Python UI 自动化框架

本框架支持功能：基于 selenium 和 pytest 的二次功能开发，拥有简易的 selenium 二次封装 api（元素基础操作，智能等待定位，下拉选定位，文本、元素属性获取，切换网页句柄、js 执行，滚动条移动等），并且自带打印日志，方便定位问题。脚本基于 pageobject 模式编写，用例和元素定位分离，易于维护，配合 pytest-rerunfailures 实现用例的失败重跑，提高脚本的稳定性。脚本执行完毕，使用 allure2 收集执行结果并生成测试报告。

技术栈：
1.语言：python3.x
2.自动化工具：selenium 3.141.0
3.单元测试框架：pytest 3.6.0
4.报告生成插件：pytest-allure-adaptor 1.7.10
5.失败重跑插件：pytest-rerunfailures 5.0

待开发：
1.邮件功能
2.数据驱动(参数化)
3.告罄功能
4.数据库断言

使用流程大致分为以下：

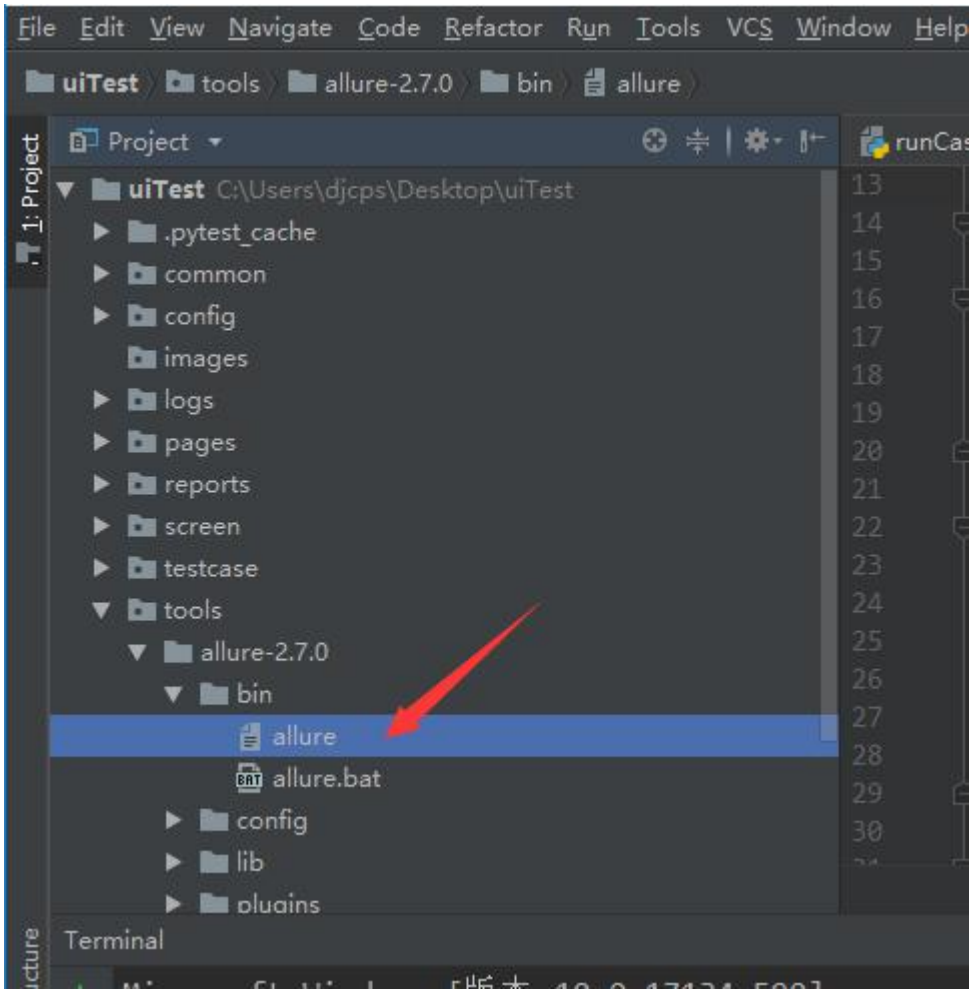
框架环境配置：

1. python 3.6 运行环境，pip 安装以下包(版本要对应，不然运行会报错)，命令如 pip install selenium == 3.141.0：

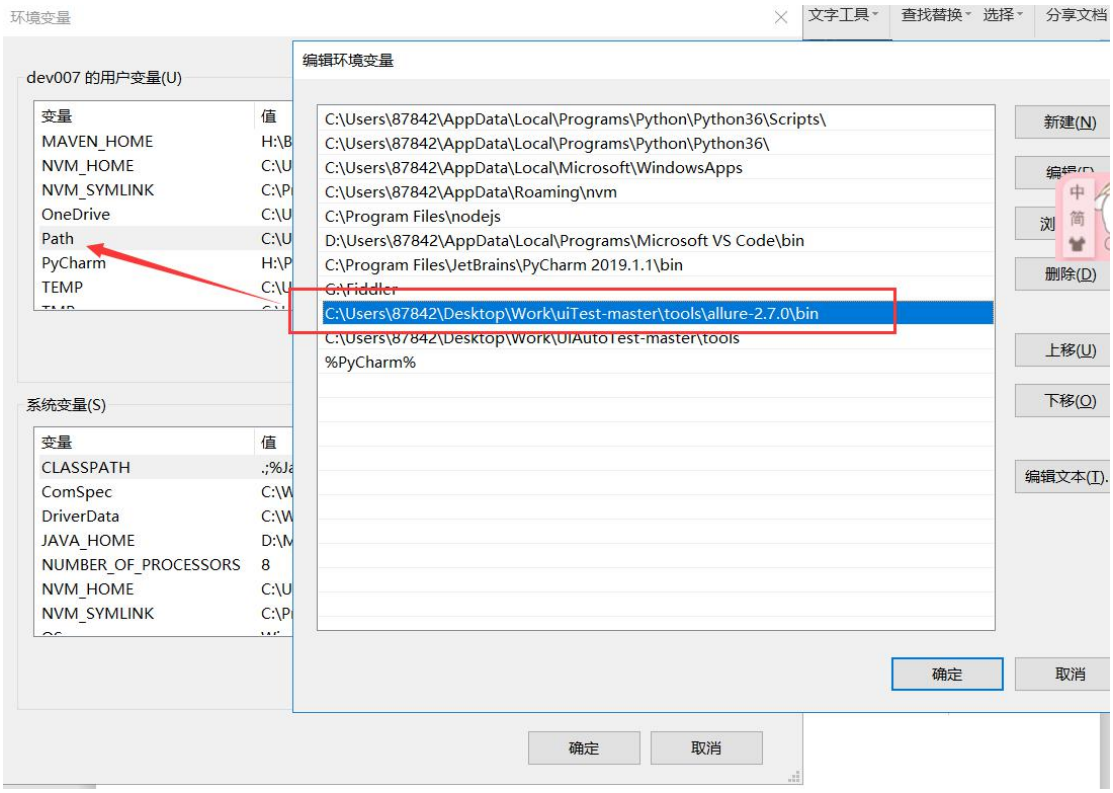
- selenium 3.141.0
- pytest 3.6.0
- pytest-allure-adaptor 1.7.10
- pytest-rerunfailures 5.0
- Jdk1.8 （allure 是 java 开发的，需要 java 环境支持）

- PyYAML 3.13

2. allure 环境变量配置



找到 tools/allure-2.7.0/bin/的绝对路径，并配置到系统环境变量中

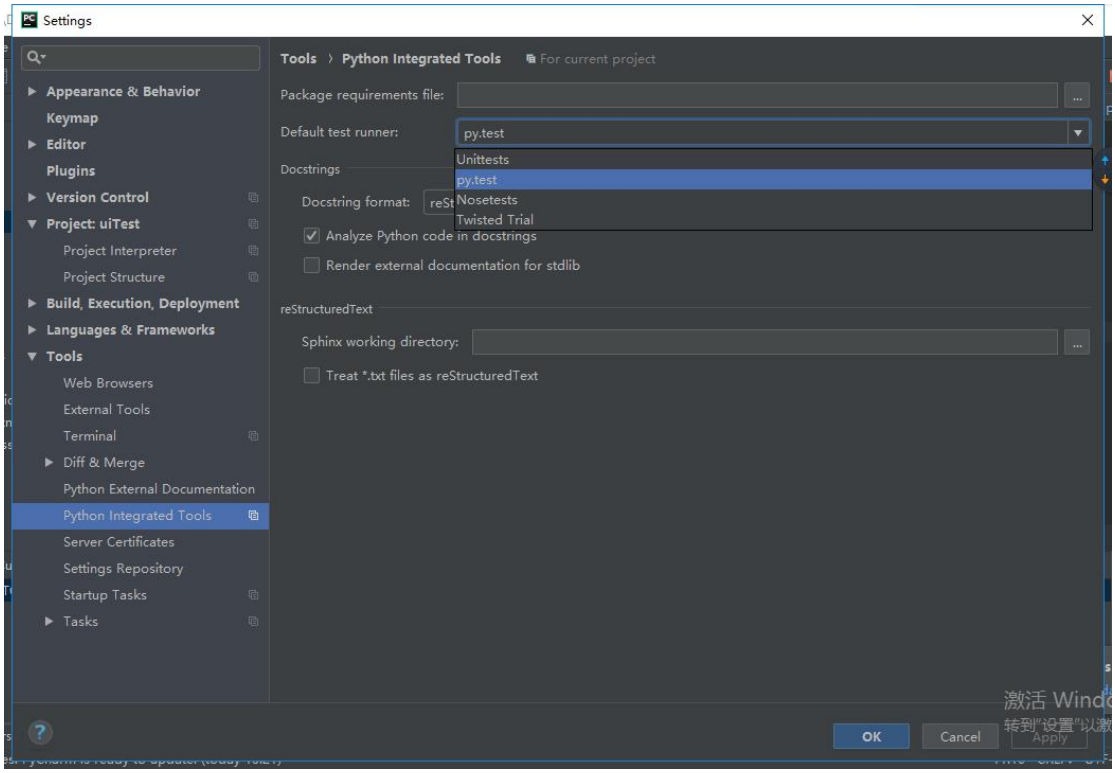


C:\Users\87842\Desktop\Work\uiTest-master\tools\allure-2.7.0\bin

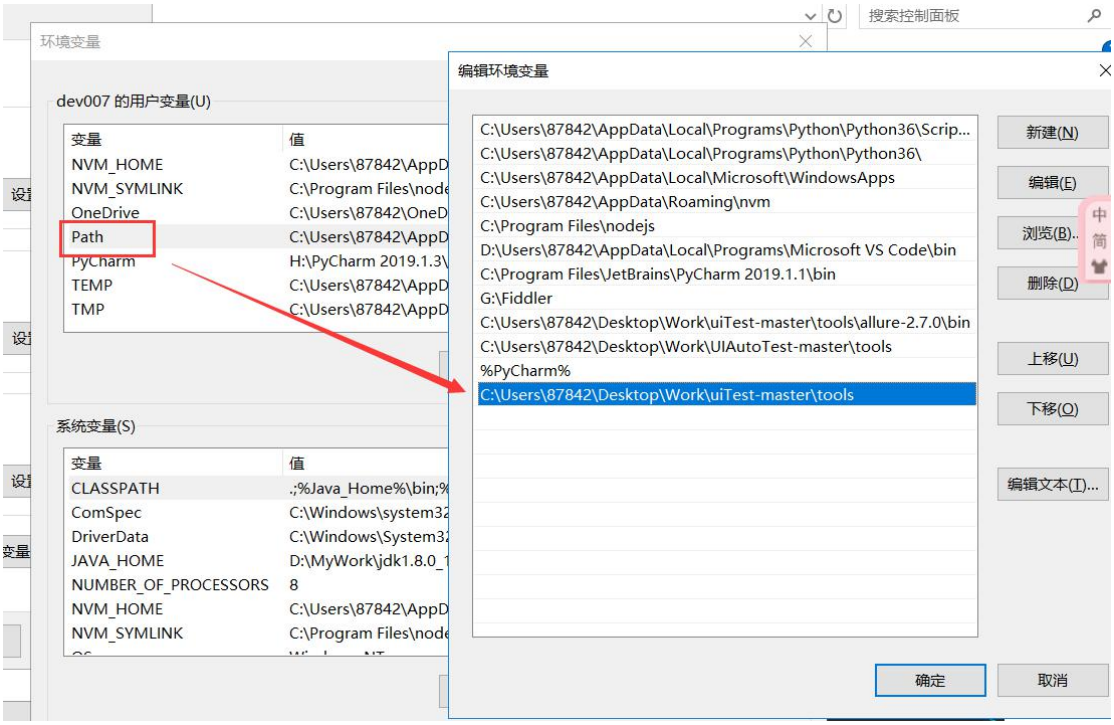
3. 设置 pycharm 默认 runner 为 pytest

找到以下路径进行设置：

Settings-Tools-Python Integrated Tools-Default test runner



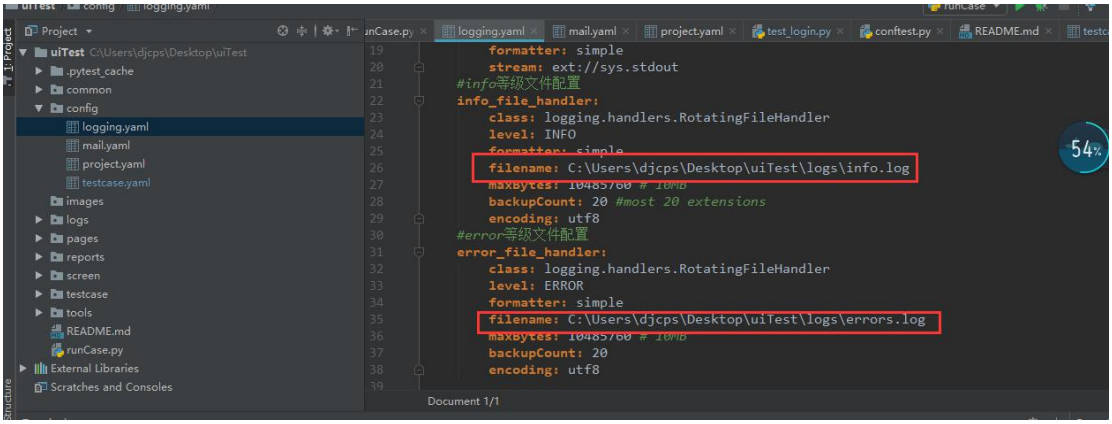
4. 设置 webdriver 配置文件的路径



5. C:\Users\87842\Desktop\Work\uiTest-master\tools

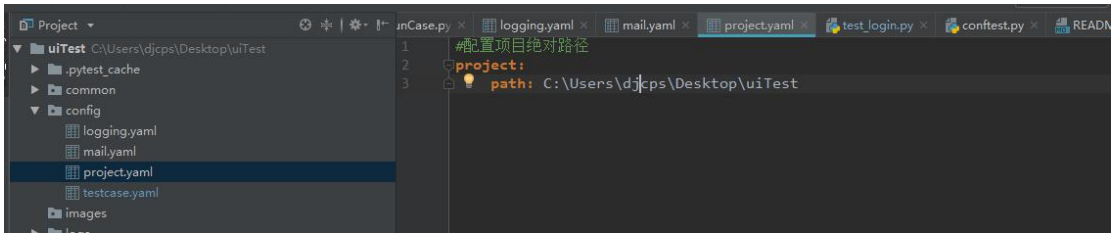
框架 config 配置文件：

logging.yaml：



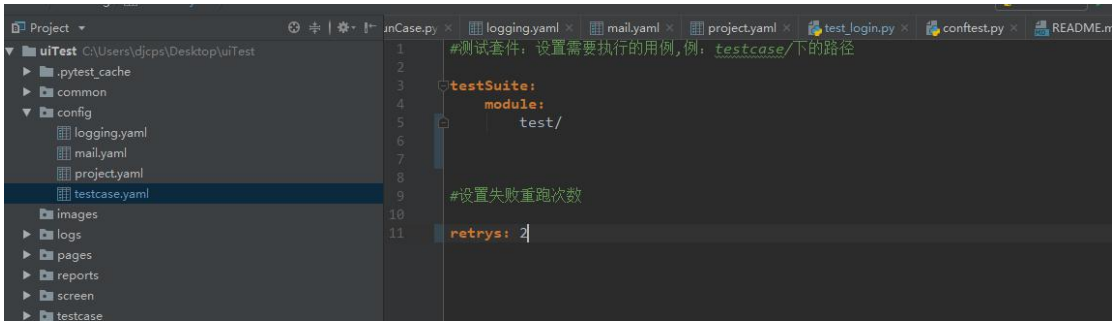
根据本地项目路径，配置对应的路径

project.yaml:



配置项目绝对路径

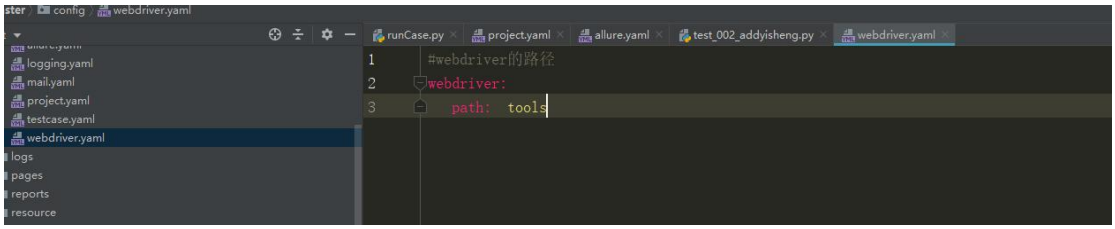
testcase.yaml：



添加测试套件配置：按顺序添加需要执行的用例相对路径，如 test/test_001.py

设置失败重跑次数：可以控制失败的重跑次数，如设置为 0 则不进行重跑

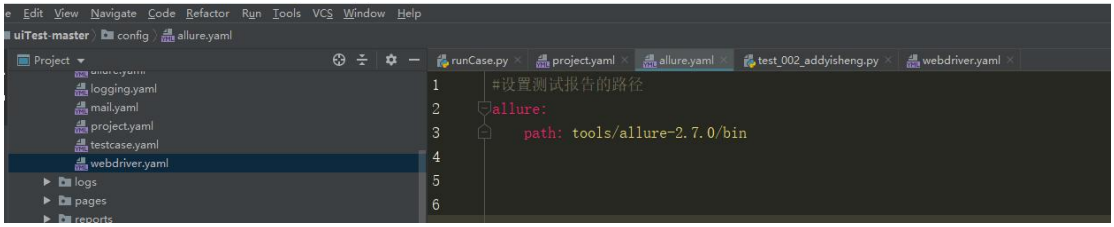
Webdriver.yaml：



读取 webdriver 的路径：目的为了后期获取 webdriver 的路径，再获取绝对路径，再根据 python 的 os.putenv 方法，添加到环境变量中。

```
webdri = dit4['webdriver']['path']      # tools/chromedriver.exe

abswebdri=os.path.abspath(webdri)

os.putenv('path',abswebdri)
```

Allure.yaml:



读取 allure 的路径：目的为了后期获取 allure 的路径，再获取绝对路径，再根据 python 的 os.putenv 方法，添加到环境变量中。

如何编写脚本：

前期知识学习：学习 pytest 测试框架（具体内容自行学习），框架二次封装的 api 熟练使用，allure 报告注解使用

框架基类 api：

基类 api	参数说明	api 说明

get(url)	指定 url	访问某个网址
quit()		退出浏览器
get_title()		获取网页 title
get_current_url()		获取当前 url
findElement(locator)	locator 为一个元组，形如（“id”,”login”）	元素定位，10s 后定位不到则抛出 timeout 异常
findElements(locator)	locator 为一个元组，形如（“class”,”btn”）	多个元素定位,10s 后定位不到则抛出 timeout 异常
sendKeys(locator,text=””)	locator 为元素信息元组，text 为输入文本，默认为空	对 input 元素进行文本输入
click(locator)	locator 为元素信息元组	点击元素
clear(locator)	locator 为元素信息元组	清除元素内容
isSelected(locator)	locator 为元素信息元组	判断元素是否被选中，返回 boolean 值
isElementExist(locator)	locator 为元素信息元组	判断元素是否存在，返回 boolean 值
is_title(_title=“”)	_title 为标题文本	判断标题是否为，返回 boolean 值
is_title_contains(_title=“”)	_title 为标题文本	判断标题是否包含,返回 boolean 值
is_text_in_element(locator,_text=“”)	locator 为元素信息元组,_text 为文本	判断元素文本是否是，返回 boolean 值
Is_value_in_element(locator,_value=“”)	locator 为元素信息元组,_value 为元素值	判断元素 value 是否为，针对 input 元素，返回 boolean 值
is_alert(timeout=3)	timeout 为超时时间，默认为 3s	判断 alert,存在返回 alert 实例，不存在则返回 false
get_text(locator)	locator 为元素信息元组	获取元素文本
get_attribute(locator,name)	locator 为元素信息元组，name 为属性名称	获取元素属性
js_focus_element(locator)	locator 为元素信息元组	聚焦元素
Js_scroll_top()		调用 js 滚动到顶部
Js_scroll_end(x=0)	x 为滚动高度，默认为 0	调用 js 滚动到底部
select_by_index(locator,iindex=0)	locator 为元素信息元组，index 为下拉选下标，默认为 0	根据下标查找下拉选
select_by_value(locator,value)	locator 为元素信息元组，value 为下拉选的 value 值	根据 value 查找下拉选
select_by_text(locator,text)	locator 为元素信息元组，text 为下拉选的文本值	根据文本查找下拉选

switch_iframe(id_index_locator)	id_index_locator 为传入的 id，Iframe 下标，或 iframe 信息元组	根据 Id、下标、信息元组切换 iframe
switch_handle(window_name)	window_name 为窗口名	根据窗口名切换 handle
switch_alert()		切换到 alert
move_to_element(locator)	locator 为元素信息元组	悬停在元素
get_screen(file_name)	file_name 为截图名称	生成截图

Allure2 定制报告：

要使用 Allure 定制报告，需要先进行导入

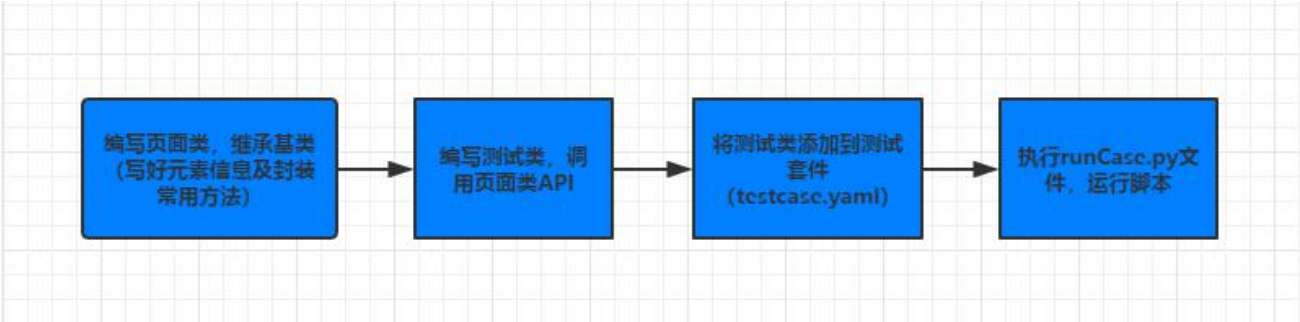
```
from allure import MASTER_HELPER as helper
```

利用以下 allure 注解进行报告定制

```
from allure import MASTER_HELPER as helper

@helper.severity("critical")                # 优先级，包含 blocker, critical, normal, minor, trivial 几个不同的等级
@helper.feature("测试模块_demo1")            # 功能块，feature 功能分块时比 story 大,即同时存在 feature 和 story 时,feature 为父节点
@helper.story("测试模块_demo2")              # 功能块，具有相同 feature 或 story 的用例将规整到相同模块下,执行时可用于筛选
@helper.issue("BUG 号：123")                 # 问题表识，关联标识已有的问题，可为一个 url 链接地址
@helper.testcase("用例名：测试字符串相等")    # 用例标识，关联标识用例，可为一个 url 链接地址
@helper.step("步骤")                        #操作步骤，用于标明脚本进行到哪一步
@helper.environment(**env_dict)             #传入环境变量,显示在报告中
@helper.description("描述")                 #为测试用例添加描述
@helper.attach()                            #为测试用例追加文字，图片，html 等
```

具体学习可参考博客：
https://testerhome.com/topics/15649?locale=zh-TW

脚本编写执行流程如下：



页面类（以团购为例）：

```
#coding=utf-8
'''封装团购登陆页面类'''
from common import basePage
from allure import MASTER_HELPER as helper
from selenium import webdriver

'''基于 PageObject 模式,定义页面类，继承基类'''
class LoginPage(basePage.BasePage):


    '''相关元素'''
    url="https://www.djcps.com/login.html"
    username=("id","username")
    password=("id","password")
    login_button=("id","login-button")

    '''封装登陆方法'''
    @helper.step("登陆团购系统")
    def login(self,username,password):
        '''访问网址，输入账号密码，点击登录按钮'''
        self.get(url=self.url)
        self.sendKeys(self.username,text=username)
        self.sendKeys(self.password,text=password)
        self.click(self.login_button)

'''测试'''
if __name__ == '__main__':
    driver=webdriver.Chrome()
    page=LoginPage(driver)
    page.login("17826826147","zyk123456")
```

conftest.py 文件(全局函数配置)

```
#coding=utf-8
import pytest
from pages.groupbuy import loginPage
```

```
from selenium import webdriver
from allure import MASTER_HELPER
import time

'''登录函数，供全局使用'''
@pytest.fixture(scope="module")
def login():
    driver=webdriver.Chrome()
    driver.maximize_window()
    page=loginPage.LoginPage(driver)
    page.login("17826826147","zyk123456")
    return driver
```

测试类

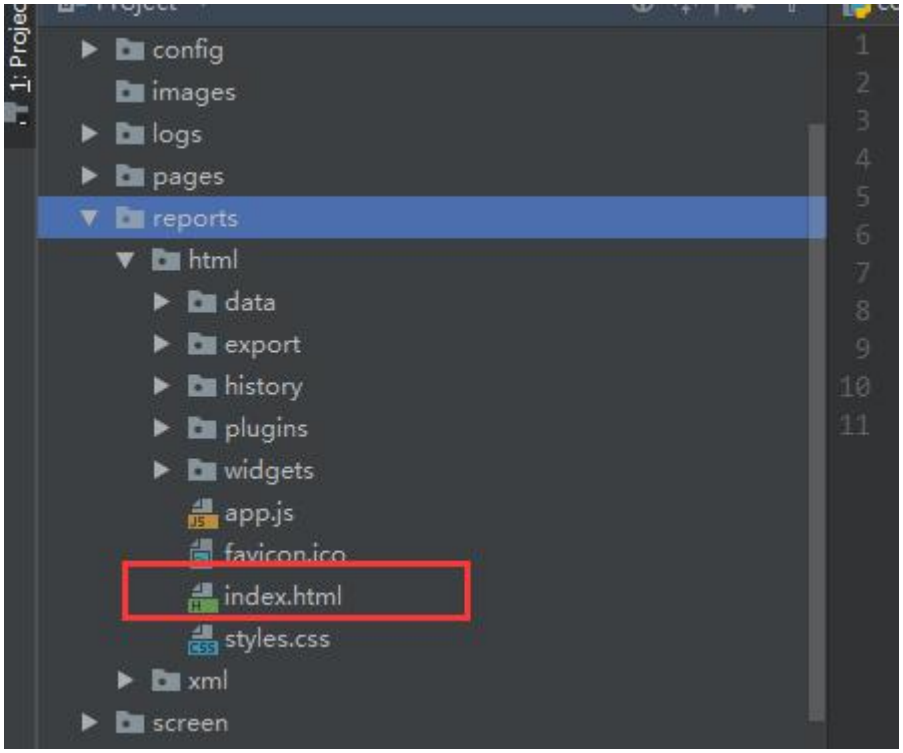
```
#coding=utf-8
import pytest
from pages.groupbuy import indexPage
from allure import MASTER_HELPER as helper

'''定义测试类'''
@helper.feature("主页面测试用例集")
class TestIndex():


    '''测试用例：搜索，需将fixture传入'''
    @helper.testcase("用例名：搜索东富1号")
    @helper.step("搜索东富1号")
    def test_search(self,login):
        self.page = indexPage.IndexPage(login)
        self.page.search_goods("东富1号")


    @helper.step("用例结束后，关闭浏览器")
    def teardown(self):
        self.page.quit()

if __name__ == '__main__':
    pytest.main(['-s','test_index.py'])
```

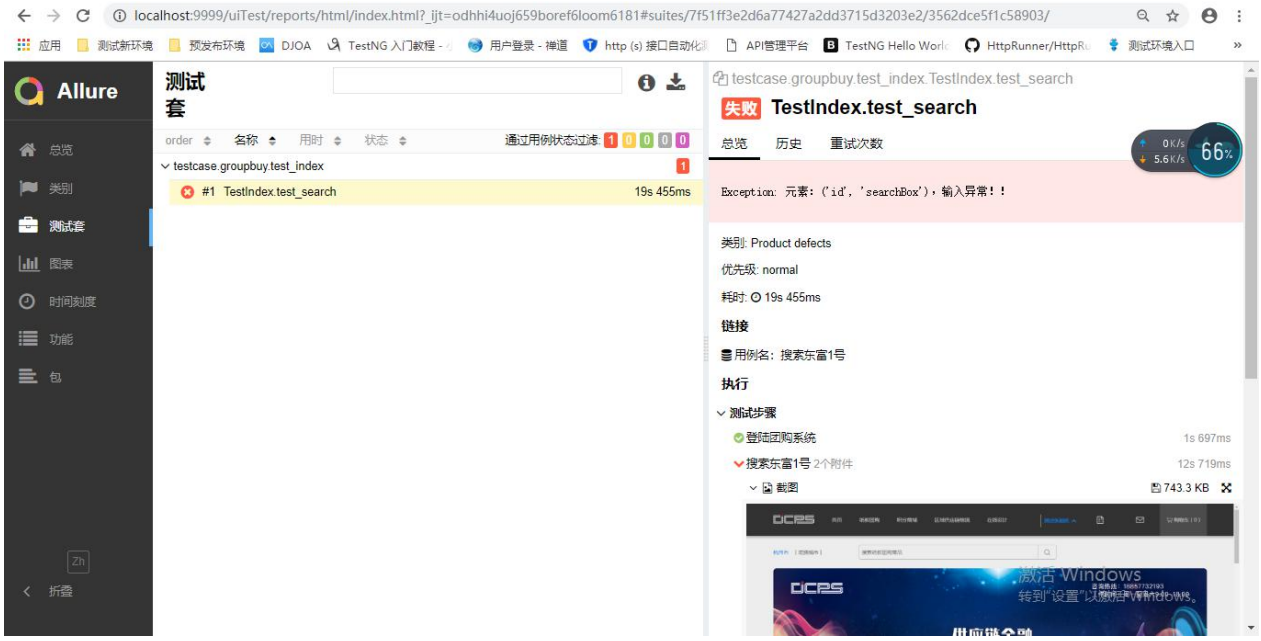
添加脚本到测试套件，设置重跑次数

```
#测试套件：设置需要执行的用例,例：testcase/下的路径

testSuite:
    module:
```

```
        groupbuy/test_index.py


#设置失败重跑次数

retrys: 0
```

运行 runCase.py，等待脚本执行完毕，查看报告





脚本编写注意事项：

1. 元素编写尽量见名知意，以降低维护成本
2. 脚本中添加必要注释
3. 脚本执行的关键节点需添加断言
4. 脚本文件名称以 test_开头，测试类以 Test 开头
5. 脚本和脚本之间不要有关联，保持低耦合度
6. 报告定制功能层级及描述等要清晰，以便于定位问题