单元测试工具 Junit 教程和测试覆盖率工具 EclEmma 教程

查阅了很多文章，没有一篇文章是系统介绍单元测试工具 junit 和测试覆盖率工具 eclemma 的。经过一番努终于有一个比较清楚的认识，现总结出来分享给大家，希望能够给学习单元测试的朋友有一些帮助。

第一节：junit 学习--Eclipse 下的环境搭建

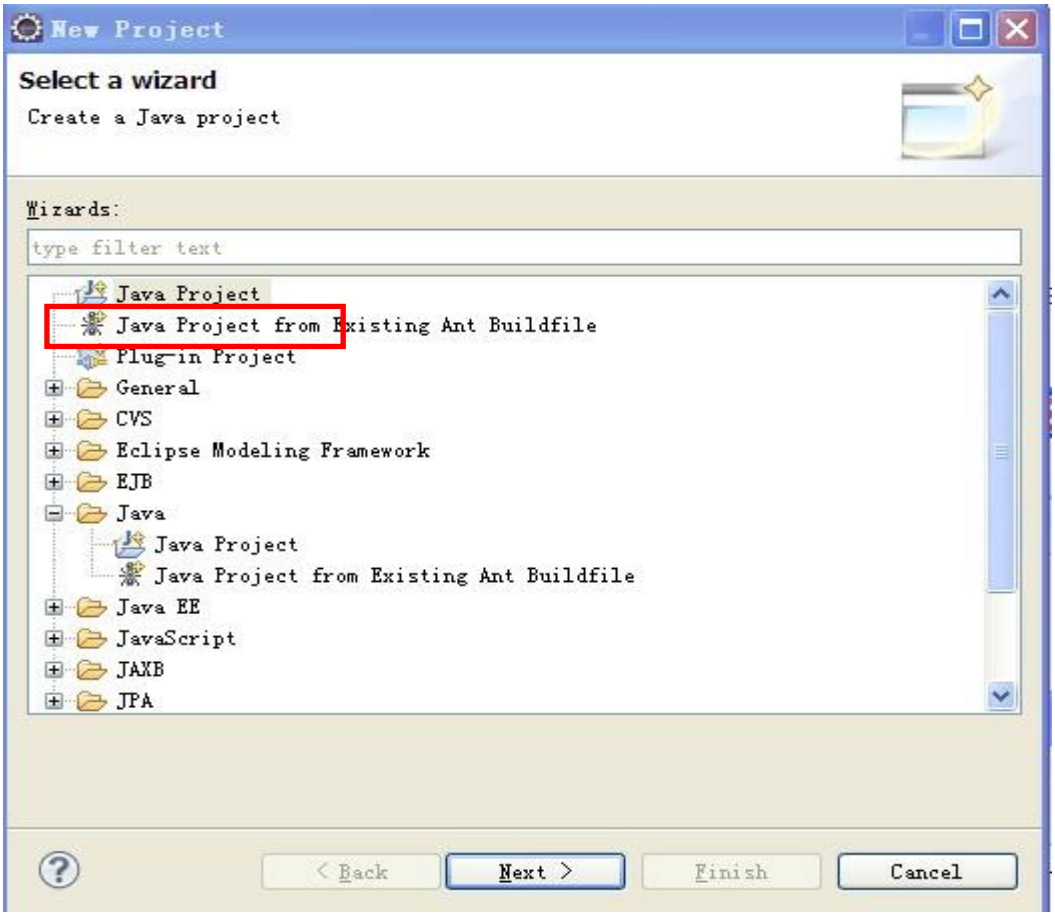
下面要写的目的是：在 Eclipse 中，演示如何创建一个 junit 的测试用例。

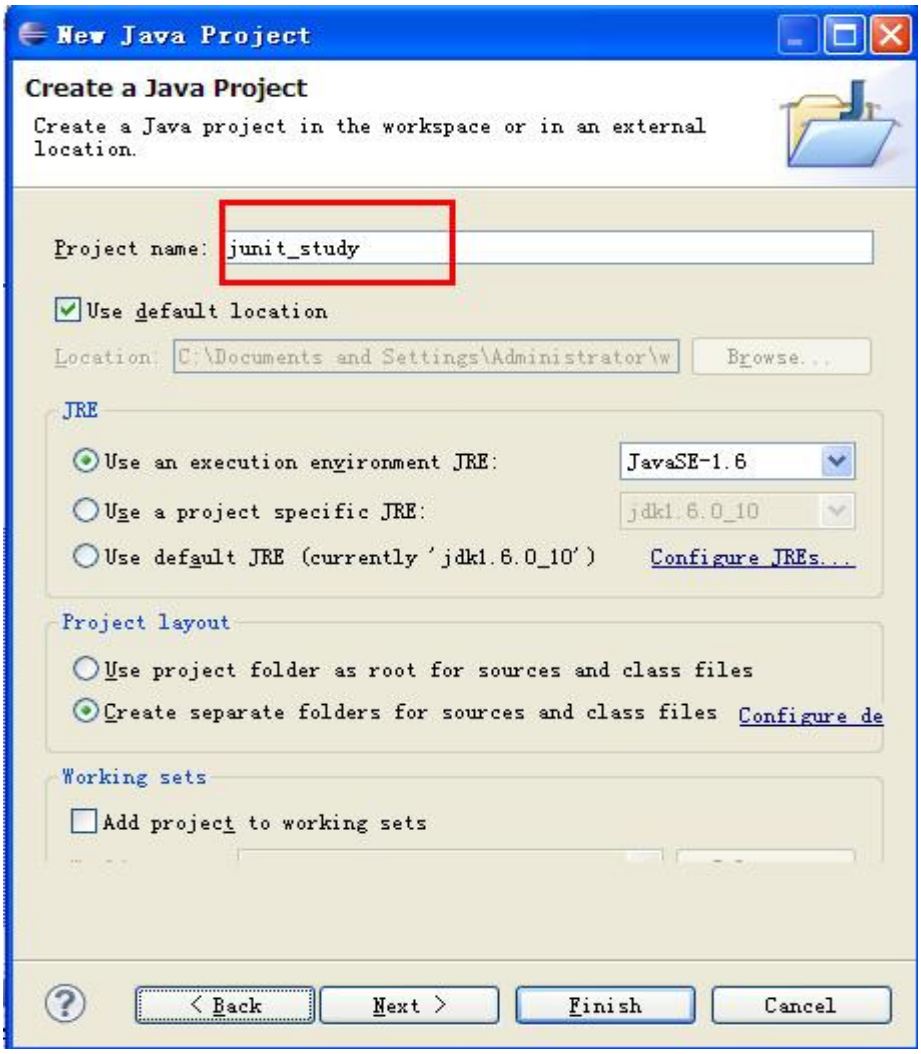
大体的步骤

- 创建 project
- 添加 junit 包
- 创建一个待测试的类 JDemo.java
- 为 JDemo.java 创建一个 Junit 测试用例 JDemoTest.java
- 运行测试

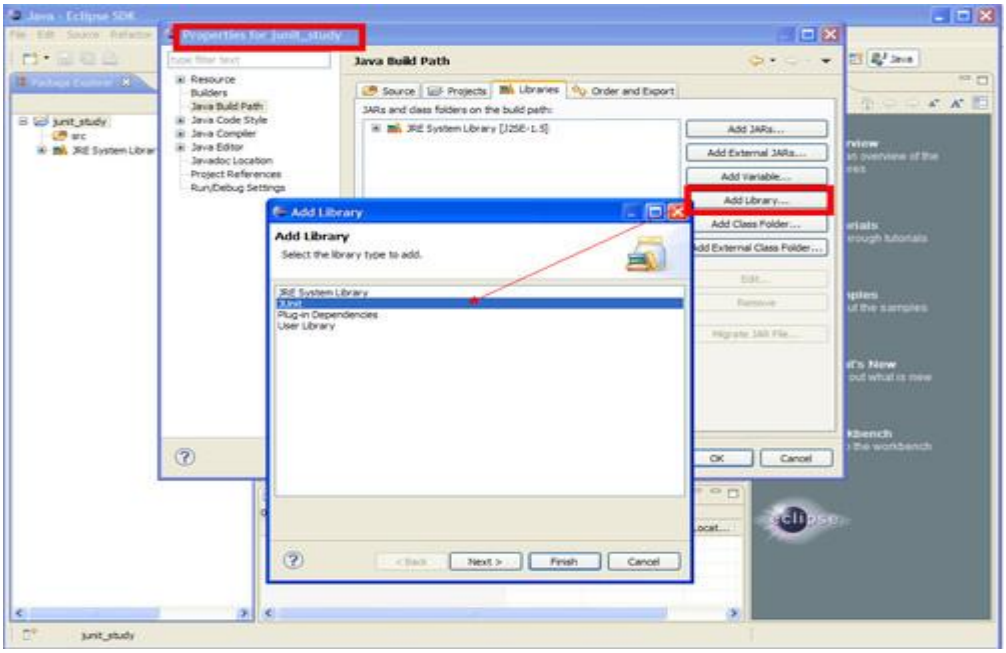
1. 创建一个 project ：【new】-》【project】 （以下截图是操作过程的主要步骤，并非所有）

跳出如下文本框，选择 Java Project，选择 next





2. 然后在 project 的名字上，右键，打开属性窗口，通过【Java build path】-》【library】-》【Add library】添加 junit 的包。



添加的时候，我们选择 junit4 （目前最高版本的 junit）
要说明的，其中添加 junit 包在创建 project 的时候也可以添加的，就是点击第一张图片的时候“Next”进入添加页面。

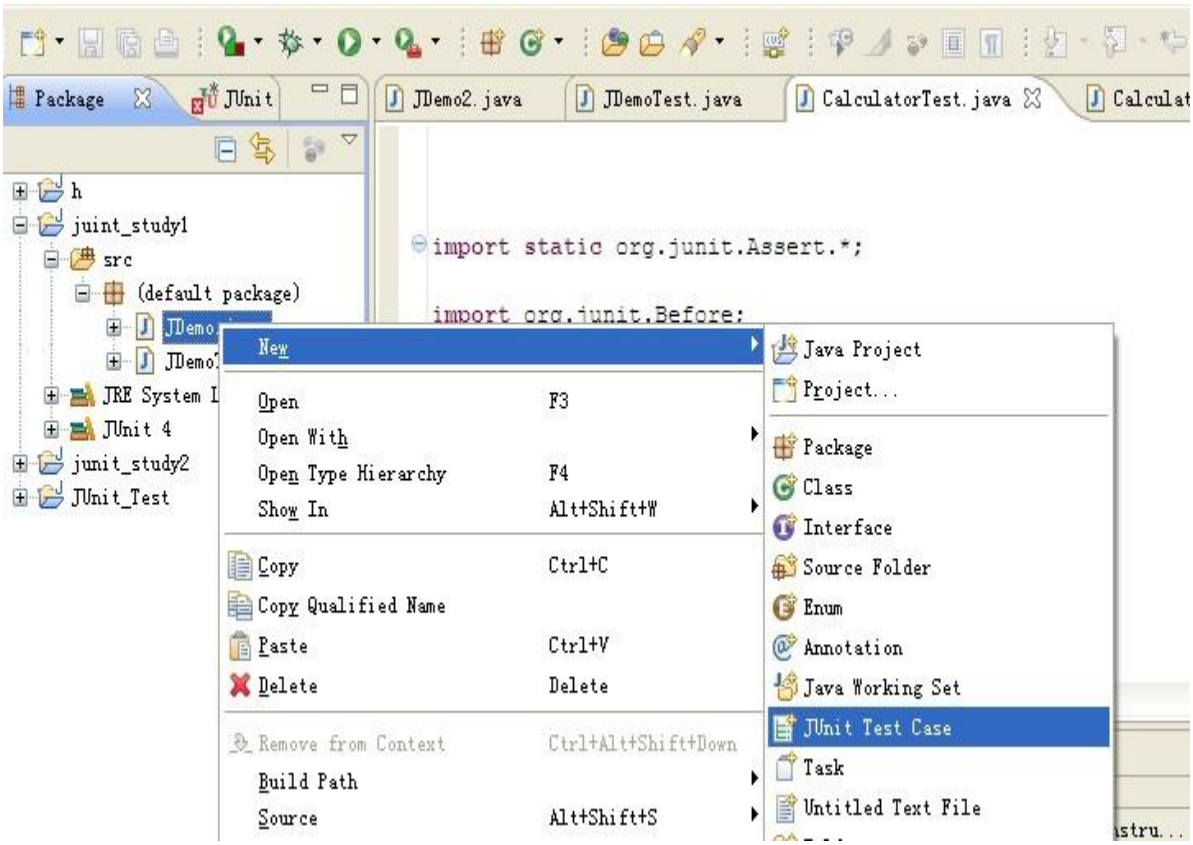
3. 创建一个 java 文件 JDemo.java，后面要利用 junit 对其进行测试

JDemo.java

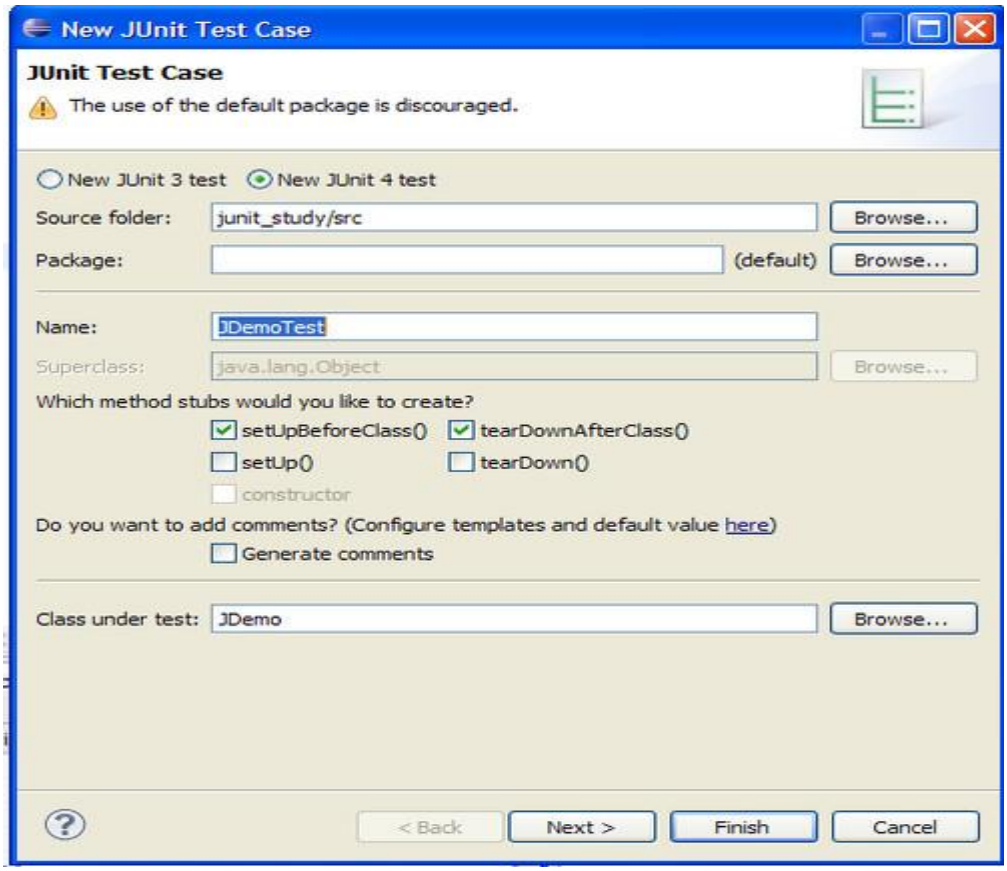
它实现了一个加法的算法。

```
public class JDemo  {
    int a;
    int b;
    int result;

    public int add(int a, int b){
        result = a+b;
        return result;
    }
}
```

4. 为此 JDemo.java 类创建一个 junit 测试用例，在该类的名称上【右键】-》【new】-【Junit Test case】

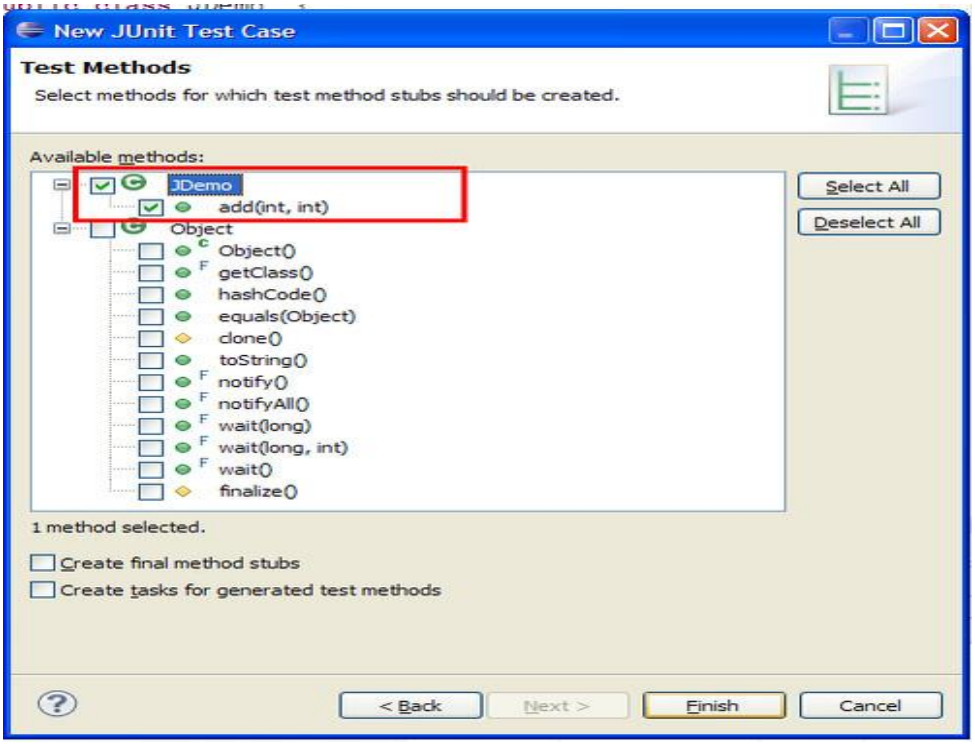


参照下图进行设置



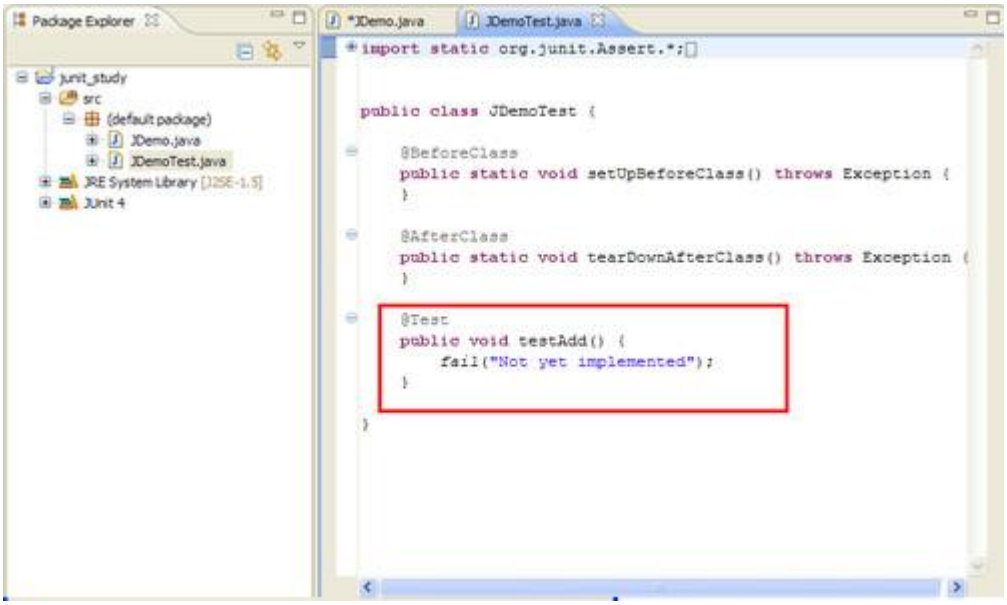
（主要，如果没有在该类的名称上点击，窗口下方的“Class under test”就会为空，如果为空，“Next”键就不可用，也就没有下图的操作。当然，下图的操作也不是必须的，只是按下图操作，Eclipse 会自动为我们在测试用例中创建一个测试方法而已，后面可以看到）

点击【next】



选择我们想要测试的 JDemo.java 中想要测试的方法 add()

点击【Finish】后，Eclipse 就会为我们创建一个用例测试 JDemo.java 的测试用例 JDemoTest.java,并且还自动创建了一个测试方法 TestAdd()



正如前面所说，如果没有选择要测试的方法的那一步，只是这里会少一个测试方法 testAdd()而已，这是可以自己手动加入的，并无实质的影响。

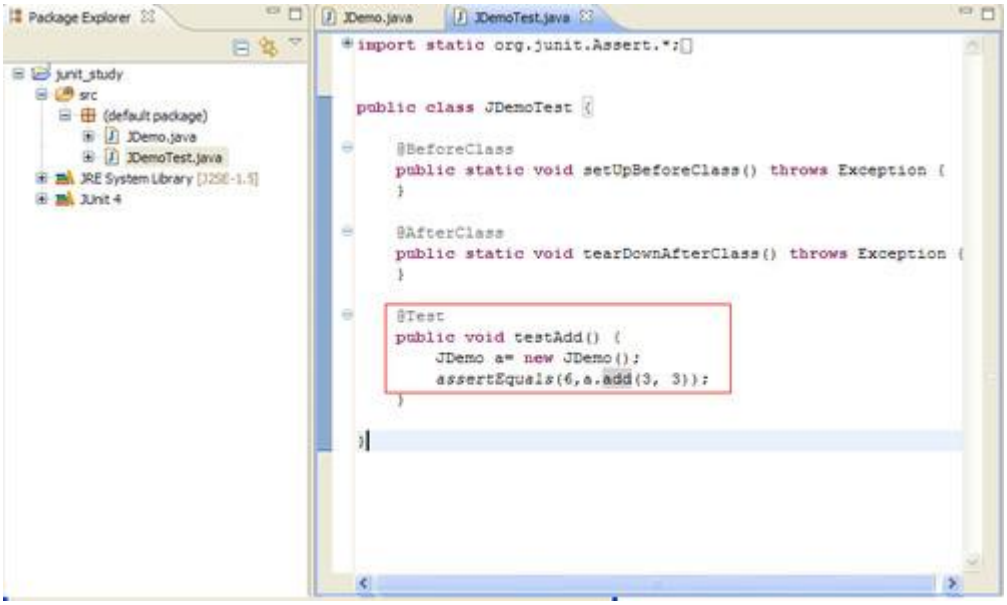
然后再修改 JDemoTest 测试用例中的 testAdd 就行了，按如下修改

JDemoTest.java

```
import static org.junit.Assert.*;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;


public class JDemoTest {

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

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

	@Test
	public void testAdd() {
		JDemo a= new JDemo();
		assertEquals(6,a.add(3, 3));
	}
```

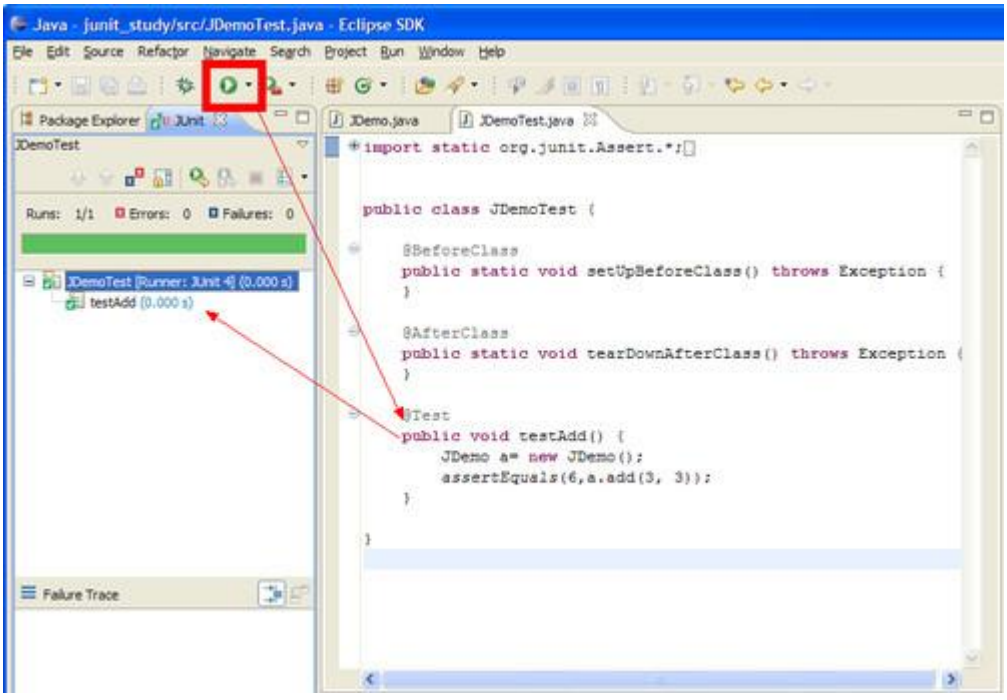
```
}
```

修改完成后，即得到以下显示（注意左边的结构框架：）



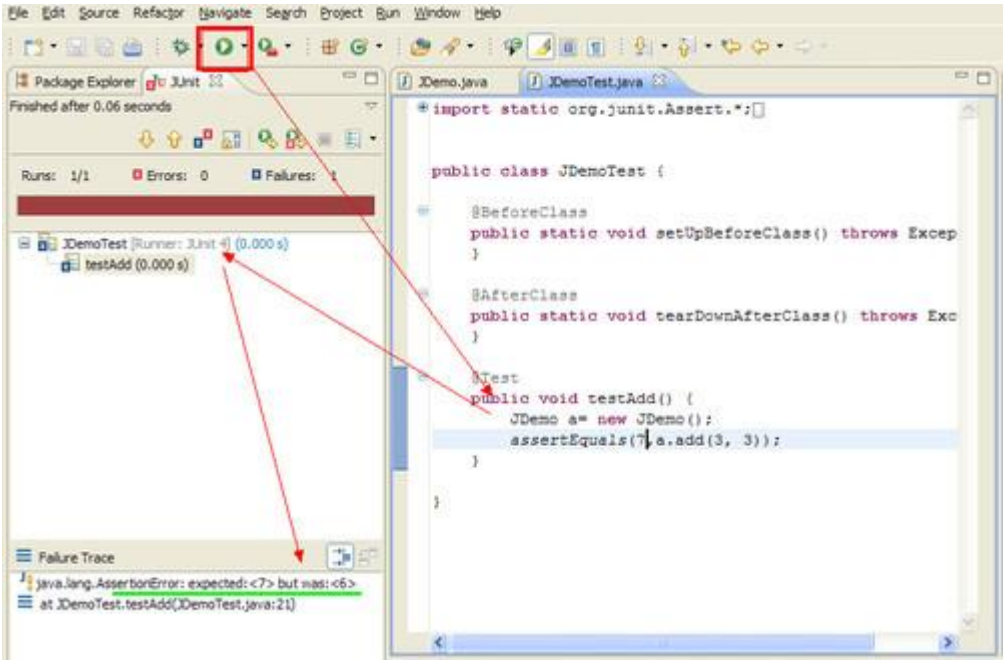
5 运行这个用例

通过点击菜单【运行】，或者右键中的 Run，执行，结果如下



上面用到了 junit 的断言 assertEquals，且上面我们的预期和实际结果是一直的。

下面我们故意修改预期结果，测试错误的情况 junit 是如何处理的



当测试失败后，还可以看到失败的原因。

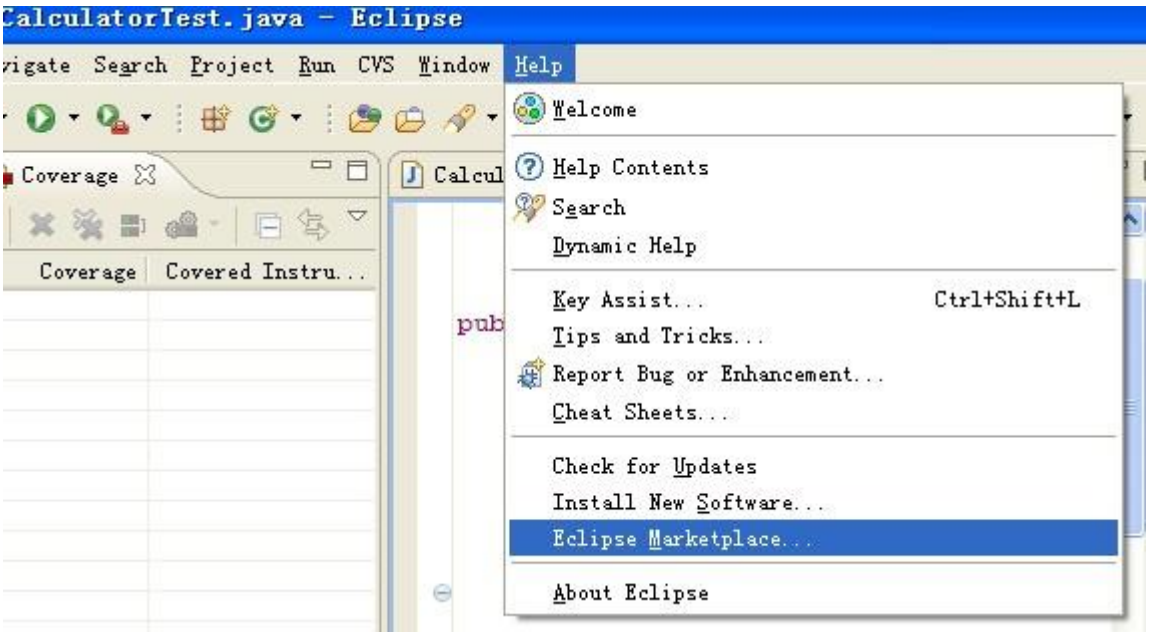
==========

第二节：用测试覆盖率工具 EclEmma 测试覆盖率并生成报告

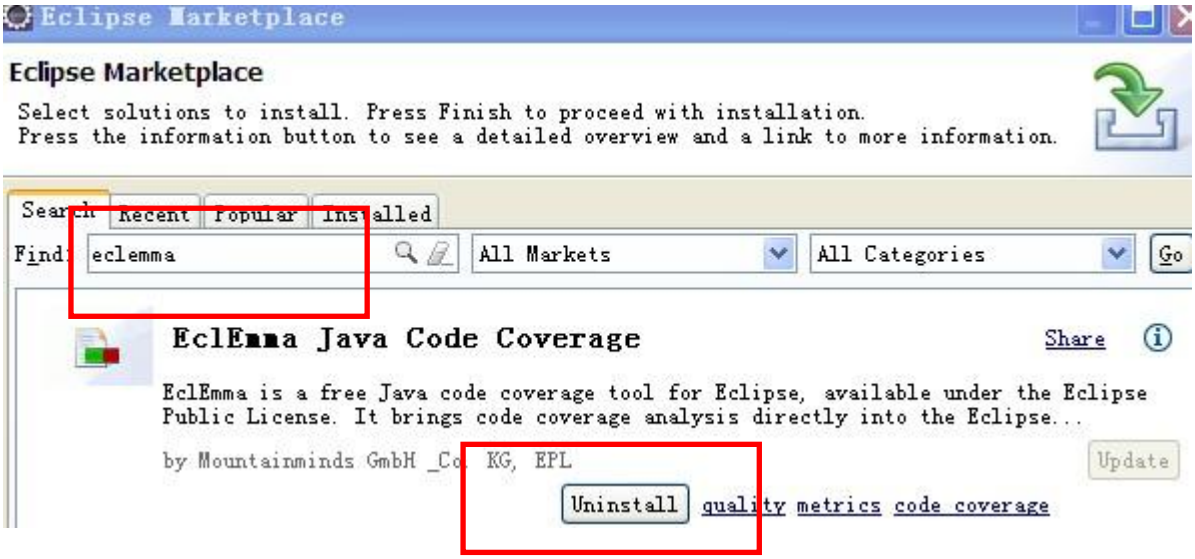
目的：用 EclEmma 测试覆盖率

- EclEmma 安装
- 运行 EclEmma 执行测试
- 查看测试结果
- 导出报告

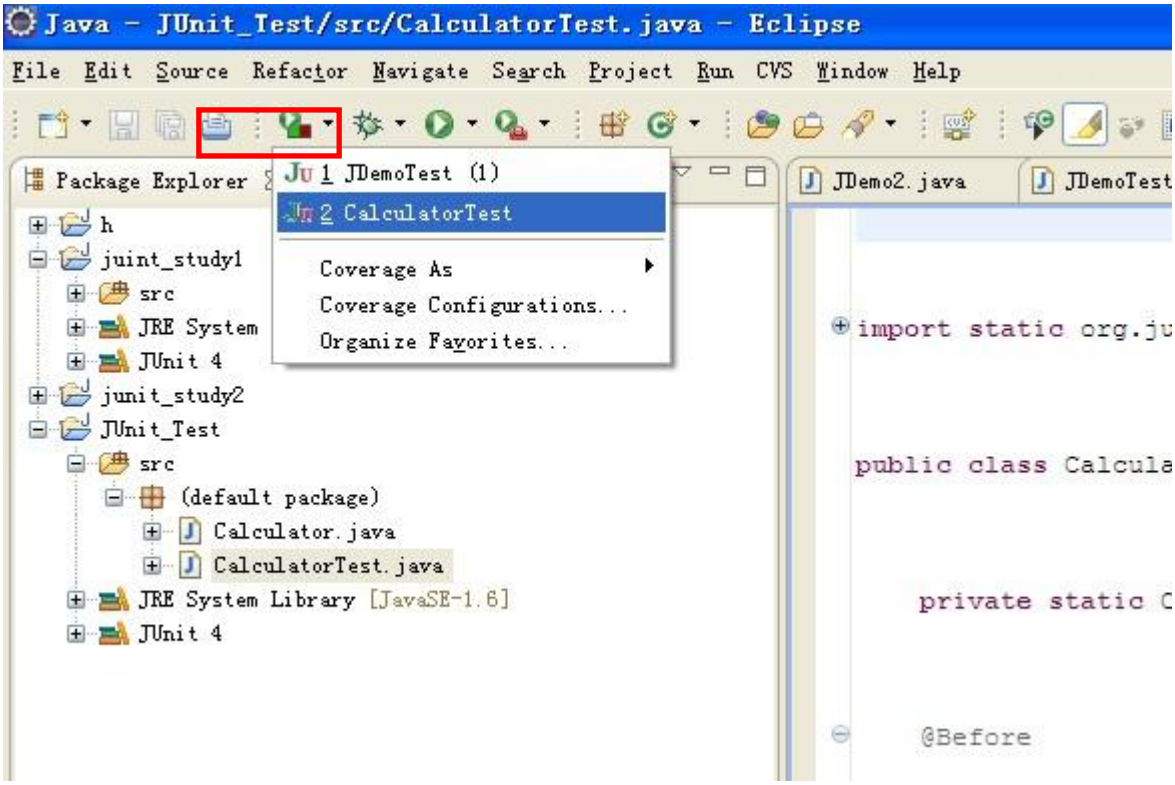
1、 测试覆盖率工具 EclEmma 安装（Eclipse 下 Update 机制远程安装）



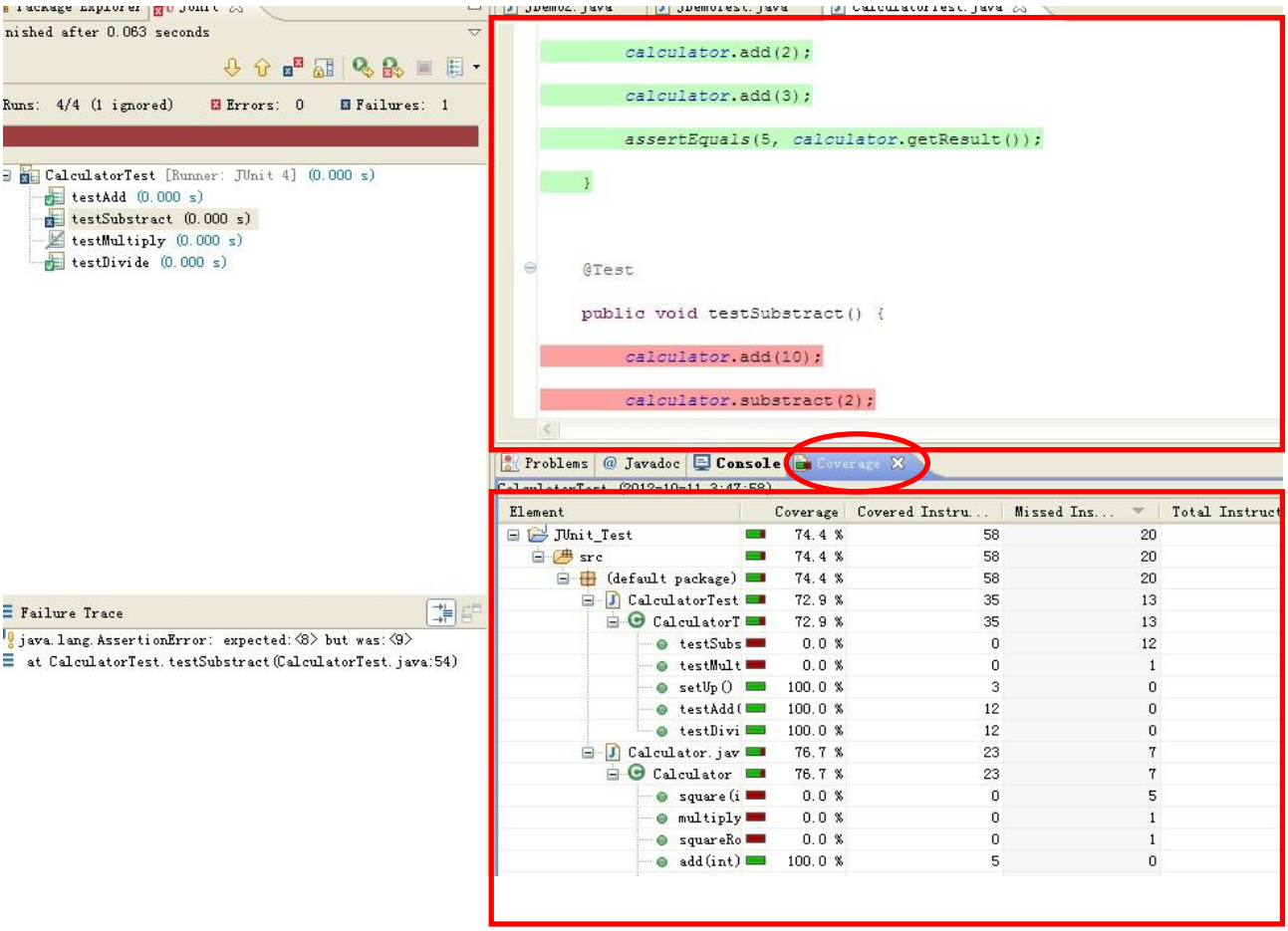
2、在 find 框中输入 eclemma，搜索到以后，点击 uninstall 下载



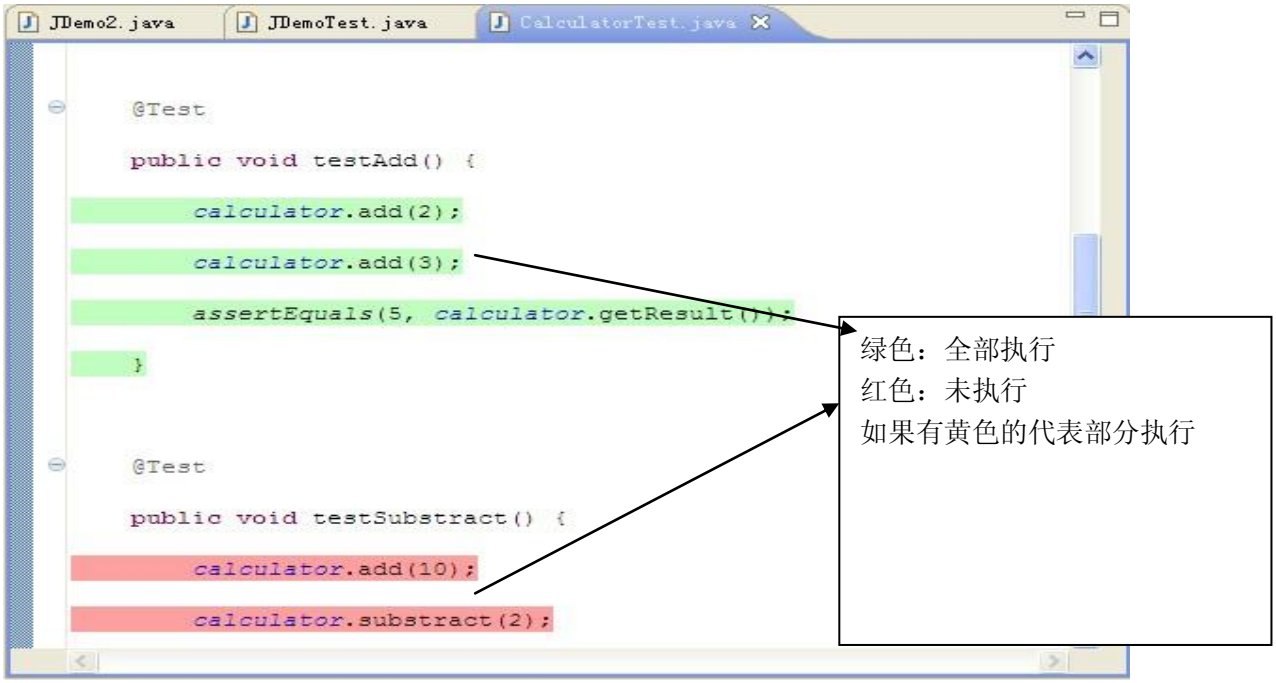
3、安装成功后会看到下面一个小图标。下面以具体的项目为例，也可以使用第一节中的测试用例，为了方便查看测试报告，下面我们以一个稍微复杂的测试用例 CalculatorTest 运行测试程序（代码和测试用例附在本文最下边）



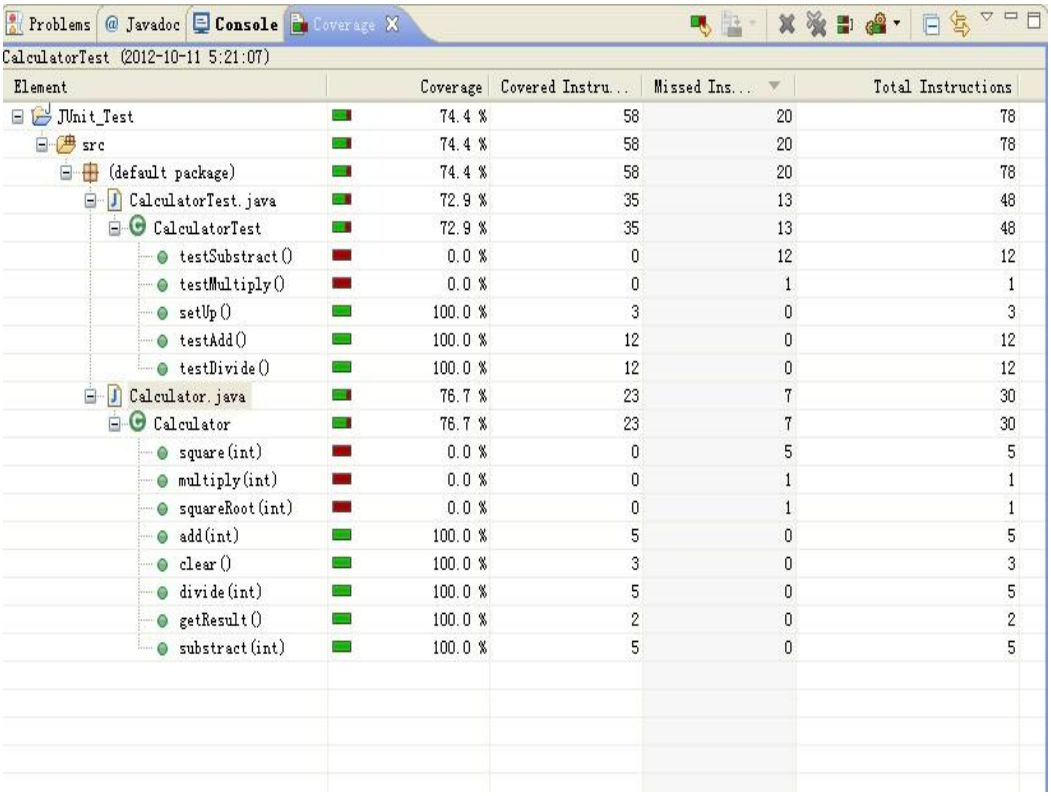
4、查看结果，coverage 显示的就是测试覆盖率结果



5、标注源代码



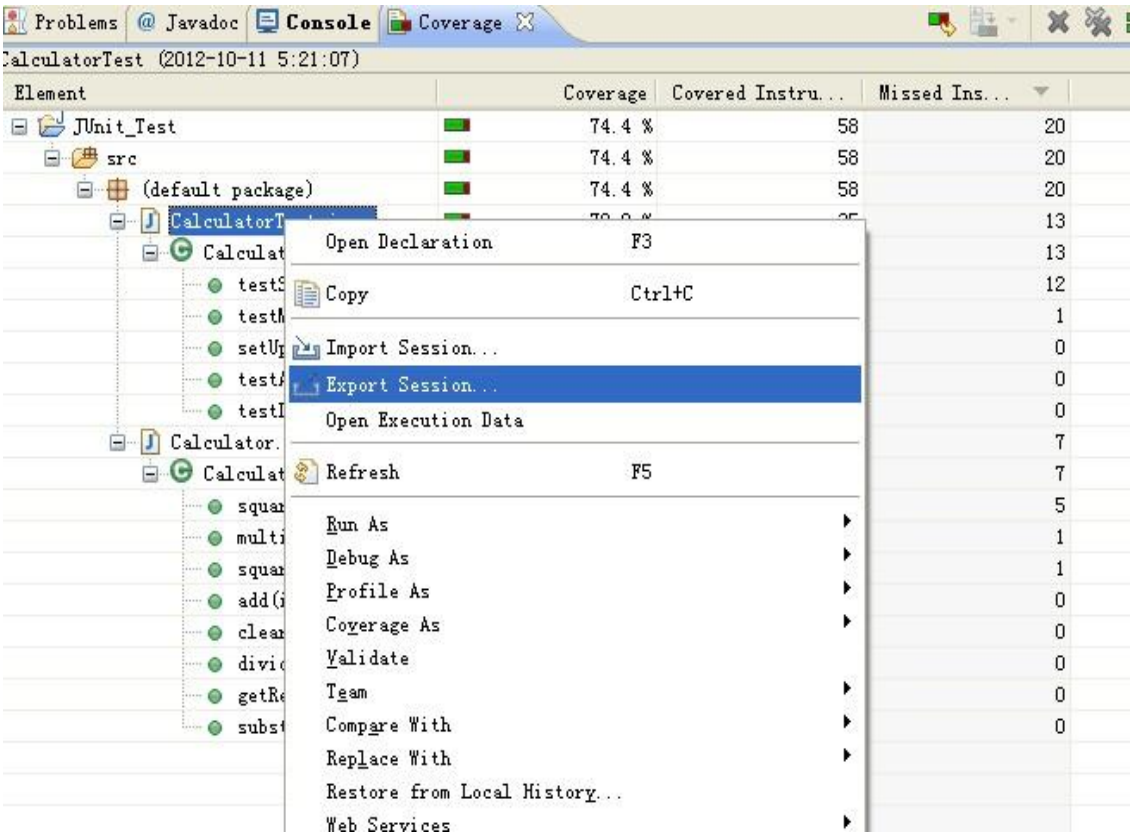
6、Coverage 视图，查看测试覆盖率



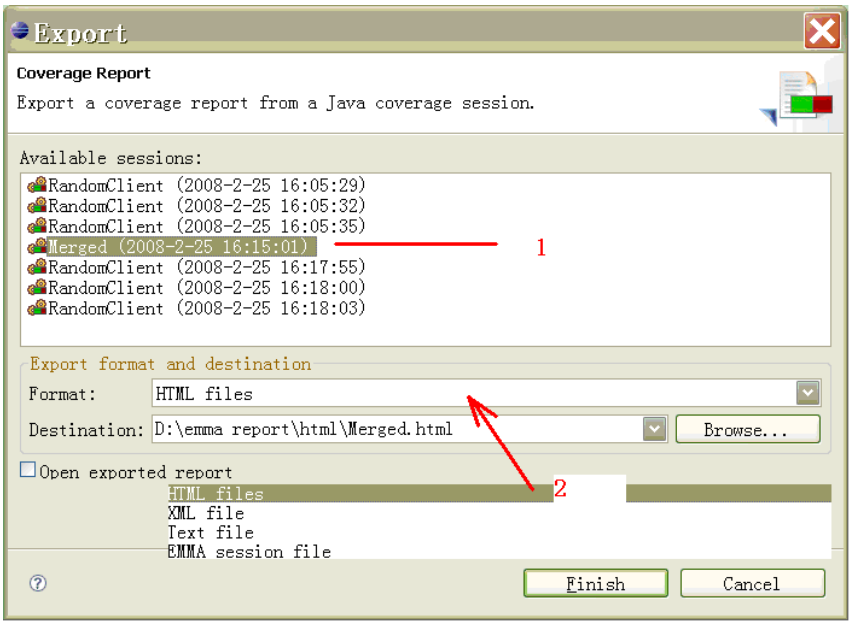
Element	Coverage	Covered Instru...	Missed Ins...	Total Instructions
JUnit_Test	74.4 %	58	20	78
src	74.4 %	58	20	78
(default package)	74.4 %	58	20	78
CalculatorTest.java	72.9 %	35	13	48
CalculatorTest	72.9 %	35	13	48
testSubstract()	0.0 %	0	12	12
testMultiply()	0.0 %	0	1	1
setUp()	100.0 %	3	0	3
testAdd()	100.0 %	12	0	12
testDivide()	100.0 %	12	0	12
Calculator.java	76.7 %	23	7	30
Calculator	76.7 %	23	7	30
square(int)	0.0 %	0	5	5
multiply(int)	0.0 %	0	1	1
squareRoot(int)	0.0 %	0	1	1
add(int)	100.0 %	5	0	5
clear()	100.0 %	3	0	3
divide(int)	100.0 %	5	0	5
getResult()	100.0 %	2	0	2
substract(int)	100.0 %	5	0	5

、

7、导出报告，在 Coverage 视图主区域中点击右键，出现的快捷菜单中选择”Export Session…”，



8、选择存储位置，导出报告的类型。



9、代码和测试用例

Calculator.jave 代码：

```
public class Calculator {

    private static int result; // 静态变量，用于存储运行结果
```

```
public void add(int n) {
    result = result + n;
}
public void substract(int n) {
    result = result - 1;  //Bug: 正确的应该是 result =result-n
}
public void multiply(int n) {
}        // 此方法尚未写好
public void divide(int n) {
    result = result / n;
}
public void square(int n) {
    result = n * n;
}
public void squareRoot(int n) {
    for (; ;) ;           //Bug : 死循环
}
public void clear() {     // 将结果清零
    result = 0;
}
public int getResult() {
    return result;
```

```
    }
}
```

CalculatorTest.jave 测试用例：

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;

public class CalculatorTest {

    private static Calculator calculator = new Calculator();

    @Before
    public void setUp() throws Exception {
        calculator.clear();
    }

    @Test
    public void testAdd() {
        calculator.add(2);
        calculator.add(3);
```

```
        assertEquals(5, calculator.getResult());
    }

    @Test
    public void testSubstract() {
        calculator.add(10);
        calculator.substract(2);
        assertEquals(8, calculator.getResult());
    }

    @Ignore("Multiply() Not yet implemented")
    @Test
    public void testMultiply() {
    }

    @Test
    public void testDivide() {
        calculator.add(8);
        calculator.divide(2);
        assertEquals(4, calculator.getResult());
    }
}
```