14.4 将测试输出用日志记录到文件中¶

问题¶

你希望将单元测试的输出写到到某个文件中去，而不是打印到标准输出。

解决方案¶

运行单元测试一个常见技术就是在测试文件底部加入下面这段代码片段：

```
import unittest

class MyTest(unittest.TestCase):
    pass

if __name__ == '__main__':
    unittest.main()
```

这样的话测试文件就是可执行的，并且会将运行测试的结果打印到标准输出上。 如果你想重定向输出，就需要像下面这样修改 `main()` 函数：

```
import sys

def main(out=sys.stderr, verbosity=2):
    loader = unittest.TestLoader()
    suite = loader.loadTestsFromModule(sys.modules[__name__])
    unittest.TextTestRunner(out,verbosity=verbosity).run(suite)

if __name__ == '__main__':
    with open('testing.out', 'w') as f:
        main(f)
```

讨论¶

本节感兴趣的部分并不是将测试结果重定向到一个文件中， 而是通过这样做向你展示了 `unittest` 模块中一些值得关注的内部工作原理。

`unittest` 模块首先会组装一个测试套件。 这个测试套件包含了你定义的各种方法。一旦套件组装完成，它所包含的测试就可以被执行了。

这两步是分开的，`unittest.TestLoader` 实例被用来组装测试套件。 `loadTestsFromModule()` 是它定义的方法之一，用来收集测试用例。 它会为 `TestCase` 类扫描某个模块并将其中的测试方法提取出来。 如果你想进行细粒度的控制，可以使用 `loadTestsFromTestCase()` 方法来从某个继承TestCase的类中提取测试方法。 `TextTestRunner` 类是一个测试运行类的例子， 这个类的主要用途是执行某个测试套件中包含的测试方法。 这个类跟执行 `unittest.main()` 函数所使用的测试运行器是一样的。 不过，我们在这里对它进行了一些列底层配置，包括输出文件和提升级别。 尽管本节例子代码很少，但是能指导你如何对 `unittest` 框架进行更进一步的自定义。 要想自定义测试套件的装配方式，你可以对 `TestLoader` 类执行更多的操作。 为了自定义测试运行，你可以构造一个自己的测试运行类来模拟 `TextTestRunner` 的功能。 而这些已经超出了本节的范围。`unittest` 模块的文档对底层实现原理有更深入的讲解，可以去看看。