1. 哪些代码需要添加单元测试

第一步，应该对所有程序集中的公开类以及公开类里面的公开方法添加单元测试。

第二步，对于构造函数和公共属性进行单元测试。

第三步，添加全面单元测试。

1. junit 3.x

dome:

public class TestMyNumber extends TestCase {

private MyNumber myNumber;

public TestMyNumber(String name) {

super(name);

}

// 在每个测试方法执行 [之前] 都会被调用

@Override

public void setUp() throws Exception {

// System.out.println("欢迎使用Junit进行单元测试…");

myNumber = new MyNumber();

}

// 在每个测试方法执行 [之后] 都会被调用

@Override

public void tearDown() throws Exception {

// System.out.println("Junit单元测试结束…");

}

public void testDivideByZero() {

Throwable te = null;

try {

myNumber.divide(6, 0);

Assert.fail("测试失败");

} catch (Exception e) {

e.printStackTrace();

te = e;

}

Assert.assertEquals(Exception.class, te.getClass());

Assert.assertEquals("除数不能为 0 ", te.getMessage());

}

}

1. junit 4.x

(1)、使用junit4.x版本进行单元测试时，不用测试类继承TestCase父类，因为，junit4.x全面引入了Annotation来执行我们编写的测试。

(2)、junit4.x版本，引用了注解的方式，进行单元测试；

(3)、junit4.x版本我们常用的注解：

A、@Before 注解：与junit3.x中的setUp()方法功能一样，在每个测试方法之前执行；

B、@After 注解：与junit3.x中的tearDown()方法功能一样，在每个测试方法之后执行；

C、@BeforeClass 注解：在所有方法执行之前执行；

D、@AfterClass 注解：在所有方法执行之后执行；

E、@Test(timeout = xxx) 注解：设置当前测试方法在一定时间内运行完，否则返回错误；

F、@Test(expected = Exception.class) 注解：设置被测试的方法是否有异常抛出。抛出异常类型为：Exception.class；

G、@Ignore 注解：注释掉一个测试方法或一个类，被注释的方法或类，不会被执行。

dome:

package com.an.junit;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

public class TestMyNumber {

private MyNumber myNumber;

@BeforeClass

// 在所有方法执行之前执行

public static void globalInit() {

System.out.println("init all method...");

}

@AfterClass

// 在所有方法执行之后执行

public static void globalDestory() {

System.out.println("destory all method...");

}

@Before

// 在每个测试方法之前执行

public void setUp() {

System.out.println("start setUp method");

myNumber = new MyNumber();

}

@After

// 在每个测试方法之后执行

public void tearDown() {

System.out.println("end tearDown method");

}

@Test(timeout=600)// 设置限定测试方法的运行时间 如果超出则返回错误

public void testAdd() {

System.out.println("testAdd method");

int result = myNumber.add(2, 3);

assertEquals(5, result);

}

@Test

public void testSubtract() {

System.out.println("testSubtract method");

int result = myNumber.subtract(1, 2);

assertEquals(-1, result);

}

@Test

public void testMultiply() {

System.out.println("testMultiply method");

int result = myNumber.multiply(2, 3);

assertEquals(6, result);

}

@Test

public void testDivide() {

System.out.println("testDivide method");

int result = 0;

try {

result = myNumber.divide(6, 2);

} catch (Exception e) {

fail();

}

assertEquals(3, result);

}

@Test(expected = Exception.class)

public void testDivide2() throws Exception {

System.out.println("testDivide2 method");

myNumber.divide(6, 0);

fail("test Error");

}

public static void main(String[] args) {

}

}