

大连海事大学

---

硕士学位论文

---

JavaEE多层架构Struts2+Spring3+Hibernate3+Ajax的整合

---

姓名：王向兵

---

申请学位级别：硕士

---

专业：管理科学与工程

---

指导教师：张升文

---

20090601

---

## 摘 要

随着软件开发技术的发展,可复用、易扩展的而且经过良好测试的软件组件,越来越为开发者所青睐。其中最受人们关注的是 Struts、Spring 和 Hibernate 框架。随着 WEB2.0 时代的到来, Ajax 技术带给了用户更高的客户体验, Ajax 框架受人瞩目成为必然。设计一个基于 MVC 模式的 SSH+Ajax 框架非常具有现实意义。

MVC 模式分离了数据访问和数据表现给系统提供了更好的解耦,在实现多层 Web 应用系统中具有明显的优势。Struts2 是一个基于 MVC 模式并且成熟的实现了控制器层和 Web 表现层的集大成者的框架,它不但提供了灵活自然的控制器分配方式,而且提供了强大的标签表示技术,并且为其它框架的整合预留了方便的接口。Spring 主要基于 IoC 和 AOP,很容易实现 Bean 的装配和事务管理等特性;同时它对不同的数据访问技术提供了统一的接口。Hibernate 框架是一个面向 Java 环境的对象/关系数据库映射工具。它不仅可以管理 Java 类到数据库表的映射,还提供数据查询和获取数据的方法,可以大幅度减少开发时开发者使用 SQL 和 JDBC 处理数据的时间。Ajax 框架能够使应用开发人员更好的解决企业级应用的灵活开发,增加开发的满意度,解决在 Struts 框架或者 Spring 框架中的不足,从而更完美地完成软件开发任务。

本论文在深入研究各个框架系统理论和设计模式的基础上,主要针对当前开发模式中表示层与业务逻辑层、业务逻辑层与数据持久层之间不能完全分离,设计了基于 MVC 模式的一套框架,该框架以 Struts2, Spring3, Hibernate3 为主, Ajax 为辅,引入了 Spring 的 IoC 技术和 Java5 的新特性 Annotation。通过 IoC 技术的引用,降低了系统模块之间的依赖性。通过在 POJO 类使用 Annotation 技术,大大降低了系统的代码量以及提高了系统的开发效率,并且通过 Struts2 把 AOP 思想引入到企业级应用的开发中,分离了业务逻辑代码和基础业务代码(交叉业务代码),提高了系统代码的可复用性,可维护性和可读性,解决了代码的分散混乱的问题。最后以大连鑫轮模具公司实际的 ERP 项目为例,对整合的架构应用进行具体的实现。开发结果表明,整个系统具有平台无关性,并提高了应用系统的灵活性,可维护性,可扩展性,可移植性和组件的可复用性。

**关键词:** Struts2; Spring; Hibernate; Ajax; Annotation; IoC; AOP

---

## Abstract

With the development of software technology, reusable, scalable and easy to well-tested software components, are more and more popular for developers. In Which the most popular are Struts, Spring and Hibernate frameworks. However, with the coming of WEB 2.0 era, Ajax technologies bring our enterprise's developers a higher customer satisfaction. It is inevitable that Ajax framework is attention, therefore the design of SSH + Ajax framework based on MVC model is a very practical significance.

MVC pattern, which separated the data access layer and data presentation layer, provides a better decoupling and a distinct advantage in the realization of multi-layered Web applications. Struts2 is an excellent realization based on the controller layer and the presentation layer of MVC pattern. And it not only provides a flexible and natural way of the controller distribution, and also provides a powerful label presentation technology, and sets aside interfaces for other frameworks in order to integrate facilitate. Spring is another excellent framework mainly based on the thinking of IoC and AOP. It is easy to achieve the assembly of Java Bean and transaction management, and at the same time it provides a unified interface for different data access technology. Hibernate framework is an ORM (Object/Relational Mapping) tool. It not only can manage the Java objects mapped to database tables, also provides methods of data query and access to data, which can significantly reduce the time of using SQL and JDBC technology to access to data manually. Ajax framework provides us a better and more flexible developing in the enterprise web applications, and it also can increase the development of satisfaction and work out problems in the Struts framework or Spring framework in order to complete software development.

This paper, based on in-depth study of the various frameworks' system theory and design patterns, aiming at that presentation layer and Business Logic layer, Business Logic layer and Data layers can not be completely separated, will design a framework based on MVC, which mainly in Struts2, Spring3, Hibernate3 framework, Ajax framework as a supplement, and introduced Spring's IoC technology and Annotation technology in java 5's new features. It reduces the dependency between modules

---

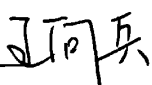
through the use of IoC thinking, and greatly reduces amounts of application codes, improve the efficiency of the application through the use of Annotation technology in POJO. It will bring the thinking of AOP in struts2 to the development of enterprise web application in order to the separation of business logic code and the basic operational code (cross-business code), improve the system code reusability, maintainability and readability, Finally we will use this integrated framework to complete an actual ERP project on Dalian XinLun enterprise. The result shows the system does not depend on platform and has the characteristics of high maintainability, expansibility and reusability.

**Key Words: Struts2; Spring; Hibernate; Ajax; Annotation; IoC; AOP**

# 大连海事大学学位论文原创性声明和使用授权说明

## 原创性声明

本人郑重声明：本论文是在导师的指导下，独立进行研究工作所取得的成果，撰写成硕士学位论文 “JavaEE 多层架构 Struts2+Spring3+Hibernate3+Ajax 的整合”。除论文中已经注明引用的内容外，对论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。本论文中不包含任何未加明确注明的其它个人或集体已经公开发表或未公开发表的成果。本声明的法律责任由本人承担。



学位论文作者签名：王向兵 

## 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大连海事大学有关保留、使用研究生学位论文的规定，即：大连海事大学有权保留并向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。本人授权大连海事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。同意将本学位论文收录到《中国优秀博硕士学位论文全文数据库》（中国学术期刊（光盘版）电子杂志社）、《中国学位论文全文数据库》（中国科学技术信息研究所）等数据库中，并以电子出版物形式出版发行和提供信息服务。保密的论文在解密后遵守此规定。

本学位论文属于： 保 密 ☐ 在\_\_\_\_\_年解密后适用本授权书。

不保密 ☐ （请在以上方框内打“√”）

论文作者签名：王向兵  导师签名：张静文   
日期：2019年6月9日

## 第 1 章 绪论

### 1.1 课题背景及意义

JavaEE 多层架构是目前流行的 Web 应用开发架构之一，已成为解决电子商务和企业级应用的标准平台。但这个平台并不能满足所有需求特点，其中的业务逻辑层的解决方案 EJB 机制存在过度设计的缺陷，而且 EJB 的机制是一种针对代码和设计强侵入性的规范，使业务组件移植不是很方便。EJB 制订了很多的接口和编码规范，要求实现者遵守，这样就导致了业务逻辑移植困难，成本较高。伴随着软件开发技术的发展，在多层的软件开发项目中，可复用、易扩展的而且是经过良好测试的软件组件，越来越为开发者所青睐。这意味着开发人员可以将充裕的时间用来分析、构建业务逻辑的应用，而非繁杂的代码工程。于是开发人员将相同类型问题的解决途径进行抽象，抽取成一个个应用框架。用这些框架构建的分布式应用程序完美地实现了应用程序高内聚、低耦合、高弹性、易维护的优点。其中最受人们关注的是 Struts、Spring 和 Hibernate 框架。但是随着 WEB2.0 时代的到来，Ajax 技术带给了用户更高的客户体验，Ajax 框架受人瞩目成为必然。因此设计一个基于 MVC 模式的 SSH+Ajax 框架非常具有现实意义。

Struts 是 MVC 模式的实现框架，它由一系列框架类、辅助类和定制的 JSP 标记库构成，为 Model-2 Web 应用开发提供一个强大的、通用的 Web 应用框架，从整体上减轻构造企业 Web 应用的负担。Spring 是一个服务于所有层面的应用框架，允许开发者根据需要使用它的部分模块。Spring 主要基于 IoC(Inversion of Control, 控制反转)和 AOP(Aspect Oriented Programming, 面向方面编程)，很容易实现 Bean 的装配和 Transaction Management(事务管理)等特性；同时它对不同的数据库访问技术提供了统一的接口。Hibernate 框架是一个面向 Java 环境的对象/关系数据库映射工具。它不仅可以管理 Java 类到数据库表的映射，还提供数据查询和获取数据的方法，可以大幅度减少开发时人工使用 SQL 和 JDBC 处理数据的时间。它可以在应用 EJB 的 JavaEE 架构中取代 CMP(Container-managed persistence)，完成数据持久化的重任。Ajax 是一种客户端方法，可以与 JavaEE，.NET 等脚本语

言进行交互，是一种真正的能够实现与服务器无关的技术，使用 Ajax 框架能够更好的解决 web 应用的灵活开发，增强客户使用体验，增加客户开发满意度，解决在 Struts 框架或者 Spring 框架中的不足，从而更好地完成软件开发任务。

随着系统规模的日益复杂，为了快速提高开发效率，将一些优秀的框架进行整合以适应不同的应用需求已成为目前系统开发中流行的方向之一。

框架是一种针对特定领域，基于架构的、解决某类应用问题的半成品，是大粒度的复用，它为 Web 应用开发提供了一个能够使用的架构模板和软件包，让开发从编码中解脱出来，将注意力主要集中在业务逻辑分析上，从而减轻了开发者处理复杂问题的负担，提高了工作效率；其次开发者在分析框架的基础上，可以对其进行改进和扩展，以适应实际应用的需求。因此也为扩展和维护系统奠定了良好的基础。

本文研究的目的是提出一个新型的框架体系，用以克服 JavaEE 架构的 Web 多层应用解决方案中存在的一些缺点，使其能够最大限度地满足符合企业级应用软件系统需求特点的开发方法。研究的意义在于：通过 JavaEE 多层架构中 Web 表现层、业务逻辑层和数据持久层的四个优秀的框架，充分发挥每个框架的优势，以实现：

- (1) 复用设计，以简化开发的复杂性；
- (2) 复用代码，减少编码和测试时间，提高工作效率；
- (3) 提高系统的可扩展性、可维护性和可移植性；
- (4) 降低层间的耦合度；
- (5) 提高系统开发的灵活性，增加客户体验，提高客户满意度。

这种整合框架具有很强的生命力。但应该同时看到，虽然这种整合框架的思想很好，但是发展的时间较短，还不是很成熟，与 JavaEE 架构相比，还有不足之处，所以应该与 JavaEE 架构的优势结合起来才能产生更好的效果。

### 1.2 国内外研究现状

当前，在国内外存在多种实现 Web 应用系统的技术途径，其中最具有代表性、使用最广泛的两大类分别是 Microsoft 公司提出的 .NET 平台和 SUN, IBM 等公司

提出的 JavaEE 平台。

JavaEE 是一个基于组件的容器模型的系统平台，其核心概念是容器。容器是指特定组件提供服务的一个标准化的运行时环境，Java 虚拟机就是一个典型的容器。组件是一个可以部署的程序单元，它以某种方式运行在容器中，容器封装了 JavaEE 底层的 API，为组件提供事务处理、数据访问、安全性、持久性等服务。在 JavaEE 中，组件和组件之间并不直接访问，而是通过容器提供的协议和方法来相互调用。组件和容器间的关系通过“协议”来定义。容器的底层是 JavaEE 服务器，它为容器提供 JavaEE 中定义的各种服务和 API。

由于 JavaEE 的开放性，有很多公司和开发者汇聚在一起推动着 JavaEE 技术的发展，开源产品众多，源代码的普遍开放，使得 JavaEE 技术得到了迅速的发展，出现了各种各样的开发框架。首先是 SUN 公司推出的 EJB 规范，并在其基础上推出的 JavaEE 多层架构，然后是 Struts、WebWork、Tapestry、Spring、JSF 框架，还有数据库访问的 Hibernate、iBatis、TopLink 等框架。基本上针对不同的应用，不同的层次都有很多相应的框架。这些框架整合了 JavaEE 所提供的基础服务组件，基于设计模式和架构模式，提出了最佳开发实践路径。应用这些框架去满足企业级应用开发的时候，也出现了一个问题，那就是如何来合理高效地进一步整合这些具有特定功能的开发框架，从而实现出一个完整高效的基于 JavaEE 面向 Web 的企业级应用开发框架。因此，合理地选择不同的开发框架并将其整合为一个高效的企业级应用框架对于开发基于 JavaEE 的 Web 应用显得尤为迫切和必要。

### 1.3 主要研究内容

本文主要研究基于 JavaEE 平台的 WEB 应用框架的设计和实现，并设计了以 Struts2, Hibernate3, Spring3 以及用来增加系统灵活开发和增加客户体验的 Ajax 框架的多层整合架构体系。利用该架构实现了大连鑫轮模具公司的 ERP 系统，其中包括的主要模块有：生产管理子系统、销售管理子系统、成本管理子系统、报表分析管理子系统、系统维护子系统和用户权限子系统。

本文主要研究内容：



本论文在深入研究各个框架系统理论和设计模式的基础上,主要针对当前开发模式中 web 表现层与业务逻辑层、业务逻辑层与数据持久层之间不能完全分离,使得系统维护成本增高等问题,设计了基于 MVC 模式和 JavaEE 多层架构的一个框架体系,该体系以 Struts2, Spring3, Hibernate3 为主, Ajax 为辅,依托 JavaEE 分层体系,引入了 Spring 的 IoC 技术和 java5 的新特性 Annotation。通过 IoC 技术的引用,降低了系统模块之间的依赖性。通过在 POJO 类使用 Annotation 技术,大大降低了系统的代码量以及提高了系统的开发效率,并且通过 Struts2 把 AOP 思想引入到 ERP 软件系统的开发中,提高了代码的可读性,解决了代码的分散且混乱的问题。

### 1.4 论文的章节结构

本文共分五章。论文的章节结构如下:

第1章:绪论。主要对基于 JavaEE 的 Web 应用的背景,发展和现状进行了分析。

第2章:首先介绍企业及应用架构的发展状况,然后研究基于 MVC 模式而建立起来的 Struts2、Spring3、Hibernate3 框架的原理及其新特性,通过 Ajax 技术及其实现框架的介绍,来增加客户的体验和提高系统的开发效率,并且进一步分析它们的关键技术。

第3章:研究基于 Struts2、Spring3、Hibernate3 和 Ajax 框架的架构设计,通过综合分析 Struts2、Spring3、Hibernate3 与 Ajax 各自的优势和缺陷,提出将这四个框架有效整合后的架构,并对架构的工作流程和优点作了阐述。

第4章:介绍基于 Struts2, Spring3, Hibernate3 以及 Ajax(ExtJs)整合框架的大连鑫轮模具公司的 ERP 系统的设计。首先介绍该公司 ERP 系统所涉及的业务过程,然后对当前 ERP 管理应用系统进行需求分析和详细设计,最后使用本文设计的多层整合框架来实现 ERP 系统。

第5章:总结与展望。概括总结了作者在应用该架构进行项目实施中的体会以及不足,并提出今后需要进一步研究和发展的方向。

## 第 2 章 企业级应用架构发展与 WEB 应用框架

### 2.1 企业级应用架构的发展

随着计算机的快速发展，在企业中的各种应用和计算机软件之间的桥梁开始跨越了企业的整个业务，把这种企业级应用与当前的计算机技术更加完美的融合，成为企业级应用架构发展的一种必然。

随着 Internet 技术的兴起，B/S(浏览器/服务器模式)结构在企业应用中得到了大规模应用。B/S 结构是对 C/S 结构的一种改进。在这种结构下，软件应用的业务逻辑完全在应用服务器端实现，用户表现完全在 Web 服务器端实现，客户端只需要浏览器即可进行业务处理，是一种全新的软件系统构造技术。这种结构是当今应用软件的首选体系结构。

B/S 架构的企业级应用迅速得到了推广，其原因就在于相对于 C/S 架构方式具有得天独厚的优势，其数据安全性大大提高，并且克服了 C/S 架构软件中的数据不一致性以及数据不实时性，企业业务需要更新时，只需更新服务器端程序就能够达到企业应用的安全升级。B/S 架构把企业级应用软件发布于网络中，使我们随时实时得到企业信息<sup>[1]</sup>。但是由于早期开发的 B/S 架构的企业级应用使用技术混乱，把所有的业务逻辑、数据持久化、控制逻辑混在一起，这些处理逻辑都通过页面的脚本实现，因此导致早期的 B/S 结构应用面临着后期维护困难、难以扩充的问题<sup>[2]</sup>。

MVC 设计模式重新定义了 B/S 结构应用的开发模式。MVC 模式规定 B/S 结构应用应该分成 3 个部分：Model(模型 M)、View(视图 V)和 Controller(控制器 C)<sup>[3]</sup>。MVC 模式分离了数据访问和数据表现，给系统提供了更好的解耦。MVC 架构的核心思想是：将程序分成相对独立而又能协同工作的 3 个部分。通过使用 MVC 架构，可以降低模块之间的耦合，提供应用的可扩展性。MVC 的每个组件只关心组件内的逻辑，不应与其它组件的逻辑混合。

JavaEE 的出现则更加规范了 B/S 架构应用的开发，JavaEE 是一种利用 Java

平台来简化企业解决方案的开发、部署和管理相关的复杂问题的体系结构。典型的 JavaEE 规范定义了四个层次，包括客户层、web 层、企业组件层、企业资源层<sup>[4]</sup>。下面分别介绍这四个层的作用：

客户层用户界面的开发和简单的业务逻辑都可以在该层得到实现，主要用来处理客户请求，调用相应的逻辑模块，并把结果以动态网页的形式返回到客户端。JavaEE 支持多种客户端，既可以通过 Internet 访问的 Web 浏览器客户端，也可以通过企业 Intranet 运行客户端。

Web 层是基于 Web 的应用程序服务的，它是由 JavaEE 中的 JSP 和 Java Servlet 等技术来实现。它可以访问封装所有的商业逻辑的组件，即企业组件层，并负责响应来自 Web 客户端的请求。

企业组件层处理应用的核心是商务逻辑。企业组件层为底层服务组件提供必要的接口。企业组件通常被实现为 EJB 容器内的 EJB 组件。其中，EJB 容器提供组件生命周期、管理持久性、事物和资源分配等。

EIS 层即企业信息系统(Enterprise Information System)层，包括企业已有系统、数据库系统、文件系统等。JavaEE 提供了多种技术来访问这些系统，如利用 JCA 或者 JMS 技术来访问其它已有系统(例如：CRM/ERP)。

JavaEE 多层架构的缩略图如图 2.1 所示。

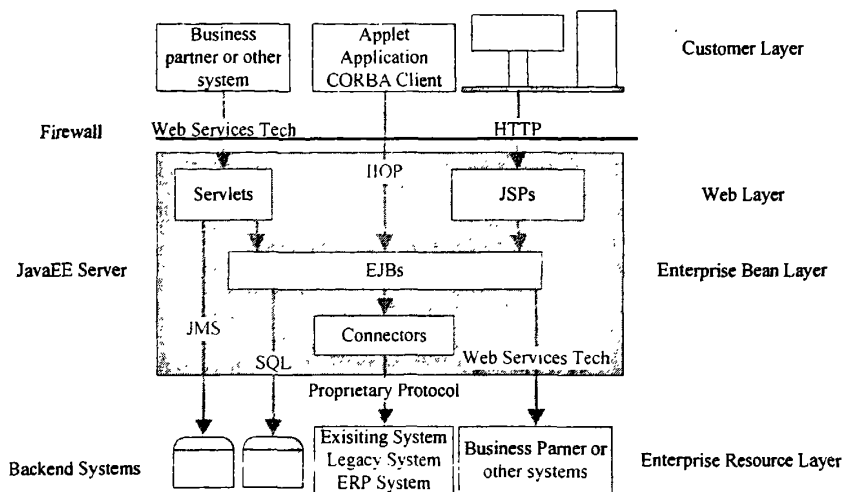


图 2.1 JavaEE 多层架构

Fig.2.1 JavaEE multi-layer architecture

目前, Ajax 技术的出现再次完善了传统的 web 应用。Ajax 应用强调异步发送用户请求, 用户可以在浏览页面的同时, 发送请求。在第一个请求的服务器响应还没有完全结束时, 浏览器可以再次发送请求。这种请求的发送方式非常类似于传统的 C/S 结构应用<sup>[5]</sup>。

## 2.2 MVC 设计模式研究

软件设计中的一个核心问题是能否使用重复的体系架构, 即能否达到体系架构级的软件复用。许多学者们开始研究和实践软件体系架构的模式问题, 设计模式的理论也就相应的被提了出来。一个设计模式描述了对于特定设计问题被验证成功的解决方案, 它综合了所有开发者对于这个问题已有的知识和见解, 同时它也是对于常见问题的可重用方案。设计模式一般适用于单个问题, 但是把多种设计模式组织在一起就可以提供整个企业系统的解决方案。MVC 设计模式在使用 B/S 结构开发模式的企业级应用中, 已经成为毋庸置疑的首选方案。

MVC 英文即 Model-View-Controller, 它是目前非常流行的一种软件设计模式。

MVC 的设计思想是将应用的输入、处理和输出流程进行强制的分离。这样一个应用被分成：**Model**（模型）、**View**（视图）、**Controller**（控制）三个层次，这三个层次以最少的耦合协同工作，从而提高应用的可扩展性及可维护性。**MVC** 模式的整体效果可如图 2.2 所示。

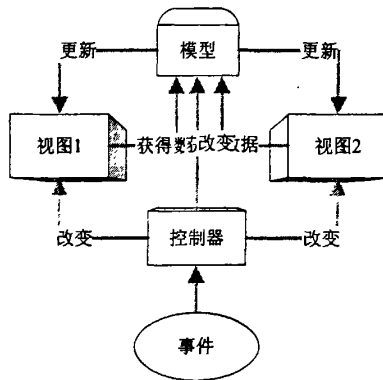


图 2.2 MVC 结构

Fig.2.2 The Structure of MVC

**视图(View):** 代表用户交互界面，典型的视图技术包括：jsp 技术，html 静态页面，包含有 Ajax 框架的动态页面以及一些报表技术页面等等，视图技术主要负责数据的输入和输出。

**模型(Model):** 就是业务流程和业务状态的处理以及业务规则的制定。业务流程的处理过程对其它层来说是黑箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 的核心。业务模型还有一个很重要的模型那就是数据模型。数据模型主要指实体对象的数据保存（持久化）。比如将一条数据保存到数据库。可以将这个模型单独列出，所有有关数据库的操作只限制在该模型中。

**控制(Controller):** 通过控制层可以起到控制整个业务流程并且实现 Model 和 View 的协同工作。控制层并不做任何的数据处理。因此，一个模型可能对应多个

视图，一个视图可能对应多个模型<sup>[6-8]</sup>。

需要说明的是在 B/S 架构中控制层通过使用前端控制器（Front Controller）模式来实现，这个模式包含了一个分发器（在 Java 的 Web MVC 实现中，通过 Servlet 或者 Filter 来实现分发器），而分发器将请求 URL 映射至需要被执行的命令实例（Command Instance Or Action），命令实例在 WebWork 或者 Struts 中就是 Action。Action 与系统后端的服务进行交互，通常这些服务会组合在一起作为模型。命令实例在处理完业务逻辑之后返回一个码值，而这个返回码会映射到某一个视图（通常是一个 Web 页面模板，譬如 JSP）。最后，结合控制器和模型，视图将会呈现给用户。通常视图会使用标签库，以便更简单地访问数值<sup>[9]</sup>。

目前常用的 MVC 框架，除了 Struts1，WebWork，还有一些非常流行的 MVC 框架，这些框架都提供非常好的层次分隔能力，并且在实现良好 MVC 分割的基础上，还提供一些辅助类库来帮助应用的开发。例如：JSF，Tapestry，Spring MVC，Struts2，其中本文的着重研究的框架是集大成者 Struts2。

## 2.3 Struts2 框架

### 2.3.1 Struts2 简介

Struts2 以 WebWork 优秀的设计思想为核心，吸收了 Struts1 的部分优点，建立了一个兼容 WebWork 和 Struts1 的 MVC 框架，基于 Struts1 和 WebWork 框架的成熟性，Struts2 不仅保留了 Struts1 的简单易用性，并且充分利用了 WebWork 的拦截器机制(其实就是 AOP 思想)，将 Struts2 发展成一个具有高度可扩展性的框架。基于这种背景，Struts2 将会在短时间内迅速成为 MVC 领域最流行的框架<sup>[10]</sup>。

Struts2 的体系与 Struts1 体系的差别非常大，因为 Struts2 使用了 WebWork 的设计核心，大量使用拦截器来处理用户请求，从而允许用户的业务逻辑控制器与 Servlet API 分离。

Struts2，类同与 WebWork，同样使用了拦截器作为处理，以用户的业务逻辑控制器为目标，创建一个控制器代理。控制器代理负责处理用户请求，处理用户

请求时回调业务控制器的 `execute` 方法, 该方法的返回值将决定 Struts2 将怎样的视图资源呈现给用户。Struts2 的体系概括<sup>[11]</sup>如图 2.3 所示。

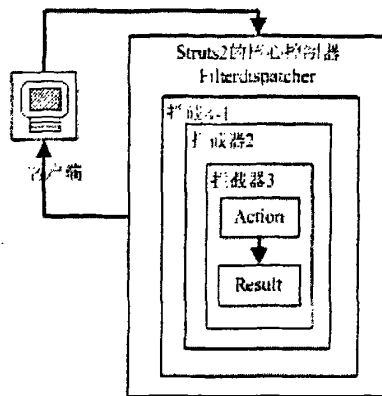


图 2.3 Struts2 体系概括

Fig.2.3 The general Graphic of Strus2

Struts2 框架的大致处理流程与 WebWork 类似, 首先从客户端 (浏览器) 发出一个请求, 请求经过一系列过滤器 (ActionContextCleanUp, SiteMesh 等), 由 struts2 的核心控制器 `FilterDispatcher` 根据请求来调用合适的 `Action`, Struts2 的拦截器栈自动对请求应用通用功能 (例如 workflow, validation 或者文件上传等功能), 然后回调 `Action` 的 `execute()` 方法, 该 `execute()` 方法先获取用户请求参数, 然后执行某种数据库操作, 既可以是将数据保存到数据库, 也可以从数据库中检索信息。实际上, 因为 `Action` 只是一个控制器, 它会调用业务逻辑组件来处理用户请求 `Action` 的 `execute()` 方法, 然后处理结果信息将被输出到浏览器中, 可以是多种视图技术中的任何一种 (JSP, PDF, Velocity, FreeMarker 等)<sup>[10]</sup>。

### 2.3.2 Struts2 核心组成

对于一个 MVC 框架而言, 重点就是实现两个部分: 控制器部分和视图部分<sup>[10]</sup>。同样 Struts2 把重点放在了这两个部分: 控制器部分是由 `Action` (以及隐藏的一系列拦截器) 来提供支持, 而视图部分则通过大量标签来提供支持。

大致上, Struts2 框架由三个部分组成: 核心控制器 `FilterDispatcher`、业务控制器 `Action`(也称 web 层业务逻辑控制器)和用户实现的业务逻辑组件。Struts2 本身已经提供了核心控制器部分,该核心控制器作为一个 `Filter` 负责过滤用户的应用请求,如果用户请求以 `action`(默认状态下)结尾,该请求将被转入 Struts2 框架处理。而业务控制器组件就是用户实现 `Action` 类的实例, `Action` 类中一般包括一个 `execute()` 方法,该方法返回一个字符串,每个字符串对应一个视图名。而用户实现的业务逻辑组件,即 JavaEE 应用里的模型组件,已经超出了 MVC 框架的覆盖范围,可能还包含了 DAO、领域对象组件。通常, MVC 框架里的业务控制器会调用模型组件的方法来处理用户请求。先来具体研究一下 Struts2 的 `Action` 机制。

### 1. Action

一般一个 `Action` 代表用户的一次请求或调用<sup>[9]</sup>。相对于 Struts1 来说, Struts2 采用了低侵入式设计, Struts2 的 `Action` 可以完全是一个 POJO 类,从而有很好的代码复用性。为了让用户开发的 `Action` 类更加规范, Struts2 的 `Action` 需要实现 `Action` 接口,或者直接继承基础类 `ActionSupport`。这是因为 `Action` 接口提供了开发应用中非常常见的静态字符串(对应一个视图资源),并且还提供了每个 `Action` 必须实现的 `execute` 方法,而基础类 `ActionSupport` 更加提供了具有丰富功能的底层方法,例如: 获取国际化信息方法、数据校验方法、默认的用户请求处理方法等,这样就会大大简化 `Action` 的开发。

Struts2 同样支持 WebWork 中的 `ModelDriven`, 所谓模型驱动,就是使用单独的 `JavaBean` 实例来贯穿整个 MVC 流程; 与之相对应的属性驱动模式,则使用属性作为贯穿整个 MVC 流程的信息携带者。当然属性无法独立存在,必然依附于一个对象,这个对象就是 `Action` 实例。简单来讲,模型驱动使用单独的 VO 来封装请求参数和处理结果,属性驱动则使用 `Action` 实例来封装请求参数和处理结果。当然还需要为这些请求参数实现它们的 `set` 和 `get` 方法。在实际开发中非常实用的是属性驱动模式。

当 `Action` 处理完用户请求后,将返回一个普通字符串,整个普通字符串就是



一个逻辑视图名。Struts2 通过配置逻辑视图名和物理视图之间映射关系，一旦系统收到 Action 返回的某个逻辑视图名，系统就会把对应的物理视图呈献给浏览者。相对于 WebWork 框架而言，Struts2 可以使用表达式语法提供 Action 的动态配置视图资源，这样不仅大大减少了配置资源文件(struts.xml)的长度，而且可以大大规范 Action 的统一配置，进而加速了开发的进度。例如：

```
<package name="/individual" extends="struts-default">
    <action name="*/*" class="{1}Action" method="{2}" >
        <result name="list">/{1}/{2}/{1}list.jsp</result>
        <result name="view">/{1}/{2}/{1}prop.jsp </result>
        .....
    </action>
    .....
</package>
```

当用户请求 Action 的 URL 为 “/individual/recruit/recruitDetail.action”，那么 Struts2 框架首先会寻找 name 为 “/individual” 的 package，然后在这个包下面寻找相应的 Action 的 name，由于 package 包下 Action 的 name 为 “\*/\*”，所以左边的\*就成为第一个动态的参数表达式，在以后的表达式中调用{1}就是指 “recruit”，而右边的\*就是第二个参数，其表达式为{2}，于是这个 Action 请求的 Action 名字为 “recruitAction”，而调用的方法名字为 “recruitDetail”，当这个方法返回视图资源名为“view”时，框架就会自动在路径 WebContent/recruit/recruitDetail/recruitlist.jsp 获得相对应的物理视图资源名。这样在一个 Action 中无论有多少方法，都可以应用一个动态配置映射到更加复杂的物理视图资源。

## 2. Interceptors

在软件开发领域有一项 DRY(Don't Repeat Yourself)规则，这就是拦截器的由来<sup>[10]</sup>。在软件开发中，经常会遇到需要把许多相同的并非业务代码混合于业务逻辑代码中，例如事务处理，安全管理，日志及审核等相关代码，为了集中管理这些代码，并且消除重复代码，软件领域兴起了 AOP(Aspect Orient Program，面向切

面编程),即将基础代码(交叉业务逻辑代码)封装成切面透明的植入到具体的业务逻辑代码中,这样就使得业务逻辑代码非常清晰,而且需要修改基础代码时,只需要修改切面即可<sup>[12]</sup>。

拦截器与 AOP 就是密切相关的,且事实上 Struts2 的拦截器机制来自于 WebWork,这种拦截器机制就是基于 AOP 思想。当业务控制器 Action 提出基础业务代码的需要时,Struts2 可以通过在配置文件中指定拦截器,从而可以让拦截器方法在目标方法执行之前或者执行之后自动执行,从而完成基础业务代码的动态插入,降低了 Action 与基础代码的耦合,而且可以抽象出大量的基础代码构造出许多公用的拦截器,例如:类型转换拦截器,数据校验处理拦截器,文件上传拦截器,阻止表单多次提交的拦截器等,然后在 Action 的配置文件中动态指定相应的拦截器<sup>[13]</sup>。拦截器与 Action 之间的关系如图 2.4 所示。

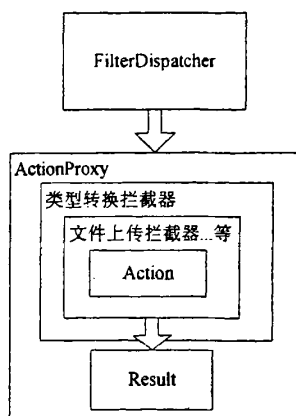


图 2.4 Struts2 拦截器与 Action 的关系

Fig.2.4 The Relationship of Struts2 Interceptors and Action

Struts2 允许以一种可插拔的方式来管理 Action 需要完成的通用操作,将这些拦截器定义成拦截器方法,然后在 struts.xml 文件中配置 Action 时引入该 Action 即可。当然在一个 Action 中可以配置多个拦截器,而 Struts2 甚至可以把这些拦截器组织成一个拦截器栈来应用于 Action。Struts2 提供了许多公用的拦截器,例如:

params 拦截器将 HTTP 请求中的参数解析出来, 设置为 Action 的属性; fileUpload 拦截器则负责解析请求参数中的文件域; 同样 Struts2 也允许来构造自己的业务拦截器, 这些只需要该拦截器 (其实本质还是一个 java 类) 实现 com.opensymphony.xwork2.interceptor.Interceptor 接口即可<sup>[10]</sup>。

### 3. Struts2 的标签库

Struts2 内建的标签库极大的简化了 JSP 页面输出的业务逻辑实现, 并且通过内建的 OGNL 表达式, 大大加强了 Struts2 的数据访问功能。

Struts2 标签库的标签并不依赖于任何表现层技术, 可以在常用的表现层技术, 例如: JSP, Velocity 和 FreeMarker 等模板技术中使用。并且默认支持 OGNL、JSTL、Groovy 和 Velocity 表达式, Struts2 的标签大致可以分为三类: 用户界面标签库(表单标签和非表单标签)、非用户界面标签库(控制标签库和数据访问标签库)和 Ajax 支持标签库。

与 Struts1 标签相比, Struts2 标签使用 OGNL 表达式作基础。因此, 对集合、对象的访问非常便利。Struts2 标签库将所有标签都统一到一个标签库下, 简化了用户对标签库的使用。

Struts2 标签库整合了 Dojo 的支持, 因此, 可以生成更多页面表示效果。Struts2 提供了许多额外的标签, 包括: 如日期时间选择器, 树形结构等。除此之外, 借助于底层的 DWR 支持, Struts2 标签库提供了 Ajax 支持, 可以通过使用 Struts2 标签库来非常轻松地完成各种 Ajax 效果。

Struts2 提供了主题、模板支持, 极大简化了视图页面的编写, 而且它们有很好的可扩展性, 如果现有的主题、模板不能满足项目需求, 完全可以开发自定义的主题、模板、可以实现更好的代码复用; Struts2 还允许在页面中使用自定义组件, 这完全能满足项目中页面显示复杂多变的需求。

同 WebWork 一样, Struts2 对多种表现层技术: JSP、Velocity 和 FreeMarker 等都有很好的支持, 从而给开发者更多的选择, 提供了更好的兼容性。

### 2.3.3 Struts2 的工作原理

Struts2 的体系结构如图 2.5 所示，下面依据 Struts2 体系结构说明 Struts2 的工作机制<sup>[14]</sup>。

- (1)用户在客户端产生一个服务器端(例如 Tomcat, WebLogic)的请求;
- (2)客户端产生的请求经过框架的一系列过滤器(Filter) (这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器, 这个过滤器对于 Struts2 和其它框架的集成很有帮助, 例如: SiteMesh Plugin ), 为请求经过 Struts2 安全处理打下基础;
- (3)接着这个请求经过 Struts2 核心控制器 FilterDispatcher, FilterDispatcher 询问 ActionMapper 来决定这个请求是否需要调用某个 Action;
- (4)如果 ActionMapper 决定需要调用某个 Action , FilterDispatcher 把请求的处理交给 ActionProxy;
- (5)ActionProxy 通过 Configuration Manager 询问框架的配置文件(struts.xml), 找到需要调用的 Action 类。
- (6)ActionProxy 创建一个 ActionInvocation 的实例。
- (7)ActionInvocation 实例使用命名模式来调用, 在调用 Action 的过程前后, 涉及到相关拦截器( Interceptor) 的调用。
- (8)一旦 Action 执行完毕 ActionInvocation, 负责根据 struts.xml 中的 result 配置找到对应的返回结果。返回结果通常是(但不总是, 也可能是另外的一个 Action 链)一个需要被表示的 JSP 或者 FreeMarker 的模板。在表示的过程中可以使用 Struts2 框架中继承的标签。在这个过程中需要涉及到 ActionMapper 。

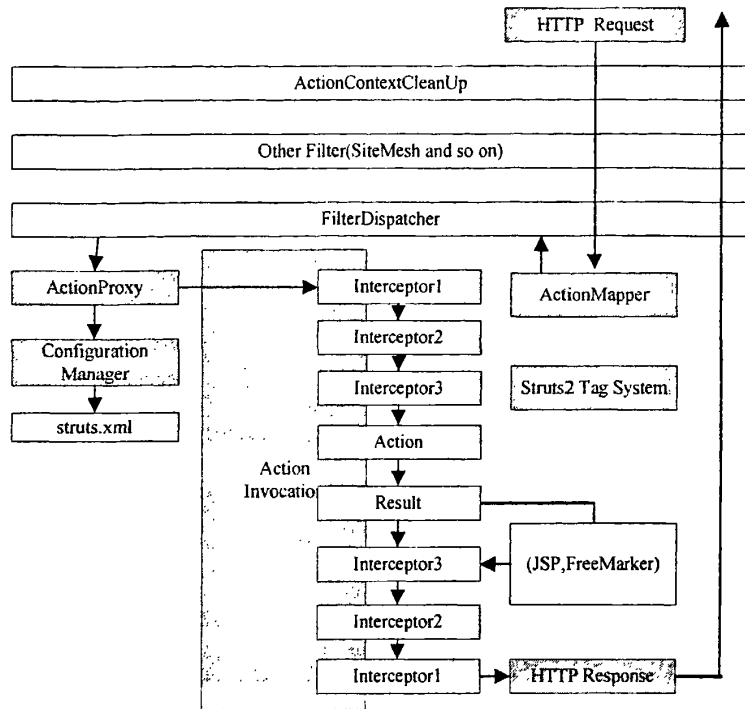


图 2.5 Struts2 架构

Fig.2.5 Struts2 frame

## 2.4 Spring 框架

Spring 框架起源于其缔造者 Rod Johnson 2002 年所写的《<Expert One-on-One J2EE Design and Development>>》一书的基础代码<sup>[15]</sup>。在这本书中，Rod 提出了他自己的 J2EE 思想，他坚信一种轻量级的，基于 JavaBean 的框架完全可以满足大多数开发人员的要求。Spring 为企业应用的开发提供了一个轻量级的解决方案，该解决方案包括：基于依赖注入的核心机制，基于 AOP 的声明式的事务管理，与多种持久层技术的整合，以及优秀的 Spring MVC 框架等。Spring 致力于 JavaEE 应用各层的解决方案，而不是仅仅专注于某一层的方案。可以说：Spring 是企业应用开发的“一站式”选择，Spring 贯穿于 web 表现层、业务逻辑层和数据持久层<sup>[15]</sup>。

虽然 Spring 提供了几乎所有各层的解决方案，但是 Spring 并没有想取代别的框架；而是可以以高度的开发性与其它框架无缝整合。所以用户可以根据自己的实际需求来灵活的选择 Spring 中的各个部分。Spring 中的许多特性，如：JDBC 抽象层或者与 Hibernate 集成，都可以作为一个库单独使用，当然也可以作为 Spring 整体方案的一个部分。从这里可以看出，Spring 提供了极大的灵活性，开发者可以选择不同领域优秀的工具，又可以选择 Spring 本身的各个部分。下面来看一下 Spring 的基本结构<sup>[16]</sup>。

#### 2.4.1 Spring 的基本结构

Spring 框架的基本结构图如图 2.6 所示。

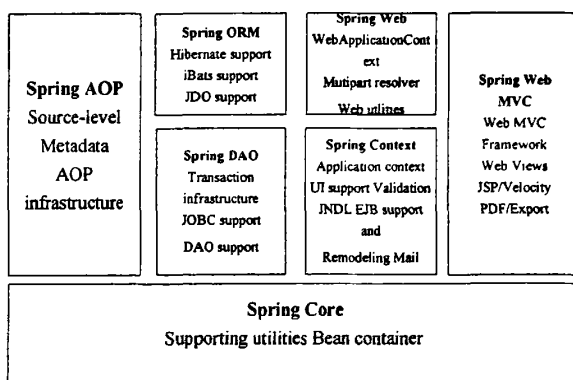


图 2.6 Spring 框架组件结构图

Fig.2.6 Spring Framework

由图 2.6 中所示，Spring 框架有 7 个基本模块，每个模块都有一个对应的 jar 文件<sup>[17]</sup>。当使用 Spring 框架时，必须使用 Spring Core，它代表了 Spring 框架的核心机制。而其它部分的组件则可选择使用。Spring 框架的其它部分，例如 Spring MVC、Spring DAO 等，都是建立在 Spring Core 基础之上的。其中 Spring Core 提供使用 Factory 模式所创建的 BeanFactory 来进行 Bean 类的构建、销毁以及装配功能；Spring AOP 提供了把交叉业务逻辑代码封装为切面透明的植入到具体的业务

逻辑代码的功能，减少了交叉业务逻辑代码和业务逻辑代码的耦合，使程序代码能够容易的集成并提供灵活的中间件服务；Spring ORM 模块提供了和其它 ORM 持久性框架(例如：Hibernate、JDO 和 iBatis 等)的集成，并且更加优秀的封装和发展了原先的框架，提供了安全的事务管理；Spring DAO 提供了 JDBC 的抽象层，消除了原先使用 JDBC 进行编程的繁琐以及错误，提供了简单的并且安全的声明式事务管理办法；Spring 中的 Web 包提供了基础的针对 Web 开发的集成特性，例如多方文件上传，利用 Servlet listeners 进行 IoC 容器初始化和针对 Web 的 Application Context。当与 WebWork 或 Struts 一起使用 Spring 时，这个包使 Spring 可与其它框架结合；SpringContext 模块提供了对事件处理、国际化、JNDI、Javamail 等的支持；Spring 中的 MVC 封装包提供了 Web 应用的实现。Spring 的 MVC 框架并不是仅仅提供一种传统的实现，它提供了一种清晰的分离模型，在领域模型代码和 Web Form 之间。并且，还可以借助 Spring 框架的其它特性<sup>[18]</sup>。

总之，Spring 虽然提供了很强大的机制，但是其最根本，最重要的核心机制还是 IoC(Inversion of Control)容器和 AOP(Aspect Oriented Programming)。

### 2.4.2 Spring 的核心机制 IoC 与 AOP

IoC 模式与 AOP 在 Spring 中占有着举足轻重的地位，是 Spring 框架的核心部分。其中 IoC 模式是贯穿 Spring 框架始终的一个概念，它秉承了 GoF 模式的基本理念，即：针对接口编程，而不是实现。

#### 1. IoC 模式简介

当进行应用开发时，在模块之间以及系统整体划分的层次之间不可避免的会出现耦合现象，于是模块与模块，层与层之间必定会形成依赖关系，如果模块之间依赖过强，就会为对应用的理解以及维护形成很大困难，因此，如何来尽可能消除模块彼此之间的耦合，成为在开发中首先考虑的问题。

IoC(Inversion of Control) 模式<sup>[19-21]</sup>，即控制反转，就是为了要解决模块间的耦合，依赖注入(Dependency Injection) 是对 IoC 模式的一种扩展的解释。依赖注入的基本原则就是：应用对象不应该负责查找资源或者其它依赖的协作组件。配

置对象的工作应该由 IoC 容器完成,“查找资源”的逻辑应该从应用代码中抽取出来,交给容器负责。使用依赖注入时, IoC 容器负责查找组件需要的资源,并将必要的资源注入到业务对象中。这样,只要容器重新配置,用不同的方式获得资源,就可以让组件获得不同的资源,而不必修改应用代码<sup>[22]</sup>,从而使得模块之间,甚至层与层之间解耦。

依赖注入有两个方式实现,一种是设置值注入(Setter Injection),另一种是构造子注入(Constructor Injection)。

设值注入的优势:

(1) 对于习惯了传统 JavaBean 开发的开发者而言,通过 setter 方法设定依赖关系显得更加直观,更加自然。

(2) 如果依赖关系(或继承关系)较为复杂,那么构造子注入模式的构造函数也会相当庞大(需要在构造函数中设定所有依赖关系),此时设值注入模式往往更为简洁。

(3) 对于某些第三方类库而言,可能要求组件必须提供一个默认的构造函数(如 Struts 中的 Action),此时构造子注入类型的依赖注入机制就体现出其局限性,难以完成开发者期望的功能。

构造子注入的优势:

在构造期即创建一个完整、合法的对象<sup>[23]</sup>,对于这条 Java 设计原则,构造子注入无疑是最好的响应者。避免了繁琐的 setter 方法的编写,所有依赖关系均在构造函数中设定,依赖关系集中呈现,更加易读。

由于没有 setter 方法,依赖关系在构造时由容器一次性设定,因此组件在被创建之后即处于相对“不变”的稳定状态,无需担心上层代码在调用过程中执行 setter 方法对组件依赖关系产生破坏,特别是对于 Singleton 模式的组件而言,这可能会对整个系统产生重大的影响。

同样,由于关联关系仅在构造函数中表达,只有组件创建者需要关心组件内部的依赖关系。对调用者而言,组件中的依赖关系处于黑盒之中。对上层屏蔽不



必要的信息，也为系统的层次清晰性提供了保证。

通过构造子注入，意味着可以在构造函数中决定依赖关系的注入顺序，对于一个大量依赖外部服务的组件而言，依赖关系的获得顺序可能非常重要，比如某个依赖关系注入的先决条件是组件的 `DataSource` 及相关资源已经被设定。

可见，构造子注入和设值注入模式各有千秋，Spring 框架对设值注入与构造子注入都提供了完善的支持，用户可以根据自己的实际情况来进行选择。

### 2. AOP 简介

简单地讲，AOP 就是将那些与业务无关，却为业务模块所共同调用的逻辑或责任，例如事务处理、日志管理、权限控制等，封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。

实质上，AOP 只是 OOP 的一种补充或某种改进，它转换了编程的范式和视角，关注了一直以来被 OOP 忽略或者说未能解决好的角落，使开发人员可以更好地将本不该彼此纠缠在一起的责任(如银行业务和事务处理)分离开来。通过面向方面的编程，可以将程序的责任分开，对象与方面互不干扰。面向方面的模块并非显式地为对象所调用，而是通过或注入或截取的方式，去获得被封装的对象内部方法间的消息，然后做出相应地处理。也许面向方面的模式破坏了对对象的封装，却正其如此，方才能降低模块与模块之间的耦合度。同样地，通过对“方面”的封装，将这些通用的功能从不同的类中分离出来，使不同的模块都能共享同样的“方面”，这也极大地减少了重复代码。

AOP 是 Spring 框架的重要组成部分，它实现了 AOP 联盟约定的接口。Spring AOP 是由纯 Java 开发完成的。Spring AOP 只实现了方法级别的连接点，在 JavaEE 应用中，AOP 拦截到方法级的操作已经足够。OOP 倡导的是基于 setter/getter 的方法访问，而非直接访问域，而 Spring 有足够理由仅提供方法级的连接点。为了使控制反转(IOC)很方便的使用到非常健壮、灵活的企业服务，则需要 Spring AOP 的实现。Spring AOP 在运行时才创建 Advice 对象。Spring AOP 的优点如下<sup>[24]</sup>：

- (1) 允许开发者使用声明式企业服务，比如事务服务、安全性服务。

(2) 开发者可以开发满足业务需求的自定义方面。

(3) 开发 Spring AOP Advice 很方便,可以借助代理类快速搭建 Spring AOP 应用。

## 2.5 数据持久化与 Hibernate

### 2.5.1 数据持久化与 ORM

何谓“持久化”? Wikipedia 给出了持久化的定义: In Computer science, Persistence refers to the characteristic of data that outlives the execution of the program that created it. Without this capability, Data only exists in memory, and will be lost when the memory loses power, such as on computer shutdown<sup>[25]</sup>.概括的说,即把数据(如内存中的对象)保存到可永久保存的存储设备中(如磁盘),并且在适当的时候可以把外部持久设备中的数据以原先的状态恢复到内存中。在实际应用开发中,持久化的主要应用是将内存中的数据存储在关系型的数据库中,当然也可以存储在磁盘文件中、XML 数据文件中等等。在大多数情况下,特别是企业级应用,数据持久化往往意味着将内存中的数据保存到磁盘上加以“固化”,而持久化的实现过程则大多通过各种关系型数据库来完成。

软件应用,或者企业级应用,作为处理数据的工具,从根本上分析它主要具有三个功能:展示数据、处理数据和管理数据<sup>[26]</sup>。随着 OOP(面向对象的编程)提出和发展,在企业级应用中,分别用三个层次来描述这三个功能,分别为:表示层、业务逻辑层和对象持久层,其中对象持久层就负责对象的持久化,即提供对面向对象的业务数据进行保存、读取和检索等功能<sup>[26]</sup>。

在企业级开发应用中, JDBC 可以说是访问数据持久层最原始、最直接的方法。在企业级应用开发中,一般使用 DAO 并适当应用继承和 Business Delegate 设计模式来封装对象的 CRUD 操作,虽然最大可能的避免了重复编写底层数据库操作代码,但是底层代码,例如管理 Connection 和 Statement 对象等等,仍然必须被编写至少一次,仍然属于跨越项目的重复操作。

目前，开源组织针对于不同项目需求开发出风格各异的对象持久化方案。如基于 JDBC API 的框架软件 iBatis, Spring JDBC Template, EJB2 中的 Entity Bean 以及 Hibernate 框架等。但是由于工业标准 EJB2 的过度设计、使用起来太过复杂；而 iBatis 和 Spring JDBCTemplate 只是对对象关系映射的简单映射，使用起来虽然简单灵活，但是功能太少。对象关系映射框架(ORM)Hibernate 正集合了以上二者之长，因此 Hibernate 的作者 Gavin King 被力邀参加制定新一代的工业标准 EJB3，在 EJB3 中，Hibernate 成为了数据持久化标准<sup>[27]</sup>。

何谓对象关系映射 ORM(Object/Relational Mapper)，即“对象-关系型数据映射组件”。对于 O/R，即 Object(对象)和 Relational(关系型数据)，表示必须同时使用面向对象编程和关系型数据进行开发。对象/关系映射技术是随着面向对象软件开发方法的发展而产生的。它在解决面向对象技术和关系数据库之间的“阻抗不匹配”时起到了关键的作用，是目前解决关系数据库中持久化对象存储和访问的主要技术。目前，国内对于该技术的研究主要集中在应用方面。国外实现了该技术的持久层框架产品已有不少，其中有一部分产品是商品化了的，也有一部分是开源源码的<sup>[28-30]</sup>。

在 Java 世界中，Hibernate 可以说是在众多 ORM 软件中，获得关注最多、使用最为广泛的框架，它成功的实现透明持久化，使用面向对象的 HQL 封装 SQL，为开发人员提供了一个简洁灵活且面向对象的数据访问接口。现在的 Hibernate 不仅具有专业打造的优秀性能，而且是商业支持的开源产品，使用简单，并且逐步提供了完善而全面的解决方案。

### 2.5.2 Hibernate 的体系结构与实现

Hibernate 是一个开放源码的 ORM 持久层框架。作为优秀的持久层框架实现，Hibernate 框架提供了强大、高性能的对象到关系型数据库的持久化服务，开发人员可以使用面向对象的设计进行持久层开发。简单的说，Hibernate 是一个将持久化类与数据库表相映射的工具，其中每个持久化类在关系型数据库中映射为一张表，而此类的实例映射为对应表中的一条记录，类中的映射属性映射为表中

的字段，开发者只需直接使用面向对象的方法操作此持久化类实例，即可完成对数据库表数据的插入、删除、修改、读取等操作。当然实际的 Hibernate 框架非常复杂，用分层的概念划分的话，它相当于在业务逻辑处理层和数据库底层 JDBC 驱动之间的一层，即通常说的持久化层，而开发人员通过 XML 配置文件或者 Annotation 标签将具体的持久化类与数据库表映射起来。

### 1. Hibernate 的体系结构

Hibernate 架构<sup>[31]</sup>如图 2.7 所示。

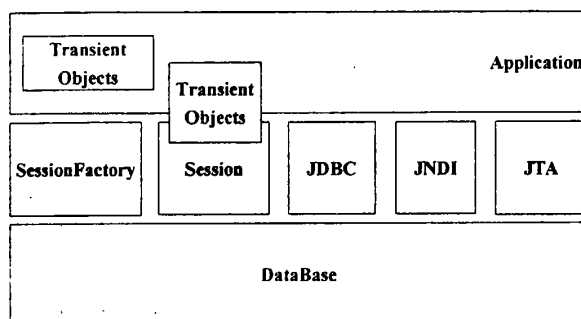


图 2.7 Hibernate 架构

Fig. 2.7 Hibernate FrameWork

图 2.7 是对 Hibernate 高层的概览。图中说明了 Hibernate 提供持久对象 (Persistent Objects) 给应用程序 Application 的方法结构。在进行应用的持久化层开发时，只需要编写简单的 POJO 类以及建立类之间的关联关系，并通过简单的设置配置文件 hibernate.properties 和创建 POJO 类所对应的 XML 映射(或者使用 Annotation)文件，这样就可以实现对象的持久化，从而代替大量复杂的 JDBC 编程，Hibernate 支持 17 种数据库，同时支持连接池应用<sup>[26]</sup>。

Hibernate 的核心类主要分为两部分：一部分负责初始化 Hibernate，另一部分为应用程序提供通用数据接口。Configuration 接口负责配置并启动 Hibernate，创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先

定位映射文档位置、读取配置，然后创建 `SessionFactory` 对象；`SessionFactory` 接口负责初始化 `Hibernate`。它充当数据存储源的代理，并负责创建 `Session` 对象。由于 `SessionFactory` 是线程安全的对象，因此在整个应用中拥有唯一即可；`Session` 接口负责执行被持久化对象的 `CRUD` 操作，但是 `Session` 对象是非线程安全的，所以每次事物完毕就需要关闭当前 `Session`；`Transaction` 接口负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码；`Query` 和 `Criteria` 接口负责执行各种数据库查询。它可以使用 `HQL` 语言或 `SQL` 语句两种表达方式。

### 2. Hibernate 工作原理

#### (1) 初始化 Hibernate

初始化 `Hibernate` 在程序代码中可以仅有一行代码，那就是：

```
Configuration config = new Configuration().configure();
```

虽然仅有一行代码，但是 `Hibernate` 底层却做了很多。首先 `Hibernate` 需要加载用来配置数据库连接的配置文件，一般可以为 `Properties` 文件或者为 `XML` 文件，创建了一个新的 `Configuration` 对象后，这个对象会创建一个新的 `SessionFactory` 实例，这个实例用来负责初始化 `Configuration` 对象内部用于保存映射信息的变量，并通过调用 `Environment` 的 `getProperties()` 方法来加载当前根目录下的 `hibernate.Properties` 文件，如果不存在这个文件，`Configuration` 对象则会使用 `configure()` 来加载 `hibernate.cfg.xml` 文件，如果这个文件位于当前根目录下，则会顺利加载，如果不在，则可以把这个文件包装成 `File` 对象写在这个方法的参数中。则可顺利加载配置文件，但是注意这个方法只能调用一次。

#### (2) 创建 Session 工厂 SessionFactory

`SessionFactory` 负责创建 `Session` 实例。可以通过 `Configuration` 实例构建 `SessionFactory`：

```
SessionFactory sessionFactory = config.buildSessionFactory();
```

`Configuration` 实例 `config` 会根据当前的配置信息，构造 `Session Factory` 实例并返回。`Session Factory` 一旦构造完毕，即被赋予特定的配置信息。也就是说，之后

config 的任何变更将不会影响到已经创建的 SessionFactory 实例(sessionFactory)。如果需要使用基于改动后的 config 实例的 SessionFactory, 需要从 config 重新构建一个 SessionFactory 实例。

### (3) Session

Session 是持久层操作的基础, 相当于 JDBC 中的 Connection。

Session 实例通过 Session Factory 实例构建:

```
Configuration config =new Configuration().configure();
```

```
SessionFactory sessionFactory =config.buildSessionFactory();
```

```
Session session = sessionFactory.openSession();
```

之后就可以调用 Session 所提供的 save、find、flush 等方法完成持久层操作。

## 3. Hibernate 的映射机制

Hibernate3 版本支持两种形式的映射元数据: 一种是基于 java5 注释新特性, 另一种是基于 XML 文件。

### (1) 对象关系映射

传统的 Hibernate 使用 XML 文件为实体类添加映射元数据, 映射文件通常以 “.hbm.xml” 作为后缀。以下是映射文件的一个简单例子:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN">

<hibernate-mapping>
    <class name="com.allwin.recruit.RecruitPost" table="acegi.recruitpost" >
        <id name="oid">
            <generator class="sequence">
                <param name="sequence">SEQUENCE</param>
            </generator>
        </id>
        <property name="name" type="string" length="20" />
        <property          name="publishDate"          type="timestamp"
```

```
column="PUBLISHDATE">
    </property>
</class>
</hibernate-mapping>
```

通过使用这个映射文件，Hibernate 将会在应用程序第一次启动时创建一个表格如图 2.8 所示。

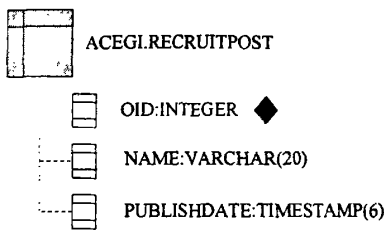


图 2.8 Hibernate 映射表结构

Fig. 2.8 The Table of Hibernate ORM

由图 2.8 可以看出一个类对应的映射文件中的 Table 属性可以设置这个类映射的表名，而这个类的属性在映射文件中根据它的 type 属性来映射它们各自在数据库中的类别，从而也可以得到类何种属性对应于所映射表结构的一个字段。如果用户需求有所改变，可以根据需求改变类的属性，并且在映射文件中修改，那么再次加载应用时就会获得一个全新的表结构。

注释(Annotation)可以说是 JavaEE5 带来的一项全新特性，它允许在 Java 源代码中直接写入元数据，可以确保数据与元数据集中存放于同一个文件夹，元数据就是描述数据的数据。在全新的 EJB3 规范中，注释已经渗透到各种类的每一个角落。

JPA 是 EJB3 规范中负责对象持久化的应用程序编程接口，它定义了一系列的注释。这些注释大体可分为：类级别注释、方法级别注释、字段级别注释。给实体类添加适当的注释可以在程序运行时告诉 Hibernate 如何将一个实体类保存到数据库中以及如何将数据以对象的形式从数据库中读取出来<sup>[32]</sup>。

目前有两种注释方案可以确定对象与表格之间的对应关系：一种是注释实体类的属性字段，成为字段访问方式，另一种是注释实体类的属性访问方法，称为属性访问方式。不过在应用开发中一般使用字段访问方式，例如：

```
@Entity
@SequenceGenerator(name="sequence", sequenceName="sequence")
@Table(name="acegi.recruitpost")
uniqueConstraints={@UniqueConstraint(columnNames={"name", "publish"})})
Public class RecruitPost implements Serializable{
@Id
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="sequence")
private into id;
@Column(name="name", length="20", nullable=false)
private String name;
@Temporal(value=TemporalType.TIMESTAMP)
@Column(name="PUBLISHDATE")
Private Date publish;
.....
}
```

其中省略号为各个属性的 set 和 get 方法，以上的 POJO 类一经加载就会形成与以上 XML 配置文件所形成表结构完全相同的结构。其中@Entity 是定义一个 POJO 类成为实体类所必需的注释，@Table 是定义实体类对应到数据库中的表结构属性，@Id 注释是用来标记表结构的主键属性，@Column 注释则用来标记实体类属性映射到数据库中字段的属性，还有许多非常复杂的其它注释属性，由于实体之间必然会产生关联，下文就着重研究使用注释来映射实体类的数据关联。

## (2) 数据关联

### 1 一对一关联

持久化对象之间一对一的关联关系可以分为单向一对一关联和双向一对一关



联，其中单向一对一关联是最简单的类关联。在关系型数据库中可以有两种方式实现这种一对一关系，一种策略是存在对应关系的表格记录各自拥有不同的逐渐，关联关系通过在某个表格当中添加额外的外键列来实现；另外一种策略则是通过使用共用主键来确定一对一关联关系<sup>[32]</sup>。

例如：

策略 1：

```
@Entity
public class RecruitPost implements Serializable{
    private into id;
    private String name;
    private Date publish;
    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="tenant_oid")
    private Tenant tenant;
    .....
}
```

策略 2：

```
@Entity
public class RecruitPost implements Serializable{
    private into id;
    private String name;
    private Date publish;
    @OneToOne(cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Tenant tenant;
    .....
}
```

从上面代码可以看出，使用@JoinColumn 可以指定数据库采用策略 1 来实现

一对一关联，而使用@PrimaryKeyJoinColumn 可以指定数据库采用策略 2 来实现一对一关联。而@OneToOne 却是标注了两个对象之间的关联关系为一对一关联关系。

双向的一对一关联，数据库采用的策略则只有通过在双方映射表中加入对方的主键作为外键来形成双向一对一关联，但是这样的策略很明显违反了数据库设计的设计规范，因此在对象映射中要尽量避免使用双向一对一关联策略。

## 2 一对多关联

一对多的关联关系在对象世界中也有单向和双向之分。在一对多关联中，主类对象含有多个目标类对象，所有的目标类对象均包含在集合对象中，一般以 set, list 集合居多。例如：

单向一对多：

```
@Entity
public class RecruitPost implements Serializable{
    private int id;
    private String name;
    private Date publish;
    private Tenant tenant;
    @OneToMany(targetEntity=Demand.class, cascade=CascadeType.ALL)
    @JoinColumn(name="recruitpost_id")
    Private List<Demand> demands;
    .....
}
```

双向一对多：

```
@Entity
public class RecruitPost implements Serializable{
    private int id;
    private String name;
    private Date publish;
```

```

private Tenant tenant;

@OneToMany(targetEntity=Demand.class          cascade=CascadeType.ALL    ,
mappedBy="recruitpost")
private List<Demand> demands;
.....
}

@Entity
public class Demand implements Serializable{
private int id;
private String name;
@ManyToOne
@JoinColumn(name="recruitpost_id")
Private RecruitPost recruitpost;
.....
}

```

由以上一对多注释中，可以知道`@OneToMany` 是用来标注关联中处于多的一方，也就是在类世界中，使用集合包装的属性，其中 `targetEntity` 属性用来指定被集合包含的目标类，而`@JoinColumn` 注释用来设定目标对象在关系型数据库表中拥有源对象的主键作为外键来进行关联，其中 `name` 属性用来标注外键字段名字。而在双向一对多关联中，`@ManyToOne` 注释则用来标注处于关联关系“一”的一方，即源对象，表明这个对象与目标对象为一对多的关联关系，而在`@OneToMany` 注释中，属性 `mappedBy` 用来描述目标对象关联对象的属性名字，从而获得该关联对象的 `oid` 作为外键。

但是如果看一下这两种方式所映射的数据库表结构，就会发现尽管在 Java 语言里面可以定义对象引用实现双向关联，但是这种双向关联反映到数据库表关联上仍然是单向的，也就是说，无论单向双向，所映射的是数据库表结构是一样的。

### 3 多对多关联

Hibernate 关联关系中虽然提供了多对多关联关系的注释`@ManyToMany`，但是

由于在关系型数据库中实现多对多关联关系需要借助中间表来实现，即是把多对多关联关系划分为两个多对一关联关系，也可以说是两个一对多关联关系，因此在实际应用开发时，会避免使用多对多关联注释，而是把多对多关联关系拆分为两个一对多关联，从而提高数据库查询性能。

虽然两个对象之间的关联关系最普遍存在的就是以上的三种关联关系，但是在 java 语言中，仍然存在其它关联关系映射，例如组件关系映射，继承关系映射，值类型集合映射等，EJB3 都提供了相应的注释，在这里就不一一详述了。

## 2.6 Ajax 框架研究

### 2.6.1 Ajax 简介与 RIA

随着 Web 技术的飞速发展，网络逐渐由一个发布信息平台转变成为了网络交互的平台，这就是 Web1.0 到 Web2.0 的提升，企业级应用开发的软件逐渐向 SAAS(SoftWare As A Service)发展，企业开发的产品更应该是提供一种交互的服务，如何使 Web 的响应更加灵敏、数据传输更加快捷，成为当今关注的热点。Ajax 技术正是为了满足用户的这种需求应运而生的，它为浏览器提供了与服务器端异步通信的能力，从而使用户从请求/响应的循环中解脱出来，这样，使得 Web 浏览器看起来就像是即时响应的桌面应用程序一样<sup>[33]</sup>。

为了改善用户体验，出现了一种新类型的 Web 应用，那就是 RIA(Rich Internet Application)，即“富 Internet 应用”，这些应用程序吸收了桌面应用程序响应快，交互性强的优点。RIA 改进了 Web 应用程序的用户交互，提供了一种更丰富、更具有交互性和响应性的用户体验。现在典型的 RIA 技术有：Microsoft 的 ClickOnce 技术、Sun 的 Java Web Start 技术、Adobe 的 Flash 技术和 Ajax 技术。可以说 RIA 应用代表着目前 WEB 应用的发展趋势。Ajax 技术与其它 RIA 应用不同的是，基于 Ajax 技术的应用完全基于现有浏览器，所以兼容性最好，无需下载任何客户端，这也是 Ajax 魅力所在<sup>[34]</sup>。

Ajax 是 Asynchronous JavaScript and XML(异步 JavaScript 和 XML)的缩写，由

著名的用户体验专家 Jesse JamesGarrett 在 2005 年 2 月 18 日发表的一篇名为 <<Ajax:A New Approach to Web Applications>>的文章中首先提出<sup>[35]</sup>。在广义上可以将 Ajax 定义为:基于标准 web 技术创建的、能够以更少的响应时间带来丰富的用户体验的一类 web 应用程序所使用的技术的集合。

确切的说, Ajax 不是一种技术,而是将一系列相关的技术组合应用的技巧,“老技术,新技巧”是对 Ajax 比较恰如其分的描述。基于 Ajax 的 web 应用中使用的技术,如 XMLHttpRequest、JavaScript、CSS 等,确实是 Web 开发领域中的“老”技术,而 Ajax 的“新”体现在它综合运用这些“老”技术的方式,以及这种运用方式所带来的独特用户体验, Ajax 的成功之处就在于此。

标准的 Ajax 包括:应用 XHTML 和 CSS 标准化、使用 DOM 实现动态显示和交互、采用 XML 和 XSLT 进行数据交换与处理、用 XMLHttpRequest 实现异步数据读取和用 JavaScript 绑定和处理所有数据<sup>[36]</sup>。

### 2.6.2 Ajax 工作方式

Ajax 的核心就是 JavaScript 对象 XMLHttpRequest,它允许通过 JavaScript 向服务器发送请求,并处理服务器响应,避免阻塞用户动作。通过使用 XMLHttpRequest 对象,浏览器通过客户端脚本与服务器交换数据,Web 页面无须频繁重新加载,其内容也由客户端脚本动态更新。

异步是指基于 Ajax 的应用与服务器通信的方式。对于传统的 Web 应用,每次用户发送请求或向服务器请求获得新数据时,浏览器都会完全丢弃当前页面,而等待重新加载新的页面。在服务器完全响应之前,用户浏览器将是一片空白,用户的动作必须中断。异步是指用户发送请求后,完全无须等待,请求在后台发送,不会阻塞用户的当前活动。用户无需等待第一次请求得到完全相应,可以立即发送第二次请求。整个 Ajax 应用的工作过程如下<sup>[34]</sup>:

(1) JavaScript 脚本使用 XMLHttpRequest 对象向服务器发送请求。发送请求时,既可以发送 GET 请求,也可以发送 POST 请求。

(2) JavaScript 脚本使用 XMLHttpRequest 对象解析服务器响应数据。

(3) JavaScript 脚本通过 DOM 动态更新 HTML 页面, 也可以为服务器响应数据增加 CSS 样式表, 在当前的网页某个部分加以显示。

XMLHttpRequest 是整个 Ajax 技术的灵魂, 只有依赖于 XMLHttpRequest 对象, Ajax 才能实现异步发送请求; JavaScript 脚本是 Ajax 技术中另一个重要组成部分, 是 Ajax 技术的编程脚本, 它主要用来创建 XMLHttpRequest 对象并向服务器端发送请求并通过回调脚本和 DOM(Document Object Model)技术来动态更新 HTML, 因此 JavaScript 技术是 Ajax 技术的粘合剂; DOM 技术可以以一种结构化的方式来动态操作 HTML 文档, 例如(增加、删除文档节点、属性或者事件), 因此 DOM 也是 Ajax 技术的核心技术; 在 web 页面上使用 CSS (Cascading Style Sheet, 级联样式单)和 XML(可扩展标记语言)技术, 可以有效的对页面的布局、字体、颜色、背景和其它效果实现更加精确地控制, 使得页面更加友好。XML 文档是一种结构化文档, 其主要作用就是用于简单数据的表示和交换, 也是 Ajax 技术中不可或缺的技术内容。

### 2.6.3 Ajax 主要框架简介

在 Ajax 技术为用户带来了全新 Web 体验的同时, 该技术就已经呈现了爆炸式发展的趋势, 新的开发框架不断涌现。常用的 Ajax 开发框架组要分为两类: 基于浏览器端的框架和基于服务器端的框架。常用的浏览器端框架有: Prototype 框架, ExtJs 框架, Dojo 框架等, 基于服务器端的框架有: DWR 框架, JSON-RPC-Java 框架, Ruby on Rails 等框架。

Prototype.js 是由 Same Stephen 写的一个 JavaScript 类库, 它构思巧妙, 并且兼顾标准、能跨浏览器使用。借助于 Prototype.js 的帮助, 开发者能够轻松建立有高度互动的 Web2.0 特性的富客户端页面<sup>[37]</sup>。它的主要功能可以体现在三个方面: 一些方便的工具函数、扩展了一些原有类, 增加了一些自定义对象、Ajax 应用相关对象。Ajax 应用相关对象更是大大简化了 Ajax 应用的开发, 避免了用户实现 XMLHttpRequest 对象, 避免用户手动打开与服务器的连接等繁琐、通用的步骤。用户只需要确定请求发送的 URL, 并为应用确定回调函数, 整个 Ajax 过程就可以

在 Prototype.js 的帮助下完成，甚至无需使用回调函数。

ExtJs 框架是真正的开发框架，它不仅仅解决了底层的浏览器兼容问题、通用的一些开发功能，也为开发者提供了大量的内建控件库，例如，用于布局的 BorderLayout、Viewport，用于生成表单的 FormPanel 等等，这些控件都是内建的，而非基于 ExtJs 开发，这个差别直接导致了整个开发框架的完整性有很大程度的提高，对于开发人员来说，与其把很多来自第三方基于一套库开发的控件放到一个项目中来，倒不如直接用内建的控件来得方便，而且整合度更高，代码可靠性也高。ExtJs 还有一点值得称道，那就是它为其它 JavaScript 开发库提供了兼容适配器，主要为 jquery，prototype+Script.aculo.us 和 Yahoo UI 提供了适配器，这是其它开发库所没有涉及的，这直接说明 ExtJs 的开发者们很有远见，融百家之众长，把其它库好的地方直接引入为我所用。因此在现在的许多 erp 应用开发中，大部分都会采取 ExtJs 框架来进行 RIA 开发<sup>[38]</sup>。

Dojo 框架是最老的框架之一，于 2004 年 9 月开始开发，不断完善。这个项目的目标是建立充分利用 XHR 的 DHTML 工具包，并把重心放在可用性问题上。Dojo 工具包中包含了一些 API 和工具来支持开发富客户端的 Web 应用。Dojo 包含了一个智能打包系统、页面特效、拖放效果支持、小应用程序(widget)API、事件抽象、客户端存储和 Ajax 交互 API<sup>[39]</sup>。Dojo 解决了一些 Ajax 应用中常见的可用性问题，如浏览器导航问题，包括前进、后退按钮的支持，更改地址栏的 URL 来收藏书签等；当客户端不完全支持 Ajax 和 JavaScript 时，可以在不提供 Ajax 功能的情况下降级使用，能够很好地尊重客户的意愿(关闭 JavaScript)和支持各种新旧版本的浏览器，这也是 web 标准所倡导的更具亲和力的 web 应用模式。Dojo 看上去是相对成熟的工具包之一，它把重点放在可用性上，这一点很不错。Dojo 表现得相当稳定，它的邮件列表相当活跃。

DWR(Direct Web Remoting)是 Apache 许可下的一个开源项目，它是一个非常专业的 JavaEE Ajax 框架。通过 DWR 的帮助，可以将 Java 组件地方法直接暴露给远程的 JavaScript 客户端。这种方法非常类似于 Java 的远程方法调用，不同的是：

DWR 中调用的客户端是 JavaScript 代码，而不是 Java 代码。通过使用 DWR，可以通过一种简单的方式开发 Ajax 应用。它的主要功能是将服务器端的 java 方法暴露给客户端。当然，它也提供了一套 JavaScript 函数集，从而简化 DOM 元素的操作。DWR 的专业还体现在与其它 JavaEE 框架的整合上，DWR 可以与 Spring 无缝整合，允许直接调用 Spring 容器中的 Bean。不仅如此，他还可以与 Struts、Webwork 和 JSF 等框架整合<sup>[40]</sup>。

JSON-RPC-Java 是 JavaEE 领域的另一个非常优秀的 Ajax 框架，它是一种类似于 RPC 的调用框架：允许 JavaScript 客户端直接调用远程 Java 方法。当然这种直接调用是以 JSON-RPC-Java 框架的底层工作为基础。通过让 JavaScript 客户端直接异步调用远程 Java 方法，从而提供一种 Ajax 能力。JSON-RPC-Java 以简单的 JSON 格式为数据交换格式，允许 JavaScript 对象和 java 对象之间的互相转换，因此，JavaScript 客户端不仅可以调用简单参数和返回值的 java 方法，也可以调用自定义类的参数和返回值的 java 方法。

虽然 Ajax 提供的一些技术，例如淡出技术(YFT-Yellow Fade Technique)，自动刷新(Auto Refresh)技术，部分页面绘制技术(Partial Page Paint)，以及可拖放 DOM(Draggable DOM)等技术极大的改善了用户的体验，但是不可避免的是 Ajax 也存在几个陷阱，例如：滥用 Ajax 技术或许会可能导致无法连接的页面，无效的后退按钮，JavaScript 代码大量膨胀，或许还会暴露业务层导致了整个系统的不安全。因此在使用 Ajax 技术的同时，也要兼顾以上各方面所提出的陷阱，尽量避免出现不安全的因素，导致整个应用“千里之堤，溃于蚁穴”<sup>[33]</sup>。



## 第3章 基于 Struts2, Spring3, Hibernate3 和 Ajax 的 Web 应用 框架设计

在前一章中分别介绍了 Struts2、Spring3、Hibernate3 以及 Ajax 框架的特点及其工作方式，因此在本章中会在理论上提出一个整合这四种框架的新模型，这个新模型应该能够解决现在企业级应用开发中普遍存在的问题，例如：如何使得系统更加灵活可扩展、如何使得具有关联关系的模块之间耦合性更低、整个应用如何能够更加安全的进行管理以及在整个应用在运行之后的维护更加方便。然后会介绍 Struts2, Spring3, Hibernate3 和 Ajax 框架的集成方式，最后介绍这个新模型中 web 表现层、业务逻辑层、数据持久层和域模型层中具体的设计以及实现方式以及各层之间如何使用面向接口的实现方式来进行解耦，并且为了提高客户的体验，必须安全使用 Ajax 框架，以保证整个系统的安全性。然后在下一章会使用一个具体的 ERP 项目系统来进行实体验证本文提出的整合框架。

### 3.1 Web 应用的层次结构

在前面第二章中，在数据持久化中讲到软件的分层时，认为软件作为处理数据的工具，从根本上分析它主要具有 3 个功能：展示数据、处理数据和管理数据。其中处理数据也包含传输数据，管理数据包括存储、读取、查询数据。随着网络的出现，“层次结构”这个概念开始广泛使用，人们开始根据这三个基本功能分工将软件形象的分成如下 3 个层次。

**表现层：**表现层主要是为人机交互提供数据接口。用户可以通过键盘输入数据，通过显示器获取数据。图形化用户接口(GUI)的广泛使用更进一步提升了用户界面使用的友好程度、改善了用户的体验。表现层通常不包含处理业务逻辑的功能模块，但是在需要表现很复杂的数据关系时，例如在目前流行的 RIA 应用中，表现层会含有许多“表现逻辑”。这时，可以根据实际情况将表现层分为表现层和表现逻辑层。

**逻辑层：**逻辑层主要负责实现软件逻辑的处理过程，例如将货物放入购物车、

生成订单、网上付款等等。该层通常从其它层(或者是通过网络从其它软件的逻辑层)获取数据,然后根据定义好的业务逻辑对数据进行加工处理,随后将加工过的数据传递给其它层(或其它软件的逻辑层)。

数据层:数据层负责数据的持久化,目前主要指数据库。数据层提供对数据进行保存、读取和检索等功能。

这种结构称为三层结构。随着对象持久化概念的发展,整个逻辑层分解为业务逻辑层和对象持久层。业务逻辑层只负责处理与业务相关的逻辑,而对象持久层只关心对象的持久化,包括面向对象的数据存储、读取和检索等功能。与此同时,关系型数据库击败了其它所有的对手,成为业界标准数据存储设备,在大多数项目开发中,数据库只作为数据存储设备而直接采用。软件中负责管理数据的逻辑逐渐集中在对象持久层,开发的焦点也就集中到对象持久层,而非数据库。这样一来,从软件开发的角度而言,3层结构就演变为:表现层、业务逻辑层、对象持久层,如图 3.1 所示。

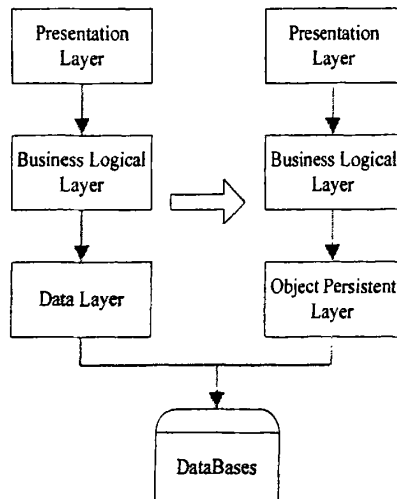


图 3.1 增加持久层的结构对比

Fig. 3.1 Compare for add permanence layer

这种根据功能进行分解的 3 层结构具有很多优点:

**易于开发** 3 个不同层的软件部分可以同时进行开发,甚至可以选用不同的技术、不同的编程语言来开发。

**易于维护** 不同层使用各自有针对性的 API,因此当程序运行发生故障时,很容易定位错误之所在。

**易于升级** 对于一个设计良好的软件而言,单独某个层的升级甚至内部框架软件的替换,从理论上讲不会迫使其其它的层随之升级。同样对于一个设计良好的对象持久层而言,更换底层数据库软件,不会迫使对持久层内部进行相应重构。

**易于扩展,提高重用性** 一个逻辑层可以为多种数据层如 Web 应用、Swing 应用、手机应用提供同样的逻辑功能。随着访问量的不断增加,逻辑层可以通过 Cluster 进行扩展,来满足日益增加的数据处理需求。

随着面向对象技术的发展,层次之间的数据耦合逐渐由原来的较细粒度的数据变量逐渐转化为较粗粒度的数据业务对象,即实例化的类对象,这样更加减小了各个层次之间耦合,并且为数据持久化提供了数据持久化的对象,更加提高了数据持久化的安全。

因此,当前流行的 Web 应用通常被分为四层,分别是:表现层(Presentation Layer),业务逻辑层(Business Logic Layer),数据持久层(Data Persistence Layer)和域模型层(Domain Model Layer)。通过分层,可以降低系统各部分之间的耦合程度,有利于开发人员的分工,增加系统的可维护性及可扩展性<sup>[41-42]</sup>。

#### 3.1.1 表现层

表现层主要是为人机交互提供数据接口,其主要职责为:

- (1) 根据用户的请求,做出相应的响应;
- (2) 根据框架提供的控制器,将页面的请求转发给表现逻辑层进行处理;
- (3) 根据框架提供的控制器把表现逻辑层的显示数据显示在 WEB 页面上;
- (4) 要对即将请求处理的数据进行识别和验证。

### 3.1.2 业务逻辑层

业务逻辑层是根据 web 应用数据流程处理抽象出来的对于业务对象最基本的数据处理，基于这一层次的设置，主要是出于降低表现层和数据持久层的耦合，并且实现基本业务逻辑的复用。因此抽象应用的基本业务逻辑是软件分层的必然选择。业务逻辑层主要负责如下几个内容：

- (1) 根据应用业务逻辑，处理业务逻辑和业务校验；
- (2) 根据数据库处理事务，对业务对象进行事务管理；
- (3) 管理业务层对象间的依赖关系；
- (4) 分离数据持久层和数据表现层，使得更多的基本业务逻辑得到复用；
- (5) 向表现层提供具体业务服务的实现类。

### 3.1.3 数据持久层

数据持久层负责软件应用数据的持久化，是直接与数据库发生关系的层，它主要负责：

- (1) 根据需求分析建立基本业务类，并且分析业务类之间关系与数据库中表及其字段的对应关系；
- (2) 为基本业务类与数据库持久化提供基本业务操作；
- (3) 实现基本业务类的的 CRUD 操作(创建、读取、更新及删除)；
- (4) 建立与管理数据库连接并且实现与底层数据库的选择进行解耦。

### 3.1.4 域模型层

域模型层的 Java 类由实际需求中的业务对象组成，如用户、角色、权限等，其主要职责为：

- (1) 业务领域相关对象的 OO(面向对象)表现；
- (2) 不同层之间传递数据，实现粗粒度的传递方式，提高系统的性能；
- (3) 为表示层提供表现所需要的数据源；
- (4) 为持久层提供被持久化的对象。

## 3.2 SSH+Ajax 组合框架的总体设计

### 3.2.1 SSH+Ajax 框架总体设计

在上文中详细介绍了 Struts2、Spring3、Hibernate3 与 Ajax 各自的特点，然后着手有效集成这四个框架。结合 Struts2、Spring3 和 Hibernate3 的相关技术特点，完成 JavaEE 多层架构的具体设计<sup>[43-45]</sup>。

根据 JavaEE 框架具体结构，并且依据软件代码复用的原则，应采用三层结构：Web 表现层、业务逻辑层、数据持久层。各个层次负责各自相应的功能，各层之间采用接口进行通信，屏蔽了内部的实现细节。架构设计图如图 3.2 所示。

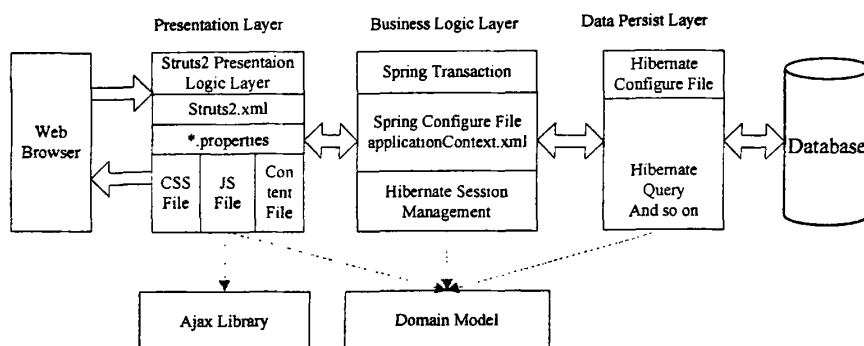


图 3.2 架构总设计图

Fig. 3.2 Chief frame blueprint

**Web 表现层：**应用中 Web 表现层用于显示模型数据，并负责提供用户界面同用户交互。由于使用了 Struts2 框架作为整个架构的表现层，于是就由 Struts2 的控制器 `FilterDispatcher` 来负责和管理用户的请求和显示模型数据，并且在这其中还需要利用 Struts2 的 `Interceptor` 来进行基础业务逻辑(例如日志处理，权限处理等)的调用。在这其中还可以在表现层内部化分复用的层次，分别为业务逻辑表现层(主要用来处理用户请求的业务逻辑)、CSS 文件(主要用来控制和显示模型数据的内容和格式)、JavaScript File(主要用来客户端数据验证以及利用 Ajax 技术来 RIA 显示

和管理数据)以及用来显示模型数据的 Content File, 即用来显示数据的 JSP 或者 HTML 文档。其中还需要配置相应的国际化文档(\*.properties)和相应的控制器转发文档(struts.xml)。

**业务逻辑层:** 业务逻辑层是根据 Web 应用本身业务流程抽象出来的最基本的业务逻辑处理, 业务逻辑层并不亲自和数据库发生关系, 而是利用数据持久层提供的接口来进行基本的业务逻辑处理。使用 Spring 作为业务逻辑层框架, 可以方便的配置数据接口, 通过读取配置文件, Spring 容器自动生成 Bean 实例, 供业务逻辑和表现逻辑使用; 管理事务操作; 提供了与表现层交互的接口; 管理业务级对象之间的依赖性; 隔离了表示层与持久层; 调用持久层接口方法, 向表示层使用的模型层填充数据。

**数据持久层:** 数据持久层是 Web 应用中直接与数据库打交道的逻辑单元, 实现了 O/R 映射, 将面向对象的操作转化为面向关系的操作。由于 Spring 包装了 Hibernate 并且提供了非常容易使用的 Hibernate 模板, 可以很方便的利用 Spring 配置文件来配置数据库的连接以及使用 Hibernate 中提供的诸多 API 来进行数据持久化处理。使用 Hibernate 作为持久层架构能完全屏蔽具体的数据库实现: 通过 Hibernate 提供的 API 和 HQL 查询接口使操作数据更有效、便捷; 向下屏蔽了不同数据库的差异; 向上为业务逻辑层提供需要的访问接口。

让 Struts2 负责 Web 应用强大的表现处理, 让 Spring 负责装配解耦复杂的业务逻辑处理对象, 让 Hibernate 负责处理数据对象的 O/R 映射及持久化, 使用 Ajax 技术来更加方便有效地提高开发效率以及客户满意度。采用这个集成架构开发出来的 Web 应用, 不仅在划分的层次之间拥有更低的耦合性, 而且整个应用的灵活性和可维护性也得到了更大的提高, 最可喜的是利用 Ajax 技术不仅提高了整个应用的开发效率, 而且提高了整个系统的人性化满意度。整合后系统架构的工作流程如图 3.3 所示。

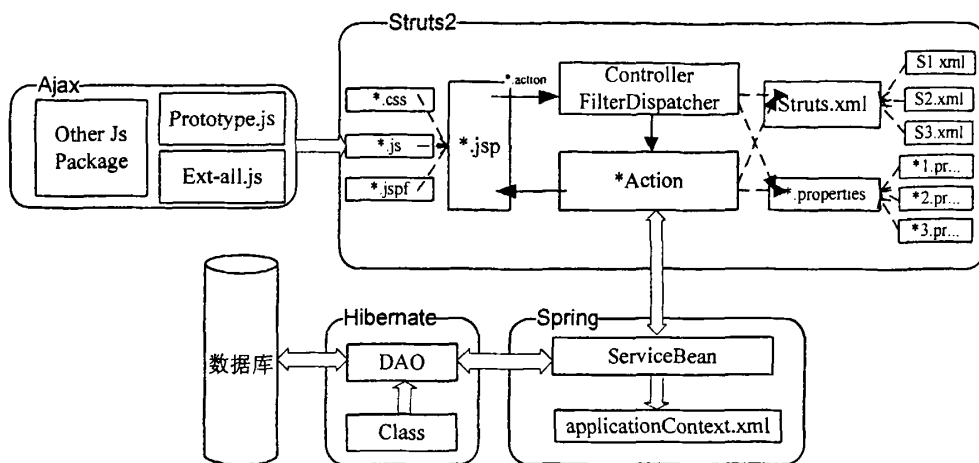


图 3.3 整合框架的工作流程

Fig.3.3 Work flow for conformity frame

### 3.2.2 Struts2 与 Spring 的集成

Struts2 框架只是一个 MVC 框架，只是一个 WEB 层的解决方案，如果不与其它第三方框架组合，将会严重影响 Struts2 框架功能的发挥。因此，struts2 提供了一种非常简单的方式与第三方框架整合。struts2 通过一种“可插拔式”的插件，实现与第三方框架的整合。

Spring 框架是一个非常优秀的轻量级 JavaEE 容器，大部分 JavaEE 应用，都会考虑使用 Spring 容器来管理应用中的组件，从而保证各组件之间的低耦合。Spring 的核心机制就是 IoC 容器，也可以叫做依赖注入，其含义是：当某个 Java 实例(调用者)需要调用另一个 Java 实例(被调用者)时，代替了传统的在调用者实例当中创建被调用实例(或者使用工厂模式创建)，而是由 Spring 容器来依赖注入，这样就大大减少了类之间的耦合程度。通过依赖注入容器的帮助，Spring 可以采用动态、灵活的方式来管理各种对象、对象与对象之间的具体实现互相透明。

Struts2 提供了两种基本的整合策略，一种策略是将 Action 实例交给 Spring 容器来负责生成，管理，通过这种方式，可以充分利用 Spring 容器的 IoC 特性，提

供最好的解藕；另外一种策略是利用 Spring 插件的自动装配方式，当 Spring 插件创建 Action 实例后，立即将 Spring 容器中对应的业务逻辑组件注入 Action 实例。通过以上两种方式，即可让 Struts2 的 Action 访问到 Spring 容器中的 Bean。

为了在 web 应用中使用 Spring 框架，需要把 Spring 框架的 jar 文件复制到 web 应用中，其中主要包括，Spring.jar—Spring 框架最全面的类库，commons-logging.jar，还有用于整合 Spring 框架的 struts2 插件 Struts2-spring-plugin-X-X-X.jar。除此之外，在使用 Spring 容器前，必须先完成 Spring 容器的初始化。struts2 提供了两种初始化方式。一般利用 ContextLoaderListener，Spring 提供一个 ContextLoaderListener 类，该类可以作为 web 应用的 Listener 使用，它会在 web 应用启动时自动查找 WEB-INF/下的 applicationContext.xml 配置文件(Spring 配置文件)，并且根据该文件来创建 Spring 容器。因此需要在 web.xml 文件中增加如下代码：

```
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
```

如果 Spring 配置文件是根据应用模块来划分的文件，那么就会存在很多 Spring 配置文件，要在应用启动时就加载这些文件，需要在 web.xml 文件中进行配置，例如一个应用需要加载的配置文件有：applicationContext\_conn.xml，applicationContext\_dao.xml，applicationContext\_service.xml，...，那可以在 web.xml 文件中进行如下配置(不过要注意文件加载的顺序)：

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>    WEB-INF/applicationContext_conn.xml    ,    /WEB-INF/
applicationContext_dao.xml, /WEB-INF/ applicationContext_service.xml ,...
</param-value>
</context-param>
```

这样就把应用中的控制器(Action)交由 Spring 容器来管理，因此在配置



struts.xml 文件时, 如果 action 元素的 class 属性还是使用 Action 类的全类名(包名+类名), 那么就会导致配置文件冗余。这时 class 属性内容只能变成 spring 容器提供的该类的 bean 的 id, 即例如以下的例子:

applicationContext\_dao.xml:

```
<beans>
    <bean                                id="shipmentDao"
class="com.allwin.magnesium.hb3.HbShipmentDao">
        .....
</beans>
```

applicationContext\_service.xml:

```
<beans>
    <bean                                id="shipmentService"
class="com.allwin.magnesium.ShipmentServiceImpl">
        <property name="shipmentDao" ref="shipmentDao"/>
    </bean>
    .....
</beans>
```

applicationContext\_action.xml:

```
<beans>
    <bean                                id="shipmentAction"          class=
"com.allwin.magnesium.action.ShipmentAction " scope="request">
        <property name="shipmentService" ref="shipmentService" />
    </bean>
    .....
</beans>
```

struts.xml:

```
<package>
    <action name="shipment" class="shipmentAction" >
```

```
<interceptor-ref name="beforeBasicStack" />
<result name="success">/shipment/shipmentProp.jsp</result>
<result name="error">/index.jsp</result>
<interceptor-ref name="AfterBasicStack" />
</action>
</package>
```

即当客户端调用 shipment.action 时, 请求会把它分发给 name 为 shipment 的 Action, 但是系统一看到其 class 属性提供的是一个伪类—即由 Spring 容器提供的, 那么 Spring 容器会寻找 id 为 shipmentAction 的 bean, Spring 容器会自动创建此类的实例, 其中还会把一个属性名为 shipmentService 的类动态装配到这个 Action 实例中。同样, shipmentService 也是一个伪类, Spring 容器会自动再次寻找 id 为 shipmentService 的 bean, 直到最后可以找到一个可以实例化的 Bean, 最终完成 struts.xml 中 Action 的实例化。当表现业务逻辑的 Action 实例化后, 就会灵活调用相应的处理方法, 例如本例中就会调用 ShipmentAction 中的 execute()方法, 根据方法执行的结果(即返回的对应不同物理视图的字符串), 应用会跳往不同的 jsp 页面。这样就实现了 Spring 框架与 Struts2 框架的整合。

同样, struts.xml 文件也可以包含众多根据模块划分的配置文件, 例如: struts\_tenant.xml, struts\_front.xml, struts\_individual.xml..., 可以同样将众多这些模块化配置文件配置与 struts.xml 文件中。这样不仅大大减小了首次加载的 struts.xml 文件的大小, 并且为各个模块分工提高工作进度提供了更好的选择。

Spring 的工作, 主要是解决对象之间的依赖问题, 实例化 Action 对象, 为其装配相关的类成员变量, 例如将实例化完整的 DAO 装配给 Service, 再将装配好的 Service 装配给 Action。这样 Spring 就很好的解决了各个层之间相互调用增加耦合的缺陷, 只需要负责实现业务逻辑的方法, Spring 负责装配和实例化业务类, 这样各个层之间的耦合性大大减小, 并且实现了业务逻辑的复用。

### 3.2.3 Spring 与 Hibernate 的集成

Hibernate 中用于数据库连接和配置持久化类的配置文件是 hibernate.cfg.xml。

Spring 通过提供相应的类从而使得开发者并不需要手动来管理 Hibernate 会话。在 Spring 应用程序上下文中, Hibernate 通常是通过 Spring 的 LocalSessionFactoryBean 来设置, 引用要使用的 DataSource 以及要载入的 Hibernate 映射文件。通常还有一些已经定义的 Hibernate 属性。但是现在使用的是 Hibernate3 中最具有代表性的 Annotation 进行 O/R 映射, 那么 Hibernate 就会通过 Spring 的 AnnotationSessionFactoryBean 来设置, 如下面的 applicationContext\_global.xml 文件的片段。

```
<beans>

<!-- 配置数据库连接properties文件-->

    <bean                                id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:/resources/jdbc.properties</value>
            </list>
        </property>
    </bean>

<!-- Data source init information -->

<!--调用数据库连接 properties 文件中键值对为初始化 Data source 属性赋值 -->

    <bean                                id="c3p0DataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="{jdbc.driverClassName}"/>
        <property name="jdbcUrl" value="{jdbc.url}"/>
        <property name="user" value="{jdbc.username}"/>
        <property name="password" value="{jdbc.password}"/>
    </bean>

<!-- Transaction manager for a single Hibernate SessionFactory (alternative to JTA) -->
```

```
<bean id="hb3TransactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory"><ref
local="hb3SessionFactory"/></property>
</bean>
<!-- Hibernate SessionFactory -->
<!--配置 Hibernate SessionFactory -->
<bean id="hb3SessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"
">
    <property name="dataSource"><ref local="c3p0DataSource"/></property>
<!--注册使用到 Annotation 映射的 JavaBean-->
    <property name="annotatedClasses">
        <list>
            <value>com.ollwin.model.user.UserRole</value>
            <value>com.ollwin.model.permission.Permission</value>
            ...
        </list>
    </property>
</bean>
</beans>
```

这样就完成了对于数据源的配置，然后 DAO 可以直接根据数据源来调用 `HibernateTemplate` 来持久化处理数据对象。Spring 处于业务逻辑层之上，它负责整个工序的步骤和进度，根据不同的时机和返回的结果，决定下一步的动作。而真正工作的则是 `Hibernate`，它处理持久层数据，更像一个工作者，根据管理者的要求去执行具体的任务，在这里是去执行对数据库的操作，将返回的结果报告给管理者。当 Web 层的 Action 委托业务逻辑层处理单元进行业务处理时，处理的任务可能很复杂，业务逻辑层在把握流程的同时，将具体和数据库打交道的实现委托

给 Hibernate 的持久层来处理, 通过调用 Spring 自带的 HibernateTemplate 方法(要说明一点就是, 模板方法只是对 Hibernate 的操作的封装, 并不能替代 Hibernate), 将对数据的处理结果返回给业务逻辑单元。

#### 3.2.4 SSH 与 Ajax 的集成

在第二章中介绍了 Ajax 框架主要分为两种: 浏览器端框架和服务器端框架。浏览器端框架主要针对 JavaScript 语法进行封装, 这些封装不仅仅减小了实际应用开发的代码量, 而且为兼容各种浏览器提供了可靠的保障。浏览器端框架的集成比较简单, 只需要在使用这些框架的表现页面中引入这些 JavaScript 包, 即可安全使用。例如短小精悍的 prototype.js, 在 jsp 页面或者 html 页面中作如下的引用即可:

```
<html>
  <head>
    <title></title>
    <script type="text/JavaScript" src="/prototype.js"></script>
    ..... //可以继续添加浏览器端 Ajax 框架
  </head>
</html>
```

当然如果在一个页面中要引入许多的浏览器端 Ajax 框架, 可以利用以上方法继续加入, 不过要注意由于 Ajax 框架的开发是由不同组织完成的, 有可能造成方法名的相同, 这样会导致调用方法失效, 一定要多加注意这方面的使用安全。经常使用的 Ajax 浏览器端框架, 例如: prototype.js, ExtJs, JQuery, DoJo 等框架都可以使用这种方式进行配置。

服务器端框架的集成比较麻烦, 但是能够利用表现层页面的 JavaScript 代码来调用服务器端的方法, 增加开发的灵活性和高效性也是值得商榷。服务器端框架也有很多, 但是仅以 DWR Ajax 框架来进行介绍集成。

DWR 会根据 Java 类动态地生成 JavaScript 代码, 通过这些动态生成的 JavaScript 代码, 将服务器端的 Java 方法直接暴露给客户端的 JavaScript 代码, 让

用户产生一种错觉：DWR 可以通过 JavaScript 代码直接调用远程 Java 方法，实际上，DWR 依然依赖 XMLHttpRequest 和服务端通信。其基本原理是，当开发者直接调用远程 Java 方法时，DWR 会负责将这种调用转换成对应的 XMLHttpRequest 请求，并将请求发送到远程服务器端。当服务器处理完成后，DWR 再负责将处理结果传回客户端的 JavaScript 代码。

在整个 Ajax 交互过程中，DWR 负责数据的传递和转换。这种远程的调用从外表看，非常类似于 RMI 或 SOAP 的 RPC 机制。因此，DWR 的 Ajax 交互也被称为 RPC 风格的 Ajax 调用<sup>[46]</sup>。

远程的 Java 方法是同步的，而 Ajax 调用是异步的。因此，当调用一个远程 Java 方法时，应该增加一个回调函数，远程调用的结果成功返回时，这个函数会自动执行，负责将服务器返回的数据显示在当前页面上。

DWR 为服务器端的 AjaxService 类生成了一个相应的客户端 AjaxService 对象，从而将服务器端的 Java 对象暴露给客户端的 JavaScript 对象。当客户端 JavaScript 代码调用这个 JavaScript 对象时，DWR 负责处理整个远程调用的细节。

把 DWR 框架集成到 Web 应用中是非常简单的事情，首先需要把 dwr.jar 文件复制到应用的 WEB-INF/lib 下，然后在 web.xml 文件中进行如下配置：

```
<!--配置 DWR 的核心 Servlet-->
<Servlet>
<!--指定 DWR 核心 Servlet 的名字-->
<servlet-name>dwr-invoker</servlet-name>
<!--指定 DWR 核心 Servlet 的实现类-->
<servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
<!--指定 DWR 核心 Servlet 处于调试状态-->
<init-param>
<Param-name>debug</Param-name>
<Param-value>true</Param-value>
</init-param>
```

```
</servlet>
<!--指定核心 Servlet 的 URL 映射-->
<servlet-mapping>
<servlet-name>dwr-invoker</servlet-name>
<!--指定核心 Servlet 映射的 URL-->
<url-pattern>/leedwr/*</url-pattern>
</servlet-mapping>
```

除此之外还需要另一个重要的配置文件 `dwr.xml` 负责用于 `JavaBean` 和 `JavaScript` 对象之间的转换。可以利用如下的配置文件进行简单的转换：

```
<!--dwr 是 DWR 配置文件的根元素-->
<dwr>
<!--allow 元素是核心元素，用于定义 Java 类和 JavaScript 对象的对应关系-->
<allow>
<create creator= “...” JavaScript= “...” >
<Param name= “...” value= “...” >
</create>
<filter class= “...” >
<Param name== “...” value= “...” >
</filter>
<convert converter= “...” match= “...” JavaScript= “...” >
<param name= “...” value= “...” >
</convert>
</allow>
<!-- signatures 元素列出所有的方法声明-->
<signatures>
<!--列出所有的方法声明-->
</signatures>
</dwr>
```

在上面通用的配置模板中，最重要的元素就是 `allow` 元素。`allow` 元素里两个常用的元素是 `create` 和 `convert`，其中，`create` 用于定义如何将一个 Java 类转换成一个 JavaScript 对象，而 `convert` 用于完成 Java 类和 JavaScript 之间的转换。

通常有如下术语：方法参数会被 `converted`，远程 Java 类会被 `created` 成一个 JavaScript 对象。因此，如果有一个叫 A 的 bean，它有一个方法叫 `A.blah(B)`，那么应该为 A 指定一个 `Creator`(创建器)，并为 B 指定一个 `Converter`(转换器)。

在表现层页面中调用以上创建的转换的 JavaScript 对象时需要引用相应的 js 文件，可以如下引入：

```
<script type="text/JavaScript" src="/应用名/配置的 dwr Servlet 路径/created
JavaScript 对象.js"></script>
```

这样即可在表现层页面中用 JavaScript 对象直接调用服务前端的方法。现在以上所讲的都是使用简单的 `JavaBean` 对象，在 `dwr` 应用中更为重要的是可以直接使用 `Spring` 容器中的 bean。`DWR` 提供了一个 `Spring` 创建器，一旦使用了 `Spring` 创建器，`DWR` 便负责搜索 Web 应用中的 `Spring` 容器，并将 `Spring` 容器中的 `Bean` 转换成一个可以远程调用的 JavaScript 对象。在 `Springbean` 已经在 `Spring` 的配置文件中配置好以后(例如这个 bean 的 id 为“hello“)，那在 `dwr` 应用中使用 `hello` 对象时，可以进行如下配置：

```
<dwr>
<allow>
<create creator= “Spring” JavaScript= “hello” >
<Param name= “beanName” value= “hello” />
</create>
</allow>
</dwr>
```

这样就实现了 `DWR` 服务器端 Ajax 框架与 `Spring` 框架的整合。



### 3.3 SSH+Ajax 组合框架的具体设计

#### 3.3.1 Web 表现层设计

从上面的研究中知道, Struts2 是一个优秀的 WEB 层的解决方案, 利用强大的 OGNL 和值堆栈机制, 为 View 层和 Model 之间实现了完美的数据共享, 基于 AOP 机制的拦截器更为开发人员针对基本业务需求提供了灵活的开发和配置。Struts2 是传统应用请求开发模式最臻成熟的产品, 是 Web 应用表现层技术的集大成者。Ajax 技术改变了传统的开发模式, 从人性化角度和客户满意度出发提出了类似于 C/S 架构的开发模式, 极大地丰富了 web 应用的开发, 提高了 web 应用的开发效率, 增加了开发的灵活度, 最主要的是客户对于应用的满意度得到极大提高。Struts2 框架和 Ajax 框架的联手组合使得现代 web 应用表现层的开发呈现很好的前景。但是它们的工作原理都是将表现逻辑处理委托给 Spring 容器进行管理, 然后由 struts2 的 FilterDispatcher 控制器根据 struts.xml 来寻找和分配执行的物理逻辑视图。在执行表现逻辑处理之前, Spring 容器负责表现逻辑处理 Action 的装配和实例化。

其次, 在应用开发中还是要着重于复用, 这种复用的概念并不局限于某个层次之内, 而是贯穿于整个应用开发的不同层之间, 当然 Web 表现层同样可以复用。由于表现层技术包括 html, jsp 标签, Struts2 标签以及 Ajax 技术等, 为了提高代码的可维护性, 需要把表现层页面分为三种不同的文档, 然后在总的表现层页面中包含进来, 分别为: CSS 文档, 主要负责整个应用的 web 表现风格和管理表现层页面的表现形式。JavaScript 文档, 主要负责每个表现层页面的验证以及利用 Ajax 技术 RIA 显示数据模型。JSPF 文档, 主要用来局部显示表现层页面主要数据内容。有了这三种文档的划分使得在开发应用时细分表现层逻辑处理和表现层页面显示提供了复用的可能。表现层的处理流程如图 3.4 所示。

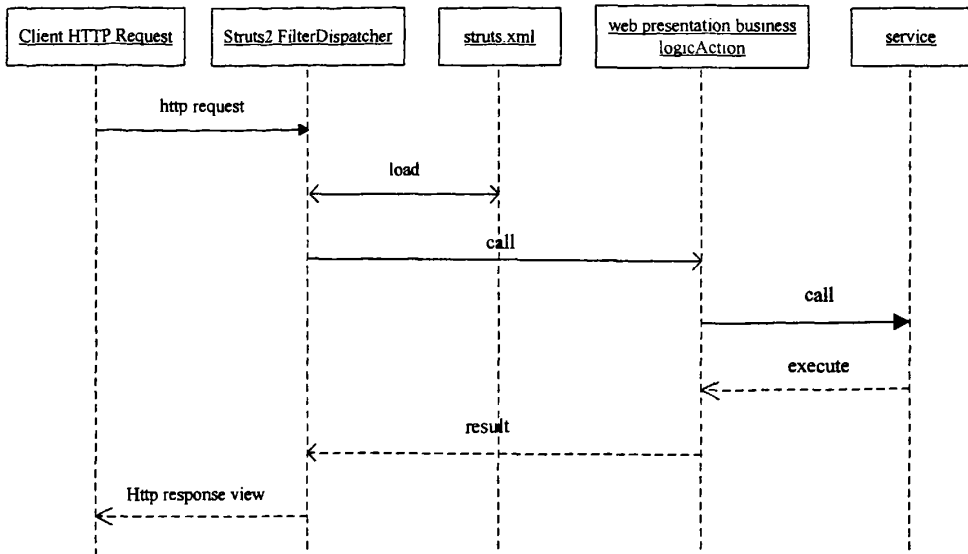


图 3.4 表现层的处理流程图

Fig. 3.4 Disposal flow chart for presentation layer

### 3.3.2 业务逻辑层设计

业务逻辑层负责处理从整个 web 应用数据流程中抽象出来的基本业务逻辑操作处理，它是位于数据持久层之上，是用来被表现业务逻辑 Action 调用的接口层。Spring 框架依赖本身的 IoC 机制和 AOP 机制在整个 web 应用开发中占有一席之地，但是 Spring 框架并不仅仅负责业务逻辑层上的业务逻辑装配，它在整个应用中几乎涉及到了所有层之间。Spring 框架在 web 表现层负责装配和实例化表现业务逻辑 Action，在业务逻辑层负责装配和实例化业务逻辑 Service，并负责进行业务之间的事务处理，在数据持久层负责装配和实例化数据对象 DAO，并调用 HibernateTemplate 来持久化数据对象。从以上可知：Spring 框架通过与 Struts2 框架和 Hibernate 框架的集成，根据 IoC 机制统一实现了数据持久 DAO 和业务逻辑 Service 的装配以及装配对象的实例化。因此业务逻辑层的 Service 方法的实现就来

的更加方便，只需要负责实现基础业务逻辑方法，让 Spring 来统一管理这些 Service，这样不但实现了组件间的松散耦合，还大大加快了开发进度。Spring 框架的优势不止于此，如事务操作、AOP 和集成的持久层实现等给开发人员带来极大的方便。不用拘泥于设计事务机制、繁琐的 try-catch 代码和数据库连接的获取、关闭等。业务层的工作流程如图 3.5 所示。

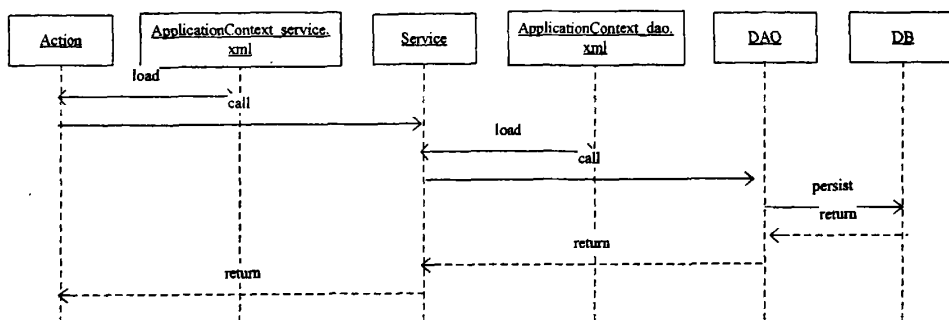


图 3.5 业务逻辑层的工作流程图

Fig. 3.5 Work flow for Business Logical layer

对 Web 层的统一接口表现为 Service。可以在 Service 接口内定义所需业务方法，通过对 DAO 的组装实现业务逻辑。对 O/R 模型对象的调用通过 Spring 的配置文件 applicationContext\_service.xml 的配置得到实现。Spring 支持三种依赖注入——setter、构造函数和方法注入。在本文集成架构中，采用 setter 注入方式对 bean 进行管理。

### 3.3.3 数据持久层设计

数据持久层主要负责域模型层域模型对象的持久化，即把域模型对象存储到数据库中或者从数据库中查询并重新装配为原来的域模型对象。在设计的整个 Web 架构中，Spring 框架集成了 Hibernate 框架。针对于 Hibernate3 Annotation 进行 O/R 映射的特点，Spring 也给出了集成方案，利用 Spring 已经提供好的

HibernateTemplate, 可以非常方便的持久化域模型对象。Spring 统一管理 Hibernate 会话, Hibernate 事务管理等方面, 使得整个开发更加容易, 并且屏蔽了系统在不同数据库之间移植存在的问题。

数据持久层只负责域模型对象的持久化, 即在数据库事物中在基本的 CRUD 基本操作, 并不涉及到基本的业务操作, 它是数据库事物的最基本的操作, 是被业务逻辑层 Service 调用的最基本的数据库操作。数据持久层流程如图 3.6 所示。

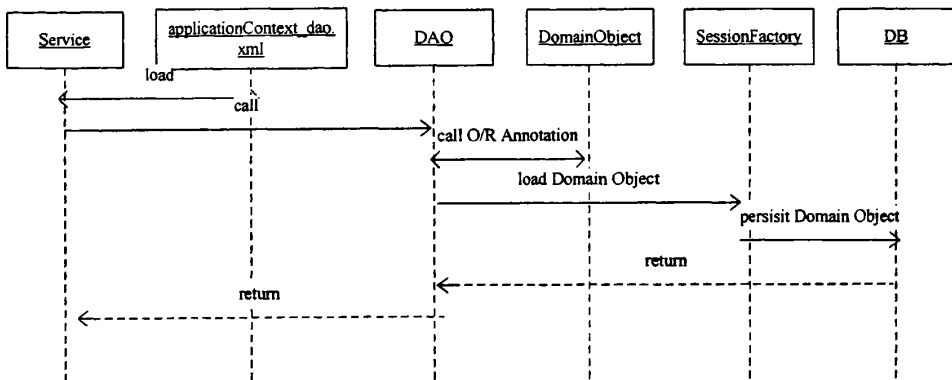


图 3.6 数据持久层的工作流程

Fig. 3.6 Work flow for permanence layer

从上图可以看出, 定义了系统中各个模块的业务入口 Service, 业务逻辑类通过调用 DAO 提供的接口完成业务逻辑处理。所有的 DAO 都继承于 HibernateDaoSupport 类。HibernateDaoSupport 类为进行数据库操作, 封装了必要的方法, 如下:

(1) get/set SessionFactory

获取/设置 SessionFactory, 用来创建 Session 实例。

(2) create/get/set HibernateTemplate

创建/获取/设置 HibernateTemplate, 而 HibernateTemplate 封装了对数据库的基本操作: 增加, 读取, 更新, 删除。

#### (3) get/release Session

获取/释放 Session(连接)利用 HibernateDaoSupport 类来获得 SessionFactory, Session 等。对 O/R 模型对象的调用通过 Spring 的配置文件 Service.xml 的配置得到实现。

#### 3.3.4 域模型层设计

领域层是实现对象的持久性封装,是根据 web 应用需求分析产生的数据封装对象,对应于 java 应用中的 class。一个域对象(Domain Object)实际上是一个具有 Setter/Getter 风格的 JavaBean,它代表一个实体对象。对于其它层而言,一个 Domain Object 相当于数据库表中的一条记录。持久层具体的 DAO 类将通过 Getter/Setter 来访问对象的持久化数据。每一个 Domain Object 必须继承 Serializable(序列化)接口。Serializable 接口没有任何的方法或属性,它只是声明了序列化接口。序列化是把一个对象的状态写入一个字节流的过程。只有一个实现 Serializable 接口的对象可以被序列化工具存储和恢复。

这样以 Struts2, Spring3, Hibernate3 和 Ajax 框架整合而成的一个非常成熟的架构贯穿于 web 应用开发的层次之间,Struts2 和 Ajax 框架负责 web 表现层的丰富表现, Spring3 框架贯穿于整个层次之间,负责 SpringBean 的装配和实例化, Hibernate3 框架以其独特的 Annotation 特点简化了数据持久化的开发。这样的设计中, Struts2 以其强大的标签库和基于 MVC 架构的控制转发,再加上基于 Spring AOP 机制的 Interceptor,非常成熟的解决了表现层的表现技术。Ajax 技术弥补了传统应用开发的同步请求模式,创建了客户端 JavaScript 代码异步发送请求开发模式,不仅使得应用开发更加灵活方便,而且客户满意度大大提高。Spring 框架依赖于 IoC 机制,非侵入性的特点实现了 Spring Bean 的装配和实例化,使得各个层次之间代码调用实现解耦, Hibernate3 框架以其独特的方式解决了数据对象的 O/R 映射,屏蔽了底层数据库选择的不同所造成的问题,更加简化了数据对象持久化的开发。整个应用开发从横向上可以划分为 web 表现层,业务逻辑层,数据持久层和领域模型层,从纵向上可以依据应用不同的需求分析划分为不同的模块,这样应用开

发就可以依据横向和纵向的划分，实现分工合作，大大提高了开发的速度。

但是，整合 Struts2, Spring3, Hibernate3 和 Ajax 框架的架构也并非完美无缺，如果使用工具没有一个统一的规划，那么就会造成滥用工具的危险。因此在使用整个框架集成的架构中，开发者需要遵守一定的使用工具的规范，这样才能够统一管理和使用开发代码，使应用系统的维护性提高。因此开发者在进行开发中一定要有一个统一的开发规划规范。这个规范须以整个应用的安全性为前提，例如：过度使用 Ajax 异步请求技术就有可能造成系统安全性方面的危害，因此开发者在开发使用 Ajax 技术时，一定要以安全性为前提，不要在浏览器客户端暴露任何有关系统安全信息。除了在使用整个框架时考虑安全性以外，开发者需要对 MVC 和分层技术有一定了解，才能正确的将该架构应用于在企业系统开发中。

## 第 4 章 基于 SSH+Ajax 框架的 ERP 设计与实现

本章通过大连鑫轮模具公司具体的 ERP 软件的设计与实现,详细分析了 SSH+Ajax 架构在应用中的使用方法,同时通过具体模块来分析 SSH+Ajax 架构的优势。并且结合当前 ERP 的发展状况,分析了使用这种多层集成架构的可扩展性和可维护性以及灵活性。本项目使用的开发环境为: Struts2, Spring3, Hibernate3 以及 ExtJs 框架。

### 4.1 项目简介

中小企业度过初创期后,多数已经有了稳定的产品和项目,当企业进入成长期后,原有的管理模式已不再适应企业内外环境的变化,迫切需要借助信息化手段在企业内部形成更为规范的决策、执行与监督制衡的治理结构。故而,ERP 不再是大企业的专利,它已经越来越受到众多中小企业的青睐<sup>[47]</sup>。

大连鑫轮模具公司是一家主要以铁裙为生产对象的中小型企业,在经过企业的初创期以后,公司规模得到了迅速的发展,涉及到的信息量也越来越多,通过手工作业已经不能够有效快速的完成生产周期,因此公司决定开发一套根据自己业务建立起来的 ERP 系统。作为一家生产企业,这个应用涉及的业务方面就包含生产的各个阶段,分别为:计划,采购,质量,库存,生产,产品,销售,报表,人力资源等。因此公司的 ERP 系统开发就是根据生产的各个阶段来划分为不同的开发模块,分别为:产品管理,销售管理,采购管理,库存管理,计划管理,质量管理,生产管理,模具管理,报表分析,人力资源管理,售后服务管理等。

虽然 ERP 是一种以管理思想为主导的应用开发,但是它也是基于企业生产业务流程上的一种管理重组<sup>[48]</sup>,因此本文着重于 ERP 应用使用本文提出的多层架构 SSH+Ajax 在生产流程上的业务实现,并不着重于 ERP 的精髓——生产流程重组。因此,根据企业的生产流程,整个应用的业务结构图如图 4.1 所示。

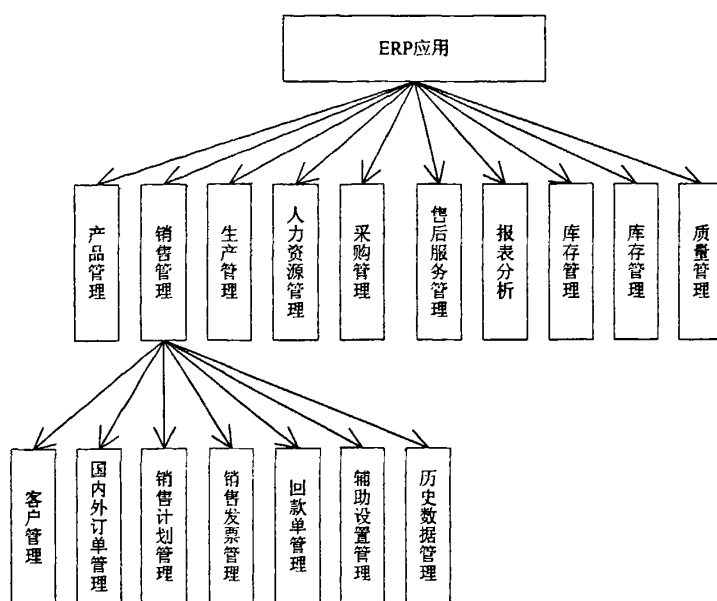


图 4.1 ERP 业务结构

Fig.4.1 Logical structure for ERP

在企业的整个业务过程中，销售管理是位于企业的核心业务而存在的，它是企业创造利润的直接渠道。由于 ERP 应用涉及太多的生产业务，因此本文只选择这个 ERP 应用的销售管理子模块来作为引导，具体来介绍多层架构的具体应用与实施，进而探讨使用此多层架构的优异性。

根据图 4.1，销售管理子模块在业务上进行具体划分，可以分为：客户管理，国内外销售订单管理，销售计划管理，销售发票管理，回款单管理，相应的辅助数据设置管理和历史数据管理。其中客户管理负责对于企业的销售对象进行管理，国内外销售订单管理负责当客户要和企业签订销售订单时的业务管理，销售计划管理负责对于企业内每个年度的预测性计划销售，销售发票管理负责企业对于销售产品创建发票进行管理，回款单管理负责企业的销售最终现金流的回收进行管理，辅助数据设置管理负责销售区域、客户组和杂费等项目的初始化设置进行管理，历史数据管理负责查看销售订单生命周期已结束的数据管理。在销售管理



子模块中,着重利用本文 SSH+Ajax 多层架构介绍国内外销售订单管理业务的需求与实现。

### 4.2 国内外销售订单管理的需求分析

#### 4.2.1 国内外销售订单管理业务需求

(1)可以根据销售订单诸多属性,例如:序号,客户名称,销售代表和类型等属性进行模糊查询。并且根据销售订单条目进行销售订单的详细信息查看。

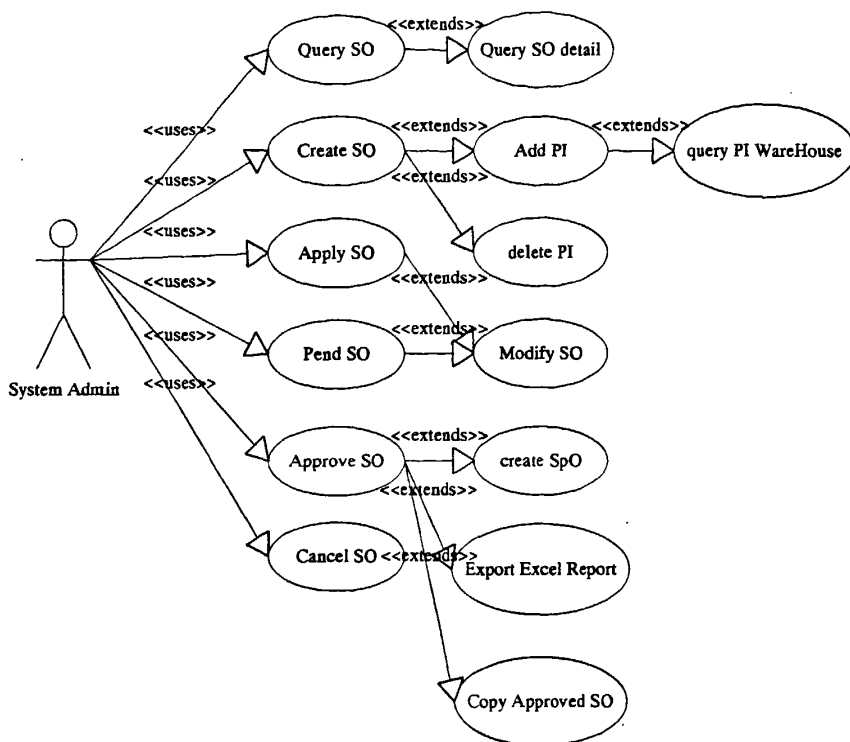
(2)可以新建销售订单,销售订单可以添加销售产品条目,并且根据销售产品条目可以查看产品库存明细,根据销售产品明细来决定是否添加杂费项目,销售订单有三个状态,分别为:草案,确认和审核,经过不同的领导进行确认和审核后,销售订单生效。

(3)销售订单审核以后,可以根据销售订单来生成出库单,也可以根据销售订单导出 Excel 报表,还可以复制一份销售订单进行存档,在销售订单处于草案和确认两个状态期间,可以对销售订单进行修改。

(4)销售订单经过确认之后,需要同步到查询订单条目之中。

#### 4.2.2 系统用例图与活动图

ERP 系统是针对在企业业务中不同的执行人员而设计的权限不同的系统。也就是说:不同的业务人员需要执行不同的业务进行。例如:销售人员可以创建销售订单的草案,销售地方部经理可以确认销售订单,而销售总经理最终审核整个销售订单。但是针对于本系统的开发人员来讲,本系统根据业务的执行操作进行了权限系统的合理设计,因此在开发人员来讲,可以把系统用力人员定为一个,即系统的管理者,系统的用例<sup>[49-51]</sup>如图 4.2 所示。

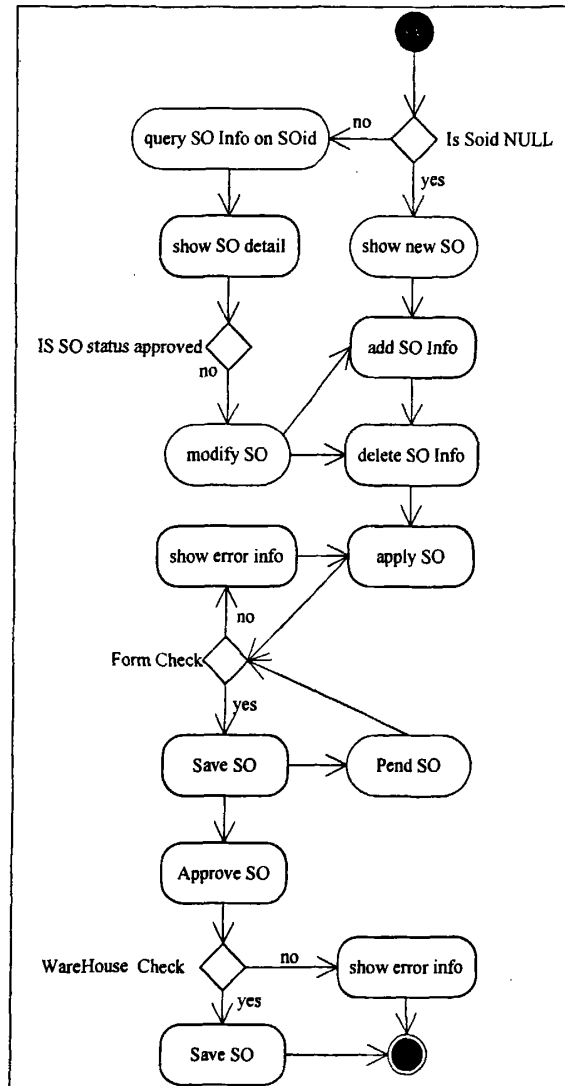


注 1) : SO: Sales Order(销售订单), PI: Product Item(产品条目), SpO: Shipment Order(销售出库单)

图 4.2 国内外销售订单用例模型图

Fig.4.2 Use case chart for Sales Orders

系统管理员可以根据查询条件来查询已存在销售订单，并且根据订单条目查询详细的销售订单信息，也可以创建一个全新的销售订单，在销售订单详细页面可以根据销售订单状态来进行销售订单的录入数据，应用、确认和审核销售订单，因此可以分析订单状态可以分为四种：新订单，草案，确认和审核，因此可以根据销售订单的状态销售订单页面设置为一个页面，和根据不同的状态来具体显示不同的信息，这样不仅达到了表现层页面的复用，而且使得整个开发进度得到大大提高。这里分别以创建新销售订单和根据订单条目查询具体销售订单信息为例来绘制活动图如图 4.3 所示。



注 2): SO: Sales Order(销售订单)

图 4.3 销售订单活动图

Fig. 4.3 Activity chart for SO

当然通过确定的销售订单会同步到销售订单查询页面，系统管理员可以根据

销售订单的诸多要素对销售订单进行模糊查询。

### 4.3 系统的实现

本文所提出的 SSH+Ajax 多层架构集成,都是轻量级的开源框架整合。由于采用 Spring 框架技术,使得应用各层之间耦合度大大降低;采用的 Struts2 web 表现层框架更加成熟了应用的表现形式, Ajax 技术从客户端发出异步请求的形式丰富了应用的表现形式,增加了客户的满意度。Hibernate 框架屏蔽了数据持久层的诸多问题。这些框架在关注自身实现的同时,只要在各自配置文件中指定连接属性,就可以实现相互关联。在应用系统需求分析和系统设计的基础上,本节主要通过具体的实例代码,详细分解了 SSH+Ajax 架构在系统中的实现,同时通过实例分析了 SSH+Ajax 架构的优势,如何体现 SSH+Ajax 四种框架之间的紧密配合,如何体现四种框架的灵活性、兼容性、可扩展性等等,并抽象出可供重复应用的组件。在当前 Web 多层框架设计中就 SSH+Ajax 框架所具有的优势进行深入探讨,为实现高效、灵活的多层 Web 体系结构在实践中的应用提出了一种新的解决方案。根据详细设计和系统架构图,从数据持久层、业务逻辑层和表现层(模型层、视图层和控制器层)三部分分别加以实现,详细分析 SSH+Ajax 架构的应用。

系统的分层模型如图 4.4 和图 4.5 所示,其中,图 4.4 是表现层与业务层的分层模型图,图 4.5 是业务层与持久层的分层模型图。

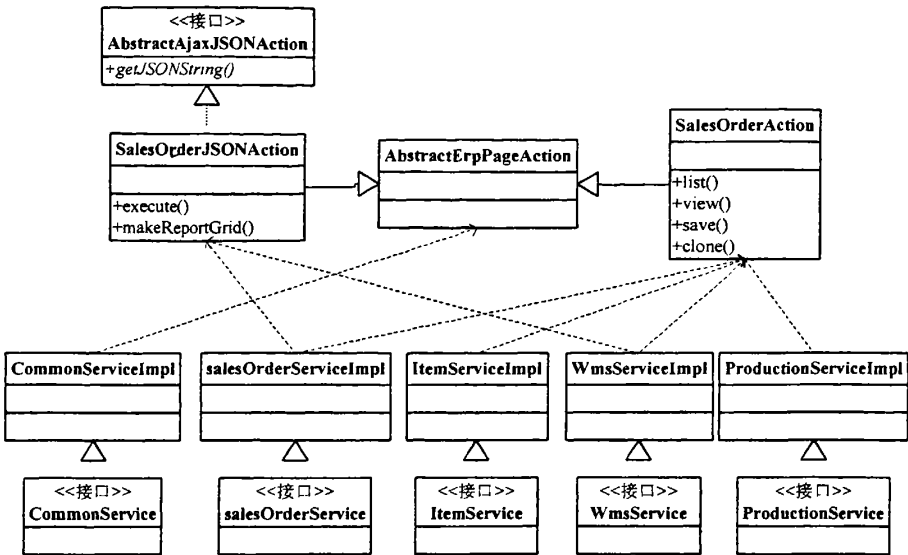


图 4.4 表现层与业务层的分层模型图

Fig. 4.4 Delamination chart for Presentation Layer and Business Logical Layer

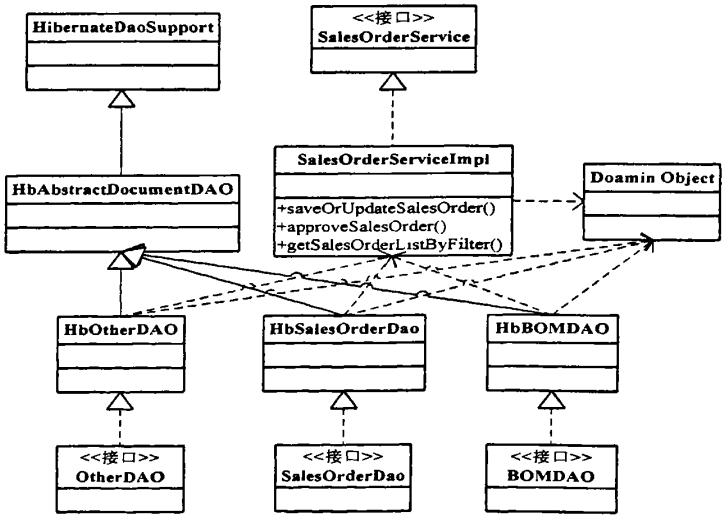


图 4.5 业务层与持久层的分层模型图。

Fig. 4.5 Delamination chart for Business Logical Layer and Persisence Layer

#### 4.3.1 表现层的实现

在表现层使用 Struts2 实现 MVC 模式，使用 ExtJs 框架来搭建表现形式，就新建并录入销售订单信息页面来说，从 MVC 三部分加以说明：

V：视图层：

视图层根据复用原则，也可以分为三部分：CSS 文件，JavaScript 文件和表现内容(Content)文件。由于 CSS 文件用于控制表现内容的格式，本节将省去 CSS 文件的说明。

表现内容文件(Content File)：

```
<%@include file="/_includes/commonStruts2JspHead.jspf" %>
<html>
<head>
<title><s:text name="%{getText('SalesOrder.Order')}" /> ${so.sequence}</title>
<%@include file="/_includes/commonStruts2Js.jspf" %>
<script type="text/javascript">
<%@include file="_includes/_salesOrder.js" %>
<%@include file="_includes/_OH_TabsScript.js" %>
</head>
<body>
<form name="form" method="post">
<%@include file="/_templates/_abstractPageStrutsHiddenFields.jsp"%>
<input type="hidden" id="lineJson" name="lineJson" value="">
<input type="hidden" name="id" value="${so.id}" />
<input type="hidden" name="otherRequest" value="${so.otherRequest}" />
<div id="divWinHeader"><s:text name="%{getText('SalesOrder.Order')}" />
${so.code} </div>
<div id="divWinContent">
<div id="divTabs"></div>
<fieldset id="divOrderLine">
<legend><s:text name="%{getText('SalesOrder.Line')}" /></legend>
```

```

<div id="divGrid" class="gridContainer"></div>
<div id="divGridCtrl" class="gridCtrl">
    <html:btnButton2 type="button" image="/common/images/icon/icon16_add.gif"
imgClass="icon" value="" id="btnAddLine" onclick="addLine()"/>
    <html:btnButton2 type="button"
image="/common/images/icon/icon16_subtract.gif" imgClass="icon" value=""
id="btnRemoneLine" onclick="removeLine()"/>
</div>
</fieldset>
<%@include file="_includes/_OL_tradeReveivable.jspf" %>
<%@include file="_includes/_OL_verify.jspf" %>
</div>
<div id="divWinFooter">
<div id="divActionBtns">
    <%@ include file="_includes/popupmenus.jspf" %>
    <%@ include file="_includes/popupmenuFunctionImpl.jspf" %>
    <html:btnButton2 id="btnModify"
image="/common/images/icon/icon16_eraser.png"
text="${erp:getResStr3(requestScope.session,'Com.Order.Modify')}}"
onClick="page_action('START_MODIFY')"/>
</div>
    <html:btnButton id="btnOk"
text="${erp:getResStr3(requestScope.session,'Com.Ok')}}"
onClick="page_action('OK')"/>
    <html:btnButton id="btnClose"
text="${erp:getResStr3(requestScope.session,'Com.Cancel')}}"
onClick="page_action('CLOSE')"/>
    <html:btnButton id="btnApply"
text="${erp:getResStr3(requestScope.session,'Com.Apply')}}"
onClick="page_action('APPLY')"/>

```

```
</div>
</form>
</BODY>
</HTML>
脚本文件(Javascript File):
function page_OnLoad() {
    initTabStrip();
    init_ConextMenu();
    initWarehousePagingSelect();
    initDatePicker();
    manageButtons(true);
    initExtGrid();
    manageFields();
}
function initExtGrid() {
    Ext.Ajax.request({
        url:
"/ajax/salesOrderJSON/execute.action?pageType=soLine&pageMode=header",
        success: extGrid_onreloadHeader
    });
}
var extgrid;
function extGrid_onreloadHeader( request, response, options) {
    var ds = Ext.util.JSON.decode(request.responseText);
    _extGrid_create(ds);
    extgrid.store.load({params:{beginIndex:0,pageSize:50},          callback:
extGridLoad_callback});
}
function _extGrid_create(ds) {
```



```
var fieldsArr = ext2_dsMapping(ds);
var store = new Ext.ux.DataSetStore({
    url:
'/ajax/salesOrderJSON/execute.action?pageMode=&pageType=soLine&id='+document.
getElementById("id")[0].value,
    root:"rows",
    fields:fieldsArr,
    totalProperty: 'totalRecordSize',
    id:"cells[0].value"
});

var cm = new Ext.grid.ColumnModel(ds.cols);
cm.defaultSortable = true;
ds.cols[5].summaryRenderer =function(v){
    return "<s:text name=\"%{"+getText('BomLine.total')+"}/>"+(" "+v+"");
};

extgrid = new Ext.grid.EditorGridPanel({
    ds: store,
    cm: cm,
    renderTo:'divGrid',
    clicksToEdit:1,
    sm: new Ext.grid.RowSelectionModel({singleSelect:true}),
    plugins:new Ext.ux.grid.GridSummary(),
    height:getContentHeight() *0.4,
    width:getWindowWidth()-10,
    loadMask: true
});
});
}
```

```
extgrid.getColumnModel().on('columnmoved', setModifiedLineCSS);
}
function page_action(name) {
    if(name != 'CLOSE') {
        document.getElementsByName("so.requestedDeliveryDate")[0].value=formatDate
(requestedDelivery_Date.value);
        // validate
        if(!validate()) {
            return;
        }
        document.getElementById("lineJson").value=modifiedRecordsToJson(extgrid);
        if (name == 'Pending') {
            form.action="/sales/salesOrder/pending.action";
            page_OnSubmit();
        }else if (name == 'APPLY') {
            form.action="/sales/salesOrder/save.action";
            page_OnSubmit();
        } else if (name == 'Approve') {
            form.action="/sales/salesOrder/approve.action";
            page_OnSubmit();
        } else if (name=='START_MODIFY'){
            form.action="/sales/salesOrder/stepToModifyOrder.action";
            page_OnSubmit();
        }
        } else {
            window.close();
        }
    }
}
```

以上是所有视图页面的部分代码，其中页面需要调用国际化文档中的 key 值，

需要使用 Struts2 国际化标签 `<s:text name="%{getText('key')}" />`<sup>[52]</sup>, 而且在整个内容页面中嵌入了 jspf 文档, 其中 jspf 文档是用来描述销售订单某一部分的部分表现文档, 主要是为了各个页面的复用, 开发者可以在任何一个使用此部分文档的页面调用此部分文档。整个表现层页面为了增加客户的体验, 先利用客户端代码加载 ExtGrid 头部信息, 然后再加载 ExtGrid 内容信息, 从而达到良好的客户体验, 这个增加客户体验客户端方法由 `initExtGrid()` 方法来实现。当销售订单填写完毕, 有客户端的表单验证方法 `validate()` 来进行验证, 然后进行表单的提交。页面表单的提交统一由 `page_action()` 方法来进行提交。

#### C: 控制器层

Struts2 使用前端控制器 `FilterDispatcher` 接收用户请求后, 通过读取配置文件 `struts_sales_model.xml`, 找到对应的 Action 类, `struts_sales_model.xml` 中部分配置如下:

```
<?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="sales" namespace="/sales" extends="struts-default-mg">
        <action name="*/*" class="{1}Action" method="{2}">
            <interceptor-ref name="myParamsPrepareParamsStack"/>
            <result name="list">/magnesium/order/so/{1}List.jsp</result>
            <result name="view">/magnesium/order/so/{1}Prop.jsp</result>
        </action>
    </package>
</struts>
```

如果在表单中应用销售订单, 也即保存生成销售订单的草案, 那么会把表单数据保存到数据库中, 并在读取出来显示在当前表单页面上, 也即进入 `salesOrderProp.jsp`。如果确定表单数据, 则会保存数据后然后关闭页面, 并把销售

订单同步到销售订单查询页面(salesOrderList.jsp)。

M: 模型层

由于销售订单页面调用了客户端请求加载, 因此销售订单的模型层是存在多个表现业务逻辑 Action 的, 为了更加清晰程序调用过程, 本节只节选调用方法生成 ExtGrid 的 JSONAction 进行说明:

```
public class SalesOrderJSONAction extends AbstractErpPageAction implements
AbstractAjaxJSONAction {
public String execute() {
    if (!Strings.isEmpty(pageType)&&pageType.equals("ORDER_HEADER")) {
        makeOrderHeaderGrid();
    } else {
        makeOrderLineGrid();
    }
    return AJAXSUCCESS;
}

private void makeOrderHeaderGrid() {
    DataSet ds = new DataSet();
    DataSetCol dsColumn = new DataSetCol("id", "id");
    dsColumn.hidden = true;
    ds.cols.put(dsColumn);
    dsColumn = new DataSetCol("sequence", getText("Com.Sequence"));
    dsColumn.w = 80;
    ds.cols.put(dsColumn);
    dsColumn = new DataSetCol("code", getText("SalesOrder.Code"));
    dsColumn.hidden = true;
    dsColumn.w = 130;
    ds.cols.put(dsColumn);
    . . .
    jsonString = ds.toString();
}
```

效果如图 4.6 所示。

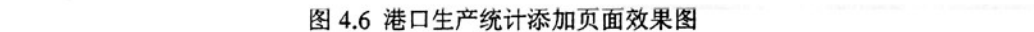


Fig.4.6 View for ship produce

业务逻辑层是整个应用的核心，接收表现层的用户请求，并通过持久层完成数据的存取操作<sup>[53]</sup>。就保存统计数据而言，先调用接口

```
public interface SalesOrderService {
    public void saveOrUpdateSalesOrder(SalesOrder salesOrder);
}
```

```
public class SalesOrderServiceImpl implements SalesOrderService {
    private SalesOrderDao salesOrderDao;

    public void setSalesOrderDao(SalesOrderDao salesOrderDao) {
```

```
this.salesOrderDao = salesOrderDao;
}
public void saveOrUpdateSalesOrder(SalesOrder so) {
    salesOrderDao.saveOrUpdateSalesOrder(so);
}
```

saveOrUpdateSalesOrder 方法通过调用 salesOrderDao 实现，而 salesOrderDao 通过 Spring 容器进行配置，实现了 dao 和 Service 的解耦。开发者可以通过 applicationContext\_sales.xml 配置：

```
<bean id="salesOrderService" class="com.service.SalesOrderServiceImpl">
    <property name="salesOrderDao"><ref bean="salesOrderDao"/></property>
    <property name="bomDAO"><ref bean="bomDAO"/></property>
    . . .
</bean>
```

#### 4.3.3 数据持久层的实现

数据持久层是系统与数据库进行交互的层，它直接负责域模型对象的持久化操作，即域模型对象的增删查改等最基本的数据库操作，本框架通过 Hibernate3 提供的独有的 Annotation 映射来进行 O/R 映射，Hibernate3 利用反射技术和接口技术提取出映射对象与数据库进行映射的属性，然后利用底层的 JDBC 技术来持久化映射对象<sup>[54]</sup>。由于 Spring 框架很好的封装了 Hibernate 会话，可以采用 Spring 封装的 HibernateTemplate 来进行对象的持久化，为了解耦 DAO 和 Service，开发者一般采用接口技术，即在 Service 中使用接口来实例化 DAO 实现类。因此，在销售订单 Action 中 SalesOrderService 就是调用 SalesOrderDao 来实现 SO 的保存和更新。由于 HbAbatractDocumentDAO 实现了 HibernateDaoSupport，所以只要继承自 HbAbatractDocumentDAO 的 DaoImpl 都可以使用 Spring 提供的 getHibernateTemplate()方法来持久化域模型对象。具体实现代码如下：

```
Public interface SalesOrderDao{
    public void saveOrUpdateSalesOrder(SalesOrder so)
```

```

    }

    public class HbSalesOrderDao extends HbAbstractDocumentDAO implements
SalesOrderDao {

        public void saveOrUpdateSalesOrder(SalesOrder so){
            if(so == null) throw new IllegalArgumentException("");
            try{
                getHibernateTemplate().saveOrUpdate(so);
            }catch(Exception ex){
                String errorMessage = o.toString();
                throw new DBException(errorMessage, ex);
            }
        }
    }

```

HbSalesOrderDao 类要使用 SessionFactory 工厂类实例变量，这个类对象由 Spring 容器来管理，通过注入方式供实现类使用。在 applicationContext\_sales.xml 中，配置文件

```

<bean id="salesOrderDao" class="com. dao.document.HbSalesOrderDao">
    <property name="sessionFactory"><ref bean="hb3SessionFactory"/></property>
</bean>

hb3SessionFactoryBean 的定义在初始配置文件(也可以放在同一文件)中定义。

<bean                                id="hb3SessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean
">

    <property name="dataSource"><ref local="c3p0DataSource"/></property>
<!--注册使用到 Annotation 映射的 JavaBean-->
    <property name="annotatedClasses">
        .... <list>
            <value>com.ollwin.model.user.UserRole</value>
            <value>com.ollwin.model.permission.Permission</value>
            ...

```

```
</list>
</property>
</bean>
```

#### 4.3.4 域模型层实现

在诸多域模型对象中，本节选择了 `SalesOrder` 作为域模型对象进行映射的代表，根据在第二章对于 `Hibernate3` 进行的 Annotation 映射，`SalesOrder.java` 的具体实现代码如下：

```
@Entity
@Table(name = "magnesium.sales_order")
@Proxy(lazy = true)
@PrimaryKeyJoinColumn(name="sales_order_id")
public class SalesOrder extends AbstractDocument {
    @ManyToOne(optional=false,fetch = FetchType.EAGER)
    @JoinColumn(name="customer_id")
    private CustomerRelationship customer;

    @ManyToOne(optional=true,fetch = FetchType.EAGER)
    @JoinColumn(name="sales_person_id")
    private MagnesiumEmployeeRelationship salesPerson;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="currency")
    private Currency currency;

    @Column(name = "requested_delivery_date")
    private Date requestedDeliveryDate;

    @Column(name="nation_category")
    @Type(type="nationCategory")
```



```

private NationCategory nationCategory;

@Column(name="delivery_mode")
@Type(type="deliveryMode")
private DeliveryMode deliveryMode;

@Column(name="document_group")
@Type(type="documentGroup")
private DocumentGroup documentGroup;

... //同样其它属性的 Annotation 映射
... //诸多属性的 get 和 set 方法
}

```

从以上的部分代码可以清楚的看到：类 `SalesOrder` 继承了抽象类 `AbstractDocumen` 并实现了 `serializable`(序列化)接口，序列化允许将实现了 `serializable` 接口的对象转换为字节序列，这些字节序列可以被完全存储以备以后重新生成原来的对象<sup>[55]</sup>。所以，域对象与其它类相区别的一个地方就是要实现 `Serializable` 接口，只有实现了该接口，域对象才能持久化。

从以上各个层次的开发过程来看，开发者可以根据需求分析来获得域模型对象，通过分析域模型对象之间的关系，开发者可以使用 `Annotation` 技术来划定域模型对象之间的关系，并且把它们之间的关系同步映射到数据库中，因此，开发者在域模型对象层所做的开发任务就是确立需求分析中的个体，确定个体之间的关系，并且依从 `Hibernate` 映射规则来书写映射规则；域模型层确立以后，持久层所做的工作也只是书写各个对象的 `Dao`，并且实现 `Dao`，使用 `Spring` 框架提供的 `HibernateTemperlate` 来持久化各个持久化对象，最后把 `Dao` 配置入 `Spring` 提供的配置文档；当数据持久层实现完毕，开发者可以根据需求分析中的具体业务来确定业务逻辑层的数据操作方法<sup>[56]</sup>，依次书写相应的 `Service` 和其实现，然后通过 `Spring` 提供的配置文档把 `Dao` 配置入 `Service` 中；业务逻辑层实现完毕，开发者就

可以借助 Struts2 提供的 MVC 开发模式来成熟表现视图层，表现业务逻辑 Action 可以通过调用 Service 来进行复杂的业务操作，因此还需要利用 Spring 的配置文档来把 Service 配置入各个 Action，同时还需要利用 Struts2 的配置文档 struts.xml 把 Action 的抽象视图(不同的字符串)与物理视图相对应。因此从上面分析中可以看出：开发者通过利用本文提供的多层架构，极大地从复杂的开发逻辑中解脱出来，只要按照各自的层次要求书写相应的文档，即可完美的完成应用的开发，分层的实现使得开发者可以无论从那个层都可以进行开发，并且形成共同协作，大大减少了开发时间，提高了开发效率，而且通过使用开源的 Ajax 框架大大提高了客户的体验。

## 第 5 章 总结与展望

随着 web 技术的飞速发展,越来越多的 web 应用采用轻量级集成框架来进行架构,从最初的基于 MVC 模式的 Struts,到后来趋于成熟的 WebWork,都曾经为 web 应用的敏捷开发打下了坚实的基础<sup>[57]</sup>。随着软件理论的发展以及应用开发的实践,应用中代码可重用度低,维护任务繁重、扩展性和兼容性差等问题困扰着开发人员,如何快速选择轻量级框架,并且把它们集成起来构建稳定、高效、可扩展性和维护性强的应用系统是众多开发人员追求的目标。

本文针对以上问题,结合作者开发经验,认为 web 的开发可以从两个方向来进行划分,横向上可以把 web 应用逐级分成一个个开发模块,以利于分工协作,当然也可设置优先度级别,纵向上可以从应用架构来进行分层,分别为 web 表现层、业务逻辑层、数据持久层和域模型层。Struts2 是基于 MVC 框架的集大成者,非常成熟的解决了 web 表现层的灵活交互, Spring 贯穿于整个业务逻辑层,为业务逻辑的层之间解耦提供了有力工具, Hibernate 更为面向对象的数据持久提供了简便易行的方案, Ajax 框架的出现弥补了 web 传统开发的缺陷,大大加强了客户与应用的交互和开发者应用开发的灵活度,提高了客户体验。而且分层的应用和框架的集成为软件开发带来了高度的重用性,同时它也需要更高技巧的设计。架构是由不同领域的框架组成,因此架构的建立及维护其稳定性,是构建高质量软件产品关键<sup>[58]</sup>。

面对传统开发越来越难以适应现代化软件开发的要求。本文通过对 JavaEE 框架和 MVC 模式的分析,通过对比 MVC 的实现技术,集成 Struts2、Spring、Hibernate 框架和 Ajax 框架形成了一个具有一定集成度和通用性的 SSH+Ajax 架构,并结合大连鑫轮模具公司 ERP 项目的开发,具体阐述了如何使用 Struts2 和 Ajax 框架作为 Web 表现层框架,而将业务逻辑委托给以 Spring 为框架的业务逻辑层处理,最后将对数据的操作交给以 Hibernate 为框架的持久层来完成。通过分层处理使整个系统结构清晰,功能明确,更重要的是各层次之间相互独立,对某一层次的修改

不会影响其它层次。通过运用集成框架，使层次间的耦合性降到最低，这就为软件的重用化和组件化创造了条件。

工作中的不足及展望：

尽管论文已经完成了预期的研究目标，但是还有一些工作需要进一步完善：

1. 随着 Web 应用技术的发展，更优秀的应用开发技术不断涌现，在以后的学习中要不断深入对组件、复用等相关技术的理解<sup>[59]</sup>，力求把本文的设计和原先已有的开发技术和将来要出现的开发技术集成到本框架来，更加满足实践要求的架构技术。

2. Struts2 作为基于 MVC 模式的集大成者，虽然提供了强大的表现技术，但是仍然没有跳出传统 web 应用的开发模式，Ajax 技术弥补了这个缺陷，但是不同的应用需要选择不同的 Ajax 框架，这就为 Struts2 与不同的 Ajax 框架集成提出一个挑战。

3. Spring 框架技术所涵盖的范围非常广泛，不仅仅提供了依赖注入和模板化封装 Hibernate 框架的特性，还实现了与 ACEGI 安全框架的无缝结合<sup>[60]</sup>，这就对于应用开发来讲是极大的应用安全措施，Spring 还提供了远程访问的支持，这些技术都值得进一步学习和研究。

4. Spring 虽然模板化了 Hibernate 框架，但是对于应用相对复杂或者对于开发数据访问性能和效率有较高要求的应用，开发这就需要慎重选择是否使用 Spring 模板化了的 JdbcTemplate 技术，Spring 提供的连接池和缓冲技术能够极大提高系统的响应和数据处理速度，在以后的学习中，将进一步予以关注，并深入研究和探索。

综上所述，通过本论文，深入研究了 JavaEE 架构技术，并提出了集成 Struts2 Spring 和 Hibernate 的架构设计方案，通过大连鑫轮模具公司 ERP 项目予以验证，并在某些方面有所创新，延伸出一些新的特性，为实现高效、灵活的多层 Web 体系结构的应用提供一种新的思路及方案。

## 参考文献

- [1] 李伟, 吴庆海. 软件架构的艺术[M]. 北京: 电子工业出版社, 2009.
- [2] 温昱. 软件架构设计[M]. 北京: 电子工业出版社, 2007.
- [3] 阎宏. Java 与模式[M]. 北京: 电子工业出版社, 2002.
- [4] Raghu R. Kodali, Jonathan Wetherbee 著, 马朝晖, 杨艳译, EJB 3 基础教程[M]. 北京: 人民邮电出版社, 2008.
- [5] 陈思淼. 基于轻量级 J2EE 的 Ajax Web 框架的设计与实现: (硕士学位论文). 成都: 电子科技大学, 2007.
- [6] 何成万, 余秋惠. MVC 模式 2 及软件框架 Struts 的研究[J]. 计算机工程, 2002.
- [7] 卫素琪. 基于 MVC 模式的一种 Web 应用框架[M]. 北京: 北京工业大学, 2003. 5.
- [8] 艾拉玛等, 马树奇译. J2EE 编程指南[M]. 北京: 电子工业出版社, 2002.
- [9] Patrick Lightbody 等著, 谭颖华等译. WebWork in Action[M]. 北京: 电子工业出版社, 2006.
- [10] 李刚. Struts2 权威指南-基于 WebWork 核心的 MVC 开发[M]. 北京: 电子工业出版社. 2007. 9.
- [11] 宋彦儒, 周翔. 基于 Struts2 框架的 Web 系统安全模型分析[J]. 南昌: 中国学术期刊电子出版室, 2009.
- [12] Rod Johnson, Juergen Hoeller 等著, 蒋培译. Spring 框架高级编程[M]. 北京: 机械工业出版社, 2006.
- [13] Donald Brown, Chad Michael Davis, Scott Stanlick. Struts2 in Action[M]. America: Manning Publications Co, 2008.
- [14] 侯国赵, 李彦斌等. Struts2 的应用研究[J]. 中国学术期刊电子出版室, 2009.
- [15] 李刚. Spring2.0 宝典[M]. 北京: 电子工业出版社, 2006.
- [16] <http://www.jActiongroup.net/reference/html/>.
- [17] Howard M. Lewis Ship. Spring in Action[M]. Softbound, 2005.
- [19] Spring 开发手册. <http://www.jActiongroup.net/reference/html/>.
- [20] <http://www.jdon.com/AOPdesign/Ioc.html>.
- [21] 埃克尔, 候捷译. Java 编程思想[M]. 北京: 机械工业出版社, 2002.

- [22] [http://www.mx68.com/WebDeveloper/2006-03-10/WebDeveloper\\_38712\\_3.shtml](http://www.mx68.com/WebDeveloper/2006-03-10/WebDeveloper_38712_3.shtml).
- [23] <http://hi.baidu.com/godblessewan/blog/item/ed286c8bba942f12c8fc7ada.html>.
- [24] 吴波. 基于 Webwork+Spring+Hiberante 框架的 Web 应用的研究与实现: (硕士学位论文). 大连: 大连海事大学, 2007.
- [25] <https://www.hibernate.org/>.
- [26] 葛京. Hibernate3 和 Java Persistence API 程序开发 从入门到精通[M]. 北京: 清华大学出版社, 2007.
- [27] Rima Patel Sriganesh, Gerald Brose, Micah Silverman 等著, 罗时飞译. 精通 EJB3.0[M]. 北京: 电子工业出版社, 2006.
- [28] Ed Roman. 刘晓华等译. 精通 EJB(第二版)[M]. 北京: 电子工业出版社, 2002.
- [29] Cary E. Umrysh, 康博译. 用 J2EE 和 UML 开发 Java 企业级应用程序[M]. 北京: 清华大学出版社, 2002.
- [30] Crawford, 刘绍华译. J2EE 设计模式[M]. 北京: 中国电力出版社, 2005. 4.
- [31] [http://www.hibernate.org/hib\\_dots/v3/reference/en/html/](http://www.hibernate.org/hib_dots/v3/reference/en/html/).
- [32] Christian Bauer, Gavin King. Hibernate in Action. America[M]: Manning publications Co, 2004.
- [33] 丁娜. 基于 Ajax 的 web2.0 技术研究: (硕士学位论文). 杭州: 浙江大学, 2007.
- [34] 李业田. Ajax 技术在 J2EE 中的研究与应用: (硕士学位论文). 武汉: 武汉理工大学, 2008.
- [35] 施伟伟, 张蓓. 征服 Ajax 框架解析与实例[M]. 北京: 人民邮电出版社, 2007.
- [36] 施伟伟, 冯梅. 征服 JavaScript 高级程序设计与应用实例[M]. 北京: 人民邮电出版社, 2007.
- [37] <http://www.prototypejs.org>.
- [38] 卫军, 夏慧军, 孟腊春. ExtJS Web 应用程序开发指南[M]. 北京: 机械工业出版社, 2009.
- [39] 李刚. 基于 J2EE 的 Ajax 宝典[M]. 北京: 电子工业出版社, 2007.
- [40] <http://dojotoolkit.org/>.
- [41] Joshua Kerevsky 著. 杨光译. 重构与模式[M]. 北京: 人民邮电出版社, 2006.
- [42] Alan Shalloway 著. 徐言声译. 设计模式解析[M]. 北京: 人民邮电出版社, 2006.
- [43] Allen Holub 著. 徐迎晓译. 设计模式初学者指南[M]. 北京: 机械工业出版社, 2006.
- [44] 张敏. 基于 J2EE 多种架构特术的 Web 应用与实现: (硕士学位论文). 成都: 电子科技大学, 2007.

- [45] Barbara Liskov 著. 裘健译. 设计模式解析[M]. 北京: 电子工业出版社, 2006.
- [46] <http://getahead.org/dwr>.
- [47] 玄承浔. 中小企业 ERP 公共服务平台的分析与构建: (硕士学位论文). 济南: 山东大学, 2008.
- [48] 林飞. ERP 销售管理系统的研究与设计: (硕士学位论文). 哈尔滨: 哈尔滨工业大学, 2007.
- [49] Mike O'Docherty 著. 俞志翔 译. 面向对象分析与设计(UML 2.0 版)[M]. 北京: 清华大学出版社, 2006.
- [50] Jim Arlow 著, 方贵宾译. UML2.0 和统一过程[M]. 北京: 机械工业出版社, 2006.
- [51] Michael Blaha, James Rumbaugh .UML 面向对象建模与设计[M]. 北京: 人民邮电出版社, 2006.
- [52] 刘增才, 李晓霞, 原小龙, 郭力. 基于架构的化学数据知识框架管理[J]. 北京: 计算机与应用化学, 2008.
- [53] 艾里特著, 刘平利译. 精通 Hibernate[M]. 北京: 机械工业出版社, 2009.
- [54] Minter.D., Linwood 著, 陈剑瓯 等译. Hibernate 基础教程[M]. 北京: 人民邮电出版社, 2008.
- [55] 艾克尔著. java 编程思想[M]. 北京: 机械工业出版社, 2004.
- [56] 宋成明. 基于 Spring、iBATIS 与 Struts 的轻量级 javaEE 编程研究(硕士学位论文). 上海: 上海师范大学, 2007.
- [57] 赫姆瑞甲尼著, 韩坤, 徐琦译. java 敏捷开发-使用 Spring、Hibernate 和 Eclipse[M]. 北京: 人民邮电出版社, 2007.
- [58] 李伟著. 软件架构的艺术[M]. 北京: 电子工业出版社, 2008.
- [59] 李伟, 吴庆海著. 架构之美-软件架构的艺术[M]. 北京: 电子工业出版社, 2009.
- [60] 郭峰著. Spring 从入门到精通[M]. 北京: 清华大学出版社, 2006.

## 攻读学位期间公开发表论文

[1] 吴波, 张升文, 王向兵. 国际信息技术与应用论坛(第三卷), An E-commerce System Structure Research Based on WSH (Webwork, Spring, Hibernate). 2009: ISBN 978-0-7695-3600-2: 14-18.



## 致 谢

时光荏苒，转眼就到了毕业的时间。在这两年的时间里，我收获很多，这和一直关心我、帮助我的人是分不开的。是他们陪伴我渡过了这段美好难忘的时光，相信这段岁月是我人生道路中最宝贵的记忆。

本论文是在导师张升文教授的悉心指导下完成的。特别感谢导师张升文教授两年多来辛勤的教诲和指导，使我能够顺利完成学业和科研任务。张老师学识渊博、思维敏锐、勇于创新、作风严谨、精益求精、敬业忘我、大度儒雅，这些无不给我留下了极为深刻的印象，并将使我在今后的发展中受益非浅。课题研发和论文撰写期间，张老师给了我很多帮助，本文从立题、撰写初稿到最终定稿，都离不开他的精心指导。谨向恩师致以最诚挚的感激和最由衷的敬意！

感谢大连奥文公司朋友们给我的帮助，谢谢他们让我了解了 web 应用技术的魅力，他们始终保持着踏实的工作精神和极强的求知欲，尤其是整个团队的团结协作精神让我感动。在他们身上我学到很多，在相互交流中，我们都有很大的进步，在此我要深深地感谢他们。借此机会祝愿全体老师和同学们在事业和学业上取得更大的进步！

最后，将深深感激对我寄予厚望的父母亲，是亲人们的无私支持和关怀给了我前进的动力，谢谢他们！

## 研究生履历

姓 名	王向兵
性 别	男
出生日期	1982 年 9 月 11 日
获学士学位专业及门类	
获学士学位单位	大连海事大学
获硕士学位专业及门类	管理科学与工程 管理学
获硕士学位单位	大连海事大学
通信地址	辽宁省大连市凌海路 1 号
邮政编码	116026
电子邮箱	14233087@163.com

作者: [王向兵](#)  
学位授予单位: [大连海事大学](#)  
被引用次数: 9次

## 参考文献(2条)

1. [卫索琪](#) [基于MVC模式的一种Web应用框架](#)[学位论文] 硕士 2003
2. [刘增才](#), [李晓霞](#), [袁小龙](#), [郭力](#) [基于SSH+ExtJS架构的化学数据知识框架管理](#)[期刊论文]-[计算机与应用化学](#) 2008 (09)

## 本文读者也读过(10条)

1. [孙璐](#) [基于SSH2的企业人事管理系统研究与实现](#)[学位论文]2010
2. [于东超](#) [基于Struts2\\_Spring\\_Hibernate三种框架的通用Web框架的研究及应用](#)[学位论文]2008
3. [申海杰](#) [基于Struts2+Spring3+Hibernate3+Ajax的DRP系统](#)[学位论文]2010
4. [邓斯红](#) [基于AJAX和SSH集成框架的国有资产管理信息系统](#)[学位论文]2010
5. [陈莉](#) [基于Struts2框架的应用研究](#)[学位论文]2008
6. [钱蓉蓉](#) [基于Struts2框架的企业协同办公系统的设计与实现](#)[学位论文]2007
7. [张洪梅](#) [面向Struts2框架的模型驱动开发方法研究](#)[学位论文]2009
8. [李绍平](#), [彭志平](#), [LI Shao-ping](#), [PENG Zhi-ping](#) [S2SH: 一种Web应用框架及其实现](#)[期刊论文]-[计算机技术与发展](#) 2009, 19 (8)
9. [武宝珠](#), [梁声灼](#), [牛德雄](#), [WU Bao-zhu](#), [LIANG Sheng-zhuo](#), [NIU De-xiong](#) [基于Struts2+Spring+Hibernate架构构建Web应用系统](#)[期刊论文]-[计算机与现代化](#)2009 (8)
10. [叶长春](#) [基于MVC的Struts框架的应用研究](#)[学位论文]2008

## 引证文献(9条)

1. [周叶菲](#) [基于SST的档案管理系统的研究与实现](#)[期刊论文]-[科技传播](#) 2012 (06)
2. [蒋材星](#) [档案管理系统的设计与实现](#)[学位论文] 硕士 2010
3. [雷明强](#) [基于SST的档案管理系统的研究与实现](#)[学位论文] 硕士 2009
4. [袁渡](#) [铝合金材料数据管理系统的设计与实现](#)[学位论文] 硕士 2014
5. [田燕](#), [梁晶晶](#), [张新刚](#), [贾松浩](#) [多层框架在远洋船务系统中的应用](#)[期刊论文]-[测控技术](#) 2013 (06)
6. [刘妍东](#) [基于J2EE的通用问题管理平台设计与实现](#)[学位论文] 硕士 2011
7. [赵海](#) [中小实体书店网上书店的设计与实现](#)[学位论文] 硕士 2011
8. [陈晓东](#) [基于SSH框架的银行信贷管理系统设计与实现](#)[学位论文] 硕士 2014
9. [王凌川](#) [面向MES的工艺信息管理系统开发](#)[学位论文] 硕士 2012

引用本文格式: [王向兵](#) [JavaEE多层架构Struts2+Spring3+Hibernate3+Ajax的整合](#)[学位论文] 硕士 2009