14.3 在单元测试中测试异常情况¶

问题¶

你想写个测试用例来准确的判断某个异常是否被抛出。

解决方案¶

对于异常的测试可使用 `assertRaises()` 方法。 例如，如果你想测试某个函数抛出了 `ValueError` 异常，像下面这样写：

```
import unittest

# A simple function to illustrate
def parse_int(s):
    return int(s)

class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        self.assertRaises(ValueError, parse_int, 'N/A')
```

如果你想测试异常的具体值，需要用到另外一种方法：

```
import errno

class TestIO(unittest.TestCase):
    def test_file_not_found(self):
        try:
            f = open('/file/not/found')
        except IOError as e:
            self.assertEqual(e.errno, errno.ENOENT)

        else:
            self.fail('IOError not raised')
```

讨论¶

`assertRaises()` 方法为测试异常存在性提供了一个简便方法。 一个常见的陷阱是手动去进行异常检测。比如：

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        try:
            r = parse_int('N/A')
        except ValueError as e:
            self.assertEqual(type(e), ValueError)
```

这种方法的问题在于它很容易遗漏其他情况，比如没有任何异常抛出的时候。 那么你还得需要增加另外的检测过程，如下面这样：

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        try:
            r = parse_int('N/A')
        except ValueError as e:
            self.assertEqual(type(e), ValueError)
        else:
            self.fail('ValueError not raised')
```

`assertRaises()` 方法会处理所有细节，因此你应该使用它。

`assertRaises()` 的一个缺点是它测不了异常具体的值是多少。 为了测试异常值，可以使用 `assertRaisesRegex()` 方法， 它可同时测试异常的存在以及通过正则式匹配异常的字符串表示。例如：

```
class TestConversion(unittest.TestCase):
```

```
    def test_bad_int(self):
        self.assertRaisesRegex(ValueError, 'invalid literal .*',
                                       parse_int, 'N/A')
```

`assertRaises()` 和 `assertRaisesRegex()` 还有一个容易忽略的地方就是它们还能被当做上下文管理器使用：

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        with self.assertRaisesRegex(ValueError, 'invalid literal .*'):
            r = parse_int('N/A')
```

但你的测试涉及到多个执行步骤的时候这种方法就很有用了。