，接收消息的窗体句柄

Msg：整型，要发送的消息，这些消息都是 windows 预先定义好的，可以参见系统定义消息（System-Defined Messages）

wParam：整型，消息的 wParam 参数

lParam：整型，消息的 lParam 参数

```
from selenium import webdriver
import win32gui as win32gui
import win32con as win32con
import time
```

```
dr = webdriver.Chrome()
dr.get('http://sahitest.com/demo/php/fileUpload.htm')
upload = dr.find_element_by_id('file')
upload.click()
time.sleep(1)
dialog = win32gui.FindWindow('#32770', u'打开')  # 对话框
ComboBoxEx32 = win32gui.FindWindowEx(dialog, 0, 'ComboBoxEx32', None)
ComboBox = win32gui.FindWindowEx(ComboBoxEx32, 0, 'ComboBox', None)
# 上面三句依次寻找对象，直到找到输入框 Edit 对象的句柄
Edit = win32gui.FindWindowEx(ComboBox, 0, 'Edit', None)
button = win32gui.FindWindowEx(dialog, 0, 'Button', None)  # 确定按钮 Button
win32gui.SendMessage(Edit, win32con.WM_SETTEXT, None, r'C:\Users\Arrow.doc')  # 往输入框输入绝对地址
win32gui.SendMessage(dialog, win32con.WM_COMMAND, 1, button)  # 按 button
print(upload.get_attribute('value'))
dr.quit()
```

2.14 下载文件

2.14.1 Firefox 文件下载

Firefox，需要我们设置其 Profile

browser.download.dir：指定下载路径

browser.download.folderList：设置成 2 表示使用自定义下载路径；设置成 0 表示下载到桌面；设置成 1 表示下载到默认路径

browser.download.manager.showWhenStarting：在开始下载时是否显示下载管理器

browser.helperApps.neverAsk.saveToDisk：对所给出文件类型不再弹出框进行询问

```
profile = webdriver.FirefoxProfile()
profile.set_preference('browser.download.dir', 'd:\\')
profile.set_preference('browser.download.folderList', 2)
profile.set_preference('browser.download.manager.showWhenStarting', False)
profile.set_preference('browser.helperApps.neverAsk.saveToDisk', 'application/zip')
driver = webdriver.Firefox(firefox_profile=profile)
driver.get('http://sahitest.com/demo/saveAs.htm')
driver.find_element_by_xpath('//a[text()="testsaveas.zip"]').click()
sleep(3)
driver.quit()
```

2.14.2 Chrome 文件下载

download.default_directory：设置下载路径

profile.default_content_settings.popups：设置为 0 禁止弹出窗口

```
options = webdriver.ChromeOptions()
prefs = {'profile.default_content_settings.popups': 0, 'download.default_directory': 'd:\\'}
options.add_experimental_option('prefs', prefs)
```

```
driver = webdriver.Chrome(executable_path=r'C:\Windows\System32\chromedriver.exe', chrome_options=options)
driver.get('http://sahitest.com/demo/saveAs.htm')
driver.find_element_by_xpath('//a[text()="testsaveas.zip"]').click()
sleep(3)
driver.quit()
```

2.15 调用 JS

driver.execute_script(js) #调用 js，无返回值

2.15.1 使用 js 拖动滚动条

http://www.cnblogs.com/yoyoketang/p/6128655.html

获取页面高度

document.body.scrollHeight

document.documentElement.scrollHeight 获取页面高度

参考 http://www.cnblogs.com/EricaMIN1987_IT/p/3593431.html

所有浏览器通用拖动滚动条

$(window).scrollTop(300);

$(document).scrollTop(300)

$("html,body").scrollTop(300);

$(window).scrollTop(document.body.scrollHeight) 拖动到底部

document.documentElement.scrollTop=10000 滚动条距离顶部 10000

window.scrollTo(100,400); # scrollTo(x, y）拖动

实例 1 将浏览器滚动条拖到底部

```
driver.get('http://www.cnblogs.com/EricaMIN1987_IT/p/3593431.html')
a=driver.execute_script('$(window).scrollTop(document.body.scrollHeight)') # 调用 js
print(a)
```

实例 2 讲 div 的滚动条拖动到底部

gundong. html

gundongtiao.html

```
<!DOCTYPE html>
<html><head>
    <style type="text/css">
        div.scroll {
            background-color: #00FFFF;
            width: 100px;
            height: 100px;
            overflow: auto; }
    </style></head><body>
<p>overflow:scroll</p>
<div class="scroll">You can use the overflow property when you want to have better control of the layout. The default
    value is visible.aaaaaaaaaaaaaaaaaaaaaaaaaa
```

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

```
driver.get(r'gundong.html')
js='document.getElementsByClassName("scroll")[0].scrollTop=10000'
driver.execute_script(js)
```

2.15.2 通过 js 操作元素

$(".letter").click() 单击
$("#tooltip").fadeOut() 隐藏
driver.execute_script(“arguments[0].click();”,Element) # 通过获取的元素点击

2.16 cookie 处理

driver.get_cookies（） 获得 cookie 信息
add_cookie(cookie_dict) 向 cookie 添加会话信息
delete_cookie(name) 删除特定(部分)的 cookie
delete_all_cookies() 删除所有 cookie

实例：

```
cookie = driver.get_cookies()  # 获得 cookie 信息
print(cookie)
driver.add_cookie({'name': 'key-name', 'value': 'value-123456'})  # 向 cookie 的 name 和 value 添加会话信息。
for cookie in driver.get_cookies():
    print("%s -> %s" % (cookie['name'], cookie['value']))
driver.delete_cookie("CookieName")  # 删除一个特定的 cookie
driver.delete_all_cookies()  # 删除所有 cookie
```

2.17 Chrome 模拟手机浏览器测试手机网页

通过 device name 来模拟的手机样式

```
mobileEmulation = {'deviceName': 'Apple iPhone 4'}
options = webdriver.ChromeOptions()
options.add_experimental_option('mobileEmulation', mobileEmulation)
driver = webdriver.Chrome(executable_path='chromedriver.exe', chrome_options=options)
driver.get('http://m.baidu.com')
```

直接指定分辨率以及 UA 标识

```
WIDTH = 320
HEIGHT = 640
```

```
PIXEL_RATIO = 3.0
UA = 'Mozilla/5.0 (Linux; Android 4.1.1; GT-N7100 Build/JRO03C) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/35.0.1916.138 Mobile Safari/537.36 T7/6.3'
mobileEmulation = {"deviceMetrics": {"width": WIDTH, "height": HEIGHT, "pixelRatio": PIXEL_RATIO}, "userAgent": UA}
options = webdriver.ChromeOptions()
options.add_experimental_option('mobileEmulation', mobileEmulation)
driver = webdriver.Chrome(executable_path='chromedriver.exe', chrome_options=options)
driver.get('http://m.baidu.com')
```

2.18 经验 switch_to 方法

alert　返回浏览器的 Alert 对象，可对浏览器 alert、confirm、prompt 框操作
default_content()　切到主文档
frame(frame_reference)　　切到某个 frame
parent_frame()　　切到父 frame，多层 frame 的时候很有用（多层的 frame 都是有问题的，会影响到性能）
window(window_name)　　切到某个浏览器窗口
active_element　返回当前焦点的 WebElement 对象

2.18.1 active_element 返回当前焦点，即返回 WebElement 对象

对于有些需要点击进去，然后变为输入框的控件有时候会失去焦点，无法输入值或者找不到对象
可以采用：

```
driver.find_element_by_class_name('kw').click() #获取对象
time.sleep(1)
driver.switch_to.active_element.send_keys('name') # 通过 active_element 获取焦点对象，输入值
```

2.19 截图

dr.get_screenshot_as_file(path) # 截图
path 要求是绝对路径,如果截图失败返回 False,成功者 True

```
dr = webdriver.Chrome()
dr.get("http://www.baidu.com")
time.sleep(3)
path=os.path.abspath(r"./jietu/baidu1.png")
dr.get_screenshot_as_file(path)
dr.quit()
```

3.元素定位 提高篇 xpath 篇

3.1、xpath，css，JavaScript 元素定位简介

自动化测试的问题大部分都集中在元素定位，开发的不规范页面经常缺少 ID 等唯一标识，也可能是采用集成框架造成 ID 等属性是动态，这些都会对元素定位带不便。

元素定位通过 ID 和 name 定位是最高效也是首选的定位方式，不过由于 name 不一定唯一，在定位时匹配条件的元素可能有多个，因此这种情况下只会定位到匹配条件的第一个元素。针对多个元素具有相同 name（或链接文本）属性的情况还需额外增加其他的过滤器才能进行精确定位。

ID、name、link_text、class_name 定位比较简单前面已经讲过了，这节内容不多做介绍。 本节重点在 xpath、css、JavaScript 定位元素。

XPath 最初是用来在 XML 文档中定位 DOM 节点的语言，由于 HTML 也可以算作 XML 的一种实现，所以 Selenium 也可以利用 XPath 这一强大的语言来定位 Web 元素。xpath 的强大在于它可以通过父节点或者兄弟节点根据前后的关联性定位到元素。

CSS (Cascading Style Sheets) 是一种用于渲染 HTML 或者 XML 文档的语言，CSS 利用其选择器可以将样式属性绑定到文档中的指定元素，即前端开发人员可以利用 CSS 设定页面上每一个元素的样式。所以理论上说无论一个元素定位有多复杂，既然开发人员能够定位到并设置样式，那么测试人员同样应该也能定位继而操作该元素。加上不同的浏览器 XPath 引擎不同甚至没有自己的 Xpath 引擎，这就导致了 XPath 定位速度较慢，所以 Selenium 官方极力推荐使用 CSS 定位。理论永远与实际脱节，很多情况下都无法使用 CSS 定位到元素，而 xpath 非常容易做到。

JavaScript 是通过 JavaScript 返回值来定位元素，比 xpath 还繁琐，不推荐使用，这章节也不会做过多的介绍。

3.2、xpath 定位元素

本节内容来自：http://www.w3school.com.cn/xpath/index.asp 精简后的内容。

XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是通过沿着路径 (path) 或者步 (steps) 来选取的。

3.2.1、选取节点

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。

..	选取当前节点的父节点。
@	选取属性。

示例：

例子	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。 注释：假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

3.2.2、谓语（Predicates）

谓语用来查找某个特定的节点或者包含某个指定的值的节点。
谓语被嵌在方括号中。

例子	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。

//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
//a[text()='新闻']	节点中的文本为新闻
//*[contains(@class,'input-box')]	class 属性包含 input-box 字段
//*[starts-with(@class,'input-box')]	class 属性包含某字段开始
//*[ends-with(@class,'input-box')]	class 属性包含某字段结束
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

3.2.3、选取未知节点

XPath 通配符可用来选取未知的 XML 元素。

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。

示例：

例子	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

3.2.4、XPath 轴

轴可定义相对于当前节点的节点集。

轴名称	结果
ancestor	选取当前节点的所有先辈（父、祖父等）。
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身。
attribute	选取当前节点的所有属性。
child	选取当前节点的所有子元素。
descendant	选取当前节点的所有后代元素（子、孙等）。
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身。
following	选取文档中当前节点的结束标签之后的所有节点。
namespace	选取当前节点的所有命名空间节点。
parent	选取当前节点的父节点。
preceding	选取文档中当前节点的开始标签之前的所有节点。
preceding-sibling	选取当前节点之前的所有同级节点。
following-sibling	选取当前节点之后的所有同级节点。
self	选取当前节点。

示例：

例子	结果
child::book	选取所有属于当前节点的子元素的 book 节点。
attribute::lang	选取当前节点的 lang 属性。
child::*	选取当前节点的所有子元素。
attribute::*	选取当前节点的所有属性。
child::text()	选取当前节点的所有文本子节点。

child::node()	选取当前节点的所有子节点。
descendant::book	选取当前节点的所有 book 后代。
ancestor::book	选择当前节点的所有 book 先辈。
ancestor-or-self::book	选取当前节点的所有 book 先辈以及当前节点（如果此节点是 book 节点）
child::*/child::price	选取当前节点的所有 price 孙节点。

3.2.5、综合运用

示例：搜狗搜索页面为示例，比较复杂点的例子

例子	结果
//input	所有 input 元素
//form/input	form 子元素 input
//form/input[2]	form 子元素中为 input 的第二个 input
//form/input[last()]	form 子元素的最后一个 input。
//form//input[@name="query"]	form 后代元素中 name="query"的元素
//form/../div	form 父元素下的子元素 div(div 与 form 同级)
//form/../../div[1]//a	form 父元素的父元素的子元素第一个 div 下的后代 a
//input[@name="pageNum"][@type="hidden"]	type="hidden"和 name="pageNum"的 input 元素
//a[text()='新闻']	节点中的文本为新闻
//*[contains(text(),'搜索 APP')]	包含'搜索 APP'文本的元素
//*[contains(@class,'input-box')]	class 属性包含 input-box 字段
//*[starts-with(@class,'input-box')]	class 属性包含某字段开始

//*[ends-with(@class,'input-box')]	class 属性包含某字段结束
//input[@name="w"]/following-sibling::input	name="w"的 input 元素同级的弟弟元素
//input[@name="w"]/preceding-sibling::input	name="w"的 input 元素同级的哥哥元素
//form[@id="sf"]//input[@type="text"]	id="sf"的 form 后代中 type="text"的 input 节点元素
//form[@id="sf"]/span[2]/input	id="sf"的 form 子元素 span 中的第 2 个下子元素中所有 input

3.3、CSS 定位元素

3.3.1、CSS 元素选择器

input 选择 input 元素
p 选择 p 元素

3.3.2、CSS ID 与类选择器

ID 选择器以 "#" 来定义。
class 类选择器以一个‘.’点号显示
示例：搜狗搜索页面元素示例
#query 表示 id 为 query 的元素
.query 表示 class 为 query 的元素

3.3.3、CSS 属性选择器

属性选择器可以根据元素的属性及属性值来选择元素。
示例：
[title] 将 title 属性放到中括号中，表示选择具有该属性的元素
[title=value] title 属性等于 value 的元素
[title~=value] title 属性包含 value 单词的元素，注意是单词
[title|=value] title 属性以 value 单词开头的元素
[title^=value] title 属性以 value 开头的元素
[title$=value] title 属性以 value 结尾的元素
[title*=value] title 属性包含 value 的元素
input[title*=value] input 元素下有 title 属性包含 value 的元素
input[type="ext][name=query]#query input 元素下有 type="text"、name='query'、id="query"的元素

3.3.4、CSS 后代选择器

后代选择器（descendant selector）又称为包含选择器。后代选择器可以选择作为某元素后代的元素。
注意：后代不一定是儿子，也有可能是孙子。
示例：搜狗搜索页面元素示例
span input 选择 span 下的 input 元素
div.content span.enter-input input#stb 选择 div.content 下 span.enter-input 的下的 input ID=stb 的元素。（搜狗

搜索示例）

3.3.5、CSS 子元素选择器

与后代选择器相比，子元素选择器（Child selectors）只能选择作为某元素子元素的元素。
如果您不希望选择任意的后代元素，而是希望缩小范围，只选择某个元素的子元素，就使用子元素选择器（Child selector）。
与后代的区别在于，子元素选择器只能选择儿子。
示例：搜狗搜索页面元素示例
div>input　　只作为 div 元素子元素的 input 元素
div>form>span>input　　div 下的子元素 form 下的子元素 span 下的子元素 input
div>form[name='sf']>span>input#query　　div 下的子元素 form[name='sf']下的子元素 span 下的子元素 input#query
div#content form>span>input#query　　div#content 下的子元素 form 下的子元素 span 下的子元素 input#query

3.3.6、CSS 相邻兄弟选择器

相邻兄弟选择器（Adjacent sibling selector）可选择紧接在另一元素后的元素，且二者有相同父元素。
示例：搜狗搜索页面元素示例
input+input　　input 后出现的 input 的元素
div#content form>span+input　　div#content 下的 form 元素下的子元素 span 同级相邻的第一个弟弟元素

3.3.7、CSS 伪类

CSS 伪类用于向某些选择器添加特殊的效果。selenium 中元素 CSS 定位用的伪类只有:first-child（元素的第一个子元素），其他的可靠性非常低要不就是不能使用。
示例：搜狗搜索页面元素示例
input:first-child　所有第一个 input 元素
form > input:first-child　所有 form 子元素中的第一个 input 元素
form input:first-child　所有 form 元素中以 input 作为第一个元素的元素
form span:first-child input:first-child　form 下的第一个 span 元素下的第一个 input 元素

3.3.8、CSS 选择器运用示例

例子是搜狗搜索中的元素为例。

例子	结果
input	所有 input 元素
#query	id=query 的元素
.sec-input	class=sec-input 元素

form>input	form 子元素 input
form>input:nth-child(2)	form 子元素中为 input 的第二个 input
form>input:last-child	form 子元素的最后一个 input。
form input[name="query"]	form 后代元素中 name="query"的元素
input[name=pageNum][type=hidden]	type="hidden"和 name="pageNum"的 input 元素
[class*=input-box]	class 属性包含 input-box 字段
[class^=input]	class 属性包含 input 字段开始
[class$=box]	class 属性包含 box 字段结束
input[name=w]+input	name="w"的 input 元素同级的弟弟元素
form#sf input[type=text]	id="sf"的 form 后代中 type="text"的 input 节点元素
form#sf>span:nth-child(2)>input	id="sf"的 form 子元素 span 中的第 2 个下子元素中所有 input

3.3.9、CSS 选择器参考手册

参考 w3school 的选择器参考手册为样本，进行了删减，将不适合自动化测试的选择器删除。
原文参考手册见： http://www.w3school.com.cn/cssref/css_selectors.asp

例子是搜狗搜索中的元素为例：

选择器	例子	例子描述
.class	.sec-input	选择 class="sec-input" 的所有元素。
#id	#query	选择 id="query" 的所有元素。
*	form *	form 下的所有元素。
element	p	选择所有 <p> 元素。
element,element	div,p	选择所有 元素和所有 <p> 元素。

element element	div p	选择 元素内部的所有 <p> 元素。
element>element	div>p	选择父元素为 元素的所有 <p> 元素。
element+element	div+p	选择紧接在 元素之后的所有 <p> 元素。
[attribute]	[target]	选择带有 target 属性所有元素。
[attribute=value]	[target=_blank]	选择 target="_blank" 的所有元素。
[attribute~=value]	[class~=gjss-sz]	选择 class 属性包含单词 "gjss-sz" 的所有元素。 value 要求是个完整的单词(空格分割)
[attribute|=value]	[class|=gjss]	选择 class 属性值以 "gjss" 开头的所有元素。 value 只能为纯字母(非字母分割)
:first-child	p:first-child	选择属于父元素的第一个子元素的每个 <p> 元素。
:lang(language)	html:lang(cn)	选择带有以 "cn" 开头的 lang 属性值的每个 <html> 元素。
element1~element2	p~ul	选择前面有 <p> 元素的每个 元素。
[attribute^=value]	p[class^="gjs"]	选择其 class 属性值以 "gjs" 开头的每个 <p> 元素。
[attribute$=value]	p[class$="-mask"]	选择其 class 属性以 "-mask" 结尾的所有 <p> 元素。
[attribute*=value]	p[class*="err"]	选择其 class 属性中包含 "err" 子串的每个 <p> 元素。
:first-of-type	p:first-of-type	选择属于其父元素的首个 <p> 元素的每个 <p> 元素。
:last-of-type	p:last-of-type	选择属于其父元素的最后 <p> 元素的每个 <p> 元素。
:only-of-type	p:only-of-type	选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。
:only-child	div:only-child	选择属于其父元素的唯一子元素的每个 元素。
:nth-child(n)	p:nth-child(2)	选择属于其父元素的第二个子元素的每个 <p> 元素。
:nth-last-child(n)	p:nth-last-child(2)	同上，从最后一个子元素开始计数。

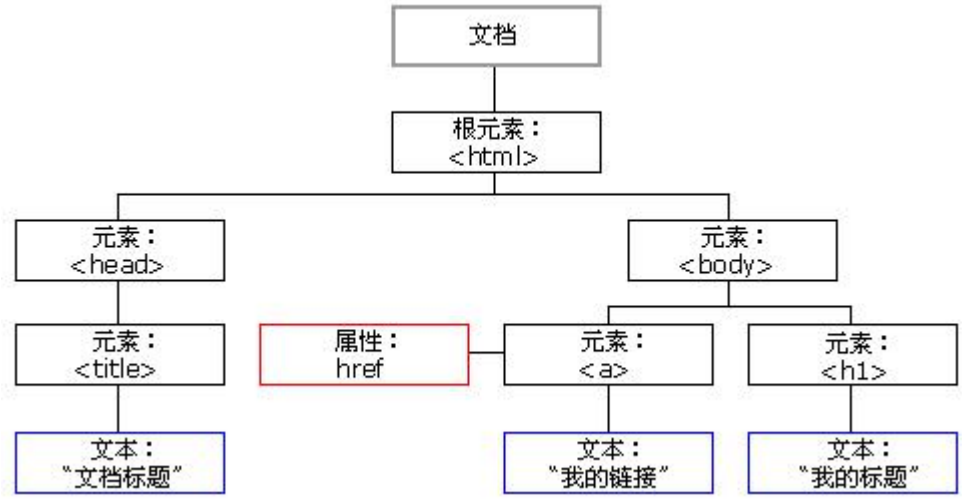
:nth-of-type(n)	p:nth-of-type(2)	选择属于其父元素第二个 <p> 元素的每个 <p> 元素。
:nth-last-of-type(n)	p:nth-last-of-type(2)	同上，但是从最后一个子元素开始计数。
:last-child	p:last-child	选择属于其父元素最后一个子元素每个 <p> 元素。
:root	:root	选择文档的根元素。
:empty	div:empty	选择没有子元素的每个 元素（包括文本节点）。
:enabled	input:enabled	选择每个启用的 <input> 元素。
:disabled	input:disabled	选择每个禁用的 <input> 元素

四、JavaScript 获取页面元素

JavaScript(js)获取页面元素是通过 js 的返回值获取元素，有两种方式：DOM、JQuery

1、DOM 获取元素

HTML DOM 定义了访问和操作 HTML 文档的标准。每个页面都是一个 DOM Tree 结果，如下图



通过 HTML DOM，树中的所有节点均可通过 JavaScript 进行访问。我们通过 JavaScript 返回该节点就是我们的元素。

document.getElementById()	获取带有指定 ID 的元素
document.getElementsByClassName()	获取包含带有指定类名的所有元素的节点列表
document.getElementsByName()	获取指定 Name 的所有元素的节点列表
document.getElementsByTagName()	获取带有指定标签名称的所有元素的节点列表

注意：getElementsByClassName() 在 Internet Explorer 5,6,7,8 中无效

示例：通过 DOM 的 JavaScript 脚本获取元素

```
js='return document.getElementById("query")'
```

```
js='return document.getElementsByClassName("sec-input")[0]'
js='return document.getElementsByName("query")[0]'
js='return document.getElementsByTagName("input")[0]'
js='return document.getElementById("sf").getElementsByTagName("input") ' #    id=sf 的 input 子元素
element=driver.execute_script(js)   #  执行 js 获取元素
```

注意：由于 getElements 返回的为列表所以需要使用[0]指定为第 1 个元素，如果查找出对应元素存在多个可以指定第几个。

2、JQuery 获取元素

JQuery 获取元素也是通过返回值获取，比 DOM 的兼容性高。

常用语法格式如下：

return $("p:contains('包含的字符')")[0]　　根据页面文本获取元素

return $(css 定位)[0]　　css 定位获取对象

return $x(xpath 定位)[0]　　xpath 定位获取对象

示例：

```
js = 'return $("input[name=w]+input")[0]'
js = 'return $("form#sf input[type=text]")[0]'
js = 'return $x("//input[@name=\'w\']/following-sibling::input")[0]'
js='$("p:contains(\'搜狗搜索 APP\')")[0]'
element=driver.execute_script(js)   #  执行 js 获取元素
```

附件：JQuery 选择器

来自于：http://www.w3school.com.cn/jquery/jquery_ref_selectors.asp

选择器	实例	选取
*	$("*")	所有元素
#id	$("#lastname")	id="lastname" 的元素
.class	$(".intro")	所有 class="intro" 的元素
element	$("p")	所有 <p> 元素
.class.class	$(".intro.demo")	所有 class="intro" 且 class="demo" 的元素
:first	$("p:first")	第一个 <p> 元素
:last	$("p:last")	最后一个 <p> 元素
:even	$("tr:even")	所有偶数 <tr> 元素
:odd	$("tr:odd")	所有奇数 <tr> 元素
:eq(index)	$("ul li:eq(3)")	列表中的第四个元素（index 从 0 开始）
:gt(no)	$("ul li:gt(3)")	列出 index 大于 3 的元素
:lt(no)	$("ul li:lt(3)")	列出 index 小于 3 的元素
:not(selector)	$("input:not(:empty)")	所有不为空的 input 元素
:header	$(":header")	所有标题元素 <h1> - <h6>
:animated		所有动画元素
:contains(text)	$(":contains('W3School')")	包含指定字符串的所有元素
:empty	$(":empty")	无子（元素）节点的所有元素
:hidden	$("p:hidden")	所有隐藏的 <p> 元素
:visible	$("table:visible")	所有可见的表格
s1,s2,s3	$("th,td,.intro")	所有带有匹配选择的元素
[attribute]	$("[href]")	所有带有 href 属性的元素
[attribute=value]	$("[href='#']")	所有 href 属性的值等于 "#" 的元素
[attribute!=value]	$("[href!='#']")	所有 href 属性的值不等于 "#" 的元素

选择器	实例	选取
*	$("*")	所有元素
#id	$("#lastname")	id="lastname" 的元素

.class	$(".intro")	所有 class="intro" 的元素
element	$("p")	所有 <p> 元素
.class.class	$(".intro.demo")	所有 class="intro" 且 class="demo" 的元素
:first	$("p:first")	第一个 <p> 元素
:last	$("p:last")	最后一个 <p> 元素
:even	$("tr:even")	所有偶数 <tr> 元素
:odd	$("tr:odd")	所有奇数 <tr> 元素
:eq(index)	$("ul li:eq(3)")	列表中的第四个元素（index 从 0 开始）
:gt(no)	$("ul li:gt(3)")	列出 index 大于 3 的元素
:lt(no)	$("ul li:lt(3)")	列出 index 小于 3 的元素
:not(selector)	$("input:not(:empty)")	所有不为空的 input 元素
:header	$(":header")	所有标题元素 <h1> - <h6>
:animated		所有动画元素
:contains(text)	$(":contains('W3School')")	包含指定字符串的所有元素
:empty	$(":empty")	无子（元素）节点的所有元素
:hidden	$("p:hidden")	所有隐藏的 <p> 元素
:visible	$("table:visible")	所有可见的表格
s1,s2,s3	$("th,td,.intro")	所有带有匹配选择的元素
[attribute]	$("[href]")	所有带有 href 属性的元素
[attribute=value]	$("[href='#']")	所有 href 属性的值等于 "#" 的元素
[attribute!=value]	$("[href!='#']")	所有 href 属性的值不等于 "#" 的元素
[attribute$=value]	$("[href$='.jpg']")	所有 href 属性的值包含以 ".jpg" 结尾的元素
:input	$(":input")	所有 <input> 元素
:text	$(":text")	所有 type="text" 的 <input> 元素

:password	$(":password")	所有 type="password" 的 <input> 元素
:radio	$(":radio")	所有 type="radio" 的 <input> 元素
:checkbox	$(":checkbox")	所有 type="checkbox" 的 <input> 元素
:submit	$(":submit")	所有 type="submit" 的 <input> 元素
:reset	$(":reset")	所有 type="reset" 的 <input> 元素
:button	$(":button")	所有 type="button" 的 <input> 元素
:image	$(":image")	所有 type="image" 的 <input> 元素
:file	$(":file")	所有 type="file" 的 <input> 元素
:enabled	$(":enabled")	所有激活的 input 元素
:disabled	$(":disabled")	所有禁用的 input 元素
:selected	$(":selected")	所有被选取的 input 元素
:checked	$(":checked")	所有被选中的 input 元素

附件：css 与 xpath 对照表

下面以搜狗搜索页面元素为示例，

描述	Xpath	CSS Path
直接子元素	//form/input	form>input
子元素或后代元素	//div//a	div a
以 id 定位	//div[@id="wrap"]//a	div#wrap a
以 class 定位	//div[@class="wrapper"]//a	div.wrapper a
同级弟弟元素	//input[@name="w"]/following-sibling::input	input[name=w]+input
属性	//form//input[@name="query"]	form input[name="query"]
多个属性	//input[@name="pageNum"][@type="hidden"]	input[name=pageNum][type=hidden]
第 2 个子元素	//form/input[2]	form>input:nth-child(2)
第 1 个子元素	//form/input[1]	form>input:first-child
最后 1 个子元素	//form/input[last()]	form>input:last-child
属性包含某字段	//*[contains(@class,'input-box')]	[class*=input-box]
属性以某字段开头	//*[starts-with(@class,'input-box')]	[class^=input]
属性以某字段结尾	//*[ends-with(@class,'input-box')]	[class$=box]
text 中包含某字段	//*[contains(text(),'搜索 APP')]	无法定位
元素有某属性	//div[@style]	div[style]
父节点	//div[@id='cur-weather']/..	无法定位
同级哥哥节点	//input[@name="w"]/preceding-sibling::input	无法定位

描述	Xpath	CSS Path
直接子元素	//form/input	form>input
子元素或后代元素	//div//a	div a
以 id 定位	//div[@id="wrap"]//a	div#wrap a
以 class 定位	//div[@class="wrapper"]//a	div.wrapper a
同级弟弟元素	//input[@name="w"]/following-sibling::input	input[name=w]+input

属性	//form//input[@name="query"]	form input[name="query"]
多个属性	//input[@name="pageNum"][@type="hidden"]	input[name=pageNum][type=hidden]
第 2 个子元素	//form/input[2]	form>input:nth-child(2)
第 1 个子元素	//form/input[1]	form>input:first-child
最后 1 个子元素	//form/input[last()]	form>input:last-child
属性包含某字段	//*[contains(@class,'input-box')]	[class*=input-box]
属性以某字段开头	//*[starts-with(@class,'input-box')]	[class^=input]
属性以某字段结尾	//*[ends-with(@class,'input-box')]	[class$=box]
text 中包含某字段	//*[contains(text(),'搜索 APP')]	无法定位
元素有某属性	//div[@style]	div[style]
父节点	//div[@id='cur-weather']/..	无法定位
同级哥哥节点	//input[@name="w"]/preceding-sibling::input	无法定位

4、JavaScript 操作元素对象

JavaScript 操作元素对象可以使用 DOM 操作元素，如果应用有 JQuery 可以使用 JQuery 操作元素对象。

4.1、打开新窗口

DOM 操作：

```
window.open('https://www.qqcom')
```

4.2、滚动浏览器滚动条

4.2.1、DOM 操作：

```
document.body.scrollWidth;          网页正文全文宽，包括有滚动条时的未见区域
document.body.scrollHeight;         网页正文全文高，包括有滚动条时的未见区域
document.documentElement.clientWidth;          可见区域宽度，不有滚动条时的未见区域
document.documentElement.clientHeight;         可见区域高度，不有滚动条时的未见区域
document.documentElement.scrollTop=100;        设置或返回匹配元素相对滚动条顶部的偏移
document.documentElement.scrollLeft=100;       设置或返回匹配元素相对滚动条左侧的偏移
window.scrollTo(100,400);           设计滚动条 left=100，top=400
```

4.2.2、JQuery 操作：

JQueryelement.height()　设置或返回匹配元素的高度
JQueryelement.width()　设置或返回匹配元素的宽度
JQueryelement.scrollTop()　设置或返回匹配元素相对滚动条顶部的偏移。
JQueryelement.scrollLeft()　设置或返回匹配元素相对滚动条左侧的偏移。

```
$(window).scrollTop(300);　　#拖动页面距离顶部 300 像素
$(document).scrollTop(300)
$("html,body").scrollTop(300);
$(window).scrollTop(document.body.scrollHeight);　#拖动到底部
$(document).scrollTop($(window).height());　　#拖动到底部
```

4.3、拖动窗口内元素滚动条

4.3.1、DOM 操作：

```
document.getElementsByClassName("scroll")[0].scrollTop=10000
document.getElementsByClassName("scroll")[0].scrollLeft=10000
```

4.3.2、JQuery 操作：

```
$("div.id").scrollTop(300);
$("div.id").scrollLeft(300);
```

4.4、新增与修改元素属性

4.4.1、DOM 操作：

改变属性

```
document.getElementById('query').value='selenium'　　# 设置元素值，如果是输入框则输入内容
document.getElementsByClassName("sec-input")[0].disabled=false　# 取消置灰
```

使元素隐藏或可见

```
document.getElementById("query").style.display="none"　# 隐藏
document.getElementById("query").style.display="block"　# 可见
```

4.4.2、JQuery 操作：

JQueryelement.attr()　　设置或返回匹配元素的属性和值。

```
$("#query").attr('value','selenium')　# 设置元素值，如果是输入框则输入内容
$("#query").attr('disabled',false)　　# 取消置灰
$("#query").attr('style','display: none;')　# 隐藏
$("#query").attr('style','display: block;')　# 可见
```

4.5、删除元素属性

4.5.1、DOM 操作：

移除元素属性

```
document.getElementById("query").removeAttribute('readonly')   # 移除'readonly'属性，是元素可输入
document.getElementById('query').value     # 获取 value 值
```

4.5.2、JQuery 操作：

移除元素属性

JQueryelement.removeAttr() 从所有匹配的元素中移除指定的属性。

```
$("#query").removeAttr('readonly')     # 移除'readonly'属性，是元素可输入
```

4.6、获取元素的内容

4.6.1、DOM 操作：

```
document.getElementById("query").innerHTML                    # 获取 HTML 内容
document.getElementsByClassName('top-nav')[0].innerText       # 获取文本内容
document.getElementsByClassName("sec-input")[0].attributes.属性      # 获取元素属性
```

4.6.2、JQuery 操作：

JQueryelement.html()　　设置或返回匹配的元素集合中的 HTML 内容。

JQueryelement.val()　　设置或返回匹配元素的 value 属性值

JQueryelement.text()　　设置或返回匹配元素的内容。

```
    $('.top-nav').html()   # 获取 HTML
    $('.top-nav').text()   # 获取 text
    $('#query').val()   获取 val 值
```

4.7、操作页面元素

4.7.1、DOM 操作：

```
document.getElementById('stb').click()     # 单击元素
```

4.7.2、JQuery 操作：

```
$('#stb').click()
```

4.unittest 单元测试框架

4.1 unittest 介绍

unittest 又名 PyUnit， Python 单元测试框架（The Python unit testing framework），简称为 PyUnit， 是 Kent Beck 和 Erich Gamma 这两位聪明的家伙所设计的 JUnit 的 Python 版本。 而 JUnit 又是 Kent 设计的 Smalltalk 测试框架的 Java 版本。它们都是各自语言的标准测试框架。
自从 Python 2.1 版本后，PyUnit 成为 Python 标准库的一部分。
官方文档 https://docs.python.org/2/library/unittest.html
中国开源社区 https://www.oschina.net/question/12_27127

4.2 简单的实例

创建计算器函数

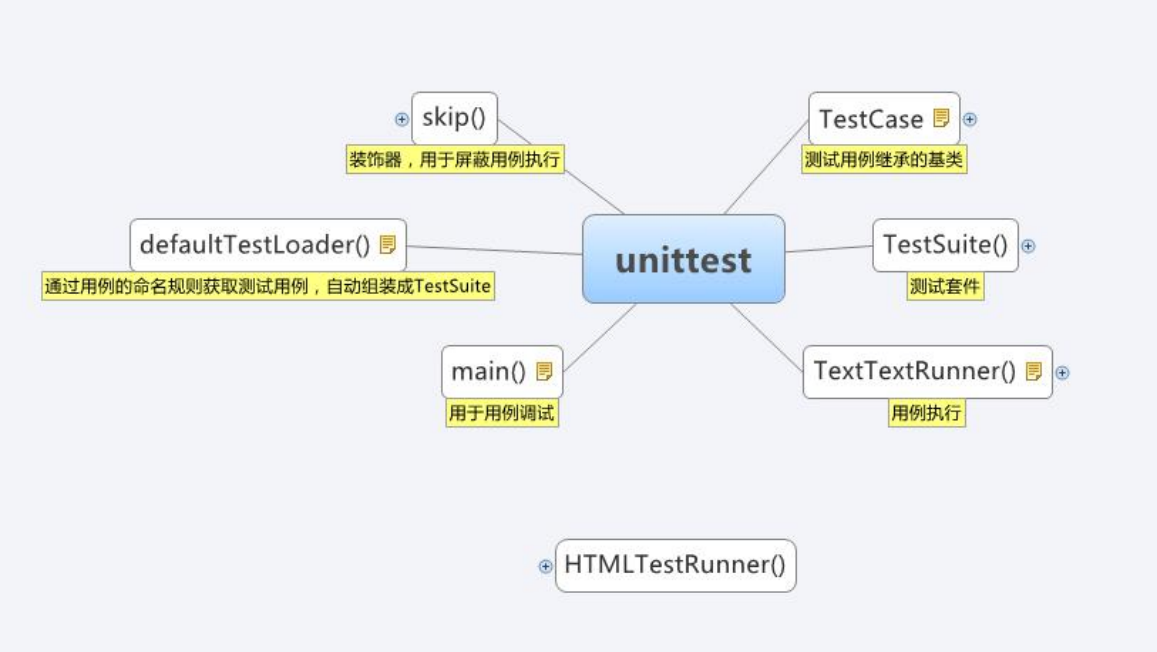
```
# Calculator.py
class calculator(object):
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def add(self):
        return (self.a + self.b)
    def minus(self):
        return (self.a - self.b)
    def multip(self):
        return (self.a * self.b)
    def divide(self):
        return (self.a / self.b)
```

创建一个简单的单元 add 测试用例

```
# coding = UTF-8
import unittest
from unittest_doc.com.Calculator.Calculator import calculator

class MyTest(unittest.TestCase):
    def add_test(self):
        a = calculator(1, 2)
        print("测试用例  add_test")
        self.assertEqual(a.add(), 3)
if __name__ == '__main__':
    unittest.main()
```

注意：如果在测试用例运行时断言（assertion）为假，AssertionError 异常会被抛出，测试框架会认为测试用例失败。其它非“assert”检查所抛出的异常会被测试框架认为是“errors”。

4.3 测试用例组成

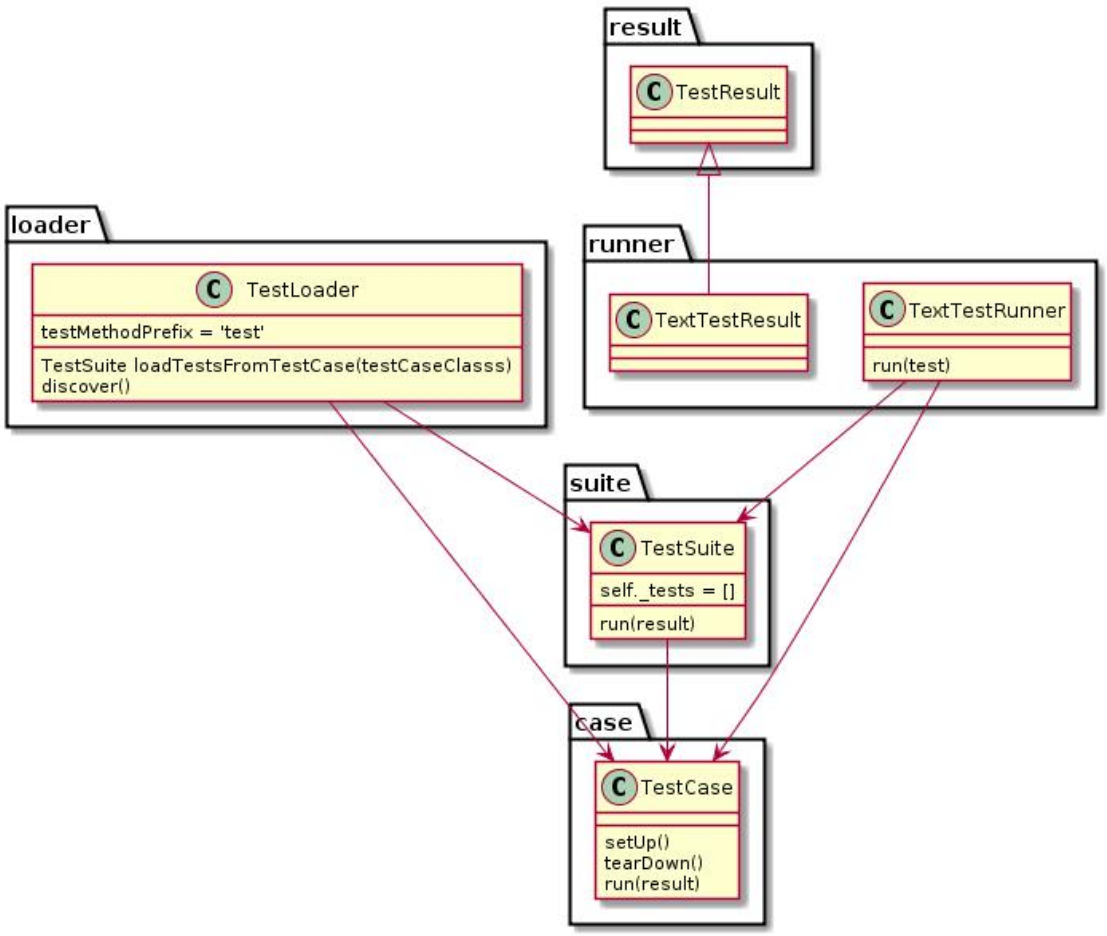


4.3.1 案例

```
#test_simple.py
class simple_test(unittest.TestCase):
    def setUp(self):
        print('@@@初始化 test_simple@@@')
        self.a = calculator(1, 2)

    def test_add(self):
        print('---测试用例 test_simple add---')
        self.assertEqual(self.a.minus(), -1, '两值不相等')
        self.assertEqual(self.a.add(), 3, '两值不相等')
        self.assertNotEqual(self.a.divide(), 1, '两值不相等')

    def test_two(self):
        print('---测试用例 test_simple two---')
        self.assertTrue(True, '为真')

    def test_divide(self):
        print('---测试用例 test_simple divide---')
        self.assertEqual(self.a.divide(), 0.5)

    def tearDown(self):
        print('@@@结束 test_simple@@@')

if __name__ == '__main__':
    suite = unittest.TestSuite()   # 建立测试套件
    suite.addTest(simple_test('test_add'))   # 添加测试用例
```

```
suite.addTest(simple_test('test_divide'))
# 执行测试
runner = unittest.TextTestRunner()
runner.run(suite)
```

4.3.2 unittest 结构



test case

一个 TestCase 的实例就是一个测试用例。测试用例就是一个完整的测试流程，包括测试前准备环境的搭建(setUp)，实现测试过程的代码(run)，以及测试后环境的还原(tearDown)。元测试(unittest)的本质也就在这里，一个测试用例是一个完整的测试单元，通过运行这个测试单元，可以对某一个功能进行验证。

test case 的组成部分

setUp():用于测试用例执行前的初始化工作。

如测试用例中需要访问数据库，可以在 setUp 中建立数据库连接并进行初始化。如测试用例需要登录 web，可以先实例化浏览器。如 app 测试需先要启动 app，可先实例化 app。

tearDown():用于测试用例执行之后的善后工作。如关闭数据库连接。关闭浏览器，关闭 app。

test_one(),test_two(): 测试脚本，已 test 开头或者结尾

test suite

对一个功能的验证往往是需要很多多测试用例，可以把测试用例集合在一起执行，这就产生了测试套件 TestSuite 的概念，它是用来组装单个测试用例，而且 TestSuite 也可以嵌套 TestSuite。

可以通过 addTest 加载 TestCase 到 TestSuite 中，再返回一个 TestSuite 实例。

```
#TestSuite 用来创建测试套件的
suite=unittest.TestSuite()
#addTest() 方法是将测试用例添加到测试套件中
suite.addTest(MyTest('test_two'))
```

```
#makeSuite：相识性创建测试套件
suite =unittest.makeSuite(MyTest,'test') # MyTest 测试用例类名
```

test runner

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

```
# run(): 方法是运行测试套件的测试用例，入参为 suite 测试套件
Runner=unittest.TextTestRunner()
runner.run(suite)
#main 用于调试执行测试用例 在 if __name__==’__main__’ 中
unittest.main()
```

test fixture

对一个测试用例环境的搭建和销毁，是一个 fixture，通过覆盖 TestCase 的 setUp()和 tearDown()方法来实现。比如说在这个测试用例中需要访问数据库，那么可以在 setUp()中建立数据库连接以及进行一些初始化，在 tearDown()中清除在数据库中产生的数据，然后关闭连接。注意 tearDown 的过程很重要，要为以后的 TestCase 留下一个干净的环境。

4.3.3 unittest 与 web 自动化测试结合

```
#test_baidu.py
import unittest, time
from selenium import webdriver

class baiduTest(unittest.TestCase):
    u'''百度测试'''
    def setUp(self):
        self.driver = webdriver.Chrome()  # 实例化浏览器
        self.driver.maximize_window()  # 最大化浏览器
        self.url = 'http://www.baidu.com'

    def test_baidu(self):
        driver = self.driver
        driver.get(self.url)
        driver.find_element_by_id("kw").clear()
        driver.find_element_by_id("kw").send_keys("selenium")
        driver.find_element_by_id("su").click()
        time.sleep(2)
        title = driver.title
        self.assertEqual(title, u"selenium_百度搜索")

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

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

4.3.4 批量用例执行

defaultTestLoader 用于匹配执行目录下的用例

unittest.defaultTestLoader.discover(start_dir, pattern='test*.py', top_level_dir=None)

start_dir ：要测试的模块名或测试用例目录。

pattern='test*.py' ：表示用例文件名的匹配原则。星号“*”表示任意多个字符。

top_level_dir=None：测试模块的顶层目录。如果没顶层目录（也就是说测试用例不是放在多级目录中），默认为 None。

```
#构建测试用例集
discover=unittest.defaultTestLoader.discover('../baidu', pattern='*_test.py')
#执行测试
runner = unittest.TextTestRunner()
runner.run(discover)
```

4.3.5 Html 测试报告

采用 HTMLTestRunner 实现产生 Html 测试报告，python3.X 需要改写

HTMLTestRunner 下载地址 http://tungwaiyip.info/software/HTMLTestRunner.html

HTMLTestRunner 针对 Python3 进行修改

第 94 行，将 import StringIO 修改成 import io

第 539 行，将 self.outputBuffer = StringIO.StringIO()修改成 self.outputBuffer= io.StringIO()

第 642 行，将 if not rmap.has_key(cls):修改成 if notcls in rmap:

第 766 行，将 uo = o.decode(‘latin-1‘)修改成 uo = e

第 775 行，将 ue = e.decode(‘latin-1‘)修改成 ue = e

第 631 行，将 print >> sys.stderr, ‘\nTime Elapsed: %s‘ %(self.stopTime-self.startTime)修改成 print(sys.stderr, ‘\nTimeElapsed: %s‘ % (self.stopTime-self.startTime))

使用方法

```
fp=open(filename, 'wb')
runner = HTMLTestRunner.HTMLTestRunner(stream=fp,title='报告标题',description='报告说明' )
#执行测试
runner.run(Suite)    # Suite 为 TestSuite 测试套件
```

产生不同文件测试报告

```
#构建测试用例集
suite1=unittest.TestSuite()
suite2=unittest.TestSuite()
suite1.addTest(simple_test('test_add'))
suite1.addTest(simple_test('test_two'))
#测试套件组合在一起
alltests = unittest.TestSuite((suite1, suite2))
now=time.strftime("%Y%m%d%H%M%S", time.localtime())
filename=r'./report/'+now+r'result.html'
fp=open(filename, 'wb')
runner = HTMLTestRunner.HTMLTestRunner(stream=fp,title='报告标题',description='报告说明' )
#执行测试
runner.run(alltests)
```

4.3.6 屏蔽测试用例执行

skip 装饰器，用来暂时屏蔽不想执行的测试用例
@unittest.skip(reason): skip(reason) 无条件跳过装饰的测试，并说明跳过测试的原因。
@unittest.skipIf(reason): skipIf(condition,reason) 条件为真时，跳过装饰的测试，并说明跳过测试的原因。
@unittest.skipUnless(reason): skipUnless(condition,reason) 条件为假时，跳过装饰的测试，并说明跳过测试的原因。
@unittest.expectedFailure(): expectedFailure()测试标记为失败。

```
class MyTest(unittest.TestCase):
    def test_add(self):
        a = calculator(1, 2)
        print("测试用例 add_test")
        self.assertEqual(a.add(), 3)
    @unittest.skip
    def test_minus(self):
        a = calculator(1, 2)
        print("测试用例 minus_test")
        self.assertEqual(a.minus(), 3)

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

4.3.7 断言

assertEqual(a,b，[msg]):断言 a 和 b 是否相等，相等则测试用例通过。
assertNotEqual(a,b，[msg]):断言 a 和 b 是否相等，不相等则测试用例通过。
assertTrue(x，[msg])：断言 x 是否 True，是 True 则测试用例通过。
assertFalse(x，[msg])：断言 x 是否 False，是 False 则测试用例通过。
assertIs(a,b，[msg]):断言 a 是否是 b，是则测试用例通过。
assertNotIs(a,b，[msg]):断言 a 是否是 b，不是则测试用例通过。
assertIsNone(x，[msg])：断言 x 是否 None，是 None 则测试用例通过。
assertIsNotNone(x，[msg])：断言 x 是否 None，不是 None 则测试用例通过。
assertIn(a,b，[msg])：断言 a 是否在 b 中，在 b 中则测试用例通过。
assertNotIn(a,b，[msg])：断言 a 是否在 b 中，不在 b 中则测试用例通过。
assertIsInstance(a,b，[msg])：断言 a 是是 b 的一个实例，是则测试用例通过。
assertNotIsInstance(a,b，[msg])：断言 a 是是 b 的一个实例，不是则测试用例通过。
assertRaises(Exceptm,a,value,[msg]):value 值是否有异常
msg='测试失败时打印的信息']

4.3.8 拓展

TestCase 方法
setUpClass():必须使用@classmethod 装饰器,所有 test 运行前运行一次
tearDownClass():必须使用@classmethod 装饰器,所有 test 运行完后运行一次

4.4 unittest 框架实现参数化

nose-parameterized 实现 unittest 参数化
pip 安装
pip install nose-parameterized
参考资料 https://github.com/wolever/parameterized

```
import unittest
from nose_parameterized import parameterized

class TestAdd(unittest.TestCase):
    l = [
        ('add01', 1, 1, 2),
        ('addaa', 2, 2, 4),
        ('add789', 3, 3, 6),
    ]
    @parameterized.expand(l)
    def test_add(self, name, a, b, c):
        print(a + b, c)
        self.assertEqual(a + b, c)
        # self.assertEqual(a + b, c)
def suite():
    return unittest.makeSuite(TestAdd, "test_")
if __name__ == '__main__':
    runner = unittest.TextTestRunner(verbosity=2)
    runner.run(suite())
```

用例名称使用参数

```
import unittest
from parameterized import parameterized
def custom_name_func(testcase_func, param_num, param):
    return "%s_%s" % (
        testcase_func.__name__,
        parameterized.to_safe_name("_".join(str(x) for x in param.args)),
    )
class AddTestCase(unittest.TestCase):
    @parameterized.expand([
        (2, 3, 5),
        (2, 3, 5),
    ], testcase_func_name=custom_name_func)
    def test_add(self, a, b, expected):
        self.assertEqual(a + b, expected)
```

5.邮件发送功能

5.1 简单的邮件发送

5.1.1 简单的文本邮件发送

```
#coding=utf-8
from email.mime.text import MIMEText
import smtplib

msg = MIMEText('hello，send by python...','plain','utf-8')
from_addr = 'jiafu_082@163.com'#发送邮箱地址
password = 'pj123456'#邮箱授权码，非登陆密码
to_addr = '512800328@qq.com'#收件箱地址
smtp_server = 'smtp.163.com'#smtp 服务器
msg['From'] = from_addr#发送邮箱地址
msg['To'] = to_addr#收件箱地址
msg['Subject'] = 'the frist mail'#主题
server = smtplib.SMTP(smtp_server,25)
# server.set_debuglevel(1)
print(from_addr)
print(password)
server.login(from_addr,password)
server.sendmail(from_addr,to_addr,msg.as_string())
server.quit()
```

5.1.2 html 邮件发送

```
#coding=utf-8
from email.mime.text import MIMEText
import smtplib

msg = MIMEText('<html><h1>你好！</h1></html>','html','utf-8')
from_addr = 'jiafu_082@163.com'#发送邮箱地址
password = 'pj123456'#邮箱授权码，非登陆密码
to_addr = '512800328@qq.com'#收件箱地址
smtp_server = 'smtp.163.com'#smtp 服务器
msg['From'] = from_addr#发送邮箱地址
msg['To'] = to_addr#收件箱地址
msg['Subject'] = 'the frist mail'#主题
server = smtplib.SMTP(smtp_server,25)
# server.set_debuglevel(1)
print(from_addr)
print(password)
```

```
server.login(from_addr,password)
server.sendmail(from_addr,to_addr,msg.as_string())
server.quit()
```

5.2 使用邮件发送报告

代碼

```
# coding=utf8
import os, time, datetime
import smtplib
from email.mime.text import MIMEText

def sentmail(file_new):
    '''
    定义发送邮件函数
    '''
    mail_from = 'jiafu_082@163.com'  # 发信邮箱
    mail_to = '512800328@qq.com'  # 收信邮箱
    smtp_server = 'smtp.163.com'  # smtp 服务器
    f = open(file_new, 'rb')  # 定义正文
    mail_body = f.read()
    f.close()
    msg = MIMEText(mail_body, _subtype='html', _charset='utf-8')
    msg['Subject'] = u"wal-mart bulk po 测试报告"  # 定义标题
    msg['date'] = time.strftime('%a, %d %b %Y %H:%M:%S %z')  # 定义发送时间（不定义的可能有的邮件客户端会不显示发送时间）
    msg['From'] = mail_from  # 发送邮箱地址
    msg['To'] = mail_to  # 收件箱地址
    msg['Subject'] = 'the frist mail'  # 主题
    smtp = smtplib.SMTP(smtp_server, 25)
    # 用户名密码
    smtp.login(mail_from, 'pj123456')
    smtp.sendmail(mail_from, mail_to, msg.as_string())
    smtp.quit()
    print('email has send out !')



def sendreport():
    '''
    查找测试报告，调用发邮件功能
    '''
    result_dir = r'D:\app\client\jia\product\12.1.0\client_1'
    lists = os.listdir(result_dir)
    lists.sort(
        key=lambda fn: os.path.getmtime(result_dir + "\\" + fn) if not os.path.isdir(result_dir + "\\" + fn) else 0)
    print(u'最新测试生成的报告：  ' + lists[-1])
    # 找到最新生成的文件
```

```
    file_new = os.path.join(result_dir, lists[-1])
    print(file_new)
    # 调用发邮件模块
    sentmail(file_new)


if __name__ == '__main__':
    sendreport()
```

6.page Object 模式

6.1 config 文件读取

6.1.1.读取配置文件

-read(filename) 直接读取 ini 文件内容
-sections() 得到所有的 section，并以列表的形式返回
-options(section) 得到该 section 的所有 option
-items(section) 得到该 section 的所有键值对
-get(section,option) 得到 section 中 option 的值，返回为 string 类型
-getint(section,option) 得到 section 中 option 的值，返回为 int 类型

```
# myapp.config
[db]
host = 127.0.0.1
port = 3306
user = root
pass = root
# ssh
[ssh]
host = 192.168.1.101
user = huey
pass = huey
```

读取文件内容代码

```
config = configparser.ConfigParser()  # 实例化
path = os.path.abspath(r'./config/myapp.config')
config.read(path)  # 传入读取文件的地址
sections = config.sections()  # 得到所有的 section
print('sections:', sections)
options=config.options('db')  # 得到该 section 的所有 option
print('options:',options)
items = config.items('db')
print('items:', items)
string=config.get('db', 'host')#得到 section 中 option 的值，返回为 string 类型
print('db>host(str):',string)
int_z=config.getint('db','port') #得到 section 中 option 的值，返回为 int 类型
```

```
print('db>port(int):',int_z)
```

6.1.2.写入配置文件

-add_section(section) 添加一个新的 section
-set(section, option, value) 对 section 中的 option 进行设置
-write(open(path, "w")) 将修改的内容写回配置文件

```
# 写配置文件
config.set('db', 'host', '127.0.1.127')  # 更新指定 section，option 的值
config.set('db', "host1", "new-value")  # 写入指定 section 增加新 option 和值
config.add_section('a_new_section')  # 增加新的 section
config.set('a_new_section', 'new_key', 'new_value')  # 新的 section 加入值
config.write(open(r'./config/myapp.config', "w"))  # 写回配置文件
```

6.2 二次封装

如：启动浏览器

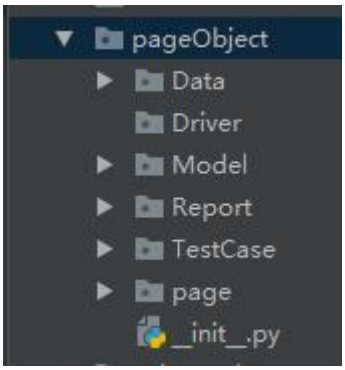
```
#Model/common/browser.py
from selenium import webdriver
def driver(name='chrome'):
    if name.lower() == 'chrome':
        return webdriver.Chrome(executable_path=r'../../Driver/chromedriver.exe')
    elif name.lower() == 'firefox':
        return webdriver.Firefox()
    elif name.lower() == 'ie':
        return webdriver.Ie()
```

如：对于查找元素进行二次封装，找不到元素截图

```
def findele(driver, located):
    try:
        return driver.find_element(located)
    except:
        # 截图
        path = os.path.abspath(r"./jietu/baidu1.png")  # 使用 config 文件制定地址
        dr.get_screenshot_as_file(path)
```

6.3 项目实践

参考：http://www.cnblogs.com/yufeihlf/p/5764099.html
目录结构



Driver 用于存放驱动
Data 用于存放测试数据
Model 功能模块，存放配置函数及公共类
Report 测试报告
TestCase 测试用例
Page 页面元素
Model：模块作用，存放相同的代码

简单的实例

页面对象存储：

```
from selenium.webdriver.common.by import By
class loginpage():
    inputs=(By.ID,'kw')
    submit=(By.ID,'su')
```

使用页面对象：

```
print(*loginpage.inputs)
driver = webdriver.Chrome()
driver.get('http://www.baidu.com')
driver.find_element(*loginpage.inputs).send_keys('selenium')
time.sleep(3)
driver.quit()
```

6.4 综合运用

1.使用配置文件参数化基本的参数
2.二次封装常用方法函数
3.unittest 管理测试用例