JUnit4详细教程

因jdk5中的新特性，JUnit4也因此有了很大的改变。确切的说，Junit4简直就不是3的扩展版本，而是一个全新的测试框架。下面详细介绍JUnit4的使用方法

1.测试方法：

在junit4之前，测试类通过继承TestCase类，并使用命名约束来定位测试，测试方法必须以“test”开头。Junit4中使用注释类识别：@Test,也不必约束测试方法的名字。当然，TestCase类仍然可以工作，只不过不用这么繁琐而已。

Junit中还因JDK5而增加了一项新特性，静态导入（static import）。

2.固件测试

所谓固件测试(Fixture)，就是测试运行运行程序（test runner）会在测试方法之前自动初始化、和回收资源的工作。JUnit4之前是通过setUp、TearDown方法完成。在Junit4中，仍然可以在每个测试方法运行之前初始化字段，和配置环境，当然也是通过注释完成。Junit4中，通过@befroe替代setUp方法；@After替代tearDown方法。在一个测试类中，甚至可以使用多个@Before来注释多个方法，这些方法都是在每个测试之前运行。说明一点，@Before是在每个测试方法运行前均初始化一次，同理@Ater是在每个测试方法运行完毕后，均匀性一次就是说，经这两个注释的初始化和注销，可以保证各个测试之间的独立性而互不干扰，他的缺点是效率低。另外，不需要在朝类中显示调用初始化和清除方法，只要他们不被覆盖，测试运行程序将根据需要自动调用这些方法。超类中的@Before方法在自来的@Before方法之前调用（与构造函数调用顺序一致），@After方法是子类中的在超类之前运行。

在JUnit4中加入了一项新特性。加入了两个注释：@BeforeClass和@AfterClass,使用这两个注释的方法，在该测试类中，的测试方法之前、后各运行一次，而不是按照方法各运行一次。对于消耗很的啊的资源，可以使用这两个注释。

3.异常测试

因为使用了注释特性，JUnit4测试异常非常的简单和明了。通过对@Test传入expected参数值，即可测试异常。通过传入异常类后，测试类如果没有抛出异常或者抛出一个不同的异常，本测试方法就将失败。见代码：

/\*\*

\*

\*/

package cn.hrmzone.junit;

import java.io.File;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.io.SAXReader;

import org.junit.Before;

import org.junit.Ignore;

import org.junit.Test;

/\*\*

\* @author hrmzone.cn

\*2010-10-1

\*/

public class ExceptionTest {

File f;

Document doc;

@Before

public void init() {

f=new File("output"+File.separator+"test.xml");

}

@Ignore("not run")

@Test(expected=DocumentException.class)

public void read() throws DocumentException {

SAXReader reader=new SAXReader();

doc=reader.read(f);

}

@Test(expected=ArithmeticException.class)

public void divideZero() {

int n=2/0;

}

}

在第二个测试方法中，用0做除数，将会抛出ArithmeticException异常，所以对expected参数传入该类。测试抛出此异常，说明本次测试成功。如图：

测试成功后，方法前会出现一个小沟，大家可能注意到了，read（）方法，代码中有一个奇怪的@Ignore注释，测试后也会出现一个标识。@Ignore注释表示忽略注释，运势测试类时，被@Ignore注释的方法将不会被测试，所以运行测试类后，会出行一个奇怪的标识。

4.超时测试

通过在@Test注释中，为timeout参数指定时间值，即可进行超时测试。如果测试运行时间超过指定的毫秒数，则测试失败。超时测试对网络链接类的非常重要，通过timeout进行超时测试，简单异常。如下例子：

/\*\*

\*

\*/

package cn.hrmzone.junit;

import static org.junit.Assert.assertTrue;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.junit.Before;

import org.junit.Test;

/\*\*

\* @author hrmzone.cn

\*2010-10-1

\*/

public class RegularExpressionTest {

private String dateReg;

private Pattern pattern;

@Before

public void init() {

dateReg="^\\d{4}(\\-\\d{1,2}){2}";

pattern=Pattern.compile(dateReg);

}

// timeout测试是指在指定时间内就正确

@Test(timeout=1)

public void verifyReg() {

Matcher matcher=pattern.matcher("2010-10-2");

boolean isValid=matcher.matches();

// 静态导入功能

assertTrue("pattern is not match",isValid);

}

}

5.测试运行器

测试运行器：JUNit中所有的测试方法都是由它负责执行。JUnit为单元测试提供了默认的测试运行器，但是没有限制必须使用默认的运行器。自己定制的测试运行器必须继承自org.junit.runner.Runner。而且还可以为每一个测试类指定某个运行器：@Runwith(CustomTestRunner.class).在JUnit中，有两个高级特性会需要自定义运行器。

6.测试套件

JUnit4中最显著的特性是没有套件(套件机制用于将测试从逻辑上的分组并将这这些测试作为一个单元测试来运行)。为了替代老版本的套件测试，套件被两个新注释代替：@RunWith、@SuteClasses。通过@RunWith指定一个特殊的运行器：Suite.class套件运行器，并通过@SuiteClasses注释，将需要进行测试的类列表作为参数传入。

使用方法为：

a.创建一个空类作为测试套件的入口

b.将@RunWith、@SuiteClasses注释修饰这个空类；

c.把Suite.class作为参数传入@RunWith注释，以提示JUnit将此类指定为运行器；

d.将需要测试的类组成数组作为@SuiteClasses的参数。

注意：这个空类必须使用public修饰符，而且存在public的无参构造函数（类的默认构造函数即可）。测试代码如下：将先前的两个测试类：RegularExpressionTest,Exception作为一个测试套件进行测试：

/\*\*

\*

\*/

package cn.hrmzone.junit;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

/\*\*

\* @author hrmzone.cn

\*2010-10-2

\*/

@RunWith(Suite.class)

@SuiteClasses({RegularExpressionTest.class,ExceptionTest.class})

public class SuiteTest {

}

@SuiteClasses参数为需要测试类的数组，可以使用{}，将两个测试类传入。

测试套件类为空类，需要一个无参的public构造函数，使用默认构造函数即可。

运行结果如图，可以带到作为参数的两个测试类均运行，与单独测试一致。图2

7.参数化测试

为测试程序健壮性，可能需要模拟不同的参数对方法进行测试，如果在为每一个类型的参数创建一个测试方法，呵呵，人都疯掉了。幸好有参数化测试出现了。它能够创建由参数值供给的通用测试，从而为每个参数都运行一次，而不必要创建多个测试方法。注：测试方法（@Test注释的方法）是不能有参数的。

参数化测试编写流程如下：

a.为参数化测试类用@RunWith注释指定特殊的运行器：Parameterized.class；

b.在测试类中声明几个变量，分别用于存储期望值和测试用的数据，并创建一个使用者几个参数的构造函数；

c.创建一个静态（static）测试数据供给（feed）方法，其返回类型为Collection，并用@Parameter注释以修饰；

d.编写测试方法（用@Test注释）。

测试示例代码如下：

/\*\*

\*

\*/

package cn.hrmzone.junit;

import static org.junit.Assert.assertEquals;

import java.util.Arrays;

import java.util.Collection;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

/\*\*

\* @author hrmzone.cn

\*2010-10-2

\*/

@RunWith(Parameterized.class)

public class ParameterTest {

private String dateReg;

private Pattern pattern;

// 数据成员变量

private String phrase;

private boolean match;

// 使用数据的构造函数

public ParameterTest(String phrase, boolean match) {

super();

this.phrase = phrase;

this.match = match;

}

@Before

public void init() {

dateReg="^\\d{4}(\\-\\d{1,2}){2}";

pattern=Pattern.compile(dateReg);

}

// 测试方法

@Test

public void verifyDate() {

Matcher matcher=pattern.matcher(phrase);

boolean isValid=matcher.matches();

assertEquals("Pattern don't validate the data format",isValid,match);

}

// 数据供给方法（静态，用@Parameter注释，返回类型为Collection

@Parameters

public static Collection dateFeed() {

return Arrays.asList(new Object[][] {

{"2010-1-2",true},

{"2010-10-2",true},

{"2010-123-1",false},

{"2010-12-45",false}

});

}

}

运行结果如图：

因为代码只测试日期格式，并未测试日期范围的正确性，2010-12-45这数值测试与期望值不符。

8.数组断言

JUnit4中添加了一个用于比较数组的新断言（Assert），这样不必使用迭代比较数组中的条目。

public static void assertEquals(Object[] expected, Object[] actual)

public static void assertEquals(String message, Object[] expected,Object[] actual)

JUnit4是向前兼容的，在JUnit4中依然可以使用JUnit3的测试代码，而不需要做任何改动。

JUnit测试框架将测试更加便捷和容易，编写测试代码也是简单、明了，但功能依然强大。