一、测试原则

Java语言是一个支持面向对象的语言，通常情况下我们可以将程序的一个单元看成是一个独立的类，因此进行单元测试的重点就是对这些类进行测试。

1、不需要测试get和set这样的行为

2、一个方法至少需要测试一次

二、测试步骤

1、判断组件的功能：通过定义应用的整体需求，然后将系统划分成几个对象，我们需要对组件的基本功能十分清楚。因此，J2EE单元测试实际上也属于设计过程的一部分。

2、设计组件行为：依据所处理的过程，可以通过一个正规或者非正规的过程实现组件行为的设计，我们可以使用UML或者其他文档视图来设计组件行为，从而为组件的测试打下基础。

3、编写单元测试程序（或测试用例）确认组件行为：这个阶段，应该假定组件的编码已经结束而组件工作正常，我们需要编写单元测试程序来确定其功能是否和预定义的功能相同，测试程序需要考虑所有正常和意外的输入，以及特定的方法能产生的溢出。

4、编写组件并执行测试：首先，创建类及其所对应的方法标识，然后遍历每个测试实例，为其编写相应代码使其顺利通过，然后返回测试。继续这个过程直至所有实例通过。此时，停止编码。

5、测试替代品：对组件行为的其他方式进行考虑，设计更周全的输入或者其他错误条件，编写测试用例来捕获这些条件，然后修改代码使得测试通过。

6、重整代码：如果有必要，在编码结束时，对代码进行重整和优化，改动后，返回单元测试并确认测试通过。

7、当组件有新的行为时，编写新的测试用例：每次在组件中发现故障。编写一个测试实例重复这个故障，然后修改组件以保证测试实例通过。同样，当发现新的需求或已有的需求改变时，编写或修改测试实例以响应此改变，然后修改代码。

8、代码修改,返回所有的测试：每次代码修改时，返回所有的测试以确保没有打乱代码。

三、使用junit进行单元测试

摘要：1、每次只对一个对象进行UT测试；2、给测试方法起个好名字；3、明确写出出错原因；4、一个UT测试方法只应该测试一种情况；5、测试任何可能的错误；6、让你的测试帮助改善你的代码；7、一样的包，不同的位置……

以下是对JUnit实践的一个很好的总结，信息来源于一些比较权威的JUnit书籍和网上资料。这里集合如下：

1、每次只对一个对象进行UT测试（unit-test one object at a time）。这样能使你尽快发现问题，而不被各个对象之间的复杂关系所迷惑。

2、给测试方法起个好名字（choose meaningful test method names）。应该是用形如testXXXYYY()，这样的格式来命名你的测试方法。前缀test是Junit查找测试方法的依据，XXX应该是你测试 的方法名，YYY应该是你测试的状态。当然如果你只有一种状态需要测试可以直接命名为testXXX()。

3、明确写出出错原因（explain the failure reason in assert calls）。在使用assertTrue,assertFalse,assertNotNull,assertNull方法时，应该将可能的错误的描述 字符串，以第一个参数传入相应的方法。这样你可以迅速的找出出错原因。

4、一个UT测试方法只应该测试一种情况（one unit test equals one testMethod）。一个方法中的多次测试，只会混乱你的测试目的。

5、测试任何可能的错误（test anything that could possibly fail）。你的测试代码不是为了证明你是对的，而是为了证明你没有错。因此对测试的范围要全面，比如边界值、正常值、错误值；对代码可能出现的问题要全 面预测。

6、让你的测试帮助改善你的代码（let the test improve the code）。测试代码永远是我们代码的第一个用户，所以不仅让他帮组我们发现Bug,还要帮组我们改善我们的设计，就是有名的测试驱动开发（Test-Driven Development，TDD）。

7、一样的包，不同的位置（same package, separate directories）。测试的代码和被测试的代码应该放到不同的文件夹中，建议使用这种目录 src/java/代码 src/test/测试代码。 这样可以让两份代码使用一样的包结构，但是放在不同的目录下。

8、关于setup与teardown

a) 不要用TestCase的构造函数初始化Fixture，而要用setUp()和tearDown()方法。

b) 在setUp和tearDown中的代码不应该是与测试方法相关的，而应该是全局相关的。如：针对与测试方法都要用到的数据库链接等等。

c) 当继承一个测试类时，记得调用父类的setUp()和tearDown()方法。

9、不要在mock object中牵扯到业务逻辑（don’t write business logic in mock objects）。

10、只对可能产生错误的地方进行测试（only test what can possibly break）。如：一个类中频繁改动的函数。对于那些仅仅只含有getter/setter的类，如果是由IDE（如Eclipse）产生的，则可不测； 如果是人工写，那么最好测试一下。

11、尽量不要依赖或假定测试运行的顺序，因为JUnit利用Vector保存测试方法。所以不同的平台会按不同的顺序从Vector中取出测试方法。

12、避免编写有副作用的TestCase，你要确信保持你的测试方法之间是独立的。

13、将测试代码和工作代码放在一起，一边同步编译和更新（使用Ant中有支持junit的task）。

14、确保测试与时间无关，不要依赖使用过期的数据进行测试。导致在随后的维护过程中很难重现测试。

15、如果你编写的软件面向国际市场，编写测试时要考虑国际化的因素。不要仅用母语的Locale进行测试。

16、尽可能地利用JUnit提供地assert/fail方法以及异常处理的方法，可以使代码更为简洁。

17、测试要尽可能地小，执行速度快。