JUnit

JUnit 是 Java 程序的一种**格式 format** 和**框架 framework**

用于**自动地 automatically** **指定 specifying** 和**执行 executing** 测试用例

介绍

为什么要用 JUnit

开发人员测试比 println-ing 更高级、更简单

- 自然支持连续测试
- 帮助定位错误 fault
- 自动化测试执行过程
- 辅助对测试系统的替代思考
- 是一个软件规范（预期的行为）
- 整合测试结果和执行统计数据

JUnit vs Printgln-ing

```
@Test
public void testAdd(){
    Calculator calculator = new Calculator();
    double result = calculator.add(10, 50);
    assertEquals(60, result, 0);
}
```

- 正确使用 `main()` 方法
- 自动测试执行和生成报告
- 无需人工检查

```
public class CalculatorTest{
    public static void main(String[] args){
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        if(result != 60){
            System.out.println("Bad result: " + reqult);
        }
    }
}
```

- 不必要的 `main()` 方法

- 检查手动输出

逻辑结构

Assert

一组检查条件的方法

```
1   assertEquals("", result);
```

- 如果断言失败 fails （检查条件不满足）
    - 当前测试被标记为 `filed`
    - JUnit 跳过其余测试方法的执行，继续执行其余的测试方法

类型

对于 boolean

```
1   assertTrue(“message for fail”, condition);
2   assertFalse(“message”, condition);
```

对于 object, int, long, byte, array 等

```
1   assertEquals(expected_value, expression);
```

对于 float 和 double

```
1   assertEquals(expected, expression, error);
```

对于引用对象

```
1   assertNull(obj_ref);
2   assertNotNull(obj_ref);
3   assertSame(obj_ref, obj_ref2);
```

assertEquals vs. assertSame

- 对于值，使用 `assertEquals`
    - 检查值是否相同
- 对于引用对象，使用 `assertSame`
    - 检查对象地址是否相同

在比较字符串时要谨慎

```
@Test
public void testEquality() {
    String a = "abcde";
    String b = new String(a);
    assertTrue(a.equals(b)); // true
    assertFalse(a == b); // true
    assertEquals(a, b); // true
    String c = "abcde";
    assertNotSame(a, b); // true
    assertSame(a,c); // true
}
```

测试类 Test Class

测试方法 ∈ 测试类

聚合 `@Test` 方法和特殊辅助 JUnit 方法（设置和删除）的公共类

```
public class ListTest {
  protected List<Integer> fEmpty;
  protected List<Integer> fFull;
  protected static List<Integer> fgHeavy;

  @Before
  public void setup() {
    fEmpty = new ArrayList<Integer>();
    fFull = new ArrayList<Integer>();
    fFull.add(1);
    fFull.add(2);
    fFull.add(3);
  }

  @After
  public void tearDown() {
    fFull = null;
    ....
  }
```

- Setup 方法
  - 为测试执行准备公共对象/资源
  - `@Before` 在每个测试方法之前执行
  - `@BeforeClass` 对测试类执行一次
- Tear-down 方法
  - 方法在测试执行后释放公共对象/资源
  - `@After`

测试套件 Test Suite

允许将测试类分组在不同的集合中进行测试执行

```
@RunWith(Suite.class)
@SuiteClasses({CSVRendererTest.class, EmacsRendererTest.class,
XMLRendererTest.class,
TextPadRendererTest.class})

public class RenderersTests {
}
```

测试异常

有两种情况

- 我们希望代码有一个正常的行为，不要有异常
- 我们期待给出一个异常行为，然后产生异常

例如有这样一个类

```
class TheClass {
    public void method(String p) throws PossibleException{
        // ......
    }
}
```

正常的行为

```
try {
    // We call the method with correct parameters
    object.method("Parameter");
    assertTrue(true); // OK
} catch(PossibleException e){
    fail("method should not fail !!!");
}

```

希望出现异常

```
try {
    // we call the method with wrong parameters
    object.method(null);
    fail(“method should fail!!");
} catch(PossibleException e){
    assertTrue(true); // OK
}
```