第 5 章

WebDriver API 初级应用案例

在 Selenium 的 WebDriver 类中包含了大部分 UI 自动化 API 操作。第 4 章讲解元素定位时所用到的定位方法属于与定位相关的 API 操作。本章主要目标是将众多 API 初级操作中常用的方法分类并进行讲解。这类初级方法有一个共同的特点,就是功能实现单一,实际应用过程中需组合以实现具体应用场景。根据使用属性将这些常用方法分成 4 类进行讲解。

5.1 获取页面属性操作

页面属性多用于自动化测试用例的断言,即预期结果与实际结果的对比。获取页面上的关键信息来判断用例执行结果,在本节中使用 Python 自带的 assert 断言关键字来完成代码执行结果的判断。

5.1.1 获取页面 Title 属性值

前面提到过自动化测试操作的范围是 HTML 页面中的 body 标记对部分。head 标记对中只有 Title 标记对可视,它可协助测试用例脚本完成代码执行结果的断言。以百度首页搜索为例,代码如下:

```
#第 5 章/Obtain_title.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#搜索内容
driver.find_element_by_xpath('//*[@id="kw"]').send_keys('网易云思课帮')
sleep(2)
#单击进行搜索
driver.find_element_by_xpath('//*[@id="su"]').click()

#获取页面的 title 标签值,并将实际结果存入 actual 变量
```

```
actual = driver.title
#将预期结果存入 expect 变量
expect = '网易云思课帮_百度搜索'

#使用 assert 进行断言
assert actual,expect

sleep(2)
driver.quit()
```

5.1.2 获取页面源码

获取页面源码和获取 Title 文本的作用相似,都是用来对操作结果页面进行断言。当 Title 信息无法完成结果判断时,可选用此方法查看预期结果在结果页面中是否存在,代码如下:

```
#第 5 章/Obtain_source.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#搜索内容
driver.find_element_by_xpath('//*[@id="kw"]').send_keys('网易云思课帮')
sleep(2)
#单击进行搜索
driver.find_element_by_xpath('//*[@id="su"]').click()

#获取页面的 title 标签值,并将实际结果存入 actual 变量
actual = driver.page_source
#将预期结果存入 expect 变量
expect = '网易云思课帮_百度搜索'

#使用 assert 进行断言
assert expect in actual

sleep(2)
driver.quit()
```

5.1.3 获取页面元素文本信息

当页面元素有固定文本信息时,也可以选择获取元素文本的方式进行断言。这种方法和获取页面源码进行比较功能是一样的。以百度首页打开的正确性进行断言为例,此次选

择页脚处“关于百度”进行比较，代码如下：

```
#第 5 章/Obtain_text.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#获取页面元素文本值，并将实际结果存入 actual 变量
actual = driver.find_element_by_xpath('//*[@id="bottom_layer"]/div[1]/p[2]/a').text
#将预期结果存入 expect 变量
expect = '关于百度'
print(actual)
#使用 assert 进行断言
assert expect , actual

sleep(2)
driver.quit()
```

5.1.4 获取并设置当前窗口大小

M 版网站运行于移动端浏览器，在 UI 自动化测试时多采用模拟器或真机方式进行。事实上在自动化验证过程中，PC 端浏览器完全可以满足 M 版网站的功能测试。首先获取浏览器窗口大小，将窗口设置为手机常见的分辨率来执行自动化脚本。执行结束后再将浏览器窗口恢复至初始大小，这是因为通常浏览器有记忆功能，再次打开时会与上一次设置大小保持一致。这种情况下执行非 M 版网站脚本时会造成操作元素遮挡，从而导致执行失败。以 M 版百度为例进行演示，代码如下：

```
#第 5 章/Obtain_size.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://m.baidu.com/')

#获取当前浏览器窗口大小，返回的是字典对象
position = driver.get_window_size()
print("当前浏览器所在位置的宽:",position['width'])
print("当前浏览器所在位置的高:",position['height'])

#设置当前浏览器窗口大小，宽度 508 为 Chrome 浏览器宽度下限
driver.set_window_size(width=508, height=760,windowHandle='current')
sleep(2)
```

```
print(driver.get_window_size())

#在M版百度页执行搜索操作
driver.find_element_by_xpath('//*[@id="index-kw"]').send_keys('网易云思课帮')
sleep(2)
driver.find_element_by_xpath('//*[@id="index-bn"]').click()
sleep(2)
#恢复浏览器窗口为全屏状态
driver.maximize_window()
driver.quit()
```

5.2　输入操作

输入操作是页面交互操作中最基本的一种方式。在广义的输入概念中键盘及鼠标操作,甚至语音及视频都可以算是输入。本节所讲的输入为狭义的文本输入。有些操作在本书前面的示例中已经用到过。

5.2.1　输入文本操作

在 Selenium 中,文本输入用到的方法是 Sendkeys(text)。它的使用方法很简单,通过 text 参数将输入文本填入定位元素指定的位置,代码如下:

```
#第5章/Input_text.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
#在打开的浏览器中输入网址,这也算是一种输入
driver.get('http://www.baidu.com/')
#send_keys()的基本输入方式
#当前面定位元素不是可输入文本框时,输入失败,而执行无异常
driver.find_element_by_xpath('//*[@id="kw"]').send_keys('网易云思课帮')
driver.find_element_by_xpath('//*[@id="su"]').click()
sleep(2)

driver.quit()
```

5.2.2　单选、复选框操作

单选、复选框的操作本质上还是单击,使用 click()方法就可以完成此操作。选择是输入之外使用最多的页面交互操作。常用于搜索页面条件选择、个人设置页面信息项选择、按钮等操作。页面选择中还有一类特殊的选择操作,即文件选择,类似操作在 OA 系统中较为

常见，需要配合键盘及鼠标来完成，在后面的鼠标及键盘操作中会分别讲解。关于 click() 的使用方法以孔夫子旧书网登录页面为例，代码如下：

```
#第 5 章/Input_choice.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://login.kongfz.com/')

#输入用户名及密码
driver.find_element_by_xpath('//*[@id="username"]').send_keys('Thinkerbang')
driver.find_element_by_xpath('//*[@id="password"]').send_keys('123456')
sleep(2)
#单击完成"记住密码"框的选择操作
driver.find_element_by_xpath('//*[@id="login"]/div[2]/div[1]/input').click()
sleep(1)
driver.find_element_by_xpath('//*[@id="login"]/div[3]/input').click()
sleep(2)

driver.quit()
```

示例中所用网站是笔者在日常使用过程中为数不多的在自动化脚本执行时不弹出验证码的网站。当然，若是同一 IP 频繁执行登录脚本也会弹出验证问题。各位读者在练习过程中首选网站肯定是自己公司的待测网站。网站验证码的出现本身就是为了防止自动化脚本的执行页出现的。各网站层出不穷且花样翻新的验证方式本质上也是保障网站安全的一种方式。网上会看到一些自动化脚本绕过验证码的方法，如果方法可行，就说明此网站或验证方式存在安全漏洞。在自己日常测试的软件上执行自动化测试，最常见的方法是找开发者暂时关掉验证功能。这在 UI、接口、性能测试时是通用的方法。

5.2.3 下拉列表操作

下拉列表控件的常规操作是单击菜单项，在弹出列表中选择目标数据，以百度首页中的设置菜单为例，页面 HTML 片断如图 5.1 所示。

图 5.1 百度首页“设置”菜单 HTML 片断

在定位页面元素时，会发现无法通过开发者模式中的定位箭头完成控件定位。这是因为菜单是动态显示的，当鼠标离开“设置”时，菜单会被隐藏。鼠标带着定位箭头单击“设置”完成的是它本身的元素定位。这时可以在要定位元素上右击并选择“检查”项，同样可以完成 HTML 中的元素定位，代码如下：

```
#第 5 章/Input_drop - down.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#全屏窗口页面，设置菜单在最右侧，需要在全屏模式下操作
driver.maximize_window()
#单击"设置"菜单
driver.find_element_by_xpath('//*[@id="s-usersetting-top"]').click()
#在弹出菜单中选择开启热榜，此步执行成功的前提条件是热榜处于关闭状态
driver.find_element_by_xpath('//*[@id="s-user-setting-menu"]/a[2]').click()
sleep(2)

driver.quit()
```

5.2.4 复位操作

本节主要演示几个自动化操作中的辅助操作。辅助类的方法独立出现的意义不大，很多时候也能找到相关替代方法，这主要取决于个人编写自动化脚本的习惯。下面进行归类介绍。

1. 清除操作

常见的清除有清空输入文本，以及页面刷新、复位等，代码如下：

```
#第 5 章/Input_clear.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://news.baidu.com/')

#输入要搜索的新闻关键字并进行搜索操作
driver.find_element_by_xpath('//*[@id="ww"]').send_keys('北斗导航')
sleep(2)
driver.find_element_by_xpath('//*[@id="s_btn_wr"]').click()
sleep(2)
#再次输入搜索文本前，需执行清除文本操作
driver.find_element_by_xpath('//*[@id="kw"]').clear()
```

```
sleep(2)
driver.find_element_by_xpath('//*[@id="kw"]').send_keys('网易云思课帮')
sleep(2)
driver.find_element_by_xpath('//*[@id="su"]').click()
sleep(2)
#刷新页面操作效果类似于按 F5 键进行刷新
#有些页面输入刷新后会恢复初始状态,例如百度注册页面刷新,其效果相当于清除输入文本
driver.refresh()

driver.quit()
```

2. 等待操作

常见的时间等待操作有显式等待、隐式等待。显式等待目的性更强,在设定时间内等待某个具体定位元素的加载。这样做的优点是脚本执行过程中延时针对性强,缺点是一旦目标元素无法完成加载,脚本执行会中断。显式等待实现过程会在本书第 7 章讲解。

隐式等待是针对整个页面而言的,在设定时间内,如果整个页面加载完成,则自动中止等待,反之则执行完设定时间后继续执行后面的脚本。time 模块下的 sleep()的用法和隐式等待相似,不同之处在于 sleep()所设置的等待时间值没有弹性,设置的时间消耗完毕后才会进行后续脚本的执行,代码如下:

```
#第 5 章/Input_wait.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://news.baidu.com/')

#隐式等待 5s,页面加载在 5s 以内完成则自动进行下一步操作
#此方法适用于页面载入内容较多且等待时间不确定时使用
driver.implicitly_wait(5)

#输入要搜索的新闻关键字并进行搜索操作
driver.find_element_by_xpath('//*[@id="ww"]').send_keys('北斗导航')
sleep(2)
driver.find_element_by_xpath('//*[@id="s_btn_wr"]').click()
sleep(2)

driver.quit()
```

5.3 鼠标操作

鼠标类的操作在软件中最常见到的是单击、双击、右击 3 种。部分地图类和 OA 办公类软件在页面中也会出现拖曳操作,例如百度网盘、迅捷在线思维导图等。读者有必要掌握一

些基本的鼠标类操作方法。

Selenium 中的鼠标操作基本在 ActionChains 类的下面。

5.3.1 单击操作

Click()单击操作在之前已多次使用,代码如下:

```
#第 5 章/Mouse_click.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('https://www.baidu.com/')

driver.find_element_by_id("kw").send_keys('网易云思课帮')
#对定位对象进行单击操作
driver.find_element_by_id('su').click()
sleep(2)

driver.quit()
```

5.3.2 双击操作

双击操作分 3 步实现。首先定位需要实现双击的元素,其次声明 ActionChains 类,将定位对象传入,最后在类对象中选择 double_click()方法,执行 perform()进行双击动作并提交,代码如下:

```
#第 5 章/Mouse_double.py
from selenium.webdriver.common.action_chains import ActionChains
from selenium import webdriver
from time import sleep

driver = webdriver.Firefox()
driver.get('https://www.kongfz.com/')

#定位需要双击的元素,页面右上角的只读文本:网罗天下图书
double = driver.find_element_by_xpath("//*[@id='navHeader']/div/div[1]/span[1]")
sleep(2)
#对定位对象进行双击操作
#双击后的效果是定位文字被全部选中
ActionChains(driver).double_click(double).perform()
sleep(2)

driver.quit()
```

5.3.3 右击操作

右击操作的实现并不复杂，复杂的是后续操作。通常右击后会出现菜单项，然后在菜单项中选择需要操作的项进行下一步操作。很多 Web 页面中的右击菜单属于浏览器本身，页面中没有与之相关的元素，这时需要配合键盘进行操作。在 5.4 节键盘操作讲解完成后再实现右击菜单的操作。本节实现右击效果，代码如下：

```
#第 5 章/Mouse_right.py
from selenium.webdriver.common.action_chains import ActionChains
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('https://www.baidu.com/')

#定位需要右击的元素
right = driver.find_element_by_id("kw")
#对定位对象进行右击操作
ActionChains(driver).context_click(right).perform()
sleep(2)

driver.quit()
```

5.3.4 鼠标拖曳操作

拖曳类操作有两种方法进行支撑：第 1 种是 drag_and_drop()方法，此方法有两个传入参数，实现将第 1 个定位元素移动至第 2 个定位元素的坐标处；第 2 种是计算偏移量 drag_and_drop_by_offset()方法，此方法有 3 个传入参数，第 1 个参数是待移动参数，后两个参数分别是待移动参数相对于当前位置的 x、y 轴偏移量。以 jqueryui 网站实现拖曳操作为例，其代码如下：

```
#第 5 章/Mouse_drag.py
from selenium.webdriver.common.action_chains import ActionChains
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('https://jqueryui.com/resources/demos/draggable/scroll.html')

#定位页面中的前两个元素
above1 = driver.find_element_by_id("draggable")
```

```
above2 = driver.find_element_by_id("draggable2")

#执行移动操作,将第 1 个方块向左下角进行偏移,偏移量 x:100,y:200
ActionChains(driver).drag_and_drop_by_offset(above1,xoffset = 100,yoffset = 200).perform()
sleep(2)

#执行移动操作,将第 2 个方块移动到第 1 个方块的位置
ActionChains(driver).drag_and_drop(above2,above1).perform()
sleep(2)

driver.quit()
```

5.4 键盘操作

Web 页面交互操作中用到键盘的操作有 3 种：输入操作、组合热键操作和右击菜单进行选择操作。

5.4.1 输入操作

输入操作的基本用法在前面示例中已多次使用到，通过 send_keys()方法实现输入效果，代码如下：

```
#第 5 章/Keyboard_input.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#send_keys()的基本输入方式
#当前面定位元素不是可输入文本框时,输入失败,而执行无异常
driver.find_element_by_xpath('//*[@id = "kw"]').send_keys('网易云思课帮')
driver.find_element_by_xpath('//*[@id = "su"]').click()
sleep(2)

driver.quit()
```

5.4.2 组合热键操作

在 send_keys()操作中输入英文内容与直接按下键盘上相应的键效果等价。本节要讲解的组合热键的重点是非内容输入类的辅助键的使用方法，代码如下：

```
#第5章/Keyboard_hot-key.py
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.sogou.com')

#定位搜狗搜索页面输入框元素,按下F12键以便打开开发者模式
driver.find_element_by_id("query").send_keys(Keys.F12)
sleep(2)

#输入搜索内容
driver.find_element_by_id("query").send_keys("Selenium")
sleep(2)

#按下Ctrl + A键全选文字
driver.find_element_by_id("query").send_keys(Keys.CONTROL,'a')
sleep(2)

#按下Ctrl + X键剪切文字,再按下Ctrl + V键进行粘贴
driver.find_element_by_id("query").send_keys(Keys.CONTROL,'x')
sleep(1)
driver.find_element_by_id("query").send_keys(Keys.CONTROL,'v')

#按下Enter键进行内容搜索
driver.find_element_by_id("query").send_keys(Keys.ENTER)
sleep(2)

driver.close()
```

5.4.3 右击菜单进行选择操作

在5.3.3节中实现了右击操作,接下来对右击菜单中的选项进行选择操作。以百度首页为例,在百度首页LOGO上进行右击,在弹出的快捷菜单中选择“图片另存为”选项,如图5.2所示。

接下来会弹出图片另存窗口,如图5.3所示。

要完成图片保存操作,需要解决3个问题。首先要在定位图片上执行右击操作以便弹出菜单,其次需要按键盘的向上按钮选择需要的菜单项进行回车操作,最后需要回车完成图片保存操作。由于右击菜单和图片另存弹窗均不是页面内容,因此无法完成定位。此时借助键盘热键可以辅助完成这些操作,代码如下:

图 5.2　百度首页右击菜单项

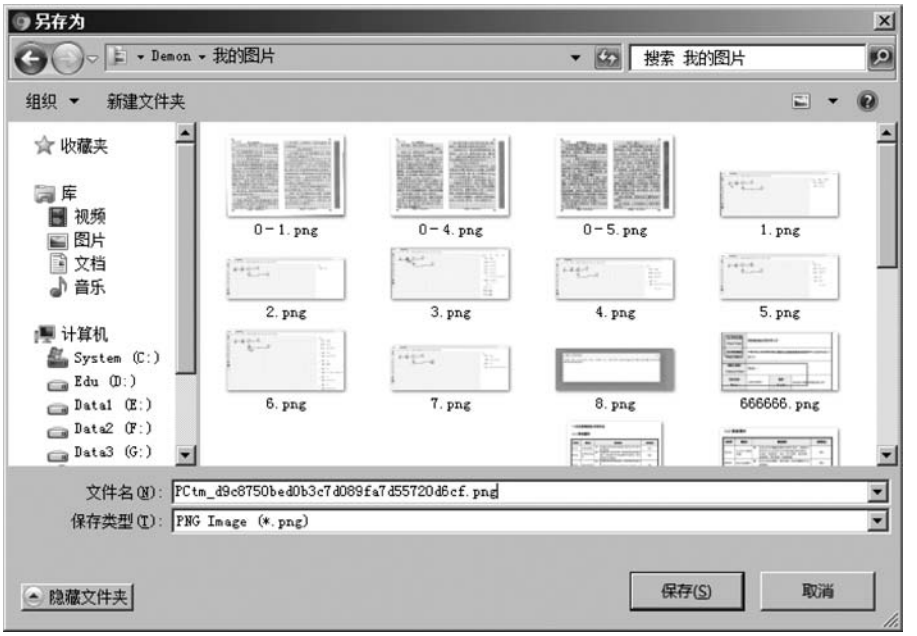

图 5.3　图片另存窗口

```
#第5章/Keyboard_right-key.py
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys
from selenium import webdriver
from time import sleep
import pyautogui

driver = webdriver.Chrome()
driver.get('https://www.baidu.com/')

#定位需要右击的图片元素
right = driver.find_element_by_xpath("//*[@id='s_lg_img']")
#对定位对象进行右击操作
action = ActionChains(driver)
action.context_click(right).perform()
action.send_keys(Keys.ARROW_DOWN)
sleep(2)

#在弹出的快捷菜单中选择"图片另存为"选项
for i in range(7):
    #此处引入'pyautogui'模块,down与Keys.ARROW_DOWN作用相同
    pyautogui.typewrite(['down'])
    sleep(1)

#'return'的作用与Keys.ENTER的作用相同
pyautogui.typewrite(['return'])
sleep(2)
#再次按下'return',效果为在弹出另存窗口上回车并保存
pyautogui.typewrite(['return'])
sleep(2)

driver.quit()
```

在本例中无法选择图片存储路径,也无法为另存文件命名。在第6章会讲解一款针对窗口元素的操作工具AutoIt,可以配合代码完成Web页面与Windows窗口之间的交互操作。

5.5 执行JavaScript脚本操作

Python语言支持多语言执行,Selenium下也保留了类似功能。WebDriver类中的excute_script()方法可以直接运行JavaScript代码,因此在UI自动化测试过程中,可以借助JavaScript语言来辅助完成一些Selenium实现困难的操作。

5.5.1 JavaScript 弹窗操作

页面弹窗操作始终是 UI 自动化操作过程中 Selenium 无法实现的一项操作。页面弹窗分 3 种情况：Web 页面模拟弹窗、Windows 弹窗和 JavaScript 弹窗。

Web 页面模拟弹窗的本质仍然是 Web 页面，常规页面元素定位就可以实现。

Windows 弹窗的操作会在第 6 章具体讲解处理方法。

JavaScript 弹窗触发的脚本在 Web 页面中，触发弹出后，弹窗本身不在页面中，Selenium 无法完成定位及后续操作。通过执行 JavaScript 代码的方式可以完成对这类弹窗的确认操作。由于公网 Web 页面中 JavaScript 弹窗没有适合的例子，所以需要编写一个 HTML 页面实现弹窗效果，代码如下：

```
<!-- 第 5 章/popup.html -->
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf8">
        <title>JavaScript 弹窗示例</title>
    </head>
    <body>
        <div align="center">
        <h4>JavaScript 弹窗示例</h4>
        <input type="button" onclick="showPro()" value="prompt 弹窗按钮"/>
        <input type="button" onclick="showAlert()" value="Alert 弹窗按钮"/>
            <br><br><br>
        <span id="textSpan"></span>

        </div>
    </body>
    <script>

        function showPro(){
            document.getElementById("textSpan").innerHTML="";
            con = prompt("这是 prompt 弹窗");
        }
        function showAlert(){
            document.getElementById("textSpan").innerHTML="";
            alert("这是 Alert 弹窗");
        }
    </script>
</html>
```

通过示例页面，对两种 JavaScript 弹窗进行操作，代码如下：

```
#第 5 章/JS_popup.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('D:/PO_test/book05/popup.html')
sleep(2)

#获取 alert 对话框的按钮,单击按钮,弹出 alert 对话框
driver.find_element_by_xpath('/html/body/div/input[2]').click()
#获取 alert 对话框
alert = driver.switch_to.alert
sleep(2)

#获取警告对话框的内容,并打印警告对话框内容
print(alert.text)
#接受 alert 弹窗
alert.accept()
sleep(2)

#获取 confirm 对话框的按钮,单击按钮,弹出 confirm 对话框
driver.find_element_by_xpath('/html/body/div/input[1]').click()
#获取 confirm 对话框
dialog_box = driver.switch_to.alert
sleep(2)

#获取对话框的内容,并打印警告对话框内容
print(dialog_box.text)
#接受弹窗
dialog_box.accept()
sleep(2)

driver.quit()
```

5.5.2 JavaScript 输入操作

JavaScript 可以模拟 Selenium 进行元素定位并实现输入、单击等操作,代码如下:

```
#第 5 章/JS_input.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')
sleep(2)
```

```
#定位搜索输入框,并输入搜索内容
driver.execute_script('document.getElementById("kw").value = "网易云思课帮"')
sleep(2)
#定位"百度一下"按钮,并单击搜索操作
driver.execute_script('document.getElementById("su").click()')
sleep(2)

driver.quit()
```

5.5.3 JavaScript 滑屏操作

页面操作中有一种特例情况,待操作元素不在本页面可视范围内显示。JavaScript 中有两种方式可以完成滑屏操作:第 1 种方式是通过定位目标位置的元素来完成滑屏;第 2 种方式是通过执行定位坐标偏移量来完成滑屏。推荐使用第 1 种方式滑屏,使用此滑屏方式定位更精确。当页面滑屏目标位置没有可定位元素时,可采用第 2 种方式。通过 Scroll() 方法实现滑屏,它通过设置 y 坐标的方式实现。此方法受显示器屏幕分辨率影响,只能算出大致的位置,可以作为元素定位滑屏方法的一个备用方案。代码如下:

```
#第 5 章/JS_move.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get("http://www.kongfz.com")
sleep(1)

#网页自动隐藏不可视栏目元素,只有第 2 种方法合适
#拖动至"民国书刊拍卖"栏目
driver.execute_script("scroll(0,7500)")
sleep(2)

#获取当前可视范围内所有 class = 'floor - big - title - name'元素
#全页面共 9 个栏目,每次可随机获取 3～4 个栏目元素
target_elem = driver.find_elements_by_xpath("//span[@class = 'floor - big - title - name']")
#打印获取栏目标题元素数
print(len(target_elem))
sleep(2)

#使用第 1 种元素定位滑屏方式将页面滑至可视范围内的位置
driver.execute_script("return arguments[0].scrollIntoView();", target_elem[1])
sleep(5)

driver.quit()
```

5.5.4 JavaScript 辅助操作

JavaScript 还有很多对页面的操作可用作 UI 自动化测试过程中的辅助操作。此处简单列出两种用法。有兴趣的读者可以参考 JavaScript 使用手册学习更多使用方法，代码如下：

```
#第 5 章/JS_aux.py
from selenium import webdriver
from time import sleep

driver = webdriver.Chrome()
driver.get('http://www.baidu.com/')

#标红定位元素,当某一操作断言异常时,此方法可在截屏前对异常进行标注
js = 'var q = document.getElementById(\"kw\"); q.style.border = \"2px solid red\";'
driver.execute_script(js)
sleep(2)

#隐藏元素,将获取的图片元素隐藏
img = driver.find_element_by_xpath("//*[@id='lg']/img")
driver.execute_script('$(arguments[0]).fadeOut()', img)
sleep(2)

driver.quit()
```