**Pyunit使用总结**

1. **介绍**

* 测试是一个贯穿于整个开发过程的连续过程，所以在python中我们使用pyunit这个测试框架，用它来比较输出结果和预期结果是否相同。程序员可能会按照自己的方法写出一些静态测试代码，但是这样写出来的测试代码杂乱无章，因此，我们使用pyunit测试框架来标准测试代码，编写动态测试代码。
* 静态方法和动态方法的比较：静态方法需要为每个测试方法编写一个测试类，费时枯燥，动态方法的测试类结构好，所有代码可以一个类中实现。

1. **使用方法**

* 我们所有的测试用例必须都得继承unittest
* 在pyunit测试框架运行测试用例时TestCase的setUp( ),runTest( ),tearDown( )依次执行。 注：这里的setUp( ),runTest( ),tearDown( )三个方法中的U 、T、 D需要大写。
* 每个测试方法中使用assertEqual检测运行结果是否与预期相符
* 执行伪代码：

Class TestCase(unittest.TestCase):

def setUp():

def tearDown():

def testA():

assertEqual(method，except)

def testB():

assertEqual(method，except)

def suite():#全局方法，使用TestSuite来组织测试方法，测试过程中用来确定有多少测试用例，需要返回TestSuite的对象

也可以直接定义一个TestSuite的子类，并在其初始化方法（\_\_init\_\_）中完成所有测试用例的添加：

class TestSuite(unittest.TestSuite):

|  |
| --- |
| def \_\_init\_\_(self):  unittest.TestSuite.\_\_init\_\_(self, map(TestCase, ("testSize"，"testResize")))这样只要在suite方法中该类的实例：  def suite():  return TestSuite() |
|  |

* Ok,到这里我们就要执行测试类了

if \_\_name\_\_==”\_\_main\_\_”:

runner = unittest.TextTestRunner()

runner.run(suite( ))

* 执行结果：

