**web自动化测试应该怎么做-unitest提升**

单元测试的重要性就不多说了，可恶的是python中有太多的单元测试框架和自动化测试工具，什么unittest, testtools, subunit, coverage, testrepository, nose, mox, mock, fixtures, discover，再加上setuptools, distutils等等这些，先不说如何写单元测试，光是怎么运行单元测试就有N多种方法，再因为它是测试而非功能，是很多人没兴趣触及的东西。但是作为一个优秀的测试人员，写好测试代码可以彰显你的实力。如此多的框架和工具，很容易让人困惑，困惑的原因是因为并没有理解它的基本原理，如果一些基本的概念都不清楚，怎么能够写出思路清晰的测试代码？

今天的主题就是unittest，作为标准python中的一个模块，是其它框架和工具的基础，参考资料是它的官方文档：http://docs.python.org/2.7/library/unittest.html和源代码，文档已经写的非常好了，我在这里记录的主要是它的一些重要概念、关键点以及可能会碰到的一些坑，目的在于对unittest加深理解，而不是停留在泛泛的表面层上。

unittest是一个python版本的junit，junit是java中的单元测试框架，对java的单元测试，有一句话很贴切：Keep the bar green，相信使用eclipse写过java单元测试的都心领神会。unittest实现了很多junit中的概念，比如我们非常熟悉的test case, test suite等，总之，原理都是相通的，只是用不同的语言表达出来。

在文档的开篇就介绍了unittest中的4个重要的概念：test fixture, test case, test suite, test runner，我觉得只有理解了这几个概念，才能真正的理解单元测试的基本原理，下面就主要围绕这几个概念来展开这篇文章。

首先通过查看unittest的源码，来看一下这几个概念，以及他们之间的关系，他们是如何在一起工作的，其静态类图如下：



一个TestCase的实例就是一个测试用例。什么是测试用例呢？就是一个完整的测试流程，包括测试前准备环境的搭建(setup)，执行测试代码(run)，以及测试后环境的还原(tearDown)。元测试(unit test)的本质也就在这里，一个测试用例是一个完整的测试单元，通过运行这个测试单元，可以对某一个问题进行验证。

而多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。

TestLoader是用来加载TestCase到TestSuite中的，其中有几个loadTestsFrom\_\_()方法，就是从各个地方寻找TestCase，创建它们的实例，然后add到TestSuite中，再返回一个TestSuite实例。

TextTestRunner是来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的run(result)方法。

测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

这样整个流程就清楚了，首先是要写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中，整个过程集成在unittest.main模块中。

现在已经涉及到了test case, test suite, test runner这三个概念了，还有test fixture没有提到，那什么是test fixture呢？？在TestCase的docstring中有这样一段话：

Test authors should subclass TestCase for their own tests. Construction and deconstruction of the test's environment ('fixture') can be implemented by overriding the 'setUp' and 'tearDown' methods respectively.

可见，对一个测试用例环境的搭建和销毁，是一个fixture，通过覆盖TestCase的setUp()和tearDown()方法来实现。这个有什么用呢？比如说在这个测试用例中需要访问数据库，那么可以在setUp()中建立数据库连接以及进行一些初始化，在tearDown()中清除在数据库中产生的数据，然后关闭连接。注意tearDown的过程很重要，要为以后的TestCase留下一个干净的环境。关于fixture，还有一个专门的库函数叫做fixtures，功能更加强大，以后会介绍到。

至此，概念和流程基本清楚了，下面通过简单的例子再来实践一下，就拿unittest文档上的例子吧：



TestSequenceFunctions继承自unittest.TestCase，重写了setUp()方法，并且定义了三个以'test'开头的方法，那这个TestSequenceFunctions类到底是个什么呢？它是一个测试用例，还是三个测试用例？说是三个测试用例的话，它本身继承自TestCase，说是一个测试用例的话，里面又有三个test\_\*()方法，明显是三个测试用例。其实，我们只要看一些TestLoader是如何加载测试用例的，就一清二楚了，在loader.TestLoader类中有一个loadTestsFromTestCase()方法：



getTestCaseNames()是从TestCase这个类中找所有以“test”开头的方法，然后注意第9行，在构造TestSuite对象时，其参数使用了一个map方法，即对testCaseNames中的每一个元素，使用testCaseClass为其构造对象，其结果是一个TestCase的对象集合，可以用下面的代码来分步说明：



可见，对每一个以test开头的方法，都为其构建了一个TestCase对象，值得注意的是，如果没有定义test开头的方法，而是将测试代码写到了一个名为runTest的方法中，那么会为该runTest方法构建TestCase对象，如果定义了test开头的方法，就会忽略runTest方法。

到此，基本就清楚了，每一个以test开头的方法，都会为其构建TestCase对象，也就是说TestSequenceFunctions类中其实定义了三个TestCase，之所以写成这样，是为了方便，因为这几个测试用例的fixture是相同的，如果每一个测试用例单独写成一个TestCase的话，会有很多的冗余代码。