

# 《Spring 从入门到精通》 清华大学出版社

<http://www.dearbook.com.cn/book/122892>

<http://www.china-pub.com/computers/common/info.asp?id=32510>



## 目录

### 第一篇 Spring 入门

第1章 Spring 概述.....	1
1.1 Spring 的历史.....	1
1.2 Spring 简介.....	1
1.2.1 Spring 框架介绍.....	1
1.2.2 Spring 的特点.....	2
1.3 如何学习 Spring.....	3
1.4 网络上的资源.....	3
1.5 小结.....	4
第2章 开始 Spring 之旅.....	5
2.1 建立 Spring 开发环境.....	5

2.1.1	下载 JDK .....	5
2.1.2	安装 JDK .....	5
2.1.3	设定 Path 与 Classpath .....	6
2.1.4	下载 Eclipse .....	6
2.1.5	配置 Eclipse .....	7
2.1.6	下载 Tomcat .....	7
2.1.7	设定 TOMCAT_HOME .....	7
2.1.8	下载 Eclipse 的 Tomcat 插件 .....	8
2.1.9	为 Eclipse 配置 Tomcat 插件 .....	8
2.1.10	下载 Spring .....	9
2.1.11	简单介绍 Spring 包 .....	10
2.1.12	在 Eclipse 中配置 Spring .....	10
2.2	第一个使用 Spring 实现 HelloWorld 的例子 .....	13
2.2.1	编写存放 HelloWorld 的 Java 文件 HelloWorld.java .....	13
2.2.2	配置 Spring 的 config.xml .....	14
2.2.3	编写测试程序 TestHelloWorld.java .....	15
2.2.4	运行测试程序并查看输出结果 .....	15
2.3	改写第一个 Spring 例子实现中英文输出 .....	16
2.3.1	编写接口文件 Hello.java .....	16
2.3.2	编写实现接口的两个类 ( ChHello、 EnHello ) .....	16
2.3.3	修改 Spring 的 config.xml .....	17
2.3.4	修改测试程序 TestHelloWorld.java .....	18
2.3.5	运行测试程序并查看输出结果 .....	18
2.4	小结 .....	19

## 第二篇 Spring 技术详解

第 3 章	Spring 基础概念 .....	20
3.1	反向控制/依赖注入 .....	20
3.1.1	反向控制 ( IoC ) .....	20
3.1.2	依赖注入 ( DI ) .....	25
3.2	依赖注入的 3 种实现方式 .....	25
3.2.1	接口注入 ( interface injection ) .....	25
3.2.2	Set 注入 ( setter injection ) .....	26
3.2.3	构造注入 ( constructor injection ) .....	26
3.3	将 HelloWorld 实例改为构造注入方式实现 .....	27
3.3.1	修改 HelloWorld.java .....	27
3.3.2	修改 config.xml .....	28
3.3.3	编写测试程序 TestHelloWorld.java .....	28
3.3.4	运行测试程序并查看输出结果 .....	28
3.4	使用哪种注入方式 .....	29
3.5	小结 .....	29

第 4 章 Spring 的核心容器 .....	30
4.1 什么是 Bean .....	30
4.2 Bean 的基础知识 .....	30
4.2.1 Bean 的标识 ( id 和 name ) .....	30
4.2.2 Bean 的类 ( class ) .....	31
4.2.3 Singleton 的使用 .....	32
4.2.4 Bean 的属性 .....	33
4.2.5 对于 null 值的处理 .....	35
4.2.6 使用依赖 depend-on .....	36
4.2.7 一个较为完整的 Bean 配置文档 .....	36
4.3 Bean 的生命周期 .....	37
4.3.1 Bean 的定义 .....	37
4.3.2 Bean 的初始化 .....	38
4.4.3 Bean 的使用 .....	41
4.4.4 Bean 的销毁 .....	42
4.4 用 ref 的属性指定依赖的 3 种模式 .....	45
4.4.1 用 local 属性指定 .....	45
4.4.2 用 bean 属性指定 .....	45
4.4.3 用 parent 属性指定 .....	46
4.4.4 3 种模式的比较 .....	46
4.5 Bean 自动装配的 5 种模式 .....	46
4.5.1 使用 byName 模式 .....	46
4.5.2 使用 byType 模式 .....	47
4.5.3 使用 constructor 模式 .....	48
4.5.4 使用 autodetect 模式 .....	49
4.5.5 使用 no 模式 .....	50
4.5.6 对 5 种模式进行总结 .....	51
4.6 Bean 依赖检查的 4 种模式 .....	51
4.6.1 为什么要使用依赖检查 .....	51
4.6.2 使用 simple 模式 .....	51
4.6.3 使用 object 模式 .....	52
4.6.4 使用 all 模式 .....	52
4.6.5 使用 none 模式 .....	52
4.6.6 对 4 种模式进行总结 .....	53
4.7 集合的注入方式 .....	53
4.7.1 List .....	53
4.7.2 Set .....	54
4.7.3 Map .....	54
4.7.4 Properties .....	55
4.7.5 对集合的注入方式进行总结 .....	56
4.8 管理 Bean .....	56
4.8.1 使用 BeanWrapper 管理 Bean .....	56
4.8.2 使用 BeanFactory 管理 Bean .....	57
4.8.3 使用 ApplicationContext 管理 Bean .....	58

4.8.4	3 种管理 Bean 的方式的比较.....	59
4.9	ApplicationContext 更强的功能.....	59
4.9.1	国际化支持.....	59
4.9.2	资源访问.....	62
4.9.3	事件传递.....	62
4.10	小结.....	65
第 5 章	Spring 的 AOP.....	66
5.1	AOP 基本思想.....	66
5.1.1	认识 AOP.....	66
5.1.2	AOP 与 OOP 对比分析.....	66
5.1.3	AOP 与 Java 的代理机制.....	67
5.2	从一个输出日志的实例分析 Java 的代理机制.....	67
5.2.1	通用的日志输出方法.....	67
5.2.2	通过面向接口编程实现日志输出.....	68
5.2.3	使用 Java 的代理机制进行日志输出.....	70
5.2.4	对这 3 种实现方式进行总结.....	72
5.3	AOP 的 3 个关键概念.....	72
5.3.1	切入点 (Pointcut) .....	72
5.3.2	通知 (Advice) .....	73
5.3.3	Advisor.....	73
5.4	Spring 的 3 种切入点 (Pointcut) 实现.....	73
5.4.1	静态切入点.....	74
5.4.2	动态切入点.....	74
5.4.3	自定义切入点.....	74
5.5	Spring 的通知 (Advice) .....	74
5.5.1	Interception Around 通知.....	75
5.5.2	Before 通知.....	75
5.5.3	After Returning 通知.....	75
5.5.4	Throw 通知.....	75
5.5.5	Introduction 通知.....	76
5.6	Spring 的 Advisor.....	76
5.7	用 ProxyFactoryBean 创建 AOP 代理.....	76
5.7.1	使用 ProxyFactoryBean 代理目标类的所有方法.....	76
5.7.2	使用 ProxyFactoryBean 代理目标类的指定方法.....	77
5.7.3	正则表达式简介.....	78
5.8	把输出日志的实例改成用 Spring 的 AOP 来实现.....	80
5.8.1	采用 Interception Around 通知的形式实现.....	80
5.8.2	采用 Before 通知的形式实现.....	84
5.8.3	采用 After Returning 通知的形式实现.....	87
5.8.4	采用 Throw 通知的形式实现.....	90
5.9	Spring 中 AOP 的 2 种代理方式.....	95
5.9.1	Java 动态代理.....	95
5.9.2	CGLIB 代理.....	95
5.10	Spring 中的自动代理.....	97

5.11	一个用 Spring AOP 实现异常处理和记录程序执行时间的实例.....	103
5.11.1	异常处理和记录程序执行时间的实例简介 .....	103
5.11.2	定义负责异常处理的 Advice 为 ExceptionHandler.java.....	104
5.11.3	定义记录程序执行时间的 Advice 为 TimeHandler.java.....	104
5.11.4	定义业务逻辑接口 LogicInterface.java.....	105
5.11.5	编写实现业务逻辑接口的类 Logic1.java.....	105
5.11.6	编写一个不实现业务逻辑接口的类 Logic2.java .....	106
5.11.7	使用自动代理定义配置文件 config.xml.....	107
5.11.8	编写测试类 Logic1 的程序 TestAop.java .....	108
5.11.9	输出自动代理时类 Logic1 异常处理和记录程序执行时间的信息 .....	108
5.11.10	编写测试类 Logic2 的程序 TestAop.java .....	109
5.11.11	输出自动代理时类 Logic2 异常处理和记录程序执行时间的信息 ..	109
5.11.12	使用 ProxyFactoryBean 代理定义配置文件 config.xml.....	110
5.11.13	编写测试类 Logic1 的程序 TestAop.java .....	111
5.11.14	输出 ProxyFactoryBean 代理时类 Logic1 记录程序执行时间的信息 .....	112
5.11.15	编写测试类 Logic2 的程序 TestAop.java .....	112
5.11.16	输出 ProxyFactoryBean 代理时类 Logic2 异常处理的信息 .....	113
5.12	小结.....	113
第 6 章	Spring 的事务处理 .....	114
6.1	简述事务处理 .....	114
6.1.1	事务处理的基本概念 .....	114
6.1.2	事务处理的特性 .....	114
6.1.3	对事务处理特性的总结 .....	115
6.2	事务处理的 3 种方式 .....	116
6.2.1	关系型数据库的事务处理 .....	116
6.2.2	传统的 JDBC 事务处理 .....	116
6.2.3	分布式事务处理 .....	117
6.3	Spring 的事务处理 .....	118
6.3.1	Spring 事务处理概述 .....	118
6.3.2	编程式事务处理 .....	119
6.3.3	声明式事务处理 .....	124
6.4	使用编程式还是声明式事务处理 .....	128
6.5	小结 .....	128
第 7 章	Spring 的持久层封装 .....	129
7.1	传统的 Jdbc 数据访问技术 .....	129
7.2	通过 XML 实现 DataSource (数据源) 注入 .....	130
7.2.1	使用 Spring 自带的 DriverManagerDataSource .....	130
7.2.2	使用 DBCP 连接池 .....	132
7.2.3	使用 Tomcat 提供的 JNDI .....	133
7.3	使用 JdbcTemplate 访问数据 .....	134
7.3.1	Template 模式简介 .....	134
7.3.2	回顾事务处理中 TransactionTemplate 的实现方式 .....	135
7.3.3	JdbcTemplate 的实现方式 .....	137

7.3.4	使用 JdbcTemplate 查询数据库 .....	144
7.3.5	使用 JdbcTemplate 更改数据库 .....	145
7.4	使用 ORM 工具访问数据 .....	146
7.4.1	ORM 简述 .....	146
7.4.2	使用 Hibernate .....	146
7.4.3	使用 iBATIS .....	150
7.5	小结 .....	153
第 8 章	Spring 的 Web 框架 .....	154
8.1	Web 框架介绍 .....	154
8.1.1	MVC 模式简介 .....	154
8.1.2	MVC 模式的结构图 .....	154
8.1.3	MVC 模式的功能示意图 .....	155
8.1.4	使用 MVC 模式的好处 .....	155
8.1.5	Model1 规范 .....	156
8.1.6	Model2 规范 .....	157
8.1.7	Spring MVC 的特点 .....	158
8.2	一个在 Jsp 页面输出 “HelloWord” 的 Spring MVC 实例 .....	158
8.2.1	配置 Web.xml .....	158
8.2.2	编写实现输出的 jsp 页面 index.jsp .....	160
8.2.3	编写控制器 HelloWorldAction.java .....	160
8.2.4	配置 Spring 文档 dispatcherServlet-servlet.xml .....	161
8.2.5	启动 Tomcat .....	162
8.2.6	运行程序 .....	162
8.2.7	把 index.jsp 改为使用 jstl .....	163
8.2.8	修改配置文档 dispatcherServlet-servlet.xml .....	163
8.2.9	运行修改后的程序 .....	164
8.2.10	使用 Log4j 时应该注意的问题 .....	165
8.3	Spring MVC 的模型和视图 (ModelAndView) .....	165
8.3.1	模型和视图 .....	166
8.3.2	Jstl 简介 .....	167
8.3.3	视图解析 .....	169
8.4	Spring MVC 的控制器 (Controller) .....	169
8.4.1	Controller 架构 .....	169
8.4.2	表单控制器 (SimpleFormController) .....	170
8.4.3	多动作控制器 (MultiActionController) .....	174
8.5	Spring MVC 的分发器 (DispatcherServlet) .....	180
8.5.1	分发器的定义方式 .....	180
8.5.2	分发器的初始化参数 .....	181
8.5.3	分发器的工作流程 .....	182
8.5.4	分发器与视图解析器的结合 .....	182
8.5.5	在一个 Web 应用中使用不同的视图层技术 .....	184
8.5.6	在 DispatcherServlet 中指定处理异常的页面 .....	185
8.6	处理器映射 .....	186
8.6.1	映射示例 .....	186

8.6.2	映射原理.....	187
8.6.3	添加拦截器.....	188
8.7	数据绑定.....	191
8.7.1	绑定原理.....	192
8.7.2	使用自定义标签.....	192
8.7.3	Validator 应用.....	196
8.8	本地化支持.....	200
8.8.1	在头信息中包含客户端的本地化信息.....	201
8.8.2	根据 cookie 获取本地化信息.....	201
8.8.3	从会话中获取本地化信息.....	203
8.8.4	根据参数修改本地化信息.....	203
8.9	一个用 Spring MVC 实现用户登录的完整示例.....	205
8.9.1	在 Eclipse 中建立 Tomcat 工程项目 myMVC.....	205
8.9.2	编写日志文件放在 myMVC/WEB-INF/src 下.....	206
8.9.3	配置 web.xml.....	207
8.9.4	编写登录页面 login.jsp.....	208
8.9.5	编写显示成功登录的页面 success.jsp.....	208
8.9.6	编写存放用户登录信息的 Bean.....	208
8.9.7	编写用户输入信息验证 UserValidator.java.....	209
8.9.8	编写用户登录逻辑 Login.java.....	210
8.9.9	编写配置文件 dispatcherServlet-servlet.xml.....	210
8.9.10	运行程序.....	212
8.9.10	国际化支持.....	213
8.9.11	运行国际化后的程序.....	216
8.10	小结.....	218
第 9 章	Spring 的定时器.....	219
9.1	定时器简述.....	219
9.2	定时器的 2 种实现方式.....	219
9.2.1	Timer.....	220
9.2.2	Quartz.....	221
9.2.3	两种方式的比较.....	223
9.3	利用 Spring 简化定时任务的开发.....	223
9.3.1	在 Spring 中使用 Timer 实现定时器.....	224
9.3.2	在 Spring 中使用 Quartz 实现定时器.....	225
9.4	小结.....	227

## 第三篇 Spring 与其他工具整合应用

第 10 章	Spring 与 Struts 的整合.....	228
10.1	Struts 介绍.....	228
10.1.1	Struts 的历史.....	228
10.1.2	Struts 的体系结构.....	228

10.2	Struts 的下载和配置 .....	229
10.2.1	下载 Struts .....	229
10.2.2	配置 Struts .....	229
10.3	一个在 Jsp 页面输出 “HelloWord” 的 Struts 实例.....	233
10.3.1	配置 web.xml.....	233
10.3.2	编写实现输出的 jsp 页面 index.jsp.....	234
10.3.3	编写控制器 HelloWorldAction.java .....	234
10.3.4	配置 Struts 文档 struts-config.xml .....	235
10.3.5	启动 Tomcat.....	235
10.3.6	运行程序.....	236
10.4	Struts 的几个主要类简介 .....	236
10.4.1	ActionServlet ( 控制器 ) .....	236
10.4.2	Action ( 适配器 ) .....	239
10.4.3	Action Mapping ( 映射 ) .....	240
10.4.4	ActionForm ( 数据存储 ) .....	243
10.4.5	DispatchAction ( 多动作控制器 ) .....	245
10.5	国际化支持.....	248
10.6	Struts 的自定义标签 .....	252
10.6.1	Bean 标签.....	253
10.6.2	Logic 标签 .....	254
10.6.3	Html 标签.....	255
10.7	Spring 与 Struts 整合的三种方式.....	257
10.7.1	通过 Spring 的 ActionSupport 类.....	257
10.7.2	通过 Spring 的 DelegatingRequestProcessor 类 .....	260
10.7.3	通过 Spring 的 DelegatingActionProxy 类 .....	264
10.7.4	比较三种整合方式.....	267
10.8	采用第三种整合方式编写一个用户注册的例子 .....	267
10.8.1	Spring 与 Struts 整合环境的配置 .....	268
10.8.2	编写 web.xml .....	271
10.8.3	编写用户注册画面 regedit.jsp .....	271
10.8.4	编写用户注册成功画面 success.jsp .....	272
10.8.5	编写用户类 User.java.....	272
10.8.6	编写 Struts 的配置文件 struts-config.xml .....	273
10.8.7	编写 Spring 的配置文件 config.xml .....	273
10.8.8	编写控制器 RegeditAction.java.....	274
10.8.9	编写业务逻辑接口 Regedit.java.....	275
10.8.10	编写具体的业务逻辑类 RegeditImpl.java .....	275
10.8.11	运行用户注册的例子 .....	276
10.9	小结.....	277
第 11 章	Spring 与 Hibernate 的整合 .....	278
11.1	Hibernate 介绍 .....	278
11.2	Hibernate 的下载和配置 .....	278
11.2.1	下载 Hibernate .....	278
11.2.2	配置 Hibernate .....	279



11.3	一个实现数据新增的 Hibernate 示例 .....	280
11.3.1	建立数据库表 .....	280
11.3.2	编写表对应的 POJO .....	282
11.3.3	编写 POJO 对应的 Xml .....	283
11.3.4	编写 Hibernate 的配置文件 .....	283
11.3.5	编写测试案例 .....	284
11.3.6	运行测试程序 .....	285
11.4	Hibernate 的配置 .....	285
11.5	Hibernate 的映射 .....	287
11.5.1	集合映射 .....	288
11.5.2	组件映射 .....	296
11.5.3	关联映射 .....	298
11.5.4	继承映射 .....	318
11.6	Hibernate 的工具 .....	321
11.6.1	从数据库到映射 .....	322
11.6.2	从映射到 POJO .....	329
11.7	Hibernate 的几个主要类简介 .....	332
11.7.1	Configuration ( 管理 Hibernate ) .....	332
11.7.2	SessionFactory ( 创建 Session ) .....	332
11.7.3	Session ( 提供 Connection ) .....	332
11.8	通过 xml 来整合 Spring 和 Hibernate .....	334
11.8.1	配置文件 Hibernate-Context.xml 的编写方式 .....	334
11.9	整合 Struts、Spring 和 Hibernate 实现用户注册的示例 .....	335
11.9.1	Spring、Struts 和 Hibernate 整合环境的配置 .....	336
11.9.2	编写 web.xml .....	340
11.9.3	编写用户注册画面 regedit.jsp .....	340
11.9.4	编写用户注册成功画面 success.jsp .....	341
11.9.5	建立数据库表结构 .....	341
11.9.6	生成映射文件 User.hbm.xml 和 POJO .....	341
11.9.7	编写接口 UserDAO.java 和实现类 UserDAOImpl.java .....	343
11.9.8	编写 Struts 的配置文件 struts-config.xml .....	344
11.9.9	编写 Spring 的配置文件 config.xml .....	345
11.9.10	编写控制器 RegeditAction.java .....	347
11.9.11	编写业务逻辑接口 Regedit.java .....	347
11.9.12	编写具体的业务逻辑类 RegeditImpl.java .....	348
11.9.13	运行用户注册的例子 .....	349
11.10	小结 .....	350
第 12 章	在 Spring 中使用 Ant .....	351
12.1	Ant 介绍 .....	351
12.2	Ant 的下载和安装 .....	351
12.2.1	下载 Ant .....	351
12.2.2	安装 Ant .....	352
12.3	在 Spring 中使用 Ant .....	353
12.3.1	在 Eclipse 中配置 Ant .....	353

12.3.2	建立 build.xml .....	353
12.3.3	运行 Ant.....	354
12.4	小结.....	355
第 13 章	在 Spring 中使用 Junit .....	356
13.1	Junit 介绍.....	356
13.2	Junit 的下载和安装.....	357
13.2.1	下载 Junit.....	357
13.2.2	安装 Junit.....	357
13.3	在 Spring 中使用 Junit .....	357
13.3.1	在 Eclipse 中配置 Junit .....	358
13.3.2	扩展 TestCase 类 .....	358
13.3.3	运行 Junit.....	359
13.4	使用 Junit 时常用的一些判定方法 .....	360
13.5	利用 Ant 和 Junit 进行自动化测试 .....	360
13.5.1	修改 build.xml .....	360
13.5.2	运行 Ant.....	362
13.5.3	自动生成测试报告.....	363
13.6	小结.....	364

## 第四篇 Spring 实例

第 14 章	用 Spring 实现新闻发布系统的例子 .....	365
14.1	新闻发布系统的介绍.....	365
14.2	检查环境配置.....	365
14.2.1	检查 JDK 配置 .....	365
14.2.2	检查 Tomcat 配置.....	366
14.2.3	检查 Ant 配置.....	367
14.3	在 Eclipse 下建立项目 myNews.....	367
14.3.1	在 Eclipse 下建立项目 .....	367
14.3.2	编写 Ant build 文件.....	371
14.3.3	配置 Web.xml 文件 .....	372
14.4	设计新闻发布系统.....	373
14.4.1	设计画面.....	373
14.4.2	设计持久化类.....	381
14.4.3	设计数据库.....	387
14.4.4	新闻发布系统在持久层的整体 UML 图 .....	387
14.5	编写新闻发布系统的 Jsp 页面.....	388
14.5.1	新闻发布的展示画面 show.jsp.....	388
14.5.2	发布新闻画面 release.jsp .....	389
14.5.3	用户注册画面 regedit.jsp .....	390
14.5.4	管理员登录页面 login.jsp .....	391
14.5.5	错误处理页面 error.jsp .....	392

14.6	建立数据库表并生成 xml 和 POJO .....	393
14.6.1	存放用户信息的数据库表 .....	394
14.6.2	存放新闻的数据库表 .....	396
14.6.3	存放新闻类别的数据库表 .....	397
14.6.4	存放用户权限的数据库表 .....	399
14.6.5	建立表之间的关系 .....	401
14.6.6	生成对应的 xml .....	404
14.6.7	生成 POJO .....	415
14.7	编写新闻发布系统的 VO 和 DAO .....	422
14.7.1	用户类 User.java .....	422
14.7.2	用户权限类 UsersAuthor.java .....	423
14.7.3	新闻类 News.java .....	424
14.7.4	新闻类别类 NewsType.java .....	425
14.7.5	用户 DAO 接口 UserDAO.java .....	426
14.7.6	新闻 DAO 接口 NewsDAO.java .....	426
14.7.7	新闻类别 DAO 接口 NewsTypeDAO.java .....	427
14.7.8	用户 DAO 实现类 UserDAOImpl.java .....	427
14.7.9	新闻 DAO 实现类 NewsDAOImpl.java .....	428
14.7.10	新闻类别 DAO 实现类 NewsTypeDAOImpl.java .....	429
14.8	编写新闻发布系统的业务逻辑类 .....	430
14.8.1	登录接口 Login.java .....	431
14.8.2	注册接口 Regedit.java .....	431
14.8.3	发布接口 Release.java .....	431
14.8.4	登录实现类 LoginImpl.java .....	432
14.8.5	注册实现类 RegeditImpl.java .....	432
14.8.6	发布实现类 ReleaseImpl.java .....	434
14.9	编写新闻发布系统的控制器类 .....	435
14.9.1	登录控制器类 LoginController.java .....	435
14.9.2	注册控制器类 RegeditController.java .....	437
14.9.3	发布控制器类 ReleaseController.java .....	438
14.9.4	显示控制器类 ShowController.java .....	440
14.10	编写辅助类 NewsUtil.java .....	441
14.11	编写配置文件 dispatcherServlet-servlet.xml .....	446
14.12	运行验证程序 .....	450
14.13	小结 .....	456

## 第 7 章 Spring 的持久层封装

在第 6 章，介绍了事务处理的一些理论知识，本章主要结合 Spring 的持久层来讲解一下如何使用 Spring 的事务处理。首先对传统的数据库连接进行一下介绍，然后再介绍使用 Spring 怎样来进行持久层封装。

### 7.1 传统的 Jdbc 数据访问技术

前面讲过，传统的 Jdbc 数据访问技术的一般的流程是：首先获取数据源，然后根据数据源获取数据连接，接着设定事务开始，执行相应的操作，最后执行成功则提交，执行失败则回滚。下面，通过示例来看 JDBC 中是怎么使用事务处理的，示例代码如下：

```
//***** TimeBook.java*****
Public Class HelloWorld {
    Connection conn = null;
    Statement stmt = null;
    try {
        //获取数据连接
        Class.forName(com.microsoft.jdbc.sqlserver.SQLServerDriver);
        conn = DriverManager.getConnection(jdbc:microsoft:sqlserver://localhost:1433/stdb,
admin, admin);
        //开始启动事务
        conn.setAutoCommit(false);
        stmt = conn.createStatement();
        //执行相应操作
        stmt.executeUpdate("insert into hello values(1,'gf', 'HelloWorld')");
        //执行成功则提交事务
        conn.commit();
    } catch (SQLException e) {
        if (conn != null) {
            try {
                //执行不成功，则回滚
                conn.rollback();
            } catch (SQLException ex) {
                System.out.println("数据连接有异常" + ex);
            }
        }
    } finally {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                System.out.println("执行操作有异常" + ex);
            }
        }
        if (conn != null) {
            try {
```

```

        conn.close();
    } catch (SQLException ex) {
        System.out.println("数据连接有异常" + ex);
    }
}
}
}

```

上面代码只是一个使用 Jdbc 连接的示例，在实际的应用中，是不会这样用的，一方面编写代码繁琐，一方面效率太低，而 Spring 在持久层这方面提供了更好的支持，对 Jdbc 进行了良好的封装。

注意：本章示例中使用的数据库表名为 hello，该表包含 3 个字段：id、name、msg。其中 id 为整型，其余 2 个都是字符串类型，以后不再重复说明。

## 7.2 通过 XML 实现 DataSource（数据源）注入

这里介绍 Spring 提供的 3 种通过 Xml 实现 DataSource（数据源）注入的方式：使用 Spring 自带的 DriverManagerDataSource、使用 DBCP 连接池和使用 Tomcat 提供的 JNDI。下面分别来进行介绍。

### 7.2.1 使用 Spring 自带的 DriverManagerDataSource

在第 6 章的例子中，所有示例的配置文档对于 DataSource 的注入，使用的都是 Spring 提供的 DriverManagerDataSource。使用 DriverManagerDataSource 的效率上和直接使用 Jdbc 没有多大的区别，使用 DriverManagerDataSource 的配置文档示例代码如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!--设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </property>
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <property name="name">
            <value>admin</value>
        </property>
        <property name="msg">
            <value>admin</value>
        </property>
    </bean>
    <!--设定 transactionManager -->
    <bean
                                                id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

```

```

        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- 示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
    </bean>
</beans>

```

配置文档中 id 为 helloDAO 的 Bean 的示例代码在第 6 章已经讲解过，这里只是把示例代码展示出来，以示过程的完整性。HelloDAO.java 的示例代码如下：

```

//***** HelloDAO.java*****
package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;

public class HelloDAO {
    private DataSource dataSource;
    private PlatformTransactionManager transactionManager;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }

    public int create(String msg) {
        TransactionTemplate transactionTemplate = new
TransactionTemplate(transactionManager);
        Object result = transactionTemplate.execute(
            new TransactionCallback() {
                public Object doInTransaction(TransactionStatus status) {
                    // 执行新增的操作，向数据库新增一笔记录
                    .....
                    // 返回值是 resultObject
                    return resultObject;
                }
            });
    }
}

```

## 7.2.2 使用 DBCP 连接池

Spring 也提供了对 DBCP 连接池的支持，可以直接在配置文档中配置 DBCP 数据库连接池，要在 Spring 中使用 DBCP 连接池，需要将 spring-framework-2.0-m1\lib\jakarta-commons 文件夹中的 commons-collections.jar、commons-dbc.jar 和 commons-pool.jar 用前面介绍的方法加入到 ClassPath 中，使用 DBCP 连接池的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <!-- 设定 dataSource -->
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
    <!-- 使用 sqlserver 数据库 -->
    <property name="driverClassName">
      <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
    </property>
    <!-- 设定 Url -->
    <property name="url">
      <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
    </property>
    <!-- 设定用户名 -->
    <property name="name">
      <value>admin</value>
    </property>
    <!-- 设定密码 -->
    <property name="password">
      <value>admin</value>
    </property>
  </bean>
  <!-- 设定 transactionManager -->
  <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>
  </bean>
  <!-- 示例中的一个 DAO -->
  <bean id="helloDAO" class="com.gc.action.HelloDAO">
    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>
    <property name="transactionManager">
      <ref bean="transactionManager"/>
    </property>
  </bean>
</beans>
```

HelloDAO 类的代码和上面的一样，不用改变，这里就不在展示了。

### 7.2.3 使用 Tomcat 提供的 JNDI

与使用 DBCP 连接池外相比，使用 Spring 来进行 Web 开发，更多的是使用 Web 容器提供的数据库连接池功能，这里以使用 Tomcat 容器为例，来讲解一下在 Spring 中，使用 Tomcat 提供的 JNDI 应该如何配置。首先要在 Tomcat 的 server.xml 中添加一下代码：

```
<Context path="/myApp" reloadable="true"
docBase="D:\eclipse\workspace\myApp" workDir="D:\eclipse\workspace\myApp\work" >
<Resource name="jdbc/opedb" auth="Container"
type="javax.sql.DataSource"
factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
driverClassName="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://localhost:1433/stdb"
<!--设定用户名-->
name="admin"
<!--设定密码-->
msg="admin"
<!--设定最大连接数-->
maxActive="10000"
<!--连接最大空闲时间-->
maxIdle="10000"
<!--连接最大等待时间-->
maxWait="10000"
removeAbandoned="true"
removeAbandonedTimeout="10"
logAbandoned="true"
/></Context>
```

然后 Spring 的配置文档的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
<!--设定 dataSource -->
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName">
<value>jdbc/opedb</value>
</property>
</bean>
<!--设定 transactionManager -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource">
<ref bean="dataSource"/>
</property>
</bean>
<!--示例中的一个 DAO -->
<bean id="helloDAO" class="com.gc.action.HelloDAO">
<property name="dataSource">
<ref bean="dataSource"/>
</property>
</bean>
```



```

        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
    </bean>
</beans>

```

同样，HelloDAO 的代码不用改变。

上面介绍的这 3 种实现 DataSource 注入的方式，给开发人员的 Jdbc 编程带来了极大的方便，主要是因为 Spring 对 Jdbc 进行了良好的封装。

## 7.3 使用 JdbcTemplate 访问数据

在介绍 JdbcTemplate 之前，有必要先介绍一下 Template 模式，因为在 Spring 中，这种模式有着大量的应用，比如前面讲过的 TransactionTemplate。然后再介绍如何使用 JdbcTemplate 查询、更新数据库。

### 7.3.1 Template 模式简介

Template 模式指的就是在父类中定义一个操作中算法的骨架或者说操作顺序，而将一些步骤的具体实现延迟到子类中。这个模式可能是目前最简单的模式了，这里以一个示例做说明。假如要设计一个事务处理类，这个类要处理事务，这个是确定的，但是要处理哪些事务却是不知道的，所以可以在父类中编写事务处理的骨架代码，指明事务处理的操作顺序，而具体的事务则可以延迟的子类中去实现。具体实现步骤如下：

(1) 在父类 Transaction 中编写事务处理的骨架代码，指明事务处理的操作顺序，Transaction.java 的示例代码如下：

```

//***** Transaction.java *****
public abstract class Transaction{
    //事务处理的骨架代码，指明事务处理的操作顺序
    public Object execute() throws TransactionException {
        this.transactionManager.getTransaction(this);
        Object result = null;
        try {
            result = doInTransaction();
        } catch (Error err) {
            // Transactional code threw error -> rollback
            this.transactionManager.rollback(this);
            throw err;
        }
        this.transactionManager.commit(this);
        return result;
    }
    //负责传入具体的事务
    Public abstract Object doInTransaction ();
}

```

(2) 在子类 SubTransaction 中编写具体要进行事务处理的代码，SubTransaction.java 的示例

代码如下：

```
/****** SubTransaction.java*****  
public class SubTransaction extends Transaction {  
    //负责传入具体的事务  
    Public Object doInTransaction () {  
        //具体对数据库进行新增、修改、删除的代码  
        .....  
    }  
}
```

(3) 这样当有具体的业务逻辑调用 SubTransaction.Execute() 方法时，就会进行事务处理。

注意：在子类中，是必须要延迟到子类进行的才在子类中做，否则没有必要在子类中做。而接口就无法像抽象类似的这样做，因为接口并不能具体实现任何操作。

### 7.3.2 回顾事务处理中 TransactionTemplate 的实现方式

然而 Spring 提供了另外一种实现 Template 模式的方法，利用接口回调函数，也是很有意思的，来看一下前面讲过的 Spring 的事务处理方式：

(1) TransactionTemplate 的示例代码如下：

```
/****** TransactionTemplate.java*****  
public class TransactionTemplate extends DefaultTransactionDefinition implements  
InitializingBean{  
    .....  
    //进行事务处理的骨架代码，指明了事务处理的顺序  
    public Object execute(TransactionCallback action) throws TransactionException {  
        TransactionStatus status = this.transactionManager.getTransaction(this);  
        Object result = null;  
        try {  
            result = action.doInTransaction(status);  
        }  
        catch (RuntimeException ex) {  
            // Transactional code threw application exception -> rollback  
            rollbackOnException(status, ex);  
            throw ex;  
        }  
        catch (Error err) {  
            // Transactional code threw error -> rollback  
            rollbackOnException(status, err);  
            throw err;  
        }  
        this.transactionManager.commit(status);  
        return result;  
    }  
    //事务处理回调时调用  
    private void rollbackOnException(TransactionStatus status, Throwable ex) throws  
TransactionException {  
        try {  
            this.transactionManager.rollback(status);  
        }  
    }  
}
```

```

        catch (RuntimeException ex2) {
            throw ex2;
        }
        catch (Error err) {
            throw err;
        }
    }
}

```

代码说明：

- ❑ 这里 TransactionTemplate 没有使用抽象类，在它的 execute ( ) 方法里定义的事务处理的骨架代码。
- ❑ 但是 execute ( ) 方法的 TransactionCallback 参数却是个接口，在这个接口中定义了 doInTransaction ( ) 方法。

(2) 接着来看一下 TransactionCallback 的实现，TransactionCallback.java 的示例代码如下：

```

//***** TransactionCallback.java*****
public interface TransactionCallback {
    Object doInTransaction(TransactionStatus status);
}

```

(3) 只要实现 TransactionCallback 接口，并在 doInTransaction ( ) 方法里编写具体要进行事务处理的代码就可以了。简单的示例代码如下：

```

//***** HelloDAO.java*****
package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;

public class HelloDAO {
    private DataSource dataSource;
    private PlatformTransactionManager transactionManager;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }

    public int create(String msg) {
        TransactionTemplate transactionTemplate = new
TransactionTemplate(transactionManager);
        Object result = transactionTemplate.execute(
            new TransactionCallback() {
                public Object doInTransaction(TransactionStatus status) {
                    // 执行新增的操作，向数据库新增一记录

```

```

        .....
        // 返回值是 resultObject
        return resultObject;
    });
}
}

```

代码说明：

- 在 HelloDAO 的 create ( ) 方法里，对 TransactionCallback 接口的 doInTransaction ( ) 方法进行了实现。

多多钻研 Spring 的源代码会对开发人员的编程能力有很大的提高，接下来将要讲解的 JdbcTemplate 也同样用类似的方式实现了 Template 模式。

### 7.3.3 JdbcTemplate 的实现方式

在前面的很多示例中，都会看到 JdbcTemplate 的身影，JdbcTemplate 封装了传统 Jdbc 的功能，却提供了更强大的功能，从名字可以看出 JdbcTemplate 也实现了 Template 模式，但它和 TransactionTemplate 又有了些不同。JdbcTemplate 的示例代码如下：

```

/***** JdbcTemplate.java *****/
public class JdbcTemplate extends JdbcAccessor implements JdbcOperations {
    .....
    //使用回调方法
    public Object execute(ConnectionCallback action) throws DataAccessException {
        Connection con = DataSourceUtils.getConnection(getDataSource());
        try {
            Connection conToUse = con;
            if (this.nativeJdbcExtractor != null) {
                // Extract native JDBC Connection, castable to OracleConnection or the like.
                conToUse = this.nativeJdbcExtractor.getNativeConnection(con);
            }
            else {
                // Create close-suppressing Connection proxy, also preparing returned
Statements.
                conToUse = createConnectionProxy(con);
            }
            return action.doInConnection(conToUse);
        }
        catch (SQLException ex) {
            // Release Connection early, to avoid potential connection pool deadlock
            // in the case when the exception translator hasn't been initialized yet.
            DataSourceUtils.releaseConnection(con, getDataSource());
            con = null;
            throw getExceptionTranslator().translate("ConnectionCallback", getSql(action), ex);
        }
        finally {
            DataSourceUtils.releaseConnection(con, getDataSource());
        }
    }
    //使用回调方法
}

```

```

public Object execute(StatementCallback action) throws DataAccessException {
    Connection con = DataSourceUtils.getConnection(getDataSource());
    Statement stmt = null;
    try {
        Connection conToUse = con;
        if (this.nativeJdbcExtractor != null &&
            this.nativeJdbcExtractor.isNativeConnectionNecessaryForNativeStatements()) {
            conToUse = this.nativeJdbcExtractor.getNativeConnection(con);
        }
        stmt = conToUse.createStatement();
        applyStatementSettings(stmt);
        Statement stmtToUse = stmt;
        if (this.nativeJdbcExtractor != null) {
            stmtToUse = this.nativeJdbcExtractor.getNativeStatement(stmt);
        }
        Object result = action.doInStatement(stmtToUse);
        SQLWarning warning = stmt.getWarnings();
        throwExceptionOnWarningIfNotIgnoringWarnings(warning);
        return result;
    }
    catch (SQLException ex) {
        // Release Connection early, to avoid potential connection pool deadlock
        // in the case when the exception translator hasn't been initialized yet.
        JdbcUtils.closeStatement(stmt);
        stmt = null;
        DataSourceUtils.releaseConnection(con, getDataSource());
        con = null;
        throw getExceptionTranslator().translate("StatementCallback", getSql(action), ex);
    }
    finally {
        JdbcUtils.closeStatement(stmt);
        DataSourceUtils.releaseConnection(con, getDataSource());
    }
}

//直接传入 sql 语句执行
public void execute(final String sql) throws DataAccessException {
    if (logger.isDebugEnabled()) {
        logger.debug("Executing SQL statement [" + sql + "]");
    }
    class ExecuteStatementCallback implements StatementCallback, SqlProvider {
        public Object doInStatement(Statement stmt) throws SQLException {
            stmt.execute(sql);
            return null;
        }
        public String getSql() {
            return sql;
        }
    }
    execute(new ExecuteStatementCallback());
}

.....
}

```

代码说明：

- ❑ 这里只是简单的罗列出 JdbcTemplate 的一些代码以供分析使用。
- ❑ 和 TransactionTemplate 类似，同样有 execute ( ) 方法，并且 execute ( ) 方法的产生是一个接口。
- ❑ 但和 TransactionTemplate 不同的是，JdbcTemplate 提供了更简单的操作方式，不需要在代码中使用回调方法，可以只是把 sql 语句传入，直接执行，当然，如果使用回调方法也是可以的。

上面介绍了 JdbcTemplate 的实现方式，那 JdbcTemplate 该如何使用呢？下面主要通过示例代码来进行讲解，实现思路是：首先编写配置文档，然后通过在程序中使用 JdbcTemplate，并和事务处理结合在一起，具体编写步骤如下：

( 1 ) Spring 配置文档的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!--设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!--使用 sqlserver 数据库 -->
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </property>
        <!--设定 Url -->
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <!--设定用户名-->
        <property name="name">
            <value>admin</value>
        </property>
        <!--设定密码-->
        <property name="msg">
            <value>admin</value>
        </property>
    </bean>
    <!--设定 transactionManager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!--示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
    </bean>
```

```
</beans>
```

因为在第 6 章进行过讲解，所以这里不再解释配置文档中的具体含义。

(2) 在类 HelloDAO 中使用 JdbcTemplate，并和事务处理结合在一起，HelloDAO.java 的示例代码如下：

```
/****** HelloDAO.java*****  
package com.gc.action;  
  
import javax.sql.DataSource;  
import org.springframework.jdbc.core.*;  
import org.springframework.transaction.*;  
import org.springframework.transaction.support.*;  
import org.springframework.dao.*;  
  
public class HelloDAO {  
    private DataSource dataSource;  
    private PlatformTransactionManager transactionManager;  
  
    public void setDataSource(DataSource dataSource) {  
        this.dataSource = dataSource;  
    }  
  
    public void setTransactionManager(PlatformTransactionManager transactionManager) {  
        this.transactionManager = transactionManager;  
    }  
    //使用 JdbcTemplate  
    public int create(String msg) {  
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();  
        TransactionStatus status = transactionManager.getTransaction(def);  
        try {  
            JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);  
            jdbcTemplate.update("INSERT INTO hello VALUES(1, 'gf', 'HelloWord')");  
        } catch (DataAccessException ex) {  
            // 也可以执行 status.setRollbackOnly();  
            transactionManager.rollback(status);  
            throw ex;  
        } finally {  
            transactionManager.commit(status);  
        }  
    }  
}
```

代码说明：

- 首先把配置文档中定义的 dataSource，通过 JdbcTemplate 的构造方法进行注入，然后直接执行 JdbcTemplate 的 update（）方法即可实现对数据库的操作，是不是比传统的 Jdbc 方式简单多了，只用了 2 行代码就实现了执行数据库的操作。

(3) 开发人员还可以把配置文档中定义的 dataSource，通过 JdbcTemplate 的构造方法进行注入也省掉，直接把 JdbcTemplate 在配置文档中配置，并使 JdbcTemplate 依赖于 dataSource，配置文档的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!-- 设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!-- 使用 sqlserver 数据库 -->
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </property>
        <!-- 设定 Url -->
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <!-- 设定用户名 -->
        <property name="name">
            <value>admin</value>
        </property>
        <!-- 设定密码 -->
        <property name="msg">
            <value>admin</value>
        </property>
    </bean>
    <!-- 设定 transactionManager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- 设定 jdbcTemplate -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- 示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name=" jdbcTemplate ">
            <ref bean=" jdbcTemplate "/>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
    </bean>
</beans>

```

代码说明：

- ❑ 这里把 jdbcTemplate 通过配置文档定义，把 dataSource 注入到 jdbcTemplate 中。
- ❑ 把 jdbcTemplate 通过配置注入到 HelloDAO 中。

(4) 在类 HelloDAO 中使用 JdbcTemplate，但是不用编写代码把 dataSource 注入到 jdbcTemplate 中，并和事务处理结合在一起，HelloDAO.java 的示例代码如下：

```
//***** HelloDAO.java*****
```



```

package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;
import org.springframework.jdbc.core.JdbcTemplate;

public class HelloDAO {
    private JdbcTemplate jdbcTemplate;
    private PlatformTransactionManager transactionManager;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }
    //这里把 jdbcTemplate 通过依赖注入来实现
    public int create(String msg) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
        try {
            jdbcTemplate.update("INSERT INTO hello VALUES(1, 'gf', 'HelloWord')");
        } catch (DataAccessException ex) {
            // 也可以执行 status.setRollbackOnly();
            transactionManager.rollback(status);
            throw ex;
        } finally {
            transactionManager.commit(status);
        }
    }
}

```

这里只有 1 行代码就实现了对数据库的操作，由此可以看出 Spring IoC 功能的强大，通过依赖注入，大大简化了开发人员的编码工作量，而且在 Spring 的配置文档中配置也是非常容易的。

(5) 开发人员甚至还可以 sql 语句也通过配置文档来进行配置，这样当需要更改 sql 语句时，就不需要改变代码了，而只需要修改配置文档。通过配置文档进行 sql 语句注入的示例代码如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!--设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!--使用 sqlserver 数据库 -->
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>

```

```

        </property>
        <!-- 设定 Url -->
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <!-- 设定用户名-->
        <property name="name">
            <value>admin</value>
        </property>
        <!-- 设定密码-->
        <property name="msg">
            <value>admin</value>
        </property>
    </bean>
    <!-- 设定 transactionManager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- 设定 jdbcTemplate -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!-- 示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name=" jdbcTemplate ">
            <ref bean=" jdbcTemplate "/>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
        <property name="sql">
            <value> INSERT INTO hello VALUES(1, 'gf', 'HelloWord')</value>
        </property>
    </bean>
</beans>

```

代码说明：

□ 把 sql 语句通过 IoC 注入到 HelloDAO 中。

( 6 ) HelloDAO.java 的示例代码如下：

```

//***** HelloDAO.java*****
package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;

```

```

import org.springframework.jdbc.core.JdbcTemplate;

public class HelloDAO {
    private JdbcTemplate jdbcTemplate;
    private PlatformTransactionManager transactionManager;
    private String sql;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }

    public void setSql(String sql) {
        this.sql = sql;
    }
    //把 sql 语句也通过依赖注入的方式来实现
    public int create(String msg) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
        try {
            jdbcTemplate.update(this.sql);
        } catch (DataAccessException ex) {
            transactionManager.rollback(status); // 也可以执行 status.setRollbackOnly();
            throw ex;
        } finally {
            transactionManager.commit(status);
        }
    }
}

```

上述 3 种方式都实现了同样的功能，当业务逻辑调用类 HelloDAO 中的 create ( ) 方法时，都会向数据库增加一笔数据，而且代码量逐渐的减少，因为在 Spring 的配置文档中都通过依赖注入来进行设置。

### 7.3.4 使用 JdbcTemplate 查询数据库

通过对 JdbcTemplate 源代码的研读，可以看到，在 JdbcTemplate 提供了很多用来查询的数据库，比如：queryForMap ( ) queryForLong ( ) queryForInt ( ) queryForList ( ) 等等。这里只是简单的进行示例说明，使用 queryForList 查询的示例代码如下：

```

List rows = jdbcTemplate.queryForList("select * from hello");
Iterator it = rows.iterator();
while(it.hasNext()) {
    Map map = (Map) it.next();
    String id = map.get("id");
    String name = map.get("name");
    String msg = map.get("msg");
}

```

使用 queryForInt 查询获得 hello 表中记录数量的示例代码如下：

```
int count = jdbcTemplate.queryForInt("select count(*) from hello");
```

这里只简单介绍这 2 个查询的使用方法，如果想获得更多的查询功能，研读 JdbcTemplate 的源代码是很有用的。

### 7.3.5 使用 JdbcTemplate 更改数据库

使用 JdbcTemplate 的 update ( ) 方法是进行数据库更改常用的方式。比如要往数据库插入一笔数据，则可以使用以下示例代码：

```
jdbcTemplate.update("inset into hello values('1, 'gf','HelloWorld '");
```

可以使用下面的这种示例代码来实现同样的功能：

```
jdbcTemplate.update("inset into hello values (?, ?, ?)",  
    new Object[] {1,'gf', 'HelloWorld'});
```

还可以使用下面的这种示例代码来实现同样的功能：

```
jdbcTemplate.update("inset into hello values (?, ?, ?)",  
    new PreparedStatementSetter() {  
        public void setValues(PreparedStatement ps) throws SQLException {  
            ps.setInt(1, 1);  
            ps.setString(2, 'gf');  
            ps.setString(3, 'HelloWorld');  
        }  
    });
```

对数据库的修改，同样可以使用上面的方法：

```
jdbcTemplate.update("update hello set name = 'gd', msg = 'HelloWorld' where id = 1");
```

可以使用下面的这种示例代码来实现同样的功能：

```
jdbcTemplate.update("update hello set name = ?, msg = ? where id = ?",  
    new Object[] {'gf', 'HelloWorld',1});
```

注意：新增和修改时 Object[] 中参数的先后顺序是不一样的。

## 7.4 使用 ORM 工具访问数据

前面讲述了使用 JdbcTemplate 对数据库进行操作的方法，可是很多情况下，开发人员使用 JdbcTemplate 还是有一定的麻烦。Spring 还提供了与其他 ORM 工具结合的功能，本节就主要讲述 Spring 与 Hibernate、iBATIS 的无缝结合。在讲解 Spring 与 Hibernate、iBATIS 的结合前，先来看一下什么是 ORM。

## 7.4.1 ORM 简述

假如读者对 ORM 已经有一定的了解，则可以跳过这一节。

ORM 的英文全称是 Object-Relational Mapping，中文名称是对象关系映射。

为什么要使用 ORM 呢？是因为开发人员使用的技术的面向对象技术，而使用的数据库却是关系型数据库。使用 SQL 带来了很大的麻烦，可是目前面向对象的数据库还没有完全成熟，那么有什么办法既使用 SQL 又能去掉或减少这些麻烦呢？使用 ORM，通过在对象和关系型之间建立起一座桥梁。Hibernate、iBATIS 就是这样的 ORM 工具。

ORM 包括以下四部分：

- ❑ 一个对持久类对象进行 CRUD 操作的 API。
- ❑ 一个语言或 API 用来规定与类和类属性相关的查询。
- ❑ 一个规定 mapping metadata 的工具。
- ❑ 一种技术可以让 ORM 的实现同事务对象一起进行 dirty checking, lazy association fetching 以及其他的优化操作。

ORM 模型的简单性简化了数据库查询过程。使用 ORM 查询工具，用户可以访问期望数据，而不必理解数据库的底层结构，开发人员不再需要关心底层数据库是怎么工作的了，甚至不需要知道数据库的结构，一切都交给 ORM 去管理了。

## 7.4.2 使用 Hibernate

Hibernate 也是一个开源的框架，主要应用在持久层方面。关于 Hibernate，在后面的章节中还有详细的介绍，本节只是简单的列出示例代码，让读者快速了解 Spring 和 Hibernate 结合在一起的应用。这里仍然介绍如何配置 Spring 的配置文档和 HelloDAO 如何编写，另外因为使用 Hibernate，需要增加一个 Hibernate 的配置文件 Hello.hbm.xml 和存放数据的类 Hello。具体步骤如下：

(1) 加入 Hibernate 后的 Spring 配置文档的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!--设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!--使用 sqlserver 数据库 -->
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </property>
        <!--设定 Url -->
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <!--设定用户名-->
        <property name="name">
            <value>admin</value>
        </property>
        <!--设定密码-->
```

```

        <property name="msg">
            <value>admin</value>
        </property>
    </bean>
    <!--使用 Hibernate 的 sessionFactory -->
    <bean id="sessionFactory"
class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref local="dataSource" />
        </property>
        <property name="mappingResources">
            <list>
                <value>com/gc/action/ Hello.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    net.sf.hibernate.dialect.SQLServerDialect
                </prop>
                <prop key="hibernate.show_sql">
                    true
                </prop>
            </props>
        </property>
    </bean>
    <!--设定 transactionManager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
    <!--示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
    </bean>
</beans>

```

代码说明：

- Hibernate 中通过 SessionFactory 创建和维护 Session，而 Spring 对 SessionFactory 的配置进行了整合。
  - com/gc/action/ Hello.hbm.xml，表示 Hello.hbm.xml 放在包 com.gc.action 下。
- (2) 把配置文件 Hello.hbm.xml 放在包 com.gc.action 下，Hello.hbm.xml 的示例代码如下：

```

<hibernate-mapping>
<class name="com.gc.action.Hello" table="hello" dynamic-update="false" dynamic-insert="false">
<id name="id" column="id" type="java.lang.Integer"/>

```

```

<property name="name"
type="java.lang.String"
update="true"
insert="true"
access="property"
<!--栏位名称为 msg-->
column="msg"
<!--字段长度为 50-->
length="50"/>
<property name="msg"
type="java.lang.String"
update="true"
insert="true"
access="property"
<!--栏位名称为 name-->
column="name"
<!--字段长度为 50-->
length="50" />
</class>
</hibernate-mapping>

```

( 3 ) 把类 Hello 放在包 com.gc.action 下 , Hello.java 的示例代码如下 :

```

/***** Hello.java *****/
/**
 * @hibernate.class table="hello"
 */
public class Hello {
    public Integer id;
    public String name;
    public String msg;
    /**
     * @hibernate.id
     * column="id"
     * type="java.lang.Integer"
     */
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    /**
     * @hibernate.property column="msg" length="50"
     */
    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }
    /**
     * @hibernate.property column="name" length="50"

```

```

    */
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

(4) 在类 HelloDAO 中使用 SessionFactory，并和事务处理结合在一起，HelloDAO.java 的示例代码如下：

```

//***** HelloDAO.java*****
package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;
import org.springframework.orm.*;

public class HelloDAO {
    private SessionFactory sessionFactory;
    private PlatformTransactionManager transactionManager;

    public void setSessionFactory(DataSource sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }
    //使用 HibernateTemplate 代替 JdbcTemplate
    public int create(String msg) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
        try {
            HibernateTemplate hibernateTemplate = new HibernateTemplate(sessionFactory);
            Hello hello = new Hello();
            hello.setId(1);
            hello.setName("gf");
            hello.setMsg("HelloWorld");
            hibernateTemplate.saveOrUpdate(hello);
        } catch (DataAccessException ex) {
            transactionManager.rollback(status); // 也可以执行 status.setRollbackOnly();
            throw ex;
        } finally {
            transactionManager.commit(status);
        }
    }
}

```



上述代码实现的功能和前面使用 JdbcTemplate 是一样的。

### 7.4.3 使用 iBATIS

iBATIS 也是一个开源的框架，和 Hibernate 一样主要应用在持久层方面。关于 iBATIS，在后面的章节中还有详细的介绍，本节只是简单的列出示例代码，让读者快速了解 Spring 和 iBATIS 结合在一起的应用。这里仍然介绍如何配置 Spring 的配置文档和 HelloDAO 如何编写，另外因为使用 iBATIS，需要增加一个 iBATIS 的配置文件 sqlMapConfig.xml 和 Hello.xml 和存放数据的类 Hello。具体步骤如下：

(1) 加入 iBATIS 后的 Spring 配置文档的示例代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <!--设定 dataSource -->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <!--使用 sqlserver 数据库 -->
        <property name="driverClassName">
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </property>
        <!--设定 Url -->
        <property name="url">
            <value>jdbc:microsoft:sqlserver://localhost:1433/stdb</value>
        </property>
        <!--设定用户名-->
        <property name="name">
            <value>admin</value>
        </property>
        <!--设定密码-->
        <property name="password">
            <value>admin</value>
        </property>
    </bean>
    <!--使用 iBATIS-->
    <bean id="sqlMap" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocation">
            <value>WEB-INF/sqlMapConfig.xml</value>
        </property>
    </bean>
    <!--设定 transactionManager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
    </bean>
    <!--示例中的一个 DAO -->
    <bean id="helloDAO" class="com.gc.action.HelloDAO">
        <property name="dataSource">
```

```

        <ref bean="dataSource"/>
    </property>
    <property name="transactionManager">
        <ref bean="transactionManager"/>
    </property>
    <property name="sqlMap">
        <ref bean="sqlMap"/>
    </property>
</bean>
</beans>

```

代码说明：

□ sqlMapConfig.xml，是 iBATIS 的配置文件，放在 WEB-INF 下。

( 2 ) sqlMapConfig.xml 的示例代码如下：

```

<sqlMapConfig>
    <sqlMap resource="com/gc/action/Hello.xml"/>
</sqlMapConfig>

```

( 3 ) 把配置文件 Hello.xml 放在包 com.gc.action 下，Hello.xml 的示例代码如下：

```

<sqlMap namespace="Hello">
    <typeAlias alias="hello" type="com.gc.action.Hello" />
    <insert id="insertHello" parameterClass="hello">
        insert into hello ( id,name, msg)  values ( #id#,#name#,#msg# )
    </insert>
</sqlMap>

```

( 4 ) 类 Hello 和前面 Hibernate 使用的一样，Hello.java 的示例代码如下：

```

//***** Hello.java*****
public class Hello {
    public Integer id;
    public String name;
    public String msg;

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

    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {

```

```

        this.name = name;
    }
}

```

(5) 使类 HelloDAO 继承，并和事务处理结合在一起，HelloDAO.java 的示例代码如下：

```

//***** HelloDAO.java*****
package com.gc.action;

import javax.sql.DataSource;
import org.springframework.jdbc.core.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.*;
import org.springframework.dao.*;
import org.springframework.orm.*;

public class HelloDAO extends SqlMapClientDaoSupport {
    private PlatformTransactionManager transactionManager;
    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }
    //通过 getSqlMapClientTemplate 来代替 JdbcTemplate
    public int create(String msg) {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        TransactionStatus status = transactionManager.getTransaction(def);
        try {
            Hello hello = new Hello();
            hello.setId(1);
            hello.setName("gf");
            hello.setMsg("HelloWorld");
            getSqlMapClientTemplate().update("insertHello", hello);
        } catch (DataAccessException ex) {
            transactionManager.rollback(status); // 也可以执行 status.setRollbackOnly();
            throw ex;
        } finally {
            transactionManager.commit(status);
        }
    }
}

```

上述代码实现的功能和前面使用 JdbcTemplate 以及 Hibernate 是一样的，读者可以仔细比较一下它们之间的用法。

## 7.5 小结

本章讲述了 Spring 对 Jdbc 的封装，其核心是 JdbcTemplate，又讲了 Spring 与 Hibernate、iBATIS 的无缝结合，主要目的是让读者了解 Spring 对持久层的封装，并学会 JdbcTemplate 和事务处理结合在一起使用的方法。

通过前面几章的介绍，读者对 Spring 的全貌有了大概的了解，当然 Spring 目前应用最广泛的应该是进行 Web 开发了，在接下来的这一章，将主要介绍 Spring 的 Web 框架。

## 第 14 章 用 Spring 实现新闻发布系统的例子

前面的 13 章内容，都是对 Spring 和其相关工具的介绍，通过对这些内容的讲述，目的是让读者对 Spring 有一个全面的了解。前面讲过，实现实例是对 Spring 最好的理解方式，本章就是通过一个实例来对 Spring 进行一个整体的演示，主要使用 Spring 和 Hibernate 来实现新闻发布系统的实例，从而使读者对 Spring 和 Hibernate 有一个全面的掌握。

因为 Spring 的 MVC 完全可以代替 Struts，所以这里只做 Spring 和 Hibernate 的整合，而不再考虑 Struts。

### 14.1 新闻发布系统的介绍

这个实例是一个新闻发布系统，主要包括用户的注册、权限的划分、新闻类别的设定、新闻的发布和新闻的浏览。首先注册用户，然后对用户进行授权，没有设定权限的用户只能浏览新闻，授权用户可以发布新闻，浏览新闻，普通用户不用授权即可浏览新闻。

具体实现步骤如下：

### 14.2 检查环境配置

前面的章节中，都是在实例中进行环境配置的，这里再从头进行一下总结。很多时候程序没有跑通，都是因为环境配置的问题造成的。

这里主要检查 JDK、Tomcat、Spring、Hibernate、Ant 是否配置成功。

#### 14.2.1 检查 JDK 配置

具体的安装步骤可以参看第 2 章的介绍，查看 JDK 是否配置成功，可以通过 cmd 命令框来进行检查，在 cmd 命令框中输入 java，如果出现 java 相关选项的介绍信息，则说明 JDK 安装成功，如图 14.1 所示：

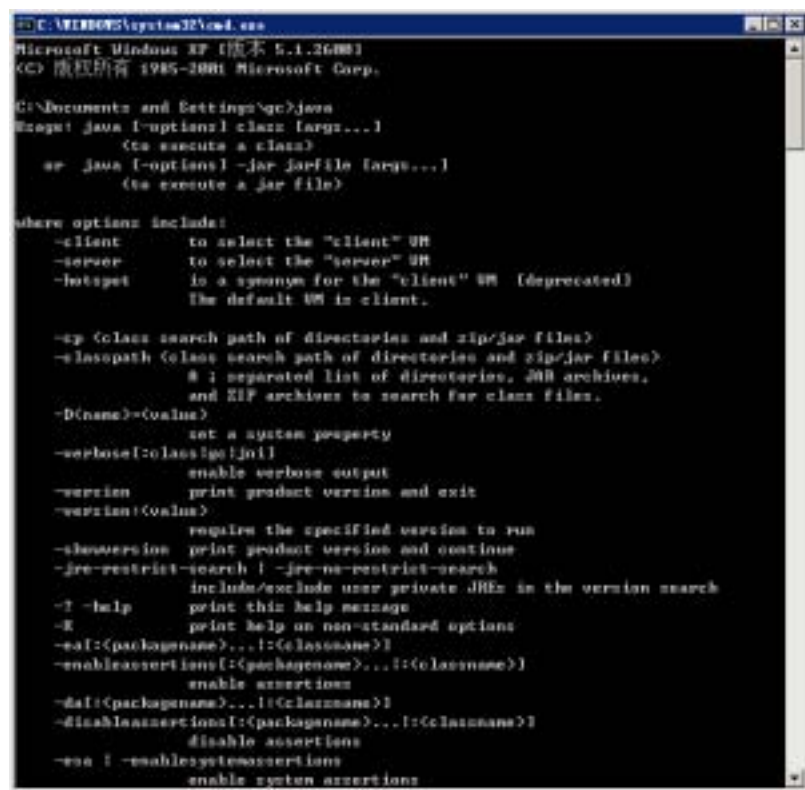


图 14.1 出现 java 相关选项的介绍信息

## 14.2.2 检查 Tomcat 配置

具体的安装步骤可以参看第 2 章的介绍，Tomcat 启动后，在 IE 地址栏中输入 <http://localhost:8080>，即可测试是否配置成功，Tomcat 启动成功的画面如图 14.2 所示。



图 14.2 Tomcat 启动成功的画面

### 14.2.3 检查 Ant 配置

在 cmd 命令框中输入 ant 命令，即可查看 Ant 是否安装成功，如图 14.3 所示。

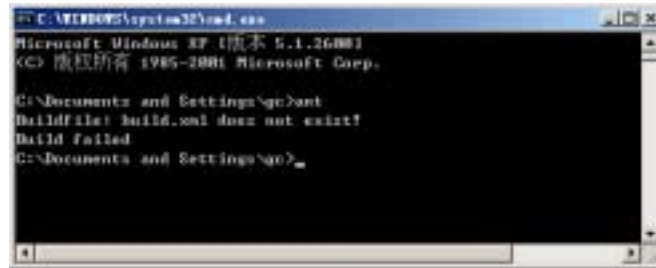


图 14.3 查看 Ant 是否安装成功

如果输入一下内容，则说明 Ant 已经安装成功。

```
Buildfile: build.xml does not exist!  
Build failed
```

注意：虽然输出内容为 Build 失败，只是说明没有找到 build.xml，所以 Build 失败，但说明 Ant 已经可以运行。

## 14.3 在 Eclipse 下建立项目 myNews

在 Eclipse 下建立项目 myNews，并配置 Spring 和 Hibernate。

### 14.3.1 在 Eclipse 下建立项目

- (1) 运行 Eclipse，单击菜单栏中的“File”菜单，Eclipse 将显示“File”菜单。
- (2) 移动鼠标到“New”，在出现的子菜单中单击“Project”，Eclipse 将弹出“New Project”对话框，如图 14.4 所示。
- (3) 用鼠标选择列表框中“Java”下的“Tomcat Project”，然后单击“Next”按钮。将弹出“New Tomcat Project”对话框，如图 14.5 所示。

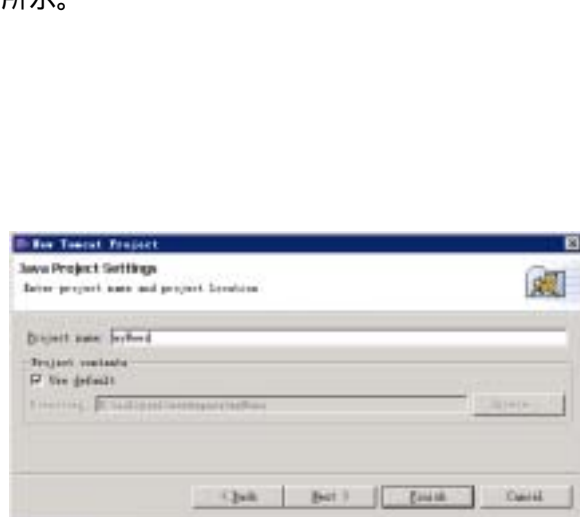
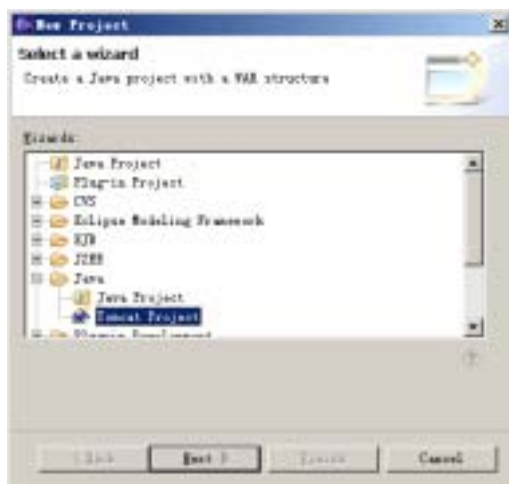


图 14.4 “New Project”对话框

图 14.5 “New Tomcat Project”对话框

(4) 在“New Tomcat Project”对话框中，“Project name”文本框中输入“myNews”，然后单击“Finish”按钮，项目即建立成功，myNews 的目录结构如图 14.6 所示。

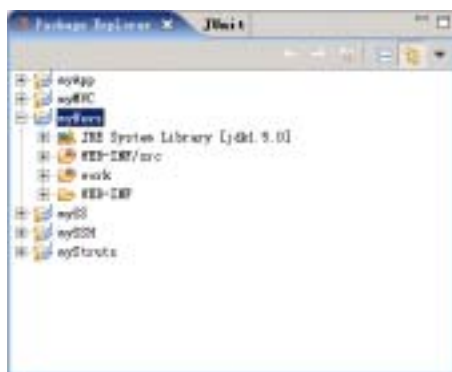


图 14.6 myNews 的目录结构

(5) 把 log4j-1.2.9.jar、commons-logging.jar、spring.jar、antlr.jar、asm.jar、spring-hibernate3.jar、asm-attrs.jar、cglib.jar、commons-collections.jar、dom4j.jar、ehcache.jar、jta.jar、mysql-connector-java-5.0.0-beta-bin.jar、hibernate3.jar。这 14 个 jar 放在 /WEB-INF/lib/ 下复制到 myNews\WEB-INF\lib 目录下，即 CLASSPATH 中。

(6) 用 Windows 自带的文本编辑器，建立一个文件 log4j.properties，把 log4j.properties 放到 myNews\WEB-INF\src 目录下。

(7) 编辑 log4j.properties 文件，内容如下：

```
log4j.rootLogger=DEBUG,stdout,R

log4j.logger.org=ERROR, A1
log4j.logger.com.gd =DEBUG,A2
log4j.appender.A1=org.apache.log4j.RollingFileAppender
log4j.appender.A1.File=org.log
log4j.appender.A1.MaxFileSize=500KB
log4j.appender.A1.MaxBackupIndex=50
log4j.appender.A1.Append=true
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{ISO8601} - [%p] [%C{1}] - %m%n
log4j.appender.A2=org.apache.log4j.RollingFileAppender
log4j.appender.A2.File=gc.log
log4j.appender.A2.MaxFileSize=500KB
log4j.appender.A2.MaxBackupIndex=50
log4j.appender.A2.Append=true
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d{ISO8601} - [%p] [%C{1}] - %m%n
#-----stdout-----
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=[%-5p] %d{yyyy-MM-dd HH:mm:ss} %c - %m%n
#-----R-----
#log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
#this log file will be stored in web server's /bin directory,modify to your path which want to store.
log4j.appender.R.File=gf.log
```

```
#log4j.appender.R.datePattern='yyyy-MM-dd-HH-mm
log4j.appender.R.datePattern='yyyy-MM-dd
log4j.appender.R.append=true
## Keep one backup file
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=[%-5p] %d{yyyy-MM-dd HH:mm:ss} %c - %m%n
#[%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n
```

(8) 在 myNews 上单击鼠标右键，Eclipse 将弹出右键菜单。

(9) 在弹出的右键菜单中，单击“Properties”，将弹出“Properties for myNews”对话框，如图 14.7 所示。



图 14.7 “Properties for myNews”对话框

(10) 在“Properties for myNews”对话框中，选择对话框右边列表框中的“Java Build Path”。

(11) 选择“Properties for myNews”对话框左边的“Libraries”标签。

(12) 在“Libraries”标签中，单击“Add JARs...”按钮，弹出“JAR Selection”对话框，如图 14.8 所示。

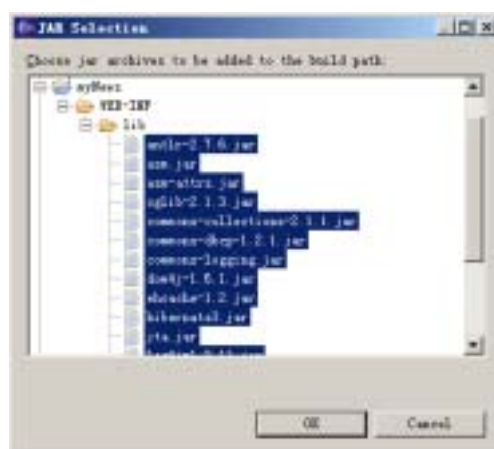


图 14.8 “JAR Selection”对话框

(13) 在“JAR Selection”对话框中，打开列表框“myNews”一直到 lib 目录下出现 14 个 jar：log4j-1.2.9.jar、commons-logging.jar、spring.jar、antlr.jar、spring-hibernate3.jarasm.jar、



asm-attrs.jar 、 cglib.jar 、 commons-collections.jar 、 dom4j.jar 、 ehcache.jar 、 jta.jar 、 mysql-connector-java-5.0.0-beta-bin.jar、 hibernate3.jar。

(14) 按住“Ctrl”键,选中这14个jar,然后单击“OK”按钮,返回“Properties for myNews”对话框,如图14.9所示。

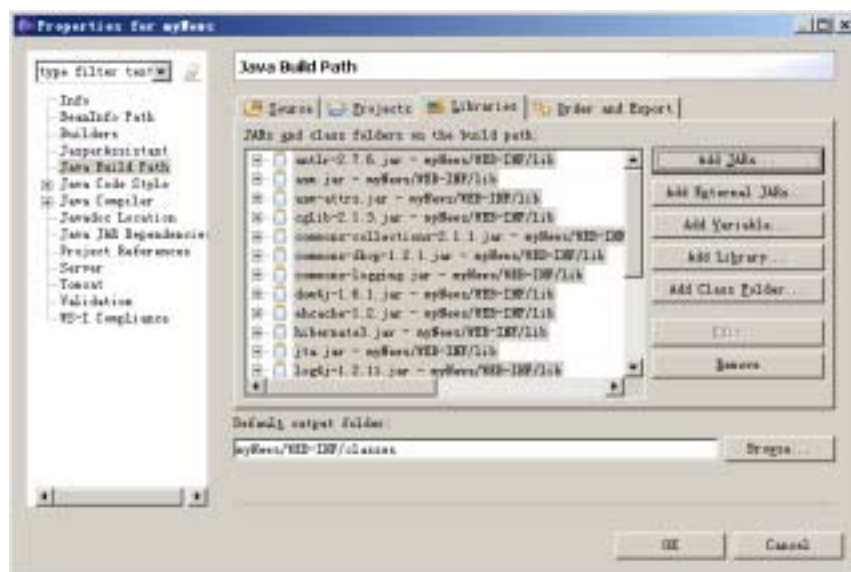


图 14.9 “Properties for myNews”对话框

(15) 在“Properties for myNews”对话框,单击“OK”按钮,即可完成对 Spring 和 Hibernate 的配置。

(16) 再次在 myNews 上单击鼠标右键,在弹出右键菜单中,将鼠标移动到“New”子菜单中的“Package”,单击“Package”,将弹出“New Java Package”对话框,如图14.10所示。

注意:在 spring.jar 中,没有包含对 Hibernate 的支持,如果需要使用 Hibernate,则需要把 spring-framework-2.0-m1\dist\extmodules\spring-hibernate3.jar 也包含进 ClassPath 中。



图 14.10 “New Java Package”对话框

(17) 在“New Java Package”对话框中的“Name”文本框中输入“com.gd.action”,然后单击“Finish”按钮,即可建立 com.gd.action 包。

(18) 用同样的方法建立 com.gd.service 包、com.gd.service.impl 包、com.gd.dao 包、com.gd.dao.impl 包和 com.gd.vo 包、com.gd.po 包。

(19) 在 WEB-INF 目录下建立 jsp 文件夹。

(20) 最终配置好 Spring 和 Hibernate 的 myNews 项目的目录结构如图 14.11 所示。

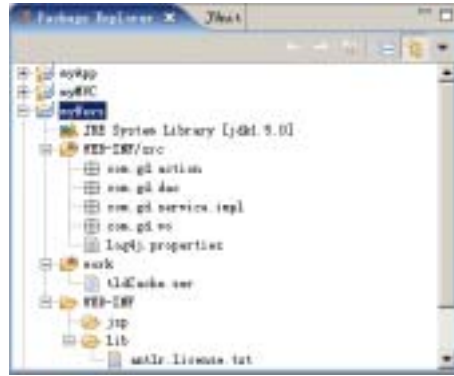


图 14.11 配置好 Spring 和 Hibernate 的 myNews 项目的目录结构

### 14.3.2 编写 Ant build 文件

在 myNews 目录下新建一个 ant 文件 build.xml，示例代码如下：

```
<?xml version="1.0"?>
<project name="myNews" default="init" basedir=".">
  <property name="myNews.home" value="." />
  <property name="myNews.lib" value="${myNews.home}/WEB-INF/lib" />
  <property name="myNews.jar" value="${myNews.home}/WEB-INF/lib" />
  <property name="myNews.classes" value="${myNews.home}/WEB-INF/classes" />
  <property name="tomcat.home" value="D:/jakarta-tomcat-5.5.5" />
  <!--<property file="build.properties" />以上内容还可以定义在 build.properties 中-->
  <target name="init">
    <path id="all">
      <fileset dir="${myNews.lib}">
        <include name="**/*.jar" />
      </fileset>
      /*用来定义所包含的 jar*/
      <fileset dir="${tomcat.home}/common/lib">
        <include name="*.jar" />
      </fileset>
    </path>
    <mkdir dir="${myNews.classes}" />
  </target>
  <target name="clean">
    <delete dir="${myNews.classes}">
    </delete>
  </target>
  <target name="compile" depends="init">
    /*用来定义要编译的源文件和编译后的路径*/
    <javac srcdir="${myNews.home}/WEB-INF/src" destdir="${myNews.classes}"
target="1.5">
      <classpath refid="all" />
    </javac>
  </target>
  <target name="jar" depends="compile">
    /*用来定义打包后 jar 的名字和路径*/
    <jar jarfile="${myNews.jar}/gd.jar" basedir="${myNews.classes}" includes="com/gd/**">
```

```

        </jar>
    </target>
    <!--将 myNews 项目打成 war 文件-->
    <target name="war" depends="jar">
        <war
                                destfile="${myNews.home}/myNews.war"
webxml="${myNews.home}/WEB-INF/web.xml">
            <fileset dir="${myNews.home}" casesensitive="yes">
                <include name="WEB-INF/**" />
                <exclude name="*.war" />
            </fileset>
            <lib dir="${myNews.home}/WEB-INF/lib">
                <include name="*.jar" />
            </lib>
        </war>
    </target>
</project>

```

### 14.3.3 配置 Web.xml 文件

建立 web.xml 文件，放在 myNews \WEB-INF 目录下，web.xml 的示例代码如下所示：

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
    .....
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/conf/dispatcherServlet-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcherServlet </servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <taglib>
        <taglib-uri>/spring</taglib-uri>
        <taglib-location>/WEB-INF/tld/spring.tld</taglib-location>
    </taglib>
    .....
</web-app>

```

代码说明：

- /WEB-INF/conf/dispatcherServlet-servlet.xml，为了统一管理，在 WEB-INF 下新建一个文件夹 conf，把配置文件放在 /WEB-INF/conf/ 下。
- /WEB-INF/tld/spring.tld，为了统一管理，在 WEB-INF 下新建一个文件夹 tld，把自

定义标签文件放在/WEB-INF/tld/下。

## 14.4 设计新闻发布系统

上面环境配置完毕，在开始编码之前，先来设计新闻发布系统，包括设计画面、设计业务逻辑、设计数据库。

### 14.4.1 设计画面

为了示例方便，这里的画面都没有使用图片。

从前面的实例说明，可以知道，这个实例需要如下一些画面：新闻发布的展示画面 show.html、发布新闻画面 release.html、用户注册画面 regedit.html、管理员登录页面 login.html、错误处理页面 error.html。

(1) 新闻发布的展示画面 show.html，如图 14.12 所示。



图 14.12 新闻发布的展示画面

该画面主要用来显示用户发布的新闻，并按照新闻类别来显示。每个新闻类别下显示 5 条新闻，其他新闻使用更多内容来进行查看。其中新闻类别分为：新闻中心、公司文件、规章制度、会议纪要。

show.html 的源代码如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>新闻发布的展示画面</title>

<style type="text/css">
<!--
.style1 {font-family: "隶书"}
-->
```

```

</style>
</head>

<body>
<table width="100%" height="100%" border="1" cellpadding="0" cellspacing="0" >
  <tr height="100%">
    <td height="20"><strong>新闻中心</strong></td>
    <td><strong>公司文件</strong></td>
  </tr>
  <tr height="100%">
    <td height="150"><ol>
      <li>中共 x x 公司支部委员会胜利召开</li>
      <li>x x 公司自主管理活动显成效</li>
      <li>普及法律合同知识，增强规避风险能力</li>
      <li>自主管理活动工具介绍</li>
      <li>自主管理活动遍地开花</li>
    </ol></td>
    <td><ol>
      <li>关于改变中午午休的通知</li>
      <li>关于任命 x x 为厂长的通知</li>
      <li>关于开展自主管理活动的通知</li>
      <li>关于对上半年工作进行总结的通知</li>
      <li>召开人力资源专题会议的通知</li>
    </ol></td>
  </tr>
  <tr height="100%" style="border-top-width:0">
    <td height="15" style="border-top-width:0"><div align="right" class="style1" >》更多内容
  </div></td>
    <td style="border-top-width:0"><div align="right" class="style1" >》更多内容</div></td>
  </tr>
  <tr height="100%">
    <td height="20"><strong>规章制度</strong></td>
    <td><strong>会议纪要</strong></td>
  </tr>
  <tr height="100%">
    <td height="150"><ol>
      <li>设备维修管理制度</li>
      <li>计算机管理制度</li>
      <li>办公用品发放管理制度</li>
      <li>工程项目审计实施办法</li>
      <li>会计档案管理知道</li>
    </ol></td>
    <td><ol>
      <li>x x 公司部长例会会议纪要</li>
      <li>x x 公司双周例会会议纪要</li>
      <li>2006 年 5 月人力资源专业例会会议纪要</li>
      <li>2006 年一季度财务专业例会会议纪要</li>
      <li>x x 公司 2005 年品种开发会议纪要</li>
    </ol></td>
  </tr>
  <tr height="100%">
    <td height="15" style="border-top-width:0"><div align="right" class="style1" >》更多内容
  </div></td>

```

```

        <td style="border-top-width:0"><div align="right" class="style1">》更多内容</div></td>
    </tr>
</table>
</body>
</html>

```

(2) 发布新闻画面 release.html，如图 14.13 所示。

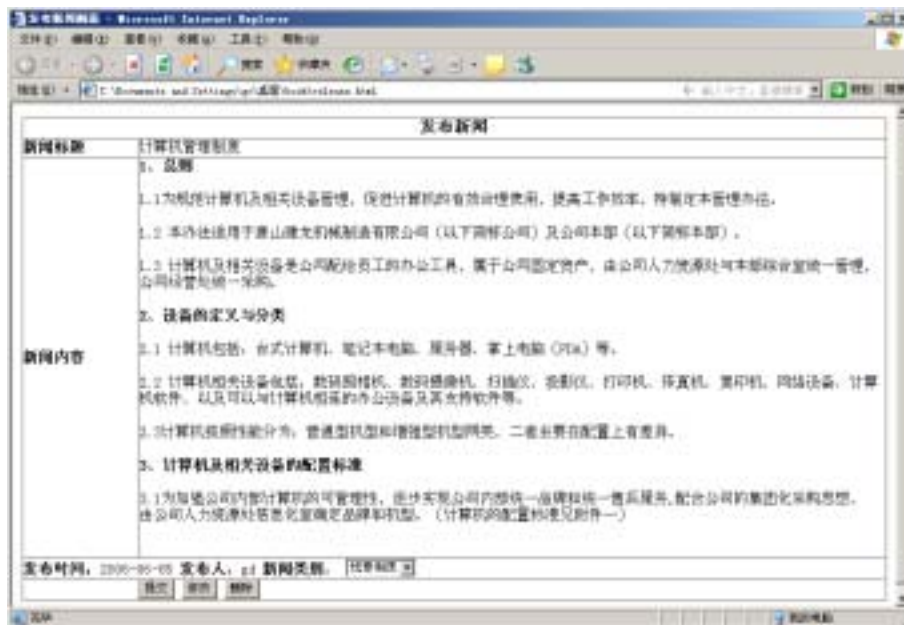


图 14.13 发布新闻画面

该画面主要用来发布新闻，用户填写新闻标题、新闻内容和选择新闻类别，系统自动代出发布时间 and 发布人。

release.html 的源代码如下所示：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>发布新闻画面</title>
<style type="text/css">
<!--
.style1 {
    font-size: large;
    font-weight: bold;
}
-->
</style>
</head>

<body>
<form name="form1" method="post" action="">
    <table width="100%" height="160" border="1" cellpadding="0" cellspacing="0">
        <tr>
            <td height="17" colspan="2"><div align="center" class="style1">发布新闻</div></td>

```

```

</tr>
<tr>
  <td width="126" height="19"><strong>新闻标题</strong></td>
  <td width="560">计算机管理制度</td>
</tr>
<tr>
  <td height="73"><strong>新闻内容</strong></td>
  <td><p><strong>1</strong><strong>、总则</strong></p>
    <p align="left">1.1 为规范计算机及相关设备管理，促进计算机的有效合理使用，提高工作效率，特制定本管理办法。</p>
    <p>1.2 本办法适用于唐山建龙机械制造有限公司（以下简称公司）及公司本部（以下简称本部）。</p>
    <p>1.3 计算机及相关设备是公司配给员工的办公工具，属于公司固定资产，由公司人力资源处与本部综合室统一管理，公司经营处统一采购。</p>
    <p><strong>2</strong><strong>、设备的定义与分类</strong></p>
    <p>2.1 计算机包括：台式计算机、笔记本电脑、服务器、掌上电脑（PDA）等。</p>
    <p>2.2 计算机相关设备包括：数码照相机、数码摄像机、扫描仪、投影仪、打印机、传真机、复印机、网络设备、计算机软件、以及可以与计算机相连的办公设备及其支持软件等。</p>
    <p>2.3 计算机按照性能分为：普通型机型和增强型机型两类，二者主要在配置上有差异。<strong>&nbsp;</strong></p>
    <p><strong>3</strong><strong>、计算机及相关设备的配置标准</strong></p>
    <p>3.1 为加强公司内部计算机的可管理性，逐步实现公司内部统一品牌和统一售后服务,配合公司的集团化采购思想，由公司人力资源处信息化室确定品牌和机型。（计算机的配置标准见附件一）</p>
    <p>&nbsp;</p>
  </td>
</tr>
<tr>
  <td height="19" colspan="2"><strong>发布时间：</strong>2006-06-05 <strong>发布人</strong>：gd <strong>新闻类别</strong>：
    <select name="select">
      <option>新闻中心</option>
      <option selected>规章制度</option>
      <option>会议纪要</option>
      <option>公司文件</option>
    </select></td>
</tr>
<tr>
  <td height="18">&nbsp;</td>
  <td><input type="submit" name="Submit" value="提交">
    <input type="submit" name="Submit" value="修改">
    <input type="submit" name="Submit" value="删除"></td>
</tr>
</table>
</form>
</body>
</html>

```

（3）用户注册画面 regedit.html，如图 14.14 所示。



图 14.14 用户注册画面

该画面主要用来注册用户，包括注册用户名和密码。

regedit.html 的源代码如下所示：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>用户注册画面</title>
</head>

<body>
<form name="form1" method="post" action="">
  <table width="100%" height="251" border="1" cellpadding="0" cellspacing="0">
    <tr>
      <td height="17" colspan="2"><div align="center"><strong> 注 册 用 户
</strong></div></td>
    </tr>
    <tr>
      <td width="18%"><strong>用户名：</strong></td>
      <td width="82%"><input type="text" name="textfield"></td>
    </tr>
    <tr>
      <td><strong>密码：</strong></td>
      <td><input type="password" name="textfield"></td>
    </tr>
    <tr>
      <td><strong>确认密码：</strong></td>
      <td><input type="password" name="textfield"></td>
    </tr>
    <tr>
      <td colspan="2"><div align="center">
        <input type="submit" name="Submit" value="注册">
        <input type="reset" name="Submit" value="重置">
      </div></td>
    </tr>
  </table>
```



```

</form>
</body>
</html>

```

(4) 管理员登录页面 login.html，如图 14.15 所示。

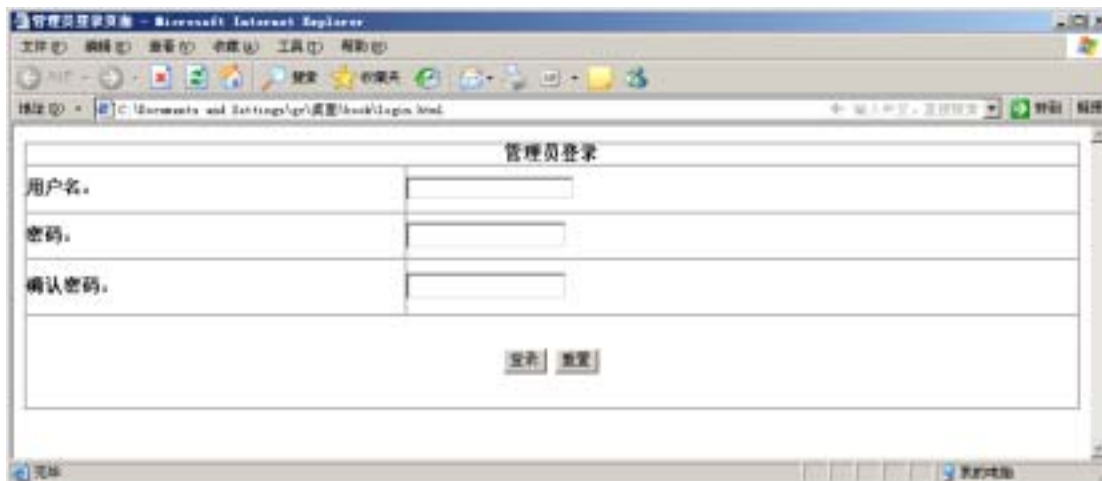


图 14.15 管理员登录页面

该画面主要用来进行新闻发布前的登录，如果管理员登录成功，则直接跳转至发布新闻的画面。

login.html 的源代码如下所示：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>管理员登录页面</title>
</head>

<body>
<form name="form1" method="post" action="">
  <table width="100%" border="1" cellpadding="0" cellspacing="0">
    <tr>
      <td colspan="2"><div align="center"><strong>管理员登录</strong></div></td>
    </tr>
    <tr>
      <td height="41"><strong>用户名：</strong></td>
      <td><input type="text" name="textfield"></td>
    </tr>
    <tr>
      <td height="40"><strong>密码：</strong></td>
      <td><input type="password" name="textfield"></td>
    </tr>
    <tr>
      <td height="49"><strong>确认密码：</strong></td>
      <td><input type="password" name="textfield"></td>
    </tr>
  </table>

```

```

        <td height="83" colspan="2"><div align="center">
            <input type="submit" name="Submit" value="登录">
            <input type="reset" name="Submit" value="重置">
        </div></td>
    </tr>
</table>
</form>
</body>
</html>

```

(5) 错误处理页面 error.html，如图 14.16 所示。



图 14.16 错误处理页面

该画面主要用来显示在示例运行过程中的错误信息。

error.html 的源代码如下所示：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>错误处理页面</title>
<style type="text/css">
<!--
.style1 {
    color: #000000;
    font-weight: bold;
}
.style2 {color: #FF0000}
-->
</style>
</head>

<body>
<table width="100%" border="1">
    <tr>
        <td colspan="2"><div align="center"><strong>错误信息显示</strong></div></td>
    </tr>
    <tr>

```

```

<td width="22%" height="141"><span class="style1">错误信息是：</span></td>
<td width="78%"><span class="style2">SQL 出错，请查看 SQL 语句是否符合语法规则。
</span></td>
</tr>
</table>
</body>
</html>

```

## 14.4.2 设计持久化类

通过上面的介绍和分析，可以知道，在该应用程序中，至少需要以下一些持久化类：负责用户基本信息的类 User.java、负责用户权限的类 UsersAuthor.java、负责新闻的类 News.java、负责新闻类别的类 NewsType.java。

这里使用 Together 来画 UML 图，对于 Together 的使用方法这里不再进行讲解，读者可以使用其他的工具，方法基本类似，这里首先画一个整体包结构图，如图 14.17 所示：

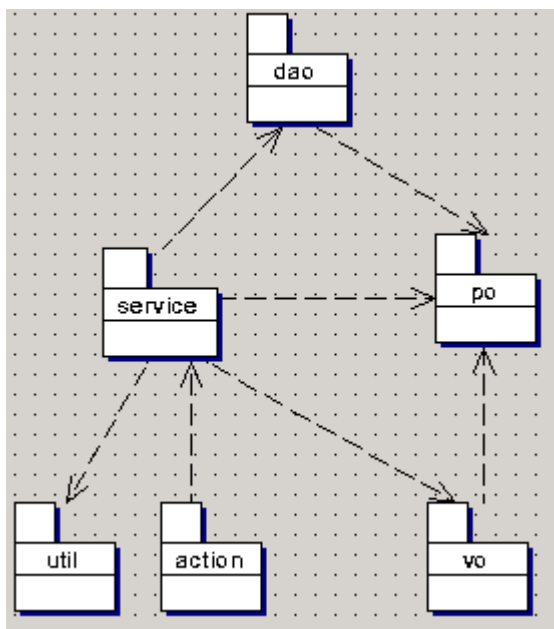


图 14.17 整体包结构图

(1) 负责用户基本信息的类 User.java，主要用来存储用户的基本信息以及对用户的信息进行验证，用户类的 UML 图如图 14.18 所示：

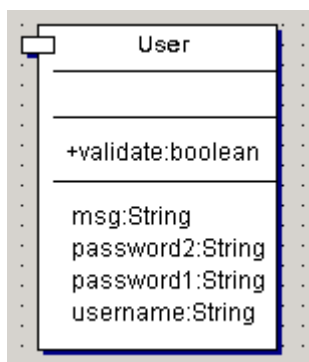


图 14.18 用户类的 UML 图

由 Together 自动生成的用户类的源代码如下：

```
//***** User.java *****  
package com.gd.vo;  
public class User {  
    public String getMsg(){  
return msg;  
    }  
    public void setMsg(String msg){  
this.msg = msg;  
    }  
    public String getPassword2(){  
return password2;  
    }  
    public void setPassword2(String password2){  
this.password2 = password2;  
    }  
    public String getPassword1(){  
return password1;  
    }  
    public void setPassword1(String password1){  
this.password1 = password1;  
    }  
    public String getUsername(){  
return username;  
    }  
    public void setUsername(String username){  
this.username = username;  
    }  
    public boolean validate() {  
    }  
    private String msg;  
    private String password2;  
    private String password1;  
    private String username;  
}
```

代码说明：

- ☐ msg，用来存储该用户类的消息。
- ☐ password1，用来存储用户第一次输入的密码。
- ☐ password2，用来存储用户第二次输入的密码。
- ☐ username，用来存储用户名。

(2) 负责用户权限的类 UsersAuthor.java，主要用来存储用户的权限信息，用户权限类的 UML 图如图 14.19 所示：



图 14.19 用户权限类的 UML 图

因为 UsersAuthor 类依赖于 User 类，所以这里给出用户类和用户权限类之间的关联图，如图 14.20 所示：

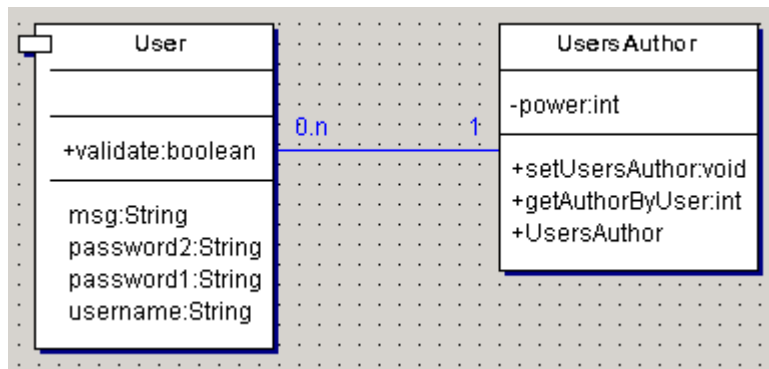


图 14.20 用户类和用户权限类之间的关联图

由 Together 自动生成的用户权限类的源代码如下：

```

//***** UsersAuthor.java*****
package com.gd.vo;
public class UsersAuthor {
    public void setUsersAuthor(User lnkUser, int power) {
        lnkUser = lnkUser;
        power = power;
    }
    public int getAuthorByUser(User lnkUser) {
        return power;
    }
    public UsersAuthor(User lnkUser, int power) {
        lnkUser = lnkUser;
        power = power;
    }
    private int power;
    /**
     * @clientCardinality 1
     * @supplierCardinality 0..n
     */
    private User lnkUser;
}
  
```

代码说明：

- power，表示用户拥有的权限，如果 power 为 0 表示普通用户，如果 power 为 1 表示超级用户。

(3) 负责新闻类别的类 NewsType.java , 主要用来存储新闻类别的信息 , 新闻类别类的 UML 图如图 14.21 所示 :

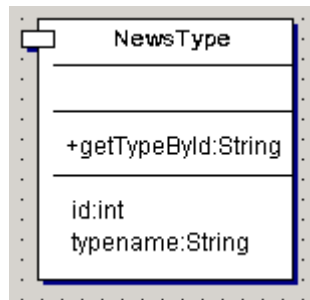


图 14.21 新闻类别类的 UML 图

由 Together 自动生成的新闻类别类的源代码如下 :

```
/******* NewsType.java*****  
  
package com.gd.vo;  
public class NewsType {  
    public int getId(){  
        return id;  
    }  
    public void setId(int id){  
        this.id = id;  
    }  
    public String getTypeName(){  
        return typename;  
    }  
    public void setTypename(String typename){  
        this.typename = typename;  
    }  
    public String getTypeById (int id) {  
        return typename;  
    }  
    private int id;  
    private String typename;  
}
```

(4) 负责新闻的类 News.java , 主要用来存储新闻的信息 , 新闻类的 UML 图如图 14.22 所示 :

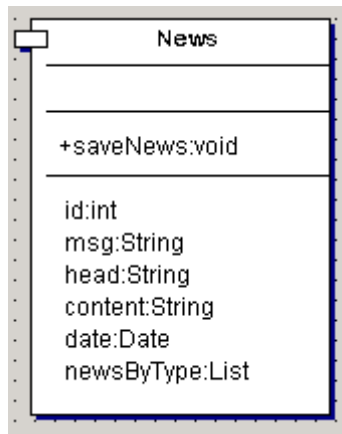


图 14.22 新闻类的 UML 图

因为 News 类依赖于 User 类和新闻类别类，所以这里给出用户类、新闻类和新闻类别类之间的关联图，如图 14.23 所示：

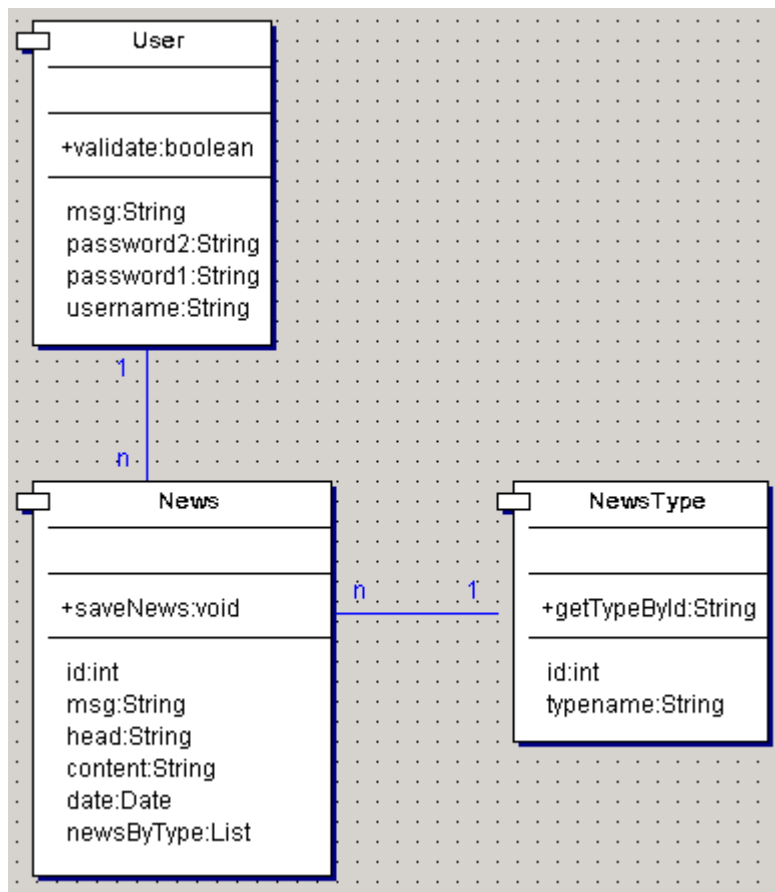


图 14.23 用户类、新闻类和新闻类别类之间的关联图

由 Together 自动生成的新闻类的源代码如下：

```

//***** News.java*****

package com.gd.vo;
public class News {
public int getId(){
  
```

```

        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getMsg(){
        return msg;
    }
    public void setMsg(String msg){
        this.msg = msg;
    }
    public String getHead(){
        return head;
    }
    public void setHead(String head){
        this.head = head;
    }
    public String getContent(){
        return content;
    }
    public void setContent(String content){
        this.content = content;
    }
    public Date getDate(){
        return date;
    }
    public void setDate(Date date){
        this.date = date;
    }
    //根据新闻类别的 id 获取新闻
    public List getNewsByType(int id) {
    }
    //保存新闻
    public void saveNews(New new) {
    }
    private int id;
    private String msg;
    private String head;
    private String content;
    private Date date;
    /**
     * @clientCardinality n
     * @supplierCardinality 1
     */
    private NewsType InkNewsType;
    /**
     * @clientCardinality n
     * @supplierCardinality 1
     */
    private User InkUser;
}

```



14.4.3 设计数据库

经过分析可以知道，在该应用中，主要有用户信息、用户的授权信息、新闻类别、新闻等内容需要存储，所以对数据库表的设计也主要从这几个方面来考虑。

- (1) 存储用户信息的表，表名为 user，主要字段有：username、password，主键为 username。
- (2) 存储用户授权信息的表，表名为 userAuthor，主要字段有：username、power，主键为 username、power。
- (3) 存储新闻类别的表，表名为 newsType，主要字段有：id、type，主键为 id。
- (4) 存储新闻的表，表名为 news，主要字段有：id、head、content、issuedate、issueuser、newstype，主键为 id。

注意：上面设计的数据库表是为了演示使用，如果在实际的应用中，笔者建议每个表都加上 id，然后表之间的关联用 id 来实现，这样就把表之间的管理和业务逻辑的变化分离开了，降低了业务逻辑和表之间的耦合性。

14.4.4 新闻发布系统在持久层的整体 UML 图

经过上面对画面、持久类和数据库的设计，最终可以得到一个新闻发布系统在持久层的整体 UML 图，如图 14.24 所示：

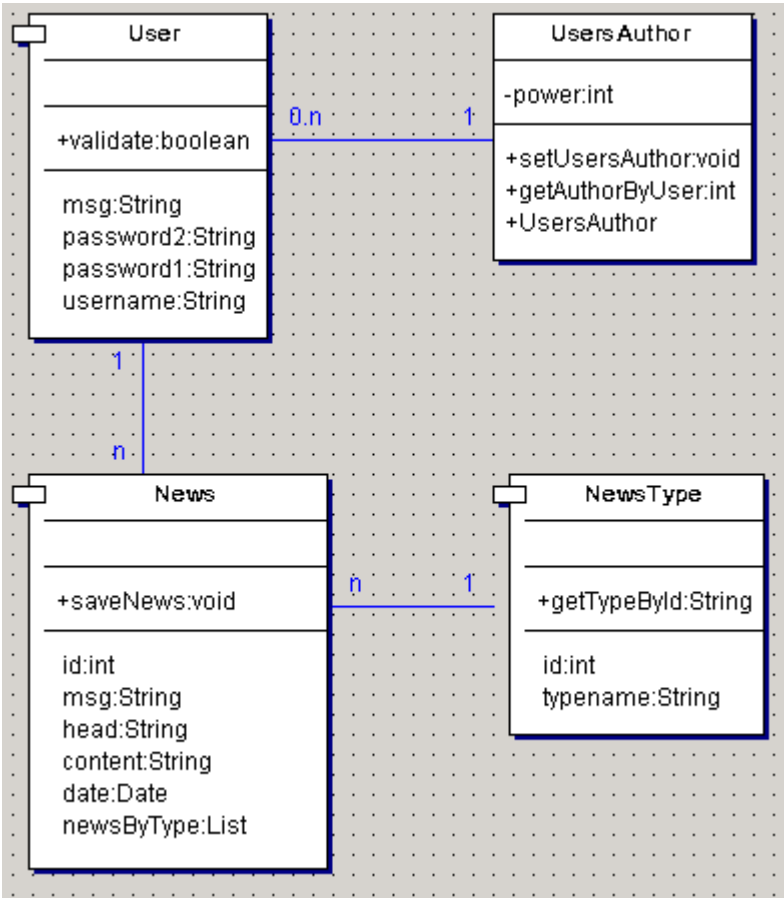


图 14.24 新闻发布系统在持久层的整体 UML 图

## 14.5 编写新闻发布系统的 Jsp 页面

根据上面设计的新闻发布系统的画面，编写这些画面的 jsp 页面。

### 14.5.1 新闻发布的展示画面 show.jsp

该画面存放在 WEB-INF/jsp 下，用来展示已经发布的新闻，并按照新闻类别进行显示，这里将新闻标题放在 Map 数组中，建立新闻类别 id 和新闻标题之间的对应关系，首先将新闻类别通过循环展示出来，同时每显示一个新闻类别，将新闻类别对应的新闻标题再通过循环显示出来。show.jsp 的代码如下所示：

```
<%@page contentType="text/html;charset=GBK"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.util.*,com.gd.util.*,com.gd.vo.*,com.gd.po.Newstype,com.gd.po.New"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>新闻发布的展示画面</title>

<style type="text/css">
<!--
.style1 {font-family: "隶书"}
-->
</style>
</head>
<%
List listNewsType = (List)request.getAttribute("listNewsType");
Map mapNews = (Map)request.getAttribute("mapNews");
%>
<body>
<table width="100%" height="100%" border="1" cellpadding="0" cellspacing="0" >
<%
    for (int i = 0; listNewsType != null && i < listNewsType.size(); i++) {
%>
<tr height="100%">
<td height="20"><strong><%=((Newstype)listNewsType.get(i)).getType()%></strong></td>
<td><strong><%=((Newstype)listNewsType.get(i + 1)).getType()%></strong></td>
</tr>
<tr height="100%">
<td height="150"><ol>
<%
        List newsHeads = (List)mapNews.get(((Newstype)listNewsType.get(i)).getId());
        for (int j = 0; newsHeads != null && j < newsHeads.size(); j++) {
%>
<li><%=((New)newsHeads.get(j)).getContent() %></li>
<%}%>
</ol></td>
<td><ol>
```

```

        <%
            newsHeads = (List)mapNews.get((((Newstype)listNewsType.get(++i)).getId());
            for (int j = 0; newsHeads != null && j < newsHeads.size(); j++) {
                %>
                <li><%=((New)newsHeads.get(j)).getContent() %></li>
            <%}%>
        </ol></td>
    </tr>
    <tr height="100%" style=" border-top-width:1">
        <td height="15" style=" border-top-width:1"><div align="right" class="style1" >》更多内容
    </div></td>
        <td style=" border-top-width:1"><div align="right" class="style1" >》更多内容</div></td>
    </tr>
    <%}%>
</table>
</body>
</html>

```

代码说明：

- Map mapNews = (Map)request.getAttribute("mapNews")，这里通过 Map 来循环把每种新闻类别的内容显示出来。

## 14.5.2 发布新闻画面 release.jsp

该画面存放在 WEB-INF/jsp 下，主要通过表单来提交用户填写的新闻标题和内容，已经发布人，这里添加了一个辅助类 NewsUtil.java，主要用来获取当前日期，release.jsp 的代码如下：

```

<%@page contentType="text/html; charset=GBK"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.util.List,com.gd.util.*,com.gd.vo.User,com.gd.po.Newstype"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>发布新闻画面</title>
<style type="text/css">
<!--
.style1 {
    font-size: large;
    font-weight: bold;
}
-->
</style>
</head>
<%
List newsTypes = (List)request.getAttribute("newsTypes");
User user = (User)request.getAttribute("user");
%>
<body>
<form name="form1" method="post" action="/myNews/release.do">

```

```

<table width="100%" height="160" border="1" cellpadding="0" cellspacing="0">
<tr>
<td height="17" colspan="2"><div align="center" class="style1">发布新闻</div></td>
</tr>
<tr>
<td width="126" height="19"><strong>新闻标题</strong></td>
<td width="560"><input name="head" type="text" size="100%"></td>
</tr>
<tr>
<td height="73"><strong>新闻内容</strong></td>
<td><p>
<textarea name="content" cols="100%" rows="30"></textarea>
</p>
</td>
</tr>
<tr>
<td height="19" colspan="2"><strong>发 布 时 间 :</strong><%=NewsUtil.getCurrentDate()%><strong>发布人</strong> :<%=user.getUsername()%>
<strong>新闻类别</strong> :
<select name="newsType">
<%
for (int i = 0; newsTypes != null && i < newsTypes.size(); i++) {
Newstype newsType = (Newstype)newsTypes.get(i);
%>
<option value = '<%=newsType.getId()%>'><%=newsType.getType()%></option>
<%}%>
</select></td>
</tr>
<tr>
<td height="18">&nbsp;</td>
<td><input type="submit" name="insert" value="提交">
<input type="submit" name="update" value="修改">
<input type="submit" name="delete" value="删除"></td>
</tr>
</table>
</form>
</body>
</html>

```

代码说明：

- NewsUtil.getCurrentDate()，这里添加一个辅助类 NewsUtil，用来负责 myNews 系统的一些辅助方法，该类在包 com.gd.util 里。

### 14.5.3 用户注册画面 regedit.jsp

该画面存放在 WEB-INF/jsp 下，主要通过表单来提交用户填写的用户名和密码，regedit.jsp 的示例代码如下：

```

<%@page contentType="text/html;charset=GBK"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>

```

```

<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>用户注册画面</title>
</head>

<body>
<form name="form1" method="post" action="/myNews/regedit.do">
  <table width="100%" height="251" border="1" cellpadding="0" cellspacing="0">
    <tr>
      <td height="17" colspan="2"><div align="center"><strong>注册用户</strong></div></td>
    </tr>
    <tr>
      <td width="18%"><strong>用户名 : </strong></td>
      <td width="82%"><input type="text" name="username"></td>
    </tr>
    <tr>
      <td><strong>密码 : </strong></td>
      <td><input type="password" name="password1"></td>
    </tr>
    <tr>
      <td><strong>确认密码 : </strong></td>
      <td><input type="password" name="password2"></td>
    </tr>
    <tr>
      <td colspan="2"><div align="center">
        <input type="submit" name="Submit" value="注册">
        <input type="reset" name="Submit" value="重置">
      </div></td>
    </tr>
  </table>
</form>
</body>
</html>

```

#### 14.5.4 管理员登录页面 login.jsp

该画面存放在 WEB-INF/jsp 下，主要通过表单来提交用户填写的用户名和密码，用于验证用户是否填写正确，login.jsp 的示例代码如下：

```

<%@page contentType="text/html; charset=GBK"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>管理员登录页面</title>
</head>

<body>
<form name="form1" method="post" action="/myNews/login.do">
  <table width="100%" height="251" border="1" cellpadding="0" cellspacing="0">

```

```

<tr>
  <td height="17" colspan="2"><div align="center"><strong>管理员登录</strong></div></td>
</tr>
<tr>
  <td width="18%"><strong>用户名 : </strong></td>
  <td width="82%"><input type="text" name="username"></td>
</tr>
<tr>
  <td><strong>密码 : </strong></td>
  <td><input type="password" name="password1"></td>
</tr>
<tr>
  <td><strong>确认密码 : </strong></td>
  <td><input type="password" name="password2"></td>
</tr>
<tr>
  <td colspan="2"><div align="center">
    <input type="submit" name="Submit" value="登录">
    <input type="reset" name="Submit" value="重置">
  </div></td>
</tr>
</table>
</form>
</body>
</html>

```

### 14.5.5 错误处理页面 error.jsp

该画面存放在 WEB-INF/jsp 下，主要用来捕捉并显示程序出现的 Exception 信息，error.jsp 的示例代码如下：

```

<%@page contentType="text/html; charset=GBK"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>错误处理页面</title>
<style type="text/css">
<!--
.style1 {
  color: #000000;
  font-weight: bold;
}
.style2 {color: #FF0000}
-->
</style>
</head>
<% Exception ex = (Exception)request.getAttribute("Exception"); %>

<body>
<table width="100%" border="1">
  <tr>

```

```

<td colspan="2"><div align="center"><strong>错误信息显示</strong></div></td>
</tr>
<tr>
<td width="22%" height="141"><span class="style1">错误信息是 : </span></td>
<td width="78%"><span class="style2"><%=ex.getMessage();%></span></td>
</tr>
</table>
</body>
</html>

```

如果要使用该画面，则需要在 Sprng 的配置文档中增加以下代码：

```

<bean id="exceptionResolver"
class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
  <property name="exceptionMappings">
    <props>
      <prop key="java.sql.SQLException">error</prop>
      <prop key="java.sql.IOException">error</prop>
    </props>
  </property>
</bean>

```

代码说明：

- 只要发生了 SQLException 异常或 IOException 异常，就会连接至 /WEB-INF/jsp/error.jsp。

## 14.6 建立数据库表并生成 xml 和 POJO

这里仍然通过 Xampp 来建立 mysql 数据库，步骤如下：

(1) 在浏览器地址栏输入 <http://localhost/xampp/>，进入 xampp 的管理画面，如图 14.25 所示。



图 14.25 xampp 的管理画面

(2) 在 Xampp 画面左边单击工具下的 phpMyAdmin，将会出现创建数据库的向导画面，如图 14.26 所示。

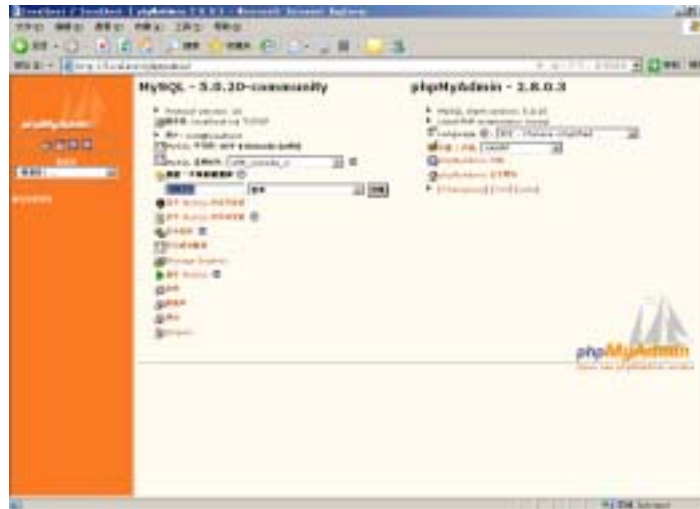


图 14.26 创建数据库的向导画面

(3) 这里设定创建的数据库名为 myNews ,然后单击“ 创建 ”按钮 ,即可创建数据库 myNews。接下来根据向导，分别创建前面设计的数据库表。

### 14.6.1 存放用户信息的数据库表

(1) 根据创建一个存储用户信息的表，表名为 user，主要包括 3 个字段，创建 user 表的向导如图 14.27 所示：

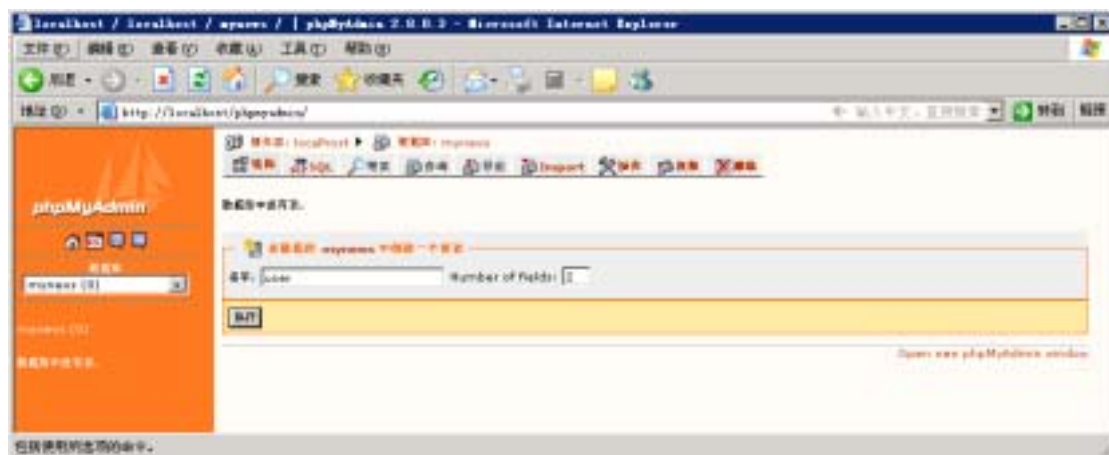


图 14.27 创建 user 表的向导

(2) 单击创建 user 表的向导画面的“ 执行 ”按钮，出现设定 user 表字段的画面，如图 14.28 所示：



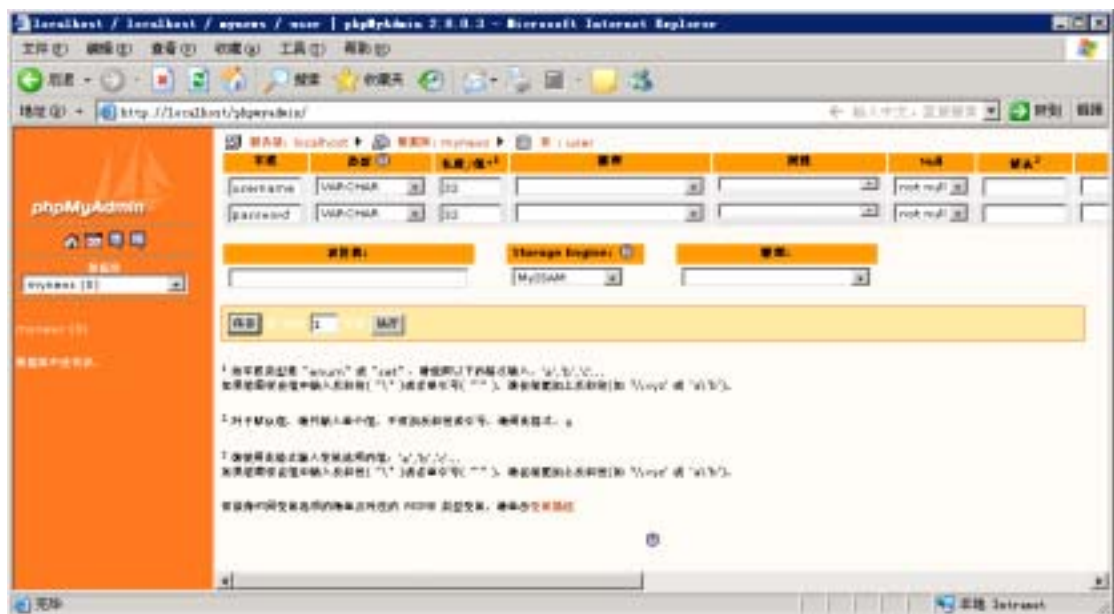


图 14.28 设定 user 表字段的画面

(3) 输入字段名称：username、password，都为 varchar 类型，长度为 32，主键为 username，然后单击“保存”按钮，创建 user 表，生成的 sql 语句如下所示：

```
CREATE TABLE `user` (
  `username` VARCHAR(32) NOT NULL ,
  `password` VARCHAR(32) NOT NULL ,
  PRIMARY KEY ( `username` )
) ENGINE = MYISAM ;
```

(4) 最终的 user 表结构如图 14.29 所示：

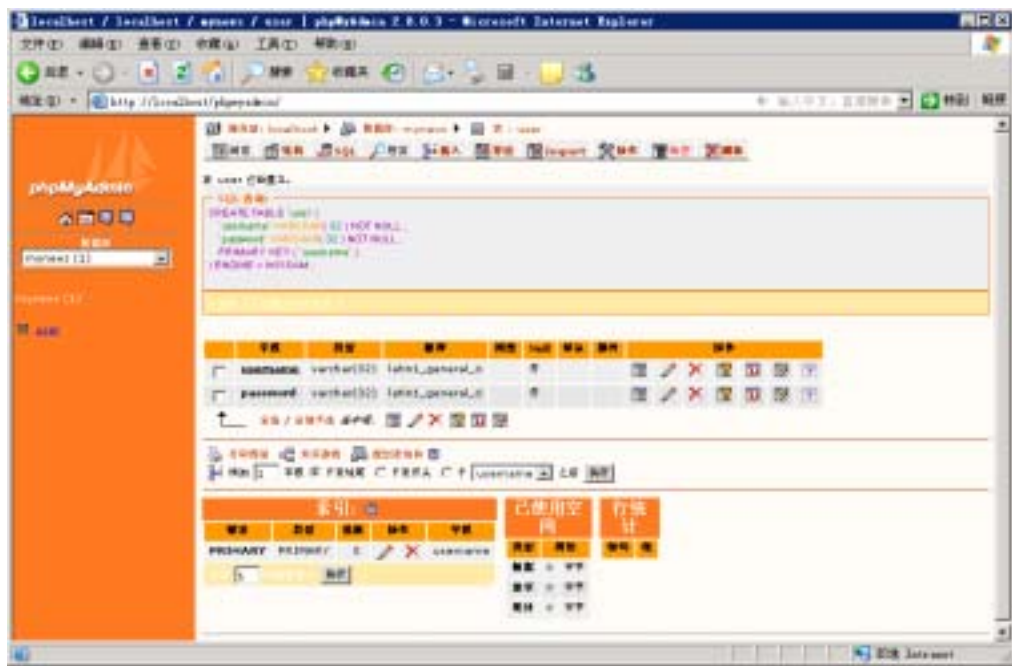


图 14.29 user 表结构

14.6.2 存放新闻的数据库表

(1) 根据创建一个存储新闻的表，表名为 news，主要包括 6 个字段，创建 news 表的向导如图 14.30 所示：

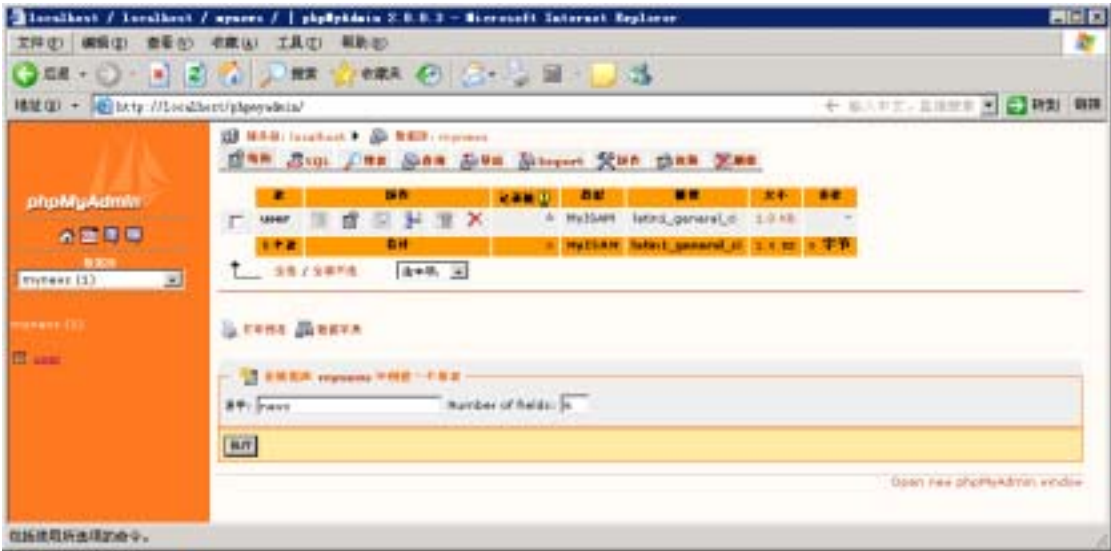


图 14.30 创建 news 表的向导

(2) 单击创建 news 表的向导画面的“执行”按钮，出现设定 news 表字段的画面，如图 14.31 所示：



图 14.31 设定 news 表字段的画面

(3) 输入字段名称：id、head、content、issuedate、issueuser、newstype，id 为 int 类型，长

度为 4；head 为 varchar 类型，长度为 200；content 为 varchar 类型，长度 8000；issuedate 为 Date 类型；issueuser 为 varchar 类型，长度 32；newstype 为 int 类型，长度为 4，主键为 id，，然后单击“保存”按钮，创建 news 表，生成的 sql 语句如下所示：

```
CREATE TABLE `news` (  
  `id` INT( 4 ) NOT NULL ,  
  `head` VARCHAR( 200 ) NOT NULL ,  
  `content` VARCHAR( 8000 ) NOT NULL ,  
  `issuedate` DATE NOT NULL ,  
  `issueuser` VARCHAR( 32 ) NOT NULL ,  
  `newstype` INT( 4 ) NOT NULL ,  
  PRIMARY KEY ( `id` )  
) ENGINE = MYISAM ;
```

(4) 最终的 news 表结构如图 14.32 所示：

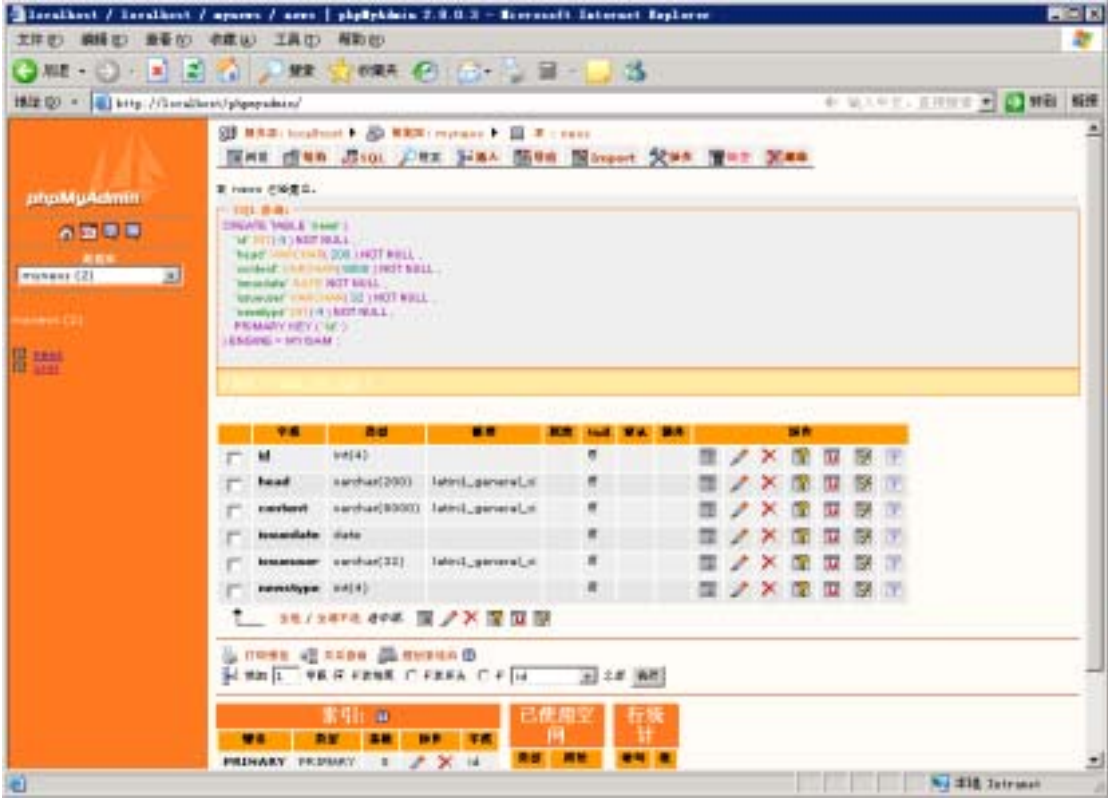


图 14.32 news 表结构

### 14.6.3 存放新闻类别的数据库表

(2) 存储用户授权信息的表，表名为 userAuthor，主要字段有：username、power，主键为 username、power。

(3) 存储新闻类别的表，表名为 newsType，主要字段有：id、type，主键为 id。

(4) 存储新闻的表，表名为 news，主要字段有：id、head、content、issuedate、issueuser、newstype，主键为 id。

(1) 根据创建一个存储新闻类别的表，表名为 newsType，主要包括 2 个字段，创建 newsType

表的向导如图 14.33 所示：

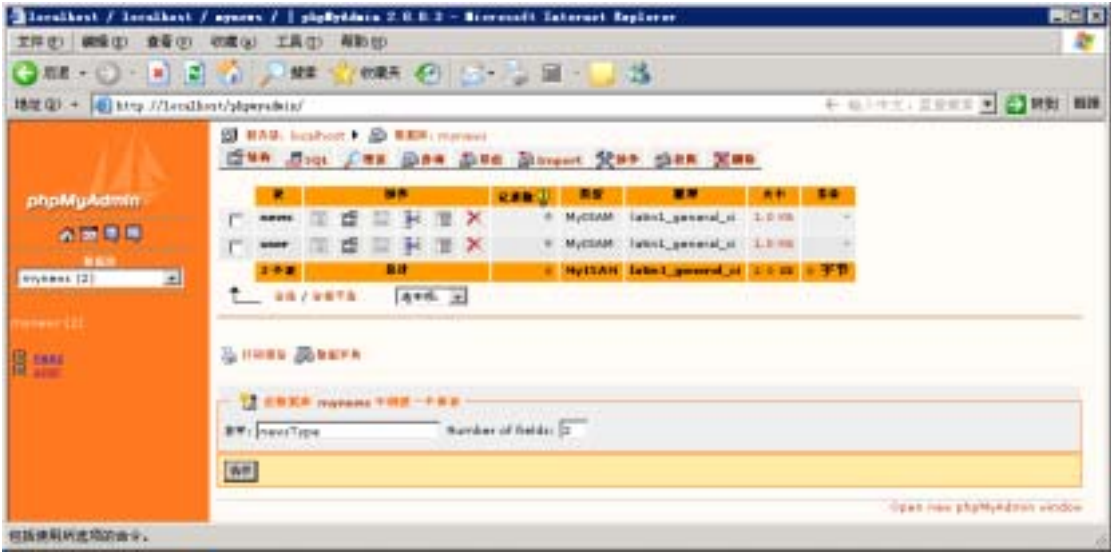


图 14.33 创建 newsType 表的向导

(2) 单击创建 newsType 表的向导画面的“执行”按钮，出现设定 newsType 表字段的画面，如图 14.34 所示：

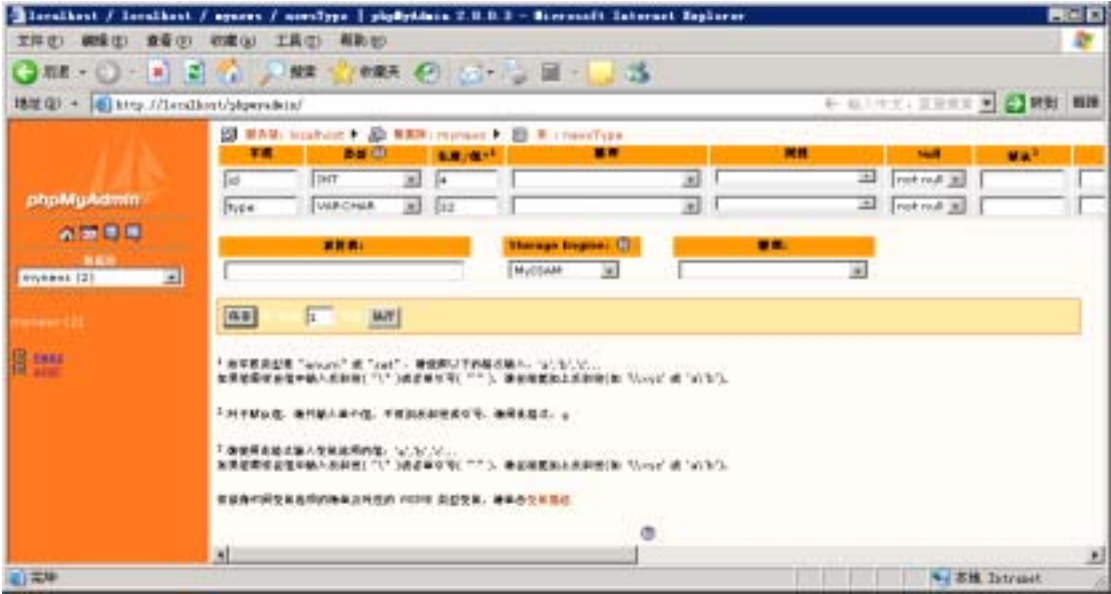


图 14.34 设定 newsType 表字段的画面

(3) 输入字段名称：id、type，id 为 int 类型，长度为 4；type 为 varchar 类型，长度为 32，主键为 id，然后单击“保存”按钮，创建 newsType 表，生成的 sql 语句如下所示：

```
CREATE TABLE `newsType` (  
  `id` INT(4) NOT NULL ,  
  `type` VARCHAR(32) NOT NULL ,  
  PRIMARY KEY (`id`)  
 ) ENGINE = MYISAM ;
```

(4) 最终的 newsType 表结构如图 14.35 所示：

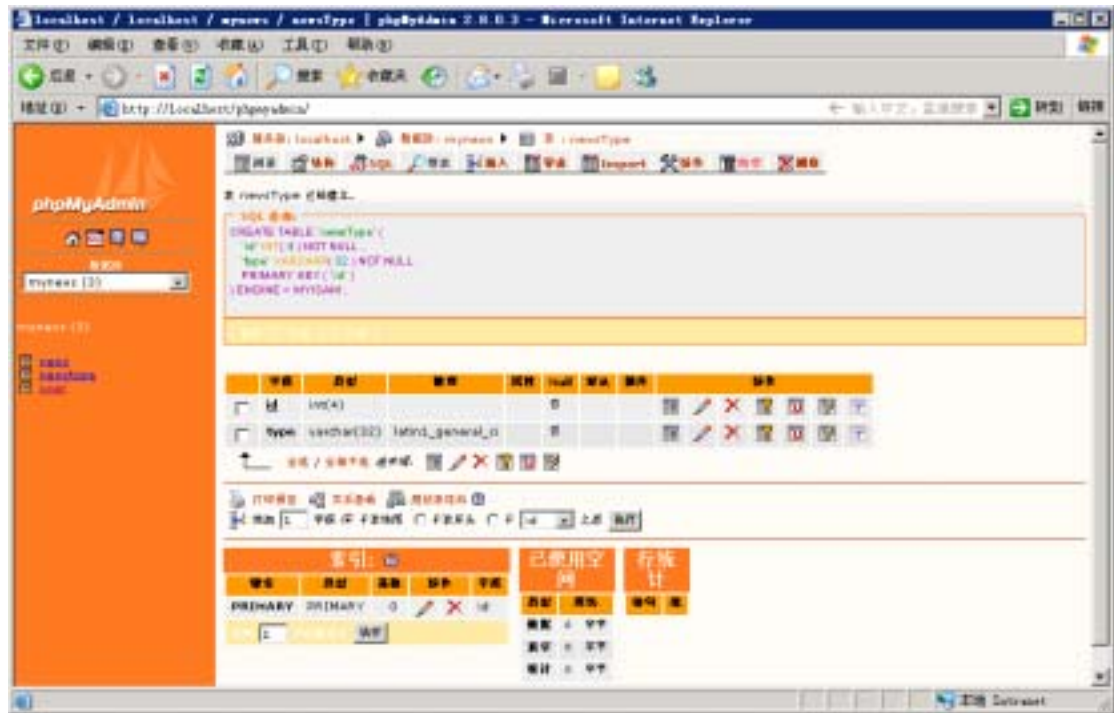


图 14.35 newsType 表结构