UI 自动化 day03 随堂笔记

今日内容

- 浏览器和元素的操作方法(API)
- 键盘和鼠标操作
- 元素等待

元素操作方法

```
"""
元素操作方法
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 需求：打开注册A页面，完成以下操作
# 1).通过脚本执行输入用户名：admin；密码：123456；电话号码：
18611111111；电子邮件：123@qq.com
driver.find_element_by_id('userA').send_keys('admin')
driver.find_element_by_id('passwordA').send_keys('123456')
tel = driver.find_element_by_id('telA')
tel.send_keys('18611111111')
```

```
driver.find_element_by_name('emailA').send_keys('123@qq.com')

# 2).间隔3秒，修改电话号码为：18600000000
# 注意：在使用操作中，一般对于输入框元素，都要先执行清空，再执行输入，避免操作错误
# 清空操作：元素对象.clear()
sleep(3)
tel.clear()
tel.send_keys('18600000000')

# 3).间隔3秒，点击‘注册’按钮
sleep(3)
driver.find_element_by_css_selector('button').click()

# 4).间隔3秒，关闭浏览器
# 5).元素定位方法不限

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

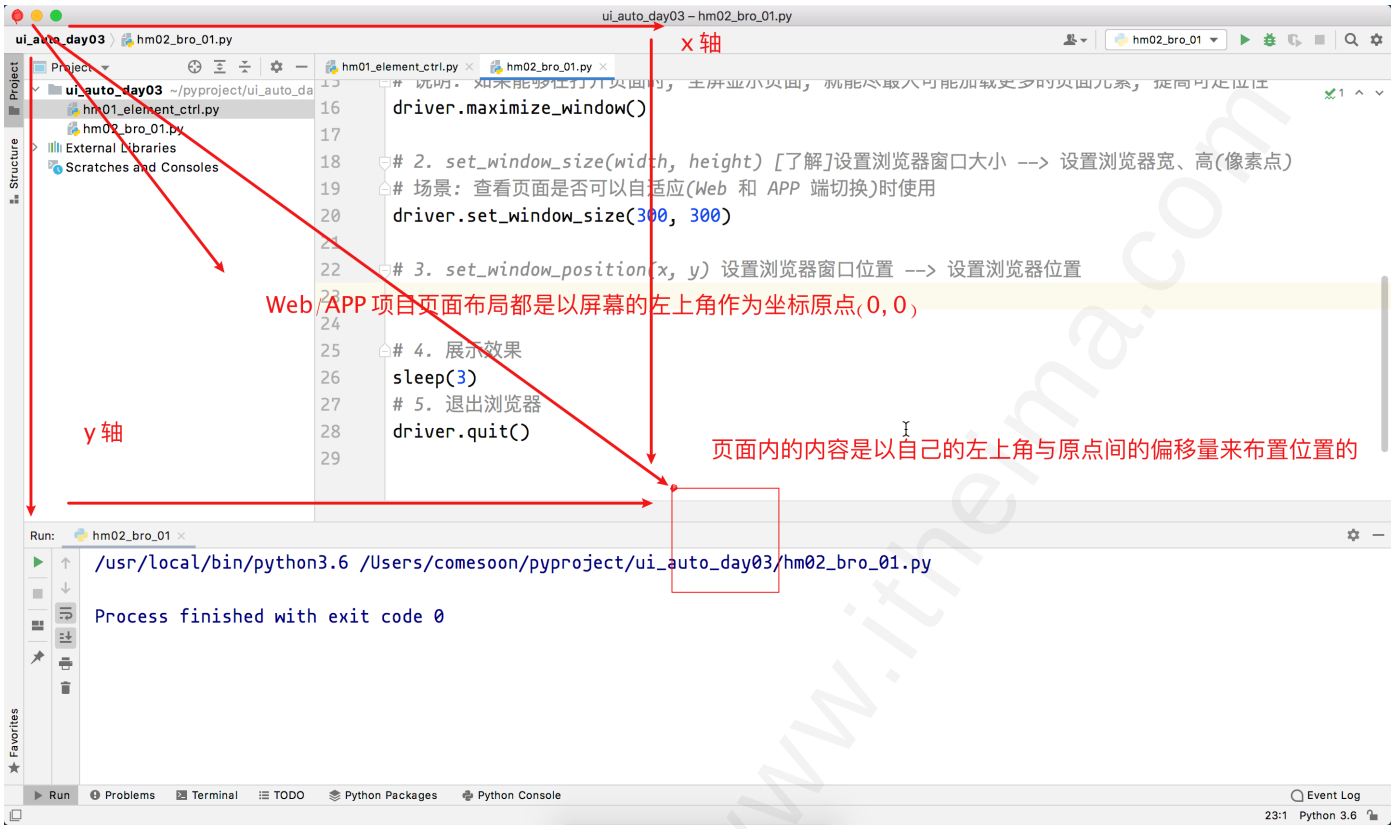
浏览器操作方法

Part1: 设置最大化/大小/位置

```
"""
浏览器操作方法 Pat1:
"""
# 1. 导入模块
```

```python
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%8CA.html')

# 1. maximize_window() [重点]最大化浏览器窗口 --> 模拟浏览器最大化按钮
# 说明: 如果能够在打开页面时, 全屏显示页面, 就能尽最大可能加载更多的页面元素, 提高可定位性
driver.maximize_window()

# 2. set_window_size(width, height) [了解]设置浏览器窗口大小 --> 设置浏览器宽、高(像素点)
# 场景: 查看页面是否可以自适应(Web 和 APP 端切换)时使用
driver.set_window_size(300, 300)

# 3. set_window_position(x, y) [了解]设置浏览器窗口位置 --> 设置浏览器位置
driver.set_window_position(300, 300)

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

扩展: Web/APP 项目页面布局坐标系示意



Part2: 后退/前进/刷新

```
"""
浏览器操作方法: Part2
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('http://www.baidu.com')
```

```
# 定位搜索框输入内容
driver.find_element_by_id('kw').send_keys('黑马马')
# 点击搜索按钮
driver.find_element_by_id('su').click()
sleep(2)

# 后退: 浏览器对象.back()
driver.back()
sleep(2)

# 前进: 浏览器对象.forward()
driver.forward()
sleep(2)

# 刷新[重点]: refresh()
# 说明: 刷新动作是重新向服务器发起当前页面的请求!
driver.refresh()

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

Part3: 关闭/退出/获取页面标题和 URL 地址

```
"""
浏览器操作方法: Part3
"""
# 1. 导入模块
from time import sleep
```

```python
from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%8CA.html')

# 点击新开新浪网页
driver.find_element_by_partial_link_text('访问').click()
sleep(2)

# 7. close() 关闭当前窗口 --> 模拟点击浏览器关闭按钮
# 8. quit() 关闭浏览器驱动对象 --> 关闭所有程序启动的窗口
# 9. title 获取页面title
# 10. current_url 获取当前页面URL

# 场景: 浏览器的标题和 URL 地址属性, 可以用来做断言使用
print('关闭前页面标题:', driver.title)
print('关闭前页面地址:', driver.current_url)

# 说明: 在没有实现浏览器页面切换操作前, close() 方法关闭的是原始页面!
# 场景: 关闭单个页面时使用
driver.close()

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
# 说明: 关闭所有页面
driver.quit()
```

获取元素信息方法

Part1: 获取大小/文本/属性值

```
"""
获取元素信息方法: Part1
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 1. size 返回元素大小
# 场景: 用于判断页面元素布局尺寸是否合理时使用
username = driver.find_element_by_id('userA')
print('目标元素的尺寸为:', username.size)

# 2. text 获取元素的文本
# 场景: 用于切换页面后, 对页面内容特定元素的文本信息的获取(用作断言使
用)
btn = driver.find_element_by_tag_name('button')
print('目标元素的文本为:', btn.text)

# 3. get_attribute("xxx") 获取属性值, 传递的参数为元素的属性名
# 场景: 有些情况下, 需要获取目标元素的特定属性值作为判断依据或数据
# 语法: 元素对象.get_attribute("属性名")
link = driver.find_element_by_link_text('新浪')
```

```
print('目标元素的地址为:', link.get_attribute('href'))

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

Part2: 判断元素是否可见/可用/可选中

```
"""
获取元素信息方法: Part2
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 4. is_displayed() 判断元素是否可见
# 说明: 该方法多用于对元素在页面内显示效果的判断时使用(元素不显示不意
味着一定无法定位)
span = driver.find_element_by_name('sp1')
print('目标元素是否显示:', span.is_displayed())

# 5. is_enabled() 判断元素是否可用
# 说明: 该方法多用于判断目标元素是否可以进行交互时使用
```

```
can_btn = driver.find_element_by_id('cancelA')
print('目标元素是否可用:', can_btn.is_enabled())

# 6. is_selected() 判断元素是否选中，用来检查复选框或单选按钮是否被选中
# 场景：如购物车页面，不全选商品，不让结算
check = driver.find_element_by_id('lyA')
print('目标元素是否被选中:', check.is_selected())

# 扩展：判断条件
if check.is_selected():  # 选中的判断
    pass
if not check.is_selected():  # 未选中的判断
    pass

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

鼠标操作

右键单击[了解]

```
"""
鼠标操作：鼠标右键
"""
# 1. 导入模块
from time import sleep
```

```
from selenium import webdriver
from selenium.webdriver import ActionChains

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 需求：打开注册页面A，在用户名文本框上点击鼠标右键
# 0> 定位目标元素
username = driver.find_element_by_id('userA')
# 1> 实例化鼠标对象(关联浏览器对象)
action = ActionChains(driver)
# 2> 调用鼠标方法
# 说明：鼠标右键只能展示右键菜单内容，而菜单中的元素无法操作!
action.context_click(username)
# 3> 执行方法：该方法必须调用，否则上述代码无效!!!!!!
action.perform()

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

左键双击[了解]

```
"""
鼠标操作：左键双击
"""
# 1. 导入模块
```

```python
from time import sleep

from selenium import webdriver
from selenium.webdriver import ActionChains

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%8CA.html')

# 需求：打开注册页面A，输入用户名admin，暂停3秒钟后，双击鼠标左键，选中admin
username = driver.find_element_by_id('userA')
username.send_keys('admin')
sleep(3)

# 说明：使用键盘快捷键 Ctrl + A，也能实现全选
# 1> 实例化鼠标对象(关联浏览器对象)
action = ActionChains(driver)
# 2> 调用鼠标动作(传入目标元素)
action.double_click(username)
# 3> 执行方法
action.perform()

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

拖拽操作[了解]

```
"""
鼠标操作：拖拽操作
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver
from selenium.webdriver import ActionChains

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/drag.html')

# 需求：打开‘drag.html’页面，把红色方框拖拽到蓝色方框上
red = driver.find_element_by_id('div1')
blue = driver.find_element_by_id('div2')

# 1> 实例化鼠标对象(关联浏览器对象)
action = ActionChains(driver)
# 2> 调用方法(传入目标元素)
action.drag_and_drop(red, blue)
# 3> 执行方法
action.perform()

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

悬停操作[重点]

```
"""
鼠标操作：悬停操作
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver
from selenium.webdriver import ActionChains

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 定位目标元素
btn = driver.find_element_by_tag_name('button')

# 实例化鼠标对象
action = ActionChains(driver)
# 调用鼠标方法
# 说明：该方法在实际应用中，处理悬停鼠标才会出现的菜单时使用
# 注意：该方法执行时，不要动鼠标!!!!!!
action.move_to_element(btn)
# 执行方法
action.perform()

# 另一种鼠标操作的写法:(在其他编程语言中称为链式编程)
# ActionChains(driver).move_to_element(btn).perform()

# 4. 展示效果
```

```
sleep(3)
# 5. 退出浏览器
driver.quit()
```

键盘操作

案例实现

```
"""
键盘操作
"""
# 1. 导入模块
from time import sleep

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

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%
8CA.html')

# 需求：打开注册A页面，完成以下操作
# 1). 输入用户名：admin1，暂停2秒，删除1
username = driver.find_element_by_id('userA')
username.send_keys('admin1')
sleep(2)
# 删除：BACK_SPACE 等价于 BACKSPACE
username.send_keys(Keys.BACK_SPACE)
```

```
# 2). 全选用户名: admin, 暂停2秒
# 说明: macOS 系统需要使用 Command + a
username.send_keys(Keys.CONTROL, 'a')  # Windows 系统
# username.send_keys(Keys.COMMAND, 'a')  # macOS 系统
sleep(2)

# 3). 复制用户名: admin, 暂停2秒
username.send_keys(Keys.CONTROL, 'c')  # Windows 系统
# username.send_keys(Keys.COMMAND, 'c')  # macOS 系统
sleep(2)

# 4). 粘贴到密码框
# 说明: 之所以能够复制完内容后, 在任意位置处可以进行粘贴, 是通过系统的剪切板实现的
password = driver.find_element_by_id('passwordA')
password.send_keys(Keys.CONTROL, 'v')  # Windows 系统
# password.send_keys(Keys.COMMAND, 'v')  # macOS 系统

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

元素等待

隐式等待

说明：定位元素时，如果能定位到元素则直接返回该元素，不触发等待； 如果不能定位到该元
素，则间隔一段时间[不可控制]后再去定位元素； 如果在达到最大时长时还没有找到指定元素，则抛出元素不存在的异常 NoSuchElementException 。

案例实现

```
"""
元素等待：隐式等待
"""
# 1. 导入模块
from time import sleep

from selenium import webdriver

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%8CA%E7%AD%89%E5%BE%85.html')

# 设置隐式等待
driver.implicitly_wait(10)

# 定位元素并输入
driver.find_element_by_id('userA').send_keys('admin')

# 注意：
# 1. 隐式等待是全局有效，只需要设置一次即可
# 2. 当隐式等待被激活时，虽然目标元素已经出现了，
# 但是还是会由于当前页面内的其他元素的未加载完成，而继续等待，进而增加代码的执行时长
```

```
# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

显式等待

说明：定位指定元素时，如果能定位到元素则直接返回该元素，不触发等待； 如果不能定位到该
元素，则间隔一段时间[可以控制]后再去定位元素； 如果在达到最大时长时还没有找到指定元素，则抛出超时异常 TimeoutException

案例实现

```
"""
元素等待：显式等待
"""
# 1. 导入模块
from time import sleep

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

# 2. 实例化浏览器对象
driver = webdriver.Chrome()
# 3. 打开页面
driver.get('file:///Users/comesoon/Desktop/page/%E6%B3%A8%E5%86%8CA%E7%AD%89%E5%BE%85.html')
```

```
# 需求：打开注册A页面，完成以下操作
# 1).使用显式等待定位用户名输入框，如果元素存在，就输入admin
# driver.find_element_by_id('userA').send_keys('admin')

# 设置显式等待：按住 Ctrl，鼠标左键点击类名，拷贝示例代码，根据实际需求修改对应参数即可
"""
driver: 浏览器对象
timeout: 超时时长
poll_frequency: 定位方法执行间隔时长(默认 0.5 秒)
"""
element = WebDriverWait(driver, 10, 1).until(lambda x:
x.find_element_by_id('userA'))
# 说明：元素操作方法没有代码提示，需要手写
element.send_keys('admin')

# 4. 展示效果
sleep(3)
# 5. 退出浏览器
driver.quit()
```

隐式等待和显示等待的对比

对比	元素个数	调用方式	异常类型
隐式等待	全局	浏览器对象调用	NoSuchElementException
显式等待	单个	实例化对象调方法	TimeoutException
扩展: 强制等待(sleep)	全局	方法	NoSuchElementException

今日任务

- 总结今日内容 -> 截图提交
- 完成今日练习 -> 百度设置修改 -> 代码贴图提交
- 预习任务 -> 截止到验证码处理
- 思考题: 能不能用自动化完成每日反馈提交(此段内容为自动化提交)