第 36 卷 第 4 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.4 Computer Engineering February 2010
·软件技术与数据库· 文章编号：1000—3428(2010)04—0059—04 文献标识码：A 中图分类号：N945.15

基于 Spring 的 MVC 框架设计与实现

张 宇[1]，王映辉[1,2]，张翔南[2]
(1. 陕西师范大学计算机科学学院，西安 710062；2. 西安理工大学计算机科学与工程学院，西安 710048)

摘 要：为构建高效、灵活且易于使用的模型-视图-控制器(MVC)模式实现机制，提出一种基于 Spring 框架的 MVC 框架。该框架使用 JSP 技术构建视图层，采用总体控制与局部控制相结合的方法增强控制层的功能，采用 O/R Mapping 类完成数据访问操作。基于 IOC/AOP 技术实现事务管理，并应用该框架开发网络收藏夹。

关键词：模型-视图-控制器模式；Spring 框架；控制反转；面向方面编程

Design and Implementation of MVC Framework Based on Spring

ZHANG Yu[1], WANG Ying-hui[1,2], ZHANG Xiang-nan[2]
(1. School of Computer Science, Shaanxi Normal University, Xi'an 710062;
2. Faculty of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048)

【Abstract】 In order to build the mechanism of Model-View-Controller(MVC) pattern implementation that is efficient, flexible and easy to use, this paper gives a MVC framework based on Spring framework. This framework builds view layer by using JSP technology, enhances the function of control layer by the methods of combining overall control and local control and uses O/R Mapping class to achieve data access operation. It implements transaction management based on IOC/AOP technology and applies the framework to the development of network bookmark.

【Key words】 Model-View-Controller(MVC) pattern; Spring framework; inversion of control; aspect-oriented programming

1 概述

目前流行的应用程序大多基于模型-视图-控制器(Model-View-Controller, MVC)模式进行设计与实现[1-2]，包括 Web 程序框架。轻量级 Spring 框架提供了对 MVC 模式的实现机制，它主要围绕分发器进行设计，包括可配置的处理器映射、视图解析、本地化、主题解析、支持文件上传等[3]。该机制具有角色划分清晰、可重用业务代码等优点，但在具体应用与开发中，其实现难度大、配置复杂，且没有提供实用的数据持久化方法。

Spring 框架提供了数据持久化机制[3]，其实现方式分为 2 种，一种以对象关系映射(Object Relational Mapping, ORM)工具为基础，另一种则对 Java 数据库连接(Java Data Base Connectivity, JDBC)类进行封装和抽象。前者能有效解决面向对象与关系数据库之间的不匹配问题，但必须依赖 ORM 工具。后者虽然简单易用，但不能有效解决面向对象与关系数据库之间的不匹配问题。

鉴于此，本文对 Spring 框架中的 MVC 实现机制进行扩展研究，提出一种新的 MVC 模式实现框架，论述其组成结构和运行机理，并以网络收藏夹为例阐述各组成部分的实现过程和方法。

2 框架结构与运行机理

2.1 框架总体结构

本文给出的 MVC 模式实现框架如图 1 所示，包括视图层、控制层、业务逻辑层、数据访问层和数据存储层 5 个部分。

视图层是用户与系统交互的接口。在 Web 应用中，视图层主要由各种 Web 页面组成。在该实现框架中，视图层主要由 JSP(JavaServer Pages)页面构成。

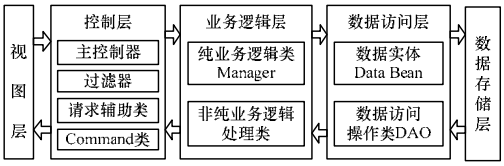

图 1 基于 Spring 的 MVC 模式实现框架

控制层实现 MVC 中控制器的功能，其主要是接收用户请求，依据不同的请求，调用对应的业务处理对象来执行业务逻辑，获取执行结果，并依据当前的状态数据及业务逻辑的处理结果，选择适合的视图组件返回给用户[1]。在该实现框架中，控制层由主控制器、过滤器、请求辅助类和命令处理类 Command 组成。

主控制器是框架中的总体控制类，所有符合要求的用户请求都要先转发到该类进行处理，再由该类分发给各个部分控制类进行处理。

过滤器用于在用户请求被处理前或处理后完成数据的编码转化、数据加密、身份验证、数据压缩、日志记录等工作。用户请求由主控制器接收处理之后，先交由过滤器进行处理，然后转发给具体的业务处理类进行业务处理，处理结果返回给用户之前，可依据实际需要由过滤器对处理结果进行修改。

请求辅助类用于完成处理用户请求的辅助工作，其一方面封装用户请求信息和配置参数信息供命令处理类使用，另

基金项目：陕西省科技基金资助项目(2007F51, 2008K4-11)；西安市创新支持计划基金资助重点项目(CXY080030)；教育部博士点基金资助项目(20070700002)
作者简介：张 宇(1982－)，男，硕士研究生，主研方向：软件工程；王映辉，教授、博士、博士生导师；张翔南，学士
收稿日期：2009-07-26 **E-mail：**zhangyu_288@163.com

一方面解析配置文件，将用户请求提交给具体的命令处理类进行处理，并提供获取业务逻辑对象的方法。

命令处理类 Command 负责处理页面逻辑动作，页面中的每个动作，如登录、注册等对应一个具体的 Command 类。Command 类调用请求辅助类提供的方法获取业务逻辑对象，利用业务逻辑对象来处理页面逻辑动作，依据处理结果选择视图层页面返回给用户。

业务逻辑层是应用系统中的核心，负责处理系统具体的业务逻辑。在该实现框架中，业务逻辑层由纯业务逻辑类 Manager 和 AOP 处理程序组成。Manager 类负责完成与具体业务密切相关的业务逻辑处理，AOP 处理程序负责权限管理、事务管理、日志管理等非纯业务的逻辑处理。

数据访问层主要是封装业务处理逻辑与数据存储层之间的交互过程，向业务逻辑层提供数据服务。在具体的应用开发中，数据访问层有多种不同的实现方式。在该实现框架中，数据访问层由数据实体 Data Bean 和数据访问类 DAO 组成。

数据实体主要用于在各层之间传递数据信息。在该实现框架中，数据实体扮演着多种角色：作为 POJO(Plain Old Java Object)，通过数据访问类中的 O/R 映射功能将数据持久化到关系数据库中；作为 Value Bean 在各层之间进行传输数据信息；作为 View Object 将数据传输到 JSP 页面并进行显示；作为 BO(Business Object)与 Manager 类的属性部分和行为部分相对应。

数据访问类负责完成数据的存取操作，实现与数据存储层的交互。数据访问类基于 Java 的反射机制和 JDBC 提供的数据访问对象进行实现，先利用反射机制构建 O/R 映射关系，形成标准的 SQL 语句，然后使用 JDBC 提供的数据访问对象完成数据访问操作。

数据存储层用于存放应用系统所要处理的数据信息，一般采用关系型数据库进行存储。由于框架中采用数据访问层隔离了数据存储层，因此数据存储层可选用不同的数据库。

2.2 框架运行机理

框架中各组成部分在运行过程中的调用关系见图 2。

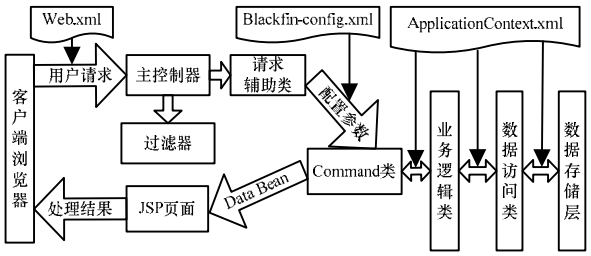

图 2 框架各组成部分的调用关系

用户通过浏览器向服务器发送请求，服务器解析 Web.xml 文件，将用户请求转发给主控制器，主控制器将用户请求交给过滤器进行处理。主控制器委托请求辅助类解析用户请求中的 URL 地址，请求辅助类查找 Blackfin-config.xml 文件，将用户请求转发给对应的命令处理类 Command。命令处理类 Command 获取请求辅助类传递过来的用户请求和配置参数信息，创建业务处理逻辑类的实例(业务对象)，并调用业务对象处理具体的用户请求。业务对象依据业务规则处理用户请求，期间若需要与数据存储层进行交互，则由调用数据访问类 DAO 完成，业务对象将请求处理结果返回给 Command 类。Command 类依据配置信息将处理结果转发给视图层的 JSP 页面。JSP 页面将取来的数据结合页面信息生成 HTML 文件返回给客户端浏览器。

3 框架具体实现

网络收藏夹是用于管理用户所需保存的网页和网址信息的工具，其主要功能包括用户管理、网络书签管理等。下文结合网络收藏夹中的用户管理功能来说明新框架中各组成部分的具体实现方法。

3.1 视图层的实现

视图层的实现主要是依据具体的功能需求构建 JSP 页面的过程。

3.2 控制层的实现

3.2.1 主控制器的实现

主控制器的实现包括构建和配置 MainController 类 2 个部分。MainController 类继承于 HttpServlet 类，主要实现 Init(), doPost(), doGet(), processRequest()等方法。Init()用于初始化公共资源、服务、过滤器和系统配置参数等。doPost()和 doGet()用于接收用户请求，并交由 processRequest()方法进行统一处理。processRequest()用于统一处理用户请求。

配置 MainController 类主要是在配置文件 Web.xml 中注册主控制器对象和设置主控制器所作用的 URL 样式，前者使用<servlet>元素，后者使用<servlet-mapping>元素。

3.2.2 过滤器的实现

过滤器 Filter 由 Filter 实现类和管理类组成，前者负责实现具体的过滤功能，后者负责初始化注册的 Filter 实现类。

Filter 实现类是 Javax.servlet.Filter 接口的实现类，包括 Init(), destory()和 doFilter()3 个方法。Init()用于初始化过滤器，参数为 FilterConfig 对象，destory()用于销毁过滤器，释放使用完的资源，doFilter()用于实现具体的过滤处理逻辑。

Filter 实现类构建完成之后，在配置文件 Web.xml 中注册过滤器对象和设置过滤器对象作用的 URL 样式，前者使用<filter>元素，后者使用<filter-mapping>元素。

3.2.3 请求辅助类的实现

请求辅助类 RequestHelper 主要包括 getAttribute()和 getBean()方法。前者用于封装 Request, Response 和配置参数信息并传递给命令处理类，后者用于获取 ApplicationContext.xml 中配置的业务对象实例。

请求辅助类解析配置文件 Blackfin-config.xml，将用户请求提交给相应的命令处理类进行处理。

在如下所示的配置中：

```
<command-mappings><command path="/command/login" scope="request"
    type="test.command.LoginCommand" >
    <forward    name="success" path="main" />
    <forward    name="fail" path="login.jsp"/>
    </command></command-mappings>
```

RequestHelper 将用户请求的 URL 地址与 Path 值进行比较，若请求的 URL 与 Path 值相匹配，则将用户请求提交给 type 值指定的 test.command.LoginCommand 命令处理类进行处理。

3.2.4 Command 类的实现

Command 类针对页面逻辑中的每个动作进行实现，包括建立页面动作与命令处理类之间的映射关系和构建命令处理类 2 部分内容。

下文以网络收藏夹中的命令处理类 test.command.showTestListCommand 为例，说明具体的实现过程和方法。

先在配置文件 Blackfin-config.xml 中进行配置，将用户

通过页面 Userlist.jsp 发送的显示所有用户信息的页面请求动作映射到 test.command.showTestListCommand 进行处理，并设置不同处理结果所返回的视图层页面。然后结合业务处理过程构建 test.command.showTestListCommand 命令处理类，读取用户信息返回给视图层页面。

建立页面动作与命令处理类的映射关系的代码如下：

```
<command  parameter="execute"
path="/command/showTestListCommand"
type=" test.command.showTestListCommand" scope="request" >
<forward  name="success"
path="/jsp/userlist.jsp" />
<forward name="fail" path="/jsp/fail.jsp" />
</command>
```

构建命令处理类的代码如下：

```
public class ShowTestListCommand
  implements Command{
public String execute(RequestHelper helper) throws
ApplicationException {
//创建 testManager 实例
TestManager testManager=(TestManager)
helper.getBean("testManager")
//调用 testManager 读取所有用户信息
ArrayList list = testManager.getUser(new Test());
//保存获取的用户信息到显示页面
helper.setAttribute("userlist", list);
//读取跳转页面信息
String successPage =
helper.getCommandConfig().getPath("success");
return successPage; //跳转到指定页面}}
```

3.3 业务逻辑层的实现

3.3.1 纯业务逻辑类的实现

纯业务逻辑类 Manager 与具体业务密切相关，需依据特定的业务处理过程进行实现，其实现步骤如下：

(1)依据业务功能需求定义 xxxManager 原型接口方法，每个接口方法处理一项具体的业务需求，如添加用户等。

(2)依据特定的业务处理过程编写 xxxManagerImpl 类实现 xxxManager 中的接口方法。在接口实现过程中，若需要与数据存储层进行交互，则调用数据访问类 DAO 中的数据访问方法完成。若现有方法不能满足应用需求，则可依据实际情况编写新类继承 DAO 类，并在新类中添加所需的方法。每个 xxxManagerImpl 类都要实现 setxxxDao(Dao dao)方法，用于将管理和调用数据访问类 DAO 的工作交由 Spring 框架来完成。

(3)在 ApplicationContext.xml 文件中进行配置：为 xxx ManagerImpl 类配置<bean>对象和注入数据访问对象，使用声明式事务机制实现对 Manager 类的事务处理。

3.3.2 事务 AOP 的实现

非纯业务的逻辑处理是基于控制反转(Inversion of Control, IOC)机制和 AOP 技术进行实现的。

控制反转 IOC 是一种用于控制业务对象之间依赖关系的机制，其将设计的类以及类之间的关系都交由外部容器进行管理，仅需调用类在容器中注册的名字就可以得到类的实例，有效降低了业务对象之间的依赖程度，实现了业务对象之间的松散耦合。

AOP 是一种全新的编程技术，通过为开发者提供横切关注点描述和动态注入机制来实现横切关注点的模块化，解决了横切关注点代码分散产生的问题。

Spring 框架的核心机制是 IOC 和 AOP[3]，为实现各种非纯业务的逻辑处理提供了有力支撑。使用 Spring 中的声明式事务处理机制为实现事务管理提供了有效途径。

Spring 中实现声明式事务的主要步骤如下：首先使用 Spring 中默认的事务类创建事务管理对象，然后使用声明代理类 TransactionProxyFactoryBean 为组件配置事务代理对象，该配置包括 3 个属性 transactionManager, transactionAttributes 和 target，分别用于指向事务管理对象、设置事务管理的细节和指向具体的事务管理目标对象。通常需要在 transaction Attributes 中添加<props>的列表，每个<prop>元素表示一个权限设置，key 使用通配符表示目标对象中的函数名，属性值表示匹配到的函数的操作权限。

上文阐明了纯业务逻辑类和事务处理的实现方法，下文以网络收藏夹中负责用户管理的纯业务逻辑类 UserManager 为例说明具体的实现过程：

(1)依据用户管理的功能需求定义 UserManager 的原型接口方法，代码如下：

```
void setTestDAO(DataAccessDAO dao)
boolean updateUser(String username,String passwd,String email)
boolean addUser(String username,String passwd,String email)
boolean deleteUsers(String[] userName)
…
```

(2)结合用户管理的处理过程编写 UserManagerImpl 类实现接口方法。

(3)在 ApplicationContext.xml 文件中对 UserManager 类进行配置，具体如下：

1)配置 Manager 实现类<bean>对象

```
<bean  id="userManagerTarget"  class="test.service.impl.User
ManagerImpl"><property name="testDAO">
<ref local="testDAO" />//注入数据访问对象
</property></bean>
```

2)配置事务管理对象

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransacti
onManager" >
<property name="dataSource">
<ref local="dataSource"/></property></bean>
```

3)配置事务代理对象

```
  <bean id="userManager"
class="org.springframework.transaction.interceptor.Transaction
ProxyFactoryBean">
<property name="transactionManager">
<ref local="transactionManager" />
</property>
<property name="target">
<ref local="userManagerTarget"/></property>
<property name="transactionAttributes">
<props><prop key="get*">
PROPAGATION_REQUIRED,readOnly,-ApplicationException</
prop>
…//其他代码略去
```

```
</props></property></bean>
```

3.4 数据访问层的实现

3.4.1 Data Bean 的实现

Data Bean 是对数据库表的抽象，包括与数据库表字段相对应且名称一致的成员变量以及设置、读取变量值的 set 和 get 方法，是符合固定规则的 Java Bean。

为确保 Data Bean 的准确性和快速生成，可采用工具自动批量生成。先依据数据库模型生成 XML Schema 文件，然后通过 Castor 工具生成 Data Bean。在生成 XML Schema 文件时，一种方法是先采用 Power Designer 工具生成物理模型，然后由物理模型自动生成 XML 模型，最后由 XML 模型生成 XML Schema 文件。另一种方法是使用 XML Spy 工具依据数据库中的物理表反向生成 XML Schema 文件。

在网络收藏夹中，数据库表 User 包括 3 个字段，即 USERNAME, PASSWD, EMAIL，则与其相对应的 Data Bean 描述如下：

```
String USERNAME;
String PASSWD;
String EMAIL;
Void setUSERNAME(String USERNAME);
void setPASSWD(String PASSWD);
void setEMAIL(String EMAIL)
String getUSERNAME();
String getPASSWD();
String getEMAIL();
```

3.4.2 数据访问类 DAO 的实现

数据访问类 DAO 的实现包括构建 O/R Mapping 类、构建数据访问工厂类和配置数据源及相关信息 3 部分内容。O/R Mapping 类的构建步骤如下：先采用 Java 中的反射机制获取 Data Bean 中的属性及属性值，生成操作单个数据表的标准 SQL 语句，然后使用 JDBC 提供的数据访问对象完成与数据库的交互。数据访问工厂类提供在运行时读取数据源和创建数据访问对象的方法。配置信息是指在 ApplicationContext.xml 文件中对数据源、O/R Mapping 类和数据工厂类进行相关配置以及为 O/R Mapping 类设置代理对象。

网络收藏夹中的数据访问类在 ApplicationContext.xml 中的部分配置信息如下：

(1)配置数据源

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName">
<value>com.mysql.jdbc.Driver</value>
</property>
<property name="url">
<value>jdbc:mysql://127.0.0.1:3306/demo?characterEncoding=gbk</value>
</property>
<propertyname="username">
<value>root</value></property>
<property name="password">
<value>123</value></property></bean>
```

(2)配置事务管理对象

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
<property name="dataSource">
<ref local="dataSource"/></property></bean>
```

(3)配置数据源的工厂类

```
<bean class="mvc.dataaccess.DbFactory"id="dbFactory">
<property name="dataSource">
<ref local="dataSource" /></property>
</bean>
```

(4)指定 DAO 的实现类，并为其注入数据工厂对象

```
<bean id="testDAOTarget" class="test.dao.impl.DataAccessDAOJDBC"><property name="dbFactory">
<ref local="dbFactory" /></property></bean>
```

(5)指定 Spring 框架来代理 DAO 接口类，并将其关联到 DAO 的实现类

```
<bean id="testDAO" class="org.springframework.aop. framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>test.dao.DataAccessDAO</value>
</property>
<property name="target">
<ref local="testDAOTarget"/></property>
</bean>
```

4 结束语

本文框架利用 Spring 框架的特性将业务逻辑对象和数据访问对象交由 IOC 容器管理，降低了对象之间的依赖程度。采用 IOC/AOP 技术将日志管理、事务处理等横切关注点与具体的业务逻辑代码进行分离，增加了代码的可读性，易于实现重构。利用反射原理，通过构建 O/R Mapping 类生成标准的 SQL 语句，实现数据访问操作，便于开发人员学习和掌握。

实践表明，该框架简单易用，具有较高灵活性和可扩展性，对构建中小型 Web 应用具有良好的支撑作用和借鉴意义，但其功能有待改进和完善。

参考文献

[1] 刘　宁, 陆荣国, 缪万胜. MVC 体系架构从模式到框架的持续抽象进化[J]. 计算机工程, 2008, 34(4): 107-110.

[2] 王映辉, 王英杰, 王彦君, 等. 基于 MVC 的软件界面体系结构研究与实现[J]. 计算机应用研究, 2004, 21(8): 188-190.

[3] Johnson R, Hoeller J, Arendsen A. Spring, Java/J2EE Application Framework[EB/OL]. [2008-05-14]. Http://static.springframework.org/spring/docs/2.0.x/reference/index.html.

编辑　陈　晖