# 初识unittest

## 1.1 unittest相关概念

unittest是python内置的一个单元测试框架，在学习怎么使用它之前，我们先来了解它的一些概念和原理。

* **Test Case：**测试用例，一个TestCase的实例就是一个测试用例。什么是测试用例呢？就是一个完整的测试流程，包括测试前准备环境的搭建(setUp)，执行测试代码(run)，以及测试后环境的还原(tearDown)。单元测试(unit test)的本质也就在这里，一个测试用例是一个完整的测试单元，通过运行这个测试单元，可以对某一个问题进行验证。
* **Test Suite：**测试套件，就是多个测试用例集合在一起
* **Test Runner：**测试执行，用来执行测试用例，其中的run(test)会执行TestSuite/TestCase
* **Test Fixture：**对一个测试用例环境的搭建和销毁，是一个fixture，通过覆盖TestCase的setUp()和tearDown()方法来实现。这个有什么用呢？比如说在这个测试用例中需要访问数据库，那么可以在setUp()中建立数据库连接以及进行一些初始化，在tearDown()中清除在数据库中产生的数据，然后关闭连接。注意tearDown的过程很重要，要为以后的TestCase留下一个干净的环境。

## 1.2 登录脚本示例：

# coding = utf-8

import unittest

import warnings

from selenium import webdriver

from time import sleep

# 驱动文件路径

driverfile\_path = r'D:\coship\Test\_Framework\drivers\IEDriverServer.exe'

class CmsLoginTest(unittest.TestCase):

def setUp(self):

# 这行代码的作用是忽略一些告警打印

warnings.simplefilter("ignore", ResourceWarning)

print("test start!")

self.driver = webdriver.Ie(executable\_path=driverfile\_path)

self.driver.get("http://172.21.13.83:28080/")

def tearDown(self):

self.driver.quit()

print("test end!")

def test\_login1(self):

'''用户名、密码为空'''

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message1 = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

error\_message2 = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message1, '用户名不能为空')

self.assertEqual(error\_message2, '密码不能为空')

print("用例test\_login1执行结束")

def test\_login3(self):

'''用户名、密码正确'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("111111")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

sleep(1)

self.driver.switch\_to.frame("topFrame")

username = self.driver.find\_element\_by\_css\_selector("#nav\_top>ul>li>a").text

self.assertEqual(username,"autotest")

print("用例test\_login3执行结束")

def test\_login2(self):

'''用户名正确，密码错误'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '密码错误,请重新输入!')

print("用例test\_login2执行结束")

def login4(self):

'''用户名不存在'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("test007")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '用户不存在!')

print("用例login4执行结束")

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

## 1.3 执行结果

"C:\Program Files\Python36\python.exe" D:/Git/Test\_Framework/utils/1.py

test start!

用例test\_login1执行结束

.test end!

test start!

用例test\_login2执行结束

.test end!

test start!

用例test\_login3执行结束

test end!

.----------------------------------------------------------------------

Ran 3 tests in 34.044s

OK

Process finished with exit code 0

## 1.4 结论

* 要使用unittest这个测试框架时，需要先导入unittest
* 定义一个测试类时，要继承unittest.TestCase这个类
* setUp()是测试用例执行前的环境准备，tearDown()是测试用例执行结束后的环境恢复，每个测试用例都会执行一次setUp()和tearDown()
* 测试用例必须以test开头，否则不会被执行。
* 测试用例执行顺序是根据用例名称顺序来执行的

# 跳过测试用例（了解）

## 2.1 相关注解函数

在执行测试用例时，有时候有些用例是不需要执行的，那我们怎么办呢？难道删除这些用例？那下次执行时如果又需要执行这些用例时，又把它补回来？这样操作就太麻烦了。

unittest提供了一些跳过指定用例的方法

* @unittest.skip(reason)：强制跳转。reason是跳转原因
* @unittest.skipIf(condition, reason)：condition为True的时候跳转
* @unittest.skipUnless(condition, reason)：condition为False的时候跳转
* @unittest.expectedFailure：如果test失败了，这个test不计入失败的case数目

## 2.2 示例

# coding = utf-8import unittestimport warningsfrom selenium import webdriverfrom time import sleep# 驱动文件路径

driverfile\_path = r'D:\coship\Test\_Framework\drivers\IEDriverServer.exe'

class CmsLoginTest(unittest.TestCase):

def setUp(self):

# 这行代码的作用是忽略一些告警打印

warnings.simplefilter("ignore", ResourceWarning)

self.driver = webdriver.Ie(executable\_path=driverfile\_path)

self.driver.get("http://172.21.13.83:28080/")

def tearDown(self):

self.driver.quit()

@unittest.skip("用户名密码都为空用例不执行")

def test\_login1(self):

'''用户名、密码为空'''

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message1 = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

error\_message2 = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message1, '用户名不能为空')

self.assertEqual(error\_message2, '密码不能为空')

@unittest.skipIf(3 > 2, "3大于2，此用例不执行")

def test\_login3(self):

'''用户名、密码正确'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("111111")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

sleep(1)

self.driver.switch\_to.frame("topFrame")

username = self.driver.find\_element\_by\_css\_selector("#nav\_top>ul>li>a").text

self.assertEqual(username,"autotest")

@unittest.skipUnless(3 < 2,"2没有大于3，此用例不执行")

def test\_login2(self):

'''用户名正确，密码错误'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '密码错误,请重新输入!')

@unittest.expectedFailure

def test\_login4(self):

'''用户名不存在'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("test007")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '用户名不存在!')

def test\_login5(self):

'''用户名为空'''

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

self.assertEqual(error\_message, '用户不存在!')

def test\_login6(self):

'''密码为空'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message, '密码不能为空')

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main(verbosity=2)

## 2.3 执行结果

"C:\Program Files\Python36\python.exe" D:/Git/Test\_Framework/utils/cmslogin.py

test\_login1 (\_\_main\_\_.CmsLoginTest)

用户名、密码为空 ... skipped '用户名密码都为空用例不执行'

test\_login2 (\_\_main\_\_.CmsLoginTest)

用户名正确，密码错误 ... skipped '2没有大于3，此用例不执行'

test\_login3 (\_\_main\_\_.CmsLoginTest)

用户名、密码正确 ... skipped '3大于2，此用例不执行'

test\_login4 (\_\_main\_\_.CmsLoginTest)

用户名不存在 ... expected failure

test\_login5 (\_\_main\_\_.CmsLoginTest)

用户名为空 ... FAIL

test\_login6 (\_\_main\_\_.CmsLoginTest)

密码为空 ... ok

======================================================================

FAIL: test\_login5 (\_\_main\_\_.CmsLoginTest)

用户名为空----------------------------------------------------------------------

Traceback (most recent call last):

File "D:/Git/Test\_Framework/utils/cmslogin.py", line 71, in test\_login5

self.assertEqual(error\_message, '用户不存在!')

AssertionError: '用户名不能为空' != '用户不存在!'

- 用户名不能为空+ 用户不存在!

----------------------------------------------------------------------

Ran 6 tests in 32.663s

FAILED (failures=1, skipped=3, expected failures=1)

Process finished with exit code 1

# 测试套件

## 3.1 为什么要使用测试套件

我们知道测试用例的执行顺序是根据测试用例名称顺序执行的，在不改变用例名称的情况下，我们怎么来控制用例执行的顺序呢？

一个测试文件，我们直接执行该文件即可，但如果有多个测试文件，怎么进行组织，总不能一个个文件执行吧？

要解决上面两个问题，我们就要用到测试套件（TestSuite）了

## 3.2 示例代码

# coding = utf-8import unittestimport warningsfrom selenium import webdriverfrom time import sleep# 驱动文件路径

driverfile\_path = r'D:\coship\Test\_Framework\drivers\IEDriverServer.exe'

class CmsLoginTest(unittest.TestCase):

def setUp(self):

# 这行代码的作用是忽略一些告警打印

warnings.simplefilter("ignore", ResourceWarning)

self.driver = webdriver.Ie(executable\_path=driverfile\_path)

self.driver.get("http://172.21.13.83:28080/")

def tearDown(self):

self.driver.quit()

def test\_login1(self):

'''用户名、密码为空'''

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message1 = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

error\_message2 = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message1, '用户名不能为空')

self.assertEqual(error\_message2, '密码不能为空')

def test\_login3(self):

'''用户名、密码正确'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("111111")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

sleep(1)

self.driver.switch\_to.frame("topFrame")

username = self.driver.find\_element\_by\_css\_selector("#nav\_top>ul>li>a").text

self.assertEqual(username,"autotest")

def test\_login2(self):

'''用户名正确，密码错误'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '密码错误,请重新输入!')

def test\_login4(self):

'''用户名不存在'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("test007")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '用户不存在!')

if \_\_name\_\_ == "\_\_main\_\_":

# 构造测试套件

suite = unittest.TestSuite()

suite.addTest(CmsLoginTest("test\_login1"))

suite.addTest(CmsLoginTest("test\_login2"))

suite.addTest(CmsLoginTest("test\_login4"))

suite.addTest(CmsLoginTest("test\_login3"))

# 执行测试

runner = unittest.TextTestRunner(verbosity=2)

runner.run(suite)

注：verbosity 参数可以控制输出的错误报告的详细程度，默认是 1；如果设为 0，则不输出每一用例的执行结果；如果设为 2，则输出详细的执行结果

## 3.3 执行结果

"C:\Program Files\Python36\python.exe" D:/Git/Test\_Framework/utils/1.py

test\_login1 (\_\_main\_\_.CmsLoginTest)

用户名、密码为空 ... ok

test\_login2 (\_\_main\_\_.CmsLoginTest)

用户名正确，密码错误 ... ok

test\_login4 (\_\_main\_\_.CmsLoginTest)

用户名不存在 ... ok

test\_login3 (\_\_main\_\_.CmsLoginTest)

用户名、密码正确 ... ok

----------------------------------------------------------------------

Ran 4 tests in 44.818s

OK

Process finished with exit code 0

## 3.4 结论

从用例的执行结果中我们可以看到，用例的执行顺序是按照添加用例时的顺序来执行的

一个一个地添加测试用例到测试套件中，有点麻烦，其实我们可以把要执行的测试用例用个列表来管理，然后再把这个列表添加到测试套件

## 3.5 测试套件使用方式改进

把要执行的测试用例用个列表来管理，然后再把这个列表添加到测试套件

if \_\_name\_\_ == "\_\_main\_\_":

# 构造测试套件

suite = unittest.TestSuite()

test\_cases = [CmsLoginTest("test\_login1"),CmsLoginTest("test\_login2"),CmsLoginTest("test\_login4"),

CmsLoginTest("test\_login3")]

suite.addTests(test\_cases)

# 执行测试

runner = unittest.TextTestRunner(verbosity=2)

runner.run(suite)

## 3.6 测试套件使用方式再改进

下面我们就来讲讲多个文件的测试用例组织。

假如我有两个系统的登录需要测试，测试用例分别放在两个文件中(cmslogin.py，smelogin.py)，现在我需要把这两个文件中的用例添加到一个测试套件中来执行，为此我们要重新建立一个叫run\_all.py的文件

### 3.6.1示例代码

import unittestfrom cmslogin import CmsLoginTestfrom smelogin import SmeLoginTest

if \_\_name\_\_ == "\_\_main\_\_":

# 构造测试套件

suite = unittest.TestSuite()

test\_cases = [CmsLoginTest("test\_login1"),CmsLoginTest("test\_login2"),CmsLoginTest("test\_login4"),

CmsLoginTest("test\_login3"),SmeLoginTest("test\_login1"),SmeLoginTest("test\_login2")]

suite.addTests(test\_cases)

# 执行测试

runner = unittest.TextTestRunner(verbosity=2)

runner.run(suite)

### 3.6.2 示例代码2

还可以用addTests + TestLoader方法来添加用例，但是这种方法是无法对case进行排序的

import unittest

from cmslogin

import CmsLoginTest

from smelogin import SmeLoginTest

if \_\_name\_\_ == "\_\_main\_\_":

# 构造测试套件

suite = unittest.TestSuite()

# 第一种方法：传入'模块名.TestCase名'

suite.addTests(unittest.TestLoader().loadTestsFromName('cmslogin.CmsLoginTest'))

suite.addTests(unittest.TestLoader().loadTestsFromName('smelogin.SmeLoginTest'))

# 这里还可以把'模块名.TestCase名'放到一个列表中

suite.addTests(unittest.TestLoader().loadTestsFromNames(['cmslogin.CmsLoginTest','smelogin.SmeLoginTest']))

# 第二种方法：传入TestCase suite.addTests(unittest.TestLoader().loadTestsFromTestCase(CmsLoginTest))

# 执行测试

runner = unittest.TextTestRunner(verbosity=2)

runner.run(suite)

# 批量执行测试用例-discover

前面我们说了，对于不同文件用例，我们可以通过addTest()把用例加载到一个测试套件(TestSuite)来统一执行，对于少量的文件这样做没问题，但是如果有几十上百个用例文件，这样做就太浪费时间了。

unittest中的discover()方法可以批量加载用例

discover(start\_dir, pattern='test\*.py', top\_level\_dir=None)

* start\_dir：测试模块名或测试用例所在目录
* pattern='test\*.py'：表示用例文件名的匹配方式，此处匹配的是以test开头的.py类型的文件，\*表示匹配任意字符
* top\_level\_dir：测试模块的顶层目录

示例代码：

import unittest

if \_\_name\_\_ == "\_\_main\_\_":

# 测试用例目录

test\_dir = r"D:\Git\Test\_Framework\test\_case"

# 加载测试用例

discover = unittest.defaultTestLoader.discover(test\_dir, 'test\*.py')

runner = unittest.TextTestRunner(verbosity=2)

runner.run(discover)

# 装饰器@classmethod（了解）

我们知道setUp()和setDown()的作用是在每条测试用例执行前准备测试环境以及用例测试结束后恢复测试环境，如果我们执行的测试类下所有测试用例的环境准备和环境复原的操作都是一样的，那么我们就没必要每条测试用例执行前都执行一次setUp()和setDown()的操作。在运行测试类前只执行一次环境的准备，测试类运行结束后只执行一次环境复原的操作，这时我们就可以引入装饰器@classmethod

## 5.1示例代码：

# coding = utf-8import unittestimport warningsfrom selenium import webdriverfrom time import sleep# 驱动文件路径

driverfile\_path = r'D:\coship\Test\_Framework\drivers\IEDriverServer.exe'

class CmsLoginTest(unittest.TestCase):

@classmethod

def setUpClass(cls):

print("test start!")

# 这行代码的作用是忽略一些告警打印

warnings.simplefilter("ignore", ResourceWarning)

cls.driver = webdriver.Ie(executable\_path=driverfile\_path)

cls.driver.get("http://172.21.13.83:28080/")

@classmethod

def tearDownClass(cls):

cls.driver.quit()

print("test end!")

def test\_login1(self):

'''用户名、密码为空'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").clear()

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message1 = self.driver.find\_element\_by\_css\_selector("[for='loginName']").text

error\_message2 = self.driver.find\_element\_by\_css\_selector("[for='textfield']").text

self.assertEqual(error\_message1, '用户名不能为空')

self.assertEqual(error\_message2, '密码不能为空')

def test\_login2(self):

'''用户名正确，密码错误'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '密码错误,请重新输入!')

def test\_login3(self):

'''用户名不存在'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("test007")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("123456")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

error\_message = self.driver.find\_element\_by\_css\_selector(".errorMessage").text

self.assertEqual(error\_message, '用户不存在!')

def test\_login4(self):

'''用户名、密码正确'''

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").clear()

self.driver.find\_element\_by\_css\_selector("[name='admin.loginName']").send\_keys("autotest")

self.driver.find\_element\_by\_css\_selector("[name='admin.password']").send\_keys("111111")

self.driver.find\_element\_by\_css\_selector("#imageField").click()

sleep(1)

self.driver.switch\_to.frame("topFrame")

username = self.driver.find\_element\_by\_css\_selector("#nav\_top>ul>li>a").text

self.assertEqual(username,"autotest")

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main(verbosity=2)

## 5.2执行结果：

"C:\Program Files\Python36\python.exe" D:/Git/Test\_Framework/test\_case/test\_cmslogin.py

test start!

test\_login1 (\_\_main\_\_.CmsLoginTest)

用户名、密码为空 ... ok

test\_login2 (\_\_main\_\_.CmsLoginTest)

用户名正确，密码错误 ... ok

test\_login3 (\_\_main\_\_.CmsLoginTest)

用户名不存在 ... ok

test\_login4 (\_\_main\_\_.CmsLoginTest)

用户名、密码正确 ... ok

test end!

----------------------------------------------------------------------

Ran 4 tests in 22.462s

OK

Process finished with exit code 0

## 5.3结论：

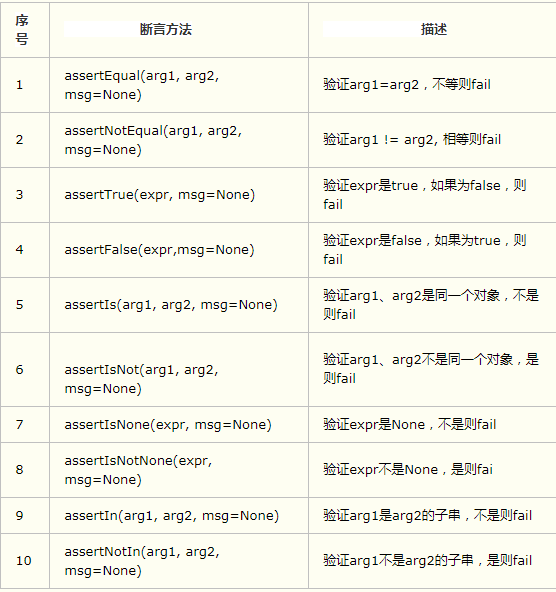
从执行结果中可以看到，整个测试类的运行过程中，只执行了一次setUpClass()和tearDownClass()

# 关于断言

## 6.1 断言概述

我们在执行测试用例时，怎么来判断这条用例是否通过呢？唯一的办法就是拿实际结果和预期结果进行比较，如果一致用例就是通过的，否则用例就是失败的。在python中这种比较的方法就叫做断言，unittest框架提供了一系列的断言方法

## 6.2 常见的断言方法



# 自动生成测试报告

## 7.1 默认生成文本测试报告

用例执行完成后，执行结果默认是输出在屏幕上，其实我们可以把结果输出到一个文件中，形成测试报告。

unittest自带的测试报告是文本形式的

示例：

import unittest

if \_\_name\_\_ == "\_\_main\_\_":

# 测试用例目录

test\_dir = r"D:\Git\Test\_Framework\test\_case"

# 加载测试用例

discover = unittest.defaultTestLoader.discover(test\_dir, 'test\*.py')

# 测试报告路径

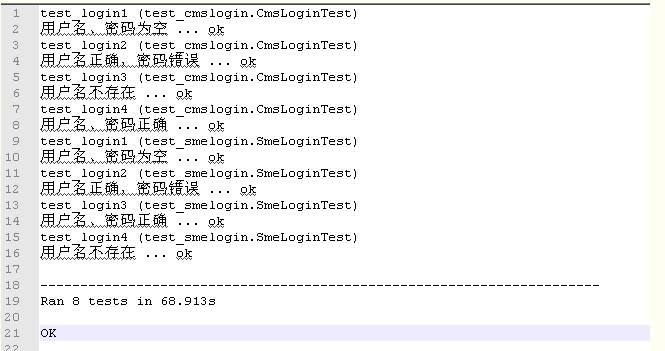
report\_path = r"D:\Git\Test\_Framework\report\report.text"

with open(report\_path,"a") as report:

runner = unittest.TextTestRunner(stream=report,verbosity=2)

runner.run(discover)

生成的txt测试报告如下：



这种TXT文件测试报告不能直观地展示用例执行情况，引入第三方模块HTMLTestRunner，可以展示一份优美的html格式的测试报告。

## 7.2 自动生成HTML格式的测试报告

### 7.2.1 下载HTMLTestRunner模块

下面我们就来介绍下HTMLTestRunner模块

HTMLTestRunner模块不能通过pip安装，必须先下载下来下载地址：<http://tungwaiyip.info/software/HTMLTestRunner.html>



### 7.2.2 配置HTMLTestRunner环境

直接右键另存为就行了，下载完成后放到python安装目录下的../Python36\Lib

因为HTMLTestRunner模块是基于Python2开发的，目前停止更新了，在Python3上运行会报错。为了使其支持Python3环境，需要对其中的部分内容进行修改：

第94行，将import StringIO修改成import io

第539行，将self.outputBuffer = StringIO.StringIO()修改成self.outputBuffer= io.StringIO()

第631行，将print >> sys.stderr, ‘\nTime Elapsed: %s‘ %(self.stopTime-self.startTime)修改成print(sys.stderr, ‘\nTimeElapsed: %s‘ % (self.stopTime-self.startTime))

第642行，将if not rmap.has\_key(cls):修改成if not cls in rmap:

第766行，将uo = o.decode(‘latin-1‘)修改成uo = e

第772行，将ue = e.decode(‘latin-1‘)修改成ue = e

修改完成后保存。

现在我们就可以生成一份优美的测试报告了

### 7.2.3 示例代码

import unittestfrom HTMLTestRunner import HTMLTestRunner

if \_\_name\_\_ == "\_\_main\_\_":

# 测试用例目录

test\_dir = r"D:\Git\Test\_Framework\test\_case"

# 加载测试用例

discover = unittest.defaultTestLoader.discover(test\_dir, 'test\*.py')

# 测试报告路径

report\_path = r"D:\Git\Test\_Framework\report\report.html"

with open(report\_path,"wb") as report:

runner = HTMLTestRunner(stream = report,

title = "测试报告",

description = "系统登录测试用例执行")

runner.run(discover)

### 7.2.4 测试报告生成

