

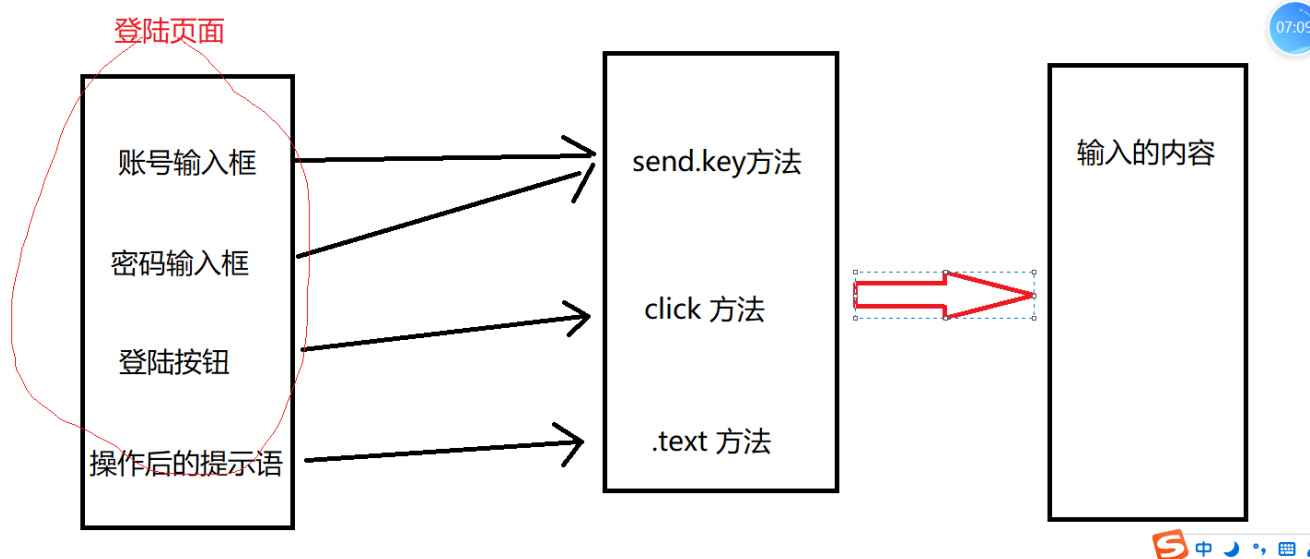
POM模型

一、POM模型

P: page, 页面, 把每个页面的**元素**封装成类或者函数

O: objec, 对象, 把传入的内容/操作看作每页对象

M: mode, 模型



汇总: 把每个页面里面的元素都定位, 并封装成函数/类, 到时候先要哪个页面元素就直接进行调用, 并且使用相应的操作方法

二、pom模型的思路, 实现步骤

pom只是一种思想/思维习惯, 并不是具体的框架, 是需要通过pytest框架来实现的。

- 1、定位元素的方法, 可以封装成函数
- 2、需要封装定位的元素属性
- 3、传入数据

三、pytest模块的划分, 需要依赖的包

common: 存放公共方法、元素定位方法、数据操作、驱动方法

config: 配置文件, 存放配置、IP, 域名, 数据库的配置

data: 数据层, 用于测试数据

report: 生成测试报告存放的目录

run_case: 执行测试用例主程序的目录

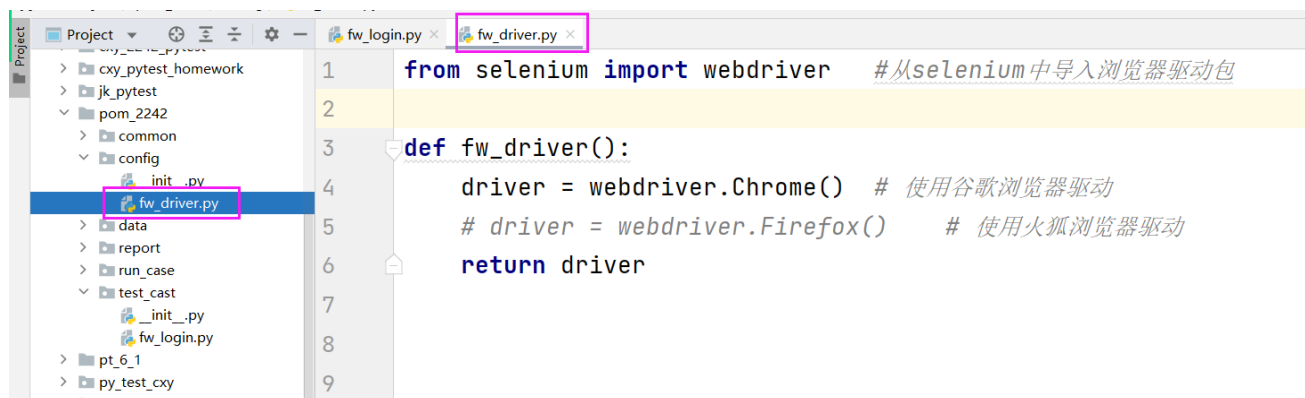
test_case: 存放测试用例

生成allure报告, 需要安装allure-pytest包

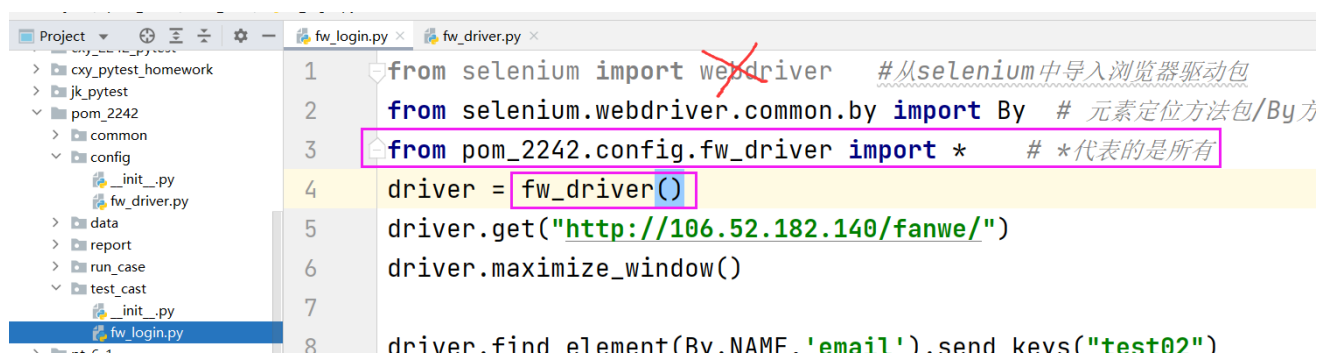
四、搭建pytest框架的操作步骤

1、把方维登陆断言成功的测试用例放到test_case下面

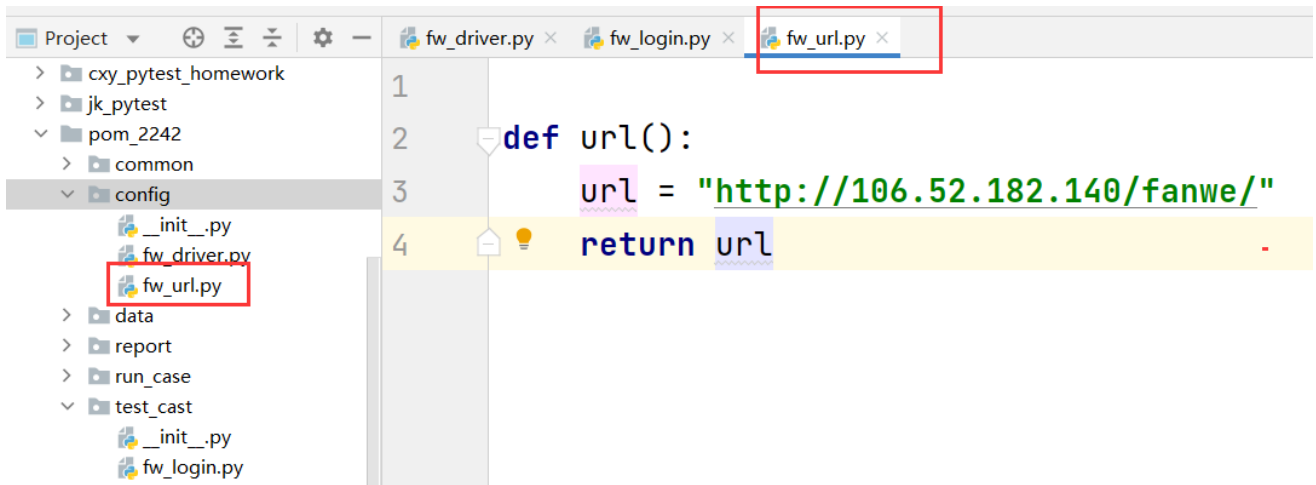
2、为了统一管理每个测试用例中的浏览器驱动, 可以把驱动封装成方法, 放在config下面



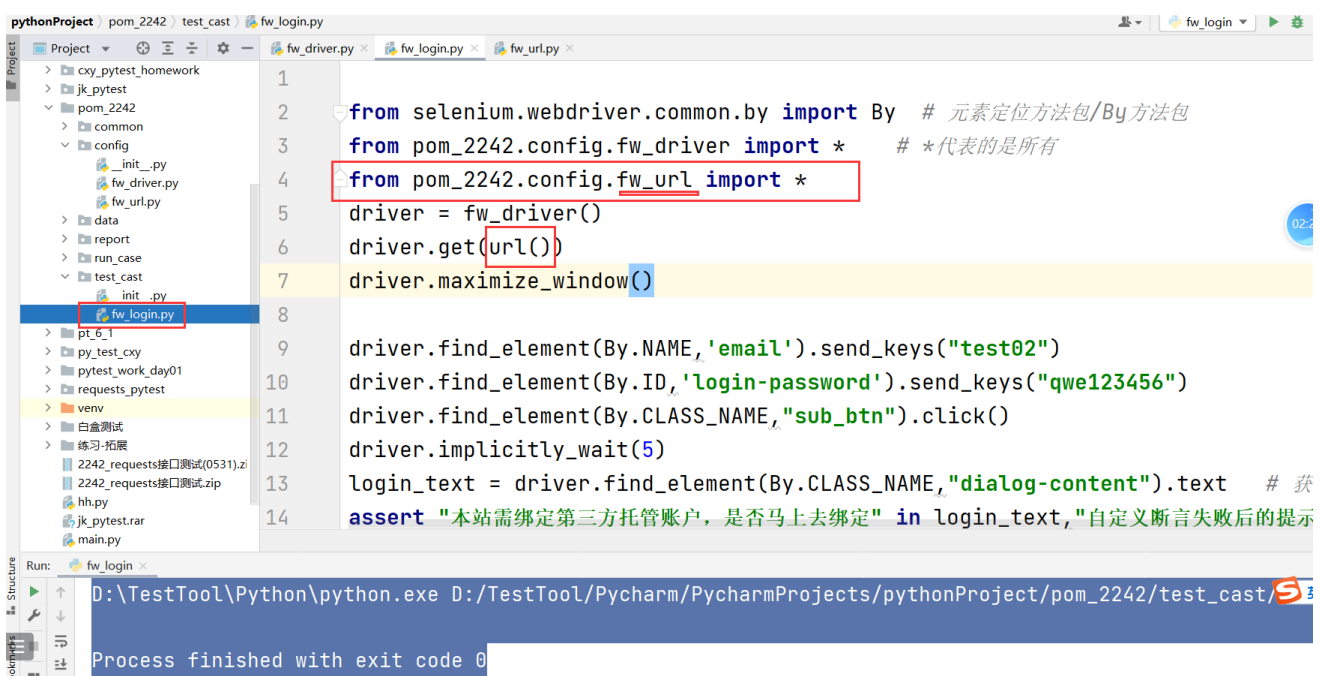
调用的是时候, 需要先导包



3、封装方维URL, 放在config下



导入后调用



4、需要把定位元素方法的操作封装成函数

定位元素方法, 发现缺少driver驱动, 以及By方法包, 都需要进行导入

完成后再进行封装



```
1 from pom_2242.config.fw_driver import *
2 driver = fw_driver()
3 from selenium.webdriver.common.by import By
4
5 def name():
6     return driver.find_element(By.NAME, 'email') # 定位账号
7
8 def pwd():
9     return driver.find_element(By.ID, 'login-password')
10
11 def button():
12     return driver.find_element(By.CLASS_NAME, 'sub_btn')
13
14 def login_text():
15     return driver.find_element(By.CLASS_NAME, 'dialog-content').text
```

Annotations in the image:

- Line 3: `from selenium.webdriver.common.by import By` is highlighted in blue. A red note says "缺少driver跟By, 需要导入" (Missing driver and By, need to import).
- Line 5: `def name():` is highlighted in blue. A red note says "再进行封装" (Further encapsulation).
- Line 6: `return driver.find_element(By.NAME, 'email')` is highlighted in green. A red note says "# 定位账号" (Locate account).
- Line 8: `return driver.find_element(By.ID, 'login-password')` is highlighted in green.
- Line 10: `return driver.find_element(By.CLASS_NAME, 'sub_btn')` is highlighted in green.
- Line 12: `return driver.find_element(By.CLASS_NAME, 'dialog-content').text` is highlighted in green.

封装好后进行调用

调用的时候, 发现有使用到两个浏览器驱动, 那么需要把原来fw_login下面多余的驱动给删除



```
1 from selenium.webdriver.common.by import By # 元素定位方法包/By方法包
2 from pom_2242.config.fw_driver import * # *代表的是所有
3 from pom_2242.config.fw_url import *
4 from pom_2242.common.login_common import *
5 driver = fw_driver()
6 driver.get(url())
7 driver.maximize_window()
8
9 name().send_keys("test02")
10 pwd().send_keys("qwe123456")
11 button().click()
12 driver.implicitly_wait(5)
13 login_text = login_text() # 获取文本内容
14 assert "本站需绑定第三方托管账户, 是否马上去绑定" in login_text, "自定义断言失败后的提示"
15 driver.quit()
```

Annotations in the image:

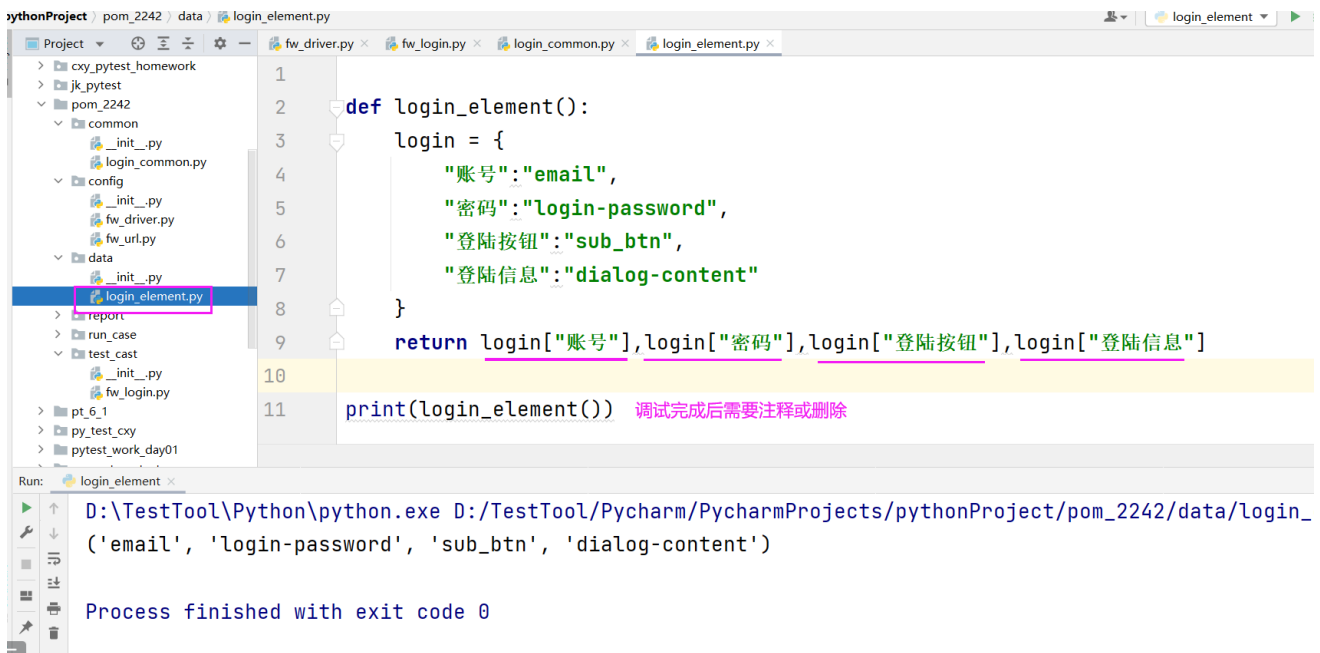
- Line 1: `from selenium.webdriver.common.by import By` is highlighted in red. A red note says "# 元素定位方法包/By方法包".
- Line 2: `from pom_2242.config.fw_driver import *` is highlighted in red. A red note says "# *代表的是所有".
- Line 3: `from pom_2242.config.fw_url import *` is highlighted in red.
- Line 4: `from pom_2242.common.login_common import *` is highlighted in red. A red note says "导入包含了一个浏览器驱动".
- Line 5: `driver = fw_driver()` is highlighted in red. A red note says "X".
- Line 6: `driver.get(url())` is highlighted in red.
- Line 7: `driver.maximize_window()` is highlighted in red.
- Line 9: `name().send_keys("test02")` is highlighted in green.
- Line 10: `pwd().send_keys("qwe123456")` is highlighted in green.
- Line 11: `button().click()` is highlighted in green.
- Line 12: `driver.implicitly_wait(5)` is highlighted in green.
- Line 13: `login_text = login_text()` is highlighted in green. A red note says "# 获取文本内容".
- Line 14: `assert "本站需绑定第三方托管账户, 是否马上去绑定" in login_text, "自定义断言失败后的提示"` is highlighted in green.
- Line 15: `driver.quit()` is highlighted in green.

5、实现数据分离：

定位跟操作分离

定位方法跟元素属性分离

把元素属性再次进行封装：通过字典格式中键值对中的key值



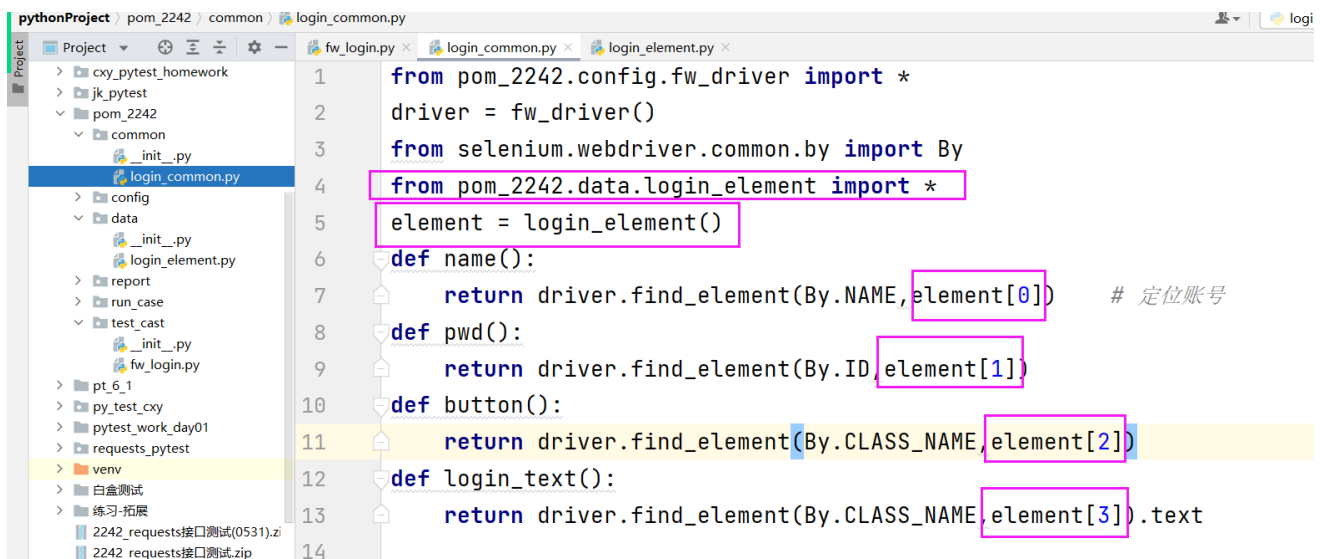
```
1 def login_element():
2     login = {
3         "账号": "email",
4         "密码": "login-password",
5         "登陆按钮": "sub_btn",
6         "登陆信息": "dialog-content"
7     }
8     return login["账号"], login["密码"], login["登陆按钮"], login["登陆信息"]
9
10 print(login_element()) 调试完成后需要注释或删除
```

Run: login_element x

D:\TestTool\Python\python.exe D:/TestTool/Pycharm/PycharmProjects/pythonProject/pom_2242/data/login_ ('email', 'login-password', 'sub_btn', 'dialog-content')

Process finished with exit code 0

调用需要导入，并进行赋值，通过索引进行调用



```
1 from pom_2242.config.fw_driver import *
2 driver = fw_driver()
3 from selenium.webdriver.common.by import By
4 from pom_2242.data.login_element import *
5 element = login_element()
6 def name():
7     return driver.find_element(By.NAME, element[0]) # 定位账号
8 def pwd():
9     return driver.find_element(By.ID, element[1])
10 def button():
11     return driver.find_element(By.CLASS_NAME, element[2])
12 def login_text():
13     return driver.find_element(By.CLASS_NAME, element[3]).text
14
```

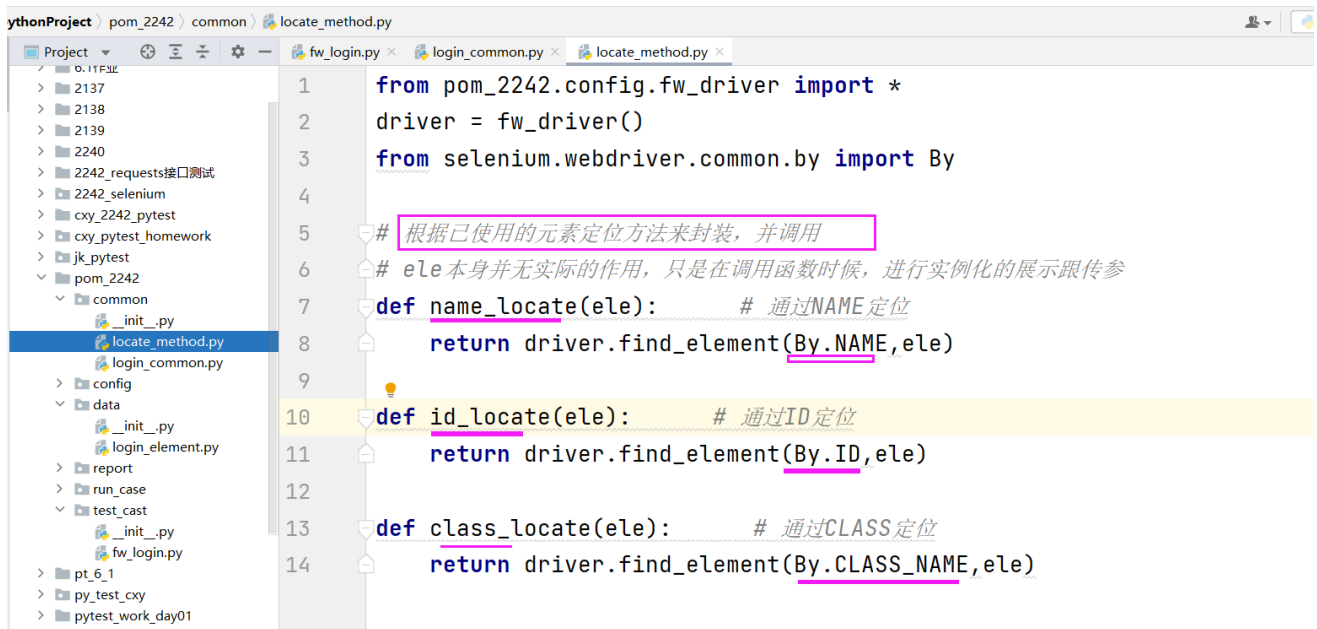
Run: login_common x

D:\TestTool\Python\python.exe D:/TestTool/Pycharm/PycharmProjects/pythonProject/pom_2242/common/login_common.py

Process finished with exit code 0

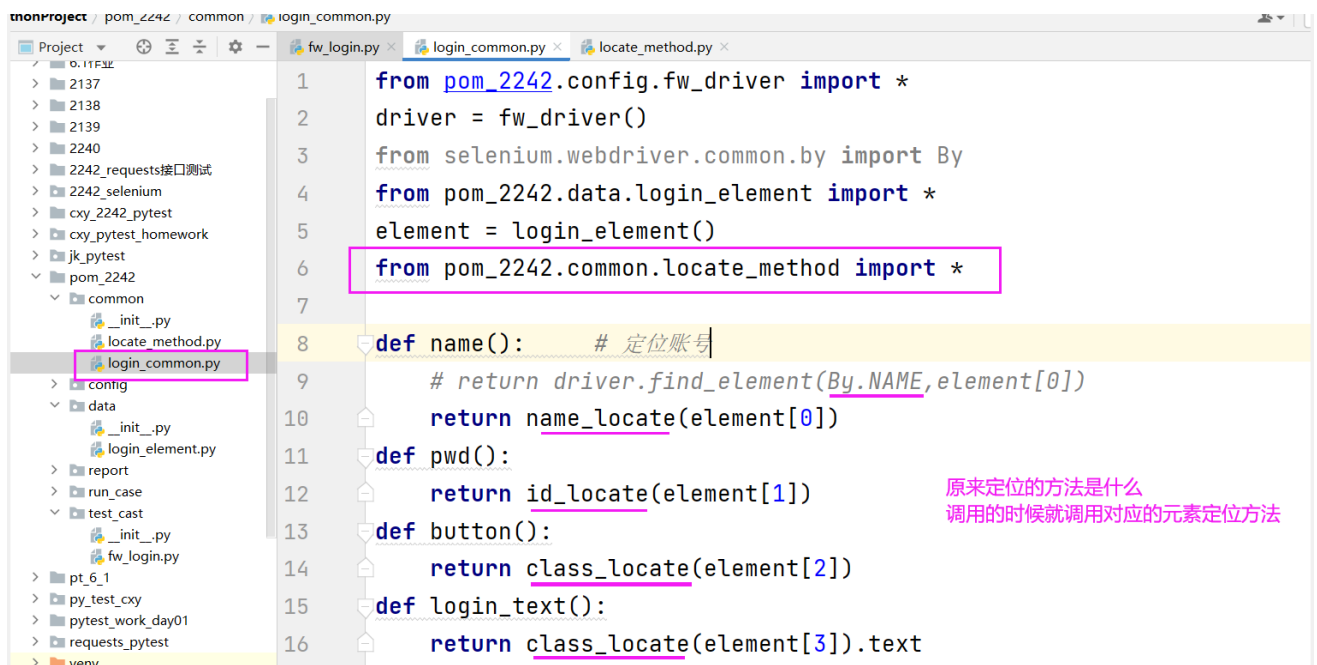
6、将元素定位方法（八大定位方法）进一步进行封装

使用了什么定位方法，就封装对应的定位方法



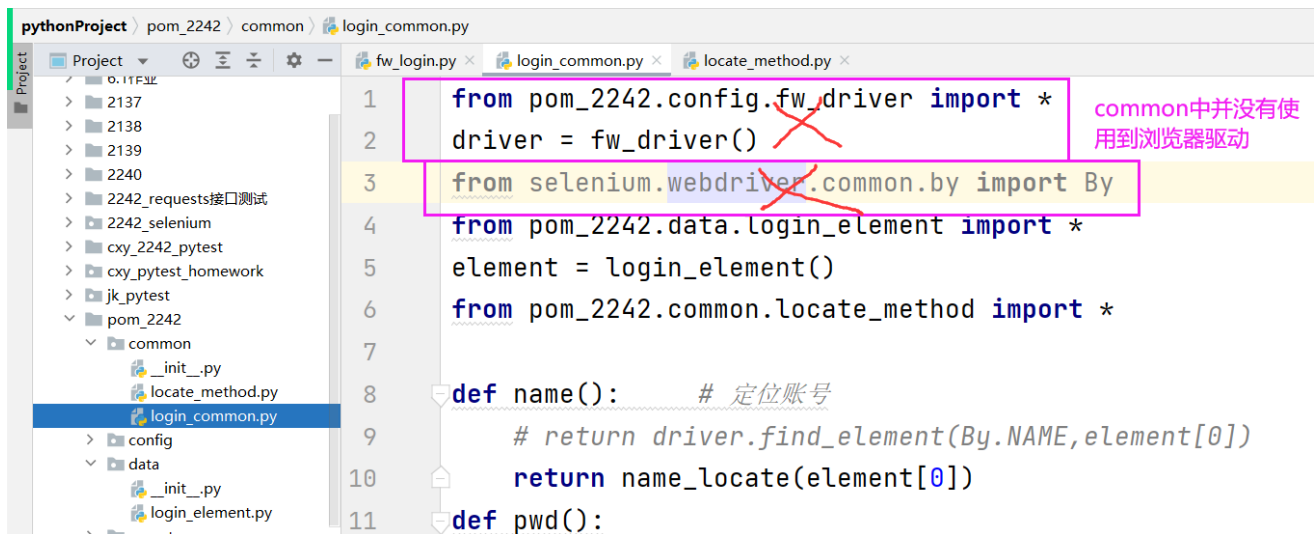
```
1 from pom_2242.config.fw_driver import *
2 driver = fw_driver()
3 from selenium.webdriver.common.by import By
4
5 # 根据已使用的元素定位方法来封装，并调用
6 # ele本身并无实际的作用，只是在调用函数时候，进行实例化的展示跟传参
7 def name_locate(ele): # 通过NAME定位
8     return driver.find_element(By.NAME, ele)
9
10 def id_locate(ele): # 通过ID定位
11     return driver.find_element(By.ID, ele)
12
13 def class_locate(ele): # 通过CLASS定位
14     return driver.find_element(By.CLASS_NAME, ele)
```

调用的时候，根据原来使用的元素定位方法来选择调用



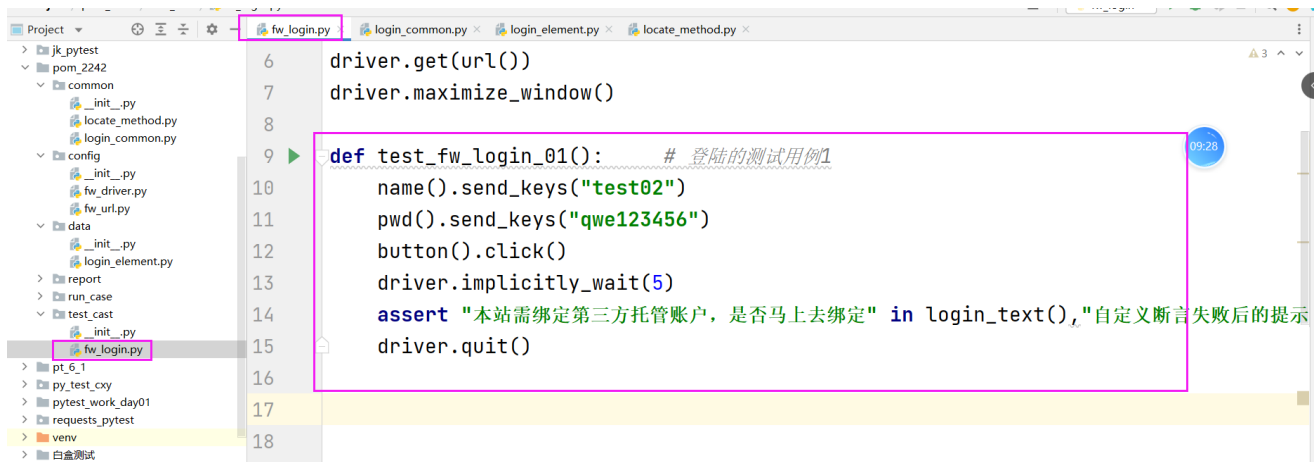
```
1 from pom_2242.config.fw_driver import *
2 driver = fw_driver()
3 from selenium.webdriver.common.by import By
4 from pom_2242.data.login_element import *
5 element = login_element()
6 from pom_2242.common.locate_method import *
7
8 def name(): # 定位账号
9     # return driver.find_element(By.NAME, element[0])
10    return name_locate(element[0])
11
12 def pwd():
13    return id_locate(element[1])
14
15 def button():
16    return class_locate(element[2])
17
18 def login_text():
19    return class_locate(element[3]).text
```

再次发现在login_common下没有使用到浏览器驱动，需要将其删除



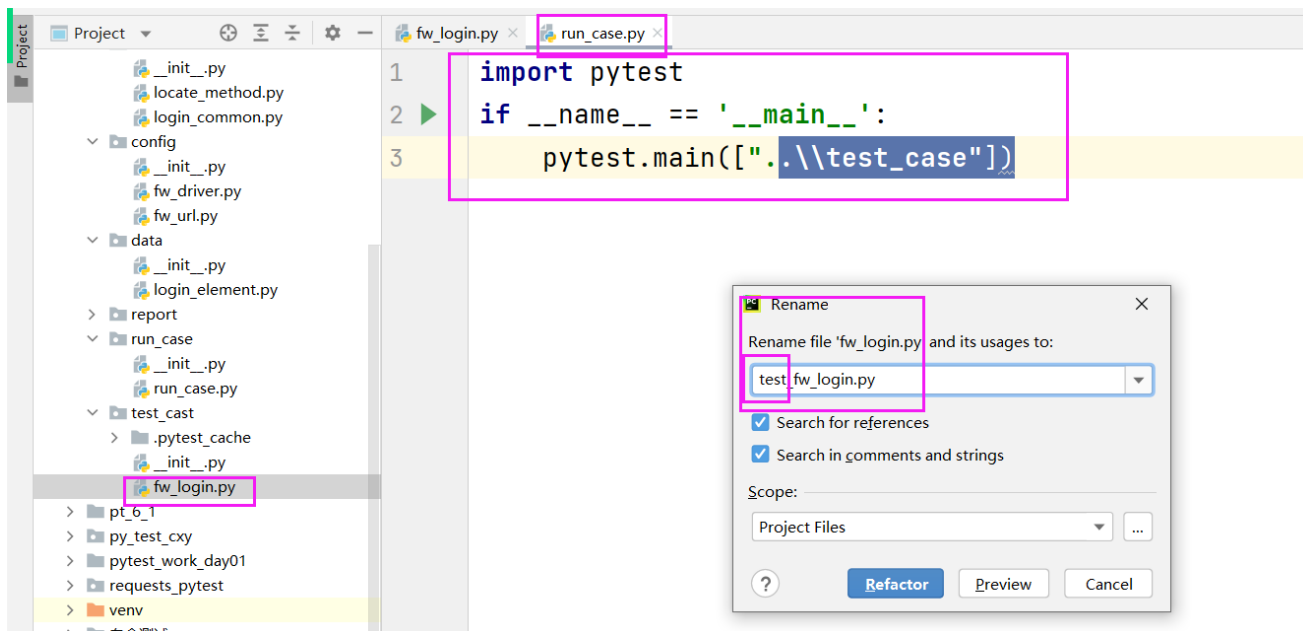
7、将测试用例进行封装

test_case下的方维登陆用例，务必记住，需要test开头

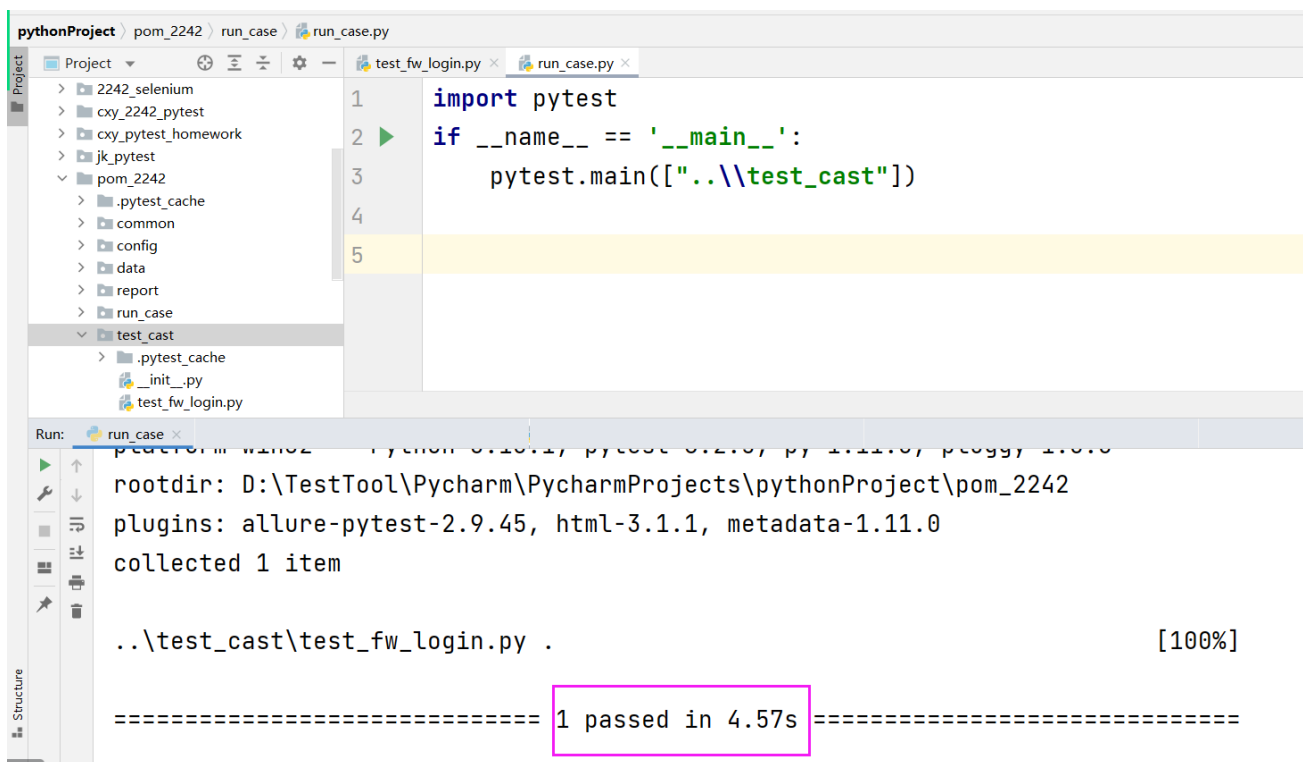


8、在run_case下创建执行测试用例的主程序

需要注意下：测试用例脚本文件需要以test开头



执行通过:



9、生成allure报告

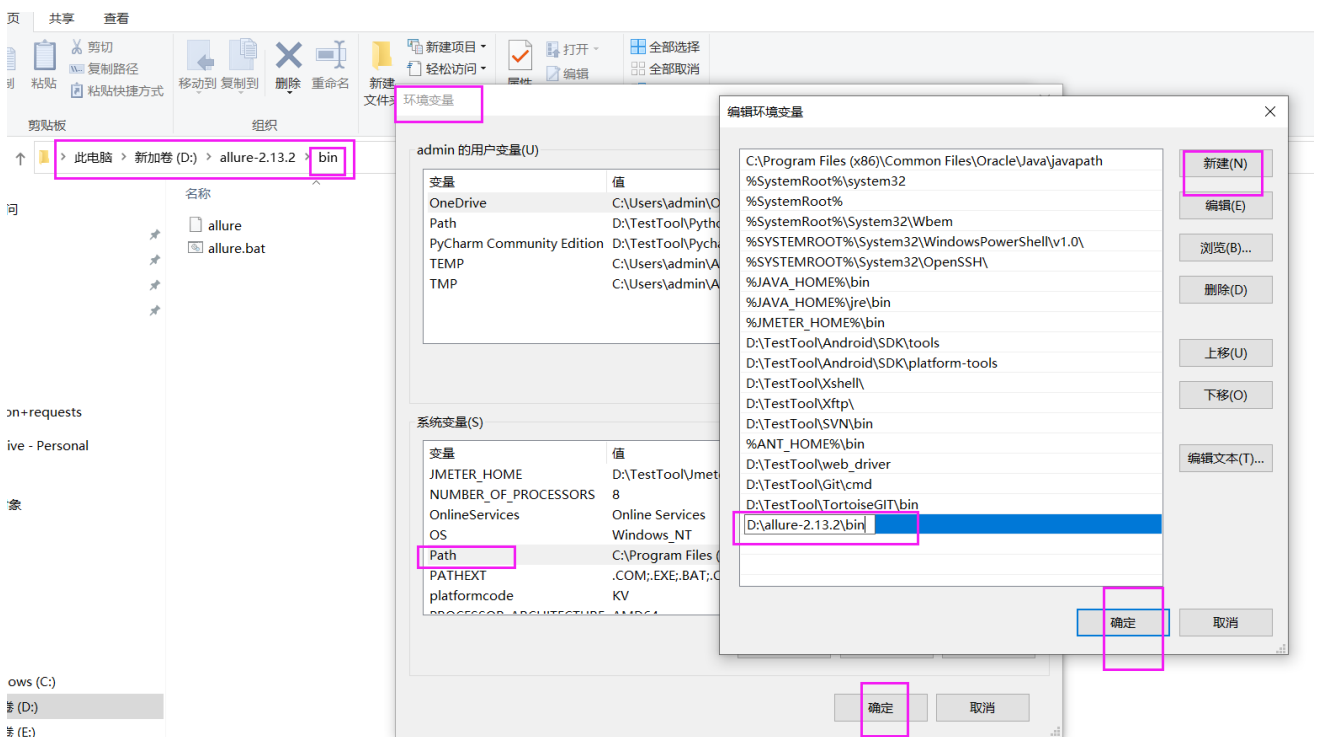


10、发现生成报告是JSON格式的，需要使用allure进行转换

1、先搭建allure环境

将allure_2.13.2.zip包解压到非C盘非中文路径下

将其中的bin目录添加到path环境变量中



验证，dos命令窗口验证：allure，没有报非内部命令即成功

```
C:\windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19044.1706]
(c) Microsoft Corporation。保留所有权利。

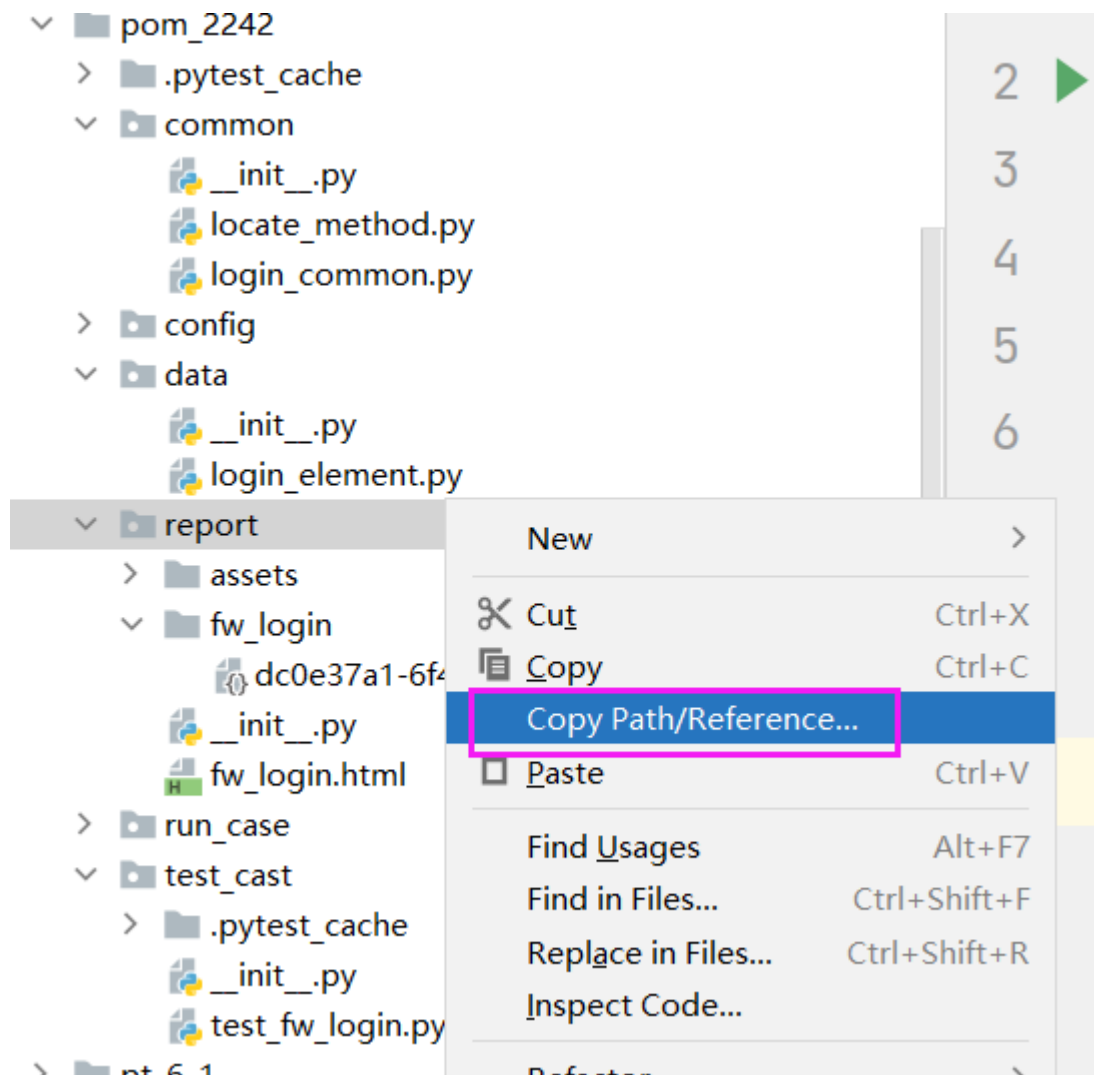
C:\Users\admin>allure
Usage: allure [options] [command] [command options]
Options:
  --help
    Print commandline help.
  -q, --quiet
    Switch on the quiet mode.
    Default: false
  -v, --verbose
    Switch on the verbose mode.
    Default: false
```

2、转换allure报告

在dos命令下进行转换：

①、需要先确认JSON报告格式的路径，**report**的路径，并切换到该路径下

举例：D:\TestTool\Pycharm\PycharmProjects\pythonProject\pom_2242\report



C:\windows\system32\cmd.exe

Microsoft Windows [版本 10.0.19044.1706]
(c) Microsoft Corporation。保留所有权利。

C:\Users\admin>D:

D:\>cd D:\TestTool\Pycharm\PycharmProjects\pythonProject\pom_2242\report

D:\TestTool\Pycharm\PycharmProjects\pythonProject\pom_2242\report>

②、进行转换：

allure generate JSON报告格式的路径 -o 新生成的报告路径 --clean

JSON报告格式的路径：.\fw_login

-o：表示指定文件路径

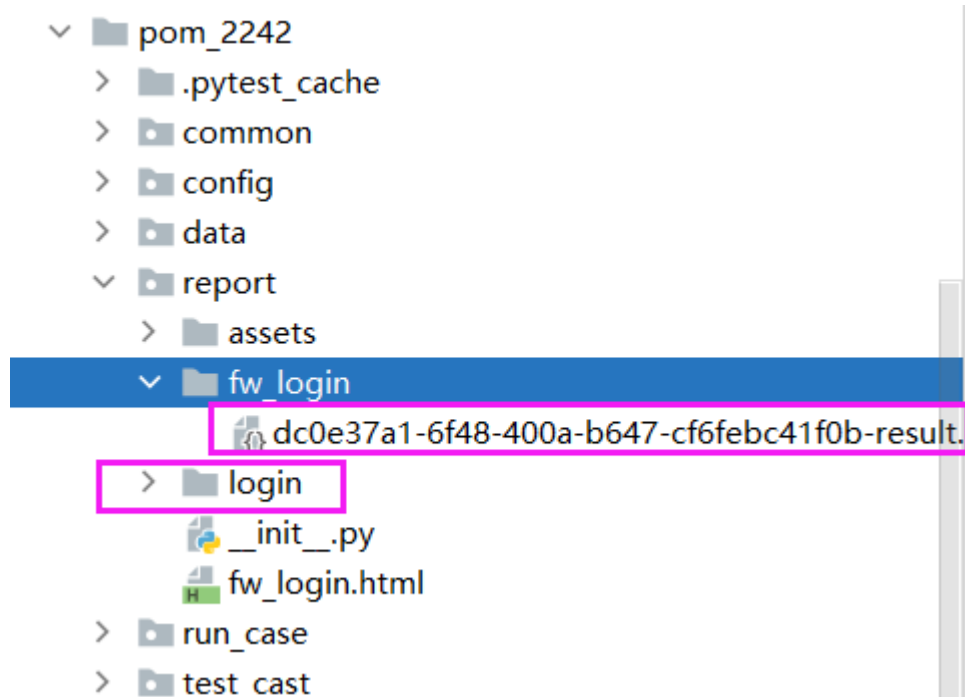
新生成的报告路径：.\login

--clear：如果新路径下存在报告，会进行清楚，没有的话就忽略

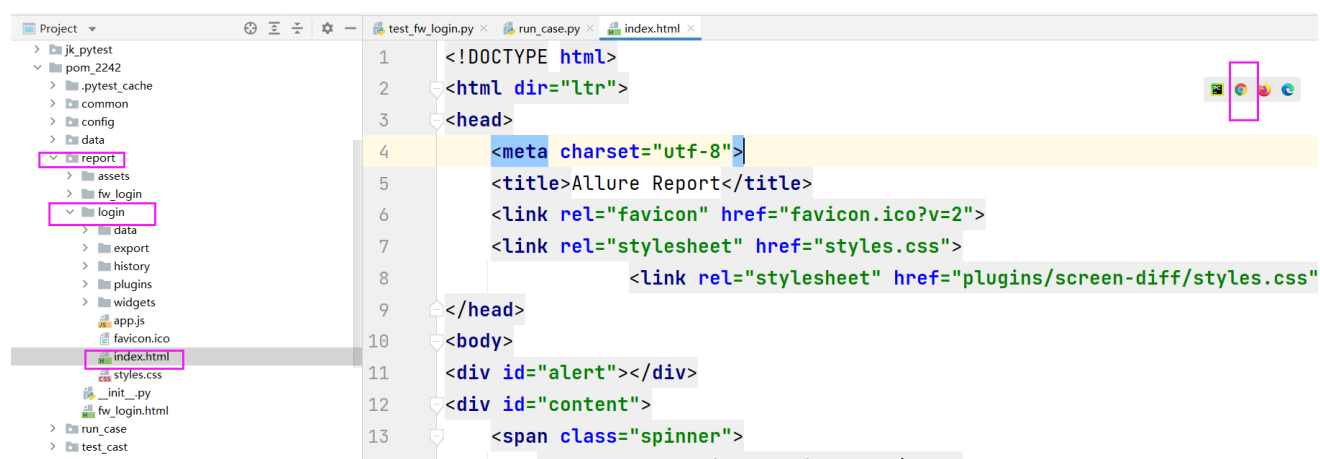
举例：allure generate .\fw_login -o .\login --clean

```
选择 C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19044.1706]
(c) Microsoft Corporation. 保留所有权利。

D:\TestTool\Pycharm\PycharmProjects\pythonProject\pom_2242\report>allure generate .\fw_login -o .\login --clear
--clear does not exists
Report successfully generated to .\login
D:\TestTool\Pycharm\PycharmProjects\pythonProject\pom_2242\report>_
```

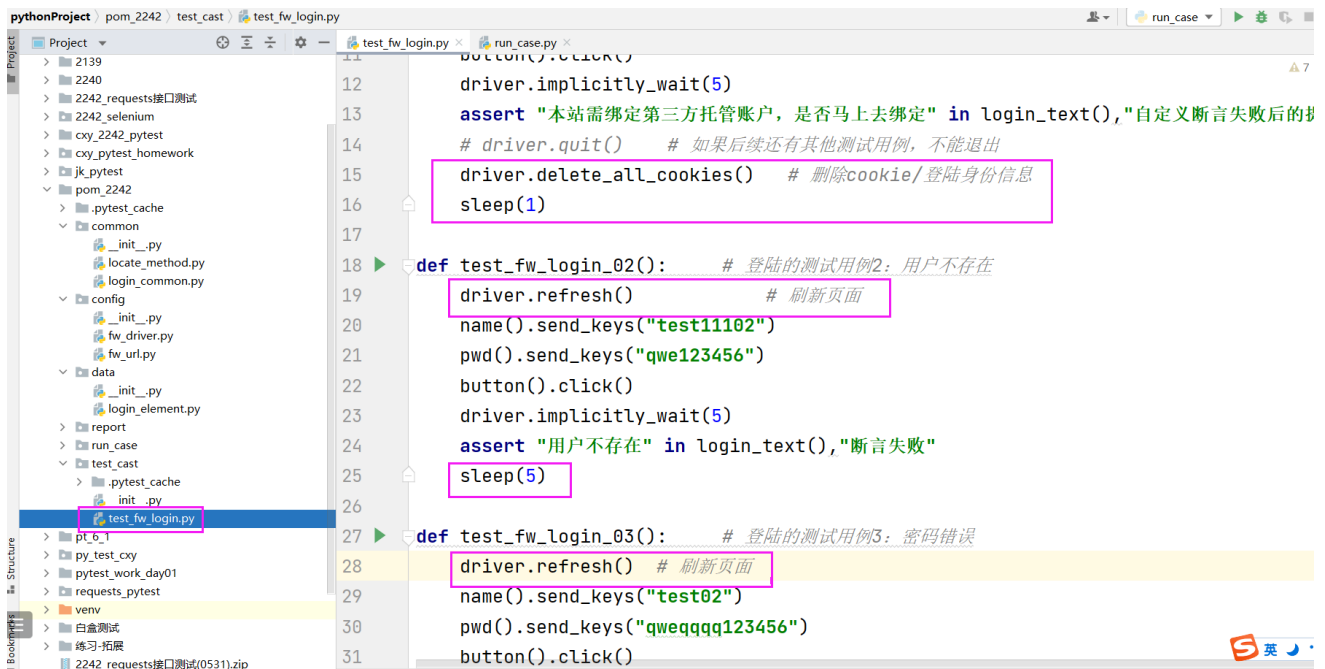


allure报告可以通过浏览器进行打开



11、补充测试用例

登陆成功后，如果想要执行登陆失败用例，可以使用删除身份信息(cookie)，并刷新，就会返回到登陆界面



五、自动化知识的拓展(接口自动化、UI自动化):

1、数据闭环

在测试的过程中, 因为测试操作, 参数了大量无效的数据, 这些数据会成为垃圾数据, 影响到服务器的性能、数据库的读写速度。

干扰测试效率以及测试的准确性。所以, 需要将无用的垃圾数据进行清理删除, 从而形成数据闭环

2、参数化

使用最少正确的数据来进行测试验证, 而这些数据的提供方式是各种各样的

没有固定格式: txt文本数据、Excel表格数据(csv/xls)、元组、列表、字典

少量: 最合适的数据, 而不是过多重复的数据

3、面试常问问题

①、接口自动化, 是使用什么框架实现的? 框架是怎么封装处理? 你在其中担任什么角色? 接口自动化用例写了多少条?

框架: pytest框架 (框架的作用: 框架可以通过固定格式去收集测试用例, 执行测试用例, 并可以生成测试报告)

先说担任角色:

(一) 只是负责编写执行测试用例, 不负责测试环境的搭建。可以直接说, 公司的框架是有专门的自动化测试小组负责搭建好的, 然后跟我们同步培训(培训会议、使用文档), 我们直接使用现成的框架。

可以清楚明白框架中每个模块的具体作用。大概划分了六个模块

common: 存放公共方法、数据操作

config: 配置文件, 存放配置、IP, 域名, 数据库的配置

data: 数据层, 用于测试数据

report: 生成测试报告存放的目录

run_case: 执行测试用例主程序的目录

test_case: 存放测试用例

在实际的接口自动化测试过程中, 我只负责补充修改有涉及新增、更新接口的接口测试用例

(二) 负责公司pytest框架的搭建跟维护, 需要详细说明下搭建的步骤, 以及每个方法的封装跟调用。

接口自动化测试 用例:

项目0~1: 无到有, 接口自动化测试, 用例时大量, (项目一年左右+)五六百以上, 将近一千都可以;

后续迭代过程(多个批次迭代), (项目时间半年)功能新增, 涉及到接口修改/新增, 负责的接口自动用例不会很多, 一般在一两百。

②、跑接口自动化测试用例花费的时间?

300左右的用例, 花费时间? $0.15 \times 300 = 45$ 秒, 回答面试官问题的时候, 不要说准确数据, 说大概就好:

可以说, 用例大概写了三百多条, 具体数据没有统计/关注, 只是大概记得, 执行时间不到一分钟(一分钟左右)。

③、UI自动化测试脚本运行花费多少时间?

使用什么框架实现的? 框架是怎么封装处理? 你在其中担任什么角色?

框架: pytest框架 (框架的作用: 框架可以通过固定格式去收集测试用例, 执行测试用例, 并可以生成测试报告)

先说担任角色:

(一) 只是负责编写执行测试用例, 不负责测试环境的搭建。可以直接说, 公司的框架是有专门的自动化测试小组负责搭建好的, 然后跟我们同步培训(培训会议、使用文档), 我们直接使用现成的框架。

可以清楚明白框架中每个模块的具体作用。大概划分了六个模块

common: 存放公共方法、数据操作

config: 配置文件, 存放配置、IP, 域名, 数据库的配置

data: 数据层, 用于测试数据

report: 生成测试报告存放的目录

run_case: 执行测试用例主程序的目录

test_case: 存放测试用例

在实际的接口自动化测试过程中, 我只负责补充修改有涉及新增、更新接口的接口测试用例

(二) 负责公司pytest框架的搭建跟维护，需要详细说明下搭建的步骤，以及每个方法的封装跟调用。

用例数：整个项目只实现了**核心的功能流程**UI自动化测试用例，用例数在150~300之间，自己说个大概数数量

举例：之前项目只涵盖了核心功能流程的**正向场景**的用例，用例数不到300/用例数大概不到200，具体没有统计过

时间：200X3.33=666秒，11分钟。

可以说，用例数不到200，执行时间大概十几二十分钟。

用例数在300左右，大概跑了半小时左右

(项目比较大，看起来比较复杂，可以在这个基础上增加20~50%)

项目周期长，可以说涵盖正向、反向场景的用例，用例数跟执行时间适当增加)

90%以上的公司，做自动化，只做接口自动化。不做UI自动化测试。

原因：UI自动化需要同时基于**后台服务接口**跟**前端UI界面元素**，其中之一发送了变化，脚本都需要调整改动。

④、自动化测试在什么阶段？

接口自动化测试、UI自动化测试，都是在版本批次迭代上线后进行编写，是为了方便后续的迭代，使用自动化测试脚本，对原来项目批次功能、接口的回归点检（代替人工回归验证）。

