# 作业二

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | **Web技术** | **班级** | **计科2001** | **学号** | **8204200213** | | **姓名** | **韩非江** |
| **名称** | **作业3 登录界面设计** | | | **时间** | | 2022.12.13前 | | |
| **环境** | | | | | | | | |
| Windows | | | | | | | | |
| **目的和内容要求** | | | | | | | | |
| 本次实验任务为实现一个基础的登录系统，可完成前后端数据交互。  上传两份文件  1. 实验报告.doc （截图+说明+总结）  2. 源码压缩包（带注释）  实验内容：  1. 实现登录界面，布局**类似**于学校门户.  2. 完成前后端数据交互（用JSON格式）  3. 数据库不限，后端技术不限，但不能用框架。  功能包括：  （1）显示当前城市及天气信息；  （2）登录界面；  （3）手机号码（邮箱）+验证码登录，验证通过后才能登录；  （4）登录成功后提示及跳转（可跳转至CSU学校主页作业 或者个人简历）；  （5）数据统一存储在后端数据库中。  \*（6）忘记密码、修改密码？  \*（7）用户注册、用户注销、用户信息修改？  \*（8）查找/搜索  \*（9）管理员功能  \*（10）不仅限于以上功能。。。  前五项为基础功能，后五项为可选功能（加分项目）。  其他说明：  1. 实验独立完成；  2. 实验检查时间 周二12月6日至 12月13日。  3. 第17周开卷考试 | | | | | | | | |
| **实验过程、结果分析与总结** | | | | | | | | |
| 1. **程序运行过程和结果（请另外提供所完成的html/css等文件（压缩包），在此贴上运行结果的界面截图）：**   **（一）主页面：**    **（二）用户登录后页面：**  **（三）用户修改密码界面：**    （四）动态码登录界面：    （五）忘记密码界面：    **（六）管理员登录后界面：**    **（七）用户增删改查界面：**    **（八）模糊查询：**    **（九）修改用户信息：**    （十）添加用户界面：     1. **实验整体设计**   本次实验为前后端交互的实验，前端主要是jsp+js+css，后端用java的servlet和数据库操作。在实现之前，我先学习了Java Web项目的运行原理和组织架构，如下图：    浏览器中显示的是视图层，常见的如html、js、css等前端代码就是这一层；控制层可以控制视图层显示的内容，Servlet就在此层来处理传入的request和response请求；模型层是控制层的基础，使用java实现额外的附加功能如数据库查询、实体类构建、用户附加服务等操作。  学完大致的项目流程后，开始构建具体的Java Web项目。首先我想构建最熟悉的控制层，每个Servlet文件都对应与登录页面中的一种请求，我梳理出如下请求：  **用户：**  登录请求处理：LoginServlet  获取天气请求处理：GetWeather  生成动态图片验证码请求：CreateCode  获得手机和邮箱验证码请求：getDynamicCode-动态登录；ForgetServlet-忘记密码  更新密码请求：UpdatePsw-用户更新  忘记密码请求  **管理员：**  查询请求：AdminServlet  修改用户请求：UpdateUserF  添加用户请求：AddUserServlet  删除用户请求：DeleteServlet    图2.2请求对应的servlet类  然后开始设计模型层，我认为模型层中共有五个部分：  dao：数据库操作函数封装  entity：对用户和管理员进行封装，内部使用dao中的数据库操作。  JSON：根据http请求获得json数据，一般用于通过url请求API接口  service：用户和管理员的功能函数封装，内部调用数据库操作函数  utils：工具。如获取电话、邮箱验证码；获取天气；读取database.properties文件进行数据库连接断开操作。    图2.3模型层设计  最后是视图层，登录界面的视图层设计如下，这一层是我最熟练的一层，因此只做简单展示：    图2.4视图层设计  感悟：如果要用javaWeb复现一个新网站，我首先会从请求种类入手，每种请求都可以设计一种servlet处理对应的request和response；然后根据需求用java引入数据库操作，并在数据库操作的基础上进行封装，进而构建出用户、管理员、其他人员的实体；再然后实现前端代码即可，在前端jsp文件和js文件中都可以和servlet进行交互，因为前面已经设计了servlet，所以这一步会比较简单；最后，再反过头来添加各种java功能类，如天气查询、短信邮件收发等功能。  （我寒假还要使用django+React开发实验室的聚类系统，刚才仔细梳理了一遍开发过程，发现和javaWeb有很大的不同，继续努力。） 实验细节和代码实现说明3.1 概述 本实验所用的技术按照重要性的顺序（考试重要性）为Servlet、Ajax、html、js、css，因此本章内容组织如下：  3.1小节除了进行内容概述，还分析了实验中每个模块的具体实现思路，主要包括登录模块、验证码模块、增删改查模块、天气模块等。  3.2小节介绍实验中Servlet相关内容，包括对Servlet原理的再次理解、写代码的一般流程、实验中典型Servlet类的实现、servlet向jsp传值几种方式（request和session）这几个部分；  3.3小节介绍实验中Ajax相关的内容，包括Ajax的基础流程，以及实验中用到的Ajax两部分；  3.4小节介绍实验中的前端技术，包括jsp和js、css的构建，主要总结jsp中如何使用servlet传入的值，以及jsp中如何和servlet进行沟通，此外还有表单的设计和实现、变灰的实现等内容；  此处需要说明的是，针对java数据库的操作和抽象实体的方式并非本实验的重点，此外我以前也很少用java，今后的工作也是前端或者后端c++/python，因此此部分放在最后简单展示和说明。  下面先来介绍各个模块的实现思路，具体的代码部分将在之后几个小节按照所用技术进行分类说明。   1. 登录的实现思路：  * 首先前端需要用表单收集登录所需用户名和密码，将输入值作为参数提交给后端servlet。 * Servlet中初始化数据库对象，获取前端输入的用户名和密码，判断数据库中是否存在此用户，若存在则跳转至登陆成功页面，同时存储数据库中的user到session中；如果用户不存在则使用request传递错误信息给前端jsp页面，提示用户名密码错误。  1. 忘记密码的实现思路：  * 首先前端需要输入学号、手机号或邮箱、图片验证码、手机或邮箱验证码、新密码和确认密码，并将表单提交至servlet，servlet依旧需要先获取参数再进行数据库操作。 * 获取手机验证码时使用Ajax，首先获取输入框内容，然后构造get请求到servlet，在servlet中判断输入的图片验证码是否正确、学号对应的实体是否存在于数据库，如果都符合要求则判断用户输入的是手机号还是邮箱，最后再调用API实现验证码发送。  1. 图片验证码的实现思路：  * 首先在本地生成随机字母+数字的组合 * 然后调用接口在本地生成指定大小和数字的图片 * 使用write函数将本地图片写到前端页面中的img标签中  1. 手机验证码和短信验证的实现思路：  * 配置token和模式码 * 本地随机生成n位数字 * Ajax辅助获取用户输入的手机号或邮箱 * 将手机号/邮箱和验证码传入API接口，API自动进行验证码发送  1. 天气显示的实现思路：  * 找到长沙的对应天气编号 * 调用工具类，传入天气查询URL，返回查询得到的json数据 * 使用request和redirect将数据传输到前端jsp页面  1. 页面跳转显示用户名的思路：  * 登录判断成功时先用session将用户实体存储在服务器 * 使用redirect或者dispatch进行页面跳转 * 新的jsp页面中获取并显示session中user的信息  1. 增删改查的实现思路：  * 表单提交，servlet处理，调用数据库，更新页面  3.2 Servlet的实现 **（一）对Servlet原理的再次理解**  第一部分中想阐述做完实验后对servlet和tomcat的个人理解，因为我们这部分的课时较少，课堂上我并没有学的特别清楚，正好利用这次实验理解一些原理。  Web 服务器是整个动态网站的“大门”，用户的 HTTP 请求首先到达 Web 服务器，Web 服务器判断该请求是静态资源还是动态资源：如果是静态资源就直接返回，此时相当于用户下载了一个服务器上的文件；如果是动态资源将无法处理，必须将该请求转发给 Servlet 容器。而我认为Tomcat就是一个web服务器，可以接受http请求，并做出响应。对于普通的电脑来说没有web服务器，因此无法得知浏览器的http请求何时到达，更不能调用对应的java程序执行。Tomcat可以监听端口，及时接受请求。在《计算机网络》这门课中我曾实现过一个web服务器，可以解析浏览器的请求并提供请求的本地资源，在接触《web技术》这门课程之前我要从html中请求资源就是用我写的web服务器实现的，有了Tomcat以后就不需要那么麻烦了。  不过我们通常所说的 Web 服务器，比如Nginx等，它们的功能往往都比较单一，只能提供 http(s) 服务，让用户访问静态资源（HTML 文档、图片、CSS 文件、JavaScript 文件等），它们不能执行任何编程语言，也不能访问数据库，更不能让用户注册和登录。如果只有 Web 服务器，那就只能部署静态网站，不能部署动态网站。要想部署动态网站，必须要有编程语言运行环境和数据库管理系统的支持。  而servlet就是依托于Tomcat专门用来接收客户端的请求，然后调用底层service处理数据并生成结果，大致流程为：浏览器http请求----tomcat服务器----到达servlet----执行doGet，doPost方法----服务器得到返回数据----反馈给浏览器。  IMG_256IMG_256   1. 写代码的一般流程   **public** **class** MyServlet **extends** HttpServlet {   1. **public** void doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException { 2. } 3. **public** void doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException { 4. doGet(req, resp); 5. } 6. }   **在很多本实验的大多数servlet文件中我都用到了以下五个步骤：**  **1.设置编码格式**  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");  **2.接受参数**  String parameter1 = request.getParameter("key1");  String parameter2 = request.getParameter("key2");  **3.生成session会话**  HttpSession session = request.getSession();  **4.获取session中的数值**  String parameter3 = (String) session.getAttribute('key3');  **5.调用各种java类的service进行操作**  例如数据库的初始化、查询  **6.根据service操作的结果分支判断，进行dispatch或者redirect**  **例如：**在登录时通过判断用户是否存在决定用redirect跳转到成功页面还是用dispatch输出错误信息。  if(user!=null) {  // 用户存在  // 保存用户名字  session.setAttribute("user",user);  session.setAttribute("username",user.getUsername()); // 跳转到登录成功的页面 response.sendRedirect(request.getContextPath() + "/success.jsp");  } else {  // 用户不存在，提示错误信息  request.setAttribute("login\_error","您提供的用户名或者密码有误"); request.getRequestDispatcher("/index.jsp").forward(request,response);  }  **对于第5步，这里要强调一下转发和重定向的区别，经过和群里小伙伴的讨论以及在网上查资料，我得到了如下结论：**   1. 转发时共享request；而重定向相当于请求了两次，不共享request。 2. 转发不改变网址；重定向改变网址。 3. 转发不跨域；重定向跨域。 4. 转发到jsp页面后，本质上请求的还是servlet文件，每次刷新都会重新请求servlet类；而重定向到jsp页面后，每次刷新只会重新请求jsp页面。 5. 重定向开销比转发略高，且**重定向只能使用session传递数据不能使用request**，而使用session传值对需要持久访问的值比较好，但是对于一次性访问的值就不好。 6. 实验中典型Servlet类的实现   第一种典型的servlet就是用来进行数据库逻辑判断的，例如用户和管理员的登录、忘记密码后重新提交、修改信息等表单操作；获取所有用户信息这样的查询操作等。   1. 登录表单处理   @WebServlet(name = "LoginServlet", value = "/LoginServlet") public class LoginServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");   // 接收参数  String userid = request.getParameter("userid");  String password = request.getParameter("password");   // 调用业务逻辑  UserService userService = new UserServiceImpl();  User user = userService.login(userid,password);   // 处理结果  HttpSession session = request.getSession();  if(user!=null){ // 用户存在  // 保存用户名字  session.setAttribute("user",user);  session.setAttribute("username",user.getUsername());  // 跳转到登录成功的页面  response.sendRedirect(request.getContextPath() + "/success.jsp");  }  else{ // 用户不存在，提示错误信息  request.setAttribute("login\_error","您提供的用户名或者密码有误");  request.getRequestDispatcher("/index.jsp").forward(request,response);  }  }   @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request,response);  } }  **分析：**  首先使用request.getParameter获得从input输入的参数，然后new UserServiceImpl()  初始化数据库，调用userService.login判断是否能再数据库中查询到输入的账号和密码。如果查询成功，则使用session.setAttribute把查询到的user和userName存储到session中，再使用response.sendRedirect重定向到成功jsp页面；如果查询失败，则使用request.setAttribute设置错误值，并在jsp页面显示。   1. 动态验证码登录   @WebServlet(name = "LoginDynamicServlet", value = "/LoginDynamicServlet") public class LoginDynamicServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");   HttpSession session = request.getSession();   String rightCode = (String) session.getAttribute("rightDCode");  String code = request.getParameter("DCode");   User user = (User) session.getAttribute("user");  if(rightCode.equals(code)){  // 动态码输入正确  // 保存用户名字  session.setAttribute("username",user.getUsername());  // 跳转到登录成功的页面  response.sendRedirect(request.getContextPath() + "/success.jsp");  }  else{  request.setAttribute("DCode\_error","您输入的动态码有误");  request.getRequestDispatcher("/loginDynamic.jsp").forward(request,response);  }   }   @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request,response);  } }  分析：验证码登录的过程和普通登录过程相似，只多了使用session.getAttribute从session读取生成正确验证码来进行匹配的过程。   1. 用户修改自己的密码   @WebServlet(name = "UpdatePsw", value = "/UpdatePsw") public class UpdatePsw extends HttpServlet {  @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");   HttpSession session = request.getSession();   String newpsw1 = request.getParameter("newpsw1");  String newpsw2 = request.getParameter("newpsw2");  User user = (User) session.getAttribute("user");   boolean isEmpty = newpsw1.length()==0 || newpsw2.length()==0;   if(!isEmpty && newpsw1.equals(newpsw2)){  UserService userService = new UserServiceImpl();  int res = userService.updatePsw(user.getUserid(),newpsw1);  response.sendRedirect(request.getContextPath() + "/success.jsp");  }  else if(!isEmpty){  request.setAttribute("error","您两次输入的密码不同");  request.getRequestDispatcher("/updatePsw.jsp").forward(request,response);  } else {  request.setAttribute("error","密码不能为空！");  request.getRequestDispatcher("/updatePsw.jsp").forward(request,response);  }  }  分析：依旧是前面总结的5步，先接受参数，在进行逻辑判断，只是这里不需要进行数据库操作了。  **总结：前面三种servlet的处理方式类似，都是收参+逻辑判断。对用于表单处理的servlet而言，我认为最重要不是收参而是逻辑判断，前面用到的逻辑判断主要用if分支实现，涉及到数据库查询、等值比较、判空等判断条件，分支的结果往往对应于request、session的属性设置和redirect、dispatch的设置。**  D.管理员修改用户信息  @WebServlet(name = "UpdateUserServlet", value = "/UpdateUserServlet") public class UpdateUserServlet extends HttpServlet {   @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");  String userid = request.getParameter("userid");  String username = request.getParameter("username");  String password = request.getParameter("password");  String tel = request.getParameter("tel");  String email = request.getParameter("email");  UserService userService = new UserServiceImpl();  User user = userService.haveId(userid);  user.setUsername(username);  user.setPassword(password);  user.setTel(tel);  user.setEmail(email);  userService.updateUser(user);  request.getRequestDispatcher("/AdminServlet").forward(request,response);  }  @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  doGet(request,response);  } }  @WebServlet(name = "AdminServlet", value = "/AdminServlet") public class AdminServlet extends HttpServlet { // 产生所有用户  @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");   // 加载得到所有的用户信息  UserService userService = new UserServiceImpl();  List<User> user = userService.showAllUser();   //创建Jackson的核心对象 ObjectMapper  ObjectMapper mapper = new ObjectMapper();  String json = mapper.writeValueAsString(user);   HttpSession session = request.getSession();  session.setAttribute("json",json);  session.setAttribute("users",user);  request.getRequestDispatcher("/admin.jsp").forward(request,response);  //response.sendRedirect(request.getContextPath() +"/admin.jsp");  }  在adminjs中动态绘制页面:  <%-- 展示用户信息列表--%> <table class="table">  <tr class="thead">  <th>学号</th>  <th>姓名</th>  <th>密码</th>  <th>手机号</th>  <th>邮箱</th>  <th>操作</th>  </tr>  <**c:forEach** items="${sessionScope.users}" var="user" varStatus="s">  <tr>  <td>${user.userid}</td>  <td>${user.username}</td>  <td>${user.password}</td>  <td>${user.tel}</td>  <td>${user.email}</td>  <td>  <a class="btn btn-default btn-sm" href="${pageContext.request.contextPath}/UpdateUserF?userid=${user.userid}" >修改</a>&nbsp;  <a class="btn btn-default btn-sm" href="${pageContext.request.contextPath}/DeleteServlet?userid=${user.userid}">删除</a>  </td>  </tr>  </**c:forEach**>  </table>  分析：这两个servlet共同来处理管理员对用户的更新请求。第一个servlet类中只调用数据库中的updateUser进行用户更新，然后dispatch到另一个servlet中继续处理，把新的user重新写入session中。  **总结：从这里例子中，我们可以看到两个servlet是通过数据库进行user的传递的，首先现在一个servlet中修改数据库中的user，再从另一个servlet中使用数据库查询操作获得此user。不过servlet之间的传值方式不止这一种，下面进行一个小总结。如下是一个博客中的总结，我认为里面应该再加一项通过数据库在两个servlet中传值。**    **虽说有很多种传值方式，但是本次实验中构建登录界面用的最多的是Session的传递。**  例子1：图片验证码存于Session之中，创建code以后将其记录与session中：   // 将验证码存储在session中的codes属性, 后期匹配校验时用到  HttpSession session = request.getSession();  session.setAttribute("codes",codes);  获取短信验证码前需要先验证图片验证码是否正确，因此从session中取出Rightcode与用户输入的Usercode进行判断：   String rightDCode = (String) session.getAttribute("codes");  例子2：用户登录后存储user信息到session中。在用户登录输入密码后，去数据库查找是否存在user，若存在则将其保存于session的user字段：   User user = userService.login(userid,password);  // 处理结果  HttpSession session = request.getSession();  if(user!=null){     // 用户存在     // 保存用户名字     session.setAttribute("user",user);     session.setAttribute("username",user.getUsername());     // 跳转到登录成功的页面     response.sendRedirect(request.getContextPath() + "/success.jsp");  }  在用户登录后修改密码时，读取session中user字段：   User user = (User) session.getAttribute("user");、  userService.updatePsw(user.getUserid(),newpsw1);  第二种servlet是只用来调用API返回生成或请求的数据，例如天气显示这种调用外接API的；发送短信、邮箱验证码这种调用自己写的接口的。   1. 天气查询   在servlet中直接使用了url请求接口获取API返回的json数据，然后把想要显示在前端的数据通过request传输到jsp页面即可：  public class GetWeather extends HttpServlet {  @Override  protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.setCharacterEncoding("UTF-8");  // 手动设置域对象内容，此内容可使用请求转发发送给其他servlet或者jsp页面  String url="https://v0.yiketianqi.com/api?unescape=1&version=v61&appid=73742951&appsecret=gfUOps6r&cityid=101250101";  String weatherJsonStr= NetUtil.*get*(url);  JSONObject weatherJson = JSONObject.*fromObject*(weatherJsonStr);  req.setAttribute("cityEn",weatherJson.get("cityEn"));  req.setAttribute("update\_time",weatherJson.get("update\_time"));  req.setAttribute("humidity",weatherJson.get("humidity"));  req.setAttribute("visibility",weatherJson.get("visibility"));  req.setAttribute("tem",weatherJson.get("tem1"));  req.setAttribute("air\_tips",weatherJson.get("air\_tips"));；。  System.*out*.println(weatherJson);  // 请求转发  // 后端使用setAttribute设置域对象内容，通过请求转发把内容传给前端jsp，jsp可以动态把内容画在html上显示出来。  // 前端收到后端的req以后也可以直接使用getParameter读取浏览器查询参数，然后使用response.getWriter().println画到浏览器上  req.getRequestDispatcher("index.jsp").forward(req,resp);  } }  B.发送验证码  只需要调用封装好的函数，传入手机号/邮箱+随机生成的验证码即可：   1. 邮箱验证码   String code = *generateRandomCode*(6); // 生成邮箱验证啊 *sendMail*(usertel,code); // 发送邮件 session.setAttribute("rightDCode",code); session.setAttribute("user",user);  （2）手机验证码  String telcode = *generateTelCode*(6); // 调用函数生成验证码 try {  String sendCoderes = *sendCode*(usertel,"",telcode); // 调用函数发送验证码 } catch (Exception e) {  e.printStackTrace(); } session.setAttribute("rightDCode",telcode); session.setAttribute("user",user);  **总结：实验中用到的第二种servlet中没有数据库操作，而是用于调用API来返回数据或发送数据，此时应该做好逻辑判断条件，即什么时候该发什么东西、什么时候返回什么数据。例如短信验证码只有在满足图片验证码正确且用户输入手机号正确的前提下才会发送。**   1. servlet向jsp传值几种方式（request和session）   1.使用dispatch：例如-错误信息提示  Servlet：   request.setAttribute("login\_error","您提供的用户名或者密码有误");  request.getRequestDispatcher("/index.jsp").forward(request,response);  Jsp:    <%--   提示输入的信息错误--%>   <div id="formErrorTip" class="form-errorTip">   <span id="showErrorTip" class="form-error text-ellipsis">${requestScope.login\_error}</span>   </div>  2.使用Redirect：例如-登录后从session获取信息  批注：重定向 (Redirect)是路径的跳转，内容和url都改变。不允许带 request参数，即不允许在servlet里给 request对象使用setAttribute方法传给下一页面。但是可以使用session参数  Servlet：   HttpSession session = request.getSession();  // 保存用户名字  session.setAttribute("user",user);  session.setAttribute("username",user.getUsername());  // 跳转到登录成功的页面  response.sendRedirect(request.getContextPath() + "/success.jsp");  Jsp：   <h2>${sessionScope.username},你好呀！</h2>  **3.2总结：本节我学会了servlet编程的一般套路：收参+调用业务+逻辑+跳转。也明白了servlet的两种应用方式：有数据库操作的servlet、调用API接口的servlet。本节也结合项目总结了servlet之间传值的四种方式、servlet到jsp传值的两种方式。我也理解了session的作用以及重定向和转发的区别。**  **3.3 Ajax的实现**  我在本实验中Ajax用到的地方较少，因此此处除了结合实验中具体用到Ajax的地方以外，还结合了赵启涵同学课上讲解的案例，最终整理出自己理解的Ajax的一般实现流程，再此感谢赵启涵同学。  **Ajax可以**在不重新加载整个页面的情况下与服务器交换数据并更新部分网页内容，一般在javascript中使用XMLHttpRequest 来实现Ajax，但是XMLHttpRequest 只是实现 Ajax 的一种方式。菜鸟教程描述如下：    本实验在“忘记密码”模块中获取手机验证码时，就会使用Ajax 的异步请求，进而在不提交表单的时候也可以调用servlet中的判断逻辑。  下面用一个实验中具体的例子来整理Ajax的使用：  function checktelcaptcha(){  var userid = ***document***.querySelector("#userid").value;  var usertel = ***document***.querySelector("#usertel").value;  var captcha = ***document***.querySelector("#captcha").value;  // 送异步请求  // 1.创建核心对象  var xmlhttp;  if (***window***.***XMLHttpRequest***)  {// code for IE7+, Firefox, Chrome, Opera, Safari  xmlhttp=new XMLHttpRequest();  }  else  {// code for IE6, IE5  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  }  xmlhttp.open("GET","ForgetServlet?usertel="+usertel+"&&captcha="+captcha + "&&userid=" + userid,true);   //3.发送请求  xmlhttp.send();  // 4.回调函数  xmlhttp.onreadystatechange=function()  {  //判断readyState就绪状态是否为4，判断status响应状态码是否为200  if (xmlhttp.readyState==4 )  {  if(xmlhttp.status==200){  //获取服务器的响应结果  var responseText = xmlhttp.responseText;  document.querySelector("#showWarnTip").innerText = responseText;  }  }  }  }  ***window***.addEventListener("load",function (){  var getCodeText = ***document***.querySelector("#getCodeText");  getCodeText.addEventListener("click",checktelcaptcha); // 获取动态码  var captchaImg = ***document***.querySelector("#captcha-img");  captchaImg.addEventListener("click",freshVCode); // 刷新验证码  var newpsw1 = ***document***.querySelector("#newpsw1");  var newpsw2 = ***document***.querySelector("#newpsw2"); })  从上面可以看出，只要点击“动态获取验证码”，就会调用异步Ajax请求，把输入的用户名、手机号/邮箱、图片验证码当做参数传递给处理请求的servlet中，在servlet中接受传入的参数并调用数据库业务进行逻辑判断、调用API进行验证码发送。  其实天气的获取也可以用Ajax，如下是我改写的Ajax获取天气，首先创建请求对象，初始化XMLHttpRequest；然后构造请求头，指定城市和key；然后使用open向API发送get请求；最后判断状态是否为4/200，用JSON.parse(rqst.response);得到返回的json数据：  let weather = document.getElementById("weather")  weather.addEventListener("click", function (e) {      // Step1 创建请求对象      let rqst;      if (window.XMLHttpRequest) rqst = new XMLHttpRequest();      else rqst = new ActiveXObject("Microsoft.XMLHTTP");      // Step2 设置请求对象的请求行/请求头      const myKey = "91661a899863b23d58673be0e52e43a5"      let myCity = "430104"      const api = "https://restapi.amap.com/v3/weather/weatherInfo?"      let url = api + `?key=${myKey}` + `&city=${myCity}`      rqst.open('GET', url, true)      // Step3 设置回调函数      rqst.onreadystatechange = () => {          if (rqst.readyState === 4 && rqst.status === 200) {              const rsps = JSON.parse(rqst.response);              console.log(rsps)              const weatherData = rsps.lives[0]              // console.log(weatherData)              let ul = document.createElement("ul")              for (let item in weatherData) {                  // console.log(item, weatherData[item])                  let li = document.createElement('li')                  li.innerText = item + " : " + weatherData[item]                  ul.appendChild(li)              }              document.body.appendChild(ul)          }      }      // Step4 发送请求（请求体作为参数, Get方法则为null）      rqst.send(null)  })  let map = document.getElementById("map")  受赵启涵同学的启发，我又用Ajax实现了地图的显示。地图图片是二进制数据，所以需要用到Blob对象，Blob（Binary Large Object）表示二进制类型的大对象。在数据库管理系统中，将二进制数据存储为一个单一个体的集合。Blob 通常是影像、声音或多媒体文件，在 JavaScript 中 Blob 类型的对象表示不可变的类似文件对象的原始数据。此外，要将从API获取的二进制在html中显示出来，此时需要借用URL.createObjectURL  把传入的二进制图片数据生成本地url，用img标签显示。  let map = document.getElementById("map")  map.addEventListener("click", function (e) {      // Step1 创建请求对象      let rqst;      if (window.XMLHttpRequest) rqst = new XMLHttpRequest();      else rqst = new ActiveXObject("Microsoft.XMLHTTP");      // Step2 设置请求对象的请求行/请求头      const myKey = "91661a899863b23d58673be0e52e43a5"      let url = "https://restapi.amap.com/v3/staticmap?location=112.941658,28.149713&zoom=13&key=91661a899863b23d58673be0e52e43a5"      rqst.responseType = 'blob'      rqst.open('GET', url, true)      // Step3 设置回调函数      rqst.onreadystatechange = () => {          if (rqst.readyState === 4 && rqst.status === 200) {              //let blob = new Blob([rqst.response], { type: "image/png" });              let img = document.createElement("img");              img.src = URL.createObjectURL(rqst.response);              document.body.appendChild(img);          }      }      rqst.send(null)  })    通过实验中涉及到的三个Ajax案例，我感觉我已经摸清楚Ajax的一般流程了：  Step1：创建请求对象  Step2：设置url请求，可以是api，也可以是servlet。可以传参、设置token  Step3：发送Get请求到前面的url  Step4：设置回调函数，当rqst.readyState === 4 && rqst.status === 200时进行dom操作  Step5：发送请求  **3.4 Jsp和javaScript**  这个章节首先介绍实验中jsp中是如何使用servlet传递的参数的，然后总结jsp向servlet转换的方式，然后再整理一些以前没有用到过或者用的少的html操作，例如有form表单的使用、input-hidden的使用等。   1. **在jsp中使用request和session获取servlet设置的数值**   **示例1：使用requestScope，实现输错账号密码是提示错误**  <div id="formErrorTip" class="form-errorTip">  <span id="showErrorTip" class="form-error text-ellipsis">${requestScope.login\_error}</span> </div>  **示例2：使用sessionScope，实现登录后显示用户名**  <h2>${sessionScope.username},你好呀！</h2>  **刚开始学的时候还见过直接用request去获取参数的，容易和requestScope混淆，这里做一个总结：**  （1）request对象通常用来接收从客户端通过form提交过来的数据，然后在servlet或者action中用request.getParameter()的方法获取获取参数内容；  （2）而requestScope通常是在servlet和action中通过request.setAttribute()方法把数据放到request对象中供客户端获取，然后客户端获取的方法就是requestScope.getAttribute()。  （1）getParameter()： 可以获取从URL传递过来的String参数 提示：对于post和get提交的表单，我们都可以通过request.getParameter()来获取表单的参数值。 当我们进行一个请求重定向（redirect）的时候，可以通过request.setParameter()设置要传递的参数，而如果用request.setAttribute()在重定向页面取不到传递的参数值；  （2）getAttribute()： 获取的是一个对象（Object） 当我使用请求转发（forward）的时候，可以使用request.setAttribute()传递参数；  （4）requestScope: 是EL表达式中的一个隐含对象，类似request，如${requestScope.username }表示从request域中获取username属性对应的值，相当于request.getAttribute(“username”);  结论就是：  **①request.getParameter(“username”)用在服务器获取页面或客户端的内容，这些内容都是String的；**  **②request.getAttribute(“username”)等价于${requestScope.username}，一般是从服务器传递数据到页面，在页面中获取服务器保存在其中的数据内容。**   1. **jsp转向servlet的方式**   **A.**使用form表单。method：post；action：需要提交的Servlet类路径；input中设置name属性，Servlet接受name参数。  jsp文件：   <form class="loginFromClass" method="post" id="pwdFromId" action="${pageContext.request.contextPath}/LoginServlet">     ...      <input id="userid" name="userid" type="text" placeholder="请输入学号/工号" title="请输入学号/工号" value="">      <input id="password" name="password" type="password" placeholder="请输入密码" title="请输入密码">  </form>  Servlet文件：   String userid = request.getParameter("userid");  String password = request.getParameter("password");  **B.** img中使用src：例如图片验证码  例如在显示验证码的部分，jsp中的img通过指定src到Servlet文件，便可以获得加工后的验证码  jsp：   <img class="captcha-img" id = "captcha-img" src="/CreateCode">  Servlet：   @WebServlet(name = "CreateCode", value = "/CreateCode")  public class CreateCode extends HttpServlet {      @Override      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {          // 1.创建验证码图片，保存于本地内存          ValidateCode code = new ValidateCode(200,80,4,20);          String codes = code.getCode();          System.out.println("正确的验证码为：" + codes);          // 将验证码存储在session中的codes属性, 后期匹配校验时用到          HttpSession session = request.getSession();          session.setAttribute("codes",codes);          // 2.输出内存中的图片          code.write(response.getOutputStream());     }  ​      @Override      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {          doGet(request,response);     }  }  **C.**使用JavaScript构造http请求到Servlet文件(Ajax)：例如-点击获得验证码  批注：前面可以使用form配合submit时间jsp到servlet的数据传输，但是有时并不需要提交表单，例如点击“获取验证码”的按钮即可给手机发送验证码。此时可以使用XMLHttpRequest构造html请求，然后手动把Jsp中的参数发给Servlet类。XMLHttpRequest 用于在后台与服务器交换数据，教程：<http://c.biancheng.net/view/5995.html>  Jsp：   1. 输入手机号和邮箱  <input type="text" id="usertel" name="usertel" autocomplete="off" placeholder="请输入手机号/邮箱" title="请输入手机号/邮箱" value="">  2. 动态验证码  <input type="text" id="captcha" name="captcha" placeholder="请输入验证码" title="请输入验证码" maxlength="10">  <img class="captcha-img" id = "captcha-img" src="/CreateCode">  3. 获取验证码、输入验证码  <input type="text" id="DCode" name="DCode" placeholder="请输入短信或邮箱动态码" title="请输入短信或邮箱动态码" maxlength="10">  <a class="get-code" href="javascript:void(0)"><em id = "getCodeText" class="getCodeText">获取动态码</em></a>  JavaScript：   function checktelcaptcha(){      var userid = document.querySelector("#userid").value;      var usertel = document.querySelector("#usertel").value;      var captcha = document.querySelector("#captcha").value;      console.log(usertel + " " + captcha);      // 送异步请求      // 1.创建核心对象      var xmlhttp;      if (window.XMLHttpRequest)     {// code for IE7+, Firefox, Chrome, Opera, Safari          xmlhttp=new XMLHttpRequest();     }      else     {// code for IE6, IE5          xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");     }  ​      xmlhttp.open("GET","ForgetServlet?usertel="+usertel+"&&captcha="+captcha + "&&userid=" + userid,true);  ​      //3.发送请求      xmlhttp.send();  ​      xmlhttp.onreadystatechange=function()     {          //判断readyState就绪状态是否为4，判断status响应状态码是否为200          if (xmlhttp.readyState==4 )         {              if(xmlhttp.status==200){                  //获取服务器的响应结果                  var responseText = xmlhttp.responseText;                  document.querySelector("#showWarnTip").innerText = responseText;             }  ​         }     }  ​  }  Servlet：   // 收参  String userid = request.getParameter("userid");  String usertel = request.getParameter("usertel");  String captcha = request.getParameter("captcha");  **D.**在Jsp中使用href跳转到Servlet：例如-查询用户  Jsp1：   <div class = "bt"><a href="${pageContext.request.contextPath}/AdminServlet">查询用户</a></div>  Servlet：   ...从数据库请求数据，存储于json  HttpSession session = request.getSession();  session.setAttribute("json",json);  session.setAttribute("users",user);  request.getRequestDispatcher("/admin.jsp").forward(request,response);  Jsp2：    <c:forEach items="${sessionScope.users}" var="user" varStatus="s">                  <tr>                      <td>${user.userid}</td>                      <td>${user.username}</td>                      <td>${user.password}</td>                      <td>${user.tel}</td>                      <td>${user.email}</td>                      <td>                          <a class="btn btn-default btn-sm" href="${pageContext.request.contextPath}/UpdateUserF?userid=${user.userid}" >修改</a>&nbsp;                          <a class="btn btn-default btn-sm" href="${pageContext.request.contextPath}/DeleteServlet?userid=${user.userid}">删除</a>                      </td>                  </tr>  </c:forEach>   1. **input-hidden的使用**   Input里另外加一个 type=hidden 的 input; onsubmit 的时候, 加密密码放入hidden字段; 服务器用此hidden字段做密码验证; 同时把以前的password字段 disable=true(则password字段不提交)。  首先明文密码肯定不能上传, 不然md5就没意义了; 每次登录应该产生并记录一个salt, 前端md5(password+salt), 服务器也加入salt做验证，这样确保每次 md5产生值不同, 即使攻击者截获了 某个md5值, 也无法用于下次登录。  举例：<input type="hidden" name="pwd1" id="pwd1"/>  **3.5展示java的数据库操作**  **注：本章没有个人总结，可快速略去。**  **用户数据库操作：**  public interface UserDao { // 操作数据库的类  public int delete(String userid);  public int update(String userid,String password);  public int updatename(String userid,String username);  public int updateTel(String userid,String tel);  public int updateEmail(String userid,String email);   public int add(User user); // 注册一个用户  public User select(String userid) throws SQLException;  public List<User> selectAll();  public User selectByTel(String tel);  public User selectByEmail(String email);  public List<User> selectPart(User user); }  **管理员数据库操作：**  public interface AdminDao {  public Admin select(String adminid); }  **用户数据库实现：**  public class UserDaoImpl implements UserDao {  private QueryRunner queryRunner = new QueryRunner();   @Override  public int delete(String userid) {  try {  queryRunner.update(DbUtils.*getConnection*(),  "DELETE FROM user where userid = ?",userid);  } catch (SQLException e) {  e.printStackTrace();  }  return 0;  }   @Override  public int update(String userid, String password) {  User user = select(userid);  int res = 0;  try {   res = queryRunner.update(DbUtils.*getConnection*(),"update user set password=? where userid = ?",password,userid);  } catch (SQLException e) {  e.printStackTrace();  }  return res;  }   @Override  public int updatename(String userid, String username) {  User user = select(userid);  try {  queryRunner.update(DbUtils.*getConnection*(),"update user set username=? where userid = ?",username,userid);  } catch (SQLException e) {  e.printStackTrace();  }  return 0;  }   @Override  public int updateTel(String userid, String tel) {  User user = select(userid);  try {  queryRunner.update(DbUtils.*getConnection*(),"update user set tel=? where userid = ?",tel,userid);  } catch (SQLException e) {  e.printStackTrace();  }  return 0;  }   @Override  public int updateEmail(String userid, String email) {  User user = select(userid);  try {  queryRunner.update(DbUtils.*getConnection*(),"update user set email=? where userid = ?",email,userid);  } catch (SQLException e) {  e.printStackTrace();  }  return 0;  }     @Override  public int add(User user) {  int res = 0;  try {  res = queryRunner.update(DbUtils.*getConnection*(),"INSERT INTO user values(?,?,?,?,?)",user.getUserid(),user.getUsername(),user.getPassword(),user.getTel(),user.getEmail());  System.*out*.println("参数："+user.getUserid()+" "+user.getUsername()+" "+user.getPassword()+" "+user.getTel()+" "+user.getEmail());  } catch (SQLException e) {  e.printStackTrace();  }   return res;  }   @Override  public User select(String userid){  User user = new User();// 能够执行  try {  user = queryRunner.query(DbUtils.*getConnection*(),"select \* from user where userid=?",new BeanHandler<User>(User.class),userid);  } catch (SQLException e) {  e.printStackTrace();  }  return user;  }   @Override  public List<User> selectAll() {  List<User> users = null;  try {  users = queryRunner.query(DbUtils.*getConnection*(),"select \* from user;",new BeanListHandler<User>(User.class));  } catch (SQLException e) {  e.printStackTrace();  }  return users;  }   @Override  public User selectByTel(String tel) {  User user = new User();// 能够执行  try {  user = queryRunner.query(DbUtils.*getConnection*(),"select \* from user where tel=?",new BeanHandler<User>(User.class),tel);  } catch (SQLException e) {  e.printStackTrace();  }  return user;  }   @Override  public User selectByEmail(String email) {  User user = new User();// 能够执行  try {  user = queryRunner.query(DbUtils.*getConnection*(),"select \* from user where email=?",new BeanHandler<User>(User.class),email);  } catch (SQLException e) {  e.printStackTrace();  }  return user;   }   @Override  public List<User> selectPart(User user) {  List<User> users = null;  String sql = "select \* from user where";  sql += " userid like \"%" + user.getUserid() + "%\" ";  sql += "and username like \"%" + user.getUsername() + "%\" ";  sql += "and tel like \"%" + user.getTel() + "%\" ";  sql += "and email like \"%" + user.getEmail() + "%\" ";   try {  users = queryRunner.query(DbUtils.*getConnection*(),sql,new BeanListHandler<User>(User.class));  } catch (SQLException e) {  e.printStackTrace();  }   return users;  } }  **用户实体：**  public class User {  private String userid;  private String username;  private String password;  private String tel;  private String email;  public User(){  }  public User(String userid, String username, String password, String tel, String email) {  this.userid = userid;  this.username = username;  this.password = password;  this.tel = tel;  this.email = email;  }   public String getUserid() {  return userid;  }   public void setUserid(String userid) {  this.userid = userid;  }   public String getUsername() {  return username;  }   public void setUsername(String username) {  this.username = username;  }   public String getPassword() {  return password;  }   public void setPassword(String password) {  this.password = password;  }   public String getTel() {  return tel;  }   public void setTel(String tel) {  this.tel = tel;  }   public String getEmail() {  return email;  }   public void setEmail(String email) {  this.email = email;  }   @Override  public String toString() {  return "User{" +  "userid='" + userid + '\'' +  ", username='" + username + '\'' +  ", password='" + password + '\'' +  ", tel='" + tel + '\'' +  ", email='" + email + '\'' +  '}';  } }  **用户操作：**  public interface UserService {  public User login(String userid, String password);  public List<User> showAllUser();  public List<User> showPartUser(User user);  public User haveEmail(String email);  public User haveTel(String tel);  public User haveId(String userid);  public int updatePsw(String userid,String password);  public int updateUser(User user);  public int add(User user);  public void delete(String userid);  **用户操作的实现：**  public class UserServiceImpl implements UserService {  private UserDao userDao = new UserDaoImpl();   @Override  public User login(String userid, String password) { // 登录函数，实现登录业务功能，检测输入的用户id和密码是否正确  DbUtils.*begin*();  User resUser = null;  try {  User user = userDao.select(userid);  if(user!=null){ // 如果存在该用户id  if(user.getPassword().equals(password)) // 且密码正确  resUser = user;  }  DbUtils.*commit*(); // 整个事务完成，进行事务提交  } catch (SQLException e) {  DbUtils.*rollback*(); // 事务出现异常，进行回滚  e.printStackTrace();  }  return resUser;  }   @Override  public List<User> showAllUser() {  DbUtils.*begin*();  List<User> users = null;  users = userDao.selectAll();   return users;  }   @Override  public List<User> showPartUser(User user) {  DbUtils.*begin*();  List<User> users = null;  users = userDao.selectPart(user);   return users;  }   @Override  public User haveEmail(String email) {  DbUtils.*begin*();  User user = null;  user = userDao.selectByEmail(email); // 如果存在，就存储在user中，如果不存在，就为空  DbUtils.*commit*(); // 整个事务完成，进行事务提交  return user;  }   @Override  public User haveTel(String tel) {  DbUtils.*begin*();  User user = null;  user = userDao.selectByTel(tel); // 如果存在，就存储在user中，如果不存在，就为空  DbUtils.*commit*(); // 整个事务完成，进行事务提交  return user;  }  public User haveId(String userid){  DbUtils.*begin*();  User user = null;  try {  user = userDao.select(userid); // 如果存在，就存储在user中，如果不存在，就为空  } catch (SQLException e) {  e.printStackTrace();  }  DbUtils.*commit*(); // 整个事务完成，进行事务提交  return user;  }   @Override  public int updatePsw(String userid, String password) {  DbUtils.*begin*();  int res = userDao.update(userid,password);  DbUtils.*commit*(); // 整个事务完成，进行事务提交  return res;  }   @Override  public int updateUser(User user) {  DbUtils.*begin*();  String userid = user.getUserid();  userDao.update(userid, user.getPassword());  userDao.updatename(userid, user.getUsername());  userDao.updateTel(userid, user.getTel());  userDao.updateEmail(userid, user.getEmail());  DbUtils.*commit*(); // 整个事务完成，进行事务提交   return 0;  }   @Override  public int add(User user) {  DbUtils.*begin*();  int res = userDao.add(user);  DbUtils.*commit*(); // 整个事务完成，进行事务提交  return res;  }   @Override  public void delete(String userid) {  DbUtils.*begin*();  userDao.delete(userid);  DbUtils.*commit*(); // 整个事务完成，进行事务提交  }  }   1. **实验过程中的发现与收获，未解决或需进一步解决的问题：**   **注：具体的实现细节总结已经在前面的板块用红字标出，此处只说明抽象的收获。**  **收获:**   1. **第一次动手实现javaWeb项目，这是一次全栈的开发经历，受益匪浅。** 2. **通过多个案例学会了Servlet的原理、作用、写法；学会了如何在servlet中使用数据库和外接API。** 3. **通过案例理解了两组混淆概念：request和requestScope的区别；转发和重定向的区别。** 4. **学会了Ajax的一般写法，学会了如何使用Ajax请求API获取信息。** 5. **学会了jsp页面中使用servlet，servlet给jsp传值，servlet给servlet传值的方法。** 6. **理解了session的作用。** 7. **理解了input-hidden的作用。**   **问题:**  虽然实现了功能要求，但是有些细节没有做好，比如没有使用正则表达式判断是否为中国手机号，没有发送验证码后的倒计时动画。此外，在设计css的时候我依旧感到有些吃力，有时候需要边翻文档边写，而且还要尝试很多次，为此我在网上买了一门外国人讲的css课程，寒假好好学习一下。  这是我们最后一次实验了，回想这学期做过的五个实验，我感慨颇多：  从个人简历中我学会了基本html语法和css的布局方法，尤其是掌握了flex和grid布局的使用......  从日历中我学会了怎么用js生成dom元素并插入dom树中，学会了怎么用css实现隐藏效果，学会了js内置日期函数的用法......  从猜数字中我又强化了js对dom元素的操纵，也学会了label和input一起使用的方法......  从CSU官网中我学会了如何用js实现动画效果，主要代表作就是是连续轮播图还是间隔轮播图，以及下拉框的实现啦，此外我还掌握了一个功能完整的网站的布局设计方法，再次强化了flex实现复杂布局的方法......  从CSU登录界面中我学会了如何使用servlet和Ajax与前端jsp页面进行交互，以及如何实现数据库的操作、实体的构建......  学习之路没有终点，我会继续努力学习，努力实现自己的目标！正如我在第一份实验报告中最后哪里写到的“我感觉html太弱了，我要赶快自学js了”，现在我依然想说：我感觉前端三件套+servlet+Ajax太弱了，我要赶紧自学React/Vue+Django了。  最后给老师提个建议，下半年的人工智能可以安排一个神经网络的实践实验，不仅仅是在写好的老系统上简单的进行反向传播的可视化（虽然手算梯度下降在机器学习课程考试中也是10-15分的必考题），而是加上手写数字/莺尾花识别这种基础的实践实验，就用简单的全连接网络实现就行，因为如果人工智能课做了这个实验，后续的机器学习再学到相关知识的时候会非常高效且清晰。 | | | | | | | | |
| **指导老师评阅意见** | | | | | | | | |
|  | | | | | | | | |
| 指导老师： 年 月 日 | | | | | | | | |