6/23/2018 10:54:44 PM

（一）JUnit 介绍

单元测试：可以对重要的程序分支进行测试以发现模块中的错误。

单元测试框架：可以帮助我们完成自动化测试

Junit 官网：http://junit.org/

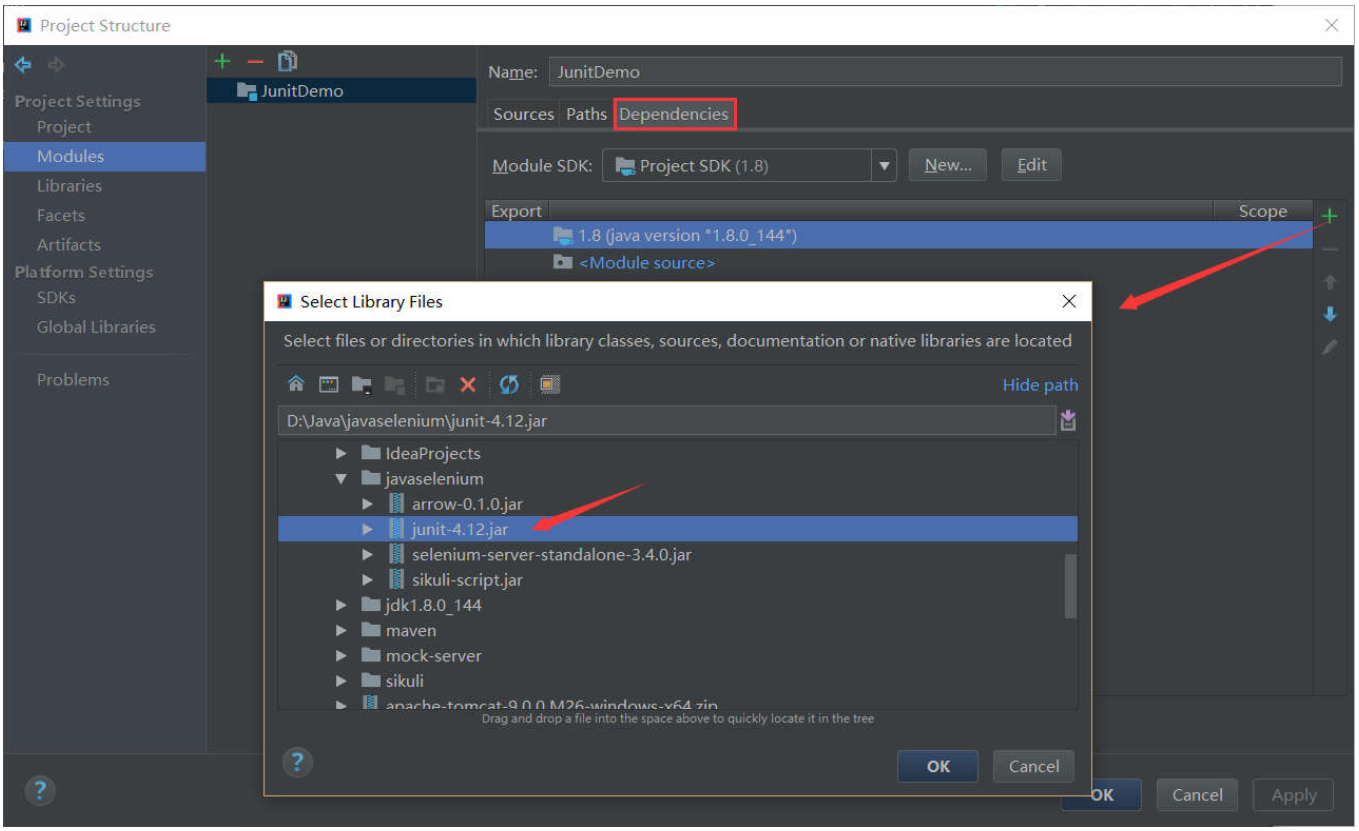
JUnit 是一个编写可重复测试的简单框架。它是单元测试框架的 xUnit 架构的一个实例。

（二）JUnit 安装

1、下载 junit-4.12.jar 文件：https://github.com/junit-team/junit4/releases

2、 菜单栏：File菜单 –> Porject Structure 选项 –> Dependencies 标签 –>

点击 “+” 号 –> Library… –> Java 。 选择下载的 junit-4.12.jar 进行添加。



3、以同样的方式下载和导入 hamcrest： 推荐使用maven下载jar包，然后引入jar包

hamcrest-core-1.3.ORC2.jar：hamcrest的核心包，使用hamcrest框架必须引入的包。

hamcrest-library-1.3.ORC2.jar：包含各种断言，补充hamcrest core包中的断言。

这次我们从中选两个：hamcrest-core-1.3.jar和hamcrest-library-1.3.jar

```
<dependency>
<groupId>org.hamcrest</groupId>
<artifactId>hamcrest-all</artifactId>
<version>1.3</version>
<scope>test</scope>
</dependency>
```

（三）JUnit 编写单元测试

编写单元测试

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

//创建 JunitDemo 类
public class JunitDemo {

    //@Test 用来注释一个普通的方法为一条测试用例。
    @Test
    public void myFirstTest() {
        //assertEquals() 方法用于断言两个值是否相关。
        assertEquals(2+2, 4);
    }

}
```

测试功能模块

```
//创建一个被测试类: Count
public class Count {

    /**
     * 计算并返回两个参数的和
     */
    public int add(int x ,int y){
        return x + y;
    }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

//创建 CountTest 类，用于测试 Count 类。
public class CountTest {

    @Test
```

```
    public void testAdd() {
        //new 出 Count 类，调用 add() 方法并传参，
        //通过 assertEquals() 断言 返回结果。
        Count count = new Count();
        int result = count.add(2,2);
        assertEquals(result, 4);
    }

}
```

（四）JUnit 注解

JUnit 注解说明：

注解	说明
@Test :	标识一条测试用例。 (A) (expected=XXException.class) (B) (timeout=xxx)
@Ignore:	忽略的测试用例。
@Before:	每一个测试方法之前运行。
@After :	每一个测试方法之后运行。
@BefreClass	所有测试开始之前运行。
@AfterClass	所有测试结果之后运行。

```
//创建被测试类 Count
public class Count {

    /**
     * 计算并返回两个参数的和
     */
    public int add(int x ,int y){
        return x + y;
    }

    /**
     * 计算并返回两个数相除的结果
     */
    public int division(int a, int b){
        return a / b;
    }
}
```

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

import org.junit.Ignore;
```

```
import org.junit.Test;

//创建测试类 CountTest
public class CountTest {

    //验证超时
    // timeout=100 , 说明的用例的运行时间不能超过 100 毫秒
    @Test(timeout=100)
    public void testAdd() throws InterruptedException {
        //添加 sleep() 方法休眠 101 毫秒 测试用例执行失败
        Thread.sleep(101);
        new Count().add(1, 1);
    }

    //验证抛出异常
    @Test(expected=ArithmeticException.class)
    public void testDivision() {
        //被除数不能为0 抛出异常符合预期
        new Count().division(8, 0);
    }

    // 直接跳过当前用例
    @Ignore
    @Test
    public void testAdd2() {
        Count count = new Count();
        int result = count.add(2,2);
        assertEquals(result, 5);
    }

}
```

（五）JUnit 注解之Fixture

每次测试开始时都处于一个固定的初始状态；测试结果后需要将测试状态还原.

测试执行所需要的固定环境称为 Test Fixture。

```
import static org.junit.Assert.*;
import org.junit.*;
//被测试类同样使用上一小节的 Count
//创建 TestFixture 测试类。
public class TestFixture {

    //在当前测试类开始时运行。
    @BeforeClass
    public static void beforeClass(){
        System.out.println("-------------------beforeClass");
```

```
    }

    //在当前测试类结束时运行。
    @AfterClass
    public static void afterClass(){
        System.out.println("--------------------afterClass");
    }

    //每个测试方法运行之前运行
    //浏览器驱动的定义放到 @Before 中
    @Before
    public void before(){
        System.out.println("=====before");
    }

    //每个测试方法运行之后运行
    //浏览器的关闭放到 @After 中
    @After
    public void after(){
        System.out.println("=====after");
    }

    @Test
    public void testAdd1() {
        int result=new Count().add(5,3);
        assertEquals(8,result);
        System.out.println("test Run testadd1");
    }

    @Test
    public void testAdd2() {
        int result=new Count().add(15,13);
        assertEquals(28,result);
        System.out.println("test Run testadd2");
    }

}
```

（六）JUnit 用例执行顺序

JUnit 通过 @FixMethodOrder 注解来控制测试方法的执行顺序的。

@FixMethodOrder 注解的参数如下

```
MethodSorters.JVM 按照代码中定义的方法顺序

MethodSorters.DEFAULT(默认的顺序)
```

MethodSorters.NAME_ASCENDING 按方法名字母顺序执行

```
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;
import static org.junit.Assert.assertEquals;

// 按字母顺序执行
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class TestRunSequence {

    @Test
    public void TestCase1() {
        assertEquals(2+2, 4);
    }

    @Test
    public void TestCase2() {
        assertEquals(2+2, 4);
    }

    //TestAa() 先被执行，虽然它在代码中是最后一条用例。
    @Test
    public void TestAa() {
        assertEquals("hello", "hi");
    }
}
```

（七）JUnit 断言方法

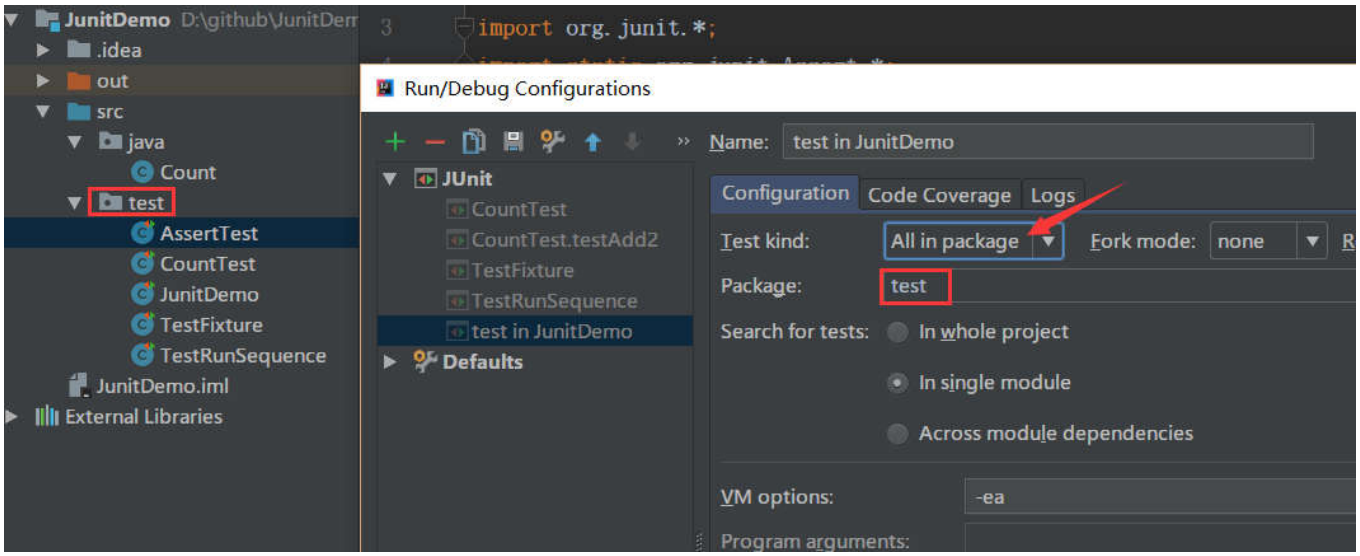
JUnit 所提供的断言方法:

方法	说明
assertArrayEquals(expecteds, actuals)	查看两个数组是否相等。
assertEquals(expected, actual)	查看两个对象是否相等。类似于字符串比较使用的equals()方法。
assertNotEquals(first, second)	查看两个对象是否不相等。
assertNull(object)	查看对象是否为空。
assertNotNull(object)	查看对象是否不为空。
assertSame(expected, actual)	查看两个对象的引用是否相等。类似于使用“==”比较两个对象。
assertNotSame(unexpected, actual)	查看两个对象的引用是否不相等。类似于使用“!=”比较两个对象。
assertTrue(condition)	查看运行结果是否为true。
assertFalse(condition)	查看运行结果是否为false。
assertThat(actual, matcher)	查看实际值是否满足指定的条件。
fail()	让测试失败。

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

//创建 AssertTest 测试类
//包含被测试方法
public class AssertTest {

    /**
     * 判断一个数是否为素数
     */
    public static Boolean Prime(int n) {
        for (int i = 2; i < Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    @Test
    public void testPrime(){
        int n = 7;
        //通过 assertTrue 来断言结果
        assertTrue(AssertTest.Prime(n));
    }

}
```

（八）JUnit 测试批量运行

菜单栏：Run菜单 –> Edit Configurations...选项。



JUnit 提供了一种批量运行测试类的方法，叫测试套件。

通过测试套件运行

```
package test;

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

//下面的为模板代码 实际开发中直接套用即可
@RunWith(Suite.class)
@SuiteClasses({
        //需要批量测试的类
        CountTest.class,
        TestFixture.class,
        AssertTest.class,
        TestRunSequence.class,
})

//创建一个测试类 runAllTest
//保证这个空类使用public修饰，而且存在公开的不带任何参数的构造方法。
public class runAllTest {

}
```