web自动化测试框架训练营之实战基础篇

1.selenium的基本介绍

selenium是一个用于Web应用的工具：可以实现web自动化测试，以及爬虫。

特点：

- 开源软件
- 跨平台
- 核心功能：
  - 可以在多个浏览器上进行自动化测试
- 支持多语言

核心：

- 自动化控制浏览器
- 自动化控制页面内容
- 执行高级指令，比如js脚本代码

2.搭建web自动化测试的环境

- 基于Python环境搭建（Windows系统）
  - python环境：解释器
  - 安装pycharm：编写python代码的工具
  - 安装第三方库：selenium
    - pip install selenium
  - 浏览器：谷歌浏览器
  - 浏览器的驱动

注意：浏览器的版本一定要跟驱动版本号保持一致

浏览器驱动最新地址：https://googlechromelabs.github.io/chrome-for-testing/#stable

浏览器的驱动要放置项目中才能运行

验证环境搭建创建连接：

```
# 导包
import time

from selenium import webdriver

# 创建webdriver对象
# 获取驱动对象
driver = webdriver.Chrome()

# 通过驱动访问页面
driver.get("https://www.baidu.com/")
# driver.get("http:")

# 将页面最大化
driver.maximize_window()

# 强制等待
time.sleep(5)

# 关闭驱动对象
driver.quit()
```

3.元素的定位方式

八大定位方式：

- 通过id定位
- 通过name属性定位
- 通过class name属性定位
- 通过标签定位（不常用）
- 通过超链接的精确及模糊定位
- 常用：
    - 通过xpath定位
    - 通过css定位

定位器	描述
id	定位 id 属性与搜索值匹配的元素
name	定位 name 属性与搜索值匹配的元素
tag name	定位标签名称与搜索值匹配的元素
class name	定位class属性与搜索值匹配的元素（不允许使用复合类名）
link text	定位link t·ext可视文本与搜索值完全匹配的锚元素
partial link text	定位link text可视文本部分与搜索值部分匹配的锚点元素
xpath	定位与 XPath 表达式匹配的元素
css selector	定位 CSS 选择器匹配的元素

```
ID = "id"
XPATH = "xpath"
LINK_TEXT = "link text"
PARTIAL_LINK_TEXT = "partial link text"
NAME = "name"
TAG_NAME = "tag name"
CLASS_NAME = "class name"
CSS_SELECTOR = "css selector"
```

由于元素定位不到出现的原因：

- 元素不可用，只读，不可见
  - 通过js脚本执行让元素改变原来的属性之后再进行定位操作
- 元素出现动态的属性，比如id
  - M32iFFAJVyZi6gqL
  - 4N5vs7uinB7Q9aNs
  - 通过手写xpath进行定位
    - 调用判断元素开头以及结束或者内容进行定位
- 其他：
  - 加滚动进行定位
  - 元素未加载完成，使用等待完成定位
  - 元素在iframe子页面中需要切换
  - 元素在不同的页面中需要切换
  - 警告框，选择框特殊元素标签需要进行特殊处理

4.元素的其他操作

元素操作

- 点击元素：click()
- 输入内容：send_keys（）
- 清除文本：clear

元素属性获取

- 元素大小：size
- 元素文本：text
- 获取属性值：get_attribute（）
- 元素是否可见：is_display
- 元素是否可用：is_enabled（）

5.等待

因为在web中看到的元素，不一定写在html中，有可能是通过js代码的dom操作产生出来的，而js产生的元素，很可能先要获取数据，处理之后才会显示，那么不一定页面打开所有的元素就加载完全，需要有一定的等待时间。

强制等待

- 通过time.sleep(5)
- 固定等待5秒钟

隐式等待

- 在创建好驱动对象之后，调用隐式等待的方式传入等待时间即可

```
driver.implicitly_wait(5)
```

- 原理：
  - 元素在第一次就定位到的时候不会触发隐式等待的时长，直接操作元素
  - 如果第一次定位失败，那么会触发隐式等待的有效时长
  - 如果在有效时长定位到元素，那么执行操作
  - 如果没有在有效市场定位到元素，那么会报错：NoSuchElementException

显示等待

- 使用方式：
  - 导包

```
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

    - 使用：

```
el1 = WebDriverWait(driver, 5).until(EC.presence_of_element_located((By.XPATH, '//*[@id="1"]/div/h3/a')))
driver.find_element(By.XPATH, '//*[@id="1"]/div/h3/a').click()
```

- 原理
-
    - 每隔一定的时间不断尝试查找元素
    - 找到了就返回元素
    - 未找到就会报错：TimeoutException

6.后台登录案例

流程测试用例的设计以及验证码的处理方式

设计流程用例：

- 编写自动化的用例线性脚本
    - 完成功能冒烟测试
    - 输入账号
    - 输入密码
    - 输入验证码
    - 点击登录

```
# 导包
import time

from selenium import webdriver
from selenium.webdriver.common.by import By

# 创建webdriver对象
# 获取驱动对象
driver = webdriver.Chrome()

# 通过驱动访问页面
driver.get("http:")

# 将页面最大化
driver.maximize_window()

# 编写自动化的用例线性脚本
#
# - 完成功能冒烟测试
# - 输入账号
driver.find_element(By.XPATH,
"/html/body/form/table/tbody/tr/td[3]/table/tbody/tr[2]/td[2]/input").send_keys("admin")
# - 输入密码
driver.find_element(By.XPATH,
"/html/body/form/table/tbody/tr/td[3]/table/tbody/tr[3]/td[2]/input").send_keys("123456")
```

```
# - 输入验证码
driver.find_element(By.XPATH,
"/html/body/form/table/tbody/tr/td[3]/table/tbody/tr[5]/td[2]/input").send_keys("8888")
# - 点击登录
driver.find_element(By.XPATH, '//*[@id="login_btn"]').click()

# 强制等待
time.sleep(5)

# 关闭驱动对象
driver.quit()
```

7.验证码处理方式

- 关闭验证码功能
- 设置万能验证码
- 通过第三方打码平台识别验证码

验证码图片获取提取

```
# 截取验证码图片
driver.find_element(By.XPATH, '//*[@id="verify"]').screenshot("verify.png")
```

验证码图片的识别

通过第三方平台：

- 超级鹰：https://www.chaojiying.com/price.html
- 通过发送一个第三方的接口请求处理识别验证码
    - http://upload.chaojiying.net/Upload/Processing.php
- 然后获取响应信息的验证码结果

```
# 识别验证码
url = "http://upload.chaojiying.net/Upload/Processing.php"

data = {
    # 用户名
    "user": "tianqiu2",
    # 密码
    "pass": "ltqiu123456",
    # 用户id
    "sofid": "949627",
    # 验证码类型编号
    "codetype": 1902
}
```

```
# 打开读取图片
files = {"userfile": open("verify.png", "rb")}

resp = requests.post(url, data=data, files=files)
# print(resp.json())
res = resp.json()
if res["err_no"] == 0:
    print("识别成功")
    code = res["pic_str"]
    print(f"验证码数字为：{code}")
else:
    print("识别失败")
```

一般线性脚本和处理功能的脚本不会写在一起

具体的功能实现，比如验证码识别会进行独立的封装进行调用

封装验证码识别函数：

```
import requests


def imgcode(file):
    # 识别验证码
    url = "http://upload.chaojiying.net/Upload/Processing.php"

    data = {
        # 用户名
        "user": "tianqiu2",
        # 密码
        "pass": "ltqiu123456",
        # 用户id
        "sofid": "949627",
        # 验证码类型编号
        "codetype": 1902
    }
    # 打开读取图片
    files = {"userfile": open(file, "rb")}

    resp = requests.post(url, data=data, files=files)
    # print(resp.json())
    res = resp.json()
    if res["err_no"] == 0:
        print("识别成功")
        code = res["pic_str"]
        print(f"验证码数字为：{code}")
        return code
    else:
        print("识别失败")
        return False
```

调用验证码识别函数，获取验证码实际值：

```
# 截取验证码图片
driver.find_element(By.XPATH, '//*[@id="verify"]').screenshot("verify.png")
# 调用验证码识别函数，获取返回结果值
code = imgcode("verify.png")
```