Junit的基本使用

# 基本的Junit测试类

## 声明测试方法

|  |
| --- |
| #测试方法必须使用@Test修饰  #测试方法必须使用public void修饰，并且不带参数  #新建test源代码目录来存放测试代码  #测试类的包名应该和被测试类的报名保持一致  #测试单元中的每个方法必须可以独立测试，测试方法之间不能有任何依赖  #测试类一般使用Test作为后缀  #测试方法一般使用test作为前缀  public class CalculatorTest {  private static Calculator calculator;  @BeforeClass  public static void BuildCalculator() {  calculator = new Calculator();  }  @ Test  public void testAdd() {  Assert.assertEquals(8, calculator.add(3, 5));  }  } |

## Junit测试的生命周期

|  |
| --- |
| public class MethodTest {    @BeforeClass //用static修饰  public static void setUpBeforeClass() throws Exception {  System.out.println("beforeClass...");  }    @AfterClass //用static修饰  public static void tearDownAfterClass() throws Exception {  System.out.println("afterClass...");  }    @Before  public void setUp() throws Exception {  System.out.println("before...");  }    @After  public void tearDown() throws Exception {  System.out.println("after...");  }    @ Test  public void test1() {  System.out.println("test1...");  // fail("Not yet implemented");  }    @ Test  public void test2() {  System.out.println("test2...");  // fail("Not yet implemented");  }    }  输出如下：  beforeClass...  before...  test1...  after...  before...  test2...  after...  afterClass... |
| 注解的执行顺序和作用：  @Test将一个普通的方法修饰为一个测试方法  @BeforeClass会在所有方法运行前执行，static修饰  @AfterClass会在所有方法运行结束后执行，static修饰  @Before会在每个测试方法运行前执行一次  @After会在每个测试方法运行后被执行一次 |

# Junit断言使用

## 常用断言API

|  |
| --- |
| assertEquals(? expected, ? actual)：断言两个对象或者基本对象是否相等  assertArrayEquals(? expected, ? actual):断言两个数组是否包含相同的元素  assertTrue(boolean condition)：断言语句真  assertFalse(boolean condition)：断言语句为假  assertNull(obj)：断言对象为空  assertNotNull(obj)：断言对象不为空  assertSame(? expected, ? actual)：断言两个对象引用指向同一实例  assertNotSame(? expected, ? actual);断言两个对象引用指向不同实例 |

## 抛出断言异常

有时你想要的代码就只在某些情况抛出异常，比如，无效的输入，你希望抛出IllegalArgumentException。如果没有抛出异常或者抛出其他异常，则测试图不通过。

|  |
| --- |
| **public** **void** throwSomeException() **throws** IllegalArgumentException{  **throw** **new** IllegalArgumentException("抛出异常了");    }    @Test(expected=IllegalArgumentException.**class**)  **public** **void** test3(){  throwSomeException();  } |

# Junit扩展使用

## 使用运行器来控制测试的执行

@Ignore作用是让测试运行器忽略其修饰的测试方法。@Ignore所修饰的方法不会执行。@RunWith可以用来更改运行测试器org.junit.runner.Runner。

|  |
| --- |
| #@RunWith来修改测试运行器。  #然后使用@Suite.SuiteClasses来指定要批量执行的测试类（数组的形式）  @RunWith(Suite.**class**)  @Suite.SuiteClasses(StringTest.**class**)  **public** **class** SuiteTest {    } |
| #创建一个名为ParameterTest的测试类，并用RunWith注解来改变测试运行器。  #声明变量来存放预期值和结果值。  #声明一个返回值为Collection的公共静态方法，并使用 @Parameters进行修饰。  #为测试类声明一个带有参数的公共构造方法，并在其中为之声明变量赋值。  @RunWith(Parameterized.class)  public class ParameterTest {    private static Calculator calculator;    @BeforeClass  public static void setUpBeforeClass() {  calculator = new Calculator();  }    private int expected;  private int input1;  private int input2;    @Parameters  public static Collection<Object[]> setParameters() {  Object[][] objs = {{8,3,5},{5,3,2}};    return Arrays.asList(objs);  }    public ParameterTest(int expected, int input1, int input2) {  this.expected = expected;  this.input1 = input1;  this.input2 = input2;  }    @ Test  public void testParameters() {  Assert.assertEquals(expected, calculator.add(input1, input2));  }  } |

## 用规则来装饰测试

### 配置全局超时

@Test(timeout=XX)可以用于某一个测试方法。全局的超时配置如下：

|  |
| --- |
|  |

第一步：定义一个MethodRule类型的公有字段

第二步：字段带有注解@Rule

### 预期异常

## 临时文件夹