



**CHANGJI COLLEGE**

**JavaWeb程序设计**

期 末 大 作 业

**系 （院）： 信息工程学院**

**专 业 ： 网络工程**

**班 级 ：**

**姓 名 ：**

**学 号 ：**

**授课老师： 车 全 江**

**开课时间：** **2023~2024年春季学期**

**大作业成绩：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 系（院） | 信息工程学院 | | 专业 | 网络工程 | 班级 |  |
| 学生姓名 |  | | 学号 |  | 授课老师 | 车全江 |
| 大作业课题 | | **XX数据管理系统设计与实现** | | | | |
| 大作业需求及相应评分标准 | | （1）要求完成一个数据管理系统，功能要求如下：  登录功能：通过js需要完成前端的校验，通过cookie会话技术实现“记住密码”功能；  注册功能：通过js需要完成前端的校验，通过session会话技术实现“验证码”功能；  过滤功能：通过Filter技术实现过滤功能，使得用户必须完成登录之后才能够有权限访问其它页面（除登录和注册页面除外）；  添加功能：在数据显示主页面，通过前端页面及后端逻辑实现一行数据的添加；  修改功能：在数据显示主页面，通过前端页面及后端逻辑实现一行数据的修改；  删除功能：在数据显示主页面，通过前端页面及后端逻辑实现一行数据的删除；  查询所有：在数据显示主页面，通过前端页面及后端逻辑实现查询所有的数据；   1. 具体评分标准如下：  |  |  |  | | --- | --- | --- | | 任务点 | 评分点 | 评分（满分） | | 登录功能 | 完成登录前端页面及js前端校验 | 8 | | 完成登录的后端逻辑 | 8 | | 完成"记住密码"功能 | 6 | | 注册功能 | 完成注册前端页面及js前端校验 | 8 | | 完成登录的后端逻辑 | 8 | | 完成"验证码"功能 | 6 | | 过滤功能 | 完成过滤功能 | 6 | | 数据显示主页面 | 完成查询所有功能（包括前端+后端） | 10 | | 完成添加一条数据功能（包括前端+后端） | 10 | | 完成删除一条数据功能（包括前端+后端） | 10 | | 完成修改一条数据功能（包括前端+后端） | 10 | | 使用Vue框架、Element UI框架、AJAX技术 | | 10 | | 总分 | | 100 |   **批注：（内容自定义，每位同学不能雷同，雷同视为抄袭，成绩为0分）** | | | | |
| **大作业内容**  **大作业内容**  **大作业内容**  **大作业内容** | | 1. 登录功能   1.1 完成登录前端页面及js前端校验（附代码+截图）（8分）   |  | | --- | | 批注：代码要求Times New Roman字体，字体大小为小四号，单倍行距（正式打印，删除此部分删除）；  页面设计 + 样式代码：  <template>  <div id="loginPage">  <div class="login-header">  <el-image :src="Logo" class="logo-image" />  <h2 style="margin-left: 20px; display: flex; align-items: center">  学生数据管理系统 登录  </h2>  </div>  <div class="login-box">  <el-form  label-position="right"  label-width="auto"  :model="loginParams"  style="min-width: 360px; max-width: 460px"  >  <el-form-item label="手机号">  <el-input v-model="loginParams.phone" placeholder="请输入手机号" />  </el-form-item>  <el-form-item label="密码">  <el-input  v-model="loginParams.password"  placeholder="请输入密码"  type="password"  show-password  />  </el-form-item>  <el-form-item>  <el-row justify="space-between">  <el-checkbox  style="margin-left: 70px; margin-right: 70px"  v-model="remberMe"  label="记住我"  size="large"  />  <el-text  >没有账号？前往  <el-link style="color: blue" @click="toRegister">注册</el-link>  一个吧！</el-text  >  </el-row>  </el-form-item>  </el-form>  </div>  <div style="display: flex; justify-content: center">  <el-button color="#626aef" @click="doLogin"> 登录</el-button>  </div>  </div>  </template>  <style lang="less">  #loginPage {  position: absolute;  left: 0;  top: 0;  right: 0;  bottom: 0;  background: url("@/assets/bg.png") no-repeat center center fixed;  background-size: cover;  }  #loginPage .logo-image {  width: 50px;  cursor: pointer;  }  #loginPage .login-header {  margin-top: 60px;  display: flex;  justify-content: center;  }  #loginPage .login-box {  display: flex;  justify-content: center;  margin-top: 20px;  }  #loginPage .el-form-item\_\_label {  color: black;  font-size: larger;  font-weight: 500;  }  .el-checkbox\_\_input.is-checked + .el-checkbox\_\_label {  color: black;  }  </style>  页面效果：    登录逻辑代码：  const doLogin = () => {  if (loginParams.value.phone && loginParams.value.password) {  login(loginParams.value)  .then((res) => {  if (res.code === 0) {  const data = res.data;  const user = data["user"] as User;  userStore.setUserInfo(user);  localStorage.setItem("token", data["token"]);  if (remberMe.value) {  Cookies.set("phone", loginParams.value.phone, { expires: 7 });  Cookies.set("password", loginParams.value.password, { expires: 7 });  } else {  Cookies.remove("phone");  Cookies.remove("password");  }  ElMessage({  type: "success",  message: "登录成功",  });  if (user.role === "ADMIN") {  router.push({  path: "/",  replace: true,  });  } else {  router.push({  path: "/demo",  replace: true,  });  }  } else {  ElMessage({  type: "error",  message: res,  });  }  })  .catch((err) => {  ElMessage({  type: "error",  message: err,  });  console.log(err);  });  } else {  ElMessage({  type: "error",  message: "手机号或密码为空",  });  }  }; |   1.2 完成登录的后端逻辑（附代码）（8分）   |  | | --- | | @Override  public BaseResponse login(LoginDto loginDto) {  String phone = loginDto.getPhone();  User userByPhone = userMapper.selectOne(new QueryWrapper<User>().eq("phone", phone));  if(userByPhone == null){  return ResultUtils.error(ErrorCode.PARAMS\_ERROR,"用户不存在");  }  // 验证用户名和密码 验证失败时会抛异常  try {  authenticationManager.authenticate(  new UsernamePasswordAuthenticationToken(  phone,  loginDto.getPassword()  )  );  } catch (Exception e) {  return ResultUtils.error(ErrorCode.OPERATION\_ERROR, "用户名或密码错误");  }  String token = jwtUtil.generateToken(userByPhone);  Map<String, Object> mp = new HashMap<>();  mp.put("token", token);  mp.put("user", userByPhone);  return ResultUtils.success(mp);  } |   1.3 完成"记住密码"功能（附代码+截图）（6分）   |  | | --- | | 组件被挂载到DOM时从Cookies中获取登录表单数据  onMounted(() => {  // 从本地Cookies中获取登录信息  const savedPhone = Cookies.get("phone");  const savedPassword = Cookies.get("password");  if (savedPhone && savedPassword) {  loginParams.value.phone = savedPhone;  loginParams.value.password = savedPassword;  remberMe.value = true;  }  });  登录成功时，根据是否记住密码对登录数据进行保存  // 表示登录成功  if (res.code === 0) {  const data = res.data;  const user = data["user"] as User;  userStore.setUserInfo(user);  localStorage.setItem("token", data["token"]);  if (remberMe.value) {  Cookies.set("phone", loginParams.value.phone, { expires: 7 });  Cookies.set("password", loginParams.value.password, { expires: 7 });  } else {  Cookies.remove("phone");  Cookies.remove("password");  }  ElMessage({  type: "success",  message: "登录成功",  });  if (user.role === "ADMIN") {  router.push({  path: "/",  replace: true,  });  } else {  router.push({  path: "/demo",  replace: true,  });  }  } else {  ElMessage({  type: "error",  message: res,  });  } |  1. 注册功能   2.1 完成注册前端页面及js前端校验（附代码+截图）（8分）   |  | | --- | | 前端页面+样式代码  <template>  <div id="registerPage">  <div class="register-header">  <el-image :src="Logo" class="logo-image" />  <h2 style="margin-left: 20px; display: flex; align-items: center">  学生数据管理系统 注册  </h2>  </div>  <div class="register-box">  <el-form  label-position="right"  label-width="auto"  :model="registerUserParam"  style="min-width: 360px; max-width: 460px"  >  <el-form-item label="手机号">  <el-input  v-model="registerUserParam.phone"  placeholder="请输入手机号"  />  </el-form-item>  <el-form-item label="密码">  <el-input  v-model="registerUserParam.password"  placeholder="请输入密码"  type="password"  show-password  />  </el-form-item>  <el-form-item label="验证码">  <div style="display: flex; align-items: center">  <el-input  v-model="code"  placeholder="请输入验证码"  style="flex: 1; margin-right: 10px"  />  <el-image  :src="codeSrc"  alt="验证码"  style="width: 100px; height: 30px; cursor: pointer"  @click="refreshCode"  />  </div>  </el-form-item>  <el-form-item style="display: flex">  <el-text style="text-align: right; width: 100%"  >已有账号？前往  <el-link  style="color: blue; vertical-align: baseline"  @click="toLogin"  >登录</el-link  >  吧！</el-text  >  </el-form-item>  </el-form>  </div>  <div style="display: flex; justify-content: center">  <el-button color="#626aef" @click="toRegister"> 注册</el-button>  </div>  </div>  </template>  <style lang="less">  #registerPage {  position: absolute;  left: 0;  top: 0;  right: 0;  bottom: 0;  background: url("@/assets/bg.png") no-repeat center center fixed;  background-size: cover;  }  #registerPage .logo-image {  width: 50px;  cursor: pointer;  }  #registerPage .register-header {  margin-top: 60px;  display: flex;  justify-content: center;  }  #registerPage .register-box {  display: flex;  justify-content: center;  margin-top: 20px;  }  #registerPage .el-form-item\_\_label {  color: black;  font-size: larger;  font-weight: 500;  }  </style>  效果图：    前端登录逻辑代码：  onst doRegister = () => {  if (  registerUserParam.value.phone &&  registerUserParam.value.password &&  code.value  ) {  register(registerUserParam.value, code.value)  .then((res) => {  if (res.code === 0) {  ElMessage({  type: "success",  message: res.data,  });  router.push({  path: "/user/login",  replace: true,  });  } else {  ElMessage({  type: "error",  message: res.messsage,  });  }  })  .catch((err) => {});  } else {  ElMessage({  type: "error",  message: "请填写信息",  });  }  }; |   2.2 完成注册的后端逻辑（附代码）（8分）   |  | | --- | | @Transactional  @Override  public BaseResponse register(RegisterDto registerDto) {  String phone = registerDto.getPhone();  User storeUser = userMapper.selectOne(new QueryWrapper<User>().eq("phone", phone));  if(storeUser != null){  return ResultUtils.error(ErrorCode.OPERATION\_ERROR,"手机号已注册");  }  String password = registerDto.getPassword();  long currTime = System.currentTimeMillis();  User user = new User();  user.setRole(Role.STUDENT.name());  user.setPhone(phone);  user.setPassword(passwordEncoder.encode(password));  user.setUpdateTime(new Date(currTime));  user.setCreateTime(new Date(currTime));  userMapper.insert(user);  return ResultUtils.success("注册成功");  } |   2.3 完成"验证码"功能（附代码+截图）（6分）   |  | | --- | | 后端使用kaptcha生成图形验证码.  Kaptcha配置类：  package com.example.config;  import com.google.code.kaptcha.impl.DefaultKaptcha;  import com.google.code.kaptcha.util.Config;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import java.util.Properties;  @Configuration  public class CaptchaConfig {  public final static String CAPTCHA\_SESSION\_KEY = "captcha\_key";  @Bean  public DefaultKaptcha getKaptchaBean(){  DefaultKaptcha defaultKaptcha = new DefaultKaptcha();  Properties properties = new Properties();  properties.setProperty("kaptcha.border", "no");  properties.setProperty("kaptcha.border.color", "105,179,90");  properties.setProperty("kaptcha.textproducer.font.color", "blue");  properties.setProperty("kaptcha.image.width", "100");  properties.setProperty("kaptcha.image.height", "54");  properties.setProperty("kaptcha.session.key", "code");  properties.setProperty("kaptcha.textproducer.char.length", "4");  properties.setProperty("kaptcha.textproducer.font.names", "宋体,楷体,微软雅黑");  properties.setProperty("kaptcha.textproducer.font.size","35");  defaultKaptcha.setConfig(new Config(properties));  return defaultKaptcha;  }  }  验证码生成接口：  package com.example.controller;  import com.example.config.CaptchaConfig;  import com.google.code.kaptcha.impl.DefaultKaptcha;  import lombok.RequiredArgsConstructor;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.RestController;  import javax.imageio.ImageIO;  import javax.servlet.ServletOutputStream;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;  import java.awt.image.BufferedImage;  import java.io.IOException;  @RestController  @RequiredArgsConstructor  public class CaptchaController {  private final DefaultKaptcha captchaProducer;  @GetMapping(value="/code")  public void kaptcha(HttpSession session, HttpServletResponse response) throws IOException {  response.setDateHeader("Expires", 0);  response.setHeader("Cache-Control", "no-store, no-cache, must-revalidate");  response.addHeader("Cache-Control", "post-check=0, pre-check=0");  response.setHeader("Pragma", "no-cache");  response.setContentType("image/jpeg");  String capText = captchaProducer.createText();  session.setAttribute(CaptchaConfig.CAPTCHA\_SESSION\_KEY,capText);  try(ServletOutputStream out = response.getOutputStream()){  BufferedImage bufferedImage = captchaProducer.createImage(capText);  ImageIO.write(bufferedImage,"jpg",out);  out.flush();  }  }  }  图形验证码过滤器：该过滤器会拦截注册的请求，并在进入handler前对存储在Session里的code与用户输入的code进行比对和校验  package com.example.handler;  import com.example.config.CaptchaConfig;  import org.springframework.security.web.authentication.session.SessionAuthenticationException;  import org.springframework.stereotype.Component;  import org.springframework.util.StringUtils;  import org.springframework.web.filter.OncePerRequestFilter;  import javax.servlet.FilterChain;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import java.io.IOException;  @Component  public class CaptchaCodeFilter extends OncePerRequestFilter {  @Override  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  if("/user/register".equals(request.getRequestURI())&&request.getMethod().equalsIgnoreCase("post")){  validate(request);  }  filterChain.doFilter(request,response);  }  private void validate(HttpServletRequest request) throws IOException {  //用户输入的验证码  String userCode = request.getParameter("code");  //session里的验证码对象  String code = (String) request.getSession().getAttribute(CaptchaConfig.CAPTCHA\_SESSION\_KEY);  if(StringUtils.isEmpty(userCode)){  throw new SessionAuthenticationException("验证码不能为空");  }  if(code==null){  throw new SessionAuthenticationException("验证码不存在");  }  if(!code.equalsIgnoreCase(userCode)){  throw new SessionAuthenticationException("验证码错误");  }  }  }  前端获取图形验证码：  import { BASE\_URL } from "@/http/config";  import axios from "axios";  import { ApiUrl } from "../constant";  export const verifyCode = async (): Promise<string> => {  try {  const response = await axios.get(BASE\_URL + ApiUrl.code, {  responseType: "arraybuffer",  });  const imageBlob = new Blob([response.data], { type: "image/jpeg" });  const imageUrl = URL.createObjectURL(imageBlob);  return imageUrl;  } catch (error) {  return "";  }  }; |  1. 过滤功能   3.1 完成过滤功能（附代码+截图）（6分）   |  | | --- | | JWT过滤器：该过滤器用于校验请求头中的token，当Token过期或Token不正确时，会及时返回403无权限错误；当Token检验通过时，会利用Token中的数据查询出对应的UserDetail并保存到SecurityContextHolder中让Spring Security知道该UserDetail已通过了校验。  package com.example.handler;  import com.example.utils.JwtUtil;  import lombok.NonNull;  import lombok.RequiredArgsConstructor;  import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  import org.springframework.security.core.context.SecurityContextHolder;  import org.springframework.security.core.userdetails.UserDetails;  import org.springframework.security.core.userdetails.UserDetailsService;  import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;  import org.springframework.stereotype.Component;  import org.springframework.web.filter.OncePerRequestFilter;  import javax.servlet.FilterChain;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import java.io.IOException;  @RequiredArgsConstructor  @Component  public class JwtAuthenticationFilter extends OncePerRequestFilter {  private final JwtUtil jwtUtil;  private final UserDetailsService userDetailsService;  @Override  protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain filterChain) throws ServletException, IOException {  final String authHeader = request.getHeader("Authorization");  final String userEmail;  final String jwt;  if (authHeader == null || !authHeader.startsWith("Bearer ")) {  filterChain.doFilter(request, response);  return;  }  jwt = authHeader.substring(7);  userEmail = jwtUtil.extractUsername(jwt);  if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {  UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);  if (jwtUtil.isTokenValid(jwt, userDetails)) {  UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(  userDetails,  null,  userDetails.getAuthorities()  );  authToken.setDetails(  new WebAuthenticationDetailsSource().buildDetails(request)  );  SecurityContextHolder.getContext().setAuthentication(authToken);  }  }  filterChain.doFilter(request, response);  }  }  由于后端采用了Spring Security权限校验框架，因此只要Details权限通过，所有的接口访问权限的判断都可以由Security自动完成，而判断的规则则是由我们自已配置。Security的配置如下：  package com.example.config;  import com.example.handler.CaptchaCodeFilter;  import com.example.handler.JwtAuthenticationFilter;  import lombok.RequiredArgsConstructor;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  @Configuration  @RequiredArgsConstructor  @EnableWebSecurity  public class SecurityConfig extends WebSecurityConfigurerAdapter {  private final CaptchaCodeFilter captchaCodeFilter;  private final JwtAuthenticationFilter jwtAuthenticationFilter;  private final String[] ALLOW\_URL = {"/code","/user/login","/user/register"};  @Override  protected void configure(HttpSecurity http) throws Exception {  http.cors()  .and()  .csrf().disable()  .httpBasic()  .and()  .rememberMe() // 开启记住我  .and()  .authorizeRequests()  .antMatchers(ALLOW\_URL).permitAll()  .antMatchers("/student/\*\*").hasAnyAuthority("ADMIN")  .anyRequest().authenticated()  .and()  .addFilterBefore(captchaCodeFilter, UsernamePasswordAuthenticationFilter.class)  .addFilterAfter(jwtAuthenticationFilter,CaptchaCodeFilter.class);  }  }  配置类中规定：”/code”、”/user/login”、”/user/register”三个接口是公开的，也就是说不需要经过权限校验就能访问。  “/student/\*\*”接口则是要求必须由ADMIN权限才能访问。”\*\*”是一种泛解析的方式，表示以”/student/”开头的请求都满足条件，例如:”/student/1”,”/student/2”,....都要求必须有ADMIN权限才能访问。  效果图：  ADMIN用户：10000000000 123123      非ADMIN用户：12312301230 123123 |  1. 数据显示主页面   4.1 完成查询所有功能（包括前端+后端、附代码+截图）  （10分）   |  | | --- | | 后端查询的核心代码:学号、姓名模糊查询、手机号，QQ号精准查询，并且采用分页。  @Override  public BaseResponse queryStudent(QueryStdDto queryStdDto) {  int page = queryStdDto.getPage();  int pageSize = queryStdDto.getPageSize();  Page<Student> currPage = new Page<>(page, pageSize);  String number = queryStdDto.getNumber() == null ? "" : queryStdDto.getNumber();  String name = queryStdDto.getName() == null ? "" : queryStdDto.getName();  QueryWrapper<Student> wp = new QueryWrapper<>();  wp.like("number",number);  wp.like("name",name);  if(!ObjectUtils.isEmpty(queryStdDto.getQq())){  wp.eq("qq",queryStdDto.getQq());  }  if(!ObjectUtils.isEmpty(queryStdDto.getPhone())){  wp.eq("phone",queryStdDto.getPhone());  }  System.out.println("查询参数:" + queryStdDto);  Page<Student> studentsByPage = studentMapper.selectPage(currPage, wp);  return ResultUtils.success(studentsByPage);  }  前端对接代码：  export const queryStudent = async (data: QueryStdDto): Promise<any> => {  const resp = await request.get(ApiUrl.queryStd, data);  return resp;  };  管理页面调用代码：  const loadStdData = () => {  loading.value = true;  queryStudent(queryStdDto.value)  .then((res) => {  if (res.code === 0) {  const data = res.data;  total.value = data.total;  stdData.value = data.records;  } else {  ElMessage({  type: "error",  message: res.message,  });  }  })  .catch((err) => {})  .finally(() => {  loading.value = false;  });  };  学号模糊查询效果图：    姓名模糊查询效果图：    QQ精准查询效果图：    手机号码精准查询效果图： |   4.2 完成添加一条数据功能（包括前端+后端、附代码+截图）（10分）   |  | | --- | | 后端添加功能的核心代码：  @Transactional  @Override  public BaseResponse addStudent(AddStdDto addStdDto) {  Student storeStudent = studentMapper.selectOne(new QueryWrapper<Student>().eq("number", addStdDto.getNumber()));  if(storeStudent == null){  Student student = new Student();  BeanUtils.copyProperties(addStdDto,student);  long currTime = System.currentTimeMillis();  student.setCreateTime(new Date(currTime));  student.setUpdateTime(new Date(currTime));  studentMapper.insert(student);  return ResultUtils.success("插入成功");  }else {  return ResultUtils.error(ErrorCode.OPERATION\_ERROR,"学号:" + addStdDto.getName() + " 已存在");  }  }  前端对接代码：  export const addStudent = async (data: AddStdDto): Promise<any> => {  const resp = await request.post(ApiUrl.addStd, data);  return resp;  };  前端添加操作的调用代码：  const addStd = () => {  addStdDto.value.sex = selectSex.value === "男" ? 1 : 0;  addStudent(addStdDto.value)  .then((res) => {  if (res.code === 0) {  ElMessage({  type: "success",  message: res.data,  });  } else {  ElMessage({  type: "error",  message: res.message,  });  }  })  .catch((err) => {  ElMessage({  type: "error",  message: err,  });  })  .finally(() => {  isAdd.value = false;  loadStdData();  addStdDto.value.name = "";  addStdDto.value.number = "";  addStdDto.value.sex = 0;  addStdDto.value.phone = "";  addStdDto.value.qq = "";  addStdDto.value.address = "";  selectSex.value = "男";  });  };  效果图： |   4.3 完成删除一条数据功能（包括前端+后端、附代码+截图）（10分）   |  | | --- | | 后端删除功能的核心代码：  @Transactional  @Override  public BaseResponse delStudent(DelStdDto delStdDto) {  List<Long> ids = delStdDto.getIds();  int cnt = studentMapper.deleteBatchIds(ids);  return ResultUtils.success("成功删除" + cnt + "条记录");  }  前端对接代码：  export const delStudent = async (data: DelStdDto): Promise<any> => {  const resp = await request.post(ApiUrl.delStd, data);  return resp;  };  前端删除功能调用代码：  const delStd = (val: any) => {  const delStdDto = {  ids: [val],  } as DelStdDto;  delStudent(delStdDto)  .then((res) => {  console.log(res);  if (res.code === 0) {  ElMessage({  type: "success",  message: res.data,  });  loadStdData();  } else {  ElMessage({  type: "error",  message: res.message,  });  }  })  .catch((err) => {  console.log(err);  });  };  效果图： |   4.4 完成修改一条数据功能（包括前端+后端、附代码+截图）（10分）   |  |  | | --- | --- | | 后端修改功能核心代码：  @Transactional  @Override  public BaseResponse updateStudent(UpdateStdDto updateStdDto) {  long currTime = System.currentTimeMillis();  String number = updateStdDto.getNumber();  Student student = studentMapper.selectOne(new QueryWrapper<Student>().eq("number", number));  student.setQq(updateStdDto.getQq());  student.setAddress(updateStdDto.getAddress());  student.setPhone(updateStdDto.getPhone());  student.setUpdateTime(new Date(currTime));  studentMapper.updateById(student);  return ResultUtils.success("修改成功");  }  前端对接代码：  export const updateStudent = async (data: UpdateStdDto): Promise<any> => {  const resp = await request.post(ApiUrl.updateStd, data);  return resp;  };  前端修改功能调用代码：  const handleSaveInfo = () => {  updateStudent(updateStdDto.value)  .then((res) => {  console.log(res);  if (res.code === 0) {  ElMessage({  type: "success",  message: res.data,  });  } else {  ElMessage({  type: "error",  message: res.message,  });  }  })  .catch((err) => {  console.log(err);  })  .finally(() => {  isShow.value = false;  loadStdData();  });  };  效果图： |  | | | | | |