JUnit 框架是 Java 语言单元测试当前的一站式解决方案。这个框架值得称赞，因为它把测试驱动的开发思想介绍给 Java 开发人员并教给他们如何有效地编写单元测试。但是，在过去的几年中，JUnit 的改进不大；所以，为当今复杂的环境编写测试已经变成一个越来越困难的任务，即 JUnit 必须与其他一些补充性测试框架集成起来。而 TestNG 是一个测试 Java 应用程序的新框架。TestNG 不仅确实强大、创新、可扩展、灵活，它还展示了 Java Annotations（JDK 5.0 中的重大新特性）的有趣应用。

TestNG 的创造者是 Cedric Beust，他在 Java 编程领域非常出名，是 EJB 3 专家组的成员，也是其他一些流行的开源项目（例如 EJBGen 和 Doclipse）的创造者。

示例测试代码：

package example1;  
  
import org.testng.annotations.\*;  
  
public class SimpleTest {  
  
 @BeforeClass  
 public void setUp() {  
   // code that will be invoked when this test is instantiated  
 }  
  
 @Test(groups = { "fast" })  
 public void aFastTest() {  
   System.out.println("Fast test");  
 }  
  
 @Test(groups = { "slow" })  
 public void aSlowTest() {  
    System.out.println("Slow test");  
 }  
  
}

    TestNG需要有自己的配置文件，最方便的办法即是从eclipse上直接下载一下插件。直接下载跟插件下载的地址都可以在<http://testng.org/doc/download.html>上找到。

    http://www.mkyong.com/tutorials/testng-tutorials/ 这个教程写得很好，挺通俗易懂的。

    大致摘抄记录一下：

    1. 基本用法。

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. import java.util.\*;
2. import org.testng.Assert;
3. import org.testng.annotations.\*;
5. public class TestNGTest1 {
7. private Collection collection;
9. @BeforeClass
10. public void oneTimeSetUp() {
11. // one-time initialization code
12. System.out.println("@BeforeClass - oneTimeSetUp");
13. }
15. @AfterClass
16. public void oneTimeTearDown() {
17. // one-time cleanup code
18. System.out.println("@AfterClass - oneTimeTearDown");
19. }
21. @BeforeMethod
22. public void setUp() {
23. collection = new ArrayList();
24. System.out.println("@BeforeMethod - setUp");
25. }
27. @AfterMethod
28. public void tearDown() {
29. collection.clear();
30. System.out.println("@AfterMethod - tearDown");
31. }
33. @Test
34. public void testEmptyCollection() {
35. Assert.assertEquals(collection.isEmpty(),true);
36. System.out.println("@Test - testEmptyCollection");
37. }
39. @Test
40. public void testOneItemCollection() {
41. collection.add("itemA");
42. Assert.assertEquals(collection.size(),1);
43. System.out.println("@Test - testOneItemCollection");
44. }
45. }

    @BeforeClass @AfterClass 等这些从字面上都可以很好理解，Class是整个测试类运行时的前后，而Method则在每个测试方法被调用前都会被调用。

    所以这一段代码执行后的结果如下：

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. @BeforeClass - oneTimeSetUp
2. @BeforeMethod - setUp
3. @Test - testEmptyCollection
4. @AfterMethod - tearDown
5. @BeforeMethod - setUp
6. @Test - testOneItemCollection
7. @AfterMethod - tearDown
8. @AfterClass - oneTimeTearDown
9. PASSED: testEmptyCollection
10. PASSED: testOneItemCollection

    2.测试预期的异常

    可以检测某一方法检测到某一异常时是否能按预期地抛出。

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. import org.testng.annotations.\*;
3. /\*\*
4. \* TestNG Expected Exception Test
5. \* @author mkyong
6. \*
7. \*/
8. public class TestNGTest2 {
10. @Test(expectedExceptions = ArithmeticException.class)
11. public void divisionWithException() {
12. int i = 1/0;
13. }
15. }

    在这一示例中，divisionWithException()将会抛出一个ArithmetricException的预期异常，这个单元测试也将顺利通过。

    3. 忽略某一测试方法

    TestNG是通过直接在方法上加标注的方式来进行测试，而这里也可以设置某个测试方法不工作。可以通过如下方式：

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. import org.testng.annotations.\*;
3. /\*\*
4. \* TestNG Ignore Test
5. \* @author mkyong
6. \*
7. \*/
8. public class TestNGTest3 {
10. @Test(enabled=false)
11. public void divisionWithException() {
12. System.out.println("Method is not ready yet");
13. }
15. }

    4. 时限测试

    可以设置一个特定时长的限制（以毫秒ms为单位），一旦测试的内容运行超过了该 时间长度，那么将会终止，同时标记为failed。

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. import org.testng.annotations.\*;
3. /\*\*
4. \* TestNG TimeOut Test
5. \* @author mkyong
6. \*
7. \*/
8. public class TestNGTest4 {
10. @Test(timeOut = 1000)
11. public void infinity() {
12. while (true);
13. }
15. }

    运行后将会有如下的提示：

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. FAILED: infinity
2. org.testng.internal.thread.ThreadTimeoutException:
3. Method public void TestNGTest4.infinity() didn't finish within the time-out 1000
4. ... Removed 18 stack frames

    5. 测试套件（Suite Test）

   即是将一些单元测试用例绑定并一起运行。定义suite test是在xml文件中，参见如下文件，表示将TestNG1和TestNG2一起执行。

**[html]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. <!DOCTYPE suite SYSTEM "http://beust.com/testng/testng-1.0.dtd" >
2. <suite name="My test suite">
3. <test name="testing">
4. <classes>
5. <class name="TestNG1" />
6. <class name="TestNG2" />
7. </classes>
8. </test>
9. </suite>

   6. 依赖测试（Dependency Test）

    这也可以用于解决如何决定一个测试类中各测试方法调用顺序的问题，可以指定某一个方法依赖于另一个方法的预先执行。

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. import org.testng.annotations.\*;
3. /\*\*
4. \* TestNG Dependency Test
5. \* @author mkyong
6. \*
7. \*/
8. public class TestNGTest7 {
10. @Test
11. public void method1() {
12. System.out.println("This is method 1");
13. }
15. @Test(dependsOnMethods={"method1"})
16. public void method2() {
17. System.out.println("This is method 2");
18. }

21. }

    运行的结果为：

**[java]** [view plaincopy](http://blog.csdn.net/bigapplestar/article/details/7300137)

1. PASSED: method1
2. PASSED: method2

method仅当在method1方法执行成功的前提下才会运行。

    这里翻译得顺序有些调整，关于参数传递的6,7小节下次再翻译，还没有具体测试过，同时也还有一些疑问，若传递的是数据，类等，试试再来整理出来了。