# Python unittest模块实现单元测试

通过上一节《单元测试是什么？》我们对单元测试的概念有了基本的了解，我们知道 Django 的单元测试是基于 Python 的标准库模块 unittest 实现的。所以在本节我们将使用该模块编写测试用例代码，在单元测试过程中必须使用断言。unittest 单元测试框架中的 TestCase 类提供了很多断言方法，便于检验测试是否满足预期结果，并能在断言失败后抛出失败的原因。了解该模块的基本使用方法，从而对单元测试的测试流程有更深的理解。

断言(assert)，表示为一些布尔表达式编写代码时，开发人员总是会在某些特定点做出一些假设，来判断程序是否达到预期。

## 1. 编写测试用例代码

### 1) 编写待测功能函数

首先需要我们定义一个待测的功能函数，在 index 应用下新建 test.py 文件，定义一个传递分数就能够进行评级的功能函数，如下所示：

```
1. def score_grade(score):
2.     if not isinstance(score,int):
3.         raise TypeError("你应该输入一个整数类型")
4.     if not (0<=score<=100):
5.         raise Exception("你输入的整数范围应该在0-100之间")
6.     if 0<=score<60:
7.         return "D"
8.     elif 60<=score<=75:
9.         return 'C'
10.    elif 75<score<=85:
11.        return 'B'
12.    else:
13.        return 'A'
```

上面的代码就不做解释了，非常简单易懂。编写单元测试，需要从 unittest.TestCase 继承。使用 unittest 单元测试框架的时候，需要注意 unittest 规定只有以 test 开头的方法才是测试方法，所以我们使用的时候要注意命名规则，编写下面代码：

```
1. import unittest
2. class TestScore(unittest.TestCase):
3.     def test_exception(self):
```

```
        with self.assertRaises(TypeError):
            score_grade('x')
    def test_score(self):
        self.assertEqual(score_grade(86),'A')
        self.assertNotEqual(score_grade(86),'B')
        self.assertTrue(score_grade(86) is 'A')
```

我们对上述代码做一下解析：我们定义了一个测试类 TestScore，它继承自 unittest.TestCase 即测试用例类。可以这样理解，每一个继承 TestCase 类的子类里面实现的具体的方法（也就是以 test 开头的方法）都是一条测试用例。

### 2) unittest提供的断言方法

上面定义的 test_exception 与 test_score 都是测试用例，它们都使用了 unittest 提供的断言方法，从而得到最终的测试结果，常用的断言方法如下所示：

| 断言方法 | 含义 |
| --- | --- |
| assertEqual(first,second,msg=None) | 测试first == second，否则抛出断言异常信息msg； |
| assertNotEqual(first,second,msg = None) | 测试first!=second，否则抛异常信息msg； |
| assertTrue(expr,msg=None) | 测试表达式expr为True，否则抛出断言异常信息msg； |
| assertFalse(expr,msg=None) | 测试表达式expr为False，否则抛出断言异常信息msg； |
| assertIs(a,b,msg=None) | 测试a和b是同一对象，否则抛出断言异常信息msg； |
| assertIsNot(a,b,msg=None) | 测试a和b不是同一对象，否则抛出断言异常信息msg； |
| assertIsNone(expr,msg=None) | 测试表达式expr结果为None，否则抛出断言异常信息msg； |
| assertIsNotNone(expr,msg=None) | 测试表达式expr结果不为None,否则抛出断言信息异常； |
| assertIn(a,b,msg=None) | 测试a包含在b中，否则抛出断言异常信息msg； |

| | |
|---|---|
| assertIsInstance(obj,cls,msg=None) | 断言obj为cls类型，否则抛出断言异常信息msg； |
| assertRaisesRegexp(exc,r[,fun,*args,**kwds]) | 测试函数fun(*args,**kwds)抛出exc异常，同时可以用正则r去匹配异常信息exc，否则抛出断言异常； |
| assertRaises(exc[,fun,*args,**kwds]) | 测试函数fun(*args,**kwds)抛出exc异常，否则抛出断言异常； |

## 2. 执行单元测试的方法

我们可以使用 unittest 单元测试模块提供的方法来执行单元测试，如下所示：

```
python -m unittest test
```

如果只想执行 TestScore 类中定义的测试用例，可以使用如下方式：

```
python -m unittest test.TestScore
```

最终的执行结果如下所示：

```
C:¥Users¥Administrator¥Book¥BookStore>python -m unittest
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s
OK
```

若最终显示结果为 OK 表示测试成功，并且上面信号还包含了执行测试方法的个数和执行所需的时间，如果测试不通过呢？我们可以进行以下稍微的改动，将 test_score中的第一个断言更改为，如下所示：

```
1. self.assertEqual(score_grade(86),'B')
```

当我们再去执行测试的时候，就会得到如下结果：

```
C:¥Users¥Administrator¥Book¥BookStore>python -m unittest
.F
======================================================================
FAIL: test_score (index.test.TestScoreGrade)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:¥Users¥Administrator¥Book¥BookStore¥index¥test.py", line 24, in test_score
    self.assertEqual(score_grade(86),'B')
AssertionError: 'A' != 'B'
- A
+ B
----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

从返回结果可以得知其中有一个测试用例失败了，用例名称是 test_score，并且给出了失败的原因 AssertionError: 'A' != 'B'。

在下一节中，我们将完成如何针对 Django 项目编写单元测试代码，Django 对于单元测试环节做了哪些支持呢？在编写过程中又需要开发者注意哪些地方呢？这些知识点在下一节将逐一给大家揭晓。