三 江 学 院

本科毕业设计（论文）

题　　目　用测试驱动方式开发 Struts 2 应用

计算机科学与工程 院（系）计算机科学与技术 专业

学　　号　B05051073

学生姓名　陈　峰

指导教师　杨　老　师

起讫日期　2009 年　1 月　7 日～　6 月　07　日

设计地点　无锡软通动力

联系方式　QQ：562116039　邮箱：freechf@126.com

摘　要

测试驱动开发是一种不同于传统软件开发流程的新型的开发方法。它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。Struts 2 则是传统的 Struts 的替代者，是一个优秀的用于开发 Java 企业级应用的 Web MVC 框架，可以极大地提高开发效率。本文大量引用中外专业文献，力求在总结前人经验的基础上，有所整合、有所创新，探讨基于 Struts 2 的系统的测试途径以及用测试驱动的方式开发 Struts 2 应用的最佳解决方案。本文所提方案将在三江园丁网开发中进行验证。

关键词：测试驱动开发　测试优先编程　单元测试　重构　Junit　Struts 2　Ant

论文作者：陈峰
QQ：562116039

Abstract

Test-Driven Development is a new software development methodology which is different from traditional others. Using TDD we must create a test before writing code, this test will drive the code needed to implement the functionality. Not only does this improve the quality and design of the software, but it also simplifies the development process. Struts 2 is a combination of the popular Sturts framework and Webwork. It's very extensible and elegant for the development of java enterprise web application of any size. This article will discuss how to develop Struts 2 application by TDD, and these solutions will all be authenticated in the yuanding project.

Key words : Test-Driven Development Test-First Programming Unit Testing Refactoring Junit Struts 2 Ant

论文作者：陈峰
QQ：562116039

目　　录

论文作者：陈峰
QQ：562116039

论文作者：陈峰
QQ：562116039

第一章 测试驱动开发原理

1.1 结构化开发过程及其弊端

重量级软件开发中的结构化开发过程方法，是软件工程发展到较成熟阶段的产物，也是多年来软件业界的普遍标准，已经发展出了很多优秀的开发模型，如瀑布开发模型、V 模型、原型法、螺旋模型、RUP 等等。

但是随着当代软件开发对快速工作、响应变化及高效率高可靠性等的要求急剧增强，传统的结构化开发过程方法的诸多弊端也日渐凸显出来。

需求难以量化和准确把握，而且随时可能发生大的变化；在编码过程中甚至项目后期阶段经常发现严重的设计缺陷，必须中途改变设计思路。结构化开发过程很难适应这种种“变数”，而且将会导致成果难以检验，进度难以度量，破坏了其严谨和规范。而且重量级的开发过程降低了团队的开发效率，进而导致快速响应客户能力的下降，特别是我们所熟知的瀑布开发模型，其重量级的特征令许多软件团队望而生畏。

于是，不断返工重做、通宵达旦加班、技术骨干救急、预算严重超支、项目一再延期……这些现象可谓是司空见惯。

1.2 敏捷宣言和测试驱动开发

为了解决许多软件团队陷入不断增长的过程泥潭以及上文所述的各种问题，业界提出了一些可以让软件开发团队具有快速工作、响应变化能力的价值观和原则——敏捷宣言。

敏捷开发正越来越受到各国软件企业的青睐，成为软件工程领域一支后起之秀。敏捷开发过程的方法很多，其中最著名的是 Kent Beck 首先倡导的极限编程，而测试驱动开发，正是极限编程中所提出的一项最重要的核心实践和技术，同时它也是敏捷开发最重要的部分。通过测试驱动开发可以很好地解决以上所提及的种种问题。

Kent Beck 先生最早在其极限编程（XP）方法论中，向大家推荐“测试驱动”这一最佳实践，还专门撰写了《测试驱动开发》一书，详细说明如何实现。经过几年的迅猛发展，测试驱动开发已经成长为一门独立的软件开发技术，其名气甚至盖过了极限编程。

1.3 测试驱动开发基本原理

测试驱动开发（Test-Driven Development）简称 TDD，其基本思想就是在开发功能代码之前，先编写测试代码，然后只编写使测试通过的功能代码，从而以测试来驱动整个开发过程的进行。这有助于编写简洁可用和高质量的代码，有很高的灵活性和健壮性，能快速响应变化，并加速开发过程。

论文作者：陈峰
QQ：562116039

测试驱动开发的基本过程如下：

1. 快速新增一个测试
2. 运行所有的测试（有时候只需要运行一个或一部分），发现新增的测试不能通过
3. 做一些小小的改动，尽快地让测试程序可运行，为此可以在程序中使用一些不合情理的方法
4. 运行所有的测试，并且全部通过
5. 重构代码，以消除重复设计，优化设计结构

简单来说，就是不可运行/可运行/重构——这正是测试驱动开发的口号。

1.4 测试驱动开发的一个生动比喻

举个比较俗套的例子，这个例子或许你已经在关于 TDD 的文献资料上都看到过，但它确实是一个不错的比喻。

盖房子的时候，工人师傅砌墙，会先用桩子拉上线，以使砖能够垒的笔直，因为垒砖的时候都是以这根线为基准的。TDD 就像这样，先写测试代码，就像工人师傅先用桩子拉上线，然后编码的时候以此为基准，只编写符合这个测试的功能代码。

而一个新手，却可能不知道拉线，而是直接把砖往上垒，垒了一些之后再看是否笔直，这时候可能会用一根线，量一下砌好的墙是否笔直，如果不直再进行校正。使用传统的软件开发过程就像这样，我们先编码，编码完成之后才写测试程序，以检验的此段代码是否正确，如果有错误再进一步修改。

测试驱动开发与结构化软件过程方法最显著的区别正在于这里，我们只是改变了原始的砌墙方式，转而先拉线再砌墙，这样可以达到很好的效果，而且实际上提高了效率。

1.5 测试驱动开发的本质和优势

或许只有了解了测试驱动开发的本质和优势之后，你才会领略到她的无穷魅力。

测试驱动开发不是一种测试技术，它是一种分析技术、设计技术，更是一种组织所有开发活动的技术。相对于传统的结构化开发过程方法，它具有以下优势：

1. TDD 根据客户需求编写测试用例，对功能的过程和接口都进行了设计，而且这种从使用者角度对代码进行的设计通常更符合后期开发的需求。因为关注用户反馈，可以及时响应需求变更，同时因为从使用者角度出发的简单设计，也可以更快地适应变化。
2. 出于易测试和测试独立性的要求，将促使我们实现松耦合的设计，并更多地依赖于接口而非具体的类，提高系统的可扩展性和抗变性。而且 TDD 明显地缩短了设计决策的反馈循环，是我们几秒或几分钟之内就能获得反馈。

论文作者：陈峰
QQ：562116039

3. 将测试工作提到编码之前，并频繁地运行所有测试，可以尽量地避免和尽早地发现错误，极大地降低了后续测试及修复的成本，提高了代码的质量。在测试的保护下，不断重构代码，以消除重复设计，优化设计结构，提高了代码的重用性，从而提高了软件产品的质量。

4. TDD 提供了持续的回归测试，使我们拥有重构的勇气，因为代码的改动导致系统其他部分产生任何异常，测试都会立刻通知我们。完整的测试会帮助我们持续地跟踪整个系统的状态，因此我们就不需要担心会产生什么不可预知的副作用了。

5. TDD 所产生的单元测试代码就是最完美的开发者文档，它们展示了所有的 API 该如何使用以及是如何运作的，而且它们与工作代码保持同步，永远是最新的。

6. TDD 可以减轻压力、降低忧虑、提高我们对代码的信心、使我们拥有重构的勇气，这些都是快乐工作的重要前提。

1.6 测试驱动开发现状和前景

测试驱动开发的技术已得到越来越广泛的重视，但由于发展时间不长，相关应用并不是很成熟。现今越来越多的公司都在尝试实践测试驱动开发，但由于测试驱动开发对开发人员要求比较高，更与开发人员的传统思维习惯相违背，因此实践起来有一定困难。

美国不少著名软件公司如 IBM 很早就开始向敏捷转型，在此过程中，TDD 通常是最重要也最艰难的一个，正如 IBM 开发转型部门副总裁 Sue Mckinney 所言：测试驱动开发前景非常诱人，但是“在这个过程中我们的付出可能也是最多的。” Forrester 的高级分析师 Dave West 认为，测试驱动开发（TDD）就像是“圣杯”，但是“如果能达到这个目标，付出再多的辛苦也是值得的。”

测试驱动开发的推广过程中，首要的问题是将开发人员长期以来形成的思维观念和意识形态转变过来，开发人员只喜欢编码，不喜欢测试，更无法理解为什么没有产品代码的时候就先写单元测试；其次是相关的技术支持，测试驱动开发对开发人员提出了更高的要求，不仅要掌握测试和重构，还要懂得设计模式等设计方面的知识。

论文作者：陈峰
QQ：562116039

第二章 测试驱动开发实践

工欲善其事，必先利其器。

在开始测试驱动开发的实践之前，我们得做好充分的准备工作，我们首先要熟练掌握两项重要技能——单元测试和重构。这两项技术，是进行 TDD 实践的两大手段，缺一不可。

2.1 单元测试技术

软件测试已经成为确保软件质量、可靠性的重要手段和过程，如今已经产生了许多优秀的测试理论和测试方法，逐渐形成了一整套完整的体系。

根据软件测试在软件生命周期各阶段的不同表现，软件测试可以分为四种：单元测试、集成测试、系统测试和验收测试。

其中单元测试是最基础的，是所有测试的起点，并且越来越来受到重视。

2.1.1 单元测试与 JUnit

测试驱动开发中的“测试”主要指单元测试，因此所谓测试驱动，即是由单元测试驱动的。

单元测试是指对组成程序的基本单元（比如一个类或者一个方法）进行测试，验证每个单元是否完成了预期的功能。单元测试是由开发人员进行的，因此有人又称之为程序员测试。

单元测试要达到的目标，总体来说就是保证单元内部的处理是正确的、没有遗漏和多余功能。细分而言，单元测试要达到以下几个目标：

1. 信息能否正确地流入和流出单元。
2. 在单元工作过程中，其内部数据能否保持其完整性，包括内部数据的形式、内容及相互关系不发生错误，也包括全局变量在单元中的处理和影响。
3. 在为限制数据加工而设置的边界处，能否正确工作。
4. 单元的运行能否做到满足特定的逻辑覆盖。
5. 单元中发生了错误，其中的出错处理措施是否有效。

我们一般不需要全程手动编写测试代码，或者自己实现测试框架，使用现成的单元测试框架会为我们提供极大的方便。目前最流行的单元测试框架是 xUnit 系列，常用的根据语言不同分为 JUnit（java），CppUnit（C++），DUnit（Delphi ），NUnit（.net）等等。面向 Java 语言的 JUnit，是其中最成功和最著名的一个。

JUnit 由 XP 和 TDD 的创始人 Kent Back 以及 Eclipse 架构师之一、设计模式之父 Erich Gamma 共同打造，已经无可争议地成为 Java 领域单元测试的标准框架。Martin Fowler 如

论文作者：陈峰
QQ：562116039

此评价 JUnit：在软件开发领域，从来就没有如此少的代码起到了如此重要的作用。它大大简化了开发人员执行单元测试的难度，特别是 JUnit 4 使用 Java 5 中的注解（annotation）使测试变得更加简单。正是 JUnit 把我们带入了测试驱动开发的时代。

下面是对 JUnit 一些特性的总结：

1. 提供的 API 可以让你写出测试结果明确的可重用单元测试用例
2. 提供了三种方式来显示你的测试结果，而且还可以扩展
3. 提供了单元测试用例成批运行的功能
4. 超轻量级而且使用简单，没有商业性的欺骗和无用的向导
5. 整个框架设计良好，易扩展，对不同性质的被测对象，Junit 有不同的使用技巧。
6. 使用断言方法判断期望值和实际值差异，返回 Boolean 值。
7. 测试驱动设备使用共同的初始化变量或者实例。
8. 测试包结构便于组织和集成运行。
9. 支持图型交互模式和文本交互模式。

2.1.2 Mock objects 与 EasyMock

Junit 的确很强大，为我们进行单元测试打开了方便之门，但实际应用中经常遇到很多复杂的问题——被测试的类要使用真实的对象，依赖于外部环境（比如 Web 和数据库），这些外部对象并不总是被提供的，或者在单元测试中使用会很昂贵。这时就可以采用模拟对象的技术——mock objects。

Mock objects 是一种模拟一些在应用中不容易构造或者比较复杂的对象，从而把测试与测试边界以外的对象隔离开的策略。理论上使用 Mock objects 策略可以模拟任何外部对象，但需要你熟悉被调用的 API 的行为，并且创建对象带来的开销是无法忽略的。不过已经产生了许多基于此策略的框架(mock object framework)，可以帮助你快速地生成需要模拟的外部对象，比如 EasyMock、jMock、MockRunner 等。

对于 Struts 1，因为与其本身的 API 以及 Servet API 耦合较多，测试困难，可以使用 StrutsTestCase 进行测试，Struts 2 则没有相应的测试框架，因为它本身的松耦合等特性使得我们可以进行独立的单元测试。本文中将主要使用 JUnit，并结合 EasyMock 或 jMock 两种模拟对象的框架，以及数据库测试组件 DbUnit。

2.1.3 TDD 中测试的原则

测试驱动开发中中的单元测试与一般的单元测试不完全等同，主要是我们是在没有产品代码的前提下写测试代码的，在 TDD 过程写单元测试代码还需要遵循一定的原则：

1. 测试优先原则。这是 TDD 中的测试最显著的特征，测试驱动开发的过程中，测试

论文作者：陈峰
QQ：562116039

代码永远是在产品代码之前编写的，然后只需要编写使测试通过的产品代码就可以了。

2. 用户需求驱动。测试应该从用户的角度出发，根据用户需要的功能编写测试代码，然后以此驱动产品代码的开发，这样开发出的代码就更加符合用户需求。

3. 小步前进的原则。把所有的规模大、复杂性高的工作，分解成小的任务来完成。对于一个类来说，一个功能一个功能的完成，如果太困难就再分解。通过分解降低整个系统开发的复杂性。

4. 测试的独立性。测试之间应该互不干扰，这样一个测试失败了就对应一个问题，两个测试失败了就对应两个问题，而不应该牵连出更多的问题。独立测试会促使我们实现高内聚、松耦合的设计。

5. 避免过度设计。只关注当前的工作，只为当前需要实现的功能编写测试，不要过多地考虑其他方面的细节，也不要过多地考虑后期的扩展，只着眼于现下，保证头上只有一顶帽子。避免考虑无关细节过多，无谓地增加复杂度。

6. 感觉和经验。那些重要的功能、核心的代码就应该重点测试。感到疲劳就应该停下来休息一下。感觉没有必要更详细的测试，就停止本轮测试。TDD 强调测试并不应该是负担，而应该是帮助我们减轻工作量的方法。而对于何时停止编写测试用例，也是应该根据你的经验，功能复杂、核心功能的代码就应该编写更全面、细致的测试用例，否则测试流程即可。

2.2 重构技术

重构（Refactoring）是对软件内部结构的一种调整，目的是在不改变外部行为的前提下，提高其可理解性,降低其修改成本。开发人员可以使用一系列重构准则，在不改变软件行为的前提下，调整软件的结构。

Martin Fowler 在其著名的《重构——改善既有代码的设计》[7]一书中谈到了为何重构的几点原因：

1. 重构可以改进软件的设计。
2. 重构可以使代码更易被理解。
3. 重构可以协助找到 Bugs 。
4. 重构可以提高编程的速度。

至于何时重构，怎样重构等问题，《重构——改善既有代码的设计》书中都给出了详尽的讲解。

在学习和掌握了重构相关的触发信号以及相应重构方法之后，会使我们在测试驱动开发的过程中更加得心应手。

论文作者：陈峰
QQ：562116039

2.2.1 TDD 中的重构

对测试驱动开发当中所经常使用到的重构方法，Kent Beck 在《测试驱动开发》[1]一书第 31 章也进行了讲解，掌握了这些模式，就基本够用了。这些模式主要是：调和差异、隔离变化、数据迁移、提取方法、内联方法、提取接口、转移方法、方法对象、添加参数以及把方法中的参数转变为构造函数中的参数等。

在测试驱动开发的过程中，需要不断重构我们的代码，其目标很明确——消除重复设计，优化设计结构。但测试驱动开发中的重构还有一些需要注意比较独特的地方：

首先，重构的过程中也要遵循小步前进的原则。重构应该是由无数个微小的步骤组成的。不要囫囵吞枣，试图一次性完成所有代码的重构，一小步一小步地去完成，积以跬步，以至千里。你会从“小步前进”原则的实践中尝到甜头。当然，重构的步骤大小可以根据自己以及代码的具体情况来调整。

其次，很重要的一点，所有重构都应该是在测试代码的支持下进行的。否则，我们重构时犯了错误，也不会知道，等后期发现时将花费更多的成本。这就不是测试驱动开发了。因此，在重构之前一定要补上需要的测试。

然后，及时重构。无论是功能代码还是测试代码，对结构不合理，有重复的代码等情况，在测试通过后，要及时进行重构。有单元测试的支持，你可以放心地重构，如果产生什么错误，测试程序会立马告诉它在哪里，因此你不必担心不可预知的情况。

2.3 测试与重构

测试可以给予我们对代码质量的自信，以及对代码进行重构的勇气，测试能够让我们相信较大的重构不会改变系统的行为；我们信心越足，就越敢于尝试能延长系统生命周期的大规模的重构；通过重构，又可以使下一轮的测试编写工作更容易。

TDD 正是在这种测试与重构的交互作用下，不断推动我们的开发，并使我们获得“整洁可用的代码”的。

2.4 Strtus 2 应用 TDD 实践

2.4.1 Struts 2 框架概述

在 Web 应用开发领域， MVC 已经成为了构架 Web 体系结构的最基本模式。在 Java 开源社区，已经出现众多优秀的基于 MVC 模式的 Web 框架，比如 Struts 1、WebWork、Spring MVC 等，其中最著名的便是 Struts 1。Struts 1 框架得到了广泛的应用，已经成为了事实上的标准。

但 Struts 1 自身也存在不少的缺点：需要编写的代码过多，容易引起“类爆炸”，与 Servelt

论文作者：陈峰
QQ：562116039

API 和 Struts 1 API 严重耦合，单元测试困难等等。这些缺点随着 Web 的发展越来越突出，并严重制约了开发者的手脚，于是就促生了新的 Struts 2 框架。Struts 2 的能够很好的解决上述问题，它已经完全超出了 Struts 1 框架原有的高度，建立在 Struts 1 和 WebWork 两个框架整合的基础上，是两大社区合并后的产物，因此提供了很多优秀的机制，拥有其它框架无可比拟的优势。

Struts 2 框架主要由三个部分组成：核心控制器、业务控制器和用户实现的业务逻辑组件。核心控制器 FilterDispatcher 负责拦截用户请求，如以 action 结尾则转入 Struts 2 框架处理；然后会调用相应业务控制器（用户实现的 Action）来处理用户请求；业务控制器 Action 只作为中间负责调度的调度器，一般不对用户请求进行实际处理，而是调用模型组件处理具体的业务逻辑；之后核心控制器根据 Action 的处理结果返回相应视图给用户显示最终处理结果。

Struts 2 框架到处体现着 POJO、AOP、低侵入、低耦合等优秀编程思想以及轻量级的设计技巧，同时作为 Struts 1 的替代者和两大社区整合后的产物，它拥有其他框架所无法具备的先天优势。基于这些，相信 Struts 2 将会在短期内成为 Web MVC 领域最流行的框架，将会比原有的 Struts 1 框架更流行，更强大。这些优势，也使得单元测试变得简单可行，使得 Struts 2 可以脱离容器进行测试，从而使测试驱动开发成为可能。

2.4.2 Struts 2 与 TDD 结合的意义

测试驱动开发可以极大地提高开发效率和保证代码质量，能够造就简单、清晰、高质量的代码，有很高的灵活性，能快速响应变化，并加速开发过程。

而作为 Struts 1.x 的替代者，Struts 2 则不愧为一款优秀的 MVC 框架，它采用了大量轻量级的设计技巧以及 AOP 等优秀的编程思想，从而降低了单元测试的难度，使测试可以脱离 Web 容器进行。这些都极大地简化了单元测试的难度，使得采用测试驱动开发变得可行。

现今国内 Struts 2 正逐渐取代 Struts 1.x 成为 Java 领域主流 MVC 框架，越来越多的 Java 企业级项目都开始采用 Struts 2 框架。在开发 Struts 2 应用的过程中，如果采用测试驱动开发的方法，更加可以提高软件产品的质量和加速开发过程，而且可以快速响应变化，避免了结构化开发过程方法带来的种种弊端，因此研究这两者的结合拥有很大的现实意义和实用价值。

因此，以 Struts 2 作为框架，并采用测试驱动开发的软件开发方法，可以明显地提高我们开发 Java 企业级应用的效率，同时可以构造出松耦合、高内聚的系统，而且具有瑕疵率低、维护成本低等特点。

论文作者：陈峰
QQ：562116039

2.4.3 本工作的目的和研究重点

本文就将探讨基于Struts 2的系统的测试途径以及用测试驱动的方式开发Struts 2应用的最佳解决方案。限于本人水平可能难以把握其精髓，但还是期望能够摸索出一些可行的途径。

Struts 2框架主要的三个部分是核心控制器FilterDispatcher以及用户实现的Action和业务逻辑组件，其中FilterDispatcher是框架的核心，负责将用户请求转发到Action，并负责将Action的处理结果转换成对用户的响应。我们只是使用Struts 2框架，它是成熟的产品，应该信赖它，所以没有必要去测试它的API。

因此我们研究的重点在于由用户实现的Action和业务逻辑，我们又将业务逻辑分为业务逻辑层和数据库访问层，因此我们将从业务控制器、业务逻辑层、数据库访问层三方面来具体探讨Struts 2应用的TDD解决方案。

至于除控制器和模型之外的另一个重要模块——视图，由于本文旨在探讨Struts 2框架应用的TDD解决方案，所以未作研究。Russell Gold、Thomas Hammell 和Tom Snyder三位作者合著的《Test Driven Development: A J2EE Example》一书，对J2EE系统的TDD解决方案有完整论述，其中便包括视图层的TDD解决方案。

2.5 园丁网项目TDD实践

2.5.1 园丁网相关概述

理论若不扎根于具体实践，必将脱离实际，成为纸上谈兵。

本文将结合园丁网项目的具体开发工作，来探究和验证测试驱动开发Struts 2应用的具体施行方案。为了文章的简洁清晰，同时又能够浅显明白地说明问题（这也正是秉持TDD思想的），我将截取园丁网中的登录注册模块的开发工作进行演示。而且我还会进行一定程度地简化，因为我们只是探讨通用解决方案，不需要展示复杂的代码，那只会徒增文章理解的难度。

园丁网项目使用MyEclipse 6.0作为集成开发环境，这会为我们带来很多的方便，比如它集成了JUnit插件，因此编写单元测试后可以直接运行。通过插件的方式，MyEclipse几乎可以集成所有我们将要用到的工具，方便了各种资源的综合管理。另外MyEclipse还可以进行自动代码检查等工作，可以极大地节省我们的时间和精力。正如单元测试需要工具的支持，重构也一样需要专门的工具， MyEclipse本身就是一款优秀的重构工具，它支持众多的自动化重构技巧。

论文作者：陈峰
QQ：562116039

2.5.2 以下章节的内容概要

掌握了 TDD 和 Struts 2 的相关理论知识，以及单元测试和重构两大技术，同时拥有了 MyEclipse 集成开发环境，我们就可以正式开始 Strtus 2 应用的 TDD 实践了。

接下来，将针对上面所提及的几个研究重点，逐步探讨通用的 Strtuts 2 应用的 TDD 解决方案，以下几章的主要内容介绍如下：

第三章：主要探讨业务控制器即 Action 的 TDD 解决方案；

第四章：专门探讨业务逻辑部分的 TDD 解决方案；

第五章：本章将探讨数据库访问层或 DAO 组件的 TDD 解决方案；

第六章：本章介绍在项目中实践 TDD 需要掌握的其他几个技能，如怎样组合测试源代码、怎样组合多个单元测试、如何进行自动化测试和自动化构建等。

论文作者：陈峰
QQ：562116039

第三章 业务控制器的 TDD 方案

这里将使用登录模块进行案例演示，首先来说明登录模块的相关功能需求。登录很简单，在登录页面 login.jsp 中输入用户名、密码（有时还需输入校验码，此处略过），点击登录后转到 LoginAction 进行登录验证；用户名密码验证正确则成功登录并显示园丁网主界面，如 main.jsp；验证失败则留在本登录页面并显示错误信息，如“用户名密码不能为空”、“用户名或密码不正确”等。

由以上分析可见，Action 真实反映了应用程序的功能和流程，清晰地表达了功能需求；另外，Struts 2 的 Action 只负责调度，具体的处理调用业务逻辑组件进行，因此结构简单清晰，因此 Action 最适合作为我们工作的起点。

这里将用登录功能的实现进行案例演示，所以我们首先实现来 LoginAction，它有 username、password 两个私有属性，execute 方法负责验证登录是否成功并返回处理结果。

Struts 2 的 Action 根据处理结果返回的是字符串，而不是 Struts 1 中的 ActionMapping，因此要验证返回处理结果的正确性，只需要比较返回的字符串是否与预期的字符串相同。先设计单元测试用例如下：

测试用例	**测试方法名**	**期望返回值**
登录成功	`testExecuteSuccess`	success
登录失败	`testExecuteFailure`	input

3.1 方案一：将 Action 作为 POJO

现在可以开始编写测试代码 LoginActionTest 了，它继承 junit.framework 包里的 TestCase 类。先完成登录成功的测试用例，根据我们所要实现的功能需求编写测试代码。此处将 LoginAction 类作为普通的 POJO（一个简单的 Java 类）来处理：

论文作者：陈峰
QQ：562116039

```
public class LoginActionTest extends TestCase {
    LoginAction login = null;
    @Before
    public void setUp() throws Exception {
        login = new LoginAction();
    }
    @After
    public void tearDown() throws Exception {
    }
    @Test
    public final void testExecute() throws Exception {
        String result = login.execute();
        Assert.assertEquals("登录成功！", "success",result);
    }
}
```

JUnit 单元测试的基石是断言，比如这里通过 Assert.*assertEquals*("登录成功！", "success", result)一句，来验证 execute 方法的返回结果是否为 success 字符串。

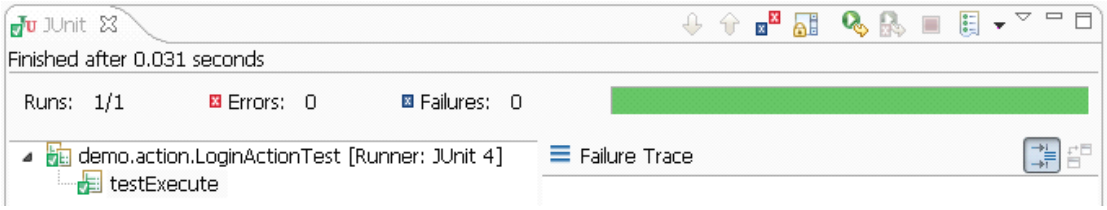
此时运行测试肯定失败！那是必然的，因为我们还没有实际的类呢。其实不用运行测试，MyEclipse 已经提示我们错误了。

好了，现在新建 LoginAction 类，我们不用管怎么实现具体的功能，现在当务之急是想尽一切办法使测试通过，那就直接返回“success”字符串吧：

```
public class LoginAction {
    public String execute(){
        return "success";
    }
}
```

这里直接返回了一个字符串常量，其实是使用了“伪实现”[1]的方法。因为当前的中心任务是尽快地使测试通过，此时甚至“可以使用一些不合情理的方法”[1]。

现在运行测试，进度条显示绿色——成功：



登录当然要有用户名、密码啦！还好 Struts 2 的 Action 通过属性封装用户请求以及要

传入下个页面的处理信息[5]，因此单元测试代码直接通过 set 方法向 Action 传入请求参数，用 get 方法从 Action 获取处理结果信息。

因此在 LoginActionTest 的 testExecute 方法起始处添加两行代码，用于设定用户名和密码，真实场景中这两个参数是由用户登录页面传递过来的：

```
@Test
public final void testExecute() throws Exception {
    login.setUsername("root");
    login.setPassword("chenfeng");
    String result = login.execute();
    Assert.assertEquals("登录成功！", "success",result);
}
```

现在测试会失败，因为LoginAction 类中还未添加相应属性和 set 方法。修改 LoginAction 类，添加如下代码：

```
private String username;
private String password;
public String getUsername() {
    return username;
}
public void setUsername(String username) {
    this.username = username;
}
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
}
```

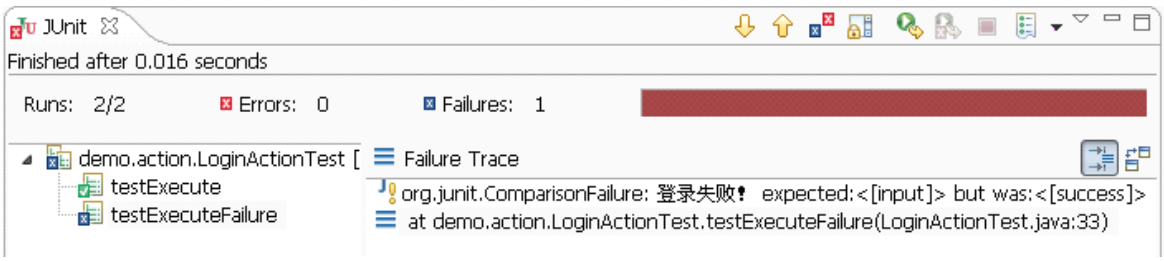
现在我们再来添加登录失败的用例：

```
@Test
public final void testExecuteFailure() throws Exception {
    login.setUsername("saas");
    login.setPassword("123456");
    String result = login.execute();
    Assert.assertEquals("登录失败！", "input",result);
}
```

此用例命名为 testExecuteFailure，表示测试登录失败的情况，而登录成功的用例名称为 testExecute，我们将它改名为 **testExecuteSuccess**。不要看只是命名，这很重要，这有助于增强代码的可读性，而且这是使得单元测试代码可作为文档的前提条件。

此时运行测试，进度条显示红色：

论文作者：陈峰
QQ：562116039



失败了！不过不要仅，JUnit 会告诉我们是哪里发生了错误：是“登录失败”的用例。“expected:<[input]> but was:<[success]> ”提示我们：预期结果是 input，而实际返回的却是 success。

因此用单元测试来驱动开发的时候，我们不用担心发生什么难以预料的错误，只要有错误，测试会及时地告诉我们它在哪里。当然，前提是我们的测试代码足够强大，覆盖了所有可能的情况。

现在来修改 LoginAction 的 execute 方法：

```
public String execute() {

    if (this.username.equals("root")
            && this.password.equals("chenfeng") ) {
        return "success";
    } else {
        return "input";
    }
}
```

运行 LoginActionTest 测试程序，成功了。

目前为止，我们的每个测试用例中都只有一个断言，我们模拟只有输入用户名为 root、密码为 chenfeng 才是正确的，实际当中这应该是从数据库中去读取的，而现在我们只是要测试此段代码的“行为是否正确”[4]（也就是说看输入正确时有什么反应，输入错误时又有何反应，是否如我们所预期的一样），并且我们成功了。

3.2 在 Action 中调用业务逻辑组件

现在可以考虑实现具体功能代码了，我们可以在 execute 方法里直接查询数据库进行登录验证，但那将会造就极差的代码，没有任何扩展性、重用性可言，而且违背了 Struts 2 的设计精神。因此我们调用一个 LoginManage 来处理登录相关的所有业务逻辑，比如输入校验、登录验证等。

我刚开始是尝试将 LoginManage 作为一个类的，然后在 LoginAction 中 new 一个实例出来就行了。但是考虑到写单元测试的时候，我束手无策了！我要测试的代码里面调用了其他对象，而那个类还没有写出来呢，难道现在要转移注意力去完成那个类吗？如果那个类还要再调用另一个外部对象呢？

这里你可以很快地写出 LoginManage 类，但实际项目中经常遇到当前模块所依赖的外部资源还根本没有提供，或者这些外部资源使用起来比较昂贵，比如数据库。况且单元测试当中是不应该再调用其他自己所要实现的模块的。

此时 mock objects 策略就可以大展身手了。Mock objects 是一种模拟一些在应用中不容易构造或者比较复杂的对象，从而把测试与测试边界以外的对象隔离开的策略。

3.2.1 被测试对象拥有其他对象

J.B.Rainsberger 和 Scott Stirling 所著的《JUnit Recipes》给了我们以下宝贵的建议：

1. 如何创建被测试对象所包含对象的替身？

1) 创建一个接口的测试对象：只要创建一个类，用最简单的方法实现这个接口就可以了。

2) 创建一个类的测试对象：可以创建它的一个子类，然后仿造它的全部方法，或者干脆屏蔽它的全部方法。

2. 如何将它传递给要测试的对象？

1) 仿造构造函数。

2) 添加一个 set 方法。

两位作者还提出：在要测试的对象中声明接口，然后在测试时转变为具体类的对象是有好处的，因为这样我们就可以使用 Easy Mock 了。如果在对象中要使用其他的对象，最好将其声明为接口，这样我们的设计就更有弹性，也更容易测试。

TDD 过程中，代码的可测试性和单元测试的独立性会要求我们实现松耦合的设计，否则测试程序的编写将是一件颇为头疼的事情。即使写出了一个单元测试，但却相当冗长复杂，那就毫无疑问的说明：我们的设计是有问题的，需要重构了。

这正是 TDD 作为一种设计技术的体现。

3.2.2 将外部对象声明为接口

虽然 TDD 可以驱动我们优化设计，但那是一个摸索的过程。在这里，我们还是听取前人的建议，少走弯路，因此我们将 LoginManage 作为一个接口，它提供了验证用户名密码

论文作者：陈峰
QQ：562116039

的方法声明：

```
public interface LoginManage {
    public boolean loginValid(String name,String pass);
}
```

我们再新建一个类 LoginManageMockImpl，它实现了 LoginManage 接口，当然也实现了 loginValid 方法，现在先采用伪实现写一段简陋的代码：

```
public class LoginManageMockImpl implements LoginManage {
    public boolean loginValid(String name, String pass) {
        if (name.equals("root")
                && pass.equals("chenfeng") ) {

            return true;
        } else {
            return false;
        }
    }
}
```

这只是一个业务逻辑组件的模拟，写得越简单越好，只要能测试当前目标即 LoginAction 的行为即可。

修改 LoginAction 的代码，添加一个 LoginManage 的属性和它的 set 方法：

```
private LoginManage loginManage;
public void setLoginManage(LoginManage mgr){
    this.loginManage = mgr;
}
```

setLoginManage 方法可以提供给测试程序使用，将一个模拟的对象实例传给它。这也为将来整合 Spring 使用 IoC 容器提供了方便，这正是 Struts 2 框架官方所提倡的。可见这种设计极大地提高了应用的可扩展性。这里先假设我们将使用 Spring 实现依赖注入。

然后要修改 LoginAction 的 execute 方法，调用外部对象进行实际的登录验证处理，这样所有的实际处理工作都仍给了传入的 loginManage 实例对象：

论文作者：陈峰
QQ：562116039

```
public String execute() {
    if (loginManage.loginValid(this.username, this.password)) {
        return "success";
    } else {
        return "input";
    }
}
```

此时运行测试，显示失败，因为测试程序还没有给 LoginAction 传入一个实例，修改 LoginActionTest，在两个测试方法中都添加了同样的一句：

```
login.setLoginManage(new LoginManageMockImpl());
```

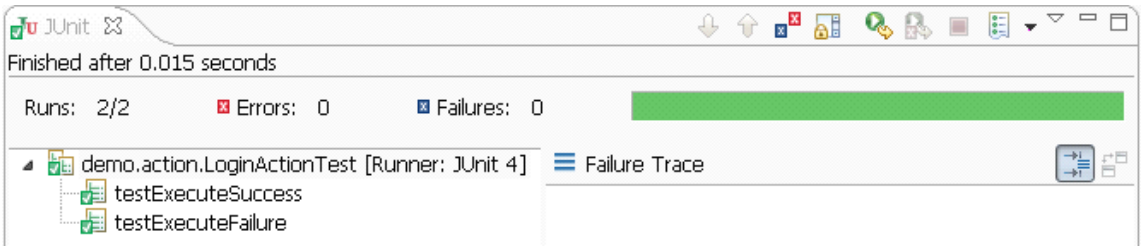
，

但这造成了重复，因此我将这一句移到 setUp 方法体里：

```
@Before
public void setUp() throws Exception {
    login = new LoginAction();
    login.setLoginManage(new LoginManageMockImpl());
}
```

这是对测试代码进行的最简单的重构，正如产品代码一样，测试代码也是需要不断重构的，从而使测试代码也一样达到简单清晰而有效的终极目标。

此时运行测试，成功了：



实际项目中最好不要在控制器中直接创建业务逻辑组件的实例，而是通过工厂模式管理业务逻辑组件实例，或者通过依赖注入将业务逻辑组件实例注入控制器组件。[5]

3.2.3 调用的外部对象为具体类

如果业务逻辑组件比较简单，我们不想将它声明为接口，而是直接将 LoginManage 实现为一个具体的类，那我们只要修改我们的模拟类 LoginManageMockImpl，其他代码都不需要作任何改动。

论文作者：陈峰
QQ：562116039

使用这种方法的前提条件是 LoginManage 类已经创建了，我们只需要修改 LoginManageMockImpl 为继承自 LoginManage 类的子类（而不再是接口 LoginManage 的实现类），然后仿造父类 LoginManage 的方法。这样可以达到相同的效果。

不使用依赖注入的情况 ：

如果我们不想使用 Spring 的 IoC 容器，也不想使用工厂模式，只想简单的在控制器中直接实例化。那我们可以这样做：

新建 LoginMange 接口的实现类 LoginManageImpl（这才是真正的业务逻辑组件），然后修改 LoginAction 的构造函数：

```
public LoginAction(){
    this.loginManage = new LoginManageImpl();
}
//测试专用的构造函数
public LoginAction(LoginManage mgr){
    this.loginManage = mgr;
}
```

真实情况下会调用上面的默认构造函数，它会自己 new 一个实例出来，而测试时可以使用第二个构造函数，直接传入一个实例。

还要修改 LoginActionTest 方法的 setUp 方法：

```
@Before
public void setUp() throws Exception {
    login = new LoginAction(new LoginManageMockImpl());
}
```

运行一下测试，还是成功的。

3.2.4 目前为止完整的代码

LoginActionTest

论文作者：陈峰
QQ：562116039

```
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import junit.framework.TestCase;


public class LoginActionTest extends TestCase {
    LoginAction login = null;

    @Before
    public void setUp() throws Exception {
        login = new LoginAction(new LoginManageMockImpl());
    }

    @After
    public void tearDown() throws Exception {
    }


    @Test
    public final void testExecuteSuccess() throws Exception {
        login.setUsername("root");
        login.setPassword("chenfeng");
        String result = login.execute();
        Assert.assertEquals("登录成功！", "success", result);
    }

    @Test
    public final void testExecuteFailure() throws Exception {
        login.setUsername("saas");
        login.setPassword("123456");
        String result = login.execute();
        Assert.assertEquals("登录失败！", "input", result);
    }
}
```

LoginAction

论文作者：陈峰
QQ：562116039

```
public class LoginAction {
    public LoginAction() {
        this.loginManage = new LoginManageImpl();
    }

    // 测试专用的构造函数
    public LoginAction(LoginManage mgr) {
        this.loginManage = mgr;
    }


    private String username;
    private String password;
    private LoginManage loginManage;

    public void setLoginManage(LoginManage mgr) {
        this.loginManage = mgr;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }


    public String execute() {

        if (loginManage.loginValid(this.username, this.password)) {
            return "success";
        } else {
            return "input";
        }
    }
}
```

LoginManage

```
public interface LoginManage {
    public boolean loginValid(String name,String pass);
}
```


论文作者：陈峰
QQ：562116039

LoginManageMockImpl

```
public class LoginManageMockImpl implements LoginManage {
    public boolean loginValid(String name, String pass) {
        if (name.equals("root")
                && pass.equals("chenfeng") ) {

            return true;
        } else {
            return false;
        }
    }
}
```

3.3 在 Action 中访问 Servlet API

实际应用当中我们的 Action 不可能都是这样简单的，有时候肯定是要访问 HttpSession 等对象的。比如这里我们实现登录功能时，登录成功后要将用户名添加为 Session 状态信息。

为了访问 HttpSession 实例，可以使用 ActionContext 类的 getSession 方法，它返回一个 Map，访问这个 Map 对象就相当于访问 HttpSession 实例，Struts 2 的系列拦截器会负责该 Map 和 HttpSession 之间的转换。因此单元测试中只要想办法替换了这个 Map 对象即大功告成。

单元测试用例设计如下：

测试用例	期望返回结果	Session 状态信息	Session 属性的值
登录成功	success	user	root
登录失败	input	无	无

修改 LoginActionTest ，新增加载包如下 ：

```
import java.util.Map;
import java.util.HashMap;
import com.opensymphony.xwork2.ActionContext;
```

新增一个 Map 实例，并用它去替代 LoginAction 中将要访问的 Session：

论文作者：陈峰
QQ：562116039

```
Map<String,Object> session;
@Before
public void setUp() throws Exception {
    login = new LoginAction(new LoginManageMockImpl());
    session = new HashMap<String,Object>();
    ActionContext.getContext().setSession(session);
}
```

这样名为 session 的 Map 就替代了 LoginAction 中将要访问的 Session，并且可以在 LoginAction 和 LoginActionTest 中同样地操作并共用此 Map 对象，使得涉及 Session 对话的单元测试变得异常简单。

再修改两个测试用例，成功登录时应该访问到 session 中添加了一个名为 user 的状态信息，其值为 root ；登录失败时则不存在此状态信息：

```
@Test
public final void testExecuteSuccess() throws Exception {
    login.setUsername("root");
    login.setPassword("chenfeng");
    String result = login.execute();
    Assert.assertEquals("登录成功！ ", "success", result);
    Assert.assertEquals("成功时验证Session！ ","root", session.get("user"));
}
@Test
public final void testExecuteFailure() throws Exception {
    login.setUsername("saas");
    login.setPassword("123456");
    String result = login.execute();
    Assert.assertEquals("登录失败！ ", "input", result);
    Assert.assertNull("失败时验证Session！ ", session.get("user"));
}
```

运行测试，失败：

因为我们在 LoginAction 中成功登录时并没有添加 session 信息。

修改 LoginAction，先添加两个包：

论文作者：陈峰
QQ：562116039

```
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
```

然后修改 execute 方法：

```
public String execute() {
    Map<String, Object> session;
    session = ActionContext.getContext().getSession();

    if (loginManage.loginValid(this.username, this.password)) {
        session.put("user", getUsername());
        return "success";
    } else {
        return "input";
    }
}
```

再一次运行测试，进度条显示绿色了，成功。

再看一下现在的完整代码：

LoginActionTest

```
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import junit.framework.TestCase;
import java.util.Map;
import java.util.HashMap;
import com.opensymphony.xwork2.ActionContext;
```


论文作者：陈峰
QQ：562116039

```
public class LoginActionTest extends TestCase {
    LoginAction login = null;

    Map<String, Object> session;

    @Before
    public void setUp() throws Exception {
        login = new LoginAction(new LoginManageMockImpl());
        session = new HashMap<String, Object>();
        ActionContext.getContext().setSession(session);
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public final void testExecuteSuccess() throws Exception {
        login.setUsername("root");
        login.setPassword("chenfeng");
        String result = login.execute();
        Assert.assertEquals("登录成功! ", "success", result);
        Assert.assertEquals("成功时Session! ","root",session.get("user"));
    }

    @Test
    public final void testExecuteFailure() throws Exception {
        login.setUsername("saas");
        login.setPassword("123456");
        String result = login.execute();
        Assert.assertEquals("登录失败! ", "input", result);
        Assert.assertNull("失败时验证Session! ", session.get("user"));
    }
}
```

LoginAction

论文作者：陈峰
QQ：562116039

```
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;

public class LoginAction {
    public LoginAction() {
        this.loginManage = new LoginManageImpl();
    }

    // 测试专用的构造函数
    public LoginAction(LoginManage mgr) {
        this.loginManage = mgr;
    }


    private String username;
    private String password;
    private LoginManage loginManage;

    public void setLoginManage(LoginManage mgr) {
        this.loginManage = mgr;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }


    @SuppressWarnings("unchecked")
    public String execute() {
        Map<String, Object> session;
        session = ActionContext.getContext().getSession();

        if (loginManage.loginValid(this.username, this.password)) {
            session.put("user", getUsername());
            return "success";
        } else {
            return "input";
        }
    }
}
```


论文作者：陈峰
QQ：562116039

LoginManage

```
public interface LoginManage {
    public boolean loginValid(String name,String pass);
}
```

LoginManageMockImpl

```
public class LoginManageMockImpl implements LoginManage {
    public boolean loginValid(String name, String pass) {
        if (name.equals("root")
                && pass.equals("chenfeng") ) {

            return true;
        } else {
            return false;
        }
    }
}
```

3.4 方案一存在的问题：

本方案将 Action 作为普通的 POJO，即作为简单的 Java 类来处理。在测试程序中直接 new 一个实例出来，这种方式比较简单、更容易实现。

但这样做的话也存在一些问题，例如我们并没有测试 struts.xml 配置是否正确、拦截器是否运用得当等情况。

下面简略地介绍一下更复杂的解决方案。

3.5 方案二：复杂的解决方案

我们也可以采取复杂一些的方法，主要有基于 struts 的 mock 和基于 webwork 的 ActionProxyFactory 两种方式。

如果再深入一些，我们可以直接模拟出真实情况下 Struts 2 框架核心生成 Action 的场景。但这要求对 Struts 2 的源码理解得比较透彻，因为我们需要知道 Struts 2 框架核心处理一个请求时的具体流程，使用了哪些类来处理请求、如何生成 Action 实例、如何调用 Action 实例以及如何转发处理结果等，然后才能模拟出相关部分功能。

这种方法实现起来比较困难，但更具有普遍适用性，我们甚至可以自己开发出一套 Struts 2 的 Action 的测试框架，类似于 Struts 1 的 StrutsTestCase。关于这方面的资料，我在一些外国的技术博客上找到两篇比较有价值的文章：《Unit Testing Struts 2 Actions》《Unit Testing Struts 2 Actions wired with Spring using JUnit》，可作参考。

对于此种方案，本文暂不作探讨。

由于时间有限，还有水平有限（其实后者才是主要原因，呵呵），对于这种方案我还没有探索出自己的解决途径，假如我能在这方面有所突破的话，那我的论文就真正具有了自主创新的成果，就可以上升一个档次了，嘿嘿，自恋一把。哎，可惜，由于种种原因……如果再给我一次（从头开始上大学，嘿嘿），我一定会说——我会努力学习，努力成长，努力创新，在整整四年里！这段是我发上网与君共享之前加上的，有兴趣的朋友可以联系我，本人做不更名站不改姓——陈峰，QQ 是 562116039，邮箱是 freechf@126.com，大家可以一起切磋一起提高嘛！

很希望正在看我写的废话的您可以给我指点一下迷津，O(∩_∩)O~ 有什么好的想法和途径可千万不要藏着掖着啊，中国人应该团结起来，大家一起分享成果，集思广益，然后一起提高一起成长，然后才能去壮大中国的高端软件行业嘛。这样我们才能早点赶上老美，然后再超过老美啊！您说对不？哈哈，又说大话了，不好意思啊。。。

3.6 方案一和方案二的取舍：

方案二要求我们对 Struts 2 的 API 必须非常熟悉，要求比较高，不容易上手。

我想方案一已经可以应付一般的情况了，我们的 Action 就应该写得足够简单，尽量写成一个普通的 POJO，不与 Struts 2 API 耦合，更不要与 Servlet API 耦合，这样不仅易于测试，而且增强了 Action 的可扩展性、可重用性。

方案一已经测试了 Action 接受了相应请求参数后的具体行为，因此已经基本达到了我们的单元测试的目的。

至于方案一没有测试的问题，比如配置文件是否正确、拦截器是否运用得当等，并不一定要在单元测试中全部解决，因为我们在开发完成之后还是需要进行功能测试和 End-to-End 的用户测试的，这样就可以验证所有的这些问题了。不管我们的单元测试如何完美，它们并不能验证我们是否已经真正实现了用户所需要的功能，这是 End-to-End 的测试所要做的工作。[3]

考虑到园丁网项目规模比较小，开发人员也不多，以及时间有限，所以在本次项目开发过程中我选择了方案一。

我们可以根据项目的具体情况，综合考虑项目组人员的技术水平，以及其他各种因素，从两种方案当中选择更合适的一个。

3.7 阶段性小结：TDD 的原则

前面就园丁网的具体开发工作，通过详细的迭代步骤展示了测试驱动开发的真实流程，

而且严格遵循了 TDD 的一些模式或策略，比如测试优先、小步前进、独立测试、模拟对象以及伪实现、及时重构等。

这些原则不是让我们生搬硬套的，当掌握到一定的程序，就可以“忘记”它们了，要能融会贯通、运用自如、信手拈来，正像我们学习设计模式的过程一样。“无招胜有招”，或许用武侠世界的这句名言来形容，会显得更形象一些。正像习武之人，刚开始肯定是练习某门派武术套路，当修炼到一定阶段便可以运用自如，已经没有了那些套路，达到无招胜有招的境界，甚至可以自创武功。

论文作者：陈峰
QQ：562116039

第四章 业务逻辑层的 TDD 方案

接下来我将直接给出阶段性工作后的整体代码，而不是每写一点或修改一点就展示出一小段代码。但所有的代码我都是依照以上所展示的开发迭代步骤，遵循“不可运行/可运行/重构”的节奏进行开发的。

4.1 业务逻辑与数据库访问代码分离

我们一般不在这些业务逻辑组件中直接访问数据库，而是提供一个 DAO 层，封装所有的数据库访问操作，并向上层的业务逻辑组件提供数据访问接口。这样使用 DAO 模式，实现了底层数据访问操作与高层业务逻辑的分离。

采用这种做法，使得我们可以在不改变应用程序其他部分的情况下改变持久性策略，例如我们既可以用 JDBC 实现数据库访问操作，也可以使用 Hibernate 等 ORM 框架。同时，这样做还极大地简化了单元测试。

LoginManageImpl 是登录功能所使用的业务逻辑组件，负责处理登录相关的所有业务逻辑，如输入长短校验、非法字符串校验、用户名密码验证等。此处我们只实现简单的非空校验和用户名密码验证。

我们先创建了 UserDao 接口，它提供了所有登录和注册相关的所有数据库访问，如用户名密码验证、用户名是否已存在验证、插入新用户等：

UserDao

```
import java.util.List;
import sju.sjumis.yuanding.model.User;

public interface UserDao {
    //根据用户名密码查找用户
    public boolean findByNameAndPass(String name,String pass);
    //根据用户名查找用户
    public boolean findByName(String name);
    //插入新的用户
    public int insert(User user);
    //查询所有用户列表
    public List<String> findAllUsers();
}
```

但我们还没有写 UserDao 的具体实现类，所以需要使用模拟对象的技术。

这次我们不是自己手动模拟外部对象，因为实际情况下我们经常遇到外部对象比较复

论文作者：陈峰
QQ：562116039

杂，难以模拟，特别是当我们对外部对象的 API 不是很了解的时候。因此我们可以使用一些现成的基于 mock objects 策略的框架(mock object framework)。如起步早发展比较成熟的 EasyMock，还有产生时间不长但也比较优秀的 jMock、MockRunner 等等。

这里我们将使用 EasyMock 作为模拟对象工具。EasyMock 为 Mock Objects 提供接口并在 JUnit 测试中利用 Java 的 proxy 设计模式生成它们的实例。通过它可以方便地模拟接口和类，比如 Request、ResultSet 等，从而使单元测试顺利进行，可见 EasyMock 很适合于测试驱动开发。

设计 LoginManageImpl 的登录验证功能的测试用例：

测试用例	**期望返回值**	**期望 tip 提示信息**
用户名密码正确	true	成功登录
用户名密码错误	false	用户名或密码不正确
用户名密码为空	false	用户名或密码不能为空

完整的测试代码，LoginManageImplTest ：

```
import junit.framework.TestCase;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.easymock.IMocksControl;
import org.easymock.EasyMock;
import sju.sjumis.yuanding.dao.UserDao;
```


论文作者：陈峰
QQ：562116039

```
public class LoginManageImplTest extends TestCase {
    private LoginManageImpl loginManage;
    private IMocksControl control;
    private UserDao mockDao;
    @Before
    public void setUp() throws Exception {
        loginManage = new LoginManageImpl();

        // 使用 EasyMock 生成 Mock 对象
        control = EasyMock.createControl();
        mockDao = control.createMock(UserDao.class);

        // 将 Mock 对象传递给测试对象
        loginManage.setDao(mockDao);
    }
    @After
    public void tearDown() throws Exception {
        // 对 Mock 对象的行为进行验证
        control.verify();
    }
```

```
    @Test
    public void testLoginValidSuccess() {
        String name = "root";
        String pass = "chenfeng";

        // 设定 Mock 对象的预期行为和输出
        mockDao.findByNameAndPass(name, pass);
        EasyMock.expectLastCall().andReturn(true).times(1);

        // 将 Mock 对象切换到 Replay 状态
        control.replay();

        // 调用 Mock 对象方法进行单元测试
        assertTrue(loginManage.loginValid(name, pass));
        assertEquals("成功登录", loginManage.getTip());
    }
```

```
    @Test
    public void testLoginValidFailure() {
        String name = "saas";
        String pass = "123456";

        mockDao.findByNameAndPass(name, pass);
        EasyMock.expectLastCall().andReturn(false).times(1);

        control.replay();

        assertFalse(loginManage.loginValid(name, pass));
        assertEquals("用户名或密码不正确", loginManage.getTip());
    }
```

```
    @Test
    public void testLoginValidNull() {
    String name = "";
    String pass = "";

    // mockDao.findByNameAndPass(name, pass);
    // EasyMock.expectLastCall().andReturn(false).times(1);

    control.replay();

    assertFalse(loginManage.loginValid(name, pass));
    assertEquals("用户名或密码不能为空",loginManage.getTip());
    }
```

完整的产品代码，LoginManageImpl：

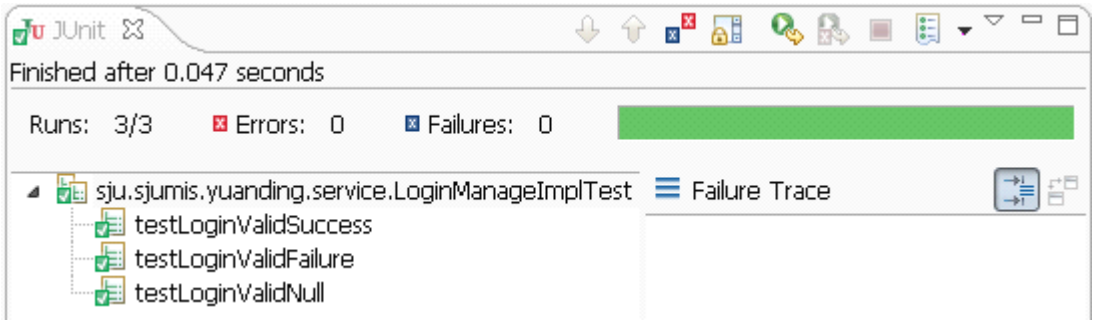
```
import sju.sjumis.yuanding.dao.UserDao;

public class LoginManageImpl implements LoginManage {
    private UserDao dao;
    private String tip;
    public LoginManageImpl(){
        //此处暂未实现，因为还没有写UserDao接口的实现类
    }
    public LoginManageImpl(UserDao dao){
        this.dao = dao;
    }
    public void setDao(UserDao dao) {
        this.dao = dao;
    }
    public String getTip() {
        return tip;
    }
```

```
    //输入校验
    public boolean inputCheck(String name,String pass){
        if(name == null || pass == null
                || name == "" || pass == ""){
            tip = "用户名或密码不能为空";
            return false;
        }
        return true;
    }
```

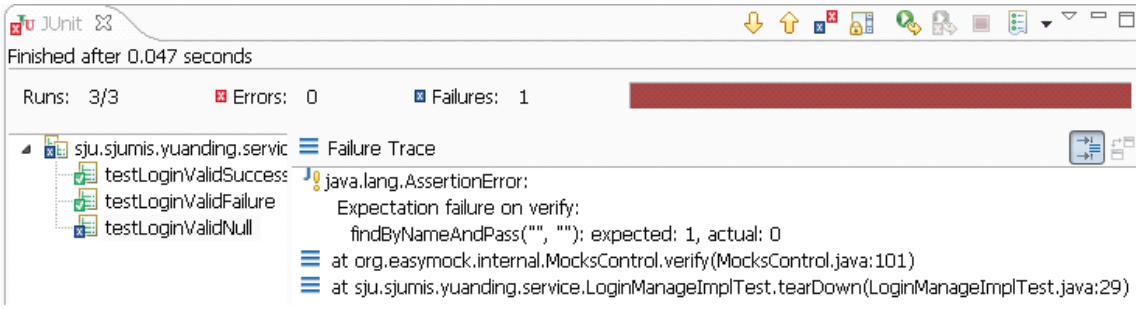

论文作者：陈峰
QQ：562116039

```
    //登录验证
    public boolean loginValid(String name, String pass) {
        name = name.trim();
        pass = pass.trim();
        if(inputCheck(name,pass)){
            if(dao.findByNameAndPass(name, pass)){
                tip = "成功登录";
                return true;
            }else{
                tip = "用户名或密码不正确";
            }
        }
        return false;
    }
}
```

运行测试，进度条显示绿色，通过：



注意：用户名密码为空的测试用例中，被注释的两行不能去除注释，否则运行测试时产生错误：



因为用户名密码为空时直接返回 false，并没有调用 dao 对象进行用户名密码验证。而 tearDown 方法中的 control.verify()一句表明将对 Mock 对象的方法的调用次数进行验证。若不将两句注释掉，则说明 findByNameAndPass 方法将必须被调用一次，但实际并没有调用此方法，所以测试便会报错。

实际应用中这一功能将是非常有用的，可以确保所有的事情都是如我们在测试中所预期

的进行下去的。

4.2 增加同类型测试用例

我们的每个测试方法中只有测试了一种情况，这样的测试并不全面，应该增加更多的测试用例，比如对于用户名密码为空的情况，我们应当测试三种可能：用户名密码皆为空、只有用户名为空、只有密码为空。

我们没有必要为每种可能的情况写一个新的测试方法，比如为“只有用户名为空”写一个 testLoginValidNameNull、为“只有密码为空”又写一个 testLoginValidPassNull，因为它们都是相似的情况，所以全部放到 `testLoginValidNull` 里就可以了。

因此我们修改 testLoginValidNull 如下：

论文作者：陈峰
QQ：562116039

```
public void testLoginValidNull() {
    String name;
    String pass;
    //mockDao.findByNameAndPass(name, pass);
    //EasyMock.expectLastCall().andReturn(false).times(1);

    control.replay();

    name = "";
    pass = "";
    assertFalse(loginManage.loginValid(name, pass));
    assertEquals("用户名或密码不能为空",loginManage.getTip());

    name = "";
    pass = "chenfeng";
    assertFalse(loginManage.loginValid(name, pass));
    assertEquals("用户名或密码不能为空",loginManage.getTip());

    name = "root";
    pass = "";
    assertFalse(loginManage.loginValid(name, pass));
    assertEquals("用户名或密码不能为空",loginManage.getTip());
}
```

很明显，此段测试代码中存在不必要的重复，两个断言分别出现了三次，却并没有任何变动，如果增加更多的测试用例，这种重复会更烦人，因此需要重构了：

```
public void testLoginValidNull() {
    String name;
    String pass;

    control.replay();

    String cases[][] = {
            {"",""},
            {"","chenfeng"},
            {"root",""}
    };
    for(int i=0; i < cases.length ; i ++){
        name = cases[i][0];
        pass = cases[i][1];
        assertFalse(loginManage.loginValid(name, pass));
        assertEquals("用户名或密码不能为空",loginManage.getTip());
    }
}
```

运行此测试，进度条显示绿色，成功。

我们将所有可能的输入值放入一个名为 cases 的数组当中，这样就分离出了断言的公

论文作者：陈峰
QQ：562116039

共结构，避免了不必要的重复。这里的期望输出值都是一样的，如果每种输入值对应了不同的输出值，只需要将输出值也放入 cases 数组中即可。

这样，同一类型的测试用例放到一个测试方法中，要增加一个新的测试用例，只需要在 cases 数组中添加一条数据即可。这是 Kent Beck 在《测试驱动开发》最后一个实例演示中给出的建议。

4.3 测试的方法没有返回值

以上的被测试方法有返回值，因此直接通过输入输出就可以编写单元测试了。但是我们经常会遇到这样的情形——要测试的方法没有返回值。

对此《JUnit Recipes》也提供了解决途径：

1. 如果要测试的方法没有返回值，那它一定有一些“可观测的副作用”，比如说改变了一个对象的状态，那我们就可以观测方法执行前后这个对象的状态变化，以此来设置断言，测试方法的行为。

2. 即使没有可观测的副作用，还可以使用另一个方法的返回值来测试这个方法。

3. 如果没有办法观测一个方法的副作用，那么就为了测试创建一个出来，将一个不可观测的副作用转化为可观测的。

论文作者：陈峰
QQ：562116039

第五章 数据库访问层 TDD 方案

对于普通的逻辑组件，因为它们只需要内存资源，所以通过输入输出即可编写出有效的单元测试。而对于进行数据库访问的 DAO 组件，则要复杂得多。因为它们严重依赖于底层数据库，以及 JDBC 的 API 或者 Hibernate 等 ORM 框架，因此增加了单元测试的难度。尤其是涉及到事务等复杂引用时更是如此。

5.1 数据库单元测试的几种可行方案

我们先总结当前主要的几种数据库单元测试编写策略，并比较各方优劣，然后探寻最适合于测试驱动开发的方案，以及最适合于当前项目即园丁网的解决途径。

(一) 直接使用真实数据库环境

这是最原始的方法，可能也是比较常用的方法，在单元测试的时候，连接真实的数据库并对其作出实际操作。但使用这种方法，弊端很明显：

1. 我们需要准备数据库环境，并准备初始数据。很多时候这是难以实现的，我们无法保证在我们进行 TDD 的工作时数据库已经准备就绪了。

2. 每次运行单元测试后，其操作将直接影响到下一次测试，破坏了单元测试的独立性，难以实现“即时运行，反复运行”单元测试的良好实践。

如果做些许改进，此方法也是可以使用的：

1. 利用 setUp()和 tearDown()方法做建立环境和清楚垃圾数据的工作。

2. 利用数据库事务处理，做回退。在测试方法中对数据库做了修改，但在测试出前回退，使修改不生效。

如今不少公司都在采用此种方法，因为它很简单，我们不需要学习更多的工具和技术，而只是采用传统的思维来写单元测试。本人现在所在公司专门给华为做外包，正是采用的此法。

(二) 使用 DbUnit 数据库测试组件

DbUnit 是对用数据库相关的应用提供的一个对 JUnit 扩展的测试组件，使用 DbUnit 进行测试时，测试脚本的框架和使用步骤与 JUnit 基本相同，DbUnit 只不过增加了专门来处理于数据库相关的测试功能。

DbUnit 的设计理念就是在测试之前，备份数据库，然后给对象数据库植入我们需要的准备数据，最后，在测试完毕后，读入备份数据库，回溯到测试前的状态；而且因为 DbUnit

论文作者：陈峰
QQ：562116039

是对 JUnit 的一种扩展，开发人员可以通过创建测试用例代码，在这些测试用例的生命周期内来对数据库的操作结果进行比较。

可见 DbUnit 只是在 JUnit 框架的基础上为我们自动地完成方案一中的需要手工来做的建立环境、清除数据、回退事务等工作。而且不管我们的 DAO 组件是使用 JDBC 实现的，还是使用的 Hibernate，使用 DbUnit 都一样可以解决。

Vincent Massol 在其经典书籍《JUnit IN ACTION》中提供了两种数据库整体单元测试策略：

前提条件是我们必须将数据库访问代码清晰地与业务逻辑代码区分开来。

(三) mock objects / 功能测试

为业务逻辑代码编写一个 mock objects 风格的单元测试。同样为数据库访问代码编写一个 mock objects 风格的单元测试。最后，写一个功能测试以确保当整个系统在一个配置过的环境中运行时它们可以正常工作。

(四) mock objects / 数据库集成单元测试 / 功能测试

为业务逻辑部分写一个 mock objects 风格的单元测试。不要用 mock objects 为数据库访问层代码编写单元测试。取而代之，可以为之编写 Cactus 的集成单元测试（可能会用一些 mock objects 测试失败的条件）。同样为端对端测试编写一个功能测试。

执行数据库集成单元测试是指连着一个数据库的情况下执行单元测试。我们需要使用两个框架：Cactus 和 DbUnit。Cactus 允许从容器内部进行测试，DbUnit 具有预先将数据装入数据库的能力，以及测试后将数据库的内容和参考数据相比较的能力。

因为要在容器内部运行测试，所以需要挑选一个 J2EE 容器和一个数据库，Vincent Massol 在书中使用的是 JBoss 和 Hypersonic SQL，同时他还使用了 Ant 工具来构建整个环境。关于数据库集成单元测试，《JUnit IN ACTION》一书第 11 章做了详细阐释和实例讲解，这里不再赘述。

5.2 各方案优劣比较

可见方案一是最简单的，但是需要准备真实的数据库，有时候这是很难实现的；而且连接真实的数据库会降低我们测试的速度，如果测试需要花费很长的时间，肯定会使得我们降低测试的频率，而这进而会影响到 TDD 的效果。

方案二是对方案一的改进，很多工作可以自动完成，不过还是需要牺牲一定的性能。

论文作者：陈峰
QQ：562116039

至于方案三，对数据库访问代码也进行全程 MockObject，这样事件录制的代码量会相当大，而且这并没有测试我们的代码是否与数据库正确的交互。

而方案四中，进行那种数据库集成单元测试并不是一件轻松的事情，我们需要做大量的准备工作，而且没有深厚的编程功底和丰富的开发经验只会弄巧成拙。最重要的是，我们是要进行测试驱动开发，因此是在没有产品代码的前提下写测试代码的，那我们该如何编写这种复杂的数据库集成单元测试呢？

这种复杂的集成单元测试一般适合于为已有代码编写单元测试，而不适用于测试驱动开发。特别是重构的时候，如果测试代码太复杂会很麻烦，而我们在 TDD 的过程中是需要不断重构的。我们的单元测试应当足够简单，只有单纯的对象测试才能更好地驱动开发和设计。

这种复杂的集成单元测试有时也是需要的，但不是在编码之前，而是在编码完成之后。用测试驱动开发并不是说在测试的驱动下完成了编码，并且通过了测试，之后就不需要再测试了，我们还是需要一些复杂的集成单元测试、功能单元测试以及集成测试、功能测试，只不过优先于编码的单元测试已经避免了大多数的错误，提高了代码的质量，从而极大地减轻了这些后期测试的负担。

5.3 数据库 TDD 解决方案

综合考虑以上几种方案的优缺点，结合 TDD 的具体实践，我总结出如下进行数据库访问层的测试驱动开发的解决方案：

1. 首先，我们要将底层数据访问操作与高层业务逻辑完全分离，提供一个 DAO 层，封装所有的数据库访问操作，并向上层的业务逻辑组件提供统一的数据访问接口。

2. 对业务逻辑代码，将其与数据库访问代码清晰地分离开之后，用 mock objects 来模拟数据库访问代码。这正是我们在业务逻辑组件的 TDD 方案中所使用的策略。

3. 实现真实的 DAO 组件的工作，应该是在系统的其他部分已经开发完成并且真实的数据库已经被创建之后才开始的。

4. 在 TDD 的过程中，要写尽量少而又完全够用的 DAO 组件。因为它们的单元测试确实是最复杂的，而且没有一个真正完美的解决方案，要么是牺牲性能，要么是相当繁琐。所有千万不要在每个需要访问数据库的地方直接访问数据库。

5. 对于实际的数据库访问代码，有两种解决方法：

1) 隔离开数据库，同样编写 mock objects 风格的单元测试，因为我们的重点是“测试需要测试数据的代码”[4]，以 mock objects 风格的单元测试来驱动我们的 DAO 组件的开发。

2) 连接实际的数据库，使用 DbUnit 作为设置数据库前后状态的组件，以确保外部因素不影响测试对象，以及测试之间不相互影响。这种方法提供了更广的测试覆盖，但也需要

论文作者：陈峰
QQ：562116039

更多的设置。

6. 开发完成之后，我们还要写一个功能测试，以确保当整个系统在一个配置过的环境中运行时这些数据库访问代码可以正确地工作。

于是我们工作的中心就集中到了少量的数据访问层接口的实现类上了，有mock objects和 DbUnit 两种策略供选择，可以根据具体情况选择最优的方案。国内很多人普遍认为这里使用 Mock 没有任何意义，因为那样并没有测试与数据库是否正确交互，我个人也倾向于此观点。

当我们已经实现了系统的其他部分，而且已经创建了真实的数据库，那我们就可以开始实现真正的 DAO 组件了。实现这 DAO 组件的时候不会影响到其他部分，因此可以独立完成，只需要根据前面已定义的 UserDao 接口来实现即可，我们可以用 JDBC，也可以用 Hibernate，这都不会对上层业务逻辑造成任何影响。

5.4 DAO 组件 TDD 案例演示

例如在园丁网项目中，对于用户注册相关的所有数据库访问，我们提供了一个 **UserDao** 接口；然后写一个实现类，比如我们要用 JDBC 实现，则可以命名为 **UserDaoJdbcImpl**。这里将选择 DbUnit 作为数据库访问代码的测试工具。

园丁网使用的数据库名为 teacher，存放登录用户信息的表为 siteUser，但由于 siteUser 表中字段比较多，演示当中我将进行简化，只假设其中有三个字段：id、name、password。生成此表的 SQL 语句为：

```
use teacher

create table siteUser(
id int primary key,
name nvarchar(20) not null,
password nvarchar(20) not null)
```

DAO 接口为 `UserDao` ：

论文作者：陈峰
QQ：562116039

```
import java.util.List;
import sju.sjumis.yuanding.model.User;

public interface UserDao {

    //根据用户名密码查找用户
    public boolean findByNameAndPass(String name,String pass);
    //根据用户名查找用户
    public boolean findByName(String name);
    //插入新的用户
    public int insert(User user);
    //查询所有用户列表
    public List<String> findAllUsers();

}
```

用户模型为 **User** ：

```
public class User {
    int id;
    String name;
    String password;

    public int getId() {..
    public void setId(int id) {..
    public String getName() {..
    public void setName(String name) {..
    public String getPassword() {..
    public void setPassword(String password) {..
}
```

下面将演示两个具有代表性的功能的实现，一个是查询所有用户名列表，另一个是插入一个新的用户。

设计单元测试用例：

测试用例	输入条件	期望结果
testFindAllUsers	siteUser_initial.xml	返回列表长度为 2；返回列表与测试期望列表一致
testInsertSuccess	siteUser_initial.xml	返回受影响行数为 1；测试实际数据与测试期望数据一致

论文作者：陈峰
QQ：562116039

创建测试用的数据：

测试初始数据文件 siteUser_initial.xml

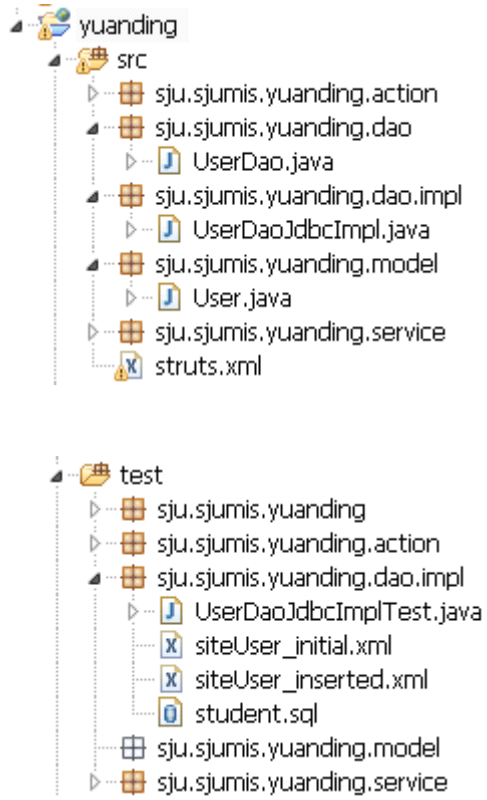
```
<?xml version="1.0" encoding="gbk"?>
<dataset>
    <siteUser id="1" name="chenfeng" password="05051073" />
    <siteUser id="2" name="yunzhu" password="05709118" />
</dataset>
```

插入新用户成功后的预想数据文件 siteUser_inserted.xml

```
<?xml version="1.0" encoding="gbk"?>
<dataset>
    <siteUser id="1" name="chenfeng" password="05051073" />
    <siteUser id="2" name="yunzhu" password="05709118" />
    <siteUser id="3" name="root" password="chenfeng" />
</dataset>
```

这两个数据文件与测试类放在一起，即 test 源代码目录下的相应的包下。

为了容易理解，先展示一下该模块实现之后的源代码目录树结构：



论文作者：陈峰
QQ：562116039

创建测试脚本框架：

首先创建测试类，并继承 **org.dbunit.DatabaseTestCase** 类。继承 DatabaseTestCase 必须实现 getConnection()方法和 getDataSet()方法。

执行每个测试用例前会调用 `getConnection()` 方法取得 DbUnit 用的数据库连接，随后调用 `getSetUpOperation()` 设置数据库测试前的状态，再调用 `getDataSet()` 方法装载测试用例的测试初始数据，接下来执行测试用例的测试脚本，在执行测试用例之后，调用方法 `getTearDownOperation()` 设置数据库测试后的状态。我们没有实现 `getSetUpOperation()` 和 `getTearDownOperation()` 方法，因此会执行默认的操作。[6]

创建我们的测试类 UserDaoJdbcImplTest：

```
import java.io.*;
import java.sql.*;
import java.util.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.Assert;
import org.dbunit.Assertion;
import org.dbunit.DatabaseTestCase;
import org.dbunit.database.DatabaseConnection;
import org.dbunit.database.IDatabaseConnection;
import org.dbunit.dataset.IDataSet;
import org.dbunit.dataset.ITable;
import org.dbunit.dataset.xml.FlatXmlDataSet;

import sju.sjumis.yuanding.dao.UserDao;
import sju.sjumis.yuanding.dao.impl.UserDaoJdbcImpl;
import sju.sjumis.yuanding.model.User;
```


论文作者：陈峰
QQ：562116039

```
public class UserDaoJdbcImplTest extends DatabaseTestCase {
    UserDao dao;

    public UserDaoJdbcImplTest(String s) {
        super(s);
    }

    @Before
    public void setUp() throws Exception {
        super.setUp();
        dao = new UserDaoJdbcImpl();
    }

    @After
    public void tearDown() throws Exception {
        super.tearDown();
    }

    protected IDatabaseConnection getConnection() throws Exception {
        Class.forName("net.sourceforge.jtds.jdbc.Driver");
        String uri = "jdbc:jtds:sqlserver://localhost:1433/teacher";
        Connection jdbcConnection = DriverManager.getConnection(uri,
                "sa","123456");
        return new DatabaseConnection(jdbcConnection);
    }

    protected IDataSet getDataSet() throws Exception {
        return new FlatXmlDataSet(new File(
            "test/sju/sjumis/yuanding/dao/impl/siteUser_initial.xml"));
    }
}
```

查询用户名列表：

实现查询用户名列表的测试用例 testFindAllUsers：

```
@Test
public void testFindAllUsers() {
    List<String> list = dao.findAllUsers();
    Assert.assertEquals(2, list.size());
    List<String> list_exp = new ArrayList<String>();

    list_exp.add("chenfeng");
    list_exp.add("yunzhu");
    Assert.assertTrue(list_exp.equals(list));
}
```


论文作者：陈峰
QQ：562116039

实现相应功能代码：

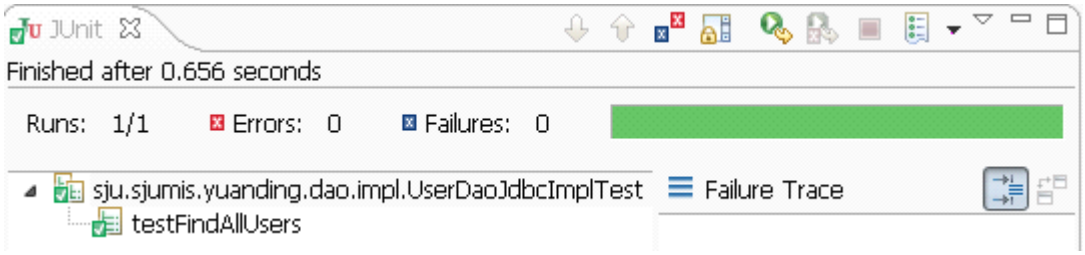
在构造函数中取得数据库连接，只实现了查询用户名列表的功能代码，其他几个功能未实现，因此以// TODO Auto-generated method stub 标识，并返回一个常量，这是由 MyEclipse 为我们自动完成的。

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

import sju.sjumis.yuanding.dao.UserDao;
import sju.sjumis.yuanding.model.User;


public class UserDaoJdbcImpl implements UserDao {

    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

    public UserDaoJdbcImpl() {
        try {
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            String uri="jdbc:jtds:sqlserver://localhost:1433/teacher";
            con = DriverManager.getConnection(uri, "sa", "123456");
        } catch (ClassNotFoundException e) {
            System.out.println("驱动未找到: \n" + e.getMessage());
        } catch (SQLException e) {
            System.out.println("连接数据库错误: \n" + e.getMessage());
        }
    }


/*
 * 查询用户名列表
 */
public List<String> findAllUsers() {
    List<String> list = new ArrayList<String>();

    try {
        pstmt = con.prepareStatement("select name from siteUser");
        rs = pstmt.executeQuery();

        while (rs.next()) {
            list.add(rs.getString(1));
        }

    } catch (SQLException e) {
        System.out.println("查询用户名列表错误: \n" + e.getMessage());
```


论文作者：陈峰
QQ：562116039

```
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                    pstmt = null;
                }
                if (con != null) {
                    con.close();
                    con = null;
                }
            } catch (SQLException e) {
                System.out.println("关闭对象错误: \n" + e.getMessage());
            }
        }
        return list;
    }


    /*
     * 插入新的用户
     */
    public int insert(User user) {
        // TODO Auto-generated method stub
        return 0;
    }

    public boolean findByName(String name) {
        // TODO Auto-generated method stub
        return false;
    }

    public boolean findByNameAndPass(String name, String pass) {
        // TODO Auto-generated method stub
        return false;
    }

}
```

运行测试：



论文作者：陈峰
QQ：562116039

插入新的用户：

实现插入新用户成功的测试用例 testInsertSuccess：

```
@Test
public void testInsertSuccess() throws SQLException, Exception {
    User user = new User();
    user.setId(3);
    user.setName("root");
    user.setPassword("chenfeng");

    int i = dao.insert(user);
    Assert.assertEquals(1, i);

    IDataSet actualDataSet = getConnection().createDataSet();
    ITable actualTable = actualDataSet.getTable("siteUser");

    IDataSet expectedDataSet = new FlatXmlDataSet(new File(
            "test/sju/sjumis/yuanding/dao/impl/siteUser_inserted.xml"));
    ITable expectedTable = expectedDataSet.getTable("siteUser");

    //这是DbUnit提供的方法，可以断言测试期望数据和测试实际数据是否一致
    Assertion.assertEquals(expectedTable, actualTable);
}
```

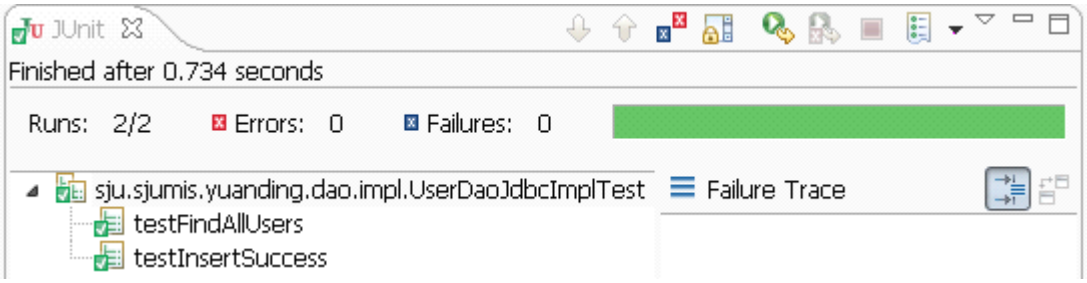
实现相应功能代码：

```
/*
 * 插入新的用户
 */
public int insert(User user) {
    int i = 0; // 受影响的行数

    try {

        pstmt = con.prepareStatement(
            "insert into siteUser(id,name,password) values(?,?,?)");

        pstmt.setInt(1, user.getId());
        pstmt.setString(2, user.getName());
        pstmt.setString(3, user.getPassword());

        i = pstmt.executeUpdate();

    } catch (SQLException e) {
        System.out.println("插入记录错误：\n" + e.getMessage());
```


论文作者：陈峰
QQ：562116039

```
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                    pstmt = null;
                }
                if (con != null) {
                    con.close();
                    con = null;
                }
            } catch (SQLException e) {
                System.out.println("关闭对象错误: \n" + e.getMessage());
            }
        }
        return i;
    }
```

再运行测试：



测试成功了，到此，查询用户名列表和插入新用户两个功能点的数据库访问代码便在 JUnit+DbUnit 单元测试地驱动下开发完成了。

注意：在比较测试实际数据与测试期望数据是否一致的时候，要注意数据记录的顺序！因为插入一条记录后，在数据库中会按照主键自然排序，所以测试期望数据文件中的各条记录也必须按照主键排序。否则，从数据库中读出的数据记录顺序与测试期望数据文件不一致，断言就会认为它们不一样，所以就认为失败。这是由 Assertion 自身的特性所造成的。

5.5 阶段性小结：

至此，我们就完成了园丁网的登录注册功能，但我在上面只展示了一部分，我截取了其中具有代表性的和比较简单易懂的部分作了演示。

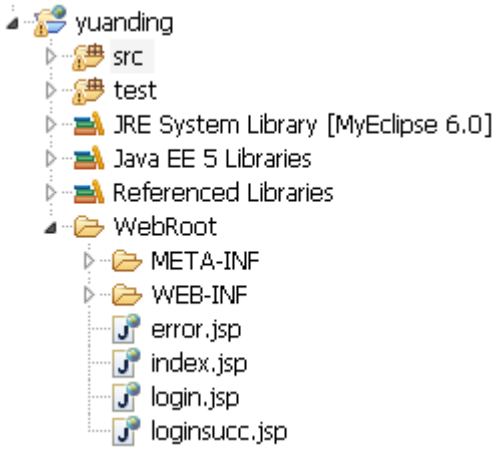
论文作者：陈峰
QQ：562116039

在测试驱动之下完成了这部分功能的开发之后，或者是同时，需要完成相应的页面显示部分。开发过程中我们只需要构建出最简单的页面，不需要考虑风格样式和美观，那应当在开发工作全部完成后去做，而且多数时候是由专门的页面美工人员负责的，程序员只应负责业务逻辑部分。

当然，开发过程中我们还需要进行相关配置，主要是 `web.xml` 和 `struts.xml` 两个配置文件。

下面将展示只完成了登录功能时的相关页面和配置文件：

1. 项目的整体目录树结构：



2. **`web.xml`** 配置文件：

论文作者：陈峰
QQ：562116039

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

3. **struts.xml** 配置文件：

```xml
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="yuanding" extends="struts-default">
        <action name="Login"
            class="sju.sjumis.yuanding.action.LoginAction">
            <result name="error">/error.jsp</result>
            <result name="success">/loginsucc.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
    </package>
</struts>
```

完成了显示页面和相关配置，再配置 Tomcat 服务器，然后就可以启动服务器进行用户真实场景下的验证了。这正是我们没有接触测试驱动开发的时候所经常做的事情——每完成一点功能就启动服务器进行验证，有错误就找到它并一个个解决，运行一遍下来没有错误就 OK 了。

首先，在登录页面输入用户名和密码：

论文作者：陈峰
QQ：562116039

点击登录按钮后转到登录成功页面：

可见，我们使用测试驱动的方式开发了一个最简单的 Struts 2 应用（只有登录功能），并且它已经通过真实场景下的检验了。

论文作者：陈峰
QQ：562116039

第六章 实践 TDD 的其他技能

使用以上技巧已经可以进行测试驱动开发的部分实践了，但要在一个真实的项目中娴熟地运用 TDD 技术，还有其他一些需要掌握的技能。

6.1 测试代码的组织

我们在软件开发的过程中，会很注重源代码的组织，我们会将类分门别类放到不同包中，并且取一个浅显易懂的名字，从而增强整个项目的可读性，使别人拿到此项目不至于无从下手，如果组织杂乱甚至连自己都会看得云里雾里，特别是过些时日之后。

组织我们的测试代码同样重要：

首先是单个测试类的命名，通用的规则是——在相应被测试的对象名后加上“Test”后缀作为相应测试类的名字。例如要测试的类为 LoginAction，那么测试类就可以命名为 LoginActionTest。

其次是测试方法的命名。测试方法应以“test”为前缀，加上被测试的方法名。如果需要为一个方法建立多个测试用例，那么每个测试方法名后面还应该加上一个后缀，能够明确表明我们的目的。

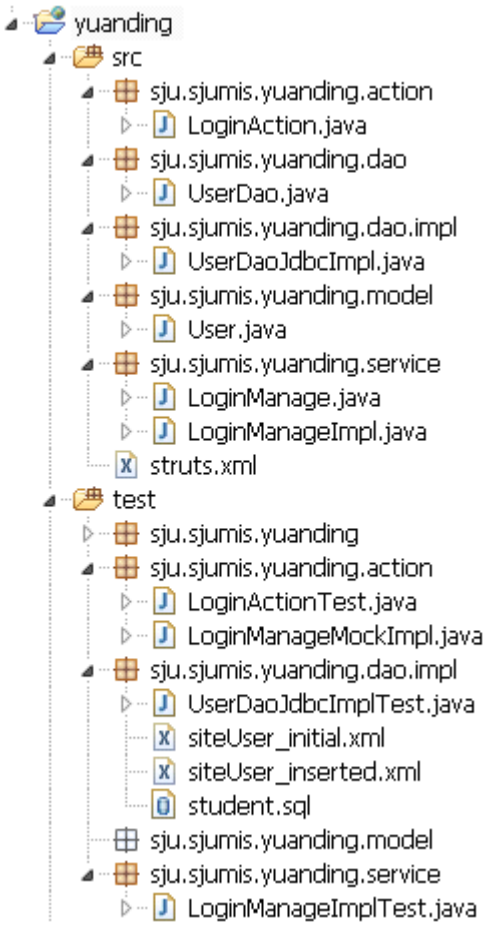
例如，要测试 LoginManageImpl 中的方法 LoginValid，有三种情况：用户名密码正确、用户名密码错误、用户名密码为空。那我们就写三个测试方法，分别测试这三种情况，它们是 testLoginValidTrue、testLoginValidFalse、testLoginValidNull。

然后是该将测试代码放到哪里。我们可以将测试代码与产品代码放到一起，但这会造成组织的杂乱。我们也可以将测试代码放到不同的包中，但更通用的做法是“同一个包，不同的目录”[3]。

也就是为测试代码创建独立的源代码目录，但其实与产品代码还是处在同一个包中的，这样做更容易管理，可以提供很多的方便，比如分发产品时可以将测试代码与产品代码一起发布，也可以只发布产品代码，而不需要做太多的工作。

例如，到我们只完成登录注册功能为止，园丁网项目的源代码目录树结构如下（在 MyEclipse 中的显示）：

论文作者：陈峰
QQ：562116039



源码目录 src 用于存放产品代码，而test 则专门存放单元测试相关的源代码。在前面的演示过程中，我正是采用这种哦个方式组织所有源代码的。

6.2 自由地组合测试

一个项目中我们会写大量的单元测试，难道要每个单元测试都单独地手动地去执行吗？当然不行。JUnit 框架提供了一个 TestSuite 核心类，可以便捷地组合多个相关测试，既可以将多个单元测试归入一组，还可以将多个单元测试中的部分测试用例归入一组，而且可以将多个 Test Suite 组合起来。一般将这种机制称为测试套件，顾名思义，就是里面包含了若干个测试。

Test Suite 是 JUnit 中测试的最小单位。其实我们在运行单个测试的时候，JUnit 也是执行的 Test Suite，虽然我们并没有创建 Test Suite，是 JUnit 自动为我们创建的，缺省的 Test Suite 会扫描我们的测试类，找出所有的以“test”开头的“public void testXXX()”形式的方法，并为之创建一个 Test Case 的实例。这也是为什么我在前面说测试方法应以 test 为前缀的原因。

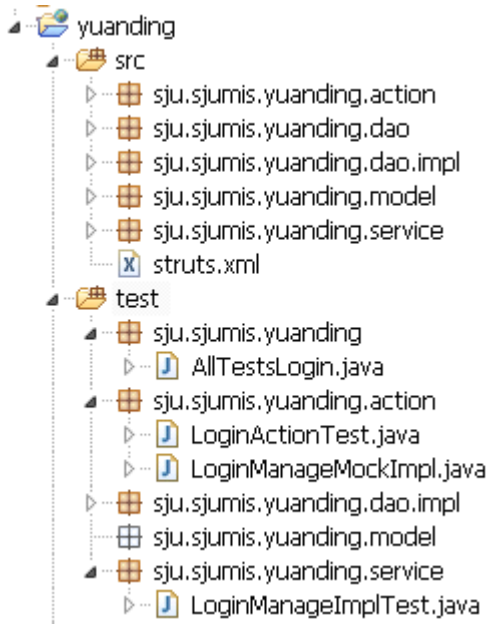
论文作者：陈峰
QQ：562116039

6.2.1 组合相关测试

首先要使我们的测试类都是继承自 junit.framework 包中的 **TestCase** 类。

例如我们有 **LoginActionTest** 和 **LoginManageImplTest** 两个测试类，都继承自 TestCase 类，我们可以将它们组合起来，一次就能运行所有的测试用例。我们将这个 Test Suite 命名为 **AllTestsLogin**：

源代码目录树：

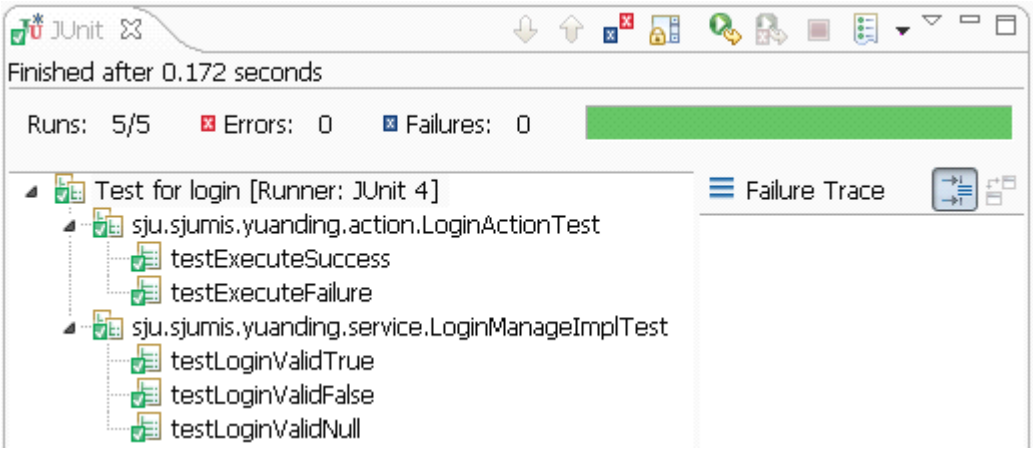


组合这两个测试类中所有测试用例的测试套件 **AllTestsLogin** ：

```
import junit.framework.Test;
import junit.framework.TestSuite;
import sju.sjumis.yuanding.action.LoginActionTest;
import sju.sjumis.yuanding.service.LoginManageImplTest;

public class AllTestsLogin {
    public static Test suite() {
        TestSuite suite = new TestSuite("Test for login");
        // $JUnit-BEGIN$
        suite.addTestSuite(LoginActionTest.class);
        suite.addTestSuite(LoginManageImplTest.class);

        // $JUnit-END$
        return suite;
    }
}
```


论文作者：陈峰
QQ：562116039

运行 **AllTestsLogin** ，它同样是作为 JUnit Test 来运行的：



可见运行 AllTestsLogin 也就运行了 LoginActionTest 和 LoginManageImplTest 中的所有测试用例，与我们单独运行每个测试程序的效果是一样的，但这种方式组合了多个测试，只需要一次运行。

6.2.2 组合全部测试

有时候我们希望一次性执行所有的测试，那么可以用 TestSuite 组合系统中的全部测试：

一般可以为每个包（不包含其他包）建立一个名为 AllTests 的 test suite，来组合该包下的所有测试类；对于包含其他包的包，它的 AllTests 不仅组合了该包下的所有测试类，还组合了所有子包的 test suite；往上依次类推，每个包就组合了该包下的所有测试，包括其子包中的，和子包的子包中的；而一个项目最顶端的包，它的 AllTests 就组合了该项目下的所有测试，因此只要运行这个 AllTests，我们就可以运行所有的测试了。

这样，整个项目的测试作为一个测试集，或者一个模块的所有测试作为一个测试集，我们就可以一次运行全部测试，然后查看哪些地方有错误，并一个个解决，等到进度条显示绿色，那说明全部的测试已经通过，那我们就可以向源代码库（比如 CVS 或者 TortoiseSVN）提交代码了。

这实现起来就跟上面组合相关测试一样简单，只要按照包层次结构逐级向上收集所有测试即可。

而且当我们要收集一个包下的测试类时，MyEclipse 可以自动帮我们找出所有的测试类（但不能递归地找出子包中的）。我们只要在相应包上单击右键，选择“新建”菜单中的“Other”，在跳出的窗口中选“JUnit Test Suite”即可，然后就可以选择要收集该包下的哪

些测试类了。要运行此 Test Suite，只要在该 Test Suite 类上单击右键，选择“Run As”菜单中的“JUnit Test”即可运行了。

将大量的单元测试组合为一些完整的测试套件，并持续地运行它们，就可以随时地监视整个系统，一有问题就立马报告，因此我们可以尽早地发现和修正大量的 Bug，这也极大地增强了我们在开发过程中的信心。[8]

6.3 自动化测试与构建

在以上的工作中，我们一直是使用的集成开发环境，所有的操作都是在 MyEclipse 6.0 当中完成的，从写测试代码到运行测试，从到编写产品代码，到进行重构，再到编译整个项目。集成开发环境的确是个强大的全功能的工具，特别是在 Eclipse 当中，通过插件几乎可以集成所有 Java 领域的框架和工具，所以我们才有了 MyEclipse。

但是我们不能永远依赖集成开发环境，正像《JUnit Recipes》一书中所说的：“我们都希望集成开发环境使我们的生活变得更轻松，但是集成开发环境的某个功能使你变懒的时候，你就有需要担心的事情了。集成开发环境是个工具，而不是离不开的拐杖；在需要的时候，应该能够脱离集成开发环境”。

的确，很多情况下我们是不能使用集成开发环境的，比如要实现自动化的可重复的编译过程，比如需要实现自动化的测试和自动化构建，又比如项目需要移植到不同平台上，而这些平台上可能没有集成开发环境可以使用等等。

这时候，Ant 可以帮我们的大忙。

Ant 原名 Another Neat Tool，是在 Java 程序员中已经成为业界标准的自动化构建工具。

使用 Ant，我们可以轻松地实现以下功能：

1. 可以配置和启动编译器
2. 在指定的目录结构里递归地编译其中所有的 Java 类
3. 可以实现自动测试，它会自动找出要运行的测试
4. 将测试结果输出为可预制的 HTML 等格式的文件
5. 可以从 CVS 等源代码管理系统获取最新源代码
6. 通过简单地配置命令彻底让编译和部署过程自动化
7. 使用构建文件完成所有的功能，在所有平台上都有同样表现
8. 它还可以生成 Java 文档，可以查询数据库

论文作者：陈峰
QQ：562116039

总之，Ant 的功能相当强大。准确地说，Ant 不仅仅是一个工具，更是一个运行工具的框架[3]。因此 Ant 和 JUnit 共同成为了 Java 程序员必备的两大工具，尤其是对于实践测试驱动开发的敏捷人士。

要使用 Ant 进行自动化的编译或构建，我们要做的就是为一个项目建立一个构建文件（buildfile），默认情况下被命名为 build.xml。Ant 会使用此构建文件来管理整个构建过程，这个构建文件可以包含多个目标，封装不同的任务：编译源码、执行测试、输出测试结果报告以及打包部署等。

Ant 对 JUnit 的全面集成，则使得一切都变得更加完美。结合上面提及的 Ant 的种种强大的功能，我们可以很容易地实现自动化单元测试。难怪有人会说 JUnit、Ant 是 Java 自动化的绝代双骄呢。

通过 Ant，我们不仅可以使自动化单元测试，更可以实现自动化构建，做到了这两点，我们就可以进一步实施每日构建或持续集成的实践了。这样，我们就又向“敏捷开发”跨进了一大步了。

论文作者：陈峰
QQ：562116039

第七章 总结与展望

测试驱动开发是一种提倡测试优先，并用测试来驱动整个开发工程的新型软件开发方法，可以让我们得到简洁可用和高质量的代码，并加速开发过程。Struts 2 则是当今 Java 领域最优秀的 Web MVC 框架之一，可以极大地减少工作量、提高开发效率。以 Struts 2 作为框架并采用 TDD 的方式，可以极大地提高我们开发 Java 企业级应用的效率，又能得到高质量的软件，同时可以减轻我们的精神压力使我们可以快乐地工作。

本文就是致力于探讨用 TDD 的方式开发 Struts 2 应用的最佳解决方案，限于本人水平和时间有限，可能在某些问题已经作出了完满的解答，在某些问题上则还有所欠缺，甚至有些应当涉及的问题本文还没有提出解决方案。

7.1 本文主要工作成果

在此列举本文的主要工作成果，如下：

1. 测试驱动开发技术的产生、原理、优势和发展现状
2. 单元测试和重构，以及 TDD 中单元测试和重构的原则
3. 怎样用 TDD 的方式开发 Struts 2 的 Action，以及在 Action 中调用业务逻辑组件和访问 Servlet API 时的解决途径
4. 怎样用 TDD 的方式开发业务逻辑层（使用 JUnit 和模拟对象框架 EasyMock）
5. 怎样用 TDD 的方式开发数据库访问层，展示了各种现有方案然后寻求最佳的，其实最适合当前系统的就是最佳的
6. 介绍在项目中实践 TDD 还需要掌握的其他技能：如何组织测试代码，怎样组合测试，以及实现自动化单元测试
7. 本文结合“三江园丁网”的开发过程，以简单的功能实现进行演示，在实践中摸索 TDD 解决途径同时进行验证

7.2 本文存在的欠缺之处

限于本人水平以及时间有限，还有资料不充足等原因，本文尚有不少欠缺之处，现将我所知道的最主要的几点列举如下：

1. 对 Struts 2 的 Action 的单元测试，只是将其看做普通的 POJO 来处理，没有测试其他部分如配置文件等，只演示了方案一，对方案二没有展开来讲，更没有作案例演示。因为我对 Struts 2 源码了解程度不够，同时此方面资料相当有限，只在外国的博客上找到三四篇文章，并且还比较复杂，所以不易施行。
2. 对于自动化单元测试，本文只是介绍了可以使用 Ant 实现，也没有详细讲解如何实现，更没有进行实例演示。因为本文主要目标是探究 Struts 2 应用的 TDD 解决方案，而对于如何使用 Ant 实现自动化单元测试，则是另一个很复杂的问题，完全可以另写一篇论文详

论文作者：陈峰
QQ：562116039

细论述。同时此方面文献资料比较丰富，而本人又学习得不够深入，难以摸索出新的东西。

另外在学习 Struts 2 源码的过程中，还发现 Strtus 2 框架本身所存在的一个问题：源码的包组织比较乱。

7.3 Struts 2 框架的一个问题

因为 Strtus 2 由 WebWork 2 发展而来，其实只是改了个名字，其核心还是调用了很多 WebWork 2 的包。所以一会儿在 Strtus 2 的包里面，一会儿又跳到 WebWork 2 的包中了，这造成了包组织命名的不一致，并且给我们学习其源代码增加了难度，相信 Apache 组织会很快对此作出改进，进行统一化。

7.4 实践 TDD 的具体流程

在学习了相关知识和进行了园丁网项目的实践之后，通过对前人经验的整合再加入自己的个人创造，我总结了在项目中实践 TDD 的具体流程，如下：

1. 首先，当然要先进行需求分析和设计。同客户和分析人员交流，确定相关的用户需求，并分解为一个个 User Story，记录在卡片上。
2. 采用极限编程中的 User Story 能够更好地匹配体现 TDD 的优势。
3. 结合你要完成的业务功能的需求，写一个包含所有你认为必须要编写的测试的清单。
4. 不是一次全部提出，而是通过下面步骤的迭代逐步加入新的工作。
5. 针对某一具体功能设计测试用例、编写测试代码，一开始这个测试程序甚至不能编译。
6. 测试代码刚开始只是很简陋的，在后面步骤的迭代中逐渐加入新的测试用例，完善测试程序。
7. 运行所有测试，发现新增的测试不能通过，因为相应的功能代码还根本没有写呢。
8. 快速地添加用于实现此功能的简陋的代码，以尽快地让测试程序可运行。
9. 这时甚至可以使用一些不合情理的方法，比如采用伪实现，直接返回一个常量。
10. 运行所有的测试，如果这个测试未通过，重复上一步，通过则继续下一步。
11. 逐步编写合格的一般化的功能代码，持续进行代码重构，以消除重复设计，优化设计结构，同时要始终保证测试通过。要保证添加或改动代码之前先添加或改动相应的测试程序，使所有的重构都被测试程序所支持。因为不管做什么，你都要让测试程序来告诉你是对是错，靠自己的逻辑分析和判断得花费多得多的时间和精力。
12. 重复以上步骤，循环完成所有业务功能的开发。

论文作者：陈峰
QQ：562116039

7.5 未来展望

此毕业论文完成之后，我将很快迎来大学生涯的终结，甚至是学生生涯的终结，从此将步入社会走上工作岗位。很庆幸可以找到一个软件开发的职位，在未来走上程序员的岗位之后，我还会继续从事本文主题方面的深入学习和研究，我会逐步解决本文所存在的欠缺之处，并着重探讨以上所提及的 Action 的 TDD 解决方案二和自动化单元测试，希望在研究 TDD 以及 TDD 与 Struts 2 结合的路上走得更远，并在此领域有所突破、有所贡献。

相信在未来几年内，在 Java 框架领域，在不断改进和完善的过程中， Struts 2 框架会很快取代 Strtus 1 成为 Java 框架中的老大，成为事实上的标准，当然，它还需要与 Spring 强强联手，这两者并没有同行相妒，相反两者结合将更加强大。Java 开源领域的群雄并起，使我们有了更多的选择的余地，但同时加大了 Java 程序员的学习量，希望 Struts 2 和 Spring 能够一起改变这样的现状。

而测试驱动开发，则会在软件工程领域刮起一阵旋风，作为软件开发方法学的革命性产物，它必然会备受推崇，成为越来越多程序员的制胜法宝，并成为我们 80 后一代软件开发人员的时代特征。

而将这两者结合一样意义非凡，虽然本人水准有限，难以实现重大突破，然而也是想在此细分领域能有所贡献，能分一杯羹。

论文作者：陈峰
QQ：562116039

结束语

经过近半年的努力，我的毕业设计终于完成了，虽然不一定能称得上圆满，但至少基本达到了目标，最重要的是——我努力了收获了，而且我也尽力了。

从寒假开始准备工作，查阅相关文献资料，整体把握毕业设计相关知识体系，构思开题报告，并且完成了开题报告的雏形。

然后在本学期未开学之前就到了软通无锡分公司，参加集中培训和实习，之后毕业设计的绝大部分工作皆是在公司完成的。在这里，基本可称得上是与世隔绝，而且面临着不小的困难：这里没有老师可以耐心地为我指点迷津，这里更没有图书馆为我提供浩瀚的文献书籍，公司电脑没有连网因此又少了互联网这一强大的助手，另外没有带电脑因此只有在公司上班时间才有电脑可用。

好在从学校带来了几本书，有自己买的 Struts 2 的书，有图书馆借的两本 TDD 的，以及两本关于 JUnit 和一本关于 J2EE 测试的是从杨老师处借的，于是所有的希望就集中于这仅有的几本书了。感谢杨老师推荐的这几本书，它们成为了支持我走下去的力量，成为了我完成毕业设计的主要知识来源。

而遇到这几本书没有涉及到什么问题，需要下载东西或者查阅资料，就全部列在一张纸上，然后等到周末去一次网吧，一次性下载所有的东西，因此网吧竟也成了一处重要的工作场所了。

由于对 TDD 从未耳闻，因此要学习的知识确实是不少，而这近半年时间的学习，主要是围绕从学校带来的那几本书，同时这个历程拥有一个清晰的脉络：

1. 刚开始（主要是寒假）通过大量网络文献以及《测试驱动开发的 3 项修炼》前面部分，大致了解了测试驱动开发的技术。

2. 之后重点关注单元测试的学习，掌握了单元测试就掌握了 TDD 的一半。此阶段以《JUnit IN ACTION》一书为主，学习了 JUnit 的原理和使用，然后又通过网上的一些资料学习了模拟对象工具 EasyMock。

3. 接着又回过头来学习 TDD 的原理。有了一些基础之后，终于可以看懂 Kent Beck 的《测试驱动开发》了，不像以前看得云里雾里，而是如醍醐灌顶，受益匪浅，总是不禁感慨大师之作果然不同凡响。

4. 再然后，通过《测试驱动开发的 3 项修炼》和其他书籍以及网络资料，系统地学习

论文作者：陈峰
QQ：562116039

了从单元测试到重构，再到自动化测试与每日构建，还有需求建模等各方面的知识。

5. 在具备了充足的知识准备以后，就可以大胆地开始 TDD 实践了。于是开始结合园丁网项目，截取最简单的一部分功能，探讨如何用 TDD 的方式并以 Struts 2 作为框架去实现它们。种种难题当然就在此阶段浮出水面，而真正的成果也将在此阶段诞生。

6. 在实践的过程中遇到了真正的问题，比如如何测试 Action，在 Action 中调用外部对象如何解决，访问 Session 又该当如何，然后还有业务逻辑组件的实现、DAO 组件的实现等等。

7. 在遇到这些问题的时候，《JUnit Recipes》就像一盏明灯，它收录了 137 个问题和解决方案，因此大部分问题都可以通过这本书中的技巧圆满解决。而在这之前我竟觉得这本书一无是处。

8. 对于 Action 的 TDD 方案，则查看了大量的网络资料，而且清一色是英文的。只有外国的一些专业博客上，才可以找到有价值的文章，因此实在不得不佩服人家确实比我们先进的多。

9. 在这个过程中，我一边学习，一边思考，一边实践，一边探索，一边撰写论文。

10. 再然后呢，再然后自然就到我写这段结束语了，于是一分耕耘一分收获，我们毕业设计已接近完成了。

在完成毕业设计的过程中，我更加相信——世上无难事，只怕有心人。

从第一次看到老师给的毕业设计课题题目时的一头雾水和茫然无措，到现在基本完成毕业设计，我是一步一个脚印走过来的，我不断学习新的知识和技术，并且付诸实践，我在项目开发实践中摸索解决方案，并且不断发现问题，再解决问题。

在这近半年时间里，我不仅完成了毕业设计课题，还学到了很多的知识和技术，更重要的是，我培养了自主学习新知识和新技术的能力，以及发现问题和自主解决问题的能力。前者是自主学习能力，后者是创新能力，我想，这两种能力，才是作为一个身在学习型社会的二十一世纪人才，作为一个新世纪的软件开发人员最重要的素质。

论文作者：陈峰
QQ：562116039

致谢

在提交论文之际，我需要感谢一些人。

感谢父母的话就不说了，那种伟大的亲情，是无法用语言来形容的，在她的面前，世间一切文字都显得苍白无力。所以我就不废话了，不过还是得感谢一下自己，毕竟我也在没有网络没有图书馆没有老师没有同学的，如此恶劣的环境下完成了这篇论文，虽说无甚作为、缺乏创新，不过写下这惶惶三万字已经很不容易啦。另加一句——我叫陈峰， QQ 号是 562116039，有志同道合的朋友可以找我切磋切磋，并为我指点迷津，先谢谢啦 O(∩_∩)O~

好了，下面开始进入正题。

首先，当然是杨老师，但我最感激的不是杨老师给我的无微关怀和细心帮助，而是杨老师给我确定了这个毕业设计课题。这是一个发展并不十分成熟的细分领域（至少我无法找到任何一篇与此相关的文章，或许是很多人已经有所研究但不想公开——这却是更大的问题），这又是一个前景无可限量的研究课题，感谢杨老师的睿智和眼光。说实话，我已经深深地被测试驱动开发的魅力所感染，我想在以后的工作中更多的使用这种技术，我坚信这会在未来的工作中帮我的大忙，使我在事业的起跑线上拥有更多的优势。

其次，要感谢杨老师为我指点大方向和细心评审。从刚开始杨老师为我指点完成毕业设计的整体思路，到杨老师为我推荐了几本 TDD 和单元测试的经典书籍，从而为我确定了大的方向。而我每个阶段的成果（如开题报告、论文初稿）提交后，杨老师都细心地研究斟酌，不仅指出大的欠缺，甚至还指出一些细微的措辞的不当，实在感谢老师的细心和关怀。

然后，要感谢的是参考文献中的几本书籍的作者，他们的著作将我领进了测试驱动开发和 Struts 2 的大门，并牢牢地掌握了相关的大量技巧，使我完成了研究本课题所需的多数技能的修炼。

另外，还要感谢无锡软通为我提供的计算机，毕竟，我的毕业设计的 90%的工作是在这台联想电脑上完成的。至于联想——太远了点，就不谢了吧。

其实，大部分的工作还是由我自主完成的，毕竟我一直都是远在异地，杨老师主要是从宏观指点了大方向。在进入杨老师的实验室之前，就听闻杨老师与其他老师不同，喜欢无为而治，但培养出来的学生却比较厉害。很庆幸能进入杨老师的实验室，并在杨老师的指点之下，成长了很多很多。而这段时间，正是我开始专攻 Java 的时期，也是我开始成长为一名 Java 程序员的过渡阶段。

还有，要感谢软通的一些前辈同事，我曾经询问过他们一些问题。虽然没有得到什么实质性的帮助，但他们的经验让我少走了弯路，因为软通还没有实施 TDD，但拥有单元测试的丰富经验，使我在此方面得到了一些帮助。

最后向所有关心我的和我所关心的人表示感谢。

论文作者：陈峰
QQ：562116039

参考文献

[1] 孙平平等 译. Kent Beck . 测试驱动开发(中文版) [M].北京:中国电力出版社.2004.3

[2] 王晓毅. 测试驱动开发的三项修炼—走出 TDD 丛林[M].北京:清华大学出版社. 2008.1

[3] 鲍志云 译. Vincent Massol . JUnit in Action 中文版[M]. 北京:电子工业出版社. 2005

[4] 陈浩等 译, J.B.Rainsberger , Scott Stirling. JUnit Recipes 中文版——程序员实用测试技巧[M] .北京:电子工业出版社.2006.9

[5] 李刚. Struts 2 权威指南:基于 WebWork 核心的 MVC 开发[M]. 北京: 电子工业出版社. 2007

[6] 白胜普. J2EE 企业级应用测试实践[M].北京.清华大学出版社.2009.1

[7] 侯捷 译. Martin Fowler. 重构－－改善既有代码的设计 [M]. 中国电力出版社. 2003

[8] Russell Gold, Thomas Hammell , Tom Snyder. Test Driven Development: A J2EE Example [M]. USA: Apress. 2005

[9] 曾广平,兰鄂. 测试驱动开发探究[J]. 开发研究与设计技术. P172-P174

[10] 黎利 , 刘振宇 . 面向 MVC++ 的测试驱动开发研究 [J]. 计算机系统应用.2006(12)P43-P46

[11] 李群 . “ 浅谈测试驱动开发 （ TDD ）” . developerWorks 中国 . http://www.ibm.com/developerworks/cn/linux/l-tdd/ . 2004.11.19

[12] 郑闽睿 . “ EasyMock 使用方法与原理剖析 ” . developerWorks 中国 . http://www.ibm.com/developerworks/cn/opensource/os-cn-easymock/ . 2007.10.25

[13] “Unit Testing Struts 2 Actions ”. http://glindholm.wordpress.com/2008/06/30/unit-testing-struts-2-actions/ .2008.6.30

[14] “Unit Testing Struts 2 Actions wired with Spring using JUnit ”. http://arsenalist.com/2007/06/18/unit-testing-struts-2-actions-spring-junit/ .2007.6.18

论文作者：陈峰
QQ：562116039