

分类号：
密 级：

单位代码：10019
学 号：S02746

中国农业大学

学位论文

基于组合框架的 J2EE 组件单元测试的
研究与应用

**Research and Application of Unit Testing for J2EE Components
Based on Combined Frameworks**

研 究 生： 史明慧

指 导 教 师： 周南 副教授

申请学位门类级别： 工学硕士

专 业 名 称： 计算机应用技术

研 究 方 向： 计算机网络技术应用

所 在 学 院： 信息与电气工程学院

二零零五 年 六月

独 创 性 声 明

本人声明所呈交的论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已发表或撰写过的研究成果，也不包含为获得中国农业大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本课题所做的任何贡献均在论文中作了明确的说明，并表示了感谢。

研究生签名：

时间：

年 月 日

关于论文使用授权的说明

本人完全了解中国农业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件和磁盘，允许论文被查阅和借阅，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。同意中国农业大学可以用不同方式在不同媒体上发表、传播学位论文的全部或部分内容。

研究生签名：

时间：

年 月 日

导师签名：

时间：

年 月 日

摘要

本文首先在研究面向对象软件开发与测试的现状以及组件测试的现状的基础上，指出了传统的测试技术已无法对面向对象软件进行有效的测试，分析了 J2EE 组件与容器交互的特点，提出了 J2EE 组件单元测试面临的问题和难点。然后，全面介绍和分析了面向对象软件测试的理论和技術以及 J2EE 组件的内容和特点。本文通过对组合框架，即 JUnit 单元测试框架及其两个扩展框架 Mock Objects 和 Cactus 深入细致的研究，比较得出了三个单元测试框架的优缺点，综合性地提出了 J2EE 组件单元测试的实施方案。最后，给出了该方案在“基于网络的物流配送管理系统平台”中的应用。

本文的内容融合了软件测试理论知识与笔者的实践经验，相信本论文的研究对于更快速有效地实施 J2EE 组件单元测试具有一定的理论意义和参考价值。

关键词：单元测试，J2EE 组件，JUnit，Mock Objects,Cactus

Abstract

In this paper, the problem and difficulty about the unit tests of J2EE components were put forward, based on the studies of the theory and technic for Objected-Oriented software testing and the character of interactant between J2EE components and their containers. Through the careful analysis and comparison of advantages and disadvantages about the combined unit test frameworks, JUnit and its extension frameworks, Mock Objects and Cactus, an integrated scheme to implement the unit test of J2EE components, which had absorbed the merits of those three frameworks, was given. Finally, put the scheme into practical project, that is, “A Management System Platform of Logistic and Dispatching based on the network” and gave several typical testing cases as illustration.

Combine the theories of software testing and the author's practices, the paper would be meaningful and can provide a useful reference for implementing J2EE components unit test more quickly and efficiently.

Key words: unit test, J2EE components, JUnit, Mock Objects, Cactus

目 录

第一章 绪论	1
1.1 研究背景	1
1.2 问题的提出	3
1.3 论文研究内容和研究思路	4
第二章 面向对象软件测试的理论与技术	7
2.1 软件测试的基本理论与技术	7
2.2 面向对象软件测试的理论	14
2.3 面向对象软件的单元测试技术	22
2.4 本章小结	24
第三章 J2EE 组件技术	25
3.1 J2EE 架构技术	25
3.2 J2EE 组件	26
3.3 本章小结	28
第四章 JUnit 单元测试框架及其扩展框架	29
4.1 JUnit 单元测试框架	29
4.2 Mock Objects 测试框架	36
4.3 Cactus 测试框架	37
4.4 本章小结	41
第五章 J2EE 组件单元测试的方案与实施	42
5.1 测试方案分析与确定	42
5.2 方案的实施	44
5.3 本章小结	58
第六章 总结与展望	59
6.1 总结	59

6.2 展望	59
参考文献	61
附录	64
致谢	70
作者简介	71

图表目录

图 1-1 面向对象测试模型	2
图 1-2 J2EE 组件/容器模型	4
图 1-3 论文研究思路	6
图 2-1 软件测试技术的分类	8
图 2-2 传统软件开发过程与软件测试	10
图 2-3 软件测试与增量开发过程	11
图 2-4 软件测试与迭代开发过程	11
图 2-5 软件测试的过程	12
图 2-6 单元测试环境	14
图 2-7 面向对象软件的测试层次	20
图 2-8 两种不同的测试模型	20
图 2-9 应用程序内部的单元测试	21
图 3-1 J2EE 多层结构框架图	25
图 3-2 J2EE 组件分布示意图	26
图 4 -1 目前支持 JUnit 的 Java IDE	30
图 4 -2 JUnit 的系统架构图	30
图 4 -3 JUnit 中的模式	31
图 4-4 测试序列图	33
图 4-6 Mock Objects 实现隔离测试的技术框架示意图	36
图 4-7 Cactus 的测试生命周期	38
图 4-8 各种运行 Cactus 的 runner	41
图 5-1 Cactus 与 Mock Objects 的比较	43
图 5-2 Cactus 和 Mock Objects 的不同适用情况示意图	44
图 5-3 系统软件体系架构	45
图 5-4 系统实现的 J2EE 框架	46
图 5-5 执行 TestSampleServlet TestCase 的结果	53
图 5-6 Cactus 测试的目录结构	56
图 5-7 用 Cactus/Ant 执行 Cactus 测试 EJB 的结果	58

第一章 绪论

1.1 研究背景

软件测试是软件工程方法中保证软件质量的重要手段。软件测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件中的错误。软件测试是保证软件质量的关键步骤，是对软件规格说明、设计和编码的最后复审。

在软件业较发达的国家，软件测试不仅早已成为软件开发的一个有机组成部分，而且在整个软件开发的系统工程中占据着相当大的比重。以美国的软件开发和生产的平均资金投入为例，通常是：“需求分析”和“规划确定”各占百分之三，“设计”占百分之五，“编程”占百分之七，“测试”占百分之十五，“投产和维护”占百分之六十七^[1]。测试在软件开发中的地位，由此可见一斑。

与国外相比，国内的软件测试业尚处于萌芽状态。从市场角度来看，中国的软件开发公司比比皆是，但软件测试公司则如凤毛麟角。2003 年举办了“首届中国软件测试与软件产业发展战略研讨会”，随后关于软件测试的国家标准也相继出台，软件测试在国内日益被重视。

1.1.1 面向对象软件开发与测试的研究现状

面向对象技术在软件工程中的推广，使得传统的测试技术和方法受到极大的冲击。面向对象技术所引入的新特点，尤其是面向对象方法所独有的多态、继承、封装等特点，使传统的测试技术已无法对面向对象软件进行有效的测试。针对面向对象软件的测试，其测试策略和测试方法都需要有相应的变革，必须考虑面向对象软件的独特特征^[2]。

面向对象软件的结构不再是传统的功能模块结构。面向对象软件作为一个整体，使得原有集成测试所要求的逐步将开发模块组装在一起进行测试的方法已成为不可能。面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（Object-Oriented Analysis，OOA），面向对象设计（Object-Oriented Design，OOD），面向对象编程（Object-Oriented Programming，OOP）和面向对象测试（Object-Oriented Tsting，OOT）四个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。针对这种开发模型，结合传统的软件测试步骤的划分，文献^[2]提出一种在整个软件开发过程中不断进行测试的面向对象的软件测试模型，使开发阶段的单元测试与编码完成后的单元测试、集成测试、系统测试成为一个整体。该测试模型给出了面向对象测试 OOT 与 OOA、OOD 和 OOP 三者的对应关系(如图 1-1 所示)。

面向对象分析的测试（Object-Oriented Analysis Test, OOA Test）和面向对象设计的测试（Object-Oriented Design Test, OOD Test）是对分析和设计的结果进行测试，即将测试扩充到分析和设计阶段。采用正式的复审技术对分析和设计产生的文本进行正确性验证，是软件开发前期的关键性测试。面向对象编程的测试（Object-Oriented Programming Test, OOP Test）忽略类功能的实现细则，将测试集中在类功能的实现和相应的面向对象程序风格，主要体现为两方面：（1）数据成员是否满足数据封装的要求；（2）类是否实现了系统分配的功能。

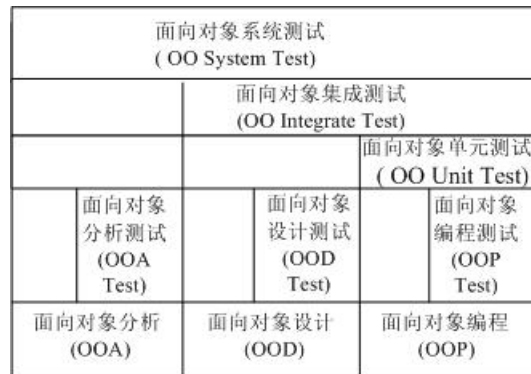


图 1-1 面向对象测试模型

从图 1-1 可以看出，面向对象单元测试（Object-Oriented Unit Test，OOUT）在 OOP Test 时进行，是针对面向对象软件设计的最小单位——程序模块（即类），进行正确性检验的测试工作。面向对象的编程特性使得对成员函数的测试与传统的函数或过程测试不同，需要考虑继承和多态特性。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试 OO Integrate Test 涵盖 OOD Test 和 OOP Test，主要对系统内部的相互服务进行测试。面向对象系统测试 OO System Test 是面向对象测试的最后阶段，主要以用户需求为测试标准，借鉴 OOA 或 OOA Test 结果，对系统的有效性进行测试。

面向对象单元测试是本论文研究的重点，它用于检测代码的开发是否符合详细设计的要求，其目的在于发现每个类内部可能存在的差错。加强单元测试同时也减轻了后续集成测试和系统测试的负担。根据业界的统计，一个 BUG 在单元测试阶段发现花费是 1 的话，到集成测试就变为 10，到系统测试就高达 100，到实际推向市场生产后就高达 1000^[3]年。当前，涌现出许多新的软件技术和方法把软件测试的角色加重，尤其是单元测试如 Agile（敏捷方法）、XP（Extreme Programming）、RUP（Rational United Process）等等。特别是一些单元测试框架的提出，使得单元测试自动化成为可能。但是，单元测试在目前国内软件企业中开展得并不好，一方面是由于对单元测试重视程度不够，测试投入不足，另一方面是由于在单元测试实践方面积累得也不够，单元测试处于一种摸索状态。

随着面向对象软件的广泛应用，面向对象测试的理论也在日益完善，并出现了许多相应的测试框架和工具。以 J2EE 组件单元测试这一方向为例，主要测试框架就有 JUnit、Mock Objects、Cactus 和 HttpUnit 等；分析覆盖率的工具有 Logiscope、TrueCoverage、PureCoverage 等；已经产品化的自动化单元测试工具有 Jtest、J2EE Tester、Rational Quality Architect、Rational

PurifyPlus 等等。这些测试框架和工具为全面和高效率地执行面向对象测试奠定了基础，也体现了面向对象软件测试技术有了很大的发展。

1.1.2 组件测试的研究现状

当前社会的信息化过程对软件的开发方法与生产能力提出了新的要求，软件复用是提高软件产品质量与生产效率的关键技术。组件技术的提出为软件复用提供了技术基础^[4]。

组件的高可靠性是组件能被成功复用的前提。组件测试是保障和提高组件的可靠性的重要手段。组件的开发者和复用者必须对组件进行充分地测试，以确保它在新的环境中工作正常。这里所指的组件测试是将面向对象软件系统分解成特定粒度单元的活动。组件本身就是一个程序单元，与传统的软件测试相比，组件测试有着自身固有的特点^[5]：（1）不能对组件的执行环境（即容器）和用户的使用模式进行完全准确的预测，故组件开发者不能完全、彻底地对组件进行测试，并且很难确定何时结束测试；（2）组件复用者和第三方测试人员通常无法得到组件的源代码及详细的设计知识，通常只能对组件进行黑盒测试，即调用组件的方法后，只能通过观察执行的结果判断组件的行为是否正确，无法检查执行过程中的组件的内部状态，使得组件执行过程中的一些故障被隐藏。这些特点对组件测试提出了严峻的挑战。

国际上于 20 世纪 90 年代后期对组件测试开展了研究^[6, 7]。近年来，出现了大量的文献报道^[5, 8, 9-18]。组件测试成为当前软件工程学术界和工业界的热点问题之一。大体上，对组件的测试可以从以下几个方面来进行分类。从组件测试的内容可分为：组件内部实现细节的测试^[8]，组件接口的测试^[6-11, 17, 18]，组件组装（构架）的测试^[10, 15]。从测试者与组件的关系可分为：组件开发者的测试（拥有组件的源代码）^[8, 9, 17]、组件复用者和第三方的测试（没有源代码）^[5, 6, 16]。从测试过程中所采用的技术手段可分为：基于变异测试的方法^[8, 17, 18]，基于组件状态机的方法^[14]，对组件的回归测试^[16]，以及组件的易测试性设计^[8, 12, 11, 12]等。

1.2 问题的提出

Sun 公司的 J2EE（Java 2 Enterprise Edition）平台是目前使用最广泛的面向对象的程序设计技术之一，它就是一个基于组件-容器模型的系统平台。其中容器是指为特定组件提供服务的一个标准化的运行时环境，Java 虚拟机就是一个典型的容器。组件是一个可以部署的程序单元，它以某种方式运行在容器中，如图 1-2 所示，不同的容器负责不同的组件的运行。容器封装了 J2EE 底层的 API，为组件提供事务处理、数据访问、安全性、持久性等服务。在 J2EE 中组件和组件之间并不直接访问，而是通过容器提供的协议和方法来相互调用。组件和容器间的关系通过“协议”来定义。容器的底层是 J2EE 服务器，它为容器提供 J2EE 中定义的各种服务和 API。一个 J2EE 服务器（也叫 J2EE 应用服务器）可以支持一种或多种容器。J2EE 平台使用这种基于 J2EE 组件的方法，来设计、开发、组装和部署企业应用。

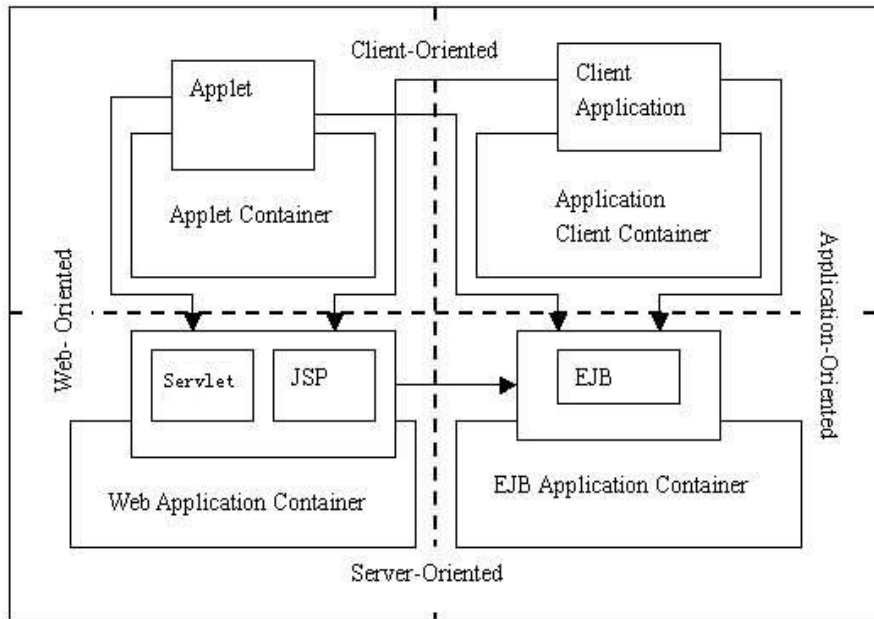


图 1-2 J2EE 组件/容器模型

由此可知，J2EE 组件和容器是密不可分的，所有的 J2EE 组件都运行在容器之上。因此 J2EE 容器对于 J2EE 组件的单元测试会产生影响是显而易见的。正因为此，对组件进行单元测试又要难于对普通 Java 类进行单元测试。J2EE 组件要和为它们提供服务的容器相互作用，而容器只有在运行时才能提供服务。所以，与容器进行交互的代码逻辑的测试就随之成为了一个难点。如何能够将被测试代码与复杂的运行环境隔离开来测试，或者如何能够提供适当的容器对象来支持单元测试；如何通过良好的单元测试框架和技术，分析解决这些问题并给出实施方案，是全方位实现 J2EE 组件单元测试时所必须要解决的难点，也是本文论述的重点。

鉴于目前关于面向对象软件测试尤其是单元测试的资料大都是一些理论介绍和一些简单的实现，没有形成体系，对 J2EE 组件单元测试的实施的经验非常缺乏，因此很有必要围绕实际项目中面向对象的 J2EE 组件单元测试所面临的问题进行研究和探讨。

1.3 论文研究内容和研究思路

1.3.1 研究目标

本文主要的研究目标：在对软件测试及面向对象软件测试理论的探讨、分析和总结的基础上，将现有良好的单元测试的框架和方法紧密地与实践相结合并且能根据需求灵活使用，解决 J2EE 组件与容器交互而产生的单元测试的问题，进而能够更全面地实施 J2EE 组件单元测试，以自动化测试替代手工测试，减轻编码人员及测试人员的工作量，增加编码人员的测试信心，提高软件开发的效率与软件的质量，增加软件的可复用性。

1.3.2 研究内容

在研究软件测试和面向对象软件的有关理论及现有测试技术的基础上，对组合框架，即当前主流的单元测试框架 JUnit 及其扩展框架 Mock Objects 和 Cactus 的特点、实现方法和工作原理进行研究，并分析比较这三者的优缺点。在遵循组件测试思想的前提下，对 J2EE 组件单元测试进行研究，重点关注容器对 J2EE 组件测试的影响。针对不同类型的 J2EE 组件，探讨如何合理使用测试框架进行单元测试以及解决因 J2EE 组件与容器交互而带来的单元测试难点。针对 J2EE 组件的单元测试给出具体的实施方案，并应用于“基于网络的物流配送管理系统平台”的单元测试中。

1.3.3 内容组织

本文的内容围绕面向对象软件单元测试的探讨和应用而展开，在研究软件测试和面向对象软件的有关理论及现有测试技术的基础上，分别介绍了组合框架中 JUnit 及其扩展框架 Mock Objects 和 Cactus 的特点、实现方法和工作原理等，并结合项目开发的实际经验，分析比较这三者的优缺点，最后给出了基于 JUnit 框架及其扩展框架对 J2EE 组件进行单元测试的实施方案和测试用例。

第一章：绪论。首先介绍面向对象软件开发和测试的研究现状及面向对象单元测试的重要性；其次介绍了组件测试的研究现状。在此基础上指出了 J2EE 组件单元测试的问题和难点。最后给出了论文的研究内容和内容组织。

第二章：面向对象软件测试的理论与技术。首先阐述了有关软件测试的一些基本概念和技术，分析研究了面向对象软件的特点和这些特点对面向对象软件测试带来的影响，概括了面向对象软件测试的内容和层次。最后介绍了面向对象软件的单元测试的技术和工具。

第三章：J2EE 组件技术。概述了 J2EE 平台、J2EE 服务技术和通讯技术，重点介绍了 J2EE 组件技术，并详细分析了容器对各组件的影响。

第四章：JUnit 单元测试框架及其扩展框架。本章从多个角度剖析了 JUnit 单元测试框架，介绍了在单元测试实施过程中积累的编写测试代码的方法和技巧。在 JUnit 理论和实现流程的基础上，分别详细介绍了 JUnit 的扩展框架 Mock Objects 的特点、实现框架以及 JUnit 的扩展框架 Cactus 的特点、测试的生命周期和工作原理。

第五章：J2EE 组件单元测试的方案与实施。首先通过对比分析，总结说明了如何合理使用 JUnit 单元测试框架及其扩展框架，来解决论文提出的问题和难点，并给出 J2EE 组件单元测试的一种实施方案。最后通过实际项目“基于网络的物流配送管理系统平台”中几个有代表性的 J2EE 组件单元测试用例来应用该实施方案。

第六章：总结和展望。

1.3.4 研究思路

本文的研究思路如下图所示：

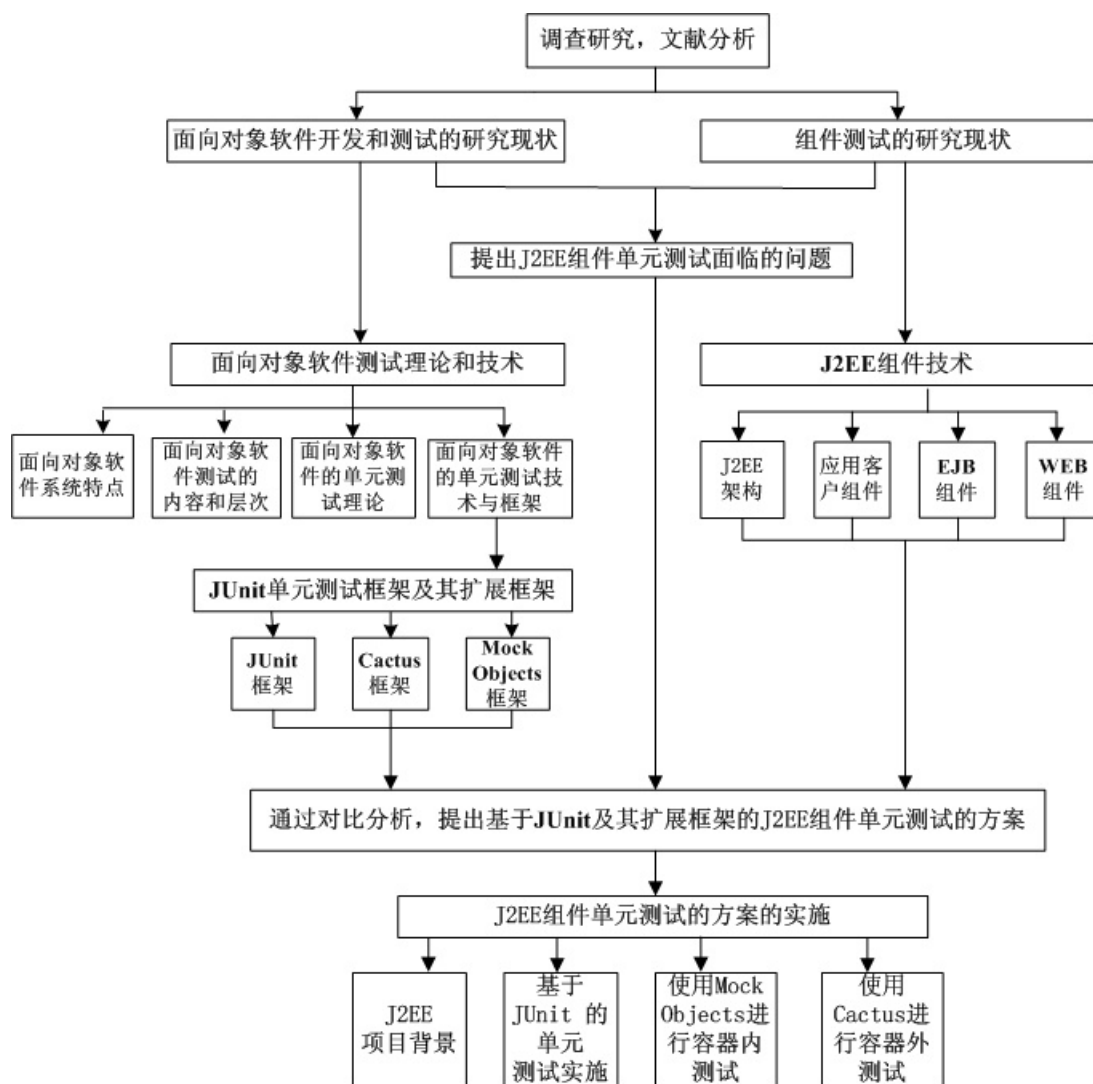


图 1-3 论文研究思路

第二章 面向对象软件测试的理论与技术

2.1 软件测试的基本理论与技术

2.1.1 软件测试的定义

关于软件测试,许多人存在着模糊甚至错误的认识。他们认为,“测试就是证明程序中没有错误”,或者“测试是证明程序能执行它的功能”,或者“成功的测试是没有发现错误的测试”。不用说,这些想法都是错误的,那么,到底什么是软件测试呢?

G.J.Myers 说:“程序测试是为了发现错误而执行程序的过程^[19]。” E.W.Dijkstra 说:“程序测试能证明错误的存在,但不能证明错误不存在^[20]。”

因此,通俗地讲,软件测试是发现并指出软件系统缺陷的过程。软件缺陷在分析、设计、开发和维护的任何阶段都有可能发生,并由此产生一个或多个 bug,表现为错误、误解和冗余,有时甚至误导开发者。软件测试包括寻找缺陷,但不包括跟踪 bug 及其修复。换句话说,测试不包括调试和修复。

2.1.2 软件测试的特点

(1) 软件的可测试性

软件可测试性就是一个计算机程序能够被测试的容易程度。可测试软件具有几个特征:可操作性、可观察性、可控制性、可分解性、简单性、稳定性和易理解性等。

(2) 软件彻底测试的不可能性

无法将所有的测试情况一一枚举,穷尽测试是不可能的。例如一个出自 Humphrey^[21]的简单程序。这个程序是分析一个由 10 个字母组成的字符串。这个程序的输入变量可产生 26^{10} (141 万亿次) 测试。可以看出,要测试所有可能的输入变量组合是根本不可能的。而且,大多数软件产品的输入变量空间远远大于这个例子。

因此,通过软件测试不可能找出所有存在的错误。这就是为什么说测试只能证明程序有错而不能证明程序无错的原因。

(3) 软件充分测试的可能性

既然软件彻底测试是不可能的,那么,测试到什么程度才算是已经对软件进行了充分测试呢? 不足的测试势必是软件带着一些未揭露的隐藏错误投入运行,过度测试则会浪费许多宝贵的资源,增加软件开发的成本。软件有无限的测试用例,要想办法把软件的相似输入、输出操作分成一组,来使无限的测试用例减少到同样有效的最小范围,这个过程称为等价分配。一般情况下,可以参考以下几项停止测试的标准:①规定测试的方法和应该达到的条件;②规定至少要查出议定书数量的错误作为停止测试的标准;③按照发现招错误的时间图来决定是否停止

测试。

2.1.3 软件测试的技术

软件测试的方法和技术是多种多样的。对于软件测试技术，可以从不同的角度加以分类：

- 按测试阶段分：需求测试、单元测试、集成测试和系统测试；
- 按测试状态分：静态测试和动态测试；
- 按测试方向分：正向测试和反向测试；
- 按测试方法分：白盒测试和黑盒测试。

图 2-1 是按照静态测试和动态测试进行的一种通常的软件测试技术分类。

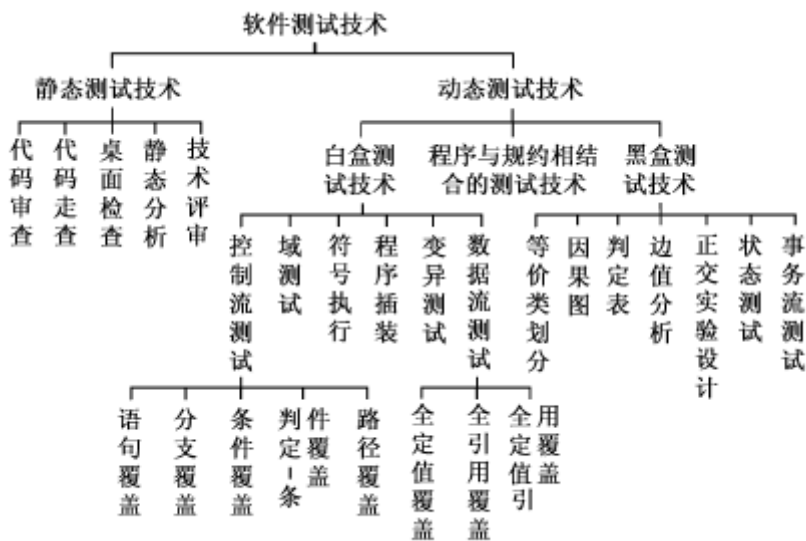


图 2-1 软件测试技术的分类

下面介绍几种常用的软件测试技术。

(1) 黑盒测试

黑盒测试也称功能测试或数据驱动测试，它是在已知产品所应具有的功能上，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能恰当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。“黑盒”法着眼于程序外部结构、不考虑内部逻辑结构，针对软件界面和软件功能进行测试。

黑盒法是穷举输入测试，不仅要测试所用合法的输入，而且还要对那些非法但是可能的输入进行测试。但是，对于实际程序而言，穷举法测试通常是不可能做到的，所以黑盒测试方法采取等价类划分、边值分析、因果图、错误推测等方法实现测试的可行性。关于黑盒测试的各种方法详细内容请参见文献^[22]。

当前使用的黑盒测试工具有：Winrunner, Bobot, WAS, e-Test, LoadSite, Aqtest, Aqtime, AQTime.net 等。

(2) 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是知道产品内部工作过程，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序。检验程序中的每条通路是否都有能按预定要求正确工作，而不考虑它的功能。白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

软件人员使用白盒测试方法，主要是对程序模块进行如下检查：

- 对程序模块的所有独立的执行路径至少测试一次；
- 对所有的逻辑判定，取“真”与取“假”的两种情况都需要至少测试一次；
- 在循环的边界和运行的界限内执行循环；
- 测试内部数据结构的有效性，等等。

就白盒测试而言，即使每条路径都测试了，程序仍可能有错。例如，要求编写一个升序的程序，错编成一个降序的程序（功能错），就是穷举路径测试也无法发现错误。再如，由于疏忽漏写了路径，白盒测试也发现不了。

因此，黑盒测试和白盒测试都不能使得测试彻底。为了有限的测试结果发现更多的错误，需要精心设计测试用例。黑盒测试和白盒测试是设计测试用例的基本策略，每一种方法对应着多种测试用例的技术。每种技术可达到一定的软件质量标准。关于白盒测试的各种方法详细内容参见文献^[22]。

当前使用的白盒测试工具有：JUnit, JTest, Test Mentor, Optimizse, CUnit, DUnit, C++Test, NUnit 等。

(3) 混合测试

混合测试在算法（或方法）粒度上检查类操作，但是仅检查公有方法和变量。从应用程序来看，混合测试可以仅看作是白盒测试，因为被测试的类是对测试人员不可见的。从类来看，混合测试可以看成黑盒测试，因为私有方法和变量是不可见的并且它也不会问结果是怎样产生的这样的问题。

例如 Windows 应用程序记事本（Notepad.exe）的混合测试就是执行图形控制对象的公有方法和观察它的公有变量。

(4) 回归测试

软件开发是一个不断变化的过程。这些变化可能是软件缺陷的修复，软件功能的改变，软件配置的改变或者是编译过程的改变。它们可能会增加一些功能并修复一些老的错误，但是也可能在软件的新版本中产生老版本未曾发生的问题。而回归测试的目的就在于及时地监控这些变化是否会影响软件质量。回归测试紧跟这些变化，验证软件原有的功能是否正常、原有的

错误是否被修复、或者已经修复的错误是否还会再发生。

回归测试由可重用的测试库以及测试的数据和结果构成。可重用的测试库包含许多测试集，每一个测试集通常代表对某一功能模块的测试，包含可反复执行的测试用例。测试库如果包含所有类型的测试集，那么就是全回归测试；如果按照需求只包含一些功能块的测试集，就是部分回归测试。测试的数据和结果则用于比较和分析回归测试，控制软件质量。测试库既要包含此次测试之前测试失败的测试用例，也需要包含测试成功的测试用例。这是因为软件中一部分的改变，可能会影响到其他部分的质量和功能。之前能够通过测试的部分，不能保证在软件发生改变之后也一定能够通过测试。然而开发时间的增加对于软件来说就意味着成本的上升。由此可见回归测试的测试库必须要包含之前所有已经测试过的测试用例。

2.1.4 软件开发过程与软件测试

软件开发过程有很多不同的模型。所有模型的共同点是，在软件生命过程中的某一阶段或者某几个阶段，必须进行不断的软件测试。完整的软件测试工作也应该贯穿整个软件生存周期，它有两方面的含义：（1）软件开发不同阶段都有软件测试工作；（2）软件测试工作的各个步骤分布在整个软件生存周期中^[23]。这体现了软件测试的一个原则：尽早开始软件测试工作，不断进行软件测试工作。

（1）传统开发过程

传统软件开发过程模型是一个自定向下、逐步细化的瀑布开发模型或其类似模型，而测试过程则是与瀑布模型象对应的自底向上、逐步集成的过程。低一级测试为上一级测试准备条件。与开发过程类似，每一个测试步骤在逻辑上是前一个步骤的继续。如图 2-2 所示。

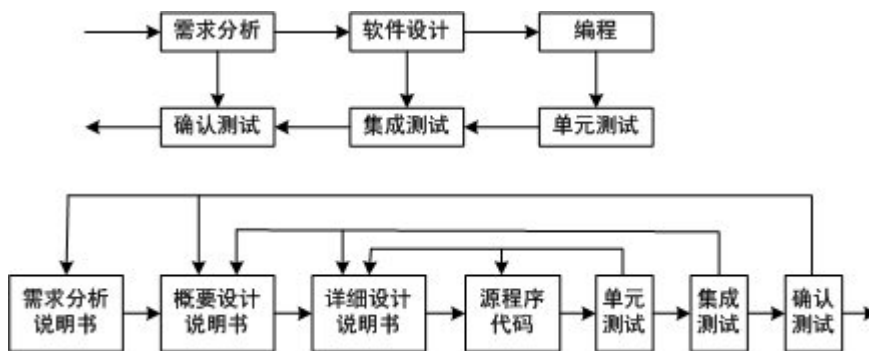


图 2-2 传统软件开发过程与软件测试

（2）增量开发过程

软件开发过程中一个普遍问题是，软件需要一个较长的时间来进行充分开发才可以交付使用，但是并没有给足够的时间去开发。而瀑布开发过程模型较理想化，它不合适需求变化快的、或者需要具有较少功能的中间系统的软件开发过程。这种问题的解决方案是交付一个中间软件产品供用户暂时使用，它带有较少的功能，但这些功能都是用户目前最迫切需要的。而且中间软件是迈向完全功能软件的坚实环节。该方法也可以大大减少软件开发风险。这种方法通常被称之为增量开发。相应过程称之为增量开发过程。在增量开发周期内，开发过程中的每一个阶

段都有自己的开发周期，一般是瀑布型或者是其简略模型。中间交付软件的实际数量依赖于开发过程。

每一个阶段软件的交付都需要进行充分的单元测试、集成测试和确认测试。这样，就把整个软件的测试过程分解为许多个小的测试过程，如图 2-3 所示。

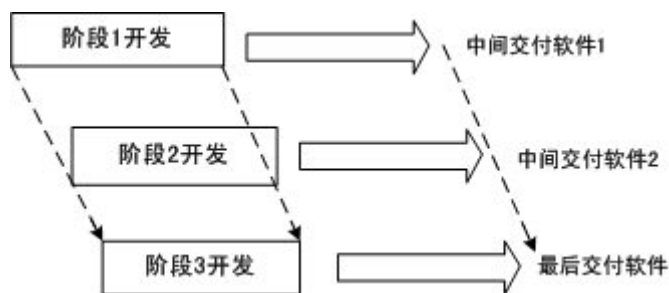


图 2-3 软件测试与增量开发过程

(3) 迭代开发过程

迭代是现在软件开发过程中常用的软件工程方法，如 RUP 和 XP 都是以此为基础来设计软件过程，以适应用户需求的变更。迭代开发模型并不需要有完整、详细的需求分析完成之后才能开始开发。相反，在仅仅明确软件一部分需求的情况下即可开始开发，不断评审已开发的程序，明确进一步的需求。这个过程不断的被重复，模型的每次循环都产生软件的一个新的版本。迭代开发过程包含重复顺序的四个阶段和一个最终的质量测试阶段，如图 2-4 所示：

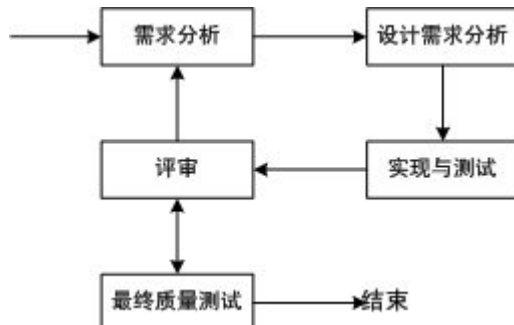


图 2-4 软件测试与迭代开发过程

软件测试在软件的增量开发和迭代开发过程模型中尤为重要，因为在这种开发过程中为了用来控制软件质量而进行的重复测试次数比传统的顺序开发模型更多、更频繁。不管软件开发用的是那种开发过程模型，软件必须进行测试且尽早测试。在软件开发的过程中越早进行测试，并在发生变化时进行全面的回归测试，就越能带来更高的效率并保证更高的软件质量。

回归测试是软件维护的主要组成部分。很多时候开发人员无法预料软件变化带来的全部结果。如果没有回归测试，软件质量就会受到损害。重复测试的容易程度对软件维护的成本有很大的影响。

2.1.5 软件测试的过程与步骤

通常，软件测试过程主要包括四个步骤，即单元测试、集成测试、确认测试和系统测试，

如图 2-5 所示。

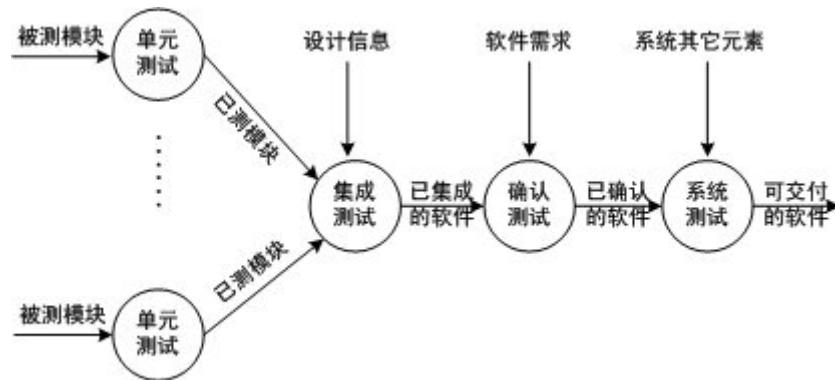


图 2-5 软件测试的过程

(1) 单元测试

单元测试是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作。其目的在于发现每个程序模块内部可能存在的差错。它主要通过对代码的逻辑结构进行分析来设计测试用例。在动态测试手段中，单元测试是一种非常高效的测试方法，并且是软件测试周期中第一个阶段。简单点说，单元测试就是测试代码撰写者依据其所设想的方式执行是否产生了预期的结果。后面将作详细的介绍。

(2) 集成测试

若要模块间接口匹配，必须保证这些模块都正确。集成测试检查所有模块的输入和输出数据并且检查类似“模块 A 提供正确的输出数据但是模块 B 不能正常工作”这样的不足。集成测试不是测试每一个模块的特性，而是模块之间的协作。

(3) 确认测试

确认测试又称为有效性测试。它的任务是验证软件的有效性，即验证软件的功能和性能及其它特性是否与用户的要求一致。在软件需求规格说明书描述了全部用户可见的软件属性，其中有一节叫做有效性准则，它包含的信息就是软件确认测试的基础。

(4) 系统测试

所谓系统测试，是将通过确认测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其它系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。系统测试的目的在于通过与系统的需求定义做比较，发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据需求分析规格说明来设计，并在实际使用环境下来运行。

下面详细阐述传统的软件测试过程中的单元测试。

一般来讲，单元测试属于编码阶段的工作，由程序编码人员自己完成。单元测试的目的是保证每个模块作为一个单元能正确运行。单元测试过程中往往能发现编码与详细设计过程中的错误。

单元测试的一般方法：首先通过编译或解释系统检查并改正程序中所有的语法错误，然后以详细设计模块说明为指南，了解该模块的 I/O 条件和模块的逻辑结构，对重要的控制路径进行测试，以便发现模块内部的错误。单元测试主要采用白盒测试的测试用例，以黑盒测试的测试用例为辅助，使之对任何合法或不合法的输入，都能鉴别和响应。有关单元测试的相关信息详见 IEEE-标准 1008-1987^[24]。

(1) 单元测试的主要内容

● 模块接口

首先应该对通过模块接口的数据流进行检测，如果数据不能正确地输入和输出，其他测试都无法进行。

● 局部数据结构

模块局部数据结构是最常见的错误来源，应该设计相应的测试用例以便发现数据结构方面的错误。

● 路径测试

由于通常不可能做到穷举测试，所以在单元测试期间要选择适当的测试用例，对模块中重要的执行路径进行测试。应当设计测试用例查找错误的计算、不正确的比较或不正常的控制流而导致的错误。

● 出错处理

比较完善的模块设计要求能预见出错的条件，并设置适当的出错处理，以便在一旦程序出错时，能对出错程序重做安排，保证其逻辑上的正确性。这种出错处理也应当是模块功能的一部分。

● 边界测试

在边界上出错是常见的，要特别注意数据流和控制流中刚好、大于或小于确定的比较值时出错的可能性。

(2) 单元测试流程

在单元测试时，模块本身并不是一个可以独立运行的程序，在进行单元测试时，同时要考虑和外界的交互问题。用一些辅助模块去模拟与被测模块相联系的其它模块。这些辅助模块就是驱动模块和桩模块。

● 驱动模块

相当于被测模块的主程序。用它接收测试用例的测试数据，把这些数据传给被测试模块，最后输出结果。

● 桩模块

用以代替被测试模块所调用的子模块。桩模块可以做少量的数据操作，不需要把子模块的所有功能都带进来。

被测模块、与被测试模块相关的驱动模块和桩模块共同构成了单元测试的测试环境，如图 2-6 所示。

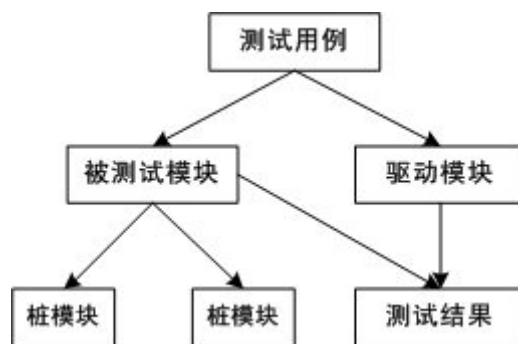


图 2-6 单元测试环境

(3) 单元测试的好处^[25]

^[25]著名的测试专家 Boris Beizer 博士认为：“软件开发历史上最臭名昭著的错误都是单元错误——即通过适当的单元测试可以发现的错误。”他引证了 Voyager 的错误（将探测器发送到太阳）、AT&T 和 DCS 的错误（曾造成美国三分之一的电话瘫痪）以及 Intel 奔腾芯片错误，都能够通过全面的单元测试排除掉。单元测试能够极大的改善软件质量，它是在最容易和成本最低的阶段帮助程序员检测和纠正错误。

首先，单元测试最接近错误，它能够有效的检测在应用级测试中很难找到的错误。类一级的测试提供了一种更有效的发现错误的方法。单独测试一个与其它对象分离的类时，由于更接近错误，找到潜在的错误就会变得容易得多。

其次，单元测试简化错误检测的另一个方面是防止错误繁衍出更多的错误，避免了总是要穿过重重迷雾去寻找一个简单的错误。因为错误之间是有相互作用的，如果代码中遗留了一个错误，它可能会导致更多的错误。因此，如果系统延迟到以后的开发阶段才去测试，可能不得不去纠正更多的错误，花费更多的时间去发现和改正每个错误，并且要修改更多的代码。如果刚开发完一个类的时候测试，发现和纠正一个错误就会容易得多，并且错误繁衍的机会也会降到最低限度。结果是：极大的减少了调试时间和成本。

2.2 面向对象软件测试的理论

面向对象技术是一种全新的软件开发技术，正逐渐代替广泛使用的面向过程的开发方法，被看成是解决软件危机的新兴技术。面向对象技术产生更好的系统结构，更规范的编程风格，极大地优化了数据使用的安全性，提高了程序代码的重用。一些人就此认为面向对象技术开发出来的程序无需进行测试。应该看到，尽管面向对象技术的基本思想保证了软件应该具有更高的质量，但是实际上无论采用什么样的编程技术，编程人员的错误都是无法避免的，而且由于面向对象技术开发的软件代码的重用率高，更需要严格的测试，避免错误的繁衍。因此，软件测试并没有因面向对象编程的兴起而丧失它的重要性。

从 1982 年在美国北卡罗来纳大学召开首次软件测试的正式技术会议至今，软件测试理论迅

速发展，并相应出现了各种软件测试方法，使软件测试得到极大的提高。然而，一度实践证明行之有效的面向过程的软件测试方法对面向对象技术开发的软件多少显得力不从心。尤其面向对象技术所独有的多态、继承、封装等特点，产生了传统语言设计所不存在的错误可能性，或者使得传统软件测试中的重点不再显得突出，或者使原来测试经验认为和实践证明的次要方面成为了主要问题。在传统的面向过程的程序中也许只需要考虑一个函数的行为特点，而在面向对象程序中，不得不同时考虑其父类相关方法的行为和子类相关方法的行为。

面向对象程序的结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起测试的方法已成为不可能的。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都不同于以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点，应该有一种新的测试模型。

要研究针对面向对象软件系统的测试技术，首先必须了解面向对象软件系统的特点和这些特点对软件测试的影响，然后确定面向对象软件测试的内容，才能更进一步地对面向对象软件的测试方法进行研究。

2.2.1 面向对象软件系统的特点及对测试的影响

2.2.1.1 面向对象软件系统的特点

事实上，面向对象是个老概念，早在 Dahl 和 Nygaard 设计的 Simula67 语言中，就提出并实现了一些面向对象概念——类。类是具有相同属性和服务的一组对象集合，它为属于该类的全部对象提供了统一的抽象描述。其内部包括属性和服务（方法）两个主要的部分^[26]。面向对象软件是有一系列类组成的，在类中定义了封装的数据（用于表示对象的状态）以及作用在数据上的操作（方法），数据和操作统称为特征。对象是类的实例，类和类之间按集成关系组成一个无环有向图结构，父类中定义了共享的公共特征，子类除了继承父类中定义的所有特征外，还可以引入新的特征，也允许对继承的方法重新定义。面向对象语言提供的动态绑定方法和对象动态的联系起来，继承和动态绑定机制的结合是面向对象语言与其他预言相比最为有力的地方，它使得程序具有较大的灵活性，特别是当用户需求变动时，程序变动最小。

面向对象技术具有的基本特征：

(1) 抽象

抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的部分。抽象并不打算了解全部的问题，而只是选择其中的一部分，暂时不用部分细节。比如，要设计一个学生成绩管理系统，考察学生这个对象时，只需要关心他的班级、学号、成绩等，而不用关心他的身高、体重这些信息。抽象包括两个方面，一是过程抽象，二是数据抽象。过程抽象是指任何一个明确定义功能的操作都可以被使用者看作单个的实体对待，尽管这个操作实际上可能由一系列更低级的操作完成。数据抽象定义了数据类型和施加于该类型对象上的操

作，并限定了对象的值只能通过使用这些操作修改和观察。

(2) 继承

继承是一种联结类的层次模型，并且运行和鼓励类的重用，它提供了一种明确表述事务共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或者增加新的方法使之更适合特殊的需要。这也体现了现实世界中一般和特殊的关系。继承很好的解决了软件的可重用问题。比如说，所有的 Windows 应用程序都有一个窗口，他们可以看作都是一个窗口类派生出来的，但是有的应用程序用于文字处理，有的应用程序用作绘图，这是由于派生出了不同的子类，各子类添加了不同的特性。继承有单继承和多继承^[27]。

(3) 封装

封装是面向对象的特征之一，是对象和类概念的主要特征之一。封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本的概念，即现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个受保护的接口可访问其它对象。一旦定义一个对象的特性，则有必要决定这些特性的可见性，即哪些特性对外部世界是可见的，哪些特性用于表示内部状态。在这个阶段定义对象的接口。通常，应禁止直接访问一个对象的实际表示，而应通过操作接口访问对象，这称为信息隐藏。事实上，信息隐藏是用户对封装性的认识，封装则为信息隐藏提供支持。封装保证了模块具有良好的独立性，使得程序维护修改较为容易。对应用程序的修改仅仅限于类的内部，因而可以将应用程序修改带来的影响减少到最低程度。

(4) 多态性和动态绑定

多态性是指允许不同类的对象都对同一消息做出响应。比如同样的加法，把两个实践加在一起和把两个整数加在一起肯定完全不同。又比如，同样的选择编辑-粘贴操作，在字处理程序和绘图程序中有不同的效果。多态性包括参数化多态性和包含多态性^[28]。多态性具有灵活、行为共享、代码共享的优势，很好的解决了应用程序函数同名的问题。

要实现运行时的多态性，获得动态多态性的对象，必须建立一个类的等级，并将父类中的有关成员方法定义为抽象方法。然后，在派生类中对其重定义。这样当基类指针指向不同的派生类对象时，就可以执行相应对象的操作，从而实现动态的多态性。

2.2.1.2 面向对象软件系统的特点对软件测试的影响

面向对象软件系统存在的特点，对面向对象软件测试产生很多深远的影响，使得面向对象系统和以往的基于过程的软件系统的软件测试相比在很多方面提出了新的问题。下面介绍面向对象软件系统的各个特点对面向对象软件测试的影响。

(1) 信息隐藏和封装对测试的影响

封装技术对于软件测试产生两方面的影响：

- 最基本的可测试单元将不再是子程序，而是类和它的实例——对象。
- 必须对原有的继承测试策略进行修改。

类的封装机制给软件测试带来了困难，它把数据和操作数据的方法封装在一起，限制对象属性对外的可见性和外界对它的操作权限，从而有效地避免了类中有关实现细节的信息被错误地使用，而这样的细节性信息正是软件测试所不可忽略的。由于面向对象的软件系统在运行时由一组协调工作的对象组成，对象具有一定的状态，所以对面向对象的程序测试来说，对象的状态是必须考虑的因素，测试应涉及对象的初态、输入参数、输出参数、对象的终态。对象的行为是被动的，它只有在接收有关消息后才被激活来完成所请求的工作，并将结果返回给发送消息者。在工作过程中，对象的状态可能被修改，产生新的状态。面向对象软件测试的基本工作就是创建对象（包括初始化），向对象发送一系列消息，然后检查结果对象的状态，看其是否处于正确的状态。但是，对象的状态往往是隐蔽的，若类中未提供足够的存取方法来表明对象的实现方式和内部状态，则测试者必须增添这样的方法^[29]。

在面向对象程序中，一个类就可以包含多个子程序单元，并且同一个子程序单元可以包含于多个类中，比如说，如果我已经测试了一个类 A 中的一个“子程序”，类 B 继承类 A，这并不能保证类 B 中相同的子程序（继承类 A 得到的）的正确性，因此，在面向对象系统中，单独测试传统概念上的子程序已经没有意义，类和它的实例——对象就是最小的可测试单元。Dewayne E. Perry 和 Gail E. Kaiser^[30]根据 Weyuker 的测试充分性公理^[31]对该问题进行了讨论，认为父类中的“子程序”（方法）创建的黑盒测试用例不一定会对子类的同一个“子程序”（方法）适合，子类中继承的方法和重新定义的方法也是充分的。同时，类中的方法实现和一些数据对用户是不可见的（隐藏的），也就是说对象的状态往往是隐藏的，Bashir^[32]提出了测试时，必须要在类定义中添加一些表明对象状态实现方式和内部状态的函数，以便在测试的时候能够确定对象的状态。也就是说对类的测试除了需要测试类中包含的类方法，还要测试类的状态，这是传统单元测试所没有的。

同时，信息隐藏和封装也会影响到集成测试，在一些非面向对象测试方法中，一旦一个单元被测试过后，它就会被集成到一个更大的系统中去，如果是基于非增量测试，这意味着对每一个单元单独测试，然后同时集成这些单元，测试总的系统的结果。面向对象系统不可能把传统意义上的子程序作为测试单元，逐步集成起来进行测试。

（2）继承性对测试的影响

继承性有很多的优点：增加软件重用性，使得软件更具有直观性和自然性。同时它给软件测试也带来新的挑战。类的继承机制增加了软件测试的复杂性。继承由扩展、覆盖和特例化三种基本机制实现，其中扩展是子类自动包含父类的特征；覆盖是子类中的方法与父类中的方法有相同的名字和消息参数以及相同的接口，但方法的实现不同；特例化是子类中特有的方法和实例变量。

在面向对象的软件测试中，未重定义的继承特征是否还需要进行测试？重定义的特征需要

重新测试是显然的,但应该如何测试重定义的特征呢?Weyuker 曾经提出了 11 条基于程序的测试数据集的充分性公理^[37], Perry 与 Kaiser 根据 Weyuker 公理对这些问题进行了讨论^[38]。

还有,面向对象软件中的多重继承(即一个子类有多个父类)和重复继承性(即类层次图中一个子类能通过多条路径继承同一个父类的特征),虽然使软件的共享程度有所增加,但也显著提高了子类的复杂性,并且会导致一些从功能角度很难发现的隐含错误。比如继承特性之间的冲突,于是研究继承关系的测试方法和测试策略是面向对象研究的一个重点。总之,继承性使测试更加复杂。

(3) 多态性和动态绑定对测试的影响

多态性给程序员编程提供了高度柔性,问题抽象和易于维护。但多态性和动态绑定所带来的不确定性,使得传统测试实践中的静态分析法遇到了不可逾越的障碍。而且它们也增加了系统运行中可能的执行路径,加大了测试用例的选取难度和数量。这种不确定性和骤然增加的路径组合也给测试覆盖率的满足带来了挑战。

多态性和动态绑定为程序的执行带来了不确定性,给软件测试带来了新的挑战。多态性和动态绑定是面向对象方法的关键特性之一。同一消息可以根据发送消息对象的不同而采用多种不同的行为方式,这就是多态的概念。如根据当前指针引用的对象类型来决定使用正确的方法,这就是多态性的行为操作。运行时系统能自动为给定消息选择合适的实现代码,这为程序员编程提供了高度柔性、问题抽象和易于维护等特性。例如:设有类 A、类 B 和类 C 三个类,类 B 继承类 A,类 C 又继承类 B。方法 a() 分别存在于这三个类中,但在每个类中的具体实现则不同。同时,在程序中存在一个方法 fn(),该函数在形参中创建了一个类 A 的实例 Ca,并在方法中调用了方法 a()。程序运行时相当于执行了一个分情况语句 Switch,首先判定传递过来的实参的类型(类 A 或类 B 或类 C),然后再确定究竟执行哪一个类中的方法 a()。而在测试时必须为每一个分支生成测试用例,以覆盖所有的分支和所有的程序代码。因而,多态性和动态绑定所带来的不确定性,使得传统测试实践中的静态分析方法遇到了不可逾越的障碍,也增加了系统运行中可能的执行路径,加大了测试用例选取的难度和数量。多态性给软件测试带来的问题仍然是目前研究的重点及难点问题之一。

从分析可知,虽然信息隐蔽与封装性使得类具有较好的相对独立性,有利于提高软件的易测试性,保证软件的质量。但是,这些机制与继承机制和动态绑定给软件测试带来了新的课题。

2.2.2 面向对象软件测试的内容与层次

2.2.2.1 面向对象软件系统测试的内容

面向对象的软件测试即在测试过程中继续运用面向对象技术,进行以对象概念为中心的软件测试。正如面向对象软件开发过程不是凭空而来的一样,面向对象软件测试也要借鉴和吸收传统的软件测试方法中可以使用的部分。就测试而言,面向对象系统的测试与传统软件系统的测试并没有本质的差别,但面向对象软件测试通过捕捉面向对象分析和面向对象设计模型,可以检查程序与模型不匹配的错误,这一点是传统软件难以达到的。在所有的开发系统中都是根

据规范说明来验证系统设计的正确性，但面向对象的开发方法为测试带来一些新的可能性，同时也引入了许多新的问题。Binder 和 Barley 研究了面向对象的特征如封装性、继承性、多态和动态绑定等，认为这些特性的引入增加了测试的复杂性^{[33][34]}。文献^[35]提出面向对象软件测试的基本工作就是创建对象（包括初始化），向对象发送一系列消息，然后检查结果对象状态，检测是否处于正确状态。通常，面向对象软件测试划分为以下几个阶段：制定测试计划、产生测试计划、执行测试和评价测试。

2.2.2.2 面向对象软件系统测试的层次

软件测试层次是基于测试复杂性分析的思想，是软件测试的一种基本模式。传统层次测试时基于功能模块的层次结构，因为面向对象软件的特点而不能再套用传统的测试思想和方法。目前，关于面向对象软件测试的层次划分还尚未达成共识。在面向对象软件中，继承和组装关系刻画了类之间内在的层次，他们既是构造系统结构的基础，也是构造测试层次的基础。Seigel 描述了从类、基础部件、系统部件到应用部件的面向对象的层次测试^[36]。面向对象的程序的执行实际上是执行一个由消息连接起来的方法序列，而这个方法序列通常是由外部事件驱动的。根据这一执行特性，P.C.Jorgensen 和 C.Erichson 认为面向对象的测试应该分为 5 个层次：方法测试、方法/消息路径（MM Path）测试、系统基本功能测试、线程测试和线程间相互作用测试。

下面是两种较为普遍的面向对象软件测试层次划分方法。

（1）面向对象软件测试层次的传统划分

根据测试层次结构，面向对象软件测试总体上呈现从单元级、集成级到系统级的分层传统测试。测试集成的过程是基于可靠部件组装系统的过程。测试可用不同的方法执行。通常的方法是按设计和实现的反方向测试。首先验证不同层，然后使用事件集成不同的程序单元，最终验证系统级。根据测试层次结构确定相应的测试活动。

（2）面向对象软件测试层次的面向对象的划分

另一种普遍的面向对象软件测试层次划分为 3 个级别：①类级，考察封装在一个类中的方法和数据之间的相互作用；②类族级，考察一组协同操作的类之间的相互作用；③系统级，考察由所有类和主程序构成的整个系统。在面向对象的系统中，最小的可测试单元已不是方法而是类和类的实例，类是面向对象方法中最重要的概念，是构成面向对象程序的基本成分。面向对象的特点：封装、继承、多态等特点都是通过类实现的。经验表明，类测试是必需的，是发现错误的最重要手段。

因此，在所提出的测试方法中将对类层的测试作为单元测试，而对于由类集成的模块测试作为集成测试，系统测试与传统测试层相同^[39]。面向对象的测试层次与传统的测试层次的对应关系如图 2-7 所示。

传统测试	单元测试	集成测试	系统测试
面向对象测试	类测试 方法测试 对象测试	类的集成模块测试	系统测试

图 2-7 面向对象软件的测试层次

2.2.3 面向对象软件的单元测试理论

2.2.3.1 面向对象的单元测试分析

面向对象软件从宏观上看是类与类之间的相互作用。在面向对象系统中，系统的基本构造模块是封装了的数据和方法的类和对象，而不再是一个个能完成特定功能的功能模块。每个对象有自己的生存周期，有自己的状态。消息是对象之间相互请求或协作的途径，是外界使用对象方法及获取对象状态的唯一方式。对象的功能是在消息的触发下，由对象所属类中定义的方法与相关对象的合作共同完成，且在不同状态下对消息的响应可能完全不同，对象的状态可能被改变，产生新的状态。对象中的数据和方法是一个有机的整体，测试过程中不能仅仅检查输入数据产生的输出结果是否与预期的吻合，还要考虑对象的状态。模块测试的概念已不适用于对象的测试。类测试将是整个测试过程的一个重要步骤，它与传统单元测试的区别用图 2-8 表示。

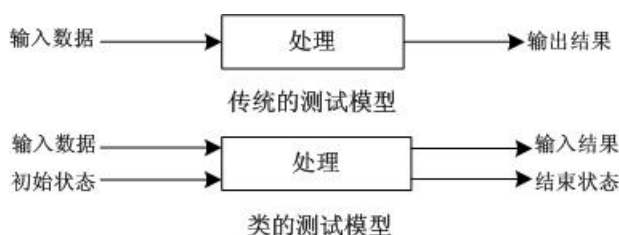


图 2-8 两种不同的测试模型

面向对象软件的类测试与传统软件的单元测试相对应，但和传统的单元测试不一样。类包含一组不同的操作，并且某个特殊操作可能作为一组不同类的一部分存在。一般地，类的测试可分为 3 级：单个成员函数、公有成员函数和公有成员函数间接口测试。

另外，一个对象有它自己的状态和依赖于状态的行为，对象操作既与对象的状态有关，又可能改变对象的状态。因此，类测试时，不仅要将操作作为类的一部分，同时要把对象与其状态结合起来，进行对象状态行为的测试^[40]。因此，从另外一个角度来看，类测试又可分为：基于服务的测试、基于状态的测试和基于响应状态的测试。

2.2.3.2 面向对象软件的单元测试的类型

当在编写代码阶段，第一个测试应该是逻辑单元测试。但当写了越来越多的测试和越来越多的代码，就会发现需要开始添加集成和功能单元测试。现将三种不同类型的单元测试总结如下：

(1) 逻辑单元测试 (Logic unit tests): 这种单元测试主要检查代码逻辑性。这些测试通常只是针对单个方法。可以通过 mock objects 或者 stub 来控制特定的方法和边界。

(2) 集成单元测试 (Integration unit tests): 这种单元测试主要是在真实环境 (或者真实环境的一部分) 下的两个组件相互交互的测试。例如: 一段访问数据库的程序已经被测试证实能够有效地访问到数据库, 那么就可以提供和数据库交互的接口。

(3) 功能单元测试 (Functional unit tests): 这种单元测试越出了集成单元测试的边界, 目的是为了确认激励-响应。例如: 一个 Web 站点是被保护的, 只能通过登录才能够进入。如果没有登录, 访问页面的结果将重新被指向登录页面。功能单元测试检验发送一个 HTTP 请求到这个页面, 并检查结果是一个请求重定向 (302 响应代码)。但是它不会检查整个的工作流是否导向登录画面。工作流属于纯粹的软件功能测试领域。

图 2-9 说明了这三者之间是如何相互作用的。并且定义了各种单元测试的边界。这些测试可以通过链的位置来定义。所有的这三种测试能够确保代码可以正确的工作。

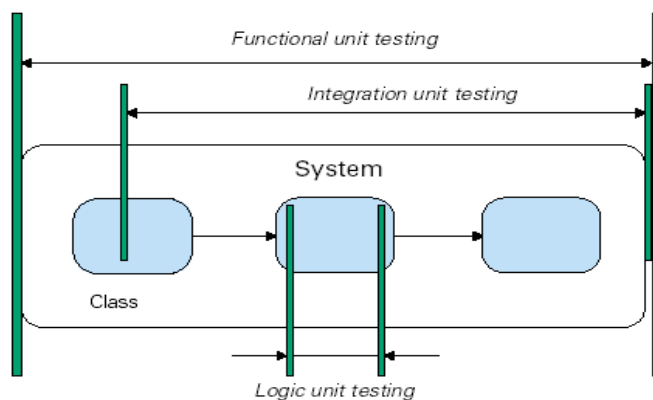


图 2-9 应用程序内部的单元测试

严格地说, 功能单元测试不是纯粹的单元测试, 也不是纯粹的功能测试。它们相对于纯粹的单元测试而言更多地依赖于外部环境。但是他们不像纯粹的功能测试那样检查完整的工作流。在本文中, 对此种单元测试不做深入的探讨。

2.2.3.3 面向对象软件的单元测试的相关理论和方法

面向测试的方法, 比如极限编程 (Extreme Programming, XP) 和测试驱动的开发 (test-driven development, TDD), 将测试, 尤其是单元测试, 看作构建应用程序过程中的关键任务, 把测试与开发置于同等地位, 使其成为风险相当的一部分。

XP (Extreme Programming) 是一个关于如何实现这类协作的模型。它更加重视工作代码的生产, 而不是过程和结果的文档。XP 提倡一个规则叫做 Test 设计^[41]。采用 Test-First 设计方法, 在编写一个新功能前先写一个单元测试, 用它来测试实现新功能需要但可能会出错的代码。这意味着, 测试首先是失败的, 写代码的目的就是为了让这些测试能够成功运行。这个方法有许多优点, 很多软件开发过程支持或要求 Test - First 方法, 如在 XP 中它就属于强制性的要求。

Test - First 要求程序员直到他们能首先编写出单元测试代码, 才可以编写任何生产代码。这意味着程序员对每一个可能出错的功能点, 都要编写单元级的测试用例。也意味着这些测试用例的编写要先于生产代码的编写, 编写生产代码之前编写测试代码对生产代码有很多的影响, 可见单元测试在 **XP** 中占有的地位。但是, **XP** 还定义了合格测试。合格测试是黑箱测试, 它们只要比较预期输出和实际输出。合格测试更多地集中在功能上, 而不是底层逻辑, 有助于确定应用程序是否实现了预定功能。

测试驱动的开发 (**Test-Driven development, TDD**) 是一项编程实践, 也是 **XP** 的一个分支, 它结合了 **XP** 的测试策略, 但是没有继承该方法的所有方面。**TDD** 要求在对应用程序编码之前编写测试, 并且通常在开发期间运行它们。如果一个应用程序没有通过某项特定的测试, 这表明该应用程序中存在缺陷。传统的开发周期是由如下步骤组成的: [编码, 测试, (重复), 交付]。开发者们使用 **TDD** 则做出了一个似乎很微小但实际上是巨大的有效调整: [测试, 编码, 重构, (重复), 交付]。这样测试就推动了设计的发展, 单元测试对其实现起到了至关重要的作用。

当然, 上述方法的实现要求测试人员就是开发人员。目前无论是工业界还是学术界都认为单元测试应该由开发人员开展, 这是因为从单元测试的过程看, 单元测试主要采用白盒测试的方法, 离不开深入被测对象的代码, 同时还需要构造驱动模块、桩函数, 因此开展单元测试需要较好的开发知识。从人员的知识结构、对代码的熟悉程度考虑, 开发人员具有一定的优势。

2.3 面向对象软件的单元测试技术

2.3.1 单元测试技术与框架

单元测试技术从整体上分为白盒测试与黑盒测试, 其中前者使用程序设计的控制结构导出测试用例, 针对程序的内在结构 (逻辑、数据流), 后者目的是验证单元实现的功能, 而不需要知道程序是如何实现它们的。黑盒测试关注的是单元的输入与输出, 不是白盒测试的替代品, 而是辅助白盒测试发现其他类型的错误。应用到具体的单元测试时, 又会体现为新的测试技术或方法, 如 **Stub** 技术、**Mock Objects** 技术以及容器内测试的方法等。它们又会发展成为新的单元测试框架或融入到新的单元测试工具中。Java 是一种完全面向对象的语言, 本文中就以基于 Java 的 **J2EE** 开发作为面向对象软件开发的代表, 结合框架和工具来了解面向对象的单元测试。下面是具有代表性的业界流行的针对 Java 开发进行面向对象的单元测试框架和工具。

为了提高单元测试的效率, 特别是提高进行回归测试时的效率, 需要在单元测试中引入自动化。而使用单元测试框架是实现自动化的首选途径。单元测试框架中首先要提到的是, 已经成为 Java 中开发单元测试的框架的事实标准的 **JUnit**。

JUnit 是一个开源 (**Open Source**) 的项目, Java 编程的最流行的单元测试框架。用其主页上的话来说就是: “**JUnit** 是由 **Erich Gamma** 和 **Kent Beck** 编写的一个回归测试框架 (**regression testing framework**), 用于 Java 开发人员编写单元测试之用。” **JUnit** 已经成为 Java 中开发单元测试的框架的事实标准。事实上, **JUnit** 背后的测试模型 (称作 **xUnit**) 正成为任何语言的标准

框架。ASP、C++、C#、Eiffel、Delphi、Perl、PHP、Python、REBOL、Smalltalk 和 Visual Basic 都已经有了 xUnit 框架。JUnit 框架的提出,使得单元测试自动化成为可能。使用 JUnit 可以很方便地构建测试用例,能够使回归测试达到最大的自动化。而且作为一个简单、实用的架构,它可以被很灵活地扩展,它给予了一种测试思想,可以利用这种思想去架构更符合需求的测试框架。

除了 JUnit 之外还有许多单元测试框架,根据单元测试的类型,主要分为以下三类:

(1) 逻辑单元测试框架。这类框架主要是处理对单个模块(例如,单个类)代码逻辑进行测试,没有多个模块之间的真正交互,测试粒度最小。其中 Mock Objects^[42]是其中最优秀的单元测试框架之一,也是 JUnit 的扩展框架。通常使用 Mock Objects 对各种组件进行隔离容器的测试。

(2) 集成单元测试框架。这类框架主要处理服务端各个模块之间交互的测试。Cactus^[43]是典型的这种类型的服务器端测试框架,也是 JUnit 的扩展框架。Cactus^[43]可以使开发人员很轻松地测试服务器端的程序。它的测试粒度比代码逻辑单元测试框架大,主要处理和容器的交互测试,测试集成代码。例如,测试 Servlet、JSP、EJB、JNDI 等。通常使用 Cactus 对各种组件提供真正的容器内的测试。

(3) 功能单元测试框架。这类框架主要对模块功能进行测试,而不再关心具体实现。这类框架的测试粒度最大。HttpUnit^[44]就是这种功能单元测试框架的例子,对服务端代码返回的值进行测试。HttpUnit 同时也处理标准的功能测试。这类单元测试已经不是纯粹的单元测试,而更偏重于功能测试。

2.3.2 单元测试工具

单元测试非常需要工具的帮助,覆盖率工具自然也不能缺少,否则用例执行后无法得到测试质量如语句覆盖、路径覆盖等情况,也就无法对被测对象进行进一步的分析。应用较广的分析覆盖率的工具有 Logiscope、TrueCoverage、PureCoverage 等,它们的功能有强有弱,可以根据实际情况采用。为了提高单元测试的效率,特别是提高进行回归测试时的效率,需要在单元测试中引入自动化。以下则是部分已经产品化的自动化的单元测试工具:

(1) Jtest, 是一个集成的、易于使用和自动化的 Java 单元测试工具。它增强代码的稳定性,防止软件错误,并首次实现了单元测试技术的自动化,包括自动的白盒测试、黑盒测试和回归测试。Jtest 的测试生成系统专利技术(patent #5, 784, 553 & #5, 761, 408)通过自动生成和执行能够全面测试类代码的测试用例,使白盒测试完全自动化。Jtest 通过读取代码中用 DbC (按合同设计)语言建立的说明信息,自动建立和执行验证代码功能性的测试用例。Jtest 的回归测试能够在类一级执行回归测试,这意味着能够更早地运行测试用例以监测代码的完整性。

Jtest 将突破性的自动化技术手段和关键的测试方法平稳地集成到开发过程中,缩短调试时间,并将测试提高到新的水平。

(2) J2EE Tester 测试器是一款用于测试发布于服务器端(如 Weblogic, Websphere)的 EJB

组件、Servlet 以及 Beans 的黑盒/白盒智能化测试工具。它的功能主要有以下三点：自动生成被测试类的覆盖度测试报告；自动生成被测试类的出错报告，记录抛错的行数，出错类型，方法名称，时间以及该行的代码；按照需要在发生错误的行产生断点，让测试者查看此时各个变量的状态。

J2EE Tester 测试器是基于 JPDA (Java Platform Debugger Architecture) 技术开发的智能全自动测试工具，它支持远程调试和本地调试，支持调试模式中的登陆 (Attach) 以及监听 (Listen) 模式。它将使得我们对于服务器端 J2EE 组件的测试更快，更准确，更好，更方便。

(3) Rational Quality Architect (RQA)，Rational Rose 中一个和 EJB 特性相关的工具，可用作设计测试 EJB，可以使用 RQA 为 EJB 作单元测试。还可以使用在 Rose 中定义的顺序图来作多个 EJB 的测试。当依赖某个软件组件，但是它还未设计好，那么可以使用 RQA 来产生框架。

(4) Rational PurifyPlus，是一个测试 Java 应用程序的完整解决方案。它包括三个应用程序：Rational Purify (内存分析工具)、Rational Quantify (应用程序执行时间分析工具)、Rational PureCoverage (代码覆盖率分析工具)。这三种工具不仅对 Java，对 Visual C/C++，Visual Basic 和 .Net 应用也都提供了全面的支持。为了利用 Rational PurifyPlus 来测试 Java Server 端应用程序，必须先服务器上安装它。

2.4 本章小结

本章从软件测试的特点、基本方法、过程以及软件开发对软件测试的影响几个方面概述了软件测试的基本理论和技术，并详细地介绍了单元测试的内容和必要性。在此理论基础上，引入了面向对象软件测试，分析研究了面向对象软件的特点和这些特点对面向对象软件测试带来的影响，概括了面向对象软件测试的内容和层次，介绍了面向对象软件的单元测试的类型和相关理论方法。然后，总结性的分析了面向对象软件的单元测试的技术，介绍了当前主流的单元测试框架 JUnit 与其扩展框架 Mock Objects 与 Cactus 等。最后列举了目前流行的自动化单元测试工具。

第三章 J2EE 组件技术

3.1 J2EE 架构技术

随着企业应用需求的发展，企业应用体系结构和开发方法发生了巨大的变化。基于层次化组件模式的 J2EE 体系结构为开发分布式的企业应用提供了一个标准的架构。它提供了组件再用、一致化的安全模型以及灵活的事务控制。它通过把业务逻辑和底层网络技术分离开来，使得在该平台上开发的系统具有可伸缩性、可扩展性、可靠性、易开发和易维护性。遵循 J2EE 标准，不仅可以比以前更快的向市场推出创造性的企业应用系统解决方案，而且平台独立的、基于组件的 J2EE 解决方案不会被束缚在任何一个特定厂商的产品和 API 上。

J2EE 定义了许多用来构建大型的、分布式的多层企业应用技术。根据使用的特点，可以把这些技术分为服务技术、通讯技术和 组件技术^[45, 46]。各种技术在 J2EE 多层结构中的分布如图 3-1。

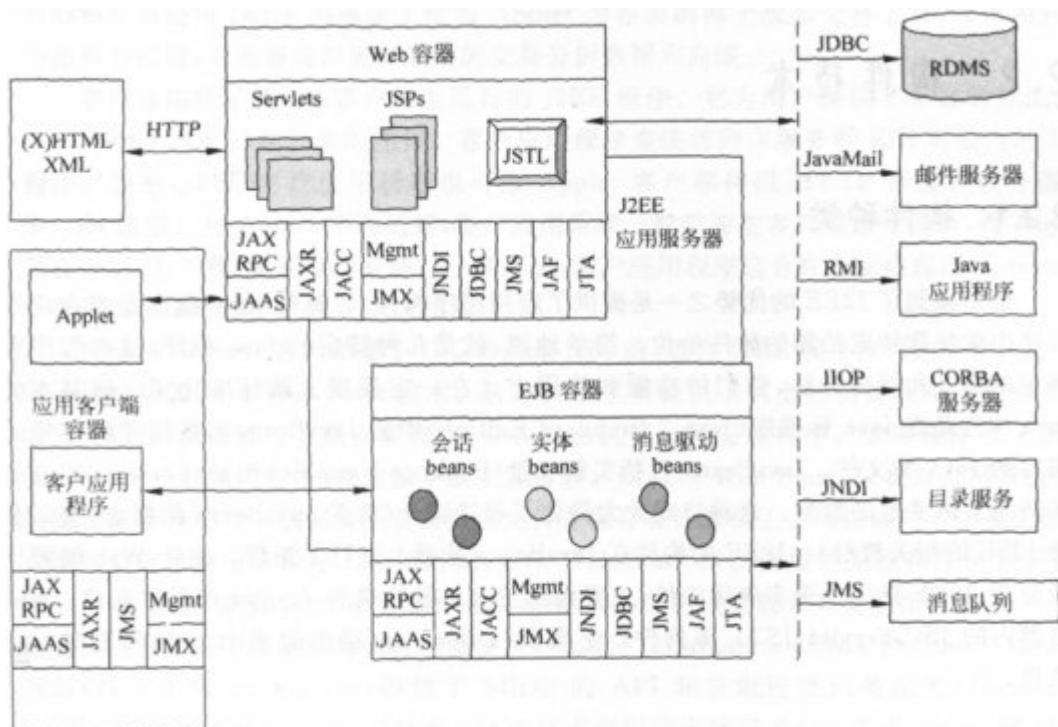


图 3-1 J2EE 多层结构框架图

(1) 服务技术

服务技术用来为应用组件提供服务，从而更有效地发挥组件的功能。J2EE 对于应用组件的服务是由容器来管理的，从而允许开发者将主要精力集中在业务逻辑上。J2EE 提供的服务技术主要有 Java 命名和目录服务接口（Java Naming Directory Interface， JNDI）、Java 数据库连接

(Java DataaBase Connectivity, JDBC) API、Java 事务 API (Java Transaction API, JTA)、和 Java 事务服务 (JTS)、Java 连接框架 (Java Connector Architecture, JCA) 等。

(2) 通讯技术

通讯技术对应用程序设计者来说是透明的。它为应用的不同部分之间提供了通信机制，而不管它们是本地的还是远程的。如 Internet 协议 (HTTP, TCP, IP, SSL 等)、远程对象协议 (RMI, RMI IIOP 等)、Java 消息服务 (Java Message Service, JMS)、JavaMail 等。

(3) 组件技术

如图 3-1 所示，不论是何种组件，它们都有相应的容器支持。所以也可以说所有的组件都共同运行在 J2EE Container 中，以获取相应的服务，如图 3-2 所示。

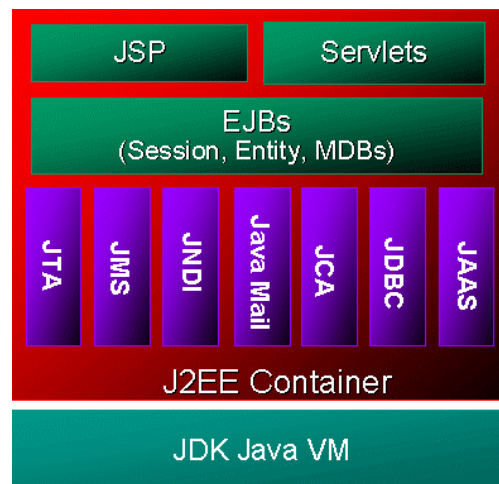


图 3-2 J2EE 组件分布示意图

组件技术将在下一节详细介绍。

3.2 J2EE 组件

J2EE 应用程序是由 J2EE 组件组成的。一个 J2EE 组件是与组件描述中定义的接口保持一致的独立功能软件单元，并且依赖于其运行环境，即容器，为 J2EE 组件提供了必需的基础服务。组件也许是一个单一的类，但也经常是类、接口和资源的集合，它随同它相关的类和文件被装配到 J2EE 应用程序中，并实现与其它组件的通信。J2EE 规范定义的组件主要包括应用客户组件、EJB (Enterprise JavaBean) 组件、WEB 组件。

3.2.1 应用客户组件

J2EE 允许多种类型的客户与服务器端组件进行交互。客户组件主要包括 Applet、客户端应用程序、无线客户等。如图 3-1 所示，它们都运行在应用客户端容器中。应用客户端容器负责所有 Web 程序在客户端构件的运行，Applet 容器可以看作特殊的应用客户端容器。它负责在

Web 浏览器和 Java 插件 (Java Plug-in) 上运行 Java Applet 程序 (Applet 是一种简化并具有安全保护的 Java 小程序), 应用客户端容器和 Applet 程序容器基本对应多层结构中的用户接口层。

Applet 是基于 Java 的小型客户端组件。它一般在 Web 浏览器上运行, 通过 HTTP 协议和服务器进行通信。从服务器传给浏览器的网页可能包括嵌入的 Applet 程序; 这些 Applet 程序在浏览器所安装的 Java 虚拟机 (Java virtual machine) 上执行。

客户应用程序是指在客户机上运行的 J2EE 程序。它为用户提供了丰富的界面 (如 JavaSwing, AWT) 和复杂的操作。客户应用程序直接访问在服务器 EJB 容器内的 EJB 程序。当然, J2EE 客户应用程序也可像 Applet 客户那样以 HTTP 连接和服务器的 Servlet 通信。与 Applet 不同的是, 客户应用程序一般需要在客户端进行安装, 而 Applet 是在 Web 上下载, 无须专门安装。一般来说, 客户应用程序适合在企业内部网中使用, Applet 适合在 WWW 上使用。

无线客户基于移动信息设备定型技术 (Mobile Information Device Profile)。Java 微型版 (Java 2 Micro Edition) 提供了 MIDP 的 API 和有限连接设备配置 (Connected Limited Device Configuration) 技术。这些技术可以使无线设备 (如手机, PDA) 同 J2EE 程序进行通信。

3.2.2 EJB (Enterprise JavaBean) 组件

EJB 组件是 J2EE 的核心, 是运行在服务器端的商业软件。它有三种类型: 会话 bean (session bean), 实体 bean (entity bean) 和消息驱动 bean (message-driven bean)。如图 3-1 所示, EJB 容器负责所有 EJB 的运行, 主要负责数据处理以及和数据库或其它 Java 程序的通信, 对应多层结构的业务层和数据访问层。

其中, 会话 bean 主要用来描述程序的业务逻辑。一个会话 bean 代表 Web 应用程序和客户的一次会话过程。会话 bean 可以是无状态的 (stateless) 或有状态的 (stateful)。无状态是指不管任何用户每次调用其方法, 会话 bean 都作同样响应。有状态是指会话 bean 需要维护和记录不同方法之间的构件状态, 这种分类主要适用不同的数据操作。

实体 bean 是用于表示和维护 Web 应用的数据实体的构件。一个实体 bean 代存放在数据库的一类数据对象。它是数据库内数据在 EJB 容器里的翻版。实体 bean 与会话 bean 不同, 如果一个客户终止使用服务或 J2EE 应用服务器被关闭, EJB 容器会保证实体 bean 的数据保存到数据库内。这就是所谓数据持久性 (data persistence)。实体 bean 根据其实现数据持久性的方法分为 BMP (bean-managed persistence) 和 CMP (container-managed persistence) 两类。BMP 指实体 bean 本身管理对数据库的访问, 这要求编程者自己写一些数据库操作指令 (如 SQL)。CMP 指对数据库的访问由 EJB 容器负责; 编程者只要定义相关设置, 而不需要写数据库操作指令。

消息驱动 bean 实现了客户和服务器更松散的方法调用, 利用消息服务器有其特定的优势, 一个消息驱动 bean 能让客户和服务器之间进行异步 (asynchronous) 通信, 服务器并不要求立刻响应; 当 Java 消息服务器 (Java message server) 收到从客户端发来的消息时, 消息驱动 bean 被激活, 客户并不像使用会话 bean 那样直接调用消息驱动 bean, 这样客户不必要知道消息驱

动 bean 中具体有什么方法可以调用。

这些不同类型的 EJB 组件的运行都离不开 EJB 容器。EJB 容器用于实现企业业务操作的程序，它在多层结构中处于业务层和数据访问层。EJB 组件在 J2EE Web 程序中实现业务逻辑，EJB 组件从客户端或 Web 容器中收到数据并将处理过的数据传送到企业信息系统来存储，EJB 还从数据库检索数据并送回到客户端；由于 EJB 依赖 J2EE 容器进行底层操作，使用 EJB 组件编写的程序具有良好的扩展性和安全性。但是 EJB 对 EJB 容器依赖也给 EJB 的测试带来了难题。

3.2.3 WEB 组件

WEB 组件主要包括包括 servlet, Java Server Page (JSP) 和 Java Server Pages Standard Tag Library (JSTL)。如图 3-1 所示，它们都运行在 WEB 容器中，这些组件主要负责程序和 Web 的通信，这一层对应多层结构中的表示层。

Servlet 是 Java 动态处理 HTTP 请求和生成网页的类 (class)。每个 servlet 就是一个在 J2EE 应用服务器 Web 容器 (又称 Web 服务器) 里的程序构件。这种构件有效地利用了 Web 服务器的 HTTP 通信功能。Web 服务器负责将 Web 请求传递给 servlet。

JSP 可以说是 servlet 的变形，JSP 在 Web 容器内会被自动编译为 servlet，编写 JSP 比编写 servlet 程序更简洁；一个 JSP 文件包括两类成分：生成 HTML 或 XML 模板和处理动态内容的 JSP 元素。JSP 元素主要用于生成动态内容或调用底层 EJB 构件，Servlet 编程将二者混在一起，而 JSP 就清楚多了。

JSTL 将常用的 JSP 功能封装成为简单的标签 (tag)。因为网页是由各种有标签的文字组成的，各种标签 (如表格、字体) 的写法基本固定，于是 JSTL 采用了相似的概念设计 JSP。JSTL 标签是最优化和标准化的，任何种类的 Web 服务器都会支持，这样就省去了不少 JSP 的麻烦；JSTL 目前提供基本的 JSP 功能，仍在不断扩充中。

所有 WEB 组件都是运行在 J2EE Web 容器上的软件程序。Web 容器主要支持多层结构的表示层。它的功能是在 HTTP 协议上对 Web 请求 (request) 进行响应 (response)。这些所谓响应其实就是动态生成的网页。在对 WEB 组件进行单元测试时，就必须考虑如何处理 Web 容器为这些组件提供的对象 (request, response)。

3.3 本章小结

概述了 J2EE 平台、J2EE 服务技术和通讯技术，重点介绍了 J2EE 组件技术，包括应用客户组件、EJB 组件、WEB 组件，并分析了各组件与容器间的关系，容器对组件的影响。

第四章 JUnit 单元测试框架及其扩展框架

JUnit 单元测试框架是 Java 开发人员用于编写单元测试的一个通用的测试框架，可以将测试代码封装入对象中，从而使开发者可以同步设计并配置自己的单元测试。JUnit 单元测试框架可以创建随着时间的流逝仍能保留其价值的测试，使得原作者之外的任何人都能运行这些测试并能解释测试的结果，从不同的作者那里组合测试而不用担心冲突的发生，使得 Java 单元测试更规范有效，并且更有利于测试的集成。

Mock Objects（简称 mocks）是在 JUnit 基础上的扩展的代码逻辑单元测试框架。Mocks 主要对单个类进行测试，避免类之间的真正交互，以使测试粒度最小。

Cactus 扩展并使用了 JUnit，是常用来对 servlet、EJB、Taglib、Filter 等服务器端 Java 代码进行集成单元测试的框架。

4.1 JUnit 单元测试框架

4.1.1 JUnit 单元测试框架的特点

JUnit 框架是一个开放源代码的通用的自动化测试框架，可以将测试代码封装入对象中，使开发者能够同步设计并配置自己的单元测试，并且无需花很多精力就可以掌握。它除了必须的测试代码外，消除不必要的代码，消除重复劳动。

JUnit 框架能够创建随着时间的流逝仍能保留其价值的测试。使得原作者之外的任何人都能运行这些测试并能解释测试的结果，从不同的作者那里组合测试而不用担心冲突的发生。也就是说让测试代码具备保值性，使测试代码的进行方式标准化，可为多人共同浏览、维护。

JUnit 框架能由已有的测试创建出新的测试，可以对于不同的测试重用测试集，将测试代码从系统代码中剥离开，二者可同步发展。

JUnit 框架可以将测试代码与产品代码分开，并易于集成到开发人员的构建过程中。目前支持 JUnit 的 Java IDE（集成开发环境）有很多（如图 4-1 所示），支持 JUnit 的方式有三种：直接支持、集成方式和插件方式。

IDE	集成方式
Forte for Java 3.0 Enterprise Edition	plug-in
Jbuilder 9 Enterprise Edition	integrated with IDE
JDeveloper Integration	plug-in
Visual Age for Java	support
Eclipse	plug-in
TogetherJ	plug-in

图 4 -1 目前支持 JUnit 的 Java IDE

4.1.2 JUnit 的系统结构

JUnit 的系统架构图如图 4-2 所示：

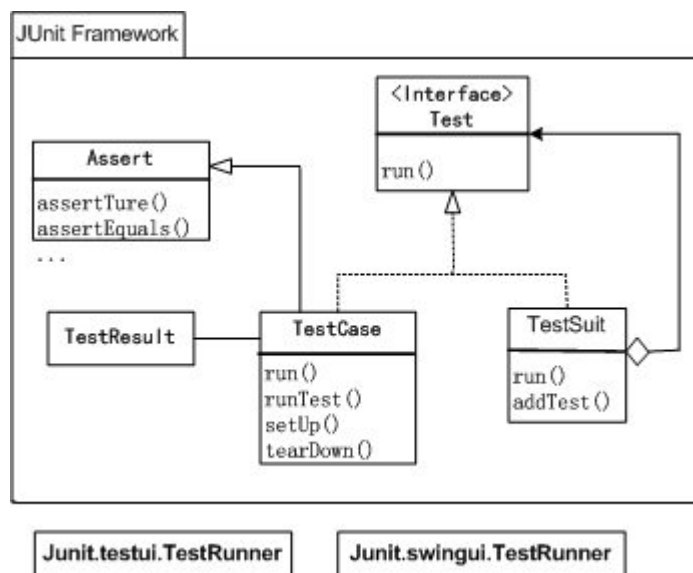


图 4 -2 JUnit 的系统架构图

JUnit 共有七个包，核心的包就是 junit.framework 和 junit.runner。Framework 包负责整个测试对象的构架；Runner 负责测试驱动。

用户编写测试用例一般从继承 TestCase 开始。在 TestCase 子类中重载 setUp() 和 tearDown() 方法，并且定义被测试类的每一个被测试方法的测试方法。一般被测方法标识前加“test”作为测试方法，并定义公有属性。同时在此子类中使用断言（Assert）、测试驱动（TestRunner）、集合测试（TestSuite）和测试输出（TestResult）。

JUnit 有四个重要的类：TestSuite、TestCase、TestResult、TestRunner。前三个类属于 Framework 包，后一个类在不同的环境下是不同的。有三个 TestRunner 类：一个用于文本控制台，另一个用于 Swing，第三个用于 AWT。各个类的功能职责如下：

(1) **TestResult**, 负责收集 **TestCase** 所执行的结果, 它将结果分为两类, 客户可预测的 **Failure** 和没有预测的 **Error**。同时负责将测试结果转发到 **TestListener** (该接口由 **TestRunner** 继承) 处理;

(2) **TestRunner**, 客户对象调用的起点, 负责对整个测试流程的跟踪。能够显示返回的测试结果, 并且报告测试的进度。

(3) **TestSuite**, 负责包装和运行所有的 **TestCase**。

(4) **TestCase**, 客户测试类所要继承的类, 负责测试时对客户类进行初始化, 以及测试方法调用。

另外还有两个重要的接口: **Test** 和 **TestListener**。

(1) **Test**, 包含两个方法: **run()** 和 **countTestCases()**, 它是对测试动作特征的提取。

(2) **TestListener**, 包含四个方法: **addError()**、**addFailure()**、**startTest()** 和 **endTest()**, 它是对测试结果的处理以及测试驱动过程的动作特征的提取。

4.1.3 JUnit 的设计模式

设计模式利用了对对象的继承、组合和代理 (delegation), 在较 OOP 高的层次上考虑问题。尤其是使用代理来对任何不稳定或不确定的方面, 如状态、对象的创建、应用平台等等, 进行封装, 从而保证了源代码的重用和设计的稳定。实际上可以理解成为是 OOP 中虚函数、多态概念的延伸。即 OOP 中的虚函数和多态实现的是方法、对象行为上的多态, 而设计模式的则对创建、结构和高层次的行为进行了多态。

JUnit 系统架构中主要用到三种设计模式, **Template Method**、**Command** 和 **Composite**。JUnit 系统架构的 UML 图表示如图 4-3, 其具体内容如下:

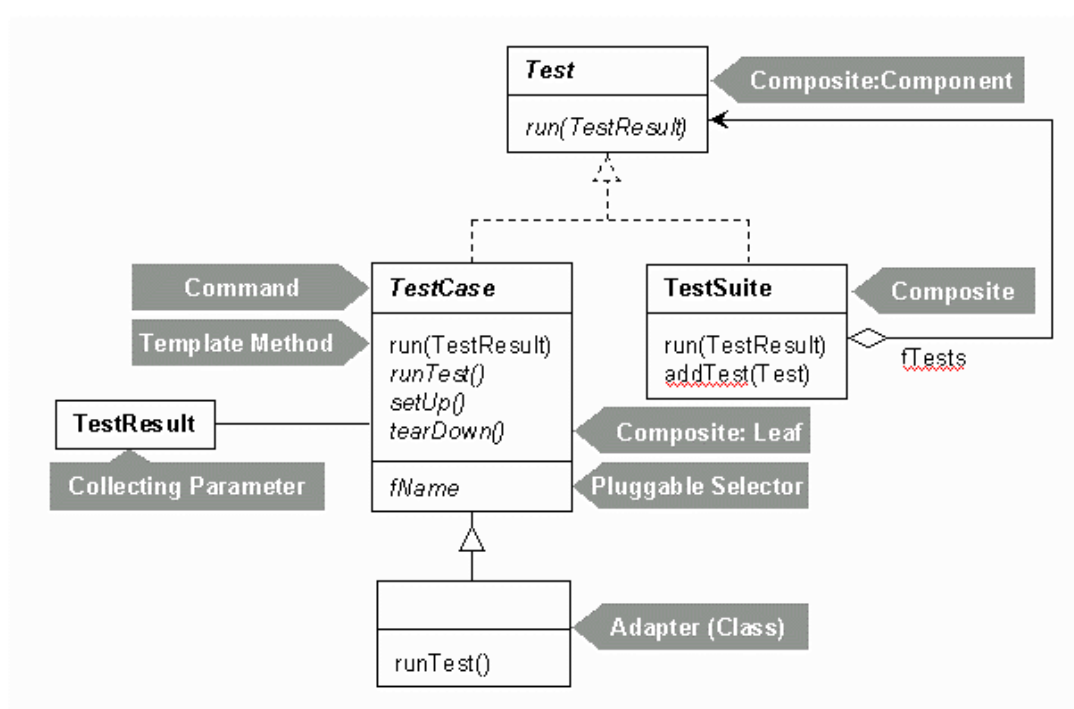


图 4 -3 JUnit 中的模式

(1) **Template Method** (模板方法) 类行为模式, 它的实质就是首先建立方法的骨架, 而尽可能地将方法的具体实现向后推移。Template Method Pattern 使用户无须了解执行框架的过

程细节，而只需重定义特性化的测试预处理，测试单元过程，以及测试完毕相应的三个 `Template Method` 就能使测试正确工作。`TestCase.runBare()`就采用了这种模式，代码如下。客户类均可以重载它的三个方法，这样使得测试的可伸缩性得到提高。

```
Public void runBare() throws Throwable
{
    setUp();
    try{runTest();}
    finally{tearDown();}
}
```

(2) `Command`（命令）对象行为模式，其实质就是将动作封装为一个对象，而不关心动作的接收者。这样动作的接收者可以一直到动作具体执行是才需确定。接口 `Test` 就是一个 `Command` 集，派生出的 `TestCase` 为 `Leaf`，是测试的执行元素，使得不同类的不同测试方法可以通过同一种接口 `Test` 构造其框架结构。

(3) 测试用例的类采用 `Composite` 模式。这样，客户的测试对象就转变成为一个“部分—整体”的层次结构。客户代码仅需要继承类 `TestCase`，就可以轻松的与已有的其它对象组合使用，从而使得单元测试的集成更加方便。`TestSuite` 为 `Composite`，其可以通过 `addTest(Test)`来容纳 `Test` 组合（`TestCase` or `TestSuite`）形成测试包。`Test` 接口与 `TestCase`、`TestSuite` 形成了 `Composite` 结构，`Run(TestResult)`则是 `Composite Methode`。

`TestCase` 可在框架中视为测试单元的运行实体。用户可以通过它派生自定义的测试过程与方式（单元），利用 `Command Pattern` 与 `Composite Pattern` 使其形成可组合装配的可扩展的测试批处理。`TestCase` 本身的运作操作为 `run(TestResult)`，其中分别执行 `setUp()`、`runTest()`、`tearDown()` 来架构测试过程。

4.1.4 JUnit 的实现流程

JUnit 的测试流程构架可以用图 4-4 加以阐述。清楚认识 JUnit 的运行机制，对实际的系统研发大有裨益。

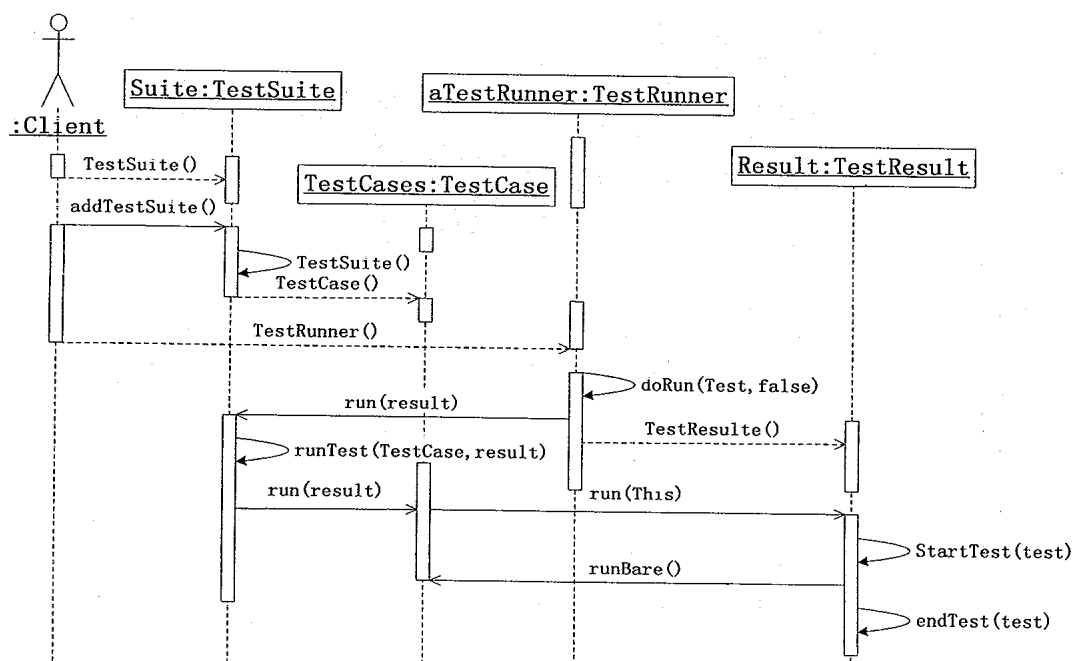


图 4-4 测试序列图

首先,从对象的创建上来分析。客户类负责创建 `Suit` 和 `aTestRunner`。注意,类 `TestRunner` 含有一个静态函数 `Run(Test)`,它自创建本身,然后调用 `doRun()`。客户类调用的一般是该函数,其代码如下:

```
static public void run(Test suit)
{
    TestRunner aTestRunner=new TestRunner();//新建测试驱动
    ATestRunner.doRun(suit, false);//用测试驱动运行测试集
}
```

`Suit` 对象负责创建中夺得测试用例,并将它们包容到本身。客户测试用例继承 `TestCase` 类,它将类,而不是对象传给 `Suit` 对象。`Suit` 对象负责解析这些类、提取构造函数和待测试方法。以待测试方法为单位构造测试用例,测试用例 `fName` 就是代测试方法名。测试结果集由 `aTestRunner` 创建。这似乎同先前阐述的类图有些矛盾,那里阐述了一个测试集可以包含很多个不同的测试驱动,似乎先创建结果集比较理想。显然,这里对测试结果的处理只采用了一种方式,所以这样做同样可行。

其次,从测试动作的执行上来分析,测试真正是从 `suit.run(result)` 开始的。其代码如下:

```
public void run(TestResult result)
{
    //从用例集中获得所有测试用例,分别执行
    for (Enumeration e=tests; e.hasMoreElements();){
        if(result.shouldStop())
```



```
        break;
        Test test=(Test)e.nextElement();
        RunTest(test, result);
    }
}
```

一旦测试用例开始执行，首先使用一个回调策略将自身交由 **Result**。这样做的每一步测试，测试驱动 **aTestRunner** 都可以跟踪处理。这无形中建立了一个庞大的监视系统，随时都可以对发生的事件给与不同等级的关注。

4.1.5 编写 JUnit 的单元测试代码的实践方法和技巧

使用 JUnit 进行测试代码编写的技巧和实践方法：

- (1) 不要用 **TestCase** 的构造函数初始化 **Fixture**，而要用 **setUp()** 和 **tearDown()** 方法。
- (2) 不要依赖或假定测试运行的顺序，因为 JUnit 利用 **Vector** 保存测试方法。所以不同的平台会按不同的顺序从 **Vector** 中取出测试方法。
- (3) 避免编写有副作用的 **TestCase**。例如：如果随后的测试依赖于某些特定的交易数据，就不要提交交易数据，简单的回滚就可以了。
- (4) 当继承一个测试类时，调用父类的 **setUp()** 和 **tearDown()** 方法。
- (5) 将测试代码和工作代码放在一起，同步编译和更新（使用 **Ant** 中有支持 JUnit 的 **task.**）。
- (6) 测试类和测试方法应该有一致的命名方案。如在工作类名前加上 **test** 从而形成测试类名。
- (7) 确保测试与时间无关，不要依赖使用过期的数据进行测试。导致在随后的维护过程中很难重现测试。
- (8) 尽可能地利用 JUnit 提供地 **assert/fail** 方法以及异常处理的方法，可以使代码更为简洁。

为了便于编写测试用例，将编写测试实例中所使用的判定方法进行了归纳，如图 4-5 所示。

assertEquals(期望原型,实际原型)	检查两个原型是否相等
assertEquals(期望对象,实际对象)	利用对象的 equals()方法检查两个对象是否相等
assertSame(期望对象, 实际对象)	检查具有相同内存地址的两个对象是否相等
assertNotSame(期望对象,实际对象)	检查具有不同内存地址的两个对象是否不相等
assertNull(对象 对象)	检查一个对象是否为空
assertNotNull(对象 对象)	检查一个对象是否为非空
assertTrue(布尔条件)	检查条件是否为真
assertFalse(布尔条件)	检查条件是否为假

图 4-5 编写测试实例中所使用的判定方法

4.1.6 JUnit 的自动化执行

每当软件变化并推出一个测试版本,回归测试都需要重新进行,而且回归测试库的测试用例也随软件功能的增加而增多。而且回归测试存在于测试每一个阶段,被全面测试的项目中每一个类至少有一个测试类相对应,开发者不可能每天手动执行整套回归测试。所以,如果不能实现自动化的回归测试,回归测试就有可能无法及时地反馈测试结果,而且频繁而重复的手动测试还会浪费大量的资源。当回归测试自动化之后,它就可以尽可能的与软件变化过程保持同步。一旦软件版本发生了新的改变,无需测试人员的干预,回归测试就可以自动完成,并及时反馈结果。

其次,每个项目都有非常多的源程序文件,所以如果编译每个文件都在 cmd 下键入 `javac ***`,在单元测试的时候也这么做,那么程序员的工作量就会很大。所以必须有一套方法或工具来自动及时地执行这些动作,而不是依赖于已经超负荷工作的人们。

上述这些要求和问题正是 JUnit 所能实现和解决的。当然前提是要遵循 JUnit 的编写规范,并且结合适当的构建工具,以此来实现单元测试部分自动化。

许多开发者已经在使用的工具有: Ant、Maven 和 Eclipse (或其它的集成开发环境)。Ant 是构建 Java 程序的事实标准,它是一个管理及自动化 JUnit 的卓越工具,它能够将源程序和测试实例统一编译并把单元测试的实施统一起来。Maven 扩展了 Ant 的功能来提供更广泛的项目管理的支持,是可以和任何 Java 编程环境配合使用的构建工具。Eclipse 是一个用于开发的功能强大的 IDE,其中集成了 JUnit。

4.2 Mock Objects 测试框架

4.2.1 Mock Objects 的特点

Mock Objects（简称 mocks）是在 JUnit 基础上的扩展的代码逻辑单元测试框架。Mocks 主要对单个类进行测试，避免类之间的真正交互，以使测试粒度最小。Mocks 可以用来代替与被测试代码协作的对象的对象。它将被测试代码中的外部对象替换成模仿对象（mock Objects），来实现真实代码的功能，这些模仿对象被传回原来的被测试代码中，就构成了被测试代码与真实测试环境（如容器）的隔离。Mocks 同样适合把部分代码逻辑的测试同其它的代码隔离开来。因此，Mocks 通过隔离被测试代码之间、被测试代码和测试代码之间的关联程度，来简化测试结构，避免被测试代码因测试环境出现意外，而导致的复杂情况。它可确保一个时间段测试一个代码特性，并且在被测试代码出现问题时，测试人员及时得到通知并解决。由于 mocks 不实现任何逻辑，它们只提供一种使测试能控制仿造类所有业务逻辑方法行为的方法的空壳，所以，mocks 本身是不需要调试的。

使用 mock objects 不仅是一种单元测试的策略，还是一种编写代码的全新方法或是一项设计技术。通过编写 mock objects 进行测试，还会迫使程序员重写部分待测试的代码。因为在实践中，代码会出现写得并不够好、并将不必要的类之间的耦合关系以及和环境的耦合关系写进了代码之中的情况。这样的代码难以在不同的环境中复用，甚至可能会对系统中其它的类产生很大的影响。而通过 mock objects，要想使用它，就必须从不同的角度思考代码，应用更好的设计模式，比如 Interface 和 Inversion of Control（IOC）。Mock objects 正是符合了“测试驱动开发”（TDD，test-driven development）思想的一种实现，即在编写代码之前先编写测试。使用 TDD，就不需要为了可以测试而重构代码，所编写的代码已经就是可以测试的了。

4.2.2 Mock Objects 的测试实现框架和测试步骤

通过项目实践和分析，可以得出一个 Mock Objects 实现与容器隔离来进行测试的技术框架，如图 4-6 所示。

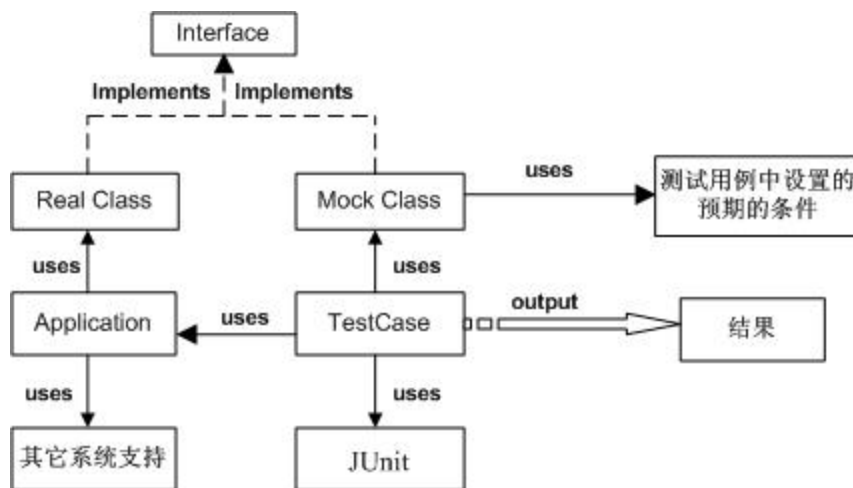


图 4-6 Mock Objects 实现隔离测试的技术框架示意图

从图 4-6 中可以看到模拟接口实现类（Mock Class）与实际接口实现类（Real Class）定义在同一个层次上。实际接口的实现类由应用程序使用，而模拟接口的实现类则模拟实现测试时需要的容器提供的测试运行环境，或需要调用的其它类的方法，以达到为测试提供支持、实现隔离容器测试的目的。在测试过程中，被测试代码可以调用 Mocks 的方法，得到 mocks 传递的结果，该结果是由测试代码设置好的。由于结果就是在测试用例中设置的预期条件，因此，通过设置不同的条件可以较全面地测试不同的情况。例如，对于可能的异常情况，就可以在测试用例中进行适当的设置，然后进行相应的测试。

在不同的情况下和不同的单元测试中，有时会使用真实对象，有时则使用 Mock Objects。在下列情形下使用 Mock Objects 要比使用真实对象更具有优势：

- 真实对象没有确定的行为；
- 真实对象难以配置创建或其一些行为的发生难以控制（比如网络错误）；
- 真实对象令程序的运行速度很慢；
- 真实对象具有（或者本身就是）用户界面；
- 测试需要查询对象，但无法查询真实对象；
- 真实对象尚不存在。

在使用 mock 对象进行测试的时候，总共有 3 个关键步骤，分别是：

- （1）使用一个接口来描述这个真实对象。
- （2）为产品代码实现这个接口。
- （3）以测试为目的，在 mock 对象中实现这个接口。

因为被测试代码只会通过接口来引用对象，所以它完全可以不知道它引用的究竟是真实对象还是 mock 对象。

4.3 Cactus 测试框架

4.3.1 Cactus 的特点

Cactus 是 Apache Software Foundation 的 Jakarta Project 中的一个开放源代码子工程。它是一个常用来对服务器端 Java 代码（即 servlet、EJB、Taglib、Filter 等等）进行集成单元测试的框架^[52]。**Cactus** 支持服务器端代码白盒测试，它扩展并使用 JUnit。JUnit 是 Cactus 分发包附带的，所以不需要去另外下载它的副本。**Cactus** 的 Ant 支持将有助于更容易实现自动化测试。Ant 支持是一个强大的功能，也是 Cactus 和 JUnit 之间的区别所在。**Cactus** 在最初设计和实现时就决定要支持集成单元测试，这种测试是代码逻辑单元测试和功能单元测试两者之间的折衷方案。

Cactus 允许实现一种容器内的策略(包括“进程内”的概念,而且是在一个 JVM 实例之内),从而在真正的容器中对组件进行单元测试,而在产品中这些组件将被部署到该容器中。这也是 **Cactus** 与 **JUnit** 相比所具有的优越性的一个方面。因为无论测试环境如何好,服务器端组件运行的情况都可能与产品模式中(也就是说,在真正的产品级运行时环境中)稍有不同,只有从应用程序服务器容器内进行测试,才能得到准确的测试结果。

Cactus 的测试分为三种不同的测试类别: **JspTestCase**、**ServletTestCase**、**FilterTestCase**, 而 **JUnit** 只有一种 **TestCase**。

4.3.2 Cactus 的测试生命周期和工作原理

Cactus 的测试生命周期如图 4-7 所示。

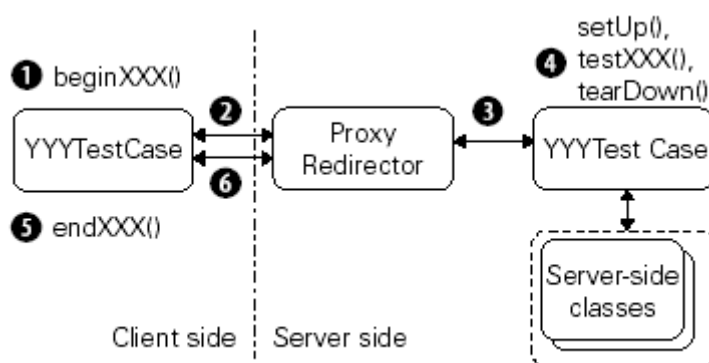


图 4-7 Cactus 的测试生命周期

下面将描述 **Cactus** 的测试生命周期的相关步骤。首先将一个 **Servlet** 应用程序（代码 4.1 见附录）部署到容器中，并且启动了容器。现在可以把用 **Cactus** 编写好的测试（代码 4.2 见附录）提交给 **JUnit test runner**，**runner** 会启动测试。

4.3.3.1 在客户端和服务端端的步骤

Cactus 生命周期的步骤分别在客户端执行和服务端端执行。客户端（**Client side**）指的是启动 **JUnit test runner** 的 **JVM**，服务器端（**Server side**）则是指容器 **JVM**。

在客户端，**Cactus** 逻辑实现于测试所继承的 **YYYTestCase** 类（**YYY** 可以是 **Servlet**、**Jsp** 或者 **Filter**）。更确切地说，**YYYTestCase** 覆写了 **JUnit** 的 **TestCase**，**runBare**，这是 **JUnit test runner** 执行测试所调用的方法。通过覆写 **runBare**，**Cactus** 可以实现自己的测试逻辑。

在服务器端，**Cactus** 逻辑实现于 **proxy redirector**（简称为 **redirector**）。

4.3.3.2 逐步执行测试

对于每项测试（**YYYTestCase** 类中 **TestXXX** 方法）都会执行图 4-7 所示的 6 个步骤。让我们依次来看这 6 个步骤。

第 1 步：执行 beginXXX

如果存在 beginXXX 方法，那么 Cactus 就会执行这个方法。通过这个 beginXXX 方法会把信息传递给 redirector。测试类（代码 4.2 中的 TestSampleServletIntegration 类）必须要继承了 ServletTestCase 并且连接到 Cactus servlet redirector。servlet redirector 被实现为一个 servlet，这是容器的入口。Cactus 客户端通过打开 HTTP 连接来调用 servlet redirector。beginXXX 方法会建立 HTTP 相关参数，这些参数在 servlet redirector 所收到的 HTTP 请求中设置。这个方法可以用来定义 HTTP POST/GET 参数、HTTP cookie 以及 HTTP header 等等。例如：

```
public void beginXXX(WebRequest request)
{
    request.addParameter("param1", "value1");
    request.addCookie("cookie1", "value1");
    [...]
}
```

在 TestSampleServletIntegration 类中，用 beginXXX 方法来告诉 redirector 不要创建 HTTP session（默认行为会创建）：

```
public void beginIsAuthenticatedNoSession(WebRequest request)
{
    request.setAutomaticSession(false);
}
```

第 2 步：打开 redirector 连接

YYYTestCase 打开到 redirector 的连接。在这个例子中，ServletTestCase 代码打开到 servlet redirector（它本身也是个 servlet）的 HTTP 连接。

第 3 步：创建服务器端的 TestCase 实例

Redirector 会创建 YYYTestCase 类的一个实例。请注意，这是 Cactus 创建的第 2 个实例。第一个是在客户端（被 JUnitTestRunner）创建的。之后，redirector 获得容器对象，并通过设置类变量把它们赋给 YYYTestCase 实例。

在 servlet 的例子中，servlet redirector 创建 TestSampleServletIntegration 的实例，并把 HttpServletRequest、HttpServletResponse、HttpSession 等对象通过设置类变量赋给它。Servlet redirector 可以做到这些事情，因为它本身是 servlet。当被 Cactus 客户端调用时，它会获得合法的 HttpServletRequest、HttpServletResponse、HttpSession 等，还会从容器获得其他的对象，并把它们传递给 YYYTestCase 实例。它的行为就像是代理服务器(proxy)或者重定向器(redirector)，并因此而得名。

然后，redirector 启动测试（见第 4 步）。在测试返回后，它会把测试结果连同测试中可能会被抛出的异常一起保存到 ServletConfig **servlet** 对象中，这样稍后就可以获取测试结果。Redirector 需要有个地方来临时保存测试结果，这是因为只有当 endXXX 方法执行完毕（参见第 5 步）后 Cactus 测试才算完整执行完。

第 4 步：在服务器端调用 setUp、testXXX 和 tearDown

如果这个 setUp 方法存在的话，Redirector 会调用 YYYTestCase 的 JUnit setUp 方法。然后，它会调用 testXXX 方法。testXXX 方法会调用被测试的类和方法。最后，redirector 会调用 TestCase 的 JUnit tearDown 方法。

第 5 步：执行 endXXX

一旦客户从到 redirector 的连接获得响应，它就会调用 endXXX 方法（如果这个方法存在的话）。使用这个方法是为了让测试可以对被测代码的结果做一些断言。例如，如果使用 ServletTestCase、FilterTestCase 或者 JspTestcase 类，那么可以对 HTTPcookie、HTTP header 或者 HTTP response 的内容作出断言：

```
public void endXXX(WebResponse response)
{
    assertEquals("value", response.getCookie("cookieName").getValue());
    assertEquals("...", response.getText());
    [...]
}
```

第 6 步：收集测试结果

在第 3 步，redirector 把测试结果存在一个变量中，并保存于 ServletConfig 对象。现在 Cactus 客户端需要测试结果并告诉 JUnit test runner 测试是否成功，以便可以在 Cactus test runner 的 GUI 或者命令行控制台显示结果。为此，YYYTestCase 会打开另一个到 redirector 的连接，并要求获得测试结果。

这个过程看起来可能会觉得很复杂，但是要进入容器内部执行并在容器内部执行测试，这些是必需的。幸运的是，作为 Cactus 的使用者，不必关心这些复杂性，因为 Cactus 框架把一切都包装起来了。可以简单地用所提供的 Cactus runner（前端）来启动和建立测试。

4.3.3 运行 Cactus 的 runner 工具

事实上，运行 Cactus 要比运行单纯的 JUnit 测试复杂，因为需要把代码打包，并且把它部署到容器中，再启动容器，然后再用 JUnit test runner 来启动测试。但是 Cactus 把很多复杂性都隐藏了起来，并且提供了几个 runner（也称作前端）工具，可以通过自动执行几乎上述所有这些不同的步骤来简化测试的执行。图 4-8 展示了目前所有可用的 Cactus runner。其中很多都是 Cactus 团队编写的：Ant 集成、浏览器集成、Eclipse 插件、Jetty 集成和 Maven 插件。

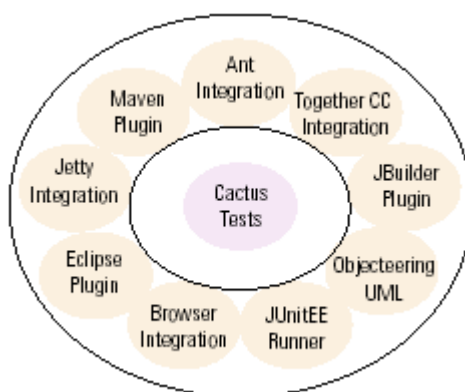


图 4-8 各种运行 Cactus 的 runner

4.4 本章小结

本章首先从多个角度剖析了 JUnit 单元测试框架。在深入理解 JUnit 理论和实现流程的基础上分别详细介绍了 Mock Objects 的特点、实现框架以及 Cactus 的特点、测试生命周期和工作原理。

第五章 J2EE 组件单元测试的方案与实施

5.1 测试方案分析与确定

论文在前面第三章已经详细地介绍和分析了各种类型的 J2EE 组件的特点及其与 J2EE 容器的相互作用的关系,明确了容器对 J2EE 组件单元测试的具体影响。论文在第四章也详尽地描述了 JUnit 框架及其扩展 Mock Objects 和 Cactus 的特点和使用方法。因此,可以说目前为止已经完全明确了单元测试对象的特性,而且掌握了该领域的测试框架,那么如何根据各自不同的特点,对症下药,选择合适的单元测试框架测试不同的 J2EE 组件对象,确定一个较全面有效的测试实施方案呢?依照以往项目中 J2EE 组件单元测试积累的经验,通过对这三个主流单元测试框架不同角度的综合分析比较,就可以找到答案了。

5.1.1 JUnit 测试 J2EE 组件

面向对象的 J2EE 组件的单元测试包含对 jsp、servlet、javabeen、EJB 等组件的测试,而且这些组件都是在服务器端的容器里面运行。但是 JUnit 并没有像其它 J2EE 组件那样被设计成在容器内执行,所以这给测试带来了一些问题和难点。因此,对于普通的 jsp、servlet 用 JUnit 来测试就已经不是那么方便了。虽然直接使用 JUnit 也可以进行 J2EE 组件测试,但是面临着如下几个困难:

(1) 一般 servlet、jsp 或者 EJB 都是运行在服务器上,如果使用 JUnit 测试的话,测试是在客户端,这使得运行环境和测试环境处于不同的系统环境中,这会导致有时会出现不同的测试结果。

(2) 对 EJB 来说,特别是 2.0 版本,很多接口都是 Local Interface,不允许远程调用,没有办法进行分布式测试。

(3) 在一个 EJB 的应用中,一般都会有一些前端应用来访问 EJB,例如: jsp、servlet、javabeen 等。这就意味着需要一个测试框架来测试这些前端组件。而这对 JUnit 来说,并非轻而易举的事情。

当然,由于 MockObjects 和 Cactus 单元测试框架都是在 JUnit 基础上的扩展,所以要进行要深入理解这两个框架就必须全面掌握 JUnit。而且在使用这两个框架进行单元测试时,JUnit 包的支持是必需的。

在对 J2EE 组件中普通的 JavaBean 组件进行测试时,常使用 JUnit 来完成。

5.1.2 Mock Objects 与 Cactus 测试框架的比较分析

作为两种常用的单元测试框架，Mock Objects 和 Cactus 所测试的重点是不同的。

(1) Mock objects 的主要目标是在其它区域对象隔离的情况下进行一个方法的单元测试，执行测试时不需要运行的容器。这一点从另一个角度讲却是对 J2EE 组件单元测试的不足，没有测试容器和组件的交互。而 Cactus 很好地弥补了这一不足，它主要测试服务器端代码，和容器交互，如：Servlet、JSP、EJB、Taglibs，Filters 等。

(2) Cactus 和 Mock Objects 的最大不同是 Cactus 测试倾向于更粗粒度，确保开发的代码可以在容器中运行。它不仅能运行单元测试而且能运行集成测试和某种程度上的功能测试。

(3) Mock objects 是一种综合性的测试框架，它适用于几乎全部的代码逻辑单元测试，如 Servlet、JDBC、Struts 单元测试等等。而 Cactus 一般适用于服务器端代码的测试。但是在 Cactus 测试中可以利用 Mock Objects 模拟一个 JDBC 连接、LDAP 连接等。

(4) 运行 Mock Objects 的测试速度非常快，它不依赖于运行的容器，这种测试可以经常运行。而 Cactus 的执行速度要慢些，它首先需要启动容器，之后运行测试，最后还要停止容器的运行。

(5) Mock Objects 测试通常是比较难写的，因为需要对 mock 的所有调用定义行为。而 Cactus 的配置比较复杂，需要在运行测试之前打包应用程序（通常打包成 war 或 ear），把测试部署到服务器上去，启动服务器，然后才能启动测试。

综合上述分析，图 5-1 列出了从不同角度进行比较的结果。

测试方法	测试粒度	是否需要配置	是否与容器交互	编写的复杂度	运行速度
Cactus	粗	是	是	简单	慢
Mock Objects	细	否	否	复杂	快

图 5-1 Cactus 与 Mock Objects 的比较

因此，针对 J2EE 组件进行单元测试时，问题的重点不在于是否应该使用 Mock Objects 或 Cactus，而是在于该在哪里使用 Mock Objects 和 Cactus。分析后认为，一个好的策略就是要将商业逻辑代码从集成代码（与容器相互作用的代码）中分离出来，然后使用 Mock Objects 测试商业逻辑，进行容器外测试（outside-the-container testing）；使用 Cactus 测试集成代码，进行容器内测试（in-container testing）。图 5-2 旨在说明这种策略。总之，这两种方法不是排斥的，而是互为补充的。明确这一使用技巧，对在实际项目中合理使用测试框架进行单元测试非常重要，进一步确定了 J2EE 组件单元测试方案的正确性。

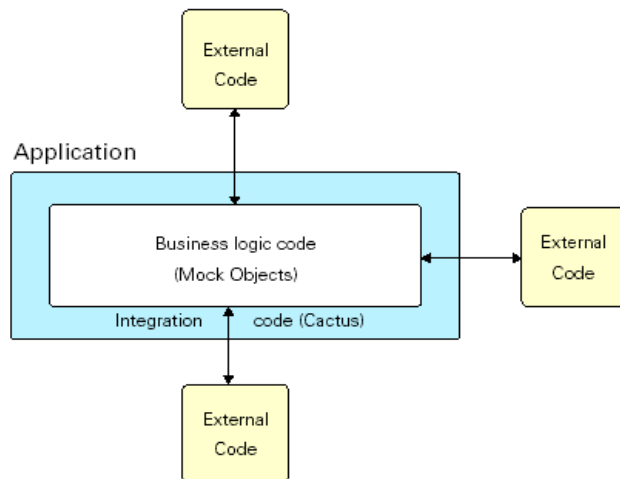


图 5-2 Cactus 和 Mock Objects 的不同适用情况示意图

5.1.3 测试实施方案的确定

综合上述内容，提出以下对 J2EE 组件进行单元测试的方案：采用 MockObjects 进行容器外测试，采用 Cactus 进行容器内测试，再结合 JUnit 对普通 Java 类进行测试。

5.2 方案的实施

5.2.1 实施背景

5.2.1.1 “基于网络的物流配送管理系统平台”简介

“基于网络的物流配送管理系统平台”是科技部“科技型中小企业技术创新基金”支持项目（项目编号：03C26211100786）。该平台可以同时为上百家物流企业提供多品种、多渠道、多种运输工具的物流配送信息化服务。该平台是采用 J2EE 组件应用企业级计算技术，使系统更具可用性和可靠性。系统采用国家物流标准开发物流配送的标准化软件组件，运用智能规划技术，实现对货物、渠道、运输工具、路线及配送中心进行智能规划和布局，实现对配送业务的全程实时监控。“物流配送管理系统平台”由四个子系统组成，分别是“数码仓库系统”、“数码配送系统”、“客户服务系统”和“物流系统调度中心”。

作者本人参与了“基于网络的物流配送管理系统平台”的开发工作，并负责该应用系统的单元测试工作，从而给了笔者一个深入学习软件测试的机会。

5.2.1.2 系统的软件架构

“基于网络的物流配送管理系统平台”的软件体系结构包括门户层、应用层、数据层、网络层（如图 5-3 所示）。其中：

- （1）门户层：包括门户网站、短消息服务等部分，为用户使用应用层提供引导；

(2) 应用层：主要提供各种应用处理，包括信息标准化管理模块、数码仓库业务管理模块、数码配送业务管理模块、短消息处理模块；

(3) 数据层：是系统数据资源中心，包括信息标准化数据库、数码仓库业务数据库、数码配送业务数据库等；

(4) 网络层：在本系统中提供了多种服务方式，因此在网络层涉及到互联网（Internet）、公众电话网（PSTN，public switching Telephone Network）、移动电话网（GSM/CDMA）等三种网络的支持。

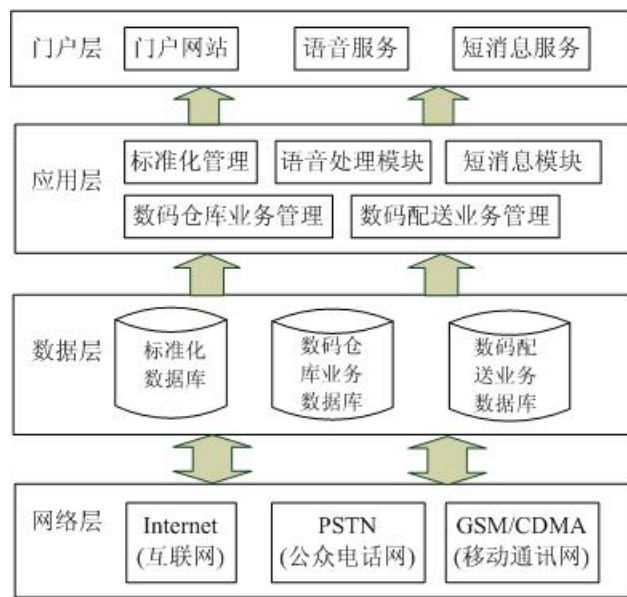


图 5-3 系统软件体系架构

系统采用了 B/S（Browser/Server）模式，进行“基于组件的软件开发”（CBSD，Component Based Software Development）。系统使用 J2EE 架构下的各种 J2EE 组件、服务和通讯技术，如 JSP、Servlet、JavaBean、EJB、JDBC、JMS 等进行开发，数据库选用 Oracle9i。J2EE 体系结构主要包括三层：客户层、中间层（包括 Web 层和业务层）、企业信息系统层。图 5-4 给出了系统实现的 J2EE 框架图。

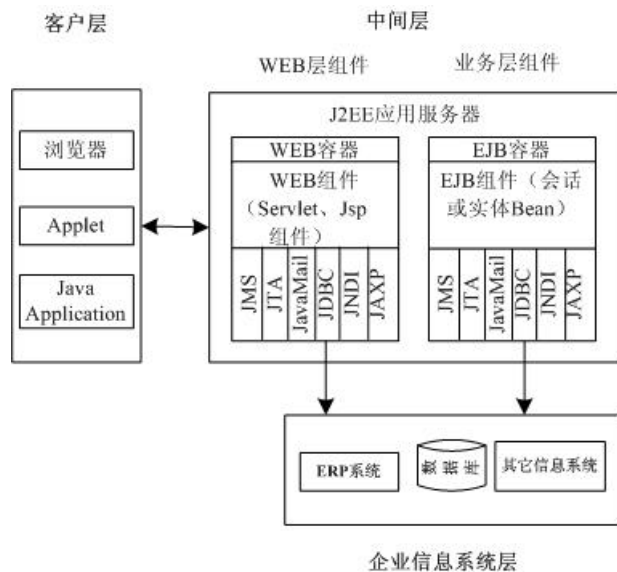


图 5-4 系统实现的 J2EE 框架

由图 5-4 可以看出，对该系统的 J2EE 组件的单元测试包含的内容多，形式复杂，而且很多种组件的测试都与承载各自的容器有关，这点已在第三章中详细说明。这也正是对 J2EE 组件实施单元测试的难点所在。由于篇幅的限制，本文不能将项目中的针对各组件的所有的单元测试用例一一列出，只选择其中几个有代表性的用例来说明 JUnit、Mock Objects、Cactus 三个测试框架是如何具体实施 J2EE 组件单元测试，如何解决测试中遇到的难点的。将分别使用 JUnit 测试 JavaBean 组件，使用 Mock Objects 对 Servlet 组件进行容器外测试，使用 Cactus 对 CMP entity bean 进行容器内测试。后两种类型的测试用例将说明如何处理 J2EE 组件与容器交互带来的问题。

5.2.2 使用 JUnit 进行单元测试

5.2.2.1 单元测试

在通常的测试中，一个单元测试一般针对于特定对象的一个特定特征。整个“基于网络的物流配送管理系统平台”软件系统的一个特定的功能就是采用三级授权机制，进行用户管理。集团公司的系统管理员可以为地区事业部的系统管理员和集团公司用户授权，地区事业部的系统管理员可以为所在辖区的子公司的系统管理员和地区事业部用户授权，子公司的系统管理员只能为子公司的用户授权。系统会根据用户级别（user_level）决定用户的使用功能。在进行角色的身份验证，需要进行数据库的连接。代码 5.1 为一段连接数据库的 javaBean 程序。Java bean 是 J2EE 组件技术的一种形式。

代码 5.1: DatabaseBean.java

.....

```

public class DatabaseBean implements Serializable
{
    public Connection makeConnection() throws DatabaseException
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection(

```

```

        "jdbc: oracle:thin:@192.168.0.6:1521:oracle", "sa", "sa");
    return con;
}
catch(Exception ex)
{throw new DatabaseException("Unable to connect to database "
                             +ex.getMessage());}
}
public void releaseConnection(Connection inCon) throws DatabaseException
{try{inCon.close();}
 catch(SQLException ex)
  { throw new DatabaseException("releaseConnection:"
                                +ex.getMessage());}

  }//releaseConnection
}
.....

```

5.2.2.2 创建测试用例

JUnit 的测试用例由扩展 JUnit 框架的 Java 类编写。这些类拥有大量的方法，每一种方法测试特定的功能或者说代码单元（unit）。通常，编写的测试用例越多，测试的完整性也就越好。应当测试产品功能的全部功能领域：只有这样才能保证系统的局部变动不至于影响到系统的其它部分。

现在就对代码 5.1 中连接数据库的程序编写一个测试用例（如代码 5.2 所示）。

代码 5.2: 数据库连接测试用例（DatabaseTest.java）

```

.....
public class DatabaseTest extends TestCase{
    public DatabaseTest(String name){
        super(name);
    }
    protected void setUp(){
    }
    protected void tearDown(){
    }
    public void testmakeConnection() throws DatabaseException{
        DatabaseBean Dbean=new DatabaseBean();
        try{
            Connection con=Dbean.makeConnection();
            System.out.println("Connection DataSource Success!");
            AssertTrue(true);
        }
        catch(Exception ex )
    }
}

```

```

        { throw new DatabaseException("Unable to connect to database"
                                         +ex.getMessage());
        }
    }
}
.....

```

进行系统的登录需要用户的身份认证，就是要访问后台的数据库中的 userTable，读取该用户的权限，检查该用户是否有使用系统的权限，部分程序代码如下。

代码 5.3：用户权限查询

```

.....
public String QueryRole(String inName, String inPassword)
    throws SQLException, DatabaseException{
    String result= "";
    Con=makeConnection();
    String selectStatement= "select password,   role from "
                            +tableName+"where name=?";
    PreparedStatement prepStmt
                            =con.prepareStatement(selectStatement);
    prepStmt.setString(1, inName);
    ResultSet rs=prepStmt.executeQuery();
    if (rs.next()){
        if (rs.getString(1).equals(inPassword))
            result=rs.getString(2);    }
    prepStmt.close();
    releaseConnection(con);
    return result;
} //QueryRole
.....

```

对于这一段权限查询的代码可以创建如下测试用例。

代码 5.4：用户权限查询测试用例（DatabaseTest.java）

```

.....
public void testQueryRole() throws SQLException, DatabaseException{
    UserBean u=new UserBean();
    String roles;
    ArrayList a;
    String name, password;
    try{
        a=u.GetAllUsers();
        System.out.println("testQueryRole result: ");
        AssertEquals("管理员", u.QueryRole("admin", "admin"));
        System.out.println("username=admin;Role="+u.QueryRole("admin", "admin"));
    }
}

```

```
        catch(Exception ex)
        {
            throw new DatabaseException(ex.getMessge());
        }
    }
    .....
```

5.2.2.3 创建测试包

所谓测试包（test suite）其实就是同一会话中应当执行的测试集合。也就是说可以在 DatabaseTest.java 中添加若干个测试用例，或者在不同的文件里构建测试用例，测试包可以把这些测试组织在一起执行，而不论测试是否在同一个文件里。代码 5.5 显示的测试包其实就是代码 5.2 和代码 5.4 中的两个测试，将 DatabaseTest 这个测试实例中的所有测试方法添加到一个 TestSuite 对象中。测试包构造器会自动装载测试，利用 Java 的内省（reflection）机制来实现，自动找出并调用该类所有的测试方法。

代码 5.5: DatabaseTestRunner.java

```
import junit.framework.Test;
import junit.framework.TestSuite;
import DatabaseTest;
public class DatabaseTestRunner{
    public static Test suite(){
        return new TestSuite(DatabaseTest.class);
    }
    public static void main(String args[]){
        junit.textui.TestRunner.run(suite);}
}
```

5.2.2.4 执行测试

执行测试需要创建可执行类来调用 JUnit 测试运行器。运行器（runner）负责执行测试包，运行所有的测试并输出测试结果，程序中要求使用文本界面（textui）来运行。

在此之前，讲一下 JUnit 的安装，这也是它易用性的一个体现。安装很简单，先到以下地址下载一个最新的 junit.zip 包，下载完以后解压缩到相应的目录下，假设是 JUNIT_HOME，然后将 JUNIT_HOME 下的 junit.jar 包加到系统的 CLASSPATH 环境变量中，对于 IDE 环境，对于需要用到的 junit 的项目增加到 lib 中，其设置不同的 IDE 有不同的设置，这里不再多讲。

执行结果如果没有问题，JUnit 就像代码 5.6 那样输出结果。但是要出现什么问题的话输出结果就可能如清单 A 那样。

代码 5.6: JUnit test


```
...
```

```
Time: 5.395
```

```
OK (3 tests)
```

Time 上的小点表示测试个数，如果测试通过则显示 OK。否则在小点的后边标上 F，表示该测试失败。只有每次的测试结果都应该是 OK 的，才能说明测试是成功的，如果不成功就要马上根据提示信息进行修正了。如果 JUnit 报告了测试没有成功，它会区分失败（failures）和错误（errors）。失败是你的代码中的 assert 方法失败引起的；而错误则是代码异常引起的，例如 ArrayIndexOutOfBoundsException。

```
清单 A: JUnit test errors
```

```
..F.E
```

```
Time : 1.403
```

```
Testsuit:DatabaseTestRunner
```

```
Test run: 3, Failures: 1, Errors:1, Time elapsed: 3.856 sec
```

```
Testcase: testmakeConnection took 2.914sec
```

```
Testcase: testGetAllUsers took 0.32 sec
```

```
FAILED
```

```
null
```

```
junit.framework.AssertionFailedError
```

```
at DataBaseTest.testGetAllUsers(Database.java:42)
```

```
Testcase: testGetAllUsersTestcase: testQueryRole took 0.351 sec
```

```
Caused an Error
```

```
ORA-00923: 未找到预期 FROM 关键字
```

```
Work.excp.DatabaseException: ORA-0923: 未找到预期 FROM 关键字
```

```
at DatabaseException.testQueryRole(Databasetest.java :77)
```

```
Testcase: testQueryRole
```

5.2.2.5 维护单元测试

需求总是变化的，因此单元测试也会根据需求的变化不断的演变。如果修改类的行为规则，可以明确的是，针对这个类的测试单元也会相应的进行修改，以适应变化。但是如果对这个类仅有调用关系的类的行为定义没有变化则相应的单元测试仍然是可靠和充分的，同时如果包含行为变化的类的对象的状态定义与其没有直接的关系，测试单元仍然起效。这种结果也是封装原则的优势体现。

5.2.3 使用 Mock Objects 进行容器外测试

Mock Objects 将被测试代码中的外部对象（即容器对象）替换成模仿对象（Mock Objects），来实现真实代码的功能，这些模仿对象被传回原来的被测试代码中，就构成了被测试代码与容器提供的真实测试环境的隔离。Mocks 同样适合把部分代码逻辑的测试同其它的代码隔离开来。因此，Mocks 通过隔离被测试代码之间、被测试代码和测试代码之间的关联程度，来简化测试结构，避免被测试代码因容器提供的测试环境出现意外，而导致的复杂情况。它可确保一个时间段测试一个代码特性，并且在被测试代码出现问题时，测试人员及时得到通知并解决。我们可以借助一些能够自动生成 mock objects 的框架来进一步简化编写工作，如 EasyMock 等。

EasyMock 是现在最著名的 mock objects 生成器之一。它是用 Java 动态代理（dynamic proxies）实现的，所以可以透明地在运行时生成 mock objects。

测试方案确定采用 Mock Objects 对 J2EE 组件进行容器外测试，这里以 servlet 的单元测试为例来具体说明测试方法和过程。

下面从本系统 servlet 应用程序中选取一个作为例子 SampleServlet sevlet，并对它的 isAuthenticated 方法（代码 5.6）进行隔离的单元测试。

代码 5.6: SampleServlet.java

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
public class SampleServlet extends HttpServlet
{
    public boolean isAuthenticated(HttpServletRequest request)
    {
        HttpSession session = request.getSession(false);
        if (session == null)
        {
            return false;
        }
        String authenticationAttribute =
            (String) session.getAttribute("authenticated");
        return Boolean.valueOf(
            authenticationAttribute).booleanValue();
    }
}
```

为了能够测试这个方法，需要得到一个合法的 HttpServletRequest 对象。然而，不可能调用 new HttpServletRequest 来创建一个可用的请求。HttpServletRequest 的生命周期是由容器管理的。无法单独使用 JUnit 为 IsAuthenticated 方法编写测试。于是，试着使用 EasyMock 框架，对其进行单元测试代码的编写，为了便于对代码的分析说明，在相应的代码行后增加了编号，如下所示：

代码 5.7: TestSampleservlet.java

```
...
import org.easymock.MockControl;
import junit.framework.TestCase;
public class TestSampleServlet extends TestCase
{
    private SampleServlet servlet;
    private MockControl controlHttpServlet; // 1
    private HttpServletRequest mockHttpServletRequest; // 2
```

```
private MockControl controlHttpSession; // 3
private HttpSession mockHttpSession; // 4
protected void setUp()
{
    servlet = new AuthServlet();
    controlHttpServlet = MockControl.createControl(HttpServletRequest.class); //5
    mockHttpServletRequest =
        (HttpServletRequest) controlHttpServlet.getMock(); //6
    controlHttpSession = MockControl.createControl(HttpSession.class); //7
    mockHttpSession =
        (HttpSession) controlHttpServlet.getMock(); //8
}
protected void tearDown()
{
    controlHttpServlet.verify(); //9
    controlHttpSession.verify(); //10
}
public void testIsAuthenticatedAuthenticated()
{
    mockHttpServletRequest.getSession(false); //11
    controlHttpServlet.setReturnValue(mockHttpSession); //12
    mockHttpSession.getAttribute("authenticated"); //13
    controlHttpSession.setReturnValue("true"); //14
    controlHttpServlet.replay(); //15
    controlHttpSession.replay(); //16
    assertTrue(servlet.isAuthenticated(mockHttpServletRequest));
}
public void testIsAuthenticatedNotAuthenticated()
{
    mockHttpServletRequest.getSession(false); //17
    controlHttpServlet.setReturnValue(mockHttpSession); //18
    mockHttpSession.getAttribute("authenticated"); //19
    controlHttpSession.setReturnValue(null); //20
    controlHttpServlet.replay(); //21
    controlHttpSession.replay(); //22
    assertFalse(
        servlet.isAuthenticated(mockHttpServletRequest));
}
public void testIsAuthenticatedNoSession()
{
    mockHttpServletRequest.getSession(false); //23
    controlHttpServlet.setReturnValue(null); //24
    controlHttpServlet.replay(); //25
```

```
controlHttpSession.replay();//26
assertFalse(
    servlet.isAuthenticated(mockHttpServletRequest));
}
```

对代码 5.7 简要分析如下，以代码行编号来标示对应的代码：

1-8: 创建了两个 mock objects，一个是 HttpServletRequest，另一个是 HttpSession。EasyMock 通过为 mock 创建动态代理来工作，动态代理使通过 MockControl 控制对象来控制的。

9-10: 让 EasyMock 来校验确实调用了被模拟的方法。如果定义一个 mock 方法但是没有被调用到，那么 EasyMock 就会报错。如果没有为一个方法定义任何行为，但史它被调用到了，那么 EasyMock 也会报错，这是在 JUnit 的 tearDown 方法中来做，这样就可以对所有测试自动执行校验。因为基于这种“针对所有测试”的工作分配方式，所以必须在 testIsAuthenticatedNoSession 中激活 HttpSession mock object，哪怕在这项测试中用不到它。使用 EasyMock，mock objects 只有被激活后才能被校验。

11-12: 使用控制对象告诉 HttpServletRequest mock object，当 getSession(false)被调用时这个方法该返回一个 HttpSession mock object。

13-14, 17-20, 23-24: 和前一步一样，告诉 mock object 当这些相应的方法被调用时该怎么做，并告诉控制对象这些对象该返回什么。这是 EasyMock 的训练模式。

15-16, 21-22, 25-26: 一旦激活了这里的 mock object，就走出了这种训练模式。调用 mock object 会返回预期的结果。

最后在 Eclipse 中执行这个 TestSampleServlet TestCase 的结果如图 5-5 所示。

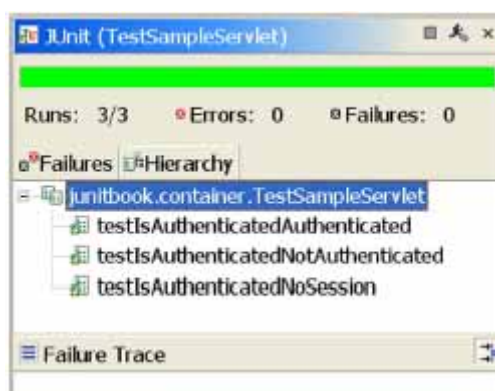


图 5-5 执行 TestSampleServlet TestCase 的结果

这种 Mock objects 方法最大的优点就是，执行测试是不需要运行的容器。可以很快就建立起测试，而且测试运行的速度也很快。然而，用 mock objects 来对 J2EE 组件进行单元测试的缺点是：

(1) 它们没有测试容器和组件的交互，而这是代码的重要部分。

(2) 它们没有测试组件的部署部分。

(3) 为了建立测试，必须要知道被模拟的 API 行为是怎么样的。要知道自己写的 API 的行为比较容易。但要了解外部 API，比如 Servlet API 就没那么容易了。

(4) 无法让你完全确信代码会在目标容器中正常运行。

这些相对的缺点在用 Cactus 实施组件单元测试时会迎刃而解。

5.2.4 使用 Cactus 进行容器内测试

EJB 的测试一直是以它的难度著称的，其中的一个主要原因是 EJB 试运行于 EJB 容器之内的组件。因此，要么必需用自己的代码来抽象出容器的服务，要么就执行容器内测试。所以，可以用 Mock Objects，也可以用 Cactus 来编写对 EJB 的单元测试。但是因为 Cactus 本身就可以提供容器对象，这样就可以既执行单元测试，同时也可以确保代码能在目标容器内正确运行，使编写单元测试更简单更快速。

由于 Cactus 运行于容器内部，所以可以使用本地接口来对 EJB 进行单元测试。虽然会因为必须得编写容器的启动和停止而带来比较复杂的 Ant 脚本，但由于 Cactus 提供了 Ant 的集成，简化了这些操作。

测试方案确定采用 Cactus 对 J2EE 组件进行容器内测试，这里以 EJB 的单元测试为例来具体说明测试方法和过程。

在本节就使用 Cactus 来测试一个容器管理持久化 (CMP) entity bean。

5.2.4.1 用 Cactus 编写 EJB 单元测试

下面的代码出自一个用于定制职业服务相关信息的 OrderEJB 的 CMP entity bean 代码（完整的代码见附录）。

代码 5.8 OrderEJB.java

```
...
public abstract class OrderEJB implements EntityBean
{
    ...
    public OrderLocal ejbCreate(Date orderDate, String orderItem)
        throws CreateException
    {
        int uid = 0;
        // Note: Would need a real counter here. This is a hack!
        uid = orderDate.hashCode() + orderItem.hashCode();
        setOrderId(new Integer(uid));
        setOrderDate(orderDate);
        setOrderItem(orderItem);
    }
}
```

```

        return null;
    }
    public void ejbPostCreate(Date orderDate, String orderItem)
        throws CreateException {}
    ...
    public void setEntityContext(EntityContext context)
        throws EJBException, RemoteException {}
    public void unsetEntityContext()
        throws EJBException, RemoteException {}
}

```

接下来就可以对 `ejbCreate` 方法进行单元测试。Cacus 的解决方案中透明地使用了 `Servlet Redirector`，在 web 容器的环境中执行测试，对于本地接口也照样可以运行。用 Cacus 编写的单元测试代码如下。

代码 5.9 作为 Cactus ServletTestCase 的 OrderEJB 单元测试：

```

...
public class TestOrderEJB extends ServletTestCase
{
    public void testEjbCreateOk() throws Exception
    {
        OrderLocalHome orderHome =
            (OrderLocalHome) new InitialContext().lookup(
                JNDINames.ORDER_LOCALHOME); // ①

        Date date = new Date();
        String item = "item 1";
        OrderLocal order = orderHome.create(date, item);
        assertEquals(date, order.getOrderDate());
        assertEquals(item, order.getOrderItem());
        assertEquals(new Integer(date.hashCode()
            + item.hashCode()), order.getOrderId());
    }
}

```

因为 Cactus 测试在容器内进行，所以不必在调用 `new InitialContext()` (①) 前设置 JNDI 的属性。但是，如果使用纯 JUnit 的解决方案，则需要增加如下的步骤：

- (1) 因为要进行 JVM 外的容器调用，所以需要制定 JNDI 的属性。
- (2) 执行一个 JNDI 查找，须获取 OrderEJB 的 home 实例。

相对应①的代码段如下：

```

...
Properties props = new Properties();

```

```

props.put(Context.INITIAL_CONTEXT_FACTORY,
    "org.jnp.interfaces.NamingContextFactory");
props.put(Context.PROVIDER_URL, "localhost:1099");
props.put(Context.URL_PKG_PREFIXES,
    "org.jboss.naming:org.jnp.interfaces");
InitialContext context = new InitialContext(props); //在调用 new InitialContext()前,指定 JNDI
                                                    的属性。

Object obj = context.lookup(JNDINames.ORDER_HOME);
OrderHome orderHome=
    (OrderHome) PortableRemoteObject.narrow(
        obj, OrderHome.class); // 获取 OrderEJB 的 home 实例。

...
    
```

5.2.4.2 定义 cactus 测试的目录结构

编写好代码后,就可以考虑应用 Ant 运行这个测试用例了,以达到测试执行的自动化。Ant 是一种基于 Java 的 build 工具,它通过基于 XML 的配置文件,调用 target 树,来执行各种 task。下面是 cactus 测试的目录结构,如图 5-6。其中 conf/cactus 目录用于存放 Cactus 测试的配置文件,而 src/testcactus/ 目录用于存放单元测试代码。目录中还需要添加 Ant 构建文件 build-cactus.xml,使得通过使用 Cactus 的 Ant 自动执行 Cactus 测试的 Ant target。同时使用 Ant 脚本 build.xml (代码 5.10 见附录) 程序打包成 ear 文件。



图 5-6 Cactus 测试的目录结构

5.4.3 打包 Cactus 测试

因为 Cactus 测试是在 web 容器中进行的,所以需要在 ear 中将它们打包成一个 war。Cactus

提供了一个 Ant 任务——cactifywar，可以很简单地创建这个 war：

```
<cactifywar version="2.3" destfile="${target.dir}/cactus.war"
  mergewebxml="${conf.dir}/cactus/web.xml">
  <classes dir="${target.classes.cactus.dir}"/>
</cactifywar>
```

其中 version 属性指定了使用 Servlet API 2.3 构建 war。按照 J2EE 规范的要求，在 war web.xml 文件（代码 5.11 见附录）中需要一个<ejb-local-ref>，包含 ejb-local-ref 入口，用于调用 OrderEJB。。

打包的最后一步是在 ear 中将 war 打包：

```
<ear update="true" destfile="${target.dir}/ejb.ear"
  appxml="${conf.dir}/cactus/application.xml">
  <fileset dir="${target.dir}">
    <include name="cactus.war"/>
  </fileset>
</ear>
```

同时向 application.xml（代码 5.12 见附录）中添加 Cactus war 定义。

然后编写完整的 Ant 脚本 ejb-jar.xml（代码 5.13 见附录）。

5.4.4 执行 Cactus 测试

这里将使用 **Cactus** 提供的 cactus 任务来执行 Cactus 测试。cactus 任务可以完成：配置 ear，启动容器，执行测试，并停止容器。cactus 任务扩展自 JUnit Ant 任务 junit，因此也就继承了 junit 的所有功能。代码 5.14 示范了如何使用 cactus 任务来自动运行对 TestOrderEJB 的测试。

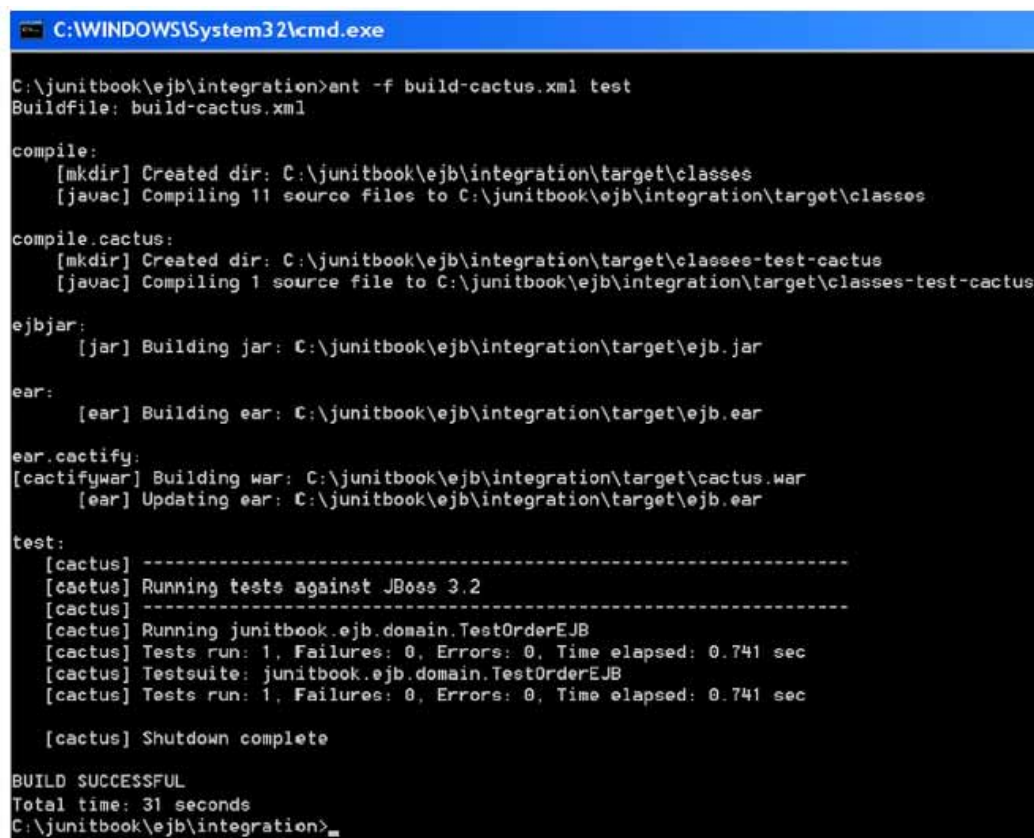
代码 5.14：

```
<?xml version="1.0"?>
<project name="Ejb" default="test" basedir=".">
[...]
  <target name="test" depends="ear.cactify">
    <cactus earfile="${target.dir}/ejb.ear" fork="yes"
      printsummary="yes" haltonerror="true"
      haltonfailure="true">
      <containerset>
        <jboss3x dir="${cactus.home.jboss3x}"
          output="jbossresult.txt"/>
      </containerset>
      <formatter type="brief" usefile="false"/>
      <test name="junitbook.ejb.domain.TestOrderEJB"/>
      <classpath>
        <pathelement location="${target.classes.java.dir}"/>
      </classpath>
    </cactus>
  </target>
</project>
```



```
<pathelement location="${target.classes.cactus.dir}"/>
</classpath>
</cactus>
</target>
</project>
```

通过键入 `ant -f build-cactus.xml test` 执行测试，得出的结果如图 5-7 所示。



```
C:\WINDOWS\System32\cmd.exe
C:\junitbook\ejb\integration>ant -f build-cactus.xml test
Buildfile: build-cactus.xml

compile:
[mkdir] Created dir: C:\junitbook\ejb\integration\target\classes
[javac] Compiling 11 source files to C:\junitbook\ejb\integration\target\classes

compile.cactus:
[mkdir] Created dir: C:\junitbook\ejb\integration\target\classes-test-cactus
[javac] Compiling 1 source file to C:\junitbook\ejb\integration\target\classes-test-cactus

ejbjar:
[jar] Building jar: C:\junitbook\ejb\integration\target\ejb.jar

ear:
[ear] Building ear: C:\junitbook\ejb\integration\target\ejb.ear

ear.cactify:
[cactifywar] Building war: C:\junitbook\ejb\integration\target\cactus.war
[ear] Updating ear: C:\junitbook\ejb\integration\target\ejb.ear

test:
[cactus] -----
[cactus] Running tests against JBoss 3.2
[cactus] -----
[cactus] Running junitbook.ejb.domain.TestOrderEJB
[cactus] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.741 sec
[cactus] Testsuite: junitbook.ejb.domain.TestOrderEJB
[cactus] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.741 sec

[cactus] Shutdown complete

BUILD SUCCESSFUL
Total time: 31 seconds
C:\junitbook\ejb\integration>
```

图 5-7 用 Cactus/Ant 执行 Cactus 测试 EJB 的结果

5.3 本章小结

本章首先通过对三个框架优缺点的对比分析，总结说明了如何合理使用 JUnit 单元测试框架及其扩展框架 Mock Objects 和 Cactus，解决论文提出的问题和难点，进而确定了使用组合框架进行 J2EE 组件单元测试的实施方案。然后通过“基于网络的物流配送管理系统平台”中几个有代表性的 J2EE 组件单元测试用例应用了该实施方案。这几个测试用例分别说明了如何使用 JUnit 测试 JavaBean 组件，如何使用 Mock Objects 对 Servlet 组件进行容器外测试以及如何使用 Cactus 对 EJB 组件进行容器内测试，并解决了因 J2EE 组件与容器交互而带来的单元测试的问题和难点。

第六章 总结与展望

自确定论文方向后，经过一年多的学习、分析、研究与实践，取得了一定的成果。下面把本人所做的工作和论文解决的问题作一个总结，并提出论文的进一步研究方向。

6.1 总结

本文在研究面向对象软件测试的理论和技术的基礎上，指出了 J2EE 组件与容器交互的特点，提出了 J2EE 组件单元测试面临的问题和难点。通过深入研究和分析组合框架，即 JUnit 及其两个扩展框架 Mock Objects 与 Cactus，给出了结合三个框架的各自优点，进行 J2EE 组件单元测试的综合性实施方案，解决了 J2EE 组件单元测试的问题和难点，并将其成功应用于实际项目“基于网络的物流配送管理系统平台”的单元测试中。

通过研究和应用，得出的主要结论如下：

(1) 以传统的软件开发方法为背景发展起来的测试技术，已不能适用于面向对象软件的测试，面向对象中的继承、多态、动态绑定等机制对面向对象软件的测试产生了深刻的影响，面向对象的测试策略和测试方法都必须进行相应的变革。

(2) J2EE 组件单元测试的难点主要在于测试时如何处理 J2EE 组件和容器之间交互作用带来的复杂性。

(3) JUnit 作为主流的单元测试框架，虽然能够很好地实现普通 Java 类的单元测试，但是对于有关容器的 J2EE 组件的测试，还存在很多的局限性。它的两个扩展框架 Mock Objects 与 Cactus 弥补了它的不足，为 J2EE 组件测试提供了好的策略。

(4) Mock Objects 与 Cactus 具有互补的特点，采用 Mock Objects 与 Cactus 结合，分别对组件进行容器外和容器内测试的方案可以更好地解决 J2EE 组件单元测试的难点。

(5) 目前，国内对 J2EE 组件单元测试的研究尚处于起步阶段。本文采用的基于 JUnit 框架及其扩展框架 Mock Objects 和 Cactus 的综合测试方案，实现了 J2EE 组件单元测试的自动化和测试过程的可重复性。通过在“基于网络的物流配送管理系统平台”的单元测试中的应用，证明了该方案的实用价值和可行性。

6.2 展望

近几年来，单元测试越来越受到重视，在 XP 等轻量级的开发过程中尤为显得重要。本文对基于 JUnit 框架及其扩展框架的 J2EE 组件单元测试进行了研究分析，提出了一个综合测试方案，并在实际项目中得到应用。但是，由于时间和精力所限，该方案仍有存在着一定的不足，有待于在今后工作中进一步研究：

(1) 本文提出的方案对编程人员或测试人员的要求较高，需要掌握的框架工具比较多，使得这个方案的推广有一定的难度。

(2) JUnit 是一个结构优良的基础框架，对它的使用还仅限于中级阶段，完全可以在这个框架的基础上灵活的扩展适合自己的测试工具，就像 mock objects、Cactus 等一样，具备各自的特点，通过与其它工具的结合，实现单元测试更全面的自动化。

参考文献

- [1] 余梦非. 软件测试期待更多关注. 中国计算机报, 2003, 9 (65)
- [2] PRESSMAN RS. 软件工程: 实践者的研究方法[M]. 黄柏素, 梅宏, 译. 北京: 机械工业出版社, 1999.
- [3] 周峰. 如何做好单元测试. 无忧软件测试网: <http://www.51testing.com>
- [4] 杨芙清, 王千祥, 梅宏, 等. 基于复用的软件生产技术. 中国科学(E 辑), 2001, 31 (4): 363—371
- [5] Ma Y2S, Oh S2U, Bae D2H, et al. Framework for Third Party Testing of Component Software. In: Proceedings of the 8th Asia2Pacific Software Engineering Conference, Macau SAR: 2001. 4312434
- [6] Voas J M. Certifying Off2the2shelf Software Components. IEEE Computer, 1998, 31 (6): 53259
- [7] Edwards S H, Shakir G, Sitaraman M, et al. A Framework for Detecting Interface Violations in Component2Based Software. In: Proceedings of the 5th International Conference on Software Reuse, British Columbia: 1998. 46255
- [8] Martins E, Toyota C M, Yanagawa R L. Constructing Self2Testable Software Components. In: Proceedings of the International Conference on Dependable Systems and Networks, Goteborg: 2001. 1512160
- [9] Yoon H, Choi B. Inter2Class Test Technique between Black2Box2Class and White2Box2Class for Component Customization Failures. In: Proceedings of the 6th Asia2Pacific Software Engineering Conference, Takamatsu: 1999. 1622165
- [10] Vitharana P, Jain H. Research Issues in Testing Business Components. Information & Management, 2000, 37 (6): 2972309
- [11] Wu Y, Pan D, Chen M2H. Techniques for Testing Component2Based Software. In: Proceedings of the 7th International Conference on Engineering of Complex Computer Systems, Los Alamitos: 2001. 2222232
- [12] Edwards S H. A Framework for Practical, Automated Black2Box Testing of Component2Based Software. Software Testing, Verification and Reliability, 2001, 11 (2): 972111
- [13] Deveaux D, Frison P, Jezequel J2M. Increase Software Trustability with Self2testable Classes in Java. In: Proceedings of the 2001 Australian Software Engineering Conference. Canberra: 3211
- [14] Beydeda S, Gruhn V. An Integrated Testing Technique for Component2Based Software. In: Proceedings of the ACSP IEEE International Conference on Computer Systems and Applications, Beirut: 2001. 3282334
- [15] Jin Z, Offutt J. Deriving Tests from Software Architectures. In: Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong: 2001. 3082313
- [16] Orso A, Harrold MJ, Rosenblum D, et al. Using Component Metacontent to Support the Regression Testing of Component2Based Software. In: Proceedings of the International Conference on Software Maintenance, Florence: 2001. 7162725
- [17] Yoon H, Choi B. An Effective Testing Technique for Component Composition in

- EJBs. In : Proceedings of the 8th Asia2Pacific Software Engineering Conference , Macau SAR: 2001. 2292236
- [18] Lee S C , Offutt J . Generating Test Cases for XML2Based Web Component Interactions using Mutation Analysis. In : Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong : 2001. 2002209
- [19] Myer, GJ..The Art of Software Testing .NY: Wiley , 1979
- [20] Dijkstra, EW. “Structured Programming”, in J.N.Buxton and B. Randell(eds), Software Engineering Techiques, Brussels, Belgium: NAO Science Committee, 1970
- [21] Hunphrey, W.S..Managing the Software Process, Reading, MA: Addison-Wesley, 1989
- [22] Roger S.Pressman.Software Engineering-A Practitioner’s Approach, 1998
- [23] 周涛. 航天型号软件测试. 北京: 宇航出版社, 1999
- [24] IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990 by IEEE, Inc.
- [25] C++Test 介绍, [http: //www.superst.com/para_ctest.htm](http://www.superst.com/para_ctest.htm)
- [26] Ronald J. Norman. Objected-Oriented analysis and design, Prentice Hall, 1996.
- [27] 王燕. 面向对象的理论与实践. 北京: 清华大学出版社, 1998
- [28] Stephen R. Schach. Object-Oriented and classical software engineering, 2002
- [29] 张毅坤. 面向对象软件测试的特点及方法[J] . 西安理工大学学报, 2002 , 18(4) : 3612365.
- [30] Perry D. E. , Kaiser G.E..Adequate testing and Objected-Oriented Programming, Journal of Object-Oriented Programming, 1990
- [31] Weyuker E.J.. Axiomatizing software test data accuracy, IEEE Transanction on Software Engineering , 1986
- [32] Bashir I. , Goel A. L. . Testing of Objected-Oriented Software: A combinatorial approach, Technical Report, CASE Center, Syracuse University, USA, 1993
- [33] Binder RV. Testing Object_Oriented Systems : A Status Report , American Programmer, 1994
- [34] Barbey S., Strohmeier A..The problematics of testing object_oriented software, Building Quality into Software, 1995
- [35] Doong R., Frankle P.G.The ASIOOI approach to testing objected-oriented programs.ACM Transactions on Software Engineering and Methodology, 1993
- [36] Seigel S..Object_Oriented integration testing .NY: John Wiley&Sons, Inc., 1996
- [37] Weyuker E J . Axiomatizing Software Test Data Accuracy[J] . IEEE Trans on Software Engineering , 1986 , SE212 (12) : 1 12821 138.
- [38] Perry D E , Kaiser G E. Adequate Testing and Object2oriented Programming[J] . Journal of Object2oriented Programming , 1990 , 2(5) : 13219.
- [39] John D , McGregor , David A , et al.面向对象的软件测试[M].北京: 机械工业出版社, 2002.
- [40] Robert V Binder. State2based Testing : Sneak Paths and Conditional Transitions[J] . Object Magazine , 1995, 5(6) : 87289.
- [41] Jeff Langer. Evolution of Test and Code Via Test-First Design, 2001
[http: //objectmentor.com/resources/articles/tfd.pdf](http://objectmentor.com/resources/articles/tfd.pdf).

-
- [42] <http://www.mockobjects.com>
- [43] <http://jakarta.apache.org/cactus/>
- [44] <http://www.httputil.org>
- [45] Paul J.Perrone, etal.著, 张志伟 谭郁松 张明杰译: J2EE 构建企业系统专家级解决方案[M], 北京, 清华大学出版社, 2001
- [46] 陈华军: J2EE 构建企业级应用解决方案[M], 北京, 人民邮电出版社, 2002 年 3 月
- [47] Dives, A..201 Principles of Software Development, McGraw-Hill, 1995
- [48] Michel Casabianca . A JUnit Step-by-Step. May/June 2003
http://www.oracle.com/technology/oramag/oracle/03-may/o33junit_2.html .
- [49] Malcolm Davis. Incremental development with Ant and JUnit. Nov 2000
<http://www-128.ibm.com/developerworks/java/library/j-ant/?n-j-11160>
- [50] David Gallardo, 任何人都可以重构. Nov 2003 .
http://www-900.ibm.com/developerWorks/cn/linux/opensource/os-ecref/index_eng.shtml
- [51] Vincent Massol. JUnit in Action.北京: 电子工业出版社. 2005
- [52] 王东刚. 软件测试与 JUnit 实践.北京: 人民邮电出版社. 2004
- [53] Christopher Alexander.etc .A Pattern Language.NY: Oxford University Press.1997
- [54] Erich Gamma. 设计模式-可复用面向对象软件的基础. 北京: 机械工业出版社.
- [55] Miller G.A..The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.Psychological Review, 1963
- [56] <http://www.junit.org>

附录

(1)代码 4.1: **SampleServlet.java**(待进行单元测试的 servlet 示例方法)

```
package junitbook.container;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
public class SampleServlet extends HttpServlet
{
    public boolean isAuthenticated(HttpServletRequest request)
    {
        HttpSession session = request.getSession(false);
        if (session == null)
        {
            return false;
        }
        String authenticationAttribute =
            (String) session.getAttribute("authenticated");
        return Boolean.valueOf(
            authenticationAttribute).booleanValue();
    }
}
```

(2)代码 4.2 : **TestSampleServletIntegration.java**(用 Cactus 来对 servlet 代码进行单元测试)

```
package junitbook.container;
import org.apache.cactus.ServletTestCase;
import org.apache.cactus.WebRequest;
public class TestSampleServletIntegration extends ServletTestCase
{
    private SampleServlet servlet;
    protected void setUp()
    {
        servlet = new SampleServlet();
    }
    public void testIsAuthenticatedAuthenticated()
    {
        session.setAttribute("authenticated", "true");
        assertTrue(servlet.isAuthenticated(request));
    }
    public void testIsAuthenticatedNotAuthenticated()
    {
        assertFalse(servlet.isAuthenticated(request));
    }
}
```

```
}  
public void beginIsAuthenticatedNoSession(WebRequest request)  
{  
    request.setAutomaticSession(false);  
}  
public void testIsAuthenticatedNoSession()  
{  
    assertFalse(servlet.isAuthenticated(request));  
}  
}
```

(3) 代码 5.8: OrderEJB.java

```
package junitbook.ejb.domain;  
import java.rmi.RemoteException;  
import java.util.Date;  
import javax.ejb.CreateException;  
import javax.ejb.EJBException;  
import javax.ejb.EntityBean;  
import javax.ejb.EntityContext;  
import javax.ejb.RemoveException;  
public abstract class OrderEJB implements EntityBean  
{  
    public abstract Integer getOrderId();  
    public abstract void setOrderId(Integer orderId);  
    public abstract Date getOrderDate();  
    public abstract void setOrderDate(Date orderDate);  
    public abstract String getOrderItem();  
    public abstract void setOrderItem(String item);  
    public OrderLocal ejbCreate(Date orderDate, String orderItem)  
        throws CreateException  
    {  
        int uid = 0;  
        // Note: Would need a real counter here. This is a hack!  
        uid = orderDate.hashCode() + orderItem.hashCode();  
        setOrderId(new Integer(uid));  
        setOrderDate(orderDate);  
        setOrderItem(orderItem);  
        return null;  
    }  
    public void ejbPostCreate(Date orderDate, String orderItem)  
        throws CreateException {}  
    public void ejbActivate()  
        throws EJBException, RemoteException {}  
    public void ejbLoad()
```



```
        throws EJBException, RemoteException {}
    public void ejbPassivate()
        throws EJBException, RemoteException {}
    public void ejbRemove()
        throws RemoveException, EJBException, RemoteException {}
    public void ejbStore()
        throws EJBException, RemoteException {}
    public void setEntityContext(EntityContext context)
        throws EJBException, RemoteException {}
    public void unsetEntityContext()
        throws EJBException, RemoteException {}
}
```

(4)代码 5.9: TestOrderEJB.java

```
package junitbook.ejb.domain;
import java.util.Date;
import javax.naming.InitialContext;
import junit.framework.TestCase;
import junitbook.ejb.util.JNDINames;
public class TestOrderEJB extends ServletTestCase
{
    public void testEjbCreateOk() throws Exception
    {
        OrderLocalHome orderHome =
            (OrderLocalHome) new InitialContext().lookup(
                JNDINames.ORDER_LOCALHOME);
        Date date = new Date();
        String item = "item 1";
        OrderLocal order = orderHome.create(date, item);
        assertEquals(date, order.getOrderDate());
        assertEquals(item, order.getOrderItem());
        assertEquals(new Integer(date.hashCode()
            + item.hashCode()), order.getOrderId());
    }
}
```

(5)代码5.10: build.xml

```
<?xml version="1.0" ?>
<project name="Ejb" default="test" basedir=".">
    <property file="build.properties" />
    <property name="conf.dir" location="conf" />
    <property name="src.dir" location="src" />
    <property name="src.java.dir" location="${src.dir}/java" />
```

```

<property name="target.dir" location="target" />
<property name="target.classes.java.dir" location="${target.dir}/classes" />
<target name="compile">
    <mkdir dir="${target.classes.java.dir}" />
    <javac destdir="${target.classes.java.dir}" srcdir="${src.java.dir}">
        <classpath>
            <path element location="${j2ee.jar}" />
        </classpath>
        Generic Ant properties pointing to location on filesystem Compiles runtime
        classes Using JUnit and remote calls 317
    </javac>
</target>
<target name="ejbjar" depends="compile">
    <jar destfile="${target.dir}/ejb.jar">
        <metainf dir="${conf.dir}">
            <include name="ejbjar.xml" />
        </metainf>
        <fileset dir="${target.classes.java.dir}" />
    </jar>
</target>
<target name="ear" depends="ejbjar">
    <ear destfile="${target.dir}/ejb.ear" appxml="${conf.dir}/application.xml">
        <fileset dir="${target.dir}">
            <include name="ejb.jar" />
        </fileset>
    </ear>
</target>
</project>

```

(6)代码5.11: web.xml (conf/cactus/ web.xml)

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <ejb-local-ref>
        <ejb-ref-name>Order</ejb-ref-name>
        <ejb-ref-type>Entity</ejb-ref-type>
        <local-home>junitbook.ejb.domain.OrderLocalHome</local-home>
        <local>junitbook.ejb.domain.OrderLocal</local>
        <ejb-link>Order</ejb-link>
    </ejb-local-ref>
</web-app>

```

(7)代码 5.12: appliction.xml

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC
'-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN'
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>
<application>
  <display-name>ejb</display-name>
  <description>EJB Chapter Sample Application</description>
  <module>
    <ejb>ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>cactus.war</web-uri>
      <context-root>test</context-root>
    </web>
  </module>
</application>
```

(8)代码 5.13: ejb-jar.xml

```
<?xml version="1.0" ?>
<project name="Ejb" default="test" basedir=".">
  [...]
  <property name="src.cactus.dir" location="${src.dir}/testcactus" />
  [...]
  <property name="target.classes.cactus.dir" location="${target.dir}/classsestestcactus" />
  [...]
  <target name="compile.cactus" depends="compile">
    <mkdir dir="${target.classes.cactus.dir}" />
    <javac destdir="${target.classes.cactus.dir}" srcdir="${src.cactus.dir}">
      <classpath>
        <pathelement location="${target.classes.java.dir}" />
        <pathelement location="${cactus.jar}" />
        <pathelement location="${j2ee.jar}" />
      </classpath>
    </javac>
  </target>
  <target name="ear.cactify" depends="compile.cactus,ear">
    <taskdef resource="cactus.tasks">
      <classpath>
        <pathelement location="${cactus.ant.jar}" />
        <pathelement location="${cactus.jar}" />
      </classpath>
    </taskdef>
  </target>
```

```
<pathelement location="${logging.jar}" />
<pathelement location="${aspectjrt.jar}" />
<pathelement location="${httpclient.jar}" />
</classpath>
</taskdef>
<cactifywar          version="2.3"          destfile="${target.dir}/cactus.war"
  mergewebxml="${conf.dir}/cactus/web.xml">
  <classes dir="${target.classes.cactus.dir}" />
</cactifywar>
<ear                update="true"          destfile="${target.dir}/ejb.ear"
  appxml="${conf.dir}/cactus/application.xml">
  <fileset dir="${target.dir}">
    <include name="cactus.war" />
  </fileset>
</ear>
</target>
</project>
```

致谢

本文的研究和撰写工作是在导师周南老师的精心指导和无微不至的关怀下完成的。在研究生学习期间，周老师对我始终严格要求、充分信任，并给我热情的鼓励和全面锻炼的机会。周老师严谨的治学态度、敏锐的学术思想、敢于创新和实践的勇气、崇高的敬业精神以及平易近人的师者风范对我都产生了深深的影响，将使我终生受益。她所教诲的研究方法的探索、创新能力的培养以及为人处世的原则对我产生的影响远远超过了论文本身。在此，特向周老师致以最诚挚的敬意和表示最由衷的感谢！

同时，还要感谢邱丽娟和叶海健两位老师在我攻读硕士学位期间给予的关怀和帮助。感谢在研究生学习期间给予我关怀和帮助的所有老师。王昌宝、朱静、宋晨光和杨冬平等同学在共同学习期间给予了我许多帮助，对此表示衷心的感谢。

最后，感谢许多不相识的朋友，他们在软件工程论坛上讨论的许多问题，给了我许多启发，对此表示深深的谢意！

作者简介

姓名：史明慧

性别：女

出生年月：1978 年 11 月

籍贯：陕西府谷

最高学历：工学硕士

毕业学校：中国农业大学

在中国农业大学就读研究生期间主要的科研工作经历如下：

参加的研究项目：

[1] 参加科技部“科技型中小企业技术创新基金”支持项目（项目编号：03C26211100786）中“基于网络的物流配送管理系统平台”的研究与开发。

[2] 参加科技部“十五”国家重大科技项目“小城镇科技发展重大项目（项目编号：**2003BA808A14**）——小城镇服务业技术研究与开发”课题

已发表的论文：

史明慧，周南，邱丽娟，邱小彬，王欣：

小城镇劳务中介管理信息系统的设计与实现，农业网络信息，2005（4）