

1. 单元测试规范

单元测试使用 Junit4 进行测试，Eclipse 内置了 Junit4 的支持。

1) 测试类命名

在项目的 DAO 与 service 层的实现类都必须编写测试用例，action 由于 struts2.0 支持测试，所以也应该测试：

1. 测试用例**命名规则为：类名 + Test**。（junit4 支持非这样的命名方式，但是为了统一管理，还是采用这样的方式命名）
2. 建立一个和 src 平行的 test 包，所有测试用例都放在相应的包内，便于统一管理，合成测试套件。
3. 同一个包的测试用例，合成一个测试套件。
4. 整个工程的测试套件，合成一个统一的测试套件。

2) 测试用例的编写

1. 测试方式都是 test 开头的方法(testXXXX)，JUnit 按照在测试用例中的顺序执行。测试方法可以和被测试的方法一一对应，**测试方法也可以包含多个被测试的方法**。
2. 测试方法中，使用断言(assertXXX 和 fail，详细资料请查阅 JUnit 文档)来进行测试结果判断，也可以辅以文字打印说明，如果测试程序抛出异常，则显示为错误，如果断言失败，则显示故障。
3. 测试用例必须覆盖被测试类、方法的所有功能，包括正常情况、异常情况和发生错误的情况都必须覆盖，才能保证测试的完整性。

3) 测试数据的准备

为了防止数据库的更改对测试结果的影响，测试数据由统一的 sql 脚本来创建，测试前执行一下脚本创建数据。也可以使用数据回滚的方式进行与数据相关的测试。

4) 实体层的测试

由于本次项目的 Entity model 由之前的贫血模型改为到充血模型，所以对于 Entity 的测试不能够再忽略，对于 Entity 自身的除 set/get 以为的方法都应该进行单元测试。

5) DAO 层的测试

每一个 DAO 类都必须编写测试用例，对 DAO 的每一个公开方法进行测试，测试用例必须使用有代表性的测试代码，覆盖的所有可能的输入和输出情况，包括创建，更改，删除对象以及输入错误数据等测试。

6) Service 层的测试

Service 层的测试必须测试 Service 对象是否满足功能要求，事务完整性等功能。

7) Struts 层和浏览器层的测试

待定...

2. Eclipse 中 JUnit 的用法

这里不详细介绍 JUnit 的用法，详细的用法自己找文档进行学习。

下面举一个简单的例子说明 Eclipse 中 JUnit 的用法：

我们假设我们要写一个整数除法的类，并且给他写测试用例：

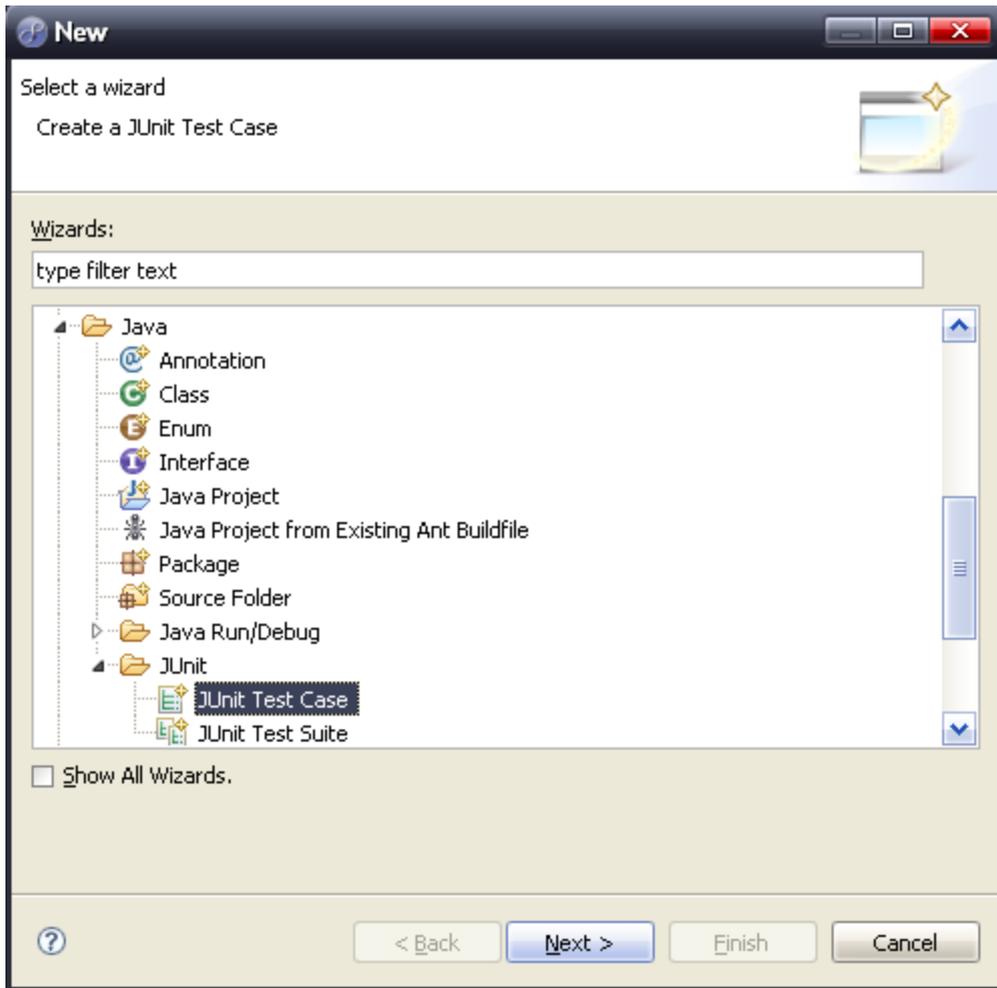
1) 建立 Math 类

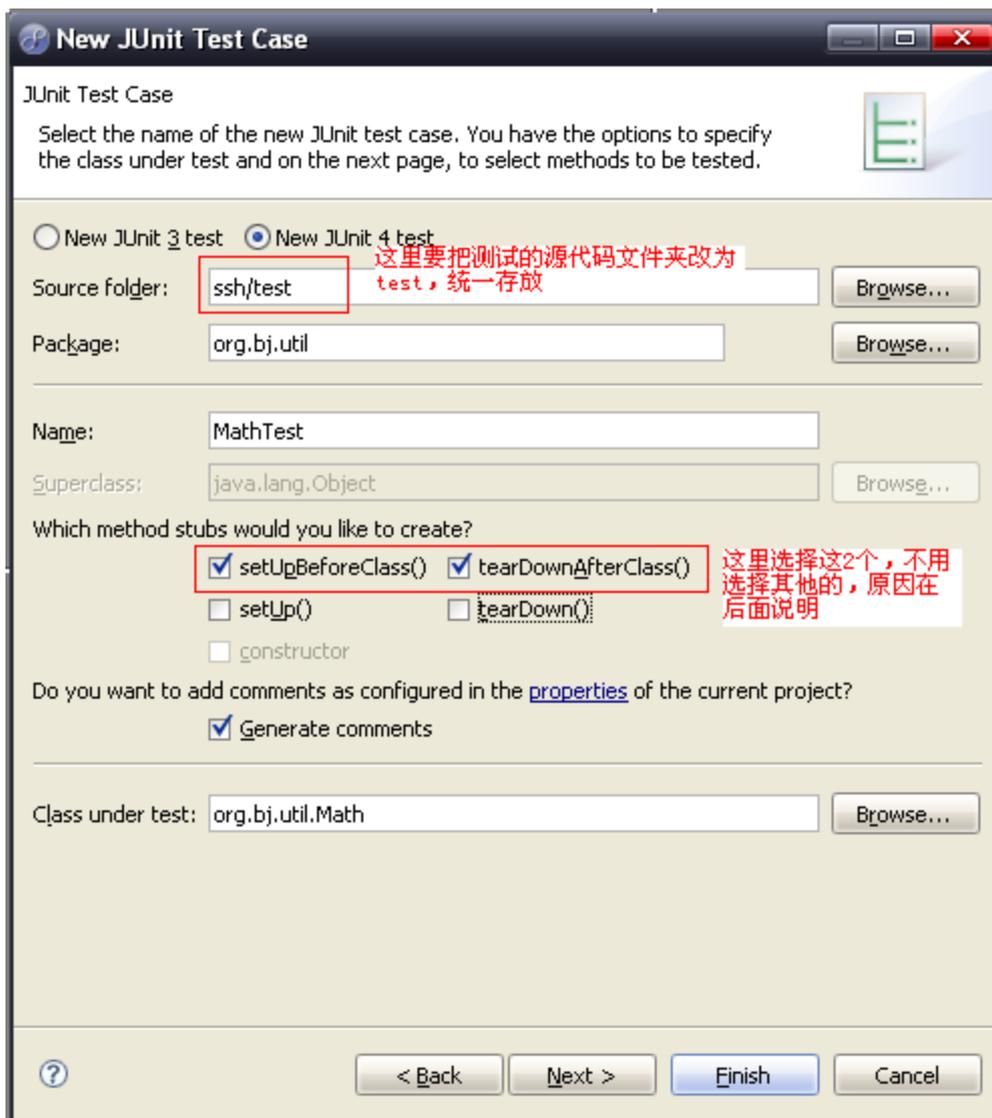
```
/**
 * @author bulargy.j.bai
 * @创建时间: Mar 10, 2008
 * @描述:
 */
public class Math {
    public static int divide(int x,int y) {
        return x/y;
    }

    public static int multiple(int x,int y) {
        return x*y;
    }
}
```

2) 建立测试用例

选中需要建立测试用例的包，选择 new->other。





这里 main 方法不需要，我们可以直接用 IDE 进行测试。

setUp()方法在测试方法前调用，一般用来做测试准备工作。

tearDown()方法在测试方法后调用，一般作测试的清理工作。

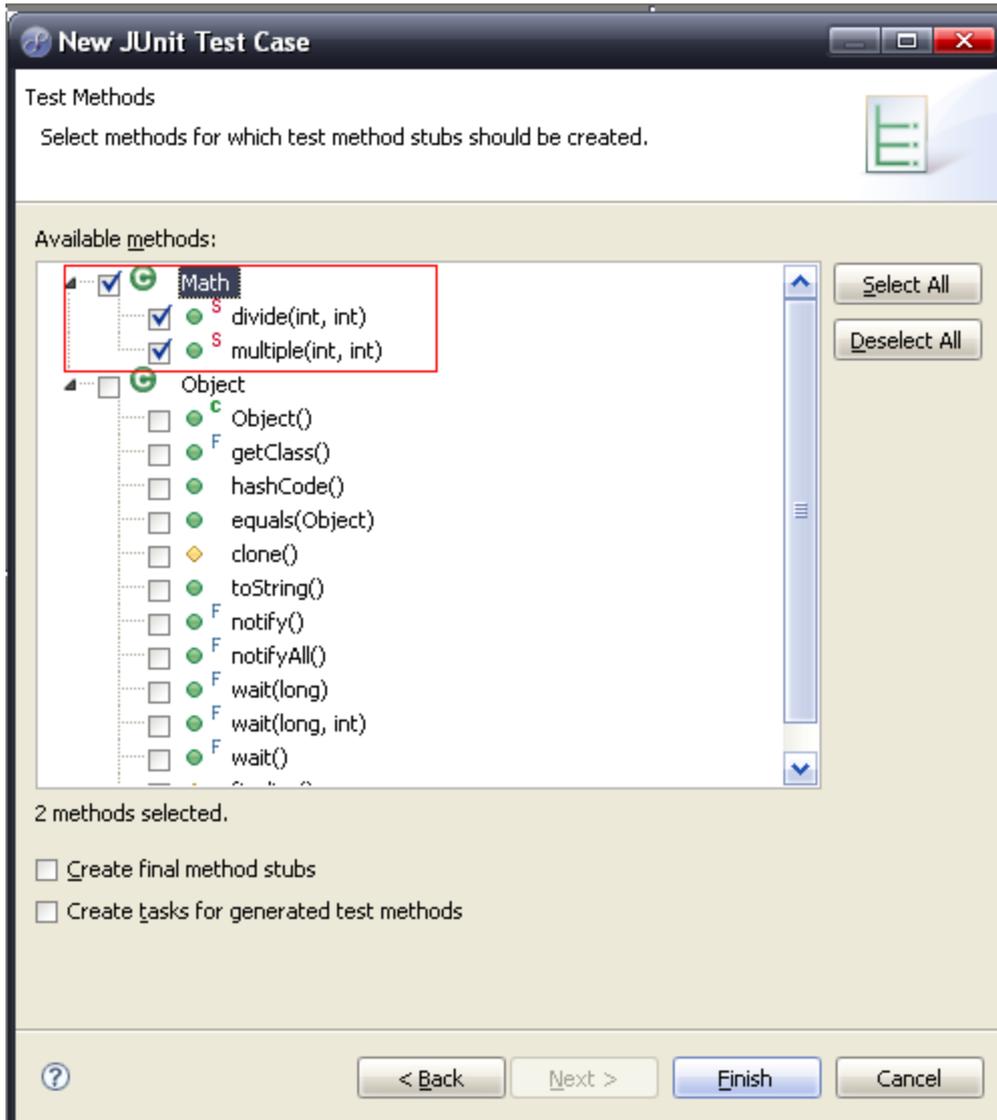
setUpBeforeClass()方法在整个类初始化之后调用，一般用来做测试准备工作。

tearDownAfterClass()方法在整个类结束之前调用，一般作测试的清理工作。

constructor()为是否包含构造方法。

选择下一步：

选择需要测试的方法，完成。



系统生成以下代码：

```
/**
 * @author bulargy.j.bai
 * @创建时间: Mar 11, 2008
 * @描述:
 */
public class MathTest {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Test
    public void testDivide() {
        fail("Not yet implemented");
    }

    @Test
    public void testMultiple() {
        fail("Not yet implemented");
    }
}
```

说明:

`@BeforeClass` 标签注释的方法用于在整个类测试过程的初始化后调用一次，`@AfterClass` 标签注释的方法则是整个测试类结束之前调用一次。这 2 个标间的搭配可以避免使用 `@Before`、`@After` 标签组合在每个测试方法前后都调用的弊端，减少系统开销，提高系统测试速度。（不过对环境独立性的测试还是应当使用 `@Before`、`@After` 来完成）

`@Test` 标签用来标注待测试的方法，按照类中声明的顺序执行。

我们在 `testDivide` 方法加入测试代码，分别测试三种情况：

- 完全正确也没有可能出错的数据，如：9 除 3 结果必须等于 3
- 可能有问题的边缘数据，如：10 除 3 结果也必须等于 3
- 错误的的数据，如：10 除 0 必须抛出异常

忽略 `testMultiple` 方法

代码如下：

```

@Test(expected=ArithmeticException.class)
public void testDivide() {
    assertEquals(3,Math.divide(9,3));
    assertEquals(3,Math.divide(10,3));
    Math.divide(10,0); //除数不能为0，会抛出异常
}

@Ignore("忽略乘法测试")
@Test
public void testMultiple() {
}

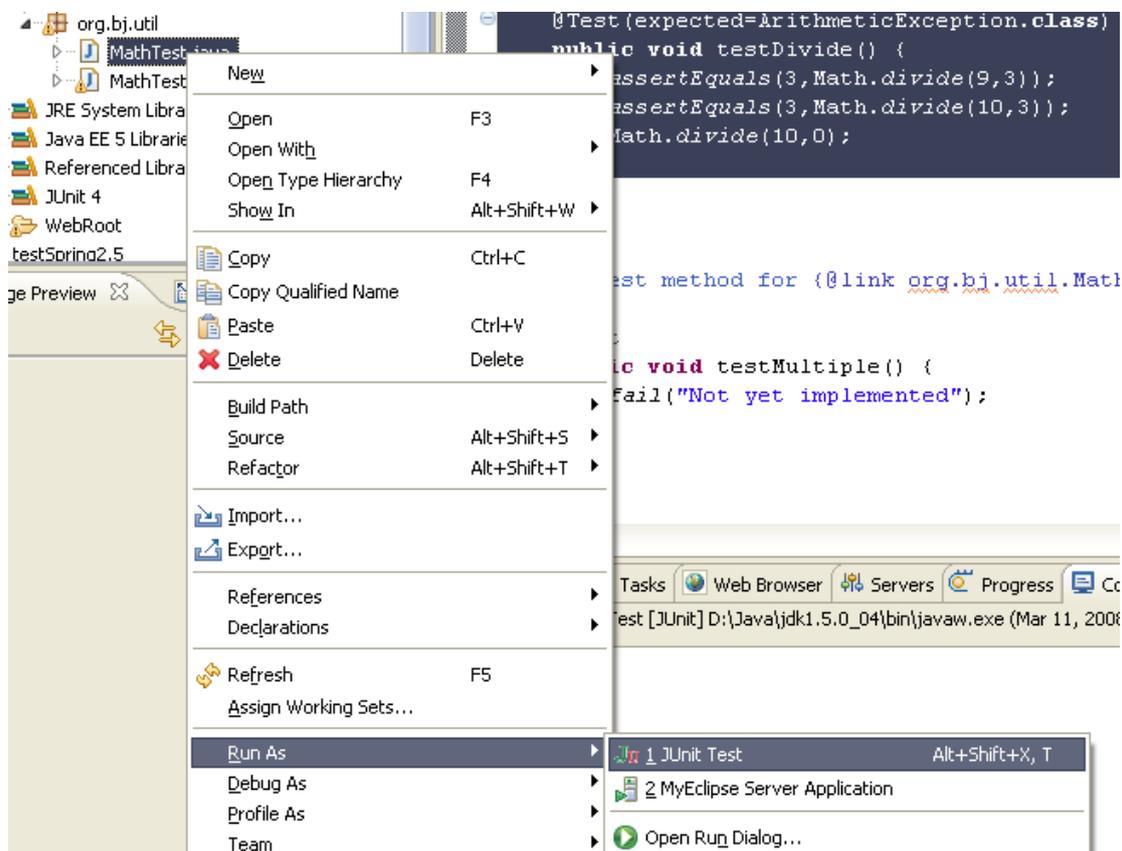
```

说明:

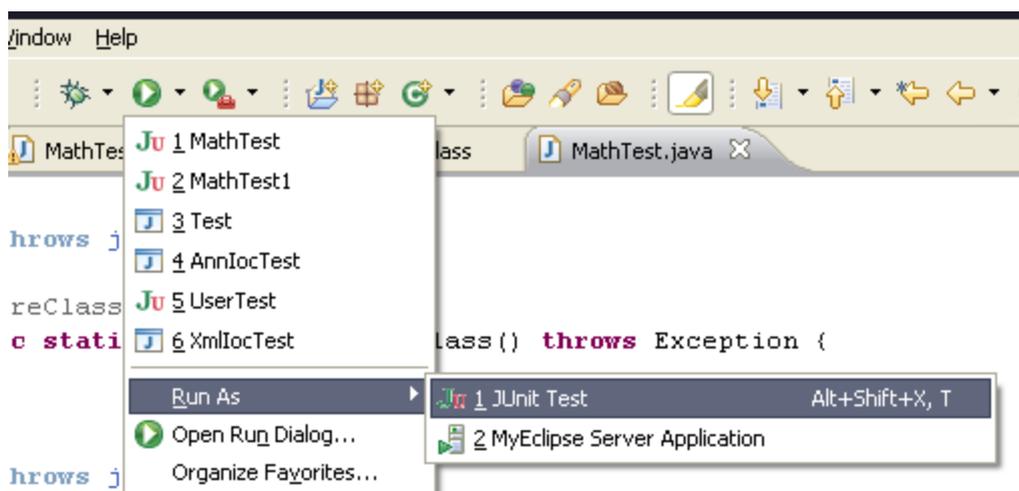
JUnit4 为测试方法增加了判断异常的方式，避免了以前还要通过 `try/catch` 块捕捉异常再抛出的复杂方式，简单的这样声明 “`@Test(expected=ArithmeticException.class)`” **JUnit4** 就会检查此方法是否抛出 `ArithmeticException` 异常，如果抛出则测试通过，没抛出则测试不通过（`@Test` 标签还有一些其他参数，例如超时测试，但是由于并不能准确反应实际时间，所以应用较少）

`@Ignore` 标签会告诉 **JUnit4** 忽略它所标注的方法，例如数据库不可用时可以用此标注标注一些测试数据库连接的方法来避免测试失败。

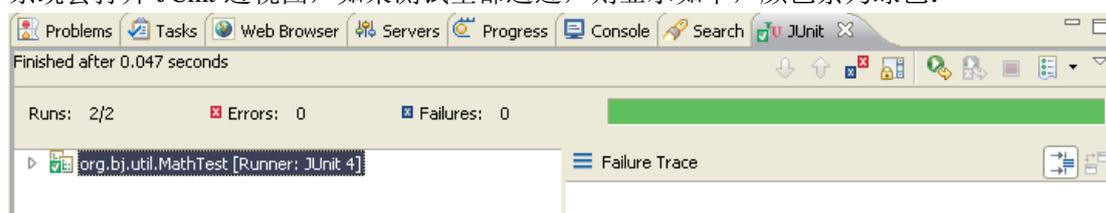
3) 运行测试



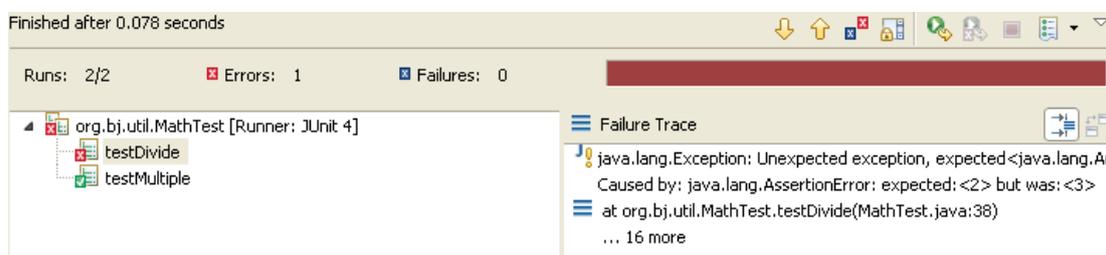
或者



系统会打开 JUnit 透视图，如果测试全部通过，则显示如下，颜色条为绿色：



我们将 `assertEquals(3,Math.divide(9,3));`改成 `assertEquals(2,Math.divide(9,3));`;则显示如下，颜色条为红色：



可以对错误或者故障的地方进行追踪。

4) 创建测试套件

测试套件可以将多个测试用例合在一起测试，将相关的测试用例合成一个测试套件，在做一次修改后，只需要运行测试套件就可以，不需要运行每一个测试用例。

JUnit4 没有采用以前的套件测试方法，同样使用 `annotation` 的方式来进行。简单在你所要构建测试套件的包里创建一个文件，一般以 `包名+4Suite`

下面我在上面的测试包中复制一下之前的测试类并且一个改名字叫做 `MathTestAnother`，新建一个 `class` 类叫做 `Uit4Suite`，代码如下：

```

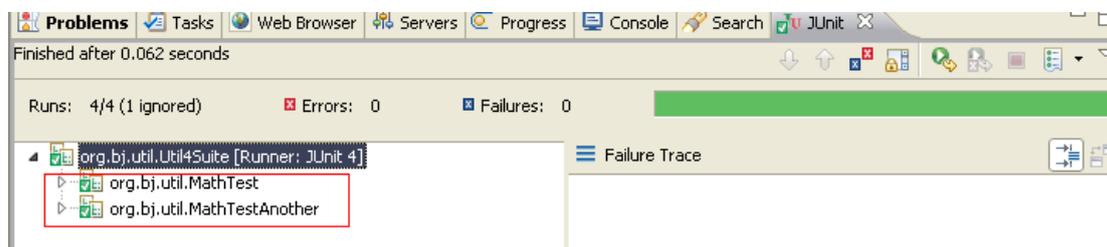
package org.bj.util;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
/**
 * @author bulargy.j.bai
 * @创建时间: Mar 11, 2008
 * @描述:
 */
@RunWith(Suite.class)
@SuiteClasses({MathTest.class,
               MathTestAnother.class})
public class Util4Suite {
}

```

说明:

通过 `@RunWith` 和 `@SuiteClasses` 标签来注释一个空的包含无参数构造函数的类来作为套件类，将需要组成套件运行的类加到 `@SuiteClasses` 的属性中即可。

运行正确后的结构如图:



可以看到运行套件类的结果是 2 个测试类都进行了测试。

5) 参数测试

修改 `testMultiple`

```

//@Ignore("忽略乘法测试")
@Test
public void testMultiple() {

    assertEquals(result, Math.multiply(faciend, multiplicator));
}

```

编写参数方法:

```
@Parameters
public static Collection multipleValues() {
    return Arrays.asList(new Object[][] {
        {3, 2, 6 },
        {4, 3, 12 },
        {5, 7, 35 },
        {6, 7, 42 },
        {11, 7, 77 },
        {25, 4, 100 },
        {34, 3, 102 },
        {21, 5, 105 },
        {11, 22, 242 },
        {8, 9, 72 }});
}
```

说明:

需要使用@Parameters 标签注解一个静态的返回集合对象的方法

增加成员变量和构造函数:

```
int faciend;
int multiplicator;
int result;

public MathTest(int faciend, int multiplicator, int result) {
    this.faciend = faciend;
    this.multiplicator = multiplicator;
    this.result = result;
}
```

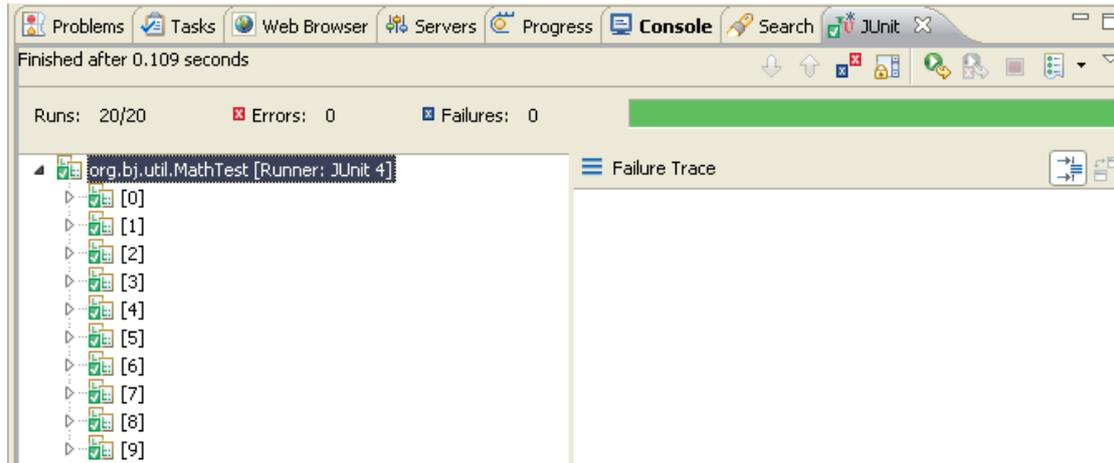
说明:

注意类型要匹配好，不要搞错了

最后在给测试类增加如下注释:

```
@RunWith(Parameterized.class)
```

OK,大功告成。测试看看，结果如下:



成功运行了 9 次。