第七章节 自动化测试常用函数（C++方向）

本节重点

- 元素定位
- 操作测试对象
- 窗口
- 等待
- 导航
- 弹窗
- 文件上传
- 浏览器参数

1. 元素的定位

web自动化测试的操作核心是能够找到页面对应的元素，然后才能对元素进行具体的操作。

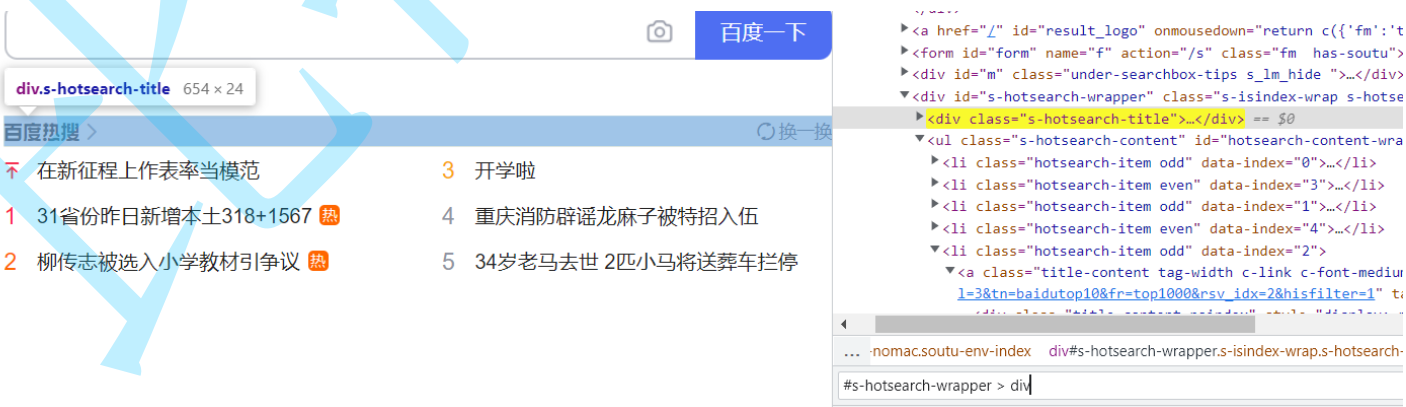
常见的元素定位方式非常多，如id，classname，tagname，xpath，cssSelector

常用的主要由cssSelector和xpath

1.1 cssSelector

选择器的功能：选中页面中指定的标签元素

选择器的种类分为基础选择器和复合选择器，常见的元素定位方式可以通过id选择器和子类选择器来进行定位。



定位百度首页的“百度热搜”元素，可以使用通过id选择器和子类选择器进行定位：`#s-hotsearch-wrapper > div`

“搜索输入框元素”：`#kw`

“百度一下按钮”：`#su`

1.2 xpath

XML路径语言，不仅可以在XML文件中查找信息，还可以在HTML中选取节点。

xpath使用路径表达式来选择xml文档中的节点

xpath语法中：

1.2.1 获取HTML页面所有的节点

`//*`

1.2.2 获取HTML页面指定的节点

`//[指定节点]`

> //ul：获取HTML页面所有的ul节点
>
> //input：获取HTML页面所有的input节点

1.2.3 获取一个节点中的直接子节点

`/`

> //span/input

1.2.4 获取一个节点的父节点

`..`

> //input/.. 获取input节点的父节点

1.2.5 实现节点属性的匹配

`[@...]`

> //*[@id='kw'] 匹配HTML页面中id属性为kw的节点

1.2.6 使用指定索引的方式获取对应的节点内容

注意：xpath的索引是从1开始的。

> 百度首页通过：//div/ul/li[3] 定位到第三个百度热搜标签

`更便捷的生成selector/xpath的方式：右键选择复制"Copy selector/xpath"`

案例：如果想要匹配到百度首页指定的新闻文本或者节点集:，直接使用`#hotsearch-content-wrapper > li`能够满足吗？

> 问题：既然可以手动复制`selector/xpath的方式`，为什么还有了解语法？

手动复制的selector/xpath表达式并不一定满足上面的唯一性的要求，有时候也需要手动的进行修改表达式

案例：百度首页（需要登陆百度账号）右侧的热搜，复制li标签下的a标签，复制好的的selector为：`#title-content`，xpath为：`//*[@id="title-content"]`，同学们可以手动操作一下，手动复制的表达式是否唯一呢？

2. 操作测试对象

获取到了页面的元素之后，接下来就是要对元素进行操作了。常见的操作有点击、提交、输入、清除、获取文本。

2.1 点击/提交对象

`click()`

```
#找到百度一下按钮并点击
driver.find_element(By.CSS_SELECTOR, "#su").click()
```

2.2 模拟按键输入

`send_keys("")`

```
driver.find_element(By.CSS_SELECTOR, "#kw").send_keys("迪丽热巴")
```

2.3 清除文本内容

输入文本后又想换一个新的关键词，这里就需要用到`clear()`

```
driver.find_element(By.CSS_SELECTOR, "#kw").send_keys("迪丽热巴")
time.sleep(1)
driver.find_element(By.CSS_SELECTOR,"#kw").clear()
time.sleep(1)
driver.find_element(By.CSS_SELECTOR, "#kw").send_keys("古力娜扎")
```

2.4 获取文本信息

如果判断获取到的元素对应的文本是否符合预期呢？获取元素对应的文本并打印一下~~

获取文本信息：`text`

```
text = driver.find_element(By.XPATH,'//*[@id="hotsearch-content-
wrapper"]/li[1]/a/span[2]').text
```

```
print(f"text:{text}")
```

问题：是否可以通过`text`获取到“百度一下按钮”上的文字“百度一下”呢？尝试一下

注意：文本和属性值不要混淆了。获取属性值需要使用方法`get_attribute("属性名称")`；

2.5 获取当前页面标题

`title`

```
title = driver.title
```

2.6 获取当前页面URL

`current_url`

```
url = driver.current_url
```

3. 窗口

> 打开一个新的页面之后获取到的title和URL仍然还是前一个页面的?

当我们手工测试的时候，我们可以通过眼睛来判断当前的窗口是什么，但对于程序来说它是不知道当前最新的窗口应该是哪一个。对于程序来说它怎么来识别每一个窗口呢？每个浏览器窗口都有一个唯一的属性句柄（handle）来表示，我们就可以通过句柄来切换

3.1 切换窗口

1）获取当前页面句柄：

`driver.current_window_handle`

3）获取所有页面句柄：

`driver.window_handles`

3）切换当前句柄为最新页面：

```
curWindow = driver.current_window_handle
allWindows = driver.window_handles

for window in allWindows:
    if window != curWindow:
```

```
6           driver.switch_to.window(window)
```

注意：执行了`driver.close()`之前需要切换到未被关闭的窗口

3.2 窗口设置大小

1）窗口的大小设置

```
#窗口最大化
driver.maximize_window()
#窗口最小化
driver.minimize_window()
#窗口全屏
driver.fullscreen_window()
#手动设置窗口大小
driver.set_window_size(1024,768)
```

3.3 屏幕截图

> 我们的自动化脚本一般部署在机器上自动的去运行，如果出现了报错，我们是不知道的，可以通过抓拍来记录当时的错误场景

```
driver.save_screenshot('../images/image.png')
```

代码演示

```
#简单版本
driver.save_screenshot('../images/image.png')

#高阶版本
filename = "autotest-"+datetime.datetime.now().strftime('%Y-%m-%d-%H%M%S')+'.png'
driver.save_screenshot('../images/'+filename)
```

3.4 关闭窗口

```
driver.close()
注意：窗口关闭后driver要重新定义
```

4. 弹窗

弹窗是在页面是找不到任何元素的，这种情况怎么处理？使用selenium提供的Alert接口

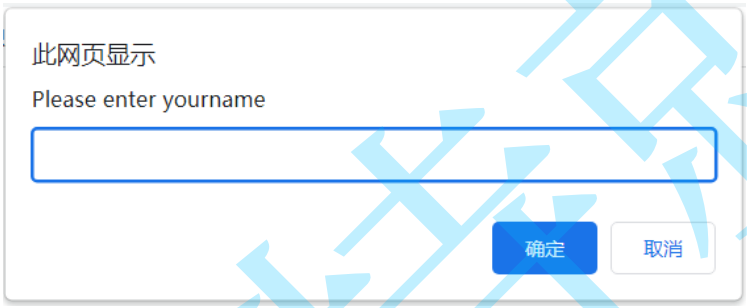
4.1 警告弹窗+确认弹窗

警告弹窗

确认弹窗

```
alert = driver.switchTo.alert
//确认
alert.accept()
//取消
alert.dismiss()
```

4.2 提示弹窗




```
alert = driver.switchTo.alert
alert.send_keys("hello")
alert.accept()
alert.dismiss()
```

5. 等待

通常代码执行的速度比页面渲染的速度要快，如果避免因为渲染过慢出现的自动化误报的问题呢？可以使用selenium中提供的三种等待方法：

5.1 强制等待

`time.sleep ()`

优点：使用简单，调试的时候比较有效

缺点：影响运行效率，浪费大量的时间

5.2 隐式等待

隐式等待是一种智能等待，他可以规定在查找元素时，在指定时间内不断查找元素。如果找到则代码继续执行，直到超时没找到元素才会报错。

`implicitly_wait ()` 参数：秒

示例：

```
1 #隐式等待5秒
2 driver.implicitly_wait(5)
```

隐式等待作用域是整个脚本的所有元素。即只要driver对象没有被释放掉（`driver.quit()`），隐式等待就一直生效。

优点：智能等待，作用于全局

5.3 显示等待

显示等待也是一种智能等待，在指定超时时间范围内只要满足操作的条件就会继续执行后续代码

`WebDriverWait(driver,sec).until(functions)`

`functions`：涉及到selenium.support.ui.ExpectedConditions包下的`ExpectedConditions`类

> ExpectedConditions下涉及到的方法：
>
> https://www.selenium.dev/selenium/docs/api/py/webdriver_support/selenium.webdriver.support.expected_conditions.html

示例：

```
1 from selenium.webdriver.support import expected_conditions as EC
2
3 wait = WebDriverWait(driver,2)
4 wait.until(EC.invisibility_of_element((By.XPATH,'//*
  [@id="2"]/div/div/div[3]/div[1]/div[1]/div')))
```

`ExpectedConditions` 预定义方法的一些示例：

方法	说明
`title_is(title)`	检查页面标题的期望值
`title_contains(title)`	检查标题是否包含区分大小写的子字符串的期望值
`visibility_of_element_located(locator, str])`	检查元素是否存在于页面的DOM上并且可见的期望值。
`presence_of_element_located (locator, str])`	用于检查元素是否存在于页面的DOM上的期望值
`visibility_of (element)`	检查已知存在于页面DOM上的元素是否可见的期望
`alert_is_present ()`	检查是否出现弹窗

优点：显示等待是智能等待，可以自定义显示等待的条件，操作灵活

缺点：写法复杂

> 隐式等待和显示等待一起使用效果如何呢?

测试一下

```
#隐式等待设置为10s，显示等待设置为15s，那么结果会是5+10=15s吗?
driver.implicitly_wait(10)
wait = WebDriverWait(driver,15)
start = time.time()
try:
    res = wait.until(EC.presence_of_element_located((By.XPATH,'//*[@id="2"]/div/div/div[3]/div[1]/div[1]/div/div/div')))
except:
    end = time.time()
    print("no such element")

driver.quit()

print(end-start)
```

结果：重试多次，设置10秒的隐式等待和15秒的显式等待导致20秒后发生超时

> 小提示：
>
> 不要混合隐式和显式等待，可能会导致不可预测的等待时间。

6. 浏览器导航

常见操作：

1）打开网站

```
driver.get("https://tool.lu/")
```

2）浏览器的前进、后退、刷新

```
driver.back()
driver.forward()
driver.refresh()
```

> 案例：百度首页测试https://tool.lu/标签入口

7. 文件上传

点击文件上传的场景下会弹窗系统窗口，进行文件的选择。

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

但是可以使用sendkeys来上传指定路径的文件，达到的效果是一样的

```
driver.get("file:///D:/file/%E6%AF%94%E7%89%B9%E6%95%99%E5%8A%A1/%E6%B5%8B%E8%AF%95/selenium4html/selenium-html/upload.html")
ele = driver.find_element(By.CSS_SELECTOR,"body > div > div > input[type=file]")
ele.send_keys("D:\\file\\test.txt")
```

8. 浏览器参数设置

1)设置无头模式

```
options = webdriver.ChromeOptions()
options.add_argument("-headless")
driver =
webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()),options
=options)
```

2)页面加载策略

> `options.page_load_strategy = '加载方式'`

页面加载方式主要有三种类型：

策略	说明
normal	默认值, 等待所有资源下载
eager	DOM 访问已准备就绪, 但诸如图像的其他资源可能仍在加载
none	完全不会阻塞WebDriver

```
options = webdriver.ChromeOptions()
options.page_load_strategy = 'eager'
driver =
webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()),options
=options)
```