目录

TestNG 介绍 3
TestNG 是什么? 3
TestNG 的特点 3
TestNG 环境设置（配置安装） 4
系统要求 4
步骤 1 -验证 Java 安装在你的机器上 4
第二步：设置 JAVA 环境 5
第 3 步：下载 TestNG 的归档文件 5
步骤 4：设置 TestNG 的环境 6
第 5 步：设置 CLASSPATH 变量 6
步骤 6：测试 TestNG 的设置 6
第 7 步：检查结果 7
TestNG 编写测试 7
TestNG 基本注解(注释) 11
使用注释的好处 12
TestNG 执行程序 12
TestNG 执行测试 14
创建一个类 15
创建测试例类 16
创建 testng.xml 16
TestNG 套件测试 17
estNG 套件测试 17
创建一个类 17
创建测试用例类 18
TestNG 忽略测试 20
创建一个类 20
创建测试案例类 21
创建 testng.xml 22
TestNG 组测试 22
创建一个类 22
创建测试案例类 23
TestNG 异常测试 24
创建一个类 24
创建测试案例类 25
创建测试运行 25
TestNG 依赖测试 26
使用属性 dependsOnMethods 例如 26
创建一个类 26

创建测试案例类.....27
创建 TESTNG.XML.....28
dependsOnGroups Vs dependsOnMethods.....28
TestNG 参数化测试.....28
传递参数使用 testng.xml.....29
创建测试案例类.....29
创建 TESTNG.XML.....29
实例 2.....30
创建 Java 类.....30
创建测试案例类.....31
创建 TESTNG.XML.....31
TestNG 运行 JUnit 测试.....32
创建 JUnit 测试用例类.....32
TestNG 测试结果报告.....33
TestNG 插件与 ANT.....34
步骤 1：下载 Apache Ant.....34
步骤 2：设置 Ant 环境.....34
第 3 步：下载 TestNG.....35
第 4 步：创建项目结构.....35
创建 ANT build.xml.....37
TestNG Eclipse 插件.....37
步骤 1：下载 TestNG 的归档文件.....37
第二步：设置 Eclipse 环境.....38
步骤 3：确认 Eclipse 已经安装 TestNG.....39

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 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 环境设置（配置安装）

TestNG 是一个 Java 的框架，所以第一个要求是 JDK 要安装在你的机器上。

系统要求

JDK	1.5 或以上
内存	没有最低要求
磁盘空间	没有最低要求
操作系统	没有最低要求

步骤 1 -验证 Java 安装在你的机器上

现在，打开控制台并执行以下的 java 命令。

OS	任务	命令
Windows	打开命令控制台	c:> java -version
Linux	打开命令终端	$ java -version
Mac	打开命令终端	machine:~ joseph$ java -version

让我们来验证所有的操作系统的输出：

OS	输出
Windows	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)

Linux	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
Mac	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)

如果你没有安装 Java，安装 Java 软件开发工具包（SDK）点击： http://www.oracle.com/technetwork/java/javase/downloads/index.html. 我们假设本教程中安装和使用 Java1.7.0_25 版本。

第二步：设置 JAVA 环境

设置 JAVA_HOME 环境变量指向的基本目录的位置，在你的机器上安装 Java。例如：

OS	输出
Windows	设置环境变量 JAVA_HOME 为 C:Program FilesJavajdk1.7.0_25
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

添加 Java 编译器的位置，系统路径。

OS	输出
Windows	Append the string; C:Program FilesJavajdk1.7.0_25in to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

验证 Java 安装使用命令 java-version 如上所述。

第 3 步：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. 。在写这篇教程的时候，我下载 TestNG 中-6.8.jar，并将 testng-6.8.jar 其复制到 C:>TestNG 目录。

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

步骤 4：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-6.8.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

第 5 步：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-6.8.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% estng-6.8.jar;
Linux	export CLASSPATH=$CLASSPATH:$TESTNG_HOME/testng-6.8.jar:
Mac	export CLASSPATH=$CLASSPATH:$TESTNG_HOME/testng-6.8.jar:

步骤 6：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 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>
```

第 7 步：检查结果

类编译使用 javac 编译如下：

```
===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

TestNG 编写测试

编写 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）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:

```
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)

传统的方式来表示 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 语言和 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 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。

```
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 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 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

```
Hello World


===============================================
Sample test Suite
Total tests run: 1, Failures: 0, Skips: 0
```

```
==========================================
```

TestNG 套件测试

estNG 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，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。

```
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

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 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。

```
Inside testSalutationMessage()
Hi!Manisha

===============================================
Suite1
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

TestNG 组测试

在 TestNG 中组测试是一个新的创新功能，它不存在于 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 "tutorialsyiibai" to the message
    public String salutationMessage() {
        message = "tutorialsyiibai" + 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

```
<?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

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

```
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 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

```
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 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。

```
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。

```
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 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 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。

```
===============================================
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 默认情况下，会产生不同类型的测试执行报告。这包括 HTML 和 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	这个例子说明如何编写您自己的记录。
自定义记录器	这个例子说明了如何编写自己的记录器。
HTML 和 XML 报告	这个例子说明了默认的 HTML 和 XML 报告 TestNG 产生。
JUnit 报告	这个例子说明了 TestNG 的报告生成 Junit 的报告。

TestNG 插件与 ANT

在这个例子中，我们将演示如何使用 ANT 运行 TestNG。让我们遵循的步骤：

步骤 1：下载 Apache Ant

下载 Apache Ant

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 FilesApache Software Foundationapache-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_HOMEin 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.

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：

```
test:
   [testng] [TestNG] Running:
   [testng]   C:TestNG_WORKSPACETestNGWithAntsrc    estng.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 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

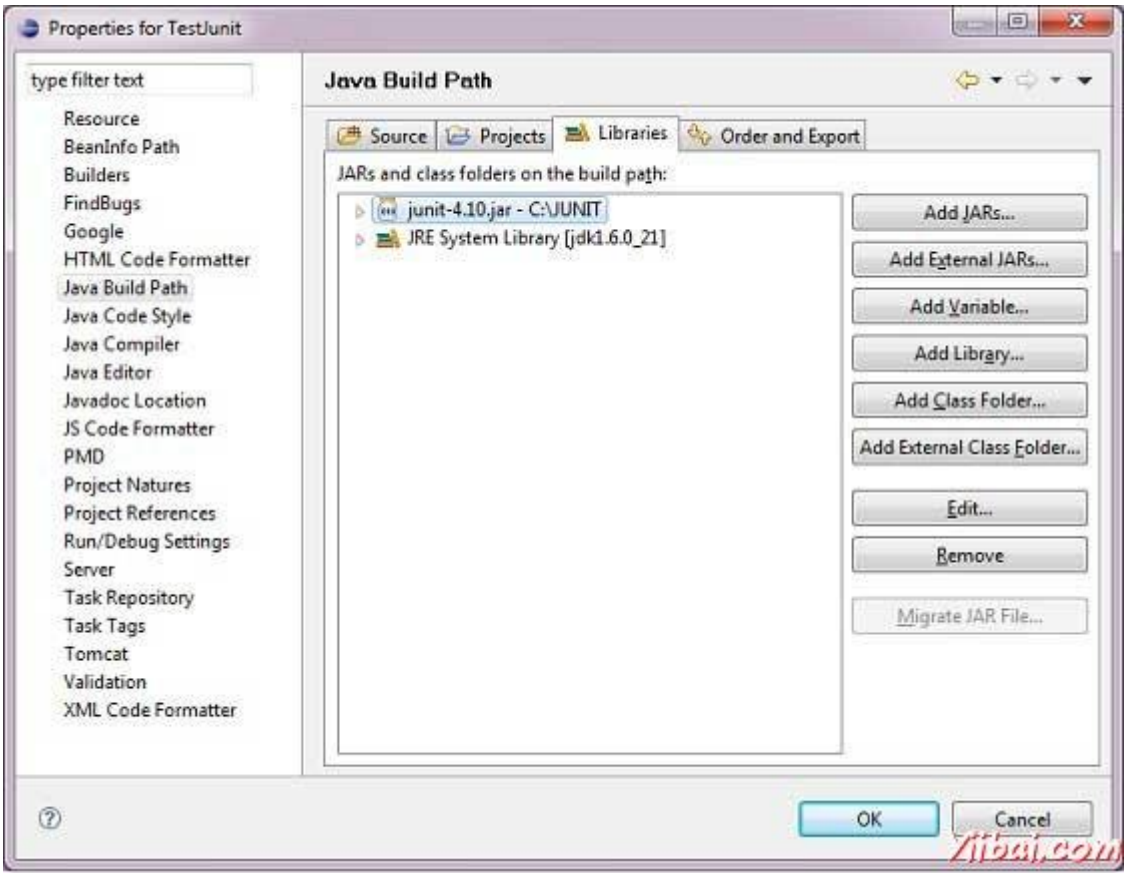
OS	压缩文件名

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。

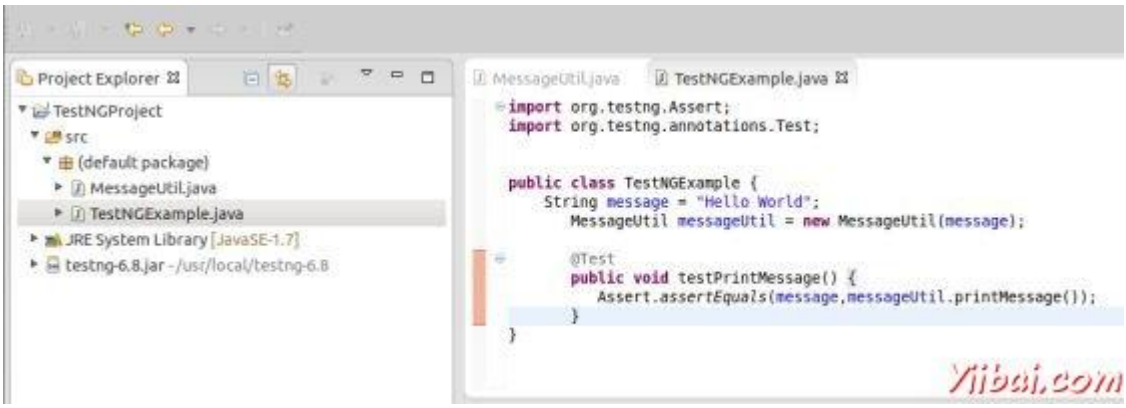
现在，你的 eclipse 已经可以使用 TestNG 测试用例的开发做好准备。

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。

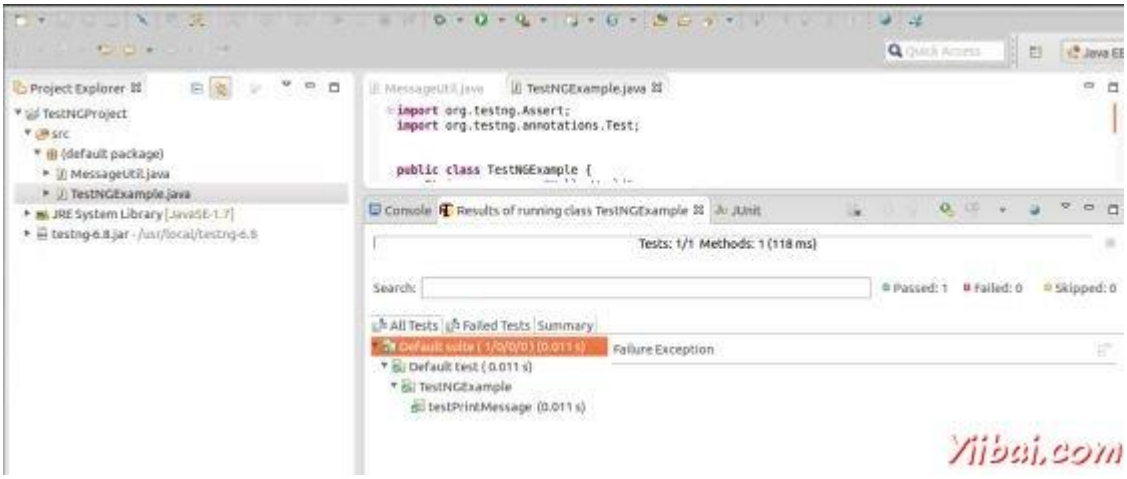
```
import org.testng.Assert;
import org.testng.annotations.Test;
public class TestNGExample {
    String message = "Hello World";
    MessageUtil messageUtil = new MessageUtil(message);
    @Test
    public void testPrintMessage() {
       Assert.assertEquals(message,messageUtil.printMessage());
   }
}
```

下面应该是项目结构：



最后，通过右击程序和 TestNG 的运行验证程序的输出。

验证结果。


Project Explorer
TestNGProject
src
(default package)
MessageUtil.java
TestNGExample.java
JRE System Library [JavaSE-1.7]
testng-6.8.jar - /usr/local/testng-6.8
MessageUtil.java
TestNGExample.java
import org.testng.Assert;
import org.testng.annotations.Test;
public class TestNGExample {
Console
Results of running class TestNGExample
JUnit
Tests: 1/1 Methods: 1 (118 ms)
Search:
Passed: 1
Failed: 0
Skipped: 0
All Tests
Failed Tests
Summary
Failure Exception
Default test (0.011 s)
TestNGExample
testPrintMessage (0.011 s)
Yiibai.com