14.1 测试stdout输出¶

问题¶

你的程序中有个方法会输出到标准输出中（sys.stdout）。也就是说它会将文本打印到屏幕上面。 你想写个测试来证明它，给定一个输入，相应的输出能正常显示出来。

解决方案¶

使用 `unittest.mock` 模块中的 `patch()` 函数， 使用起来非常简单，可以为单个测试模拟 `sys.stdout` 然后回滚， 并且不产生大量的临时变量或在测试用例直接暴露状态变量。

作为一个例子，我们在 `mymodule` 模块中定义如下一个函数：

```
# mymodule.py

def urlprint(protocol, host, domain):
    url = '{}://{}.{}'.format(protocol, host, domain)
    print(url)
```

默认情况下内置的 `print` 函数会将输出发送到 `sys.stdout` 。 为了测试输出真的在那里，你可以使用一个替身对象来模拟它，然后使用断言来确认结果。 使用 `unittest.mock` 模块的 `patch()` 方法可以很方便的在测试运行的上下文中替换对象，并且当测试完成时候自动返回它们的原有状态。下面是对 `mymodule` 模块的测试代码：

```
from io import StringIO
from unittest import TestCase
from unittest.mock import patch
import mymodule

class TestURLPrint(TestCase):
    def test_url_gets_to_stdout(self):
        protocol = 'http'
        host = 'www'
        domain = 'example.com'
        expected_url = '{}://{}.{}\n'.format(protocol, host, domain)

        with patch('sys.stdout', new=StringIO()) as fake_out:
            mymodule.urlprint(protocol, host, domain)
            self.assertEqual(fake_out.getvalue(), expected_url)
```

讨论¶

`urlprint()` 函数接受三个参数，测试方法开始会先设置每一个参数的值。 `expected_url` 变量被设置成包含期望的输出的字符串。

`unittest.mock.patch()` 函数被用作一个上下文管理器，使用 `StringIO` 对象来代替 `sys.stdout` . `fake_out` 变量是在该进程中被创建的模拟对象。 在with语句中使用它可以执行各种检查。当with语句结束时， `patch` 会将所有东西恢复到测试开始前的状态。 有一点需要注意的是某些对Python的C扩展可能会忽略掉 `sys.stdout` 的配置而直接写入到标准输出中。 限于篇幅，本节不会涉及到这方面的讲解，它适用于纯Python代码。 如果你真的需要在C扩展中捕获I/O，你可以先打开一个临时文件，然后将标准输出重定向到该文件中。 更多关于捕获以字符串形式捕获I/O和 `StringIO` 对象请参阅5.6小节。