第4章 Struts,Spring,Hibernate 互整合

前三章的主要目的在分别介绍 Struts,Spring,Hibernate 技术，而本章的主旨在于讲解如何将这三个组件两两组合构成一个完整的应用。本章的各小节都将基于一个简易的个人博客示例，同时结合 MyEclipse 开发工具图文并茂的阐述开发过程。

4.1 Spring 与 Hibernate 整合

个人博客主要实现的功能是：本人登陆后可以在日志栏中浏览所有日志，添加日志， 编辑日志，和删除日志。而其它游客只能浏览日志。下面介绍如何在 MyEclipse 中使用 Spring 与 Hibernate 的组合开发一个这样的简易博客系统。

4.4.1 设计和配置数据库

使用第五章所介绍的 SQLyog 在 mysql 数据库中新建一个数据库命名为 shdb。然后在这个数据库中创建一张表：日志信息表(blog_info)效果如图 4-1 所示：

id	title	content	modify_date
1	这是标题一	这是内容一	2007-11-23
2	这是标题二	这是内容二	2007-11-09
3	这是标题三	这是内容三	2007-11-23

图 4-1 日志信息表(blog_info)

表中 id 是 int 类型的自增主键，title 表示文章标题，content 表示文章内容，modiy_date 表示文章最后修改日期，如果一个文件是新增加的，那么就是新建日期。在 mysql 中创建此表的 SQL 如下所示：

```
CREATE TABLE `blog_info` (
                `id` int(11) NOT NULL auto_increment,
                `title` varchar(100) default NULL,
                `content` varchar(500) default NULL,
                `modify_date` date default NULL,
                PRIM1ARY KEY   (`id`)
              ) ENGINE=InnoDB DEFAULT CHARSET=gbk
```

后面的工作都是针对这张表进行增加，删除，修改，查询操作。

打开 Myeclipse 工作台中的 DataBase Explorer 透视图，使用前章配置 testdb 的办法新配置一个数据库连接命名位 blogdb。配置过程的关键对话框如图 4-2 所示：

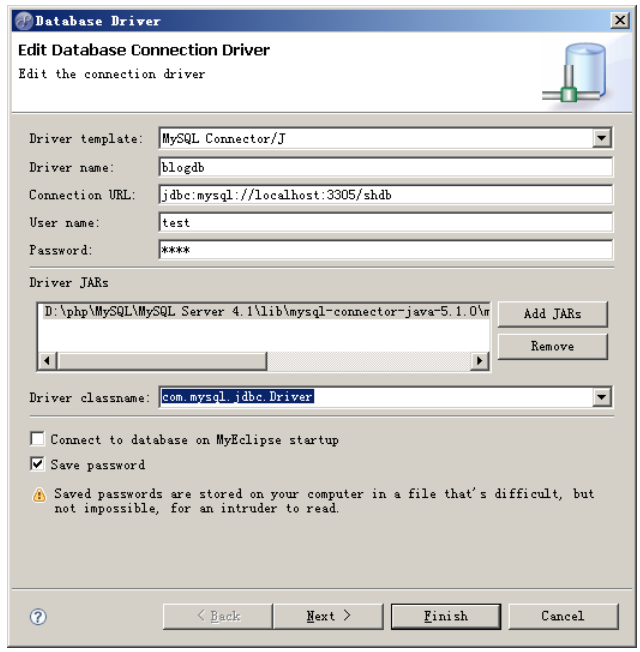

图 4-2 在 MyEclipse 中配置 blogdb

到此关于数据库的设计和配置工作结束。

4.4.2 搭建基础代码

下面开始介绍在 MyEclipse 中搭建一个由 Spring 和 Hibernate 构成的框架代码，这里也称其为基础代码。它主要包括一些 jar 包和 POJO，DAO，以及配置文件。

打开 MyEclipse（本书所使用到的 MyEclipse 版本都为 5.5）如图 4-3 所示：

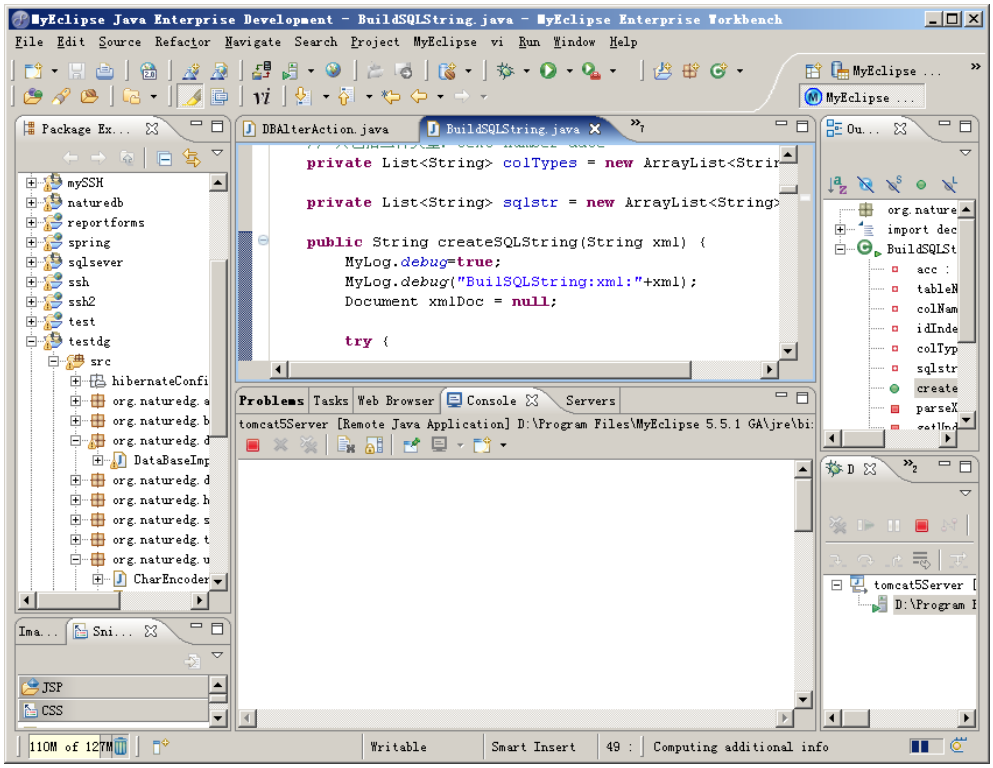

图 4-3 MyEclipse 工作台

使用前面章节所介绍的方法新建一个 web 工程，命名为 shblog，新建后的工

程如图 4-4 所示：

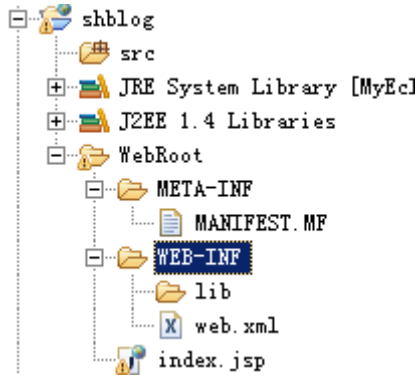

图 4-4 新建的 shblog

新建的 shblog 的工程仅具有一个普通 web project 所具有的代码架构，主要包括一个空的 src 文件夹，一个 JRE 库，J2EE 库，和 web project 必须具备的 META-INF，WEB-INF 文件夹。以及 web.xml 文件。还有 MyEclipse 自动生成的 index.jsp 文件。

下面要为这个“裸工程”添加 Hibernate 和 Spring 能力。这两个模块的添加顺序建议为先 Spring 然后 Hibernate。尽管倒过来也是可以的。但是 Hibernate 与 Spring 组合方式中事实上是有着 Spring 集成 Hibernate 的意味。通过下面的过程也可以发现在 MyEclipse 中先添加 Spring 然后添加 Hibernate 要方便许多。

选中 shblog 这个工程，在 MyEclipse 菜单栏中选择 MyEclipse|Add Spring Capabilities 命令，将弹出如图 4-5 所示的对话框：

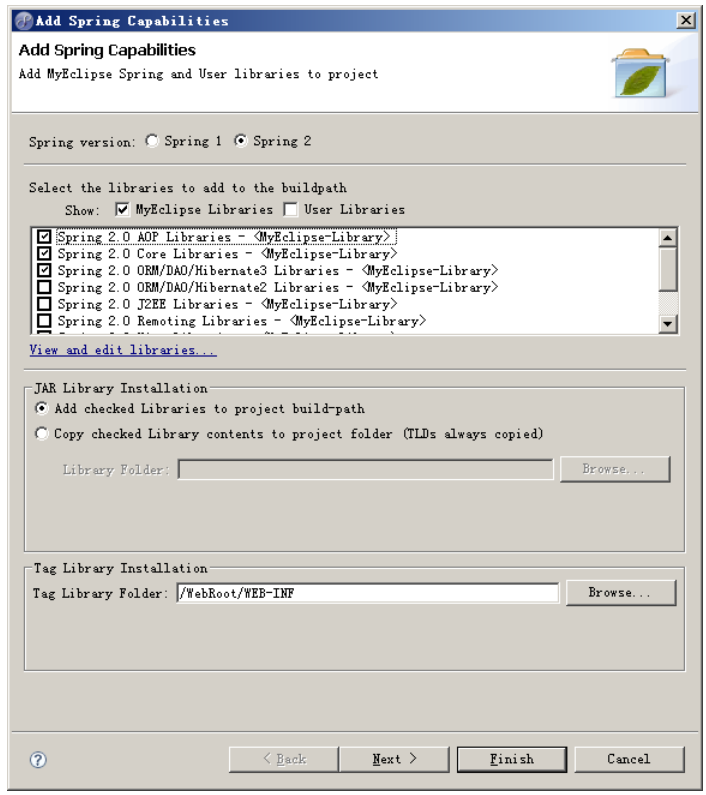

图 4-5 添加 Spring 能力的对话框

需要勾选的包括

- Spring 2.0 Core Libararies：它包括 Spring 的核心功能
- Spring 2.0 ORM/DAO/Hibernate3 Libraries：它包括 Hibernate 的一些服务
- Spring 2.0 AOP Libraries：在选择上面那个库时候，会自动选中这个库，

它提供一些 AOP 服务

- Spring 2.0 Web Libraries：应用于 web 环境下的一些服务

单击 Finish 按钮完成对 Spring 的添加。Spring 能力的添加过程在前面的章节中也是介绍过了的。添加完 Spring 能力后的工程 shblog 如图 4-6 所示：

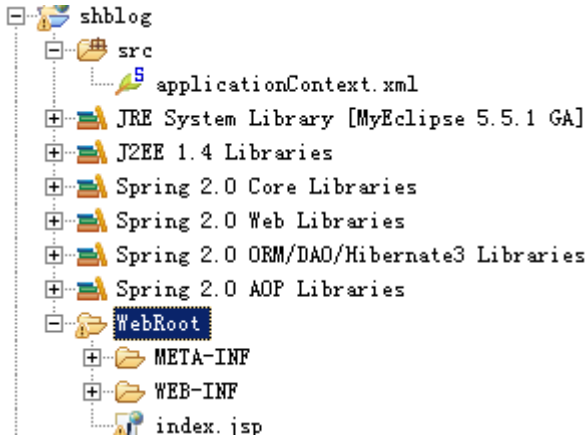

图 4-6 具有 Spring 能力的 shblog

下面添加 Hibernate 能力，选中 shblog 这个工程，在 MyEclipse 菜单栏中选择 MyEclipse|Add Hibernate Capabilities 命令，将弹出如图 4-7 所示的对话框：

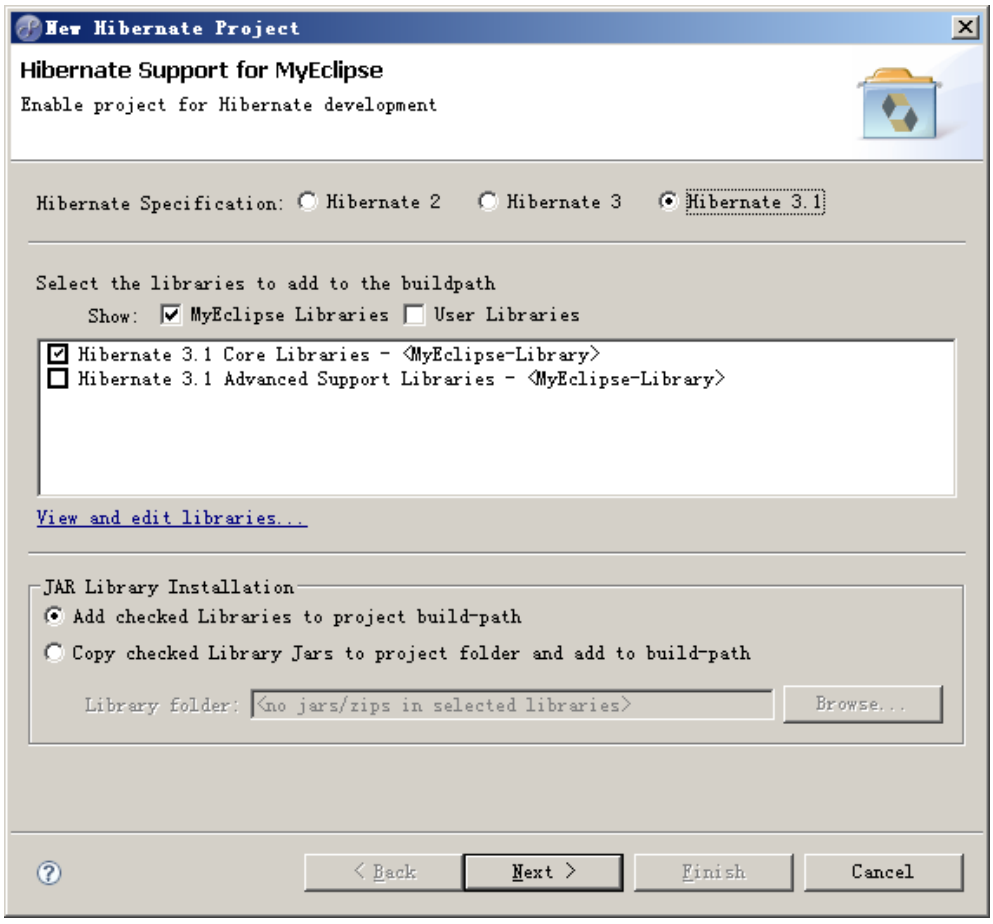

图 4-8 添加 Hibernate 能力

保持默认配置单击 Next 按钮进入下一步，如图 4-9 所示：

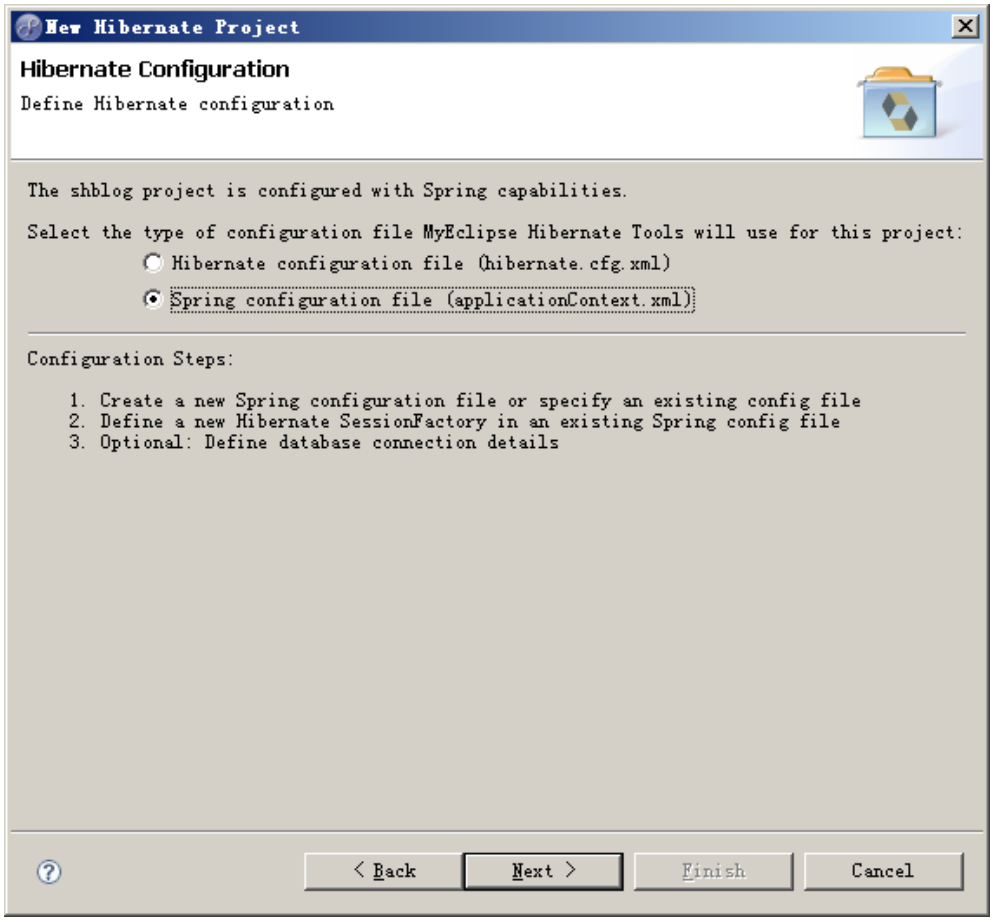

图 4-9 配置文件设置对话框

上面的对话框中有两个选项，由于这里是要将 Hibernate 整合到 Spring 中，所以勾选 Spring configuration file 这个选项，这个动作将导致 hibernate 的配置文件被 Spring 的配置文件取代，hibernate 的配置信息将写入到 Spring 配置文件中。单击 Next 按钮，进入下一步，如图 4-10 所示：

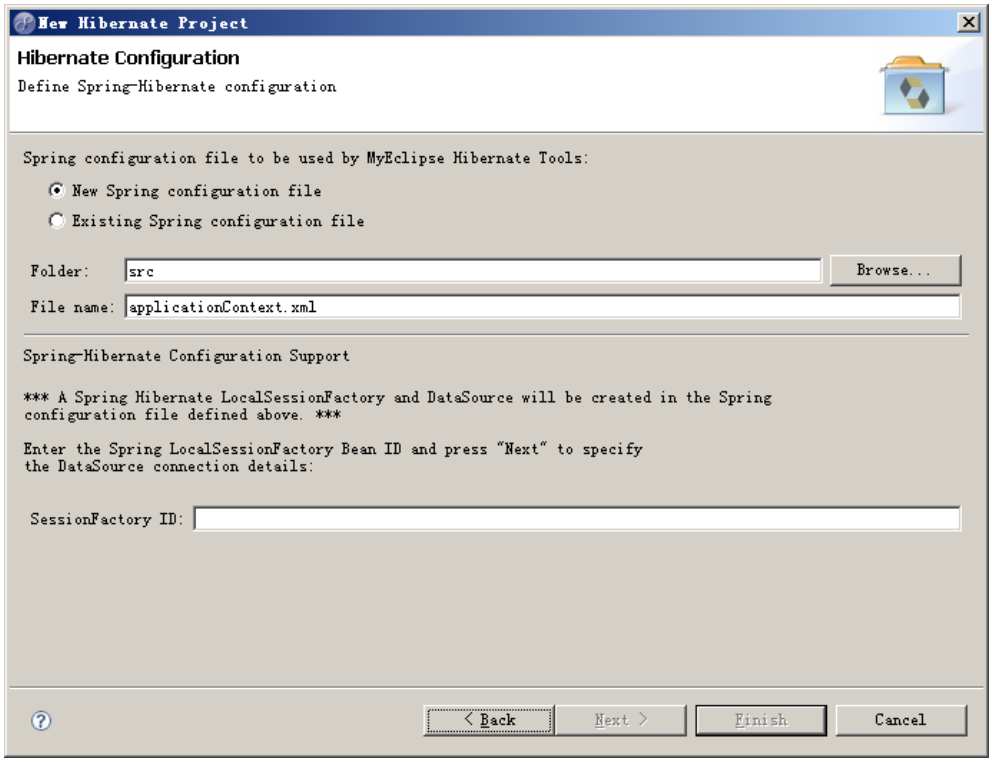

图 4-11 是否新建 applicationContext.xml 对话框

这个对话框是在问是否需要新建一个 applicationContext.xml 文件，还是用已经存在的配置文件。这里当然使用已经存在的，所以选中 Exisiting Spring configuration file 如图 4-12 所示：

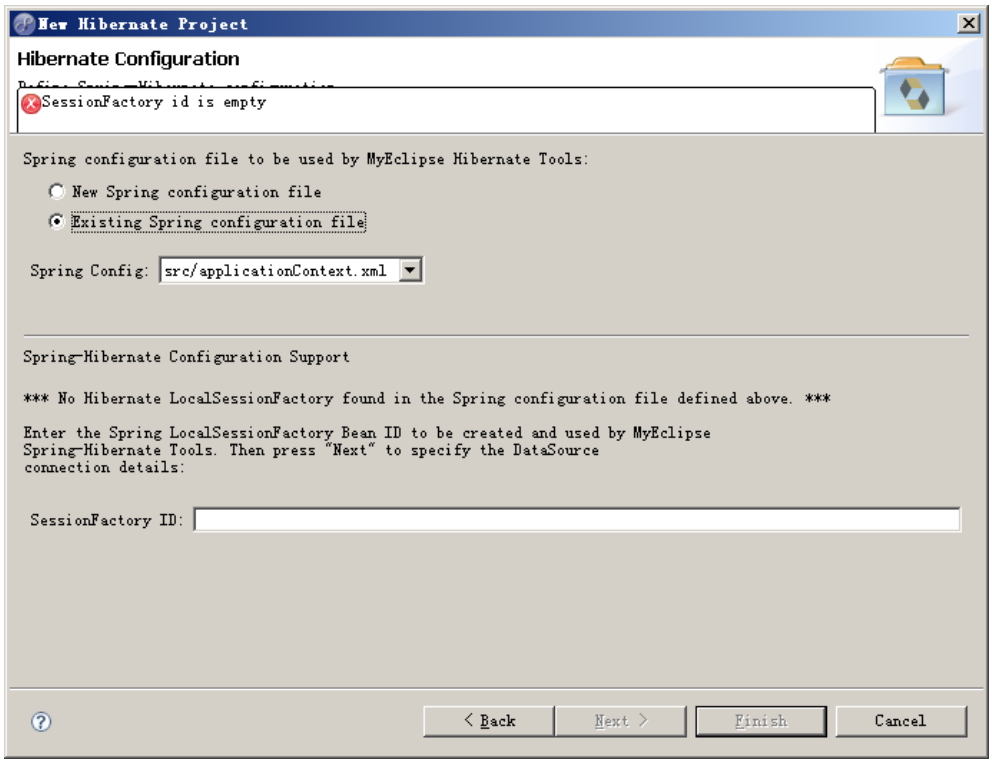

图 4-12 Spring config 对话框

选中 Exisiting Spring configuration 后还需要填写 SessionFactory ID，它表示在 applicationContext.xml 中需要为 Hibernate 的 sessionFactory 注册一个 bean，这个

id 就是 bean 的 id。这里填 hsid 单击 Next 按钮进入下一步，如图 4-13 所示：

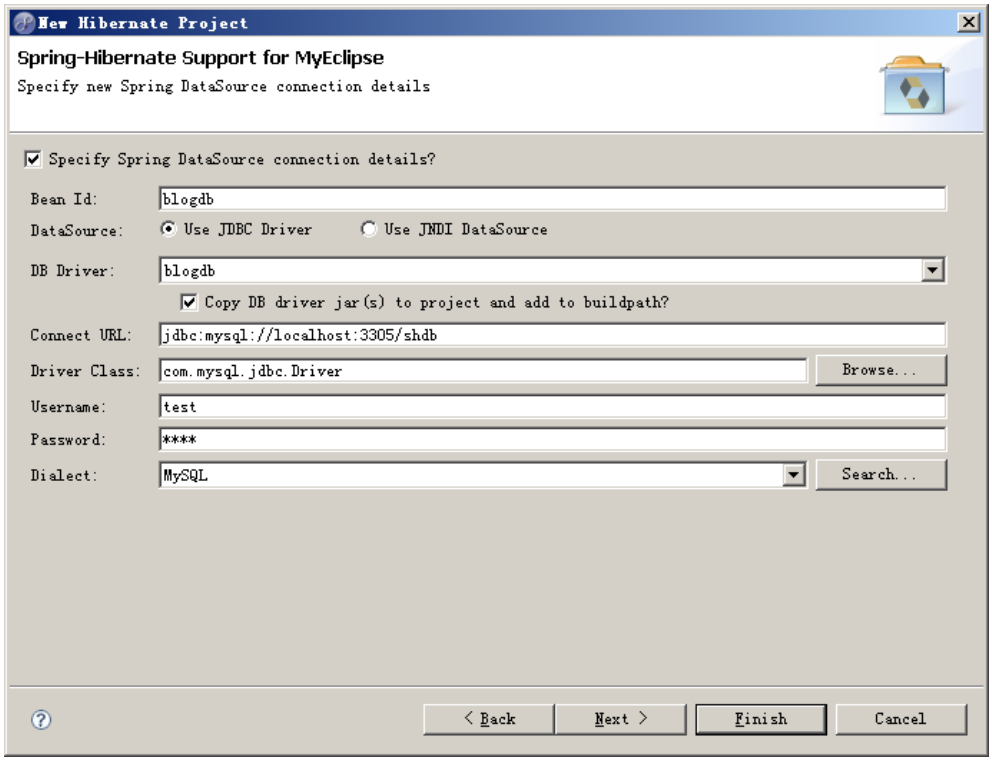

图 4-13 关联数据库

这个对话框的意图非常明显，它包含数据库连接 url，用户名，密码等信息，这些信息将会被写入 applicationContext.xml。这里的 blogdb 是在前一节配置好的数据库驱动。在 DB Driver 一栏中选择它后，其后的 connect URL 等字段将自动填充，唯一需要敲入的是 Bean Id，随便为其指定一个名称即可。单击 Next 按钮进入下一步如图 4-14 所示：

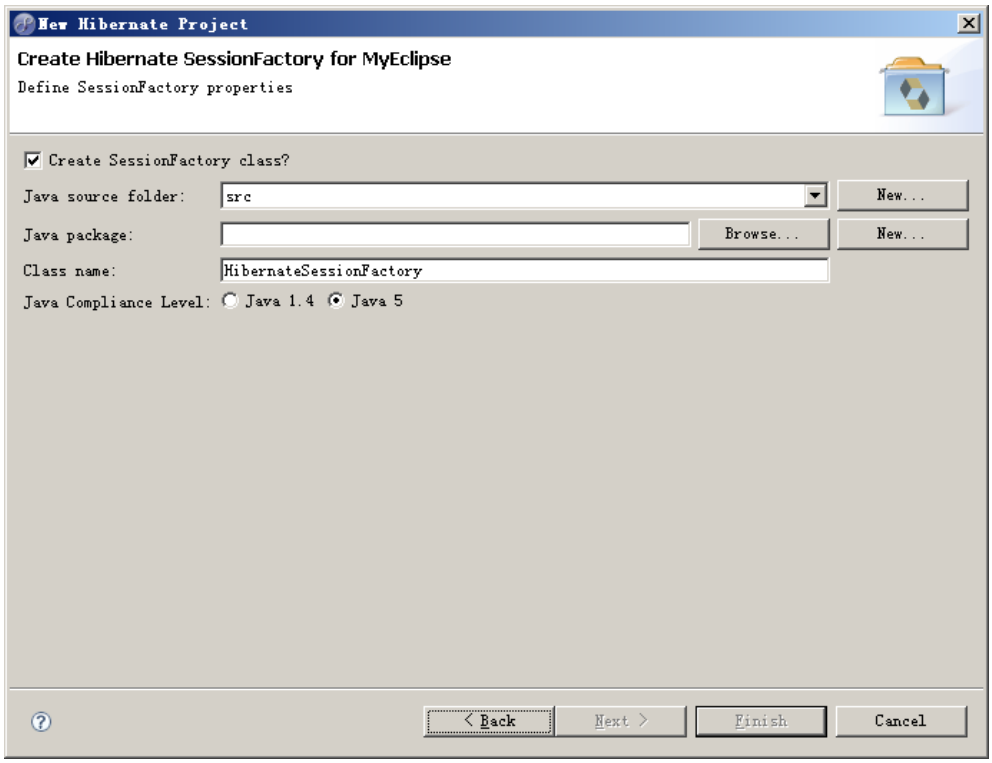

图 4-14 创建 sessionFactory 类对话框

这个对话框是要问是否创建一个 SessionFactory 类，一般都会创建，但是在 Hibernate 与 Spring 的整合项目中，往往是使用 Spring 内置的 sessionFactory 类。所以对于这个工程来说它不是必须的，为了应对未来某些不可预测的需求，这里还是建议创建。做法是在 Java package 通过单击 New..按钮新建一个 hibernate 包，将这个 HibernateSessionFactory 类创建在这个包下，然后单击 Finish 按钮完成对 Hibernate 的添加，添加后的 shblog 工程如图 4-15 所示：

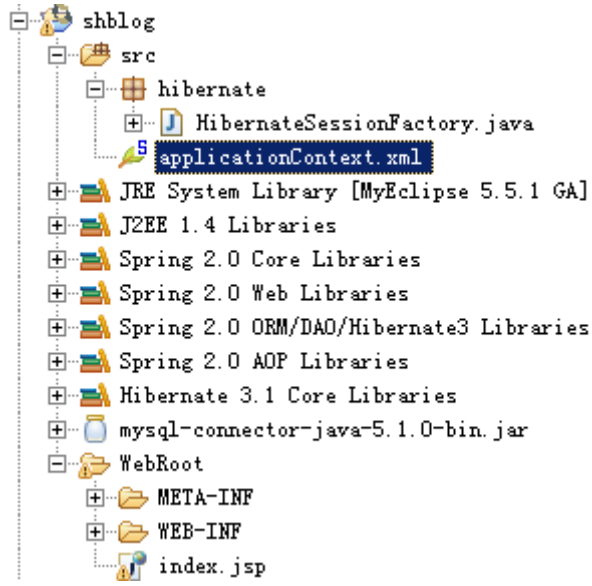

图 4-16 具有 Hibernate 和 Spring 能力的 shblog

添加完 Hibenrate 能力后的 shblog 工程下的 applicationContext.xml 文件，现在已经具有了实质性的内容，代码如下所示：

```
<?xml version="4.0" encoding="UTF-8"?>
<!-- Spring DTD 定义 -->
<beans
 xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<!-- 配置数据库连接的 bean 开始 -->

 <bean id="blogdb" class="org.apache.commons.dbcp.BasicDataSource">
 <!--  -->
        <property name="driverClassName"
              value="com.mysql.jdbc.Driver">
        </property>
        <!--  数据库连接地址-->
        <property name="url" value="jdbc:mysql://localhost:3305/shdb"></property>
        <!-- 数据库用户名 -->
        <property name="username" value="test"></property>
        <!--  数据库密码-->
        <property name="password" value="test"></property>
 </bean>
<!-- 配置数据库连接的 bean 结束 -->
```

```
<!-- Spring 的 hibernate 支持的相关配置  -->
 <bean id="hsid"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <!--   对 blogdb 的引用-->
        <property name="dataSource">
              <ref bean="blogdb" />
        </property>
        <property name="hibernateProperties">
              <props>
              <!--   mysql 数据库方言配置-->
                    <prop key="hibernate.dialect">
                          org.hibernate.dialect.MySQLDialect
                    </prop>
              </props>
        </property>
  </bean></beans>
```

这个配置文件主要包括连个 bean.对于一个 blogdb 的配置不应该感到陌生，与 hibernate 的配置文件内容非常类似。第一个 bean 仅仅是第二个 bean 的一个属性配置。hsid 的配置的配置的主要内容是使用 Spring 中的 LocalSessionFactoryBean 作为 sessionFactory。

下一个重要的工作是对数据表进行逆向工程。关键配置的对话框如图 4-17 所示：

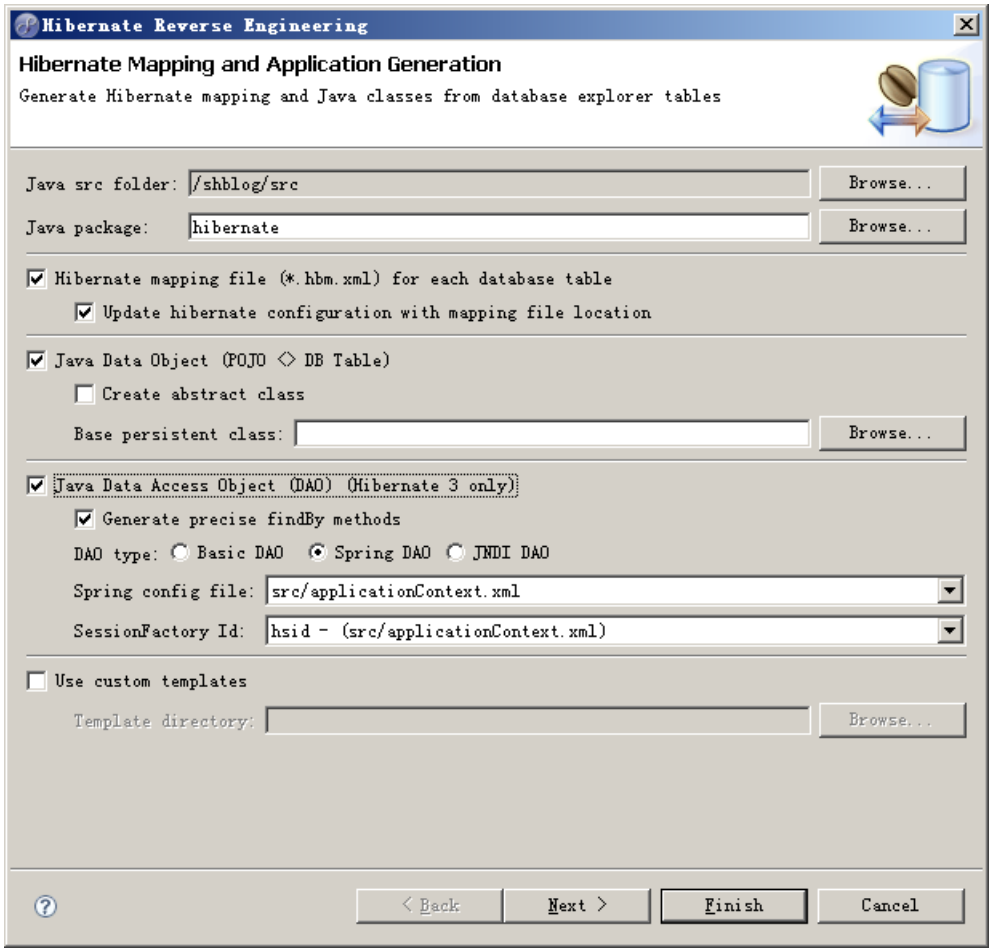

图 4-17Hibernate 逆向工程

这里和以前不同的地方，需要特别指出的是在选择生成 DAO 的时候，请选择 Spring DAO。而不是 BasicDAO。然后单击 Finish 按钮即可。逆向工程后的 shblog 工程如图 4-18 所示：

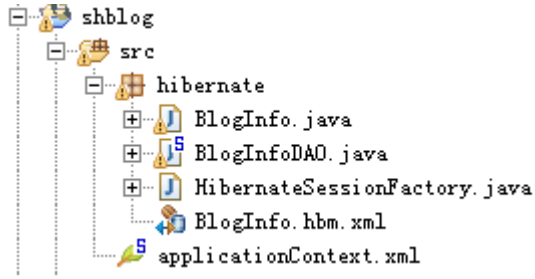

图 4-18 逆向工程后的 shblog

其中 BlogInfo 是自动生成的 POJO，代码如下所示：

```
package hibernate;

import java.util.Date;

/**
 * BlogInfo generated by MyEclipse Persistence Tools
 */
//博客信息类
public class BlogInfo implements java.io.Serializable {
        public static final long serialVersionUID = 1L;//这条语句需要自动添加，它表示序列版本号，没有则会发生警告。
    // Fields

    private Integer id;//主键 id

    private String title;//文章标题

    private String content;//文章内容

    private Date modifyDate;//最后修改日期

    // Constructors

    /** default constructor */
    public BlogInfo() {
    }

    /** minimal constructor */
    public BlogInfo(Integer id) {
        this.id = id;
    }

    /** full constructor */
    public BlogInfo(Integer id, String title, String content, Date modifyDate) {
        this.id = id;
        this.title = title;
```

```
        this.content = content;
        this.modifyDate = modifyDate;
    }

    // Property accessors

    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return this.content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getModifyDate() {
        return this.modifyDate;
    }

    public void setModifyDate(Date modifyDate) {
        this.modifyDate = modifyDate;
    }

}
```

对于这个自动生成的 JavaBean 没有过多的地方需要解释，唯一需要说明的是黑体部分代表，需要手动添加，它表示的是序列化版本号，如果没有这条语句则会出现警告。

另一个自动生成制品是 DAO 文件，内容如下所示：

```
package hibernate;

import java.util.Date;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
```

```
import org.hibernate.LockMode;
import org.springframework.context.ApplicationContext;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

/**
 * Data access object (DAO) for domain model class BlogInfo.
 *
 * @see hibernate.BlogInfo
 * @author MyEclipse Persistence Tools
 */
//继承 Spring 框架中的 HibernateDAO 支持
public class BlogInfoDAO extends HibernateDaoSupport {
 private static final Log log = LogFactory.getLog(BlogInfoDAO.class);

 //
 public static final String TITLE = "title";

 public static final String CONTENT = "content";

 protected void initDao() {
        // do nothing
 }
 //保存一个顺时对象

 public void save(BlogInfo transientInstance) {
        log.debug("saving BlogInfo instance");
        try {
                getHibernateTemplate().save(transientInstance);
                log.debug("save successful");
        } catch (RuntimeException re) {
                log.error("save failed", re);
                throw re;
        }
 }
//删除一个持久对象
 public void delete(BlogInfo persistentInstance) {
        log.debug("deleting BlogInfo instance");
        try {
                getHibernateTemplate().delete(persistentInstance);
                log.debug("delete successful");
        } catch (RuntimeException re) {
                log.error("delete failed", re);
                throw re;
        }
 }
//通过主键找对象
 public BlogInfo findById(java.lang.Integer id) {
        log.debug("getting BlogInfo instance with id: " + id);
        try {
                BlogInfo instance = (BlogInfo) getHibernateTemplate().get(
```

```
                        "hibernate.BlogInfo", id);
                return instance;
        } catch (RuntimeException re) {
                log.error("get failed", re);
                throw re;
        }
 }
//通过对象样例找对象
 public List findByExample(BlogInfo instance) {
        log.debug("finding BlogInfo instance by example");
        try {
                List results = getHibernateTemplate().findByExample(instance);
                log.debug("find by example successful, result size: "
                                + results.size());
                return results;
        } catch (RuntimeException re) {
                log.error("find by example failed", re);
                throw re;
        }
 }

 //通过属性找对象
 public List findByProperty(String propertyName, Object value) {
        log.debug("finding BlogInfo instance with property: " + propertyName
                        + ", value: " + value);
        try {
                String queryString = "from BlogInfo as model where model."
                                + propertyName + "= ?";
                return getHibernateTemplate().find(queryString, value);
        } catch (RuntimeException re) {
                log.error("find by property name failed", re);
                throw re;
        }
 }
//通过标题找对象
 public List findByTitle(Object title) {
        return findByProperty(TITLE, title);
 }
//通过内容找对象
 public List findByContent(Object content) {
        return findByProperty(CONTENT, content);
 }
//列出所有的对象
 public List findAll() {
        log.debug("finding all BlogInfo instances");
        try {
                String queryString = "from BlogInfo";
                return getHibernateTemplate().find(queryString);
        } catch (RuntimeException re) {
                log.error("find all failed", re);
```

```
            throw re;
        }
    }
    //其它方法
    public BlogInfo merge(BlogInfo detachedInstance) {
        log.debug("merging BlogInfo instance");
        try {
            BlogInfo result = (BlogInfo) getHibernateTemplate().merge(
                    detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
        }
    }

    public void attachDirty(BlogInfo instance) {
        log.debug("attaching dirty BlogInfo instance");
        try {
            getHibernateTemplate().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public void attachClean(BlogInfo instance) {
        log.debug("attaching clean BlogInfo instance");
        try {
            getHibernateTemplate().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public static BlogInfoDAO getFromApplicationContext(ApplicationContext ctx) {
        return (BlogInfoDAO) ctx.getBean("BlogInfoDAO");
    }
}
```

上面的这个 DAO 类和在 Hibernate 那一章中就看到的 DAO 有两个最明显的区别，第一个它们继承的父类不一样，第二是这里的类多了一个 getFromApplicationContext 方法，它的作用是通过 Spring 容器得到这个 DAO 的实例。

Hibernate 逆向工程生成的 hbm.xml 文件内容如下所示：

```
<?xml version="4.0" encoding="utf-8"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<!-- hibernate 映射配置开始 -->
<hibernate-mapping>
    <class name="hibernate.BlogInfo" table="blog_info" catalog="shdb">
    <!-- 主键配置 -->
        <id name="id" type="java.lang.Integer">
            <column name="id" />
             <generator class="increment" />
        </id>
        <!-- 标题属性 -->
        <property name="title" type="java.lang.String">
            <column name="title" length="100" />
        </property>
        <!-- 内容属性配置 -->
        <property name="content" type="java.lang.String">
            <column name="content" length="500" />
        </property>
        <!-- 修改日期属性 -->
        <property name="modifyDate" type="java.util.Date">
            <column name="modify_date" length="0" />
        </property>
    </class>
</hibernate-mapping>
<!-- hibernate 映射配置结束 -->
```

这里的主键生成策略修改为自动增长如黑体所示。到此，所有的基础代码都已经完成，换言之，能够由 IDE 工具自动生成的代码都已经完成。接下来介绍业务逻辑代码的开发。

4.4.3 编写控制器代码

在这个博客系统中，可以设计两个控制器，一个是用户登陆控制器，一个是博客操作请求控制器。本节的重点是讲解 Spring 和 Hibernate 的耦合，在控制器部分使用最原始的 HttpServlet 技术即可，用户登陆控制器代码如下所示：

```
package org.blog.controller;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.blog.services.BlogServImpl;
```

```
import org.blog.services.IBlogServ;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

/**
 *  用户登陆控制器
 *
 *
 */
public class LoginController extends HttpServlet {
 public static final long serialVersionUID = 1L;//序列化版本号

 /**
  * Constructor of the object.
  */
 public LoginController() {
        super();
 }

 /**
  * Destruction of the servlet. <br>
  */
 public void destroy() {
        super.destroy(); // Just puts "destroy" string in log
        // Put your code here
 }

 /**
  * The doGet method of the servlet. <br>
  *
  * This method is called when a form has its tag value method equals to get.
  *
  * @param request
  *                    the request send by the client to the server
  * @param response
  *                    the response send by the server to the client
  * @throws ServletException
  *                     if an error occurred
  * @throws IOException
  *                     if an error occurred
  */
 public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        process(request, response);
 }

 /**
  *  实际的处理代码
  * @param request
  * @param response
```

```
     */
    private void process(HttpServletRequest request,
            HttpServletResponse response) {

        WebApplicationContext
wac=WebApplicationContextUtils.getRequiredWebApplicationContext(request.getSession().getServletCont
ext());
        IBlogServ bs =(IBlogServ)wac.getBean("BlogServImpl");//从 Spring 环境中取得 BlogServ 对象

        if (request.getParameter("type").equals("com")) {//如果登陆用户是游客
            try {

                request.setAttribute("content", bs.getBlogContent());//取出日志内容
                request.setAttribute("type", "com");//说明是游客
                request.getRequestDispatcher("blog.jsp").forward(request,
                        response);//跳转到博客显示页面
            } catch (Exception e) {
                e.printStackTrace();
            }
            return;
        }

        String username = request.getParameter("username");//取得用户名
        String password = request.getParameter("password");//取得密码

        if (username == null || !username.equals("yxf") || password == null
                || !password.equals("123")) {//如果用户名和密码不都正确（在实际开发中这里的
用户名和密码应该取自数据库）
            try {
                request.getRequestDispatcher("failed.jsp").forward(request,
                        response);//失败页面
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        else {
            try {
                request.getSession().setAttribute("user", "yxf");//将用户名存入到 session 中，它将
用于别处
                request.setAttribute("content", bs.getBlogContent());//取得内容
                request.setAttribute("type", "admin");//说明是博客主人
                request.getRequestDispatcher("blog.jsp").forward(request,
                        response);//博客内容显示页面
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
```

```
    }

    /**
     * The doPost method of the servlet. <br>
     *
     * This method is called when a form has its tag value method equals to
     * post.
     *
     * @param request
     *                the request send by the client to the server
     * @param response
     *                the response send by the server to the client
     * @throws ServletException
     *                 if an error occurred
     * @throws IOException
     *                 if an error occurred
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        process(request, response);
    }

    /**
     * Initialization of the servlet. <br>
     *
     * @throws ServletException
     *                 if an error occure
     */
    public void init() throws ServletException {
        // Put your code here
    }

}
```

通过这段代码可以看出，它主要使用一个 servlet 作为控制器。它主要判断登陆用户的用户名和密码是否正确，如果一个用户尝试以博客主人的身份登陆且密码或者用户名错误，那么这个控制器将跳转到错误页面。如果成功将跳转到博客显示页面，并告知其当前用户类型为博客主人。如果是登陆者是以游客身份登陆则直接跳转到博客显示页面，并告知其用户类型为游客。

第二个控制器是博客操作控制器，代码如下所示：

```
package org.blog.controller;

import hibernate.BlogInfo;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

import org.blog.services.BlogServImpl;
import org.blog.services.IBlogServ;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

/**
 * 日志操作控制器
 *
 *
 */
public class BlogController extends HttpServlet {
 public static final long serialVersionUID = 1L;

 /**
  * 对业务逻辑类的引用
  */
 private IBlogServ bs;

 /**
  * Constructor of the object.
  */
 public BlogController() {
        super();
 }

 /**
  * Destruction of the servlet. <br>
  */
 public void destroy() {
        super.destroy(); // Just puts "destroy" string in log
        // Put your code here
 }

 /**
  * The doGet method of the servlet. <br>
  *
  * This method is called when a form has its tag value method equals to get.
  *
  * @param request
  *                    the request send by the client to the server
  * @param response
  *                    the response send by the server to the client
  * @throws ServletException
  *                     if an error occurred
  * @throws IOException
  *                     if an error occurred
  */
 public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
            throws ServletException, IOException {
        process(request, response);

    }

    /**
     * The doPost method of the servlet. <br>
     *
     * This method is called when a form has its tag value method equals to
     * post.
     *
     * @param request
     *                the request send by the client to the server
     * @param response
     *                the response send by the server to the client
     * @throws ServletException
     *                 if an error occurred
     * @throws IOException
     *                 if an error occurred
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        process(request, response);

    }

    /**************************************************************************
     *  实际起到控制器作用的代码
     * @param request
     * @param response
     */
    private void process(HttpServletRequest request,
            HttpServletResponse response) {

        if (request.getSession().getAttribute("user") == null) {//未登陆用户的访问是不能对日志进行操作的，除了查看。

            System.out.println("致命的非法请求");//实际开发中请使用具有实际意义的代码替换这句

            return;
        }


        try {
            //在一个大项目中，字符集的设定一般采用过滤器和配置文件实现
            request.setCharacterEncoding("gbk");//设置字符集避免中文乱码
            response.setCharacterEncoding("gbk");//设置字符集避免中文乱码
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }

        WebApplicationContext
wac=WebApplicationContextUtils.getRequiredWebApplicationContext(request.getSession().getServletCont
ext());
          bs =(IBlogServ)wac.getBean("BlogServImpl");//从 Spring 环境中取得 BlogServ 对象

        String action = request.getParameter("action");//取得动作命令

        if ("add".equals(action)) {//添加动作
            String title = request.getParameter("title");//取得新增的标题
            String content = request.getParameter("content");//取得新增内容
            processAdd(title, content);//处理添加
            try {
                request.setAttribute("type", "admin");//告诉 blog.jsp，是博客主人在使用博客
                request.setAttribute("content", bs.getBlogContent());//刷新博客内容
                request.getRequestDispatcher("blog.jsp").forward(request,
                        response);//   跳转到博客页面
            } catch (Exception e) {
                e.printStackTrace();
            }
        }


        else if ("to_edit".equals(action)) {//编辑动作
            String id = request.getParameter("id");//取得主键值
            BlogInfo blog = processToEdit(id);//处理编辑
            request.setAttribute("blog", blog);//将要编辑的日志封装在一个对象中
            try {

                request.getRequestDispatcher("editBlog.jsp").forward(request,
                        response);//跳转到编辑页面
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        else if ("submit_edit".equals(action)) {//完成博客的编辑，并提交
            String id = request.getParameter("id");//取得主键
            String title = request.getParameter("title");//取得标题
            String content = request.getParameter("content");//取得内容
            processSubmitEdit(id, title, content);//处理日志的更新
            try {
                request.setAttribute("type", "admin");//告诉 blog.jsp 是博客主人在使用博客
                request.setAttribute("content", bs.getBlogContent());//刷新博客内容
                request.getRequestDispatcher("blog.jsp").forward(request,
                        response);//跳转到博客页面
            } catch (Exception e) {
                e.printStackTrace();
            }
```

```
        }


        else if ("del".equals(action)) {//删除动作
            String id = request.getParameter("id");//取得待删除的日志主键
            processDelete(id);//处理删除
            try {
                request.setAttribute("type", "admin");//告诉 blog.jsp 是博客主人在使用博客
                request.setAttribute("content", bs.getBlogContent());//刷新博客内容
                request.getRequestDispatcher("blog.jsp").forward(request,
                        response);//跳转到博客页面
            } catch (Exception e) {
                e.printStackTrace();
            }

        }



    }
    /**
     * 处理博客删除请求
     * @param id
     */
    private void processDelete(String id) {
        bs.deleteContent(id);

    }
    /**
     * 处理博客更新请求
     * @param id
     * @param title
     * @param content
     */
    private void processSubmitEdit(String id, String title, String content) {
        bs.modifyContent(id, title, content);
    }
    /**
     * 取得待更新的博客
     * @param id
     * @return
     */
    private BlogInfo processToEdit(String id) {
        return bs.getBlogInfoById(id);
    }
    /**
     * 处理添加请求
     * @param title
     * @param content
     */
```

```
    private void processAdd(String title, String content) {
        bs.addContent(title, content);

    }

    /**
     * Initialization of the servlet. <br>
     *
     * @throws ServletException
     *                 if an error occure
     */
    public void init() throws ServletException {
        // Put your code here
    }

}
```

这个控制器的主要作用是根据博客主人的不同操作请求（包括添加一个日志，删除一个日志，修改一个日志）调用业务逻辑代码将操作反映到数据库中，然后跳转到博客主页。

4.4.4 编写业务逻辑代码

一个好的 MVC 架构中的控制器不应该出现业务逻辑处理代码，正确的做法是为其单独设计一个服务类。在设计这个类前先要设计一个接口，这个服务类的接口代码如下所示：

```
package org.blog.services;

import hibernate.BlogInfo;

import java.util.List;

public interface IBlogServ {

 /**
  * 获得博客内容
  *
  * @return
  */
 public List getBlogContent();

 /**
  * 添加日志
  *
  * @param title
  * @param content
  */
 public void addContent(String title, String content);

 /**
```

```
     * 修改内容
     *
     * @param id
     * @param title
     * @param ChangedContent
     */
    public void modifyContent(String id, String title, String ChangedContent);

    /**
     * 删除一条日志
     *
     * @param id
     */
    public void deleteContent(String id);

    /**
     * 通过主键找对象
     *
     * @param id
     * @return
     */
    public BlogInfo getBlogInfoById(String id);

}
```

使用接口隔离可以使得提供服务的类和调用的客户类保持疏散耦合。下面这个类是其实现类，代码如下所示：

```
package org.blog.services;

import java.util.Date;
import java.util.List;

import hibernate.BlogInfo;
import hibernate.BlogInfoDAO;

/**
 * 博客服务类，主要包含对博客日志的增，删，改，查服务
 *
 *
 */
public class BlogServImpl implements IBlogServ{

/**
 * 对 hibernate 逆向工程制品 DAO 的引用
 */
 private BlogInfoDAO blogdao;
```

```
    /**
     * 获得博客内容
     * @return
     */
    public List getBlogContent() {
        return blogdao.findAll();//获得全部对象
    }
    /**
     * 添加日志
     * @param title
     * @param content
     */
    public void addContent(String title, String content) {
        BlogInfo instance = new BlogInfo();//实例化一个博客对象
        //下面的设置没有包含 id，是因为采用了 increment 策略
        instance.setTitle(title);//设置新的标题
        instance.setContent(content);//设置内容
        instance.setModifyDate(new Date());//使用当前日期
        blogdao.save(instance);//持久化此对象

    }
    /**
     * 修改内容
     * @param id
     * @param title
     * @param ChangedContent
     */
    public void modifyContent(String id, String title, String ChangedContent) {

        BlogInfo instance = blogdao.findById(Integer.parseInt(id));//通过主键找到对象
        instance.setTitle(title);//设置新标题
        instance.setContent(ChangedContent);//设置新内容
        instance.setModifyDate(new Date());//使用当前日期
        blogdao.getHibernateTemplate().update(instance);//更新对象，并将这个游离对象
持久化

    }
    /**
     * 删除一条日志
     * @param id
     */
    public void deleteContent(String id) {
        BlogInfo instance = blogdao.findById(Integer.parseInt(id));//通过主键找到对象
        blogdao.delete(instance);//删除对象

    }
    /**
     * 通过主键找对象
     * @param id
```

```
     * @return
     */
    public BlogInfo getBlogInfoById(String id) {
        return blogdao.findById(Integer.parseInt(id));//使用 DAO 的方法
    }
    public BlogInfoDAO getBlogdao() {
     return blogdao;
    }
    public void setBlogdao(BlogInfoDAO blogdao) {
     this.blogdao = blogdao;
    }

    }
```

通过代码内容可以看出这个服务类主要封装对 DAO 的访问，它首先通过 Spring 容器获得 DAO 的实例。最后利用这个 DAO 对数据库进行添加，删除，修改操作。

这个服务类编写完成后，需要将其注册到 Spring 环境中，其中的 blogdao 也要通过 Spring 配置进行属性注入。添加完配置后的 Spring 配置文件 applicationContext.xml 文件如下所示：

```
    <?xml version="4.0" encoding="UTF-8"?>
    <!-- Spring DTD 定义 -->
    <beans
     xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 配置数据库连接的 bean 开始 -->

     <bean id="blogdb" class="org.apache.commons.dbcp.BasicDataSource">
     <!--  -->
          <property name="driverClassName"
               value="com.mysql.jdbc.Driver">
          </property>
          <!--  数据库连接地址-->
          <property name="url" value="jdbc:mysql://localhost:3305/shdb"></property>
          <!-- 数据库用户名 -->
          <property name="username" value="test"></property>
          <!--  数据库密码-->
          <property name="password" value="test"></property>
     </bean>
     <!-- 配置数据库连接的 bean 结束 -->

    <!-- Spring 的 hibernate 支持的相关配置 -->
     <bean id="hsid"
          class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

          <!--  对 blogdb 的引用-->
          <property name="dataSource">
```

```
            <ref bean="blogdb" />
        </property>
        <property name="hibernateProperties">
            <props>
            <!--  mysql 数据库方言配置-->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
        <property name="mappingResources">
            <list>
                <value>hibernate/BlogInfo.hbm.xml</value></list>
        </property></bean>
    <bean id="BlogInfoDAO" class="hibernate.BlogInfoDAO">
        <property name="sessionFactory">
            <ref bean="hsid" />
        </property>
    </bean>
    <!-- 注册服务类 -->
    <bean id="BlogServImpl" class="org.blog.services.BlogServImpl">
    <property name="blogdao">
    <ref   bean="BlogInfoDAO"/>
    </property>
    </bean>


    </beans>
```

配置部分如粗体部分所示，这里的属性直接引用已配置了的 BlogInfoDAO 这个 bean。

4.4.5 编写视图层页面

Spring 与 Hibernate 的组合在视图技术上有比较多的选择，主要是 Spring 对视图技术的支持比较丰富。这里的示例仍然使用最为典型的 JSP 视图技术。视图页面包括：用户登陆页面，错误显示页面，博客主页，博客编辑页面，博客添加页面。

下面是用户登陆页面的 index.jsp 代码：

```
    <%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>
    <%
    String path = request.getContextPath();
    String                                    basePath                                    =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>

    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">
```

```
    <title>My JSP 'index.jsp' starting page</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
    <script type="text/javascript">
    function comLogin(){//游客登陆控制 js
    var login_form=document.getElementById("loginForm");
    login_form.action="LoginController?type=com";//修改表单的 action
    login_form.submit();//提交表单
    }
    </script>
  </head>

  <body>

<table width="100%" border="0" cellspacing="0" cellpadding="4">

  <tr>

    <td bgcolor="#000099">

      <table width="100%" border="0" cellspacing="0" cellpadding="4">

        <tr>

          <td bgcolor="#FFFFFF">o <b>*</b> </td>
<!-- 标题设置 -->
          <td width="100%"><font color="#CCCCCC">  <font color="#FFFFFF">yxf 的个人博客</font></font></td>

        </tr>

      </table></td>

  </tr>

  <tr>

    <td width="100%" bgcolor="#EAEAEA" colspan="2">
<!-- 用户登陆表单 -->
      <form id="loginForm" action="LoginController?type=admin" method="post"><p>

        <label for="textfield">博客主人：</label>
```

```
                    <br>
                    <input type="text" name="username" >
                    </p>
                    <p>
                    <label for="password">密码：</label>
                    <br>
                    <input type="password" name="password" >
                    </p>
                    <p>
                    <input type="submit" name="Submit" value=" 登 陆 "><input type="button" onclick="comLogin()" value="游客登陆">
                    </p>
                    <p>  </p>
                    </form>

            </td>
        </tr>
    </table>




    </body>
</html>
```

这个页面比较简单，主要包含一个表单和一个游客用户登陆控制 javascript 代码。作为视图技术，Javascript 是一个强大的客户端语言，它使得视图变得生动，当然不仅仅如此，它最大的优点是可以方便的操作 HTML 元素。

错误信息提示页面 failed.jsp 如下所示：

```
<%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>
<%
String path = request.getContextPath();
```

```
    String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>

    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">

        <title>My JSP 'failed.jsp' starting page</title>

     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
     <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
     <meta http-equiv="description" content="This is my page">
     <!--
     <link rel="stylesheet" type="text/css" href="styles.css">
     -->

      </head>

      <body>
    用户名或者密码错误，请返回<a href="index.jsp">登陆页面</a>
      </body>
    </html>
```

此页面仅包含一条输出语句，并含有一个返回到首页的超级链接，这样的错误提示页面并不是必须的，尤其是使用 Ajax 这样 web2.0 技术，完全可以在出错的当页显示。

博客编辑页面 editBlog.jsp 如下所示：

```
    <%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
    <jsp:directive.page import="hibernate.BlogInfo"/>
    <%
    String path = request.getContextPath();
    String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>
    <!--  博客编辑页面-->
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">

        <title>My JSP 'editBlog.jsp' starting page</title>

     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
```

```
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
    <%BlogInfo blog=(BlogInfo)request.getAttribute("blog"); %>
  <table width="100%" border="0" cellspacing="0" cellpadding="4">

    <tr>

        <td bgcolor="#000099">

            <table width="100%" border="0" cellspacing="0" cellpadding="4">

                <tr>

                    <td bgcolor="#FFFFFF"> <b>*</b> </td>

                        <td width="100%"><font color="#CCCCCC">  <font color="#FFFFFF">编辑日志</font></font></td>

                </tr>

            </table></td>

    </tr>

    <tr>

        <td width="100%" bgcolor="#EAEAEA" colspan="2">

            <form action="BlogController?action=submit_edit&id=<%=blog.getId() %>" method="post"><p>

                <label for="textfield">标题</label>

                <br>

                <input type="text" name="title" value="<%=blog.getTitle() %>">

                </p>

                <p>

                <label for="textfield2">内容</label>
```

```
                    <br>
                    <textarea rows="15" cols="100" name="content"><%=blog.getContent() %></textarea>
                    </p>
                    <p>
                    <input type="submit" name="Submit" value="提交">
                    </p>
                    <p>  </p>
                    </form>
                </td>
            </tr>
        </table>

        </body>
    </html>
```

这个页面的主要元素是表单，它包含了三个标题，内容这两个输入字段。当用户输入内容后，次表单被提交到BlogController控制器，并调用相应的submit_edit方法。

下面是博客主页 blog.jsp 代码：

```
    <%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>

    <jsp:directive.page import="hibernate.BlogInfo"/>
    <jsp:directive.page import="java.text.SimpleDateFormat"/>
    <!-- 博客内容页面-->
    <%
    String path = request.getContextPath();
    String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>

    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">

        <title>My JSP 'blog.jsp' starting page</title>
```

```
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
    <!-- 定义样式 -->
  <style type="text/css">
  .title{
  background-color:"#000088";
  color:"#FFFFFF";
  }
  a{
  color:white;
  }

  </style>

    </head>

    <body>

  <%
    String type=(String)request.getAttribute("type");//取得用户类型，分为博客主人和游客两者
    List blogList=(List)request.getAttribute("content");//取得博客内容
    if(type.equals("admin")){//是博客主人
      out.print("<a href='addNewBlog.jsp' ><font color='red'>添加新日志</font></a>");//可以添加博客
    }


  SimpleDateFormat format=new SimpleDateFormat("yyyy-MM-dd");//格式化日期输出
    for(int i=0;i<blogList.size();i++){//依次输出博客内容
    BlogInfo blog=(BlogInfo)blogList.get(i);//类型转化
    String datestr="";//代表日期的字符串
    Date modifydate=blog.getModifyDate();//得到修改日期
    if(modifydate!=null){//防止空值异常
    datestr=format.format(modifydate);//格式日期
    }
    //输出博客标题和日期
    out.print("<div class='title' id='"+i+"'>"+blog.getTitle()+"<span style='text-align:right;width:100%'>"+datestr);
    //如果是博客主人还可以对日志进行删除和编辑
    if(type.equals("admin")){
    out.print("<a href='BlogController?action=del&id="+blog.getId()+"'> 删 除</a> ");
    out.print("<a href='BlogController?action=to_edit&id="+blog.getId()+"' >编辑</a>");
```

```
        }
        out.print("</span></div>");
        //输出日志内容
        out.print("<div style='content'>"+blog.getContent()+"</div>");
        }


    %>



        </body>
    </html>
```

这段 jsp 代码主要的功能是显示博客内容，并根据用户角色显示不同的交互界面。如果是博客主人，则生成可修改界面，如果是游客则只生成查看页面。

下面是博客新增页面 addNewBlog.jsp，这个页面主要由一个表单组成，包含标题和内容两个字段，表单提交给 BlogController 这个多控制器处理，并调用 add 方法进行实质性的控制处理。代码如下所示：

```
    <%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
    <%
    String path = request.getContextPath();
    String                                    basePath                                    =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>
    <!-- 添加新日志 -->
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">

        <title>My JSP 'addNewBlog.jsp.jsp' starting page</title>

     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
     <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
     <meta http-equiv="description" content="This is my page">
     <!--
     <link rel="stylesheet" type="text/css" href="styles.css">
     -->

      </head>

      <body>
    <table width="100%" border="0" cellspacing="0" cellpadding="4">

     <tr>
```

```
            <td bgcolor="#000099">

                <table width="100%" border="0" cellspacing="0" cellpadding="4">

                    <tr>

                        <td bgcolor="#FFFFFF"> <b>*</b> </td>

                            <td width="100%"><font color="#CCCCCC">  <font color="#FFFFFF">新增日志</font></font></td>

                    </tr>

                </table></td>

        </tr>

        <tr>

            <td width="100%" bgcolor="#EAEAEA" colspan="2">
      <!-- 新增日志表单 -->
                <form name="Name" action="BlogController?action=add" method="post"><p>

                    <label for="textfield">标题</label>

                    <br>

                    <input type="text" name="title" >

                    </p>

                    <p>

                    <label for="textfield2">内容</label>

                    <br>

                    <textarea rows="15" cols="100" name="content"></textarea>

                    </p>

                    <p>

                    <input type="submit" name="Submit" value="提交">

                    </p>

                    <p>  </p>
```

```
                    </form>

                </td>

            </tr>

        </table>


            </body>
        </html>
```

4.4.6 运行效果

登陆首页 index.jsp 运行后效果如图：4-19 所示：

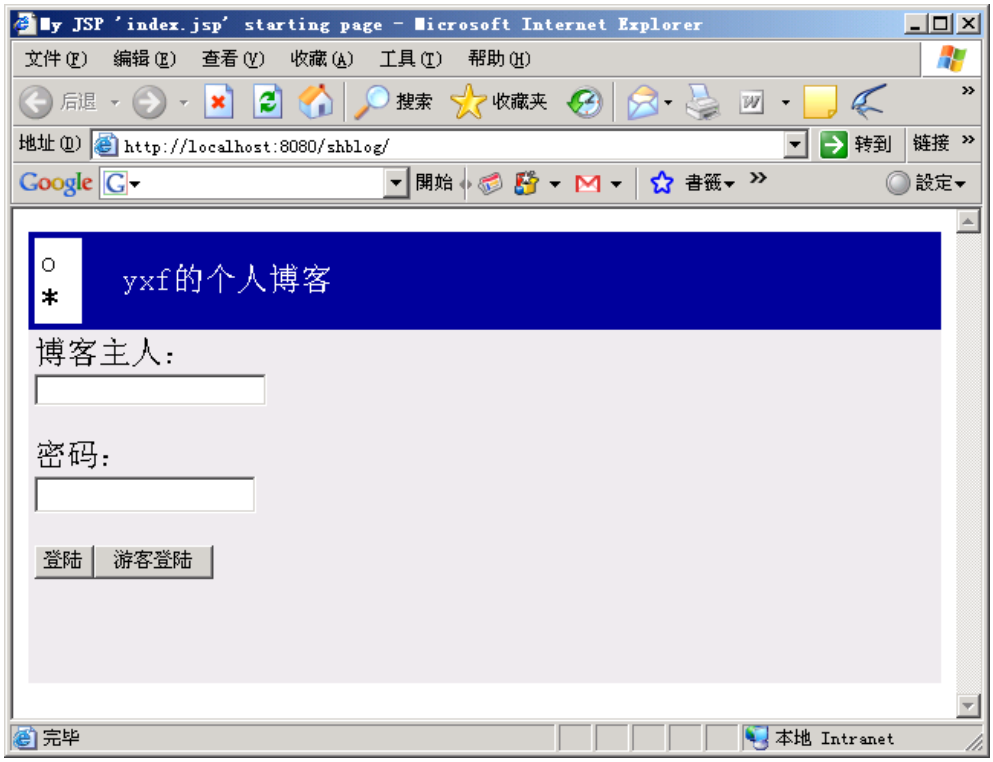

图 4-19 博客登陆页面

在博客主人字段中输入 yxf，密码字段中输入 456（错误密码），单击“登陆”按钮，将跳转到 failed.jsp 效果如下所图 4-20 所示：

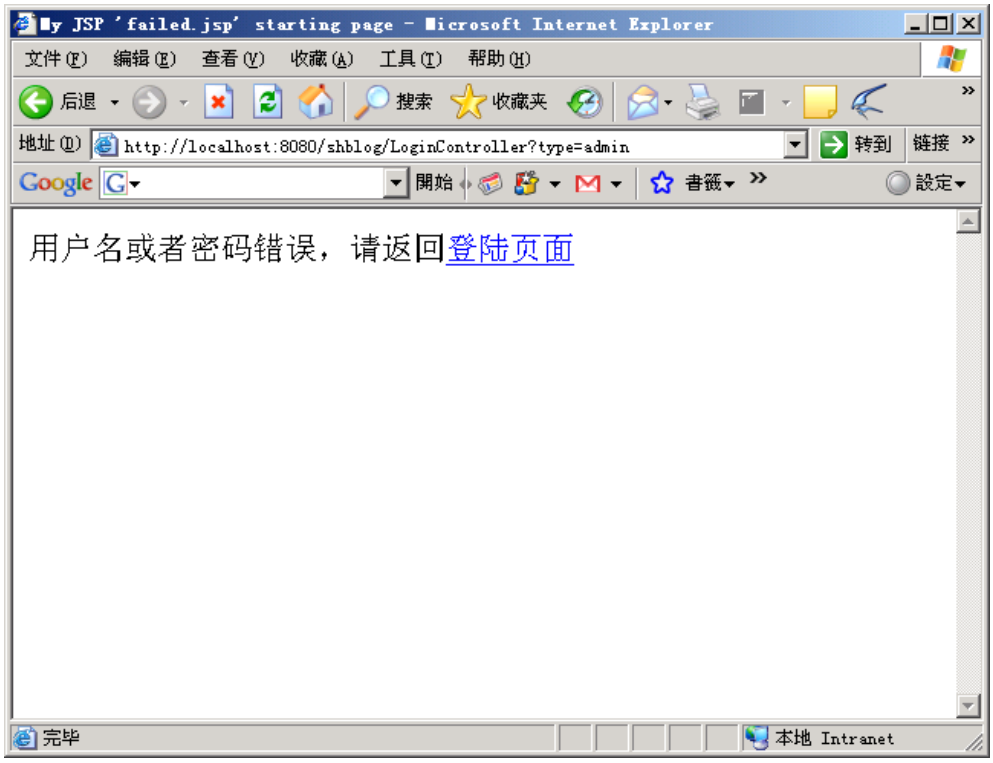

图 4-20 登陆失败页面

单击登陆页面返回到登陆首页以正确的用户名 yxf 和密码 123 输入，将跳转到 blog.jsp 页面。效果如图 4-21 所示：

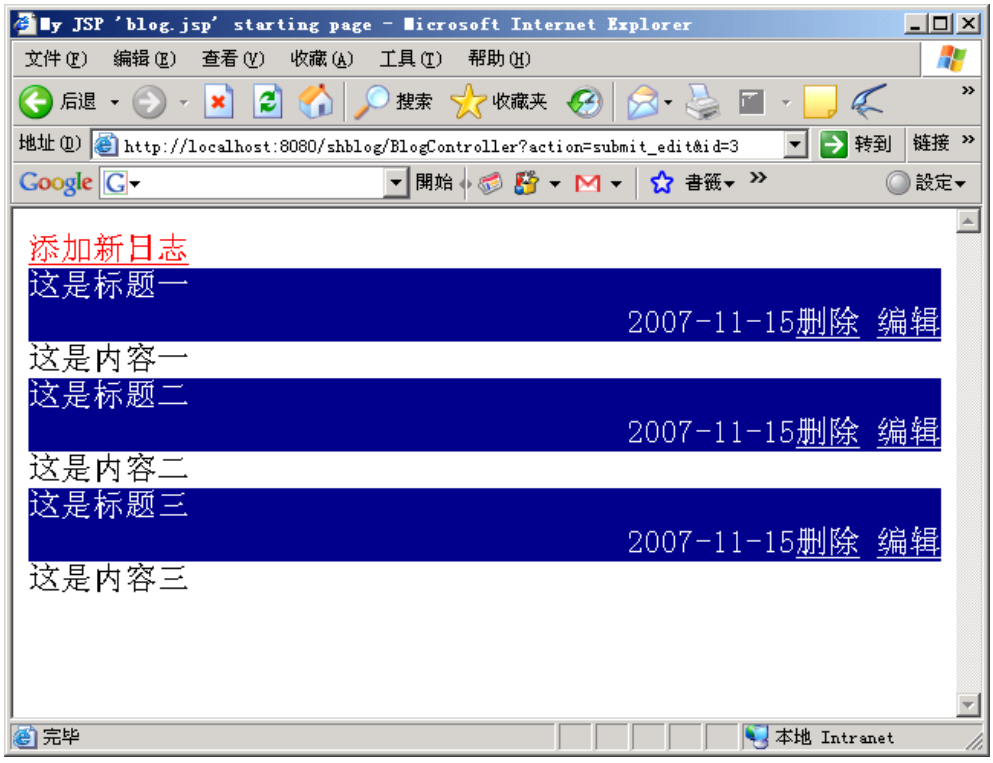

图 4-21 博客首页

单击“添加新日志”链接将进入博客添加页面 addNewBlog.jsp 效果如图 4-22 所示：

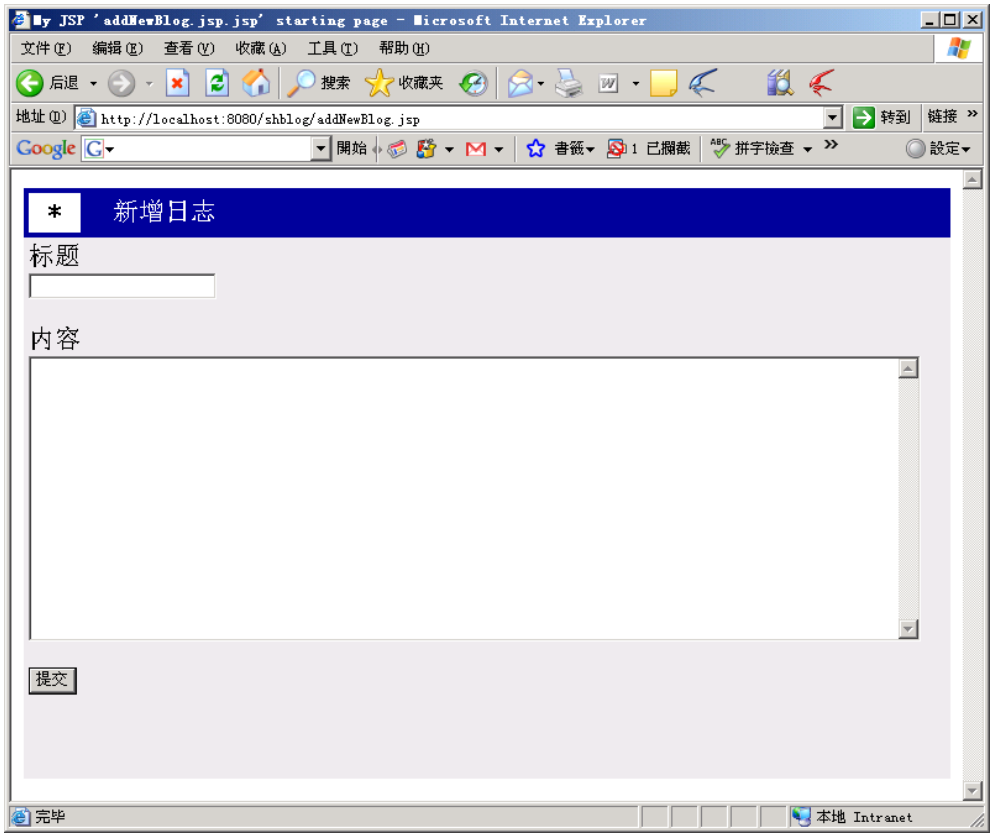

图 4-22 新增日志页面

新增一个标题为“这是新增标题”，新增内容为“这是新增内容”然后单击“提交”按钮，将重新进入到博客首页，新增后的 blog.jsp 页面效果如下图 4-23 所示：

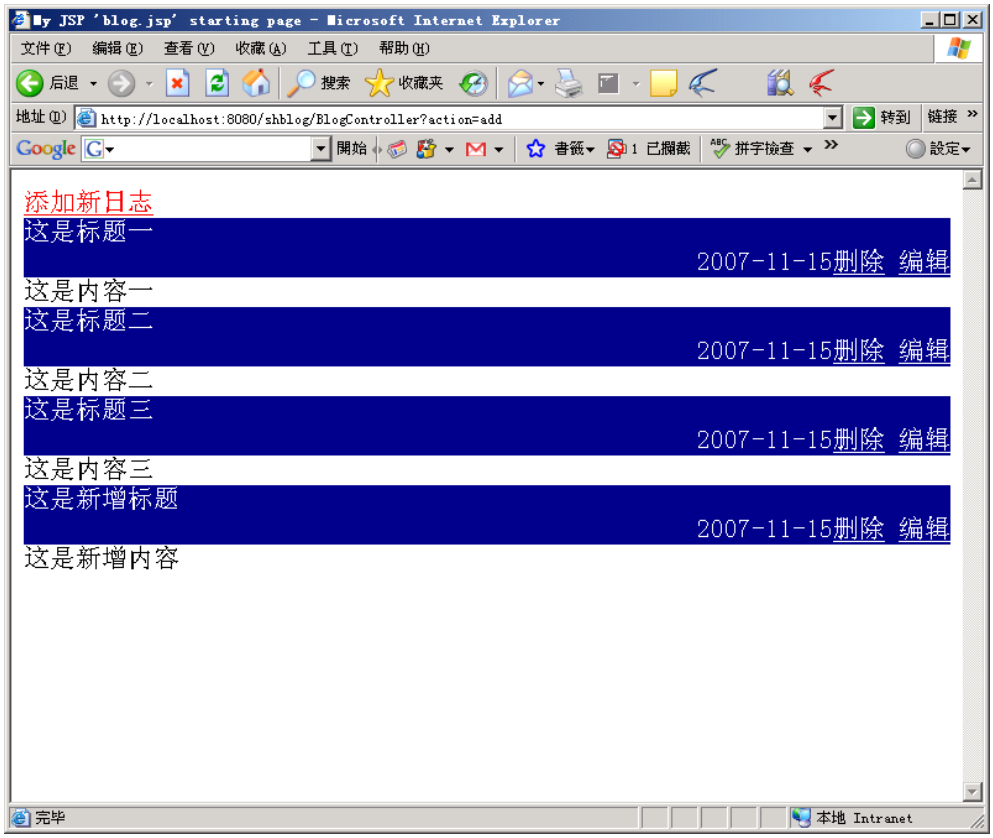

图 4-23 新增后的博客首页

在标题二选项中单击“编辑”链接，将进入博客的编辑页面 editBlog.jsp 效果

如图 4-24 所示：

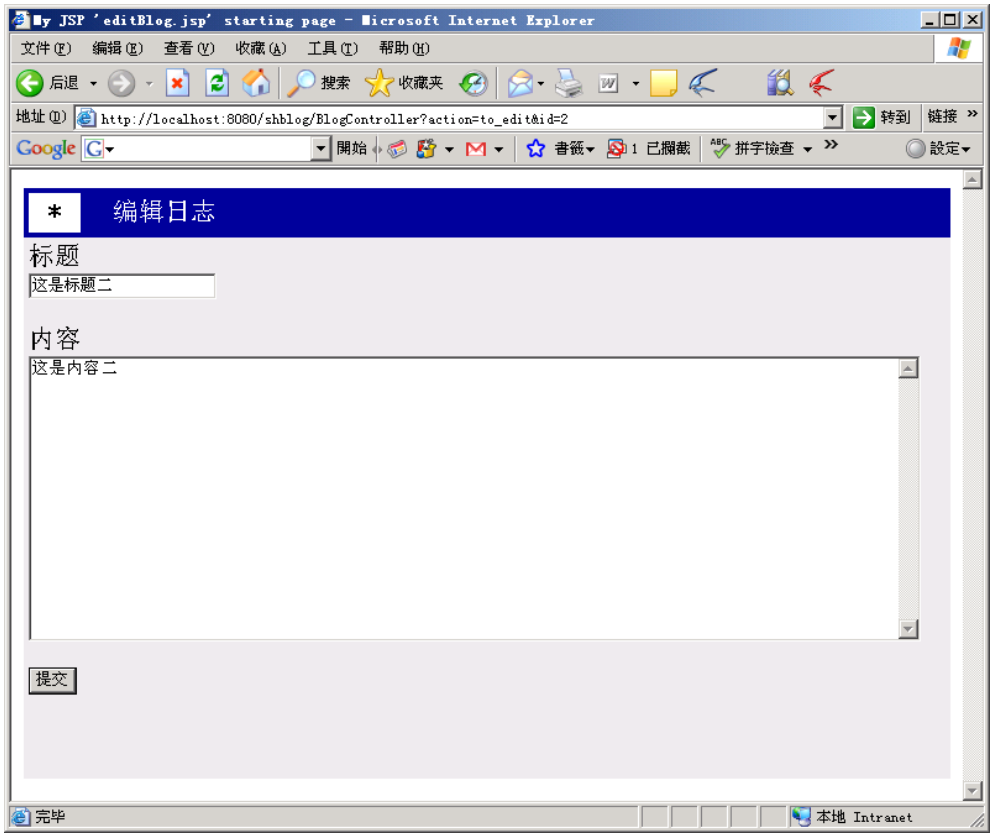

图 4-24 编辑日志页面

修改内容为“这是内容二（修改后）”然后单击“提交”按钮，进入 blog.jsp 页面，效果如图 4-25 所示：

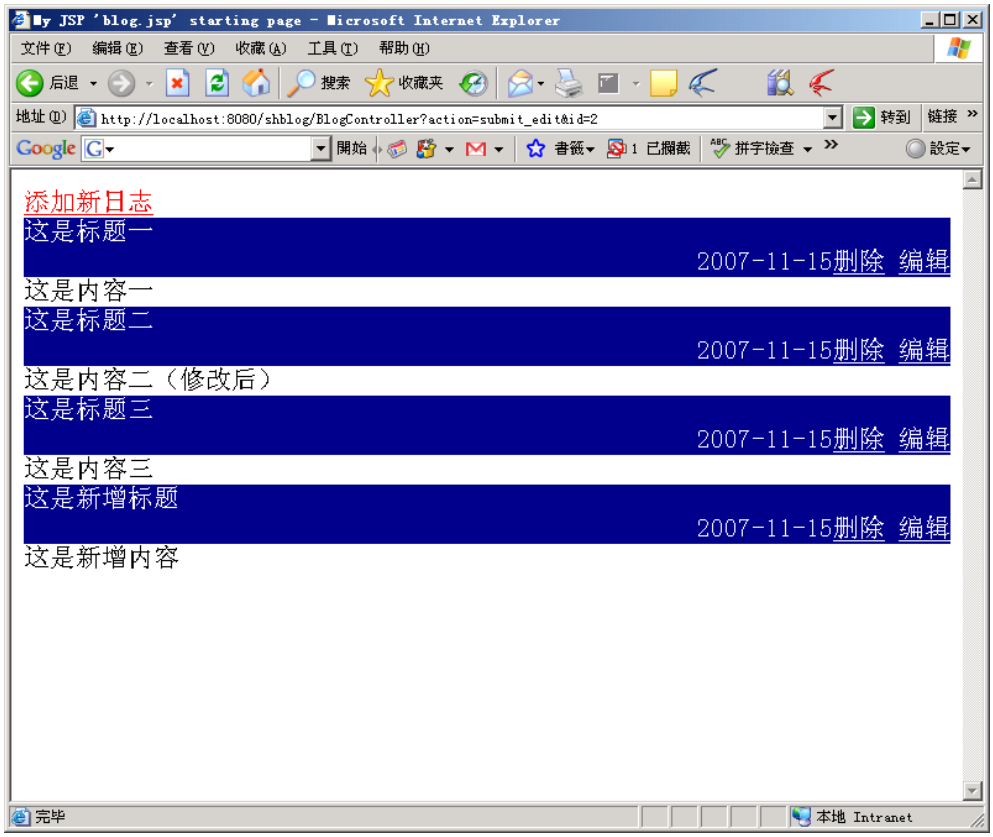

图 4-25 修改后的首页

在标题三选项中单击“删除”按钮，页面出现短暂的刷新，刷新后的首页如图 4-26 所示：

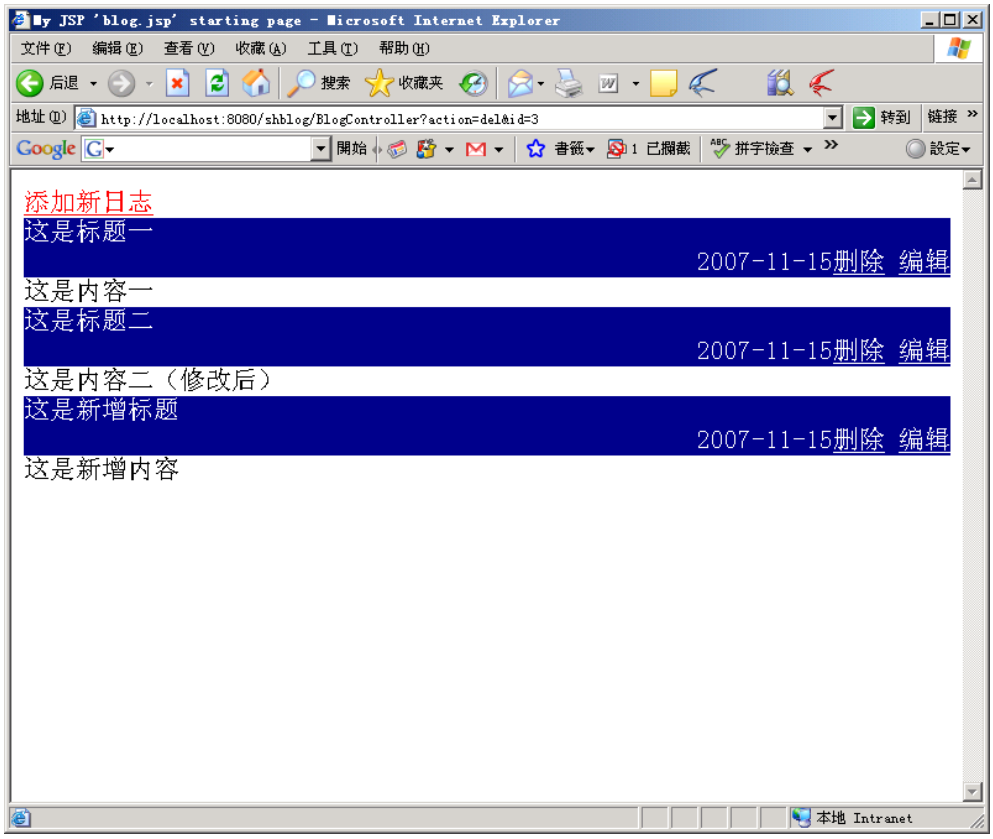

图 4-26 删除日志

重新回到首页以游客的身份登陆，登陆后的页面如图 4-27 所示：

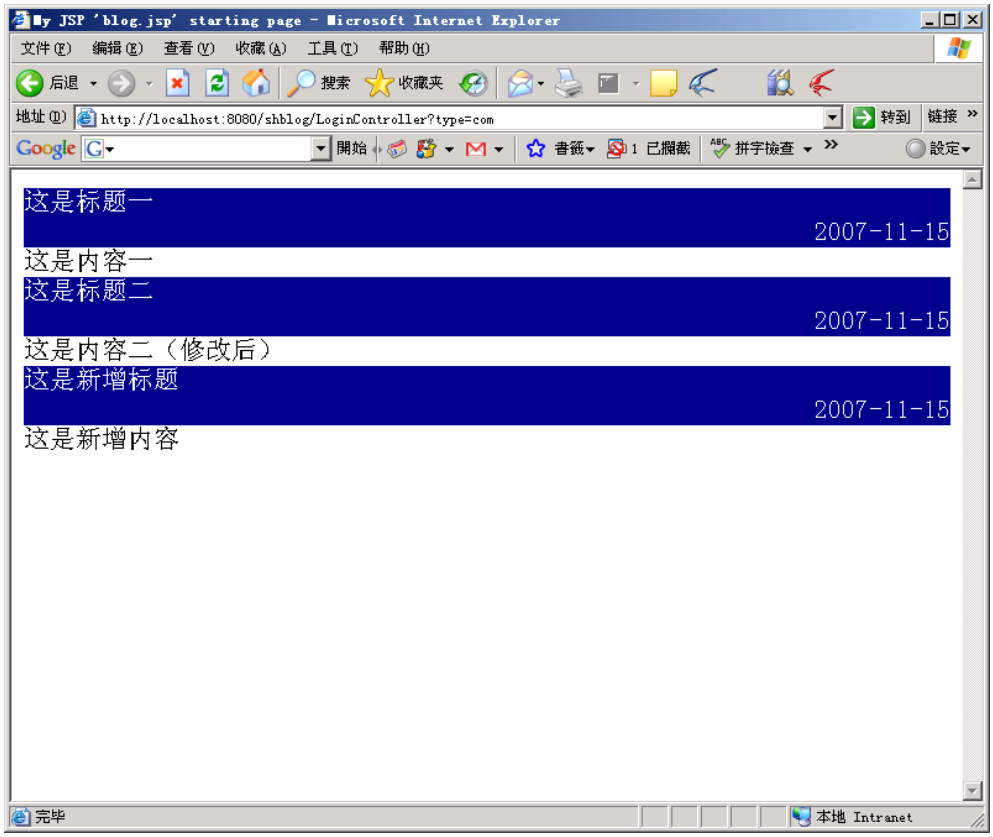

图 4-27 游客浏览博客首页

可见，游客仅有浏览的权限。读者可以扩展和重构这个例子，比如为游客添加留言功能。添加各种安全控制，对控制器中的多动作进行分离等。

4.2 Spring 与 Struts 的整合

通过前面章节的学习可以知道 Spring 完全具有 Struts 所具有的 MVC 框架，但由于 Struts 被广泛的开发人员接受和使用所以有必要介绍这两种框架的组合。下面将讲解如何使用这两种组合实现上一节的简易博客系统。并在实现方式上做一定的优化调整。

4.2.1 搭建框架环境

在 MyEclipse 中新建一个 web 工程名为 ssblog。为其添加 Spring 和 Struts 能力。添加 Spring 能力的关键对话框如图 4-28 所示：

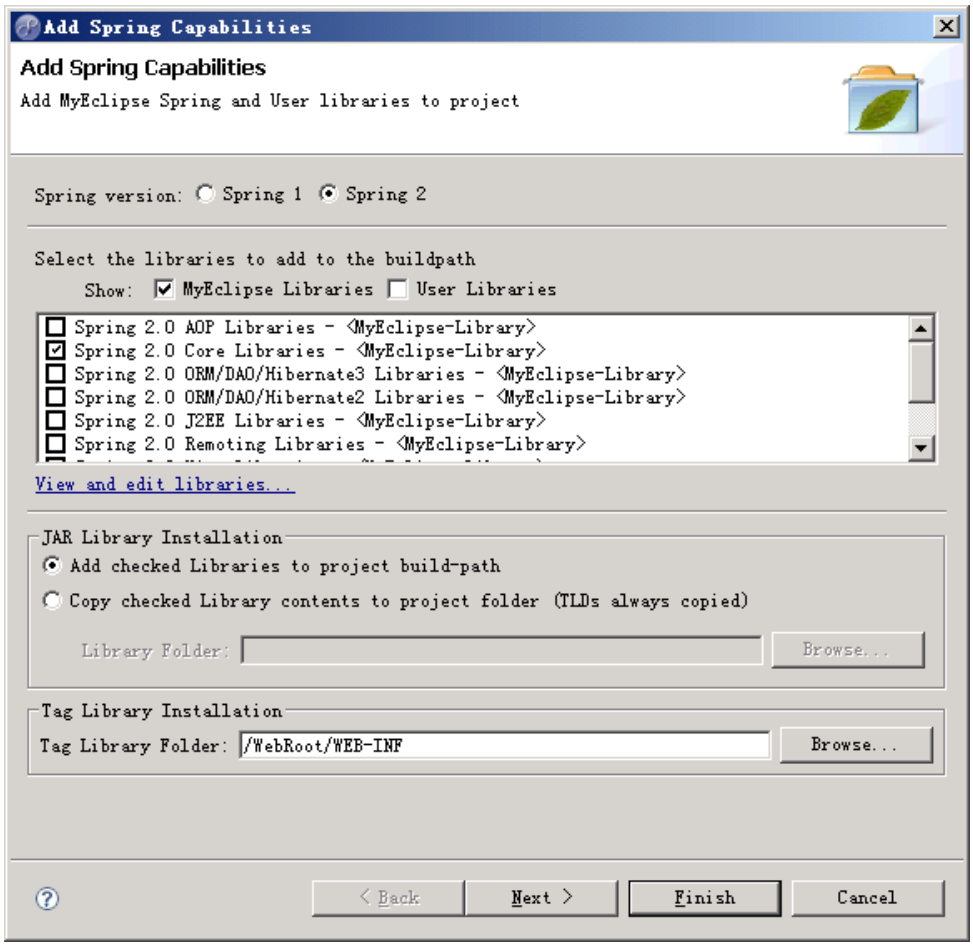

图 4-27 为 ssblog 添加 Spring 能力

与上一节不同的是，这里不再需要 Spring2.0 ORM/DAO/Hibernate3 和 Spring 2.0 AOP Libraries 这两个库（但仍然保留 Spring2.0 Web Libraries）因为本节的示例是基于 Spring 和 Struts 框架，并不再使用 Hibernate。添加 Struts 能力的关键对话框如图 4-28 所示：

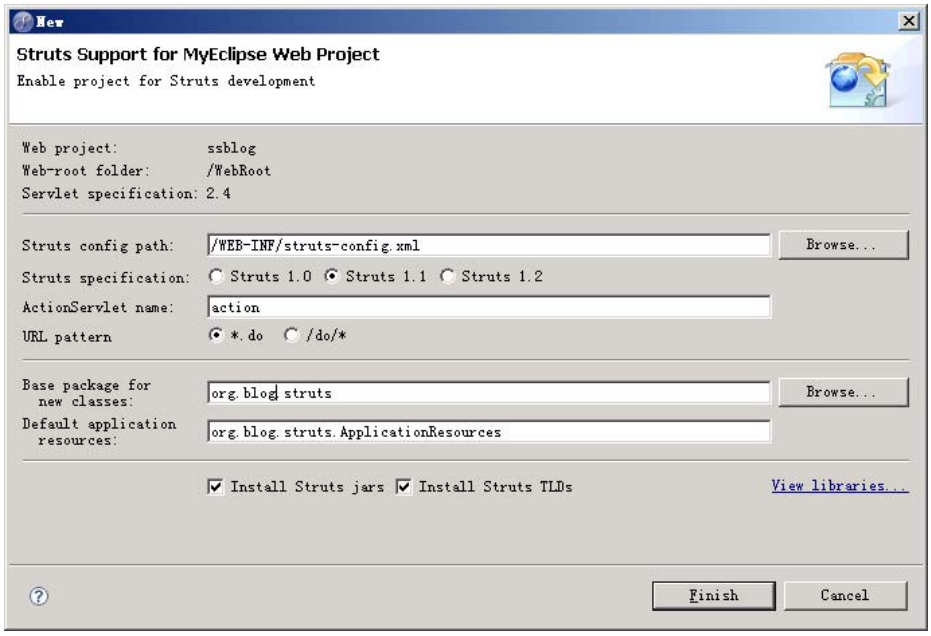

图 4-28 为 ssblog 添加 struts 能力

添加完后的 ssblog 工程雏形如图 4-29 所示：

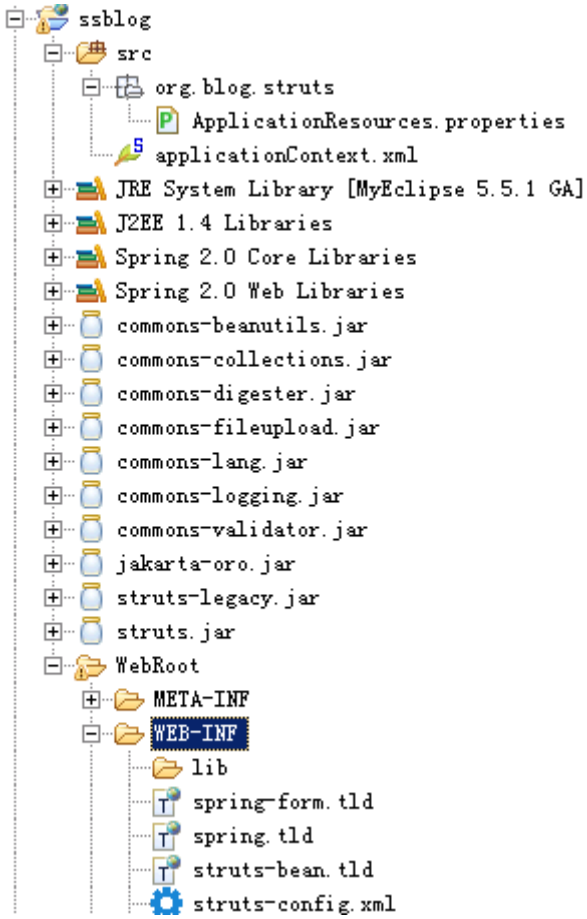

图 4-29 ssblog 工程

由于不再有 Hibernate 的支持，所以需要手动编写数据库访问类，这里采用传统的 JDBC 技术进行对数据的访问操作。当然在此之前需要将 mysql 的驱动 jar 导入到工程。如图 4-30 所示：

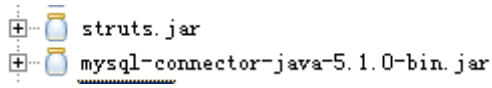

图 4-30 导入 mysql 驱动 jar 包

加入 struts 和 Spring 能力后，要使得它们能够工作还需要在 web.xml 中进行相关的配置，web.xml 文件内容如下所示：

```
<?xml version="4.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <!-- 配置 Spring 配置文件 -->
    <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/classes/applicationContext.xml
    </param-value>
  </context-param>
  <!-- 字符编码过滤器 -->
    <filter>
        <filter-name>charEncoder</filter-name>
        <filter-class>org.blog.services.CharEncoderFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>charEncoder</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

<!-- struts 配置 -->
    <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

```
<!-- 错误页面配置  -->
<error-page>
<error-code>500</error-code>
<location>/error.jsp</location>
</error-page>
</web-app>
```

在个配置文件中关于 struts 和 Spring 的配置读者并不陌生，至于其它的配置暂时不用关心，会在后面的开发过程中讲解到。下面讲解具体的开发。

4.2.2 编写数据库访问类

在 src 目录下新建一个 org.blog.database.DataBaseAccessImpl 类，这个类主要实现对数据的查询，变更（新增一条记录，删除一条记录，更新一条记录），它采用的是普通的 JDBC 方式连接数据库，基本流程是装载驱动，建立连接，建立语句，执行查询或者更新，然后关闭资源。代码如下所示：

```
package org.blog.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;



/**
 * 基于 JDBC 技术的数据库访问类
 *
 *
 */
public class DataBaseAccessImpl implements IDataBaseAccess{
 /**
  * 连接数据库的库名，用户，密码信息
  */
 private    String url ;
 private    String username;
 private    String password;
 private    String dateFormat;




 /**
  * 一个连接 一个对象只会产生一个连接
  */
```

```
private Connection con;

/**
 * 一个语句 每个操作都会产生一个语句
 */
private Statement stmt;
/**
 * 装载驱动 并确保系统启动后只装载一次。
 */
static {
	try {
		Class.forName("com.mysql.jdbc.Driver");

	} catch (Exception e) {
		System.out.println("DataBaseAccess:" + e.getMessage());
	}
}



/**
 * 建立连接
 *
 */
public void createConnection(){
	try {

		setCon(DriverManager.getConnection(url,username,password));
	} catch (Exception e) {
		System.out.println("DataBaseAccess:" + e.getMessage());
		System.exit(0);
	}
}

/**
 * 执行一个查询 sql
 * @param sqlstr
 * @return
 */
public String[][] execQuery(String sqlstr) {
	try {

		try{
		setStmt(getCon().createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
				ResultSet.CONCUR_UPDATABLE)); // 游标类型/并发组合
		}catch(Exception ee){
			ee.printStackTrace();
		}
		ResultSet rs = getStmt().executeQuery(sqlstr);
		ResultSetMetaData rsmd = rs.getMetaData();
```

```
                int collen=rsmd.getColumnCount();

                rs.last();
                String result[][]=new String[rs.getRow()][collen];
                   rs.beforeFirst();

                while(rs.next()){
                        for(int i=1;i<=collen;i++){
                        String data="";
                                Object obj=rs.getObject(i);
                                if(obj instanceof Date){
                                        Date ts=(Date)obj;
                                        SimpleDateFormat                                format=new
SimpleDateFormat(getDateFormat());
                                        data=format.format(ts);
                                 }
                                 else{
                                        try{

                                        data=(String)obj;
                                         }catch(Exception e){
                                                data=obj.toString();
                                         }

                                 }

                                result[rs.getRow()-1][i-1]=data;
                        }
                }


                return result;


        } catch (Exception e) {
                e.printStackTrace();
        }

        return new String[0][0];
    }









    public Connection getCon() {
        return con;
    }
```

```
public void setCon(Connection con) {
        this.con = con;
}

public Statement getStmt() {
        return stmt;
}

public void setStmt(Statement stmt) {
        this.stmt = stmt;
}
```

```
/**
 * 实现更新
 * @param sqlstr 更新语句
 * @return   成功为 true,失败为 false
 */
        public boolean execUpdate(String sqlstr){
                try{
                        getCon().setAutoCommit(false);
                        setStmt(getCon().createStatement());
                       getStmt().executeUpdate(sqlstr);
                        getCon().commit();
                        getCon().setAutoCommit(true);
                        return true;
                }catch(Exception e){
                        try{
                        getCon().rollback();
                        getCon().setAutoCommit(true);
                        }catch(Exception ee){
                                ee.printStackTrace();
                        }
                        e.printStackTrace();
                }finally{
                        try{
                                if(getStmt()!=null)
                                        getStmt().close();
                        }catch(Exception eee){
                                eee.printStackTrace();
                        }
                }
                return false;
        }
```

```
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getDateFormat() {
        return dateFormat;
    }

    public void setDateFormat(String dateFormat) {
        this.dateFormat = dateFormat;
    }









}
```

它所实现的接口 IdataBaseAccess 代码如下所示：

```
package org.blog.database;

/**
 * 数据库访问接口
 *
 *
```

```
 */
public interface IDataBaseAccess {
 /**
  * 创建一个数据库连接
  *
  */
 public void createConnection();
 /**
  * 执行一个查询 sql 语句，并将结果以二维数组的形式返回
  * @param sqlstr
  * @return 如果没有结果返回的是 0 行 0 列的二维数组，而不是 null
  */
 public String[][] execQuery(String sqlstr);
 /**
  * 执行一个更新 sql 语句，包括 insert,update,delete 等
  * @param sqlstr
  * @return 执行成功为 true 失败为 false
  */
 public boolean execUpdate(String sqlstr);
}
```

这个接口主要包括两个方法：

- String[][] execQuery(String sqlstr) 它只能接受一个查询 sql 语句，并将结果类型转换为字符串数组
- boolean execUpdate(String sqlstr) 执行一个变更 sql 语句，包括 insert，update，delete 语句。

为了可以轻松的从使用 Spring+Hibernate 的框架的 shblog 移植到使用 Struts+Spring 的 ssblog，尽管 shblog 中的 DAO 已经变更成使用 sql 方式的 DataBaseAccessImpl，但 POJO 类应该保留下来，代码如下所示：

```
package org.blog.database;

import java.util.Date;


//博客信息类
public class BlogInfo implements java.io.Serializable {
        public static final long serialVersionUID = 1L;//这条语句需要自动添加，它表示序列版本号，没有则会发生警告。
 // Fields

 private Integer id;//主键 id

 private String title;//文章标题

 private String content;//文章内容

 private Date modifyDate;//最后修改日期

 // Constructors
```

```
/** default constructor */
public BlogInfo() {
}

/** minimal constructor */
public BlogInfo(Integer id) {
        this.id = id;
}

/** full constructor */
public BlogInfo(Integer id, String title, String content, Date modifyDate) {
        this.id = id;
        this.title = title;
        this.content = content;
        this.modifyDate = modifyDate;
}

// Property accessors

public Integer getId() {
        return this.id;
}

public void setId(Integer id) {
        this.id = id;
}

public String getTitle() {
        return this.title;
}

public void setTitle(String title) {
        this.title = title;
}

public String getContent() {
        return this.content;
}

public void setContent(String content) {
        this.content = content;
}

public Date getModifyDate() {
        return this.modifyDate;
}

public void setModifyDate(Date modifyDate) {
        this.modifyDate = modifyDate;
```

```
    }

}
```

这个类没有任何变化。这个类和上面提到的 DataBaseAccessImpl 类可以充当 shblog 中 Hibernate 的角色。尽管从功能上来说是非常之寒碜的，但完全能满足对数据的基本操作需求。

4.2.3 编写业务逻辑类

在 shblog 中，业务逻辑类包含一个接口和一个接口的实现类，这里只需要修改接口实现类就行。接口代码如下所示：

```
package org.blog.services;

import org.blog.database.BlogInfo;

import java.util.List;

public interface IBlogServ {

 /**
  * 获得博客内容
  *
  * @return
  */
 public List getBlogContent();

 /**
  * 添加日志
  *
  * @param title
  * @param content
  */
 public void addContent(String title, String content);

 /**
  * 修改内容
  *
  * @param id
  * @param title
  * @param ChangedContent
  */
 public void modifyContent(String id, String title, String ChangedContent);

 /**
  * 删除一条日志
  *
  * @param id
  */
 public void deleteContent(String id);
```

```
    /**
     * 通过主键找对象
     *
     * @param id
     * @return
     */
    public BlogInfo getBlogInfoById(String id);

}
```

这个接口没有什么变化，也尽量不要修改否则不方便从 shblog 项目重构到 ssblog。

由于对数据库的访问不再是通过 Hibernate，因此必须修改这个接口的实现类，修改后的代码如下所示：

```
package org.blog.services;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.blog.database.BlogInfo;
import org.blog.database.IDataBaseAccess;



/**
 * 博客服务类，主要包含对博客日志的增，删，改，查服务
 *
 *
 */
public class BlogServImpl implements IBlogServ{
/**
 * 数据库操作类的访问接口
 */
 private IDataBaseAccess db;
 /**
  * 日期类型格式
  */
     private String dateFormat;


 /**
  * 获得博客内容
  * @return
  */
 public List getBlogContent() {
        try {
              List<BlogInfo> blogList=new ArrayList<BlogInfo>();
```

```
                String sqlstr="select * from blog_info";
                db.createConnection();
                String result[][]=db.execQuery(sqlstr);
                 SimpleDateFormat format=new SimpleDateFormat(getDateFormat());

            for (String[] blogString : result) {
                    BlogInfo blog=new BlogInfo();
                    blog.setId(Integer.parseInt(blogString[0]));
                    blog.setTitle(blogString[1]);
                    blog.setContent(blogString[2]);
                    blog.setModifyDate(format.parse(blogString[3]));
                    blogList.add(blog);
            }

            return blogList;
            } catch (Exception e) {
            e.printStackTrace();
            }
       return null;

       }
      /**
       * 添加日志
       * @param title
       * @param content
       */
       public void addContent(String title, String content) {
       SimpleDateFormat format=new SimpleDateFormat(getDateFormat());
       String now=format.format(new Date());

            String sqlstr="insert into blog_info(title,content,modify_date)
values('"+title+"','"+content+"','"+now+"')";
            db.createConnection();
            db.execUpdate(sqlstr);



       }
      /**
       * 修改内容
       * @param id
       * @param title
       * @param ChangedContent
       */
       public void modifyContent(String id, String title, String ChangedContent) {
      String sqlstr="update blog_info set title='"+title+"',content='"+ChangedContent+"' where
id="+id;
      db.createConnection();
      db.execUpdate(sqlstr);
```

```
 }
/**
 *  删除一条日志
 * @param id
 */
 public void deleteContent(String id) {
 String sqlstr="delete from blog_info where id="+id;
 db.createConnection();
 db.execUpdate(sqlstr);

 }
/**
 *  通过主键找对象
 * @param id
 * @return
 */
 public BlogInfo getBlogInfoById(String id) {
        BlogInfo blog=new BlogInfo();
        try {

                String sqlstr="select * from blog_info where id="+id;
                db.createConnection();
                String results[][]=db.execQuery(sqlstr);
                        blog.setId(new Integer(results[0][0]));
                        blog.setTitle(results[0][1]);
                        blog.setContent(results[0][2]);
                        SimpleDateFormat format=new SimpleDateFormat(getDateFormat());
                        blog.setModifyDate(format.parse(results[0][3]));

                        return blog;
        } catch (Exception e) {
        e.printStackTrace();
        }

 return null;
 }


public IDataBaseAccess getDb() {
 return db;
}
public void setDb(IDataBaseAccess db) {
 this.db = db;
}
public String getDateFormat() {
 return dateFormat;
}
public void setDateFormat(String dateFormat) {
 this.dateFormat = dateFormat;
```

```
    }


    }
```

可以看到，这个实现类的面貌较 Hibernate 自动生成的 DAO 完全不同，它是依靠 sql 方式调用前面编写的数据库访问类，达到数据操作目的的。由于它只是修改了实现方式，因此对于这个服务类的调用者控制器类（下一节内容）而言，不需要修改与之相关的代码，这是接口隔离的原因。

上面的服务类所具有的属性是通过 Spring 进行依赖注入的，因此需要在 Spring 的配置文件中进行配置，applicationContext.xml 文件内容如下所示：

```
    <?xml version="4.0" encoding="UTF-8"?>
    <beans
     xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <!-- 配置数据库操作类 -->
    <bean id="DataBaseAccessImpl" class="org.blog.database.DataBaseAccessImpl">
    <!-- 数据库连接 url -->
    <property name="url" value="jdbc:mysql://localhost:3305/shdb"></property>
    <!--  数据库连接用户名-->
    <property name="username" value="test"></property>
    <!--  数据库连接密码-->
    <property name="password" value="test"></property>
    <!--  日期类型格式-->
    <property name="dateFormat" value="yyyy-MM-dd"></property>

    </bean>

    <!-- 博客服务类 -->
    <bean id="BlogServImpl" class="org.blog.services.BlogServImpl">
    <property name="db">
    <ref bean="DataBaseAccessImpl"/>
    </property>
    <property name="dateFormat" value="yyyy-MM-dd" >

    </property>
    </bean>

    </beans>
```

这个配置文件中包含两个 bean，第一个是对 DataBaseAccessImpl 的配置，它将数据库连接 url，用户名，密码信息进行依赖注入。Hibernate 也是这么做的，目的是避免硬编码。第二个 bean 是这个应用的服务类，它具有一个属性 DataBaseAccessImpl。这个属性的值直接引用第一个 bean。

4.2.4 编写控制类

在 shblog 中的控制类是非常淳朴的，它们采用的是原始的 HttpServlet 技术，一个没有任何框架支持的控制器一般就会像它们一样编写，从功能角度而言，是

没有什么差别的，何况就算使用下面要讲的 Struts 框架所提供 Action 类或者是 Spring 所提供的 ActionSupport 类其根本也是采用 HttpServlet 技术

在 shblog 中，一共包含两个控制器：用户登陆控制器 LoginController 和博客操作控制器 BlogController。通过分析不难发现 BlogController 实际上是一个多动作控制器，这样做的一个明显的好处是减少类的量。但是却违背了单一职责（SRP）原则，鉴于此，在 ssblog 中将会对它进行解剖。下面一一讲解。

在 Struts 一章当中已经学习过如何在 Struts-config.xml 文件中向导式生成 Action,ActionForm 和 JSP.这里不再详述这个过程。生成用户登陆的 Action 后，在 Struts-config.xml 的设计模式下查看这个 Action 的属性如图 4-31 所示

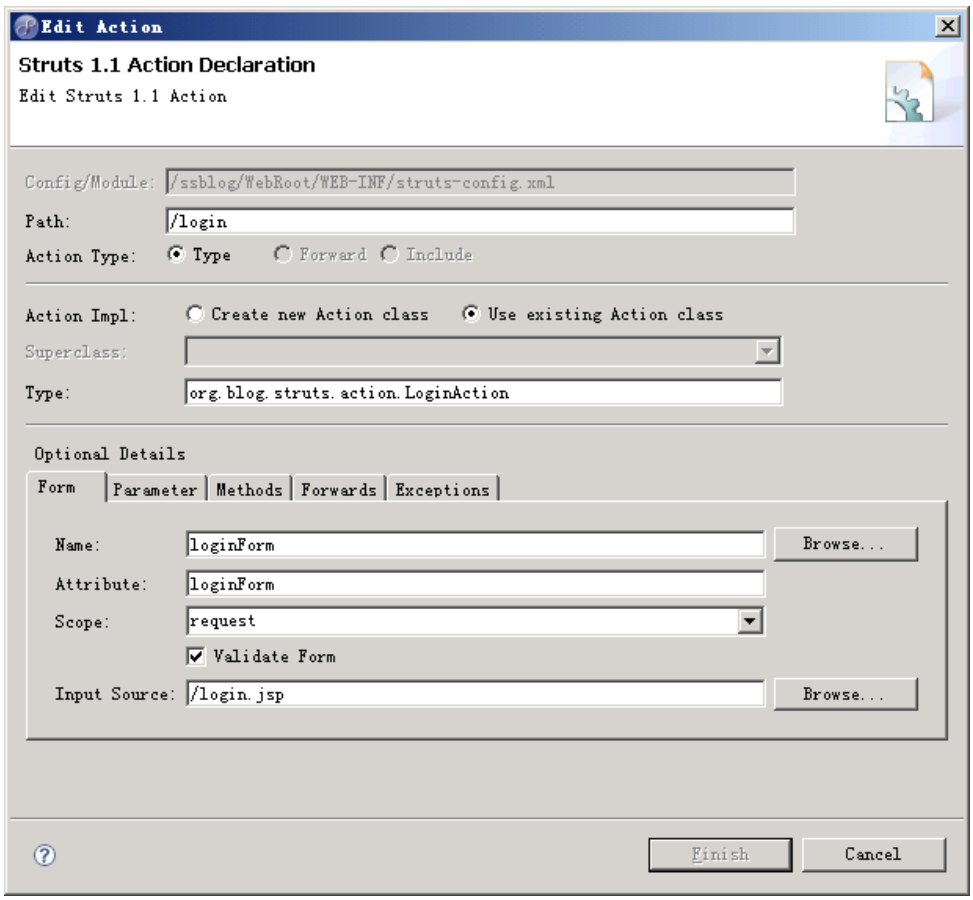

图 4-31LoginAction 属性

LoginAction 的代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.blog.services.IBlogServ;
import org.blog.struts.form.LoginForm;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 *  用户登陆控制器
 * XDoclet definition:
 *  @struts.action  path="/login"  name="loginForm"  input="/login.jsp"  scope="request"
validate="true"
 * @struts.action-forward name="failed" path="/failed.jsp"
 * @struts.action-forward name="blog" path="/blog.jsp"
 */
public class LoginAction extends ActionSupport {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
         HttpServletRequest request, HttpServletResponse response) {
     LoginForm loginForm = (LoginForm) form;// TODO Auto-generated method stub
     ApplicationContext context=getWebApplicationContext();
     IBlogServ  bs  =(IBlogServ)context.getBean("BlogServImpl");//从 Spring 环境中取
得 BlogServ 对象

     if (request.getParameter("type").equals("com")) {//如果登陆用户是游客
         try {

             request.setAttribute("content", bs.getBlogContent());//取出日志内容
             request.setAttribute("type", "com");//说明是游客
             return mapping.findForward("blog");//跳转到博客显示页面
         } catch (Exception e) {
             e.printStackTrace();
         }
         return null;
     }

     String username = loginForm.getUsername();//取得用户名
     String password = loginForm.getPassword();//取得密码

     if ( !username.equals("yxf") || !password.equals("123")) {//如果用户名和密码不都
正确（在实际开发中这里的用户名和密码应该取自数据库）
```

```
                try {
                    return mapping.findForward("failed");

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            else {
                try {
                    request.getSession().setAttribute("user", "yxf");// 将用户名存入到session中，它将用于别处
                    request.setAttribute("content", bs.getBlogContent());//取得内容
                    request.setAttribute("type", "admin");//说明是博客主人
                    return mapping.findForward("blog");    //博客内容显示页面
                } catch (Exception e) {
                    e.printStackTrace();
                }

            }

            return null;
        }
    }
```

对于这个LoginAction，可能让读者不能理解是为什么它所继承的不是在Struts中介绍的 Action 而是一个称为 ActionSupport 的类。

ActionSupport 是 Spring 框架所提供的一个将 Spring 和 Struts 整合到一起的控制类。继承 ActionSupport 最明显的好处是可以通过 getWebApplicationContext()方法直接获得 Spring 的容器。要使得这个类可以工作，还需要在 struts-config.xml 文件中编写一个插件配置。代码如下所示：

```
    <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
      <set-property property="contextConfigLocation" value="/WEB-INF/classes/applicationContext.xml" />
    </plug-in>
```

这个插件的目的是告诉 struts 关于 Spring 的存在，和存在何处。如果没有这个配置应用在运行期间将导致异常：

```
java.lang.IllegalStateException: No WebApplicationContext found: no ContextLoaderListener registered?
 org.springframework.web.context.support.WebApplicationContextUtils.getRequiredWebApplicationContext(WebA
 org.springframework.web.struts.DelegatingActionUtils.findRequiredWebApplicationContext(DelegatingActionUtil
 org.springframework.web.struts.ActionSupport.initWebApplicationContext(ActionSupport.java:98)
```

这个用户登陆 Action 对应的 ActionForm 代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.form;

import javax.servlet.http.HttpServletRequest;
```

```
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 *  用户登陆
 * XDoclet definition:
 * @struts.form name="loginForm"
 */
public class LoginForm extends ActionForm {
 public static final long serialVersionUID = 1L;
 /*
  * Generated fields
  */

 /** password property */
 private String password;

 /** username property */
 private String username;

 /*
  * Generated Methods
  */

 /**
  * Method validate
  * @param mapping
  * @param request
  * @return ActionErrors
  */
 public ActionErrors validate(ActionMapping mapping,
             HttpServletRequest request) {
        if(request.getParameter("type").equals("admin")){

             ActionErrors errors = new ActionErrors();

             if(username == null || username.trim().length() < 1)//用户名不能为空
                  errors.add("username", new ActionError("login.username.null"));
             if(password == null || password.trim().length() < 1)//用户密码不能为空
                  errors.add("password", new ActionError("login.password.null"));

             return errors;

        }
```

```
        return null;
    }

    /**
     * Method reset
     * @param mapping
     * @param request
     */
    public void reset(ActionMapping mapping, HttpServletRequest request) {
        // TODO Auto-generated method stub
    }

    /**
     * Returns the password.
     * @return String
     */
    public String getPassword() {
        return password;
    }

    /**
     * Set the password.
     * @param password The password to set
     */
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * Returns the username.
     * @return String
     */
    public String getUsername() {
        return username;
    }

    /**
     * Set the username.
     * @param username The username to set
     */
    public void setUsername(String username) {
        this.username = username;
    }
}
```

这是一个经典的用户登陆所使用到的 ActionForm，它包含最基本的用户名和密码这两个属性，一般而言对这两个属性应该要做非常充分的验证，这里的 LoginForm 仅对用户名和密码进行空值验证，读者可以完善这个验证工作。使用到的属性文件内容如下所示：

```
login.username.null=用户名为空
login.password.null=密码为空
```

上面的内容存在于 ApplicationResources.properties 文件中。

博客操作控制器在这里将被分解为：添加博客控制器 AddBlogAction，删除博客控制器 DeleteBlogAction 和编辑博客控制器 EditBlogAction。其中 AddBlogAction 的代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.action;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.blog.services.IBlogServ;
import org.blog.struts.form.AddBlogForm;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 * 功能描述：添加博客控制器，用户在表单中填充新增博客后调整到这个控制器
 * XDoclet definition:
 * @struts.action path="/addBlog" name="addBlogForm" input="/addBlog.jsp"
scope="request" validate="true"
 * @struts.action-forward name="blog" path="/blog.jsp"
 */
public class AddBlogAction extends ActionSupport {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
                 HttpServletRequest request, HttpServletResponse response) {
```

```
        AddBlogForm addBlogForm = (AddBlogForm) form;// TODO Auto-generated
method stub
        ApplicationContext context=getWebApplicationContext();//得到 Spring 容器
        IBlogServ bs=(IBlogServ)context.getBean("BlogServImpl");    //获得服务类
        bs.addContent(addBlogForm.getTitle(), addBlogForm.getContent());//调用服务类
提供的添加博客服务
        List content=bs.getBlogContent();//返回新的博客内容
        request.setAttribute("content",content);//封装在 request 对象中
            request.setAttribute("type", "admin");//以博客主人身份返回到博客页面

        return mapping.findForward("blog");//返回博客主页
    }
   }
```

这个控制器主要用来响应从视图层提交的添加博客表单请求，它有两个动作，一个是将新的内容插入到数据库，另一个是刷新博客主页。对应的 ActionForm 代码如下所示：

```
   /*
    * Generated by MyEclipse Struts
    * Template path: templates/java/JavaClass.vtl
    */
   package org.blog.struts.form;

   import javax.servlet.http.HttpServletRequest;

   import org.apache.struts.action.ActionError;
   import org.apache.struts.action.ActionErrors;
   import org.apache.struts.action.ActionForm;
   import org.apache.struts.action.ActionMapping;

   /**
    * MyEclipse Struts
    * Creation date: 11-17-2007
    *  功能描述：博客添加表单验证
    * XDoclet definition:
    * @struts.form name="addBlogForm"
    */
   public class AddBlogForm extends ActionForm {
    public static final long serialVersionUID = 1L;
    /*
     * Generated fields
     */

    /** title property */
    private String title;

    /** content property */
    private String content;
```

```
/*
 * Generated Methods
 */

/**
 * Method validate
 * @param mapping
 * @param request
 * @return ActionErrors
 */
public ActionErrors validate(ActionMapping mapping,
        HttpServletRequest request) {

    ActionErrors errors=new ActionErrors();
    if(title==null||title.trim().length()<1){//新增的博客标题不能为空
        errors.add("title", new ActionError("blog.title.null"));

    }
    if(content==null||content.trim().length()<1){//新增的博客内容不能为空
        errors.add("content", new ActionError("blog.content.null"));

    }



    return errors;
}

/**
 * Method reset
 * @param mapping
 * @param request
 */
public void reset(ActionMapping mapping, HttpServletRequest request) {
    // TODO Auto-generated method stub
}

/**
 * Returns the title.
 * @return String
 */
public String getTitle() {
    return title;
}

/**
 * Set the title.
 * @param title The title to set
 */
```

```
    public void setTitle(String title) {
        this.title = title;
    }

    /**
     * Returns the content.
     * @return String
     */
    public String getContent() {
        return content;
    }

    /**
     * Set the content.
     * @param content The content to set
     */
    public void setContent(String content) {
        this.content = content;
    }
}
```

这个 AddBlogForm 仅包括 title 和 content 属性，对其验证也只是进行简单的空值验证。其使用到的错误提示相关属性文件内容如下所示：

```
blog.title.null=标题不能为空
blog.content.null=内容不能为空
```

删除博客控制器 DeleteBlogAction 代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.action;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.blog.services.IBlogServ;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 *  功能描述：响应用户的删除博客的请求
 * XDoclet definition:
 * @struts.action
```

```
 * @struts.action-forward name="blog" path="/blog.jsp"
 */
public class DeleteBlogAction extends ActionSupport {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {

        ApplicationContext context=getWebApplicationContext();//获得 Spring 容器
        IBlogServ bs=(IBlogServ)context.getBean("BlogServImpl");//获得服务类
        bs.deleteContent(request.getParameter("id"));//或者日志 id

        List content=bs.getBlogContent();//获得新的日志
        request.setAttribute("content",content);//传递新的日志
        request.setAttribute("type", "admin");//以博客主人的身份
        return mapping.findForward("blog");//返回到博客主页
 }
}
```

这个控制器也是两个动作，一个是从数据库中删除指定 id 的博客，另一个是刷新博客主页，刷新的办法是从新获得新的博客内容，然后跳转到博客主页。这里没有使用到表单，所以没有 ActionForm。

编辑博客控制器 EditBlogAction 代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.action;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.blog.services.IBlogServ;
import org.blog.struts.form.EditBlogForm;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;
```

```
/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 * 功能描述：响应用户编辑博客请求
 * XDoclet definition:
 * @struts.action path="/editBlog" name="editBlogForm" input="/editBlog.jsp" scope="request" validate="true"
 * @struts.action-forward name="blog" path="/blog.jsp"
 */
public class EditBlogAction extends ActionSupport {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
         HttpServletRequest request, HttpServletResponse response) {
     EditBlogForm editBlogForm = (EditBlogForm) form;
     ApplicationContext context=getWebApplicationContext();//  获得 Spring 容器
     IBlogServ bs=(IBlogServ)context.getBean("BlogServImpl");//获得服务类
     bs.modifyContent(request.getParameter("id"), editBlogForm.getTitle(), editBlogForm.getContent());//获得修改服务

     List content=bs.getBlogContent();//获得新的博客列表
     request.setAttribute("content",content);//封装内容
     request.setAttribute("type", "admin");//以博客主人身份
     return mapping.findForward("blog");//返回到博客主页
 }
}
```

这个控制类与添加博客控制器非常类似，但是不同的是它是更新一条记录，而不是新增一条记录，它们的差别仅在于调用的服务不同。编辑博客也需要使用到表单，其对应的 ActionForm 代码如下所示：

```
/*
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.form;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionError;
```

```
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 * 功能描述：编辑博客
 * XDoclet definition:
 * @struts.form name="editBlogForm"
 */
public class EditBlogForm extends ActionForm {
 public static final long serialVersionUID = 1L;
 /*
  * Generated fields
  */

 /** title property */
 private String title;

 /** content property */
 private String content;

 /*
  * Generated Methods
  */

 /**
  * Method validate
  * @param mapping
  * @param request
  * @return ActionErrors
  */
 public ActionErrors validate(ActionMapping mapping,
                HttpServletRequest request) {

        ActionErrors errors=new ActionErrors();
        if(title==null||title.trim().length()<1){//修改后的博客标题不能为空
                errors.add("title", new ActionError("blog.title.null"));

        }
        if(content==null||content.trim().length()<1){//修改后的博客内容不能为空
                errors.add("content", new ActionError("blog.content.null"));

        }



        return errors;
 }
```

```
    /**
     * Method reset
     * @param mapping
     * @param request
     */
    public void reset(ActionMapping mapping, HttpServletRequest request) {
            // TODO Auto-generated method stub
    }

    /**
     * Returns the title.
     * @return String
     */
    public String getTitle() {
            return title;
    }

    /**
     * Set the title.
     * @param title The title to set
     */
    public void setTitle(String title) {
            this.title = title;
    }

    /**
     * Returns the content.
     * @return String
     */
    public String getContent() {
            return content;
    }

    /**
     * Set the content.
     * @param content The content to set
     */
    public void setContent(String content) {
            this.content = content;
    }
}
```

可以看得出来，这个 ActionForm 和添加博客 ActionForm 在内容上是没有什么区别的。

除了上面提到的 Action。在 ssblog 中还存在一个称为 GetEditPageAction 的控制器。博客主人在博客首页针对一项具体的日志进行编辑请求时，需要跳转到一个编辑页面，且此时的编辑页面中需要包含被编辑日志的标题和内容，那么这个过程需要一个控制器进行实现，于是就有了 GetEditPageAction。代码如下所示：

```
    /*
```

```
 * Generated by MyEclipse Struts
 * Template path: templates/java/JavaClass.vtl
 */
package org.blog.struts.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.blog.database.BlogInfo;
import org.blog.services.IBlogServ;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;

/**
 * MyEclipse Struts
 * Creation date: 11-17-2007
 *  功能描述：获得编辑页面
 * XDoclet definition:
 * @struts.action
 * @struts.action-forward name="edit" path="/editBlog.jsp"
 */
public class GetEditPageAction extends ActionSupport {
 /*
  * Generated Methods
  */

 /**
  * Method execute
  * @param mapping
  * @param form
  * @param request
  * @param response
  * @return ActionForward
  */
 public ActionForward execute(ActionMapping mapping, ActionForm form,
                 HttpServletRequest request, HttpServletResponse response) {
        ApplicationContext context=getWebApplicationContext();//获得 Spring 容器
        IBlogServ bs=(IBlogServ)context.getBean("BlogServImpl");//获得服务
        BlogInfo  blog=bs.getBlogInfoById(request.getParameter("id"));//获得被编辑博客
的 id
        request.setAttribute("blog",blog);//封装被编辑博客对象
        return mapping.findForward("edit");//跳转到编辑页面
 }
}
```

这个控制器首先获得被编辑博客的主键 id，然后调用服务类的 getBlogInfoById 方法获得一个 BlogInfo 对象，最后将这个对象传递给了博客编辑页面。

关于上面的所有 Action,和 ActionForm 都需要在 struts-config.xml 文件中进行配置，文件内容如下所示：

```
<?xml version="4.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 4.1//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_4.dtd">

<struts-config>
  <data-sources />
  <form-beans >
    <form-bean name="loginForm" type="org.blog.struts.form.LoginForm" />
    <form-bean name="addBlogForm" type="org.blog.struts.form.AddBlogForm" />
    <form-bean name="editBlogForm" type="org.blog.struts.form.EditBlogForm" />

  </form-beans>

  <global-exceptions />
  <global-forwards

  >
    <forward name="blog" path="/blog.jsp" />

  </global-forwards>

  <action-mappings >
    <action
      attribute="loginForm"
      input="/login.jsp"
      name="loginForm"
      path="/login"
      scope="request"
      type="org.blog.struts.action.LoginAction">
      <forward name="failed" path="/failed.jsp" />

    </action>
    <action
      attribute="addBlogForm"
      input="/addBlog.jsp"
      name="addBlogForm"
      path="/addBlog"
      scope="request"
      type="org.blog.struts.action.AddBlogAction">

    </action>
    <action
      attribute="editBlogForm"
      input="/editBlog.jsp"
      name="editBlogForm"
      path="/editBlog"
      scope="request"
      type="org.blog.struts.action.EditBlogAction">
```

```
        </action>
        <action
          path="/getEditPage"
          type="org.blog.struts.action.GetEditPageAction"
          validate="false">
          <forward name="edit" path="/editBlog.jsp" />
        </action>
        <action
          path="/deleteBlog"
          type="org.blog.struts.action.DeleteBlogAction"
          validate="false">

        </action>

      </action-mappings>

      <message-resources parameter="org.blog.struts.ApplicationResources" />

      <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
value="/WEB-INF/classes/applicationContext.xml" />
      </plug-in>

    </struts-config>
```

上面的内容除了前面提到的<plug-in>是手写方式添入的外，其它的配置信息都是通过 Struts-config.xml 文件设计模式下向导式开发过程中自动添加的。

4.2.5 编写视图层 JSP 页面

在视图上由于使用了 Struts 技术所以需要做一定的修改。使用 Struts 后的用户登陆页面 login.jsp，如下所示：

```
    <%@ page language="java" pageEncoding="gbk"%>
    <%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
    <%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

    <html>
     <head>
          <title>JSP for LoginForm form</title>

               <script type="text/javascript">
     function comLogin(){//游客登陆控制 js
     var login_form=document.forms[0];
     login_form.action="login.do?type=com";//修改表单的 action
     login_form.submit();//提交表单
     }
     </script>
     </head>
```

```
    <body>
        <html:form action="/login?type=admin" method="post">

            用 户 名 : <html:text property="username"/><html:errors property="username"/><br/>
            密 码 : <html:password property="password"/><html:errors property="password"/><br/>
            <html:submit value="登陆"/><button onclick="comLogin()">游客登陆</button>
        </html:form>
    </body>
</html>
```

这是一个仅含有两个字段的表单。它将提交给 LoginAction 控制器，如果用户登陆失败，将跳转到下面的 faild.jsp 代码如下所示：

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>My JSP 'failed.jsp' starting page</title>

 <meta http-equiv="pragma" content="no-cache">
 <meta http-equiv="cache-control" content="no-cache">
 <meta http-equiv="expires" content="0">
 <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
 <meta http-equiv="description" content="This is my page">
 <!--
 <link rel="stylesheet" type="text/css" href="styles.css">
 -->

  </head>

  <body>
用户名或者密码错误，请返回<a href="login.jsp">登陆页面</a>
  </body>
</html>
```

如果用户登陆时候的用户名和密码都正确那么用户单击登陆后就会跳转到成功页面，这里没有什么称之为成功页面的 jsp 文件，而是博客首页 blog.jsp。代码如下所示：

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk"%>

<jsp:directive.page import="java.text.SimpleDateFormat" />
<jsp:directive.page import="org.blog.database.BlogInfo;" />
<!-- 博客内容页面-->
<%
 String path = request.getContextPath();
 String basePath = request.getScheme() + "://"
                + request.getServerName() + ":" + request.getServerPort()
                + path + "/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
        <base href="<%=basePath%>">

        <title>My JSP 'blog.jsp' starting page</title>

        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <!--
 <link rel="stylesheet" type="text/css" href="styles.css">
 -->
        <!-- 定义样式 -->
        <style type="text/css">
.title{
background-color:"#000088";
color:"#FFFFFF";
}
a{
color:white;
}

</style>

 </head>

 <body>

        <%
                String type = (String) request.getAttribute("type");//取得用户类型，分为博客主人和游客两者
                List blogList = (List) request.getAttribute("content");//取得博客内容
                if (type.equals("admin") && session.getAttribute("user") != null) {//是博客主人
                        out
```

```
                    .print("<a href='addBlog.jsp' ><font color='red'> 添 加 新 日 志
</font></a>");//可以添加博客
                }
                out
                        .print("   <a href='login.jsp'><font color='red'>
退出登陆</font></a>");

                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");//格式
化日期输出
                for (int i = 0; i < blogList.size(); i++) {//依次输出博客内容
                    BlogInfo blog = (BlogInfo) blogList.get(i);//类型转化
                    String datestr = "";//代表日期的字符串
                    Date modifydate = blog.getModifyDate();//得到修改日期
                    if (modifydate != null) {//防止空值异常
                        datestr = format.format(modifydate);//格式日期
                    }
                    //输出博客标题和日期
                    out.print("<div class='title' id='" + i + "'>"
                    + blog.getTitle()
                    + "<span style='text-align:right; width:100%'>"
                    + datestr);
                    //如果是博客主人还可以对日志进行删除和编辑
                    if (type.equals("admin")
                    && session.getAttribute("user") != null) {
                        out.print("<a href='deleteBlog.do?id=" + blog.getId()
                        + "'>删除</a> ");
                        out.print("<a href='getEditPage.do?id=" + blog.getId()
                        + "' >编辑</a>");
                    }
                    out.print("</span></div>");
                    //输出日志内容
                    out.print("<div style='content'>" + blog.getContent()
                    + "</div>");
                }
            %>



     </body>
    </html>
```

这个博客首页有一个用户角色判断，如果登陆的用户为博客主人，则提供博客的添加，删除，编辑服务。添加页面 addBlog.js 代码 p 如下所示：

```
    <%@ page language="java" pageEncoding="gbk"%>
    <%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
    <%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

    <html>
     <head>
          <title>JSP for AddBlogForm form</title>
```

```
    </head>
    <body><br>
        <html:form action="/addBlog">
            标题: <html:text property="title"/><html:errors property="title"/><br/>
            内 容 : <html:textarea property="content" rows="15" cols="100"/><html:errors property="content"/><br/>
            <html:submit value="提交"/>
        </html:form>
    </body>
</html>
```

删除一条记录不需要设计视图页面，编辑一条记录也需要一个类似于上面这段代码所示的页面。编辑页面 editBlog.jsp 代码如下所示：

```
<%@ page language="java" pageEncoding="gbk"%>
<jsp:directive.page import="org.blog.database.BlogInfo"/>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

<html>
 <head>
    <title>JSP for EditBlogForm form</title>


 </head>
 <body>
 <% BlogInfo blog=(BlogInfo)request.getAttribute("blog");
 String id=blog.getId().toString();
 %>
    <html:form action="/editBlog" >
        标 题 : <html:text property="title" value="<%=blog.getTitle() %>"/><html:errors property="title"/><br/>
        内 容 : <html:textarea property="content" rows="15" cols="100" value="<%=blog.getContent() %>"/><html:errors property="content"/><br/>
        <html:hidden property="id" value="<%=id %>"/>
        <html:submit value="提交"/>
    </html:form >
 </body>
</html>
```

这个页面没什么特殊或者难以理解之处，所以不做过多的解释。

4.2.6 其它非功能组件

这里首先要介绍的一个有用的页面是 error.jsp 代码如下所示：

```
<%@ page language="java" import="java.util.*" pageEncoding="GB18030"%>
<%
String path = request.getContextPath();
String basePath =
```

```
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
    %>

    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
      <head>
        <base href="<%=basePath%>">

        <title>My JSP 'error.jsp' starting page</title>

     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
     <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
     <meta http-equiv="description" content="This is my page">
     <!--
     <link rel="stylesheet" type="text/css" href="styles.css">
     -->

      </head>

      <body>
    <h1> 出错啦！ </h1>
      </body>
    </html>
```

不错，它几乎没有任何内容仅含有一句出错信息，但是这里我们要将他派上大用场。众所周知，在使用 JSP+tomcat 开发 web 应用时候，如果后台程序出现编译错误，或者运行时异常，将会直接在浏览器的页面上打印非常难看的错误堆栈。类似于图 4-32 所示：

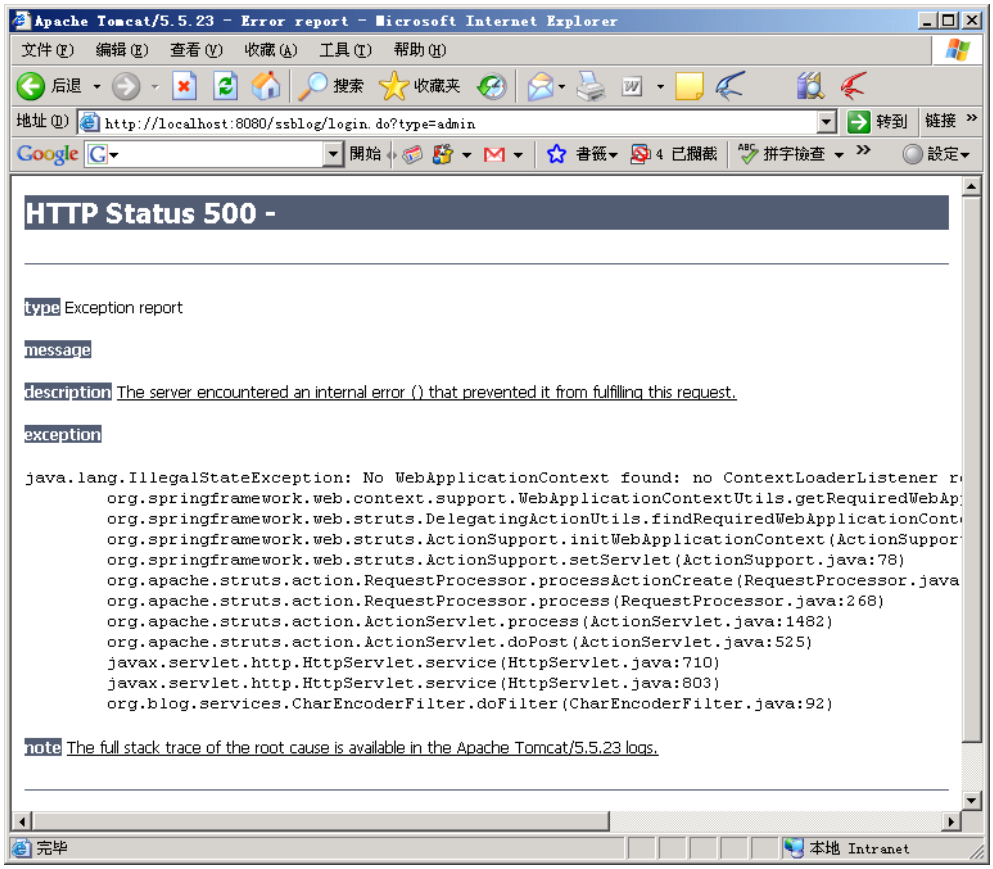

图 4-32 异常提示页面

这不仅仅是“面子上的问题”，更有可能引发安全事故。为了不在这样的页面上显示如此不友好的错误信息的一个办法就是编写一个类似于刚刚提到的 error.jsp。每次出现某种在 web.xml 文件配置的错误时，就会跳转到 error.jsp，于是图 4-32 取而代之的是图 4-33 所示的页面：

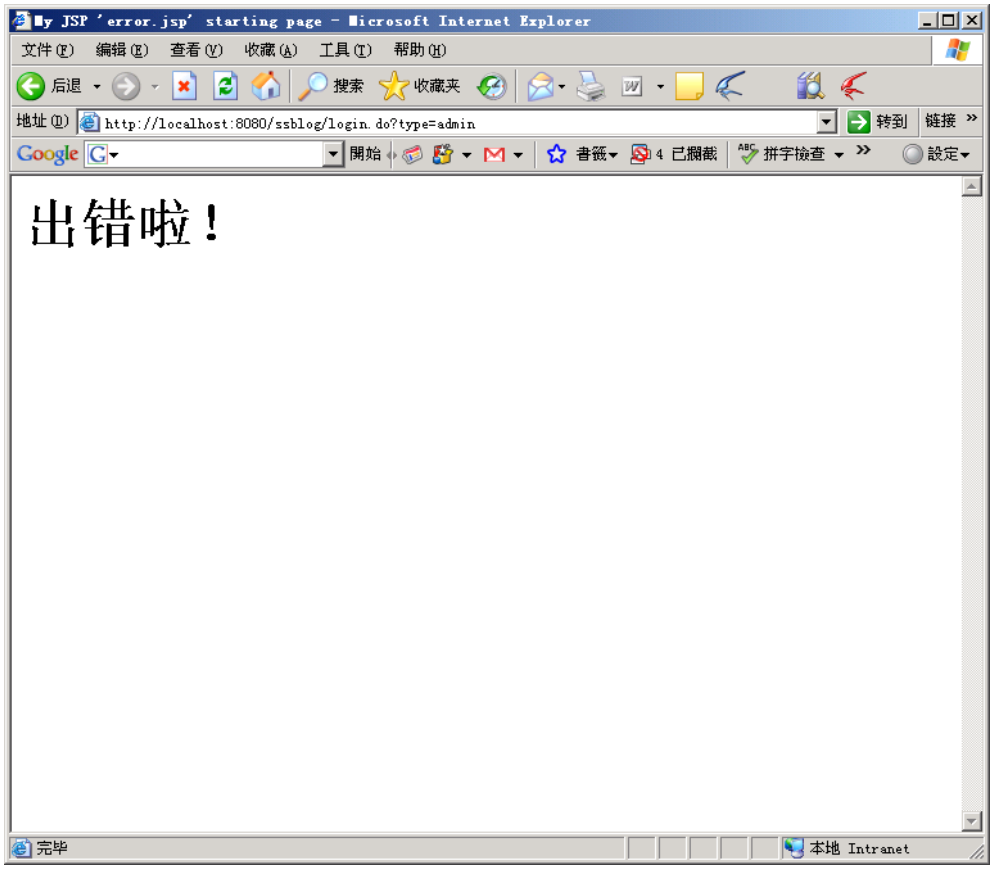

图 4-33 有好错误提示页面

上面提到要使得这个 error.jsp 达到这种功效，需要在 web.xml 进行相关配置，这里的配置内容如下所示：

```
<!-- 错误页面配置  -->

<error-page>
<error-code>500</error-code>
<location>/error.jsp</location>
</error-page>
```

这个配置表示当出现 500（编译错误）时 web 服务器自动跳转到 error.jsp 这个页面。

下面要介绍的是一个有用的过滤器，在 web 开发中，经常出现表单提交中文出现乱码，或者其它方式的乱码问题，这都是由于字符编码没有设置恰当成的，一个有效的解决办法是在需要的地方使用 request.setCharacterEncoding(encoding);

response.setCharacterEncoding(encoding);这样的语句。但如果要在每个需要的地方编写这样的代码，就显得非常冗余和修改困难。因此最佳办法是为这种需求编写一个过滤器。这个过滤器代码如下所示：

```
package org.blog.services;


import java.io.IOException;
import javax.servlet.Filter;
```

```
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
/**
 * 功能描述： 设置字符集编码过滤器<br>
 *
 *
 *
 */



public class CharEncoderFilter implements Filter {


    // ---------------------------------------------------- Instance Variables


    /**
     * The default character encoding to set for requests that pass through
     * this filter.
     */
    protected String encoding = "gbk";


    /**
     * The filter configuration object we are associated with.  If this value
     * is null, this filter instance is not currently configured.
     */
    protected FilterConfig filterConfig = null;


    /**
     * Should a character encoding specified by the client be ignored?
     */
    protected boolean ignore = true;


    // --------------------------------------------------------- Public Methods


    /**
     * Take this filter out of service.
     */
    public void destroy() {

        this.encoding = null;
        this.filterConfig = null;
```

```
        }


        /**
         * Select and set (if specified) the character encoding to be used to
         * interpret request parameters for this request.
         *
         * @param request The servlet request we are processing
         * @param result The servlet response we are creating
         * @param chain The filter chain we are processing
         *
         * @exception IOException if an input/output error occurs
         * @exception ServletException if a servlet error occurs
         */
        public void doFilter(ServletRequest request, ServletResponse response,
                                        FilterChain chain)
    throws IOException, ServletException {

            // Conditionally select and set the character encoding to be used
            if (ignore || (request.getCharacterEncoding() == null)) {
                String encoding = selectEncoding(request);

                if (encoding != null){
                    request.setCharacterEncoding(encoding);
                    response.setCharacterEncoding(encoding);
                }
            }



    // Pass control on to the next filter

            chain.doFilter(request, response);

        }


        /**
         * Place this filter into service.
         *
         * @param filterConfig The filter configuration object
         */
        public void init(FilterConfig filterConfig) throws ServletException {

    this.filterConfig = filterConfig;

                String configEncoding=filterConfig.getInitParameter("encoding");
            if(configEncoding!=null)//如果没有在 web.xml 文件中配置字符编码，则使用
默认
```

```
            this.encoding=configEncoding;

        String value = filterConfig.getInitParameter("ignore");
        if (value == null)
            this.ignore = true;
        else if (value.equalsIgnoreCase("true"))
            this.ignore = true;
        else if (value.equalsIgnoreCase("yes"))
            this.ignore = true;
        else
            this.ignore = false;

    }


    // ---------------------------------------------------- Protected Methods


    /**
     * Select an appropriate character encoding to be used, based on the
     * characteristics of the current request and/or filter initialization
     * parameters.  If no character encoding should be set, return
     * <code>null</code>.
     * <p>
     * The default implementation unconditionally returns the value configured
     * by the <strong>encoding</strong> initialization parameter for this
     * filter.
     *
     * @param request The servlet request we are processing
     */
    protected String selectEncoding(ServletRequest request) {

        return (this.encoding);

    }


}
```

这个过滤器默认使用 gbk 编码格式，如果在 web.xml 文件中指定了其它编码格式，则优先使用配置的编码格式。

最后需要对这个过滤器在 web.xml 中进行配置方能工作，配置内容如下所示：

```
<!-- 字符编码过滤器 -->
    <filter>
        <filter-name>charEncoder</filter-name>
        <filter-class>org.blog.services.CharEncoderFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>charEncoder</filter-name>
```

```
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

这个过滤器的 url-pattern 说明它拦截一切 HTTP 请求。

4.3 Struts 与 Hibernate 的整合

有了前两节的基础，再使用 Struts 和 Hibernate 整合开发前两节所实现的简易博客系统，只需要做少量调整即可。办法如下：

4.3.1Hibernate 逆向工程

为了和 shblog 区分，这里新建一个 web 工程名为 strutsHblog。使用前面所学，逆向工程 Hibernate 制品将产生如图 4-34 所示的代码清单：

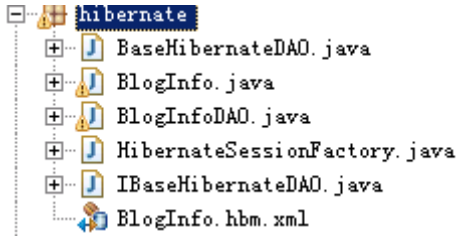

图 4-34Hibernate 制品

清单中还应该包括一个 hibernate.cfg.xml 文件。内容如下所示：

```
<?xml version='4.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

 <session-factory>
        <property name="connection.username">test</property>
        <property name="connection.url">
            jdbc:mysql://localhost:3305/shdb
        </property>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <property name="myeclipse.connection.profile">blogdb</property>
        <property name="connection.password">test</property>
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>

        <mapping resource="hibernate/BlogInfo.hbm.xml" />
 </session-factory>

</hibernate-configuration>
```

这段代码相信不需要再做过多解释。此外与 shblog 对比，这里新增了

BaseHibernateDAO 与 IBaseHbiernateDAO，但是 BlogInfo 和 BlogInfo.hbm.xml 文件的内容应该保持一致。最大的不同是 BlogInfoDAO，代码如下所示：

```
package hibernate;

import java.util.Date;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.LockMode;
import org.hibernate.Query;
import org.hibernate.criterion.Example;

/**
 * Data access object (DAO) for domain model class BlogInfo.
 *
 * @see hibernate.BlogInfo
 * @author MyEclipse Persistence Tools
 */

public class BlogInfoDAO extends BaseHibernateDAO {
 private static final Log log = LogFactory.getLog(BlogInfoDAO.class);

 // property constants
 public static final String TITLE = "title";

 public static final String CONTENT = "content";

 public void save(BlogInfo transientInstance) {
        log.debug("saving BlogInfo instance");
        try {
                getSession().save(transientInstance);
                log.debug("save successful");
        } catch (RuntimeException re) {
                log.error("save failed", re);
                throw re;
        }
 }

 public void delete(BlogInfo persistentInstance) {
        log.debug("deleting BlogInfo instance");
        try {
                getSession().delete(persistentInstance);
                log.debug("delete successful");
        } catch (RuntimeException re) {
                log.error("delete failed", re);
                throw re;
        }
 }

 public BlogInfo findById(java.lang.Integer id) {
```

```
        log.debug("getting BlogInfo instance with id: " + id);
        try {
            BlogInfo instance = (BlogInfo) getSession().get(
                    "hibernate.BlogInfo", id);
            return instance;
        } catch (RuntimeException re) {
            log.error("get failed", re);
            throw re;
        }
    }

    public List findByExample(BlogInfo instance) {
        log.debug("finding BlogInfo instance by example");
        try {
            List results = getSession().createCriteria("hibernate.BlogInfo")
                    .add(Example.create(instance)).list();
            log.debug("find by example successful, result size: "
                    + results.size());
            return results;
        } catch (RuntimeException re) {
            log.error("find by example failed", re);
            throw re;
        }
    }

    public List findByProperty(String propertyName, Object value) {
        log.debug("finding BlogInfo instance with property: " + propertyName
                + ", value: " + value);
        try {
            String queryString = "from BlogInfo as model where model."
                    + propertyName + "= ?";
            Query queryObject = getSession().createQuery(queryString);
            queryObject.setParameter(0, value);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find by property name failed", re);
            throw re;
        }
    }

    public List findByTitle(Object title) {
        return findByProperty(TITLE, title);
    }

    public List findByContent(Object content) {
        return findByProperty(CONTENT, content);
    }

    public List findAll() {
        log.debug("finding all BlogInfo instances");
```

```
        try {
            String queryString = "from BlogInfo";
            Query queryObject = getSession().createQuery(queryString);
            return queryObject.list();
        } catch (RuntimeException re) {
            log.error("find all failed", re);
            throw re;
        }
    }

    public BlogInfo merge(BlogInfo detachedInstance) {
        log.debug("merging BlogInfo instance");
        try {
            BlogInfo result = (BlogInfo) getSession().merge(detachedInstance);
            log.debug("merge successful");
            return result;
        } catch (RuntimeException re) {
            log.error("merge failed", re);
            throw re;
        }
    }

    public void attachDirty(BlogInfo instance) {
        log.debug("attaching dirty BlogInfo instance");
        try {
            getSession().saveOrUpdate(instance);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }

    public void attachClean(BlogInfo instance) {
        log.debug("attaching clean BlogInfo instance");
        try {
            getSession().lock(instance, LockMode.NONE);
            log.debug("attach successful");
        } catch (RuntimeException re) {
            log.error("attach failed", re);
            throw re;
        }
    }
}
```

通过这段代码可以看出，它与 shblog 中的 BlogInfoDAO 最大的不同之处是它们继承的父类不同。前者是一个 Spring 框架提供的 HibernateDaoSupport，后者是 BaseHibernateDAO，它实现了 IBaseHbiernateDAO 接口。因此后者也不再具有 getFromApplicationContext(ApplicationContext ctx)这样的方法，其它方法如 save 在实现方式上都有所区别。这将影响到后面的业务逻辑层代码

4.3.2 微调业务逻辑层代码

首先将 shblog 项目中的 org.blog.service 包内容复制到新的工程中。这时候将发现在 BlogServImpl 类中有错误提示，错误的原因是 DAO 的改变不再支持某些方法。修改 BlogServImpl 后的代码如下所示：

```
package org.blog.services;

import java.util.Date;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import hibernate.BlogInfo;
import hibernate.BlogInfoDAO;

/**
 * 博客服务类，主要包含对博客日志的增，删，改，查服务
 *
 *
 */
public class BlogServImpl implements IBlogServ{

/**
 * 对 hibernate 逆向工程制品 DAO 的引用
 */
 private BlogInfoDAO blogdao;



 /**
  * 获得博客内容
  * @return
  */
 public List getBlogContent() {
        return blogdao.findAll();//获得全部对象
 }
/**
 * 添加日志
 * @param title
 * @param content
 */
 public void addContent(String title, String content) {
        BlogInfo instance = new BlogInfo();//实例化一个博客对象
        //下面的设置没有包含 id，是因为采用了 increment 策略
        instance.setTitle(title);//设置新的标题
        instance.setContent(content);//设置内容
        instance.setModifyDate(new Date());//使用当前日期

        blogdao.save(instance);//持久化此对象
```

```
 }
/**
 *  修改内容
 * @param id
 * @param title
 * @param ChangedContent
 */
 public void modifyContent(String id, String title, String ChangedContent) {

        BlogInfo instance = blogdao.findById(Integer.parseInt(id));//通过主键找到对象
        instance.setTitle(title);//设置新标题
        instance.setContent(ChangedContent);//设置新内容
        instance.setModifyDate(new Date());//使用当前日期
        blogdao.getSession().update(instance);;//更新对象，并将这个游离对象持久化


 }
/**
 *  删除一条日志
 * @param id
 */
 public void deleteContent(String id) {
        BlogInfo instance = blogdao.findById(Integer.parseInt(id));//通过主键找到对象
        blogdao.delete(instance);//删除对象

 }
/**
 *  通过主键找对象
 * @param id
 * @return
 */
 public BlogInfo getBlogInfoById(String id) {
        return blogdao.findById(Integer.parseInt(id));//使用 DAO 的方法
 }
public BlogInfoDAO getBlogdao() {
 return blogdao;
}
public void setBlogdao(BlogInfoDAO blogdao) {
 this.blogdao = blogdao;
}

}
```

修改后的代码如黑体部分所示，这里需要使用事务，而之前的代码是 blogdao.getHibernateTemplate().update(instance);。

4.3.3 微调控制器代码

将 ssblog 中的 Struts 相关代码复制或者覆盖到新项目 strutsHblog 中，其中包括 org.blog.struts.action 包，org.blog.struts.form 包，struts-config.xml 文件，和

org.blog.struts 包，这时候会发现所有的 Action 都有错误提示，原因很简单,在 ssblog 中的所有 Action 都是继承自 Spring 框架的 ActionSupport 类。首先将它们都改回到 Struts 本身的 Action 类，但这样还不够。还需要将下面这段代码:

```
ApplicationContext context=getWebApplicationContext();//得到 Spring 容器
IBlogServ bs=(IBlogServ)context.getBean("BlogServImpl");    //获得服务类
```

改成如下的代码：

```
BlogServImpl bs=new BlogServImpl();
bs.setBlogdao(new BlogInfoDAO());
```

可以看到修改后的代码主要依靠 new 操作实现对象实例化，所需要的 DAO 属性也不能靠什么依赖注入。

最后不用修改 ActionForm，并将 ssblog 中的所有 jsp 文件和 4.2.6 节中讲述的非功能组件(CharEncoderFilter 和 error.jsp)以及 web.xml 文件直接添加或者覆盖到新的工程中相应的目录下，其中 web.xml 文件还需要删除与 Spring 相关的部分配置，如下所示：

```
<!-- 配置 Spring 配置文件 -->
    <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/classes/applicationContext.xml
    </param-value>
 </context-param>
```

此外还需要修改的是所有对 BlogInfo 的引用都要替换成 hibernate 包下的 BlogInfo。到此则完成了基于 Struts 和 Hibernate 框架的 strutsHblog 项目的开发。最终的代码 src 下的内容如图 4-35 所示

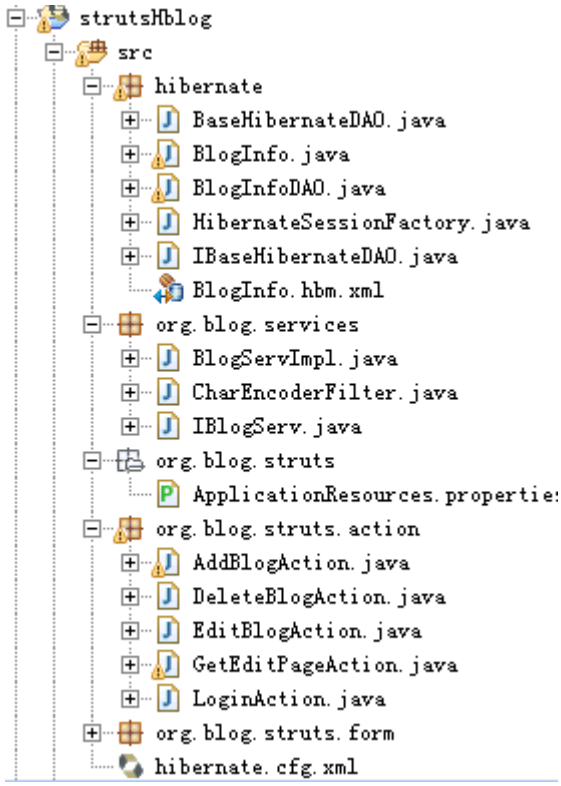

图 4-35strutsHblog 的 src 目录内容

webRoot 下的内容如图 4-36 所示：

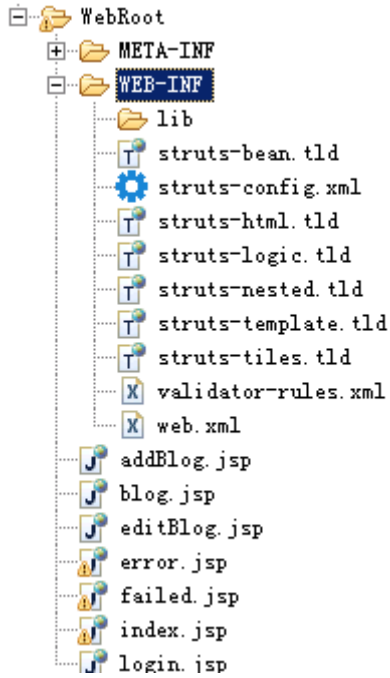

图 4-36strutsHblog 的 WebRoot 目录内容