南京大学 NANJING UNIVERSITY · SOFTWARE INSTITUTE 软件学院

Computing And Software Engineering

单元测试

- 单元测试
  - 开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确过度滥用，含义混乱
- 单元是一段方法？一个类？一段代码？
- 包含类得交互的就是集成测试？

南京大学 NANJING UNIVERSITY · SOFTWARE INSTITUTE 软件学院

Computing And Software Engineering

程序员测试

- 带来比功能测试更广范围的测试覆盖
- 建立完善的测试用例集且频繁执行
- 让团队协作成为可能
- 能够防止衰退，降低对调试的需要
- 能为我们带来重构的勇气
- 能改进实现设计
- 当作开发者文档使用

程序员测试框架

- 测试必须自动化
  - 断点、按钮（调试）需要大量精力
- 测试必须自我校验
  - 不用程序员观察变量值或输出值（ok、oops）
- 多个测试很容易同时进行
  - 集中精力在怎样编写测试和保证程序正确性上
  - 高速，等待打乱程序员思路

JUnit简介

- JUnit 是 Java 社区中知名度最高的单元测试工具。由 Erich Gamma 和 Kent Beck 共同开发完成
- 开源软件
- 支持语言
  - Smalltalk, Java, C++, Perl 等等
- 支持的IDE
  - JBuilder, VisualAge ,Eclipse等

JUnit的好处

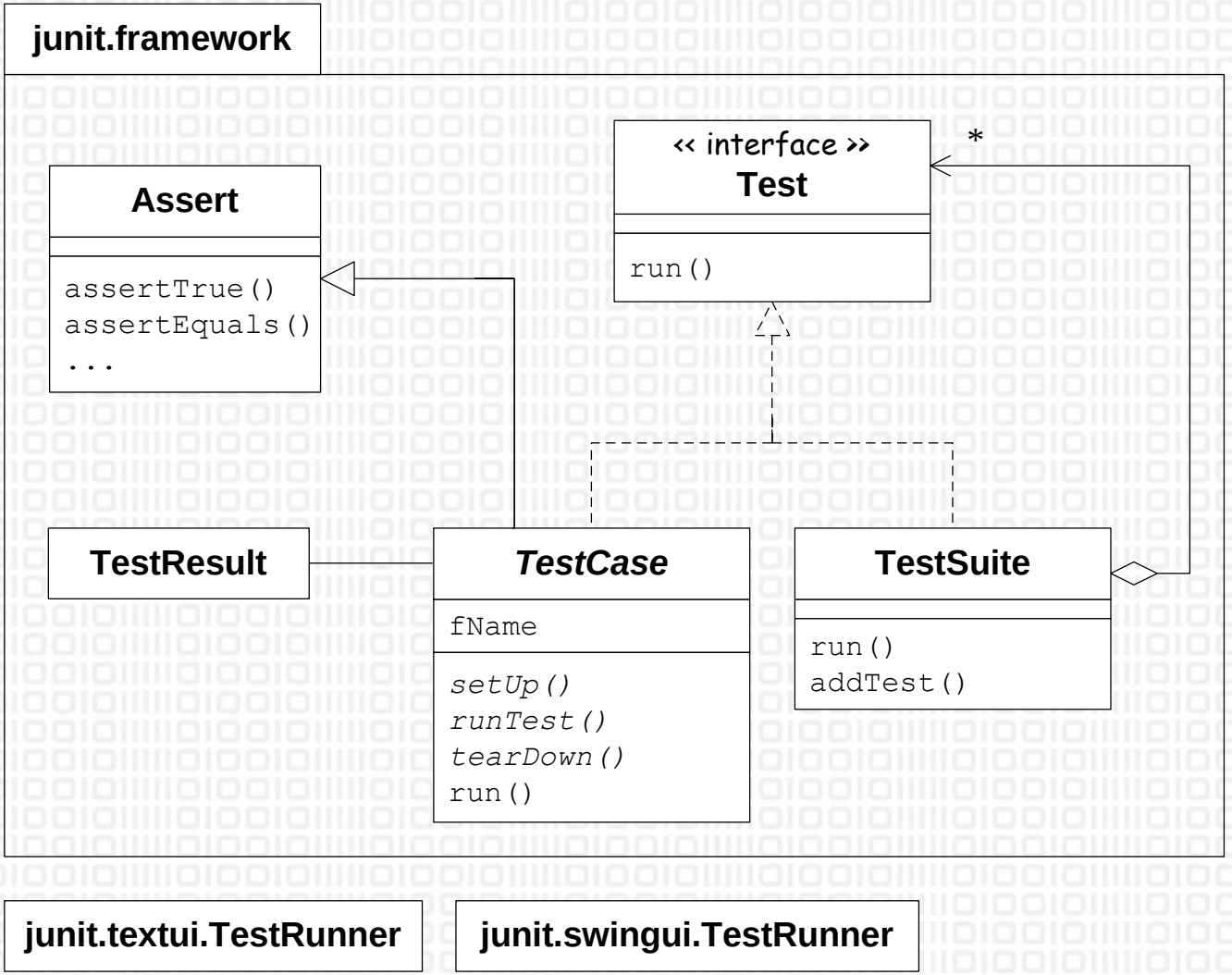
- 开源→实际项目中的应用示例、扩展JUnit功能
- 可以将测试代码和产品代码分开
- 测试代码编写容易，功能强大
- 自动检验、立即反馈
- 与开发紧密结合
- 测试包便于组织和集成运行，与IDE完美结合

JUnit 安装

- Java的JUnit可从网上免费下载 http://junit.org
- 将下载的junit.zip解压到你指定的目录
- 设置环境变量
  - Variable:CLASSPATH
  - Variable Value: .;Install Path/junit.jar
- 测试运行（进入命令提示符安装目录下）
  - java junit.swingui(textui,awtui).TestRunner junit.samples.AllTests


NANJING UNIVERSITY · SOFTWARE INSTITUTE
南京大学 软件学院
Computing
And
Software Engineering

JUnit 框架


junit.framework
<< interface >>
Test
run()
*
Assert
assertTrue()
assertEquals()
...
TestResult
TestCase
fName
setUp()
runTest()
tearDown()
run()
TestSuite
run()
addTest()
junit.textui.TestRunner
junit.swingui.TestRunner

JUnit核心类及接口(1)

类/接口	责任
Assert	当条件成立时 assert 方法保持沉默，但若条件不成立就抛出异常。
TestResult	TestResult 包含了测试中发生的所有错误或者失败。
Test	可以运行 Test 并把结果传递给 TestResult。
TestListenner	测试中若产生事件（开始、结束、错误、失败）会通知 TestListenner。
TestCase	TestCase 定义了可以用于运行多项测试的环境（或者说固定设备）。
TestSuite	TestSuite 运行一组 test case（它们可能包含其他 test suite），它是 Test 的组合。
BaseTestRunner	test runner 是用来启动测试的用户界面，BaseTestRunner 是所有 test runner 的超类。

JUnit核心类及接口(2)

- TestCase（测试用例）
  - 把具有公共行为的测试归入一组
  - 扩展了JUnit的TestCase类的类。它以testXXX方法的形式包含一个或多个测试用例
  - 典型的TestCase包含两个主要部件
    - fixture
    - 测试用例

JUnit核心类及接口(3)

- TestCase（测试用例）
  - Fixture
    - 管理资源，复用配置代码
    - 运行一个或多个测试所需的公用资源或者数据集合
    - TestCase通过setUp和tearDown方法来创建和销毁fixture
    - 典型应用数据库连接，生成输入文件
  - 测试用例
    - 继承自Assert

南京大学 NANJING UNIVERSITY · SOFTWARE INSTITUTE 软件学院

Computing And Software Engineering

Assert超类所提供的8个核心方法

方法	描述
assertTrue	断言条件为真。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertFalse	断言条件为假。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertEquals	断言两个对象相等。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNotNull	断言对象不为 null。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNull	断言对象为 null。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertSame	断言两个引用指向同一个对象。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNotSame	断言两个引用指向不同的对象。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
fail	让测试失败，并给出制定信息

JUnit核心类及接口(4)

- TestSuite（测试集合）
  - test suite是把多个相关测试归入一组便捷方式，TestRunner的runSuite()方法来执行
  - 若没有提供TestSuite，TestRunner会自动创建一个，该缺省的TestSuite将根据反射机制将测试类中的所有以testXxxxx命名的测试都加入到该TestSuite中
  - 组合模式

JUnit核心类及接口(6)

- TestSuite（测试集合）
  - 通常情况下使用仅仅包括一个静态的suite方法的TestAll类来注册应用程序需要定期执行的所有Test对象（包括TestCase对象和TestSuite对象），下面是一个典型的TestAll类

```
public class TestAll{
    public static Test suite(){
     TestSuite suite = new TestSuite("All tests frome part 1");
     suite.addTestSuite(TestCalculator.class);
     return suite;
    }

}
```

JUnit核心类及接口(7)

- TestResult
  - 所有的TestSuite都有一个对应的TestResult
  - 负责收集TestCase的执行结果。储存了所有测试的详细情况，是通过还是失败。失败则会创建一个TestFailure对象
  - TestRunner使用TestResult来报告测试结果。没有TestFailure对象进度条就用绿色，否则进度条用红色并输出失败测试的数目

JUnit核心类及接口(8)

- TestResult
  - JUnit区分失败和错误
    - 失败：
      - 包含失败的断言
    - 错误：
      - 低级别不可恢复的问题
      - 程序抛出的预期之外的异常，也被计入失败
    - 修复错误，从新执行，说不定失败也没了

JUnit单元测试的步骤

1. TestRunner+TestSuite/TestCase->Result
2. 重载setUp()，封装测试环境初始化及测试数据准备(v4.0 @Before)
3. 设计测试方法，以testXXX命名(v4.0 @Test)
4. 设计测试套件，或使用缺省的测试套件，调用TestRunner执行测试脚本，生成测试结果
5. 重载tearDown() 执行收尾动作(v4.0 @After)

南京大学 NANJING UNIVERSITY · SOFTWARE INSTITUTE 软件学院

Computing And Software Engineering

JUnit与Eclipse集成

JUnit方法

- 测试Equals方法
  - 领域对象总是实现equals方法方便测试
  - 构造对象比较、属性、toString()
- 测试没有返回值的方法
- 测试构造函数
- 总体原则：自认为有可能出错时进行测试

JUnit方法

- 测试异常

```
private void testPopEmptyStackException(Stack<Element> stack) {
    try {
            stack.pop();
            fail("Pop empty stack should cause an error!");
    } catch(EmptyStackException e) {
            assertEquals("Empty Stack!", e.getMessage());
    }
}
```

```
@Test (expected=kEmptyStacException.class)
private void testPopEmptyStackException…
```

JUnit方法

- 断言模式
  - 结果状态验证
  - 防卫断言
    - 功能验证前对夹具做出的假设
  - 交互断言
    - 对象协作行为的正确性

JUnit

- 夹具
  - 夹具是整个运行时状态（包括而不仅仅包括测试类的成员变量值）
  - 消除重复
    - 反模式——光板夹具
      - 各个测试方法的初始化过程毫无共性
  - 使测试更紧凑
    - 测试代码精炼、直指要害

JUnit方法

- 夹具模式
  - 参数化创建方法
    - 重复创建实体对象，填充不重要属性同时保证合法性
  - 对象母亲
    - 不同类的创建方法重复——单独的类作为创建方法的聚合体
  - 自动清理
    - 数据库、文件
    - 夹具创建对象时添加到对象注册表

JUnit方法

- 真正独立的测试用例
- 多次独立执行的测试需要与测试外围的执行环境无关（不依赖外围数据，不对外围产生影响）
- 内联数据文件

```
String configFile = “<?xml?><>…</>”;
Reader reader = new InputStreamReader(new
        ByteArrayInputStream(configFile.getBytes()));
```

- 测试专用数据库、+文件数据（DBUnit）

一些好的实践

- 测试命名
  - testMethodHowhow…
- 提供参数显示错误时的信息
- 测试的是行为而不是方法
  - 对象是一组相互紧密关联的方法作用在同一组数据上
  - 有些方法只是参与到特定功能中
  - 平衡测试覆盖度和重构需要的自由度

理解测试驱动开发

- 只为修复失败了的测试而写代码。
- 测试——编码——重构
- 完善测试列表
- 尽快使测试变绿
- 小步重构至设计足够优化
- 重构测试