**南 京 师 范 大 学**

**课程设计报告**



**题 目： 超市订单管理系统**

**学 院： 计算机科学与技术学院**

**专 业： 软件工程**

**姓 名： 樊世雄**

**学 号： 19170333**

**任课教师： 王**

**计算机科学与技术学院 制**

**二O二O年 七 月 二十五 日**

目录

[一、 项目开发介绍 4](#_Toc46680142)

[1.1 课程设计要求 4](#_Toc46680143)

[1.2 项目简介 4](#_Toc46680144)

[1.3 项目设计原则 4](#_Toc46680145)

[1.3.1 规范性原则 4](#_Toc46680146)

[1.3.2 可靠性原则 5](#_Toc46680147)

[1.3.3 拓展性原则 5](#_Toc46680148)

[1.3.4 开放性原则 5](#_Toc46680149)

[1.3.5 易用性原则 5](#_Toc46680150)

[1.3.6 保密安全原则 5](#_Toc46680151)

[1.4 软件实现环境与技术 5](#_Toc46680152)

[二、 系统设计 6](#_Toc46680153)

[0、 前言： 6](#_Toc46680154)

[1、模块设计： 7](#_Toc46680155)

[2、数据库设计 8](#_Toc46680156)

[三、 系统实现 8](#_Toc46680157)

[1、 操作数据库实现： 9](#_Toc46680158)

[2、登录功能实现： 13](#_Toc46680159)

[2、 注销功能实现： 18](#_Toc46680160)

[4、 登录拦截和字符过滤实现： 19](#_Toc46680161)

[5、 密码修改实现 21](#_Toc46680162)

[6、 用户管理功能实现 26](#_Toc46680163)

[6.1、获取用户数量 26](#_Toc46680164)

[6.2、获取用户列表 28](#_Toc46680165)

[6.3、获取角色操作 30](#_Toc46680166)

[6.4、用户显示 32](#_Toc46680167)

[7、 数据库： 33](#_Toc46680168)

[四、项目运行结果展示 34](#_Toc46680169)

[1、登录页面 34](#_Toc46680170)

[2、未登录却访问主页面，遭到拦截： 34](#_Toc46680171)

[3、用户管理 35](#_Toc46680172)

[4、密码修改 38](#_Toc46680173)

[四、 遇到的问题以及解决方案 38](#_Toc46680174)

[0、 Junit单元测试 38](#_Toc46680175)

[1、数据库连接问题： 38](#_Toc46680176)

[2、tomcat问题： 39](#_Toc46680177)

[3、maven部署 39](#_Toc46680178)

[4、乱码问题 39](#_Toc46680179)

[六、SSM框架 39](#_Toc46680180)

[0、 配置文件 40](#_Toc46680181)

[1、 拦截器 47](#_Toc46680182)

[2、 登录 48](#_Toc46680183)

[3、 修改密码： 50](#_Toc46680184)

[4、 用户视图 51](#_Toc46680185)

[七、收获与体会 54](#_Toc46680186)

[1、 servlet版本优缺点 54](#_Toc46680187)

[2、SSM 54](#_Toc46680188)

[2.1、SSM项目各层作用 54](#_Toc46680189)

[2.2、使用SSM框架的好处 54](#_Toc46680190)

[3、收获 55](#_Toc46680191)

# 项目开发介绍

## 1.1 课程设计要求

学习JavaWeb，学会使用tomcat部署网站，同时也了解企业框架SSM，掌握MVC设计模式，掌握JDBC数据连接，掌握Servlet，掌握JSP页面设计，掌握mysql数据库设计，搭建超市订单管理平台。使得该项目不仅满足目前的业务需要，还要满足将来业务的发展，而且需要具备良好的可扩展性、可以执行和可复用性。

## 1.2 项目简介

作为国内市场的一些中小型超市，它们在信息化过程中的步伐要落后于大中型超市，而对于这些企业的资源管理，信息的存储和处理也显得迫切需要，要适应市场竞争，就需要有高效的处理方式和管理方法，因此加快超市的信息化进程是必可少的。

超市订单管理系统依靠现代化的计算机信息处理技术来管理超市，从而节省了大量的人力、物力，改善了员工的工作条件，减轻了劳动强度，并且能够快速订单的显示管理和各种反馈信息分析，使管理人员快速对市场的变化做出相应的决策，加快超市经营管理效率。

超市订单管理系统需要满足来自多个功能。实现用户管理，供应商管理和订单管理。

## 1.3 项目设计原则

### 1.3.1 规范性原则

在项目的建设过程中，按照“总体设计，分步实施”的原则，软件工程必须标准化，遵循国际通用开发标准，并按国家标准及行业标准执行。结合项目需求，形成接口规范、数据规范、应用系统接入等规范，参与验收和评估。

### 1.3.2 可靠性原则

作为一个对公众开放的开放平台，必须保证长期大规模访问下系统的稳定性、可用性和运行性能，避免单点故障和关键信息丢失。

### 1.3.3 拓展性原则

系统应采用模块化、组件化的体系结构，在技术架构和设计模式上保证技术的延续性，灵活的扩展性和广泛的适应性，确保系统能够满足用户在数据及业务功能扩展方面的需求。

### 1.3.4 开放性原则

在内容模块设计与网上实现过程中充分考虑开放性，便于日后内容维护和扩展，同时要充分考虑与网站现有后台接口和二次开发。

### 1.3.5 易用性原则

一般用户网站访问习惯和要求，提供灵活易用页面布局。

### 1.3.6 保密安全原则

具备统一完善的多级安全机制设置，符合国家安全及保密部门要求，拒绝非法用户和合法用户越权操作，避免系统数据遭到破坏，防止系统数据被窃取和篡改，对于关键信息使用加密传输，传输的数据文件提供不可抵赖性确认。

## 1.4 软件实现环境与技术

A 系统技术平台选择

本系统是基于web开发，应用MySQL8.0数据库进行开发的一款在线学习系统，在稳定性、扩展性等方面都有可靠的保障。

B 软件环境

网络终端操作系统：Windows 10

网络服务端操作系统：Windows 10

C 数据库

MySQL 8.0

D 开发工具

DataGrip、IDEA

**E 技术**

前端：JavaScript、jQuery、bootstrap、ajax等

后端：maven、SSM、mysql等

版本控制：使用github对代码进行有效的版本控制。

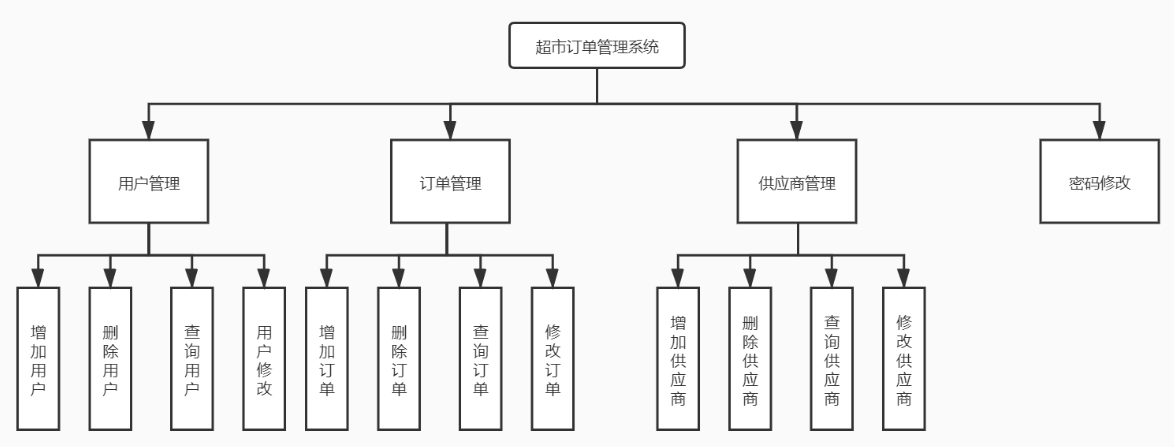
我的github主页：<https://github.com/fanshixiong>

本次项目：<https://github.com/fanshixiong/Java/tree/master/Jav_Web/javaweb-smbms>

Java目录是我在学习web时所建立的目录。

# 系统设计

超市订单管理系统需要满足来自多个功能。实现用户管理，供应商管理和订单管理。项目框架图如下：

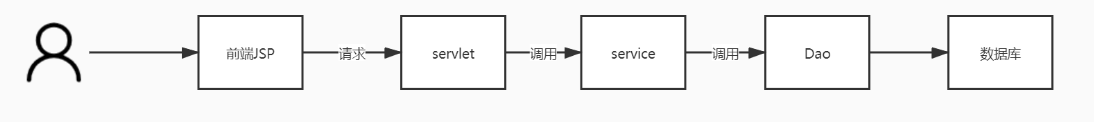


## 前言：

本项目使用servlet开发；servlet工作原理：

1. Servlet处理请求的过程
2. 客户端发送请求给服务器。
3. 容器根据请求及web.xml判断对应的Servlet是否存在，如果不存在则返回404
4. 容器根据请求及web.xml判断对应的Servlet是否已经被实例化，若是相应的Servlet没有被实例化，则容器将会加载相应的Servlet到Java虚拟机并实例化
5. 调用实例对象的service()方法，并开启一个新的线程去执行相关处理。调用servce方法，判断是调用doGet方法还是doPost方法
6. 业务完成后响应相关的页面发送给客户端。

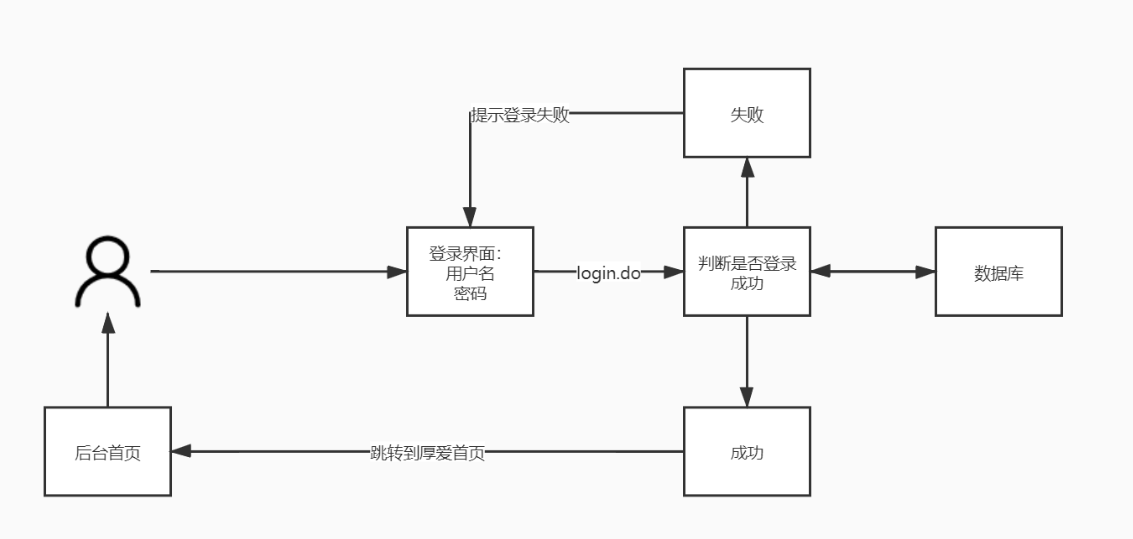
用户求一次数据流处理如图：



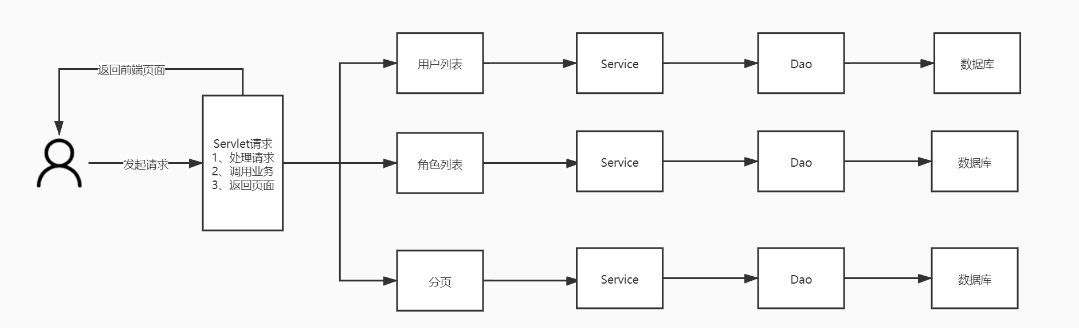
在开发中，遵循先编写dao层，再编写service层，最后写servlet的顺序。

## 1、模块设计：

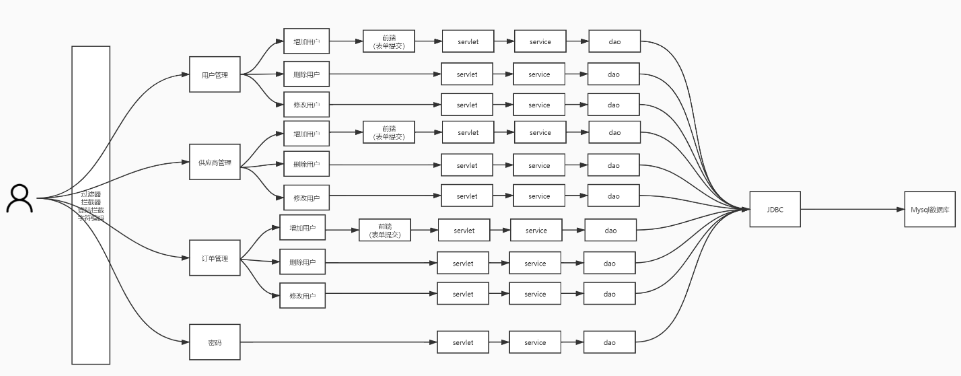
登录流程如下图：



用户管理流程如图：

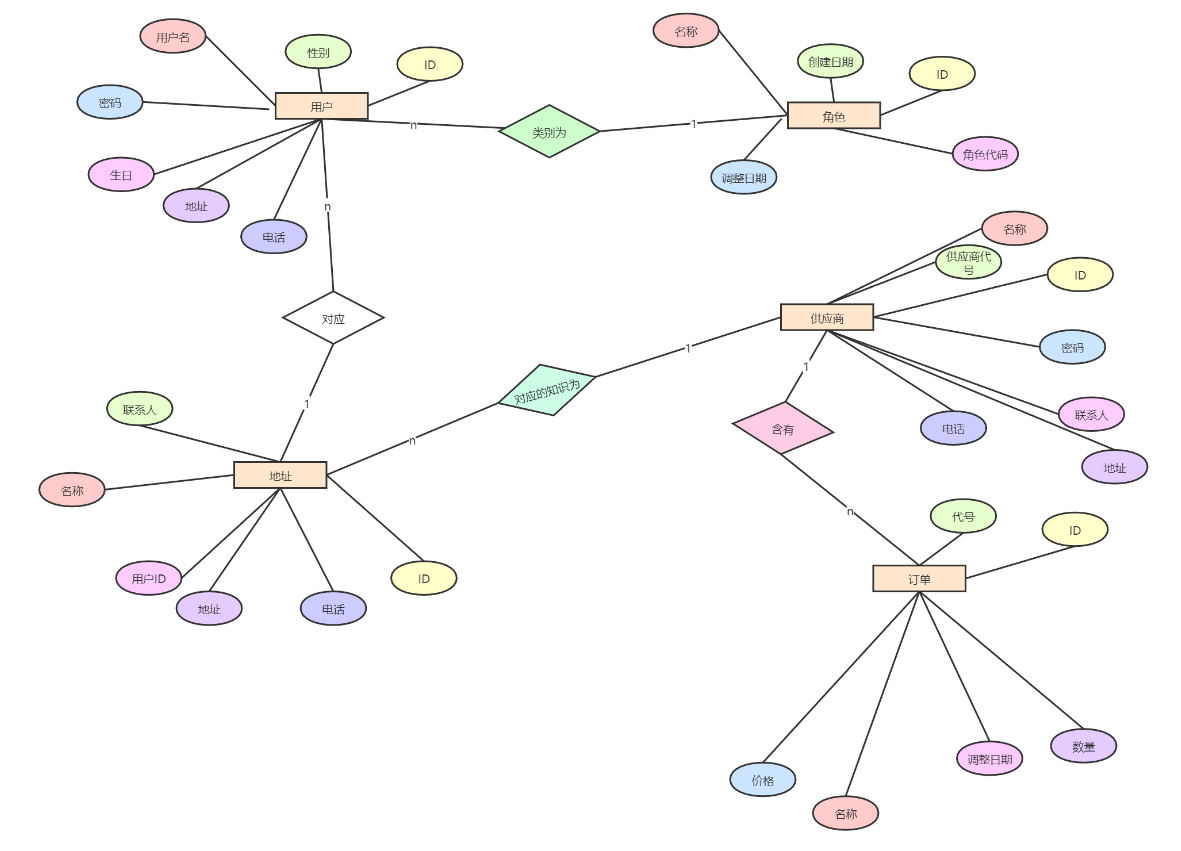


总体流程：



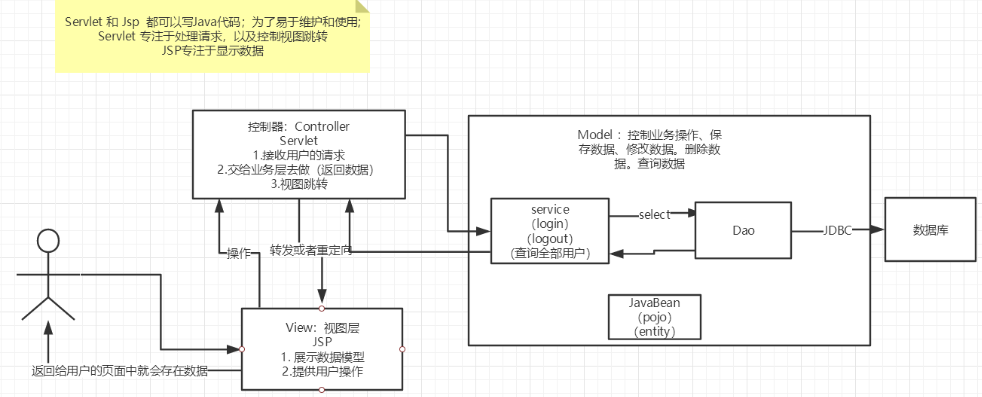
## 2、数据库设计

数据库E-R图：



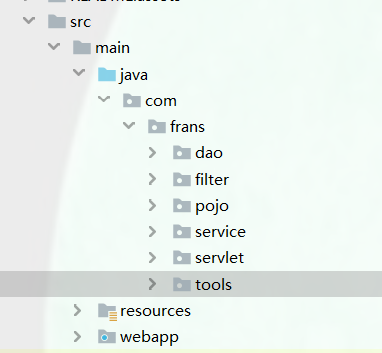
# 系统实现

本项目采用MVC设计模式进行开发。流程如下：



* **Model（模型）** - 模型代表一个存取数据的对象或 Java pojo（Plain Ordinary Java Object简单的Java对象）。它也可以带有逻辑，在数据变化时更新控制器。
* **View（视图）** - 视图代表模型包含的数据的可视化。
* **Controller（控制器）** - 控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。

**项目目录：**



## 操作数据库实现：

数据库公共类：

**package com.frans.dao**;  
  
**import java.io.IOException**;  
**import java.io.InputStream**;  
**import java.sql.**\*;  
**import java.util.Properties**;  
  
/\*\*  
 \* 操作数据库的基类--静态类  
 \* **@author** Administrator  
 \*  
 \*/  
**public class BaseDao** {  
   
 **static**{//静态代码块,在类加载的时候执行  
 **init**();  
 }  
   
 **private static String driver**;  
 **private static String url**;  
 **private static String user**;  
 **private static String password**;  
   
 //初始化连接参数,从配置文件里获得  
 **public static void** init(){  
 **Properties** params = **new** Properties();  
 **String** configFile = **"db.properties"**;  
 **InputStream** is=**BaseDao**.**class**.getClassLoader().getResourceAsStream(configFile);  
 **try** {  
 params.load(is);  
 } **catch** (**IOException** *e*) {  
 *e*.printStackTrace();  
 }  
 **driver**=params.getProperty(**"driver"**);  
 **url**=params.getProperty(**"url"**);  
 **user**=params.getProperty(**"username"**);  
 **password**=params.getProperty(**"password"**);  
  
 }   
   
   
 /\*\*  
 \* 获取数据库连接  
 \* **@return** \*/  
 **public static Connection** getConnection(){  
 **Connection** connection = **null**;  
 **try** {  
 **Class**.**forName**(**driver**);  
 //System.out.println(url +" " + user + " " + password);  
 connection = **DriverManager**.**getConnection**(**url**, **user**, **password**);  
 } **catch** (**Exception** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 }  
   
 **return** connection;  
 }  
 /\*\*  
 \* 查询操作  
 \* **@param connection** \* **@param pstm** \* **@param rs** \* **@param sql** \* **@param params** \* **@return** \*/  
 **public static ResultSet** execute(**Connection** *connection*, **PreparedStatement** pstm, **ResultSet** rs,  
 **String** *sql*, **Object**[] *params*) **throws Exception** {  
 pstm = *connection*.prepareStatement(*sql*);  
 **for**(**int** i = **0**; i < *params*.length; i++){  
 pstm.setObject(i+**1**, *params*[i]);  
 }  
 rs = pstm.executeQuery();  
 **return** rs;  
 }  
 /\*\*  
 \* 更新操作  
 \* **@param connection** \* **@param pstm** \* **@param sql** \* **@param params** \* **@return** \* **@throws Exception** \*/  
 **public static int** execute(**Connection** *connection*, **PreparedStatement** pstm,  
 **String** *sql*, **Object**[] *params*) **throws Exception** {  
 **int** updateRows = **0**;  
 pstm = *connection*.prepareStatement(*sql*);  
 **for**(**int** i = **0**; i < *params*.length; i++){  
 pstm.setObject(i+**1**, *params*[i]);  
 }  
 updateRows = pstm.executeUpdate();  
 **return** updateRows;  
 }  
   
 /\*\*  
 \* 释放资源  
 \* **@param connection** \* **@param pstm** \* **@param rs** \* **@return** \*/  
 **public static boolean** closeResource(**Connection** connection, **PreparedStatement** pstm, **ResultSet** rs){  
 **boolean** flag = **true**;  
 **if**(*rs* != **null**){  
 **try** {  
 *rs*.close();  
 rs = **null**;//GC回收  
 } **catch** (**SQLException** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 flag = **false**;  
 }  
 }  
 **if**(*pstm* != **null**){  
 **try** {  
 *pstm*.close();  
 pstm = **null**;//GC回收  
 } **catch** (**SQLException** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 flag = **false**;  
 }  
 }  
 **if**(*connection* != **null**){  
 **try** {  
 *connection*.close();  
 connection = **null**;//GC回收  
 } **catch** (**SQLException** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 flag = **false**;  
 }  
 }  
   
 **return** flag;  
 }  
  
}

通过操作数据库公共类，避免了很多重复操作。

从这里就能明显得感觉到，虽然编写了公共类，但是代码编写量还是非常大，由此学习MyBatis框架就可以明显减少代码。

## 2、登录功能实现：

2.1 编写前端页面登录界面使用bootstrap框架，更加美观。

<%@ **page** *language*="**java**" *contentType*="**text/html; charset=UTF-8**"  
 *pageEncoding*="**UTF-8**"%>  
<!DOCTYPE *html*>  
<**html**>  
<**head** *lang***="en"**>  
 <**meta** *charset***="UTF-8"**>  
 <**title**>系统登录 - 超市订单管理系统</**title**>  
  
 <**script** *src***="https://cdn.bootcss.com/jquery/2.2.4/jquery.min.js"**></**script**>  
 <**link** *href***="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css"** *rel***="stylesheet"**>  
 <**link** *href***="https://cdn.bootcss.com/font-awesome/4.7.0/css/font-awesome.min.css"** *rel***="stylesheet"**>  
 <**script** *src***="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"**></**script**>  
 <**link** *type***="text/css"** *rel***="stylesheet"** *href***="${**pageContext.request.contextPath **}/css/style.css"** />  
  
  
  
</**head**>  
<**body** *class* **= "login\_bg"**>  
<**div** *class***="container"**>  
 <**div** *class***="col-md-4"**></**div**>  
 <**div** *class***="col-md-4"** *style***="***background-color*:**#eee**;;*margin-top*:**200px**;**"**>  
 <**center**>  
 <**h3**>Login</**h3**>  
 </**center**>  
 <**form** *action***="${**pageContext.request.contextPath **}/login.do"** *method***="post"**>  
 <**div** *class***="form-group"**>  
 <**label** *for***="userCode"**>用户名:</**label**>  
 <**input** *type***="text"** *class***="form-control"** *style***="***color*:**red**;**"** *placeholder***="请输入用户名"** *id***="userCode"** *name***="userCode"** >  
 </**div**>  
 <**div** *class***="form-group"**>  
 <**label** *for***="userPassword"**>密码</**label**>  
 <**input** *type***="password"** *class***="form-control"** *style***="***color*:**red**;**"** *placeholder***="请输入密码"** *id***="userPassword"** *name***="userPassword"**>  
 </**div**>  
 <**div** *class***="form-group"**>  
 <**input** *type***="submit"** *value***="登录"** *class***="btn btn-success form-control"**>  
 <**input** *type***="reset"** *value***="重置"** *class***="btn btn-success form-control"**>  
 </**div**>  
 </**form**>  
 </**div**>  
 <**div** *class***="col-md-4"**></**div**>  
</**div**>  
</**body**>  
</**html**>

2.2 设置首页

<**welcome-file-list**>  
 <**welcome-file**>login.jsp</**welcome-file**>  
</**welcome-file-list**>

2.3 编写dao层登录用户接口

/\*\*  
 \* 通过userCode获取User  
 \* **@param connection** \* **@param userCode** \* **@return** \* **@throws Exception** \*/  
**public User** getLoginUser(**Connection** *connection*, **String** *userCode*)**throws Exception**;

2.4 编写dao层实现类

**@Override  
public User** getLoginUser(**Connection** *connection*, **String** *userCode*)  
 **throws Exception** {  
 // ***TODO Auto-generated method stub* PreparedStatement** pstm = **null**;  
 **ResultSet** rs = **null**;  
 **User** user = **null**;  
 **if**(**null** != *connection*){  
 **String** sql = **"select \* from smbms\_user where userCode=?"**;  
 **Object**[] params = {*userCode*};  
 rs = **BaseDao**.**execute**(*connection*, pstm, rs, sql, params);  
 **if**(rs.next()){  
 user = **new** User();  
 user.setId(rs.getInt(**"id"**));  
 user.setUserCode(rs.getString(**"userCode"**));  
 user.setUserName(rs.getString(**"userName"**));  
 user.setUserPassword(rs.getString(**"userPassword"**));  
 user.setGender(rs.getInt(**"gender"**));  
 user.setBirthday(rs.getDate(**"birthday"**));  
 user.setPhone(rs.getString(**"phone"**));  
 user.setAddress(rs.getString(**"address"**));  
 user.setUserRole(rs.getInt(**"userRole"**));  
 user.setCreatedBy(rs.getInt(**"createdBy"**));  
 user.setCreationDate(rs.getTimestamp(**"creationDate"**));  
 user.setModifyBy(rs.getInt(**"modifyBy"**));  
 user.setModifyDate(rs.getTimestamp(**"modifyDate"**));  
 }  
 **BaseDao**.**closeResource**(**null**, pstm, rs);  
 }  
 **return** user;  
}

2.5 业务层

/\*\*  
 \* 用户登录  
 \* **@param userCode** \* **@param userPassword** \* **@return** \*/  
**public User** login(**String** *userCode*, **String** *userPassword*);

实现类：

**@Override  
public User** login(**String** *userCode*, **String** *userPassword*) {  
 // ***TODO Auto-generated method stub* Connection** connection = **null**;  
 **User** user = **null**;  
 **try** {  
 connection = **BaseDao**.**getConnection**();  
 user = userDao.getLoginUser(connection, *userCode*);  
 } **catch** (**Exception** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 }**finally**{  
 **BaseDao**.**closeResource**(connection, **null**, **null**);  
 }  
 **System**.***out***.println(*userCode* + **" "** + *userPassword*);  
 //匹配密码  
 **if**(**null** != user){  
 **if**(!user.getUserPassword().equals(*userPassword*))  
 user = **null**;  
 }  
   
 **return** user;  
}

2.6 servlet：

**package com.frans.servlet.user**;  
  
**import com.frans.pojo.User**;  
**import com.frans.service.user.UserService**;  
**import com.frans.service.user.UserServiceImpl**;  
**import com.frans.tools.Constants**;  
  
**import javax.servlet.ServletException**;  
**import javax.servlet.http.HttpServlet**;  
**import javax.servlet.http.HttpServletRequest**;  
**import javax.servlet.http.HttpServletResponse**;  
**import java.io.IOException**;  
  
**public class LoginServlet extends HttpServlet** {  
  
  
 **public void** doGet(**HttpServletRequest** *request*, **HttpServletResponse** *response*)  
 **throws ServletException**, **IOException** {  
  
 }  
  
 **public void** doPost(**HttpServletRequest** *request*, **HttpServletResponse** *response*)  
 **throws ServletException**, **IOException** {  
  
 **System**.***out***.println(**"login ============ "** );  
 //获取用户名和密码  
 **String** userCode = *request*.getParameter(**"userCode"**);  
 **String** userPassword = *request*.getParameter(**"userPassword"**);  
 //调用service方法，进行用户匹配  
 **UserService** userService = **new** UserServiceImpl();  
 **User** user = userService.login(userCode,userPassword);  
 **if**(**null** != user){//登录成功  
 //放入session  
 *request*.getSession().setAttribute(**Constants**.***USER\_SESSION***, user);  
 //页面跳转（frame.jsp）  
 *response*.sendRedirect(**"jsp/frame.jsp"**);  
 }**else**{  
 //页面跳转（login.jsp）带出提示信息--转发  
 *request*.setAttribute(**"error"**, **"用户名或密码不正确"**);  
 *request*.getRequestDispatcher(**"login.jsp"**).forward(*request*, *response*);  
 }  
  
 }  
  
  
}

## 注销功能实现：

1. **package com.frans.servlet.user**;  
     
   **import com.frans.tools.Constants**;  
     
   **import javax.servlet.ServletException**;  
   **import javax.servlet.http.HttpServlet**;  
   **import javax.servlet.http.HttpServletRequest**;  
   **import javax.servlet.http.HttpServletResponse**;  
   **import java.io.IOException**;  
     
   **public class LogoutServlet extends HttpServlet** {  
     
    **public LogoutServlet**() {  
    **super**();  
    }  
     
    **public void** destroy() {  
    **super**.destroy();  
    }  
     
    **public void** doGet(**HttpServletRequest** *request*, **HttpServletResponse** *response*)  
    **throws ServletException**, **IOException** {  
    doPost(*request*, *response*);  
    }  
     
    **public void** doPost(**HttpServletRequest** *request*, **HttpServletResponse** *response*)  
    **throws ServletException**, **IOException** {  
    //清除session  
    *request*.getSession().removeAttribute(**Constants**.***USER\_SESSION***);  
    *response*.sendRedirect(*request*.getContextPath()+**"/login.jsp"**);  
    }  
     
    **public void** init() **throws ServletException** {  
     
    }  
     
   }

## 登录拦截和字符过滤实现：

登录拦截：

**package com.frans.filter**;  
  
**import com.frans.pojo.User**;  
  
**import javax.servlet.**\*;  
**import javax.servlet.http.HttpServletRequest**;  
**import javax.servlet.http.HttpServletResponse**;  
**import java.io.IOException**;  
  
**public class SysFilter implements Filter** {  
  
 **@Override  
 public void** init(**FilterConfig** *filterConfig*) **throws ServletException** {  
  
 }  
  
 **@Override  
 public void** doFilter(**ServletRequest** *request*, **ServletResponse** *response*,  
 **FilterChain** *chain*) **throws IOException**, **ServletException** {  
  
 **System**.***out***.println(**"SysFilter doFilter()==========="**);  
 **HttpServletRequest** rq = (**HttpServletRequest**)*request*;  
 **HttpServletResponse** rp = (**HttpServletResponse**)*response*;  
 **User** userSession = (**User**)rq.getSession().getAttribute(**"userSession"**);  
 **if**(**null** == userSession){  
 rp.sendRedirect(**"/smbms/error.jsp"**);  
 }**else**{  
 *chain*.doFilter(*request*, *response*);  
 }  
 }  
  
 **@Override  
 public void** destroy() {  
  
 }  
  
}

字符过滤：

**package com.frans.filter**;  
  
**import javax.servlet.**\*;  
**import java.io.IOException**;  
  
**public class CharacterEncoding implements Filter** {  
  
 **@Override  
 public void** init(**FilterConfig** *filterConfig*) **throws ServletException** {  
 // ***TODO Auto-generated method stub*** }  
  
 **@Override  
 public void** doFilter(**ServletRequest** *request*, **ServletResponse** *response*,  
 **FilterChain** *chain*) **throws IOException**, **ServletException** {  
 // ***TODO Auto-generated method stub*** *request*.setCharacterEncoding(**"UTF-8"**);  
 *response*.setCharacterEncoding(**"UTF-8"**);  
 *chain*.doFilter(*request*, *response*);  
 }  
  
 **@Override  
 public void** destroy() {  
 // ***TODO Auto-generated method stub*** }  
  
}

web.xml:

<**filter**>  
 <**filter-name**>CharacterEncoding</**filter-name**>  
 <**filter-class**>com.frans.filter.CharacterEncoding</**filter-class**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>CharacterEncoding</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>  
  
<**filter**>  
 <**filter-name**>SysFilter</**filter-name**>  
 <**filter-class**>com.frans.filter.SysFilter</**filter-class**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>SysFilter</**filter-name**>  
 <**url-pattern**>/jsp/\*</**url-pattern**>  
</**filter-mapping**>

## 密码修改实现

5.1 前端使用ajax进行判断，实现输入规范提示“√”，不规范提示“x”：

**var *oldpassword*** = **null**;  
**var *newpassword*** = **null**;  
**var *rnewpassword*** = **null**;  
**var *saveBtn*** = **null**;  
  
$(**function**(){  
 ***oldpassword*** = $(**"#oldpassword"**);  
 ***newpassword*** = $(**"#newpassword"**);  
 ***rnewpassword*** = $(**"#rnewpassword"**);  
 ***saveBtn*** = $(**"#save"**);  
   
 ***oldpassword***.next().html(**"\*"**);  
 ***newpassword***.next().html(**"\*"**);  
 ***rnewpassword***.next().html(**"\*"**);  
  
 ***oldpassword***.on(**"blur"**,**function**(){  
 $.ajax({  
 type:**"GET"**,  
 url:***path***+**"/jsp/user.do"**,  
 data:{method:**"pwdmodify"**,oldpassword:***oldpassword***.val()},  
 dataType:**"json"**,  
 success:**function**(data){  
 **if**(data.result == **"true"**){//旧密码正确  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"green"**},***imgYes***,**true**);  
 }**else if**(data.result == **"false"**){//旧密码输入不正确  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 原密码输入不正确"**,**false**);  
 }**else if**(data.result == **"sessionerror"**){//当前用户session过期，请重新登录  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 当前用户session过期，请重新登录"**,**false**);  
 }**else if**(data.result == **"error"**){//旧密码输入为空  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 请输入旧密码"**,**false**);  
 }  
 },  
 error:**function**(data){  
 //请求出错  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 请求错误"**,**false**);  
 }  
 });  
   
   
 }).on(**"focus"**,**function**(){  
 *validateTip*(***oldpassword***.next(),{**"color"**:**"#666666"**},**"\* 请输入原密码"**,**false**);  
 });  
   
 ***newpassword***.on(**"focus"**,**function**(){  
 *validateTip*(***newpassword***.next(),{**"color"**:**"#666666"**},**"\* 密码长度必须是大于6小于20"**,**false**);  
 }).on(**"blur"**,**function**(){  
 **if**(***newpassword***.val() != **null** && ***newpassword***.val().length > **6** && ***newpassword***.val().length < **20** ){  
 *validateTip*(***newpassword***.next(),{**"color"**:**"green"**},***imgYes***,**true**);  
 }**else**{  
 *validateTip*(***newpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 密码输入不符合规范，请重新输入"**,**false**);  
 }  
 });  
   
   
 ***rnewpassword***.on(**"focus"**,**function**(){  
 *validateTip*(***rnewpassword***.next(),{**"color"**:**"#666666"**},**"\* 请输入与上面一致的密码"**,**false**);  
 }).on(**"blur"**,**function**(){  
 **if**(***rnewpassword***.val() != **null** && ***rnewpassword***.val().length > **6** && ***rnewpassword***.val().length < **20** && ***newpassword***.val() == ***rnewpassword***.val()){  
 *validateTip*(***rnewpassword***.next(),{**"color"**:**"green"**},***imgYes***,**true**);  
 }**else**{  
 *validateTip*(***rnewpassword***.next(),{**"color"**:**"red"**},***imgNo*** + **" 两次密码输入不一致，请重新输入"**,**false**);  
 }  
 });  
   
   
 ***saveBtn***.on(**"click"**,**function**(){  
 ***oldpassword***.blur();  
 ***newpassword***.blur();  
 ***rnewpassword***.blur();  
 **if**(***oldpassword***.attr(**"validateStatus"**) == **"true"** && ***newpassword***.attr(**"validateStatus"**) == **"true"** && ***rnewpassword***.attr(**"validateStatus"**) == **"true"**){  
 **if**(*confirm*(**"确定要修改密码？"**)){  
 $(**"#userForm"**).submit();  
 }  
 }  
   
 });  
});

5.2 后端dao层：

/\*\*  
 \* 修改当前用户密码  
 \* **@param connection** \* **@param id** \* **@param pwd** \* **@return** \* **@throws Exception** \*/  
**public int** updatePwd(**Connection** *connection*, **int** *id*, **String** *pwd*)**throws Exception**;

Dao实现类：

**@Override  
public int** updatePwd(**Connection** *connection*, **int** *id*, **String** *pwd*)  
 **throws Exception** {  
 // ***TODO Auto-generated method stub* int** flag = **0**;  
 **PreparedStatement** pstm = **null**;  
 **if**(*connection* != **null**){  
 **String** sql = **"update smbms\_user set userPassword= ? where id = ?"**;  
 **Object**[] params = {*pwd*,*id*};  
 flag = **BaseDao**.**execute**(*connection*, pstm, sql, params);  
 **BaseDao**.**closeResource**(**null**, pstm, **null**);  
 }  
 **return** flag;  
}

5.3 service层：

/\*\*  
 \* 根据userId修改密码  
 \* **@param id** \* **@param pwd** \* **@return** \*/  
**public boolean** updatePwd(**int** *id*, **String** *pwd*);

实现类：

**@Override  
public boolean** updatePwd(**int** *id*, **String** *pwd*) {  
 **boolean** flag = **false**;  
 **Connection** connection = **null**;  
 **try**{  
 connection = **BaseDao**.**getConnection**();  
 **if**(userDao.updatePwd(connection,*id*,*pwd*) > **0**)  
 flag = **true**;  
 }**catch** (**Exception** *e*) {  
 *e*.printStackTrace();  
 }**finally**{  
 **BaseDao**.**closeResource**(connection, **null**, **null**);  
 }  
 **return** flag;  
}

5.4 servlet：

**private void** getPwdByUserId(**HttpServletRequest** *request*, **HttpServletResponse** *response*)  
 **throws ServletException**, **IOException** {  
 **Object** o = *request*.getSession().getAttribute(**Constants**.***USER\_SESSION***);  
 **String** oldpassword = *request*.getParameter(**"oldpassword"**);  
 **Map**<**String**, **String**> resultMap = **new** HashMap<**String**, **String**>();  
   
 **if**(**null** == o ){//session过期  
 resultMap.put(**"result"**, **"sessionerror"**);  
 }**else if**(**StringUtils**.**isNullOrEmpty**(oldpassword)){//旧密码输入为空  
 resultMap.put(**"result"**, **"error"**);  
 }**else**{  
 **String** sessionPwd = ((**User**)o).getUserPassword();  
 **if**(oldpassword.equals(sessionPwd)){  
 resultMap.put(**"result"**, **"true"**);  
 }**else**{//旧密码输入不正确  
 resultMap.put(**"result"**, **"false"**);  
 }  
 }  
   
 *response*.setContentType(**"application/json"**);  
 **PrintWriter** outPrintWriter = *response*.getWriter();  
 outPrintWriter.write(**JSONArray**.**toJSONString**(resultMap));  
 outPrintWriter.flush();  
 outPrintWriter.close();  
}

## 用户管理功能实现

### 6.1、获取用户数量

Dao：

/\*\*  
 \* 通过条件查询-用户表记录数  
 \* **@param connection** \* **@param userName** \* **@param userRole** \* **@return** \* **@throws Exception** \*/  
**public int** getUserCount(**Connection** *connection*, **String** *userName*, **int** *userRole*)**throws Exception**;

Impl：

**@Override  
public int** getUserCount(**Connection** *connection*, **String** *userName*, **int** *userRole*)  
 **throws Exception** {  
 // ***TODO Auto-generated method stub* PreparedStatement** pstm = **null**;  
 **ResultSet** rs = **null**;  
 **int** count = **0**;  
 **if**(*connection* != **null**){  
 **StringBuffer** sql = **new** StringBuffer();  
 sql.append(**"select count(1) as count from smbms\_user u,smbms\_role r where u.userRole = r.id"**);  
 **List**<**Object**> list = **new** ArrayList<**Object**>();  
 **if**(!**StringUtils**.**isNullOrEmpty**(*userName*)){  
 sql.append(**" and u.userName like ?"**);  
 list.add(**"%"**+*userName*+**"%"**);  
 }  
 **if**(*userRole* > **0**){  
 sql.append(**" and u.userRole = ?"**);  
 list.add(*userRole*);  
 }  
 **Object**[] params = list.toArray();  
 **System**.***out***.println(**"sql ----> "** + sql.toString());  
 rs = **BaseDao**.**execute**(*connection*, pstm, rs, sql.toString(), params);  
 **if**(rs.next()){  
 count = rs.getInt(**"count"**);  
 }  
 **BaseDao**.**closeResource**(**null**, pstm, rs);  
 }  
 **return** count;  
}

Service：

/\*\*  
 \* 根据条件查询用户表记录数  
 \* **@param queryUserName** \* **@param queryUserRole** \* **@return** \*/  
**public int** getUserCount(**String** *queryUserName*, **int** *queryUserRole*);

Impl：

**@Override  
public int** getUserCount(**String** *queryUserName*, **int** *queryUserRole*) {  
 // ***TODO Auto-generated method stub* Connection** connection = **null**;  
 **int** count = **0**;  
 **System**.***out***.println(**"queryUserName ---- > "** + *queryUserName*);  
 **System**.***out***.println(**"queryUserRole ---- > "** + *queryUserRole*);  
 **try** {  
 connection = **BaseDao**.**getConnection**();  
 count = userDao.getUserCount(connection, *queryUserName*,*queryUserRole*);  
 } **catch** (**Exception** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 }**finally**{  
 **BaseDao**.**closeResource**(connection, **null**, **null**);  
 }  
 **return** count;  
}

### 6.2、获取用户列表

Dao：

/\*\*  
 \* 通过条件查询-userList  
 \* **@param connection** \* **@param userName** \* **@param userRole** \* **@return** \* **@throws Exception** \*/  
**public List**<**User**> getUserList(**Connection** *connection*, **String** *userName*, **int** *userRole*, **int** *currentPageNo*, **int** *pageSize*)**throws Exception**;

Impl：

**@Override  
public List**<**User**> getUserList(**Connection** *connection*, **String** *userName*, **int** *userRole*, **int** currentPageNo, **int** *pageSize*)  
 **throws Exception** {  
 // ***TODO Auto-generated method stub* PreparedStatement** pstm = **null**;  
 **ResultSet** rs = **null**;  
 **List**<**User**> userList = **new** ArrayList<**User**>();  
 **if**(*connection* != **null**){  
 **StringBuffer** sql = **new** StringBuffer();  
 sql.append(**"select u.\*,r.roleName as userRoleName from smbms\_user u,smbms\_role r where u.userRole = r.id"**);  
 **List**<**Object**> list = **new** ArrayList<**Object**>();  
 **if**(!**StringUtils**.**isNullOrEmpty**(*userName*)){  
 sql.append(**" and u.userName like ?"**);  
 list.add(**"%"**+*userName*+**"%"**);  
 }  
 **if**(*userRole* > **0**){  
 sql.append(**" and u.userRole = ?"**);  
 list.add(*userRole*);  
 }  
 //分页 limit  
 sql.append(**" order by creationDate DESC limit ?,?"**);  
 currentPageNo = (currentPageNo-**1**)\**pageSize*;  
 list.add(currentPageNo);  
 list.add(*pageSize*);  
   
 **Object**[] params = list.toArray();  
 **System**.***out***.println(**"sql ----> "** + sql.toString());  
 rs = **BaseDao**.**execute**(*connection*, pstm, rs, sql.toString(), params);  
 **while**(rs.next()){  
 **User** \_user = **new** User();  
 \_user.setId(rs.getInt(**"id"**));  
 \_user.setUserCode(rs.getString(**"userCode"**));  
 \_user.setUserName(rs.getString(**"userName"**));  
 \_user.setGender(rs.getInt(**"gender"**));  
 \_user.setBirthday(rs.getDate(**"birthday"**));  
 \_user.setPhone(rs.getString(**"phone"**));  
 \_user.setUserRole(rs.getInt(**"userRole"**));  
 \_user.setUserRoleName(rs.getString(**"userRoleName"**));  
 userList.add(\_user);  
 }  
 **BaseDao**.**closeResource**(**null**, pstm, rs);  
 }  
 **return** userList;  
}

Service：

/\*\*  
 \* 根据条件查询用户列表  
 \* **@param queryUserName** \* **@param queryUserRole** \* **@return** \*/  
**public List**<**User**> getUserList(**String** *queryUserName*, **int** *queryUserRole*, **int** *currentPageNo*, **int** *pageSize*);

Impl：

**@Override  
public List**<**User**> getUserList(**String** *queryUserName*, **int** *queryUserRole*, **int** *currentPageNo*, **int** *pageSize*) {  
 // ***TODO Auto-generated method stub* Connection** connection = **null**;  
 **List**<**User**> userList = **null**;  
 **System**.***out***.println(**"queryUserName ---- > "** + *queryUserName*);  
 **System**.***out***.println(**"queryUserRole ---- > "** + *queryUserRole*);  
 **System**.***out***.println(**"currentPageNo ---- > "** + *currentPageNo*);  
 **System**.***out***.println(**"pageSize ---- > "** + *pageSize*);  
 **try** {  
 connection = **BaseDao**.**getConnection**();  
 userList = userDao.getUserList(connection, *queryUserName*,*queryUserRole*,*currentPageNo*,*pageSize*);  
 } **catch** (**Exception** *e*) {  
 // ***TODO Auto-generated catch block*** *e*.printStackTrace();  
 }**finally**{  
 **BaseDao**.**closeResource**(connection, **null**, **null**);  
 }  
 **return** userList;  
}

### 6.3、获取角色操作

Dao：

**package com.frans.dao.role**;  
  
**import com.frans.pojo.Role**;  
  
**import java.sql.Connection**;  
**import java.util.List**;  
  
**public interface RoleDao** {  
   
 **public List**<**Role**> getRoleList(**Connection** *connection*)**throws Exception**;  
  
}

**package com.frans.dao.role**;  
  
**import com.frans.dao.BaseDao**;  
**import com.frans.pojo.Role**;  
  
**import java.sql.Connection**;  
**import java.sql.PreparedStatement**;  
**import java.sql.ResultSet**;  
**import java.util.ArrayList**;  
**import java.util.List**;  
  
**public class RoleDaoImpl implements RoleDao**{  
  
 **@Override  
 public List**<**Role**> getRoleList(**Connection** *connection*) **throws Exception** {  
 **PreparedStatement** pstm = **null**;  
 **ResultSet** rs = **null**;  
 **List**<**Role**> roleList = **new** ArrayList<**Role**>();  
 **if**(*connection* != **null**){  
 **String** sql = **"select \* from smbms\_role"**;  
 **Object**[] params = {};  
 rs = **BaseDao**.**execute**(*connection*, pstm, rs, sql, params);  
 **while**(rs.next()){  
 **Role** \_role = **new** Role();  
 \_role.setId(rs.getInt(**"id"**));  
 \_role.setRoleCode(rs.getString(**"roleCode"**));  
 \_role.setRoleName(rs.getString(**"roleName"**));  
 roleList.add(\_role);  
 }  
 **BaseDao**.**closeResource**(**null**, pstm, rs);  
 }  
   
 **return** roleList;  
 }  
  
}

Service：

**package com.frans.service.role**;  
  
**import com.frans.pojo.Role**;  
  
**import java.util.List**;  
  
**public interface RoleService** {  
   
 **public List**<**Role**> getRoleList();  
   
}

**package com.frans.service.role**;  
  
**import com.frans.dao.BaseDao**;  
**import com.frans.dao.role.RoleDao**;  
**import com.frans.dao.role.RoleDaoImpl**;  
**import com.frans.pojo.Role**;  
  
**import java.sql.Connection**;  
**import java.util.List**;  
  
**public class RoleServiceImpl implements RoleService**{  
   
 **private RoleDao** roleDao;  
   
 **public RoleServiceImpl**(){  
 roleDao = **new** RoleDaoImpl();  
 }  
   
 **@Override  
 public List**<**Role**> getRoleList() {  
 **Connection** connection = **null**;  
 **List**<**Role**> roleList = **null**;  
 **try** {  
 connection = **BaseDao**.**getConnection**();  
 roleList = roleDao.getRoleList(connection);  
 } **catch** (**Exception** *e*) {  
 *e*.printStackTrace();  
 }**finally**{  
 **BaseDao**.**closeResource**(connection, **null**, **null**);  
 }  
 **return** roleList;  
 }  
   
}

### 6.4、用户显示

1. 获取用户前端的数据

2. 判断请求是否需要执行，看参数的值判断

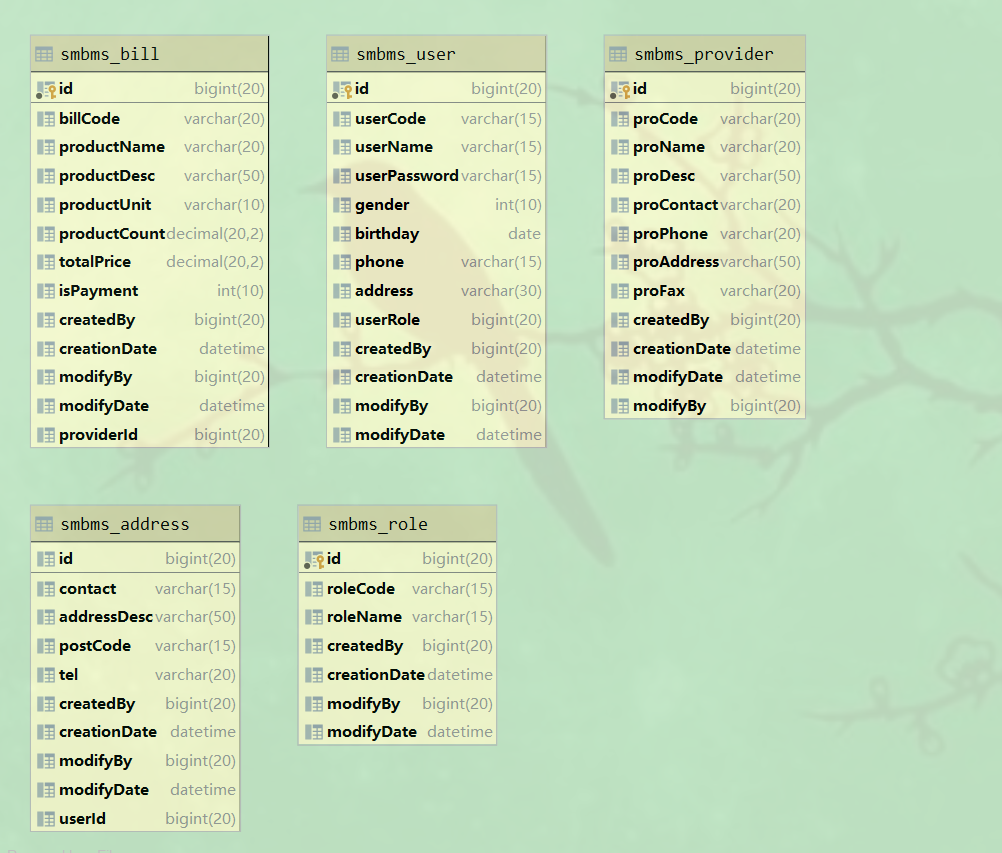
3. 实现分页

4. 用户列表展示

5. 返回前端,使用EL表达式获取数据。

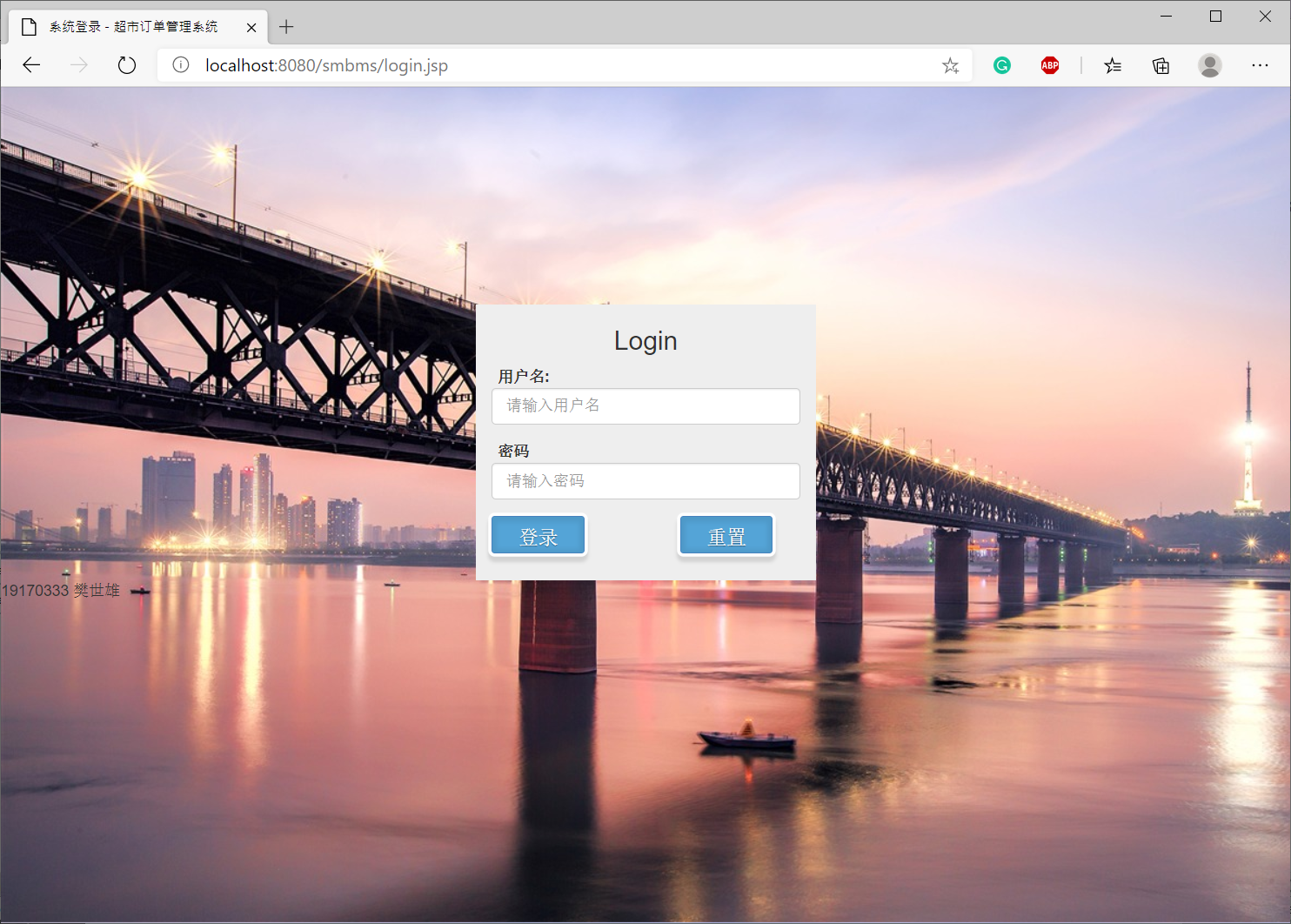
## 数据库：

数据库图：

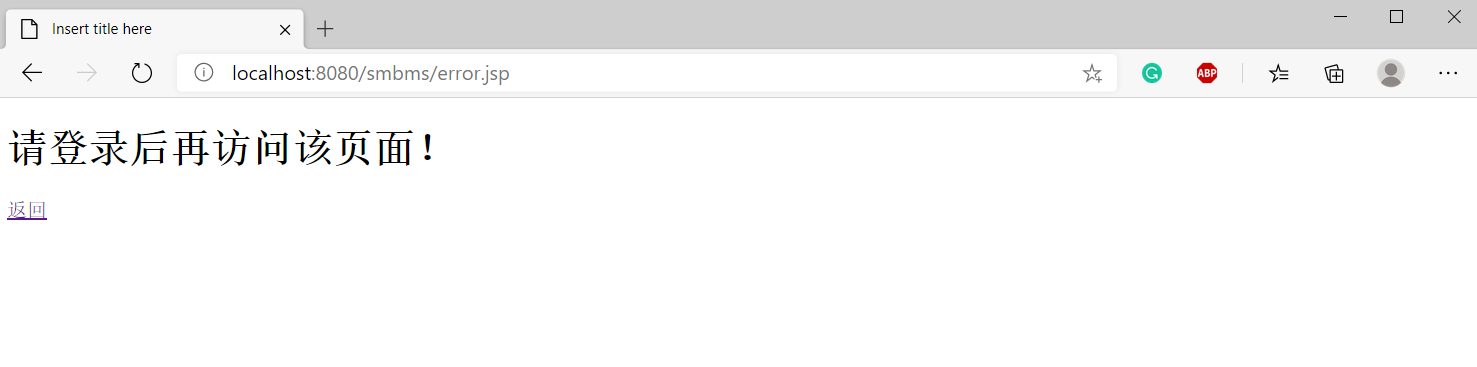


# 四、项目运行结果展示

## 1、登录页面



## 2、未登录却访问主页面，遭到拦截：



## 3、用户管理



查询会按照模糊查询：

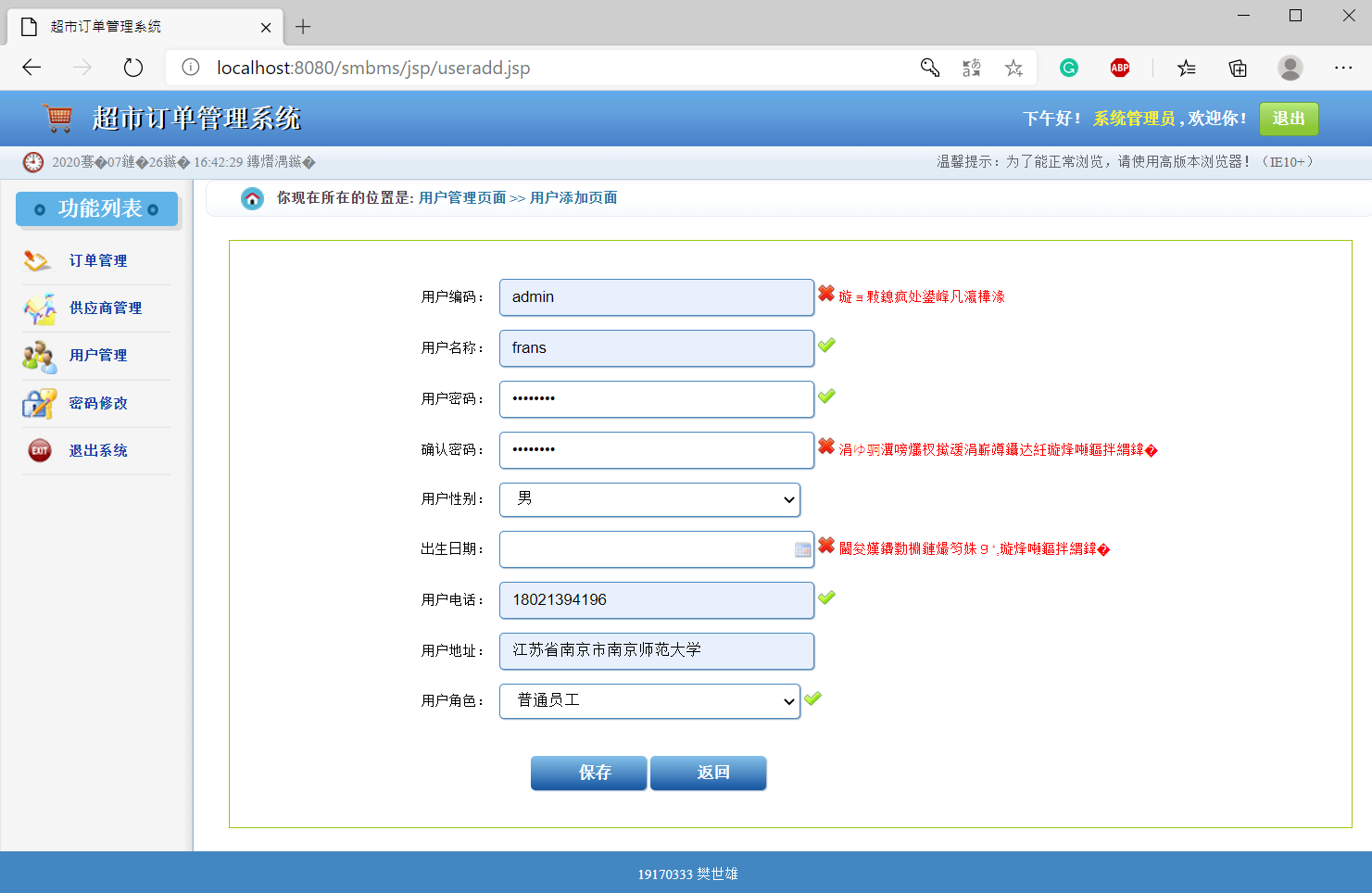
1. 查询名字中有“孙”的用户：

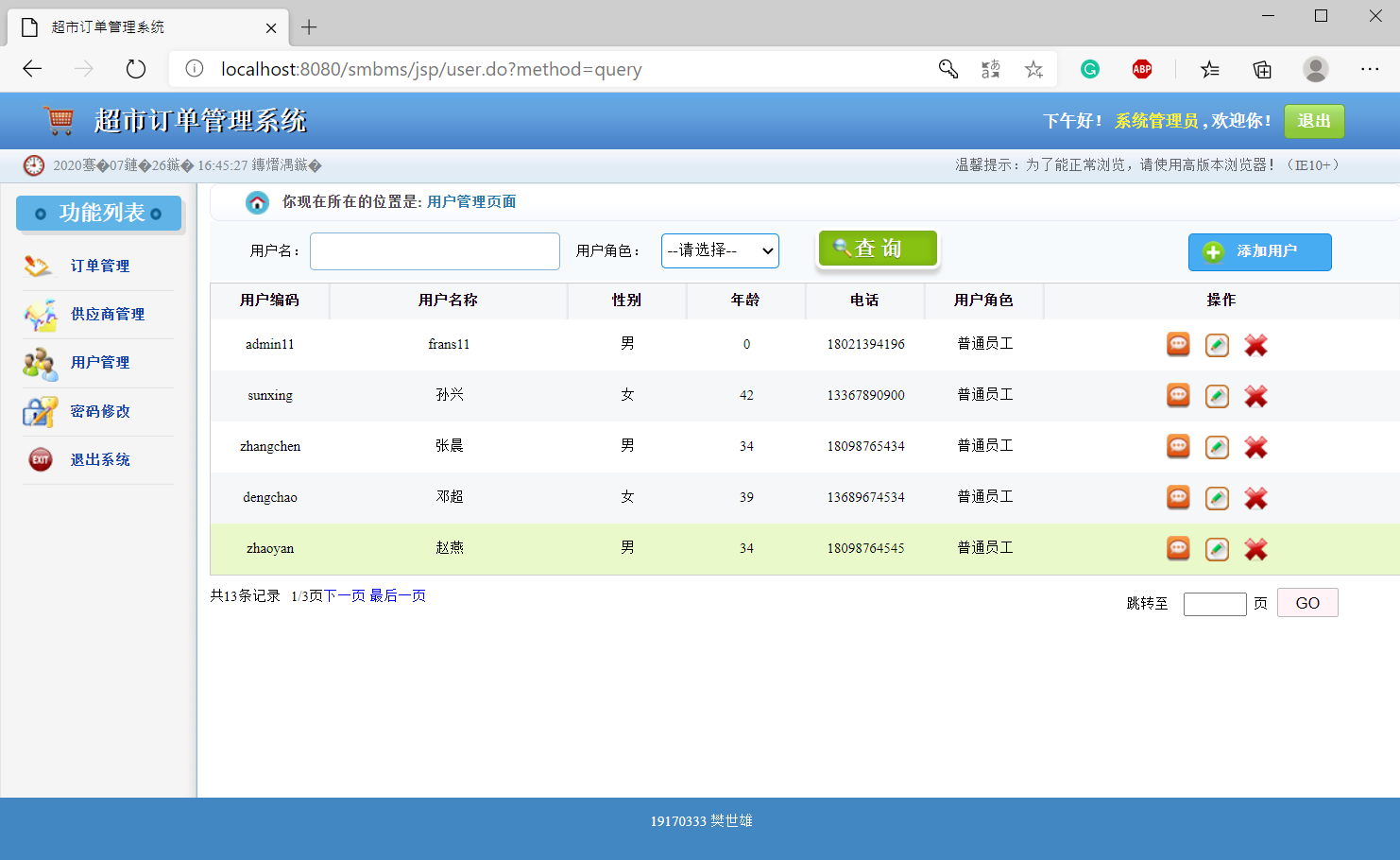


2、查询普通员工：



3、增加用户：（乱码问题仍未解决），通过ajax判断用户编码是否在数据库中存在，若存在，则提示用户，密码必须大于7位，若没有填写完成，也会提示用户。出生日期是通过日历方式由用户自己选择。

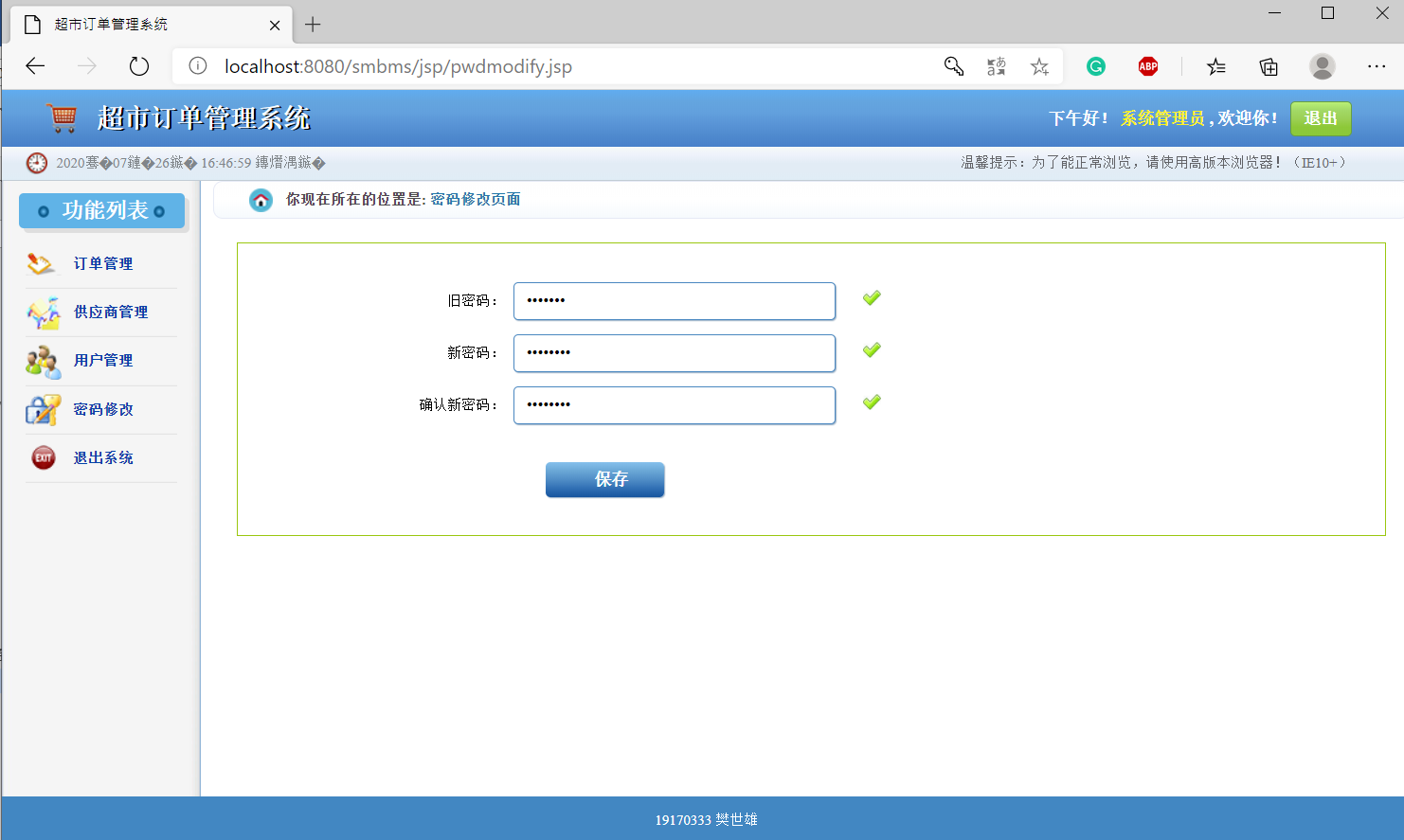


添加成功：  


4、修改：



## 4、密码修改



# 遇到的问题以及解决方案

## Junit单元测试

例如：

@Test  
**public void** getli() **throws Exception** {  
 **List**<**User**> users = getUserList(**"李"**, **0**, **1**, **5**);  
 **for** (**User** user : users) {  
 **System**.***out***.println(user);  
 }  
}

通过编写测试，可以对数据库进行查询，以便于查找问题，并且检测除了servlet之外的任何地方的数据传输问题。极大地缩短了调试BUG的时间和成本。由此可以发现servlet的一些缺点。

## 1、数据库连接问题：

Mysql8.0在jdbc连接上会出现问题

(1) jdbc驱动：Mysql5.4等8.0之前的版本：com.mysql.jdbc.Driver

8.0之后的版本：com.mysql.cj.jdbc.Driver

(2) url路径问题：8.0之后的版本要在末尾加上时区信息以及显示SSL连接。

(3) jar包问题：在jar配置中应替换Mysql所对应的jar包

## 2、tomcat问题：

同样，应放入对应tomcat的jar包

## 3、maven部署

在不同的电脑上运行maven install出现失败问题：

(1) 在对应pom.xml文件中加入需要的jdk和maven-compiler-plugin

(2) 修改maven所在位置的settings.xml文件中jdk和maven-compiler-plugin

重新构造maven，再次运行server。

(3) maven在新建项目会默认jdk1.5，需要改pom文件：

<**properties**>   
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**maven.compiler.source**>1.11</**maven.compiler.source**>  
 <**maven.compiler.target**>1.11</**maven.compiler.target**>  
</**properties**>

使JDK版本和自己配置的版本一致。

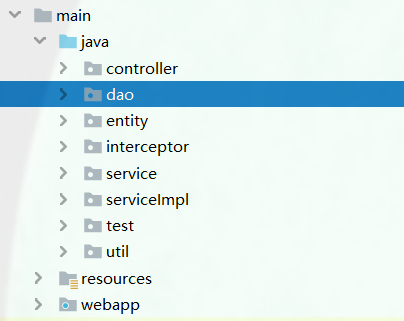
## 4、乱码问题

js代码中文乱码，至今未解决。

# 六、SSM框架

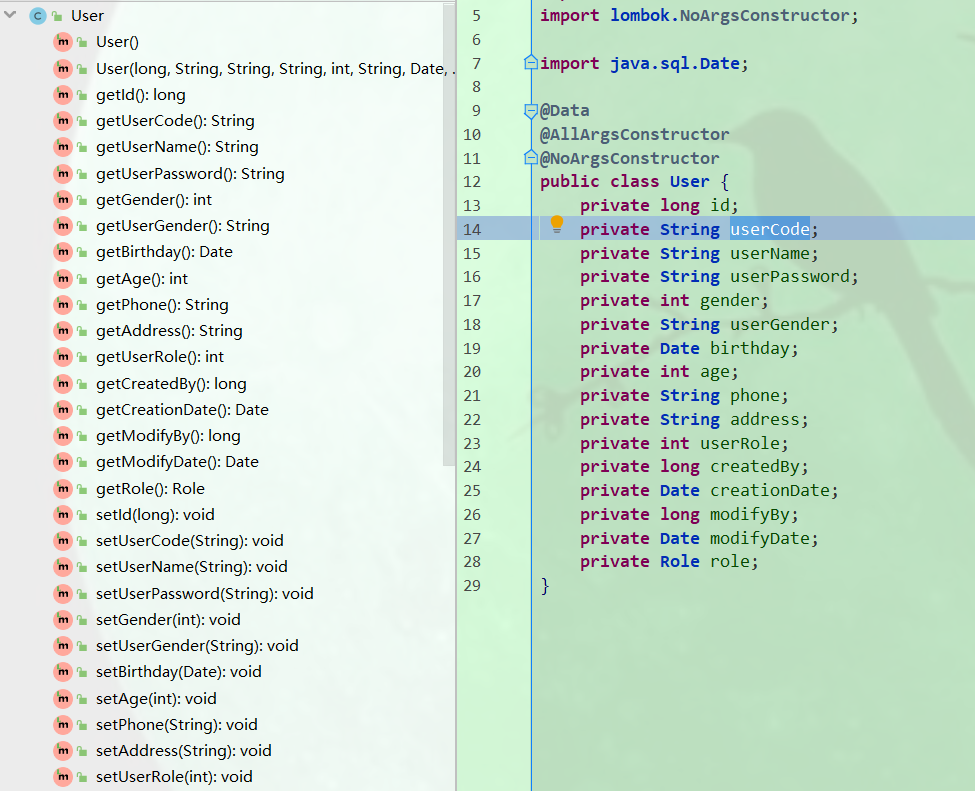
课余时间，我学习了SSM框架。将本项目后端用SSM框架重写。大大减少了代码量，体会到了框架的好处。

总的代码目录：



在建立实体类中使用了lombok插件也大大减少了代码量。

使用lombok后structure，已经包含了所有的getter/setter方法和toString方法：



## 配置文件

Log4j：

**log4j.rootLogger**=**info,con  
log4j.appender.con**=**org.apache.log4j.ConsoleAppender  
log4j.appender.con.layout**=**org.apache.log4j.PatternLayout  
log4j.appender.con.layout.ConversionPattern**=**%d{MM-dd HH:mm:ss}[%p]%c%n -%m%n**

mybatis配置文件：

*<?xml version***="1.0"** *encoding***="UTF-8"***?>***<!DOCTYPE** *configuration* **PUBLIC "-//mybatis.org//DTD Config 3.0//EN"   
"http://mybatis.org/dtd/mybatis-3-config.dtd"*>***<**configuration**>  
  
 <!--设置运行时的行为-->  
 <**settings**>  
 <**setting** *name***="logImpl"** *value***="LOG4J"**/>  
 <!--设置为FULL，自动匹配所有，则在resultMap中不进行匹配的字段也可以映射-->  
 <**setting** *name***="autoMappingBehavior"** *value***="FULL"**/>  
 </**settings**>  
  
 <!--指向包含实体类的包-->  
 <**typeAliases**>  
 <**package** *name***="entity"**/>  
 </**typeAliases**>  
</**configuration**>

Spring-mvc配置文件：

*<?xml version***="1.0"** *encoding***="UTF-8"***?>*<**beans** *xmlns***="http://www.springframework.org/schema/beans"** *xmlns:xsi***="http://www.w3.org/2001/XMLSchema-instance"** *xmlns:context***="http://www.springframework.org/schema/context"** *xmlns:mvc***="http://www.springframework.org/schema/mvc"** *xsi:schemaLocation***="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd  
 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.1.xsd"**>  
  
 <!-- 开启注解 -->  
 <**mvc:annotation-driven** />  
  
 <!-- 使用扫描机制自动扫描控制器类 -->  
 <**context:component-scan** *base-package***="controller"**/>  
  
 <!-- 对静态资源文件的访问 -->  
 <**mvc:default-servlet-handler** />  
 <**mvc:resources** *location***="/static/calendar/"** *mapping***="/calendar/\*\*"**/>  
 <**mvc:resources** *location***="/static/images/"** *mapping***="/images/\*\*"**/>  
 <**mvc:resources** *location***="/static/js/"** *mapping***="/js/\*\*"**/>  
 <**mvc:resources** *location***="/static/css/"** *mapping***="/css/\*\*"**/>  
  
 <!-- 视图解释器 -->  
 <**bean** *class***="org.springframework.web.servlet.view.InternalResourceViewResolver"** *id***="internalResourceViewResolver"**>  
 <!-- 前缀 -->  
 <**property** *name***="prefix"** *value***="/WEB-INF/view/"** />  
 <!-- 后缀 -->  
 <**property** *name***="suffix"** *value***=".jsp"** />  
 </**bean**>  
  
 <!--alibaba解析json-->  
 <**mvc:annotation-driven**>  
 <**mvc:message-converters** *register-defaults***="false"**>  
 <**bean** *class***="org.springframework.http.converter.StringHttpMessageConverter"**/>  
 <**bean** *class***="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  
 <**property** *name***="supportedMediaTypes"**>  
 <**list**>  
 <**value**> text/html; charset=UTF-8</**value**>  
 <**value**> application/json; charset=UTF-8</**value**>  
 </**list**>  
 </**property**>  
 </**bean**>  
 </**mvc:message-converters**>  
 </**mvc:annotation-driven**>  
  
  
 <!-- 拦截器 -->  
 <**mvc:interceptors**>  
 <**mvc:interceptor**>  
 <!-- 拦截所有的请求，这个必须写在前面，也就是写在【不拦截】的上面 -->  
 <**mvc:mapping** *path***="/\*\*"** />  
 <**bean** *class***="interceptor.LoginInterceptor"**></**bean**>  
 </**mvc:interceptor**>  
 </**mvc:interceptors**>  
  
 <**mvc:default-servlet-handler**/>  
</**beans**>

Spring配置文件：

*<?xml version***="1.0"** *encoding***="UTF-8"***?>*<**beans** *xmlns***="http://www.springframework.org/schema/beans"** *xmlns:xsi***="http://www.w3.org/2001/XMLSchema-instance"** *xmlns:context***="http://www.springframework.org/schema/context"** *xmlns:tx***="http://www.springframework.org/schema/tx"** *xmlns:aop***="http://www.springframework.org/schema/aop"** *xsi:schemaLocation***="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd"**>  
  
  
  
 <**context:annotation-config**/>  
  
 <**context:property-placeholder** *location***="classpath:db.properties"**/>  
  
 <**context:component-scan** *base-package***="serviceImpl"**/>  
  
 <**bean** *id***="dataSource"** *class***="org.apache.commons.dbcp2.BasicDataSource"**>  
 <**property** *name***="driverClassName"** *value***="${jdbc.driver}"**/>  
 <**property** *name***="url"** *value***="${jdbc.url}"**/>  
 <**property** *name***="username"** *value***="${jdbc.username}"**/>  
 <**property** *name***="password"** *value***="${jdbc.password}"**/>  
 </**bean**>  
  
  
 <**bean** *id***="sqlSessionFactory"** *class***="org.mybatis.spring.SqlSessionFactoryBean"**>  
 <**property** *name***="dataSource"** *ref***="dataSource"**/>  
 <**property** *name***="mapperLocations"** *value***="classpath:MybatisMapper/\*.xml"**/>  
 <**property** *name***="configLocation"** *value***="classpath:mybatis.cfg.xml"**/>  
 </**bean**>  
  
 <!--使用 MapperScannerConfigurer 扫描指定的接口，并自动生成 MapperScannerBean-->  
 <**bean** *class***="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
 <**property** *name***="basePackage"** *value***="dao"**/>  
 </**bean**>  
  
 <!--定义事务管理器-->  
 <**bean** *id***="txManager"** *class***="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property** *name***="dataSource"** *ref***="dataSource"**/>  
 </**bean**>  
  
 <!--配置事务传播特性-->  
 <**tx:advice** *id***="txAdvice"** *transaction-manager***="txManager"**>  
 <**tx:attributes**>  
 <**tx:method** *name***="find\*"** *propagation***="SUPPORTS"**/>  
 <**tx:method** *name***="add\*"**/>  
 <**tx:method** *name***="upd\*"**/>  
 <**tx:method** *name***="del\*"**/>  
 <**tx:method** *name***="\*"**/>  
 </**tx:attributes**>  
 </**tx:advice**>  
  
 <!--配置面向切面-->  
 <**aop:config**>  
 <**aop:pointcut** *id***="Transactional"** *expression***="execution(\* service.\*.\*(..))"**/>  
 <**aop:advisor** *advice-ref***="txAdvice"** *pointcut-ref***="Transactional"**/>  
 </**aop:config**>  
  
 <!--把事务管理器交给spring容器-->  
 <**tx:annotation-driven** *transaction-manager***="txManager"** *proxy-target-class***="true"**/>  
</**beans**>

分页工具类PageBean：

/\*\*  
 \* **@author** Mr.Tong  
 \* 分页显示工具类  
 \* **@param <T>** 传入的泛型集合  
 \*/  
**@Data  
public class PageBean**<*T*> {  
  
 //当前页第几页，从请求那边传过来  
 **private int** pageNum;  
  
 //每页显示的数据条数  
 **private int** pageSize;  
  
 //总的记录条数  
 **private int** totalRecord;  
  
 //总页数  
 **private int** totalPage;  
  
 //limit开始索引  
 **private int** startIndex;  
  
 //分页显示的数据存放在list集合中  
  
 **private** *T* object;  
  
 **private List**<*T*> list;  
  
 **private int** start;  
  
 **private int** end;  
  
 **public PageBean**(**int** *pageNum*,**int** *pageSize*,**int** *totalRecord*) {  
 **this**.pageNum = *pageNum*;  
 **this**.pageSize = *pageSize*;  
 **this**.totalRecord = *totalRecord*;  
  
 //计算总页数  
 **if**(*totalRecord*%*pageSize* == **0**) {  
 **this**.totalPage = *totalRecord* / *pageSize*;  
 }**else** {  
 **this**.totalPage = *totalRecord* / *pageSize* + **1**;  
 }  
  
 //开始索引  
 **this**.startIndex = (*pageNum*-**1**) \* *pageSize*;  
 **this**.start = **1**;  
 **this**.end = **5**;  
  
 //显示页码的算法  
 //显示5页  
 **if**(totalPage <= **5**) {  
 **this**.end = **this**.totalPage;  
 }**else** {  
 **this**.start = *pageNum* - **2**;  
 **this**.end = *pageNum* + **2**;  
 **System**.***out***.println(start);  
 **if**(start < **0**) {  
 **this**.start = **1**;  
 **this**.end = **5**;  
 }  
 **if**(end > **this**.totalPage) {  
 **this**.end = totalPage;  
 **this**.start = end-**5**;  
 }  
 }  
 }  
}

## 拦截器

对未登录的用户进行拦截：

**package interceptor**;  
  
  
**import org.springframework.web.servlet.HandlerInterceptor**;  
  
**import javax.servlet.ServletException**;  
**import javax.servlet.http.HttpServletRequest**;  
**import javax.servlet.http.HttpServletResponse**;  
**import java.io.IOException**;  
  
**public class LoginInterceptor implements HandlerInterceptor** {  
  
 **@Override  
 public boolean** preHandle(**HttpServletRequest** *request*,**HttpServletResponse** *response*,**Object** *handler*) **throws IOException**, **ServletException** {  
 *request*.setCharacterEncoding(**"utf-8"**);  
 *response*.setCharacterEncoding(**"utf-8"**);  
 **Object** obj = (**Object**) *request*.getSession().getAttribute(**"userSession"**);  
  
 **String** requestURI = *request*.getRequestURI();  
 **System**.***err***.println(**"===========>>"** + requestURI);  
 **if** (requestURI.contains(**"login"**) || requestURI.equals(**"/"**)) {  
 **return true**;  
 }  
  
 **if** (obj != **null** && obj != **""**){  
 **return true**;  
 }  
 *response*.sendRedirect(**"../error.jsp"**);  
 **return false**;  
 }  
}

配置对应于spring-mvc中的拦截器配置

## 登录

Dao：

/\*\*根据 userCode 查询用户名数量  
 \* **@param userCode** 用户名  
 \* **@return** 根据 userCode 查询用户名数量  
 \*/  
**int** findByUserCodeCount(**@Param**(**"userCode"**) **String** *userCode*);

/\*\*  
 \* 根据 userCode 和 userPassword 查询用户编号  
 \* **@param userCode** 用户编码  
 \* **@param userPassword** 用户密码  
 \* **@return** 用户编号( 返回值：null，用户编码和用户密码匹配不到 )  
 \*/  
**Integer** getIdByuCodeAnduPwd(**@Param**(**"userCode"**) **String** *userCode*, **@Param**(**"userPassword"**) **String** *userPassword*);

UserMapper：

<**select** *id***="findByUserCodeCount"** *resultType***="int"**>  
 select ***count***(1) from **smbms\_user** where userCode = *#{userCode}*</**select**>

<**select** *id***="getIdByuCodeAnduPwd"** *resultType***="Integer"**>  
 select id from **smbms\_user** where userCode=*#{userCode}* and userPassword = *#{userPassword}*</**select**>

Service：

/\*\* 验证登陆  
 \* **@param userCode** 用户表  
 \* **@return** 验证登陆结果(i：用户编号，0：用户名密码不匹配，-1：用户名不存在)  
 \*/  
**int** checklogin(**String** *userCode*, **String** *userPassword*);

Impl：

**@Override  
public int** checklogin(**String** *userCode*,**String** *userPassword*) {  
 // ***TODO Auto-generated method stub* int** count = userDao.findByUserCodeCount(*userCode*);  
 **if**(count == **0**) {  
 **return** -**1**;  
 }  
 **Integer** id = userDao.getIdByuCodeAnduPwd(*userCode*,*userPassword*);  
 **if**(id == **null**) {  
 **return 0**;  
 }  
 **return** id;  
}

Controller：

//登录  
**@RequestMapping**(**"/login.do"**)  
**public String** checkLogin(**HttpServletRequest** *request*, **HttpServletResponse** *response*, **HttpSession** *session*, **User** user){  
 **int** result = userService.checklogin(*user*.getUserCode(),*user*.getUserPassword());  
 **if**(result == -**1**){  
 //此账号不存在  
 *session*.setAttribute(**"error"**,**"此账号不存在！"**);  
 **return "redirect:../login.jsp"**;  
 }**else if**(result == **0**){  
 //账号密码不匹配  
 *session*.setAttribute(**"error"**,**"账号与密码不匹配！"**);  
 **return "redirect:../login.jsp"**;  
 }**else**{  
 //登陆成功  
 user = userService.getUserById(result);  
 *session*.setAttribute(**"userSession"**,user);  
 **return "frame"**;  
 }  
}

## 修改密码：

Dao：

/\*\*  
 \* 根据用户id和用户密码查询用户量  
 \* **@param id** 用户id  
 \* **@param oldpassword** 用户密码  
 \* **@return** 用户量  
 \*/  
**int** checkPwd(**@Param**(**"id"**) **long** *id*, **@Param**(**"userPassword"**) **String** *oldpassword*);

Mapper：

<**update** *id***="updatePwd"** >  
 update **smbms\_user** set userPassword = *#{newPwd}* where id = *#{id}*</**update**>

Service：

**String** checkPwd(**long** *id*, **String** *oldpassword*);

Impl：

**@Override  
public String** checkPwd(**long** *id*, **String** *oldpassword*) {  
 **int** result = userDao.checkPwd(*id*,*oldpassword*);  
 **return** result == **1**?**"true"**:**"false"**;  
}

Controller：

**@RequestMapping**(value = **"/pwdmodify.do"**,method = **RequestMethod**.***POST***)  
**public String** pwdModify(**HttpSession** *session*, **String** *oldpassword*, **String** *newpassword*, **String** *reNewPassword*){  
 **if** (!*newpassword*.equals(*reNewPassword*)){  
 *session*.setAttribute(**"message"**,**"两次输入密码不一致！"**);  
 **return "pwdmodify"**;  
 }  
 **User** user = (**User**)*session*.getAttribute(**"userSession"**);  
 **int** result = userService.pwdModify(user.getId(),*oldpassword*,*newpassword*);  
 **if** (result == -**1**){  
 *session*.setAttribute(**"message"**,**"用户密码输入错误！"**);  
 **return "pwdmodify"**;  
 }**else if**(result == **1**){  
 *session*.setAttribute(**"message"**,**"密码修改成功！"**);  
 **return "pwdmodify"**;  
 }**else**{  
 *session*.setAttribute(**"message"**,**"密码修改失败，请联系管理员！"**);  
 **return "pwdmodify"**;  
 }  
}

## 用户视图

Dao：

/\*\*  
 \* 根据用户编码和用户角色编号 分页查询  
 \* **@param pageBean** 分页  
 \* **@param queryname** 用户编码  
 \* **@param userRoleId** 用户角色编号  
 \* **@return** 用户集合  
 \*/  
**List**<**User**> getUser(**@Param**(**"pageBean"**) **PageBean**<**User**> *pageBean*, **@Param**(**"userCode"**) **String** *queryname*, **@Param**(**"userRoleId"**) **int** *userRoleId*);

Mapper： 用limit实现分页。

<**select** *id***="getUser"** *resultMap***="findByUserAllMap"**>  
 select u.\*,roleName,floor(DATEDIFF(now(),birthday)/365) as age,  
 CASE gender  
 WHEN '1' THEN '男'  
 WHEN '2' THEN '女'  
 ELSE 0  
 END AS userGender  
 from **smbms\_user** as u  
 INNER JOIN **smbms\_role** as r on userRole = r.id  
 where 1 = 1  
 <**if** *test***="userCode != ''"**>  
 and userCode = *#{userCode}* </**if**>  
 <**if** *test***="userRoleId != 0"**>  
 and userRole = *#{userRoleId}* </**if**>  
 limit *#{pageBean.startIndex}*,*#{pageBean.pageSize}*</**select**>

结果集映射：

<**resultMap** *id***="findByUserAllMap"** *type***="entity.User"**>  
 <**id** *property***="id"** *column***="id"**/>  
 <**result** *property***="userCode"** *column***="userCode"**/>  
 <**result** *property***="userName"** *column***="userName"**/>  
 <**result** *property***="userPassword"** *column***="userPassword"**/>  
 <**result** *property***="gender"** *column***="gender"**/>  
 <**result** *property***="birthday"** *column***="birthday"**/>  
 <**result** *property***="phone"** *column***="phone"**/>  
 <**result** *property***="address"** *column***="address"**/>  
 <**result** *property***="userRole"** *column***="userRole"**/>  
 <**result** *property***="createdBy"** *column***="createdBy"**/>  
 <**result** *property***="creationDate"** *column***="creationDate"**/>  
 <**result** *property***="modifyBy"** *column***="modifyBy"**/>  
 <**result** *property***="modifyDate"** *column***="modifyDate"**/>  
 <**association** *property***="role"** *javaType***="entity.Role"**>  
 <**id** *property***="id"** *column***="id"**/>  
 <**result** *property***="roleCode"** *column***="roleCode"**/>  
 <**result** *property***="roleName"** *column***="roleName"**/>  
 <**result** *property***="createdBy"** *column***="createdBy"**/>  
 <**result** *property***="creationDate"** *column***="creationDate"**/>  
 <**result** *property***="modifyBy"** *column***="modifyBy"**/>  
 <**result** *property***="modifyDate"** *column***="modifyDate"**/>  
 </**association**>  
</**resultMap**>

Service：

**PageBean**<**User**> getPageBeanUser(**PageBean**<**User**> *pageBean*, **String** *queryname*, **int** *userRoleId*);

Impl：

**public PageBean**<**User**> getPageBeanUser(**PageBean**<**User**> *pageBean*, **String** *queryname*, **int** *userRoleId*) {  
 *pageBean*.setList(userDao.getUser(*pageBean*,*queryname*,*userRoleId*));  
 **return** *pageBean*;  
}

Controller：

//展示用户视图  
**@RequestMapping**(**"/management.do"**)  
**public String** management(**HttpSession** *session*, **@RequestParam**(value = **"pageIndex"**,required = **false**)**String** *pageIndex*, **@RequestParam**(value = **"queryname"**,required = **false**)**String** *queryname*, **@RequestParam**(value = **"queryUserRole"**,required = **false**) **String** *queryUserRole*){  
 **PageBean** pageBean;  
 //表示一页展示9条数据  
 **int** pageSize = **9**;  
 **int** pageNum = *pageIndex*==**null**?**1**:**Integer**.**valueOf**(*pageIndex*);  
 **String** key = *queryname* == **null** ? **""** : *queryname*;  
  
 **int** userRoleId = *queryUserRole*==**null**?**0**:**Integer**.**valueOf**(*queryUserRole*);  
  
 **int** totalRecord = userService.getUserCount(key,userRoleId);  
  
 pageBean = **new** PageBean(pageNum,pageSize,totalRecord);  
 pageBean = userService.getPageBeanUser(pageBean,key,userRoleId);  
 *session*.setAttribute(**"userList"**,pageBean.getList());  
 *session*.setAttribute(**"pageBean"**,pageBean);  
 *session*.setAttribute(**"queryUserName"**,key);  
 *session*.setAttribute(**"queryUserRole"**,userRoleId);  
 **return "userlist"**;  
}

# 七、收获与体会

## servlet版本优缺点

servlet+jsp作为早期开发，其架构简单，比较适合小型项目开发；但是代码量大而且重复多。web.xml配置量太多，不利于团队开发，在写每一个servlet在web.xml都要做相应的配置；并且servlet具有容器依赖性，不利于单元测试。

在SSM框架中，编写如下test方法即可进行单元测试：

**@RunWith**(**SpringJUnit4ClassRunner**.**class**)  
**@ContextConfiguration**(locations = {**"classpath:spring.cfg.xml"**})  
**public class JunitTest** {  
  
 **@Test  
 public void** Test(){  
  
 }  
}

## 2、SSM

### 2.1、SSM项目各层作用

* Controller层：Controller是前后端对接的地方，接收前端请求和后端数据。转换格式、调用Service进行处理后，以Map<String, Object>格式返回。
* Mapper层：是SSM与数据库直接交互的地方，用sql语句来操作数据库。
* Service层：服务层由两部分构成，一部分是服务接口，另一部分则是服务的具体实现。

### 2.2、使用SSM框架的好处

相对于传统的JavaBean+JSP+Servlet模式而言，我认为它的优点有：

* 不需要创建那么多的Servlet，看起来很清爽。
* 不需要使用数据库JDBC操作，省略了大量的代码。
* 还可以提高程序的规范性和代码的重用性。

## 3、收获

学习了框架之后，我们编写代码的效率明显提高了。框架中封装了普通项目中程序员需要重复书写的代码和简化了调用过程。就比如说在传统的jsp项目中，我们的controller接收到前端请求后，程序就需要去开发Dao层，里面还涉及数据库的连接和存储过程的代码，大部分都是冗余的代码，而SSM框架极大地简化了controller以下层的开发，只需要一个service层和mapper层就行了，mapper层用来连接mapper.xml文件，直接用mapper.xml做sql语句的开发就行了，数据库的连接和存储过程都由mybatis负责，我们只需要负责传递形参和接收返回数据就好，这样就完成了一次完整的数据库交互，大大减少了代码量。

遗憾的是，spring框架的aop原理动态代理至今仍未理解，今后会多多研究项目理解。

注：本项目部分前端页面参考项目，后端手打。