

06 JUnit单元测试框架

本章大纲

6.1 什么是单元测试

6.2 JUnit的介绍

6.3 JUnit的常用注解

6.4 hamcrest断言

6.5 参数化

6.6 Test Suite

什么是单元测试

- 什么是单元测试？

单元测试（Unit Testing）是指在计算机编程中，针对程序模块来进行正确性检验的测试。

- 为什么要进行单元测试？

重用测试，应付将来的实现的变化。

提高士气，明确我的代码是没问题的。

单元测试的特点

- 通常采用基于类或者类的方法进行测试
- 程序单元和其他单元是相互独立的。
- 单元的执行速度很快
- 单元测试发现的问题，容易定位
- 通过了解代码的实现逻辑进行测试，通常称之为白盒测试

为什么学习单元测试框架

- 提供用例组织与执行
- 提供丰富的比较方法
- 提供丰富的日志

本章大纲

4.1 什么是单元测试

4.2 JUnit的介绍

4.3 JUnit的常用注解

4.4 hamcrest断言

4.5 参数化

4.6 Test Suite

JUnit的介绍

- JUnit是一个Java语言的单元测试框架。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。
- 最新的JUnit版本是JUnit4
- 作者:Erich Gamma 和 Kent Beck

没有使用Junit会怎么样？

```
public class Calculator {  
    public int add(int x,int y){  
        return x+y;  
    }  
  
    public int sub(int x,int y){  
        return x-y;  
    }  
}
```

```
public class CalculatorTest1 {  
    public static void main(String[] args) {  
        int result = new Calculator().add(1, 1);  
        if (result == 2) {  
            System.out.println("pass");  
        } else {  
            System.out.println("failed");  
        }  
    }  
}
```


JUnit3的使用

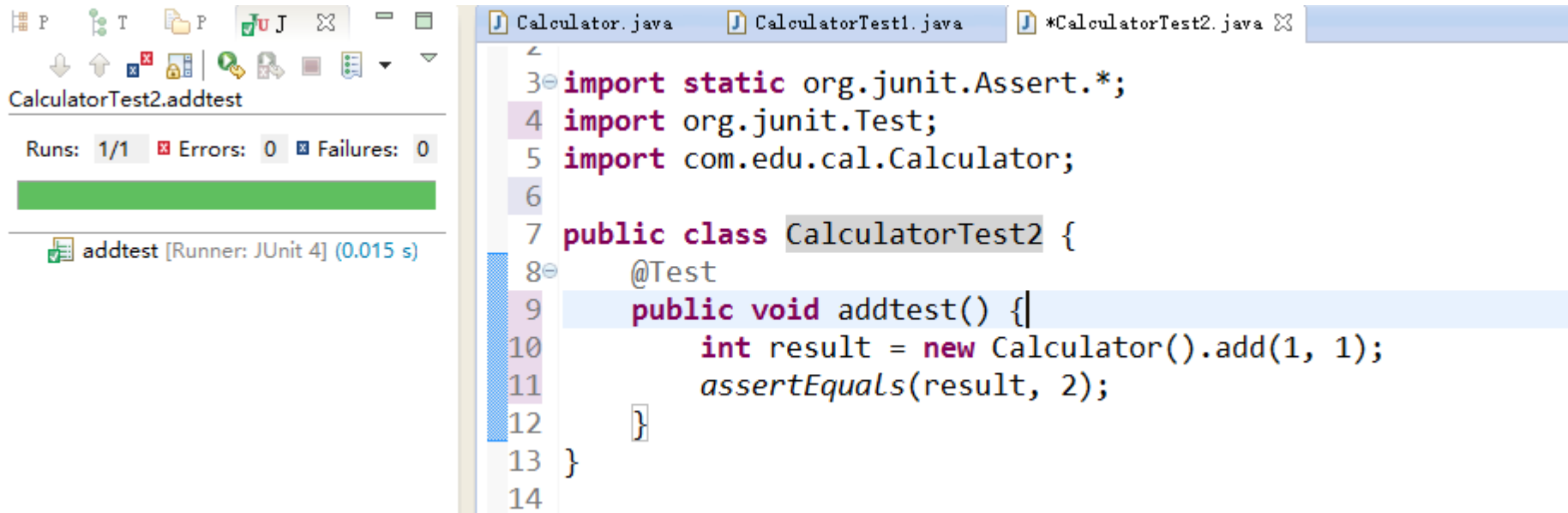
在JUnit3中规定如果是一个测试方法/用例那么必须要遵守以下4点:

- 1.方法void无返回
- 2.test开头的方法名
- 3.方法没有输入参数
- 4.测试的类必须继承于TestCase

JUnit3的使用

```
public class Test1 extends TestCase {  
    protected void setUp() throws Exception {  
        super.setUp();  
        System.out.println("setUp");  
    }  
  
    protected void tearDown() throws Exception {  
        super.tearDown();  
        System.out.println("tearDown");  
    }  
  
    public void testAdd(){  
        System.out.println("testAdd");  
    }  
}
```

使用了JUnit4后



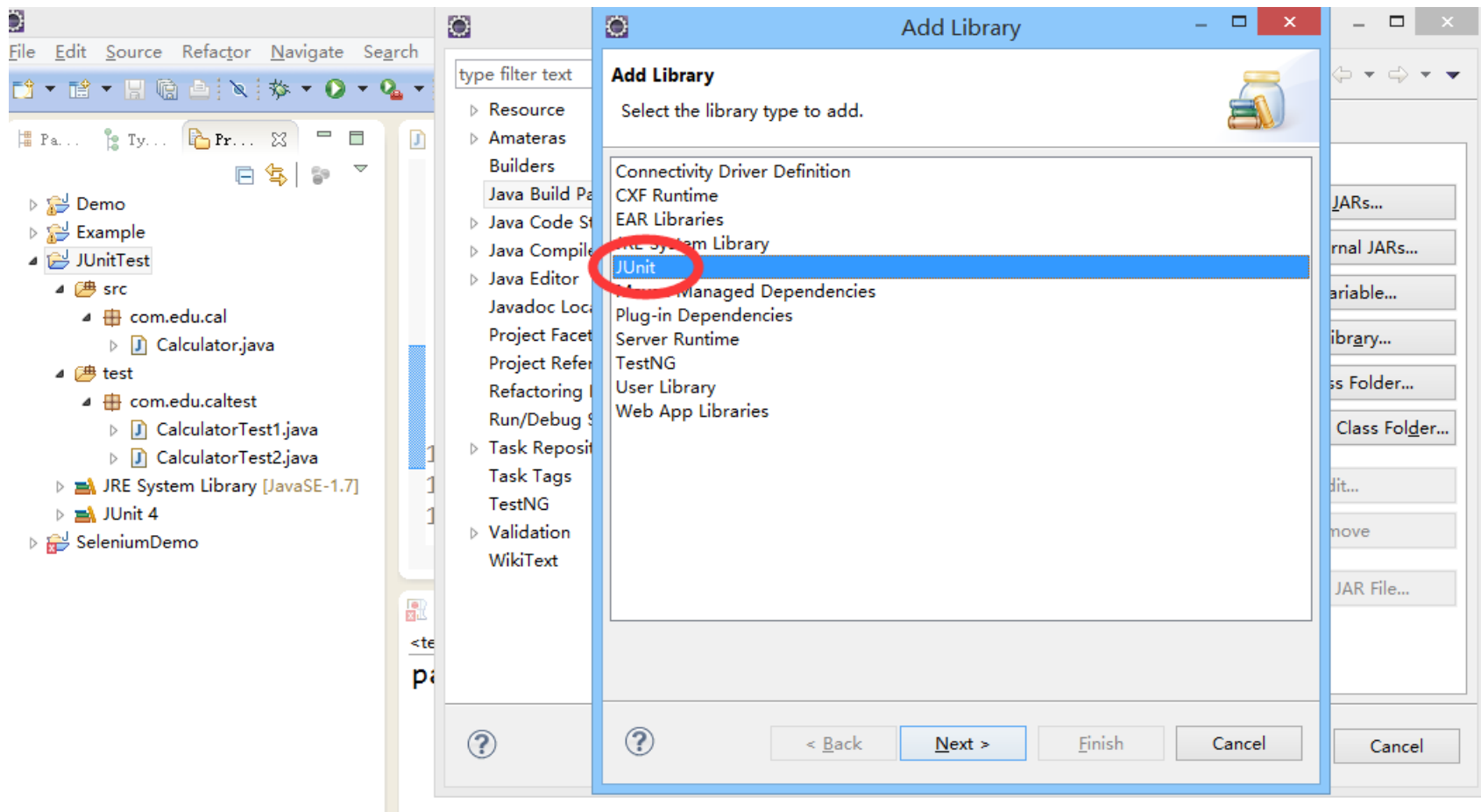
The screenshot displays an IDE interface. On the left, a toolbar contains icons for Project (P), Test (T), Package (P), JUnit (J), and other development tools. Below the toolbar, the text 'CalculatorTest2.addtest' is visible. A status bar shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. A green progress bar is present, and a message indicates 'addtest [Runner: JUnit 4] (0.015 s)'. On the right, the code editor shows three tabs: 'Calculator.java', 'CalculatorTest1.java', and '*CalculatorTest2.java'. The code in the active tab is as follows:

```
1
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import com.edu.cal.Calculator;
6
7 public class CalculatorTest2 {
8     @Test
9     public void addtest() {
10         int result = new Calculator().add(1, 1);
11         assertEquals(result, 2);
12     }
13 }
14
```

Keep the bar green to keep the code clean

导入JUnit

- Build Path->Configure->Add Library



本章大纲

4.1 什么是单元测试

4.2 JUnit的介绍

4.3 JUnit的常用注解

4.4 hamcrest断言

4.5 参数化

4.6 Test Suite

JUnit4 常用注解(JunitDemo.java)

Annotations	含义
@Test	定义一个要测试的方法
@Test(expected=XXException.class)	检测方法是不是抛出了对应的异常
@Test(timeout=100)	如果方法的执行操作毫秒数 >100ms, 那么方法失败
@Before	每一个测试方法之前运行, 常用来进行一些测试环境的准备, 例如: 读入数据, 初始化类
@After	每一个测试方法之后运行, 与@Before对应, 做一个清理/释放的工作
@BeforeClass	所有测试方法之前运行, 只执行一次, 且必须为static void, 类加载时运行。常用做一些所有的测试方法都要依赖的工作: 数据库的连接
@AfterClass	所有测试方法之后运行, 只执行一次, 且必须为static void 与@BeforeClass 相对应, 做一些类级别的清理工作
@Ignore	表明测试方法是被忽略的
Junit用例的执行顺序如下:	运行@BeforeClass->测试类实例化->运行@Before->运行@Test->运行@After->运行@AfterClass

比较预期结果与实际结果

assertEquals(a, b)	测试a是否等于b
assertNotEquals(a, b)	测试a是否不等于b
assertFalse(a)	测试a是否为false
assertTrue(a)	测试a是否为true
assertNull(a)	测试a是否为null
assertNotNull(a)	测试a是否非空
assertSame(a, b)	测试a和b是否都引用同一个对象
assertNotSame(a, b)	测试a和b是否没有都引用同一个对象

Failures和Errors

- **Failures**是指测试失败，预期结果是实际结果不一致
- **Errors**是指测试程序本身出错，代码异常引起的

Failures	Errors
<pre>@Test public void test(){ int i=10; assertEquals(1, i); }</pre>	<pre>@Test public void test(){ int[] i = null; assertEquals(1, i[0]); }</pre>

本章大纲

4.1 什么是单元测试

4.2 JUnit的介绍

4.3 JUnit的常用注解

4.4 hamcrest断言

4.5 参数化

4.6 Test Suite

hamcrest断言

一般匹配符

- `assertThat(s, allOf(greaterThan(1), lessThan(3))));`
- `assertThat(s, anyOf(greaterThan(10), lessThan(5))));`
- `assertThat(s, anything());`
- `assertThat(s, is(2));`
- `assertThat(s, not(1));`
- `assertThat(str, not("hebei"));`

hamcrest断言

数值匹配符

- `assertThat(d, closeTo(3.0, 0.5));`
- `assertThat(d, greaterThan(3.0));`
- `assertThat(d, lessThan(3.5));`
- `assertThat(d, greaterThanOrEqualTo(3.3));`
- `assertThat(d, lessThanOrEqualTo(3.4));`

hamcrest断言

字符串匹配符

- `assertThat(n, containsString("ci"));`
- `assertThat(n, startsWith("Ma"));`
- `assertThat(n, endsWith("i"));`
- `assertThat(n, equalTo("Magci"));`
- `assertThat(n, equalToIgnoringCase("magci"));`
- `assertThat(n, equalToIgnoringWhiteSpace(" M agci "));`

hamcrest断言

集合匹配符

- `assertThat(l, hasItem("Tom"));`
- `assertThat(m, hasEntry(("key", "value")));`
- `assertThat(m, hasKey("mgc"));`
- `assertThat(m, hasValue("Magci"));`

本章大纲

4.1 什么是单元测试

4.2 JUnit的介绍

4.3 JUnit的常用注解

4.4 hamcrest断言

4.5 参数化

4.6 Test Suite

JUnit参数化

- 1.测试类必须被@RunWith (Parameterized.class)修饰
- 2.定义一个方法提供数据，加上一个@Parameters注解，这个方法必须是静态static的，并且返回一个集合Collection
- 3.在测试类的构造方法中为各个参数赋值，（构造方法是由JUnit调用的），最后编写测试类，它会根据参数的组数来运行测试多次

ParaTest.java

参考文档：

<http://www.cnblogs.com/mengdd/archive/2013/04/13/3019336.html>

本章大纲

4.1 什么是单元测试

4.2 JUnit的介绍

4.3 JUnit的常用注解

4.4 hamcrest断言

4.5 参数化

4.6 Test Suite

批量依次执行不同的测试类

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({JUnitDemo.class, CalculatorTest2.class})
public class JUnitSuite {
}
```

总结 JUnit3与JUnit4区别

- JUnit3.8依赖于反射（测试方法必须以test开头），继承于TestCase 类，**JUnit4依赖于注解**@Test、@Before、@After，语法上测试方法可以任意指定(Junit3Test.java)

JUnit3.8	JUnit4
<pre>public void testAdd() { int expected = 3; int trueValue = cal.add(1, 2); assertEquals(expected, trueValue); }</pre>	<pre>@Test public void testAdd() { int expected = 2; int trueValue = cal.divide(4,2); assertEquals(expected, trueValue); }</pre>