

轻量级 J2EE 框架应用

E 6 A Simple Controller with DAO pattern & O/R mapping

学号: SA16225221

姓名: 欧勇

报告撰写时间: 2017/1/1

1. 主题概述

简要介绍主题的核心内容

1. 基于 E 5。新建一个 XML 文件名为 or_mapping.xml，格式可参考如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<OR-Mappings>
  <jdbc>
    <property>
      <name>driver_class</name>
      <value>org.postgresql.Driver</value>
    </property>
    <property>
      <name>url_path</name>
      <value>jdbc:postgresql://localhost/mydatabase</value>
    </property>
    <property>
      <name>db_username</name>
      <value>user@1</value>
    </property>
    <property>
      <name>db_userpassword</name>
      <value>user@1password</value>
    </property>
  </jdbc>
  <class>
    <name>UserBean</name>
    <table>user</table>
    <id>
      <name>userID</name>
    </id>
    <property>
      <name>userName</name>
      <column>user_name</column>
      <type>String</type>
      <lazy>>false</lazy>
    </property>
    <property>
      <name>userPass</name>
      <column>user_pass</column>
      <type>String</type>
      <lazy>>true</lazy>
    </property>
    <!-- other properties -->
  </class>
  <!-- other classes -->
</OR-Mappings>
```

2. or_mapping.xml 中定义 JDBC 节点和 Class 节点。JDBC 节点为 java jdbc 属性配置：

class 节点为 O/R 映射的实现。如示例中对 UserBean 与 user 作了映射。

3. 在工程中新建 Configuration 类与 Conversation 类。Configuration 负责解析 or_mapping.xml；Conversation 负责完成将对象操作映射为数据表操作，即在 Conversation 中定义数据操作 CRUD 方法，每个方法将对象操作解释成目标数据库的 DML 或 DDL，通过 JDBC 完成数据持久化。

4. 修改 E 5 中的 DAO 代码，使用 Conversation，将 DAO 中的数据的 CRUD 操作全部修改为对对象的 CRUD 操作。如查询用户名为 user01 的数据，代码为 Conversation.getUser(“userName”，“user01”);(如果考虑对不同类型属性都可查询，可使用泛型作为查询方法参数类型，或者使用方法重载)

5. 将修改后的代码打包为 war，部署到 web container 中测试。

6. 将 5 中测试的数据库修改为另一个 DBMS，仅修改 Conversation 代码，重新进行打包和部署，并测试结果。如将 mysql 修改为 sqlite。

7. 实现对象属性 lazy-loading（可通过代理模式 Proxy Pattern 实现）。

2. 假设

主题内容所参照的假设条件，或假定的某故事场景，如调试工具或软硬件环境

开发环境：

Win10

Eclipse kepler

JDK 1.8

Tomcat 7.0

3. 实现或证明

对主题内容进行实验实现, 或例举证明, 需描述实现过程及数据。如对 MVC 中 Controller 功能的实现及例证（图示、数据、代码等）

注：本次 E6 作业报告是在 E5 作业报告的基础上完成的。同时“延时加载”与“懒加载”同义，本文使用“懒加载”。

流程：

为了方便查看，采用 get 方式提交，可以通过浏览器 url 看到输入的用户名和密码（因为若采用 post 方式则无法通过 url 看到用户名和密码，所以采用 get 方式提交）
若登录成功则跳转 login_success.jsp 页面，页面显示 Login Success 的字符串
若登录失败则跳转 login_fail.jsp 页面，页面显示 Login Fail 的字符串

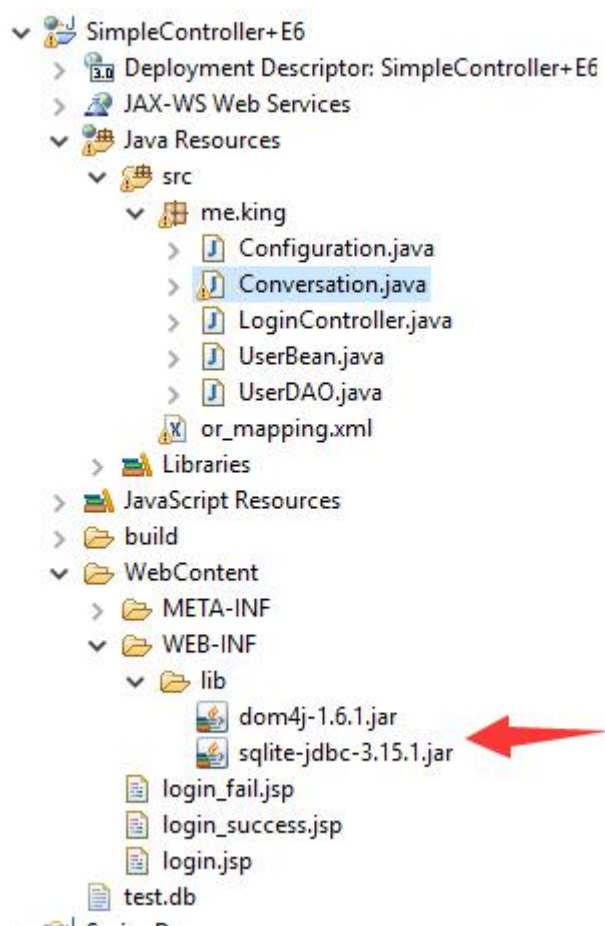


图 1：项目目录结构

图 1：项目目录结构，可以看出项目名称为 SimpleController+E6，包名为 me.king，分别有两个类，一个是 LoginController 作为控制层，一个是 UserBean 作为模型层，UserDAO 作为数据库连接类，由若干与用户有关操作，实际调用的是 Conversation 类中静态方法，

Conversation 为封装后的数据库操作方法，Configuration 类为配置类，将 or_mapping.xml 配置文件中的内容读入并保存，

在 WEB_INF/lib 下添加 sqlite-jdbc 的库，接下来将采用 sqlite 替换 mysql 测试；要读取 or_mapping.xml 配置文件，引入 dom4j 的库，

最后还有 3 个 jsp 页面作为视图层（同 E1），分别是 login_fail.jsp，login_success.jsp 和 login.jsp

```

1 package me.king;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13 * Servlet implementation class LoginController
14 @WebServlet("/LoginController")
15 public class LoginController extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     * @see HttpServlet#HttpServlet()
19     public LoginController() {
20         super();
21         // TODO Auto-generated constructor stub
22     }
23
24     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
25     protected void doGet(HttpServletRequest request,
26         HttpServletResponse response) throws ServletException, IOException {
27         // TODO Auto-generated method stub
28         ServletContext sc = getServletContext(); //获取servlet上下文
29         RequestDispatcher rd = null; //新建一个requestDispatcher对象用于内部重定向
30         //传入request中的用户名和密码新建一个UserBean对象
31         UserBean ub = new UserBean(request.getParameter("name"), request.getParameter("pwd"));
32         try { //若登录成功则设置重定向到login_success.jsp,否则为login_fail.jsp
33             if (ub.login()) rd = sc.getRequestDispatcher("/login_success.jsp");
34             else rd = sc.getRequestDispatcher("/login_fail.jsp");
35         }
36         rd.forward(request, response);
37     } catch (Exception e) {
38         // TODO Auto-generated catch block
39         e.printStackTrace();
40     }
41 }
42
43     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
44     protected void doPost(HttpServletRequest request,
45         HttpServletResponse response) throws ServletException, IOException {
46         // TODO Auto-generated method stub
47         doGet(request, response);
48     }
49 }
50
51
52
53
54
55
56
57
58
59
60

```

图 2.1: LoginController 代码截图

图 2.1: LoginController 代码截图，

使用注解的方式告知容器，LoginController servlet 映射的 url 为“/LoginController”。

代码为：@WebServlet("/LoginController")

在 doGet 方法中，新建一个 UserBean 对象 ub，传入 request 对象中的 name 和 pwd。调用 ub 的 login 方法，若 login 验证用户名和密码成功返回 true 则设置跳转至 login_success.jsp 页面，否则跳转至 login_fail.jsp 页面。最后采用 RequestDispatcher.forward 将 request 和 response 两个对象返回至指定的页面。

其他采用默认，若前端页面采用 post 方式提交，则在 doPost 方法中也需要进行转发处

理，也可以直接调用 doGet()方法。

```

1 package me.king;
2
3 public class UserBean {
4     private String name;
5     private String pwd;
6
7     UserBean(String name, String pwd){
8         this.setName(name);
9         this.setPwd(pwd);
10    }
11
12    public boolean login() throws Exception{
13        UserDAO dao = new UserDAO(); //新建DAO类，通过DAO获取实体类数据
14        //queryUser第二个参数为ignLazy, false表示不忽略懒加载配置，
15        //ture为忽略懒加载，既无论如何都从数据库中取
16        if(pwd.equals( dao.queryUser(name, false)).getPwd() ))
17            return true;
18        else return false;
19    }
20
21    public String getName() {
22        return name;
23    }
24
25    public void setName(String name) {
26        this.name = name;
27    }
28
29    public String getPwd() throws Exception {
30        System.out.println("pwd:"+pwd);
31        if(pwd == null){ //表示配置了懒加载，此时需要获取真实数据，所以ignLazy设置为true
32            System.out.println("配置了懒加载，此时需要获取真实数据");
33            pwd = ((new UserDAO()).queryUser(name, true)).getPwd();
34        }
35        return pwd;
36    }
37
38    public void setPwd(String pwd) {
39        this.pwd = pwd;
40    }
41 }
42

```

图 2.2: UserBean 代码截图

图 2.2: UserBean 代码截图，

定义一个简单的无参 login 方法，函数体内需要先 new 一个 UserDAO 对象 dao，调用 dao 对象的 queryUser 方法，传入 name 获取对应的 UserBean 对象，然后对比数据库返回的 UserBean 对象的密码与当前 UserBean 对象的密码，若一致则表示正确，返回 true，否则返回 false。

注意 1: 相比作业 E5，此处 UserDAO 类的 queryUser 方法添加了第二个参数，第二个参数名为 ignLazy，false 表示不忽略懒加载配置，ture 为忽略懒加载，既无论如何都从数据库中取。

注意 2: 在 getPwd 中增加判断是否为 null，因为没有使用代理模式 Proxy Pattern 实现（比较复杂）所以此处必须采用每次 getPwd 都需判断是否为 null，若为 null 则表示 pwd 属性配置为懒加载，则此时需要从数据库取该属性值。


```

1 package me.king;
2
3 public class UserDAO {
4
5     // 使用Conversation, 查询用户名为userName 的数据, 同时使用ignoreLazy表示是否采用了无视懒加载
6     // 若ignoreLazy设置为true, 则不管配置文件中是否标识了懒加载, 直接获取所有的属性值
7     public UserBean queryUser(String userName, boolean ignoreLazy) throws Exception {
8         System.out.println("ignoreLazy: " + ignoreLazy);
9         return (ignoreLazy) ? Conversation.getUserNoLazy("name", userName)
10            : Conversation.getUser("name", userName);
11     }
12     /**//添加用户
13     public boolean insertUser(UserBean u) {
14         return false;
15     }
16     //更新用户
17     public boolean updateUser(UserBean u) {
18         return false;
19     }
20     //删除用户
21     public boolean deleteUser(UserBean u) {
22         return false;
23     }
24 }
25

```

图 3.1: UserDAO 类概览截图

图 3.1: UserDAO 类概览截图

本次作业并没由完全实现 CRUD 的所有操作，因为实现了查询，其他 CRUD 三种操作也就没什么不一样了，在实验报告中写上也仅仅是赘述而已，所以仅仅将接口预留出。在 queryUser 方法中，都是使用 Conversation 类中的方法，查询用户名为 userName 的数据同时使用 ignoreLazy 表示是否采用了无视懒加载若 ignoreLazy 设置为 true，则不管配置文件中是否标识了懒加载，直接获取所有的属性值。

```

1 package me.king;
2
3 import java.sql.Connection;
4
5
6
7
8
9
10
11
12 public class Conversation {
13     static {
14         try {
15             Class.forName(Configuration.getDbDriver());
16         } catch (ClassNotFoundException e) {
17             // TODO Auto-generated catch block
18             e.printStackTrace();
19         }
20     }
21     //打开数据库连接
22     public static Connection openDBConnection() throws Exception {}
23     //关闭数据库连接, 静态方法无法调用非静态方法和变量, 所以改为静态
24     public static boolean closeDBConnection(Connection con, Statement stat, ResultSet rs) {}
25
26     //name为实体名, value为实体值, 不用方法重载, 所以用Object类型, 需要找到name对应的表字段名
27     public static UserBean getUser(String property, Object value) throws Exception {}
28
29     //getUserNoLazy方法大体逻辑与getUser类似, 忽视lazy标签的配置, 会直接从数据库查找到pwd的值
30     public static UserBean getUserNoLazy(String property, Object value) throws Exception {}
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115 }

```

图 3.2: Conversation 类概览截图

图 3.2: Conversation 类概览截图

先从 Configuration 类中获取数据库 driver 类名，使用静态语句块能保证仅仅在类

Conversation 中执行一次获取 driver 的操作。

getUser 方法和 getUserNoLazy 方法的逻辑基本一致，唯一差别为 getUserNoLazy 方法中不管是否配置了 lazy 属性，将该 user 的所有数据从数据库中找出。

```
22 public static Connection openDBConnection() throws Exception {
23     if(useMySQL)
24         return (Connection)DriverManager.getConnection(Configuration.getDbUrl(), Configuration.getDbName(), Configuration.getDbPwd());
25     else
26         return (Connection)DriverManager.getConnection(Configuration.getDbUrl());
27 }
28 //静态方法无法调用非静态方法和变量，所以改为静态
29 public static boolean closeDBConnection(Connection con, Statement stat, ResultSet rs) {
30     try {
31         if (stat != null) stat.close();
32         if (rs != null) rs.close();
33         if (con != null) con.close();
34         return true;
35     } catch (SQLException e) {
36         // TODO Auto-generated catch block
37         e.printStackTrace();
38         return false;
39     }
40 }
```

图 3.2.1: Conversation 类 openDBConnection、closeDBConnection 方法截图

图 3.2.1 中，22 行是打开数据库连接的方法，28~40 行是关闭数据库连接的方法，与 E5 的区别主要是都是 static 静态方法，打开连接方法中使用的参数都从 Configuration 类获取，关闭连接方法中由于采用静态方法，所以需要传入关闭的 Connection，Statement，ResultSet 对象。

```

42 //name为实体名, value为实体值,不用方法重载,所以用Object类型,需要找到name对应的表字段名
43 public static UserBean getUser(String property, Object value) throws Exception{
44     System.out.println("getUser:");
45     Connection con = openDBConnection();
46     1 String table = null; //与此类对应的表名
47     String PK = null; //主键,本例中为name
48     String PKtype = null; //主键类型
49     String field = null; //在此类中与property对应的字段名,本例中为pwd
50     String type = null; //与property属性在对应的实体中属性类型,
51     boolean isLazy = new Boolean(false); //是否懒加载,默认否
52     List<Element> properties = null; //保存所有property标签
53
54     2 for(Element ele: Configuration.getClasses())
55         if(ele.elementText("name").equals("UserBean")){
56             3 table = ele.elementText("table"); //获取表名
57             properties = ele.elements("property");
58             //获取具体属性对应的字段名,在实体中该属性的类型,
59             4 if(ele.element("id").elementText("name").equals(property)){
60                 PK = ele.element("id").elementText("column");
61                 PKtype = ele.element("id").elementText("type");
62             }
63             5 for(Element p: properties)//从property标签中查找pwd
64                 if(p.elementText("name").equals("pwd")){
65                     field = p.elementText("column");
66                     type = p.elementText("type");
67                     isLazy = (p.elementText("lazy").equals("true"))? true : false;
68                 }
69             break;
70         }
71
72     6 if(table == null) return null; //若没有找到对应的配置表名则直接返回null
73     //查数据库须使用字符串类型的value,若pwd采用懒加载,则不查询该
74     7 String sql = "select "+ PK + ((isLazy)?"":","+"field) + " from "+table+" where "+ PK +"=" + (String)value + " ";
75     UserBean ub = null;
76     Statement stat = con.createStatement();
77     8 ResultSet rs = stat.executeQuery(sql); //执行查询数据库的操作
78     if (rs.next()){
79         ub = new UserBean(null, null);
80         9 if(type.equals("String")) //根据id的type属性设置,若是其他类型则也需一一对应
81             ub.setName(rs.getString(1));
82         //else; //当主键type为其他类型的时候
83         10 if(!isLazy){ //当不是lazy懒加载模式,本例子中只对pwd属性使用,
84             if(type.equals("String")) //根据id的type属性设置,若是其他类型则也需一一对应
85                 ub.setPwd(rs.getString(2));
86         }
87         //可以继续查找其他的实体属性
88     }
89     closeDBConnection(con, stat, rs); //关闭数据库连接
90     11 return ub;
91 }

```

图 3.2.2: Conversation 类 getUser 方法截图

图 3.2.2 中, 流程为:

1. 打开数据库连接, 并初始化一些变量
2. 从 Configuration 类中获取所有的 Class 配置标签并遍历解析
3. 查找 UserBean 实体类对应的数据库表 table, 查找配置文件中所有的 property 标签
4. 获取主键对应的字段名还有其实体类属性类型
5. 从 property 标签中查找 pwd 的对应字段名和类型以及是否 lazy 加载
6. 若没有找到数据库表 table 则直接返回 null
7. 根据流程 3~5 获取的数据库表和字段名以及传入 getUser 方法的 property 属性对应的字段名以及其 value 值拼接一个 sql 字符串
8. 执行原生的数据库 sql 查询操作
9. 由结果对象 rs, 若查找到了则将获取结果中的用户名设置到新建的 UserBean 对象中, 因为用户名为主键, 所以不需要判断是否为 lazy 模式
10. 由结果对象 rs 查找剩下的其他属性, 此时需要与属性一一对应, 同时需要判断配置文件中 lazy 标签书否为 true, 否则设置为 rs 中对应的值
11. 关闭数据库并返回 ub

```
or_mapping.xml Configuration.java UserDao.java UserBean.java Conversation.java LoginController.java
import java.io.File;
9
10 @SuppressWarnings("unchecked")
11 public class Configuration {
12     private static List<Element> classes = null; //保存所有的映射类
13     private static String dbDriver = null; //数据库驱动类
14     private static String dbUrl = null; //数据库地址
15     private static String dbName = null; //数据库用户名
16     private static String dbPwd = null; //数据库密码
17     static{
18         try{
19             String file = Thread.currentThread().getContextClassLoader().getResource("").getPath() + "or_mapping.xml";
20             Document dc = (new SAXReader()).read(new File(file)); // 获取文档对象
21             List<Element> jdbc = dc.getRootElement().element("jdbc").elements("property");
22             classes = dc.getRootElement().elements("class");
23
24             for(Element ele: jdbc){
25                 String name = ele.elementText("name");
26                 String value = ele.elementText("value");
27                 if(name.equals("driver_class")) //数据库驱动类
28                     dbDriver = value;
29                 else if(name.equals("url_path")) //数据库地址
30                     dbUrl = value;
31                 else if(name.equals("db_username")) //数据库用户名
32                     dbName = value;
33                 else if(name.equals("db_password")) //数据库密码
34                     dbPwd = value;
35             }
36         } catch (Exception e) {
37             // TODO Auto-generated catch block
38             e.printStackTrace();
39         }
40     }
41
42     public static List<Element> getClasses() {
43         return classes;
44     }
45
46     public static String getDbDriver() {
47         return dbDriver;
48     }
49
50     public static String getDbUrl() {
51         return dbUrl;
52     }
53
54     public static String getDbName() {
55         return dbName;
56     }
57
58     public static String getDbPwd() {
59         return dbPwd;
60     }
61 }
```

图 3.2.2: Configuration 类截图

图 3.2.2: Configuration 类中完成配置文件的读取，并解析数据库相关配置和映射类的相关配置。

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <OR-Mapping>
3   <jdbc>
4     <property>
5       <name>driver_class</name>
6       <value>com.mysql.jdbc.Driver</value> <!-- <value>org.sqlite.JDBC</value> -->
7     </property>
8     <property>
9       <name>url_path</name>
10      <value>jdbc:mysql://localhost:3306/monitor</value> <!-- <value>jdbc:sqlite:E:test.db</value> -->
11    </property>
12    <property>
13      <name>db_username</name>
14      <value>root</value>
15    </property>
16    <property>
17      <name>db_password</name>
18      <value>hello</value>
19    </property>
20  </jdbc>
21  <class>
22    <name>UserBean</name>
23    <table>monitor_user</table>
24    <id>
25      <name>name</name>
26      <column>user_Name</column>
27      <type>String</type>
28    </id>
29    <property>
30      <name>pwd</name>
31      <column>user_Password</column>
32      <type>String</type>
33      <lazy>true</lazy>
34    </property>
35  </class>
36 </OR-Mapping>

```

图 3.3: or_mapping.xml 配置文件截图

如图 3.3: or_mapping.xml 配置所示, 依次配置数据库驱动, 数据库连接 url, 数据库用户名、密码, 以及 UserBean 与对应的 monitor_user 表的映射关系。

Id 标签设置为主键, property 设置为其他属性。

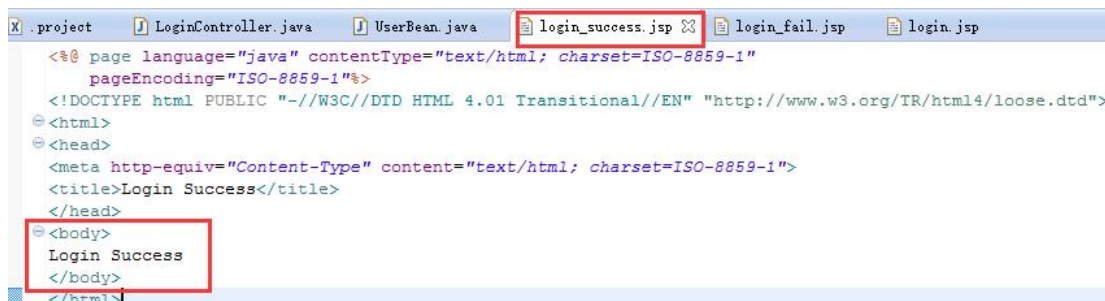
```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Login</title>
</head>
<body>
<form action="LoginController">
  <label>NAME:</label><input type="text" name="name"><br>
  <label>PASSWORD:</label><input type="password" name="pwd"><br>
  <input type="submit" value="LOGIN">
</form>
</body>
</html>

```

图 4.1: login.jsp 代码截图

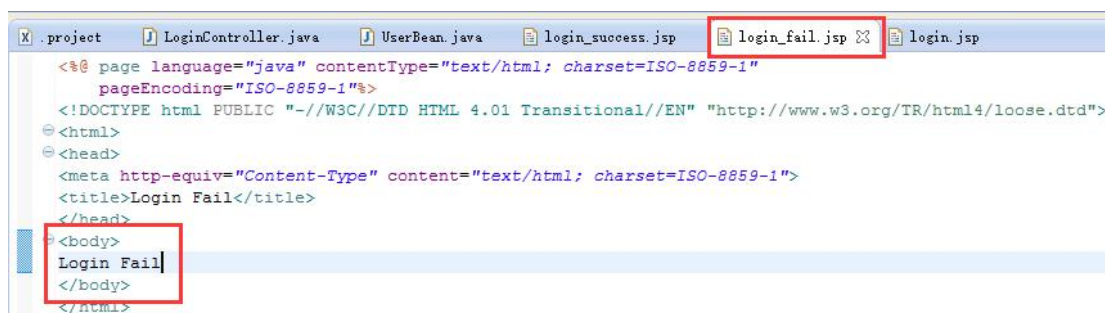
图 4.1: login.jsp 代码截图, 编写一个 form 表单, action 设置为 LoginController (就是在 LoginController 类中使用 @webServlet 注解映射的 url), 设置用户名和密码的 name 属性为 name 和 pwd。



```
.project LoginController.java UserBean.java login_success.jsp login_fail.jsp login.jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Login Success</title>
</head>
<body>
Login Success
</body>
</html>
```

图 4.2: login_success.jsp 代码截图

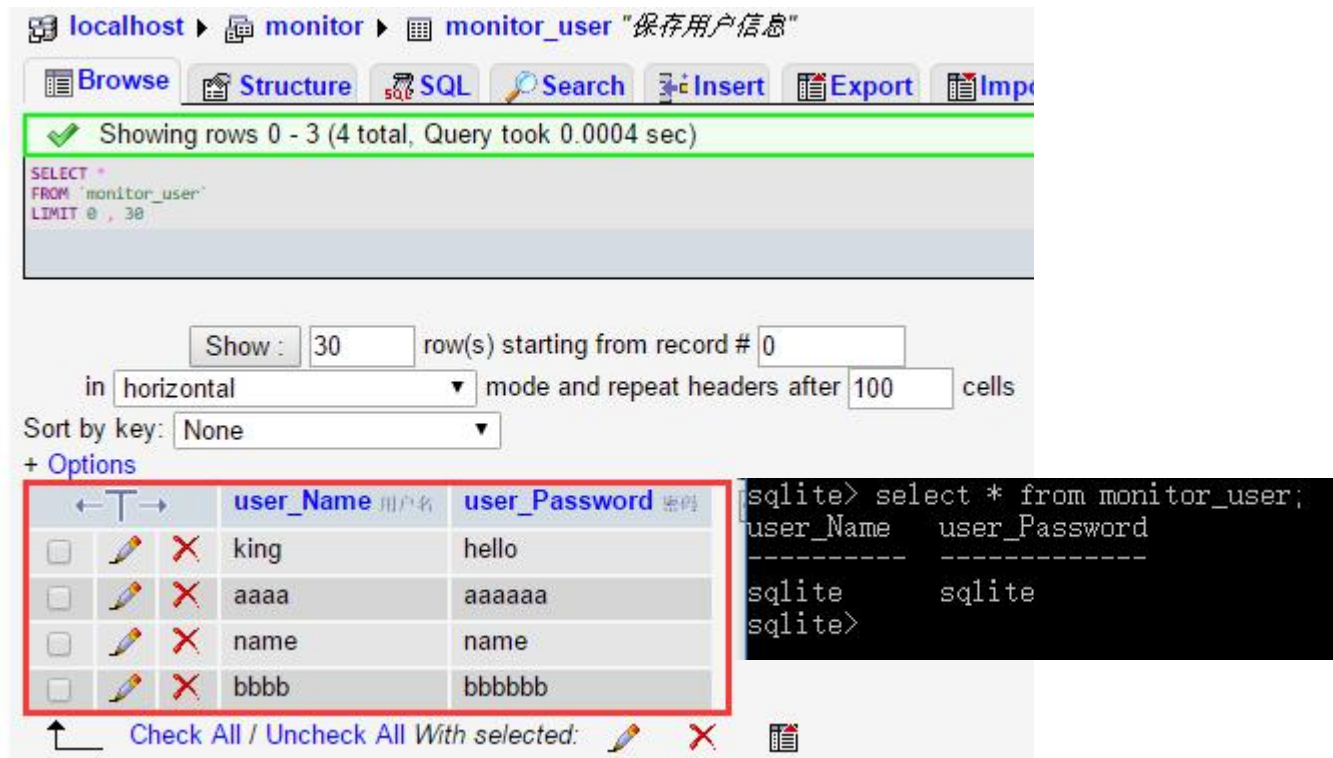
图 4.2: login_success.jsp 代码截图，
简单的在 body 中写入 Login Success。



```
.project LoginController.java UserBean.java login_success.jsp login_fail.jsp login.jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!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=ISO-8859-1">
<title>Login Fail</title>
</head>
<body>
Login Fail
</body>
</html>
```

图 4.3: login_fail.jsp 代码截图

图 4.3: login_fail.jsp 代码截图，
简单的在 body 中写入 Login Fail。



The screenshot displays a database management interface for the 'monitor_user' table. The table structure is as follows:

	user_Name	user_Password
<input type="checkbox"/>	king	hello
<input type="checkbox"/>	aaaa	aaaaaa
<input type="checkbox"/>	name	name
<input type="checkbox"/>	bbbb	bbbbbb

The interface also shows a terminal window with the following SQL query and output:

```
sqlite> select * from monitor_user;
user_Name  user_Password
-----
sqlite     sqlite
sqlite>
```

图 5. MySQL 和 SQLite 数据库的 monitor_user 表内数据展示

图 5. 用户表 monitor_user， user_Name 表示用户名， user_Password 表示密码，MySQL 数据库 4 位用户， SQLite 数据库只有 sqlite 一个用户，密码和用户名一样，为了测试方便，密码采用明文存储。

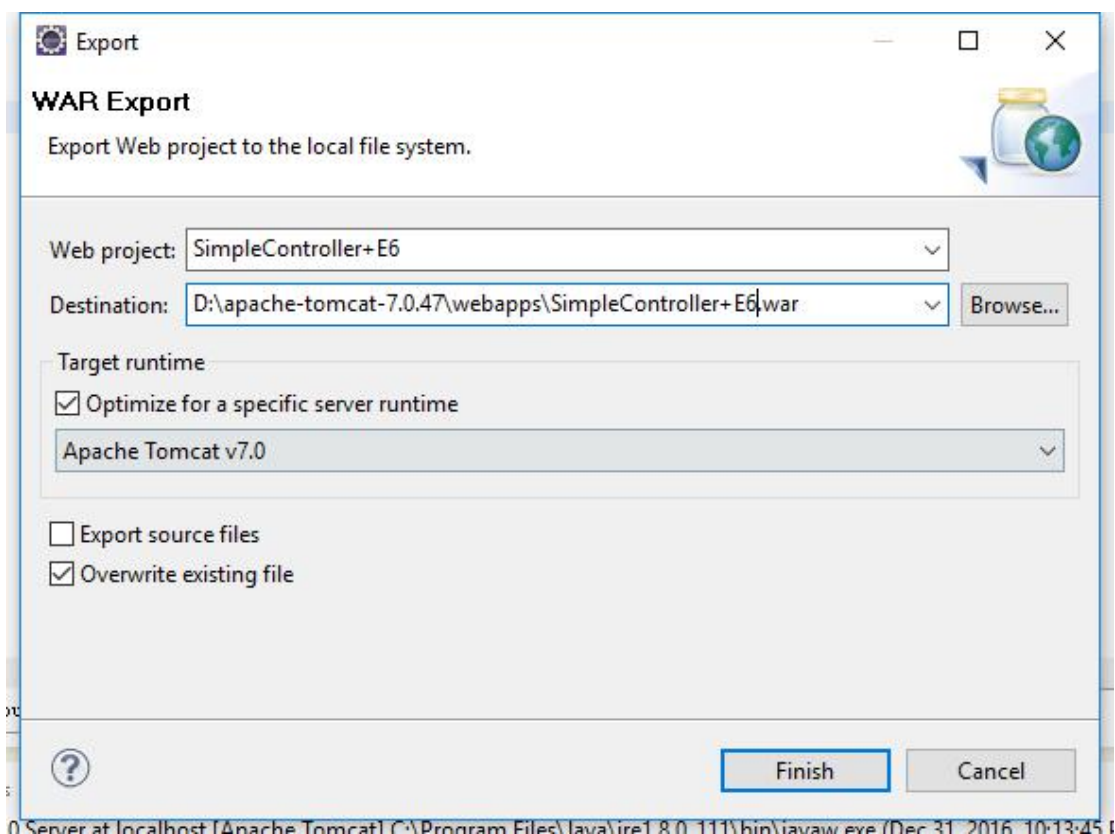


图 6.1: 导出为 War 包操作截图

图 6.1: 导出为 War 包操作截图，
在项目目录中右键选择 Export --> WAR file，然后选择导出到 Tomcat 目录下的 webapps 中即可。

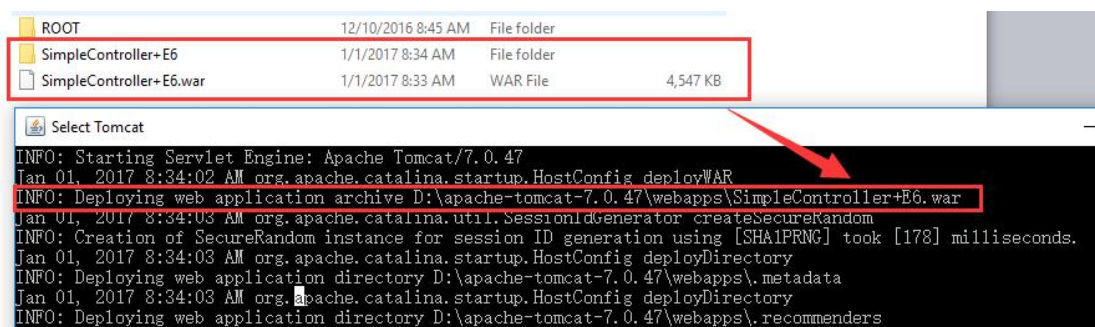


图 6.2: 部署在 Tomcat 中后使用 Chrome 浏览器测试截图

图 6.2: 部署在 Tomcat 中后使用 Chrome 浏览器测试截图，
在 Tomcat 目录下 bin 中选择 startup.bat 启动 Tomcat 服务，Tomcat 会自动将 war 包解压并将应用部署到同名文件夹下，然后在浏览器地址栏输入 SimpleController 应用的登录页面地址：

<http://localhost:8080/SimpleController+E6/login.jsp>

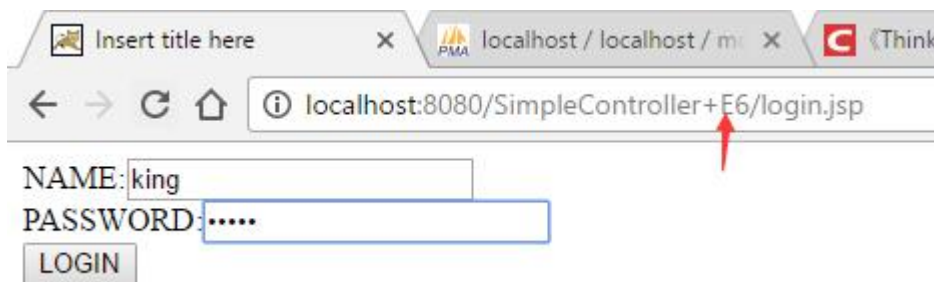


图 7.1：使用 Chrome 浏览器测试截图

图 7.1：使用 Chrome 浏览器测试截图，
可以看到 url 为：<http://localhost:8080/SimpleController+E6/login.jsp>，可以在表单中分别填入 NAME 和 PASSWORD，点击 LOGIN 按钮提交至后台服务器。

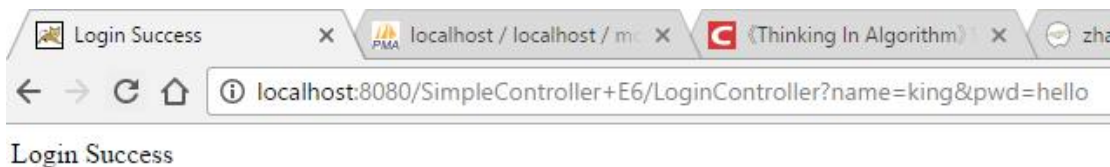


图 7.2：使用 MySQL 数据库登录成功截图

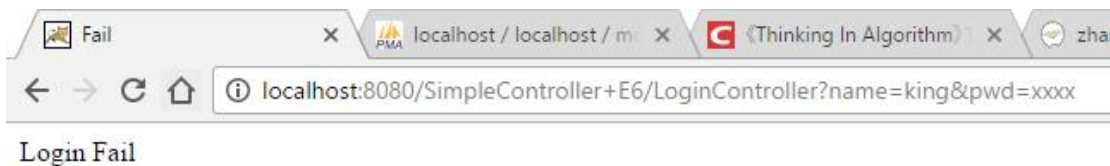


图 7.3：使用 MySQL 数据库登录失败截图

当数据库采用 MySQL 的时候，图 7.2：登录成功截图，图 7.3：登录失败截图
可以通过 url 看到，当输入的用户名 name 为 king，密码 pwd 为 hello 时验证通过时显示 Login Success
当输入的用户名 name 为 king，密码 pwd 为 xxxx 时，验证失败，显示 Login Fail
(虽然跳转至 login_fail.jsp 页面，但此时 url 没有改变为 login_fail.jsp)

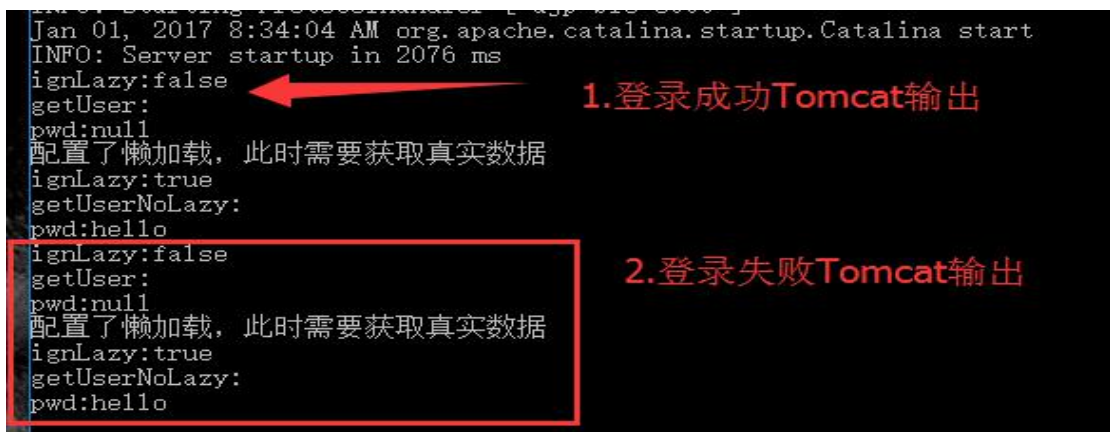


图 7.2.1: 使用 MySQL 数据库登录成功服务器控制台输出内容

图 7.2.1 所示，无论登录还是失败输出的 pwd 都是先为 null，然后是正确的密码，是因为在 UserBean 类的方法 getPwd 中的语句 `System.out.println("pwd:"+pwd);` 是输出从数据库获取的 pwd 值用于登录验证，要么为 null（懒加载模式下第一次获取用户信息，真正用到才会去数据库取），要么为正确密码（懒加载第二次获取用户信息，取自数据库），具体代码见上文图 2.2: UserBean 代码截图。

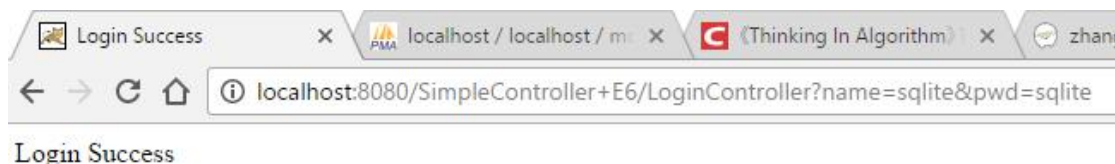


图 7.4: 使用 SQLite 数据库登录成功截图

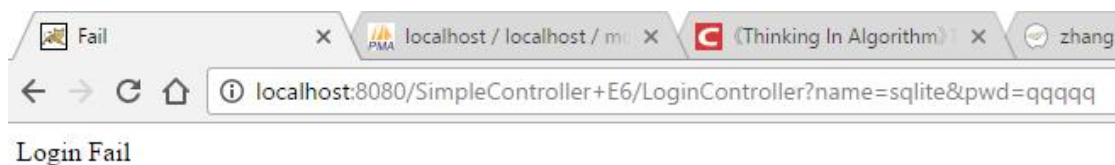


图 7.5: 使用 SQLite 数据库登录失败截图

当数据库采用 SQLite 的时候，图 7.4: 登录成功截图，图 7.5: 登录失败截图可以通过 url 看到，当输入的用户名 name 为 sqlite，密码 pwd 为 sqlite 时验证通过时显示 Login Success
当输入的用户名 name 为 sqlite，密码 pwd 为 qqqqq 时，验证失败，显示 Login Fail



图 7.6: 使用 SQLite 数据库登录 tomcat 输出截图

浏览器地址栏 URL 没有改变的原因是：

由于在 UserBean 中采用的是 `RequestDispatcher.forward()` 方式，这表示在服务器端运行。采用采用请求转发，`request` 对象始终存在，不会重新创建，前后页面共享同一个 `request` 重定向后浏览器地址栏 URL 不变。

若使用 `Response.sendRedirect()` 方式，则表示在用户的浏览器端工作。重新定向，前后页面不共享一个 `request`。重定向后在浏览器地址栏上会出现重定向页面的 URL。

4. 结论

对主题的总结，结果评论，发现的问题，或你的建议和看法。

作业中遇到的错误有：

找不到数据库表或字段，此时需要仔细检查是否是配置文件的变量名是否拼写错误；

或找不到 `or_mapping.xml` 配置文件，此时需要获取执行时的绝对路径，使用

```
Thread.currentThread().getContextClassLoader().getResource("").getPath()
```

 获取。

由于在测试中发现 `mysql` 和 `sqlite` 获取数据库连接的参数接口虽然不一致(`mysql` 需要有数据库用户名和密码，`sqlite` 不需要)，但由于一次测试失误忘记将 `mysql` 连接切换为 `sqlite` 数据库，发现并没有报错，所以取消了 E5 中用于判断切换数据库的 `useMySQL` 变量。

本次作业并没有实现添加、删除、修改用户的方法，也没有进行测试，主要是为了节约时间用于复习，因为这三种方式都比较简单，只要能顺利完成 OR 映射，并且实现懒加载，完成登录测试，其他三种操作也就没有什么困难了。

同时也为了节约时间和开发测试简便，并没有配置多个实体类，多个属性，多种类型，没有对 `type` 进行反射而是采用 `if-else` 的方式，而且也没有采用代理模式实现懒加载。

最后想到一个问题，既然采用 `or_mapping.xml` 的配置文件的方式去实现数据库连接和操作，那么定义 `Conversation.getUser()` 这样的静态方法不是有些“掩耳盗铃”了么？也就是说，理论上既然采用了 ORM 的方式，那么 `Conversation` 是不应该提前“预判”到有 `User` 类才对。

然后网上查找一番，了解到 ORM 的核心原理主要是利用反射机制，使用这种机制可以得到类的信息，包括类的修饰符、方法、属性、继承的父类以及实现接口等信息。通过映射文件得到类名和属性名，然后根据类名和属性名调用相应的 `set` 和 `get` 方法。

以上 ORM 原理主要参考博客 <http://blog.csdn.net/keywaytang/article/details/7542363>

5.参考文献

以上内容的理论知识点或技术点如果参考了网上或印刷制品，请在这里罗列出来

[1] J2EE DAO: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

[2] MySQL: <http://www.mysql.com>

[2] SQLite: <http://www.runoob.com/sqlite/sqlite-commands.html>