**unittest教程**

Python中有一个自带的单元测试框架是unittest模块，用它来做单元测试，它里面封装好了一些校验返回的结果方法和一些用例执行前的初始化操作。在说unittest之前，先说几个概念：

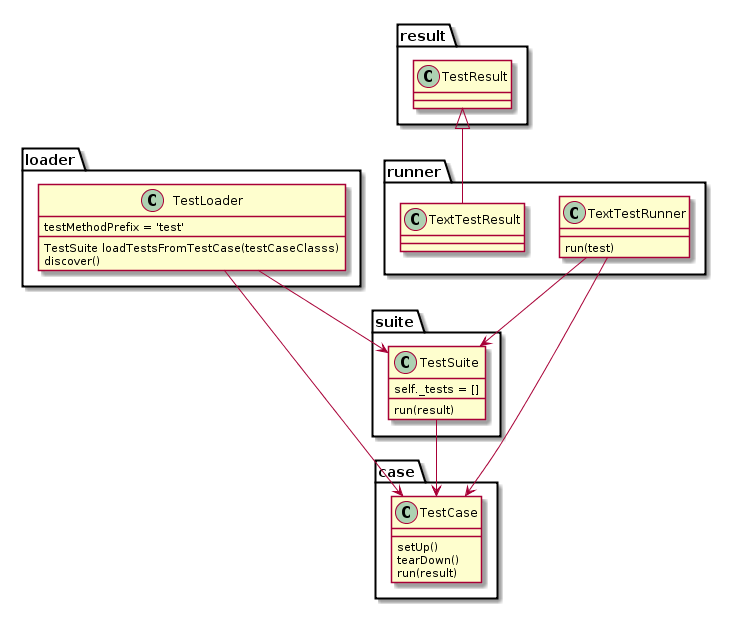
TestCase 也就是测试用例

TestSuite 多个测试用例集合在一起，就是TestSuite

TestLoader是用来加载TestCase到TestSuite中的

TestRunner是来执行测试用例的,测试的结果会保存到TestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息

下面我们分别来解释这四个概念的意思，先来看一张unittest的静态类图。



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。

而多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。 TestLoader是用来加载TestCase到TestSuite中的，其中有几个loadTestsFrom\_\_()方法，就是从各个地方寻找TestCase，创建它们的实例，然后add到TestSuite中，再返回一个TestSuite实例。

TextTestRunner是来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的run(result)方法。

测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

一个class继承了unittest.TestCase，便是一个测试用例，但如果其中有多个以 test 开头的方法，那么每有一个这样的方法，在load的时候便会生成一个TestCase实例，如：一个class中有四个test\_xxx方法，最后在load到suite中时也有四个测试用例。

在每一个测试用例中可以重写 以下函数

setUp()该测试用例执行前的设置工作、

tearDown()该测试用例执行后的清理工作、

setUpClass()所有测试用例前的设置工作、

tearDownClass()所有测试用例执行后的清洗工作

在每一个测试用例中可以通过skip，skipIf，skipUnless装饰器跳过某个测试函数，或者用TestCase.skipTest方法跳过测试函数。

@unittest.skip()

@unittest.skipIf()

@unittest.skipUnless()

到这里整个流程就清楚了：

写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中，我们通过命令行或者unittest.main()执行时，main会调用TextTestRunner中的run来执行，或者我们可以直接通过TextTestRunner来执行用例。

加个说明：

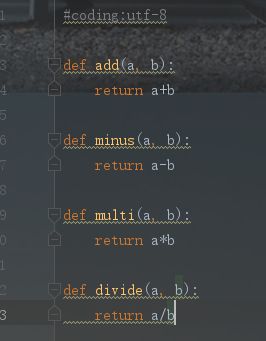
1、TestLoader加载TestCase到TestSuite可以通过TestSuite实例对象的addTest()和addTests()方法向suite中添加case或suite

2、在Runner执行时，默认将执行结果输出到控制台，我们可以设置其输出到文件，在文件中查看结果（你可能听说过HTMLTestRunner，是的，通过它可以将结果输出到HTML中，生成漂亮的报告，它跟TextTestRunner是一样的，从名字就能看出来，这个我们后面再说）。

3、在进行测试时可以传递verbosity参数，用以控制执行结果的输出，0 是简单报告、1 是一般报告、2 是详细报告。

被测方法：

这里我们随意写几个业务函数，表示我们将要进行测试的功能函数。将功能函数文件存储成myfun.py



TestCase测试用例

我们通过测试用例用代码来实现每一个测试的详细过程和针对测试目标要测试的内容。同目录下创建test\_myfun.py



注意：

skip装饰器一共有三个 unittest.skip(reason)、unittest.skipIf(condition,reason)、unittest.skipUnless(condition,reason)，skip无条件跳过，skipIf当condition为True时跳过，skipUnless当condition为False时跳过。

每个测试方法均以 test 开头，否则是不被unittest识别的。其实每一个test开头的方法都会加载为独立的测试用例。在unittest.main()中加 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。如果参数为2则表示输出详细结果。

TestSuite测试套件

TestSuite用来控制多个测试用例和多个测试文件之间的测试顺序。（这里的示例中的几个测试方法并没有一定关系，但之后你写的用例可能会有先后关系，需要先执行方法A，再执行方法B），我们添加到TestSuite中的case是会按照添加的顺序执行的。

1. **创建测试套件TestSuite()**

**语法：1、实例化、2、调用addTest方法**

**TestSuite()用法：**

1----suite = unittest.TestSuite() #实例化Testsuite类

注：TestSuite()主要用来创建测试套件集合

2----suite.addTest(TestCount('test\_add'))

suite.addTest(文件名.类名(‘类中的方法名’))

#调用TestSuite类中的addTest：suite.addTest(类名(‘类中的方法名’))

执行顺序是按照添加的顺序执行，先添加的先执行

注：test\_add是unittest脚本中的被测方法，如果是添加被导入模块中的方法，则需要指定哪个文件名,例：用例模块划分：all\_test.py

suite.addTest(文件名.类名(‘类中的方法名’))

1. **运行测试套件TextTestRunner()**

**语法：1、实例化、2、调用run方法**

1----runner = unittest.TextTestRunner() #实例化TextTestRunner类

2----runner.run(suite) #调用Testsuite类中的addTest方法

注：TextTestRunner()主要用来运行测试套件

坑：

# [使用Unittest做单元测试，addTest（）单个case的时候却执行全部的case](https://www.cnblogs.com/kaerxifa/p/11046156.html)

 首先造成这个结果的原因是pycharm配置问题

测试代码：

[复制代码](javascript:void(0);)

import unittest

class Testadd(unittest.TestCase):

def setUp(self):

self.a = 20

self.b = 10

def test\_add(self):

result = self.a + self.b

self.assertEqual(result,30)

def test\_sub(self):

result = self.a -self.b

self.assertEqual(result,10)

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

suite = unittest.TestSuite()

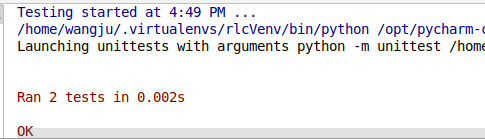
suite.addTest(Testadd("test\_add"))

runner = unittest.TextTestRunner()

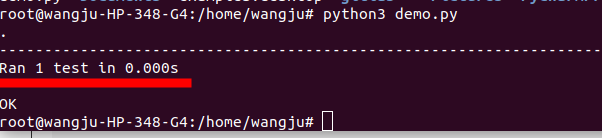
runner.run(suite)

[复制代码](javascript:void(0);)

**pycharm执行效果：**



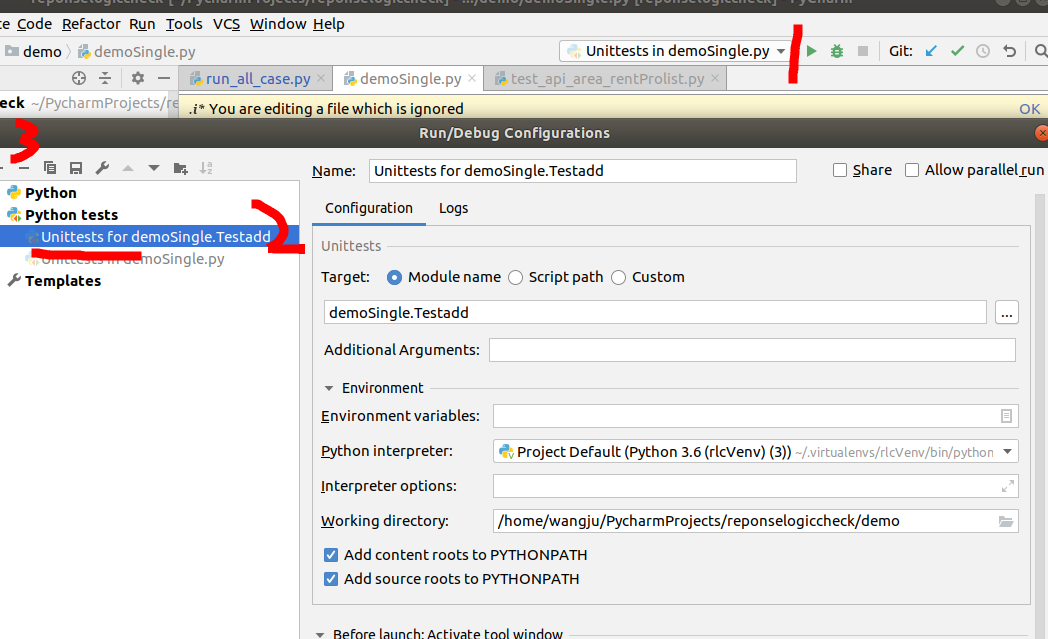
**命令行执行效果**



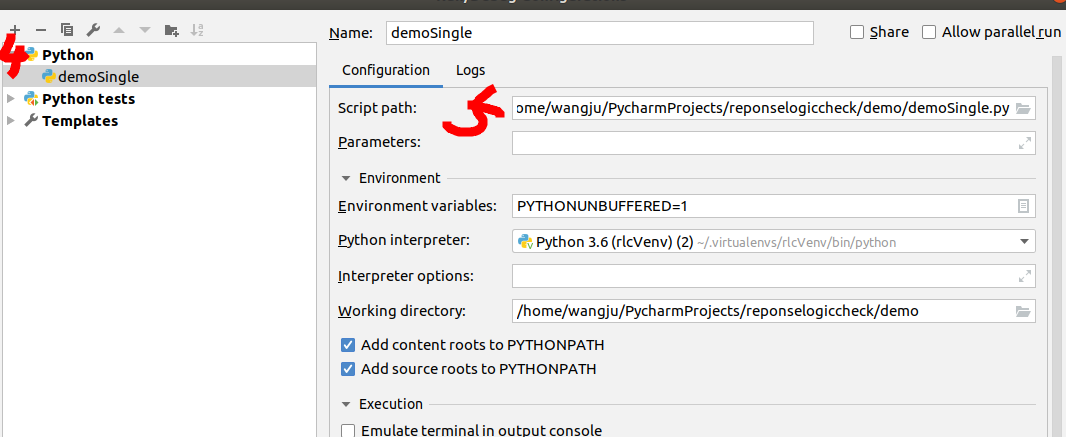
**问题解决：**

1. 点击pycharm的右上角下拉菜单，点击Edit configurations

1. 将Python tests里的对应文件的py.test for...或者unittest for...的文件删除（选中后点击左上角的减号）



1. 点击+，在下拉菜单中选择Python，然后在右边的script path里...选中所要运行的文件



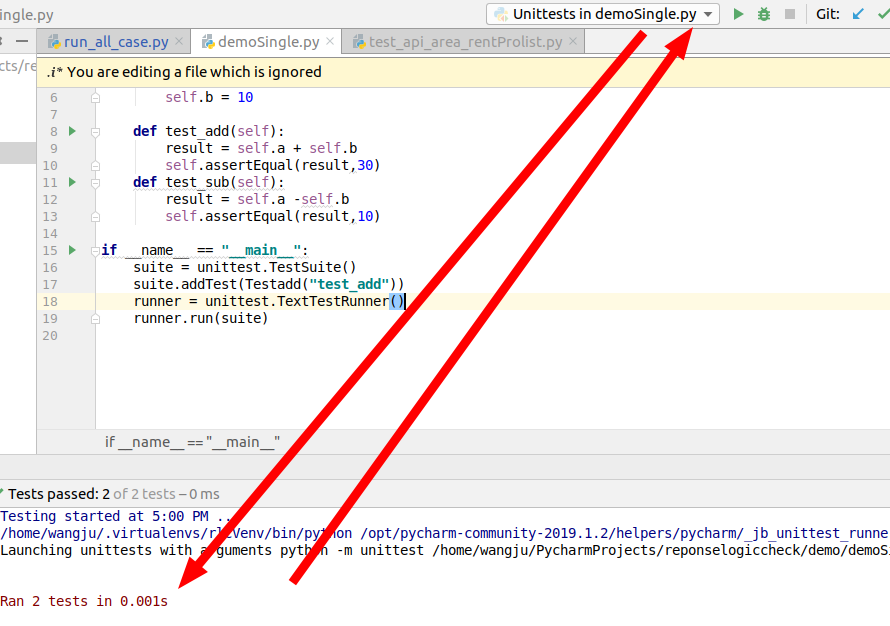
1. 最后点击ok即可，再在所要运行的文件处（最好是main处）点击右键就会发现run unittest变成了run

再次在pycharm中运行，well done

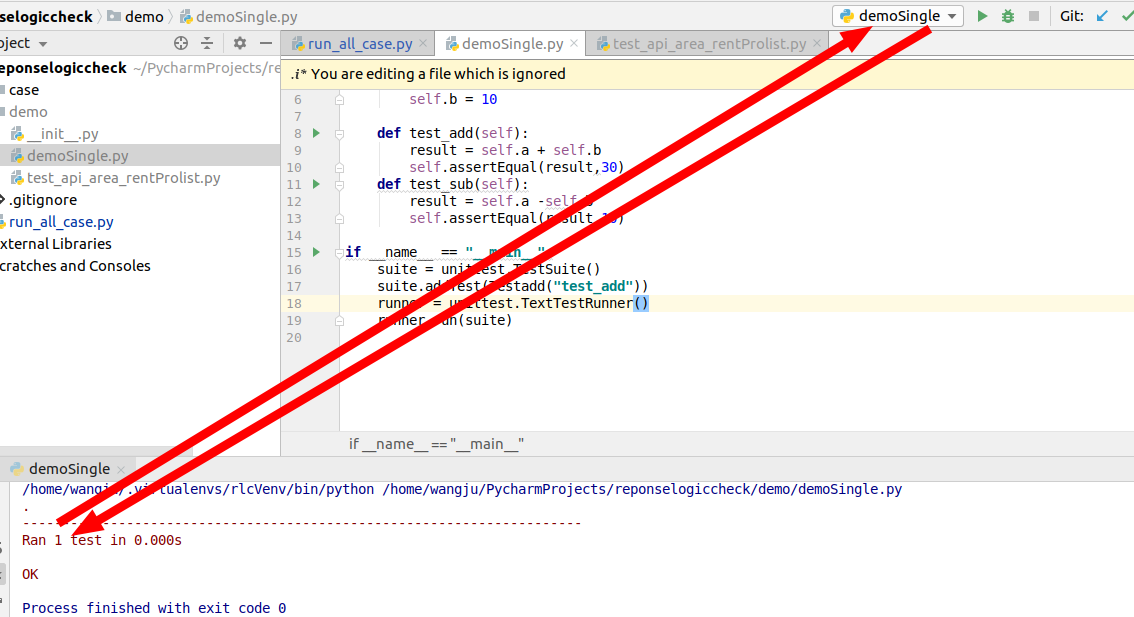


总结：

执行unittest in demoSingle就会运行全部case



而如果运行py文件本身就会只运行addsuite添加的用例



1. **单元测试框架实现模块整合---discover方法**

遍历所有测试脚本实现整合

使用测试套件单独管理全部功能的用例，利用测试套件整合

思考：即便把多条用例集成在一起，但是过多的用例不可能都放在一个脚本中，怎样对用例批量管理

**TestLoader**

该类根据各种标准负责加载测试用例，并返回给测试套件。正常情况下没有必要创建这个类的实例。unittest 提供了可以共享了 defaultTestLoader 类，可以使用其子类和方法创建实例，所以我们可以使用其下面的 discover()方法来创建一个实例。

既：

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

#运行方法同上

runner = unittest.TextTestRunner()

runner.run(discover)

找到指定目录下所有测试模块，并可递归查到子目录下的测试模块，只有匹配到文件名才能被加载。

**start\_dir ：要测试的模块名或测试用例目录。（采用双斜线或目录前加r）**

**pattern='test\*.py' ：表示用例文件名的匹配原则。星号“\*”表示任意多个字符。**

**top\_level\_dir=None：测试模块的顶层目录。如果没顶层目录（也就是说测试用例不是在该目录下则需要分别指定），默认为 None。**

见all\_test2.py文件或all\_test3.py

注：discover发现的py文件不能控制先后顺序，只能通过文件名字的assccii码值来判断先后运行顺序，所以文件命名的时候可以通过testA……等来定义

总结：单元测试的执行方法一共有3种：

第一种：通过unittest.main()加载全部test开头的用例并自动执行

第二种：通过添加测试套件的方法addTest，然后运行添加好的测试套件

第三种：通过testloader来加载指定目录下的test开头的用例，运行添加好的discover

思考：前面的脚本尽管实现了模块化，但是还不完整，比如我们想了解整个自动化测试过程中每个用例的执行情况

1. **测试报告**

**1、通过打开/写入文件，编写测试报告**

**自己完成**

**2、通过 HTMLTestRunner.py 来生成测试报告。**

HTMLTestRunner 是 Python 标准库的 unittest 单元测试框架的一个扩展。它生成易于使用的 HTML 测试报告。 HTMLTestRunner 是在 BSD 许可证下发布。

首先要下 HTMLTestRunner.py 文件，下载地址：

http://tungwaiyip.info/software/HTMLTestRunner.html

HTMLTestRunner.py 本是一个.py 文件，将它放到 Python 安装目录下即可调用。

1、语法：

首先引入HTMLTestRunner包

import HTMLTestRunner

用法：

1----------#定义个报告存放路径

**filename** = 'C:\\test\_object\\report\\result.html'

2----------#定义一个文件名，以写方式打开

fp = open(**filename**, 'wb')

3----------#定义测试报告

runner =HTMLTestRunner.HTMLTestRunner(

stream=fp,

title=u'百度搜索测试报告',

description=u'用例执行情况：')

4----------#运行测试用例

runner.run(suite)

5----------#关闭报告文件

fp.close()

例子：见unittest\_r.py

思考：生成报告后我们想更直观的去了解报告内容，从名字上来区分什么时间生成的报告