集成与测试

集成

- 软件集成（Integration）指将单独的软件构件合并成一个整体的软件开发活动。
- 个人集成：相对简单
- 团队集成：相对复杂

理想集成过程

- 自动化理想集成过程:
  - 任何人任何时刻都应该可以从一个干净的计算机上检出当前源代码快照，然后敲入一条命令（或点击一个按钮），就可以得到能在这台机器上运行的软件系统。

- 具体步骤（持续集成）
  - 从版本控制服务器签出当前最新的代码和所有的相关文件。
  - 使用自动化构建工具进行构建活动。
    - 为什么不依赖IDE?
    - 可靠性更高、取消IDE依赖
- 签入新的代码。

常见集成工具与构建

- Unix社区使用make; Java社区使用Ant, Maven; .Net社区使用MSBuild等。
  - 从头编译所有源代码。
  - 链接和部署。
  - 启动自动化测试集。
  - （如果以上步骤在执行过程中没有出现任何问题，则是一次成功的构建。）

Ant

- Apache Ant是Apache Software Foundation的一个自动化软件构建工具，它是一个Java类库，同时也是一个命令行工具，可以按照指定的一个用户编写的xml文件执行一些指定的任务。
- Ant通过一个配置脚本，称为构建文件（Buildfiles）来进行构建任务，该配置文件是一个xml文件。

Ant元素

- Ant脚本中包含项目（Projects）、目标（Targets）和任务（Tasks）。
- 每个构建文件包含一个项目元素；一个项目元素包含多个目标元素；每个目标元素由一组任务元素构成。
- 一个任务完成一个功能，例如拷贝一个文件、编译一个项目或者构建一个JAR文件。
- 一个目标是一组任务和属性的集合。一个目标可以依赖于其它目标。

例

```
public class HelloWorld{
   public static void main(String []args){
    System.out.println("Hello World!");
   }
}
```

```
<project name="hello" default="compile">

<target name="prepare">
<mkdirdir="/tmp/classes"/>
</target>
  <target name="compile" depends="prepare">
<javacsrcdir="./src" destdir="/tmp/classes"/>
</target>

</project>
```

以上Ant文件可以将当前目录下子目录“src”中的java源文件编译到“/tmp/classes”目录下。

输出

```
$ ant
Buildfile: build.xml
prepare:
[mkdir] Created dir: /tmp/classes
compile:
[javac] Compiling 1 source file to /tmp/classes
BUILD SUCCESSFUL
```

详细Ant说明

- 请参考教材

集成频率

- 间隔越短越好。
- daily build->continuous integration
- Why?
  - 快速的集成使得程序员可以尽快的发现软件中的bug，在团队开发时尤其有助于发现不同人员间的冲突。
  - bug会叠加、交错
  - 验证用户需求
  - 必须自动化!

测试分类

- 从不同的角度看问题!
- 软件开发传统上可以分为白盒测试（White Box Testing，或结构测试Structural Testing）和黑盒测试（Black Box Testing，或功能测试Functional Testing）。

- 白盒测试指当测试者知道程序的内部数据结构和算法，并且能够获取其具体源码时进行的测试。白盒测试检查程序逻辑，确定测试用例，覆盖尽可能多的代码和逻辑组合。

- 黑盒测试认为软件是一个“黑匣子”，测试人员完全不知道其内部实现。测试工程师根据需求规格说明书确定测试用例，测试软件的功能，而不需要了解程序的内部结构。

- 根据在软件开发过程中测试实施的对象不同可以分为单元测试（Unit Testing）、集成测试（Integration Testing）和系统测试（System Testing）

- 单元测试通常由程序员在编写代码时进行，用于测试某段代码的功能，通常在函数或类的级别进行。

- 集成测试用来在程序集成时测试有交互的程序模块之间的接口和交互是否正确。软件模块的集成可以使用迭代方式进行或完成所有模块再进行，通常认为迭代方式集成更为合理。

- 系统测试关注整个系统的行为，用于测试已经集成的系统是否符合其需求规格，也有人认为系统测试适合于评价系统的非功能性需求，如安全性、速度、可靠性等。

单元测试

- 单元测试是在开发过程中由程序员进行的一种测试，它主要测试程序模块的正确性。
- 敏捷软件开发实践--测试驱动开发(Test driven development, TDD)。
- 自动化单元测试。

互不关联

- 测试用例不允许调用其它测试代码。
- 否则会产生为测试代码再写“测试”的荒谬问题。

单元测试覆盖

- 最简单的白盒测试是语句覆盖（Statement Coverage），即设计一系列的测试用例，使得程序中所有的语句都会得到执行。但这样的测试不能保证测试到所有的分支。
- 改进的方法是分支覆盖（Branch Coverage），即设计一系列的测试用例，保证所有的分支都得到测试。
- 最复杂的语句测试是路径覆盖（Path Coverage），即设计一系列的测试用例，保证覆盖程序中所有的语句或它们的所有可能组合。（图11.3，11.4）

自动化单元测试工具（JUnit）

- JUnit（http://www.junit.org/）是一个开源自动化单元测试框架。单元测试是用于验证代码行为是否符合预期需求的有效手段，JUnit框架可以辅助程序员进行单元测试。
- 如果没有测试框架，大量单元测试的编写和执行会有很多工作量！

JUnit 3

- 1. 新建类，继承junit.framework.TestCase类；
- 2. 定义需要的测试方法，这些方法的名字以test开头，例如testAdd(),testPut()等，返回值为void；
- 3. 如果需要组合测试用例，可以定义teat suite。

简单例子

```
public class Catalog{
    …
    publicMessageaddBookInfo(String isbn, String title, String author, String publisher,
                    int year, booleanisRare){…}  //需要测试的方法
    …
}
```

```
importData.BookInfo;
importData.Catalog;
importjunit.framework.TestCase;
public class CatalogTest extends TestCase {
public void testAddBookInfo () {
        BookInfobooka = new BookInfo("123456", "Head First Java", "小明", "清华出版社",
                        2010, true);
        Catalog testCatalog = new Catalog();
          testCatalog.addBookInfo("123456", "Head First Java", "小明", "清华出版社", 2010,
                    true);
    assertTrue(booka.getBookInfoRecord().equals(testCatalog.searchBookInfo("123456").get
    BookInfoRecord()));
        }
}
```

断言

- ① assertTrue/False ([String message,] boolean condition);
- ②fail ([String message]);
- ③assertEquals([String message,]Object expected, Object actual);
- ④ assertNotNull/Null ([String message,] Objectobj);
- ⑤ assertSame/NotSame([String message,]Object expected, Object actual);
- ⑥ failNotSame/failNotEquals(String message, Object expected, Object actual)

骨架（fixture）

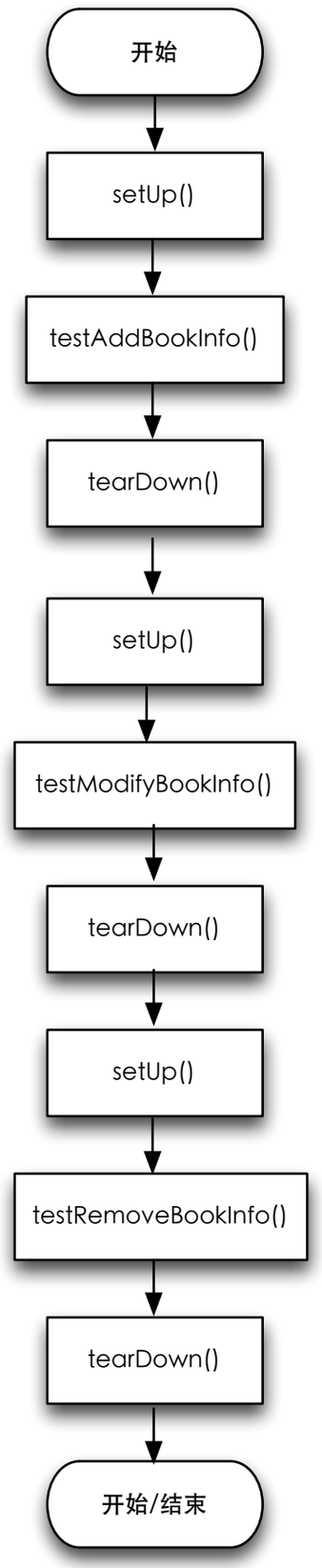
- 骨架是指测试中需要被反复运行的部分。
  - ①新建包含骨架的测试类，继承自TestCase类；
  - ②在测试类中定义需要使用的类成员变量；
  - ③重写setUp()方法，该函数在每个测试方法调用之前被调用，通常该方法负责初始化各个测试方法所需要的测试环境，用以实例化2中定义的变量；
  - ④可能会重写tearDown()方法，该函数在每个测试方法调用之后被调用，用以释放在变量资源
  - ⑤根据需要，完成其它使用骨架的测试方法。

例子

```
public class CatalogTestextends TestCase{
   privateBookInfobooka;
   privateBookInfobookb;
   privateBookInfobookc;
   private static Catalog testCatalog = new Catalog();

   publicCatalogTest(String method) {
            super(method);
}

protected void setUp(){
     //初始化
      booka = new BookInfo("123456", "Head First Java", "小明", "清华出版社", 2010, true);
          bookb = new BookInfo("234567", "Head First C++", "小力", "清华出版社", 2010, true);
          bookc = new BookInfo("345678", "Head First C#", "小白", "清华出版社", 2010, true);
     }
     protected void teardown(){
       //回收资源
     }
     //测试addBookInfo()方法
     public void testAddBookInfo ()
     {
                ...
     }
     //测试modifyBookInfo()方法
     publicvoidtestModifyBookInfo(){
          ...
     }
     //测试removeBookInfo()方法
     publicvoidtestRemoveBookInfo(){
          ...
          }
}
```


开始
setUp()
testAddBookInfo()
tearDown()
setUp()
testModifyBookInfo()
tearDown()
setUp()
testRemoveBookInfo()
tearDown()
开始/结束

套件（Suite）

- TestSuite是JUnit提供的一个用于批量运行测试用例的对象，是test的一种有效的组合方式。
- 一个测试： TestResult result = (new CatalogTest("testAddBookInfo")).run();

- 多个测试用例:
- TestSuite suite = new TestSuite();
- suite.addTest(new CatalogTest("testAddBookInfo"));
- suite.addTest(new CatalogTest("testModifyBookInfo"));
- TestResult result = suite.run();

- 自动提取套件
- 将测试用例类的类名作为套件构造函数的参数

- TestSuite suite = new TestSuite(CatalogTest.class);

- TestResult result = suite.run();

```
importjunit.framework.Test;
importjunit.framework.TestSuite;
public class TestAll{
  public static Test suite(){
    TestSuite suite=new TestSuite();
    suite.addTestSuite(CatalogTest.class);
    return suite;
  }
  public static void main(String args[]) {
    junit.textui.TestRunner.run(suite());
  }
}
```

JUnit 4

- 使用Annotation技术（请参考相应技术文档）

集成测试

- 集成测试（Integration Testing），也叫组装测试或联合测试，是在完成单元测试后，将单独模块组合成为子系统或系统时进行的测试，在系统测试前进行。
- 一次性测试（Big-bang Testing），自底向上测试（Bottom-up Testing），自顶向下测试（Top-down Testing）
- 持续集成，包含测试。

系统测试

- 系统测试关注整个系统的行为，在完整的系统上进行，测试其是否符合系统需求规格说明书。
- 可用性测试（Usability Testing）
- 验收测试（acceptance testing）
- 猴子测试（monkey testing）