作为一个测试工具，Junit是一个好的开始，它本身其实不难，但难在如何落实测试。大多数的时间中，开发人员习惯于先编写好程序，然后运行并观看结果来调试，能够编写测试程序已属难能可贵，真正能做到测试驱动（Test-Driven）的就更加罕见了。

# 单元测试(Unit Test)

一个单元(Unit)是指一个可独立进行的工作，独立进行指的是这个工作不受前一次或接下来的工作的结果影响。简单地说，就是不与程序运行时的上下文(Context)发生关系。

如果是在Java程序中，具体来说一个单元可以是指一个方法（Method）.这个方法不依赖于前一次运行的结果，也不牵涉后一次的运行结果。

举例来说，下面这个程序的gcd()方法可视为一个单元：

**public** **class** MathTool {

**public** **static** **int** gcd(**int** num1 , **int** num2){

**int** r = 0 ;

**while**(num2!=0){

r = num1 % num2 ;

num1 = num2 ;

num2 = r ;

}

**return** num1;

}

}

下面的gcd()方法不可视为一个单元，要完成gcd的计算，必须调用setNum1()、setNum2()与gcd()三个方法。

**public** **class** MathFoo {

**private** **static** **int** *num1*;

**private** **static** **int** *num2*;

**public** **static** **void** setNum1(**int** n){

*num1* = n;

}

**public** **static** **void** setNum2(**int** n){

*num2* =n;

}

**public** **static** **int** gcd(){

**int** r = 0 ;

**while**(*num2*!=0){

r = *num1* % *num2* ;

*num1* = *num2* ;

*num2* = r ;

}

**return** *num1*;

}

}

然而要完全使用一个方法来完成一个单元操作在实现上是有困难的，所以单元也可广泛解释为整个方法的集合。这数个方法组合为一个单元操作，目的是完成一个任务。

不过设计时仍优先考虑将一个公开的方法设计为单元，辅助的方法则使用设置为私有，尽量不用数个公开的方法来完成一件工作，以保持接口接口简洁与单元边界清晰。将工作以一个单元进行设计，这使得单元可以重用，并且也使得单元可以进行测试，进而增加类的可重用性。

单元测试指的是对每一个工作单元进行测试，了解其运行结果是否符合我们的要求。例如当编写完MathTool类之后，也许会这么写一个小小的测试程序：

**public** **class** MathToolTest {

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

**if** (MathTool.*gcd*(10, 5) == 5) {

System.*out*.println("GCD Test OK !");

} **else** {

System.*out*.println("GCD Test fail");

}

}

}

在文字模式下使用文字信息显示测试结果，这个动作是开发人员经常作的事情，然而你必须一行一行看着测试程序的输出结果，以了解测试是否成功。另一方面，测试程序本身也是一个程序，在更复杂的测试中，也许会遇到测试程序本身出错，而导致无法验证结果的情况。

JUnit是一个测试框架，通过它所提供的工具，可以减少编写错误的测试程序的机会。另一方面可以有更好的方法来检验测试结果，而不是看着一长串输出的文字来检验测试是否成功。Junit测试框架让测试的进行更有效率且更具可靠性。

# JUnit设置

Junit为单元测试的支持框架，用来编写与执行重复性的测试。它包括以下特性：

* 对预期结果作判断
* 提供测试装备的生成与销毁
* 易于组织与执行测试
* 图形与文字接口的测试器

要设置Junit，可先到Junit官方网站<http://junit.org/>下载JUnit的zip文件，解压后其中会含有junit.jar文件，将这个文件复制到所要的文件夹中，然后设置Classpath指向junit.jar.例如：

set classpath = %classpath%;YOUR\_JUNIT\_DIR\junit.jar

# 第一个JUnit测试

要对程序进行测试，首先要设计测试案例(Test Case) 。 一个测试案例是对程序给予假定条件，然后运行程序并看看在给定的条件下，程序的运行结果是否符合要求。

在JUnit下，可以继承TestCase来编写测试案例，并定义测试方法，每一个测试方法是以testXXX()来命名。一个例子如下所示：

**package** cn.ifinace;

**import** junit.framework.TestCase;

**public** **class** MathToolUnitTest **extends** TestCase {

**public** **void** testGcd(){

*assertEquals*(5, MathTool.*gcd*(10, 5));

}

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

junit.textui.TestRunner.*run*(MathToolUnitTest.**class**);

}

}

assertEquals()方法用来判定您的预期值与单元方法实际的返回结果是否相同，如果预期值与返回的结果不同则丢出异常，TestRunner会捕捉异常，并提取其中的相关信息以报告测试结果。这里使用的是文字模式的TestRunner。

运行测试案例，会得到以下的结果：

.

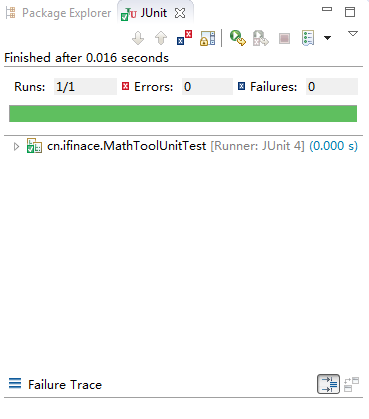
Time: 0.003

OK (1 test)

不一定要在main()中指定 TestRunner ，而可以直接启动一个TestRunner ，并指定测试案例类(继承TestCase的类).例如启动一个Swing窗口的测试结果画面：

java junit.swingui.TestRunner cn.ifinace.MathToolUnitTest

执行结果如下图：



在swing窗口的测试结果显示中，如果中间的横棒是显示绿色，表示所有的测试都已经成功，如果中间的横棒显示红色，表示测试失败。JUnit的名言是Keep the bar green to keep the code clean ,意思是保持绿色横棒以保证测试成功。

也可以指定文字模式的测试结果。例如：

java junit.textui.TestRunner cn.ifinace.MathToolUnitTest

# 自动化构建与测试

Ant可以进行自动化构建，而JUnit可以进行自动化测试，Ant可以与JUnit结合，使得自动化的构建与测试变得可行。

如果要让Ant能支持JUnit，建议直接将JUnit的junit.jar放置在Ant的lib目录，并记得改变Classpath中原先有关的junit.jar的设置。例如将Classpath重新指向%ANT\_HOME%\lib\junit.jar(假设已经设置了ant\_home环境变量)。

Ant使用<junit>标签来设置JUnit测试，下面是一个简单的例子：

<?xml version=*"1.0"*?>

<project name=*"antoBuildTest"* default=*"test"*>

<target name=*"setProperties"*>

<property name=*"src.dir"* value=*"src"* />

<property name=*"classes.dir"* value=*"classes"* />

</target>

<target name=*"prepareDir"* dependds=*"setProperties"*>

<delete dir=*"${classes.dir}"* />

<mkdir dir=*"${classes.dir}"* />

</target>

<target name=*"compile"* depends=*"prepareDir"*>

<javac srcdir=*"${src.dir}"* destdir=*"${classes.dir}"* />

</target>

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

<junit printsummary=*"yes"*>

<test name=*"cn.ifinace.MathToolUnitTest"* />

<classpath>

<pathelement location=*"${classes.dir}"* />

</classpath>

</junit>

</target>

</project>

printsummary属性会将测试的结果简单地显示出来，<test>的name属性是设置所要进行测试的测试案例类，Ant构建与调用Junit进行测试的信息如下：

c:\workspace\B\ant

Buildfile: build.xml

setProperties:

prepareDir:

[mkdir] Created dir : C:\workspace\B\classes

compile:

[javac] Compile 4 source files to C:\workspace\B\classes

test:

[junit] Running cn.ifinace.MathToolUnitTest

[junit] Tests run : 1 ,Failures : 0 ,Errors: 0 , Time elapsed: 0 sec

Build SUCCESSFUL

Total time :　1 second

# 自动生成测试报告

接上一个主题，可以将JUnit的测试过程在Ant构建过程中显示出来，只要加入<formatter>标签设置即可：

<?xml version=*"1.0"*?>

<project name=*"antoBuildTest"* default=*"test"*>

...

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

<junit printsummary=*"yes"*>

<formatter type=*"plain"* usefile=*"false"*/>

<test name=*"cn.ifinace.MathToolUnitTest"* />

<classpath>

<pathelement location=*"${classes.dir}"* />

</classpath>

</junit>

</target>

</project>

接下来Ant构建与调用JUnit进行测试的信息，就不展示了，可以自行试验。

当usefile属性设置为true时，会自动将产生的结果保存在文件中，默认是Test-\*.txt。其中\*是测试案例类名称。

<formatter>标签还可以设置将测试的结果，以XML文件保存下来。

<formatter type=*"xml"* /> ， 测试结果将保存至report目录中，文件名称为Test-\*.xml , \* 是测试案例类名称。

也可以将测试结果所产生的XML文件转换为HTML文件，使用Ant可以直接完成这个工作。

<junitreport>标签使用XSLT将XML文件转换为HTML文件。下面的例子将前面的说明做个总结，以完整呈现编写的实例：

<?xml version=*"1.0"*?>

<project name=*"antoBuildTest"* default=*"test"*>

<target name=*"setProperties"*>

<property name=*"src.dir"* value=*"src"* />

<property name=*"classes.dir"* value=*"classes"* />

<property name=*"report.dir"* value=*"report"* />

</target>

<target name=*"prepareDir"* dependds=*"setProperties"*>

<delete dir=*"${report.dir}"* />

<delete dir=*"${classes.dir}"* />

<mkdir dir=*"${report.dir}"* />

<mkdir dir=*"${classes.dir}"* />

</target>

<target name=*"compile"* depends=*"prepareDir"*>

<javac srcdir=*"${src.dir}"* destdir=*"${classes.dir}"* />

</target>

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

<junit printsummary=*"yes"*>

<formatter type=*"xml"* />

<test name=*"cn.ifinace.MathToolUnitTest"* />

<classpath>

<pathelement location=*"${classes.dir}"* />

</classpath>

</junit>

</target>

<target name=*"report"* depends=*"test"*>

<junitreport todir=*"${report.dir}"*>

<fileset dir=*"${report.dir}"*>

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

</fileset>

<report format=*"frames"* todir=*"${report.dir}/html"* />

</junitreport>

</target>

</project>

<inculde>设置搜寻Test-\*.xml文件，将之转换为HTML文件，而最后的结果被设置保存至report/html/目录下，在format属性中设置了HTML文件具有边框(Frame),如果不设置这个属性，则HTML报告文件就不具有边框。