web自动化测试框架训练营之实战升华篇

1.通过cookie保持自动登录状态

保持自动登录的方式

- 通过cookie信息的唯一标识id
- 一直保持不退出的状态，就可以实现自动登录

退出登录

- 注销登录状态
- 关闭浏览器
- 删除已登录状态的cookie信息

给页面设置cookie信息，刷新页面清除缓存实现保持登录状态

```
# 给页面设置cookie信息，刷新页面清除缓存实现保持登录状态
driver.add_cookie(
    {"name": "PHPSESSID", "value": "rh478t4dqjd729gu7ce3k8dp87"}
)
# 刷新页面
driver.refresh()
```

2.自动化获取cookie信息

在页面第一次登录成功之后，保持当前页面的cookie信息

在页面第二次及以后的登录过程中，直接使用保存的cookie信息

cookie信息的保存和使用以及判断是否登录状态：

```
# 保持页面cookie信息
def sava_cookie(driver):
    # 获取当前页面所有的cookie信息
    cookies = driver.get_cookies()
    with open("cookies.json", "w") as f:
        f.write(json.dumps(cookies))


# 使用页面的cookie信息
def load_cookie(driver):
    # 从本地文件cookies.josn文件中读取cookie信息
    try:
```

```
        driver.get("http:")
        with open("cookies.json") as f:
            cookies = json.loads(f.read())
        for cookie in cookies:
            driver.add_cookie(cookie)
        else:
            # 将页面所有cookie信息添加完成之后，刷新页面
            driver.refresh()
    except:
        pass


# 判断当前页面是否登录状态
def is_login(driver):
    if "管理员登录" in driver.title:
        print("需要进行登录")
        return True
    else:
        print("已登录状态")
        return False
```

通过获取和使用cookie信息完成自动化登录或者流程登录：

```
# 导包
import time

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

from utils import imgcode, sava_cookie, load_cookie, is_login

# 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()


# 使用cookie信息
load_cookie(driver)

if is_login(driver):
    # 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")
    # 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("msjy123")
    # - 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(code)
    # - 点击登录
    # input(":::")
    driver.find_element(By.XPATH, '//*[@id="login_btn"]').click()

    # 保存cookie信息
    sava_cookie(driver)


    # 强制等待
    time.sleep(5)

    # 关闭驱动对象
    driver.quit()
```

3.pytest测试框架的引入和基本使用

pytest是Python中的单元测试框架

pytest特点：

- 容易上手，入门简单，丰富的文档资料，文档中有很多实例可进行参考
- 支持参数化
- 执行用例过程中可以进行标记跳过用例，标记失败用例
- 支持重复执行失败的用例
- 具有很多的第三方插件，并且可以实现自定义扩展
- 便捷管理用例，方便和持续集成工具相互结合，便于生成测试报告

测试框架的核心作用：

- 找到测试用例
- 执行测试用例
- 管理测试用例
- 断言测试用例
- 生成测试报告

3.1pytest常用插件

pytest

- 测试框架本身

pytest-html

- 生成html测试报告

pytest-xdist

- 多线程运行

pytest-ordering

- 控制用例的执行顺序

pytest-rerunfailures

- 控制失败用例重跑

allure-pytest

- 生成allure测试报告

pytest-base-url

- 管理基础路径

在项目中安装所有插件：

- 新建一个requiremens.txt
- 写上所有插件的名字
- 安装命令：pip install -r requiremens.txt

3.2pytest默认的测试用例执行规则

- 包名必须是以test开头或者test结尾
- 模块名必须是以test开头或者test结尾
- 用例名必须是以test开头或者test结尾：
    - 函数
    - 方法
        - 实例方法
        - 类方法

3.3pytest两种执行方式

1.通过命令行执行：pytest -vs

2.通过主函数运行：

```
import pytest

if __name__ == "__main__":
    pytest.main()
```

3.4pytest标记跳过测试用例

无条件跳过用例

```
@pytest.mark.skip(reason="版本原因03用例不执行")
def test_login03():
    print("开始执行登录用例03")
```

有条件跳过用例

```
# 有条件跳过用例，默认条件不成立
@pytest.mark.skipif(2 > 10, reason="反例跳过不执行02")
def test_login02():
    print("开始执行登录用例02")
```

注意：条件不成立，用例执行，条件成立，那么用例不执行

3.5pytest控制测试用例的执行顺序

用例的执行默认顺序是按照包名，文件名以及函数名前后顺序执行的方式

pytest提供了可以修改执行用例顺序的插件pytest-ordering

```
import pytest


@pytest.mark.run(order=3)
def test_login01():
    print("开始执行登录用例01")


# 有条件跳过用例，默认条件成立
@pytest.mark.skipif(2 > 10, reason="反例跳过不执行02")
def test_login02():
    print("开始执行登录用例02")


@pytest.mark.run(order=2)
```

```
@pytest.mark.skip(reason="版本原因03用例不执行")
def test_login03():
    print("开始执行登录用例03")


@pytest.mark.run(order=1)
def test_login04():
    1 / 0
    print("开始执行登录用例04")
```

3.6pytest标记失败测试用例

标记预期会出现异常失败的测试用例，只有出现异常才符合预期，不出现异常反而不对

```
@pytest.mark.xfail(reason="异常用例：0不能当做被除数")
@pytest.mark.run(order=1)
def test_login04():
    # 1 / 0
    print("开始执行登录用例04")
```

如果符合预期失败：

- XFAIL结果
- 标记失败成功

如果不符合预期失败：

- XPASS结果
- 标记失败失败

3.7pytest标记参数化

对于相似的用例执行过程，但是使用的数据不一样，那么就可以使用参数化实现

参数的基本使用：

```
@pytest.mark.parametrize(["a", "b"], [(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1,
8)])
def test_login05(a, b):
    print("开始执行登录用例05")
    print(f"a的值是：{a},b的值是：{b},相加的值是：{a + b}")
```

登录案例的参数化实现流程：

```
@pytest.mark.parametrize("username, password, code", [(1, 2, 3), (4, 5, 6),(4, 5, 6),(4, 5,
6),(4, 5, 6),(4, 5, 6),(4, 5, 6),(4, 5, 6),(4, 5, 6),(4, 5, 6),(4, 5, 6),(4, 5, 6)])
def test_login06(username, password, code):
    # 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()

    # - 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(
        username)
    # - 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(
        password)
    # - 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(code)
    # - 点击登录
    # input(":::")
    driver.find_element(By.XPATH, '//*[@id="login_btn"]').click()
```

参数化一般会结合数据驱动进行自动化测试

数据驱动测试DDT:

- Data数据
- Driver驱动
- Tests用例

数据驱动测试的数据存储方式有很多种类型

- Text文本
- **Csv文件**
- Excel文件
- Json文件
- Yaml文件

通过csv的数据存储方式读取数据

```
# 实现数据驱动测试之读取csv数据符合参数化标准
def get_data():
    data_list = []
    c1 = csv.reader(open("666.csv"))
    # print(c1)
    for i in c1:
        # print(i)
        data_list.append(i)
```

```
    else:
        return data_list


print(get_data())
print(len(get_data()))
```

输出结果：

```
[['admin', '123456', '8888'], ['tinqiu', '123456', '8888'], ['beifan', '123456', '8888'],
['', '123456', '8888'], ['admin', '', '8888']]
```

```
@pytest.mark.parametrize("username, password, code", get_data())
def test_login06(username, password, code):
    # 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()

    # - 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(
        username)
    # - 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(
        password)
    # - 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(code)
    # - 点击登录
    # input(":::")
    driver.find_element(By.XPATH, '//*[@id="login_btn"]').click()

    time.sleep(2)
```