长春理工大学

硕士学位论文

基于J2EE的轻量级WEB架构研究与应用

姓名：杨泽

申请学位级别：硕士

专业：计算机软件与理论

指导教师：叶青

20100301

摘　　要

在基于 J2EE 的应用开发中，架构是技术实现的关键，随着 J2EE 的不断发展，涌现出大量的开源轻量级框架。而一些企业在应用开发中要求对架构的设计在追求尽量缩短研发周期、降低研发成本的前提下，实现应用系统的功能要全面和强大。为满足这些企业的需要，设计一套开源免费的轻量级 J2EE 架构是这些企业所需。

本文首先从 J2EE 的定义和结构入手，阐述了轻量级 J2EE 架构的基本理论。其次，通过传统 J2EE 架构和轻量级 J2EE 架构在各个方面的对比，确定轻量级 J2EE 架构的优势，选择当前流行的 Struts2、Spring、Hibernate 作为本文架构的基础，讨论了它们的相关理论。并在此基础上，提出一个符合 J2EE 规范的面向企业级应用的轻量级架构，对架构模型的建立，架构各层的细分以及开源框架的整合等实现本文架构涉及的关键技术进行了系统论述，并且对架构在分页技术、字符集过滤、整合 jBPM 及数据封装等方面进行完善，并通过架构中各层间的数据关系在设计模式中的表现，证明它的可行性。最后，通过一个再担保业务系统实例，验证它在实践中的可用性。

关键字：表示层　业务层　持久层　轻量级架构

ABSTRACT

In the development of J2EE application, the architecture is the key technology, with the continuous development of J2EE, there emerged a large number of open-source lightweight frameworks. In the development of short cycle and low-cost R & D , The function of the system should be strong In order to meet the needs of these enterprises in designing a free lightweight J2EE framework open source is that these businesses need.

Firstly, this paper started from the definition and structure of J2EE, and expounded the basic theory of lightweight J2EE framework Secondly, through the comparison for traditional J2EE architecture and lightweight J2EE framework, determined the advantages of lightweight J2EE architecture ,selected the current popular Struts2, Spring, Hibernate as the basis for this framework,and discussed their related theories. And on this basis, proposed a consistent standard for enterprise-class J2EE applications, lightweight structure, and discussed the building of the architecture model, a breakdown of architecture layers, as well as the integration of open-source frameworks involved in key technology, achieved the key technology involved in the architecture .It improved the architecture by paging technology, character set filtering, integrating jBPM and data in areas such as packaging. It prove its feasibility through the performance in the design mode data relationships of the architecture layers . Finally, an example of re-guarantee business system, verify that it is in practice availability.

Key words: presentation layer business layer persistence layer Lightweight architecture

长春理工大学硕士学位论文原创性声明

本人郑重声明：所呈交的硕士学位论文，《基于J2EE的轻量级WEB架构研究与应用》是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：杨洋　　2010年3月20日

长春理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“长春理工大学硕士、博士学位论文版权使用规定”，同意长春理工大学保留并向中国科学信息研究所、中国优秀博硕士学位论文全文数据库和 CNKI 系列数据库及其它国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权长春理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：杨洋　　2010年3月20日

导师签名：　　2010年3月20日

第一章 绪论

1.1 问题提出

在现今的大部分企业级应用开发中，都追求在尽可能缩短开发周期、降低开发成本的同时，开发出功能强大的应用系统，因此，如何使系统结构灵活，具有松耦合特性是这些企业开发中要考虑的重要问题。在基于 J2EE 的企业级应用开发中，架构是实现技术的关键，研究如何建立一个良好的架构使其具有松耦合的特性，满足企业低成本、高效益的需求具有重要的实用价值和理论意义，本文在广泛研究当前流行的各种开源框架的基础上将设计一套轻量级架构，满足以上要求，首先从它的研究背景说起。

1.2 研究背景

J2EE 是 SUN 公司推出的使用 Java 技术开发企业级应用的一种工业标准，利用 Java2 开发平台使企业级应用在研发过程中所涉及的跨平台移植、事务管理等一系列技术问题得以简化，以 Java 为技术核心，不仅能对 EJB（Enterrise JavaBeans）、Java Servlets API、JSP（Java Server Pages）提供全面技术支持，同时在安全性、健壮性、组件化等方面成熟稳定，获得了众多厂商的技术支持，在远程教育、电子商务平台的开发中它是首选。

当前，由于 J2EE 架构的跨平台特性，可以充分利用用户原有的操作系统和硬件，越来越多的企业都采用 J2EE 技术为自己的系统升级，使 J2EE 架构成为主流的架构之一。由于许多中间件供应商提供复杂的中间件服务，因此采用 J2EE 体系结构意味着一些通用的复杂的服务端功能占用开发时间短，从而大大缩短开发周期，能够满足那些不想耗费太多开销而又需要高度的稳定性、可伸缩性以及灵活性的应用需求。同时 J2EE 提供完整的 Web 服务支持，以便开发者能够在 Java 平台上方便的开发并部署自己的 Web 服务，并且通过“容器”的使用来简化开发过程，提高开发效率。

J2EE 分层设计是 J2EE 企业级应用最基本的设计思想，应用业务逻辑按功能划分为组件，可以将各个组件按照它们所在的层部署到相应的服务器上。针对传统的 C/S 结构客户端臃肿，业务逻辑和界面的重用困难等缺点，J2EE 将企业级应用切分成多层，典型的四层划分包括：客户层、WEB 层、业务层、企业信息系统层[1]。其中运行在 J2EE 服务器上的业务层组件 EJB（Enterprise JavaBeans）容器是系统的核心部分，负责 EJB 整个生命周期的一切工作。

在这个架构中，EJB 包含了各种服务并提供了一个共享中间层，能够支持各种 J2EE 客户端。然而，其缺点有二：其一，在结构中开发开销及应用性能的负担很重；其二，

为了实现分布式处理，牺牲了面向对象原则，不便于测试[2]。为了解决其缺点，通常改进方式是用本地 EJB 替代远程 EJB，这样虽然能够解决分布化问题，提高架构的可重用性，但是架构依然存在结构复杂、开销负担重等缺点。

随着技术的不断发展，对企业级应用要求的不断增加，对架构的设计也提出了新的要求，主要有以下四种表现[3]：第一，架构在追求功能强大的同时要求设计简单；第二，架构要有良好的扩展性；第三，架构在不同的服务器之间可以实现平稳的移植；第四，架构要便于对业务对象进行良好的测试。

为跟上技术发展的潮流，满足不断发展的企业应用，轻量级 J2EE 架构被提出了。在一个轻量级 J2EE 架构中，业务对象是通过普通 Java 对象（POJO）实现的，它不依赖于轻量级 IoC 容器，但能够在 IoC 容器中运行。通过面向切面编程技术（AOP）能够增强业务对象，实现企业级服务。在架构中，数据层使用 O/R Mapping 技术提供持久化支持，表示层（Web 层）只处理用户的请求信息，并将业务对象响应的结构展现给用户。

这种轻量级架构，在开发中能够更好的执行面向对象原则，它提高了程序的复用度，使开发效率大大提高。而这种轻量级架构中每个层的划分，及各个层的整合是确保以上优点的根本。因此，时至今日，基于这种架构的各个层涌现出大量的、开源的框架。各层间框架的整合应用就能够满足那些需要快速开发简单系统的企业的应用需求。

综上所述，J2EE 架构在一次革新之后，以 EJB2.x 为代表的传统 J2EE 架构已逐渐退出历史舞台，与此同时，轻量级 J2EE 架构却逐渐成为那些需要快速开发简单系统的企业的主流。IoC 容器是轻量级 J2EE 架构的核心；AOP 正成为一种日趋成熟的技术，它为轻量级 J2EE 架构提供更好的服务。开源框架的整合应用能够使轻量级 J2EE 架构在应用中提高开发效率，因此，框架的完美整合使 J2EE 应用更简单、更易用。

1.3 本文主要研究内容及论文结构

1.3.1 主要内容

本文详细论述一个轻量级 J2EE 开发架构的建立过程，它满足 J2EE 开发规范，它主要应用于一些普通的企业级 Web 开发，同时它对 J2EE 的高级特性要求不高。与传统 J2EE 架构相比，本架构开发复杂度较低，代码复用性较高，使用它可以加快企业级应用开发的速度。

因此，本文研究的主要内容是怎样建立一个满足要求的 J2EE 架构，具体内容如下：

(1) 通过 J2EE 的概念和结构了解 J2EE 架构，进而引入轻量级 J2EE 架构，从容器的角度阐述轻量级 J2EE 架构的优势。

(2) 通过阐述 MVC 架构模式，研究当前一些流行的轻量级 J2EE 框架的体系结构和工作原理，确定本文架构的各个组件，将本文架构分为表示层、业务层、持久层。

(3) 将各个组件进行整合并完善，以构建出满足企业要求的轻量级 J2EE 架构。

(4) 将整个架构应用于实际的企业级开发中。

1.3.2 论文结构

本文总共由六章组成。

第一章：绪论。提出本文的研究问题，在系统介绍本文的研究背景基础上，说明本文研究问题的理论意义和实用价值。

第二章：轻量级 J2EE 架构分析。从容器类型和容器与服务的关系的角度阐述 J2EE 结构。引入轻量级 J2EE 架构，并从容器的角度阐述轻量级 J2EE 架构的优势。

第三章：本文架构所涉及的框架和技术要点。首先确定本架构总体上要采用 MVC 三层架构模式。通过分析一些当前流行的表示层框架，确定本架构表示层框架采用 Struts2，此外，本架构的业务层框架和持久层框架分别选用当前流行的 Spring 和 Hibernate。在选择 Struts2、Spring 和 Hibernate 的同时，分析和研究它们的技术特性、体系结构及工作原理。

第四章：企业级应用轻量级架构的设计与实现。提出本架构的总体设计目标，构建本架构的基本模型，实现本架构中各个组件的整合，通过一些插件及公有类优化本架构，通过 MVC 架构模式说明本架构中各层间数据关系在设计模式中的表现，从理论上论证其各部分的重用性和可扩展性。

第五章：本架构在再担保业务系统中的应用。通过再担保业务系统，展现了本架构在企业级开发中的应用过程。从实践的角度证明了本架构的可用性。

第六章：本文总结与工作展望。分别从软件设计、实际开发和软件分层的角度总结本文架构的特点及不足。

第二章 轻量级 J2EE 架构分析

2.1 J2EE 概念

J2EE（Java2 Enterprise Edition）是一套面向企业应用的体系结构，可以将它理解为 J2SE（Java 2 平台的标准版）的扩展和延伸，它的规范和标准是由 SUN 公司倡导和制定的，它是一个主要用于构建以服务器端应用为核心的、基于互联网环境的、多层次模块化结构的企业级应用的部署和开发平台[4]。整个业界对 J2EE 每一个规范的推出都十分认同，同时许多公司（如 IBM 等）对它的标准的制定都做出了贡献。

J2EE 通过提供中间层集成框架来满足企业应用软件所需的安全可靠、高效及低成本等特性。它对现有应用程序提供了良好的支持，主要可以表现为四点：[5] 1. 对企业级 JavaBean 的完全支持；2. 对打包和部署应用的良好支持；3. 对安全机制的支持；4. 对添加目录的支持。

2.2 J2EE 结构

在第一章中本文已经介绍过传统 J2EE 架构是由四层组件组成的，J2EE 被设计成一种基于组件、平台无关的结构，它将业务逻辑封装成可复用的组件，J2EE 服务器以容器的形式为所有组件类型提供后台服务，这样可以使开发者集中精力解决复杂的业务问题。

从容器和服务的关系来看，容器就是一组提供服务的管理器，不同的容器要符合不同服务的规范和要求。J2EE 容器定制了包括事务管理、安全机制、Java 命名和目录接口寻址（JNDI）、远程服务连接、生命周期管理、数据库连接池管理等多种技术[6]。

从容器类型的角度来看，J2EE 组件可以部署到如图 2-1 所示的容器中。

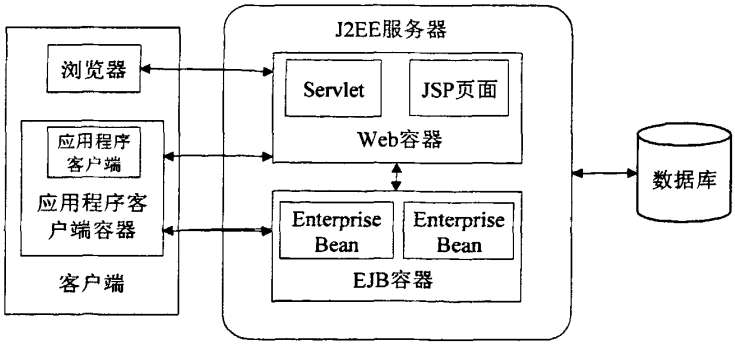

图 2-1 J2EE 结构

从图 2-1 可以看出：EJB 容器和 Web 容器都是运行在 J2EE 服务器上的，它们分别管理 J2EE 应用程序中所有 Enterprise Bean（企业级 Bean）的执行和 J2EE 应用程序中所有 JSP 与 Servlet 组件的执行。而应用程序客户端容器运行在客户端，管理 J2EE 应用程序中所有应用程序客户端组件的执行。Applet 容器也是运行在客户端的，通常是客户机上的 Web 浏览器和 Java 插件的结合。

以上从服务和容器的关系及容器类型的角度阐述了 J2EE 的结构，由此可以看出，J2EE 应用能够集成包括台式的、无线的以及基于浏览器的等一系列的客户端，并且能够提供易于扩充的、高性能的运行支撑环境以满足企业级应用开发的所需。

在企业开发过程中，一般采用多层结构开发，每一层都有独立的任务和问题，为了提高程序的效率和稳定性，缩短开发时间，企业开发通常采用现有的成熟框架方案。

2.3 轻量级 J2EE 架构

本文第一章研究背景中阐述过传统的基于 EJB 的 J2EE 架构在开发中本身占用资源多、启动慢，同时部署复杂、运行缓慢并且难以测试。基于以上问题，随着相关技术的发展，轻量级 J2EE 架构出现了。

2.3.1 轻量级 J2EE 架构的概念

BEA 公司的 Jim Rivera 为轻量级架构给出如下定义，轻量级 J2EE 架构是指简化的编程模型和更具响应能力的容器，它旨在消除与传统 J2EE API 有关的不必要的复杂性和限制，它也将缩短应用程序的部署时间[7]。轻量级 J2EE 架构是相对于传统的以 EJB 容器为主的 J2EE 架构而言的，它与传统 J2EE 架构的侧重点不同。轻量级框架侧重于减小开发的复杂度，采用轻量级框架主要有两方面的原因，第一，轻量框架在开发时不依赖于任何容器，它尽可能的采用基于普通 Java 对象的方法来开发，这样能够提高开发过程中的调试环节的效率；第二，轻量级框架大多数来自于开源社区，开源社区提供了完整的 API、开源框架的源码及快速的构建工具，这样能够提高项目的开发速度。随着技术的发展，各具特色的开源框架不断涌现，开发者可以根据实际应用选择不同的框架应用于企业开发中。

2.3.2 轻量级 J2EE 架构的优势

轻量级 J2EE 架构一般采用 IoC 容器作为它的一种实现，而传统 J2EE 架构通常采用 EJB 作为它的实现。现将它们对比如下：

1.从容器角度看，IoC 容器与 EJB 相比是一种轻量级实现，从大小与开销两方面而言轻量级 J2EE 架构都要小于传统 J2EE 架构。

2.轻量级 J2EE 架构可以实现与传统 J2EE 架构相似的功能，只不过还没有传统 J2EE

架构强大、健壮和完善。

3.从开发复杂度看，轻量级 J2EE 架构的实现比传统 J2EE 架构的实现简单，传统 J2EE 架构需要编写复杂的 EJB 组件，而轻量级 J2EE 架构只要编写普通的 JavaBean 就可以。

4.从开发成本看，轻量级 J2EE 架构比传统 J2EE 架构要节省更多的资金成本，因为传统 J2EE 架构通常使用 EJB，而购买 EJB 容器需要花费很多资金，而轻量级 J2EE 架构通常使用 Spring 一类的开源框架，可以免费使用。

2.4 本章小结

本章中首先介绍了 J2EE 的概念，并了解到 J2EE 是通过提供中间层集成框架和统一的开发平台来满足企业需求的，然后分别从容器类型和容器与服务的关系的角度阐述了 J2EE 的结构，通过了解 J2EE 的结构可知企业级开发一般采用多层结构开发。通过绪论中对传统的 J2EE 架构缺陷的了解及本章对 J2EE 结构的了解，可知轻量级 J2EE 架构是当今企业开发所需要的。最后在了解轻量级 J2EE 架构的概念之后，将轻量级 J2EE 架构和传统的 J2EE 在各个方面做了对比，确定了轻量级 J2EE 架构的优势，为本文后续研究打下了基础。在下一章中，本文将对要提出的架构的重要组件及各组件的技术做深入研究。

第三章 本文架构所涉及的框架和技术要点

3.1 MVC 架构模式

所谓 MVC 模式，即模型-视图-控制器（Model-View-Controller）模式。MVC 模式是一种架构模式，它提供了一个原则，可以按照模型、表达方式和行为等角色把一个应用系统的各个部分之间解耦、分割，使系统各部分相对独立[8]。

MVC 模式把一个应用程序分割成三个层次：视图层、模型层、控制器层，使每个层次各自处理自己的任务。其结构如图 3-1 所示。

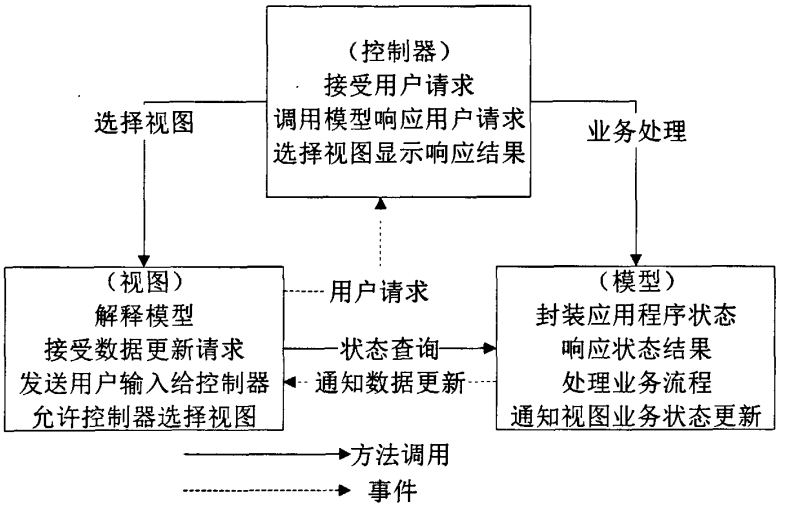

图 3-1 MVC 模式示意图

在图 3-1 中，视图就是与用户交互的界面，它可以接收用户输入，向用户显示数据，与模型层交互，向模型查询业务状态，接受模型发出的数据，更新显示用户界面。模型是应用程序的主体，用于表示业务数据和业务逻辑，一个模型可以同时为多个视图提供数据，从而提高代码的可重用性。控制器即起到控制作用，它接受用户输入，并调用模型和视图来完成用户请求[9]。在用户提交表单时，控制器本身不做任何处理和输出工作。控制器仅仅是接收请求，然后调用模型组件处理请求，最后调用视图来显示模型处理后返回的数据。

由以上分析可知，MVC 三层模型处理过程是这样的：用户在视图端发出请求。控制器接收用户请求，然后决定调用哪个模型来进行处理。模型进行相应的逻辑处理，然后返回处理后的数据。控制器调用视图将模型返回的数据呈现给用户。

由 MVC 的处理过程可知，MVC 本身就是一个比较复杂的系统，所以采用 MVC 实现 Web 应用时，最好选一个现成的 MVC 框架。在此之下进行开发，能够取得事半功倍的效果。

3.2 表示层框架

由 3.1 节可知，MVC 模式即把一个应用的输入、处理、输出流程按照 M-V-C 的方法进行分离。目前众多厂商和开源社区都基于 MVC 模式提供了表示层框架的实现，常用的表示层框架有 Struts、Turbine、COCOON、Tapestry、WebWork、Struts2、JSF 等，在这些表示层框架中，Struts 最为常用，并已成为业界的标准，然而，随着技术的不断发展，Struts2 在 Struts 及 WebWork 基础之上日趋成熟，并将得到更广泛的应用。

3.2.1 Struts 简介及存在的问题

Struts 是一个满足 MVC 模式的开源框架，使用它可以将数据的显示、控制及业务逻辑功能模块区分开，这样改动其中一个模块不会影响到其它模块[10]。然而，Struts 却存在以下问题：

1. 模型部分主要由底层的业务逻辑组件充当，它们提供数据库访问和业务逻辑的实现，可以是 JavaBean、EJB 组件或者 WebService 服务，Struts 没有为实现模型组件提供任何支持。

2. 控制器部分由核心控制器 ActionServlet 和用户自定义 Action 组合，由 Struts 框架提供支持，但是，它支持表示层的技术单一，只支持 JSP，不提供与其它表现层技术整合，如 FreeMarker。

3. 控制器部分与 Servlet Api 严重耦合，难于测试，Action 类包括 HtttpServletRequest、HttpServletResponse，代码严重依赖于 Struts Api，属于侵入式设计，Aciton 类必须继承 Action 的基类。

3.2.2 WebWork 简介

WebWork 是建立在 XWork 的 Command 模式框架之上的强大的基于 Web 的 MVC 框架[11]。它来自另一个开源组织：Opensymphony。与 Struts 框架相比，在视图部分，它支持更多的表现层技术--velocity、freeMarker 和 xslt 等。在控制器部分提供核心控制器 ServletDispatcher，无需与 Servlet Api 耦合，由 Action 拦截器负责数据初始化，它只是一个普通的实现了 webwork Action 接口的 POJO(包含了一个 execute 方法)。

3.2.3 Struts2 的简介

通过以上 3.2.1 和 3.2.2 节对 Struts 缺陷及 WebWork 的了解，本文采用 Struts2 作为表示层框架。Struts2 源于 WebWork2，融合了 Struts 和 WebWork 两大社区的力量，是这两大社区共同努力的结果。它继承了 Struts 和 WebWork 两者的优点，有着很好的发展前景。

3.2.4 Struts2 与 Struts1.x 的比较

为说明 Struts2 是如何弥补 Struts 的缺陷及进一步体现 Struts2 的优越性，将它与 Struts 在以下几个方面进行对比(为区分 Struts 和 Struts2，以下 Struts 用 Struts1.x 表示)：

(1)Action 实现及控制：支持 Struts1.x 的每一个模块都有单独的生命周期，模块中所有 Action 都必需共享相同的生命周期，并且 Action 类一般继承一个抽象类[12]，它不是面向接口编程的而是面向抽象类编程的。而 Struts2 可以通过拦截器栈为每个 Action 创建不同的生命周期，能够根据需要和不同的 Action 一起使用，它的 Action 类可以实现 Action 接口和其它接口。

(2)线程的安全与同步问题：Struts1.x 的每个 Action 类最多只能创建一个实例，因此在开发时要针对线程安全及同步问题进行专门的处理工作，而 Struts2 的 Action 类针对每一个请求创建一个实例，开发中不需要特殊处理线程安全的问题[13]。

(3)可测性：Struts1.x 的 Action 类对象在执行时依赖于 Servlet 容器，测试不方便。而 Struts2 的 Action 类和容器无关，可以通过初始化、设置属性、调用方法的方式测试，并且可以通过“依赖注入”的方式使测试更容易。

(4)用户输入信息的捕获：Struts2 直接使用 Action 属性作为输入属性，消除了 Struts1.x 中对 ActionForm 对象的需求。

(5)支持 OGNL 表达式语言：为了弥补 Struts1.x 对表达式语言支持单一的缺点，Struts2 在使用 JSTL 表达式的同时也支持表达式语言 OGNL(Object Graph Notation Language)，以加强对集合和索引属性的灵活处理。

(6)页面输出控制：Struts1.x 的页面输出是通过 JSP 机制把对象值邦定到页面中。而 Struts2 使用值栈（ValueStack）技术，不需要页面和对象值邦定即可以通过标签直接访问对象值，并且支持属性名重用。

(7)数据类型多样性：Struts1.x 的 ActionForm 属性通常都是 String 类型，Struts1.x 使用 Commons_Beanutils 工具包执行类型转换。Struts2 的模型属性可以使用多种类型并提供自动转换机制。

综上可以看出，Struts2 在各个方面都优于 Struts1.x，虽然 Struts1.x 还是业界的标准，但 Struts2 作为表示层框架更满足当今技术发展的潮流，因此，本文选择 Struts2 框架是满足当今企业级应用发展需要的。

3.2.5 Struts2 的体系结构

Struts2 的体系结构可用图 3-2 来描述。

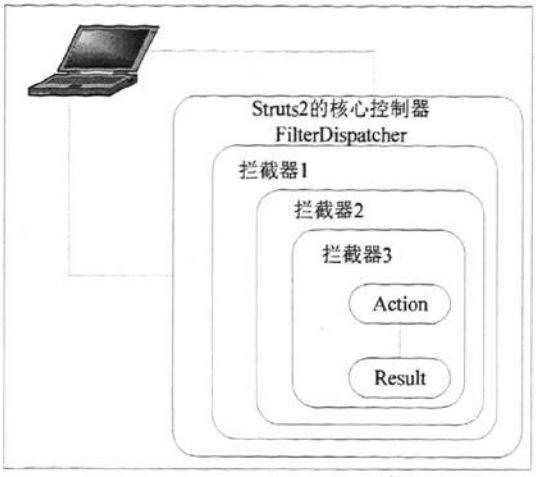

图 3-2 Struts2 的体系结构图

在图 3-2 中，FilterDispatcher 是 Struts2 的核心控制器[14]，当它被 Struts2 框架加载后，将对用户的请求进行拦截，并执行过滤分发操作；拦截器则动态的拦截 Action 对象，它可以在调用一个 Action 对象的前后加入操作，也可以在一个 Action 执行前阻止其执行，该机制提供了一种程序重复利用的方式，可以将程序通用部分提取并写成拦截器，然后作用到不同的 Action 对象中；Result 是返回 Action 处理后结果的，当 Action 的处理方法执行完毕后，将返回一个 result code（SUCCESS、INPUT 或者错误码等），Struts2 内建了常用的结果类型，默认是 Dispatcher 类型，在开发中也可以根据实际需要创建自己的 Result 类型。以上所说的 Interceptor、Action 类和 Result 等组件都是配置在 stuts.xml 里的，stuts.xml 是 Struts2 的核心配置文件[16]。

图 3-2 中描述了 Struts2 体系结构中各个组件之间的关系，具体关系可以体现在以下几个处理过程中。

(1) 浏览器请求资源，如/mypage.action、/reports/myreport.pdf 等。

(2) 核心控制器 FilterDispatcher 根据 struts.xml 文件查找处理请求的 Action 类。

(3) 拦截器 Interceptor 自动对请求使用通用功能，如 workflow、validation 和文件上传等功能。

(4) 如果 struts.xml 文件中配置 Method 参数，则调用 Method 参数对应的 Action 类中的 Method()方法，否则调用通用的 execute()方法来处理用户请求。

(5) 将 Action 类中对应方法返回的结果 Result 响应给浏览器，结果可以是 HTML 页面、图像、PDF 文档或者其它文档。

3.2.6 Struts2 拦截器

拦截器是 Struts2 框架最强大的特性之一，可以在 Action 和 Result 被执行前后进行一些权限检查或资源释放等处理[17]。同时使用拦截器可以使代码模块化，提高代码可

重用性。Struts2 框架中许多核心的特性都基于拦截器的方式实现。为进一步了解 Struts2 拦截器的特性，下面分别研究一下拦截器的调用过程、内置拦截器及自定义拦截器。

1. 拦截器的调用

由 3.2.5 节对 Struts2 体系结构的分析可知，当提交对 Action 的请求时，ServletDispatcher 首先会根据请求查找配置文件 struts.xml，然后根据 struts.xml 中相应配置调用相应的 Action 处理。在调用 Action 处理之前，首先要被拦截器截取，然后按照在配置文件中的顺序在 Action 的执行前后运行[18]，其调用的过程可用下图表示。

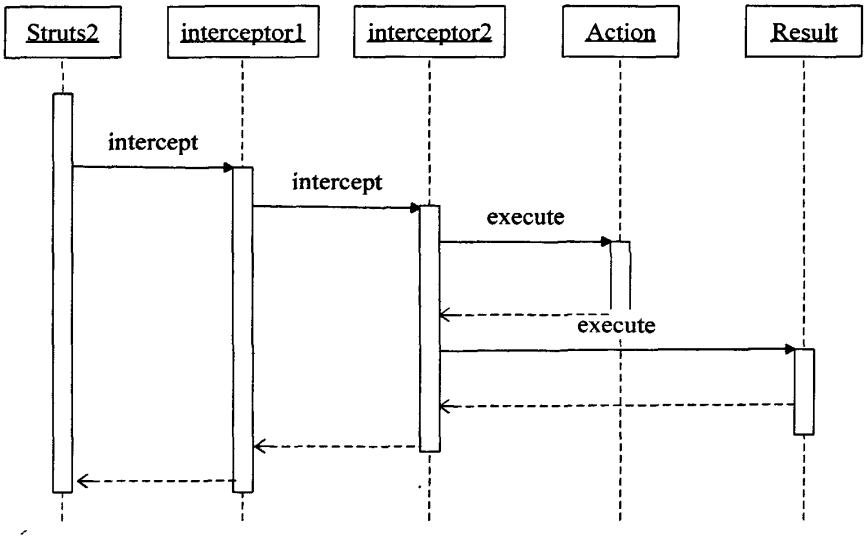

图 3-3 拦截器的调用序列图

2. 内置拦截器

Struts2 框架中提供了大量内置的拦截器和拦截器栈，它们提供了很多的核心功能和高级特性。所谓拦截器栈就是将拦截器按照一定的顺序联结成一条链，组成这条链的拦截器就会按照其定义的顺序被调用。这些拦截器和拦截器栈被定义在 struts-default.xml 文件中[19]。struts-default.xml 中定义拦截器的部分配置信息如下所示。

```
<interceptor name="execAndWait"
class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
<interceptor name="exception"
class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
<interceptor name="fileUpload"
class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
<interceptor name="i18n"
class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
<interceptor name="logger"
class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
```

如果需要使用 Struts2 框架的内置拦截器，可通过以下几个步骤完成：首先，在应用程序的 struts.xml 配置文件中通过“<include file="struts-default">”将 struts-default.xml 文件引入，Struts2 会自动引入此文件。然后，定义 package 时，继承 struts-default 包。最后，定义 Action 时，使用“<interceptor-ref name="xx">”引用拦截器或拦截器栈。

通过以上三步，就可以使用 Struts2 提供的内置拦截器。

3. 自定义拦截器

通过 3.2.6.2 节可知，Struts2 内置了功能丰富的拦截器，然而，在现实应用中有一些与系统逻辑相关的功能只使用 Struts2 提供的内置拦截器是无法实现的，此时需要通过自定义拦截器来解决。在 Struts2 中自定义拦截器比较方便，通过以下几个步骤就能完成：首先，定义一个类，实现 Interceptor 接口，或者继承 AbstractInterceptor。然后在 struts.xml 配置文件中注册拦截器。最后在需要使用的 Action 中引用上述定义的拦截器即可。[20]

通过以上过程就能实现自定义的拦截器，在实现过程中，Interceptor 接口声明了三个方法，其实现代码如下。

```
public interface Interceptor extends Serializable{
    void destroy();
    void init();
    String intercept(ActionInvocation invocation) throws Exception;
}
```

在以上接口中，init()方法在拦截器创建时调用，可用于完成拦截器的初始化操作。destroy()方法在拦截器被垃圾回收之前调用，可用于完成释放系统资源的操作。Intercept()是拦截器的主要拦截方法，如果需要调用后续的 Action 或者拦截器，需要在该方法中使用 invocation.invoke()方法。在 invocation.invoke()方法调用的前后可以插入 Action 调用前后需要做的操作。Intercept()方法执行后会返回一个 String 类型的对象，然后继续执行下一个拦截器，如果没有指定下一个拦截器，就执行 Action 中的请求处理方法，默认方法为 execute()。

除 Interceptor 接口外，Struts2 框架还提供了一个 AbstractInterceptor 类，该类提供了一种简单的 Interceptor 接口的实现。如果不需要编写 init()和 destroy()方法，则可以继承 AbstractInterceptor 类，实现 intercept()方法。

3.2.7 Struts2 的工作流程和原理

Struts2 从客户端浏览器发出请求到获得响应整个处理过程如图 3-4 所示。

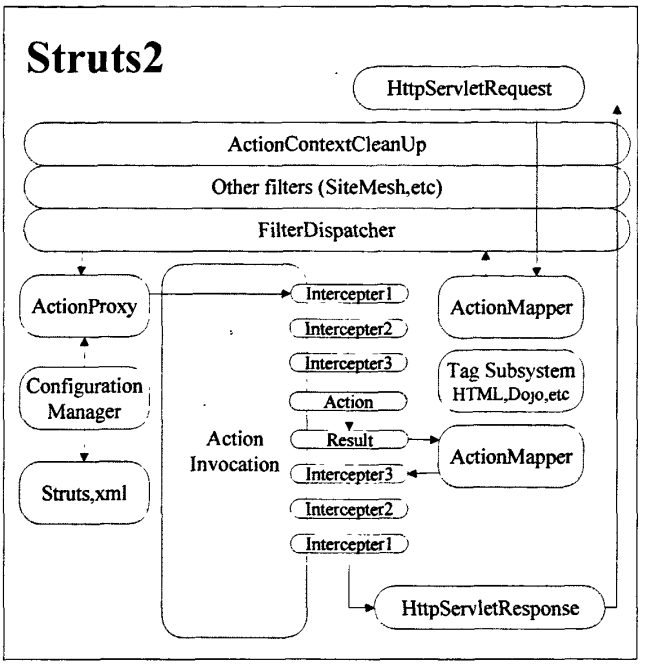

图 3-4 Struts2 的工作流程

首先由客户端浏览器发出一个 HTTP 请求，所发出的请求经过一系列标准的过滤器链后到达核心控制器 FilterDispatcher。接下来 FilterDispatcher 通过 Action 映射器（ActionMapper）决定调用哪一个 Action 类处理客户端请求，之后它将请求处理交给 Action 代理（ActionProxy）。ActionProxy 再通过配置文件管理器（Configuration Manager）即通过 Struts2 的配置文件 struts.xml，找到需要调用的 Action 类。在调用 Action 类的过程中，ActionProxy 首先要通过配置文件管理器读取 struts.xml 文件里的配置信息，然后创建一个实现了命令模式的 ActionInvocation 对象，该对象通过代理模式调用 Action，在调用 Action 之前会调用所有的拦截器（interceptor）的 before()方法。然后 Action 调用相应的业务对象，对数据库执行增删改查操作。在 Action 方法执行完毕之后返回一个结果码（result code），ActionInvocation 将根据 struts.xml 中该结果码的定义，执行 result 处理。result 会调用 JSP 或 FreeMarker 模板来呈现页面，也可以继续调用某个 Action[21]。当呈现页面时，模板可以使用 Struts2 提供的标签。最后经过一系列拦截器（Intercepter）的 after()方法后，将请求的处理结果返回客户浏览器呈现。

3.3 业务逻辑层框架

在实际开发中，业务逻辑是一个系统的核心，在第二章中，已经介绍了轻量级 J2EE 架构的优势，因此本文选取轻量级 J2EE 架构常用的 Spring 做为业务逻辑层框架，本文选取 Spring 框架的另外一个原因是 Spring 可以降低企业开发的复杂度，提高程序的灵活性，便于系统的扩展和变更，Spring 增强了系统的模块化结构，提高了功能模块的

重用性[22]。

3.3.1 Spring 的概述

Spring 是一个开源框架，由 Rod Johnson 创建[23]。Spring 为简化企业级应用开发而生。使用 Spring，可以用简单的 JavaBeans 来实现那些以前只有 EJB 才能实现的功能。不只是服务端开发能从中受益，任何 Java 应用开发都能从 Spring 的可测试和松耦合特征中得到好处。

Spring 是一个轻量级的 DI 和 AOP 容器框架[24]，它具有很多功能，主要有以下特点：

(1)轻量级：从大小和应用开支上说 Spring 都算是轻量级的，整个 Spring 框架可以打成一个 2.5MB 多一点的 JAR 包。

(2)依赖注入：Spring 提供了一种松耦合的技术，称为依赖注入（DI）。使用 DI，实现非侵入式工作方式。

(3)面向切面：Spring 支持面向切面编程，增强业务逻辑独立性，实现高内聚低耦合。

(4) Spring 是一个容器[25]：它包含并且管理应用对象的生命周期和配置，设定它们之间的关联关系。但是 Spring 有别于传统的笨重的重量级 EJB 容器。

(5)Spring 是一个开源框架：使用简单的组件配置即可组合成一个复杂的应用。在 Spring 中，应用中的对象是通过 XML 文件配置组合起来的，并且 Spring 提供了很多基础功能，这使开发人员专注于应用逻辑。

3.3.2 Spring 的体系结构

Spring 框架主要可分为七大模块，这些模块为企业级开发提供了所需要的功能，而每个模块既可以单独使用又可以和其它模块组合使用，方便灵活的部署可以使开发的程序更加灵活简洁[26]。Spring 的体系结构可用这七大模块表式如下图所示。

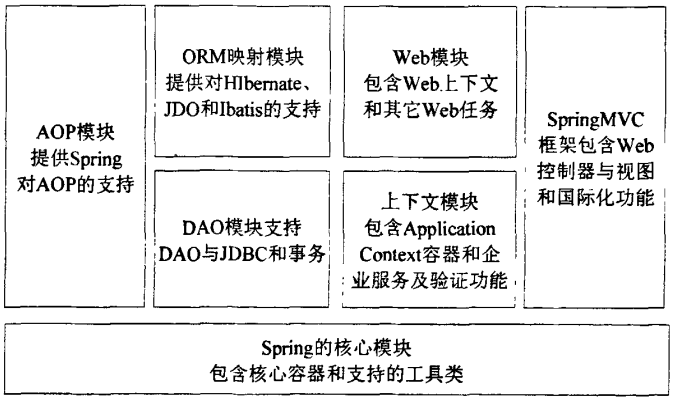

图 3-5 Spring 框架体系结构图

在图 3-5 中，核心容器（Core 模块）具有 IoC 和依赖注入的特性，它是 Spring 框

架中最基础部分，其它模块都构建于它之上，它的基础概念是 BeanFactory，BeanFactory 使用 IoC 和依赖注入特性将依赖关系和配置从程序逻辑中分离；Spring 切面编程（AOP 模块）实现横切逻辑的功能，它满足 AOP Alliance 规范，开发中可以定义切点和方法拦截器分离程序逻辑，减弱程序耦合；Spring 对象关系映射（ORM 模块）实现了对 Hibernate、iBATIS 等数据访问工具的良好集成，可以使这些数据访问工具采用相同的 Spring 特性；Spring 数据访问（DAO 模块）对 JDBC 的支持使得编写基于 JDBC 的应用更加容易，此外在 DAO 模块中还提供了一种声明性事务管理的方法；Spring Web（Web 模块）对那些依赖于复杂页面间流转的应用来说，它代表一种开发 Web 应用的新方式，它可以使用任何类型的视图技术，简化应用开发的实现，当与 Struts 或 WebWork 一起使用 Spring 时，可以使 Spring 与这些框架结合；Spring 上下文（Context 模块）提供了一种类似 JNDI 的框架式对象访问方式，此外它还包括了国际化、资源装载等一些企业服务；Spring MVC 框架能够实现 Web 开发中的 MVC，并且能够和 Spring 的其它特性更好的结合。

3.3.3 Spring IoC

IoC 全名 Inversion of Control，即反向控制或控制反转[28]。在开发时，为了便于程序重复利用，一些复杂的业务功能通常是由一系列相互关联，相互依赖的模块组成。高层模块通常与业务相关，它应该具有重用性，而不依赖于低层的实现模块，如果直接依赖低层实现会降低程序的重用性和灵活性。Spring 控制反转中的控制可以解释为程序相关模块之间的依赖关系，而反转可以解释为依赖对象发生转移，即高层依赖于抽象接口而不是依赖于具体实现。

IoC 是一个比较抽象的概念，可以用不同的方式来实现。其主要实现方式有依赖查找（Dependency Lookup）和依赖注入（Dependency Injection）两种。

Spring 的核心是一个轻量级容器，该容器通过依赖注入的方式实现了 IoC 机制，进行对象依赖关系的控制，开发时只要修改配置文件就可以改变对象之间的依赖关系。依赖注入方式有三种，分别为接口注入、setter 方法注入和构造方法注入[29]。

所谓接口注入就是在接口中定义需要注入的信息，并通过接口完成注入。它需要双方都要实现同一个接口，这样大大限制了本身的扩展性，而 Spring 是一个轻量级的非侵入式框架，使用接口注入会带来比较大的耦合，因此不建议使用这种注入方式。

所谓 setter 方式注入主要是借助于属性的 setter 方法将低层实现传递给高层。假设有类 A、类 B 和接口 IA。它们的依赖关系如图 3-6 所示。

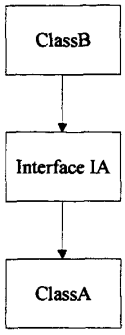

图 3-6 依赖关系图

在图 3-6 中，类 B 依赖于 IA 接口，而类 A 实现 IA 接口。采用 setter 方式注入需要在类 B 中声明一个 IA 类型的属性，并提供属性对应的 setter 方法，然后利用声明的属性调用接口 IA 的方法，接下来通过类 A 实现接口 IA。假设接口 IA 中方法为 methodA，类 B 中方法为 methodB，则类 B 和接口 IA 及其实现类的实现分别如下。

```
public class B{
    private IA a;
    public void setA(IA a){this.a = a;}
    public void methodB(){a.methodA();}
}
public interface IA{
    public void methodA();
}
public class A implements IA{
    public void methodA(){
        // 具体的实现
    }
}
```

最后要将类 A 和类 B 配置在 Spring 的配置文件中，在容器中为类 A 和类 B 声明两个对象 mya 和 myb，其中 myb 有一个名字为 a 的属性。配置信息如下。

```
<beans>
    <bean id="mya" class="com.A"></bean>
    <bean id="myb" class="com.B">
        <property name="a"><ref bean="mya"/></property>
    </bean>
</beans>
```

通过以上配置之后，Spring 会自动调用 setA()方法将 mya 对象注入给 myb 对象。

构造方法注入主要是借助构造方法将低层实例传递给高层。还是以类 A、类 B 及接口 IA 为例，构造方式与 setter 方式在实现过程中，主要是类 B 和配置信息有所不同。其实现过程为在类 B 中声明一个 IA 类型的属性，并提供类 B 的构造方法，在构造方法中把 IA 类参数赋给变量类 B 的声明变量，然后利用声明的属性调用接口 IA 的方法，接下来通过类 A 实现接口 IA。该方式下类 B 的实现如下。

```
public class B{
    private IA a;
    public B(IA a){this.a = a;}
    public void methodB(){a.methodA();
        // 其他的实现
    }
}
```

在 Spring 配置文件中，通过构造器标签引用类 A 的实例。配置信息如下。

```
<beans>
    <bean id="mya" class="com.A"></bean>
    <bean id="myb" class="com.B">
        <constructor-arg index="0"><ref bean="mya"/></constructor-arg>
    </bean>
</beans>
```

3.3.4 Spring AOP

面向方面编程（AOP，Aspected Oriented Programming）经常被定义为一种编程技术[30]。AOP 和 OOP 虽然从字面上非常相似，但却是面向不同领域的两种设计思想。面向对象编程是针对业务处理过程的实体及其属性和行为进行抽象封装，使逻辑单元划分得更加清晰高效。而 AOP 针对业务处理过程将各模块中功能相同的操作提取出来，形成一个新的处理层面，它面向的是处理过程的某个阶段或步骤，降低业务处理各实现部分之间的耦合性。AOP 从另一个角度来考虑程序的结构以完善 OOP，通常在业务处理中很多模块都需要使用日志管理、事务管理等功能，利用 AOP 思想就可以把这些公共功能提取出来，然后再切入到各个业务模块，这样在各业务模块的开发中可以集中精力实现系统的业务逻辑。

方面是 AOP 中最关键的一个术语，也可以把它称为切面，指的是对象操作过程中的截面，通常在用户进行各模块操作之前，一般都需要执行权限检查，这样在开发中就可以针对权限检查提取出一个层面，使该功能与其它各模块分离。该模块结构图可用图 3-7 表示如下。

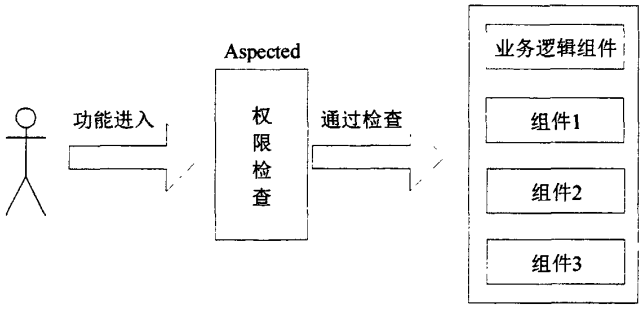

图 3-7 使用 AOP 的模块结构图

在图 3-7 中实现了一个通用的权限检查模块，在这层切面上进行统一的集中式权限管理。而业务逻辑组件则无需关心权限方面的问题。通过切面，可以将系统中各个不同层次上的问题隔离开来，实现统一的集约式处理。各切面只需关注自己领域内的逻辑实现。这样一方面使得开发逻辑更加清晰，专业化分工更加易于进行；另一方面，由于切面的隔离，降低了耦合性，可以在不同应用中将各个切面组合使用，从而使得程序可重用性增加。

除方面之外，连接点、通知、切入点也是 AOP 中的一些关键术语。其中，连接点是对象操作过程中的某个阶段点，在程序流程上的任意一点，都可以是连接点，它实际上是对象的一个操作，通常对象调用某个方法、读写对象的实例等操作都是连接点；通知是某个连接点所采用的业务逻辑，通知的调用模式通常有四种类型，分别为 Around、Before、After Return 和 Throw，在 Spring 中，是以拦截器作为通知模型，维护一个“围绕”连接点的拦截器链；切入点是指一个通知被引发的一系列连接点的集合，它指明通知在何时被触发。

在 Spring 中切面就是通过通知或拦截器实现的，Spring 中的 AOP 框架是 Spring 的一个关键组件，Spring IoC 容器并不依赖于 AOP，这意味着 AOP 既可以用又可以不用。AOP 完善了 Spring IoC，使之成为一个有效的中间件解决方案。

3.3.5 Spring 容器加载

在企业级开发中，Spring 整合系统中的各个模块，Spring 容器管理各个对象之间的关系以及 Bean 对象[31]。这里所说的 Bean 对象指的是 Spring 容器进行初始化、装配和管理的对象。BeanFactory 是 Spring 容器的核心接口，Spring Bean 对象的创建和管理是由它负责的，它还负责建立对象之间的依赖关系。XMLBeanFactory 是 BeanFactory 接口的一个主要实现，XMLBeanFactory 以 XML 结构方式描述对象与对象之间的依赖关系。Spring 在 Web 环境和非 Web 环境下提供了多种加载配置文件的方式。

1. Web 环境下配置文件的加载

在 Web 环境下配置文件可以通过 web.xml 文件进行加载。通常可利用 ContextLoaderServet 类及 ContextLoaderListener 两种配置方式加载。

利用 ContextLoaderServlet 类加载 Spring 配置文件时，首先在 web.xml 文件中指定 Spring 配置文件的位置，然后添加 ContextLoaderServlet 类的实现。实现配置如下所示。

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/spring-context/applicationContext-Spr.xml
    </param-value>
</context-param>
<servlet>
    <servlet-name>SpringContextServlet</servlet-name>
```

```
    <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
</servlet>
```

如果需要获取配置文件中定义的 Bean 对象，可实现如下：

```
WebApplicationContext ctx =
    WebApplicationContextUtils.getWebApplicationContext(this.getServlet().getServletContext());
B b = ctx.getBean("myb");
```

利用 ContextLoaderListener 方式加载与上一种方式类似，它要在 web.xml 文件中添加 ContextLoaderListener 类的监听。这种方式实现配置如下所示：

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-context/applicationContext-Spr.xml</param-value>
</context-param>
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

2. 非 Web 环境下配置文件的加载

不同于 Web 环境下加载配置文件，在非 Web 环境下加载配置文件可通过硬编码的方式实现。从 classpath 路径下加载 Spring 配置文件时，需要将 Spring 配置文件放在当前项目的 src 目录下，程序经编译后会自动将配置文件放到 classpath 路径下。这种方式加载配置文件的实现如下所示：

```
Resource resource = new ClassPathResource("applicationContext-Spr.xml");
XmlBeanFactory bean = new XmlBeanFactory(resource);
B b = Bean.getBean("myb");
```

从文件系统中加载 Spring 配置文件，可使用 FileSystemResource 类指定配置文件的位置，实现配置如下所示：

```
Resource resource = new FileSystemResource("C:\\applicationContext-Spr.xml");
XmlBeanFactory bean = new XmlBeanFactory(resource);
B b = Bean.getBean("myb");
```

从输入流中加载配置文件时，使用输入流类 InputStream 加载 Spring 配置文件。而 InputStream 是抽象类，它是不能够实例化的，因此通过它的一个子类 FileInputStream 获取一个文件流，这种实现的配置方式如下所示：

```
InputStream is = new FileInputStream("C://applicationContext-Spr.xml");
Resource resource = new InputStreamResource(is);
XmlBeanFactory bean = new XmlBeanFactory(resource);
B b = Bean.getBean("myb");
```

3.4 持久层框架

MVC 在目前的企业应用系统设计中，作为主要的系统架构模式，其中的模型层包含了复杂的业务逻辑和数据逻辑以及数据存取机制等。如何才能将这些复杂的业务逻辑和数据逻辑分离，如何才能将系统的紧耦合关系转换为松耦合关系，这都是降低系统耦合度迫切需要的，同时也是持久化的目的之一。MVC 模式实现了表示层和模型层的分离，而持久层设计则实现了模型层的业务逻辑和数据逻辑的分离。持久层封装了数据访问细节，为业务逻辑层提供了面向对象的 API。使用持久层可以让代码可重用性高，能够完成所有的数据库访问操作；持久层能够支持多种数据库平台，具有相对独立性，底层数据发生变化时，只需修改持久层代码，只要对其上层提供的 API 不变，则不用修改业务逻辑层的代码[32]。目前在持久层领域，已经出现了许多优秀的 ORM 软件，ORM 具有中间件的特性，Hibernate 就是其中的一种。Hibernate 是目前最流行的 ORM 框架，是连接 Java 对象模型和关系数据模型的桥梁，它对 JDBC 进行了轻量级的封装，不仅提供了 ORM 映射服务，还提供了数据查询和数据缓冲功能[33]。Hibernate 将 SQL 操作完全封装成对象化操作，开发者可以方便地通过 HbernateAPI 来访问数据库。现在，越来越多的企业级应用把 Hibernate 作为企业应用和关系数据库之间的中间件，以节省和对象持久化有关的 JDBC 编程工作量。

3.4.1 Hibernate 核心接口

在开发中使用 Hibernate 时，非常关键的一点就是要了解 Hibernate 的核心接口。Hibernate 的核心接口位于业务层和持久层，它的层次架构关系如下图所示。

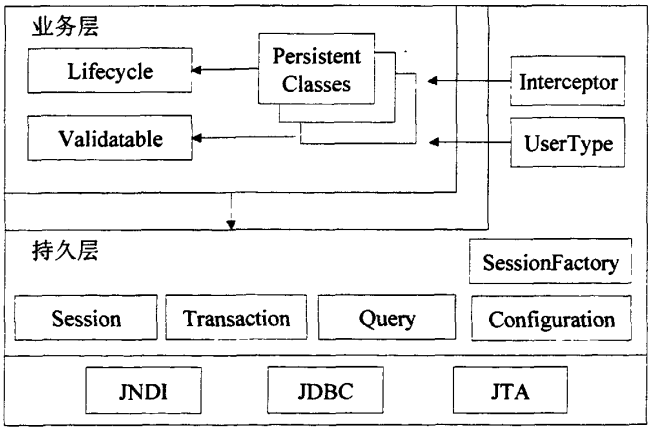

图 3-8 Hibernate 核心接口层次架构关系

从图 3-8 中可以看出，Hibernate 的核心接口一共有五个，分别为：Session、SessionFactory、Configuration、Transaction 和 Query（或 Criteria），这五个核心接口在开发中都会被用到[34]。通过这些接口，不仅可以对持久化对象进行存取，还能够进行

事务控制。五个接口的功能如下：

(1)Session 接口：专门用于完成被持久化对象的数据操纵。

(2)SessionFactory 接口：负责初始化 Hibernate，并负责创建 Session 对象。一个项目需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory[36]。

(3)Configuration 接口：负责配置并启动 Hibernate，创建 SessionFactory 对象。在 Hibernate 启动的过程中，Configuration 类的实例首先定位映射文件位置、读取配置，然后创建 SessionFactory 对象。

(4)Transaction 接口：负责事务相关的操作。Hibernate 本身并不具有管理事务的能力，只是对底层事务接口进行了封装，这样做有利于软件移植。

(5)Query 接口和 Criteria 接口：负责执行各种数据库查询。它可以使用 HQL 语言和 SQL 语言两种表达方式[37]。

3.4.2 Hibernate 的体系结构与工作流程

Hibernate 框架技术本质上是一个提供数据库服务的中间件，它的体系结构如下图所示。

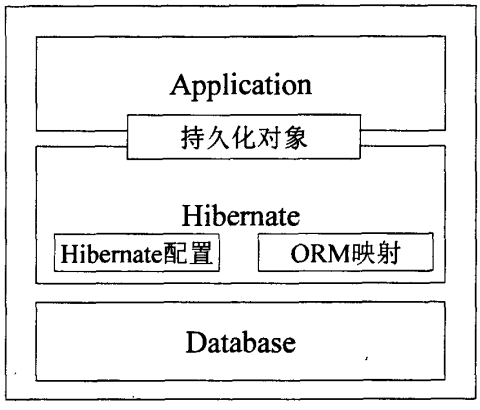

图 3-9 Hibernate 框架结构图

从 Hibernate 的体系结构图 3-9 中可以看出，当实际开发中使用 Hibernate 作为数据持久层框架时，Hibernate 通过持久化对象来操作数据库，并通过 Hibernate 配置及 ORM 映射等配置文件来处理具体的数据库操作。

在了解 Hibernate 体系结构与核心接口之后，接下来研究一下 Hibernate 的工作流程，Hibernate 的工作流程是这样的：在应用中，当 Hibernate 被启动后，Hibernate 将构建 Configuration 实例，初始化该实例中的所有变量，接下来把 hibernate.cfg.xml 文件加载到 Configuration 实例中，然后通过 hibernate.cfg.xml 文件中的 mapping 节点的配置再把.hbm.xml 文件加载到 Configuration 的实例中，并利用加载配置文件后的 Configuration 实例构建一个 SessionFactory 实例，接下来由该 SessionFactory 实例创建连接，得到 Session 实例，然后由 Session 实例创建事务操作接口 Transaction 的一个实例，接下来

通过 Session 接口提供的各种方法操纵对数据库的访问，然后通过 Transaction 实例提交数据库操作结果，最后通过 Session 实例关闭 Session 连接。这样，便完成了 Hibernate 的一次工作流程。它的具体工作流程如图 3-10 所示。

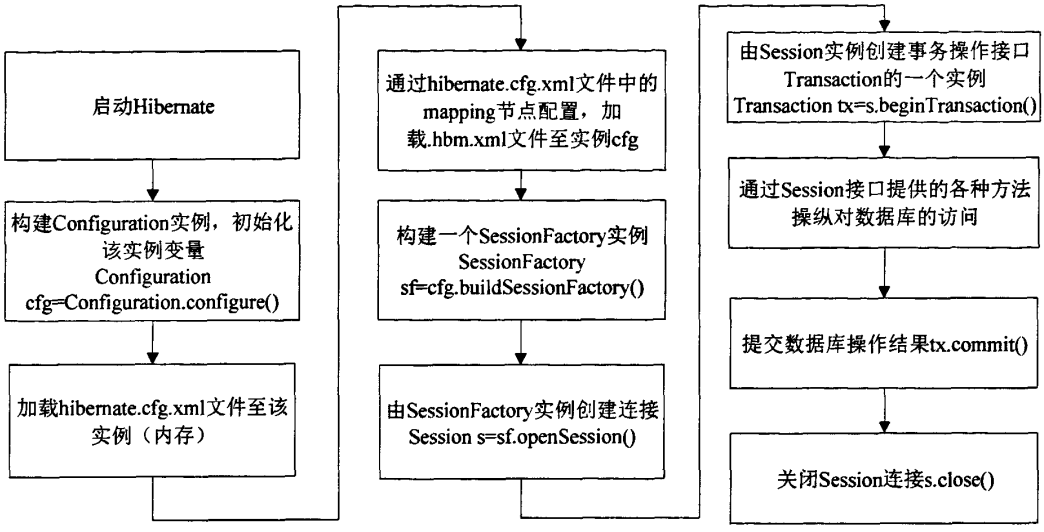

图 3-9 Hibernate 的工作流程图

3.4.3 Hibernate 对象关系映射技术

在 Hibernate 中，对象关系映射是通过一个 XML 文档来说明的。这个文档以“.hbm.xml”作为后缀，它是实体域对象与数据表之间关系的桥梁[39]。实体域对象在 Hibernate 中表现为对象关系映射中对象层的定义。下面是一个简单的商品类别的类 Sort.java 在 Hibernate 中的映射关系的实现。

```
public class Sort {
    private int id;
    private String name;
    public Sort() {}
    public void setId(int id) {this.id = id;}
    public void setName(String name) {this.name = name;}
    public int getId() {return id;}
    public String getName() {return name;}
}
```

在 Hibernate 中对应的映射文件为 Sort.hbm.xml，它的配置如下。

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="Sort" table="tb_sort">
```

```
    <id name="id" column="id" type="int">
      <generator class="increment"/>
    </id>
    <property name="name" column="name" type="string" not-null="true"/>
  </class>
</hibernate-mapping>
```

以上为实体域对象在 Hibernate 映射中的基本配置，实体域模型 Sort 类对应数据库中 tb_sort 数据表，Sort 类中封装的对象 id 与 name 对应 tb_sort 表中 id 与 name 字段。实体域对象之间的对应关系可分为一对一、一对多和多对多，因此，在 Hibernate 中的映射方式就有一对一映射、一对多映射及多对多映射[40]。在 Hibernate 中，持久化对象间的一对一关联关系通过 one-to-one 元素定义，一对多关联关系通过 one-to-many 或 many-to-one 元素定义，多对多关联关系通过 many-to-many 元素定义，具体实现时在 Hibernate 映射文件配置属性后加上相应元素配置即可。

3.5 本章小结

本章从 MVC 架构模式入手，确定要选用一个 MVC 框架作为本文架构的表示层框架，通过对现有流行的一些表示层框架的比较与分析确定本文将选用 Struts2 作为本文架构的表示层框架，根据实际情况，分别选用 Spring 和 Hibernate 作为业务层框架和持久层框架。同时对 Struts2、Spring 和 Hibernate 框架的技术特性和工作原理进行详细研究，为下面本文架构的实现提供了技术方面的基础。

第四章 企业级应用轻量级架构的设计与实现

4.1 轻量级企业级应用架构的模型构建

架构是基于 J2EE 的企业级应用中技术的根本，一个好的架构在满足企业应用需要的前提下要具有良好的松耦合的特性。本文上一章中所讨论的开源框架均可应用于企业级开发中，但各层的框架均不能完成整个企业级应用开发，只能应用于开发中的部分，因此皆不能满足完整的 J2EE 规范，所以它们不能作为企业级应用的架构。

本文要构建一个满足一些企业应用开发需要的轻量级 J2EE 架构，这个架构以上一章中所讨论的开源框架为基础，满足 J2EE 开发规范，具有良好的松耦合的特性。因此，该架构要满足以下两点要求：

(1) 通用性较强，由于在企业应用软件的实际开发中，用户业务的需求各有不同，所以要开发出满足用户所有业务需求的架构是不可能的。本架构力求做到在不涉及用户复杂业务逻辑实现的基础上能够实现用户所需的简单功能。

(2) 架构的分层结构，要使架构的各层相对独立，当一层的程序发生改变时不会影响到其它层，即架构要有良好的重用性及扩展性，使各层间都能够松耦合。

4.1.1 本架构模型的建立

通过本文 3.2.4 节的比较可知，Struts2 要比 Struts 灵活的多，因此本文将 Struts2 作为架构中表示层的实现。而 Spring 是一个实现了 IoC 模式的轻量级容器，它能够根据配置文件中的配置信息，自动装载程序所需的业务对象并对它们统一管理，业务对象访问持久对象就是通过 Spring 框架管理的。持久层实体对象由 Hibernate 统一管理，Hibernate 通过对象关系映射的方式建立实体对象和数据库之间的对应关系，当持久化的实体对象发生变化时，数据库中记录发生相应变化。实现了表现层的 Struts2 框架、业务层的 Spring 框架、持久层的 Hibernate 框架所对应的关系可用图 4-1 表示如下。

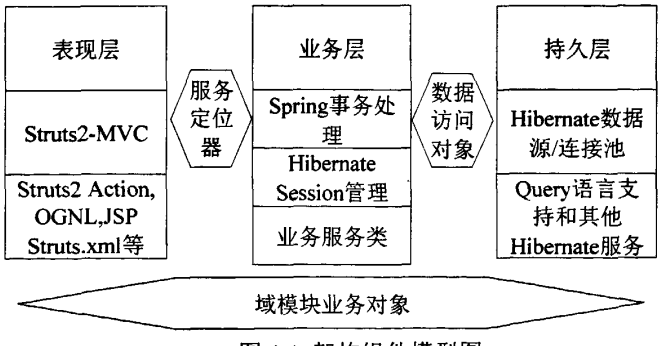

图 4-1 架构组件模型图

从图 4-1 中可以看出，实现了表示层的 Struts2 框架用到了 Struts2Action、OGNL、

JSP 及配置文件 Struts.xml 等组件。实现了业务逻辑层的 Spring 框架通过服务定位器对表示层提供业务服务，通过数据访问对象交互持久层，从而建立表现层和持久层的关系。业务层通过 Spring 框架的 IoC 控制实现与其它层的整合，通过 Spring 框架的切面编程（AOP）实现对数据访问的事务管理，在实现事务管理的过程中要与 Hibernate 框架集成。实现了持久层的 Hibernate 框架，通过对象关系映射，对数据进行持久化操作，在实现过程中，Hibernate 主要用到了 HQL 查询语言、Hibernate 连接池等自身提供的一些服务。域模型业务对象应用于架构中每一层，主要起到数据封装及数据持久化的作用，具体来说，当数据在各层间传递时，它主要应用于数据封装，当 Hibernate 对数据库进行持久化操作时，它应用于数据持久化。

4.1.2 本架构各层的细化

从 4.1.1 节中可以知道 Struts2、Spring、Hibernate 框架在架构中所起到的作用及它们所应用的组件，为使整合架构更能满足实际需要，下面对各个层深入研究。

表示层可分为页面，控制器两部分，页面主要由 JSP、JavaScript、CSS、Struts2 标签及 JSTL 标签组成，Struts2 标签结合 OGNL 表达式语言实现视图层数据的处理和展现，并可以实现界面数据的绑定，所谓界面数据的绑定，指的是将界面元素和对象的某个属性绑定在一起，实现修改和显示的自动同步。这样 Struts2 就不需要为每个页面编写专门的 FormBean，可以直接利用业务层的对象。控制器组件负责接受用户请求，更新模型以及选择合适的视图组件返回给用户。控制器组件有助于将业务层和表示层分离，这样有利于实现基于同一模型开发多种不同的视图，与此同时，也将业务逻辑的处理全部交给业务层，使业务层和表现层松耦合。

业务层主要负责具体业务的实现，从横向上看，它由业务接口和对应的业务实现类组成，接口用来体现要实现业务的具体名称，一旦接口被定义好，实现类就可以根据实际的业务需要进行编写。当具体业务发生变化时，无需改变业务接口，只需改变业务接口对应的实现类，就可以满足实际开发的需要。从纵向上看，业务层中的业务分为两大类，一类是功能单一的业务，它的实现类只实现一个单一的简单的业务，俗称细粒度业务。另一类是功能复杂的业务，它的实现类封装了很多功能单一的业务，这些被封装的简单业务所组成的复杂业务通常被称为粗粒度业务。业务层通过这样的处理，可以使复杂的业务变得简单化，同时也大大提高了程序的复用度。

持久层使用 Hibernate 的对象关系映射（ORM）功能，通过 Hibernate 提供的 HQL 来操纵实体对象。从而由 Hibernate 负责将对象的增删改查映射为数据库的操作。如同业务层一样，持久层也由接口和实现类组成，但每一个实现类的方法只负责对象的一种简单操作，即只负责增删改查中的一种。在实现本架构的过程中，提供了一个通用的持久层接口。具体的实体持久层接口可以扩展这个通用的持久层接口，并定义实现类相关的其它操作方法。这样做充分体现了本架构的可扩展性，同时也提高了程序的

复用度。

基于以上的研究，细分后的架构可用图 4-2 表示，其中，调用关系和数据传递关系分别用单向箭头线和双向箭头线表示。

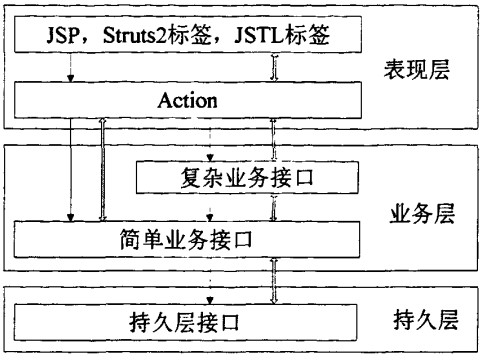

图 4-2 深入细分后的架构

由图 4-2 及各层框架的工作原理可知，架构各层间的调用关系如下：

页面中用户输入的请求事件通过过滤器链分配给合适的 Action 处理，当业务简单时，Action 直接调用简单业务接口处理业务，当业务复杂时，Action 调用复杂业务接口，复杂业务接口再调用若干简单业务接口处理业务，最后业务接口调用持久层接口来持久化数据。

在各层间的调用过程中，JSP 界面得到的数据通过 Struts2 自动进行处理，Struts2 采用直接使用域对象的方式，将域对象作为 Action 类的一个属性，这样 Action 可以方便的获取页面数据。在 Action 中，将页面获取的数据或者处理后封装到某个 VO 中再传入业务层的某个方法里或者直接将处理后的数据对象传入业务层的某个方法里。不论是复杂业务还是简单业务在其方法中的参数都是封装的 VO 或简单对象。在复杂业务中，业务方法会把 VO 对象进一步处理，或者初始化插入相应的实体对象中，并将处理后的对象传入简单业务方法中进行进一步的处理。简单业务方法直接处理数据以满足具体的业务需求，在数据处理过程中需要调用持久层中相应的方法。在业务层所调用的持久层方法中，以实体域对象作为参数传入，该持久层方法通过 Hibernate 的 O/R Mapping 技术完成对数据库的持久化操作。

由上面的各层间的调用关系和数据传递关系可以想到，在本架构中：

(1) JSP 页面上的数据经 Struts2 处理后传入到 Action 中的是简单数据，Action 中的数据经 Struts2 拦截器处理后传入到页面中的是数据集或 VO 对象。

(2) 从 Action 到业务层传入的是 VO 或简单数据对象，从业务层到 Action 传出的是 VO 或数据集。

(3) 从复杂业务层到简单业务层传入的是 VO 或简单数据对象，从简单业务层到复杂业务层传出的是单个数据或数据集。

(4) 从简单业务层到持久层传入的是实体对象或简单数据对象，从持久层到简单业务层传出的是实体对象或数据集。

上面的数据集指的是 List 或 Map，当与数据库交互一次就能满足业务需求时，返回的数据集为 List；当与数据库交互多次才能满足业务需求时，需要将数据集合封装为 Map 对象，使其只与数据库交互一次，以提高性能（典型的例子为数据字典取值）。

4.1.3 本架构的工作过程

本架构的工作过程以 Action 为界，可分为两部分，前一部分的工作过程即为 Struts2 的工作过程，这一部分在 3.2.7 节 Struts2 工作流程及原理中已经介绍，这里不再累述。后一部分为 Action 的具体执行过程，从 Action 响应页面事件开始，到业务层的调用，最后到持久层的调用，在调用上的关系可用图 4-3 表示。

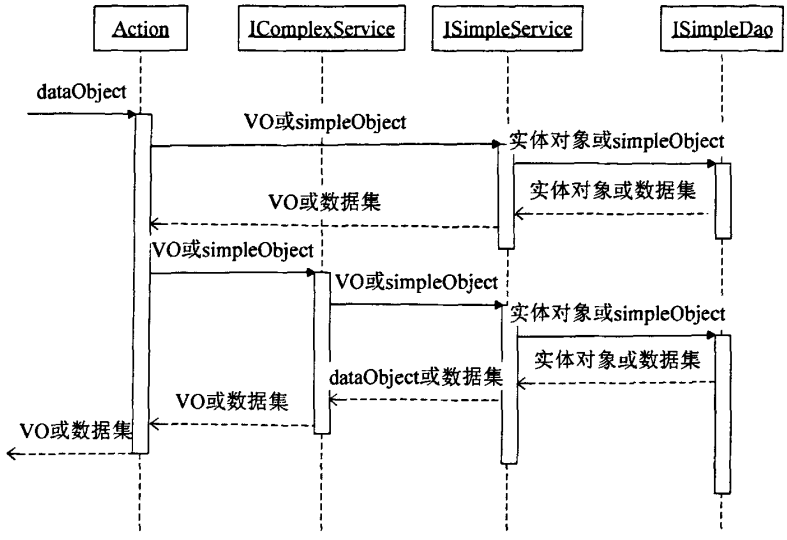

图 4-3 架构中 Action 具体执行序列图

在图 4-3 中，Action 为表示层中控制器的实现，IComplexService 表示业务层中复杂业务的实现，ISimpleService 表示业务层中简单业务的实现，ISimpleDao 表示持久层的实现。上图为本架构最基本实现过程的序列图，当 Struts2 从页面获取信息后经过一系列拦截器的处理，把处理后的数据与 Action 中域对象绑定到一起，此时 Action 中域对象值或重新封装为其它 VO 或经过其它处理，在处理之后将 VO 或处理数据传入 IComplexService 或 ISimpleService 中，当 IComplexService 接收到 VO 或处理数据后，要将 VO 或处理数据根据实际需要做进一步处理或者不做处理直接传入相应的 ISimpleService 中，ISimpleService 做简单业务处理后，调用 ISimpleDao 完成业务所需的增删改查功能。若 ISimpleDao 实现的是查询功能，则它将以实体域模型对象作为查询条件，利用 Hibernate 的 O/R Mapping 技术与数据库交互完成数据的查询并返回数据集到 ISimpleService 中，然后要么被 IComplexService 调用后返回给 Action，要么直接返回给 Action，最后将 Action 中返回的数据经 Struts2 拦截器的再次处理后返回页面并

显示。

4.2 本架构中各组件的整合配置

通过 4.1.3 节可以知道架构的工作过程，在它的工作过程中，依次要响应 JSP 事件，调用 Action 组件、业务层组件及持久层组件。JSP 事件及这些组件之间的关系是通过配置文件来确定的。其中 JSP 事件和 Action 组件的对应关系是由 Struts2 来确定的，在配置文件 struts.xml 文件中，需要对它们的关联关系进行配置；而业务层组件及持久层组件的对应关系是由 Spring 来确定的。在 Spring 的配置文件 applicationContext-Spr.xml 中，对它们的关联关系进行设定。除此之外，还有一些其它功能组件的配置，它们可以直接在 web.xml 文件中加载并在其相应组件的配置文件中配置。从文件类型来看，架构中配置文件可分为两类，一类是 xml 配置文件，一类是 properties 属性文件。

4.2.1 web.xml 的配置

web.xml 文件是一个部署描述文件，所有基于 Servlet 的 Web 应用程序都需要它。当服务器启动并载入 Web 应用时，首先要加载 web.xml 并读取它的配置信息，然后将这些配置信息应用于整个服务环境。由于本架构是基于 Struts2 和 Spring 的，因此在 web.xml 中要配入它们的初始化信息，这些信息分别是 Struts2 的核心控制器 FilterDispatcher 及初始化 Spring 的监听器 ContextLoaderListener。为了延长 action 中属性的生命周期，以便在 jsp 页面中进行访问，本架构中配置了 actionContextcleanup 过滤器，以达到和 Struts2 更好的整合。为使页面编码风格统一，防止出现乱码，本架构中还配置了自定义的字符过滤器 SetCharacterEncodingFilter。本架构中 web.xml 最基本的配置如下。

```
<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>
<filter>
    <filter-name>struts2-cleanup</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.ActionContextCleanUp
    </filter-class>
</filter>
<filter-mapping>
```

```
    <filter-name>struts2-cleanup</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext-Spr.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>com.servlet. ZdbCharEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

4.2.2 struts.xml 的配置

由 3.2.7 节可知，Struts2 是通过核心控制器 FilterDispatcher 来拦截用户请求的，FilterDispatcher 通过查找 Struts2 的核心配置文件 struts.xml 来找到处理请求的 Acition 类。因此，struts.xml 是用来完成用户请求与相应 Action 配置的，它是本架构的一个核心配置文件，它位于工程的 WEB-INF/classes 目录下。本架构根据实际业务将 struts.xml 文件分成若干个与业务相关的 Action 配置文件 struts_x.xml。本架构中 struts.xml 的配置如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<!--struts 标签是 Struts2 配置文件的根元素 -->
<struts>
<!--include 标签用于将 Struts2 应用模块化时引入其他配置文件-->
    <include file="strutsxml/struts_x.xml"/>
    <include file="strutsxml/struts_y.xml"/>
    .....
</struts>
```

通过<include../>标签引入具体业务对应的 Action 配置文件，struts_x.xml 等就是本

架构中具体业务对应的配置文件。struts_x.xml 的配置如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="x" extends="admanager-default" namespace="/x">
        <action name="list" class="xAction" method="list">
            <interceptor-ref name="params"/>
            <interceptor-ref name="islogin"/>
            <result>url</result>
            .....
        </action>
        .....
    </package>
</struts>
```

在实际开发中，会有一些公共的功能需要处理，比如登录时的身份验证，当登录成功时，能够返回相应身份对应的成功页，当登录失败或程序异常时，可以返回一个公共的失败信息显示页。在配置完具体业务对应的 Action 文件之后，本架构配置了一个全局文件 struts_base.xml，在这个配置文件中定义了一些全局的拦截器、拦截器栈和全局的返回结果类型。它的配置如下。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="admanager-default" namespace="/" extends="struts-default">
        <interceptors>
            <interceptor name="isloginInterceptor" class="com.web.interceptor.IsLoginInterceptor">
            </interceptor>
            <interceptor-stack name="islogin">
                <interceptor-ref name="isloginInterceptor"></interceptor-ref>
            </interceptor-stack>
        </interceptors>
        <global-results>
            <result name="none">/error/loginerror.jsp</result>
        </global-results>
    </package>
</struts>
```

综上可知，struts.xml 的具体配置实现过程是这样的：

(1)建立 struts.xml 文件，在 struts.xml 文件中通过<include />标签的 file 属性引入全局配置文件 struts_base.xml 及具体业务对应的 Action 配置文件 struts_x.xml 等，然后具

体配置 struts_base.xml 及 struts_x.xml 等文件。

(2)在 struts_base.xml 中，由<package />标签的 name 属性定义一个系统的全局包 admanager-default，该包通过 extends 属性继承 Struts2 自带的默认包 struts-default，由 3.2.6.2 节可知，通过这个包可获得 Struts2 提供的默认返回结果类型及拦截器。由 namespace 属性指定包所属的命名空间，admanager-default 包的命名空间是根目录，由 <interceptor />标签的 name 属性定义自定义的全局拦截器的名称 isloginInterceptor，由 class 属性指定 isloginInterceptor 的实现类的位置，由<interceptor-stack />标签的 name 属性定义拦截器栈的名称 islogin，再由<interceptor-ref />标签的 name 属性引用全局拦截器 isloginInterceptor，在全局跳转标签<global-results />下配置跳转标签<result />,由于在全局跳转中没有与之匹配的 Action，因此将<result />标签的 name 属性定义为 none，让其直接返回 JSP 页面。

(3)在 struts_x.xml 中，由<package />标签的 name 属性定义该业务的包名 x，由 extends 属性指定该包是继承于系统全局包 admanager-default 的，由 namespace 属性指定该包的命名空间是虚拟的 x 目录。由<action />标签的 name 属性指定 Action 的名字，即此 Action 能处理的 URI 名称，由 class 属性指定 Action 实现类的位置，由 method 属性指定 Action 类使用的业务处理方法名。如果没有指定该属性，系统会默认使用 excute() 方法。然后由<interceptor-ref />标签引入全局拦截器 islogin 及本业务所需要的其它拦截器，最后由<result />标签转发到相应的 Action 中。

4.2.3 Spring 整合 Hibernate 的配置

1. Spring 配置 SessionFactory

Session 是 Hibernate 持久化操作的基础，而 SessionFactory 用来创建和维护 Session 实例。Hibernate 的基础配置是围绕着 SessionFactory 展开的，SessionFactory 的配置一般在系统的 hibernate.cfg.xml 文件或 hibernate.properties 文件中。Spring 对 Hibernate 整合后，就不再需要专门修改 Hibernate 配置文件。Spring 的配置文件只有一个，默认就是/WEB_INF/applicationContext-Spr.xml， applicationContext-Spr.xml 中 SessionFactory 的配置如下。

```
<!-- 配置 Hibernate 中的 SessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <!-- 数据源引用 -->
    <property name="dataSource">
        <ref local="dataSource" />
    </property>
    <!-- 配置 Hibernate 对应的映射资源 -->
    <property name="mappingResources">
    <list>
        <value>com /dept/Dept.hbm.xml</value>
```

```
        ..................
    </list>
    </property>
    <!-- 配置 Hibernate 的属性，包括断言，show_sql 等 -->
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">${hibernate.dialect}</prop>
            <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
            <prop key="hibernate.jdbc.fetch_size">${hibernate.jdbc.fetch_size}</prop>
            <prop key="hibernate.jdbc.batch_size">${hibernate.jdbc.batch_size}</prop>
            <prop key="hibernate.connection.release_mode">${hibernate.connection.release_mode}
            </prop>
        </props>
    </property>
</bean>
```

通过上述配置，应用启动时就会自动创建 Hibernate 的 SessionFactory，Spring 将管理 SessionFactory，为架构的持久层提供支持。SessionFactory 主要由数据源、Hibernate 对应的映射资源及 Hibernate 的属性三部分组成。数据源由 dataSource 属性引入。Hibernate 映射资源由 mappingResources 属性配置，Hibernate 映射资源为 x.hbm.xml 等文件。Hibernate 的属性配置通过 hibernateProperties 属性定义。这部分关于 Hibernate 属性的定义和数据源的定义用到了 EL（Expression Language）表达式，本架构将它们的值都定义在 init.properties 文件中，init.properties 文件通过 Spring 提供的 PropertyPlaceholderConfigurer 类在配置文件中引入。它在 Spring 配置文件中的配置如下。

```
<bean id="placeholderConfig"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
        <value>classpath:init.properties</value>
    </property>
</bean>
```

在数据源中主要配置了连接数据库的驱动类、地址、数据库的用户名和密码及 Hibernate 连接池的一些相关信息。数据源的详细配置如下。

```
<!-- 配置数据源，连接池使用 c3p0 -->
    <bean id="dataSource"
        class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close" dependency-check="none">
        <property name="driverClass"><value>${datasource.driverClassName}</value></property>
        <property name="jdbcUrl"><value>${datasource.url}</value></property>
        <property name="user"><value>${datasource.username}</value></property>
        <property name="password"><value>${datasource.password}</value></property>
        <property name="acquireIncrement"><value>${c3p0.acquireIncrement}</value></property>
```

```
        <property name="initialPoolSize"><value>${c3p0.initialPoolSize}</value></property>
        <property name="minPoolSize"><value>${c3p0.minPoolSize}</value></property>
        <property name="maxPoolSize"><value>${c3p0.maxPoolSize}</value></property>
        <property name="maxIdleTime"><value>${c3p0.maxIdleTime}</value></property>
        <property name="idleConnectionTestPeriod">
            <value>${c3p0.idleConnectionTestPeriod}</value>
        </property>
        <property name="maxStatements"><value>${c3p0.maxStatements}</value></property>
        <property name="numHelperThreads"><value>${c3p0.numHelperThreads}</value>
        </property>
    </bean>
```

本架构中，Hibernate 属性及数据源的详细配置信息均放在 init.properties 文件中，init.properties 配置文件如下所示。

```
#数据源配置
datasource.type=mysql
datasource.driverClassName=org.gjt.mm.mysql.Driver
datasource.url=jdbc:mysql://127.0.0.1:3306/zdb?characterEncoding=GBK
datasource.username=root
datasource.password=
datasource.maxActive=10
datasource.maxIdle=2
datasource.maxWait=120000
datasource.whenExhaustedAction=1
datasource.validationQuery=select 1 from dual
datasource.testOnBorrow=true
datasource.testOnReturn=false
#连接池配置
c3p0.acquireIncrement=3
c3p0.initialPoolSize=3
c3p0.idleConnectionTestPeriod=900
c3p0.minPoolSize=2
c3p0.maxPoolSize=50
c3p0.maxStatements=100
c3p0.numHelperThreads=10
c3p0.maxIdleTime=600
#hibernate 相关配置
hibernate.dialect=org.hibernate.dialect.MySQLInnoDBDialect
hibernate.jdbc.batch_size=25
hibernate.jdbc.fetch_size=50
hibernate.show_sql=true
hibernate.connection.release_mode=after_transaction
```

通过上述配置，应用启动时即会自动创建 Hibernate 的 SessionFactory，Spring 将管理 SessionFactory，为系统的持久层提供支持。

2. Spring 配置事务管理

在本架构中，持久层组件并不管理事务，而将事务管理延迟到业务逻辑层管理。通过 Spring 的声明式事务管理，让业务逻辑层对象的方法具备事务性。Spring 通过 TransactionProxyFactoryBean 设置 Spring 的事务代理。定义 TransactionProxyBean 时，必须提供一个相关的事务管理器 TransactionManager 的引用和事务属性 transactionAttributes。transactionAttributes 包括对所代理的那些方法提供事务支持，例如定义一个以 add 开头的方法，那它就可以有事务管理了，对于它里面的所有操作，都可以实现事务机制，若有异常就回滚事务。在 Spring 中常用事务类型[27]有：

PROPAGATION_REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION_MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION_REQUIRES_NEW：新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，把当前事务挂起。

PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则进行与 PROPAGATION_REQUIRED 类似的操作。

Spring 配置文件中关于事务管理的配置如下。

```
<!-- 配置事务管理器 bean  -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <!-- 为事务管理器注入 sessionFactory" -->
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
    <!-- 事务控制代理抽象定义  -->
    <bean id="transactionProxy"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
        abstract="true">
        <!-- 为事务代理 bean 注入一个事物管理器 -->
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
        <property name="transactionAttributes">
        <!-- 定义事务传播属性 -->
```

```
            <props>
              <prop key="add*">PROPAGATION_REQUIRED</prop>
              <prop key="delete*">PROPAGATION_REQUIRED</prop>
              <prop key="update*">PROPAGATION_REQUIRED</prop>
              <prop key="find*">PROPAGATION_REQUIRED,readOnly</prop>
              <prop key="release*">PROPAGATION_REQUIRED</prop>
              <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
            </props>
        </property>
    </bean>
```

在本架构中选择了常用的事务类型："PROPAGATION_REQUIRED， readOnly"，它的含义是指它的数据为只读，这样有助于提高读取的性能。上述配置中定义TransactionProxyFactoryBean 时设置了"abstract=true"属性，使用 abstract 属性可以让代理对象共享一个定义好的事务属性，使配置简化。代理对象通过使用 parent="transactionProxy"设置来继承 transactionProxy 获取事务属性。

3. Spring 配置业务层及持久层的实现

以上 Spring 对 sessionFactory 及事务的配置为本架构的基本配置，它们均未涉及到具体的业务实现。在实际开发中，当完成某一模块的 Action 组件、业务层组件及持久层组件的开发后，它们之间的依赖关系由 Spring 容器来维护，需要在 Spring 配置文件中加入各层的具体业务类所对应的配置信息，假设业务对应的 Action 中实现类为 zdbAction，对应的业务层实现类为 zdbService，对应的持久层实现类为 zdbDao，则它们的配置分别如下。

(1)Action 相关配置

```
<bean id="zdbAction" class="com.web.actions. zdbAction" singleton="false">
    <property name="zdbService">
        <ref bean="zdbService"/>
    </property>
    .....
</bean>
```

(2)业务逻辑服务类相关配置

```
<bean id="zdbTarget" class="com..zdb.zdbService">
    <property name="zdbDao">
        <ref bean="zdbDao"/>
    </property>
    .....
</bean>
<bean id="zdbService" parent="transactionProxy">
    <property name="target">
        <ref bean="zdbTarget"/>
    </property>
```

```
</bean>
```

(3)持久层类相关配置

```
<bean id="zdbDao" class="com.zdb.zdbDao">
    <property name="sessionFactory">
        <ref bean="sessionFactory"/>
    </property>
</bean>
```

在上述配置中各层间组件的依赖注入均采用 Spring 的 setter 方法注入方式，因此在各层相应组件的实现中需要提供组件的 setter 方法。

4.2.4 Spring 整合 Struts2 的配置

由 4.1.3 节可知，在 Struts2 框架下开发 Web 应用时，业务逻辑组件由控制器组件 Action 来调用。在实际开发过程中通常是通过工厂模式来获得业务逻辑组件实例或利用 IoC 容器来管理业务逻辑组件的。本架构中采用 Spring IoC 容器来管理业务逻辑组件。为了让 Action 能够访问 Spring 管理的业务逻辑组件，需要由 Spring 来管理 Action，并利用依赖注入为 Action 注入业务逻辑组件。Struts2 与 Spring 的整合需要用到 Spring 插件包，这个包是同 Struts2 一起发布的。Spring 插件是通过覆盖 Struts2 的 ObjectFactory（对象工厂）来增强核心框架对象的创建。当创建一个对象的时候，它会用 Struts2 配置文件（struts.xml）中的 class 属性去和 Spring 配置文件（applicationContext-Spr.xml）中的 id 属性进行关联，如果能找到则由 Spring 创建，否则由 Struts2 框架自身创建，然后由 Spring 来装配。架构中 Struts2 整合 Spring 的 IoC 支持的实现过程如下。

首先，将 struts2-spring-plugin-*.jar 文件包放到 WEB-INF/lib 目录下。在这个插件包中有个 struts-plugin.xml 文件，该文件将 struts.properties 文件设置的框架常量 struts.objectFactory 给覆盖了，将它覆盖为“spring”。框架常量的全称是：org.apache.struts2.spring.StrutsSpringObjectFactory。该文件是在系统启动时由 Struts2 框架自动加载的，它的内容如下：

```
<struts>
  <bean type="com.opensymphony.xwork2.ObjectFactory"
    name="spring"
    class="org.apache.struts2.spring.StrutsSpringObjectFactory" />
    <!--  Make the Spring object factory the automatic default -->
    <constant name="struts.objectFactory" value="spring" />
    <package name="spring-default">
      <interceptors>
        <interceptor name="autowiring"
            class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
        <interceptor name="sessionAutowiring"
            class="org.apache.struts2.spring.interceptor.SessionContextAutowiringInterceptor"/>
      </interceptors>
```

```
    </package>
</struts>
```

默认情况下所有框架创建的对象都是由 ObjectFactory 实例化的，ObjectFactory 提供了与其它 IoC 容器如 Spring 等集成的方法。覆盖这个 ObjectFactory 的类必须继承 ObjectFactory 类或者它的任何子类，并且要带有一个不带参数的构造方法。这里用 org.apache.struts2.spring.StrutsSpringObjectFactory 代替了默认的 ObjectFactory。然后配置 Spring 监听器，将 Spring.jar 包添加到 WEB-INF/lib 目录下。这一步在初始化 web.xml 文件时已经完成。最后，利用 Spring 配置文件来注册对象。之后的步骤就和单独使用 Spring 一样，能很方便地完成 Strut2 与 Spring 的集成。在 Struts2 配置文件中配置 Action 时，只需将 class 属性和 Spring 属性文件中 bean 的 id 属性保持一致，系统即会自动通过 Spring 来管理 Action。

4.2.5 Spring 整合 Log4j 的配置

在软件运行与维护中，日志文件的管理具有重要地位。即使在软件测试中，也可以通过日志文件跟踪代码运行轨迹。Log4j 是 Apache 的一个开源项目，通过一个配置文件，可以灵活有效的管理日志记录[41]。

本文架构中 Spring 配置 Log4j 的过程如下：

1.在 src 目录下建立 log4j.properties 文件：

```
#定义缺省的日志级别和输出对象
log4j.rootLogger=info,console
#设定控制台输出模式
log4j.appender.console=org.apache.log4j.ConsoleAppender
#设定日志的输出模式
log4j.appender.console.layout=org.apache.log4j.SimpleLayout
```

2.在 Web 中为 Spring 配置 Log4j，具体操作是在 web.xml 中加入如下配置信息：

```
<!--由 Sprng 载入的 Log4j 配置文件位置-->
<context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>/WEB-INF/classes/log4j.properties</param-value>
</context-param>
<!--Spring 刷新 Log4j 配置文件的间隔,单位为 millisecond-->
<context-param>
    <param-name>log4jRefreshInterval</param-name>
    <param-value>30000</param-value>
</context-param>
<!-- 设置 Spring 的 log4j 监听器 -->
<listener>
    <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
</listener>
```

经过上述配置，就可以利用 Log4j 进行日志输出了。利用 Spring 来配置 Log4j 可以动态地改变记录级别和策略，不需要重启 Web 应用，而且配置简单，操作方便。

4.3 本架构的优化

在 4.2 节中，详细完成了本架构的整合及配置，按照架构所设计的模型实现 Struts2、Spring 及 Hibernate 的整合。在整合过程中，加入了日志管理功能，使本架构能够提供更好的服务。由于本架构是提供给企业应用的，所以还要整合一些插件并提供部分公有接口和实现类以更好的完善企业级开发规范。

4.3.1 Spring 整合 jBPM

1. jBPM 简介

在实际开发中，经常会遇到一些工作流程的应用，例如 OA 工作流系统、业务操作流程系统等。而开发一套完整的工作流体系模块费时费力，因此，在开发中通常选择一套开源的工作流产品。jBPM（Java Business Process Management）就是一款开源的工作流产品，它采用一种轻量级的 XML 结构的流程描述语言 JPDL（jBPM Process Definition Language），这种语言简单易懂[42]。jBPM 使用 Hibernate 作为持久层工具，可以在 Oracle、DB2、Sybase、Microsoft SQL Server 等常用的主流数据库下使用。jBPM 是一款功能强大、小巧灵活的工作流引擎，它既可以以嵌入式模式与业务模块运行在同一个应用之内，也可以以独立的模式单独的运行在一个应用中。当需要采用嵌入式的方式将 jBPM 与 J2EE 工程结合使用时，只需要将 jBPM 相关的 jar 文件和几个配置文件放在工程里就可以使用 jBPM 流程引擎，而不需要额外配置一个流程服务器。

2. Spring 与 jBPM 的整合

由于 jBPM 引擎本身复杂，因此本文中仅将 jBPM 作为架构中的一个额外的插件集成到本架构中，以完善本架构的功能。在集成过程中主要涉及到它与 Spring 的整合。接下来就研究一下它们在开发中是如何整合的。

首先，在 Spring 配置文件 applicationContext-Spr.xml 中的 sessionFactory 配置部分添加工作流的映射类，使业务与 jBPM 共用一个 sessionFactory。添加部分的具体实现如下。

```
<property name="mappingLocations">
    <list>
        <!-- jbmp's hbm.xml -->
        <value>classpath*:/org/jbpm/**/*.hbm.xml</value>
    </list>
</property>
```

接下来在 web.xml 中添加新的 Spring 配置文件 applicationContext-workflow.xml，在

该文件中，首先要引用业务的 sessionFactory，然后加载 jBPM 的配置文件 jbpm.cfg.xml，最后配置 jBPM 的 JbpmTemplate 类，具体实现如下。

```
<!-- jbpm configuration -->
    <bean id="jbpmConfiguration"
        class="org.springmodules.workflow.jbpm31.LocalJbpmConfigurationFactoryBean">
        <property name="sessionFactory" ref="sessionFactory" />
        <property name="configuration" value="classpath:jbpm.cfg.xml" />
        <property name="processDefinitions">
            <list><ref local="holidayWorkflow" /></list>
        </property>
    </bean>
    <bean id="holidayTemplate"
        class="org.springmodules.workflow.jbpm31.JbpmTemplate">
        <constructor-arg index="0" ref="jbpmConfiguration" />
        <constructor-arg index="1" ref="holidayWorkflow" />
    </bean>
<!-- end jbpm configuration -->
```

以上为 Spring 与 jBPM 最基础的配置，不涉及到事务的整合，由于 jBPM 在本架构中仅作为一个插件，因此，这里不再研究其更详细的配置。

4.3.2 本架构中字符过滤器的实现

在实际的开发中，对于中文汉字，经常会遇到乱码问题，一旦出现乱码，会影响页面的显示效果及数据的存储，因此，在架构设计中这一部分也很重要。

本架构在 web.xml 初始化时，加载了一个自定义的字符过滤器 SetCharacterEncodingFilter 并设置了字符集编码为 UTF-8。使用它可以将汉字转化为 web.xml 中所设置的编码方式，它的实现如下：

```
public class ZdbCharEncodingFilter implements Filter {
    private String zdbCharSet;
    public void init(FilterConfig filterConfig) throws ServletException {
        zdbCharSet = filterConfig.getInitParameter("encoding");
        if(zdbCharSet == null && zdbCharSet.length() < 1) {
            zdbCharSet = "UTF-8";
        }
    }
    public void doFilter(ServletRequest parm1, ServletResponse parm2,
        FilterChain parm3) throws IOException, ServletException {
        parm1.setCharacterEncoding(this.getZdbCharSet());
        parm3.doFilter(parm1, parm2);
    }
    public void destroy() {
        this.setZdbCharSet(null) ;
```

```
    }
    public void setZdbCharSet(String zdbCharSet) {
        this.zdbCharSet = zdbCharSet;
    }
    public String getZdbCharSet() {
        return (this.zdbCharSet);
    }
}
```

大多数架构在实现过程中，都存在字符编码不统一问题，只要定义了字符过滤器，就可以实现字符编码的统一。本架构中，使用的是 UTF-8 字符编码方式，如果需要使用其它的编码方式。只需要将 web.xml 中相应参数值设置为其它编码方式即可。

4.3.3 本架构中的数据封装

本架构中通过 Spring 框架提供的 HibernateDaoTemplate 模板封装 Hibernate 底层操作，该模板提供了数据增删改查的方法。为了使本架构中的数据可视度更加清晰明了并且在应用、调试及测试时处理时更加方便，本架构中以值/对象形式封装了查询出来的对象或者数据集中的对象。

本架构提供了对数据库中的增删改查操作的接口及实现类，在持久层中这些接口和实现类的结构可用图 4-4 表示如下。

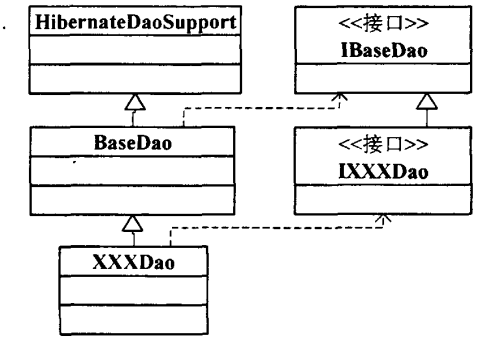

图 4-4 持久层类结构图

在图 4-4 中，IBaseDao 通过基础接口对所有实体持久层接口的通用方法进行了抽象，并提供泛型的支持。基类 BaseDao 扩展于 HibernateDaoSupport 类，并实现了通用基础接口 IBaseDao。HibernateDaoSupport 类用来简化 Hibernate 持久层的开发，IXXXDao 和 XXXDao 为实体持久层接口及其实现类。

接口 IBaseDao 和实现类 BaseDao 所包含的方法如下表 4-1 所示。

表 4-1 接口 IBaseDao 提供的方法列表

方法	描述
public void save(Object object)	根据给定的持久化对象.添加记录
public void delete(Object object)	根据给定的持久化

	对象,删除记录
public void update(Object object)	根据给定的持久化对象,更新记录
public List find(String where)	执行 HQL 查询,返回查询的结果
public void saveOrUpdate(Object object)	通过主键 ID 返回一个数据对象
public Object findById(Class cla,Inteqer id)	根据给定的对象,保存或更新对象
public Session openSession()	获得 Hibernate 中的一个 session
Public Pager getPager(String hsql,String currentPage,String pagerMethod)	获取分页对象

4.3.4 本架构中多表关联查询的优化

由 3.4.3 节可知，Hibernate 中所提供的持久化对象间的关系可分为一对一关联、一对多关联及多对多关联，由它们的配置可知在关联中一个对象要映射另一个对象，这样就表明一张关系数据表要关联另一张关系数据表的所有属性，如果所查询的信息不需要显示所有的属性，这样查询就造成资源浪费，势必降低系统性能。本架构中提供一种方法解决该问题，提升系统性能，现用如下事例说明：

假设有两个持久化类 A.java,B.java，它们如下所示：

```
public class A{
    String id; // 主键标识
    String name; // 姓名
    int age; // 年龄
    .....省略 set/get 方法.....
}
public class B{
    String id; // 主键标识
    String address; // 地址
    String school; // 学校
    String a_id; // 以 A 中主键作为外键
    .....省略 set/get 方法.....
}
```

现要在页面上统计只有姓名和地址的列表，其它信息都不显示，通常做法是将 A 和 B 两个持久化类做关联，即在配置文件 A.hbm.xml、B.hbm.xml 和持久化类 A、B 中做相应的一对一或一对多配置，假如是 A 与 B 关联，在 Hibernate 查询时就会查出 A 与 B 的持久化对象的所有属性，而除了姓名与地址外其它的信息都是无用的，这样就

降低了查询效率。

本架构中此类问题优化如下：首先，在持久化类 A 中添加成员变量 String address; 及其相应的 set/get 方法。然后在 A 中添加带有 name 和 address 属性的构造方法，既：

```
public A(String name,String address){
    this.name = name;
    this.address = address;
}
```

然后在持久层实现时用新的构造方法作为 HQL 中的查询部分，此处实现可表示如下：

```
public List<A> getAs(A a){
    String querySQL = "select new A(a.name, b.address) from A a , B b   where a.id = b.a_id";
    return getHibernateTemplate().find(querySQL);
}
```

最后页面上只要通过返回的列表对象就能得到相应结果。这样做减少了查询关系表中不必要的属性，提高了查询性能。

4.3.5 本架构中分页功能的实现

在 web 应用系统中，数据分页显示必不可少。分页技术的好坏直接影响到架构中查询部分的性能，因此在架构设计中会关注这一方面。

本架构中采用了 Hibernate 提供的分页技术，Hibernate 提供了一个支持跨系统的分页机制，这样无论底层是什么样的数据库都能用统一的接口进行分页操作。

Pager 类在本架构中处理分页功能，它是一个包含了分页信息的类，包括页面总数、记录总数、当前第几页等，它的实现如下所示：

```
public class Pager {
    private int totalRows;//记录总数
    private int pageSize = 10;//设置一页显示的记录数
    private int currentPageNum;//当前页码
    private int totalPages;//总页数
    private int startRow;//当前页码开始记录数
    private List elements;//记录列表
    public Pager() {}
    public Pager(int _totalRows) {//构造 pager 对象
        totalRows = _totalRows;
        totalPages=totalRows/pageSize;
        int mod=totalRows%pageSize;
        if(mod>0){totalPages++;}
        currentPage = 1;
        startRow = 0; }
    public void first() {//首页
```

```
        currentPageNum = 1;
        startRow = 0; }
    public void previous() {//上一页
        if (currentPageNum == 1) {return;}
        currentPageNum --;
        startRow = (currentPageNum - 1) * pageSize; }
    public void next() {//下一页
        if (currentPageNum < totalPages) { currentPageNum ++;}
        startRow = (currentPageNum - 1) * pageSize; }
    public void last() {//尾页
        currentPageNum = totalPages;
        startRow = (currentPageNum - 1) * pageSize; }
    public void refresh(int _ currentPageNum) {//刷新 pager 对象
        currentPageNum = _ currentPageNum;
        if (currentPageNum > totalPages) {last();} }
    以下省略 JavaBean 中的 set、get 方法
    .....
```

业务逻辑类通过调用相应的持久层类获取 Pager 对象，然后通过 Action 将 Pager 返回给视图进行展现。在 Action 中，分页功能作为一个共有类被提出，Action 在实现时只需继承它。这个共有类为 AbstractAction.java，它的实现如下：

```
public abstract class AbstractAction extends ActionSupport {
    protected String where = "";
    //分页需要属性
    protected Pager pagerObj;
    protected String currentPageNum;
    protected String pagerMethod;
    public Pager getPagerObj() {return pagerObj;}
    public void setPagerObj(Pager pagerObj) {this.pagerObj = pagerObj;}
    public String getPagerMethod() {return pagerMethod;}
    public void setPagerMethod(String pagerMethod) {this.pagerMethod = pagerMethod;}
    public String getWhere(){return where;}
    public void setWhere(String where){this.where = where;}
    public String getCurrentPageNum(){return currentPageNum;}
    public void setCurrentPageNum(String currentPageNum)
    {this.currentPageNum = currentPageNum;}
}
```

由上节数据封装可知，在 IBaseDao 接口中设计了一个获取 Pager 对象的通用方法 getPager()，实体持久层类通过继承 BaseDao 类来获取此方法。其中，getPager()方法的关键代码如下。

```
Query query = session.createQuery(hsql);
//从当前页记录数开始
query.setFirstResult(pager.getStartRow());
//取出 pageSize 个记录
```

```
query.setMaxResults(pager.getPageSize());
items = query.list();
pager.setList(items);
```

大多数数据库都提供数据部分读取机制，通过这种机制实现分页功能性能高。Hibernate 便通过上述代码实现了跨系统的分页机制。

4.4 本架构在设计模式中的表现

经过前面对本文设计架构的模型建立、各层细分、各层的配置整合及完善，本架构已经形成。接下来从本架构的整体结构入手研究一下它的各层间数据关系在设计模式中的表现，以验证它的通用性。

从总体来看，本架构可分三层，即表示层、业务层和持久层。表示层是由 Struts2 组件、JSTL 组件及 Action 组件等组成，表示层通过业务层接口调用相应的业务；业务层是由 Spring 框架控制，它提供了 Spring 框架标准组件，业务层通过持久层接口调用持久层中持久化对象；持久层由 Hibernate 管理。在这三层基础之上，还使用了 Log4j 组件、Util 组件等。

由以上对本架构的结构分析可知，本架构完全符合 MVC 三层架构模式，而 MVC 三层架构的关系在设计模式中的表现如下。

MVC 中视图与模型的关系为观察者模式，观察者（Observer）模式是指定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新[43]。 MVC 通过建立一个“定购/通知”机制将视图和模型分离，一个模型可以有多个视图，当模型的数据改变后会通知它的所有视图。本架构中，一个业务对象可被多个 Action 调用，从而能被多个视图响应。

视图与视图的关系为组合模式，组合（Composite）模式是指将对象组合成树形结构以表示“整体－部分”的层次结构，使得对单个对象和复合对象的使用具有一致性。MVC 用 View 类的子类 CompositeView 类来支持嵌套视图[39]。MVC 将一些对象划为一组，并将该组对象当作一个对象来使用。本架构中，Action 有时会返回多个数据集对象，将这些对象进一步封装为一个大的数据集，然后响应给页面（例如数据字典）。

视图与控制器的关系为策略模式，策略（Strategy）模式是指定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换[39]。算法的变化可以独立于使用者。MVC 将响应机制封装在 Controller 对象中，MVC 允许在不改变视图外观 View 的情况下改变视图对用户输入的响应方式。View 使用 Controller 的某个实例来实现一个特定的响应策略，实行不同的响应策略需要用不同的 Controller 实例替换即可。本架构中，Action 中将数据响应到页面可采用转发与重定向的方式。

由于本架构中各层间数据关系在实现上满足一些设计模式，因此它具有通用性。

4.5 本章小结

本章是本文的核心，通过上一章所选用的各层框架的整合，实现了本文架构。在本章中，首先确定了本架构的总体的设计目标，并根据这一目标构建了架构的总体模型，在架构模型构建过程中，建立了本架构的总体模型，细化分析了对模型的各个层次，并分析了本架构的工作原理，然后通过 Struts2、Spring 及 Hibernate 的整合配置实现了本架构。在此基础上通过与 jBPM 插件的集成、提供统一的字符集编码功能、数据封装机制和统一的分页功能进一步的完善了本架构。最后通过分析本架构的总体结构并从 MVC 架构模式在设计模式中的表现分析了本架构中各层间数据关系在设计模式中的表现，从理论上验证本架构的通用性。在下一章中，将把本章实现的架构应用到实践中。

第五章 本架构在再担保业务系统中的应用

5.1 再担保业务系统的概述

本系统是面向于再担保公司开发的，主要用于中小型企业的担保业务及再担保业务。本系统中业务类型可分为四类：直保业务；再担保业务；资格认定、评级授信业务；委托贷款、过桥、理财产品业务。为了满足再担保公司业务操作流程的便利，需要建立一套能够管理所有业务流程的系统。

5.2 系统需求分析

5.2.1 系统总体需求

再担保公司可以对企业做担保业务也可以对担保公司做再担保业务，因此，它可以接待企业和担保公司，它可以受理企业的项目和担保机构的项目。

1.对于担保机构，在受理项目之前，首先要对担保公司进行资格认定，当担保公司满足条件时才对其项目进行受理。当对担保公司的项目受理时，要对它进行评级授信，以确定它在受理时应享有的级别。对于企业，则对其项目直接受理。

2.在企业项目或机构项目受理完成之后，担保业务部负责对受理的项目进行评审，风险管理部负责对受理项目提出风险判断意见。

3.分公司担保业务部评审委员会和总公司担保业务部、风险管理部、专家委员会、评审委员会分别从不同角度对项目进行评审、复审、风险判断评议、决策。

4.实行分权受理管理制度，对于担保机构申请的再担保项目，额度在 500 万元以下的，由分公司评审，决策后报公司备案，额度在 500 万元以上的或直保项目，经分公司评审并同意担保的，报总公司复审，决策。

5.2.2 系统的流程分析

为了满足系统的需求，在系统中详细分析了项目的受理、审批及各部门领导做出的决策的每一个步骤，让直保或再担保项目的每一步在系统中都有保存记录，以便于在每一步中各部门领导都能看到对项目的处理情况。该系统中项目均是由分公司受理开始，项目流程大体上类似，因此，这里仅以再担保业务流程来说明本系统中的整体流程。其中，再担保业务分公司的流程如图 5-1 所示。

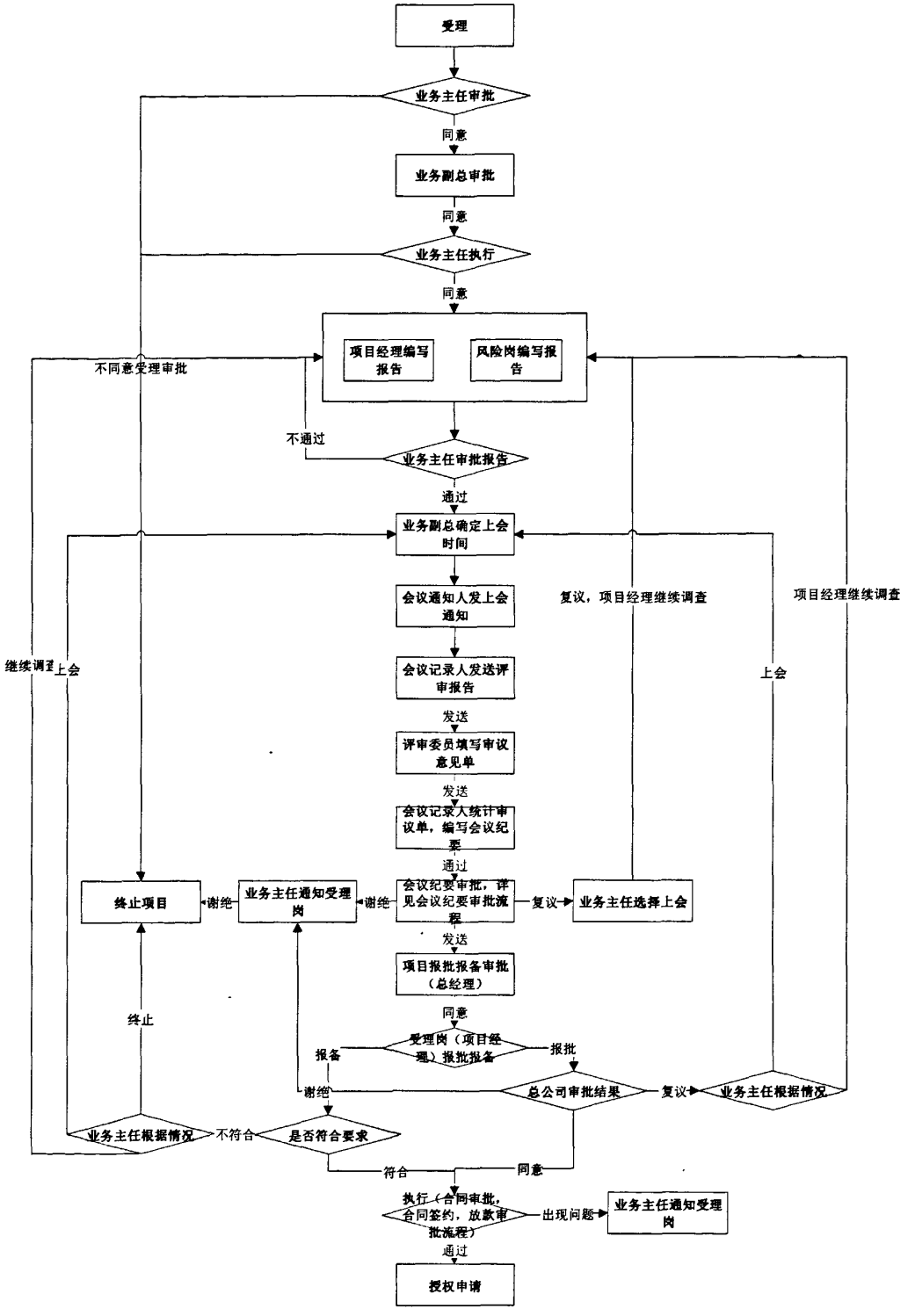

图 5-1 再担保业务分公司流程

在分公司总经理审批之后，如果同意对项目担保，便上报总公司，总公司流程中，如果上报的项目超过 500 万元，流程自动流转到限上操作，如果上报的项目低于 500

万元，则流程走限下操作。再担保业务总公司的业务流程如图 5-2 所示。

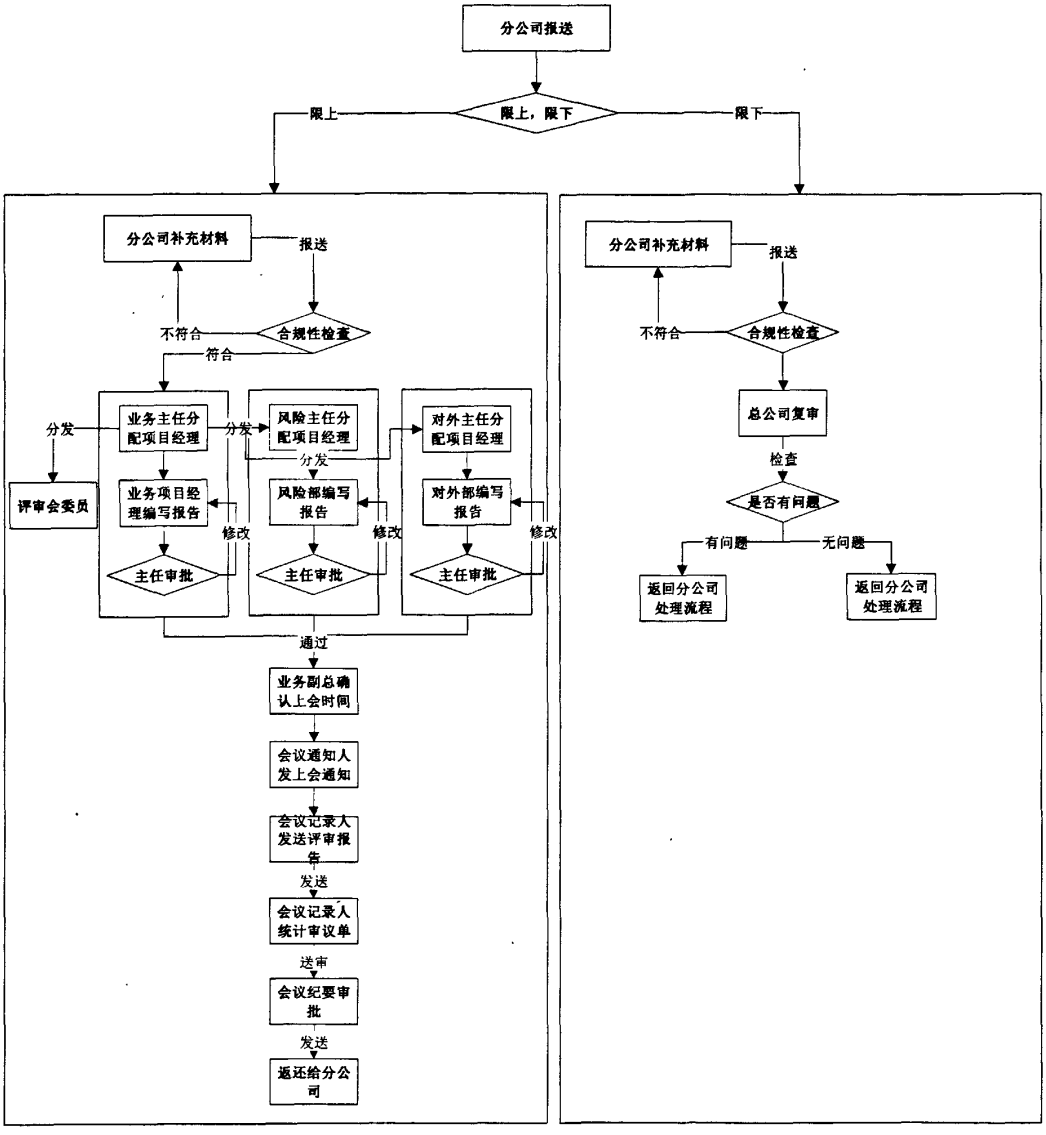

图 5-2 再担保业务总公司流程

5.2.3 系统用例分析

根据系统流程的设计及实际的需要，本系统用例可抽象为管理员与系统用户。其中，管理员负责系统中的人员维护、系统人员的角色维护、各个角色所应有权限的分配及后台数据处理中所应用的字典基础信息的维护；系统用户根据自己所拥有的权限，完成自己在流程操作对应的操作，如项目管理、风险预警、统计查询等。系统用例分析图如图 5-3 所示。

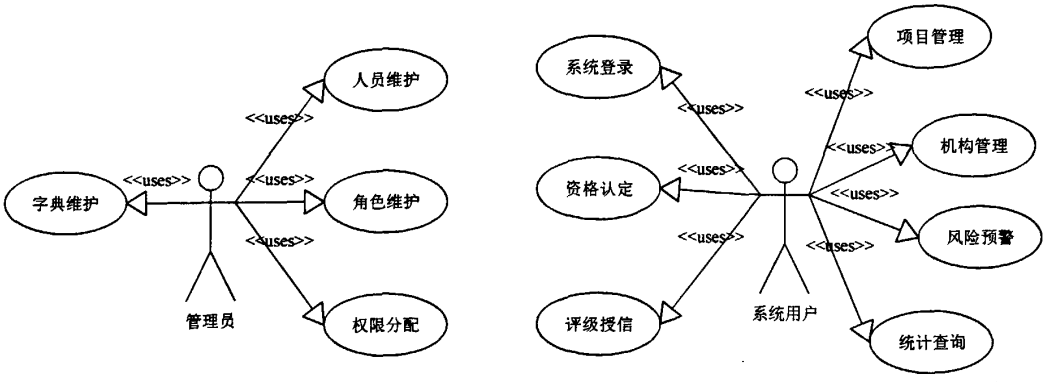

图 5-3 再担保业务系统用例分析图

5.3 系统的设计

5.3.1 系统模块的划分

根据系统流程的设计、用例分析及实际的需要，现将系统的主要部分分为以下五大模块：项目管理、机构管理、风险预警、统计查询、系统维护，这五大模块中的每一个模块根据实际需求又可分为若干个子模块。这些功能模块的结构如图 5-4 所示。

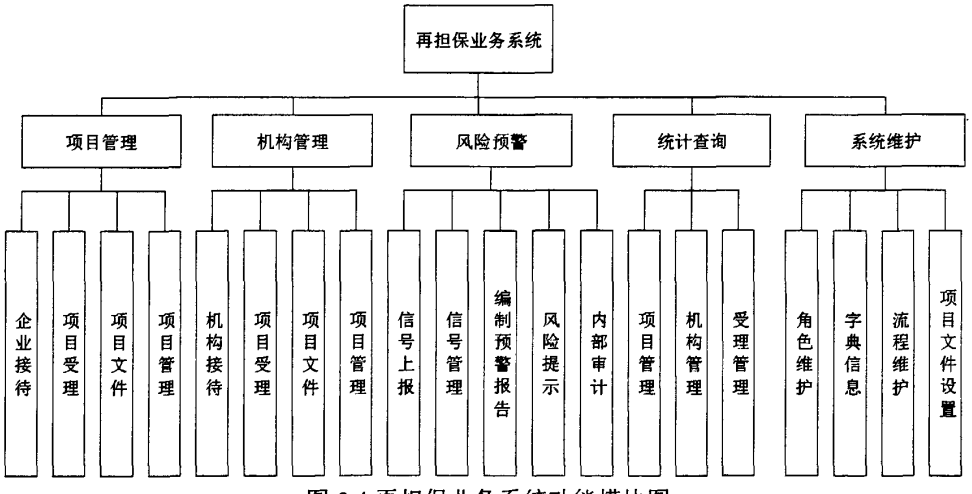

图 5-4 再担保业务系统功能模块图

图 5-4 所示功能模块中，重要的功能模块功能如下。

企业接待：再担保公司用它来接待需要做担保业务的企业，记录企业的基本信息及企业所咨询的内容，所接待的企业在再担保公司可以查询以前是否被接待过，如果以前被接待过，可以修改以前录入的企业信息，也可以添加新的企业信息。

项目受理：再担保公司对担保机构进行资格认定，如果满足受理要求，则对担保机构项目进行评级授信，确定其满足的等级，然后再对其进行受理，对企业项目则直接进行受理。受理过程中可以保存和提交受理信息，一旦提交，则开启了工作流程，审

批过程中决定项目是提交或终止。

项目文件：项目文件是在项目受理之后，业务执行人在流程中的每一步所需要的业务跟踪文件，它包括分公司的初审报告、申请单、总公司的复审报告、会议纪要、合同文件等所有与项目相关的文件。

机构接待：与企业接待类似，再担保公司用它来接待担保机构，记录被接待的担保公司的一些基本信息及咨询内容。

信号上报：如果公司人员认为项目有问题，他会向本部门主任发出预警信息，预警信号分为绿、黄信号、红、橙信号及重大事件信号。预警的严重程度可依次用绿、黄、红、橙、重大事件表示。其中，绿、黄信号上报时由本公司处理，红、橙、重大事件一定要由总公司处理。信号上报中，可以保存和提交信息。保存信息不上报，即不走工作流程。

信号管理：是对信号发起人所发起的信号信息进行的管理，通过它可以查询所发起的预警信息列表，及预警详细信息，包括所发起的信息是否上报，可以修改和删除发起的但没有上报的预警信息，一旦上报，则不能修改和删除。

项目管理：这里的项目管理是指统计查询中的部分，不同角色的用户登录后它可以查看不同阶段下直保/再担保下的项目信息。本阶段下未提交的项目可以在这里提交。

机构管理：这里的机构管理也是指统计查询中的部分，不同角色的用户登录后它可以查看不同阶段下机构项目信息。本阶段下未提交的项目可以在这里提交。

受理管理：受理管理可以查询受理中未提交的直保/再担保项目及机构项目，受理管理人员可以删除或提交所受理的项目。

角色维护：这里的角色维护仅指业务中所涉及的角色，主要用于在不同的流程不同的阶段下，角色所拥有的权限。

项目文件设置：它是用来设置项目文件在哪里显示的，通过角色信息、流程信息及阶段信息来设置文件类型。项目文件只要和设置表中的信息匹配并且登录人有查看的权限，就能够显示。

5.3.2 系统数据库设计

数据库的设计既要在功能上满足业务需求又要在效率上满足实际的需要，在本系统中，涉及业务较多，除模块划分中所述五大模块外，还涉及到 jBPM 数据信息的存取，而它们均涉及到很多数据表，因此本文只列出业务部分中最基本的最简单部分的主要的数据关系表，其余表结构的关系这里不再给出，根据系统模块的设计及实际的业务需求，所设计的主要业务方面的数据表的结构如图 5-5 所示。

图 5-5 再担保业务系统主要业务部分数据表的关系图

在图 5-5 所示的数据表中，直保/再担保项目表和机构项目表是整个业务的核心部分，在整个业务流程中都要涉及到项目流程的流转及项目信息的查询，因此，只要是涉及到项目的部分都要用到这两个表，这里仅给出它们与企业接待和机构接待表结构的关联关系，至于后期管理等辅助性功能与它们的关联关系这里不再给出。企业接待信息表和机构接待信息表是整个业务流程的入口接待模块所使用的主表，它们关联了企业和担保机构的一些扩展信息表，这些扩展信息表均非业务主体，这里不再给出。项目文件表存储整个业务过程中与项目相关的文件的部分信息，它与项目文件设置表及文件基本信息表关联，文件基本信息表除存储项目文件的基本信息还存储合同文件等非项目文件的基本信息，所有类型的文件信息表都与文件基本信息表关联，它与其它类型文件表的关联这里不再给出。风险预警报告表是预警上报部分的核心，所有涉及到的预警上报信息都要使用它，除预警月报表与它关联外，还有担保业务台账表与再担保业务台账表与它关联，而担保业务台账表与项目基本信息表关联，再担保业务

台账表与机构项目信息表关联，它们的表结构及关联关系这里不再给出。

5.3.3 系统结构的设计

要使本文第四章实现的架构应用到本系统中，只要将本系统中抽象出来的业务对象分别应用到架构的表示层、业务层、持久层中即可。将架构的模型图进行扩展即为再担保业务系统的系统总体结构图。扩展后的架构模型图如图 5-6 所示。

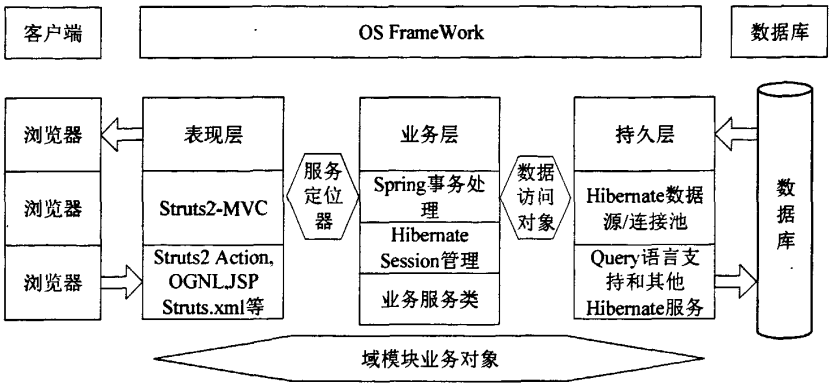

图 5-6 再担保业务系统的总体结构

在再担保业务的总体结构图 5-6 中，以风险预警模块中的上报流程功能模块为例，上报功能中存储的上报信息要写到持久层中，操作上报过程的业务要写在业务层中，已上报的信息要经过表示层中不同的 Action 传到不同的页面显示。从架构的工作流程上来说，部门人员在 JSP 页面填写上报的基本信息，当保存时将页面信息传入表示层的 Action 中，然后数据经 Action 处理后将数据传入业务层中，数据在经业务层处理后，调用持久层存储方法，最终将上报信息保存到数据库中。

5.4 系统的实现

本系统在实现过程中，各模块都使用本文架构实现，但具体业务的实现并不相同，而篇幅有限，不可能将实现一一列出。因此，这里还是以风险预警中的上报流程块为例，以展示本架构在再担保业务系统中的应用。

预警上报的主要功能是根据预警人发起预警的级别将预警信息发给本部门主任，如果本部门主任审核通过，就将预警信息发给本公司的风险部主任或总公司的风险部主任，如果本公司主任审核不通过，就将预警信息返给预警发起人。预警人可以不再发预警信息，也可以直接将预警信息发给总公司总经理。如果风险主任接到预警信息，会把预警信息发给本公司业务副总，如果本公司业务副总审核通过，会将预警信息发给本公司总经理。如果总公司业务副总审核通过，会将预警信息发给总公司的总经理。然后由本公司或总公司的总经理处理预警情况，处理后将预警信息发到总公司风险部

备案。

在这个流程中，除工作流中所走节点不同之外，在表现层中的 Action，业务层中总体业务实现及持久层中对数据库持久化的操作实现都相同。极大的体现了本架构中各层代码的重用性。

5.4.1 表现层的实现

表示层中页面均为添加信息页及查询列表页，这些页面大同小异，而且占用篇幅过长，这里就不给出它们的实现了。在表示层中仅给出查询列表部分 Action 实现的主要代码。该模块在 Action 中的主要实现代码如下。

```
public class RiskPreReportAction extends AbstractAction{
  private static Log log = LogFactory.getLog(RiskPreReportAction.class);
  private IRiskPreReportService riskPreReportService;
  private ArrayList riskPreReportList;
  public ArrayList getRiskPreReportList() {return riskPreReportList;}
  public void setRiskPreReportList(ArrayList riskPreReportList) {
    this.riskPreReportList = riskPreReportList;
  }
  public IRiskPreReportService getRiskPreReportService() {
    return riskPreReportService;
  }
  public void setRiskPreReportService(IRiskPreReportService riskPreReportService){
    this.riskPreReportService = riskPreReportService;
  }
  public String list() throws Exception{
    pager = riskPreReportService.getRiskPreReportList(currentPage,
        pagerMethod,this.getWhere());
    riskPreReportList = (ArrayList) pager.getList();
    this.setCurrentPage(String.valueOf(pager.getCurrentPage()));
    return SUCCESS;
  }
}
```

5.4.2 业务逻辑层的实现

业务逻辑层的实现继承其相应的业务接口。该模块在业务层中的主要实现代码如下。

```
public class RiskPreReportService implements IRiskPreReportService{
    private static Log log = LogFactory.getLog(RiskPreReportService.class);
    private IRiskPreReportDao riskPreReportDao;
    public void setRiskPreReportDao(IRiskPreReportDao riskPreReportDao) {
        this.riskPreReportDao = riskPreReportDao;
```

```
    }
    public List<RiskPreReport> getRiskPreReportList(){
        return riskPreReportDao.getRiskPreReports();
    }
}
```

5.4.3 持久层的实现

持久层的实现继承其相应的持久层接口。该模块在持久层中的主要实现代码如下。

```
public class RiskPreReportDao implements IRiskPreReportDao{
    public List getRiskPreReports(){
        String sqlQuery="From RiskPreReport";
        return getHibernateTemplate().find(sqlQuery);
    }
}
```

对应于 RiskPreReport 表，该模块模型的映射类由架构建立时自动生成，它就是一个简单的 JavaBean，这里不再给出它的代码。

到此，本模块中最基本的部分已经开发完成，还有一些特殊的逻辑只需在本开发基础上进行修改，就能够完成具体的功能实现。其它模块在开发中的基本实现部分与本模块类似，而具体业务部分还要根据业务的不同做具体的修改。

5.5 系统的运行效果

通过系统对风险预警上报流程的实现，该模块的预警上报部分显示效果如图 5-7 所示。

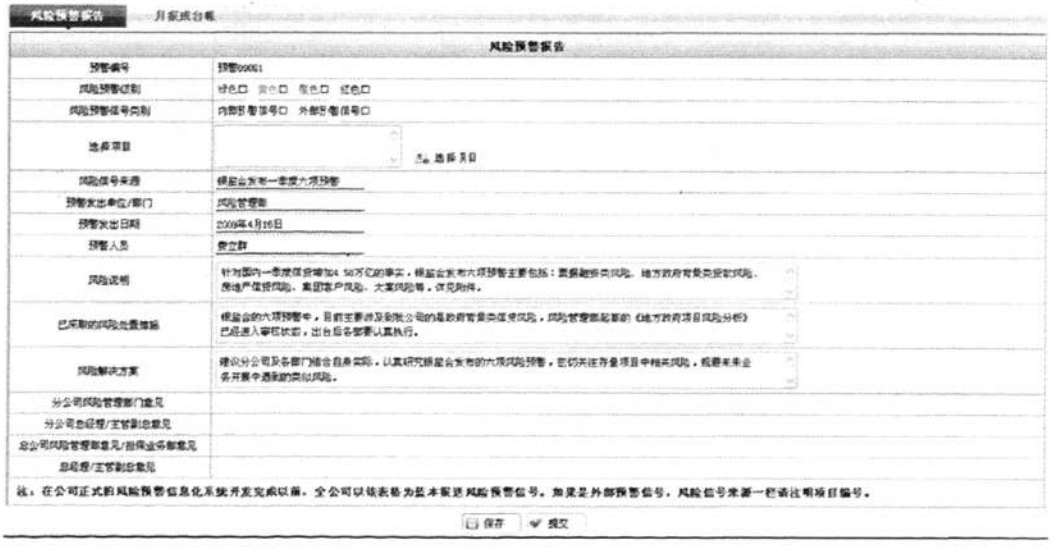

图 5-7 再担保业务系统预警上报效果图

个人预警信息维护部分的效果如图 5-8 所示。

图 5-8 再担保业务系统个人预警信息维护效果图

页面显示根据业务各不相同，其它的页面这里就不在显示了。

5.6 本章小结

在本章中，将本文架构应用到了再担保业务系统中，因此，从实践的角度验证了本架构的可用性。通过再担保业务系统的实现，可以看出本架构适用于企业级简单业务系统的开发。由于其各层次间基本实现的相似性，在实际开发中，完全可以通过 Ant 等工具来构建最基本部分代码的实现。这样，开发人员只需集中精力处理复杂业务，不必再编写基础代码的实现。

第六章 本文总结与工作展望

6.1 本文总结

本文通过对当前的一些开源的，优秀的框架的整合实现了一个企业级应用的轻量级J2EE 架构。本架构去除了传统 J2EE 架构中成本高、测试难、耗用资源大等缺点。

从软件设计角度看，本架构在实现中，总体采用 MVC 三层架构模式，因此设计简单。由于本架构每层相对独立，并且集成了一些开源组件（log4j，jBPM），所以它易于扩展，基于 J2EE 本身的特性，本架构适用于跨平台使用。本架构采用了轻量级 Spring 的 IoC 容器，因此便于测试。综上，本架构满足当今 J2EE 架构设计的规范。

从实际开发角度看，本架构可分为表示层，业务层和持久层。便于多人对不同的层同时开发，这样可以提高开发效率。由于本架构各层相对独立，因此，在开发中可以更换不同的开源框架作为不同层的组件。本架构中基本业务代码相似，在实际开发中，可以采用 Ant 等工具构建基础部分的代码的实现，以达到最大程度的代码复用。

从软件分层的角度看，本架构采用连接池机制，可以用少量的连接支持更多的用户，在一定程度上节省了资源，同时，可以将代码和数据库分在不同的服务器上使用，因此，具有良好的伸缩性。由于本架构采用分层机制，当需求发生变化时，只需修改代码的一部分，因此，具有良好的可维护性。本架构基础代码简洁，且各层代码独立，在业务交叉时，同一段代码能够满足多种需求。

6.2 工作展望

由于研究时间和本人理论知识的掌握有限，本文在实践应用和理论方面还有很多不足，需要进一步的研究完善，具体如下。

1.本架构在开发中并未采用开源框架的最新版本，因此，有些最新的技术并未应用其中，如 Spring3.x 版本的一些新特性等。

2.Spring 与 jBPM 的整合中，并未完全解决事务问题，当 Spring 整合 jBPM 做事务处理时，IoC 容器注入的 Bean 部分不好使，因此，本文中只介绍了 Spring 与 jBPM 整合中最基础的部分。

3.域模型仅使用了简单 JavaBean 和构造方法，并未做其它处理，在 Hibernate 持久化应用中并没有完全体现其优势。

4.本架构在实现中，只采用了数据字典，并没有采用静态类加载常量信息，因此，处理常量信息过于麻烦。

以上需要在将来的学习和工作中研究，进一步改进。

致　　谢

首先要由衷的感谢我的导师叶青副教授，感谢她给我的关心和论文写作上的指导。叶老师为人诚恳实在、治学严谨认真。每次对我的息心教导给我留下了深刻的印象，对我在学习生活中及工作中给予了极大的帮助。在这里，祝导师您身体健康，工作顺利。

感谢万易科技有限公司的董事长兼长春理工大学的教师王春才老师允许我用公司真实的项目作为论文的题材，以便于我能够写出真正的企业级应用。

感谢万易科技有限公司研发部的同志们，在我工作期间传授了很多保贵经验，使我能够把工作经验融入我的论文中。

感谢长春理工大学的底晓强老师，给我提供舒适安静的环境，使我能够加快论文写作的速度。

参考文献

[1]徐鹏.轻量级 J2EE 架构的研究与应用[D].电子科技大学,2007,5:1-3

[2]陈天河.Eclipse,Struts,Hibernate,Spring 集成开发宝典[M].电子工业出版社, 2008,10.

[3]李刚.轻量级 Java EE 企业应用实战:Struts 2+Spring+Hibernate 整合开发[M].电子工业出版社, 2008,11.

[4]李立华.轻量级 J2EE 开发的研究与应用[D].大连海事大学,2007,3:11-12.

[5]思志学.J2EE 整合详解与典型案例:一本书搞定 Struts+Spring+Hibernate[M].电子工业出版社,2008,1.

[6]叶健毅.精通 Java EE:Eclipse Struts2 Hibernate Spring 整合应用案例(第 2 版)[M].人民邮电出版社,2009,1.

[7] jackyli 经典的 J2EE 与轻量级的 J2ee.
http://muquanli123.blog.163.com/blog/static/771991200851107373 09/[EB/OL].2009,12

[8]阎宏.Java 与模式[M].电子工业出版社,2002,10.

[9]孙卫琴.精通 Struts:基于 MVC 的 Java Web 设计与开发[M].电子工业.2007,3.

[10]魏秋月.基于 Struts 框架的 Web 应用开发研究[J].西安邮电学院学报,2009,5

[11]Patrick Lightbody,Jason Carreira.WebWork in Action[M].Manning Publications,2005.

[12]明日科技.Struts 应用开发完全手册[M].人民邮电, 2007,09.

[13]DONALD BROWN.CHAD MICHAEL DAVIS.SCOTT STANLICK.Struts 2 in Action[M]. Manning Publications Co,2008.

[14]李绍平,彭志平.S2SH:一种 Web 应用框架及其实现[J].计算机技术与发展,2009,8.第 8 期，19 卷.

[15]李秉茂.基于 Struts2+Spring+Hibernate 轻量级框架的应用研究[J].山西煤炭管理干部学院学报,2009,2

[16]陈云芳.精通 Struts 2 基于 MVC 的 Java Web 应用开发实战[M].人民邮电出版社,2008,7.

[17]Ian Roughley.Practical Apache Struts2 Web 2.0 Projects.Apress[M], 2007.

[18]Ian Roughley,Starting Struts2[M].C4Media Inc, 2006.

[19]Budi Kurniawan,杨涛,王建桥,杨晓云.深入浅出 Struts2[M].人民邮电出版社,2009,5.

[20]Struts 2.x Draft Docs [EB/OL].http://struts.apache.org/2.x/index.html,2009,11.

[21]王春林,耿祥义.浅析 Struts2 框架[J].现代经济信息,2009,4.

[22] Rod Johnson,Juergen Hoeller. J2EE Development without EJB[M].Indian-apolis: Wiley Publishing Inc, 2004.

[23]CRAIG WALLS,RYAN BREIDENBACH.Spring in Action[M], Manning Publications Co,2005.

[24]沃尔斯(Craig Walls),布雷登巴赫(Ryan Bredenbach),毕庆红.Spring in Action(第 2 版)中文版[M].人民邮电出版社.2008,10.

[25]罗时飞,精通 Spring:深入 JavaEE 开发核心技术[M].电子工业出版社,2008,10

[26]明日科技.Spring 应用开发完全手册[M].人民邮电, 2007,9.

[27]Johnson R,Hoeller J.Spring-Java/J2EE Application Framework Reference Documentation [EB/OL]. http://www.springsource.org/documentation.2009,12.

[28]廖雪峰,Spring 2.0 核心技术与最佳实践[M].电子工业,2007,06.

[29]Rob Harrop,Jan Machacek. Pro Spring [M].Apress;1 edition,2005.

[30]马哈切克(Manchacek.J.),马连浩等.Spring 高级程序设计[M],人民邮电出版社,2009,9.

[31]Johnson R. Introduction to the Spring Framework[EB/OL].http://www.theserverside. com/articles, 2005,5.

[32]王国辉,马文强.Hibernate 应用开发完全手册[M].人民邮电, 2007,9.

[33]陶勇,李晓军.Hibernate ORM 最佳实践[M],清华大学出版社,2007,9.

[34]Emmanuel Bernard.Hibernate Core for Java [EB/OL].http: //www.hibernate.org,2009,11.

[35]Christian Bauer,Gavin King.Java Persistence with Hibernate[M].Manning Publications Co.,2007.

[36]蔡雪熬.Java 开发利器:Hibernate 开发及整合应用大全[M].清华大学出版社.2006,3.

[37]RedHat.HibernateReferenceDocumentation[S/OL].http://www.hibernate.org/hib_docs/v3/refer-ence/en-US/html,2007.

[38]Christian Bauer,Gavin King,杨春花,彭永康.Hibernate 实战(第 2 版).人民邮电出版社, 2008,4.
[39]ChristianBauer,GavinKing.Hibernatein Action.Alneriea[M].Manningpublications Co,2004.
[40]孙卫琴.精通 Hibemate:Java 对象持久化技术详解[M].北京:电子工业出版社,2005,5.
[41]梁建全,周力,孟志勇,田利军.轻量级 JavaEE 框架整合方案[M].人民邮电出版社,2008,8.
[42]高杰.深入浅出 jBPM[M].人民邮电出版社,2009,7.
[43]梅特斯克等.Java 设计模式[M].人民邮电,2007,3.