Jenkins+Python+Selenium+Unittest自动化测试框架v2.0

一、什么是web自动化测试?

自动化测试就是将手工测试的过程 ， 转换成代码来执行。

Web是指的web网页，它是在浏览器当中呈现的页面。

功能测试，都是站在用户的角度来测试系统的功能。而用户接触到就是系统的页面，通过在页面上的各种操作来使用其功能。

比如说登陆页面，测试人员是在登陆页面中，第一步：找到用户名的输入框，输入用户名。第二步：找到密码输入框，输入密码。第三步，找到登陆按钮 ，并点击。不同的登陆数据，会带来不同的测试结果。

从这个过程可见，我们做页面的功能测试，特别依赖于页面的元素。因为所有操作都是对页面的某一个元素(比如输入框，比如按钮)。

二、为什么要做web自动化?

开发人员也不能保证，转给测试的软件版本，功能都是正常的（国内现状，基本上没有开发人员会做比较全面的自测工作）。尤其是历史的功能。假设这次上线时的版本，相对于上次上线时的版本，新增了10个功能 。那么测试人员会在这期间，着重测这10个新功能。而除此之外的490个老功能，需要花大量的时间去回归测试。它并不能代替我们的手工测试，毕竟代码是我们人写的，按我们人的思维去做事情。而手工测试是我们大脑高度运转，想象无限。所以新开发的功能，一般还是手工测试为主。那回归测试的历史老功能，基本不会变。重复去做的回归测试工作，就让自动化测试代替。

三、什么样的项目适合做web自动化?

如果测试对象是B/S架构的软件 ，那么web自动化测试就是其中的一种应用。

web自动化本质上是站在用户的角度，和用户一样在页面上找到某个元素对元素进行各种操作，再在页面观察操作的结果是否正确。

所以在web自动化中，不涉及到数据库、接口这些底层相关的内容。直接从页面上看结果。因为就用户而言，在使用一个网站时只关注网站页面的状态。

所以web自动化，以页面为主，非常的依赖于页面的元素。

元素的变化会直接导致自动化用例执行失败。

因此如果是需求非常频繁变动的项目或功能，就完全不需要考虑web自动化了。你写的自动化脚本才写出来，页面就已经变了，得花时间重新更新脚本。

另外，从自动化背景中可知，web自动化主要应用是在项目长期迭代过程中的一种手段。所以如果项目周期短、功能少也可以不用考虑web自动化了。

因此，项目周期长，项目当中已基本稳定的功能模块可以考虑做web自动化。

但是，如果项目本身是以数据为主的，更多的应该考虑接口自动化更合适。比如说以报表为主的系统。

一、环境搭建:

python3及以上

https://www.python.org/downloads/

selenium pip install selenium

webdriver与Chrome对应版本如下:

https://npm.taobao.org/mirrors/chromedriver

Jenkins（Centos 7） 安装 略，可自行安装

二、框架组成：

Test_case:用于存放、管理用例

Test_data:存放参数化数据、上传的图片等

Test_report: 测试报告的存放

Common: 配置文件、邮件模块、封装函数

三、UI自动化涉及的应用：

1.Selenium 用于web自动化，可自行查阅。

2.Unittest 单元测试框架，提供用例组织执行、断言方法、丰富的日志

3.Jenkins 集成自动化框架、实现远程构建、定时无人构建、构建后触发邮件

4.~~HTMLTestRunner生成测试报告测试报告，添加截图（待）可自行选择模板~~

5.用例运行加入失败重运行机制、跳过运行机制

6测试报告采用Unittest单元测试框架拓展TestRunner 类,现在有4种模板可选择

三、部分代码

1.测试用例部分：

```python
#-*-coding:utf-8-*-
#@Time    :2020/11/11 15:04
#@Author  :JS_ErvinChiu
#@Email   :qiuxiongfei@jushiwangedu.com
#@File    :Test_Case.py
#@Software:PyCharm
#from selenium.webdriver.support.select import Select
#from selenium.webdriver.common.keys import Keys

import win32con
import win32gui
from selenium import webdriver
from unittestreport import rerun
from unittestreport import TestRunner
import unittest
import time
import HTML
import os
import HTMLTestRunnerNew

class public_def():

    #登录开始
    """ driver = webdriver.Chrome()
        driver.get("https://betaweb.jushixl.net.cn/#/home")
        driver.implicitly_wait(10)
        driver.maximize_window()
        time.sleep(5)
        driver.find_elements_by_class_name("nav_item")[3].click()
        time.sleep(3)
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[1]/div[1]/div').click()
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[2]/input').send_keys("18515817789")
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[4]/input').send_keys("a123456")
        time.sleep(3)
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/button').click()

    # 登录结束
class Test_JS_Cases(unittest.TestCase):

    @rerun(count=2,interval=5)
    def setUp(self):

        #初始化浏览器会话
        self.chrome_options = webdriver.ChromeOptions()
        self.chrome_options.add_experimental_option("excludeSwitches", ['enable-automation'])
        self.driver = webdriver.Chrome(options=self.chrome_options)
        self.url = "https://betaweb.jushixl.net.cn/#/home"
        self.driver.implicitly_wait(10)
        self.driver.maximize_window()

    def Login(self):
        # 开始登录
        driver = self.driver
        driver.get(self.url)
        time.sleep(5)
        driver.find_elements_by_class_name("nav_item")[3].click()
        time.sleep(3)
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[1]/div[1]/div').click()
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[2]/input').clear()
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[2]/input').send_keys("18515817789")
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[4]/input').clear()
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/div[4]/input').send_keys("a123456")
        time.sleep(3)
        driver.find_element_by_xpath('//*[@id="app"]/div[2]/div/div/div/div[2]/button').click()
        # 结束登录
    #=====================================================上传图片=====================================================
    def upload_chrome(self, filepath):
        # 一级窗口
        dialog = win32gui.FindWindow("#32770", "打开")
        # 二级窗口
        ComboBoxEx32 = win32gui.FindWindowEx(dialog, 0, "ComboBoxEx32", None)
        # 三级窗口
        comboBox = win32gui.FindWindowEx(ComboBoxEx32, 0, "ComboBox", None)
        # 四级窗口--文件路径输入
        edit = win32gui.FindWindowEx(comboBox, 0, "Edit", None)
        # 二级窗口-打开按钮
        button = win32gui.FindWindowEx(dialog, 0, "Button", "打开(&O)")
        # 操作--添加发送文件路径
        win32gui.SendMessage(edit, win32con.WM_SETTEXT, None, filepath)
        # 点击打开按钮
        # time.sleep(3)
        win32gui.SendMessage(dialog, win32con.WM_COMMAND, 1, button)
        time.sleep(5)
        # file_path = r"D:\PycharmProjects\JS_UIAuto_Test\test_data\001.png"
        # upload_chrome(file_path)
    #@rerun(count=2, interval=5)
    #@unittest.skip("测试跳过用例")
    def test_watch_replay(self):

        self.Login()
        driver = self.driver
        time.sleep(3)
        driver.find_elements_by_class_name("nav_item")[2].click()  # 进入我的班级列表
        time.sleep(3)
        driver.find_elements_by_class_name("fast_button")[0].click()  # 进入班级
        time.sleep(5)
        # 选择全部课（默认定位全部课程可省略）
        driver.find_element_by_xpath(
            "/html/body/div[1]/div[2]/div[2]/div[2]/div[2]/div/div[1]/div[2]/div/div[1]/div[1]/div/i").click()
        windows_handle = driver.window_handles  # 获取当前所有handles
```

2.TestRunner 类部分：

```python
"""
====================================
#-*-coding:utf-8-*-
#@Time    :2020/11/20 11:25
#@Author  :JS_ErvinChiu
#@Email   :qiuxiongfei@jushiwangedu.com
#@File    :SendEmail.py
#@Software:PyCharm
====================================
本模块主要是为了解决多线程运行unittest测试用例的问题
该模块预留了两个入口，

注意点：
使用起来非常简单，只需要调用TestRunner的run方法即可执行测试用例，运行的时候可通过参数指定开启的线程数量

"""
import os
import unittest
import time
from concurrent.futures.thread import ThreadPoolExecutor

from unittestreport.core.sendEmail import SendEmail
from unittestreport.core.testResult import TestResult, ReRunResult


class TestRunner():
    """unittest运行程序"""

    def __init__(self, suite: unittest.TestSuite,

                 filename='Js_AutoUI_Report.html',
                 report_dir="D:\PycharmProjects\JS_UIAuto_Test\Test_Report",
                 title='UI自动化测试报告',
                 tester='邱雄飞',
                 desc="聚师网UI自动化测试报告",
                 templates=3
                 ):
        """
        初始化用例运行程序
        :param suites: 测试套件
        :param filename: 报告文件名
        :param report_dir:报告文件的路径
        :param title:测试套件标题
        :param templates: 可以通过参数值1或者2，指定报告的样式模板，目前只有两个模板
        :param tester:测试者
        """
        if not isinstance(suite, unittest.TestSuite):
            raise TypeError("suites 不是测试套件")
        if not isinstance(filename, str):
            raise TypeError("filename is not str")
        if not filename.endswith(".html"):
            filename = filename + ".html"
        self.suite = suite
        self.filename = filename
        self.title = title
        self.tester = tester
        self.desc = desc
        self.templates = templates
        self.report_dir = report_dir
        self.result = []
        self.starttime = time.time()

    def classification_suite(self):
        """
        将测试套件中的用例，根据用例类位单位，拆分成多个测试套件，打包成列表类型
        :return: list-->[suite,suite,suite.....]
        """
        suites_list = []

        def wrapper(suite):
            for item in suite:
                if isinstance(item, unittest.TestCase):
                    suites_list.append(suite)
                    break
                else:
                    wrapper(item)

        wrapper(self.suite)
        return suites_list

    def classification_test_case(self):
        """
        将测试套件中的用例进行拆分，保存到列表中
        :return: list-->[case,case]
        """
        test_list = []

        def wrapper(suite):
            for item in suite:
                if isinstance(item, unittest.TestCase):
                    test_list.append(item)
                else:
                    wrapper(item)

        wrapper(self.suite)
        return test_list

    def run(self, thread_count=1):
        """
        支持多线程执行
        注意点：如果多个测试类共用某一个全局变量，由于资源竞争可能会出现错误
        :param thread_count:线程数量，默认位1
        :return:
        """
        # 将测试套件按照用例类进行拆分
        suites = self.classification_suite()
        with ThreadPoolExecutor(max_workers=thread_count) as ts:
            for i in suites:
                res = TestResult()
                self.result.append(res)
```

四、HTML测试报告生成及邮件发送

封装了TestRunner类，用来代替unittest中的TextTestRunner来执行测试用例，执行完测试用例之后会自动生成测试报告。并且有4种报告风格可选

1.模块导入

```
from unittestteport import TestRunner
```

2.使用案例

```
runner = TestRunner(test_suite)
runner.run()
```

3.关于TestRunner初始化参数

3.1 **suites**: 测试套件（必传）

3.2 **filename**: 指定报告文件名

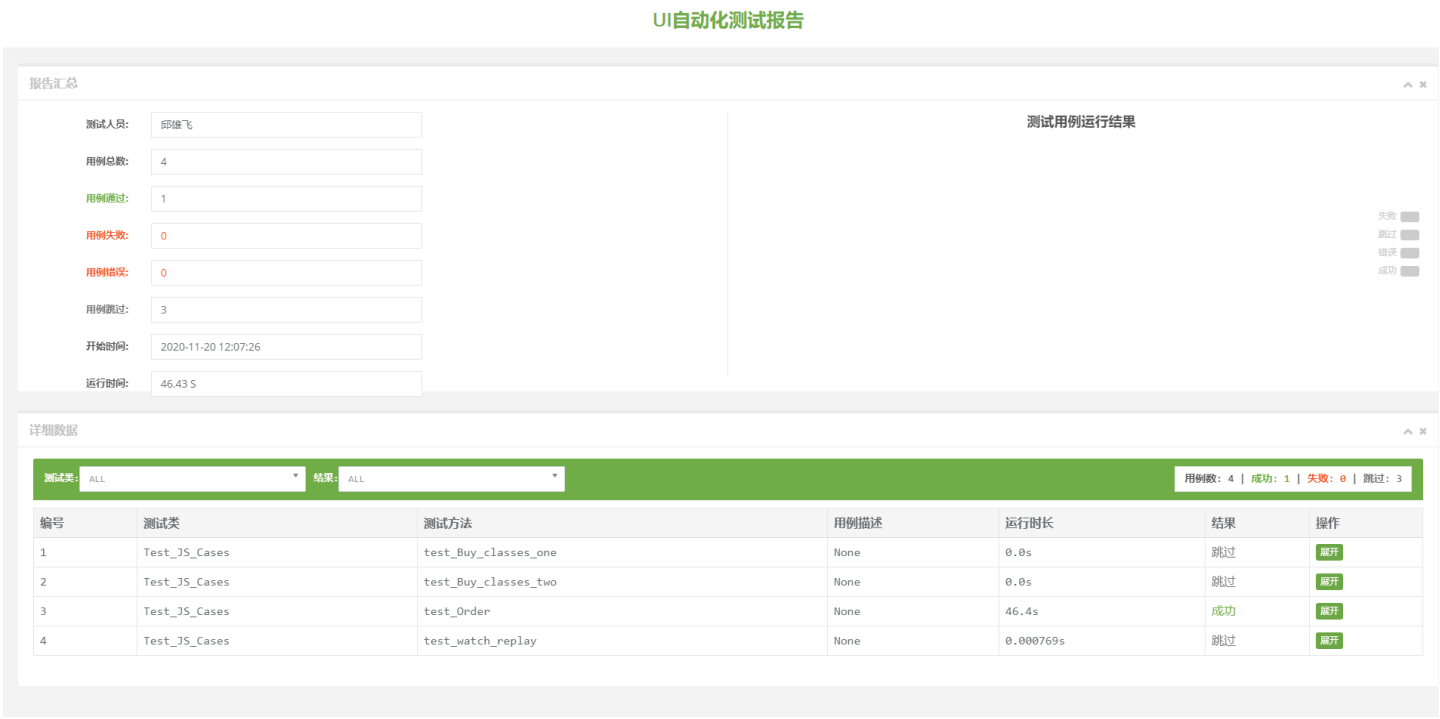
3.3 **report_dir**:指定存放报告路径

3.4 **title**:指定测试报告的标题

3.5 **templates**: 可以指定1，2，3、4四个风格的模板

3.6 **tester**:测试人员名称

4.1报告样式展示：



【UI自动化测试报告】用例执行汇总信息如下：

测试人员	邱雄飞	成功用例	1
开始时间	2020-11-20 12:09:45	失败用例	0
执行时间	47.92 S	错误用例	0
用例总数	4	跳过用例	3

UI自动化测试报告

测试结果汇总

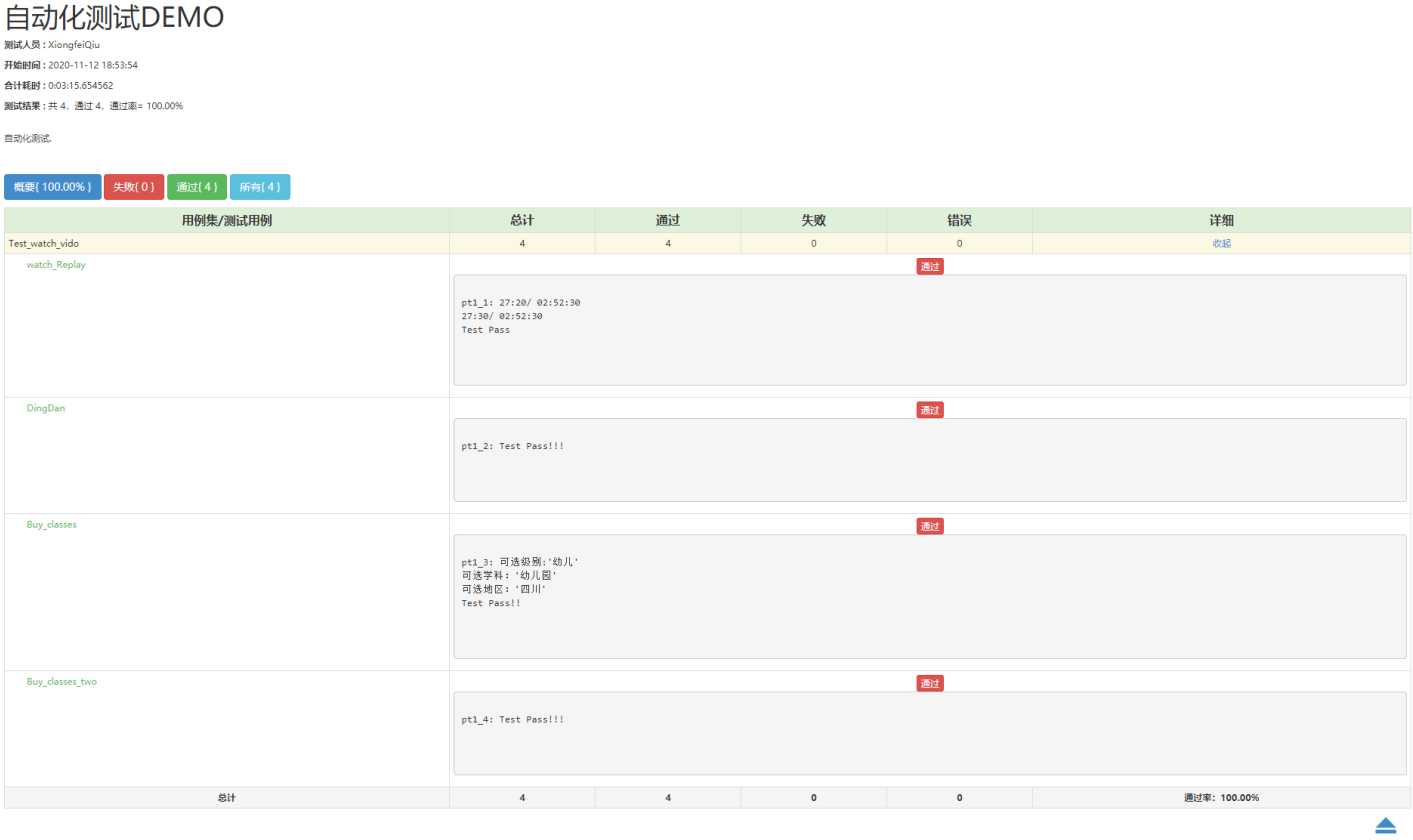
测试人员	邱雄飞	成功用例	1
开始时间	2020-11-20 11:23:28	失败用例	0
执行时间	48.00 S	错误用例	0
用例总数	4	跳过用例	3
描述信息	聚师网UI自动化测试报告		

图表展示

详细信息

编号	测试类 所有	测试方法	用例描述	执行时间	执行结果 所有	详细信息
1	Test_JS_Cases	test_Buy_classes_one	None	0.0s	跳过	查看详情
演示跳过用例						
2	Test_JS_Cases	test_Buy_classes_two	None	0.0s	跳过	查看详情
验证邮件						
3	Test_JS_Cases	test_Order	None	48.0s	成功	查看详情
'订单提交成功，请尽快付款！' 测试结果:Test Pass!! test_Order (__main__.Test_JS_Cases)执行--->【通过】						
4	Test_JS_Cases	test_watch_replay	None	0.0s	跳过	查看详情
测试跳过用例						

4.2邮件模板预览：



自动化测试脚本运行在本地环境，Jenkins运行脚本之前，需建立与本地连接，通过节点配置

自动化脚本目前分为两部分：邮件模块（独立）、用例模块（用例、报告），通过Jenkins批处理命令，构建时分别运行两个脚本：1用例脚本2.邮件脚本（邮件通过sort() 方法取到最新报告，并上传发送邮件,本次测试报告不以时间

戳区分，新报告覆盖）

五、Jenkins集成自动化：