JFZ 金斧子
WWW.JFZ.COM

自动化测试框架(数据驱动)

讲师 : 芳姐 | 时间 : 2018-8-9

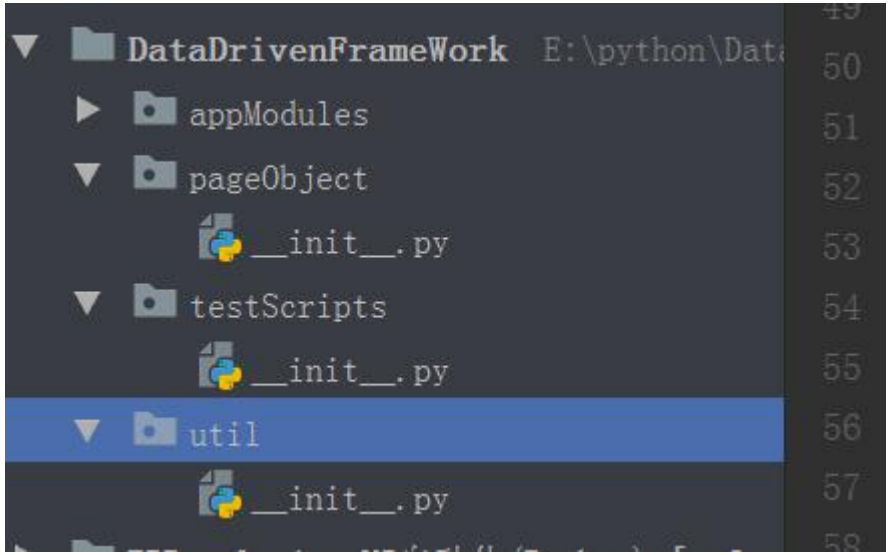
什么是数据驱动框架

使用数据数组、测试数据文件或者数据库等方式作为测试过程输入的自动化测试框架，此框架可以将所有测试数据在自动化测试执行的过程中进行自动加载，动态判断测试结果是否符合预期，并自动输出测试报告。

此框架一般用于在一个测试流程中使用多组不同的测试数据，以此来验证被测试系统是否能够正常工作。

1、新建工程文件

（1）在pycharm工具中，新建一个名叫DataDrivenFrameWork的python工程
（2）在工程中新建4个python package，分别命名为：
appModules，用于实现可复用的业务逻辑封装方法
pageObject，用于实现被测试对象的页面对象
testScripts，用于实现具体的测试脚本逻辑
util，用于实现测试过程中调用的工具类方法，例如读取配置文件，MapObject，页面元素的操作方法，解析excel文件等



2、util包中新建一个名叫ObjectMap.py的Python文件

用于实现定位页面元素的公共方法

```
from selenium.webdriver.support.wait import WebDriverWait
def getElement(driver,locateType,locatorExpression):
    try:
        element = WebDriverWait(driver, 30).until(lambda x: x.find_element(by=locateType,
value=locatorExpression))
        return element
    except Exception as e:
        raise e
# 获取多个相同页面元素对象，以list返回
def getElements(driver,locateType,locatorExpression):
    try:
        elements = WebDriverWait(driver, 30).until(lambda x: x.find_elements(by=locateType,
value=locatorExpression))
        return elements
    except Exception as e:
        raise e
```

3、在pageObjects包中新建LoginPage.py，用于封装登录页面的元素

```
from util.ObjectMap import *


class LoginPage(object):

    def __init__(self, driver):
        self.driver = driver

    def userNameObj(self):
        try:
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, "xpath", '//*[@name="username"]')
            return elementObj
        except Exception as e:
            raise e
```

```
# 密码
def passwordObj(self):
        try:
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, "xpath", '//input[@name = "password"]')
            return elementObj
        except Exception as e:
            raise e

    def loginButton(self):
        try:
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, "xpath", "//*[@class='login-btn']")
            return elementObj
        except Exception as e:
            raise e
```

4、在testScripts包中新建一个名叫TestErpApprovalflow.py文件

```
/*用于编写具体的测试操作代码*/
from selenium import webdriver
from pageObject.LoginPage import LoginPage
import time
def testErpLogin():
    try:
        # 启动谷歌浏览器
        driver = webdriver.Chrome(executable_path="E:\\Python36\\chromedriver")
        # 访问erp首页
        driver.get("https://qa1-erp.jfz.com")
        driver.implicitly_wait(30)

        driver.maximize_window()
        loginPage = LoginPage(driver)

        # 输入登录用户名
        loginPage.userNameObj().send_keys("defang1")
```

```
    # 输入登录密码
        loginPage.passwordObj().send_keys("123")

    # 点击登录按钮
        loginPage.loginButton().click()
        time.sleep(5)

        assert "德芳理财" in driver.page_source
    except Exception as e:
        raise e
    finally:
        # 退出浏览器
        driver.quit()

if __name__ == '__main__':
    # 进行单元测试
    testErpLogin()
    print("登录erp成功")
```

5、在appModules包中新建一个名叫LoginAction.py文件

```
# 实现登录模块的封装方法
import time
from selenium import webdriver
from pageObject.LoginPage import LoginPage
class LoginAction(object):
    def __init__(self):
        print("登录....")
    @staticmethod
    def login(driver, username, password):
        try:
            login = LoginPage(driver)
            # 输入登录用户名
            login.userNameObj().send_keys(username)
            # 输入登录密码
            login.passwordObj().send_keys(password)
            # 点击登录按钮
            login.loginButton().click()
        except Exception as e:
            raise e
```

6、修改testScripts包中的TestErpApprovalflow.py文件

```
from selenium import webdriver
import time
from appModules.LoginAction import LoginAction
def testErpLogin():
    try:
        # 启动谷歌浏览器
        driver = webdriver.Chrome(executable_path="E:\\Python36\\chromedriver")
        # 访问erp首页
        driver.get("https://qa1-erp.jfz.com")
        driver.implicitly_wait(30)
        driver.maximize_window()
        time.sleep(5)
        # 登录erp
        LoginAction.login(driver, "defang1", "123")
        time.sleep(5)
        assert "德芳理财" in driver.page_source
    except Exception as e:
        raise e
    finally:
        # 退出浏览器
        driver.quit()
```

```
if __name__ == '__main__':
    testErpLogin()
    print("登录erp成功")
```

比较TestErpApprovalflow.py修改前后的代码，我们可以发现登录操作的多个步骤被一个函数调用替代了，函数为LoginAction.login(driver,"defang1","123")，此种方式实现了业务逻辑的封装，只要调用一个函数就可以实现登录的操作，大大减少了测试脚本的重复编写，从而实现了代码的封装

7、在DataDrivenFrameWork工程中新建一个config文件夹

在config文件夹里面新建一个名叫PageElementLocator.ini的file，用于配置定位页面元素的定位表达式

```
[erp_login]
loginPage.username = xpath>//*[@name="username"]
loginPage.password = xpath>//input[@name = "password"]
loginPage.loginbutton = xpath>//*[@class='login-btn']
```

8、在config包中新建VarConfig.py文件

```
import os


# 获取当前文件所在目录的绝对路径
parentDirPath = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# 获取存放页面元素定位表达式文件的绝对路径
pageElementLocatorPath = parentDirPath + "\\config\\PageElementLocator.ini"
```

9、在util包中新建一个名叫ParseConfigurationFile.py文件

```
# 用于解析存储定位页面元素的定位表达式文件，以便获取定位表达式
from configparser import ConfigParser
from config.VarConfig import pageElementLocatorPath
class ParseConfigFile(object):
    def __init__(self):
        self.cf = ConfigParser()
        self.cf.read(pageElementLocatorPath)

    def getItemsSection(self, sectionName):
        # 获取配置文件中指定section下面的所有option键值对
        # 并以字典类型返回给调用者
        """注意：
        使用self.cf.items(sectionName)此种方法获取到的配置文件中
        的options内容均被转换成小写，
        比如LoginPage.frame 被转换成了LoginPage.frame """
        optionsDir = dict(self.cf.items(sectionName))
        return optionsDir
```

```
def getOptionValue(self, sectionName, optionName):
        # 获取指定section下面的指定option的值
        value = self.cf.get(sectionName, optionName)
        return value

if __name__ == '__main__':
    pc = ParseConfigFile()
    print(pc.getItemsSection("erp_login"))
    print(pc.getOptionValue("erp_login", "loginPage.username"))
```

10、修改pageObject包中的LoginPage.py文件

```
from util.ObjectMap import *
from util.ParseConfigurationFile import ParseConfigFile
class LoginPage(object):

    def __init__(self, driver):
        self.driver = driver
        self.parseCF = ParseConfigFile()
        self.loginOptions = self.parseCF.getItemsSection("erp_login")
        print(self.loginOptions)

    def userNameObj(self):
        try:
            # 从定位表达式配置文件中读取fram的定位表达式
            locateType, locatorExpression = self.loginOptions["loginPage.username".lower()].split(">")
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

```
    def passwordObj(self):
        try:
            locateType, locatorExpression = self.loginOptions["loginPage.password".lower()].split(">")
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e

    def loginButton(self):
        try:
            locateType, locatorExpression =
self.loginOptions["loginPage.loginbutton".lower()].split(">")
            # 获取登录页面的用户名输入框页面对象，并返回给调用者
            elementObj = getElement(self.driver, locateType, locatorExpression)
            return elementObj
        except Exception as e:
            raise e
```

运行查看结果：

运行TestErpApprovalflow.py代码

至此：我们就将定位页面元素的定位表达式跟程序分离出来，并存储在一个专用的配置文件PageElementLocator.ini中进行集中管理，后续如果页面结构变了，只需要修改此文件中的等号后面的定位表达式就可以，提高了脚本维护效率

作业：

1、完成上面的数据驱动框架代码，并思考

Thanks!

科　技　提　升　投　资　品　质

金斧子
WWW.JFZ.COM