# Python实验17 测试

## 一、实验介绍

编写测试检验应用程序所有不同的功能。每一个测试集中在一个关注点上验证结果是不是期望的。定期执行测试确保应用程序按预期的工作。当测试覆盖很大的时候，通过运行测试你就有自信确保修改点和新增点不会影响应用程序。

知识点

单元测试概念

使用 unittest 模块

测试用例的编写

异常测试

测试覆盖率概念

使用 coverage 模块

## 二、测试范围

如果可能的话，代码库中的所有代码都要测试。但这取决于开发者，如果写一个健壮性测试是不切实际的，你可以跳过它。就像 Nick Coghlan（Python 核心开发成员） 在访谈里面说的：有一个坚实可靠的测试套件，你可以做出大的改动，并确信外部可见行为保持不变。

## 三、单元测试

这里引用维基百科的介绍：

在计算机编程中，单元测试（英语：Unit Testing）又称为模块测试, 是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。程序单元是应用的最小可测试部件。在过程化编程中，一个单元就是单个程序、函数、过程等；对于面向对象编程，最小单元就是方法，包括基类（超类）、抽象类、或者派生类（子类）中的方法。

单元测试模块

在 Python 里我们有 unittest 这个模块来帮助我们进行单元测试。

阶乘计算程序

在这个例子中我们将写一个计算阶乘的程序 /home/shiyanlou/factorial.py：

import sys

def fact(n):

"""

阶乘函数

:arg n: 数字

:returns: n 的阶乘

"""

if n == 0:

return 1

return n \* fact(n -1)

def div(n):

"""

只是做除法

"""

res = 10 / n

return res

def main(n):

res = fact(n)

print(res)

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) > 1:

main(int(sys.argv[1]))

运行程序：

$ python3 factorial.py 5

3.1 第一个测试用例

测试哪个函数？

正如你所看到的， fact(n) 这个函数执行所有的计算，所以我们至少应该测试这个函数。

编辑 /home/shiyanlou/factorial\_test.py 文件，代码如下：

import unittest

from factorial import fact

class TestFactorial(unittest.TestCase):

"""

我们的基本测试类

"""

def test\_fact(self):

"""

实际测试

任何以 `test\_` 开头的方法都被视作测试用例

"""

res = fact(5)

self.assertEqual(res, 120)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

运行测试：

$ python3 factorial\_test.py

.

----------------------------------------------------------------------

Ran 1 test in 0.000s

OK

说明

我们首先导入了 unittest 模块，然后测试我们需要测试的函数。

测试用例是通过子类化 unittest.TestCase 创建的。

现在我们打开测试文件并且把 120 更改为 121，然后看看会发生什么？

3.2 各类 assert 语句

Method Checks that New in

assertEqual(a, b) a == b

assertNotEqual(a, b) a != b

assertTrue(x) bool(x) is True

assertFalse(x) bool(x) is False

assertIs(a, b) a is b 2.7

assertIsNot(a, b) a is not b 2.7

assertIsNone(x) x is None 2.7

assertIsNotNone(x) x is not None 2.7

assertIn(a, b) a in b 2.7

assertNotIn(a, b) a not in b 2.7

assertIsInstance(a, b) isinstance(a, b) 2.7

assertNotIsInstance(a, b) not isinstance(a, b) 2.7

3.3 异常测试

如果我们在 factorial.py 中调用 div(0)，我们能看到异常被抛出。

我们也能测试这些异常，就像这样：

self.assertRaises(ZeroDivisionError, div, 0)

完整代码：

import unittest

from factorial import fact, div

class TestFactorial(unittest.TestCase):

"""

我们的基本测试类

"""

def test\_fact(self):

"""

实际测试

任何以 `test\_` 开头的方法都被视作测试用例

"""

res = fact(5)

self.assertEqual(res, 120)

def test\_error(self):

"""

测试由运行时错误引发的异常

"""

self.assertRaises(ZeroDivisionError, div, 0)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

3.4 mounttab.py

mounttab.py 中只有一个 mount\_details() 函数，函数分析并打印挂载详细信息。

import os

def mount\_details():

"""

打印挂载详细信息

"""

if os.path.exists('/proc/mounts'):

fd = open('/proc/mounts')

for line in fd:

line = line.strip()

words = line.split()

print('{} on {} type {}'.format(words[0],words[1],words[2]), end=' ')

if len(words) > 5:

print('({})'.format(' '.join(words[3:-2])))

else:

print()

fd.close()

if \_\_name\_\_ == '\_\_main\_\_':

mount\_details()

重构 mounttab.py

现在我们在 mounttab2.py 中重构了上面的代码并且有一个我们能容易的测试的新函数 parse\_mounts()。

import os

def parse\_mounts():

"""

分析 /proc/mounts 并 返回元祖的列表

"""

result = []

if os.path.exists('/proc/mounts'):

fd = open('/proc/mounts')

for line in fd:

line = line.strip()

words = line.split()

if len(words) > 5:

res = (words[0],words[1],words[2],'({})'.format(' '.join(words[3:-2])))

else:

res = (words[0],words[1],words[2])

result.append(res)

fd.close()

return result

def mount\_details():

"""

打印挂载详细信息

"""

result = parse\_mounts()

for line in result:

if len(line) == 4:

print('{} on {} type {} {}'.format(\*line))

else:

print('{} on {} type {}'.format(\*line))

if \_\_name\_\_ == '\_\_main\_\_':

mount\_details()

同样我们测试代码，编写 mounttest.py 文件：

#!/usr/bin/env python

import unittest

from mounttab2 import parse\_mounts

class TestMount(unittest.TestCase):

"""

我们的基本测试类

"""

def test\_parsemount(self):

"""

实际测试

任何以 `test\_` 开头的方法都被视作测试用例

"""

result = parse\_mounts()

self.assertIsInstance(result, list)

self.assertIsInstance(result[0], tuple)

def test\_rootext4(self):

"""

测试找出根文件系统

"""

result = parse\_mounts()

for line in result:

if line[1] == '/' and line[2] != 'rootfs':

self.assertEqual(line[2], 'ext4')

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

运行程序

$ python3 mounttest.py

..

----------------------------------------------------------------------

Ran 2 tests in 0.001s

OK

3.5 测试覆盖率

测试覆盖率是找到代码库未经测试的部分的简单方法。它并不会告诉你的测试好不好。

在 Python 中我们已经有了一个不错的覆盖率工具来帮助我们。你可以在实验楼环境中安装它：

$ sudo pip3 install coverage

覆盖率示例

$ coverage3 run mounttest.py

..

----------------------------------------------------------------------

Ran 2 tests in 0.013s

OK

$ coverage3 report -m

Name Stmts Miss Cover Missing

--------------------------------------------

mounttab2.py 22 7 68% 16, 25-30, 34

mounttest.py 14 0 100%

--------------------------------------------

TOTAL 36 7 81%

我们还可以使用下面的命令以 HTML 文件的形式输出覆盖率结果，然后在浏览器中查看它。

$ coverage3 html

## 四、总结

知识点回顾：

单元测试概念

使用 unittest 模块

测试用例的编写

异常测试

测试覆盖率概念

使用 coverage 模块

本实验了解了什么是单元测试，unittest 模块怎么用，测试用例怎么写。以及最后我们使用第三方模块 coverage 进行了覆盖率测试。

在实际生产环境中，测试环节是非常重要的的一环，即便志不在测试工程师，但以后的趋势就是 DevOps，所以掌握良好的测试技能也是很有用的。