****

**课 程 设 计**

**题 目:**  基于B/S模式网络用户管理

**教 学 院： 数理学院**

**专业 名称： 信息与计算科学**

**姓名与学号： 赵钰 201641110201**

**邓伟 201641110212**

**付杰 201641110218**

**邓礼追 201641110216**

**叶帅 201641110229**

**王文婕 201641110225**

**王顶 201641110214**

**指导 教师： 许四平**

2019年11月10日

**摘要**

本文介绍了基于B/S模式的网络用户管理系统的设计，以满足网络用户管理需求的应用系统。

**关键词：**B/S模式 网络用户管理

# **第一章 绪论**

## 1.1研究背景与意义

### **1.1.1研究背景**

用户信息管理是网络用户安全保障至关重要的一部分 。在用户信息日常管理中，需要经常修改学生信息，如添加或修改学生课程信息、考试信息、成绩信息等，反复操作对于老师和管理员而言是一项重要但令人头疼的工作。而且，传统以手工操作为主的学生信息管理模式，存在效率低、维护性差、保密性差等缺点，随着学生人数的增加，大量学生信息数据的管理、更新、查询和维护的难度也随之增加。职业院校的学生信息管理需要投入大量人力、物力成本，给院校发展带来一一 定压力。

虽然，随着高校校园网络、数字化校园等信息化建设的加速推进，许多高校已经建立综合化信息管理平台，实现依托校园网的办公、教务、招生、就业等综合信息管理，学生信息管理信息化解决了传统手工管理模式存在的缺陷与不足。但是，仍有许多职业院校目前尚未建立校园信息化综合平台，对学生信息管理系统的需求非常迫切

本课题拟研发针对该校的学生信息管理系统，从而解决学生信息管理难题。

### **1.1.2研究意义**

由于招生规模的扩大和学生人数的不断累加，学生的信息管理量剧增，依赖传统的手工管理模式显得繁冗复杂，且效率低下，错误率高，不便于汇总和分析，缺乏系统性，为学生工作的开展带来了不便。

开发一套基于网络的学生信息管理系统，提升学生管理水平和办公效率，真正实现信息的互通、互享，从而提升整体教育教学质量，就显得尤为重要，通过对本课题的研究，创建一一个快速、 便捷的信息管理平台，学生从入学开始到毕业离校所有信息都有系统的整合，更具规范性。管理系统的开发也更加便于远程查询信息，在一定程度上降低了信息管理难度，节省了管理支出，减轻了教师负担，也为学校其他各项工作提供了很好的信息支持，为推进学校信息化、数字化建设迈出了坚实一步。

# **第二章 关键技术介绍**

## 2.1什么是B/S模式？

B/S结构(Browser/Server，[浏览器](https://baike.so.com/doc/2920715-3082096.html" \t "https://baike.so.com/doc/_blank)/[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "https://baike.so.com/doc/_blank)模式)，是WEB兴起后的一种网络结构模式，WEB浏览器是[客户端](https://baike.so.com/doc/4889711-5107810.html" \t "https://baike.so.com/doc/_blank)最主要的[应用软件](https://baike.so.com/doc/3175127-3346154.html" \t "https://baike.so.com/doc/_blank)。这种模式统一了[客户端](https://baike.so.com/doc/4889711-5107810.html" \t "https://baike.so.com/doc/_blank)，将系统功能实现的核心部分集中到[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "https://baike.so.com/doc/_blank)上，简化了系统的开发、维护和使用。客户机上只要安装一个[浏览器](https://baike.so.com/doc/2920715-3082096.html" \t "https://baike.so.com/doc/_blank)，如Netscape Navigator或[Internet Explorer](https://baike.so.com/doc/3951966-4147130.html" \t "https://baike.so.com/doc/_blank)，[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "https://baike.so.com/doc/_blank)安装SQL Server、Oracle、MYSQL等数据库。[浏览器](https://baike.so.com/doc/2920715-3082096.html" \t "https://baike.so.com/doc/_blank)通过Web Server 同数据库进行数据交互。

## 2.2什么是Tomcat服务器？

Tomcat 服务器是一个免费的开放源代码的Web 应用服务器，属于轻量级应用[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "https://baike.so.com/doc/_blank)，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好Apache 服务器，可利用它响应[HTML](https://baike.so.com/doc/5869876-6082735.html" \t "https://baike.so.com/doc/_blank)(标准通用标记语言下的一个应用)页面的访问请求。实际上Tomcat是Apache 服务器的扩展，但运行时它是独立运行的，所以当你运行tomcat 时，它实际上作为一个与Apache 独立的进程单独运行的。

## 2.3什么是Servlet?

### **2.3.1 Servlet简介**

Servlet：运行在web服务器上的组件(java类)，用来处理请求，生成响应给客户端。充当了MVC中的Controller角色

　　Servlet是sun公司提供的一门用于开发动态web资源的技术。  
　　Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：  
　　1、编写一个Java类，实现servlet接口。  
　　2、把开发好的Java类部署到web服务器中。  
　　按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

### **2.3.2 Servlet的运行流程**

Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：  
　　①Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。  
　　②实例化：第一次访问Servlet时，web服务器创建该Servlet的一个实例对象并加载到内存中。   
　　③初始化：第一次访问时调用Servlet实例对象的init()方法，只初始化一次。  
　　④服务：创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。该方法会根据请求方法来决定调用doGet()或doPost()  
　　⑤销毁：WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

总结：生命周期方法的执行顺序及次数如下

构造方法(1 次)--->init 方法(1 次) --> service 方法(n 次,每次访问都执行) -->destory(1 次)

## 2.4 什么是jsp?

JSP全名为Java Server Pages，中文名叫java服务器页面，其根本是一个简化的[Servlet](https://baike.so.com/doc/2354665-2490021.html" \t "https://baike.so.com/doc/_blank)设计，它是由Sun Microsystems公司倡导、许多公司参与一起建立的一种[动态网页](https://baike.so.com/doc/5430740-5669025.html" \t "https://baike.so.com/doc/_blank)技术标准。JSP技术有点类似ASP技术，它是在传统的[网页](https://baike.so.com/doc/3683001-3870808.html" \t "https://baike.so.com/doc/_blank)HTML（标准通用标记语言的子集）文件(\*.htm,\*.html)中插入Java[程序段](https://baike.so.com/doc/7678101-7952196.html" \t "https://baike.so.com/doc/_blank)(Scriptlet)和JSP标记(tag)，从而形成JSP文件，后缀名为(\*.jsp)。 用JSP开发的Web应用是[跨平台](https://baike.so.com/doc/5131891-5361327.html" \t "https://baike.so.com/doc/_blank)的，既能在Linux下运行，也能在其他[操作系统](https://baike.so.com/doc/5354536-5590000.html" \t "https://baike.so.com/doc/_blank)上运行。

它实现了Html语法中的java扩张（以 <%, %>形式）。JSP与Servlet一样，是在服务器端执行的。通常返回给客户端的就是一个HTML文本，因此客户端只要有浏览器就能浏览。

## 2.5 怎么理解Servlet+jsp?

事实上，JSP是Servlet的一种特殊形式，每个JSP页面就是一个Servlet实例——JSP页面由系统编译成Servlet，Servlet再负责响应用户请求。JSP其实也是Servlet的一种简化，使用JSP时，其实还是使用Servlet，因为Web应用中的每个JSP页面都会由Servlet容器生成对应的Servlet。对于Tomcat而言，JSP页面生成的Servlet放在work路径对应的Web应用下。

  Servlet是在服务器端执行的Java程序，一个被称为Servlet容器的程序（其实就是服务器） 负责执行Java程序。而JSP(Java Server Page)则是一个页面， 由JSP容器负责执行。

Servlet和JSP两者最大的区别就是，Servlet以Java程序为主， 输出HTML代码时需要使用out.println函数，也就是说Java中内嵌HTML； 而JSP则以HTML页面为主，需要写Java代码时则在页面中直接插入Java代码， 即HTML中内嵌Java。

Java [Servlet](https://baike.so.com/doc/2354665-2490021.html" \t "https://baike.so.com/doc/_blank)是JSP的技术基础，而且大型的Web应用程序的开发需要Java Servlet和JSP配合才能完成。JSP具备了[Java](https://baike.so.com/doc/2886868-3046592.html" \t "https://baike.so.com/doc/_blank)技术的简单易用，完全的面向对象，具有平台无关性且安全可靠，主要面向因特网的所有特点。

## 2.6 什么是HTTP协议？

### **2.6.1 HTTP 简介**

WEB 浏览器与 WEB 服务器之间的一问一答的交互过程必须遵循一定的规则，这个规则就是 HTTP 协议。HTTP 是 HyperText Transfer Protocol（超文本传输协议）的简写，它是 TCP/IP 的一个应用层协议，用于定义 WEB 浏览器与 WEB 服务器之间交换数据的过程及数据本身的格式。HTTP 协议是 WEB 开发的基础。

### **2.6.2 HTTP 协议特点**

HTTP 协议是基于请求/响应模型的协议，请求和响应必须成对，先有请求后有响应。

HTTP 协议默认的端口是 80。例如：<http://www.bjsxt.com:80>

简单快速：当客户端向服务器端发送请求时，只是简单的填写请求路径和请求方法即可，然后就可以通过浏览器或其他方式将该请求发送就行了。比较常用的请求方法有三种，分别是：GET、HEAD、POST。不同的请求方法使得客户端和服务器端联系的方式各不相同。因为 HTTP 协议比较简单，所以 HTTP 服务器的程序规模相对比较小，从而使得通信的速度非常快。

灵活：Http 协议允许客户端和服务器端传输任意类型的数据对象。这些不同的类型由 Content-Type 标记。

无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，服务器不知道客户端是什么状态。

## **2.6.3 HTTP 请求构成**

Http 协议由 Http 请求和 Http 响应组成。客户端连上服务器后，向服务器请求某个 web 资源，称之为客户端向服务器发送了一个 HTTP 请求。一个完整的 HTTP 请求包括：请求行、请求头、请求体(请求正文)

1.请求行：位于请求消息的第一行,格式：请求方式 资源路径 HTTP 版本号

2.请求头：客户端发送给服务器的一些基本信息，使用键值对 key：value 表示。从第二行开始到 Request Headers 结束

3.请求体：当请求方式是 post 时，请求体中有请求的参数，每一项数据都用 key=value 表示，例如：username=zhangsan&userpwd=123456.

如果请求方式为 get，那么请求参数不会出现在请求体中，会拼接在 url 地址后面

http://localhost:8080...?username=zhangsan&userpwd=123

## **2.6.4 HTTP 响应构成**

一个完整的 HTTP 响应包括：响应行、响应头、响应体

1.响应行：位于响应消息的第一行,格式：HTTP 版本号 状态码 状态描述信息

例如：HTTP/1.1 200 OK

## 2.7 GET 与 POST 方法

### **2.7.1 请求方法 GET 与 POST 的区别**

（1） Get 方式将请求参数追加在 URL 后面，通过 URL 提交数据，数据在 URL 中可以看到；POST 方式，数据放置在 Form 内提交。

（2） GET 方式提交的数据最多只能有 1024 字节，而 POST 则没有此限制。

（3） 安全性问题。正如在（1）中提到，使用 Get 的时候，参数会显示在地址栏上，而 Post 不会。所以，如果这些数据不是中文数据而且是非敏感数据，那么使用 get；反之如果是中文数据而且包含敏感数据，那么还是使用 post 为好。

### **2.7.2 如何区分GET 请求和 POST 请求**

只有表单设置为 method=”post”才是 POST 请求，其它的都是 get 请求。常见的 GET

请求有：地址栏直接访问、超链接请求、<img src=”” />等

一般来说，尽量避免使用 Get 方式提交表单，因为有可能会导致安全问题。比如说在登录表单中用 Get 方式，用户输入的用户名和密码将在地址栏中暴露无遗。但是在查询程序中，用 Get 方式就比用 Post 好。

# **第三章 基于B/S模式的用户管理需求分析**

# **第四章 基于B/S模式的用户管理详细设计**

### **4.1项目名称**

网络用户管理

### **4.2使用模式和技术**

采用B/S模式和Servlet+jsp编程技术

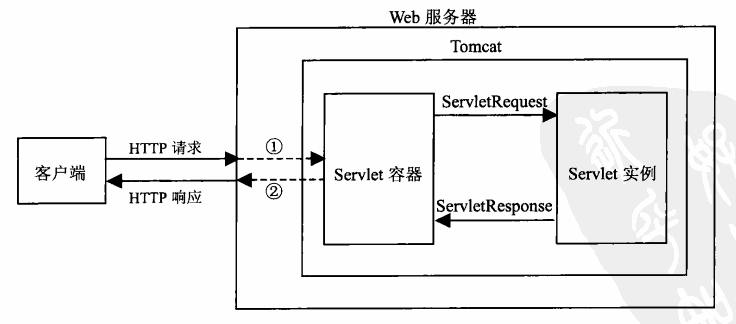
### **4.3使用工具**

MySql数据库、Navicat数据库管理工具、IntelliJ IDEA开发工具、Tomcat服务器、jdk1.8、Google Chrome

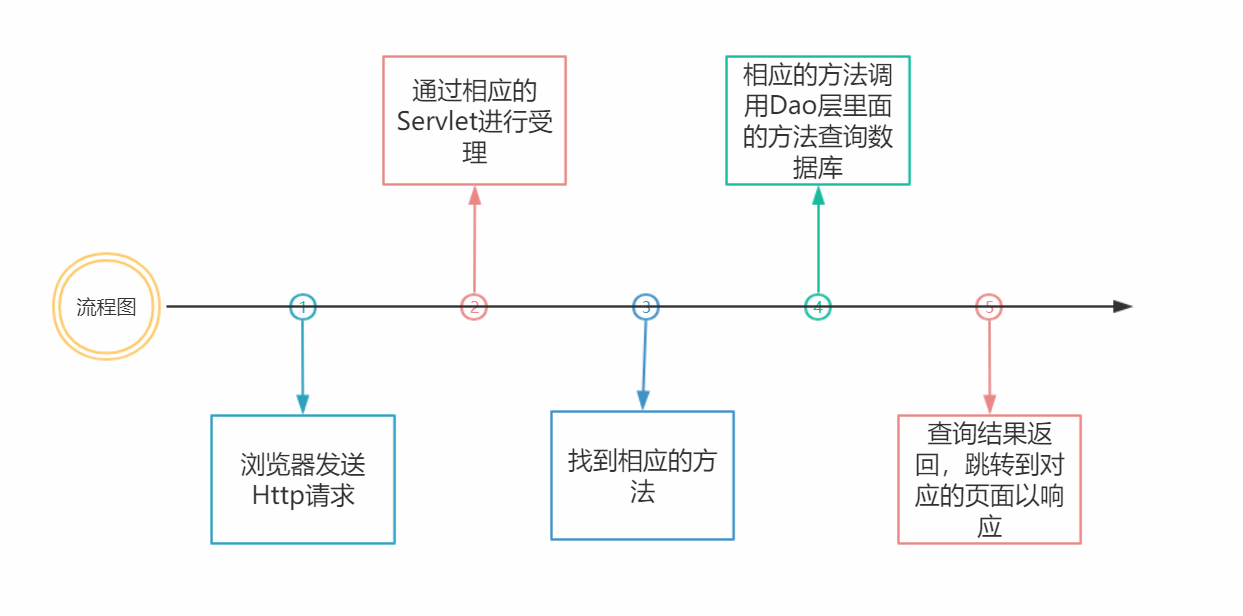
### **4.4运行底层原理：**

当web服务器找到相对应的servlet之后，就会使用反射机制创建实例，调用init( )方法将该实例装载到内存中，并且把接收到的HTTP请求封装成request对象，(同时还有一个response对象，只不过此对象中没有任何内容)作为service( )方法的参数传入。  然后执行 service( )函数，该方法会根据请求方法来决定调用doGet()或doPost()，进而获取到response对象（该对象中存储着各种信息），然后由web服务器把response对象的信息拆出，并形成HTTP响应格式返回给浏览器。

**运行原理图：**

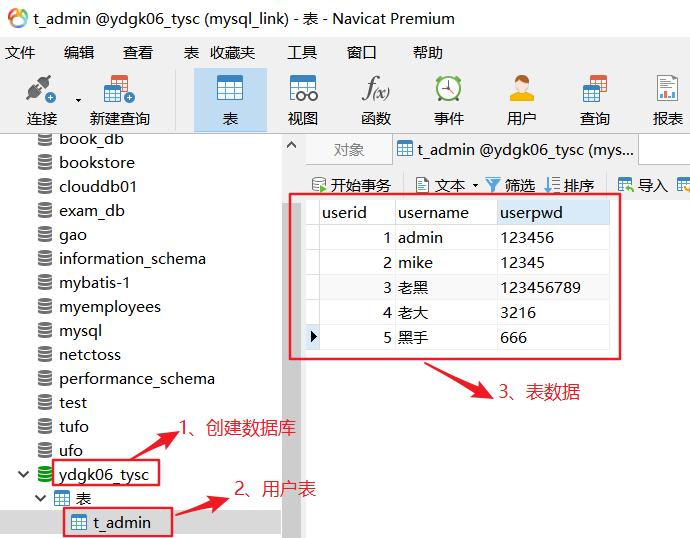


**运行流程图：**

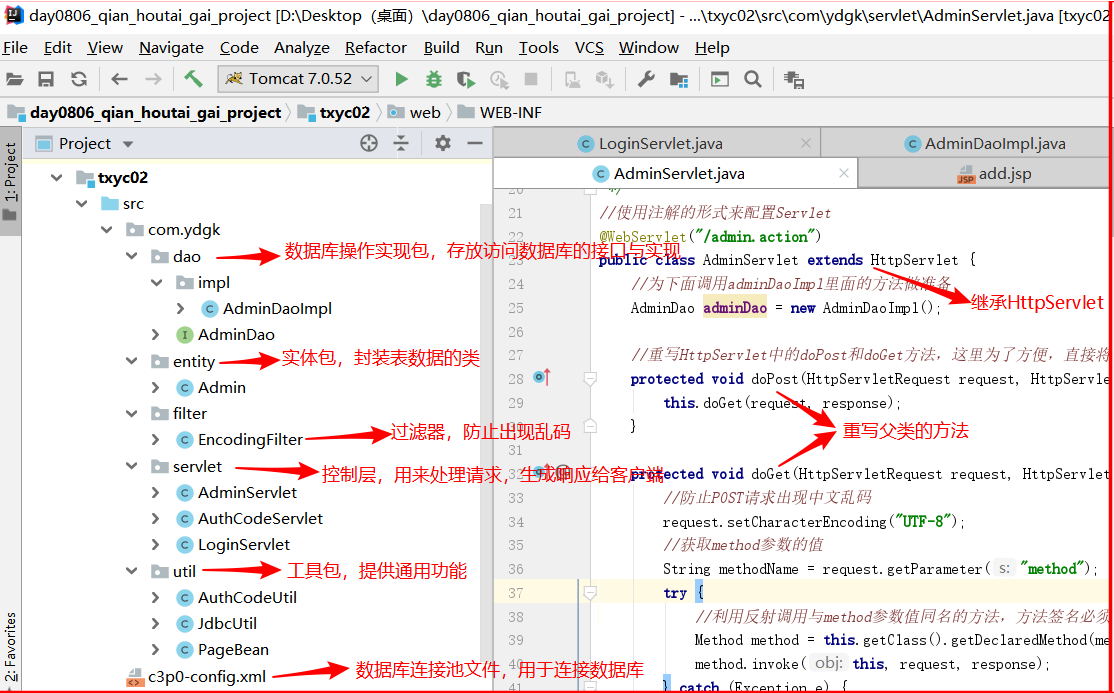


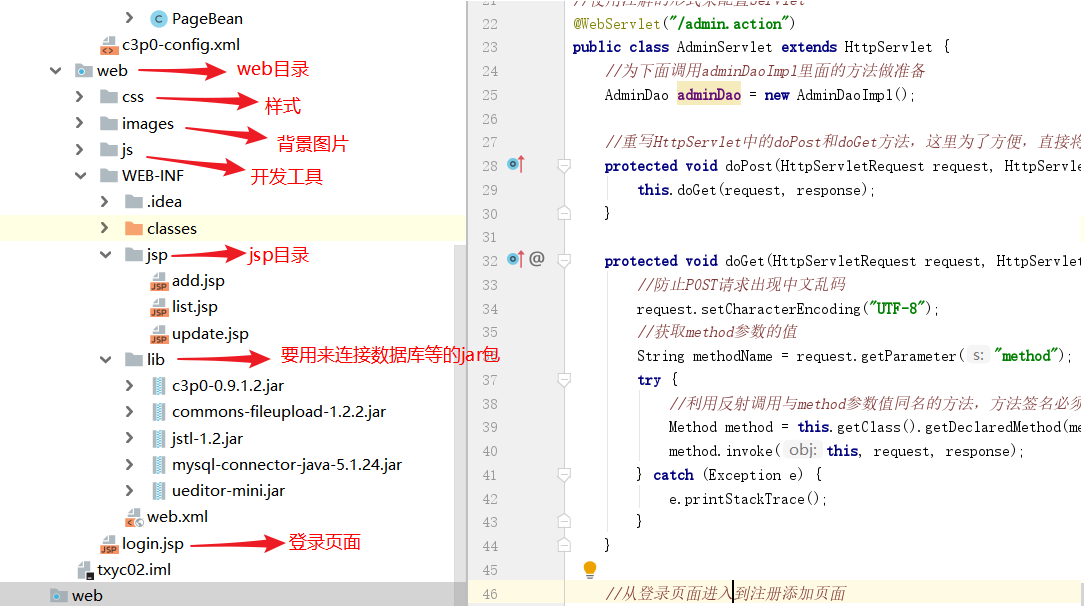
### **4.5项目建立步骤**

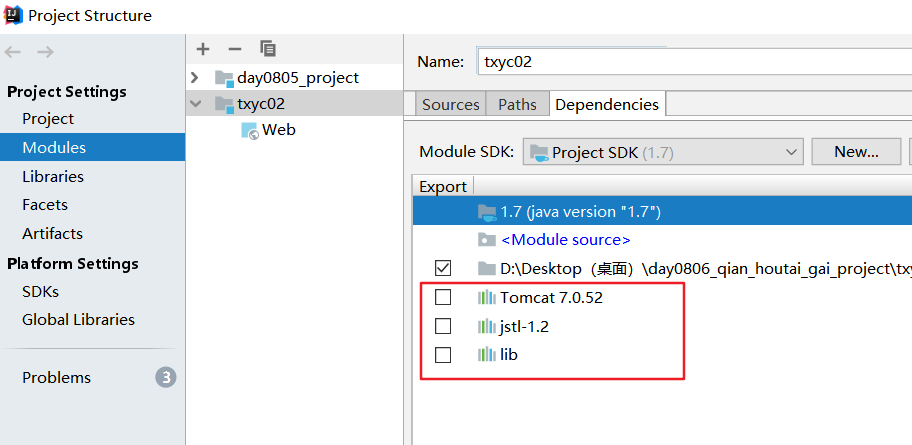
1.使用Navicat数据库管理工具在数据库创建一个t\_admin用户表。



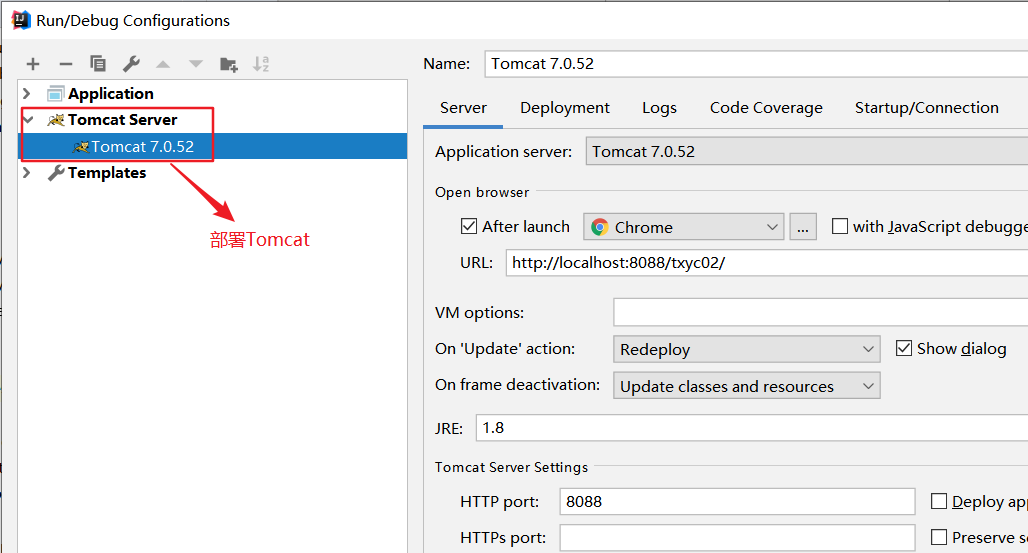
2.在IntelliJ IDEA中新建一个项目

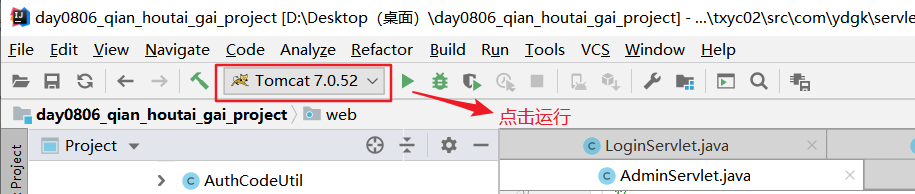


3.同时因为要用到HttpServlet,和连接数据库等，所以要配置Tomcat包和导入相关的jar包。



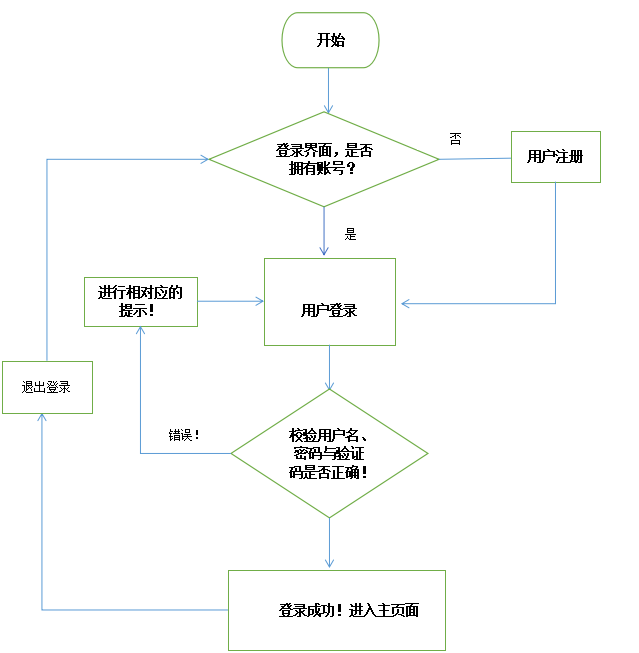
4.将项目部署到Tomcat，点击运行

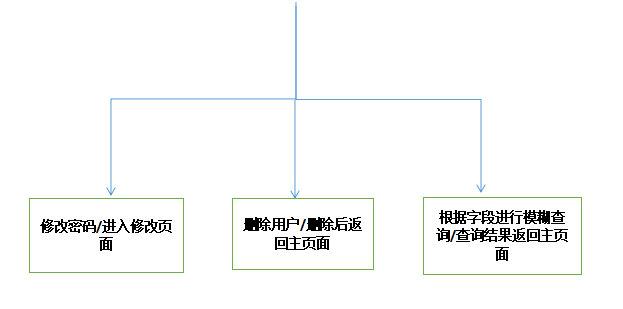




# **基于B/S模型的用户管理平台测试**

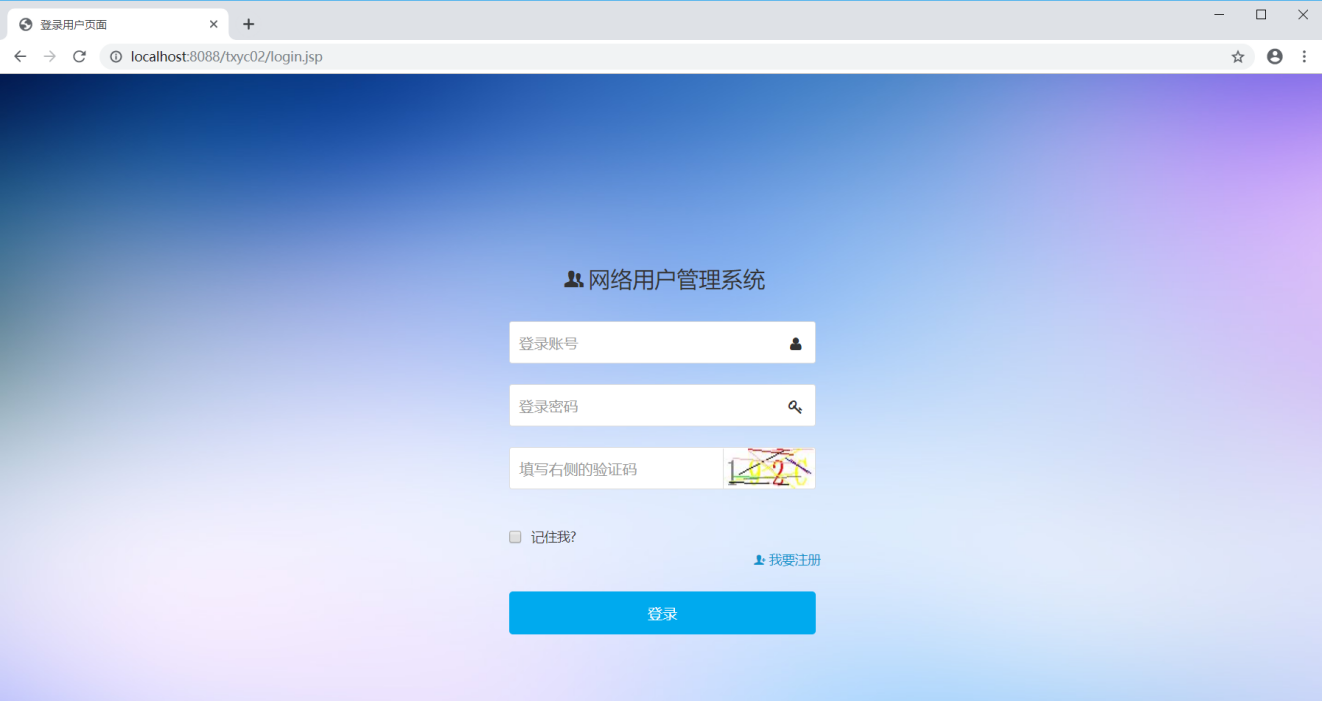
### **5.1项目运行流程图**



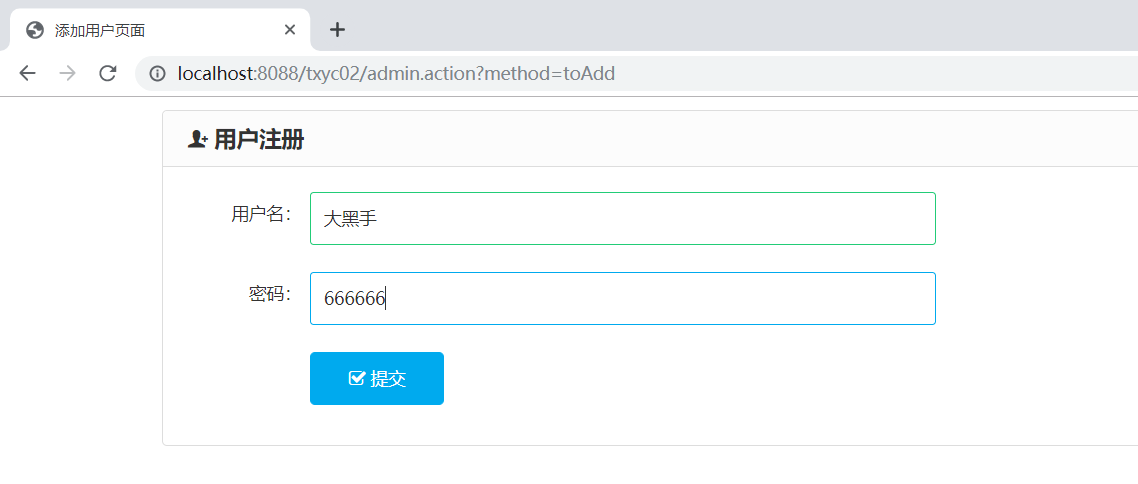
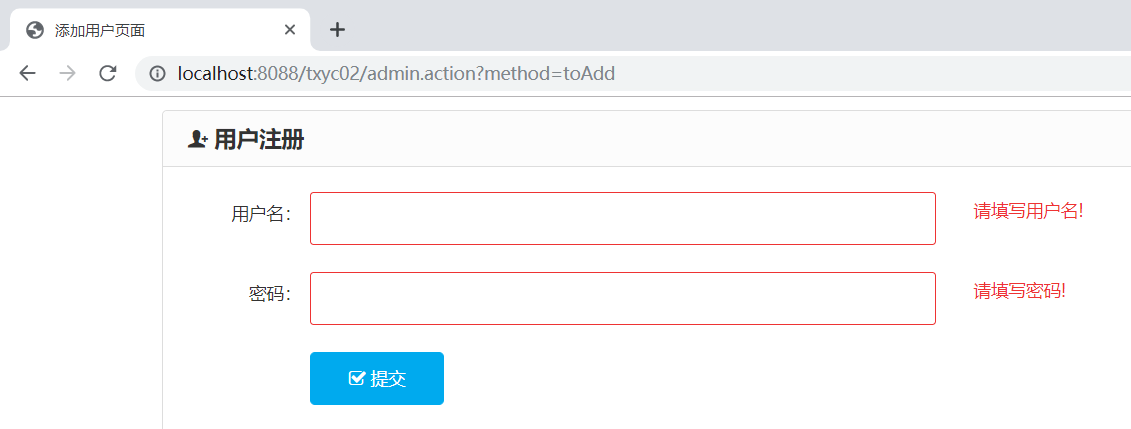


### **5.2运行界面**

**5.1.1进入用户登录页面点击进行注册**



**5.1.2用户注册与没输入提示：**



**5.1.3用户登录：**



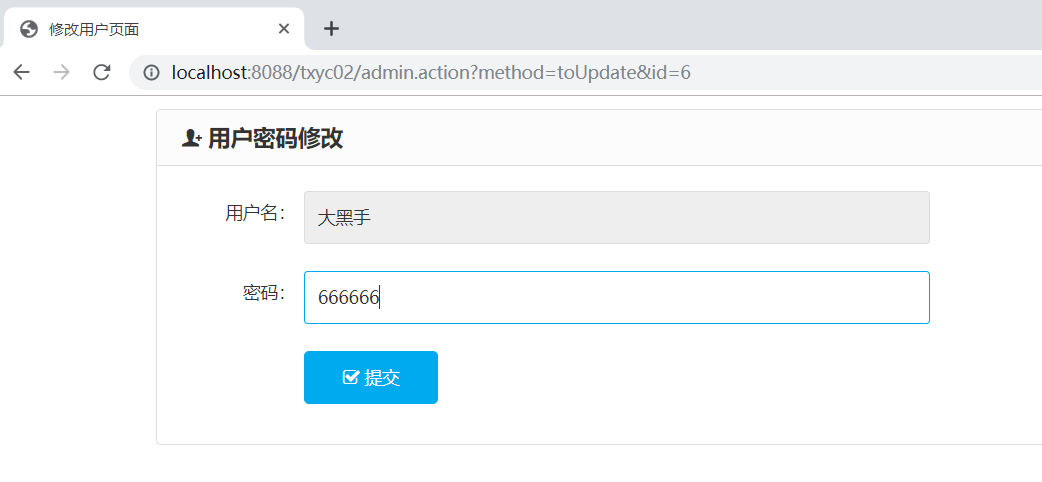
**5.1.4如果用户信息没填或者填错，能进行提示：**



**5.1.5登录成功，进入用户管理页面：**



**5.1.6能够点击进行修改密码：**



**5.1.7能够进行模糊查询用户、分页、删除用户以及退出登录：**



**附录**

**主要部分源码：**

**web.xml配置代码：**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  
 version="4.0"**>  
 *<!--设置欢迎页面，项目启动直接进入欢迎页，也就是登录页面-->*<**welcome-file-list**>  
 <**welcome-file**>login.jsp</**welcome-file**>  
</**welcome-file-list**>  
</**web-app**>

**Login.jsp登录页面代码：**

<%@ **page language**="**java**" **import**="**java.util.\***" **pageEncoding**="**UTF-8**" %>  
<!DOCTYPE **html**>  
<**html lang="zh-cn"**>  
<**head**>  
 <**title**>登录用户页面</**title**>  
 <**link rel="stylesheet" href="css/pintuer.css"**>  
 <**link rel="stylesheet" href="css/admin.css"**>  
 <**link rel="stylesheet" type="text/css" media="screen"  
 href="https://cdn.staticfile.org/ionicons/2.0.1/css/ionicons.min.css"**>  
 <**script src="js/jquery.js"**></**script**>  
 <**script src="js/pintuer.js"**></**script**>  
</**head**>  
<**body**>  
<**div class="bg"**></**div**>  
<**div class="container"**>  
 <**div class="line bouncein"**>  
 <**div class="xs6 xm4 xs3-move xm4-move"**>  
 <**div style="height**:150**px**;**"**></**div**>  
 <**div class="media media-y margin-big-bottom"**></**div**>  
 <**form action="${**pageContext.request.contextPath **}/login.action" method="post"**>  
 <**div class="panel loginbox"**>  
 <**div class="text-center margin-big padding-big-top"**>  
 <**h1** ><**i class="icon ion-person-stalker margin-small"**></**i**>网络用户管理系统</**h1**>  
 </**div**>  
 <**div class="panel-body"  
 style="padding**:30**px**; **padding-bottom**:10**px**; **padding-top**:10**px**;**"**>  
 <**div class="form-group"**>  
 <**div class="field field-icon-right"**>  
 <**input type="text" class="input input-big" name="uloginname"  
 values="admin" placeholder="登录账号"  
 data-validate="required:请填写账号!"**/> <**span  
 class="icon icon-user margin-small"**></**span**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**div class="field field-icon-right"**>  
 <**input type="password" class="input input-big" name="upsw"  
 placeholder="登录密码" values="123456"  
 data-validate="required:请填写密码!"**/> <**span  
 class="icon icon-key margin-small"**></**span**>  
 </**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**div class="field"**>  
 <**input type="text" class="input input-big" name="code"  
 placeholder="填写右侧的验证码" data-validate="required:请填写右侧的验证码!"**/>  
 <**img src="authCode.action" alt="" width="100" height="32"  
 class="passcode" style="height**:43**px**;**cursor**:**pointer**;**"  
 onclick="this**.**src**=**this**.**src**+**'?'"**>  
 </**div**>  
 *<%--回显错误信息--%>* <**span style="color**:**red**;**float**:**right"**>**${**errorMsg**}**</**span**>  
 </**div**>  
 </**div**>  
 <**BR**>  
 <**div**>  
 *<%--<div style="padding:30px;" >--%>* <**label style="padding**:30**px**;**"**>  
 <**input type="checkbox" value="remember-me" ${**requestScope.rememberFlag ? **'checked'**: **''} name="remember"**>  
 记住我?  
 </**label**>  
 <**label style="padding**:25**px**;**float**:**right"**>  
 <**a style="color**: **#0F91CB" href="${**pageContext.request.contextPath **}/admin.action?method=toAdd"** ><**i class="icon ion-person-add "**></**i**> 我要注册</**a**>  
 </**label**>  
 </**div**>  
 <**div style="padding**:30**px**;**"**>  
  
 <**input type="submit"  
 class="button button-block bg-main text-big input-big"  
 value="登录"**/>  
 </**div**>  
 </**div**>  
 </**form**>  
 </**div**>  
 </**div**>  
</**div**>  
</**body**>  
</**html**>

**LoginServlet登录控制层的代码：**

@WebServlet(**"/login.action"**)  
**public class** LoginServlet **extends** HttpServlet {  
 **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 **this**.doGet(request, response);  
 }  
  
 **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//获取表单传来的参数* String username=request.getParameter(**"uloginname"**);  
 String userpwd=request.getParameter(**"upsw"**);  
 String code=request.getParameter(**"code"**);  
 *//判断输入的验证码和session中存的验证码是否一致* HttpSession session=request.getSession();  
 String checkCode = (String) session.getAttribute(**"code"**);  
 *//比较输入的验证码和session中的验证码是否相同，equalsIgnoreCase:不区分大小写的比较* **if** (code.equalsIgnoreCase(checkCode)) {  
 *// 调用AdminDao的登录方法，传入用户名和密码进行数据库查询用户* AdminDao adminDao=**new** AdminDaoImpl();  
 Admin admin = adminDao.queryByNameAndPwd(username,userpwd);  
 **if** (admin != **null**) {  
 *// 如果登录成功，跳转到后台主页  
 // 1.将管理员对象保存在session* session.setAttribute(**"admin"**, admin);  
 *// 2.跳转到后台主页* request.getRequestDispatcher(**"admin.action?method=queryByPage"**).forward(request, response);  
  
 } **else** {  
 *// 登录失败，跳转到登录页面  
 // 1.将错误信息存到request中* request.setAttribute(**"errorMsg"**, **"用户名或密码错误"**);  
 *// 2.转发到登录页面* request.getRequestDispatcher(**"login.jsp"**).forward(request, response);  
  
 }  
  
 } **else** {  
 *// 验证码错误，跳转到登录页面  
 // 1.将错误信息存到request中* request.setAttribute(**"errorMsg"**, **"验证码错误!"**);  
 *// 2.转发到登录页面* request.getRequestDispatcher(**"login.jsp"**).forward(request, response);  
  
 }  
  
 }  
}

**List.jsp,用户界面的代码：**

<%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %>  
<%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  
<**html**>  
<**head**>  
 <**title**>用户界面</**title**>  
 <**style**>  
 .**sellink**{  
 **color**:**orangered**;  
 }  
 </**style**>  
</**head**>  
<**body**>  
<**h1 align="center"**>用户管理系统</**h1**>  
<**h2 align="center"**>  
 <**form action="admin.action?method=queryByPage" method="post"**>  
 名称：<**input type="text" name="name" value="${**name**}"**>  
 <**input type="submit" value="查询"**>  
 </**form**>  
</**h2**>  
<**table align="center" border="1" cellpadding="20" cellspacing="0" width="50%" height="10%"**>  
 <**tr**>  
 <**th**>用户ID</**th**>  
 <**th**>用户名称</**th**>  
 <**th**>用户密码</**th**>  
 <**th**>操作</**th**>  
 </**tr**>  
 <**c:forEach var="admin" items="${**pb.list**}"**>  
 <**tr align="center"**>  
  
 <**td**>**${**admin.userid**}**</**td**>  
 <**td**>**${**admin.username**}**</**td**>  
 <**td**>**${**admin.userpwd**}**</**td**>  
 <**td**>  
 <**a href="javascript:void(0)" onclick="***queryAdmin*(**${**admin.userid**}**)**"**>修改密码</**a**> |  
 *<%-- javascript:void(0)：用在href属性上，表示取消超链接的任何跳转，只响应JS事件--%>* <**a href="javascript:void(0)" onclick="***deleteAdmin*(**${**admin.userid**}**)**"**>删除用户</**a**>  
 </**td**>  
 </**tr**>  
 </**c:forEach**>  
</**table**>  
<**br**><**br**>  
*<%--上一页--%>*<**div align="center"**>  
<**c:choose**>  
 <**c:when test="${**pb.pageNum **gt** 1**}"**>  
 <**a href="admin.action?method=queryByPage&name=${**name**}&pageNum=${**pb.pageNum-1**}"**>上一页</**a**>  
 </**c:when**>  
 <**c:otherwise**>上一页</**c:otherwise**>  
</**c:choose**>  
*<%--生成所有页号的超链接，作为分页导航菜单  
 当前页的页号,就添加class="sellink"的样式;否则没有该样式  
--%>*<**c:forEach var="sn" begin="1" end="${**pb.totalPage**}"**>  
 <**c:choose**>  
 <**c:when test="${**sn **eq** pb.pageNum**}"**>  
 <**a href="admin.action?method=queryByPage&name=${**name**}&pageNum=${**sn**}" class="sellink"**>**${**sn**}**</**a**>  
 </**c:when**>  
 <**c:otherwise**>  
 <**a href="admin.action?method=queryByPage&name=${**name**}&pageNum=${**sn**}"**>**${**sn**}**</**a**>  
 </**c:otherwise**>  
 </**c:choose**>  
</**c:forEach**>  
*<%--下一页--%>*<**c:choose**>  
 <**c:when test="${**pb.pageNum **lt** pb.totalPage**}"**>  
 <**a href="admin.action?method=queryByPage&name=${**name**}&pageNum=${**pb.pageNum+1**}"**>下一页</**a**>  
 </**c:when**>  
 <**c:otherwise**>下一页</**c:otherwise**>  
</**c:choose**>  
 <----->跳转到第<**select id="selNum" onchange="***changePageNum*()**"**>  
 <**c:forEach var="sn" begin="1" end="${**pb.totalPage**}"**>  
 <**option value="${**sn**}" ${**sn **eq** pb.pageNum?**"selected"**:**""}**>**${**sn**}**</**option**>  
 </**c:forEach**>  
</**select**>页 **&nbsp;&nbsp;**<**a href="${**pageContext.request.contextPath**}/login.jsp"**>退出登录</**a**>  
</**div**>  
<**br**><**br**>  
  
<**script**>  
 *//点击删除后调用的函数* **function** *deleteAdmin*(sid){  
 **var** flag=*confirm*(**"确定要删除吗？"**)  
 **if**(flag){  
 *//请求CustomerServlet页面，使用location.href发送get请求* ***location***.**href**=**"admin.action?method=deleteAdmin&id="**+sid;  
 }  
 }  
 *//点击修改后调用的函数* **function** *queryAdmin*(id) {  
 *//请求CustomerServlet,用location.href发送get请求* ***location***.**href**=**"admin.action?method=toUpdate&id="**+id;  
 }  
 *//选择下拉列表中的页号时，调用changePageNum()函数* **function** *changePageNum*() {  
 **var** pageNum = ***document***.getElementById(**'selNum'**).**value**;  
 ***location***.**href** = **"admin.action?method=queryByPage&name=${**name**}&pageNum="** + pageNum;  
 }  
</**script**>  
</**body**>  
</**html**>

**AdminServlet用户管理的代码：**

*//使用注解的形式来配置Servlet*@WebServlet(**"/admin.action"**)  
**public class** AdminServlet **extends** HttpServlet {  
 *//为下面调用adminDaoImpl里面的方法做准备* AdminDao **adminDao** = **new** AdminDaoImpl();  
  
 *//重写HttpServlet中的doPost和doGet方法，这里为了方便，直接将请求交给doGet进行受理。* **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 **this**.doGet(request, response);  
 }  
  
 **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//防止POST请求出现中文乱码* request.setCharacterEncoding(**"UTF-8"**);  
 *//获取method参数的值* String methodName = request.getParameter(**"method"**);  
 **try** {  
 *//利用反射调用与method参数值同名的方法，方法签名必须和doGet一样* Method method = **this**.getClass().getDeclaredMethod(methodName, HttpServletRequest.**class**, HttpServletResponse.**class**);  
 method.invoke(**this**, request, response);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//从登录页面进入到注册添加页面* **protected void** toAdd(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 request.getRequestDispatcher(**"WEB-INF/jsp/add.jsp"**).forward(request, response);  
 }  
  
 *//注册添加用户* **protected void** add(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
  
 *//获取表单传来的参数* String username = request.getParameter(**"uloginname"**);  
 String userpwd = request.getParameter(**"upsw"**);  
 *//封装成admin对象* Admin admin = **new** Admin();  
 admin.setUsername(username);  
 admin.setUserpwd(userpwd);  
 *//调用dao中的方法,添加用户* **int** i = **adminDao**.insert(admin);  
 **if** (i > 0) {  
 *//i大于0表示添加成功，跳转到登录页面* response.sendRedirect(**"login.jsp"**);  
 }  
 }  
  
 *//用户分页查询，包括了模糊查询* **protected void** queryByPage(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//获取前台页面传过来的页号参数  
 //获取需要查询的字段* String name = request.getParameter(**"name"**);  
 *//将name保存到request域中，用于回显查询的参数* request.setAttribute(**"name"**,name);  
 String pageStr = request.getParameter(**"pageNum"**);  
 *//保存页号的变量* **int** pageNum = 1;  
 **if** (pageStr != **null**) {  
 pageNum = Integer.*parseInt*(pageStr);  
 }  
 *//封装一个PageBean对象* PageBean<Admin> pb = **new** PageBean<>();  
 *//封装每页条数和页号* pb.setPageNum(pageNum);  
 pb.setPageSize(3);  
 *//调用Dao方法  
 //获取当前页的数据集合* List<Admin> list = **adminDao**.queryList(pb,name);  
 *//获取总记录数* **int** totalCount = **adminDao**.queryCount();  
 *//封装当前页数据的集合和总记录数* pb.setList(list);  
 pb.setTotalCount(totalCount);  
 *//把pageBean对象保存到作用域对象中* request.setAttribute(**"pb"**, pb);  
 *//请求转发跳转到列表页面* request.getRequestDispatcher(**"/WEB-INF/jsp/list.jsp"**).forward(request, response);  
 }  
  
 *//通过id删除用户* **protected void** deleteAdmin(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//获取前台页面要删除用户的id* String id = request.getParameter(**"id"**);  
 *//调用dao方法* **int** i = **adminDao**.deleteAdminById(id);  
 **if** (i > 0) {  
 request.getRequestDispatcher(**"admin.action?method=queryByPage"**).forward(request, response);  
 }  
 }  
  
 *//通过本方法去跳转到修改页面* **protected void** toUpdate(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//获取前台页面要修改用户的id* String id = request.getParameter(**"id"**);  
 *//调用dao方法* Admin admin = **adminDao**.queryAdmin(id);  
 *//将admin对象保存到request域中* request.setAttribute(**"admin"**, admin);  
 request.getRequestDispatcher(**"/WEB-INF/jsp/update.jsp"**).forward(request, response);  
 }  
  
 *//通过id进行密码修改* **protected void** update(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *//获取前台数据* String id = request.getParameter(**"userid"**);  
 String userpwd = request.getParameter(**"upsw"**);  
 *//调用dao方法* **int** i = **adminDao**.update(id, userpwd);  
 **if** (i > 0) {  
 request.getRequestDispatcher(**"admin.action?method=queryByPage"**).forward(request, response);  
 }  
 }  
  
}

**AdminDao访问数据库的方法：**

**public interface** AdminDao {  
 *//登录用户，根据用户名和密码查询一条记录* Admin queryByNameAndPwd(String username, String userpwd);  
  
 *//添加用户* **int** insert(Admin admin);  
  
 *//查询当前页数据集合，也就是用户信息* List<Admin> queryList(PageBean<Admin> pb,String name);  
  
 *//查询总记录数* **int** queryCount();  
  
 *//根据id删除用户* **int** deleteAdminById(String id);  
  
 *//查询跳转到修改页面需要的回显的用户信息* Admin queryAdmin(String id);  
  
 *//通过id，进行密码修改* **int** update(String id, String userpwd);  
}

**Admin用户对象类：**

**public class** Admin {  
 **private** Integer **userid**;  
 **private** String **username**;  
 **private** String **userpwd**;  
  
 **public** Integer getUserid() {  
 **return userid**;  
 }  
 **public void** setUserid(Integer userid) {  
 **this**.**userid** = userid;  
 }  
 **public** String getUsername() {  
 **return username**;  
 }  
 **public void** setUsername(String username) {  
 **this**.**username** = username;  
 }  
 **public** String getUserpwd() {  
 **return userpwd**;  
 }  
 **public void** setUserpwd(String userpwd) {  
 **this**.**userpwd** = userpwd;  
 }  
}

**c3p0-config.xml代码，用于连接数据库的xml文件:**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**c3p0-config**>  
 *<!-- 命名配置，可以指定连接的数据库服务器类型 -->* <**named-config name="mysql"**>  
 *<!--JDBC驱动 -->* <**property name="driverClass"**>com.mysql.jdbc.Driver</**property**>  
 *<!--数据库地址 -->* <**property**

**name="jdbcUrl"**>jdbc:mysql://127.0.0.1:3306/ydgk06\_tysc?rewriteBatchedStatements=true  
 </**property**>  
 *<!--用户名。Default: null -->* <**property name="user"**>root</**property**>  
 *<!--密码。Default: null -->* <**property name="password"**>123456</**property**>  
 *<!-- 连接池初始化时创建的连接数，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->* <**property name="initialPoolSize"**>5</**property**>  
 *<!--连接池中保留的最大连接数。如果获得新连接时会使连接总数超过这个值则不会再获取新连接，而是等待其他连接被释放，所以这个值  
 可能会设计得很大,Default:15（建议使用） -->* <**property name="maxPoolSize"**>20</**property**>  
 *<!--连接池中保留的最小连接数,Default:3（建议使用） -->* <**property name="minPoolSize"**>3</**property**>  
 *<!-- 连接池中无空闲连接可用时， 一次性向数据库服务器申请多少个连接，Default:3 -->* <**property name="acquireIncrement"**>3</**property**>  
 *<!--最大空闲时间,如果超过这个时间，某个数据库连接还没有被使用，则会断开这个连接(从池中剔除)。若为0则永远不会断开连接。Default: 0，单位:秒 -->* <**property name="maxIdleTime"**>60</**property**>  
 *<!--JDBC的标准参数，用以控制数据源内加载的PreparedStatements数量。但由于预缓存的statements 属于单个connection而不是整个连接池。所以设置这个参数需要考虑到多方面的因素。  
 如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0 -->* <**property name="maxStatements"**>200</**property**>  
 </**named-config**>  
</**c3p0-config**>

参考文献