**TestNG介绍**

测试是检查应用程序的功能的过程是否按要求工作，以确保在开发层面，单元测试成图片。单元测试是单一实体（类或方法）的测试。单元测试是非常必要的，每一个软件公司向他们的客户提供高质量的产品。

JUnit 带动开发人员了解测试的实用性，尤其是单元测试的时候比任何其他测试框架。凭借一个相当简单，务实，严谨的架构，[JUnit](http://www.yiibai.com/html/junit/)已经能够“感染”了一大批开发人员。 JUnit的特点，可以看看Junit 特点。

其中JUnit缺点：

* 最初的设计，使用于单元测试，现在只用于各种测试
* 不能依赖测试
* 配置控制欠佳（安装/拆卸）
* 侵入性（强制扩展类，并以某种方式命名方法）
* 静态编程模型（不必要的重新编译）
* 不同的适合管理复杂项目中的测试可以是非常棘手.

**TestNG是什么?**

TestNG按照其文档的定义是：

TestNG是一个测试框架，其灵感来自JUnit和NUnit的，但引入了一些新的功能，使其功能更强大，使用更方便。

TestNG是一个开源自动化测试框架;TestNG表示下一代。 TestNG是类似于JUnit（特别是JUnit 4），但它不是一个JUnit扩展。它的灵感来源于JUnit。它的目的是优于JUnit的，尤其是当测试集成的类。 TestNG的创造者是Cedric Beust（塞德里克·博伊斯特）

TestNG消除了大部分的旧框架的限制，使开发人员能够编写更加灵活和强大的测试。 因为它在很大程度上借鉴了Java注解（JDK5.0引入的）来定义的测试，它也可以告诉你如何使用这个新功能在真实的Java语言生产环境中。

**TestNG的特点**

* 注解
* TestNG使用Java和面向对象的功能
* 支持综合类测试（例如，默认情况下，没有必要创建一个新的测试每个测试方法的类的实例）
* 独立的编译时间测试代码运行时配置/数据信息
* 灵活的运行时配置
* 主要介绍“测试组”。当编译测试，只要问TestNG运行所有的“前端”的测试，或“快”，“慢”，“数据库”等
* 支持依赖测试方法，并行测试，负载测试，局部故障
* 灵活的插件API
* 支持多线程测试

**TestNG编写测试**

编写[TestNG](http://www.yiibai.com/html/testng)测试基本上包括以下步骤：

* 测试和编写业务逻辑，在代码中插入TestNG的注解..
* 添加一个testng.xml文件或build.xml中在测试信息（例如类名，您想要运行的组，等..）
* 运行 TestNG.

在这里，我们将看到一个完整的例子了TestNG测试使用POJO类，业务逻辑类，将通过TestNG的运行测试XML。

创建 EmployeeDetails.java 在 C:\ > TestNG\_WORKSPACE 是一个 POJO 类.

public class EmployeeDetails {

private String name;

private double monthlySalary;

private int age;

/\*\*

\* @return the name

\*/

public String getName() {

return name;

}

/\*\*

\* @param name the name to set

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return the monthlySalary

\*/

public double getMonthlySalary() {

return monthlySalary;

}

/\*\*

\* @param monthlySalary the monthlySalary to set

\*/

public void setMonthlySalary(double monthlySalary) {

this.monthlySalary = monthlySalary;

}

/\*\*

\* @return the age

\*/

public int getAge() {

return age;

}

/\*\*

\* @param age the age to set

\*/

public void setAge(int age) {

this.age = age;

}

}

EmployeeDetails 类是用来

* get/set 员工的名字的值
* get/set 员工月薪的值
* get/set员工年龄的值

创建一个 EmpBusinessLogic.java 在 C:\ > TestNG\_WORKSPACE 其中包含业务逻辑

public class EmpBusinessLogic {

// Calculate the yearly salary of employee

public double calculateYearlySalary(EmployeeDetails employeeDetails){

double yearlySalary=0;

yearlySalary = employeeDetails.getMonthlySalary() \* 12;

return yearlySalary;

}

// Calculate the appraisal amount of employee

public double calculateAppraisal(EmployeeDetails employeeDetails){

double appraisal=0;

if(employeeDetails.getMonthlySalary() < 10000){

appraisal = 500;

}else{

appraisal = 1000;

}

return appraisal;

}

}

EmpBusinessLogic 类用于计算

* 员工的年薪
* 考核支付予雇员

现在，让我们创建一个TestNG 类名称为 TestEmployeeDetails.java 在 C:\ > TestNG\_WORKSPACE. TestNG类是一个Java类，它包含至少一个TestNG的注解。 这个类包含测试用例进行测试。可以配置，@BeforeXXX和@AfterXXX注解了TestNG测试 (在本章，我们会看到这样[TestNG - Execution Procedure](http://www.yiibai.com/testng/testng_execution_procedure.htm)) 它允许执行一些Java逻辑的目标点之前和之后。

import org.testng.Assert;

import org.testng.annotations.Test;

public class TestEmployeeDetails {

EmpBusinessLogic empBusinessLogic = new EmpBusinessLogic();

EmployeeDetails employee = new EmployeeDetails();

@Test

public void testCalculateAppriasal() {

employee.setName("Rajeev");

employee.setAge(25);

employee.setMonthlySalary(8000);

double appraisal = empBusinessLogic

.calculateAppraisal(employee);

Assert.assertEquals(500, appraisal, 0.0, "500");

}

// test to check yearly salary

@Test

public void testCalculateYearlySalary() {

employee.setName("Rajeev");

employee.setAge(25);

employee.setMonthlySalary(8000);

double salary = empBusinessLogic

.calculateYearlySalary(employee);

Assert.assertEquals(96000, salary, 0.0, "8000");

}

}

TestEmployeeDetails 测试方法，用于类EmpBusinessLogic，它

* 雇员测试的年薪
* 测试评估员工的金额

之前，你可以运行测试，但是必须使用特殊的XML文件，通常命名为testng.xml配置TestNG。此文件的语法很简单，其内容如下。创建这个文件C:\ > TestNG\_WORKSPACE:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="TestEmployeeDetails"/>

</classes>

</test>

</suite>

以上文件的详情如下：

* suite代表一个XML文件。它可以包含一个或多个测试，并被定义由<suite>标记
* 标签<test>代表一个测试，并可以包含一个或多个TestNG的类
* <class>的标签代表一个TestNG的类是一个Java类，它包含至少一个TestNG的注解。它可以包含一个或多个测试方法。

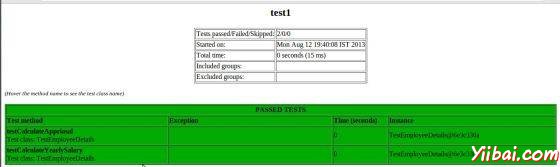
编译使用javac测试用例类。

C:\TestNG\_WORKSPACE>javac EmployeeDetails.java EmpBusinessLogic.java TestEmployeeDetails.java

现在TestNG用下面的命令：

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

如果所有配置正确的话，你应该看到测试结果，在控制台中。此外，TestNG创建了一个非常漂亮的HTML报告，会自动在当前目录下创建一个文件夹名为test-output 。如果打开​​并加载的index.html，会看到类似下面的图片中的一个页面：



**TestNG基本注解(注释)**

来源：原创文章    由 易百 更新版本    浏览：**7455**人次

**3**

传统的方式来表示[JUnit](http://www.yiibai.com/html/junit/)3中的测试方法是测试自己的名字前缀。标记一个类中的某些方法，具有特殊的意义，这是一个非常有效的方法，但命名不很好的扩展（如果我们想添加更多标签为不同的框架？），而非缺乏灵活性（如果我们要通过额外的参数测试框架）。

注释被正式加入到JDK 5中的[Java](http://www.yiibai.com/html/java/)语言和[TestNG](http://www.yiibai.com/html/testng/)作出选择使用注释注释测试类。

这里是TestNG的支持列表中的注解：

|  |  |
| --- | --- |
| **注解** | **描述** |
| **@BeforeSuite** | 注解的方法将只运行一次，运行所有测试前此套件中。 |
| **@AfterSuite** | 注解的方法将只运行一次此套件中的所有测试都运行之后。 |
| **@BeforeClass** | 注解的方法将只运行一次先行先试在当前类中的方法调用。 |
| **@AfterClass** | 注解的方法将只运行一次后已经运行在当前类中的所有测试方法。 |
| **@BeforeTest** | 注解的方法将被运行之前的任何测试方法属于内部类的 <test>标签的运行。 |
| **@AfterTest** | 注解的方法将被运行后，所有的测试方法，属于内部类的<test>标签的运行。 |
| **@BeforeGroups** | 组的列表，这种配置方法将之前运行。此方法是保证在运行属于任何这些组第一个测试方法，该方法被调用。 |
| **@AfterGroups** | 组的名单，这种配置方法后，将运行。此方法是保证运行后不久，最后的测试方法，该方法属于任何这些组被调用。 |
| **@BeforeMethod** | 注解的方法将每个测试方法之前运行。 |
| **@AfterMethod** | 被注释的方法将被运行后，每个测试方法。 |
| **@DataProvider** | 标志着一个方法，提供数据的一个测试方法。注解的方法必须返回一个Object[] []，其中每个对象[]的测试方法的参数列表中可以分配。  该@Test 方法，希望从这个DataProvider的接收数据，需要使用一个dataProvider名称等于这个注解的名字。 |
| **@Factory** | 作为一个工厂，返回TestNG的测试类的对象将被用于标记的方法。该方法必须返回Object[]。 |
| **@Listeners** | 定义一个测试类的监听器。 |
| **@Parameters** | 介绍如何将参数传递给@Test方法。 |
| **@Test** | 标记一个类或方法作为测试的一部分。 |

**使用注释的好处**

以下是一些使用注释的好处：

* TestNG的标识的方法关心寻找注解。因此，方法名并不限于任何模式或格式。
* 我们可以通过额外的参数注解。
* 注释是强类型的，所以编译器将标记任何错误。
* 测试类不再需要任何东西（如测试案例，在JUnit3）扩展。

**TestNG执行程序**

来源：原创文章    由 易百 更新版本    浏览：**6556**人次

**0**

本教程介绍了[TestNG](http://www.yiibai.com/html/testng/)中执行程序的方法，这意味着该方法被称为第一和一个接着。下面是执行程序的TestNG测试API的方法的例子。

创建一个Java类文件名TestngAnnotation.java在C:\>TestNG\_WORKSPACE测试注解。

import org.testng.annotations.Test;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.AfterSuite;

public class TestngAnnotation {

// test case 1

@Test

public void testCase1() {

System.out.println("in test case 1");

}

// test case 2

@Test

public void testCase2() {

System.out.println("in test case 2");

}

@BeforeMethod

public void beforeMethod() {

System.out.println("in beforeMethod");

}

@AfterMethod

public void afterMethod() {

System.out.println("in afterMethod");

}

@BeforeClass

public void beforeClass() {

System.out.println("in beforeClass");

}

@AfterClass

public void afterClass() {

System.out.println("in afterClass");

}

@BeforeTest

public void beforeTest() {

System.out.println("in beforeTest");

}

@AfterTest

public void afterTest() {

System.out.println("in afterTest");

}

@BeforeSuite

public void beforeSuite() {

System.out.println("in beforeSuite");

}

@AfterSuite

public void afterSuite() {

System.out.println("in afterSuite");

}

}

接下来，让我们创建的文件 **testng.xml** 在 **C:\ > TestNG\_WORKSPACE**执行注解。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="TestngAnnotation"/>

</classes>

</test>

</suite>

编译使用javac测试用例类。

C:\TestNG\_WORKSPACE>javac TestngAnnotation.java

现在运行testng.xml，将运行提供的测试用例类中定义的测试用例。

C:\TestNG\_WORKSPACE>java org.testng.TestNG testng.xml

验证输出。

in beforeSuite

in beforeTest

in beforeClass

in beforeMethod

in test case 1

in afterMethod

in beforeMethod

in test case 2

in afterMethod

in afterClass

in afterTest

in afterSuite

===============================================

Suite

Total tests run: 2, Failures: 0, Skips: 0

===============================================

见上面的输出，TestNG是执行过程如下：

* 首先所有beforeSuite（）方法只执行一次。
* 最后，afterSuite的（）方法只执行一次。
* 即使方法 beforeTest(), beforeClass(), afterClass() 和afterTest() 方法只执行一次。
* beforeMethod（）方法执行每个测试用例，但在此之前执行的测试用例。
* afterMethod（）方法执行每个测试用例，但测试用例执行后。
* In between beforeMethod() and afterMethod() each test case executes

**TestNG执行测试**

使用TestNG类执行测试用例。这个类的主入口点在[TestNG](http://www.yiibai.com/html/testng)的框架运行测试。用户可以创建自己的TestNG的对象，并调用它以许多不同的方式：

* 在现有的testng.xml
* 合成testng.xml，完全从Java创建
* 直接设定测试类

您还可以定义哪些群体包括或排除，分配参数，命令行参数：

* -d outputdir: 指定输出目录
* -testclass class\_name: 指定了一个或多个类名
* -testjar jar\_name: 指定的jar包含测试
* -sourcedir src1;src2: ; 分隔源目录列表（只有当使用的javadoc注释）
* -target
* -groups
* -testrunfactory
* -listener

testng.xml现有在下面的例子中，我们将创建TestNG的对象。

**创建一个类**

* 创建一个[Java](http://www.yiibai.com/html/java/)类进行测试为 MessageUtil.java 在 C:\ > TestNG\_WORKSPACE

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public String printMessage(){

System.out.println(message);

return message;

}

}

**创建测试例类**

* 创建一个Java测试类 SampleTest.java
* 您的测试类添加一个的测试方法testPrintMessage（）
* 添加注释@Test 到方法  testPrintMessage()
* 实现测试条件和使用的assertEquals API TestNG的检查条件

创建一个Java类文件名 SampleTest.java在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class SampleTest {

String message = "Hello World";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

Assert.assertEquals(message, messageUtil.printMessage());

}

}

**创建 testng.xml**

接下来，让我们创建testng.xml文件在 C:\ > TestNG\_WORKSPACE 执行测试用例，此文件捕获整个测试XML。这个文件可以很容易地描述所有的测试套件和它们的参数在一个文件中，你可以检查你的代码库或e-mail给同事。这也使得它容易提取测试或分裂的几个运行时配置的子集（例如，TestNG的database.xml 只能运行测试，行使数据库）。

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Sample test Suite">

<test name="Sample test">

<classes>

<class name="SampleTest" />

</classes>

</test>

</suite>

情况下使用javac编译测试

C:\TestNG\_WORKSPACE>javac MessageUtil.java SampleTest.java

现在，运行这个 testng.xml，将运行中定义的测试用例 <test> 标签

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。

Hello World

===============================================

Sample test Suite

Total tests run: 1, Failures: 0, Skips: 0

===============================================

# TestNG套件测试

来源：原创文章    由 易百 更新版本    浏览：**4660**人次

**0**

## TestNG套件测试

测试套件的测试是为了测试软件程序的行为或一系列行为的情况下，是一个集合。在[TestNG](http://www.yiibai.com/html/testng)，我们不能定义一套测试源代码，但它代表的套件是一个XML文件执行特征。这也允许灵活的配置要运行的测试。套件可以包含一个或多个测试和被定义由<suite>标签。

testng.xml中有<suite>根标签。它描述了一个测试套件，这反过来又是由多个<test>区段组成。

下表列出了所有的<suite>可接受合法属性。

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | 此套件的名称。这是一个强制性的属性。 |
| verbose | 这个运行级别或冗长。 |
| parallel | 由TestNG 运行不同的线程来运行此套件。 |
| thread-count | 使用的线程数，如果启用并行模式（忽略其他方式）。 |
| annotations | 在测试中使用注释的类型。 |
| time-out | 默认的超时时间，将用于本次测试中发现的所有测试方法。 |

在本章中，我们会告诉你一个例子，有两个Test1 & Test2测试类一起运行测试套件。

## 创建一个类

创建一个Java类进行测试 MessageUtil.java 在 C:\ > JUNIT\_WORKSPACE

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

## 创建测试用例类

创建一个Java类文件名 Test1.java 在C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class Test1 {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

Assert.assertEquals(message, messageUtil.printMessage());

}

}

创建一个Java类文件名 Test2.java 在C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class Test2 {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message,messageUtil.salutationMessage());

}

}

现在，让我们编辑写入testng.xml 在C:\ > TestNG\_WORKSPACE ，将包含<suite>标签如下：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="exampletest1">

<classes>

<class name="Test1" />

</classes>

</test>

<test name="exampletest2">

<classes>

<class name="Test2" />

</classes>

</test>

</suite>

Suite1 包括 exampletest1 和 exampletest2.

所有Java类编译使用javac。

C:\TestNG\_WORKSPACE>javac MessageUtil.java Test1.java Test2.java

现在运行 testng.xml，将运行提供的测试用例类中定义的测试用例。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

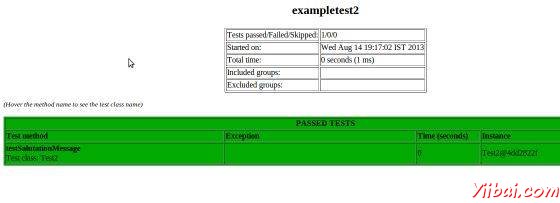
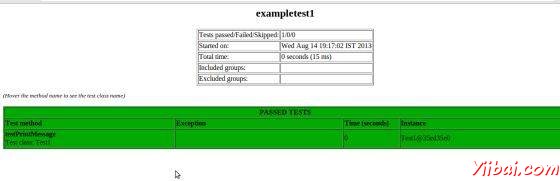
===============================================

Suite1

Total tests run: 2, Failures: 0, Skips: 0

===============================================

您也可以检查测试输出文件夹;下Suite1文件夹中，可以看到两个HTML创建的exampletest1.html 和 exampletest2.html 内容如下：



**TestNG忽略测试**

来源：原创文章    由 易百 更新版本    浏览：**3181**人次

**45**

有时，我们的代码是没有准备好，如果测试用例写入到测试方法/代码将无法运行，在这种情况下，@Test(enabled = false)有助于禁用此测试案例。

测试方法是标注了@Test(enabled = false)，那么并不是已经准备好测试的测试用例是绕过。

现在，让我们来看看测试@Test(enabled = false) 动作。

**创建一个类**

* 创建一个Java类进行测试为 MessageUtil.java 在 C:\ > TestNG\_WORKSPACE

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public String printMessage(){

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage(){

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

**创建测试案例类**

* 创建[Java](http://www.yiibai.com/html/java/)测试类为 IgnoreTest.java.
* 测试类添加测试方法testPrintMessage()，testSalutationMessage()。
* 添加注释 @Test(enabled = false) 到方法 testPrintMessage().

创建一个Java类文件名 IgnoreTest.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class IgnoreTest {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test(enabled = false)

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

}

**创建 testng.xml**

创建一个文件 testng.xml C:\ > TestNG\_WORKSPACE 用来执行测试案例

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="IgnoreTest" />

</classes>

</test>

</suite>

编译MessageUtil的测试用例类使用javac。

C:\TestNG\_WORKSPACE>javac MessageUtil.java IgnoreTest.java

现在，运行testng.xml，将无法运行testPrintMessage（）定义的测试用例在测试案例类。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。 testPrintMessage（）测试用例没有测试。

Inside testSalutationMessage()

Hi!Manisha

===============================================

Suite1

Total tests run: 1, Failures: 0, Skips: 0

===============================================

也可以忽略一组测试将在下一章中讨论

**TestNG组测试**

来源：原创文章    由 易百 更新版本    浏览：**4037**人次

**5**

在[TestNG](http://www.yiibai.com/htm/testng)中组测试是一个新的创新功能，它不存在于[JUnit](http://www.yiibai.com/htm/junit)框架，它允许调度到适当的部分方法和瓶坯复杂的测试方法分组。您不仅可以声明属于群体的那些方法，但你也可以指定一组包含其他组。然后，TestNG可调用和要求包括一组特定的群体（或正则表达式），而排除另一个集合。这给了你最大的灵活性，如何分区测试，如果想运行两套不同的测试背靠背，不要求重新编译任何东西。

组指定testng.xml文件使用<groups>标签。它可以发现无论是根据<test>或<suite>标签。组指定<suite>标签适用于所有的的<test>标签下方。

现在，让我们看一个例子，如何组测试。

**创建一个类**

* 创建一个Java类进行测试为 MessageUtil.java 在 C:\ > TestNG\_WORKSPACE

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "tutorialspoint" to the message

public String salutationMessage() {

message = "tutorialspoint" + message;

System.out.println(message);

return message;

}

// add "www." to the message

public String exitMessage() {

message = "www." + message;

System.out.println(message);

return message;

}

}

**创建测试案例类**

* 创建一个Java测试类为 GroupTestExample.java.
* 测试类添加测试方法testPrintMessage（）和 testSalutationMessage（）。
* 组的测试方法两个类别为：
  + 检入登记测试（checkintest）：提交新的代码之前，你应该运行这些测试。他们通常应快，只要确保没有被打破的基本功能。
  + 功能测试（functest）：这些测试应该涵盖软件的所有功能，每天至少运行一次，虽然理想情况下，会希望他们不断运行。

创建Java类文件名 GroupTestExample.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class GroupTestExample {

String message = ".com";

MessageUtil messageUtil = new MessageUtil(message);

@Test(groups = { "functest", "checkintest" })

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = ".com";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(groups = { "checkintest" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "tutorialspoint" + ".com";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test(groups = { "functest" })

public void testingExitMessage() {

System.out.println("Inside testExitMessage()");

message = "www." + "tutorialspoint"+".com";

Assert.assertEquals(message, messageUtil.exitMessage());

}

}

**创建testng.xml**

创建一个文件 testng.xml C:\ > TestNG\_WORKSPACE 来执行测试用例，在这里，我们将只执行这些测试，属于组functest。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<groups>

<run>

<include name="functest" />

</run>

</groups>

<classes>

<class name="GroupTestExample" />

</classes>

</test>

</suite>

编译MessageUtil的测试用例类使用javac。

C:\TestNG\_WORKSPACE>javac MessageUtil.java GroupTestExample.java

现在，运行testng.xml，只运行的方法testPrintMessage（），因为它属于组functest。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。只有的方法testPrintMessage（）被执行。

Inside testPrintMessage()

.com

Inside testExitMessage()

www..com

===============================================

Suite1

Total tests run: 2, Failures: 1, Skips: 0

===============================================

**组中组**

组也可以包含其他组。这些组称为MetaGroups。例如，您可能希望定义一个组中的所有，包括checkintest和functest。让我们修改testng.xml文件如下：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<groups>

<define name="all">

<include name="functest"/>

<include name="checkintest"/>

</define>

<run>

<include name="all"/>

</run>

</groups>

<classes>

<class name="GroupTestExample" />

</classes>

</test>

</suite>

执行上述testng.xml将执行所有三个测试会给你下面的结果：

Inside testPrintMessage()

.com

Inside testSalutationMessage()

tutorialspoint.com

Inside testExitMessage()

www.tutorialspoint.com

===============================================

Suite1

Total tests run: 3, Failures: 0, Skips: 0

===============================================

**排斥组**

可以忽略一个组使用<exclude>标签，如下图所示：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<groups>

<define name="all">

<exclude name="functest"/>

<include name="checkintest"/>

</define>

<run>

<include name="all"/>

</run>

</groups>

<classes>

<class name="GroupTestExample" />

</classes>

</test>

</suite>

**TestNG异常测试**

来源：原创文章    由 易百 更新版本    浏览：**3359**人次

**0**

[TestNG](http://www.yiibai.com/html/testng)跟踪异常处理代码提供了一个选项。可以测试是否需要代码抛出异常或不抛出。 @Test注释expectedExceptions 参数一起使用。现在，让我们来看看@Test（expectedExceptions）在动作中。

**创建一个类**

* 创建一个Java类进行测试说MessageUtil.java 在 C:\ > TestNG\_WORKSPACE
* 在printMessage（）方法里添加一个错误条件

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public void printMessage(){

System.out.println(message);

int a =0;

int b = 1/a;

}

// add "Hi!" to the message

public String salutationMessage(){

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

**创建测试案例类**

* 创建一个Java测试类为 ExpectedExceptionTest.java。
* 添加的ArithmeticException 和 testPrintMessage（）测试用例的预期异常。

创建一个Java类文件名ExpectedExceptionTest.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class ExpectedExceptionTest {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test(expectedExceptions = ArithmeticException.class)

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

messageUtil.printMessage();

}

@Test

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message,messageUtil.salutationMessage());

}

}

**创建测试运行**

创建 testng.xml 在 C:\ > TestNG\_WORKSPACE 执行测试案例。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="ExpectedExceptionTest" />

</classes>

</test>

</suite>

编译MessageUtil 测试用例类使用javac

C:\TestNG\_WORKSPACE>javac MessageUtil.java TestJunit.java

现在，运行测试运行，这将运行提供的测试用例类中定义的测试用例。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。testPrintMessage（）测试的情况下会获得通过。

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

===============================================

Suite1

Total tests run: 2, Failures: 0, Skips: 0

===============================================

# TestNG依赖测试

来源：原创文章    由 易百 更新版本    浏览：**3595**人次

**1**

有时候，你可能需要在一个特定的顺序调用方法在测试案例，或你想分享一些数据和方法之间的状态。[TestNG](http://www.yiibai.com/html/testng)支持这种依赖测试方法之间的显式依赖它支持声明。

TestNG允许指定依赖，无论与否：

* 使用属性dependsOnMethods在 @Test 注释OR
* 使用属性dependsOnGroups在@Test注解。

## 使用属性dependsOnMethods例如

### 创建一个类

创建一个Java类进行测试为 MessageUtil.java 在 C:\ > TestNG\_WORKSPACE

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

### 创建测试案例类

* 创建一个Java测试类为 DependencyTestUsingAnnotation.java.
* 添加方法 testPrintMessage(), testSalutationMessage() 和 initEnvironmentTest() 到测试类
* 添加属性 dependsOnMethods = { "initEnvironmentTest" } to the @Test 注释oftestSalutationMessage() 方法.

创建Java类文件名 DependencyTestUsingAnnotation.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(dependsOnMethods = { "initEnvironmentTest" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test

public void initEnvironmentTest() {

System.out.println("This is initEnvironmentTest");

}

}

### 创建TESTNG.XML

创建一个文件 testng.xml 在 C:\ > TestNG\_WORKSPACE 来执行测试用例

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="DependencyTestUsingAnnotation" />

</classes>

</test>

</suite>

编译MessageUtil的测试用例类使用javac

C:\TestNG\_WORKSPACE>javac MessageUtil.java DependencyTestUsingAnnotation.java

现在运行 testng.xml  这将会运行 testSalutationMessage() 只有在执行 ofinitEnvironmentTest() 方法之后

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出

This is initEnvironmentTest

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

===============================================

Suite1

Total tests run: 3, Failures: 0, Skips: 0

===============================================

## 示例，使用属性dependsOnGroups

也可以依赖于整个群组的方法。让我们来看看下面的例子：

### 创建一个类

创建一个Java类进行测试为 MessageUtil.java 在 C:\ > TestNG\_WORKSPACE

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

### 创建测试案例类

* 创建一个Java测试类说依赖TestUsingAnnotation.java.
* 添加测试方法  testPrintMessage(), testSalutationMessage() 和 initEnvironmentTest() 测试类和他们的组 "初始化"
* 添加属性 dependsOnMethods = { "init.\*" } to the @Test 注释 testSalutationMessage() 方法

创建Java类文件名 DependencyTestUsingAnnotation.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test(groups = { "init" })

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(dependsOnGroups = { "init.\*" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test(groups = { "init" })

public void initEnvironmentTest() {

System.out.println("This is initEnvironmentTest");

}

}

在这个例子中，testSalutationMessage（）被声明为根据任何一组匹配正则表达式“的init\*”，这保证了，一种方法，testPrintMessage的（）和initEnvironmentTest（）将始终前testSalutationMessage（）被调用。

如果一个方法失败，取决于你有一个很难依赖于它（alwaysRun= false，这是默认的），没有标记的方法依赖于它的失败，但作为跳过。跳过的方法将被报告为例如在最终报告（在HTML中，既不是红也不是绿的颜色），这是很重要的，因为跳过的方法不一定是失败。

### 创建TESTNG.XML

创建一个文件testng.xml C:\ > TestNG\_WORKSPACE 执行测试案例

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="DependencyTestUsingAnnotation" />

</classes>

</test>

</suite>

编译MessageUtil的测试用例类使用javac

C:\TestNG\_WORKSPACE>javac MessageUtil.java DependencyTestUsingAnnotation.java

现在，运行testng.xml，这将运行testSalutationMessage（）方法后，才执行initEnvironmentTest（）方法。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出

This is initEnvironmentTest

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

===============================================

Suite1

Total tests run: 3, Failures: 0, Skips: 0

===============================================

# TestNG参数化测试

来源：原创文章    由 易百 更新版本    浏览：**6406**人次

**5**

在[TestNG](http://www.yiibai.com/html/testng)的另一个有趣的功能是参数测试。在大多数情况下，你会遇到这样一个场景，业务逻辑需要一个巨大的不同数量的测试。参数测试，允许开发人员运行同样的测试，一遍又一遍使用不同的值。

TestNG让你直接传递参数测试方法两种不同的方式：

* 使用testng.xml
* 数据提供程序

## 传递参数使用testng.xml

有了这种技术，在testng.xml文件中定义的简单参数，然后在源文件中引用这些参数。让我们看看下面的例子中如何使用这种技术来传递参数。

### 创建测试案例类

* 创建一个Java测试类 ParameterizedTest1.java.
* 测试方法parameterTest（）添加到测试类。此方法需要一个字符串作为输入参数。
* 添加注释 @Parameters("myName") 到此方法。该参数将被传递testng.xml，在下一步我们将看到一个值。

创建Java类文件名 ParameterizedTest1.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class ParameterizedTest1 {

@Test

@Parameters("myName")

public void parameterTest(String myName) {

System.out.println("Parameterized value is : " + myName);

}

}

### 创建 TESTNG.XML

创建 testng.xml C:\ > TestNG\_WORKSPACE 执行测试案例

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<parameter name="myName" value="manisha"/>

<classes>

<class name="ParameterizedTest1" />

</classes>

</test>

</suite>

我们还可以定义参数在<suite>级别。假设我们已经定义在两个<suite>和<test>级别myName，在这种情况下，常规的作用域规则适用。这意味着，任何类里面<test>标签将查看值参数定义在<test>，而testng.xml文件中的类的其余部分将看到定义在<suite>中值

编译使用javac的测试用例类。

C:\TestNG\_WORKSPACE>javac ParameterizedTest1.java

现在，运行testng.xml，其中将运行parameterTest方法。TestNG的将试图找到一个命名myName的第一<test>标签的参数，然后，如果它不能找到它，它会搜索包围在的<suit>标签。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。

Parameterized value is : manisha

===============================================

Suite1

Total tests run: 1, Failures: 0, Skips: 0

===============================================

TestNG 对testng.xml 的参数的类型指定的值会自动尝试转换。下面是支持的类型：

* String
* int/Integer
* boolean/Boolean
* byte/Byte
* char/Character
* double/Double
* float/Float
* long/Long
* short/Short

## 传递参数与数据提供者

当你需要通过复杂的参数或参数需要创建从Java（复杂的对象，对象读取属性文件或数据库等..），在这种情况下，可以将参数传递使用数据提供者。数据提供者@DataProvider的批注的方法。这个注解只有一个字符串属性：它的名字。如果不提供名称，数据提供者的名称会自动默认方法的名称。数据提供者返回一个对象数组。

让我们看看下面的例子使用数据提供者。第一个例子是@DataProvider的使用Vector，String或Integer 作为参数，第二个例子是关于@DataProvider 的使用对象作为参数。

### 实例 1

在这里 @DataProvider 通过整数和布尔参数。

### 创建Java类

创建一个java类PrimeNumberChecker.java。这个类检查，如果是素数。创建这个类在 C:\ > TestNG\_WORKSPACE

public class PrimeNumberChecker {

public Boolean validate(final Integer primeNumber) {

for (int i = 2; i < (primeNumber / 2); i++) {

if (primeNumber % i == 0) {

return false;

}

}

return true;

}

}

### 创建测试案例类

* 创建一个Java测试类 ParamTestWithDataProvider1.java.
* 定义方法primeNumbers（），其定义为DataProvider 使用注释。此方法返回的对象数组的数组。
* 测试方法testPrimeNumberChecker（）添加到测试类中。此方法需要一个整数和布尔值作为输入参数。这个方法验证，如果传递的参数是一个素数。
* 添加注释 @Test(dataProvider = "test1") 到此方法。dataProvider的属性被映射到"test1".

创建Java类文件名ParamTestWithDataProvider1.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParamTestWithDataProvider1 {

private PrimeNumberChecker primeNumberChecker;

@BeforeMethod

public void initialize() {

primeNumberChecker = new PrimeNumberChecker();

}

@DataProvider(name = "test1")

public static Object[][] primeNumbers() {

return new Object[][] { { 2, true }, { 6, false }, { 19, true },

{ 22, false }, { 23, true } };

}

// This test will run 4 times since we have 5 parameters defined

@Test(dataProvider = "test1")

public void testPrimeNumberChecker(Integer inputNumber,

Boolean expectedResult) {

System.out.println(inputNumber + " " + expectedResult);

Assert.assertEquals(expectedResult,

primeNumberChecker.validate(inputNumber));

}

}

### 创建 TESTNG.XML

创建 testng.xml C:\ > TestNG\_WORKSPACE 执行测试案例。

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="ParamTestWithDataProvider1" />

</classes>

</test>

</suite>

编译使用javac的测试用例类。

C:\TestNG\_WORKSPACE>.javac ParamTestWithDataProvider1.java PrimeNumberChecker.java

运行testng.xml.

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。

2 true

6 false

19 true

22 false

23 true

===============================================

Suite1

Total tests run: 5, Failures: 0, Skips: 0

===============================================

### 实例 2

在这里，@DataProvider 传递对象作为参数。

### 创建Java类

创建一个Java类 Bean.java, 对象带有 get/set 方法, 在 C:\ > TestNG\_WORKSPACE.

public class Bean {

private String val;

private int i;

public Bean(String val, int i){

this.val=val;

this.i=i;

}

public String getVal() {

return val;

}

public void setVal(String val) {

this.val = val;

}

public int getI() {

return i;

}

public void setI(int i) {

this.i = i;

}

}

### 创建测试案例类

* 创建一个Java测试类 ParamTestWithDataProvider2.java.
* 定义方法primeNumbers（），其定义为DataProvider使用注释。此方法返回的对象数组的数组。
* 添加测试类中测试方法TestMethod（）。此方法需要对象的bean作为参数。
* 添加注释 @Test(dataProvider = "test1") 到此方法.  dataProvider 属性被映射到 "test1".

创建Java类文件名 ParamTestWithDataProvider2.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParamTestWithDataProvider2 {

@DataProvider(name = "test1")

public static Object[][] primeNumbers() {

return new Object[][] { { new Bean("hi I am the bean", 111) } };

}

@Test(dataProvider = "test1")

public void testMethod(Bean myBean) {

System.out.println(myBean.getVal() + " " + myBean.getI());

}

}

### 创建 TESTNG.XML

创建一个文件 testng.xml C:\ > TestNG\_WORKSPACE 来执行测试用例.

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="test1">

<classes>

<class name="ParamTestWithDataProvider2" />

</classes>

</test>

</suite>

编译使用javac的测试用例类。

C:\TestNG\_WORKSPACE>javac ParamTestWithDataProvider2.java Bean.java

运行 testng.xml.

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE" org.testng.TestNG testng.xml

验证输出。

hi I am the bean 111

===============================================

Suite1

Total tests run: 1, Failures: 0, Skips: 0

===============================================

# TestNG运行JUnit测试

来源：原创文章    由 易百 更新版本    浏览：**3304**人次

**1**

现在，您已经了解了[TestNG](http://www.yiibai.com/html/testng)和它的各种测试，如果现在担心如何重构现有的JUnit代码，那就没有必要，使用TestNG提供了一种方法，从[JUnit](http://www.yiibai.com/html/junit/)和TestNG按照自己的节奏。也可以使用TestNG执行现有JUnit测试用例。

TestNG可以自动识别和运行JUnit测试，所以你可以使用TestNG运行所有的测试，并编写新的测试使用TestNG。所有你必须做的就是把JUnit的库TestNG的类路径上，它可以发现并使用JUnit类，改变测试运行从JUnit和TestNG Ant中，然后运行TestNG的“mixed”模式。这种方式可以在同一个项目中所有的测试，即使是在同一个包中，并开始使用TestNG。这种方法还可以转换您现有的JUnit测试到TestNG。

让我们来看看下面的例子中，并尝试了上述功能：

## 创建JUnit测试用例类

创建一个Java类，这是一个JUnit测试类， TestJunit.java 在 C:\ > TestNG\_WORKSPACE

import org.junit.Test;

import static org.testng.AssertJUnit.assertEquals;

public class TestJunit {

@Test

public void testAdd() {

String str= "Junit testing using TestNG";

assertEquals("Junit testing using TestNG",str);

}

}

现在，让我们来编写 testng.xml 在 C:\ > TestNG\_WORKSPACE 应该包涵 <suite> 标签如下:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Converted JUnit suite" >

<test name="JUnitTests" junit="true">

<classes>

<class name="TestJunit" />

</classes>

</test>

</suite>

要执行JUnit测试用例定义属性 junit="true" 如上面的xml文件中. JUnit测试用例类TestJunit定义在类名。

JUnit 4中，TestNG将使用 org.junit.runner.JUnitCore 运行测试。

所有Java类编译使用javac。

C:\TestNG\_WORKSPACE>javac TestJunit.java

现在运行testng.xml，这将运行TestNG的JUnit测试用例。

C:\TestNG\_WORKSPACE>java -cp "C:\TestNG\_WORKSPACE:C:\TestNG\_WORKSPACE\lib\junit-4.11.jar" org.testng.TestNG testng.xml

在这里，我已经放在了 junit-4.11.jar 在 C:\TestNG\_WORKSPACE\lib\junit-4.11.jar下面.

验证输出。

===============================================

Converted JUnit suite

Total tests run: 1, Failures: 0, Skips: 0

===============================================

**TestNG测试结果报告**

来源：原创文章    由 易百 更新版本    浏览：**6135**人次

**4**

报告是任何测试的执行是最重要的部分，原因是它可以帮助用户了解执行测试，故障点和失败的原因的结果。记录，另一方面，重要的是要留意执行流程，或在任何故障的情况下进行调试。

[TestNG](http://www.yiibai.com/html/testng)默认情况下，会产生不同类型的测试执行报告。这包括[HTML](http://www.yiibai.com/html/html)和[XML](http://www.yiibai.com/html/xml)报表输出。 TestNG的还允许用户自己写的报告，并用它使用TestNG。还有一个选项来写你自己的记录器，在运行时通过TestNG的通知。

主要有两种方法来生成报告使用TestNG：

* 监听器: 为了实现一个监听类，类有实现theorg.testng。 ITestListener接口。这些类在运行时通知了TestNG测试开始时，结束后，失败，跳过或传递。
* 记录器: 为了实现一个报表类，实现一个org.testng.IReporter接口。这些类一整套运行结束时调用。调用时，该对象包含整个测试运行的信息传递到这个类。

下表列出了不同的情况报告和记录的例子：

|  |  |
| --- | --- |
| [自定义日志](http://www.yiibai.com/html/testng/2013/0916306.html?1379328934) | 这个例子说明如何编写您自己的记录。 |
| [自定义记录器](http://www.yiibai.com/html/testng/2013/0916307.html?1379328938) | 这个例子说明了如何编写自己的记录器。 |
| [HTML 和 XML 报告](http://www.yiibai.com/html/testng/2013/0916308.html?1379328942) | 这个例子说明了默认的HTML和XML报告TestNG产生。 |
| [JUnit 报告](http://www.yiibai.com/html/testng/2013/0916309.html?1379328945) | 这个例子说明了TestNG的报告生成Junit的报告。 |

**TestNG插件与ANT**

来源：原创文章    由 易百 更新版本    浏览：**4115**人次

**1**

在这个例子中，我们将演示如何使用[ANT](http://www.yiibai.com/html/ant)运行[TestNG](http://www.yiibai.com/html/testng)。让我们遵循的步骤：

**步骤1：下载Apache Ant**

下载 [Apache Ant](http://ant.apache.org/bindownload.cgi)

|  |  |
| --- | --- |
| **OS** | **压缩文件名** |
| Windows | apache-ant-1.8.4-bin.zip |
| Linux | apache-ant-1.8.4-bin.tar.gz |
| Mac | apache-ant-1.8.4-bin.tar.gz |

**步骤2：设置Ant环境**

设置ANT\_HOME环境变量指向参考基本目录的位置，ANT库存储在您的机器上。例如，我们已经存储了Ant库apache-ant-1.8.4，各种操作系统上的文件夹如下：

|  |  |
| --- | --- |
| **OS** | **输出** |
| Windows | 设置环境变量 ANT\_HOME to C:\Program Files\Apache Software Foundation\apache-ant-1.8.4 |
| Linux | export ANT\_HOME=/usr/local/\apache-ant-1.8.4 |
| Mac | export ANT\_HOME=/Library/\apache-ant-1.8.4 |

附加的Ant编译系统路径位置，在不同的操作系统如下：

|  |  |
| --- | --- |
| **OS** | **输出** |
| Windows | 追加字符串;%ANT\_HOME\bin 系统变量的结尾 |
| Linux | export PATH=$PATH:$ANT\_HOME/bin/ |
| Mac | not required |

**第3步：下载TestNG**

下载[http://www.testng.org.](http://www.testng.org/)

|  |  |
| --- | --- |
| **OS** | **Archive name** |
| Windows | testng-6.8.jar |
| Linux | testng-6.8.jar |
| Mac | testng-6.8.jar |

**第4步：创建项目结构**

* 创建文件夹 TestNGWithAnt 在C:\ > TestNG\_WORKSPACE
* 创建文件夹 src 在 C:\ > TestNG\_WORKSPACE > TestNGWithAnt
* 创建文件夹 test 在 C:\ > TestNG\_WORKSPACE > TestNGWithAnt
* 创建文件夹 lib 在 C:\ > TestNG\_WORKSPACE > TestNGWithAnt
* 创建MessageUtil 类在 C:\ > TestNG\_WORKSPACE > TestNGWithAnt > src 文件夹.

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public void printMessage(){

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage(){

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

* 创建TestMessageUtil 类在 C:\ > TestNG\_WORKSPACE > TestNGWithAnt > src 目录.

import org.testng.Assert;

import org.testng.annotations.Test;

public class TestMessageUtil {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

Assert.assertEquals(message,messageUtil.printMessage());

}

@Test

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message,messageUtil.salutationMessage());

}

}

* 拷贝 testng-6.8.jar 到 C:\ > TestNG\_WORKSPACE > TestNGWithAnt > lib 文件夹

**创建 ANT build.xml**

首先，我们需要定义TestNG的ant任务如下：

<taskdef name="testng" classname="org.testng.TestNGAntTask">

<classpath>

<pathelement location="lib/testng-6.8.jar"/>

</classpath>

</taskdef>

然后我们使用 <testng> TestNG的测试案例Ant来执行任务。

C:\ > TestNG\_WORKSPACE > TestNGWithAnt >\ build.xml 内容如下：

<project name="TestNGTest" default="test" basedir=".">

<!-- Define <testng> task -->

<taskdef name="testng" classname="org.testng.TestNGAntTask">

<classpath>

<pathelement location="lib/testng-6.8.jar"/>

</classpath>

</taskdef>

<property name="testdir" location="test" />

<property name="srcdir" location="src" />

<property name="libdir" location="lib" />

<property name="full-compile" value="true" />

<path id="classpath.base"/>

<path id="classpath.test">

<fileset dir="${libdir}">

<include name="\*\*/\*.jar" />

</fileset>

<pathelement location="${testdir}" />

<pathelement location="${srcdir}" />

<path refid="classpath.base" />

</path>

<target name="clean" >

<delete verbose="${full-compile}">

<fileset dir="${testdir}" includes="\*\*/\*.class" />

</delete>

</target>

<target name="compile" depends="clean">

<javac srcdir="${srcdir}" destdir="${testdir}"

verbose="${full-compile}">

<classpath refid="classpath.test"/>

</javac>

</target>

<target name="test" depends="compile">

<testng outputdir="${testdir}" classpathref="classpath.test">

<xmlfileset dir="${srcdir}" includes="testng.xml"/>

</testng>

</target>

</project>

执行下列ant命令。

C:\TestNG\_WORKSPACE\TestNGWithAnt>ant

验证输出

test:

[testng] [TestNG] Running:

[testng] C:\TestNG\_WORKSPACE\TestNGWithAnt\src\testng.xml

[testng]

[testng] Inside testPrintMessage()

[testng] Manisha

[testng] Inside testSalutationMessage()

[testng] Hi!Manisha

[testng]

[testng] ===============================================

[testng] Plug ANT test Suite

[testng] Total tests run: 2, Failures: 0, Skips: 0

[testng] ===============================================

[testng]

BUILD SUCCESSFUL

Total time: 1 second

**TestNG Eclipse插件**

来源：原创文章    由 易百 更新版本    浏览：**8942**人次

**2**

用[eclipse](http://www.yiibai.com/html/eclipse)设置TestNG，下面的步骤必须遵循：

**步骤1：下载TestNG的归档文件**

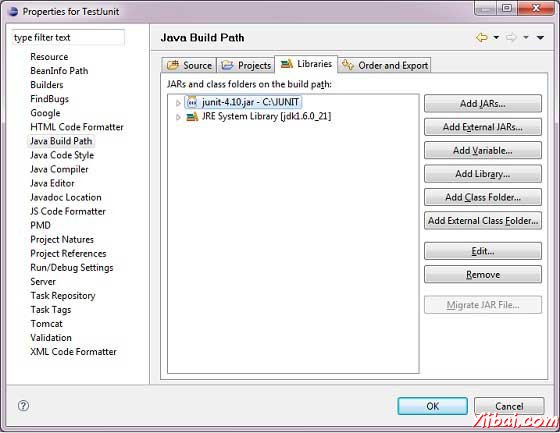
下载 [http://www.testng.org](http://www.testng.org/)

|  |  |
| --- | --- |
| **OS** | **压缩文件名** |
| Windows | testng-6.8.jar |
| Linux | testng-6.8.jar |
| Mac | testng-6.8.jar |

假设你上面复制的JAR文件到 C:\>TestNG 文件夹.

**第二步：设置Eclipse环境**

* 打开 eclipse -> 右键单击项目，然后单击property > Build Path > Configure Build Path 并添加 testng-6.8.jar 在库中使用 Add External Jar 按钮.



* 我们假设你的eclipse 中 TestNG插件已经内置，如果不是，那么请使用更新站点获取最新版本：
  + 在你的 eclipse IDE, 选择 Help / Software updates / Find and Install.
  + 搜索新功能安装。
  + 新的远程站点。
  + For Eclipse 3.4 and above, enter <http://beust.com/eclipse>.
  + For Eclipse 3.3 and below, enter <http://beust.com/eclipse1>.
  + Make sure the check box next to URL is checked and click Next.
  + 然后Eclipse会引导帮您完成整个过程。

现在，你的eclipse已经可以使用 TestNG测试用例的开发做好准备。

**步骤3：确认Eclipse已经安装TestNG**

* 在eclipse中创建一个项目TestNGProject
* 创建一类MessageUtil在项目测试。

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public String printMessage(){

System.out.println(message);

return message;

}

}

* 在项目中创建一个测试类TestNGExample

import org.testng.Assert;

import org.testng.annotations.Test;

public class TestNGExample {

String message = "Hello World";

MessageUtil messageUtil = new MessageUtil(message);

@Test

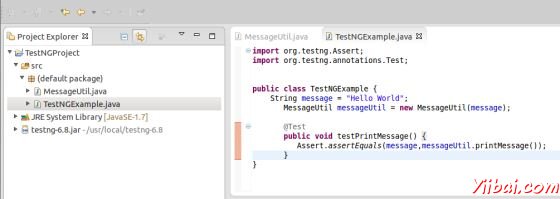
public void testPrintMessage() {

Assert.assertEquals(message,messageUtil.printMessage());

}

}

下面应该是项目结构：



最后，通过右击程序和[TestNG](http://www.yiibai.com/html/testng)的运行验证程序的输出。

验证结果。

