概述

Python单元测试框架（The Python unit testing framework），简称为PyUnit， 是Kent Beck和Erich Gamma这两位聪明的家伙所设计的 JUnit 的Python版本。 而JUnit又是Kent设计的Smalltalk测试框架的Java版本。它们都是各自语言的标准测试框架。

自从 Python 2.1 版本后，PyUnit成为 Python标准库的一部分。

所以不需要安装直接就用。

构建测试用例

在测试过程中，一般一个测试场景由多个测试用例构建而成（即Test Cases）,在PyUnit中，unittest模块中的 TestCase 类代表测试用例。

在使用PyUnit实际测试过程中，一般一个TestCase类实例代表一个场景，由一个setUp和一个tearDown方法以及N个testXXX方法(每个方法里至少一个断言)构成。

基本形式如下：

```
# -*- coding:utf-8 -*-
import unittest

class myTestCase_01(unittest.TestCase):
    def setUp(self):
        # 做一些初始化

    def tearDown(self):
        # 做一些清理动作

    def testTestCase_01(self):
        # 测试用例 1

        # ...

        # 断言验证

    def testTestCase_02(self):
        # 测试用例 2

        # ...

        # 断言验证

class myTestCase_02(unittest.TestCase):
    def setUp(self):
        # 做一些初始化

    def tearDown(self):
```

```
        # 做一些清理动作

    def testTestCase_01(self):
        # 测试用例 1

        # ...

        # 断言验证

    def testTestCase_02(self):
        # 测试用例 2

        # ...

        # 断言验证
```

构建测试套件

将多个测试用例构建在一起就形成了测试套件，下面我们看看在PyUnit中如何把多个测试用例实例构成成测试套件。

```
def suite():
    suite = unittest.TestSuite()
    suite.addTest(testTestCase_01("testTestCase_01"))
    suite.addTest(testTestCase_01("testTestCase_02"))
    suite.addTest(testTestCase_02("testTestCase_01"))
    suite.addTest(testTestCase_02("testTestCase_02"))

    return suite
```

运行

下面我们看下怎么运行上述的套件：

```
# -*- coding:utf-8 -*-

if __name__ == "__main__":
    suites = suite()
    runner = unittest.TextTestRunner()
    runner.run(suites)
```

够不够？

从实际的应用来看是不够的，但从思路来讲，总体而言，就是上述的三个过程：

- 构建测试用例集

- 由测试用例构建套件集
- 运行套件

至于PyUnit中的高级使用，例如什么动态方法、静态方法、不同的运行模式、不同的套件构建方式以及自动发现测试用例等等，请出门左转到官方网站往死里啃。

应该掌握的断言

基本断言方法

基本的断言方法提供了测试结果是True还是False。所有的断言方法都有一个msg参数，如果指定msg参数的值，则将该信息作为失败的错误信息返回。

序号	断言方法	断言描述
1	assertEqual(arg1, arg2, msg=None)	验证arg1=arg2，不等则fail
2	assertNotEqual(arg1, arg2, msg=None)	验证arg1 != arg2, 相等则fail
3	assertTrue(expr, msg=None)	验证expr是true，如果为false，则fail
4	assertFalse(expr,msg=None)	验证expr是false，如果为true，则fail
5	assertIs(arg1, arg2, msg=None)	验证arg1、arg2是同一个对象，不是则fail
6	assertIsNot(arg1, arg2, msg=None)	验证arg1、arg2不是同一个对象，是则fail
7	assertIsNone(expr, msg=None)	验证expr是None，不是则fail
8	assertIsNotNone(expr, msg=None)	验证expr不是None，是则fail
9	assertIn(arg1, arg2, msg=None)	验证arg1是arg2的子串，不是则fail
10	assertNotIn(arg1, arg2, msg=None)	验证arg1不是arg2的子串，是则fail
11	assertIsInstance(obj, cls, msg=None)	验证obj是cls的实例，不是则fail
12	assertNotIsInstance(obj, cls, msg=None)	验证obj不是cls的实例，是则fail

比较断言

unittest框架提供的第二种断言类型就是比较断言。

下面我们看下各种比较断言：

1. assertAlmostEqual (first, second, places = 7, msg = None, delta = None)

验证first约等于second。 palces: 指定精确到小数点后多少位，默认为7

2. assertNotAlmostEqual (first, second, places, msg, delta)

验证first不约等于second。 palces: 指定精确到小数点后多少位，默认为7

==注： 在上述的两个函数中，如果delta指定了值，则first和second之间的差值必须≤delta==

3. assertGreater (first, second, msg = None)

验证first > second，否则fail

4. assertGreaterEqual (first, second, msg = None)

验证first ≥ second，否则fail

5. assertLess (first, second, msg = None)

验证first < second，否则fail

6. assertLessEqual (first, second, msg = None)

验证first ≤ second，否则fail

7. assertRegexpMatches (text, regexp, msg = None)

验证正则表达式regexp搜索==匹配==的文本text。 regexp：通常使用re.search()

8. assertNotRegexpMatches (text, regexp, msg = None)

验证正则表达式regexp搜索==不匹配==的文本text。 regexp：通常使用re.search()

完了

就这么多了，其他的请参考公众号前期所发布的系列文章。

用常规的方法解决问题，少炫技巧。

扫一扫关注微信公众号：