开源测试框架及工具
之不完全手册

作者：莫映
http://morningspace.51.net/
mailto:morningspace@126.com
2006 年 10 月

目 录

开源，以时代之名

测试，是软件全生命周期中的重要一环。从传统的软工方法，到新兴的敏捷方法，尽管角度各有千秋，侧重有所不同，但是测试从来都不曾被忽视过。从方法学的支撑，到各种工具的使用，人们在测试技术上的投入也从来都不曾停止过。

在当今异常活跃的开源社区里，优秀的测试工具和测试框架层出不穷。这其中，又尤以 xUnit 家族最为耀眼。当然，在这个庞大的家族中最为璀璨的明珠自然是非 JUnit 莫属了。JUnit 测试框架系出名门，精巧的设计加之简单易学的特点，自出道至今，已经赢得了无数开发者的青睐，就连大厂商们也纷纷在各自的 IDE 中内置了对它的支持。如今，JUnit 已然成为以白盒为特征的单元测试事实上的标准。

在众多开源测试框架中，以 jMock 为代表的一类测试工具独树一帜。因为它们关注的是测试领域中一个特别的话题——mock 测试。借助以假乱真的 mock 对象，测试代码可以脱离真实环境而单独运行。这使得许多基于外部资源和其他模块的功能代码也能够方便的进行独立测试。jMock 能够帮助开发者快速有效的装配 mock 对象，从而提高测试代码的编写效率和质量。

在当前的应用领域，基于 web 的应用系统十分普及。如何针对 web 应用，进行有效的自动化验收测试和性能测试是许多测试工具致力于解决的问题。这其中就包括了后起之秀 Selenium 和开源前辈 JMeter。前者是正值蓬勃发展阶段的自动验收测试框架，而后者则是首选的免费性能、负载测试工具。

开源测试框架不断涌现，商业测试工具自然也不甘示弱。相比于开源软件，商业软件更为专业，也更加完善。像 Rational、Mercury、Compuware、Segue 这样的专业化工具厂商，都拥有一套涵盖软件测试方方面面的完整工具集。从白盒测试到黑盒测试，从静态分析到动态分析，包罗万象，应有尽有。当然，这些工具更多的是向“重型”方法靠拢，强调的是软件质量的保证。

上述内容便是笔者即将要向诸位讲述的系列故事的梗概。希望这本“不完全手册”能带给大家测试领域的一个全景图。最后面的推荐书目和相关资源网站作为这一系列文章的延伸，在给大家以参考的同时，也弥补了篇幅所限的遗憾 **J**

JUnit

什么是 JUnit？

JUnit 是一个用 Java 编写而成的单元测试框架，其早先的作者是大名鼎鼎的 Erich Gamma 和 Kent Beck（后文对此还会详述）。利用 JUnit，程序员可以通过编写简单的测试代码，方便的进行白盒测试，亦即：在了解被测代码如何工作的前提下，对其内部结构的正确性进行自动化的测试。

在 JUnit 的官方主页上，还可以找到有关 JUnit 的更为正统的释义：
JUnit 是一个开放源代码的简单框架，用来编写和运行可重复的测试（注：也被称为可回归测试）。它是致力于单元测试框架的 xUnit 架构的一种实现。其中包含了：

- 用于检测预期结果的 Assertions
- 用于共享测试用数据的 Test Fixtures
- 用于运行测试的 Test Runners

在没有 JUnit 的年代，本分的程序员也会对自己开发的代码编写测试程序。但是，这种“ad hoc”的手段多缺乏通用性，无法重用。出自名家手笔的 JUnit，强大而趁手。遵照几条简单易学的规则写就的测试代码，其中富含了各种断言，在 IDE 环境下，只消鼠标轻轻点击，便可一蹴而就——测试成功与否全凭一个直观的“green bar”或是“red bar”。程序员的目标很简单，那就是——“Keep the bar green to keep the code clean”。

不仅如此，JUnit 的出现还将 Java 程序员们带入了敏捷开发和测试驱动的时代。通过测试代码的快速反馈来驱动开发过程，业已成为敏捷开发者们编写单元测试的首选方法。而“测试”与“重构”交替进行的“敏捷韵律操”也已经为大家所熟识。JUnit 让众多程序员更加认可和信赖了敏捷开发，从这一点来看，JUnit 的出现对技术社群的影响，已非一个简单的单元测试框架这么简单了。

最新版本及新版本最大变化

目前大家所使用的 JUnit 版本多是 3.8.x，不过今年初 JUnit 发布了 4.0 版本，紧接着又在 5 月份发布了最新的 4.1 版。新版 JUnit 以 Java 5.0 为支撑，使用了像 annotations，static import 这样的新语法特性。较之以往的版本，变得更加简洁，更加丰富，更加易于使用。具体而言，JUnit4 提供的特色大致包括如下：

- 测试类不必再从 junit.framework.TestCase 派生了；
- 测试方法也不必再以“test”作为前缀，而是代之以@Test 注解来标示；
- 作为 Fixtures 的 setUp 与 tearDown 也不再强制使用这两个方法名了，只要在任何方法名称前冠以@Before 或@After，即可达到一样的效果；
- 对 setUp/tearDown 的一大改进还包括，可以限定二者只在整个 test case 范围内执行一次，这是通过@BeforeClass 和@AfterClass 注解达成的；
- @Test 注解还可以带上 timeout 参数和 expected 参数，前者代表测试方法超过指定时间即被认为失败，后者则声明了预期被抛出的异常类型；

此外，为了和以前版本的 Test Runner 兼容，JUnit4 提供了一个 JUnit4Adapter。有了它，用 JUnit4 写的测试代码就可以运行于旧版本的 Test Runner 下了。当然，以前写的测试代码在 JUnit4 的 Test Runner 里是可以直接运行的。耳听为虚，眼见为实。让我们以一个简单的例子来给大家演示一下新特性的使用方法：

```
package example.junit4;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
import junit.framework.JUnit4TestAdapter;

public class LibraryTest {    不必从 TestCase 派生
    private Library library;

    @BeforeClass public void runOnceBeforeAllTests() {    仅在所有测试方法执行前执行
        // …
    }

    @Before public void runBeforeEachTest() {    不必以 setUp 作为方法名称
        library = new Library();
    }

    @Test public void bookAvailableInLibrary () {    不必以 test 打头
        boolean result = library.checkAvailabilityByTitle(“Webster’s Dictionary”);
        assertEquals (“Our Library should have the standard Dictionary”,
                true,
                result);
    }

    @Test(expected=BookNotAvailableException.class)    指定预期抛出的异常
    public void bookNotAvailableInLibrary(){
        library.checkAvailabilityByTitle(“Some book that does not exist”);
```

```
    }

    @After public void runAfterEachTest() {    不必以 tearDown 作为方法名称
        Library = null;
    }

    @AfterClass public void runAfterAllTests() {    仅在所有测试方法执行后执行
        // ...
    }

    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(LibraryTest.class);
    }
}
```

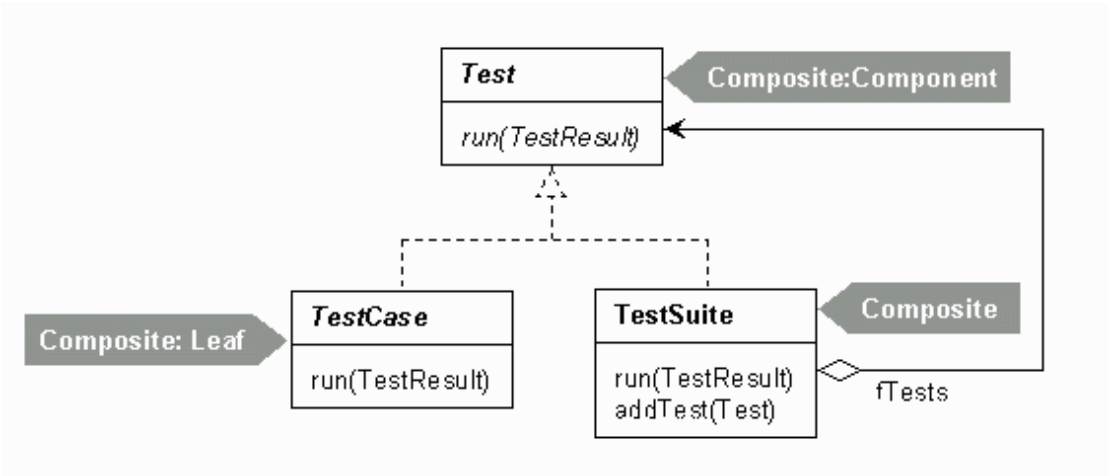
软件组织架构及应用

有关 JUnit 组织方式与设计架构的讨论，应当首推 Erich Gamma 与 Kent Beck 合写的“JUnit: A Cook’s Tour”（可以在 JUnit 的官方主页上找到）。虽然这篇文章是针对 JUnit 3.8.x 写的，但却颇具研习的价值。作者在文中带领着大家从零开始，通过逐个运用模式，最终构造出完整的 JUnit。此处，笔者就着 JUnit 的主体设计，为大家介绍两个出现其中的常用模式。更为详细的内容，请读者参阅相关资源[1]。值得一提的是，小巧精致的 JUnit 是学习设计模式的绝好素材。

Composite 模式

在 JUnit 中，有一个贯穿始终的设计模式，那就是 Composite 模式。在《设计模式》一书中对该模式是这么解释的：将对象组合成树形结构以表示“部分－整体”的层次结构。Composite 使用户对单个对象和组合对象的使用具有了一致性。在 JUnit 的框架里，测试类分为两种，某些测试类代表单个测试，称为 TestCase，另一些则由若干测试类组合而成，称为 TestSuite。彼此相关的 TestCase 共同构成一个 TestSuite，而 TestSuite 也可以嵌套包含。两者分别对应了 Composite 模式中的 Leaf 和 Composite。如下所示，TestCase 和 TestSuite 共同派生自 Test 接口：

[1]笔者在几年前也曾对 JUnit 的 C++翻版——CppUnit 做过详细的剖析，有兴趣的读者可以访问这个连接：http://morningspace.51.net/resource/cppunit/cppunit_anno.html



图：Composition 模式

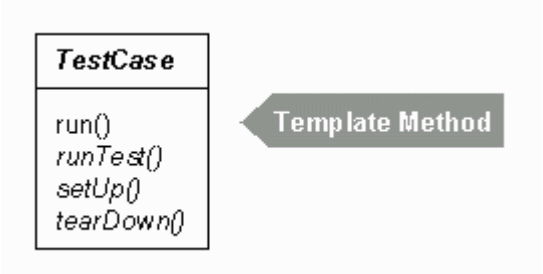
Template Method 模式

我们知道，在每一个测试方法执行之前，可能需要准备测试用的数据，而在测试方法执行完毕之后，也可能需要做些清理工作。借助 Template Method 模式，JUnit 为我们提供了所谓的 Test Fixture，使得多个测试方法可以共享同样的测试环境，并且每个测试方法的执行环境彼此独立，互不影响。

在 TestCase 的 run 方法中，执行逻辑大致如下：

```
public void run() {
      setUp();
      runTest();
      tearDown();
}
```

其中，setUp 用于在测试方法执行之前初始化测试环境，tearDown 则在测试方法执行之后清理测试环境。如此，每个测试方法在执行前都会重新建立测试环境，使得当前测试方法的执行不依赖于其他测试方法。在 TestCase 中，setUp 与 tearDown 的默认实现不做任何事情。具体实现逻辑要由测试编写人员在 TestCase 的子类中来决定。这正是 Template Method 模式的主旨所在：在父类中定义算法执行步骤，将步骤的具体执行逻辑推迟到子类中实现。实际上，该模式在一般的 framework 中十分常见。



图：Template Method 模式

主要人物及相关故事

JUnit 最初是由 Erich Gamma 和 Kent Beck 二人合力打造而成的。说起这两位大师级人物，想必大家都不会陌生。

Erich Gamma 是 Eclipse 的架构师之一，也是技术畅销书《设计模式》（Addison-Wesley, 1995）的合著者之一。该书针对软件领域通常遇到的设计问题，采用面向对象的设计思想分类理出了 23 种特定的解决方案。这本书自出版以来虽十年有余，却依旧能时常见诸 Amazon 的畅销书排行榜。正是这本经久不衰的里程碑式的书籍，让 Erich Gamma 跃上了软件业界的舞台，位列大师之中。他与该书的另三位作者常被拥趸们戏称为“Gang of Four”。Erich Gamma 目前是 IBM 的一名杰出工程师，在瑞士苏黎世的 IBM Object Technology International（OTI）实验室工作。他还担任着 Eclipse 社区的领导工作，负责 Eclipse 平台上与 Java 开发相关的事宜。

JUnit 的另一位作者 Kent Beck 也非等闲之辈。他被人们称为软件开发方法学的泰斗，长期致力于软件工程的理论研究和实践。作为极限编程（eXtreme Programming）和测试驱动开发的创始人，软件业界最富创造力，最有口碑的领导者之一，Kent Beck 极力推崇设计模式、极限编程和测试驱动开发，同时他还是多部经典技术书籍的作者或合作者，包括：《Smalltalk Best Practice Patterns》（Prentice Hall，1996）、《解析极限编程——拥抱变化》（Addison-Wesley，2nd Edition，2004）、《规划极限编程》（Addison-Wesley，2000），《Test-Driven Development: By Example》（Addison-Wesley，2002），《Contributing to Eclipse》（Addison-Wesley，2003，该书的另一位作者即是 Gamma）。Kent Beck 目前是 Three Rivers Institute（TRI）的总裁，TRI 主要从事技术和商业接合的应用研究。

从某种程度上讲，两位大师的联袂注定了 JUnit 在软件开发领域不可撼动的地位和影响力。自 1998 年诞生以来，根正苗红的 JUnit 深受 Java 程序员们的好评：2001 及 2002 年连续两届“Java World 编辑选择奖”，2003 年“Java World 最佳测试工具”，2003 年“Java Pro 最佳 Java 测试工具”。现在，JUnit 已然成为 Java 社区单元测试的“事实标准”。不仅如此，JUnit 还影响到了许多其他技术领域和技术平台，以至于人们会将其称为庞大的“xUnit 家族”。在这里，我们可以列出一串长长的名单：HttpUnit，HtmlUnit，DBUnit，CppUnit，NUnit，JUnitPerf，accessUnit，AS2Unit，CUnit，……，这其中有些是 JUnit 的直接“翻版”，而另一些则依托于 JUnit 之上，有着特殊的用途。[2]

Step by Step

一、准备工作

JUnit 的官方网站提供了最新版本的 JUnit 下载。不过，由于 JUnit 已经相当普及，一般的 IDE 开发环境，比如：Eclipse，JBuilder，NetBeans，IntelliJ 等等，都缺省提供了对 JUnit 的支持，而不必单独去下载。若非使用 JUnit4 的新功能（目前流行的 IDE 开发环境多以支持 3.8.x 版

[2] 在 http://www.xprogramming.com/software.htm，读者可以找到 xUnit 测试框架的更多成员。

本为主），否则可以直接使用 IDE 中内置的 JUnit。此外，由于一般的 IDE 都有着良好的 GUI 功能，因此 JUnit 的使用会变得更加方便。

后文将以 Eclipse 3.1.2 为例，向读者演示 JUnit 的使用方法，该版本缺省内置了 3.8.1 版的 JUnit 插件。读者可以从 Eclipse 的官方网站（http://www.eclipse.org/）下载到最新版本的 Eclipse。至于其他的 IDE 环境，使用方法大同小异，读者可以查阅相关资源。

在这里，我们使用一个简单的具有绝对值计算功能的数学运算工具类作为被测试对象，在演示如何使用 JUnit 的同时，也向读者演示一个简单的测试驱动开发过程。限于篇幅，有关测试驱动开发的知识请读者参阅其他相关资料。

二、MathUtil 的“需求”

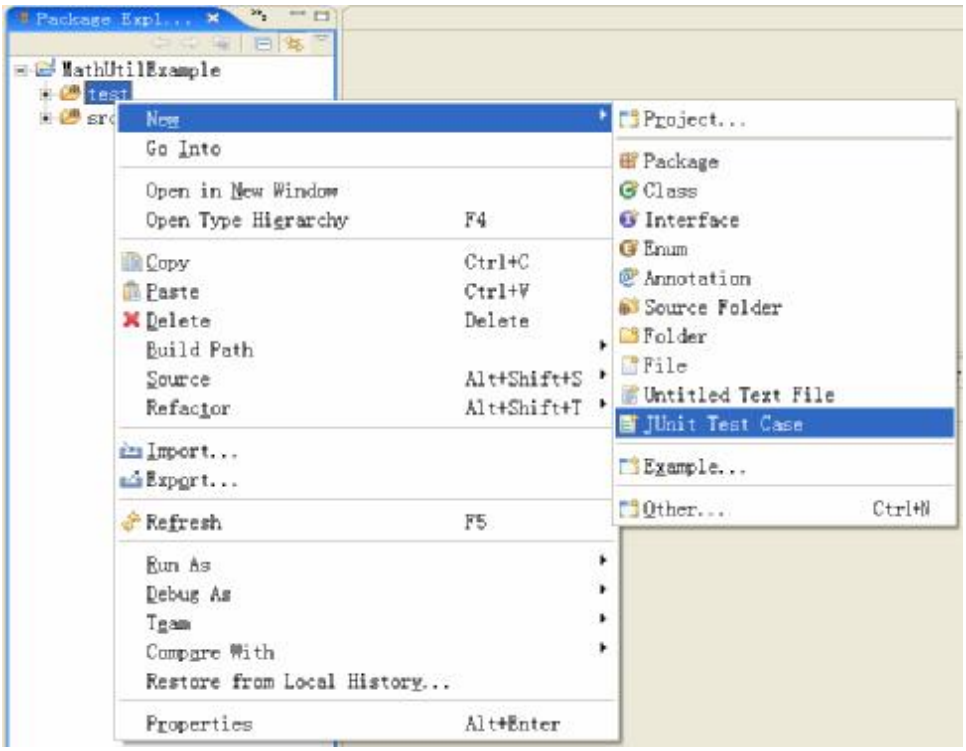
假设我们要实现一个提供了若干常用数学运算功能的工具类，名叫 MathUtil。首先，我们为其加入一个绝对值计算函数。不过，在开始编写实现代码之前，我们依据测试先行的建议，先来考虑一下绝对值运算的“需求”。一个绝对值运算，应该包含如下几项“需求”：

- 当给定一个正数时，应该返回同样大小的正数；
- 当给定一个负数时，应该返回该负数的相反数；
- 当给定 0 时，应该返回 0；

下面，我们即将开始编写第一个测试用例，测试用例的目的就是要将上述“需求”以代码的形式表达出来。不过此前，我们还需要先在 Eclipse 中创建一个名叫 MathUtilExample 的新 Java 工程，并在工程中新建两个目录：用于存放测试代码的 test 目录和用于存放源文件的 src 目录。

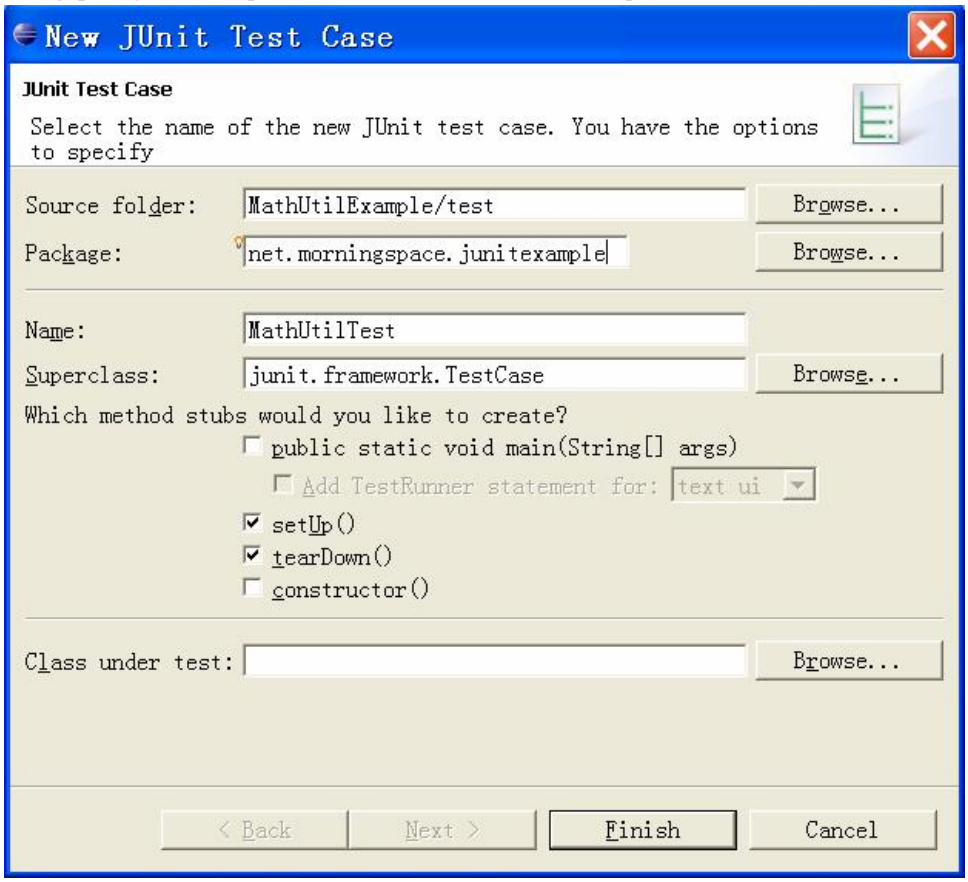
三、编写第一个 TestCase

在工程中选中 test 目录，右键点击，选择 New->JUnit Test Case。此时，Eclipse 会询问是否在工程中加入 junit.jar，点击确定即可。



图：新建 JUnit TestCase

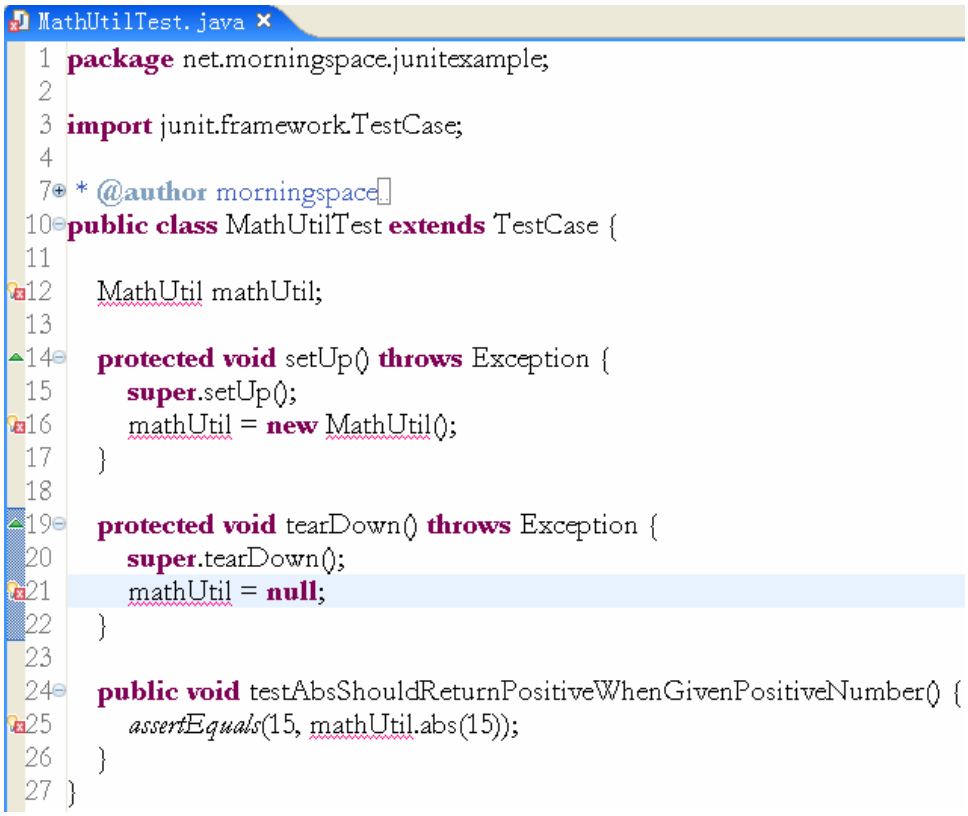
在随后打开的新建 Test Case 对话框中，我们按照对话框的提示新建一个测试类：net.morningspace.junitexample.MathUtilTest。然后选中 setUp()和 tearDown()，并点击“Finish”。



图：新建 Test Case 对话框

现在我们可以开始在新建的 MathUtilTest 类中添加测试代码了。根据前面所列的“需求”，

我们将在 MathUtilTest 中逐步添加测试方法，随着开发的推进，依次满足每项“需求”。首先我们添加第一个测试方法，以满足正数的情形：



图：添加第一个测试方法

此处，我们使用了 JUnit 的断言 assertEquals。通过该断言，我们确保了当传入的数值为 15 时，返回值必是 15。另外，利用 setUp()，我们在每个测试方法执行前都创建了新的 MathUtil 实例，同时利用 tearDown()，在执行完毕后又将该实例销毁。编辑窗口左侧的红叉告诉我们，此时还没有创建 MathUtil 类及其 abs 函数。因此，接下来我们要新建一个 MathUtil 类，并以最简单的方法让测试通过。利用 Eclipse 提供的便捷功能，我们可以很方便的新建 MathUtil 类：

图：新建 MathUtil 类

```
1  package net.morningspace.junitexample;
2
5  * @author morningspace
8  public class MathUtil {
9
10   public int abs(int number) {
11     return number;
12   }
13 }
```

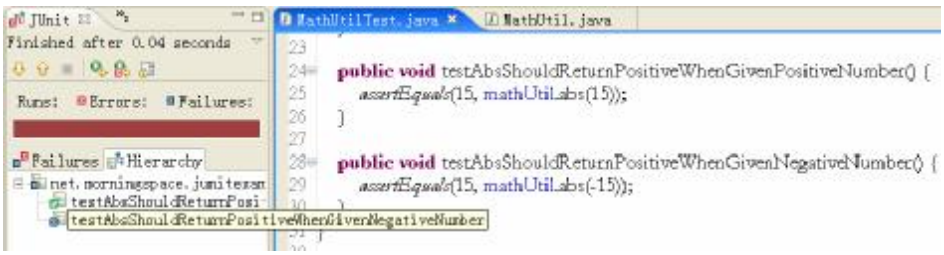
图：尝试让测试通过的最简单做法，未必是正解

随后我们运行一下单元测试，选择 Run->Run As->JUnit Test，和预想的一样，测试顺利通过了。

图：第一个测试方法通过

JUnit 单元测试的执行速度一般都很快，所以运行后立刻就能看到测试结果。JUnit 以其经典的“green bar/red bar”来表示测试通过与否。绿色表示测试通过，红色表示测试失败，假如有多个测试方法时，只要有一个测试方法没有通过，就会显示红色并列出未通过测试的方法。

接下来我们再加入第二个测试方法，以满足负数的情况。再次执行单元测试，此时的结果显示，新添加的测试方法没有通过。

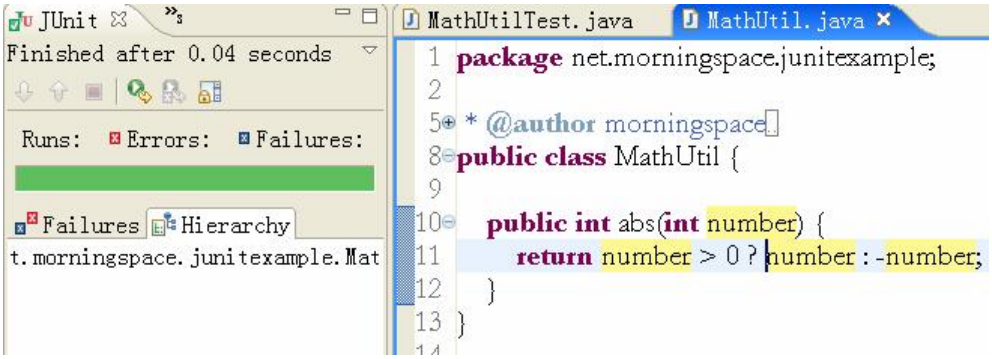


图：第二个测试方法没有通过

为了让测试通过，我们不得不修改 MathUtil.abs()原来的实现，不过这次依然采用的是让测试通过的最简单办法，我们试着在原有返回值前增加一个负号：

```
public int abs(int number) {
    return –number;
}
```

很遗憾，这次没有那么走运，再次运行测试后发现，结果依然是“red bar”。不过，与上次有所不同的是，这回没有通过测试的不是新增的测试方法，而是前一个已经通过了的测试方法。这一结果启发了我们，应该考虑引入一种 if…else 的结构，来同时应对正数和负数这两种情况：



图：前两个测试方法顺利通过

这一回，两个测试都通过了。最后，我们加入第三个测试方法，以验证传入参数为 0 的情况。结果显示为“green bar”。至此，一个简单的测试驱动过程完成了。

四、小结

在上面的例子里，我们简单演示了在 Eclipse 环境下，如何利用 Eclipse 与 JUnit 的良好集成能力，以测试驱动的方式，编写 JUnit 单元测试。也向大家演示了如何使用 setUp/tearDown，以及如何使用断言。不过由于例子十分的简单，所以无法演示 JUnit 的所有使用技巧。更为高级的使用方法，读者可以查阅相关资源。另外，后文的 FAQ 中对此也会有所涉及。

五、与 Ant 结合使用

与 JUnit 一样，另一个开源项目——自动化编译部署工具 Ant，也是大多数 Java 程序员的必备工具。利用 Ant，我们可以将许多繁琐工作自动进行，从而极大的简化了开发过程。此外，目前像极限编程这样的敏捷方法中时常倡导的持续集成，也要求有能够持续执行的自动化编译、测试和部署机制。而将 Ant 与 JUnit 结合使用，会使这一机制发挥更大的威力，产生更大的价值：只消简单的配置，从编译代码到打包部署，从自动测试到报告生成，可谓一气呵成。

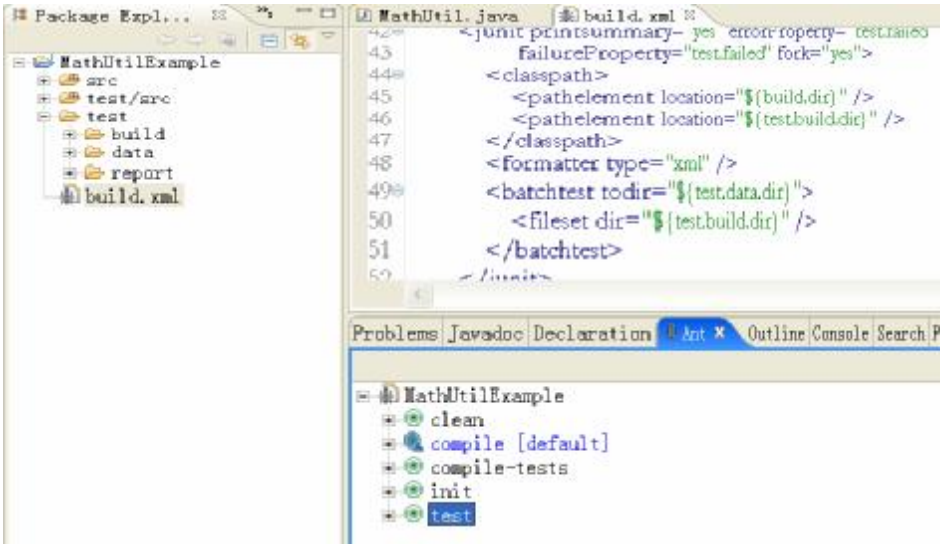
因为 Ant 早已广为流传，所以主流 IDE 对它也都有很好的内置支持，使用起来十分的方便。这里，我们仍已 MathUtil 为例，为大家演示一下在 Eclipse 环境中，如何编写 Ant 脚本，利用 Ant 提供的相关 task 自动执行单元测试，并生成测试报告。篇幅所限，有关 Ant 的详细使用方法，推荐读者阅读《Java Development with Ant》一书（Manning，2002）。

在 MathUtilExample 工程的根目录下新建一个 build.xml 文件，该文件的内容摘要如下：

```
<project name="MathUtilExample" basedir="." default="compile" >
    ... ...
    <target name="compile" depends="init" >    编译 Java 源文件
        <javac srcdir="${src.dir}" destdir="${build.dir}" >
            <include name="**/*.java"/>
        </javac>
    </target>
```

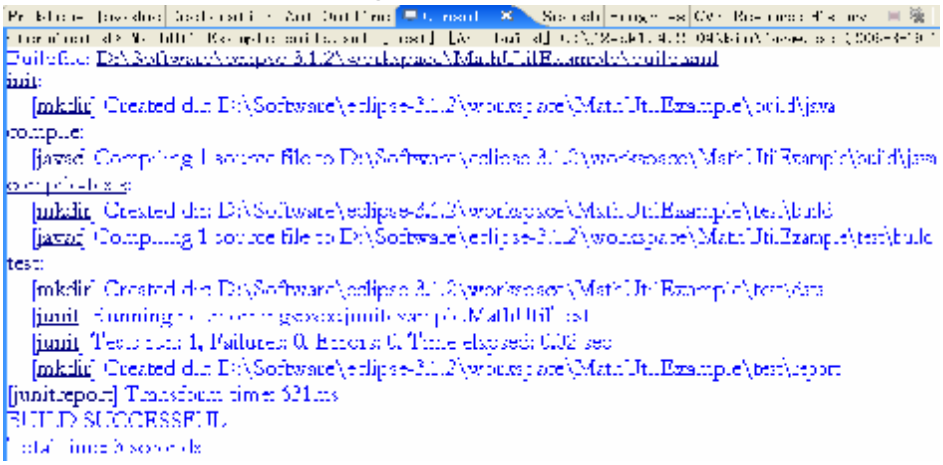
```
    <target name="compile-tests" depends="compile">   编译测试代码
        <mkdir dir="${test.build.dir}"/>
        <javac srcdir="${test.src.dir}" destdir="${test.build.dir}">
            <classpath>
                <pathelement location="${build.dir}" />
            </classpath>
            <include name="**/*.java"/>
        </javac>
    </target>

    <target name="test" depends="compile-tests">   执行单元测试
        <mkdir dir="${test.data.dir}" />   创建用于保存单元测试中间结果的目录

        <junit printsummary="yes" errorProperty="test.failed"
                  failureProperty="test.failed" fork="yes">
            <classpath>
                <pathelement location="${build.dir}" />
                <pathelement location="${test.build.dir}" />
            </classpath>
            <formatter type="xml" />
            <batchtest todir="${test.data.dir}">   批量执行指定目录下的单元测试
                <fileset dir="${test.build.dir}" />
            </batchtest>
        </junit>

        <mkdir dir="${test.report.dir}" />   创建用于存放测试报告的目录

        <junitreport todir="${test.report.dir}">   生成测试报告
            <fileset dir="${test.data.dir}">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="${test.report.dir}"/>
        </junitreport>

        <fail message="Tests failed. Check reports"   如果测试失败则终止构建过程
                 if="test.failed" />
    </target>

</project>
```

待 build.xml 编写完后，选择 Window->Show View->Ant，打开 Ant 视窗，然后将 build.xml 文件“托拽”到该视窗中。



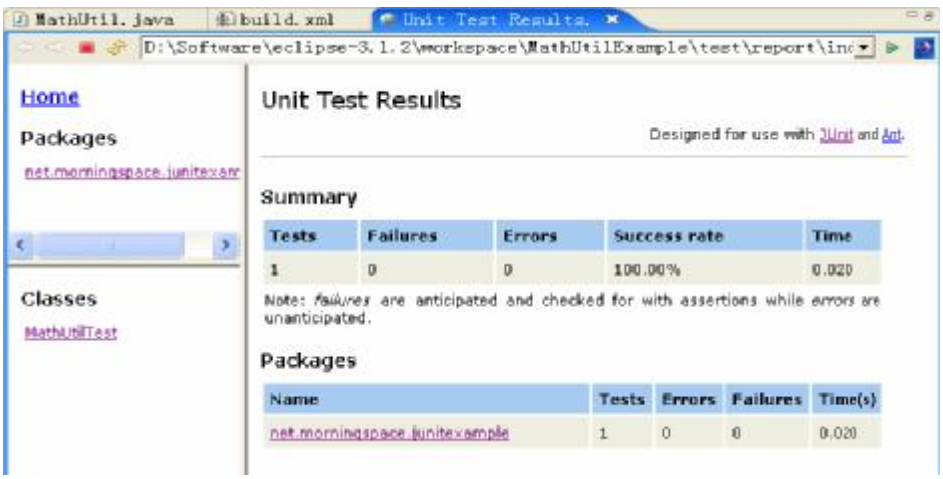
图：位于 Ant 视窗中的 build 脚本

此时，双击 Ant 视图中名为 test 的 target，即可在控制台视图中得到如下结果：



图：控制台的输出结果

此外，我们还可以在${test.report.dir}目录下找到一组漂亮的 html 格式的测试报告：



图：html 格式的测试报告

FAQ

问：从哪里可以下载到 JUnit 的最新版本？
答：最新版本的 JUnit 可以在 http://sourceforge.net/project/showfiles.php?group_id=15278 下载到。

问：在哪里可以找到 JUnit 的文档资源？
答：在 JUnit 的官方主页上，你可以找到丰富的在线文档。你还可以通过网络搜索到大量有关 JUnit 使用的资源。此外，与 JUnit 相关的技术书籍也层出不穷，对此，读者可以参考后文的推荐书目。

问：我应该把测试代码放在哪里？
答：可以把测试类与待测类放在同一个包下，这对于小规模系统而言应该足够了。例如：

```
src
    com
        .junitexample
            MathUtil.java
            MathUtilTest.java
```

如果你觉得这种方法会造成源代码目录的混乱，并且会让发布部署变得复杂，那么不妨把测试代码单独放到一个目录下，并保持与被测代码相同的包结构。例如：

```
src
    com
        junitexample
            MathUtil.java
test
    com
        junitexample
            MathUtilTest.java
```

问：什么是 Test Fixture？
答：Test Fixture 是一组测试所需的公共数据和协作对象，它为若干测试所共享。通常用测试类的实例变量（也就是数据成员）来表示。

问：Failure 和 Error 有什么区别？
答：JUnit 将测试不通过的结果分为两种类型：Failure 和 Error。其中，Failure 指的是断言结果为 false，亦即你所预期的结果没有满足，从而说明测试失败；Error 则是指意料之外的异常，这在测试运行之前是无法预期的，例如：ArrayIndexOutOfBoundsException。

问：如何测试保护类型的成员函数？
答：只要保持测试代码的包路径与待测类的包路径一致即可。保护类型的成员函数在包范围内是可见的。

问：如何测试私有类型的成员函数？

答：一般来说，测试私有类型的成员函数很可能暗示着代码需要重构，这些方法往往应该被移入另一个类中，以提高代码的重用。不过，如果你真的想测试私有类型的成员函数，并且你用的是 JDK 1.3 及其以后的版本，那么你可以用 Java 的反射机制，配合 PrivilegedAccessor[3] 来突破 Java 访问控制的限制。

问：如何编写一个测试，让其在预期异常抛出的时候通过（或失败）？
答：如果你用的是 JUnit4，可以利用@Test 的 expected 属性定义预期抛出的异常，一旦异常如期抛出，测试就通过了。假如你用的是 JUnit 3.8.x，则可以采用类似如下的方法来达到同样的目的：

```
public void testIndexOutOfBoundsException() {
  try {
    ArrayList emptyList = new ArrayList();
    Object o = emptyList.get(0);
    fail("Should throw IndexOutOfBoundsException!");
  } catch (IndexOutOfBoundsException e) {
    // gotha...
  }
}
```

假如预期异常抛出时，希望测试失败，则只需要简单的在测试方法签名中加上 throws 声明，并且确保不在方法体内 catch 异常即可。例如：

```
@Test public void testIndexOutOfBoundsExceptionNotRaised()
          throws IndexOutOfBoundsException {
          ArrayList emptyList = new ArrayList();
          Object o = emptyList.get(0);
     }
```

问：如何测试必须运行在 J2EE 容器内的代码（例如：servlets，EJBs）？
答：对 J2EE 组件进行重构，将绝大部分执行逻辑移到普通 Java 对象中（POJO），它们可以脱离容器执行，这是一种有效改善设计和系统可测试性的手段。此外，Apache Cactus 是一个开源的 JUnit 扩展框架，使用它可以模拟容器内的组件测试。

问：在什么时候应该测试 getter/setter 方法？
答：多数情况下没有必要测试 getter/setter，因为这两个方法十分简单，不会导致程序错误。假如 getter/setter 有可能导致程序运行错误，那么你可以考虑为它们编写测试代码。例如，下面的代码中，我们希望验证在 getX()调用的时候，X 的值已经通过构造函数被正确设置了。这样的测试也许是有价值的，尤其在 MyClass 有多个版本的构造函数时：

```
@Test public void testCreate() {
     assertEquals(23, new MyClass(23).getX());
}
```

问：我是否要为每个待测试的类都编写一个 TestCase？
答：用不着，虽然通常的习惯是一个待测类对应一个测试类，但这并非是必须的。TestCase 仅仅提供了一种组织测试的方法。也许开始的时候你会用一个测试类来对应待测类，但随后

[3] Source Forge 上的一个小型开源项目

你也许会发现，测试类中有一组测试方法具有共同的 Test Fixture，此时你可以将这些测试重构为一个新的测试类。

问：我应该多久运行一次单元测试？
答：尽可能频繁的运行单元测试，确保所有的单元测试都被执行。频繁的测试可以为你修改既有代码增加自信心，你不再担心因为不小心误改代码而破坏原有程序的功能了。对于较大规模的软件系统，你可以运行和目前工作相关的测试集，并且至少每天运行一次完整的测试集。

问：为什么不应该简单的使用 System.out.println()？
答：在代码中插入调试用的 System.out.println()是一种最为原始的程序调试手段。你需要通过肉眼观察输出的调试结果，来判断代码执行是否正确。这种输出结果无法用 JUnit 提供的断言机制来表达，因而也就无法充分有效的利用 JUnit。

问：为什么不应该仅仅使用调试手段？
答：人们常常使用调试的方式来深入代码内部，查看执行逻辑和变量取值情况。但是这种方式只能手工进行，本质而言，它等同于手工检查“实际值与预期值是否一致”。而且，一旦程序改动之后，我们就必须再次从头开始一步步调试。将“实际值与预期值是否一致”的判断以自动化方式执行，会弥补手工调试的不足。并且，每当这种自动判断很难编写时，往往预示着你的设计有待改进。

jMock

jMock 是什么？

jMock 是一个利用 mock 对象来测试 Java 代码的测试工具。毫不例外，jMock 也是 xUnit 家族的一员，因为它从 JUnit 发展而来，是 JUnit 的一个增强库。从 jMock 的官方网站可以下载到当前的最新版本。

mock 测试是一种常见的测试方法。通常在执行测试的时候，测试代码往往需要与一些真实对象进行交互，又或者被测代码的执行需要依赖真实对象的功能。此时，我们可以使用一个轻量级的，可控制的 mock 对象来取代真实对象，模拟真实对象的行为和功能，从而方便我们测试。jMock 便是这种方法的一种实现。

使用 jMock，我们可以很容易的快速编写出 mock 对象，而不必像以往那样，停下测试代码的编写工作，转而去写专门的 mock 对象。jMock 还允许你以一种十分灵活的方式来定义对象之间彼此交互的约束，从而更好的模拟和刻画对象间的调用关系。让测试代码变得十分简洁，同时又能很好的利用 mock 对象来达成测试意图。此外，jMock 也很容易扩展，我们可以很方便的添加自定义需求。下面我们通过一个简单的示例来看一下 jMock 的使用：

```
package name.nona.test.jmock;

import java.sql.SQLException;
```

```
import java.sql.Statement;
import org.jmock.Mock;
import org.jmock.MockObjectTestCase;

public class SqlOperationTest extends MockObjectTestCase {
      public void testSqlOperation () throws SQLException {
            Mock mockStmt = new Mock(Statement.class);
            String sql = “select * from test”;
            mockStmt.expect(
                once()).method(“execute”).with(eq(sql)).will(returnValue(false));
            Statement stmt = (Statement) mockStmt.proxy();
            assertFalse(stmt.execute(sql));
         }
}
```

在例子中，我们定义了一个 mock 对象，用来模仿 java.sql.Statement。并且断言，调用该对象的 execute 方法将返回一个 false 值。对于其中的“mockStmt.expect(…)…”调用，解释如下：

- expect，指定预期执行的次数，可以有 once()，atLeastOnce()，notCalled()三种取值；
- method，指定调用的方法名称，此处是“execute”；
- with，指定被调用方法所需的参数，若是无参方法，则不必写；
- will，指定被调用方法的返回值，若没有，则不必写；

其中，once(), eq(), returnValue()等都是继承自 MockObjectTestCase 的方法。

最新版本及新版本特性

jMock 的更新速度似乎并不是很快，目前它的最新版本是今年 4 月发布的 1.1.0RC1，该版本包含了一些过去积累下来的功能改进，以及 bug 修改，但就主体而言，与其上一个稳定版本 1.0.1 相比没有很大的差别。

jMock 是通过 CGLIB 的 DynamicProxy 来实现的 mock 功能的，除了可以 mock 接口之外，还可以 mock 具体类。但是，旧版本的 jMock 只能 mock 带有无参构造函数的具体类。这一问题，已经在新版本的 jMock 里得到了解决。新版本的 jMock 中还对约束条件做了较大的扩充，新增的约束包括：IsNothing, HasToString, StringStartsWith, StringEndsWith, IsSameFile, IsCompatibleType, HasProperty, HasPropertyValue, IsElementOf, IsCollectionContaining, IsMapContaining。此外，还增加了 doAll 和 returnIterator 两个新的 stub，以及一个新的 expectation——exactly(n)，以精确表达预期的调用执行次数。

主要人物及相关故事

在 jMock 的官方主页上有关于该项目开发团队的人员介绍。目前具有提交权限的开发人员有四个，他们分别是：Steve Freeman，Tim Mackinnon，Nat Pryce，Joe Walnes。值得一提的

是，这四位开发者几乎都是来自以敏捷实践见长的 ThoughtWorks。其中，Steve Freeman 和 Nat Pryce 共同参加了今年 4 月在英国牛津举行的 ACCU Conference，并做了主题演讲，内容是关于 jMock APIs 的演化，以及在 Java 和 C#领域，内嵌式 DSL（领域特定语言）的编写技术。

作为 jMock 项目主要开发者之一的 Steve Freeman，是敏捷软件开发方面的独立咨询师，他还是英国地区极限编程实践的早期推广者。他与其他三人共同撰写了一篇名为“Mock Roles, not Objects”的论文，探讨了有关 mock 测试技术方面的经验。这篇论文被收录于 2004 年的 OOSPA 论文集中，在 jMock 的官方主页可以找到该论文的电子版。除了 jMock 之外，四位开发者还开发了 jMock 的 c#实现版本——nMock。

除了开发人员以外，还有一些 jMock 项目的贡献者，他们为项目提供建议、补丁以及文档。不过，jMock 的在线文档资源并不是很丰富，好在 jMock 的代码简单而又精巧，因此有兴趣的读者不妨深入代码来一探究竟。

Selenium

什么是 Selenium?

JUnit 的主要用途是单元测试和组合测试，这大多是开发人员的工作。但对于一个真实的应用系统而言，这还是不够的。而在集成测试和验收测试时，尤其是目前广泛应用的 web 系统中，如何用自动化测试来替代繁重、易错而又无法回归的人工测试呢？Selenium 便是这一领域的后起之秀。

Selenium 是一个针对 Web 应用程序的自动验收测试工具，通过编写模仿用户操作的测试脚本，我们可以从终端用户的角度来对 web 应用程序进行黑盒测试。与其他测试工具相比，使用 Selenium 的最大好处是：Selenium 测试可以直接在浏览器内运行，就像真实用户操作一样。目前，Selenium 支持 Windows，Linux 和 Macintosh 上多种版本的 Internet Explorer，Mozilla 和 Firefox 浏览器，覆盖平台之多也是其他测试工具不能比拟的。Selenium 的跨平台支持，还有助于我们检查 web 应用对浏览器的兼容性支持。

Selenium 的使用有两种模式：test runner 和 driven。这两种模式在编写方式和复杂性方面有所不同。在 test runner 模式下，测试是完全在浏览器内运行的；而在 driven 模式下，测试则有一部分是在浏览器之外运行的。test runner 模式下的测试脚本是用 HTML 的 table 布局编写的；driven 模式下的测试脚本编写起来往往要更复杂些，因为它们是用编程语言编写的。但是如果使用 Python 或 Ruby 之类的高级动态编程语言，那么复杂性方面的差异就很小了。无论是哪种模式，都可以与持续集成工具很好的集成。

最新版本及新版本特性

目前，Selenium 已经演化成了几个并行的子项目，它们分别是：Selenium Core，Selenium IDE，

Selenium Remote Control 和 Selenium on Rails。

Selenium Core 是测试工具的核心，也是其他子项目的依托。它的最近一个发布版本是在今年 8 月份发布的 0.7.1，这个版本除了进一步扩大对不同浏览器的支持以外，主要是在如何能够更好的模仿用户操作方面做了改进，比如增加了像 dragAndDrop，get/setCursorPosition 这样的新命令。

Selenium IDE 是一个针对 Selenium 测试的集成开发环境。它是作为 Firefox 的扩展来实现的，允许你录制、编辑和调试 Selenium 测试。目前的最新版本是 0.8.0，该版本已经包含了 Selenium Core 的最新版本。

Selenium Remote Control 提供了一个 Selenium Server，它以 AJAX 方式来控制浏览器。你可以通过普通的 http 请求与 Selenium Server 交互，这使你能够用许多主流编程语言（比如：Java，.NET，Perl，Python，Ruby）来编写自动化的 web 应用 UI 测试。Selenium Remote Control 目前的最新版本是今年 6 月份发布的 0.8.1，这是一个维护版本。

Selenium on Rails 是一个专门用于为测试 Rails 应用程序提供便利的插件。它原来是由 Jonas Bengtsson 在其个人网站上发布的，今年 8 月份刚刚搬迁到 Selenium 的官方主页。

Selenium 的开发目前十分活跃，期待不久之后我们能够看到 Selenium 的后续版本。

主要人物及相关故事

Selenium 先前是 ThoughtWorks 员工的个人作品。2004 年，ThoughtWorks 的 Jason Huggins，Paul Gross 和 Jie Tina Wang 为当时的一个 web 报表系统共同开发了一个叫做 JavaScript Functional Test Runner 的自动测试工具，该工具能够对系统的功能进行自动化的验收测试。后来，Jason 向许多同事演示了他们的成果，不少人为该测试工具快速而直观的可视化反馈感到惊喜不已。同时，他们也感觉到，这一工具很有希望能够复用到许多其他的 web 应用系统中，从而成为一个通用的测试框架。随后，许多 ThoughtWorks 的咨询人员，还有一些外部群体，都开始加入了 Selenium 的开发队伍中。在短短的两年时间里，Selenium 已经有了长足的发展。Selenium 非常有希望成为 web 应用领域里，多语言多平台自动化测试工具的事实标准。有意思的是，今年 6 月在北京举办的第十届中国国际软件博览会上，Selenium 获得了“最具增值潜力软件产品（应用软件）”的称号。

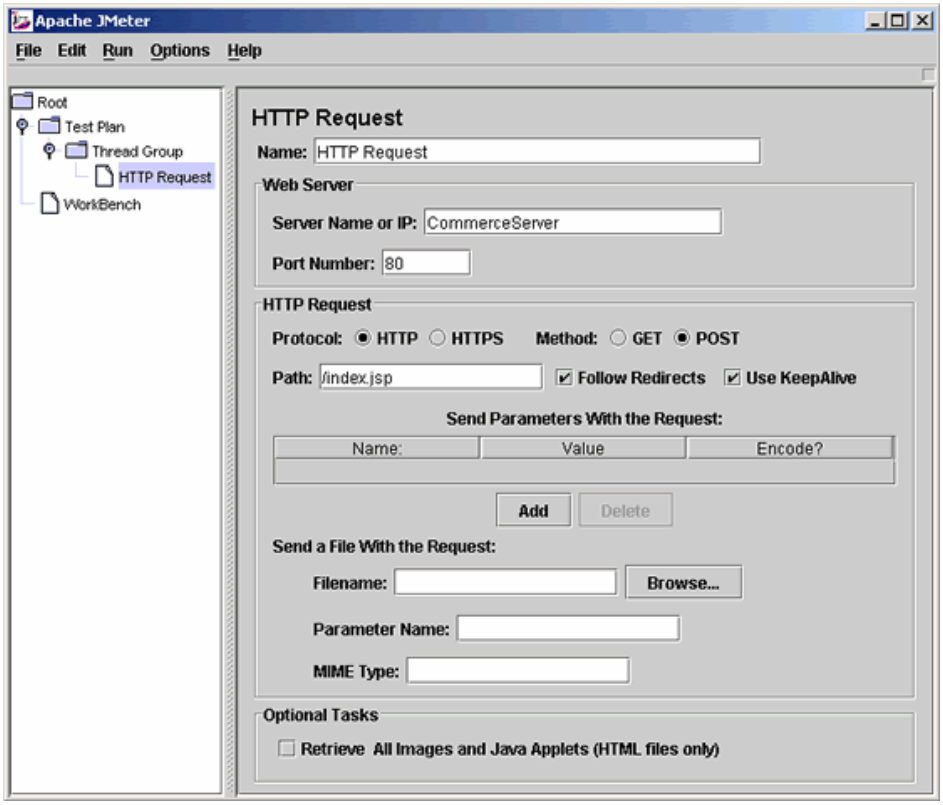
Apache JMeter

什么是 Apache JMeter？

JMeter 是 Apache 组织的开放源代码项目，用 Java 编写而成。它与 Selenium 的相似之处是，通常也用于 web 应用的自动化测试。但有所区别的是，它的主要用途是做性能测试和压力测试。

JMeter 可以用于测试静态或动态网络资源的性能，这些资源包括：普通文本、JSP、Servlet、Java 对象、Perl 脚本、数据库，以及 ftp 服务器，等等。利用多线程模式，JMeter 可以模拟在网络环境下服务器上的高负载和大并发量，从而测试服务器提供服务的受压能力，并分析这些服务在不同负载条件下的性能状况。

JMeter 是用 Swing 编写的，它提供了一个图形化的用户界面。利用它，我们能够以可视化的方式制定测试计划（Test Plan），包括：规定使用什么样的负载；测试什么内容；传入的参数，等等。同时，JMeter 还提供了多种图形化的测试结果显示方式，使我们能够简单的开始测试工作和分析测试结果。以下是 JMeter 的设置界面：



图：JMeter 的设置界面

从图中的诸项设置我们可以看到，利用 JMeter 所提供的功能，我们可以很方便的模仿来自客户端的 HTTP/HTTPS 请求。另外，通过设置 HTTP 代理，JMeter 会自动把手工操作录制成脚本，并实现脚本的回放功能。

最新版本及新版本特性

JMeter 自 1998 年底发布首个 1.0 版本以来，已经历了 8 年的时间，基本上每年都在不断的完善和补充。现在的最新版本是大约在去年下半年发布的 2.1.2。不过，CVS 中显示的最新版本是今年 6 月份上传的 2.2。从官方主页上的版本更新历史来看，2.1.2 中修正了不少以往的 bug，同时也有一些新的功能和原有功能的增强，比如：新的 XPath Extractor Post-Processor，增强的 JUnit Sampler 和 HTTP Sampler，等等。详细内容，请查看 JMeter 的版本更新历史：http://jakarta.apache.org/jmeter/changes.html

主要人物及相关故事

JMeter 项目的开发团队目前有十多个人，此外还有许多项目的贡献者，你可以在 JMeter 的 Wiki 上找到所有这些人的详细名单。平时，JMeter 的开发人员常常会通过 IRC 进行即时的在线聊天，一起讨论有关 JMeter 开发的问题。普通人也可以加入到聊天活动中，你可以在这里询问有关 JMeter 的一切问题。在 JMeter 的官方主页上有 IRC 的服务器地址和 Room 号。当然，你还可以通过 mailing list 等其他方式与 JMeter 团队沟通。

商业同类产品

在测试领域，也存在一些商业厂商提供的优秀测试工具。这些厂商的产品覆盖了完整的软件生命周期，功能齐全，当然价格也不菲。其中具有代表性的公司有：Rational，Mercury，Compuware，Segue，Empirix。这里向读者列举一二，也许未必全面。

Rational 测试工具系列

Rational 一直以来就在软件全生命周期方面见长。其系列工具贯穿了整个软件开发的生命周期，自然也覆盖了测试阶段。Rational 的自动化测试技术依托于 RUP 过程，有一套完整的软件测试流程。从测试管理，到功能测试和性能测试，再到可靠性测试和单元测试，直至实时系统的自动化测试，Rational 为每个环节都提供了相应的工具，可谓包罗万象。这里做一个简单的列举：

- TestManager，测试管理工具，包括测试计划、测试用例、测试执行的管理，性能测试的运行场景也在这里部署；
- Purfiy，用于白盒测试，测试 C、C＋＋、Java 程序的内存泄漏问题；
- Robot，用于功能回归测试和性能测试，测试脚本可自动生成，也可手工修改；
- Pure Coverage，代码覆盖测试工具，用于检查测试用例是否完全覆盖了被测试程序，并报告覆盖率；
- Rational Functional Tester，对 Java、Web 和基于 VS.NET WinForm 的应用程序进行高级自动化功能测试；
- Rational Manual Tester，使用新的测试设计技术来改进人工测试设计和执行工作；
- Rational Performance Tester，检查多用户负载情况下可接受的应用程序响应时间和可伸缩性；
- Rational Test RealTime，支持嵌入式和实时的跨平台软件的组件测试和运行时分析；

Mercury 测试工具系列

刚刚于今年 6 月被 HP 收购的 Mercury Interactive 是一家专业的软件质量保证工具开发公司，长久以来一直占据着软件测试工具领域的主导地位，在测试界的知名度颇高，它还提供专门的测试认证。Mercury TestSuite 系列的黑盒测试工具占据了同类商业产品的主流，WinRunner 和 LoadRunner 是大家所熟悉的两款自动测试工具。

WinRunner 是一款企业级的功能测试工具，用于检测应用程序是否能够正常运行，并达到预期的功能。通过自动录制、检测和回放用户操作，有效的帮助测试人员对跨平台的、复杂的企业级应用进行测试。从而提高测试人员的工作效率和工作质量，确保系统无故障发布并能长期运行。

LoadRunner 是一款负载测试工具。它能够模拟上千万用户的实时并发负载，并进行实时的性能检测，从而确认和查找问题所在。LoadRunner 适合各种体系架构，它能预测系统行为，并优化系统性能。它支持 Web、IMAP、SMTP、Oracle、EJB、Media Stream、无线等多种协议。

此外，Mercury 还提供了一款配套的测试管理工具 TestDirector。通过它的使用，可以规范测试管理流程，建立起针对具体项目的测试方案和测试计划。TestDirector 通过接口与 LoadRunner、WinRunner 交互，从而能够统一的管理各种测试用例，自动测试脚本，运行场景与测试结果，并且可以针对发生问题的部分进行错误跟踪。

Compuware 测试工具系列

Compuware 的白盒测试工具集——Numega 系列比较有代表性。其功能涵盖了指针和内存泄漏错误检查，代码运行效率检查，组件性能分析，自动错误处理和恢复，图形化的线程和事件分析，函数调用次数、所占比率统计以及稳定性跟踪，自动源代码分析，等等。

Compuware 的黑盒测试工具集 QACenter 汇集了完整的跨企业自动测试工具，专为提高软件质量而设计。QACenter 可以在整个开发生命周期中，跨越多种平台，自动执行测试任务。其中，包含于性能版中的 QALoad 是客户/服务器系统、ERP 和电子商务应用的自动化负载测试工具。它通过可重复的、真实的测试，能够彻底地度量应用的性能和可伸缩性。

此外，Compuware 提供的工具产品也覆盖了软件的全生命周期，Compuware 还是著名的 MDA 工具 OptimalJ 的开发厂商。

Segue 测试工具系列

今年上半年刚刚被 Borland 公司收购的 Segue 软件是一家提供全面 SQO（Software Quality Optimization）解决方案的公司，其产品从功能上分为两大部分。

质量管理方面的产品，均冠以 SilkCentral，包括：

- SilkCentral Test Manager，用于管理完整的测试过程，包括测试需求定义、测试计划制定、测试执行、回归测试等，贯穿于软件开发与测试的始终；
- SilkCentral Issue Manager，用于对测试过程中的缺陷和 bug 进行追踪和管理；
- SilkCentral Performance Manager，用于应用系统发布上线之后，对应用进行性能监控和管理，支持闭环测试，能有效集成 Segue 的其他测试软件产品；

测试工具方面的产品，包括：

- SilkTest & SilkTest International，用于对应用进行自动化的功能测试，分别针对单语言版

本和多语言版本进行测试，其特有的 4Test 脚本语言极为灵活，受到开发测试人员的广泛好评，脚本重用性很高；

- SilkPerformer，用于对应用系统进行复杂测试，不仅针对整个应用系统，而且对组件级压力测试提供特别的支持，使除了技术人员以外的业务人员也能够方便的进行负载测试，测试结果可视化程度很高；

Borland 目前已经开始把旗下的 ALM（Application Lifecycle Management）产品与 Segue 的 Silk 及 SilkCentral 产品线整合，成为 Borland 生命周期质量管理方案的基础。该方案将于今年稍后时间推出，相信此方案有助于客户解决质量的根本问题，能从项目开始就找出问题所在，而不是在开发后期才亡羊补牢，因为那时纠正的成本已经大大提高了。

推荐书目

JUnit In Action 中文版

本书介绍了使用 JUnit 进行测试的原则、技巧与实践。深入阐述了如何编写自动测试，把一段代码隔离开来测试有什么好处，如何判断何时需要进行集成测试，并对如何测试完整的 J2EE 应用进行了极有价值的讨论。本书富含开发实践中的真实案例，以专家的手笔讨论了实践中的测试技术，内容包括：用 mock 对象进行隔离测试；用 Cactus 进行容器内测试；用 Ant 和 Maven 进行自动构建；在 Eclipse 内进行测试；对 Java 应用程序、Filter、Servlet、EJB、JSP、数据库、Taglib 等进行单元测试。
作者：Vincent Massol，Ted Husted
原出版社：Manning
译者：鲍志云
出版社：电子工业出版社
ISBN：7-121-00483-6
出版日期：2004-10-1

Test-Driven Development: By Example（影印版）

本书的作者是极限编程的创立者 Kent Beck。该书获得第 13 届 General 效能大奖，在 Amazon 网站上持续热卖，是 Addison-Wesley 出版公司著名的大师签名系列图书之一。测试驱动开发（TDD）是极限编程的重要特点，它以不断的测试推动代码的开发，既简化了代码，又保证了软件质量。但是如何正确地进行测试，以及如何对代码中难以测试的地方进行测试，这些问题一直在困扰着开发团队。本书从头至尾跟踪介绍了两个 TDD 项目，向开发人员介绍了容易上手又能大大提高工作质量的技术。
作者：Kent Beck
原出版社：Pearson
出版社：中国电力出版社
ISBN：7-5083-1401-8
出版日期：2003-8-1

Test-Driven Development: A Practical Guide（影印版）

本书荣获第 14 届 Technical Jolt 大奖。这本测试驱动开发的指南，着重于真实项目、实现问题和实际的代码。作者通过使用 Java 和 JUnit 测试框架，从头开始创建了一个项目，向读者展示了 TDD 的精妙之处。让读者深刻理解了什么是“测试先行”，它的工作机理，其中的困难以及如何将 TDD 付诸实践。书中还讨论了针对 C++、C#/.NET、VB6、Python、Ruby 和 Smalltalk 的 TDD 框架，并介绍了以前未发表过的关于 GUI 软件的测试先行技术。
作者：Dave Astels
原出版社：Pearson
出版社：中国电力出版社
ISBN：7-5083-2193-6
出版日期：2004-4-1

单元测试之道 Java 版——使用 JUnit

本书的英文名称是《Pragmatic Unit Testing：In Java with JUnit》。该书获得第 14 届震撼大奖（Jolt Award）、生产力大奖（Productivity Award）。书中阐述了如何使用 JUnit 和 Java 语言进行单元测试，其内容广泛适用于其他语言和框架程序库。本书主要内容包括：如何更高效地撰写 bug 更少的代码；如何发现 bug 的藏身之处以及如何清除 bug；如何测试代码片断而不用牵连整个项目；如何利用 JUnit 简化测试代码；如何在团队中高效地进行测试，等等。此外，该书还有一个 C#版本——《单元测试之道 C#版——使用 NUnit》
作者：Andrew Hunt，David Thomas
原出版社：The Pragmatic Programmers
译者：陈伟柱，陶文
出版社：电子工业出版社
ISBN：7-121-00665-0
出版日期：2005-1-1

软件测试的艺术

一本“活”了 25 年的书，一本全面阐释软件测试理论与实践的书。作者提出了“测试是为发现错误而执行程序的过程”、“测试的目标是建立‘软件做了其应该做的，未做其不应该做的’信心”等观点。本书以一次自评价测试开篇，从软件测试的心理学和经济学人手，探讨了代码检查、走查与评审、测试用例的设计、模块（单元）测试、系统测试、调试等主题，以及极限测试、因特网应用系统测试等高级主题，全面展现了作者的软件测试思想。本书是软件测试领域的佳作，适合的读者群是软件测试从业人员。
作者：Glenford J.Myers 等
原出版社：John Wiley & Sons
译者：王峰，陈杰
出版社：机械工业出版社
ISBN：7-111-17319-8
出版日期：2006-1-1

有效软件测试——提高测试水平的 50 条建议（影印版）

和前一本书类似，这也是一本全面阐述软件测试的专业书籍。本书以介绍如何将测试运用到软件开发生命周期的所有阶段中为重点——从需求定义到设计直至最终代码；书中探讨了 50 个至关重要的最佳实践、缺陷及解法。这些具体项目是从作者丰富的实践经验中收集而来的，能够使质量保证专业人员和测试管理人员即刻提高其理解能力和技巧，避免重大错误，并实现高水准的测试程序。

作者：Elfriede Dustin

原出版社：Addison Wesley/Pearson

出版社：中国电力出版社

ISBN：7-5083-1054-3

出版日期：2004-1-1

自动化软件测试——入门、管理与实现（影印版）

这是一本由浅入深的学习自动化测试所涉及的工具、技术和方法的技术图书。虽然书名是“自动化软件测试”，但书中介绍的更多的并非自动化测试的具体实现，而是侧重于测试过程和测试管理方面的内容，这也是因为自动化软件测试工作的开展必须依赖于一个完善的测试过程。

作者：Elfriede Dustin，Jeff Rashka，John Paul

原出版社：Addison Wesley/Pearson

出版社：清华大学出版社

ISBN：7-89494-044-5

出版日期：2003-3-1

相关资源网站

http://www.junit.org/

JUnit 的官方主页。在这里你可以找到 JUnit 的最新下载版本，在线的 JavaDoc 文档，包括 FAQ 在内的各种文档资源，还有许多资源链接，通过这些链接，你可以找到大量与 JUnit 相关的网站，比如：各种基于 JUnit 的扩展实现，IDE 插件，以及 JUnit 的相关书籍等等。

http://www.jmock.org/

jMock 的官方主页。在这里你可以找到 jMock 的最新下载版本，了解有关 jMock 的最新消息，还有相关的文档资源，告诉你如何用 jMock 来编写测试代码，如何掌握约束，以及与另两个同类型的 mock 测试工具——DynaMock 和 EasyMock——的对比。

http://www.openqa.org

这是一个专门致力于提供开源 QA 工具的网站，创建于 2005 年 12 月。虽然时间不长，但很活跃。时下流行的 web 测试工具 Selenium，便是该网站下的一个子项目。除 Selenium 外，OpenQA 还包含了另外几个开源的 QA 项目，它们是：Floyd，pyWinAuto，Watir，WET。该网站受一些业内公司赞助和支持，它们包括：Autoriginate，Contegix，Jive Software，Caucho，ThoughtWorks 和 Atlassian。

http://fitnesse.org

FitNesse 的官方主页。FitNesse 是一个利用 Wiki 语法来编写测试脚本，自动执行测试，并以 Wiki 方式展现测试结果的自动验收测试框架。其工作方式在某些地方和 Selenium 有类似之处。不过，FitNesse 的实现思路却是来源于另一个经典的集成测试框架——Fit（Framework for Integrated Test），而 Fit 的作者便是 Wiki 的发明人 Ward Cunningham。

http://jakarta.apache.org/jmeter/

JMeter 的官方主页。在这里你可以找到 JMeter 的最新下载版本，版本变更历史，以及相关的文档资源。和其他的 Apache 子项目一样，JMeter 也有自己的 Wiki，你可以在那里找到 JMeter 用户手册，FAQ，以及其他有用的资源。

http://www.xprogramming.com/

作为敏捷方法中的代表者，极限编程极力倡导测试先行和与此相关的一系列轻量级软件开发实践。如果你是使用 JUnit 编写单元测试的 Java 程序员，那么不妨了解一下这些敏捷实践，这对你更好的编写测试，实践测试驱动开发大有助益。在这里你能找到有关敏捷方法的所有资源，其中还包括一份详尽而完整的敏捷实践工具清单。

http://www.testage.net/

国内较大的测试方面的专业网站，是以测试一线人员为主的自发的民间组织。网站提供了各种与测试相关的技术文章以及电子杂志（《测试员》），开辟了交流测试技术的论坛，还组织过多次线下的专业沙龙和交流会，具有较大影响力。

http://www.51testing.com/

另一家国内软件测试专业网站，内容涉及软件测试和质量保证等相关领域。同样拥有许多技术资源，还有电子杂志（《无忧测试》），技术交流论坛，以及线下的沙龙活动。

http://opensourcetesting.org/

这是有关开源测试工具最全面的收集网站，它用 Robot 自动收集开源项目站点的信息，最大的特点是将工具按照类别进行了分类，是一个查找工具的好去处。但该网站未提供任何评估信息，仅仅是从各项目网站上摘抄了工具说明。