单元测试学习总结

Python下的单元测试框架有很多种，如unittest, testtools, subunit, coverage, testrepository, nose, mox, mock, fixtures, discover，再加上setuptools, distutils等。本次总结挑选了使用较多的unittest、pytest、nose等框架进行总结。

# 1.Unittest学习总结(较多使用)

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。单元测试作为任何开发语言都必须的阶段，可以帮助我们很快准确的定位到问题的位置，以及出现问题的模块和单元。python单元测试是对程序中最小的可测试模块——函数，来进行测试。因为单元测试的对象是函数，也就是说被测试对象一定要有输出结果，哪怕就是异常输出，也要有输出，以便单元测试模块能够捕获返回值，并且与预期值进行比较，从而得出测试通过与否。

Unittest属于python的内建模块，是python单元测试的事实标准，也叫PyUnit。使用Unittest之前，先了解如下几个概念。

## 1.1 名词介绍

test case：测试用例，可以通过创建unitest.TestCase类的子类创建一个测试用例。

test fixture：包含执行测试用例前的测试准备工作、测试用例执行后的清理工作（分别对应TestCase中的setUp()和tearDown()方法），测试准备和测试清理的目的是保证每个测试用例执行前后的系统状态一致。

test suite：测试套，是测试用例、测试套或者两者的集合，用来将有关联的测试项打包。

test runner：负责执行测试并将结果展示给用户，可以展示图形或文字形式(unittest.TextTestRunner)的结果，或者返回一个错误码标识测试用例的执行结果。test runner提供了一个方法run()，接受一个unittest.TestSuite或unittest.TestCase实例作为参数，执行对应测试项目后返回测试结果unittest.TestResult对象。

接下来简要介绍一下unittest模块的内容：

## 1.2 Unittest模块内容

**'FunctionTestCase'**：函数测试用例，即给一个函数作为参数，返回一个testcase实例，可选参数有set-up，tear-down方法

**'TestCase'**：所有测试用例的基本类，给一个测试方法的名字，返回一个测试用例实例

**'TestLoader'**：测试用例加载器，其包括多个加载测试用例的方法。返回一个测试套件

**loadTestsFromModule(self, module)**：根据给定的模块实例来获取测试用例套件

**loadTestsFromName(self, name, module=None)**：根据给定的字符串来获取测试用例套件，字符串可以是模块名，测试类名，测试类中的测试方法名，或者一个可调用的是实例对象(这个实例对象返回一个测试用例或一个测试套件)。

**loadTestsFromNames(self, names, module=None)**：和上面功能相同，只不过接受的是字符串列表

**loadTestsFromTestCase(self, testCaseClass)**：根据给定的测试类，获取其中的所有测试方法，并返回一个测试套件

**'TestProgram'**：命令行进行单元测试的调用方法，作用是执行一个测试用例。其实unittest.main()方法执行的就是这个命令， 而这个类实例时默认加载当前执行的作为测试对象， 原型为 \_\_init\_\_(self, module='\_\_main\_\_', defaultTest=None, argv=None, testRunner=xx, testLoader=xx)其中module='\_\_main\_\_'就是默认加载自身。

**'TestResult'**：测试用例的结果保存实例，通常有测试框架调用

**'TestSuite'**：组织测试用例的实例，支持测试用例的添加和删除，最终将传递给testRunner进行测试执行

**'TextTestRunner'**：进行测试用例执行的实例，其中Text的意思是以文本形式显示测试结果。显示测试名称，即完成的测试结果，其过同执行单元测试脚本时添加-v参数

**'defaultTestLoader'**：其实就是TestLoader

**'main'**: 其实就是TestProgram

**'makeSuite'**：通常是由单元测试框架调用的，用于生产testsuite对象的实例

## 1.3 Unittest步骤

一个TestCase的实例就是一个测试用例。测试用例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)。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。

其实整个单元测试框架的逻辑出来了。分四步走：

1.第一步testloader根据传入的参数获得相应的测试用例，即对应具体的测试方法；

2.然后makesuite在把所有的测试用例组装成testsuite；多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。

3.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。

4.最后把testsiute传给testrunner进行执行。其中的run(test)会执行TestSuite/TestCase中的run(result)方法。

5.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。

而我们通常执行的unittest.main()，其实就是unittest.testprom方法，其执行的功能就是上面分析的四步：

1.在第一步中其传入的参数是自身的模块\_\_main\_\_；

2.在第二步中把自身模块中的所有测试类中中的测试方法提取出来，并生成测试套件；

3.最后再把测试套件传递给testrunner进行具体的测试。

4.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。

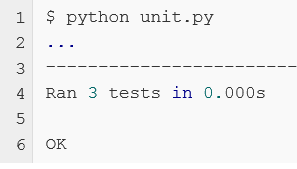
整个流程就清楚了，首先是要写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中，整个过程集成在unittest.main模块中。

## Unittest初步使用

### 定义测试用例

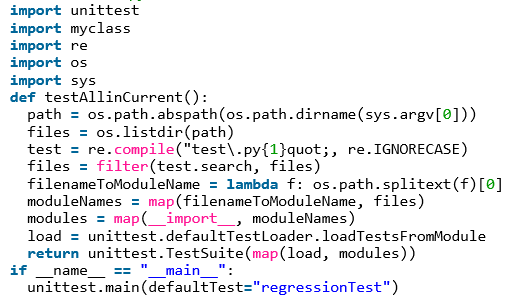


### 1.4.2 执行脚本



## 1.5 Unittest实例

测试用例文件必须为test开头，如：testxxx.py, 当然这个文件本身是一个单元测试的文件。



## 1.6 总结

单元测试目前同其他测试一样没有受到足够的重视，但是其优点不应该被忽略：

1、它是一种验证行为。

程序中的每一项功能都是测试来验证它的正确性。它为以后的开发提供支援。就算是开发后期，我们也可以轻松的增加功能或更改程序结构，而不用担心这个过程中会破坏重要的东西。而且它为代码的重构提供了保障。这样，我们就可以更自由的对程序进行改进。

2、它是一种设计行为。

编写单元测试将使我们从调用者观察、思考。特别是先写测试（test-first），迫使我们把程序设计成易于调用和可测试的，即迫使我们解除软件中的耦合。

3、它是一种编写文档的行为。

单元测试是一种无价的文档，它是展示函数或类如何使用的最佳文档。这份文档是可编译、可运行的，并且它保持最新，永远与代码同步。

4、它具有回归性。

自动化的单元测试避免了代码出现回归，编写完成之后，可以随时随地的快速运行测试。

# 2 Pytest总结

## 2.1 pytest简介

pytest 有时也被称为 py.test，是因为它使用的执行命令是 $ py.test。本文中我们使用 pytest 指代这个测试框架，py.test 特指运行命令。

pytest是一个非常成熟的全功能的Python测试框架，主要特点有以下几点：

1、简单灵活，容易上手；

2、支持参数化；

3、能够支持简单的单元测试和复杂的功能测试，还可以用来做selenium/appnium等自动化测试、接口自动化测试（pytest+requests）;

4、pytest具有很多第三方插件，并且可以自定义扩展，比较好用的如pytest-selenium（集成selenium）、pytest-html（完美html测试报告生成）、pytest-rerunfailures（失败case重复执行）、pytest-xdist（多CPU分发）等；

5、测试用例的skip和xfail处理；

6、可以很好的和jenkins集成；

## 2.2 pytest安装

使用以下3条安装命令：

pip install -U pytest

　　pip install -U pytest-html

pip install -U pytest-rerunfailures

## 2.3 pytest实例

### 2.3.1 简单功能的测试

import pytest

# content of test\_sample.py

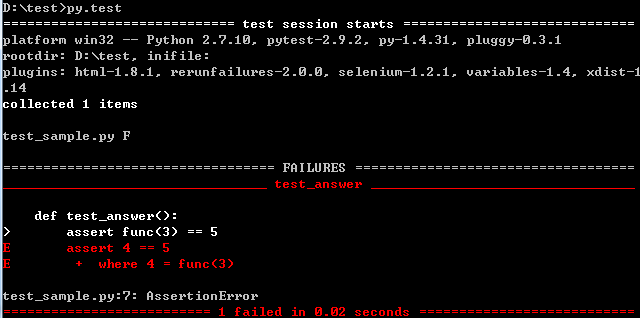
def func(x):

return x + 1

def test\_answer():

assert func(3) == 5

运行 py.test 或 指定特定文件 py.test -q test\_sample.py得出以下结果：



### 2.3.2 一个测试类中创建多个测试用例

import pytest

class TestClass:

def test\_one(self):

x = "this"

assert "h" in x

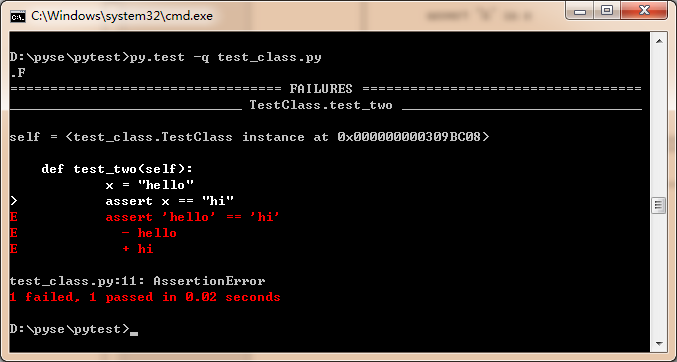
def test\_two(self):

x = "hello"

assert x == "hi"

运行测试：

>py.test -q test\_class.py



-q 为quiet。表示在安静的模式输出报告诉。加不加这个参有什么区别呢？ 读者可以对比一下两次输出的日志。其实，就是少了一些pytest的版本信息。

# 3 nose总结

## 3.1 nose简介

nose 项目是于2005 年发布的，也就是 py.test改名后的一年。它是由 Jason Pellerin 编写的，支持与 py.test 相同的测试习惯做法，但是这个包更容易安装和维护。

## 3.2 nose 安装

Nose是Python的第三方组件，在大多数Unix系统中使用以下命令来安装nose：

easy\_install nose

或者

pip install nose

或者

python setup.py install

安装完成之后nosetests就出现在了Python的安装目录下的scripts目录中。

之后运行项目的测试：

cd path**/**to**/**project

nosetests

可以看到输出以下内容：

**..................................**

**----------------------------------------------------------------------**

Ran 34 tests **in** 1.440s

OK

表示测试正在nose里边运行。

使用nosetests –h寻找帮助。

## 3.3 nose实例

在python安装目录下的Scripts目录下有一个test.py文件，内容如下

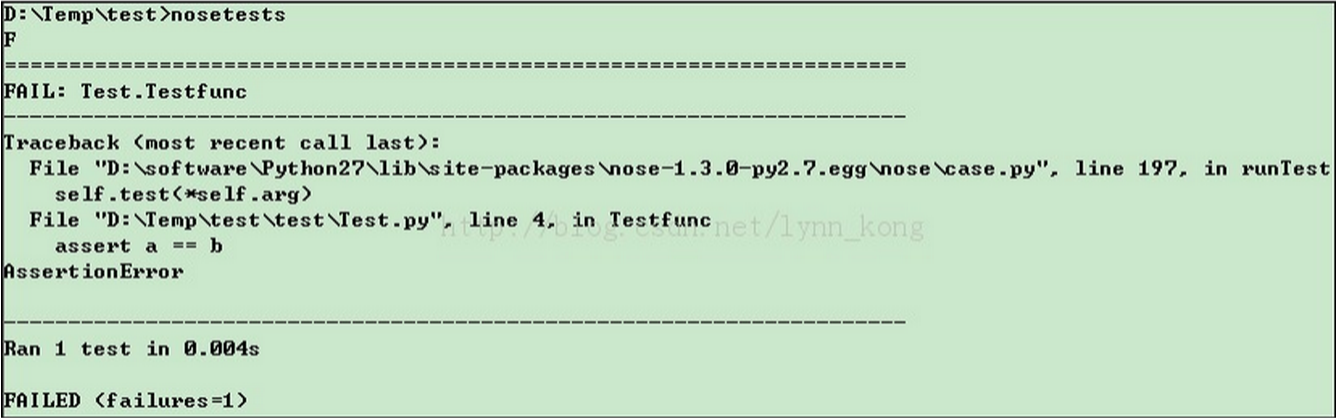
def Testfunc():

a = 1

b = 2

assert a == b

在D:\Temp\test目录中进入命令行，执行nosetests：



所有都是自动的发现和执行，返回了failed结果。Nose可以自动识别源文件、目录或包中的测试用例。