day01_基础加强

一.Junit 单元测试

JUnit是一个Java编程语言的单元测试框架，Junit是xUnit的一个子集，在c++、python、java语言中测试框架的名字都不相同，xUnit是一套基于测试驱动开发的测试框架。程序测试分为黑盒测试、白盒测试，Junit单元测试属于白盒测试的一种。

```java
// 定义一个计算器类
public class Calculator {

    // 定义一个实现加法的方法
    public int add(int a, int b) {
        return a + b;
    }

    // 定义一个实现减法的方法
    public int sub(int a, int b) {
        return a - b;
    }

}
```

```java
// 在main方法中测试Calculator类的成员方法
public class DemoMain {
    public static void main(String[] args) {

        // 创建对象
        Calculator calculator = new Calculator();

        // 调用add方法测试
        int result01 = calculator.add( a: 10,  b: 20);
        System.out.println(result01);

        // 调用sub方法测试
        int result02 = calculator.sub( a: 10,  b: 20);
        System.out.println(result02);

    }
}
```

如上图所示，以前我们测试一个方法的执行，往往是在 main 方法中创建对象调用该方法，然后运行 main 方法查看执行结果。这种测试方法有一定的局限性，下面我们将使用 Junit 来测试 Calculator 类中的成员方法。

1.Junit 的基本使用

```java
package Demo01JunitUnitTest.test;

import Demo01JunitUnitTest.Calculator;
import org.junit.Test;

public class CalculatorTest {

    // 测试add方法
    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        int result = calculator.add( a: 10,  b: 20);
        System.out.println(result);
    }

    // 测试sub方法
    @Test
    public void testSub() {
        Calculator calculator = new Calculator();
        int result = calculator.sub( a: 10,  b: 20);
        System.out.println(result);
    }

}
```

(1).定义一个测试类(测试用例)，建议如下:

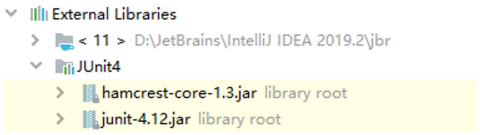
- 测试类名：被测试的类名Test
- 包名：xxx.xxx.test，将所有的测试类都放在一个名为"test"的包中

(2).定义测试方法，建议如下:

- 方法名：test被测试的方法名
- 返回值：void
- 参数：空参

(3).给方法添加 @Test 注解，表明该方法是一个测试方法，并且可以独立运行

(4).导入 Junit 依赖环境，即导入相应的 jar 包 （@Test注解是jar包中的接口，所以需要导入）



定义完测试类、测试方法后，给测试方法加上 @Test 注解，并导入相应的 jar 包，就可以单击方法左侧的箭头来单独执行测试方法了。

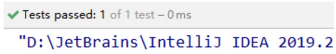
2.Junit 的常见操作

(1).断言

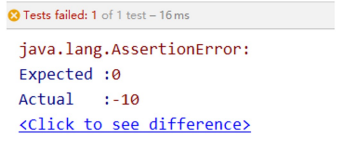
有时只看测试方法的输出结果，并不能达到测试"代码是否有错误"的目的，一般我们会采用断言操作来处理结果。调用 Assert 类中的静态方法 assertEquals(期望结果， 运算结果)，参数传递期望结果、运算结果。当期望结果和运算结果相等时，会显示 "Test passed"；当期望结果和运算结果不等时，会显示 "Test failed"。

```java
import Demo01JunitUnitTest.Calculator;
import org.junit.Assert;
import org.junit.Test;

public class CalculatorTest {

    // 测试add方法
    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        int result = calculator.add( a: 10,  b: 20);
        Assert.assertEquals( expected: 30, result);
    }

    // 测试sub方法
    @Test
    public void testSub() {
        Calculator calculator = new Calculator();
        int result = calculator.sub( a: 10,  b: 20);
        Assert.assertEquals( expected: 0, result);
    }

}
```

testAdd( )方法的执行结果：



testSub( )方法的执行结果：



(2).@Before、@After 注解

- 被 @Before 注解修饰的方法，会在所有测试方法执行之前被自动执行
- 被 @After 注解修饰的方法，会在所有测试方法执行之后被自动执行

```java
package Demo01JunitUnitTest.test;

import Demo01JunitUnitTest.Calculator;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {

    // 初始化的方法：用于资源申请，所有测试方法在执行之前都会先执行该方法
    @Before
    public void init() {
        System.out.println("init...");
    }


    // 释放资源的方法：用于资源释放，所有测试方法执行完后都会自动执行该方法
    @After
    public void close() {
        System.out.println("close...");
    }

    // 测试add方法
    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        int result = calculator.add( a: 10,  b: 20);
        Assert.assertEquals( expected: 30, result);
    }

    // 测试sub方法
    @Test
    public void testSub() {
        Calculator calculator = new Calculator();
        int result = calculator.sub( a: 10,  b: 20);
        Assert.assertEquals( expected: 0, result);
    }

}
```

二.反射