# TentNG

# TestNG介绍

## 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环境设置（配置安装）

## 下载TestNG的归档文件

下载最新版本的TestNG的jar文件，详细请点击访问 [http://www.testng.org](http://testng.org/doc/download.html).。

## 设置TestNG的环境

设置TESTNG\_HOME环境变量指向TestNG的jar 存放在您的机器上的基本目录位置。假设，我们已经储存了testng-6.9.14.jar， TestNG各种操作系统上的文件夹如下：

|  |  |
| --- | --- |
| **OS** | **输出** |
| Windows | Set the environment variable TESTNG\_HOME to C:\TESTNG |
| Linux | export TESTNG\_HOME=/usr/local/TESTNG |
| Mac | export TESTNG\_HOME=/Library/TESTNG |

## 设置CLASSPATH变量

设置CLASSPATH环境变量指向TestNG的jar文件位置。假设，我们已经储存了testng-6.9.14.jar.jar, TestNG在各种操作系统上的文件夹如下：

|  |  |
| --- | --- |
| **OS** | **输出** |
| Windows | 设置环境变量 CLASSPATH 为 %CLASSPATH%;%TESTNG\_HOME%\testng-6.9.14.jar.jar; |
| Linux | export CLASSPATH=$CLASSPATH:$TESTNG\_HOME/testng-6.9.14.jar.jar: |
| Mac | export CLASSPATH=$CLASSPATH:$TESTNG\_HOME/testng-6.9.14.jar.jar: |

## 测试TestNG的设置

创建一个Java类文件名TestNGSimpleTest  C:\ > TestNG\_WORKSPACE

import org.testng.annotations.Test;

import static org.testng.Assert.assertEquals;

public class TestNGSimpleTest {

@Test

public void testAdd() {

String str = "TestNG is working fine";

assertEquals("TestNG is working fine", str);

}

}

[TestNG](http://www.yiibai.com/html/testng)的几种不同的方法可以被调用：

* testng.xml 文件
* ant
* 命令行

让我们调用使用testng.xml文件。创建一个XML文件名称testng.xml C:\ > TestNG\_WORKSPACE 执行测试用例(s)

<?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="TestNGSimpleTest"/>

</classes>

</test>

</suite>

## 检查结果

类编译使用javac编译如下：

C:\TestNG\_WORKSPACE>javac TestNGSimpleTest.java

现在，调用testng.xml看到的结果：

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

验证输出

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

Suite1

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

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

# TestNG基本注解(注释)

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

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。
* 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套件测试

## 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 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

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

# TestNG忽略测试

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 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

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

# TestNG组测试

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 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">

<include name="checkintest"/>

</define>

<run>

<include name="all"/>

<exclude name="functest"/>

</run>

</groups>

<classes>

<class name="GroupTestExample" />

</classes>

</test>

</suite>

# TestNG异常测试

[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 ExpectedExceptionTest.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 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

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" 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（）和testPrintMessage（）方法。

C:\TestNG\_WORKSPACE>java 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

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

## dependsOnGroups Vs dependsOnMethods

* 在使用组，我们不再面临重构的问题。只要我们不修改dependsOnGroups或组属性，我们的测试将继续运行，设立适当的依赖。
* 每当一个新的方法需要添加依存关系图中，我们需要做的就是把它正确的组中，并确保它依赖于正确的组。我们不需要修改任何其他方法。

# TestNG参数化测试

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" 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" 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" 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

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: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

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自定义日志记录

我们此前读[TestNG](http://www.yiibai.com/html/testng)的记录和报告提供了不同的选项。现在，让我们了解如何开始使用它们。首先，我们将编写一个示例程序，我们将使用的ITestListener接口，以便进行记录。

## 创建测试案例类

创建一个Java类为 SampleTest.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class SampleTest {

@Test

public void testMethodOne(){

Assert.assertTrue(true);

}

@Test

public void testMethodTwo(){

Assert.assertTrue(false);

}

@Test(dependsOnMethods={"testMethodTwo"})

public void testMethodThree(){

Assert.assertTrue(true);

}

}

前面的测试类包含三种测试方法，其中testMethodOne andtestMethodThree将通过执行时，，而testMethodTwo是通过一个falseBoolean值断言失败。 assertTrue方法，该方法用于在测试中的真值条件。

## 创建自定义日志记录类

创建另一个新的类名为 CustomListener.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.ITestResult;

import org.testng.TestListenerAdapter;

public class CustomListener extends TestListenerAdapter{

private int m\_count = 0;

@Override

public void onTestFailure(ITestResult tr) {

log(tr.getName()+ "--Test method failed\n");

}

@Override

public void onTestSkipped(ITestResult tr) {

log(tr.getName()+ "--Test method skipped\n");

}

@Override

public void onTestSuccess(ITestResult tr) {

log(tr.getName()+ "--Test method success\n");

}

private void log(String string) {

System.out.print(string);

if (++m\_count % 40 == 0) {

System.out.println("");

}

}

}

上述的类扩展TestListenerAdapter，空方法实现ITestListener。因此，不需要重写其他方法从接口。如果你喜欢，可以直接实现这个接口。

## 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="Simple Logger Suite">

<listeners>

<listener class-name="CustomListener" />

</listeners>

<test name="Simple Logger test">

<classes>

<class name="SampleTest" />

</classes>

</test>

</suite>

编译SampleTest，CustomListener类使用javac

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

现在运行 testng.xml.

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

验证输出

testMethodOne--Test method success

testMethodTwo--Test method failed

testMethodThree--Test method skipped

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

Simple Logger Suite

Total tests run: 3, Failures: 1, Skips: 1

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

我们创建了一个自定义logger类，其中实现ITestListener接口和依附于作为监听器的TestNG测试套件。 TestNG的测试开始时，测试失败，在测试成功，所以这个监听器类的方法调用。可以实现多个听众，并将其添加到测试套件执行，TestNG的将调用所有侦听器连接到测试套件。

当我们需要看到的连续状态的测试执行，测试时得到执行，主要用于记录监听器。

# TestNG自定义记录器

在本节中，我们将介绍一个例子，编写自定义记录器和TestNG的方法。要编写一个定制的记录器类，我们的扩展类应实现IReporter接口。让我们继续前进，并创建一个示例使用自定义的记录器。

## 创建测试案例类

创建一个Java类为 SampleTest.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class SampleTest {

@Test

public void testMethodOne(){

Assert.assertTrue(true);

}

@Test

public void testMethodTwo(){

Assert.assertTrue(false);

}

@Test(dependsOnMethods={"testMethodTwo"})

public void testMethodThree(){

Assert.assertTrue(true);

}

}

上述测试类的包含三个测试方法，其中testMethodOne 和 testMethodThree将通过在执行时，而testMethodTwo由通过一个falseBoolean的值Assert.assertTrue方法，它是用于在测试中的真值条件失败。

## 创建自定义报告类

创建另一个新的类名为 CustomReporter.java 在 C:\ > TestNG\_WORKSPACE

**import** java.util.List;

**import** java.util.Map;

**import** org.testng.IReporter;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestContext;

**import** org.testng.xml.XmlSuite;

**public** **class** CustomReporter **implements** IReporter {

@Override

**public** **void** generateReport(List xmlSuites, List suites, String outputDirectory) {

// Iterating over each suite included in the test

**for** (Object suite : suites) {

// Following code gets the suite name

String suiteName = ((ISuite) suite).getName();

// Getting the results for the said suite

Map suiteResults = ((ISuite) suite).getResults();

**for** (Object sr : suiteResults.values()) {

ITestContext tc = ((ISuiteResult) sr).getTestContext();

System.***out***.println(

"Passed tests for suite '" + suiteName + "' is:" + tc.getPassedTests().getAllResults().size());

System.***out***.println(

"Failed tests for suite '" + suiteName + "' is:" + tc.getFailedTests().getAllResults().size());

System.***out***.println("Skipped tests for suite '" + suiteName + "' is:"

+ tc.getSkippedTests().getAllResults().size());

}

}

}

}前面的的类实现org.testng.IReporter 接口。它实现了IReporter接口定义的方法GenerateReport。这个方法有三个参数：

* 第一个是xmlSuite，这是TestNG的测试XML正在执行中提到的列表套件
* 第二个是套件，其中包含一套测试执行后信息，该对象包含了所有的信息包，类，测试方法和测试执行结果。
* 第三的outputDirectory，报告将产生的输出文件夹路径，其中包含的信息。

## 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"?>

<suite name="Simple Reporter Suite">

<listeners>

<listener class-name="CustomReporter" />

</listeners>

<test name="Simple Reporter test">

<classes>

<class name="SampleTest" />

</classes>

</test>

</suite>

编译SampleTest，CustomReporter类使用javac

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

运行 testng.xml.

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

验证输出

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

Simple Reporter Suite

Total tests run: 3, Failures: 1, Skips: 1

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

Passed tests for suite 'Simple Reporter Suite' is:1

Failed tests for suite 'Simple Reporter Suite' is:1

Skipped tests for suite 'Simple Reporter Suite' is:1

前面的例子显示了一个简单的自定义报告器，打印的数量在控制台上对每个套件包含在上述的测试执行失败，通过跳过测试。报告器主要是用于测试的执行，以生成最终的报告。扩展程序可以被用来生成XML，HTML，CHM，CSV或文本格式的文件，根据报告要求。

# TestNG HTML和XML报告

[TestNG](http://www.yiibai.com/html/testng)带有一些预定义的监听器库的一部分。默认情况下，这些监听器加入任何测试执行，并产生不同的[HTML](http://www.yiibai.com/html/html/)和[XML](http://www.yiibai.com/html/xml)报告任何测试执行。该报告所产生的名为testoutput 文件夹默认情况下，通过配置可以更改为任何其他文件夹。这些报告包含一些HTML和XML TestNG的具体报告。

## 创建测试案例类

创建一个java类名为 SampleTest.java 在C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class SampleTest {

@Test

public void testMethodOne(){

Assert.assertTrue(true);

}

@Test

public void testMethodTwo(){

Assert.assertTrue(false);

}

@Test(dependsOnMethods={"testMethodTwo"})

public void testMethodThree(){

Assert.assertTrue(true);

}

}

上述测试类的包含三种测试方法，其中将通过在执行时testMethodOne和testMethodThree，，而testMethodTwo由通过一个假布尔值的Assert.assertTrue方法，它是用于在测试中的真值条件失败。

## 创建 testng.xml

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

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

<suite name="Simple HTML-XML Suite">

<test name="Simple HTML-XML test">

<classes>

<class name="SampleTest" />

</classes>

</test>

</suite>

编译使用javac SampleTest类。

C:\TestNG\_WORKSPACE>javac SampleTest.java

现在，运行testng.xml。

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

验证输出。

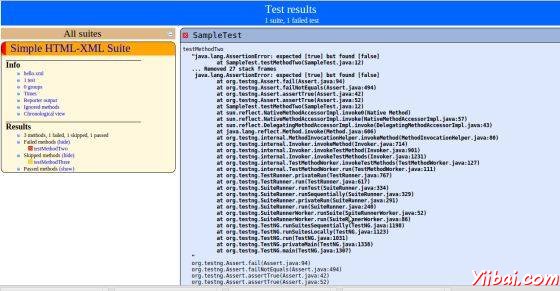
===============================================

Simple HTML-XML Suite

Total tests run: 3, Failures: 1, Skips: 1

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

现在，去到 C:\TestNG\_WORKSPACE\test-output 目录. 默认Web浏览器中打开index.html。你会看到下面的HTML报告内容如下：



现在打开 C:\TestNG\_WORKSPACE\test-output\testing-results.xml 在您的系统上默认XML编辑器，，会在XML文件中看到下面的结果：



TestNG的默认情况下生成多个报告，作为其执行测试的一部分。这些报告主要包括TestNG的[HTML](http://www.yiibai.com/html/html)报告，TestNG的电子邮件发送的报告，TestNG 报告[XML](http://www.yiibai.com/html/xml)和[JUnit](http://www.yiibai.com/html/junit)报告的XML文件。输出报告的文件夹（在这种情况下，测试输出）下可以找到这些文件。这种默认的报告生成运行测试的同时，可以禁用通过设置值的属性使用DefaultListeners的值为false。这个属性可以同时使用，如Ant或[Maven](http://www.yiibai.com/html/maven)构建工具。

# TestNG Junit报告

[JUnit](http://www.yiibai.com/html/junit)是单元框架，最初用于许多[Java](http://www.yiibai.com/html/java)应用软件作为一个单元测试框架之一。默认情况下，JUnit测试生成一个简单的XML文件测试执行报告。然后这些XML文件可以被用来生成任何自定义报表按测试要求。我们也可以使用[XML](http://www.yiibai.com/html/xml)文件生成HTML报告。Ant的有这样一个实用的任务，需要这些JUnit的XML文件作为输入，并生成一个HTML报告。

TestNG默认情况下，生成JUnit的XML执行任何测试报告（测试输出文件夹中）。我们可以使用这些XML格式的报告文件作为一个JUnit HTML报告生成的输入。让我们看看下面的例子：

## 创建测试用例类

创建一个java类名为 SampleTest.java 在 C:\ > TestNG\_WORKSPACE

import org.testng.Assert;

import org.testng.annotations.Test;

public class SampleTest {

@Test

public void testMethodOne(){

Assert.assertTrue(true);

}

@Test

public void testMethodTwo(){

Assert.assertTrue(false);

}

@Test(dependsOnMethods={"testMethodTwo"})

public void testMethodThree(){

Assert.assertTrue(true);

}

}

上述测试类的包含三个测试方法，其中将通过在执行testMethodOne和testMethodThree，其中作为testMethodTwo失败通过一个false的布尔值到Assert.assertTrue方法，它是用于在测试中的真值条件。

## 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"?>

<suite name="Simple Suite">

<test name="Simple test">

<classes>

<class name="SampleTest" />

</classes>

</test>

</suite>

编译使用javac SampleTest类。

C:\TestNG\_WORKSPACE>javac SampleTest.java

现在，运行testng.xml。

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

验证输出。

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

Simple Suite

Total tests run: 3, Failures: 1, Skips: 1

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

现在，我们已经可以从上面执行JUnit的XML报告，让我们创建一个简单的Ant构建配置XML文件来生成一个HTML报告测试执行。

创建一个新的文件名为 build.xml 在目录 C:\ > TestNG\_WORKSPACE 中

<project name="TestNG\_WORKSPACE" default="junit-report" basedir=".">

<!-- Sets the property variables to point to respective directories -->

<property name="junit-xml-dir" value="${basedir}/test-output/junitreports"/>

<property name="report-dir" value="${basedir}/html-report" />

<!-- Ant target to generate html report -->

<target name="junit-report">

<!-- Delete and recreate the html report directories -->

<delete dir="${report-dir}" failonerror="false"/>

<mkdir dir="${report-dir}" />

<mkdir dir="${report-dir}/Junit" />

<!-- Ant task to generate the html report.

todir - Directory to generate the output reports

fileset - Directory to look for the junit xml reports.

report - defines the type of format to be generated.

Here we are using "noframes" which generates a single html report.

-->

<junitreport todir="${report-dir}/Junit">

<fileset dir="${junit-xml-dir}">

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

</fileset>

<report format="noframes" todir="${report-dir}/Junit" />

</junitreport>

</target>

</project>

前面的XML定义了一个简单的Ant build.xml文件，具有特定的Ant目标名为JUnit的报告，执行时产生一个JUnit报告。目标看起来JUnit报告XML文件目录下test-output/junitreports。 Ant配置文件的默认目标执行配置JUnit的报告。

打开命令提示符窗口，然后转到 C:\ > TestNG\_WORKSPACE 目录在命令提示符下运行以下命令：

C:\TestNG\_WORKSPACE> ant

一旦执行，一个JUnit HTML报告，将产生配置目录/html-report/Junit。打开文件名为

junit-noframes.html

默认Web浏览器上。你会看到下面的HTML报告：



在这里，我们看到了如何使用JUnit XML报告由TestNG产生和使用Ant生成HTML报告。有两种类型的报告，可以使用这种方法产生的：帧和无帧。如果报表生成帧配置，为每个类和主报告生成多个文件，将他们连接到通过链接。一个无帧报告由一个单一的文件执行测试的所有结果。这可以通过提供相应的值在Ant报告任务的format属性配置。

# TestNG插件与ANT

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 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

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

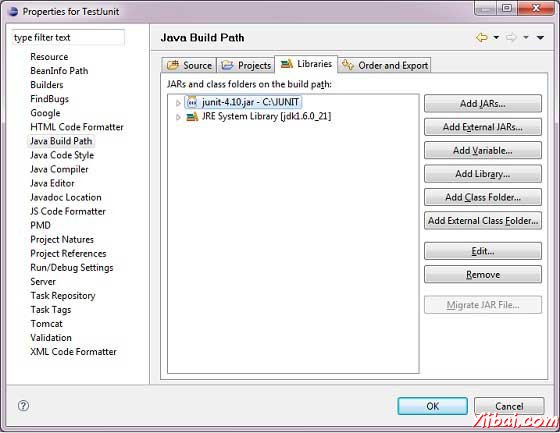
下载 [http://www.testng.org](http://www.testng.org/)

|  |  |
| --- | --- |
| **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

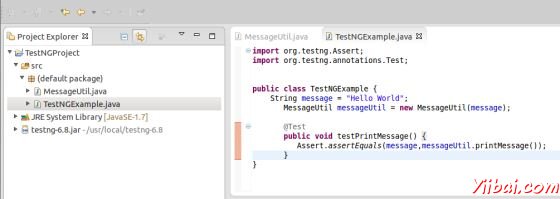
public void testPrintMessage() {

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。

