JAVA 高级编程大作业

房屋租赁系统

大连理工大学

Dalian University of Technology

目录

一、系统主要的类图

此处的类图按照包进行分类，由于每个包中的类结构类似，每个 package 只展示一副 UML 类图。

1. Common

common 包通常包含项目中通用的类和工具，包括：

图 1 common 包

- **常量类**：存放项目中使用的常量值--状态码。
- **响应封装类**： Result 类，用于统一 API 的响应格式。

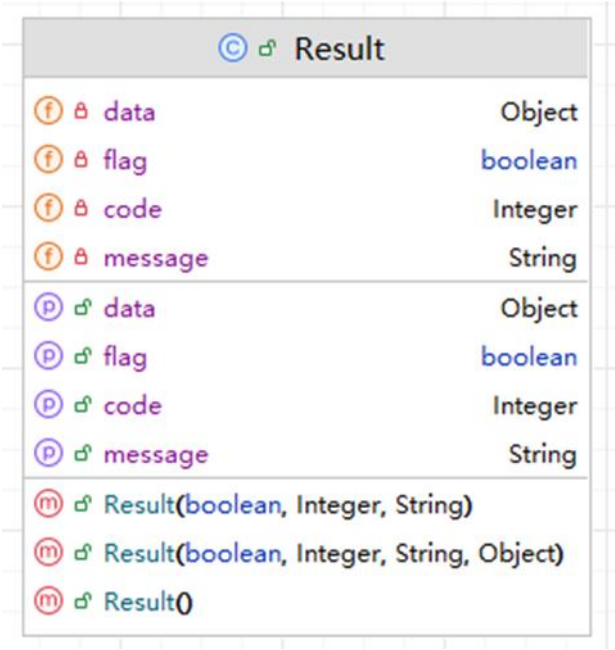


图 2 Result 类图

2. Config

`config` 包用于存放项目的配置类，本项目包含 DataSourceDConfig、WebSecurityConfig 两个配置类。

- **DataSourceDConfig**：主要配置数据库连接信息，用于与 Druid 数据源连接。
- **WebSecurityConfig**：安全配置类。

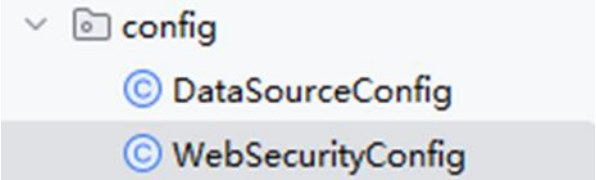


图 3 config 包

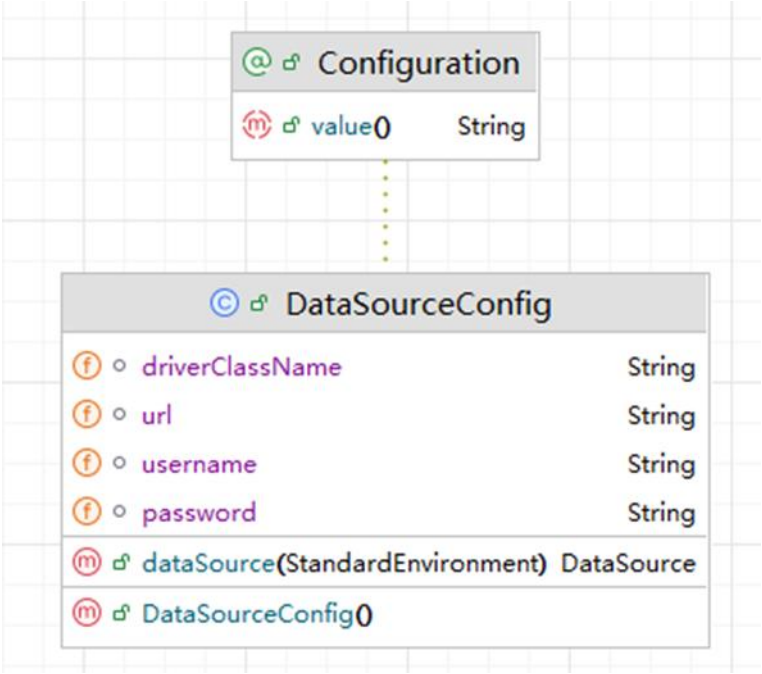


图 4 DataSourceConfig 类

3. Controller

`controller` 包包含控制器类，这些类处理 HTTP 请求，并调用服务层的逻辑来处理请求。

图 5 controller 包

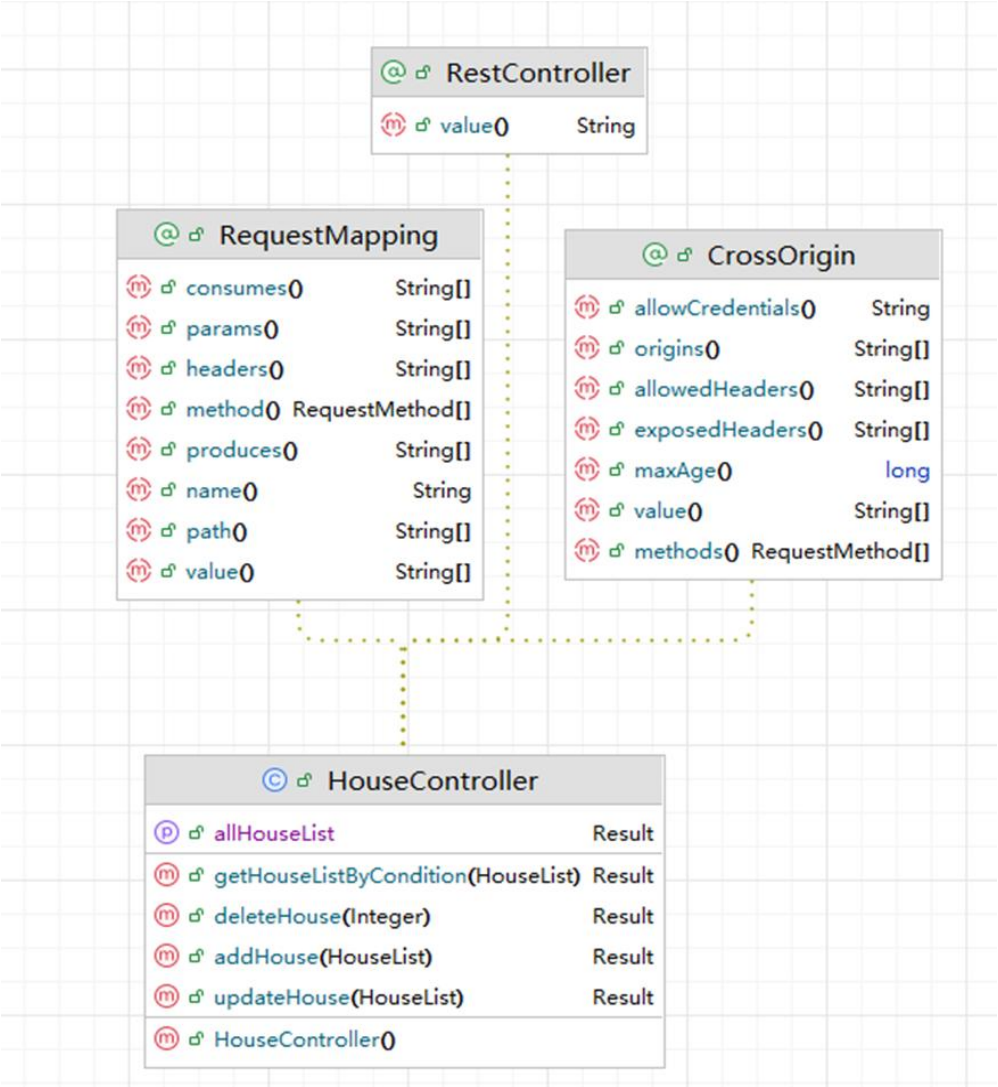


图 6 HouseController

4. Dao

dao（数据访问对象）包用于存放数据访问层的接口和实现类，这些类负责与数据库进行交互。

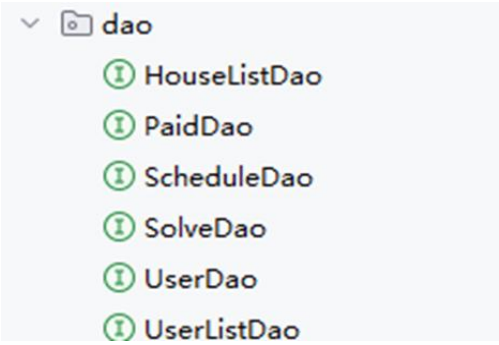


图 7 dao 包

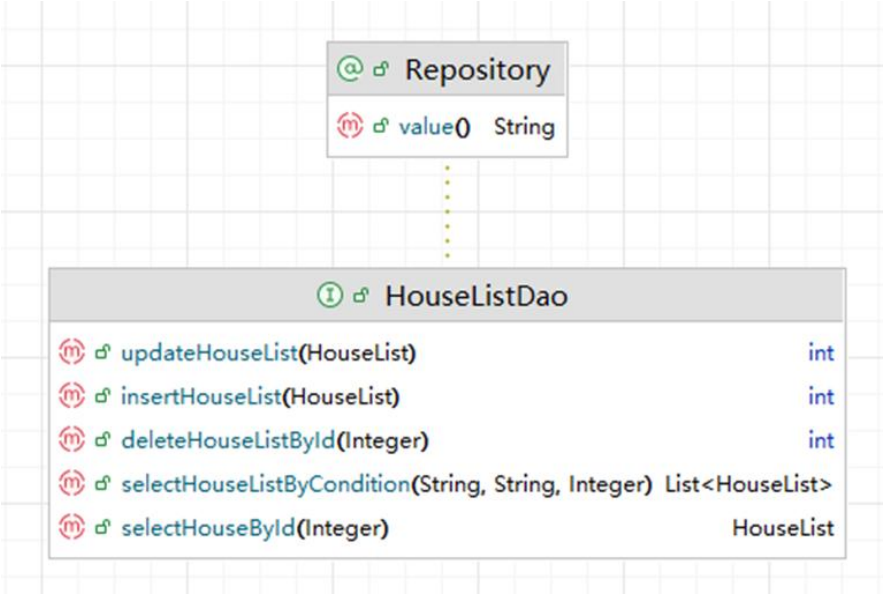


图 8 HouseListDao 类

5. dto

dto（数据传输对象）包用于存放传输数据的对象，这些对象用来在不同层之间传递数据。

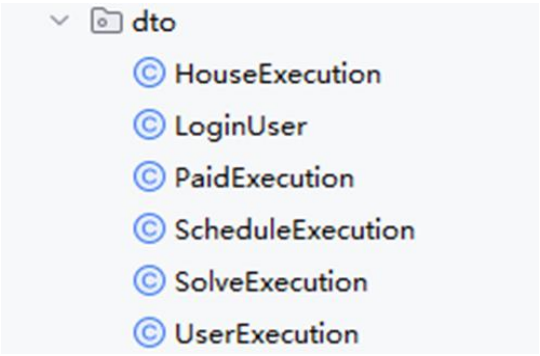


图 9 dto 包

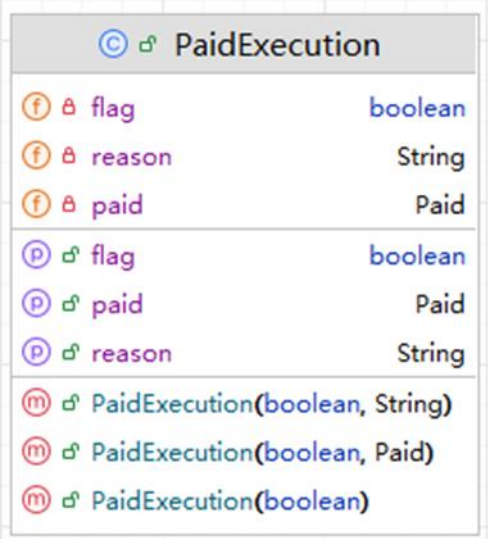


图 10 PaidExecution

6.Exception

`exception` 包用于存放项目中的异常处理类，这些类包括自定义异常和全局异常处理。

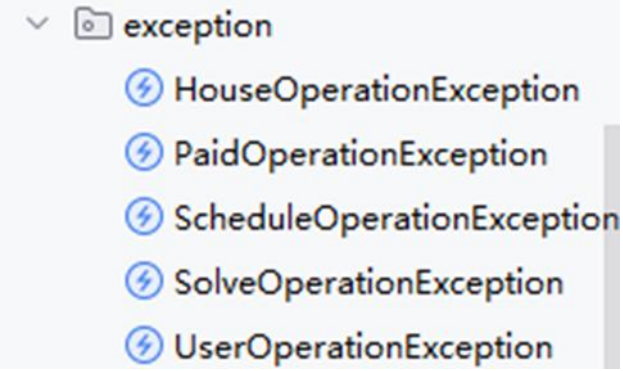


图 11 exception 包

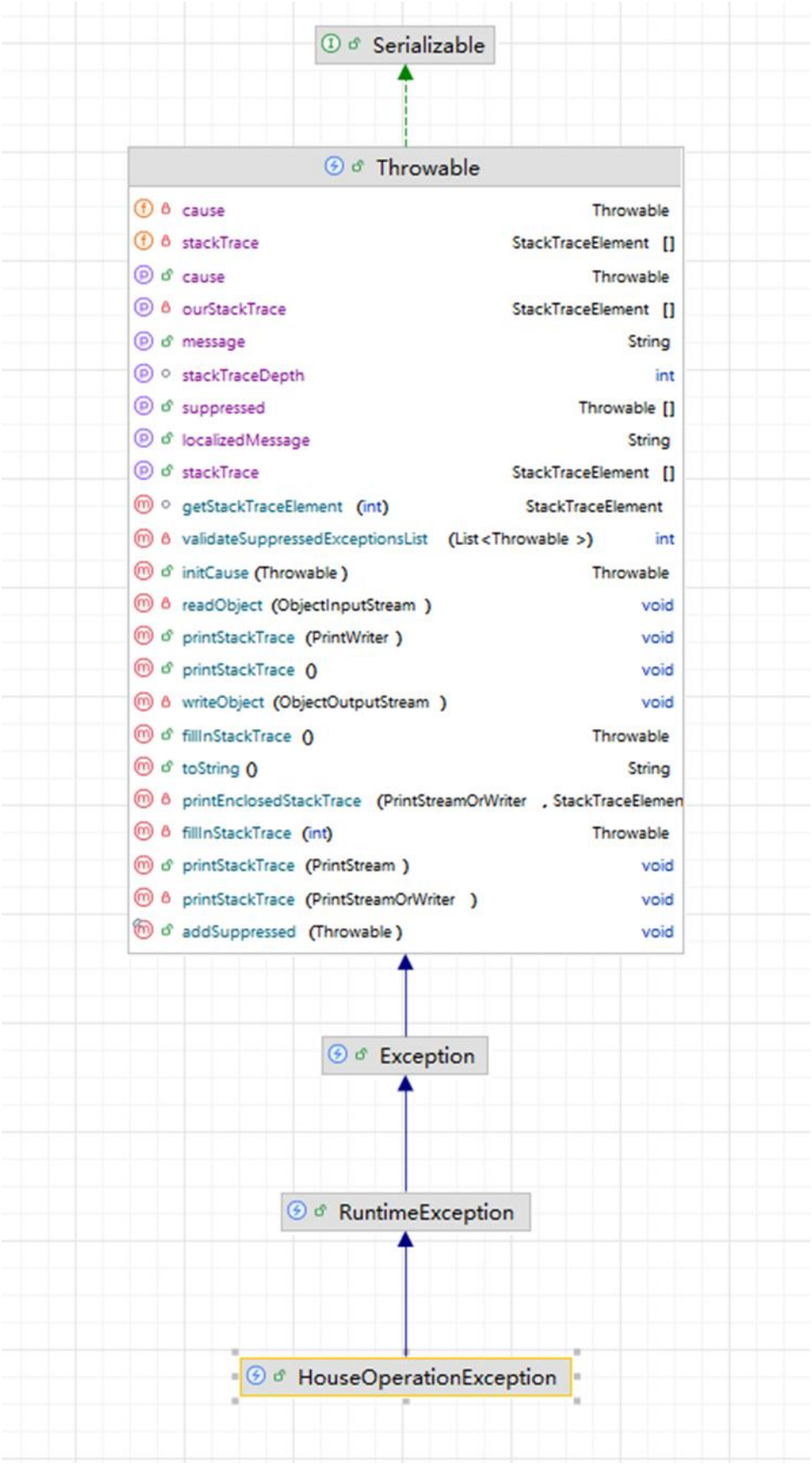


图 12 HouseOperationException

7. pojo

pojo（普通 Java 对象）包用于存放项目中的实体类，这些类与数据库中的各表结构对应。

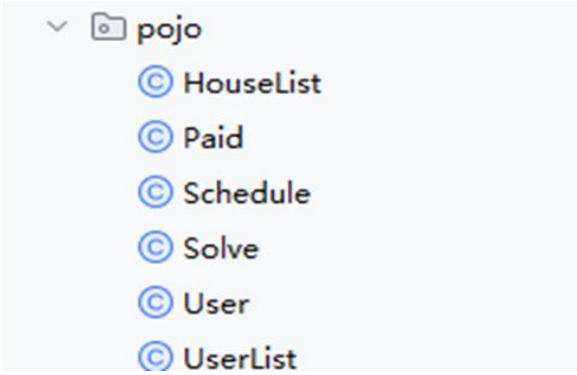


图 13 pojo 包

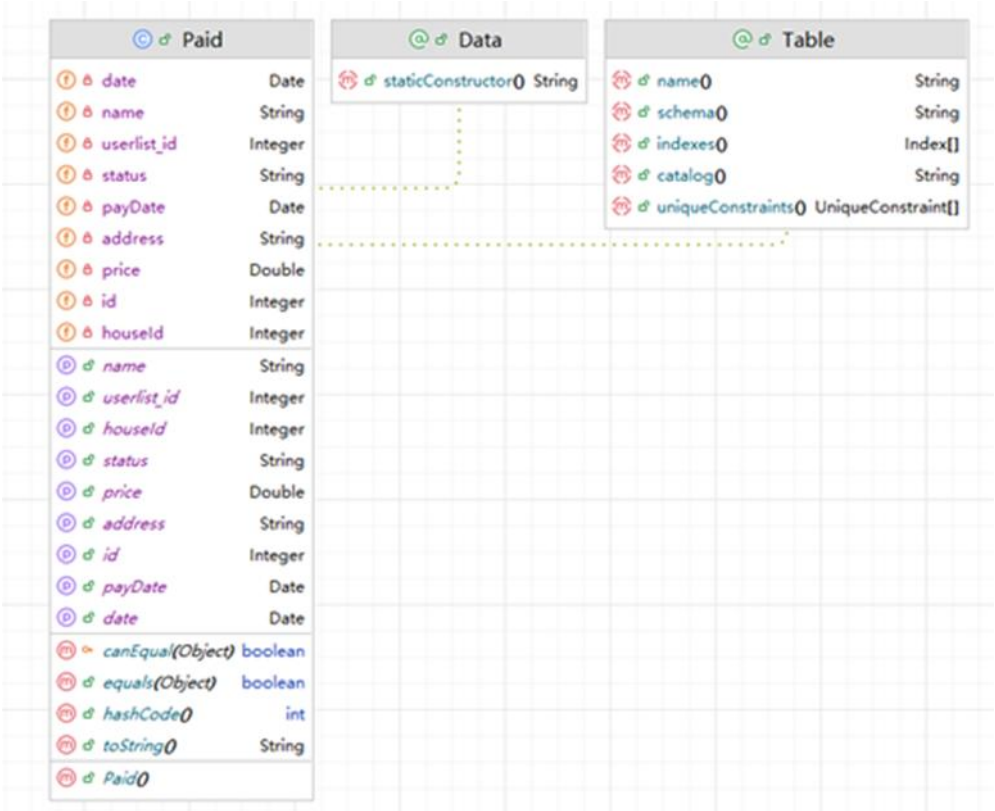


图 14 Paid 类

8. service

`service` 包用于存放业务逻辑层的接口和实现类，这些类包含项目中的业务逻辑。

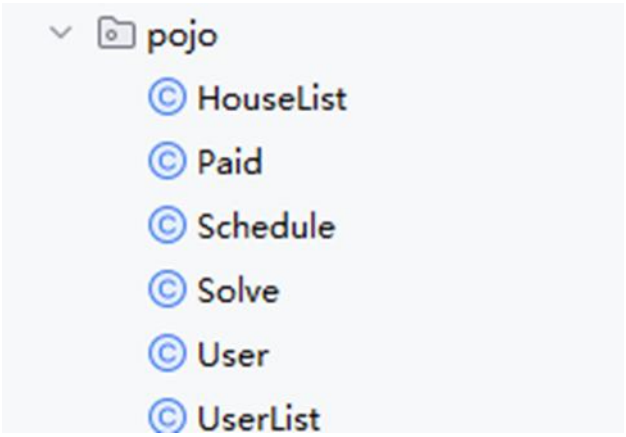


图 15 pojo 包

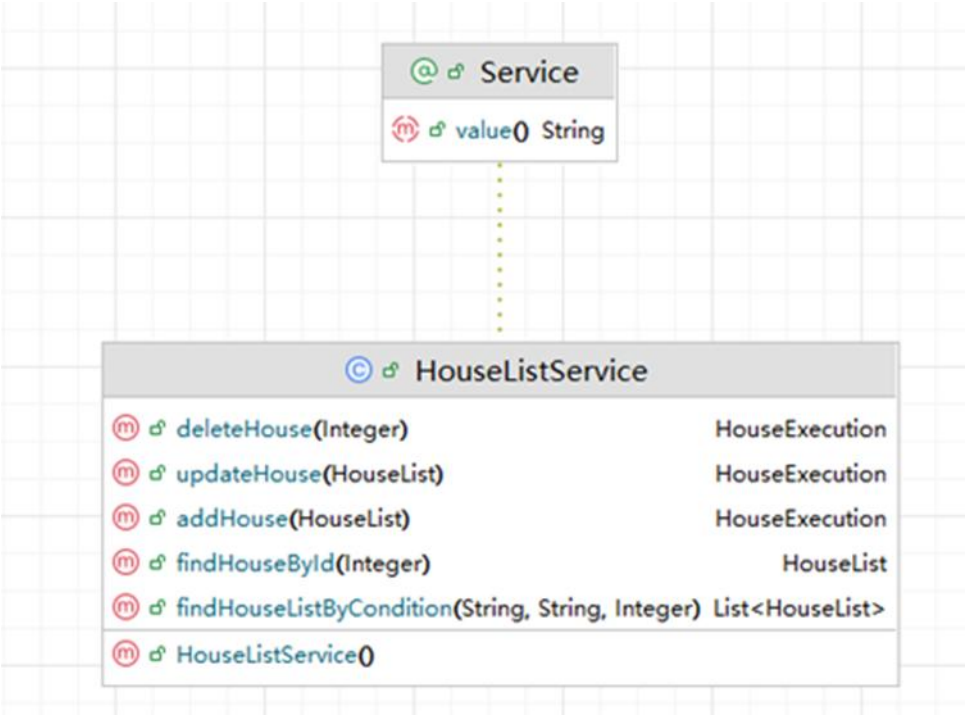


图 16 HouseListService 类

9.utils

`utils` 包用于存放项目中的工具类，这些类封装了一些常用的功能和方法。

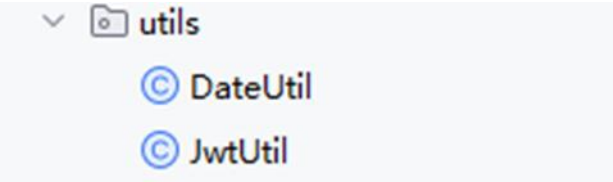


图 17 utils 包

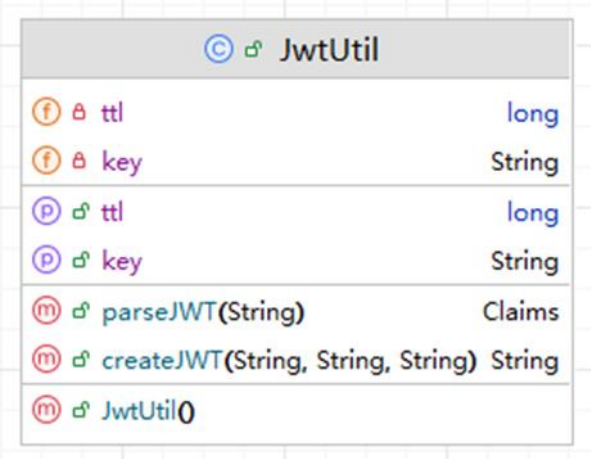


图 18 JwtUtil 类

10.vo

vo（值对象）包用于存放视图对象，本项目中主要用于向前端传送登录相关的数据。

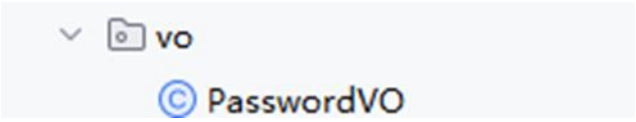


图 19 vo 包

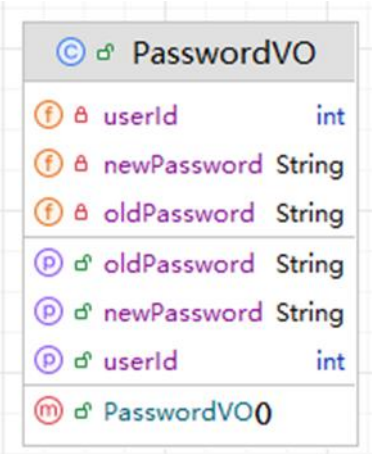


图 20 PasswordVO 类

二、系统数据库设计

数据库表汇总

表名	功能说明
fly_way_schema_history (数据库迁移信息表)	用于跟踪和记录每次数据库迁移的详细信息，确保迁移过程的可追溯性和可靠性。
paid （用户支付租金信息表）	存储房屋租金和支付基本信息
userlist （用户信息表）	存储用户个人信息
solve （用户报修信息表）	存储用户报修信息
houselist（房屋信息表）	存储房屋基本信息
schedule（系统公告信息表）	用于存储管理员发布的公告信息
user （用户账号密码表）	存储用户的账号和密码

图 21 数据库表汇总

用户信息表

编号	名称	描述	数据类型	大小	备注
1	*id*	用户记录的唯一标识符	int		主键
2	*name*	用户的姓名	varchar	255	
3	*idcard*	用户的身份证号码	varchar	255	
4	*phone*	用户的电话号码	varchar	255	
5	*userid*	用户的ID	int		用于关联user表中的用户记录

图 22 用户信息表

用户支付租金信息表

编号	名称	描述	数据类型	大小	备注
1	*id*	付款记录的唯一标识符	int		主键
2	*address*	租赁房屋的地址	varchar	255	
3	*price*	付款金额	double	10,2	最多10位数，其中小数点后有2位
4	*date*	创建日期	date		
5	*paydate*	实际付款的日期	date		
6	*name*	付款人或租户的名称	varchar	255	
7	*userlist_id*	与付款相关联的用户列表中的用户ID	int		
8	*status*	付款状态	varchar	255	
9	*houseid*	与付款记录相关联的房屋ID	int		

图 23 用户支付租金信息表

房屋信息表

编号	名称	描述	数据类型	大小	备注
1	*houseid*	房屋记录的唯一标识符	int		
2	*address*	房屋的地址	varchar	255	
3	*price*	房屋的租金价格	double	10,2	最多10位数，其中小数点后有2位
4	*status*	房屋的状态	varchar	255	例如，已租出、空闲
5	*detail*	房屋的详细信息	varchar	2048	
6	*userlist_id*	用户信息表里的id	int		与房屋记录关联的用户列表ID
7	*userlist_name*	用户信息表里的住户姓名	varchar	64	与房屋记录关联的用户姓名

图 24 房屋信息表

数据库迁移信息表

编号	名称	描述	数据类型	大小	备注
1	*installed_rank*	每个迁移的安装顺序	int		主键
2	*version*	迁移的版本号	varchar	50	
3	*description*	迁移的描述	varchar	200	
4	*type*	迁移的类型	varchar	20	指示迁移文件的类型（如 SQL 脚本、Java 代码等）
5	*script*	迁移脚本的名称或路径	varchar	1000	
6	*checksum*	迁移脚本的校验和	int		
7	*installed_by*	执行迁移的用户的名称	varchar	100	
8	*installed_on*	迁移执行的时间戳	timestamp		记录具体的执行时间
9	*execution_time*	执行迁移所花费的时间（以毫秒为单位）	int		
10	*success*	迁移是否成功执行的标志	tinyint	1	1 表示成功，0 表示失败。

图 25 数据库迁移信息表

用户账号密码表

编号	名称	描述	数据类型	大小	备注
1	*id*	用户账号唯一标识符	int		主键
2	*username*	用户账号名称	varchar	255	
3	*password*	用户密码	varchar	255	
4	*type*	用户类型	int		1表示管理员，2表示普通用户

图 26 用户账号密码表

用户报修信息表

编号	名称	描述	数据类型	大小	备注
1	*id*	报修记录的唯一标识符	int		主键
2	*address*	报修地址	varchar	255	
3	*date*	报修日期	date		
4	*detail*	报修细节	varchar	255	
5	*name*	住户名字	varchar	255	
6	*userlist_id*	用户信息表里的id	int		用于关联userlist表中的用户记录
7	*status*	报修状态	varchar	255	
8	*house_id*	房屋id	int		用于关联house_list表中的用户记录

图 27 用户报修信息表

系统公告信息表

编号	名称	描述	数据类型	大小	备注
1	*id*	系统公告的唯一标识符	int		主键
2	*date*	公告发布日期	date		
3	*content*	公告发布内容	varchar	255	
4	*time*	公告持续天数	int		

图 28 系统公告信息表

三、主要功能设计

我们设计的房屋租赁系统有五大模块：1.房屋信息管理；2. 租金管理；3. 故障管理；4. 公告管理；5. 用户管理。

首先具有登录功能，使用 mybatis 实现后端与数据库的交互,有租赁用户和管理员两种身份

a. **对于管理员：**

1、 管理员可以对房屋信息进行增删改查 (在房屋信息的详情中注明短租还是长租还是日租)、可以通过地址查询房屋信息

2、 管理员可以进行对租客的租金管理（租金显示、租金是否缴纳、延期缴纳）

3、 管理员只可以对故障的状态进行修改，而不能增加故障

4、 公告系统中可以实现增删改查

5、 管理员可以对用户信息进行增删改查，也可以新增管理员、但第一个管理员由设计者直接放入数据库

b. **对于租赁用户：**

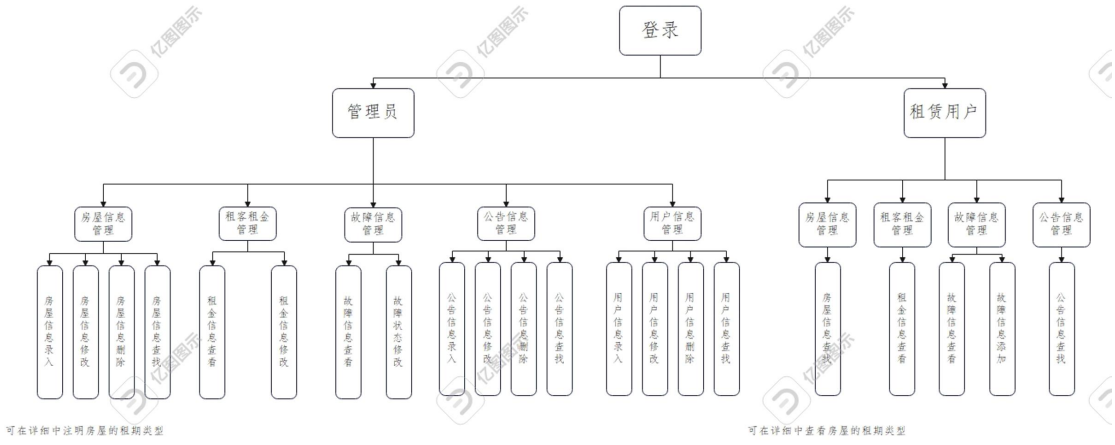
1、 用户可以查看房屋信息（如房屋地址、价格，是否已经出租、点击详细可以看否短租还是长租还是日租）、可以通过地址查询相关房屋

2、 用户可以查看当前自身租赁房屋的租金以及应缴纳时间以及缴纳状态

3、 在故障管理模块中，可以进行故障申报（增加）

4、 进行公告查询

功能框图如下：

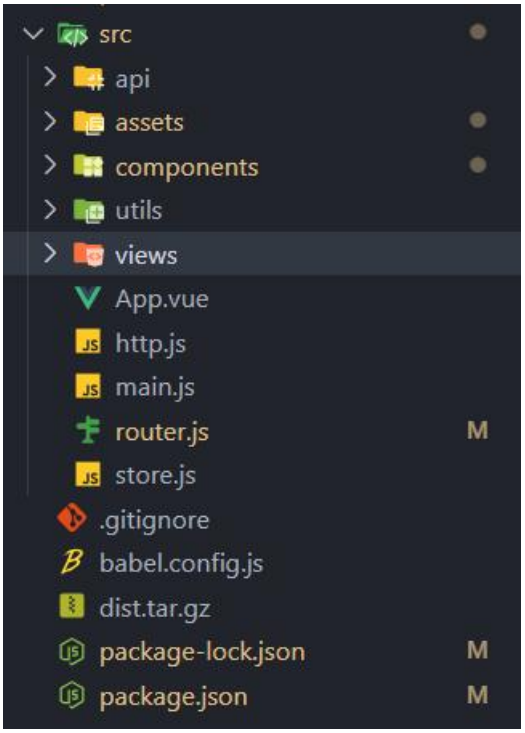


四、主要代码设计

我们的项目是基于 vue + springboot 的房屋租赁管理系统，是一个典型的 MVC 模型架构。

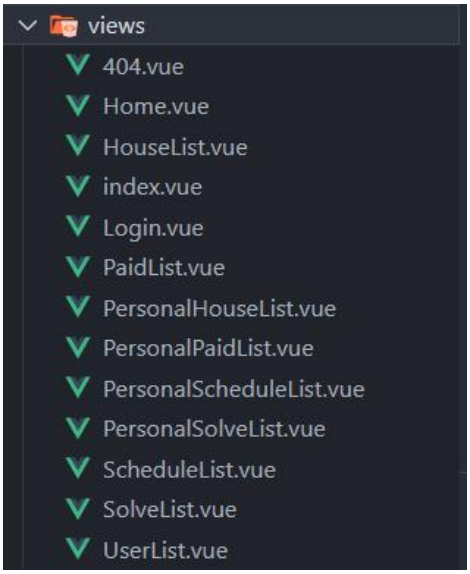
4.1 前端部分

前端使用 vue 2.0 作为 JavaScript 开发框架，下面是我们的前端代码结构：



1. src 目录：这是源代码目录，包含项目的主要代码和资源。
   ① api 目录：存放与后端接口交互的 API 请求和响应处理 js 脚本。
   ② assets 目录：用于存放静态资源，这里放置的是图像。
   ③ components 目录：存放 Vue 组件，每个组件对应一个.vue 文件，包含模板、脚本和样式。
   ④ utils 目录：存放实用工具函数和辅助代码，这些代码通常在多个地方使用。
   ⑤ views 目录：存放视图组件，这些组件通常与路由对应，定义了应用程序的不同页面。
2. App.vue：主应用组件，是应用程序的根组件。
3. http.js：用于配置和实例化 HTTP 客户端，用于处理 API 请求。
4. main.js：应用的入口文件，初始化并挂载 Vue 实例。
5. router.js：定义应用的路由，用于在不同视图组件之间导航。
6. store.js：用于状态管理，集中管理应用的状态。

在这里，我们共实现了 11 个页面视图，其中包括 4 个普通用户视图、5 个管理员视图和 2 个通用视图，每个视图由不同组件组成，具体的实现页面在实现界面中介绍。



4.2 后端部分

在后端我们首先用了实现后端的基本模板，将项目分为了 dao 层、service 层、controller 层，他们分别的作用是：

1. dao 层：其中的每个 java 文件是一个与数据库中的表相对应的接口，接口中每个对数据库操作的实现我们采用了在 resource 文件夹下的 mapper 中的 xml 文件一一实现，同时是为 service 层提供了接口。

2. service 层：在该层中对于 dao 进行进一步封装以方便为上层提供服务。

3. controller 层：在该层负责请求转发，接受页面过来的参数，传给 service 处理，接到返回值，再传给页面。

Service 层是建立在 DAO 层之上的，建立了 DAO 层后才可以建立 Service 层，而 Service 层又是在 Controller 层之下的，因而 Service 层应该既调用 DAO 层的接口，又要提供接口给 Controller 层的类来进行调用，它刚好处于一个中间层的位置。每个模型都有一个 Service 接口，每个接口分别封装各自的业务处理方法。以上是主要分层是实现功能模块的主要部分，其余层是穿插其中为他们之间的连接更为方便，简介，安全。

其中 dto 是提供 service 与 controller 之间数据传输、common 中定义了返回前端的数据格式以及相关状态码、config 中进行数据库和安全配置、exception 是为了捕获相关的异常、pojo 中定义了项目的相关实体类以及 util 中提供的工具类获取时间与密钥。

在存储我们要录入的房屋信息、录入的租金详情、用户信息、账号密码、公告信息以及故障信息时使用了泛型集合 List，如下是录入房屋信息的代码：

```
List<HouseList> selectHouseListByCondition(@Param("status") String status,@Param("address") String address,@Param("userListId") Integer userListId);
```

其中@Param 注释用于 SQL 语句的参数赋值。

为了实现 SQL 语句以及 java 语言中对于相关数据进行增删改查操作，我们在 dao 层实现了各个模块对应的操作接口，例如对于房屋信息的增删改查操作：

```
public interface HouseListDao {
    List<HouseList> selectHouseListByCondition(@Param("status")
String status,@Param("address") String address,@Param("userListId")
Integer userListId);
    HouseList selectHouseById(Integer houseId);
    int deleteHouseListById(Integer houseId);
    int insertHouseList(HouseList houseList);
    int updateHouseList(HouseList houseList);
}
```

而调用这些接口的在 service 层以及在 Mapper 中的 xml 文件对于数据库的操作。

在设计精妙之处，首先在结构方面，我们没有纯粹的套用后端模板，使得 service 层和 controller 层之间的数据传输在内部实现，这样会使得程序变得臃肿难以理解，所以我们对于其中的每一个模块在 dto 层中实现了相应的模块实现他们两层之间的数据传输，更加简洁好看、便于理解。对此我们给出其中一个方法在传递参数时所经历的历程，以下是 service 层中的语句：

```
public HouseExecution addHouse(HouseList houseList) {
    if(houseList == null){
        return new HouseExecution(false,"添加房屋信息为空");
    }
    try{
        int effectedNum = houseListDao.insertHouseList(houseList);
        if(effectedNum < 1){
            return new HouseExecution(false,"添加房屋信息失败");
        }
    }catch (Exception e){
        throw new HouseOperationException(e.toString());
    }
    return new HouseExecution(true);
}
```

以下是 controller 层中的语句

```
public Result addHouse(@RequestBody HouseList houseList){
    HouseExecution he;
    try{
        he = houseListService.addHouse(houseList);
        if(he.isFlag()){
            return new Result(true,StatusCode.SUCCESS,"添加房屋信息成
功");
        }else {
            return new Result(false,StatusCode.ERROR,"添加房屋信息失
败：" + he.getReason());
        }
```

```
    }catch (Exception e){
        return new Result(false,StatusCode.ERROR,"添加房屋信息失败："+
e.toString());
    }
}
```

以上实现的是添加房屋信息的功能，可见当添加房屋信息成功时，service 层会返回所要传递给 controller 的信息即：return new HouseExecution(true);由 controller 中的 HouseExecution 实例化对象 he 接收：
he = houseListService.addHouse(houseList);

我们还创建了一个 exception 包用于实现自定义各个模块的对应的异常处理，不在捕获对应异常的模块中定义，使得代码也更加简洁美观，也便于清晰的知道发生异常是哪个模块的问题。例如：

```
public class HouseOperationException extends RuntimeException{

    private static final long serialVersionUID = -2365905813758097384L;

    public HouseOperationException(String msg){
        super(msg);
    }
}
```

然后是代码内容方面，首先就不得不说一个非常非常好使的一个注释：

```
@Autowired
```

其对应的语句看上去是实例化注入了一个接口，但实际上是注入了一个实现相关接口的类对象，大大简化了我们实例化一个类的代码重复，非常好用！！！

```
private HouseListDao houseListDao;
```

在这之后我们可以直接只用 houseListDao 对象去实现接口中的方法。

此外，就是实现与前端交互请求转发、接受页面传递来的参数的 controller 层。

在该层中，为了实现前后端交互就得使用到@RestController 注解，例如：

```
@RestController
@CrossOrigin
@RequestMapping(value="/house")
public class HouseController {
@RequestMapping(value = "/getallhouselist",method = RequestMethod.GET)
public Result getAllHouseList(){
    List<HouseList> houseList =
houseListService.findHouseListByCondition(null,null,null);
    return new Result(true, StatusCode.SUCCESS,"查找房屋信息列表成功
",houseList);
}
}
```

这是 HouseController 中类的其中一个方法。

我们使用 @RestController 注解将 HouseController 类标记为 RESTful API 控制器。我们使用 @RequestMapping 注解将请求 URL 映射到 /house 路

径，在 getAllHouseList 方法中，我们返回在 common 中定义的 Result 类，即返回前端的信息格式，它将被转换为 JSON 响应体并发送回客户端。

还有对于前端传来的请求体我们需要做出相对应的回应，但显然请求体的格式是不符合我们在后端所定义的实体以及对该实体进行相关操作的方法的，所以我们引入了@Requestbody 注解，这个注解可以帮助我们把请求体转化为相对应的实体对象，再传递给方法

4.3 数据库部分

使用 MySQL 数据库+Druid 数据库连接池+MyBatis 框架来实现高效、可靠的数据访问层。

1. 项目分层设计

项目采用典型的三层架构设计：

- **控制层（Controller）**：负责处理客户端请求，并调用服务层完成业务逻辑。
- **服务层（Service）**：封装业务逻辑，调用数据访问层进行数据库操作。
- **数据访问层（DAO/Mapper）**：负责与数据库进行交互。

2. 泛型集合的使用

在控制层和服务层中，经常会使用泛型集合来存储和传递数据。例如，获取所有用户列表时，使用 List<User> 来存储用户数据。

```
// Service 层方法，返回所有房屋信息

 public List<HouseList> findHouseListByCondition(String status,String
 address,Integer userListId) {

        return houseListDao.selectHouseListByCondition(status,address,
userListId);

 }


// Controller 层方法，返回房屋列表

public Result getAllHouseList(){

        System.out.println("*获取所有房屋列表:");

        List<HouseList> houseList = houseListService.findHouseListByC
ondition(null,null,null);

        System.out.println("**已获取房屋列表: " + houseList);

        return new Result(true, StatusCode.SUCCESS,"查找房屋信息列表成
功",houseList);
```

```
}
```

3. 接口的使用

接口在服务层和数据访问层广泛使用，以实现松耦合和可扩展性。例如：

```
//DAO 层
public interface HouseListDao {

    List<HouseList> selectHouseListByCondition(@Param("status") Stri
ng status,@Param("address") String address,@Param("userListId") Integ
er userListId);

    HouseList selectHouseById(Integer houseId);

    int deleteHouseListById(Integer houseId);

    int insertHouseList(HouseList houseList);

    int updateHouseList(HouseList houseList);

}
```

4. 公共类的使用

公共类用于封装一些通用的功能或常量，例如 Result 类用于封装 API 的返回结果，StatusCode 类用于定义状态码。

```
// Result 类
public class Result {

    private boolean success;

    private int code;

    private String message;

    private Object data;


    public Result(boolean flag, Integer code, String message) {

        this.flag = flag;

        this.code = code;

        this.message = message;


    }
```

```
    public boolean isFlag() {
        return flag;
    }
    // getter 和 setter 方法
}

// StatusCode 类
public class StatusCode {
    public static final int SUCCESS = 20000;    //成功
    public static final int ERROR = 20001;      //失败
    public static final int LOGINERROR = 20002; //用户名或密码错误
    public static final int ACCESSERROR = 20003;//权限不足
    public static final int REMOREERROR = 20004;//远程调用失败
    public static final int REPERROE = 20005;   //重复操作
}
```

5. 分层设计的精妙之处

- **控制层 (Controller)**：控制层只处理请求和返回结果，具体的业务逻辑委托给服务层。例如：
- **服务层 (Service)**：服务层封装了具体的业务逻辑，调用数据访问层进行数据库操作。例如：
- **数据访问层 (DAO/Mapper)**：数据访问层通过 MyBatis 与数据库进行交互，使用 XML 或注解定义 SQL 语句。例如：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.house.dao.HouseListDao">
    <resultMap id="HouseResultType" type="com.house.pojo.HouseList">
        <!--省略-->
    </resultMap>
```

```
    <select id="selectHouseListByCondition" resultMap="HouseResul
    tType">
        <!--省略-->     </select>
    <!--省略-->
</mapper>
```

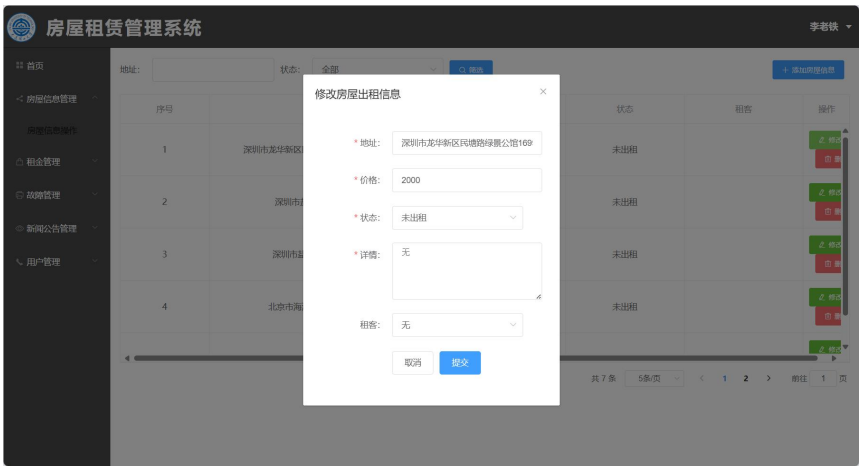
通过上述设计，可以实现一个结构清晰、松耦合、高可维护性的项目架构：这种设计使得代码结构清晰，各层职责分明，易于扩展和维护。

五、实现界面

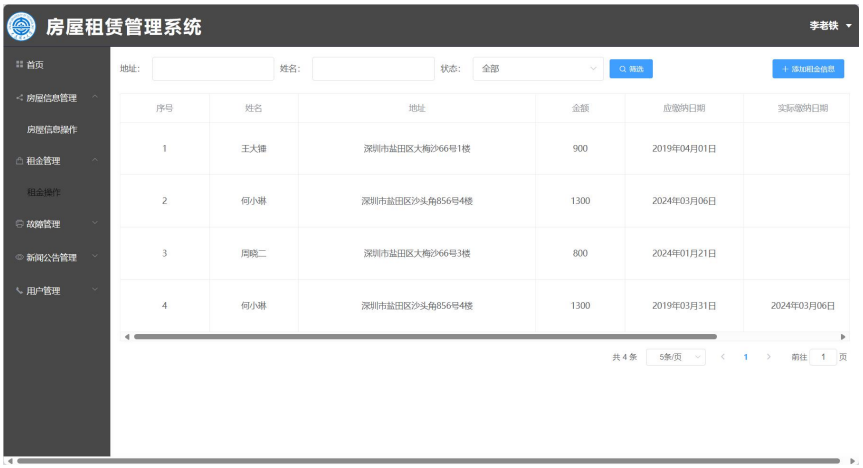
1. 登录界面

2. 主界面

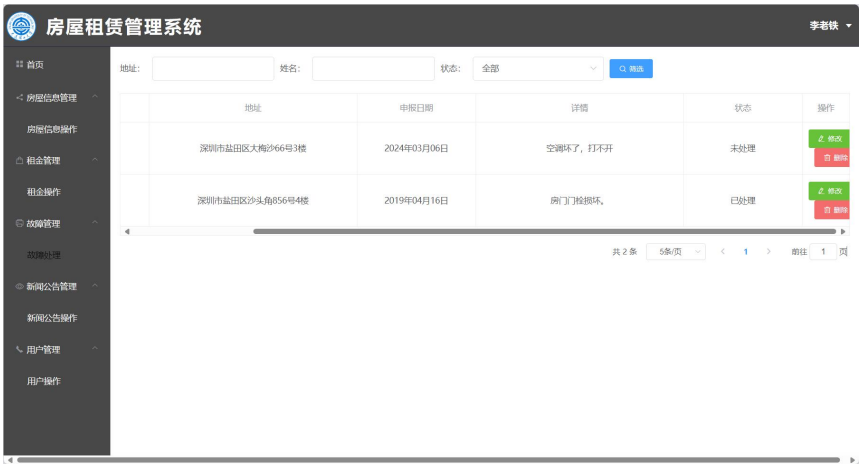
3. 管理员房屋管理界面，可添加、修改、删除和筛选房屋信息。

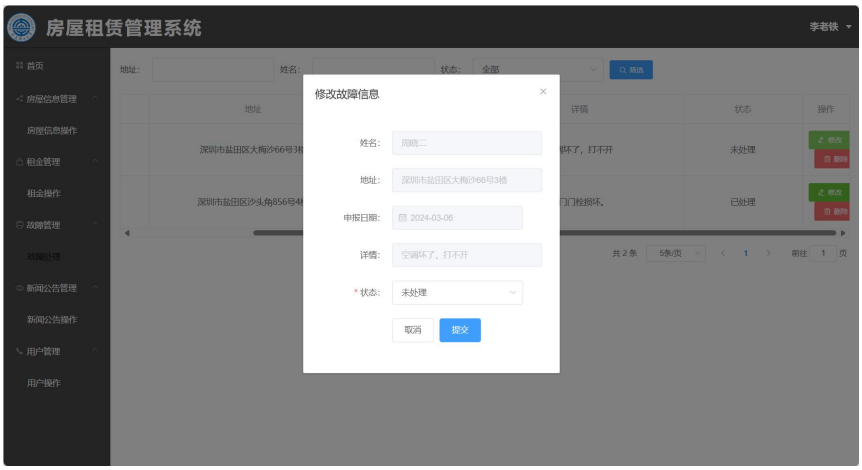


4. 管理员租金管理界面，可添加、修改、删除和筛选租金信息。

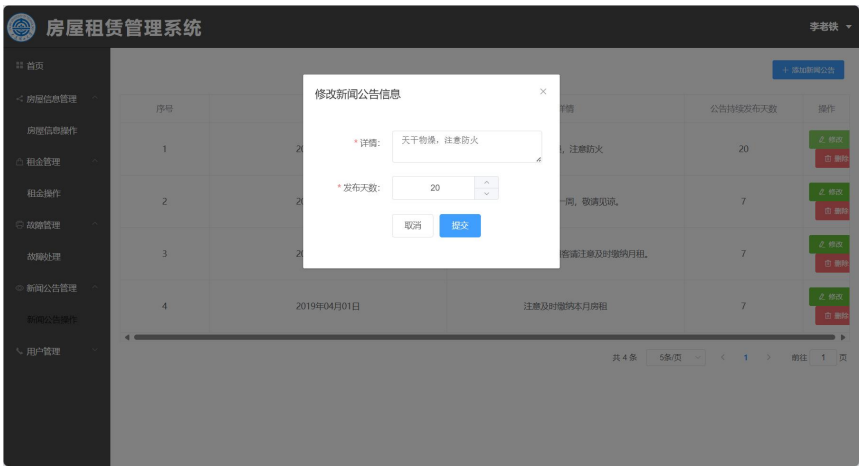


5. 管理员故障管理界面，可修改、删除和筛选故障信息。

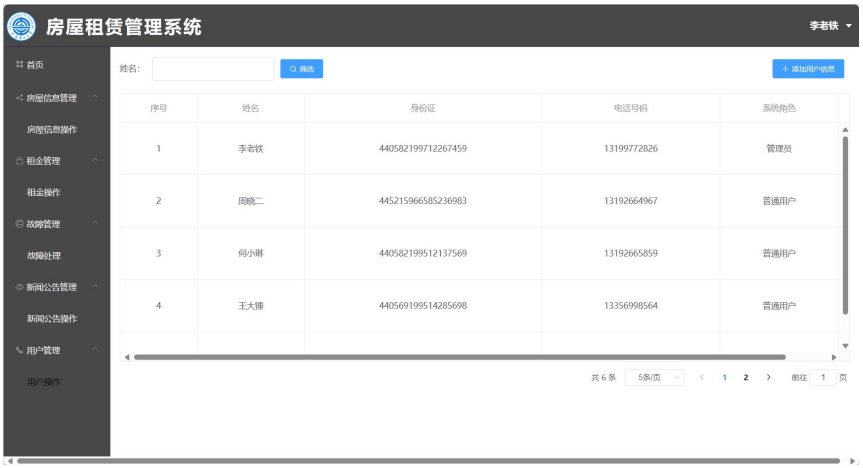




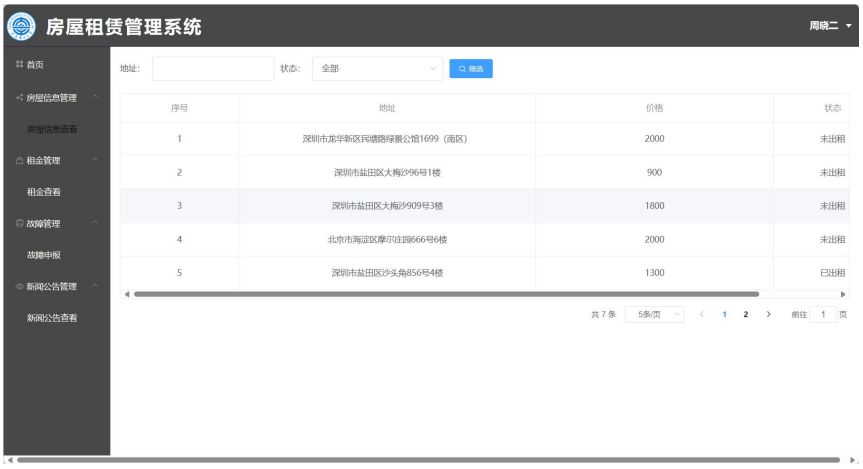
6. 管理员新闻公告界面，可添加、修改、删除和筛选新闻公告信息。



7. 管理员用户管理界面，可添加、修改、删除和筛选新闻公告信息。



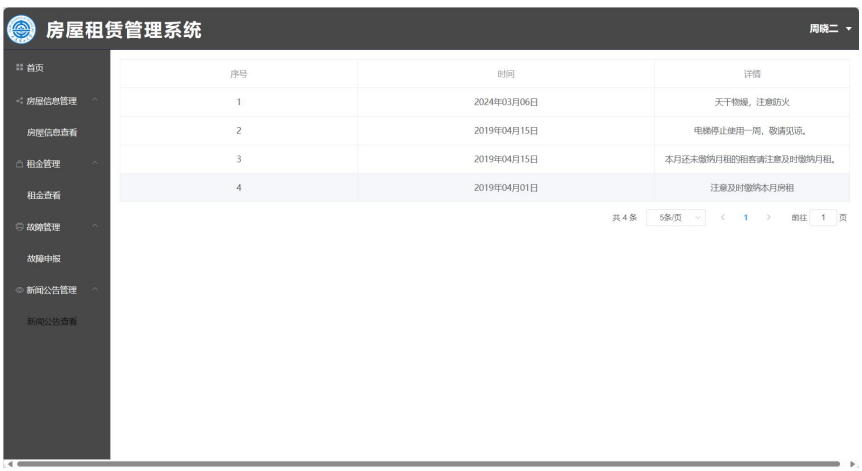
8. 普通用户可查看房屋信息



9. 普通用户可查看租金屋信息

10. 普通用户可查看各种故障申报

11. 普通用户可查看各种新闻公告



六、对课程的体会

后端部分（李瀚嵘）：

上完 Java 高级编程课程，让我受益匪浅。在课程中，我不仅深入理解了 Java 语言的高级特性，还学会了如何应用这些特性解决实际的软件开发问题。课程内容涵盖了面向对象设计原则、并发编程、异常处理等重要主题，这些知识不仅拓展了我的编程视野，也增强了我解决复杂问题的能力。

首先，在课程中，我深入学习了面向对象的设计思想和设计模式。通过案例分析和实践项目，我掌握了如何合理地设计和组织代码结构，提高了代码的复用性和可维护性。特别是学习了单例模式、工厂模式等常用设计模式，这些模式不仅在理论上加深了我的理解，而且在实际项目中能够直接应用，极大地提升了我的开发效率。

其次，课程中的并发编程内容让我对多线程编程有了更深入的理解。通过学习线程的创建与管理、同步与互斥机制等，我学会了如何处理多线程下的资源竞争和死锁问题。这些技能在现代软件开发中尤为重要，能够帮助我编写更加高效和稳定的多线程应用程序。

最后，课程还涉及了 Java 的异常处理、IO 操作以及网络编程等内容。通过这些学习，我不仅掌握了 Java 语言底层操作的技巧，还能够编写网络应用程序和文件处理程序。

总的来说，Java 高级编程课程不仅帮助我夯实了 Java 语言的基础知识，还为我今后的学习和工作打下了坚实的基础。通过这门课程，我不仅成为了一名更加熟练的 Java 开发者，也提升了解决复杂编程问题的能力和信心。这段学习经历将在我的职业生涯中持续发挥重要作用。

前端部分（陈佳辉）：

在参与这个前端项目的开发过程中，我学到了很多关于现代 Web 开发技术和项目管理的知识。以下是我的一些学习体会以及在项目中的具体分工。

这个项目使用了 Vue.js 作为前端框架。通过实际的开发，我对 Vue.js 的组件化开发模式有了更深入的理解。组件化使得代码更加模块化和可复用，特别是在开发复杂的用户界面时，组件之间的通信和状态管理变得更加清晰和简洁。

通过配置和使用 Vuex 进行全局状态管理，我学会了如何在多个组件之间共享状态，避免了因状态不一致导致的 Bug。同时，通过配置 Vue Router，我掌握了如何进行视图组件之间的导航，以及如何处理嵌套路由和动态路由。

在实现 API 接口调用的过程中，我了解了如何与后端进行数据交互，并处理各种 HTTP 请求和响应。通过编写和调试 API 请求代码，我对 Axios 库有了更深入的理解，能够更好地处理请求的拦截、错误处理和数据解析。

参与这个项目让我意识到良好的项目结构和代码规范对团队协作的重要性。通过使用 Git 进行版本控制，我学会了如何在团队中高效地进行代码合并、分支管理和冲突解决。同时，通过代码评审和协作开发，我提升了代码质量和开发效率。

数据库部分（徐星烁）：

之前没有接触过前后端分离的项目，并且也没有学习过连接数据库的知识，所以可以说这一次大作业是一次挑战。不过，得益于刚学完数据库系统这门课程，

整体流程没有想象中的困难。我先自学了 JDBC 的知识，了解了 Java 语言是怎样与数据库进行交互的，先写了几个练手的小项目；接着学习数据库连接池的知识，为了提高性能，我们把 Java 最好的连接池之一 Druid 引入了本次项目，脱离了 DriverManager 这个驱动类，便于管理数据库连接；最后学习的是 MyBatis 这个持久层框架，解决了硬编码的效率问题，通过 XML 配置文件来配置数据库查询等功能，也算是考虑到了日后可能的修改需求。

个人的体会是：难点不在于编写，而是寻找思路。在开始动手之前就需要考虑这个项目使用的是什么数据库，需不需要用到数据库连接池，需不需要通过预编译等手段提高 SQL 语句性能，为了日后修改需不需要引入框架，为了方便前端显示需不要改动返回值形式