三天WEB自动化项目实战训练营

第一天：WEB自动化实战基础篇

金牌讲师：码尚教育-北凡老师
直播时间：2022-04-25 20：00

1. Web自动化测试需求和挑战
2. 测试环境手动搭建和自动搭建
3. Selenium的八大定位策略
4. XPath语法及常用函数
5. 元素定位调试方法实例演示
6. 自动化测试用例实战

一、Web自动化测试需求和挑战

1. Web自动化测现状
    1. 属于E2E测试
    2. 过去通过点点点
    3. 好的测试，还需要记录、调试网页的细节
2. 自动化测试需求
    1. 提速增效
    2. 解决双手
    3. 技能提升
3. 目前的主流工具
    1. Cypress
    2. Playwright
    3. Selenium

	Selenium	Cypress	Playwright
浏览器兼容	几乎所有✅	仅Chrome / Firefox	仅Chrome / Safari / Firefox
多标签/多窗口	支持✅	不支持❌	支持✅
创建测试速度	快	不支持❌	快
用户行为	支持	少量支持❌	支持
并行架构	自建	付费	自建
速度	一般	很快✅	很快✅
调试	不方便	方便	方便
文档	丰富✅	一般	一般

Selenium的优势：

1. 浏览器支持多，兼容好
2. 支持多种编程语言
3. 生态成熟、文档丰富
4. 进行App自动化测试，事半功倍

本次训练营，使用Selenium 4进行讲解

二、Selenium 自动化测试环境搭建

手动：

1. 查看浏览器的版本号
2. 查询操作系统的类型
3. 根据1和2选择浏览器驱动版本
4. 下载浏览器区别，放在指定的目录

自动：

```
pip install webdriver-helper
```

自动的完成：

1. 查看浏览器的版本号
2. 查询操作系统的类型
3. 根据1和2选择浏览器驱动版本
4. 下载浏览器区别，放在指定的目录

验证是否搭建好

```
from webdriver_helper import get_webdriver

driver = get_webdriver()  # 启动浏览器
driver.get("https://baidu.com")  # 控制浏览器
input()
driver.quit()  # 关闭浏览器
```

三、Selenium自动化实战

Web自动化测试基本流程：

1. 元素定位 （前提）
2. 元素交互
3. 断言

1. 元素定位

Selenium 提供了8个元素定位的API，可以分为三种：

1. a标签定位策略
2. 属性定位策略
3. 通用定位策略

使用方法是一样

```
driver.find_element(By.ID, 'btn')
```

- find_element 用来定位单个元素
- find_elements 用来定位多个元素

1. a标签定位策略

LINK_TEXT ： 精确匹配

PARTIAL_LINK_TEXT： 模糊匹配

```
driver.get("http://101.34.221.219:8010/")  # 控制浏览器

el = driver.find_element(By.PARTIAL_LINK_TEXT, "登")  # 定位元素

print(el.tag_name, el.text)  # 打印元素的信息
```

2. 属性定位策略

ID

NAME

TAG_NAME

CLASS_NAME

都属于元素的属性

```
<input
        id="search-input"
        name="wd"
        type="text"
        placeholder="其实搜索很简单^_^ !"
        value=""
        autocomplete="off"
>
```

```python
el = driver.find_element(By.TAG_NAME, "input")
print(el.tag_name, el.text)  # 打印元素的信息

el = driver.find_element(By.ID, "search-input")
print(el.tag_name, el.text)  # 打印元素的信息

el = driver.find_element(By.NAME, "wd")
print(el.tag_name, el.text)  # 打印元素的信息
```

4. 通用定位策略

CSS：CSS语法，不够直观

XPath ： 直观的层级结构

对于大部分的定位策略，其底层，是使用CSS或者XPath实现

对于CSS和XPath的底层，是通过什么实现的?

着重讲一下XPath:

1. XPath 是一种查询语言，支持逻辑表达式和函数
2. 可以实现复杂元素的动态定位
3. 可以用于App自动化测试的定位

2. XPath

层级直观

```
//*[@id="search-input"]
/html/body/div[3]/div/div[3]/form/div/input

/html/body/div/../div
```

1.XPath 语法

- / （开头） 根路径
  - `/html`
- // 任一级
  - `//div`， 任意路径下的div元素
- / (中间) 下一级
  - `//div/p` 任意路径下的div下的p
  - `//div//p` 任意路径下的div下的任意层级下的p
- . 当前节点（同级）

- .. 上一级
- [n] 序号 表示同级元素的序号
- @ 属性
    - `//input[@name="wd"]`

2. XPath的函数

函数是XPath另一个魅力，常用函数：

- text ：精确匹配
    - `$x("//*[text()='iphoneX新品发布了']")`
- contains： 模糊匹配
    - `$x("//*[contains(text(),'新品发')]")`
- starts-with：开头一致
    - `$x("//*[starts-with(text(),'iphoneX新品')]")`

3. 实战操练

1. 简单的例子

设计用例

列出步骤

1. 访问项目首页 : http://101.34.221.219:8010/
2. 选择登录按钮 : a.text = '登录'
3. 点击登录按钮 : a.click()
4. 定位账号输入框 xpath = '/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[1]/input'
5. 输入账号
6. 定位密码输入框 xpth = '/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[2]/div/input'
7. 输入密码
8. 定位登录按钮 xpah = '/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[3]/button'
9. 点击登录

编写代码

```
from selenium.webdriver.common.by import By
from webdriver_helper import get_webdriver

driver = get_webdriver()  # 启动浏览器
driver.get("http://101.34.221.219:8010/")  # 1

el = driver.find_element(By.LINK_TEXT, "登录")  #
el.click()  # 3

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[1]/input"
```

```
)  # 4
el.send_keys("beifan")  # 5

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[2]/div/input"
)
el.send_keys("123123")  # 7

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[3]/button"
)
el.click()

input()
driver.quit()  # 关闭浏览器
```

2. 复杂的例子

> - 模拟用户操作
> - 处理弹窗
> - 获取系统提示，进行断言

- 设计用例
  - 商品下单流程：
    - 登录
    - 选择商品
    - 创建订单
    - 选择收货地址
    - 选择支付方式
    - 提交订单
- 列出步骤
  - 登录
  - 选择商品 ： text = "vivo X5MAX L 移动4G 八核超薄大屏5.5吋双卡手机vivoX5max"
  - 点击商品
  - 选择立即购买 xpath = '/html/body/div[4]/div[2]/div[2]/div/div[3]/div[2]/button[1]'
  - 处理弹窗
  - 选择付款方式 xpath = '/html/body/div[4]/div/div[4]/ul/li[1]/span'
  - 处理弹窗
  - 选择提交订单按钮 xpath = '/html/body/div[4]/div/div[6]/div/form/div/button'
  - 点击提交按钮
  - 获取系统提示
  - 断言 系统提示的内容是 ”操作成功“
- 编写代码

```
from selenium.webdriver.common.by import By
from webdriver_helper import get_webdriver


```

```python
def get_msg():
    time.sleep(0.5)  # 等待-0.5秒
    el = driver.find_element(By.XPATH, "//p[@class='prompt-msg']")
    return el.text


def alert():
    time.sleep(2)
    driver.switch_to.alert.accept()  # 弹窗点击确定


driver = get_webdriver()  # 启动浏览器
driver.maximize_window()
driver.get("http://101.34.221.219:8010/")  # 1

el = driver.find_element(By.LINK_TEXT, "登录")  #
el.click()  # 3

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[1]/input"
)  # 4
el.send_keys("beifan")  # 5

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[2]/div/input"
)
el.send_keys("123123")  # 7

el = driver.find_element(
    By.XPATH,
"/html/body/div[4]/div/div[2]/div[2]/div/div/div[1]/form/div[3]/button"
)
el.click()

# 登录成功
msg = get_msg()

assert msg == "登录成功"

time.sleep(0.5)
driver.get("http://101.34.221.219:8010/")  # 回到首页

el = driver.find_element(By.PARTIAL_LINK_TEXT, "vivo")  # 选择商品
el.click()

# 窗口最新打开的切换
driver.switch_to.window(driver.window_handles[-1])

time.sleep(0.5)

el = driver.find_element(
    By.XPATH, "/html/body/div[4]/div[2]/div[2]/div/div[3]/div[2]/button[1]"
)
el.click()  # 立即购买

alert()  # 处理弹出
```

```
60
61  el = driver.find_element(By.XPATH,
    "/html/body/div[4]/div/div[4]/ul/li[1]/span")
62  el.click()  # 付款方式
63
64  alert()  # 处理弹窗
65
66  el = driver.find_element(By.XPATH,
    "/html/body/div[4]/div/div[6]/div/form/div/button")
67  el.click()  # 提交等待
68
69  msg = get_msg()  # 获取系统提示
70  assert msg == "操作成功"
71
72  input()
73  driver.quit()  # 关闭浏览器
```

新的东西：

- 代码复用
- 等待元素加载
- 窗口切换的问题
- 处理弹窗的问题

暴露一些问题：

- 流水账
- 等待
- 用例单一

引入pytest测试框架

引入自动等待

总结

1. 什么是自动化测试，为什么进行自动化测试
2. Web自动化测试的技术选型
3. Selenium自动化基础 + 实战

chrome：

- session
    - windows

- tab