

实现过程文档、测试报告、使用说明书

项目名称：在线课程教学平台

项目负责人：郑皓天

## 目录

项目名称：在线课程教学平台 .....	1
一、 实现过程文档 .....	3
二、 测试报告 .....	21
三、 使用说明书 .....	27

# 一、实现过程文档

## 1. 后端

### ➤ 目录结构

```
|-- resources
|   |-- jdbc.properties
|-- src
|   |-- com
|       |-- szu
|           |-- se
|               |-- common
|                   |-- Result.java
|                   |-- ResultCodeEnum.java
|               |-- controller
|                   |-- AccountController.java
|                   |-- AssignmentController.java
|                   |-- BaseController.java
|                   |-- CourseController.java
|                   |-- CourseEnrollmentController.java
|                   |-- StudentController.java
|                   |-- SubmissionController.java
|                   |-- TeacherController.java
|               |-- dao
|                   |-- AccountDao.java
|                   |-- AssignmentDao.java
|                   |-- BaseDao.java
|                   |-- CourseDao.java
|                   |-- CourseEnrollmentDao.java
|                   |-- StudentDao.java
```

```
|      | |-- SubmissionDao.java
|      | |-- TeacherDao.java
|      | `-- impl
|      |     |-- AccountDaoImpl.java
|      |     |-- AssignmentDaoImpl.java
|      |     |-- CourseDaoImpl.java
|      |     |-- CourseEnrollmentDaoImpl.java
|      |     |-- StudentDaoImpl.java
|      |     |-- SubmissionDaoImpl.java
|      |     `-- TeacherDaoImpl.java
|      |-- filters
|      |   `-- CorsFilter.java
|      |-- pojo
|      |   |-- Account.java
|      |   |-- Assignment.java
|      |   |-- Course.java
|      |   |-- CourseEnrollment.java
|      |   |-- Management.java
|      |   |-- Profile.java
|      |   |-- Student.java
|      |   |-- Submission.java
|      |   |-- Teacher.java
|      |   `-- UserType.java
|      |-- service
|      |   |-- AccountService.java
|      |   |-- AssignmentService.java
|      |   |-- CourseEnrollmentService.java
|      |   |-- CourseService.java
|      |   |-- StudentService.java
```

```
|           | |-- SubmissionService.java
|           | |-- TeacherService.java
|           | `-- impl
|           |     |-- AccountServiceImpl.java
|           |     |-- AssignmentServiceImpl.java
|           |     |-- CourseEnrollmentServiceImpl.java
|           |     |-- CourseServiceImpl.java
|           |     |-- StudentServiceImpl.java
|           |     |-- SubmissionServiceImpl.java
|           |     `-- TeacherServiceImpl.java
|           |-- test
|           |   |-- TestJwtUtil.java
|           |-- util
|           |   |-- EncryptUtil.java
|           |   |-- JDBCUtil.java
|           |   |-- JwtUtil.java
|           |   `-- WebUtil.java
|-- web
|   |-- WEB-INF
|   |   |-- lib
|   |   |   |-- druid-1.1.21.jar
|   |   |   |-- jackson-annotations-2.13.2.jar
|   |   |   |-- jackson-core-2.13.2.jar
|   |   |   |-- jackson-databind-2.13.2.jar
|   |   |   |-- jaxb-api-2.3.0.jar
|   |   |   |-- jjwt-0.9.1.jar
|   |   |   |-- lombok-1.18.24.jar
|   |   |   `-- mysql-connector-j-9.0.0.jar
|   |   `-- web.xml
```

## ➤ 结构介绍

### ◆ 项目采用 MVC 设计模式：

**模型（Model）：** Model 负责数据的处理和业务逻辑。

- ❖ POJO 类（Plain Old Java Object）：这些类用于表示项目中的业务对象，每个类映射到一个数据表。
- ❖ DAO 层（Data Access Object）：负责与数据库进行交互，执行增、删、改、查等操作。
- ❖ Service 层：包含核心业务逻辑处理，负责处理与账户和课程相关的业务逻辑。每个服务层的实现类调用 DAO 层的方法，进行必要的业务处理。

**视图（View）**

- ❖ 由于采用前后端分离模式，因此没有视图层。

**控制器（Controller）：** 负责接收用户的请求并决定调用哪个业务逻辑

- ❖ 负责接收用户的 HTTP 请求，并将请求分发到相应的 Service 层进行处理。每个控制器处理特定的业务功能，如课程管理、学生注册、教师作业评分等。
- ❖ BaseController 类提供了其他控制器通用的功能，如请求参数的解析、错误处理、请求的响应等。

## ➤ 具体功能

### ◆ POJO：每个 pojo 包含与数据库中的数据相对应的属性，并定义了数据结构。

具体结构如图，使用 lombok 注解生成构造器、get、set、toString、hashCode、equals 方法

```
@AllArgsConstructor
@NoArgsConstructor
@Data
public class Account implements Serializable {
    private Integer uid;
    private UserType role;
    private String username;
    private String password;
    private Integer status;
}
```

◆ **DAO:** 负责与数据库进行交互，执行增、删、改、查等操作。

首先定义了 BaseDao 方法，每个实现类会继承此方法，实现了公共的查询方法和通用的增删改方法。

```
public class BaseDao {  
    // 公共的查询方法 返回的是单个对象  
    public <T> T baseQueryObject(Class<T> clazz, String sql, Object... args) {...}  
    // 公共的查询方法 返回的是对象的集合  
    public <T> List<T> baseQuery(Class clazz, String sql, Object... args) {...}  
    // 通用的增删改方法  
    public int baseUpdate(String sql, Object... args) {...}  
}
```

Dao 接口，定义了具体的函数类型，以及说明信息，方便管理及实现类调用。

```
public interface AccountDao {  
    /** 查询所有账户信息（不含密码）的DAO方法 ... */  
    List<Management> findAll();  
    /** 根据用户名查询账户的DAO方法 ... */  
    Account findByUserName(String userName);  
    /** 根据uid查询账户的DAO方法 ... */  
    Account findById(Integer uid);  
    /** 注册账号的DAO方法 ... */  
    Integer insertAccount(Account registerAccount);  
    /** 上传个人信息的DAO方法 ... */  
    Integer insertProfile(Profile profile);  
    /** 下载个人信息的DAO方法 ... */  
    List<Profile> findProfile(int uid);  
    /** 设置账户状态的DAO方法 ... */  
    Integer setStatus(Integer uid, Integer status);  
    /** 获取账户状态的DAO方法 ... */  
    List<Account> getStatus(Integer uid);  
    /** 重置密码为默认（123456）的DAO方法 ... */  
    Integer updatePasswordDefault(Integer uid, String pwd);  
    /** 删除账号的DAO方法 ... */  
    Integer updateDeleteAccount(Integer uid);  
    /** 根据用户名查询用户ID的DAO方法 ... */  
    List<Account> findUidByUsername(String username);  
}
```

DAO 实现类，继承 BaseDao 的通用增删改查方法，同时实现了接口定义的方法，在此编写具体的 sql 语句，通过 jdbc 与数据库连接返回查询到的信息，生成 POJO 类，或者操作数据库进行增删改。

```
public class AccountDaoImpl extends BaseDao implements AccountDao {  
    @Override  
    public List<Management> findAll() {  
        String sql = ""  
            SELECT a.uid, a.role, a.username, COALESCE(p.nickname, '') AS nickname, a.status  
            FROM account a  
            LEFT JOIN profile p ON a.uid = p.uid;  
            "";  
  
        return baseQuery(Management.class, sql);  
    }  
  
    @Override  
    public Account findByUserName(String userName) {  
        String sql = ""  
            select uid, role, username, password  
            from account  
            where username = ?  
            "";  
  
        List<Account> list = baseQuery(Account.class, sql, userName);  
        return list.size() > 0 ? list.get(0) : null;  
    }  
  
    @Override  
    public Account findById(Integer uid) {...}  
  
    @Override  
    public Integer insertAccount(Account registerAccount) {...}  
  
    @Override  
    public Integer insertProfile(Profile profile) {...}  
  
    @Override  
    public List<Profile> findProfile(int uid) {...}  
  
    @Override  
    public Integer setStatus(Integer uid, Integer status) {...}  
  
    @Override  
    public List<Account> getStatus(Integer uid) {...}  
  
    @Override  
    public Integer updatePasswordDefault(Integer uid, String pwd) {...}  
  
    @Override  
    public Integer updateDeleteAccount(Integer uid) {...}  
  
    @Override  
    public List<Account> findUidByUsername(String username) {...}  
}
```



◆ **Service:** 每个服务层的实现类调用 DAO 层的方法，进行必要的业务处理。

Service 接口，定义了具体的函数类型，以及说明信息，方便管理及实现类调用。

```
1 public interface AccountService {  
    /** 查询所有账户信息（不含密码）的方法 ... */  
    List<Management> findAll();  
    /** 根据用户名查询账户 ... */  
    Account findByUserName(String userName);  
    /** 根据uid查询账户 ... */  
    Account findByUid(Integer uid);  
    /** 注册账号 ... */  
    Integer registerAccount(Account registerAccount);  
    /** 上传个人信息 ... */  
    Integer uploadProfile(Profile profile);  
    /** 下载个人信息 ... */  
    List<Profile> downloadProfile(int uid);  
    /** 设置账户状态 ... */  
    Integer setStatus(Integer uid, Integer status);  
    /** 获取账户状态 ... */  
    List<Account> getStatus(Integer uid);  
    /** 重置密码 ... */  
    Integer resetPassword(Integer uid, String defaultPwd);  
    /** 删除账号 ... */  
    Integer deleteAccount(Integer uid);  
    /** 根据用户名查询用户ID ... */  
    List<Account> findUidByUsername(String username);  
}
```

Service 实现类，这里实现了接口，并调用 DAO 层的方法，进行必要的业务处理。

```
public class AccountServiceImpl implements AccountService {
    private AccountDao accountDao = new AccountDaoImpl();

    @Override
    public List<Management> findAll() { return accountDao.findAll(); }

    @Override
    public Account findByUserName(String userName) { return accountDao.findByUserName(userName); }

    @Override
    public Account findByUid(Integer uid) { return accountDao.findByUid(uid); }

    @Override
    public Integer registerAccount(Account registerAccount) {...}

    @Override
    public Integer uploadProfile(Profile profile) { return accountDao.insertProfile(profile); }

    @Override
    public List<Profile> downloadProfile(int uid) { return accountDao.findProfile(uid); }

    @Override
    public Integer setStatus(Integer uid, Integer status) { return accountDao.setStatus(uid, status); }

    @Override
    public List<Account> getStatus(Integer uid) { return accountDao.getStatus(uid); }

    @Override
    public Integer resetPassword(Integer uid, String defaultPwd) {...}

    @Override
    public Integer deleteAccount(Integer uid) { return accountDao.updateDeleteAccount(uid); }

    @Override
    public List<Account> findUidByUsername(String username) { return accountDao.findUidByUsername(username); }
}
```

◆ **Controller:** 每个控制器处理特定的业务功能，如课程管理、学生注册、教师作业评分等。

首先定义了 BaseController 方法，每个 Controller 会继承此方法，对 req 和 resp 进行一些通用的处理。

```
public class BaseController extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // 响应的MIME类型和乱码问题
        resp.setContentType("application/json;charset=UTF-8");

        String requestURI = req.getRequestURI();
        String[] split = requestURI.split(regex: "/");
        String methodName = split[split.length - 1];
        // 通过反射获取要执行的方法
        Class clazz = this.getClass();
        try {
            Method method = clazz.getDeclaredMethod(
                (methodName, HttpServletRequest.class, HttpServletResponse.class);
            // 设置方法可以访问
            method.setAccessible(true);
            // 通过反射执行代码
            method.invoke(obj: this, req, resp);
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e.getMessage());
        }
    }
}
```

具体的 Controller 方法，负责接收用户的 HTTP 请求，并将请求分发到相应的 Service 层进行处理。通过@WebServlet 注解指定 api 接口的调用路径。

```
@WebServlet("/account/*")
public class AccountController extends BaseController {
    private AccountService accountService = new AccountServiceImpl();

    /** 查询所有账户信息（不含密码）的业务接口实现 ...*/
    protected void findAllAccount(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 处理登录表单提交的业务接口实现 ...*/
    protected void login(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 获取账户信息的业务接口实现 ...*/
    protected void getAccountInfo(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 检验用户名是否重名业务接口实现 ...*/
    protected void checkUsername(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 注册表单提交的业务接口实现 ...*/
    protected void register(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 检验登录状态是否过期的业务接口实现 ...*/
    protected void checkLogin(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 上传个人信息的业务接口实现 ...*/
    protected void uploadProfile(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 读取个人信息的业务接口实现 ...*/
    protected void downloadProfile(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 设置账户状态的业务接口实现 ...*/
    protected void setStatus(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 获取账户状态的业务接口实现 ...*/
    protected void getStatus(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 重置密码的业务接口实现 ...*/
    protected void resetPassword(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 删除账号的业务接口实现 ...*/
    protected void deleteAccount(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}

    /** 根据账户ID获取账户名的业务接口实现 ...*/
    protected void findUidByUsername(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {...}
}
```

以处理登录表单提交的业务接口实现为例展示，先是通过 WebUtil 工具类解析前端发来的 JSON 串，将其转换为对应 POJO，然后调用服务层方法实现具体业务，最后判断返回信息是否合法，写入 JSON 串到 Result 实例，并调用 WebUtil 工具类将包含了响应信息的 Result 返回给前端。

```
/** 处理登录表单提交的业务接口实现 ...*/
protected void login(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("LOGIN FORM SUBMIT");

    // 接收用户名和密码
    Account paramAccount = WebUtil.readJson(req, Account.class);
    // 调用服务层接口实现登录
    Account loginAccount = accountService.findByUserName(paramAccount.getUsername());
    Result<Object> res = null;
    if (loginAccount != null) {
        if (EncryptUtil.encrypt(paramAccount.getPassword()).equals(loginAccount.getPassword())) {
            String token = JwtUtil.createToken(loginAccount.getId().longValue());
            // 使用Map转换token, 保证返回的是key-value格式
            Map data = new HashMap<>();
            data.put("token", token);
            res = Result.ok(data);
        } else {
            res = Result.build( body: null, ResultCodeEnum.PASSWORD_ERROR);
        }
    } else {
        res = Result.build( body: null, ResultCodeEnum.USERNAME_ERROR);
    }
    // 向客户端响应登录验证信息
    WebUtil.writeJson(resp, res);
}
```

◆ **过滤器：**处理跨域资源共享请求，告诉浏览器访问的服务器是安全的。

```
@WebFilter("/*")
public class CorsFilter implements Filter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) servletResponse;
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        response.setHeader(s: "Access-Control-Allow-Origin", s1: "*");
        response.setHeader(s: "Access-Control-Allow-Methods", s1: "POST, GET, OPTIONS, DELETE, HEAD");
        response.setHeader(s: "Access-Control-Max-Age", s1: "3600");
        response.setHeader(s: "Access-Control-Allow-Headers", s1: "access-control-allow-origin, authority, content-type, version-info, X-Requested-With, token");
        // 非预检请求，放行即可，预检请求，则到此结束，不需要放行
        if (!request.getMethod().equalsIgnoreCase( anotherString: "OPTIONS")) {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }
}
```

◆ **工具类**

JDBCUtil，在此类中会读取配置文件信息，创建 Druid 数据库连接池。

```
public class JDBCUtil {
    private static ThreadLocal<Connection> threadLocal = new ThreadLocal<>();
    private static DataSource dataSource;

    // 初始化连接池
    static {...}

    /* 向外提供连接池的方法 */
    public static DataSource getDataSource() { return dataSource; }

    /* 向外提供连接的方法 */
    public static Connection getConnection() {...}

    /* 定义一个归还连接的方法（解除和ThreadLocal之间的关联关系） */
    public static void releaseConnection() {...}
}
```

jdbc.properties 配置文件

```
url=jdbc:mysql://8.134.98.31:3306/szu_se_db?useUnicode=true&characterEncoding=utf8;
username=root
password=zht040515
driverClassName=com.mysql.cj.jdbc.Driver
initialSize=5
maxActive=20
maxWait=10000
mysql.abandonedConnectionCleanUpEnabled=false
```

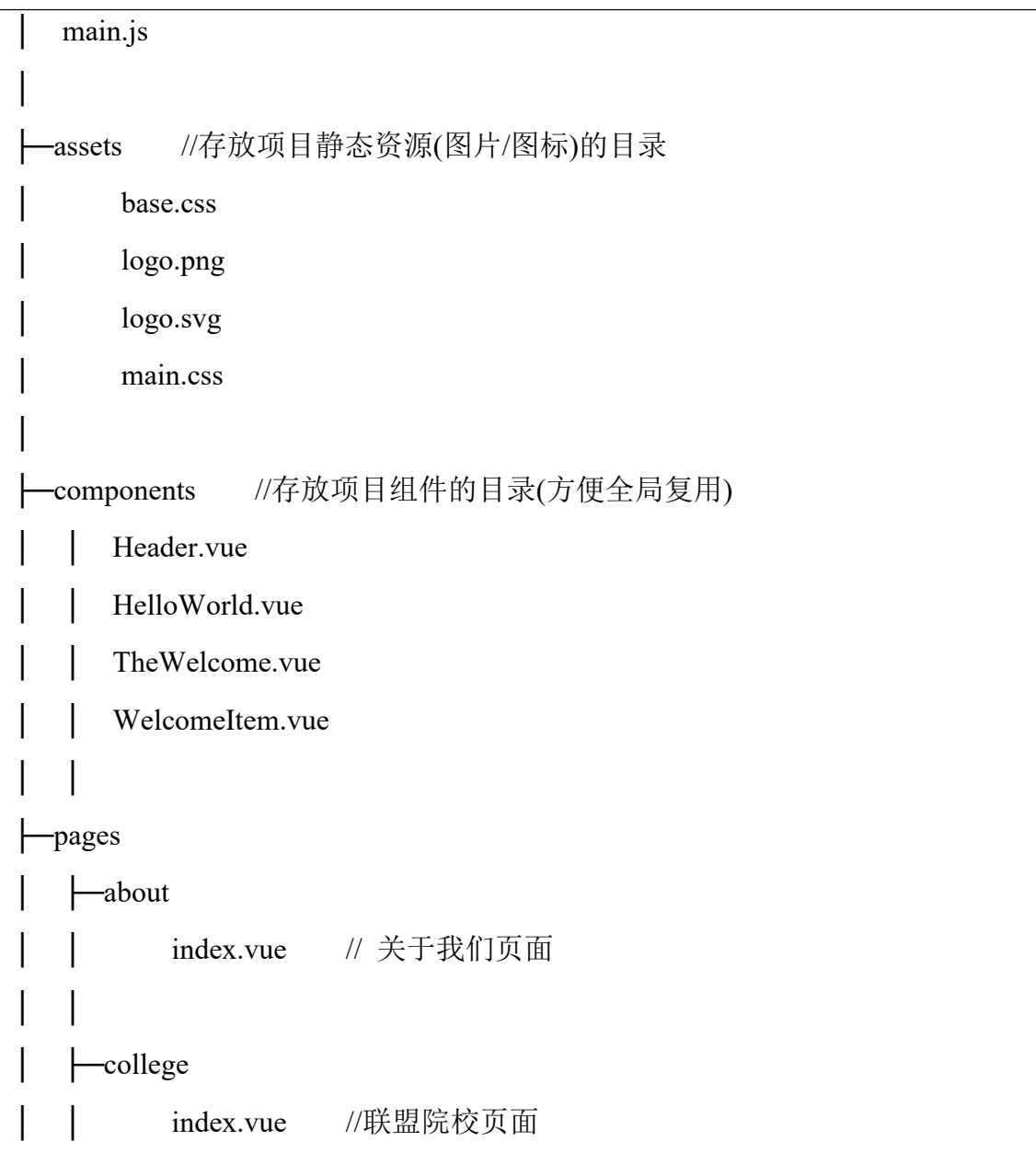


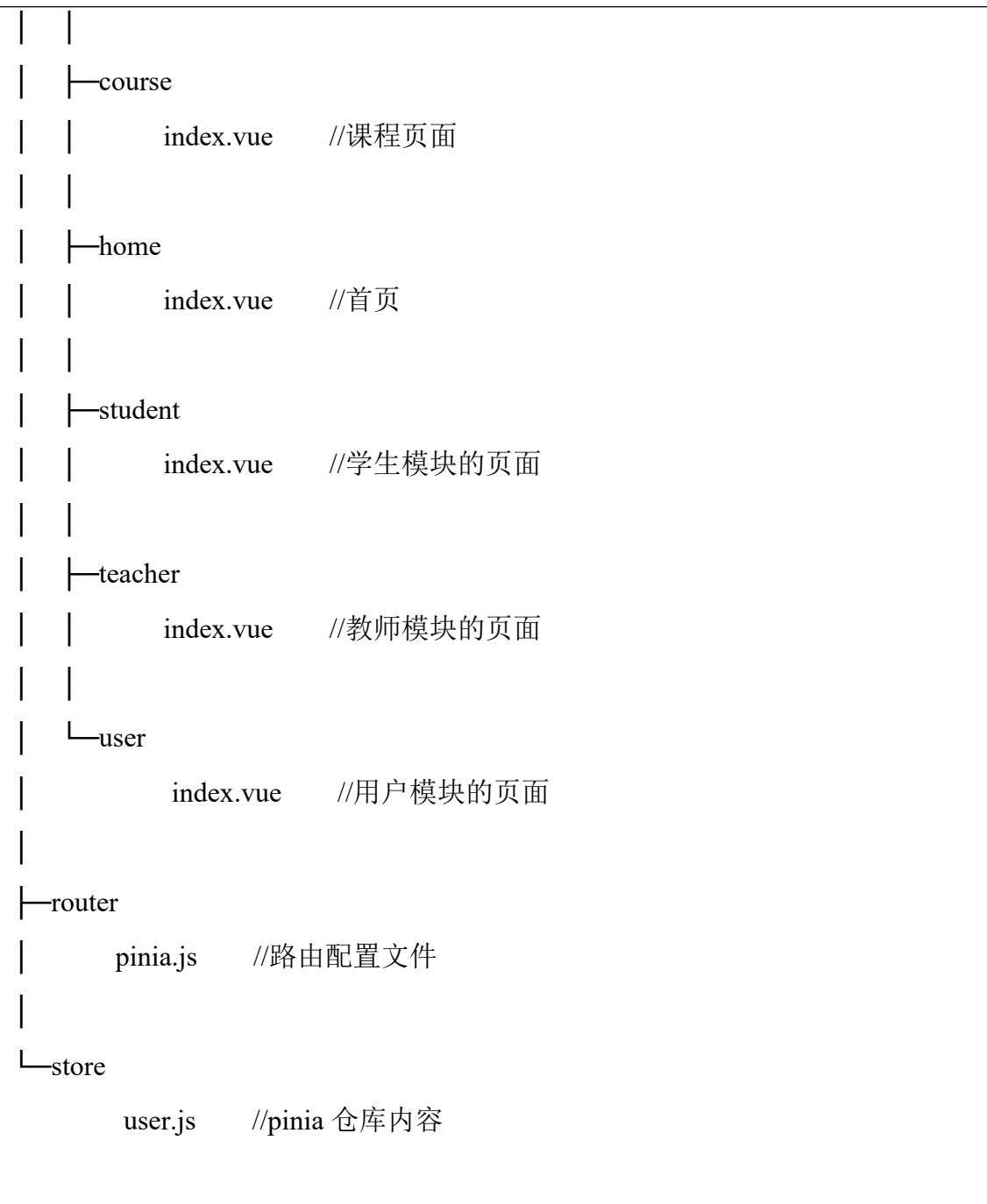
除此之外还有负责加密的 `EncryptUtil`，将密码进行加盐哈希加密；`JwtUtil` 负责生成 token 以及判断 token 是否过期；`WebUtil` 负责处理请求和相应信息，可以读写 JSON 串调用反射生成对应类，以及传入 `Result` 返回前端响应码等信息。



## 2. 前端

### ➤ 目录结构





## ➤ 结构介绍

### ◆ main.js

该文件是 Vue 应用的入口文件，负责创建和初始化 Vue 实例，并挂载到指定的 DOM 元素。在这里会导入全局配置、插件和根组件，并配置全局路由和状态管理。

### ◆ assets/

该目录存放了项目中的静态资源文件，如图片、图标、字体和全局样式等。

base.css 和 main.css 文件用于全局的样式定义，可能包括字体、排版、颜色方案等。

#### ◆ components/

该目录存放的是可以在多个页面中复用的组件，可以提高代码复用性和维护性。

#### ◆ pages/

该目录包含了项目的各个页面模块。每个页面对应一个 Vue 文件，代表一个功能区域或用户界面模块。

about/index.vue: 关于我们页面，包含公司信息、联系方式、团队介绍等。

college/index.vue: 联盟院校页面，展示与学校相关的信息。

course/index.vue: 课程页面，展示平台上提供的课程信息。

home/index.vue: 首页，是用户进入网站后看到的首个页面，包含重要的导航和内容概览。

student/index.vue: 学生模块页面，包含与学生相关的功能，如课程注册、作业提交等。

teacher/index.vue: 教师模块页面，包含教师管理课程、发布作业、查看学生成绩等功能。

user/index.vue: 用户模块页面，包括用户登录、注册、个人资料管理等功能。

#### ◆ router/

该目录用于存放路由相关配置文件，用于管理不同页面之间的导航。

pinia.js: 路由配置文件，设置应用的路由规则，定义页面路径与组件的映射关系。

#### ◆ store/

该目录用于存放 Pinia 状态管理相关的文件，用于集中管理应用中的状态。

user.js: 这个文件存储了与用户相关的状态信息，包括用户的登录状态、个人信息等，方便在全局范围内访问和修改。

#### ➤ 具体功能

这里重点介绍前后端交互部分、路由跳转以及 js 部分

#### ◆ 前后端交互

在 api.js 文件中定义了与后端交互的函数，每个函数会调用 request 的 get 或 post 方法向后端的业务接口传入参数，像下图这样。

```
// 登录的接口
export const getLogin = (info) : Promise<AxiosResponse<...>> => {
  return request.post( url: "account/login", info);
}
```

```

import request from './request.js'

//登录的接口
export const getLogin = (info) : Promise<AxiosResponse<...>> => {...};
//获取用户信息的接口
export const getUserInfo = () : Promise<AxiosResponse<...>> => {...};
//注册校验的接口 user/checkUserName
export const registerValidateApi = (username) : Promise<AxiosResponse<...>> => {...};
// 注册的接口
export const registerApi = (userInfo) : Promise<AxiosResponse<...>> => {...}
// 判断用户登录过期的接口
export const isUserOverdue = () : Promise<AxiosResponse<...>> => {...}
// 上传个人信息的接口
export const uploadProfile = (accountForm) : Promise<AxiosResponse<...>> => {...}
// 下载个人信息的接口
export const downloadProfile = (uid) : Promise<AxiosResponse<...>> => {...}
// 获取账号状态的接口
export const getStatus = (uid) : Promise<AxiosResponse<...>> => {...}
// 设置账号状态的接口
export const setStatus = (uid, status) : Promise<AxiosResponse<...>> => {...}
// 获取所有账户信息的接口
export const getAllManagementInfo = () : Promise<AxiosResponse<...>> => {...}
// 重置密码的接口
export const resetPassword = (uid, defaultPwd) : Promise<AxiosResponse<...>> => {...}
// 删除账号的接口
export const deleteAccount = (uid) : Promise<AxiosResponse<...>> => {...}
// 根据账户名查询账户ID的接口
export const findUidByUsername = (username) : Promise<AxiosResponse<...>> => {...}

```

其中 request.js 中设置了请求拦截器和响应拦截器。

请求拦截器负责在每次发出请求前将浏览器 local storage 中存储的 token 一起发送给后端。

```

// 添加请求拦截器
service.interceptors.request.use( onFulfilled: (config) => {
  NProgress.start() // 开启进度条
  // 如果有token, 通过请求头携带给后台
  const token : string = getToken()
  if (token) {
    // config.headers['token'] = token // 报错: headers对象并没有声明有token, 不能随便添加
    (config.headers)['token'] = token
  }
  return config;
});

```



响应拦截器会接受后端响应的信息，并根据状态码判断是返回接收的数据还是进行错误提示。

```
// 添加响应拦截器
service.interceptors.response.use(
  onFulfilled: (response) : ... => {
    NProgress.done() // 关闭进度条

    if (response.data.code !== 200) {
      // 判断响应状态码
      if (response.data.code === 501) return Promise.reject(ELMessage.error( options: "用户名有误"))
      else if (response.data.code === 502) return Promise.reject(ELMessage.error( options: "帐户不存在"))
      else if (response.data.code === 503) return Promise.reject(ELMessage.error( options: "密码有误"))
      else if (response.data.code === 504) return Promise.reject(ELMessage.error( options: "登录已过期"))
      else if (response.data.code === 505) return Promise.reject(ELMessage.error( options: "用户名已占用"))
      else if (response.data.code === 506) return Promise.reject(ELMessage.error( options: "上传失败"))
      else if (response.data.code === 507) return Promise.reject(ELMessage.error( options: "下载失败"))
    } else {
      return response.data.data; /* 返回成功响应数据 */
    }
  },
  onRejected: (error) : Promise<never> => {
    NProgress.done() // 关闭进度条
    return Promise.reject(error.message);
  }
);
```

## ◆ 路由跳转

路由配置里设置了当在浏览器输入路径时跳转到对应组件，requiresAuth 可以设置当前页面是否有权访问，游客和学生只能访问部分允许访问的页面。

```
// 路由配置
const routes : [{redirect: string, path: stri... = [
  // 默认重定向到 /home
  {path: '/' ...},
  {
    path: '/home',
    name: 'homeIndex',
    component: () : Promise<{...}> => import('../pages/home/index.vue'),
    meta: {requiresAuth: false}, // 不需要认证的路由
  },
  {name: 'courseIndex' ...},
  {name: 'courseDetailIndex' ...},
  {name: 'collegeIndex' ...},
  {name: 'aboutIndex' ...},
  {name: 'userIndex' ...},
  {name: 'studentIndex' ...},
  {name: 'exerciseIndex' ...},
];

const router : Router = createRouter( options: {
  history: createWebHistory(import.meta.env.BASE_URL), // 使用 import.meta.env.BASE_URL 动态获取基础路径
  routes,
});
```

路由守卫负责进行权限管理，如果当前页面无权访问，比如 token 已过期或者游客访问，会重定向到主页。

```
// 路由守卫：检查是否需要权限认证
router.beforeEach( guard: async (to : RouteLocationNormalized , from : RouteLocationNormalizedLoaded , next : NavigationGuardNext ) : Promise<...> => {
    const userStore : Store<"user", {...}, {...}, {...}> = useUserStore(); // 使用 store 获取用户认证信息
    await userStore.checkToken();
    const isAuthenticated : UnwrapRef<boolean> = userStore.isAuthenticated; // 有一个 isAuthenticated 标志来判断是否已登录
    const userRole : null | any = userStore.currentUser == null ? null : userStore.currentUser.role; // 假设用户角色存储在 userRole 中
    console.log(userRole);

    const routePaths : (...[]) = routes.map(route : ... => route.path);
    // 如果目标路径在 routes 中且不需要认证，直接放行
    if (routePaths.includes(to.path) && !to.meta.requiresAuth) {...}
    // 如果目标路径在 routes 中且需要认证，用户认证后跳转
    if (routePaths.includes(to.path) && to.meta.requiresAuth && isAuthenticated) {...}
    // 如果目标路由需要认证且用户未认证，跳转到首页
    if (to.meta.requiresAuth && !isAuthenticated) {...}
    // 如果用户未认证且当前路径不是 home，跳转到首页
    if (!isAuthenticated && to.path !== '/home') {...}
});
```

## ◆ JavaScript

登录逻辑：当用户按下登录按钮时，会先调用 api 查询当前用户状态，如果当前属于冻结状态则不可登录。

```
const onConfirm = () => {
    if (userFormRef.value) {
        userFormRef.value.validate(async (valid) => {
            if (valid) {
                const uid = await findUidByUsername(userForm.username); // 获取用户 UID
                const status = await getStatus(uid); // 获取账户状态

                if (status === 1) { // 1 表示账户被冻结
                    ElMessage.error( options: '账户已被冻结，无法登录! ');
                    return;
                }
            }
        })
    }
}
```

如果账户未被冻结，则会调用 `login` 方法传入表单尝试登录，成功则关闭登录 `ui`，重置输错次数，加载头像；失败则进入 `catch`，提示用户还有多少次输错会被冻结，输错 5 次则调用 `api` 冻结账户。

```
try {
  await userStore.login(userForm); // 尝试登录
  ElMessage.success( options: '登录成功! ');
  visible.value = false;
  errorCount = 0
  await accountStore.downloadAccountProfile(userStore.currentUser.uid)
} catch (e) {
  errorCount++; // 假设loginResult返回错误次数
  const remainingAttempts = 5 - errorCount; // 剩余尝试次数

  if (remainingAttempts > 0) {
    ElMessage.error( options: `密码错误! 剩余${remainingAttempts}次机会`);
  } else {
    await setStatus(uid, status: 1); // 错误次数超过5次, 冻结账户
    ElMessage.error( options: '错误次数过多, 账户已被冻结! ');
  }
}
```

调用 `api` 检验是否重名。

```
// 检验用户名是否重名
const checkSameUsername = async (rule, value, callback) => {
  try {
    const res = await registerValidateApi(value)
  } catch (e) {
    callback(new Error('用户名已占用! '))
  }
}
```

检验用户名是否合法。

```
// 检验用户名是否合法
const validateUsername = (rule, value, callback) => {
  // 判断用户名是否为空
  if (value.length !== 10) {
    callback(new Error('学号或工号长10位!'));
  } else if (!/^\\d{10}$/.test(value)) { // 判断是否是纯数字
    callback(new Error('学号或工号必须为纯数字!'));
  } else {
    callback(); // 验证通过
  }
};
```

比较注册时输入的两次密码是否一致。

```
// 比较密码是否一致
const validatePwd = (rule, value, callback) => {
  if (value !== registerForm.password) {
    callback(new Error('密码并不一致!')) // 自定义错误提示
  } else {
    callback() // 验证通过
  }
};
```

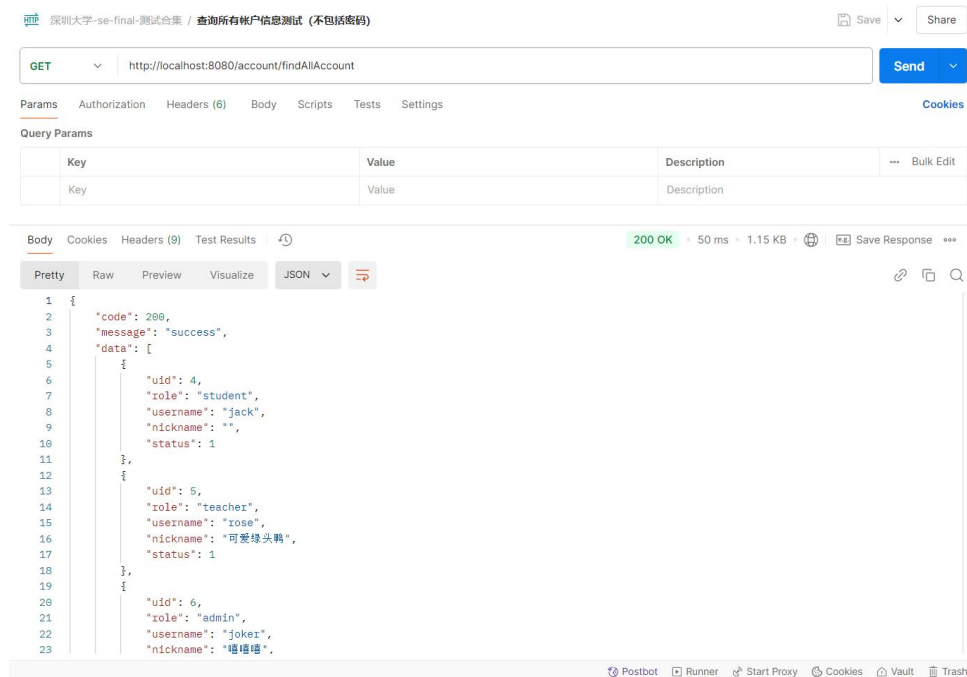
退出登录会重定向到首页，并且清除 token。

```
// 比较密码是否一致
const validatePwd = (rule, value, callback) => {
  if (value !== registerForm.password) {
    callback(new Error('密码并不一致!')) // 自定义错误提示
  } else {
    callback() // 验证通过
  }
};
```

## 二、测试报告

### ◆ 测试账户信息安全

测试获取账户信息时是否会返回密码，结果并没有返回密码，保证用户信息不被泄露。



深圳大学-se-final-测试合集 / 查询所有帐户信息测试 (不包括密码)

GET http://localhost:8080/account/findAllAccount

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (9) Test Results 200 OK • 50 ms • 1.15 KB Save Response

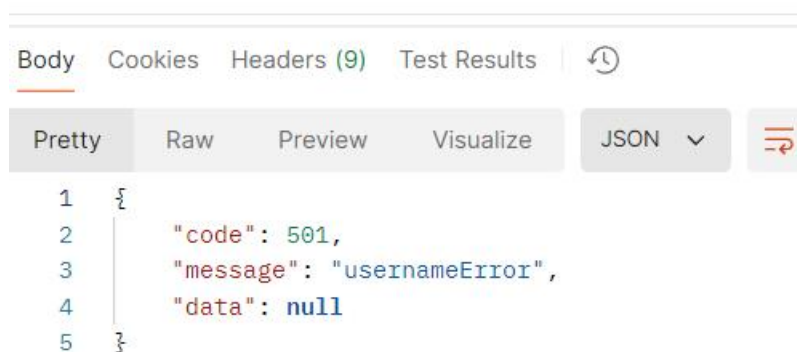
Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 200,
3   "message": "success",
4   "data": [
5     {
6       "uid": 4,
7       "role": "student",
8       "username": "jack",
9       "nickname": "",
10      "status": 1
11    },
12    {
13      "uid": 5,
14      "role": "teacher",
15      "username": "rose",
16      "nickname": "可爱绿头鸭",
17      "status": 1
18    },
19    {
20      "uid": 6,
21      "role": "admin",
22      "username": "joker",
23      "nickname": "嘻嘻嘻",
```

### ◆ 测试 SQL 注入

用户名或密码字段中的 SQL 注入（基本的 OR 1=1 攻击），这条语句会使 SQL 查询始终为真，从而绕过身份验证。结果返回 usernameError。

```
1 {
2   "username": "jerry' OR 1=1 --",
3   "password": "1111111111"
4 }
5
```



Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 501,
3   "message": "usernameError",
4   "data": null
5 }
```

联合查询注入（泄露数据库信息），这条语句会尝试获取数据库的版本信息。  
结果返回 `usernameError`。

```
1  {  
2  |    "username": "jerry' UNION SELECT null, version() --",  
3  |    "password": "1111111111"  
4  | }  
5
```

Body Cookies Headers (9) Test Results ↺

Pretty Raw Preview Visualize JSON ↕

```
1  {  
2  |    "code": 501,  
3  |    "message": "usernameError",  
4  |    "data": null  
5  | }
```

条件注入（例如通过 `OR` 语句进行绕过）。  
结果返回 `usernameError`。

```
1  {  
2  |    "username": "jerry' OR 'a'='a' --",  
3  |    "password": "1111111111"  
4  | }  
5
```

Body Cookies Headers (9) Test Results ↺

Pretty Raw Preview Visualize JSON ↕

```
1  {  
2  |    "code": 501,  
3  |    "message": "usernameError",  
4  |    "data": null  
5  | }
```



闭合引号和 SQL 注释。

结果返回 `usernameError`。

```
1  {
2    "username": "jerry' --",
3    "password": "1111111111"
4  }
5
```

Body Cookies Headers (9) Test Results ↻

Pretty Raw Preview Visualize JSON ▾ ⌵

```
1  {
2    "code": 501,
3    "message": "usernameError",
4    "data": null
5  }
```

等待注入（延迟注入），如果响应延迟，可能说明应用存在 SQL 注入漏洞。

结果返回 `usernameError`，且无延迟。

```
1  {
2    "username": "jerry' OR SLEEP(5) --",
3    "password": "1111111111"
4  }
5
```

Body Cookies Headers (9) Test Results ↻

Pretty Raw Preview Visualize JSON ▾ ⌵

```
1  {
2    "code": 501,
3    "message": "usernameError",
4    "data": null
5  }
```

◆ 测试 token

传入过期 token。

结果返回 notLogin。

HTTP 深圳大学-se-final-测试合集 / 根据token请求头获得登录的用户信息接口测试

GET http://localhost:8080/account/getAccountInfo

Params Authorization Headers (7) Body Scripts Tests Settings

Headers 6 hidden

	Key	Value
<input checked="" type="checkbox"/>	token	eyJhbGciOiJIUzUxMiIsInppcCI6IkdkaSVAifQ.H4slAAAA...
	Key	Value

Body Cookies Headers (9) Test Results 200

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 504,
3   "message": "notLogin",
4   "data": null
5 }
```

传入篡改过的 token。

结果返回 notLogin。

Headers 6 hidden

	Key	Value
<input checked="" type="checkbox"/>	token	eyJhbGciOiJIUzUxMiIsInppcCI6IkdkaSVAifQ.H4slAAAA..
	Key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 504,
3   "message": "notLogin",
4   "data": null
5 }
```



测试合法 token。

结果正确返回数据。

深圳大学-se-final-测试合集 / 根据token请求头获得登录的用户信息接口测试

GET

http://localhost:8080/account/getAccountInfo

ParamsAuthorizationHeaders (7)BodyScriptsTestsSettings

Headers

6 hidden

	Key	Value	Descr
<input checked="" type="checkbox"/>	token	eyJhbGciOiJIUzUxMiIsInpucCI6IkdkaSVAifQ.H4slAAAAA...	
	Key	Value	Descr

BodyCookiesHeaders (9)Test Results200 OK

PrettyRawPreviewVisualizeJSON

```
1 {
2   "code": 200,
3   "message": "success",
4   "data": {
5     "account": {
6       "uid": 20,
7       "role": "student",
8       "username": "jerry",
9       "password": null,
10      "status": null
11     }
12   }
13 }
```

◆ 完整测试见表格

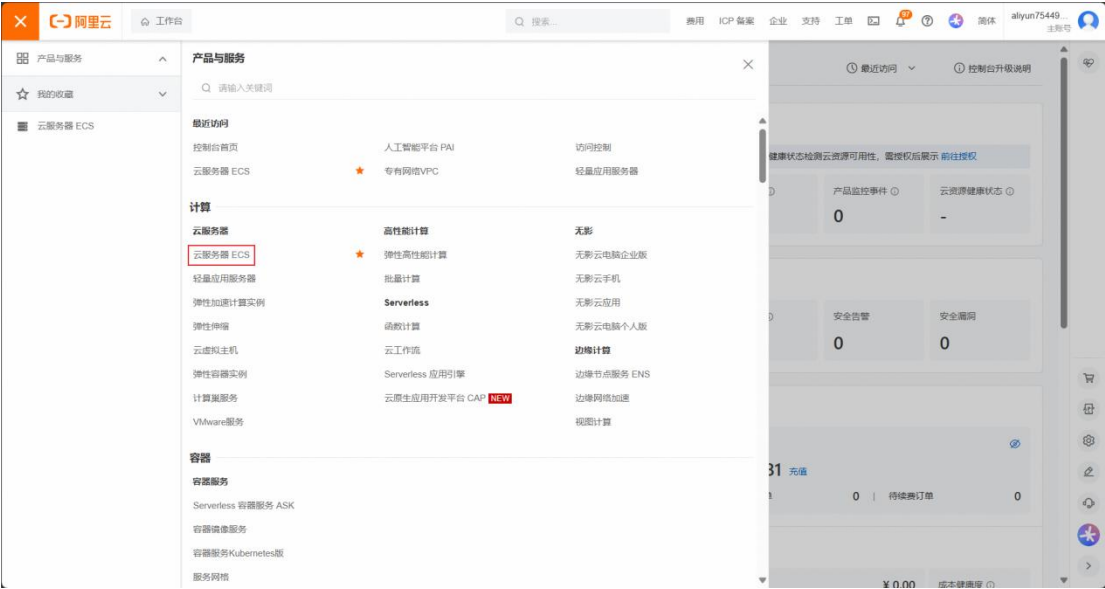
序号	测试用例	测试值	预期结果	实测结果	测试状态
1	添加新用户，所有字段正常填写	用户名：“zhanghao”， 密码：“123456”	新用户成功添加，数据库中新增一条用户记录	OK	与预期结果一致
2	用户名为空	用户名：“”，密码：“123456”	返回错误消息，提示用户名不能为空	错误消息	未通过测试
3	密码为空	用户名：“zhanghao”， 密码：“”	返回错误消息，提示密码不能为空	错误消息	未通过测试
4	邮箱格式不正确	用户名：“zhanghao”， 密码：“123456”	返回错误消息，提示邮箱格式不正确	错误消息	未通过测试

5	用户名已存在	用户名: "zhanghao", 密码: "123456"	返回错误消息, 提示用户名已存在	错误消息	未通过测试
6	注册成功后自动登录	用户名: "zhanghao", 密码: "123456"	成功注册并自动登录, 跳转到首页	OK	与预期结果一致
7	登录时密码错误	用户名: "zhanghao", 密码: "wrongpassword"	返回错误消息, 提示密码错误	错误消息	未通过测试
8	登录时用户名不存在	用户名: "nonexistentuser", 密码: "123456"	返回错误消息, 提示用户不存在	错误消息	未通过测试
9	修改个人信息, 邮箱格式错误	用户名: "zhanghao", 邮箱: "invalidemail", 电话: "123456789"	返回错误消息, 提示邮箱格式不正确	错误消息	未通过测试
10	提交作业, 所有字段正常填写	作业内容: "作业内容示例", 提交时间: "2023-12-31"	作业成功提交, 数据库中新增一条作业记录	OK	与预期结果一致
11	提交作业, 作业内容为空	作业内容: "", 提交时间: "2023-12-31"	返回错误消息, 提示作业内容不能为空	错误消息	未通过测试
12	提交作业, 提交时间在截止日期之前	作业内容: "作业内容示例", 提交时间: "2023-12-30"	作业成功提交, 数据库中记录更新	OK	与预期结果一致
13	提交作业, 提交时间晚于截止日期	作业内容: "作业内容示例", 提交时间: "2024-01-01"	返回错误消息, 提示不能晚于截止日期提交	错误消息	未通过测试
14	课程注册, 所有字段正常填写	课程 ID: "101", 学生 ID: "2022090069"	成功注册课程, 数据库中新增一条注册记录	OK	与预期结果一致
15	课程注册, 课程已满	课程 ID: "101", 学生 ID: "2022090070"	返回错误消息, 提示课程已满	错误消息	未通过测试
16	课程注册, 学生已注册该课程	课程 ID: "101", 学生 ID: "2022090069"	返回错误消息, 提示学生已注册该课程	错误消息	未通过测试
17	登录时检查	用户名: "zhanghao",	登录后下次	OK	与预期结

	记住我功能，选择记住我	密码：“123456”，勾选记住我	自动填充用户名和密码		果一致
18	登录时不选择记住我	用户名：“zhanghao”，密码：“123456”，不勾选记住我	下次登录需要重新输入用户名和密码	OK	与预期结果一致

### 三、使用说明书

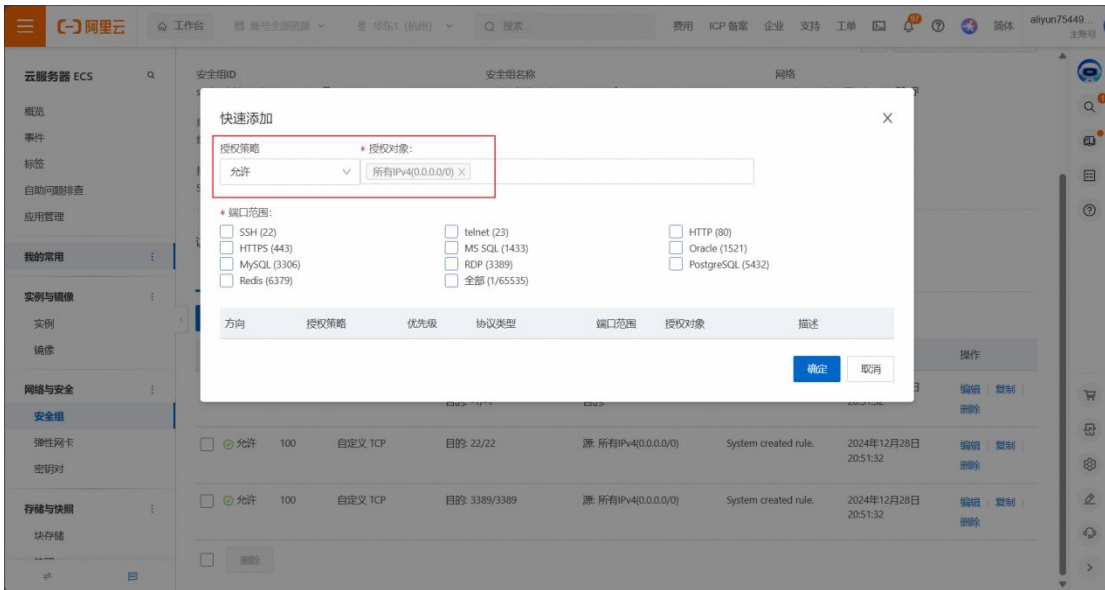
进入 aliyun，选择云服务器 ECS。



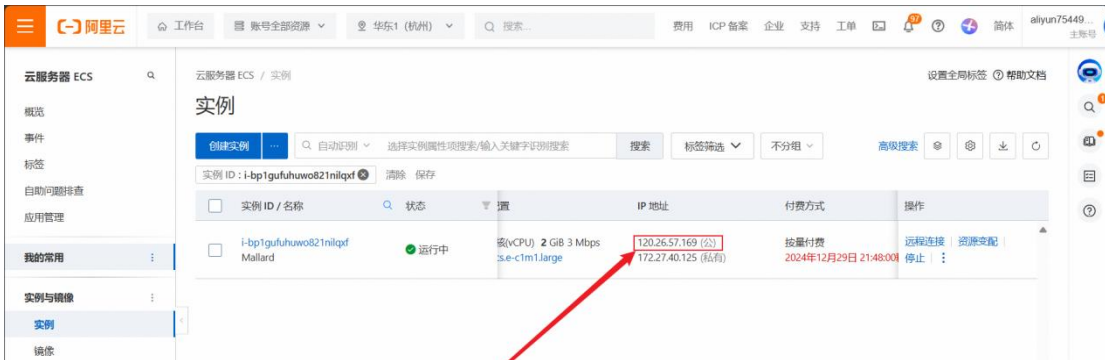
选择服务器配置，这里选择最低配置，镜像选择 window。



设置安全组，允许所有 ip（或指定 ip）访问，方便远程登录。



远程访问 ip，用户名默认是 Administrator。



## 下载 Java17

ORACLE			Products	Industries	Resources	Customers	Partners	Developers	Company	🔍	🇺🇸	👤 View Accounts	📞 Contact Sales
Linux Arm 64 RPM Package	172.62 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-aarch64_bin.rpm">https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-aarch64_bin.rpm</a> (sha256 )											
Linux x64 Compressed Archive	174.33 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.tar.gz">https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.tar.gz</a> (sha256 )											
Linux x64 Debian Package	149.69 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.deb">https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.deb</a> (sha256 )											
Linux x64 RPM Package	174.03 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.rpm">https://download.oracle.com/java/17/archive/jdk-17.0.12_linux-x64_bin.rpm</a> (sha256 )											
macOS Arm 64 Compressed Archive	168.84 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-aarch64_bin.tar.gz">https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-aarch64_bin.tar.gz</a> (sha256 )											
macOS Arm 64 DMG Installer	168.24 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-aarch64_bin.dmg">https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-aarch64_bin.dmg</a> (sha256 )											
macOS x64 Compressed Archive	170.91 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-x64_bin.tar.gz">https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-x64_bin.tar.gz</a> (sha256 )											
macOS x64 DMG Installer	170.32 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-x64_bin.dmg">https://download.oracle.com/java/17/archive/jdk-17.0.12_macos-x64_bin.dmg</a> (sha256 )											
Windows x64 Compressed Archive	172.87 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.zip">https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.zip</a> (sha256 )											
Windows x64 Installer	153.92 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.exe">https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.exe</a> (sha256 )											
Windows x64 MSI Installer	152.67 MB	<a href="https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.msi">https://download.oracle.com/java/17/archive/jdk-17.0.12_windows-x64_bin.msi</a> (sha256 )											

## 下载 Mysql 9.1.0

### MySQL Community Downloads

MySQL Community Server

**MySQL Enterprise Edition for Developers**  
Free for learning, developing, and prototyping.  
[Download Now »](#)

[General Availability \(GA\) Releases](#) [Archives](#) [📄](#)

### MySQL Community Server 9.1.0 Innovation


Select Version:

9.1.0 Innovation

Select Operating System:

Microsoft Windows

<b>Windows (x86, 64-bit), MSI Installer</b> (mysql-9.1.0-winx64.msi)	9.1.0	118.1M	<a href="#">Download</a>
MDS: 7a26420bb3440eab56f389dba05a9718   <a href="#">Signature</a>			
<b>Windows (x86, 64-bit), ZIP Archive</b> (mysql-9.1.0-winx64.zip)	9.1.0	288.4M	<a href="#">Download</a>
MDS: 0a2333afcf4ef07471bda89232697698e   <a href="#">Signature</a>			
<b>Windows (x86, 64-bit), ZIP Archive</b> <b>Debug Binaries &amp; Test Suite</b> (mysql-9.1.0-winx64-debug-test.zip)	9.1.0	822.6M	<a href="#">Download</a>
MDS: cb0327a504fc8d983f98c0cbf451a31d   <a href="#">Signature</a>			

 We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

## 下载 Tomcat 10.1.34

Download

Which version?

Tomcat 11

Tomcat 10

Tomcat 9

Tomcat Migration Tool for Jakarta EE

Tomcat Connectors

Tomcat Native

Taglibs

Archives

Documentation

Tomcat 11.0

Tomcat 10.1

Tomcat 9.0

Upgrading

Tomcat Connectors

Tomcat Native 2

Tomcat Native 1.3

Wiki

Migration Guide

Presentations

Specifications

Problems?

Security Reports

Find help

FAQ

Mailing Lists

Bug Database

IRC

Get Involved

Overview

Source code

Buildbot

Tools

Media

Twitter

YouTube

Release Integrity

You **must** [verify](#) the integrity of the downloaded files. We provide OpenPGP signatures for every release file. This signature should be matched against the [KEYS](#) file which contains the OpenPGP keys of Tomcat's Release Managers. We also provide SHA-512 checksums for every release file. After you download the file, you should calculate a checksum for your download, and make sure it is the same as ours.

Mirrors

You are currently using <https://d1cdn.apache.org/>. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors:

10.1.34

Please see the [README](#) file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
  - [32-bit Windows.zip \(pgp, sha512\)](#)
  - [64-bit Windows.zip \(pgp, sha512\)](#)
  - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
  - [tar.gz \(pgp, sha512\)](#)
- Deployer:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
- Embedded:
  - [tar.gz \(pgp, sha512\)](#)
  - [zip \(pgp, sha512\)](#)

Source Code Distributions

- [tar.gz \(pgp, sha512\)](#)
- [zip \(pgp, sha512\)](#)

## 下载 Nginx 1.26.2

Celebrating 20 years of nginx! Read about our journey and milestones in the [latest blog](#).

nginx: download

Mainline version

[CHANGES](#) [nginx-1.27.3](#) [pgp](#) [nginx/Windows-1.27.3](#) [pgp](#)

Stable version

[CHANGES-1.26](#) [nginx-1.26.2](#) [pgp](#) [nginx/Windows-1.26.2](#) [pgp](#)

Legacy versions

[CHANGES-1.24](#) [nginx-1.24.0](#) [pgp](#) [nginx/Windows-1.24.0](#) [pgp](#)

[CHANGES-1.22](#) [nginx-1.22.1](#) [pgp](#) [nginx/Windows-1.22.1](#) [pgp](#)

[CHANGES-1.20](#) [nginx-1.20.2](#) [pgp](#) [nginx/Windows-1.20.2](#) [pgp](#)

[CHANGES-1.18](#) [nginx-1.18.0](#) [pgp](#) [nginx/Windows-1.18.0](#) [pgp](#)

[CHANGES-1.16](#) [nginx-1.16.1](#) [pgp](#) [nginx/Windows-1.16.1](#) [pgp](#)

[CHANGES-1.14](#) [nginx-1.14.2](#) [pgp](#) [nginx/Windows-1.14.2](#) [pgp](#)

[CHANGES-1.12](#) [nginx-1.12.2](#) [pgp](#) [nginx/Windows-1.12.2](#) [pgp](#)

[CHANGES-1.10](#) [nginx-1.10.3](#) [pgp](#) [nginx/Windows-1.10.3](#) [pgp](#)

[CHANGES-1.8](#) [nginx-1.8.1](#) [pgp](#) [nginx/Windows-1.8.1](#) [pgp](#)

[CHANGES-1.6](#) [nginx-1.6.3](#) [pgp](#) [nginx/Windows-1.6.3](#) [pgp](#)

[CHANGES-1.4](#) [nginx-1.4.7](#) [pgp](#) [nginx/Windows-1.4.7](#) [pgp](#)

[CHANGES-1.2](#) [nginx-1.2.9](#) [pgp](#) [nginx/Windows-1.2.9](#) [pgp](#)


[CHANGES-1.0](#) [nginx-1.0.15](#) [pgp](#) [nginx/Windows-1.0.15](#) [pgp](#)

[CHANGES-0.8](#) [nginx-0.8.55](#) [pgp](#) [nginx/Windows-0.8.55](#) [pgp](#)

[CHANGES-0.7](#) [nginx-0.7.69](#) [pgp](#) [nginx/Windows-0.7.69](#) [pgp](#)

[CHANGES-0.6](#) [nginx-0.6.39](#) [pgp](#)

[CHANGES-0.5](#) [nginx-0.5.38](#) [pgp](#)

 NGINX  
20 Years

english

русский

[news](#)

[about](#)

[download](#)

[security](#)

[documentation](#)

[faq](#)

[books](#)

[community](#)

[enterprise](#)

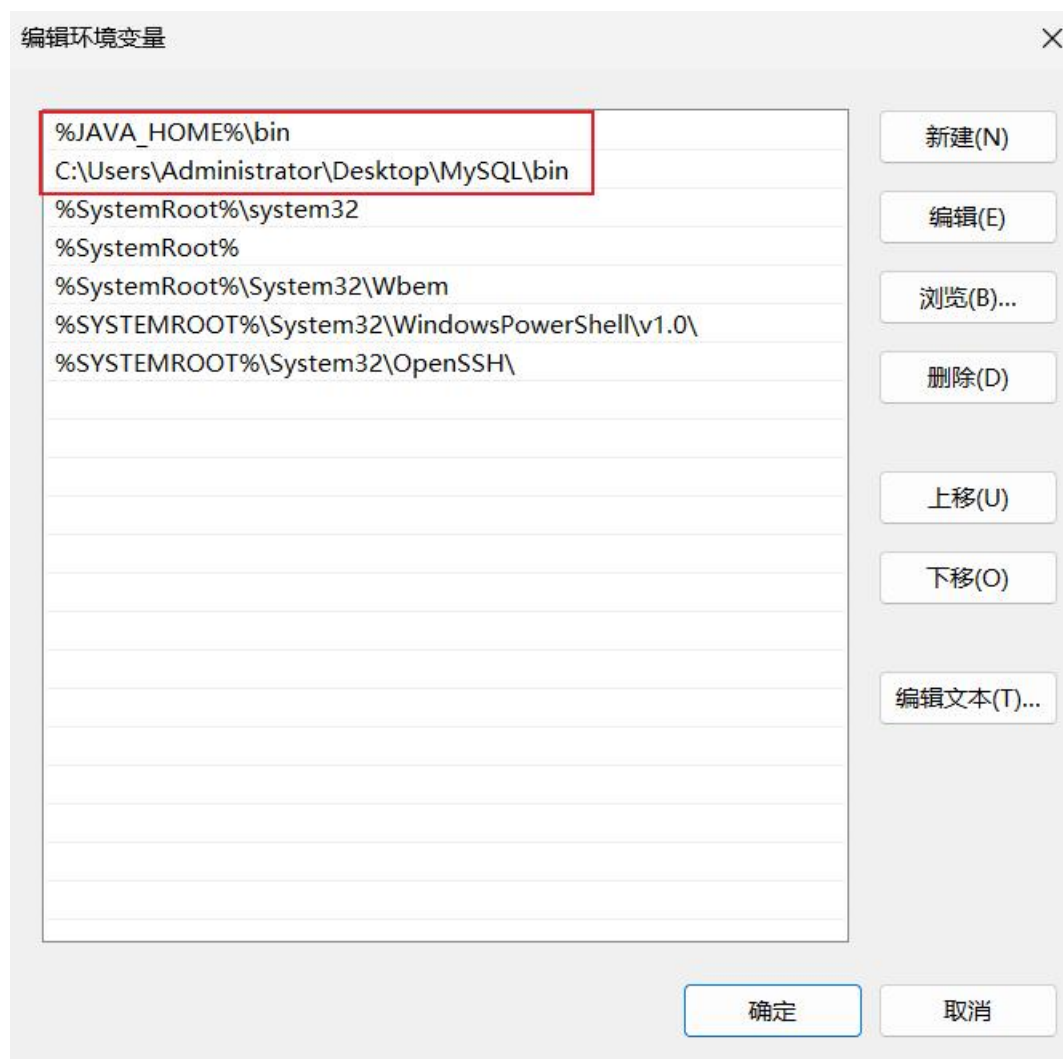
[x.com](#)

[blog](#)

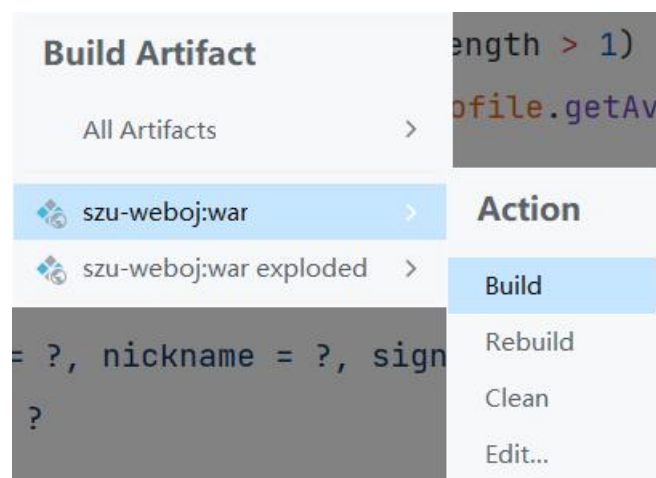
[unity](#)

[rja](#)

## 配置环境变量



## 打包后端程序



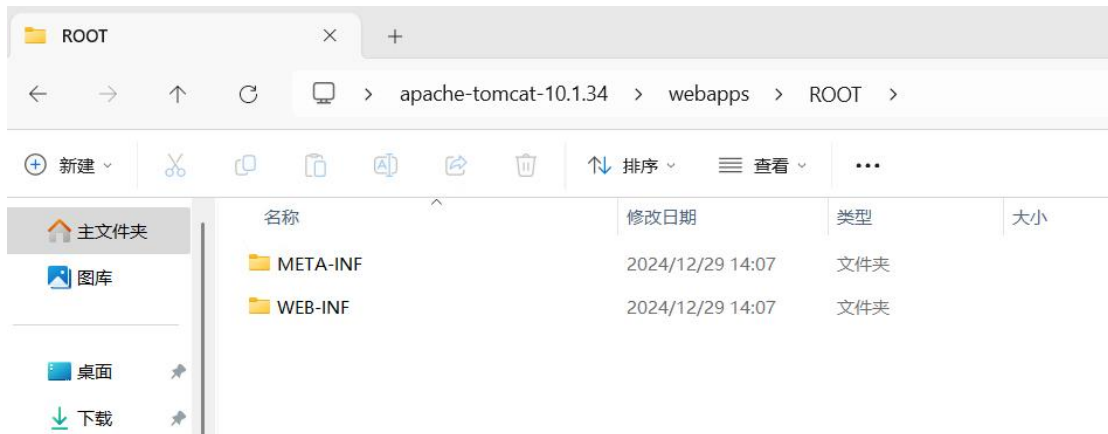
打包前端程序

```
PS D:\Projects\GitProjects\szu-web-oj> npm run build
```

```
> mooc@0.0.0 build
```

```
> vite build
```

后端程序部署到 tomcat，前端程序部署到 nginx（tomcat 也可以）



```
server {  
    listen 81;  
    server_name localhost;  
  
    location / {  
        root C:/dist/;  
        index index.html index.htm;  
    }  
}
```



访问 ip:81 即可访问资源

