目 录

[第一讲 1](#_Toc438304028)

[1.1 白盒测试 1](#_Toc438304029)

[1.1.1 白盒测试含义 1](#_Toc438304030)

[1.1.2 白盒测试与单元测试的区别 1](#_Toc438304031)

[1.2 单元测试 1](#_Toc438304032)

[1.2.1 单元测试的作用 1](#_Toc438304033)

[1.2.2 单元测试分类 2](#_Toc438304034)

[1.3 PMD 2](#_Toc438304035)

[1.3.1 PMD实现原理 2](#_Toc438304036)

[1.4 白盒测试六大法 3](#_Toc438304037)

[1.5 白盒测试针对的内容点 3](#_Toc438304038)

[第二讲 4](#_Toc438304039)

[2.1 语句覆盖 4](#_Toc438304040)

[2.2 判定覆盖 4](#_Toc438304041)

[2.3 条件覆盖 4](#_Toc438304042)

[2.4条件-分支覆盖 4](#_Toc438304043)

[2.5 条件组合覆盖 4](#_Toc438304044)

[2.6 路径覆盖 5](#_Toc438304045)

[2.7 控制流图 5](#_Toc438304046)

[2.8 复杂度 5](#_Toc438304047)

[2.9 作业 5](#_Toc438304048)

[第三讲 6](#_Toc438304049)

[3.1 Junit框架 6](#_Toc438304050)

[3.1.1 Junit3和Junit4版本 6](#_Toc438304051)

[3.1.2 使用Junit进行测试的最佳实践 7](#_Toc438304052)

[第四讲 8](#_Toc438304053)

[4.1 扩展 8](#_Toc438304054)

[4.2 测试套件（TestSuite） 8](#_Toc438304055)

[4.3 极限编程 9](#_Toc438304056)

[4.4 单元测试中的集成测试 9](#_Toc438304057)

[4.4.1 桩模块和驱动模块 9](#_Toc438304058)

[4.4.2 集成测试分类 9](#_Toc438304059)

[第五讲 10](#_Toc438304060)

[5.1 Junit4 10](#_Toc438304061)

[5.1.1 Junit4与Junit3的区别 10](#_Toc438304062)

[5.2 作业 11](#_Toc438304063)

# 第一讲

## 白盒测试

White Box Test，WBT。又称为逻辑测试、结构化测试、透明的盒子测试。

### 1.1.1 白盒测试含义

将软件看做一个盒子，只考虑内部逻辑结构的实现，不考虑外部功能的实现。

### 1.1.2 白盒测试与单元测试的区别

1. 按含义：白盒测试测试的是整个软件的代码，单元测试测试程序独立模块的代码
2. 按类型：白盒测试是按结构分的，单元测试是按阶段分的
3. 按方法：白盒测试有白盒测试六大法，单元测试没有具体的方法，是参照白盒测试的六大法进行测试的

## 单元测试

以一小段代码为代码块（在一个模块中分析出对应的段的代码块，主要具体到某一个类中的某一个方法甚至某一个属性的测试）

### 1.2.1 单元测试的作用

在测试中最先接触的是单元测试，可以集中发现错误（约80%），减少后期维护费用，节省人力物力时间资源

### 单元测试分类

单元测试分为静态测试、动态测试

1. 静态测试：采用浏览的形式检查代码是否规范（行业规范、公司规范）。其作用是提高程序代码质量、增强易读性，有助于优化程序代码。

静态测试的规则：

1. 查看程序中的警告信息，确保程序中不存在无意义的变量，并且定义变量的规则符合java变量定义的规范
2. 确保不要存在空的方法、空语句块（包括try-catch）（降低代码的执行率，增大阅读难度）
3. 是否使用了多重if嵌套，或者无意义的if判定
4. 是否使用了死循环（真死），不能使用过多的嵌套循环
5. 能用while循环的代码尽量不要使用for循环（for循环的执行点比while循环的多）
6. 当程序代码中存在从其他途径（数据库、网络等）获取资源的情况的话必须在使用链接完成后将其关闭（例如finally）
7. 不能在循环中创建对象
8. 不要写重复性代码（降低了代码的执行效率），尽量使用继承、多态
9. 采用静态测试的时候可能发现潜在bug（例如创建空对象，通过空对象调用属性和方法（空指针异常）；对象的引用错误）
10. 动态测试：执行程序代码产生的实际结果与预期结果是否一致

## PMD

静态代码分析工具：不需要采用人工浏览，自动的静态代码分析工具，大大提高了代码阅读效率

### 1.3.1 PMD实现原理

主要是依赖javacc解析生成器进行实现，其中主要是参照扩展巴克斯诺尔规则（Extended Backus Naur Formal，EBNF规则）语法进行检查，通过该语法进行编译从而得到Jtree，该语法树主要是实现了AST（abstract Syntax Tree），又称为抽象语法树。换句话说将源码文件通过一定的规则进行解析生成得到一个合法的AST的xml文件

Xml（eXtended Markup Language）可扩展标记语言，以元素的形式成对存在，实现程序代码间的数据交换

## 1.4 白盒测试六大法

1. 语句覆盖
2. 判定覆盖（分支覆盖）
3. 条件覆盖
4. 条件分支覆盖
5. 条件组合覆盖
6. 路径覆盖

白盒测试核心：逻辑

程序代码

## 1.5 白盒测试针对的内容点

1. 流程控制语句、循环语句中的语句块至少执行一遍
2. 针对流程控制语句中的bool表达式的真值假值至少执行一遍（边界值法）
3. 设计一条用例尽可能覆盖多条路径
4. 要保证白盒测试中的结构数据的一致性
5. 要有对应的白盒测试流程，大致跟黑盒测试一样，但是内容不同

不同类型的测试都应该有对应的测试计划、测试方案

# 第二讲

## 2.1 语句覆盖

定义：在程序代码中可能执行的语句至少执行一遍

语句覆盖是白盒测试中最基本的覆盖，需要进行设计尽可能多的用例去进行覆盖，是六大法中设计测试用例覆盖能力最弱的

程序流程图：通过图形符号等形式将程序进行表示出来，更为直观、更容易分析出程序的关系。使用矩形表示语句块，菱形表示判定，圆形或椭圆表示程序入口或出口，语句块与条件判定之间的连接使用有向边。

## 2.2 判定覆盖

又称为分支覆盖，指在应用程序代码中，针对流程控制语句中的每一个真值和假值都必须至少执行一遍

## 2.3 条件覆盖

在应用程序代码中，针对流程控制语句中的每个判定中的每个bool表达式的真值和假值都至少执行一遍

## 2.4条件-分支覆盖

在应用程序代码中，流程控制语句中的判定分支的真值和假值至少执行一遍，且其中的判定分支中的每个bool表达式的真值和假值至少执行一遍

## 2.5 条件组合覆盖

在应用程序代码中，针对流程控制语句中的判定条件的bool表达式的真值和假值的组合，每一次组合至少执行一遍。

总结：

1. 语句覆盖覆盖能力最弱
2. 条件覆盖、判定覆盖的覆盖能力比语句覆盖强
3. 满足条件-判定覆盖则一定满足条件覆盖和判定覆盖。
4. 条件组合覆盖是除路径覆盖之外覆盖能力最强的

## 2.6 路径覆盖

很少用于设计测试用例，通常用于验证所设计的测试用例的覆盖面

## 2.7 控制流图

控制流图是用于计算复杂度的一个重要方法。使用圆圈进行表示所执行的代码对应的行数，用有向边进行连接。

## 2.8 复杂度

用于进行衡量一段代码的质量的度量，程序中的复杂度至少要小于10，复杂度过高导致效率降低，导致性能下降

复度杂用V(G)表示，计算方法（三种计算方式等效）：

1. V(G)=有效区域数

有效区域数即封闭区域数+外围的大区域（1）

1. V(G)=E-N+2 其中E表示有向边，N表示节点数

首尾的有向边不计入E，多条边汇合结束的时候要用汇合边数减去1

1. V(G)=P+1 其中P表示二分支节点

二分支在对应分支数递增的情况下节点数也递增

## 2.9 作业

1. 画出三角形的流程图，使用五大法设计测试用例并进行筛选
2. 使用三种方法计算复杂度

# 第三讲

## 3.1 Junit框架

已经声明定义好的单元测试自动化框架，通过它可以直接进行单元测试。

衍生出Xunit（X表示任意语言）、Nunit等

### Junit3和Junit4版本

部署Junit框架：网上下载资源包、使用myeclipse自带的

1. Junit3版本使用注意点
2. 所有的测试类都必须继承于TestCase类（导包：import junit.framework.TestCase）
3. 在测试类中的方法必须是“public”类型的
4. 测试类中的测试方法必须是无返回值类型
5. 测试类中的测试方法的方法名不可以随意声明定义，必须是test+方法名（通过反射进行获取判断是不是测试方法）
6. 测试方法不能传参
7. errors表示测试代码有问题，failures表示源码有问题，runs绿色表示通过
8. 将一些需要进行初始化的对象的设置放在setUp方法中，关闭放在tearDown方法中（先运行setUp方法，再执行测试方法，再执行tearDown方法）
9. 测试代码中所有的异常都要进行捕获
10. fail方法放在不想执行的地方（常与异常一起使用）

断言（Assert）：用于进行判定运行的实际结果与预期结果是否一致的类

在类中不管创建多少个对象，都只有一个class对象（类名.class、通过对象调用getClass()方法、数字签名forName(包名.类名)）

1. Junit4

与Junit3相比多了注解（能被java虚拟机执行的，相当于元数据，即用数据去描述数据（对数据的描述和对数据的操纵）的过程）

### 3.1.2 使用Junit进行测试的最佳实践

1. 必须建立测试代码的资源文件夹（用于装测试代码，以便于测试代码与源码的区分）
2. 将资源文件夹中的结构设置成与源码中的结构一样
3. 为了使得源码和测试类更为明显区分，测试类的命名使用源码类名+test或test+源码类名

# 第四讲

## 4.1 扩展

1. 如果源码中存在私有方法的话如何测试？

目前只能使用共有方法调用私有方法

1. 如果源码中的方法为无返回值的话如何测试？

采用浏览的方式，创建对象调用该方法查看控制台输出信息是否正确。

一个类中所有属性、方法都必须测试

awt 抽象窗口工具包

## 4.2 测试套件（TestSuite）

测试套件是一组测试的集合（单元测试的自动化测试）

例子：

public static Test suite(){

TestSuite suite=new TestSuite(MaxArrayTest.class);

//suite.addTestSuite(MaxArrayTest.class);

//suite.addTest(new MaxArrayTest("testGetMax"));

//suite.addTest(new LianxiTest("testAdd"));

//实现重复性测试

suite.addTest(new RepeatedTest(new LianxiTest("testAdd"),10));

return suite;

}

## 4.3 极限编程

eXtreme Programming，XP

通过测试代码框架逆推出源码框架

《极限编程思想》

## 4.4 单元测试中的集成测试

### 4.4.1 桩模块和驱动模块

1. 桩模块：被测试模块所调用的
2. 驱动模块：大多数都存在于main方法中，主要用于调用分支模块

设计桩模块和驱动模块时要注意传入、传出参数

### 4.4.2 集成测试分类

包括渐增式集成和非渐增式集成。

1. 渐增式集成：先进行N个模块的开发与测试，再进行N+1个模块的测试并集成，分为自顶向下（设计桩模块）和自底向上（设计驱动模块）
2. 非渐增式集成：在开发过程中可能既需要设计桩模块也需要设计驱动模块
3. 渐增式集成和非渐增式集成的区别
4. 从设计上而言，渐增式集成更为方便
5. 在集成过程中出现问题的时候，渐增式集成更容易发现问题
6. 渐增式集成是将单元测试和集成测试融合了，而非渐增式集成将单元测试和集成测试分开的
7. 非渐增式集成实现的是并行开发，而渐增式集成是单线的
8. 非渐增式的代码执行效率更高

# 第五讲

## 5.1 Junit4

### 5.1.1 Junit4与Junit3的区别

1. Junit4有注解
2. Junit4兼容Junit3
3. Junit4中使用@Test标记方法（public 无返回值的方法），则该类不继承TestCase类也是一个测试类（测试类构造方法隐式调用TestCase）
4. Junit4中方法名无限制，但是定义时最好还是使用test开头，使得代码更加符合规范
5. @Test可以提供两个参数

* 测试异常类型的时候可以不捕获，直接往上抛到@Test(expected=异常类型.class)，也可以捕获处理。需要注意的是expected中捕获的异常类型与定义的异常类型一致或者是存在继承关系的父类，则判定异常类型都是正确的（如果要准确判定的话则不要写父类或者用try-catch的捕获方法）
* timeout=指定时间，用于优化代码，提升代码质量（以毫秒为单位）

1. Junit4中@Before、@After中的方法名没有限制，但是Junit3中的setUp、tearDown是不能变的
2. Junit4中@BeforeClass用于放置一些共享的昂贵的资源设置，如打开数据库，在测试方法运行之前执行一次，@AfterClass在测试方法运行完后执行
3. @ignore用于在设计代码时，测试代码框架已经完成而代码还没完成时注解不执行的方法，等完成了再进行测试（@ignore可以应用于类，整个类都不执行）
4. Junit3中的运行器（四个）：awtui、swingui、textui、Junit自带的运行器，Junit4中只有textui、Junit自带的运行器
5. 两个框架：Junit框架、java自带的框架
6. 参数化运行器

使用Parameters注解的方法可以通过Parameterized提供参数注入到一个测试类的构造方法

例子：

@Parameters

public static Collection data(){ //返回静态的集合

Object [][] obj={{1,4,5},{4,5,9}};

return Arrays.asList(obj);

}

注意：该类要用@RunWith(Parameterized.class)注解

1. 测试套件

要以套件形式运行多个类，不能以参数形式传入，传入的不只是class类型，还可以是与之对应的枚举类型（多个类.class用{}括起来）

在类前加：

@SuiteClasses({测试类名1.class,测试类名2.class,…})

@RunWith (Parameterized.class)

扩展：

线程：Thread

中断：Thread.sleep();

Integer：int的包装类

## 5.2 作业

用Junit4参数化测试数据（三组数据）、异常