## 使用 JUnit 进行单元测试 - 教程

### tanyuanji@126.com

|  |
| --- |
| **版本历史** |

**JUnit**

该教程主要讲解 JUnit 4.x 版本的使用，以及如何在Eclipse IDE 中如何使用JUnit

**目录**

[使用 JUnit 进行单元测试 - 教程 1](#_Toc372635901)

[tanyuanji@126.com 1](#_Toc372635902)

[1. Junit介绍 3](#_Toc372635903)

[1.1. 简介 3](#_Toc372635904)

[2. JUnit 优点 3](#_Toc372635905)

[2.1. 对于极限编程而言 3](#_Toc372635906)

[2.2. 对于重构而言 3](#_Toc372635907)

[2.3. 其他情况 3](#_Toc372635908)

[3. 使用 JUnit 4](#_Toc372635909)

[3.1. 使用JUnit 进行单元测试 4](#_Toc372635910)

[3.2. 可以使用的 JUnit 注解列表 4](#_Toc372635911)

[3.3. 断言(Assert)语句 5](#_Toc372635912)

[3.4. 创建测试套件 6](#_Toc372635913)

[3.5. 在 Eclipse 之外执行JUnit 6](#_Toc372635914)

[4. 安装JUnit 7](#_Toc372635915)

[4.1. Eclipse集成JUnit 7](#_Toc372635916)

[4.2. 下载Junit 7](#_Toc372635917)

[5. Eclipse对JUnit的支持 7](#_Toc372635918)

[5.1. 创建 JUnit 测试 7](#_Toc372635919)

[5.2. 运行 JUnit 测试 8](#_Toc372635920)

[5.3. JUnit静态导入的设置 9](#_Toc372635921)

[5.4. 创建JUnit 测试套件向导 10](#_Toc372635922)

[5.5. 异常的测试 12](#_Toc372635923)

## 1. Junit介绍

### 1.1. 简介

JUnit是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架（regression testing framework）。Junit测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。Junit是一套框架，继承TestCase类，就可以用Junit进行自动测试了。

JUnit是一个Java语言的单元测试框架。它由Kent Beck和Erich Gamma建立，逐渐成为源于Kent Beck的sUnit的xUnit家族中为最成功的一个。 JUnit有它自己的JUnit扩展生态圈。

多数Java的开发环境都已经集成了JUnit作为单元测试的工具。

## 2. JUnit 优点

### 2.1. 对于极限编程而言

要求在编写代码之前先写测试，这样可以强制你在写代码之前好好的思考代码（方法）的功能和逻辑，否则编写的代码很不稳定，那么你需要同时维护测试代码和实际代码，这个工作量就会大大增加。因此在极限编程中，基本过程是这样的：构思－> 编写测试代码－> 编写代码－> 测试，而且编写测试和编写代码都是增量式的，写一点测一点，在编写以后的代码中如果发现问题可以较快的追踪到问题的原因，减小回归错误的纠错难度。

### 2.2. 对于重构而言

其好处和极限编程中是类似的，因为重构也是要求改一点测一点，减少回归错误造成的时间消耗。

### 2.3. 其他情况

我们在开发的时候使用junit写一些适当的测试也是有必要的，因为一般我们也是需要编写测试的代码的，可能原来不是使用的junit，如果使用junit，而且针对接口（方法）编写测试代码会减少以后的维护工作，例如以后对方法内部的修改（这个就是相当于重构的工作了）。另外就是因为junit有断言功能，如果测试结果不通过会告诉我们哪个测试不通过，为什么，而如果是像以前的一般做法是写一些测试代码看其输出结果，然后再由自己来判断结果使用正确，使用junit的好处就是这个结果是否正确的判断是它来完成的，我们只需要看看它告诉我们结果是否正确就可以了，在一般情况下会大大提高效率。

## 3. 使用 JUnit

### 3.1. 使用JUnit 进行单元测试

JUnit 4.x 引用了注解的方式来对方法进行测试。通常测试方法都是被包含在一个 *测试类* 中

使用下面的代码来创建一个单元测试 . 通过如下操作： File → New → JUnit → JUnit Test case.

@Test

**public** **void** testMultiply() {

MyClass tester = **new** MyClass();

*assertEquals*( "10 x 5 must be 50 " , 50, tester.multiply( 10, 5 ) );

}

JUnit 假设可以使用任意的顺序执行测试方法。因而测试不依赖于其他的测试方法。

可以使用注解 @org.junit.Test 来修饰方法，以创建一个测试方法。然后使用预期的值 与 真实的值来检查方法是否正确。

你可以使用 Eclipse IDE方式来运行一个测试类，通过如下操作：右键点击需要测试的类并且选择 Run → Run As → JUnit Test。

也可以使用类 org.junit.runner.JUnitCore运行测试类，而不依赖于Eclipse。

该方法可以标记需要测试的类，选择要执行的测试。只有被选定的测试将会执行。

### 3.2. 可以使用的 JUnit 注解列表

下表给出了JUnit 4.x可以使用的注解的一个概述.

**表 1. JUnit4注解**

| **注解** | **描述** |
| --- | --- |
| @Test  public void method() | @Test 注解代表方法是一个测试方法。 |
| @Test (expected = Exception.class) | 表示预期会抛出Exception.class 的异常 |
| @Test(timeout=100) | 表示预期方法执行不会超过 100 毫秒. |
| @Before  public void method() | 表示该方法在每一个测试方法之前运行，可以使用该方法进行初始化之类的操作 |
| @After  public void method() | 表示该方法在每一个测试方法之后运行，可以使用该方法进行释放资源，回收内存之类的操作 |
| @BeforeClass  public static void method() | 表示该方法只执行一次，并且在所有方法之前执行。一般可以使用该方法进行数据库连接操作，注意该注解运用在静态方法。 |
| @AfterClass  public static void method() | 表示该方法只执行一次，并且在所有方法之后执行。一般可以使用该方法进行数据库连接关闭操作，注意该注解运用在静态方法。 |
| @Ignore | 表示该方法忽略。一般在低层代码有所改动，但是未实现，可以暂时忽略掉。也可以忽略掉执行时间过长的测试。 |

### 3.3. 断言(Assert)语句

JUnit在类 Assert提供了一些静态的方法来测试某些条件。这些方法一般以 assert开头,允许你指定消息( message )，预期的值 ( expected )，真实的值 ( actual )。一个断言方法比较返回的真实值和预期值，如果比较失败则会抛出 AssertionException 异常

下表给出了JUnit 4.x可以使用的断言方法的一个概述. 参数中的[]表示可选。

**表 2. 断言方法**

| **语句** | **描述** |
| --- | --- |
| fail(String) | 表示让测试方法失败。一般用于某些代码执行不到，或者在fail代码之前就失败了。参数是可选的。 |
| assertTrue([message], boolean condition) | 表示给出的条件断言为真。. |
| assertFalse([message], boolean condition) | 表示给出的条件断言为假。. |
| assertEquals([String message], expected, actual) | 表示测试两个值的内容相等。注意数组是检查引用相同而不是数组的内容。 |
| assertEquals([String message], expected, actual, tolerance) | 测试float或double值匹配。容差是小数点后的位数，必须是相同的。*assertEquals*( 1.1f, 1.1111d, 1); 。 |
| assertNull([message], object) | 表示检查的对象为空。 |
| assertNotNull([message], object) | 表示检查的对象为非空。 |
| assertSame([String], expected, actual) | 表示两个变量指向同一个引用。 |
| assertNotSame([String], expected, actual) | 表示两个变量指向非同一个引用。 |

### 3.4. 创建测试套件

如果你须有多个测试单元，可以合并成一个测试套件进行测试。并且可以按照指定的顺序执行所有的测试类。

下面的代码示例创建了一个测试套件来执行两个测试单元。如果你要添加其他的测试单元可以使用语句 @Suite.SuiteClasses 进行注解。

**package** cn.nsccsz;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Suite;

**import** org.junit.runners.Suite.SuiteClasses;

@RunWith( Suite.**class** )

@SuiteClasses( { JUnit1Test.**class**, StringUtilTest.**class** } )

**public** **class** JSuit {

}

### 3.5. 在 Eclipse 之外执行JUnit

JUnit 提供了使用标准的Java代码形式执行测试，而不依赖于Eclipse。通常使用类

org.junit.runner.JUnitCore 类提供的静态的方法runClasses()。

下面的代码例子展示了如何使用 JUnitCode 来测试。

**public** **class** MyTestRunner {

**public** **static** **void** main(String[] args) {

Result result = JUnitCore.*runClasses*(JUnit1Test.**class**,

StringUtilTest.**class**);

**for** (Failure failure : result.getFailures()) {

System.*out*.println(failure.toString());

}

System.*out*.println( "执行方法个数:" + result.getRunCount() );

System.*out*.println( "执行时间:" + result.getRunTime() );

}

}

可以使用Ant脚本的方式或者Java命令方式，例如：

java -cp .;D:\Users\Administrator\.m2\repository\junit\junit\4.8.2\junit-4.8.2.jar cn.nsccsz.MyTestRunner

## 4. 安装JUnit

### 4.1. Eclipse集成JUnit

Eclipse允许你使用不同版本Junit进行集成。一般Eclipse已经默认集成了JUnit3.x和JUnit4.x 不同的版本

### 4.2. 下载Junit

可以从下面的官网下载 JUnit4.x.jar. 下载好的 *junit-4.\*.jar* 添加到classpath.

http://junit.org/

## 5. Eclipse对JUnit的支持

### 5.1. 创建 JUnit 测试

* 为当前的类创建测试类

右键单击选中的类，在Package Explorer视图中右键选中 New → JUnit Test Case.

* 还可以使用 JUnit 想到进行创建，使用下面的方法

File → New → Other... → Java → JUnit.

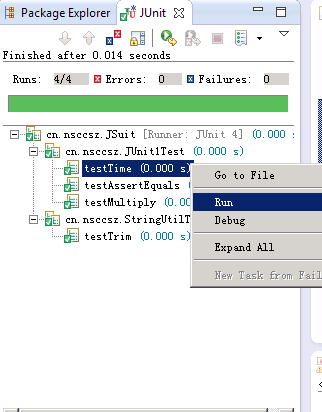
### 5.2. 运行 JUnit 测试

要运行测试类，选中包含测试方法的类。右键点击 Run-as → JUnit Test，将会执行所有的测试方法。

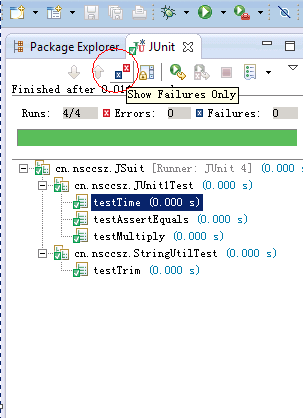
Eclipse 提供了快捷键运行测试类，快捷键为 **Alt**+**Shift**+**X,** **,T**。如果你光标选中了某个方法，则会只运行选中的方法。

To see the result of an JUnit test, Eclipse uses the JUnit view which shows the results of the tests. You can also select individual unit test in this view , right-click them and select Run to execute them again.

Eclipse提供了JUnit *view*视图来显示测试运行的结果。在这个视图中你可以选中个别的测试类，右键选中来Run运行它们。

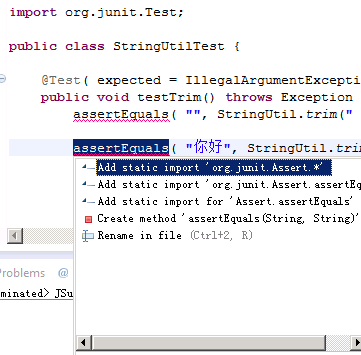


默认的情况下该视图显示所有的测试信息。你可以通过如下配置设置只显示失败的测试。



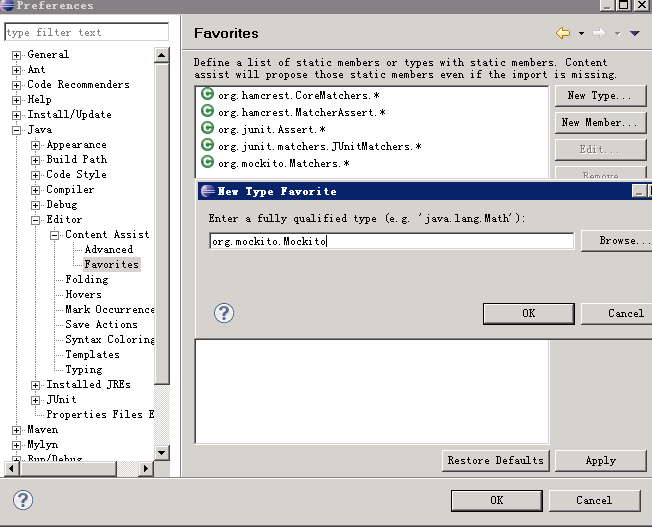
### 5.3. JUnit静态导入的设置

Eclipse 支持JUnit 的静态导入功能，使用 Ctrl+1 或者 Content Assist功能快捷键 。查看下图。



Content Assist *对*静态导入的配置。

* Window → Preferences
* Java → Editor → Content Assist → Favorites

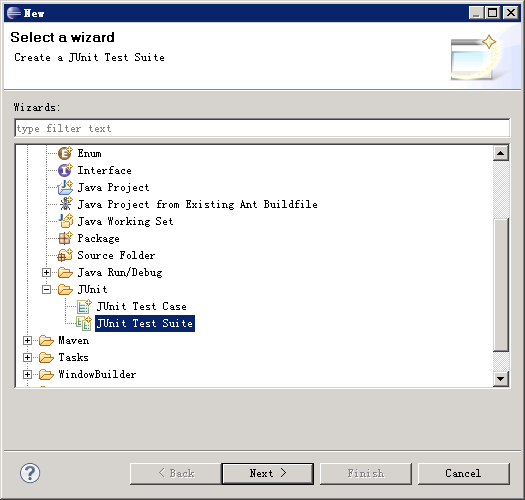


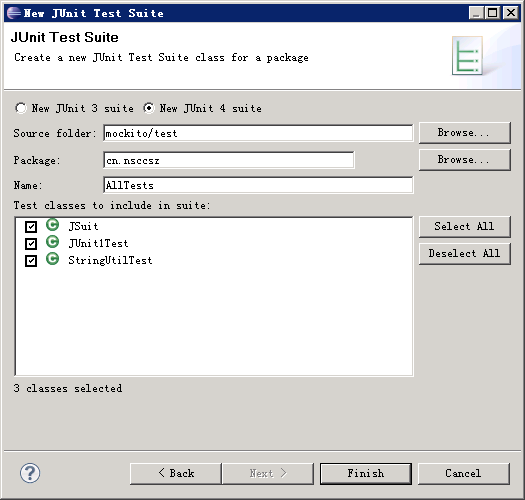
### 5.4. 创建JUnit 测试套件向导

To create a test suite in Eclipse you select the test classes which should be included into this in the Package Explorer view, right-click on them and select New → Other... → JUnit → JUnit Test Suite.

在Eclipse中创建测试套件的方法

右键选中需要创建套件的包，选中 New → Other... → JUnit → JUnit Test Suite.





### 5.5. 异常的测试

注解 @Test (expected = Exception.class) 限制了只能测试一个异常。可以使用下面的代码测试更多异常。

@Test

**public** **void** testExceptions() {

**try** {

StringUtil.*mustThrowException*();

*fail*();

} **catch**( Exception e ) {

// expected.

}

**try** {

StringUtil.*mustThrowException*();

*fail*();

} **catch**( Exception e ) {

// expected.

}

}