软件架构说明 SAD V3.0

1. 引言

1.1 目的及范围

该架构说明文档对 Rho，即 Rho 网上租房系统的系统整体结构进行描述，主要用于明确系统各功能的实现方式。本文档将全面与系统地表述目标软件系统的构架，并通过使用多种视图来从不同角度描述系统的各个主要方面，以满足相关涉众（客戶、设计人员等）对目标系统的不同关注焦点。

本文档记录并表述了架构师对系统构架方面做出的重要决策；项目经理将根据构架定义的构件结构制定项目的开发计划；设计员将据此进行各构件的详细设计；测试设计员按照构架设计系统的总体

1.2 文档结构

文档的组织结构如下：

1.2.1 第一部分 引言

本部分主要概述了文档内容组织结构，使读者能够对文档内容进行整体了解，并快速找到自己感兴趣的内容。同时，也向读者提供了架构交流所采用的视图信息。

1.2.2 第二部分 架构背景

本部分主要介绍了软件架构的背景，向读者提供系统概览，建立开发的相关上下文和目标。分析架构所考虑的约束和影响，并介绍了架构中所使用的主要设计方法，包括架构评估和验证等。

1.2.3 第三、四部分 视图及其之间的关系

视图描述了架构元素及其之间的关系，表达了视图的关注点、一种或多种结构。

1.2.4 第五部分 需求与架构之间的映射

描述系统功能和质量属性需求与架构之间的映射关系。

1.2.5 第六部分 附录

提供了架构元素的索引，同时包括了术语表、缩略语表。

1.3 视图编档说明

所有的架构视图都按照标准视图模板中的同一种结构进行编档。

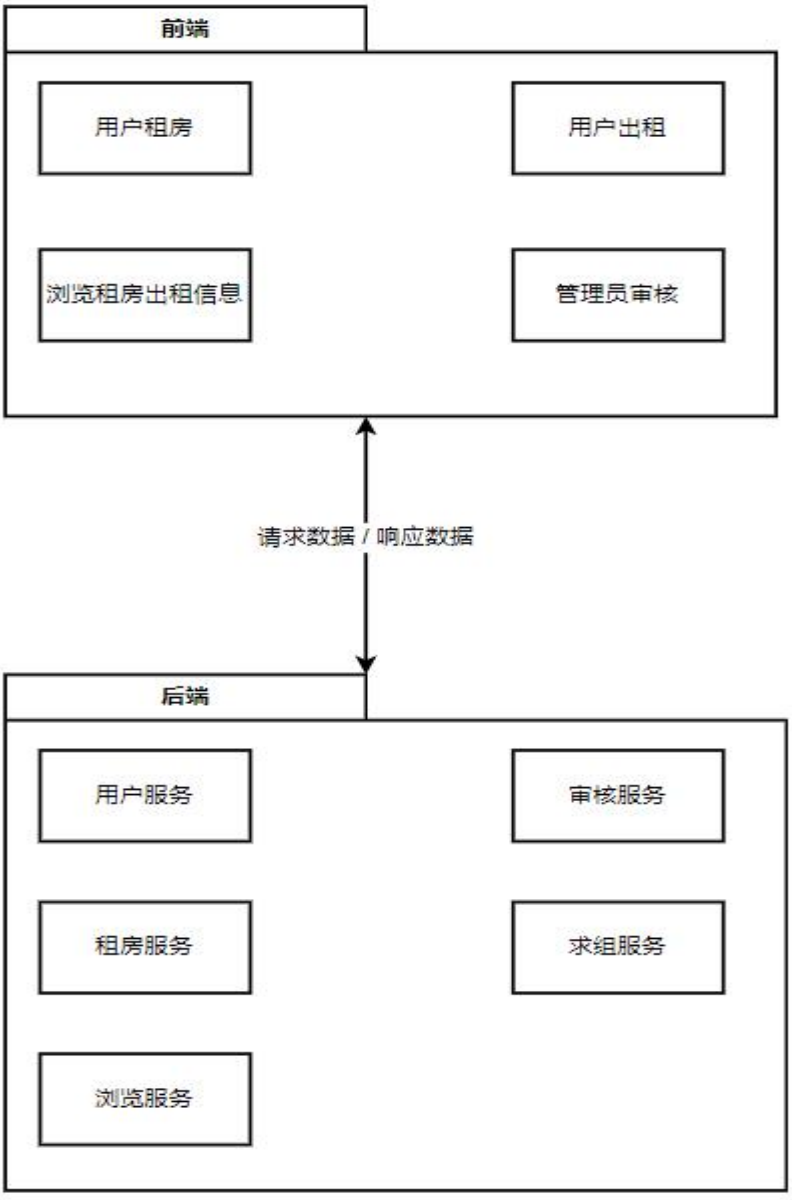
2. 架构背景

3. 视图

3.1 逻辑视图

逻辑视图从系统内在逻辑结构的角度描述系统的基本结构与动态行为，设计模型说明了系统的组成元素、组织架构和关系，并描述了各组成元素的协作以及状态转换关系等。

3.1.1 主表示



3.1.2 构建目录

构建及其特性表

构件	*描述*
前端	包括用户租房、用户出租、浏览租房出租信息、管理员审核页面
后端	包括注册登录、筛选房屋、出租、租房、浏览等服务

各类关系及其特性：

- 依赖/使用关系：
  - 前端依赖后端提供的接口
  - 后端依赖前端传值

元素接口：

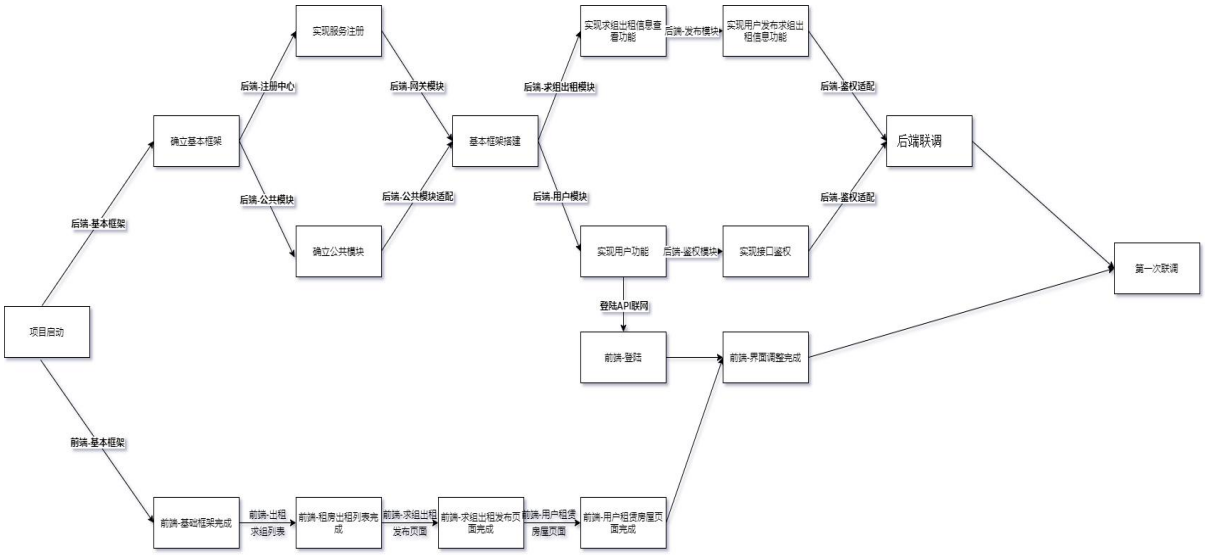
- 前端与后端
  - 查询所有出租房屋信息
  - 查询所有用户求租信息
  - 查询用户个人发布的出租房屋信息
  - 查询用户个人发布的求租信息
  - 查询个人租房信息
  - 用户租房
  - 用户提交请求成为中介
  - 用户登录/登出
  - 管理员审核用户成为中介的请求

3.1.3 可变性

在设计逻辑架构时，我们针对可变性进行了专门的设计，力求将系统耦合性降到最低。代码框架中每 一个接口都独立开来完成，因此当遇到功能扩充或接口变迁的时候，我们能够非常方便、快速地进行

3.2 开发视图

3.2.1 主表示



3.2.2 构件目录

A. 构件及其特性

构件	*描述*
前端	首先搭建整体的框架，然后逐步完善各个部件
后端	首先完成部分公共模块以及用戶登录注册的功能，随后在此基础上进行开发

各类关系及其特性：

- 依赖/使用关系：
  - 前端依赖后端提供的接口
  - 后端依赖前端提供的页面和参数

- 元素接口
  - 前端与后端
    - 查询所有出租房屋信息
    - 查询所有用户求租信息
    - 查询用户个人发布的出租房屋信息
    - 查询用户个人发布的求租信息
    - 查询个人租房信息
    - 用户租房
    - 用户提交请求成为中介
    - 用户登录/登出
    - 管理员审核用户成为中介的请求

元素行为：

- 后端
  - 数据库读写

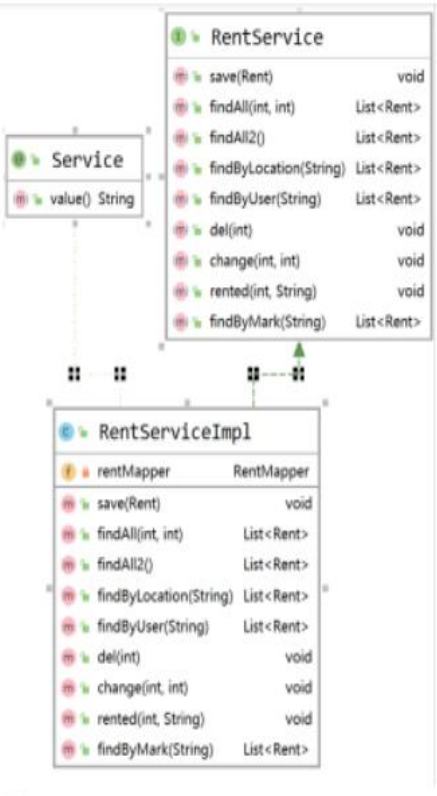
- 记录服务端的工作日志

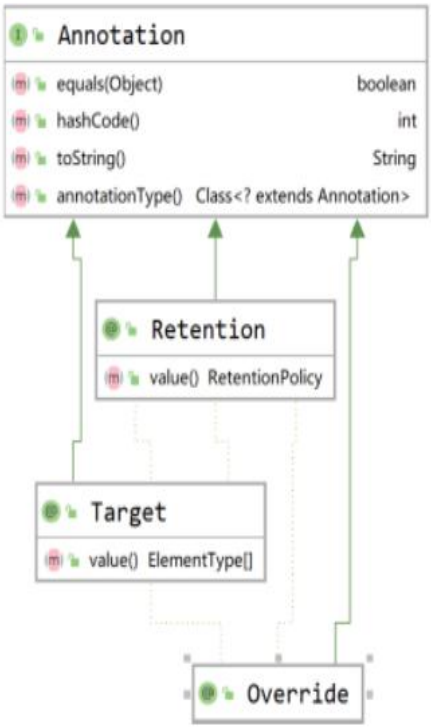
3.2.3 可变性

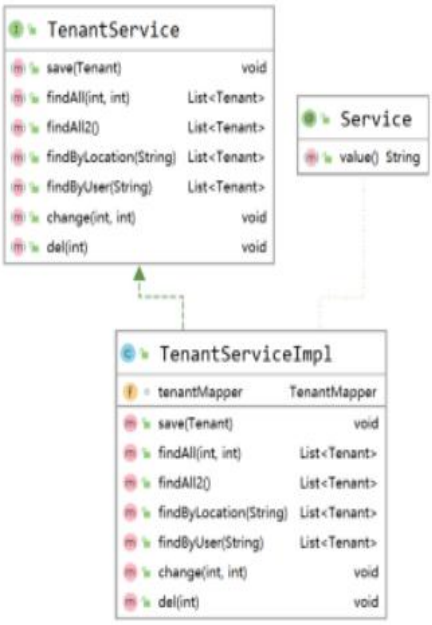
在开发视图的设计过程中，我们明确了每一部分的开发流程，各部分间的开发顺序，做到了开发流程 简单明了、开发路线简洁清晰。因此如果遇到开发技术更换，或开发部件增添的情况时，我们能够很 轻松地修改开发视图，并且不会对其它方向上的开发过程造成影响，具有极高的可变性。

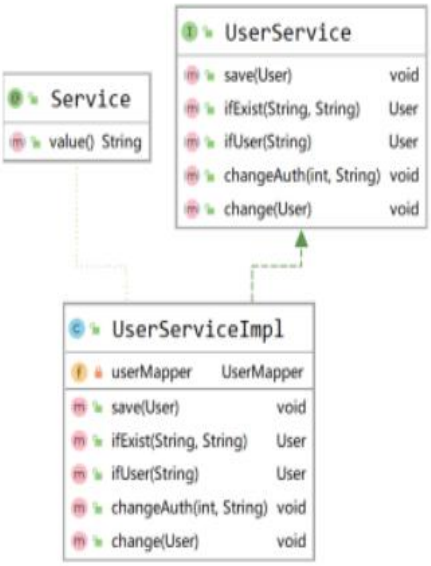
3.3 运行视图

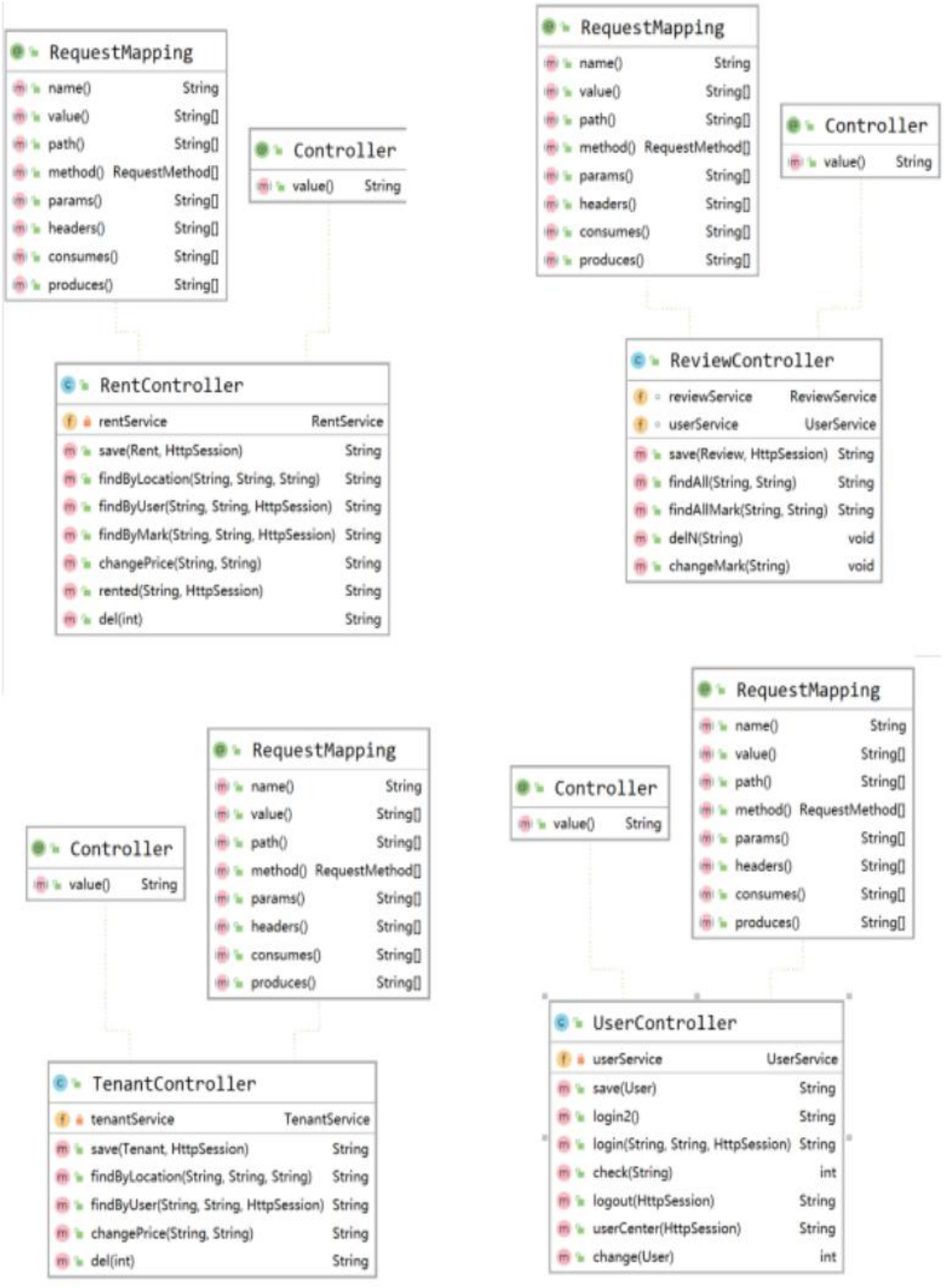
3.3.1 主表示


RentService
save(Rent) void
findAll(int, int) List<Rent>
findAll2() List<Rent>
findByLocation(String) List<Rent>
findByUser(String) List<Rent>
del(int) void
change(int, int) void
rented(int, String) void
findByMark(String) List<Rent>
Service
value() String
RentServiceImpl
rentMapper RentMapper
save(Rent) void
findAll(int, int) List<Rent>
findAll2() List<Rent>
findByLocation(String) List<Rent>
findByUser(String) List<Rent>
del(int) void
change(int, int) void
rented(int, String) void
findByMark(String) List<Rent>


Annotation
equals(Object) boolean
hashCode() int
toString() String
annotationType() Class<? extends Annotation>
Retention
value() RetentionPolicy
Target
value() ElementType[]
Override


TenantService
save(Tenant) void
findAll(int, int) List<Tenant>
findAll2() List<Tenant>
findByLocation(String) List<Tenant>
findByUser(String) List<Tenant>
change(int, int) void
del(int) void
Service
value() String
TenantServiceImpl
tenantMapper TenantMapper
save(Tenant) void
findAll(int, int) List<Tenant>
findAll2() List<Tenant>
findByLocation(String) List<Tenant>
findByUser(String) List<Tenant>
change(int, int) void
del(int) void


UserService
save(User) void
ifExist(String, String) User
ifUser(String) User
changeAuth(int, String) void
change(User) void
Service
value() String
UserServiceImpl
userMapper UserMapper
save(User) void
ifExist(String, String) User
ifUser(String) User
changeAuth(int, String) void
change(User) void


RequestMapping
name() String
value() String[]
path() String[]
method() RequestMethod[]
params() String[]
headers() String[]
consumes() String[]
produces() String[]
Controller
value() String
RentController
rentService RentService
save(Rent, HttpSession) String
findByLocation(String, String, String) String
findByUser(String, String, HttpSession) String
findByMark(String, String, HttpSession) String
changePrice(String, String) String
rented(String, HttpSession) String
del(int) String
RequestMapping
name() String
value() String[]
path() String[]
method() RequestMethod[]
params() String[]
headers() String[]
consumes() String[]
produces() String[]
Controller
value() String
ReviewController
reviewService ReviewService
userService UserService
save(Review, HttpSession) String
findAll(String, String) String
findAllMark(String, String) String
delN(String) void
changeMark(String) void
RequestMapping
name() String
value() String[]
path() String[]
method() RequestMethod[]
params() String[]
headers() String[]
consumes() String[]
produces() String[]
Controller
value() String
TenantController
tenantService TenantService
save(Tenant, HttpSession) String
findByLocation(String, String, String) String
findByUser(String, String, HttpSession) String
changePrice(String, String) String
del(int) String
Controller
value() String
UserController
userService UserService
save(User) String
login2() String
login(String, String, HttpSession) String
check(String) int
logout(HttpSession) String
userCenter(HttpSession) String
change(User) int

3.3.2 构件目录

构件及其特性表

构件	*描述*
前端	包括用户租房、用户出租、浏览租房出租信息、管理员审核页面
后端	包括注册登录、筛选房屋、出租、租房、浏览等服务

各类关系及其特性：

- 依赖/使用关系：
  - 前端依赖后端提供的接口
  - 后端依赖前端传值

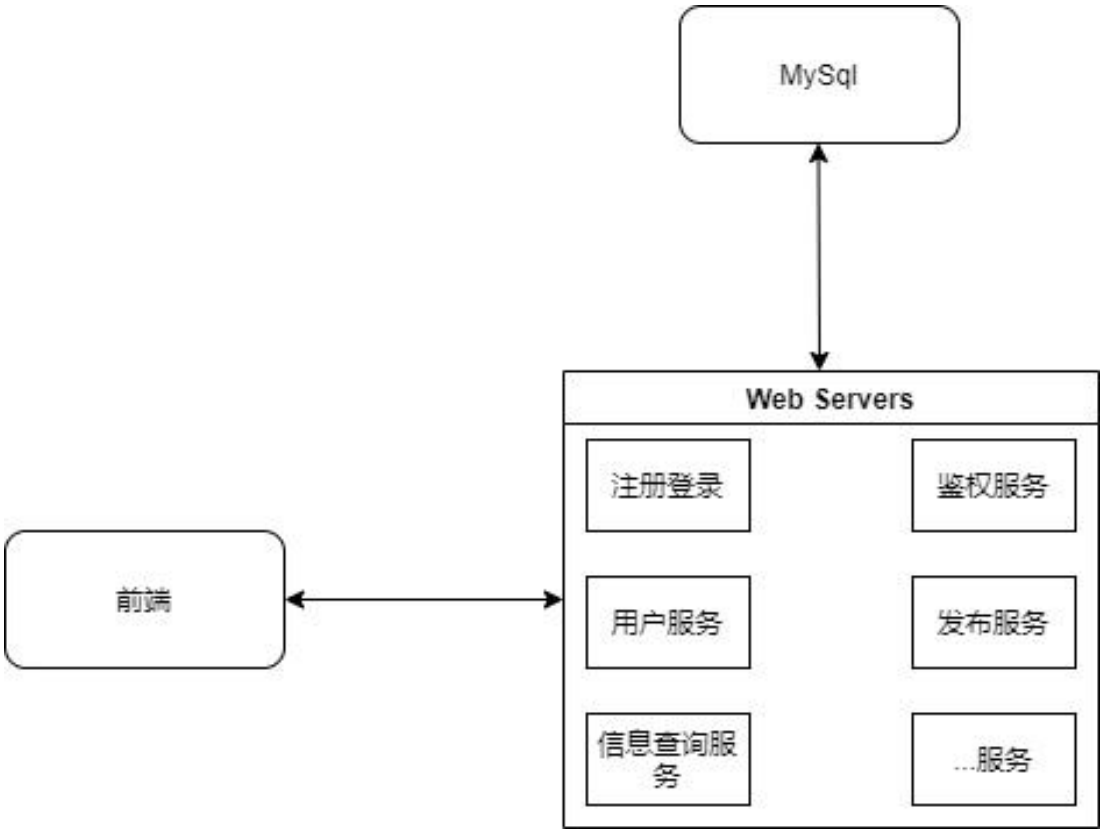
元素接口：

- 前端与后端
  - 查询所有出租房屋信息
  - 查询所有用户求租信息
  - 查询用户个人发布的出租房屋信息
  - 查询用户个人发布的求租信息
  - 查询个人租房信息
  - 用户租房
  - 用户提交请求成为中介
  - 用户登录/登出
  - 管理员审核用户成为中介的请求

3.3.3 可变性

在设计代码架构时，我们针对可变性进行了专门的设计，力求将模块间耦合性降到最低。代码框架中 每一个接口都独立开来完成，因此当遇到功能扩充或接口变迁的时候，我们能够非常方便、快速地进 行架构修改以及功能更新，可变性极高。同时可以方便的进行动态扩容，实现服务在线改变运行视图。

3.4 部署视图



3.4.1 主表示

3.4.2 构件目录

构件及其特性表：

构件	*描述*
前端	多个 Client
后端	总体运行在 Linux 系统，可以用 Docker 部署，拥有多个微服务

各类关系及其特性：

- 依赖/使用关系：
  - 前端依赖后端提供的接口
  - 后端依赖前端传值

元素接口：

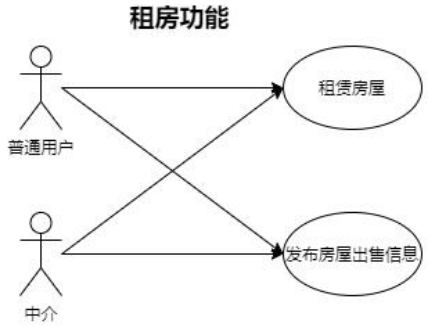
- 前端与后端
  - 查询所有出租房屋信息
  - 查询所有用户求租信息
  - 查询用户个人发布的出租房屋信息
  - 查询用户个人发布的求租信息
  - 查询个人租房信息
  - 用户租房
  - 用户提交请求成为中介
  - 用户登录/登出
  - 管理员审核用户成为中介的请求

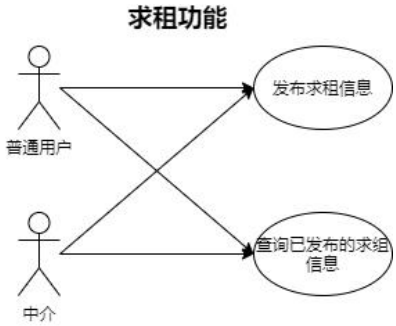
3.4.3 可变性

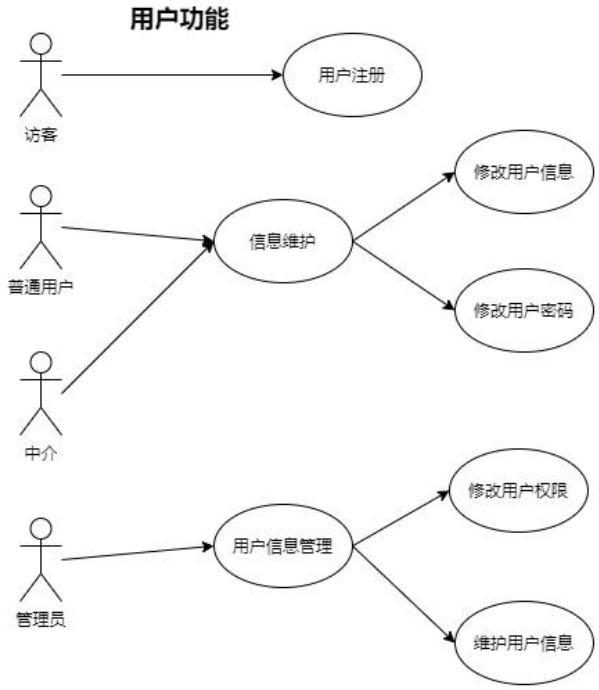
后端服务采用 docker 部署，后续可以采用微服务框架，支持各个微服务部署在不同的机器上，支持分布式部署。

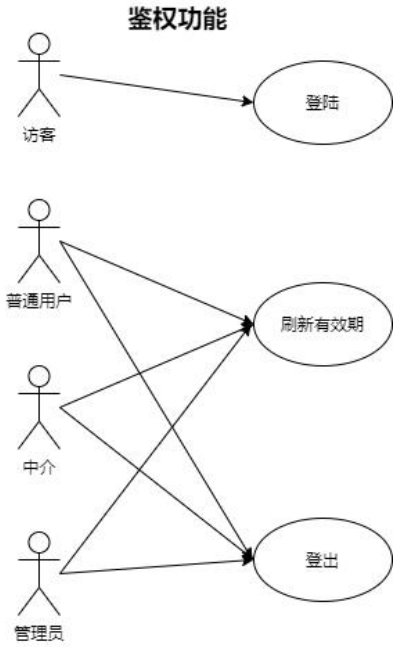
3.5 用例视图

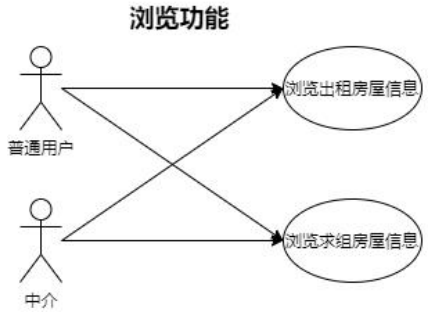
3.5.1 主表示

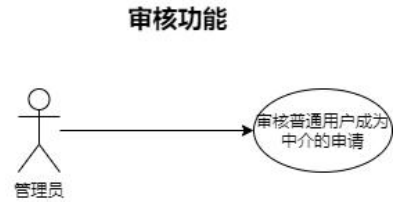












3.5.2 构建目录

构件及其特性表

构件	*描述*
租房功能	普通用户和中介可以对房屋进行租赁、可以发布房屋出租信息

求租功能	普通用户和中介可以发布求租信息、查询个人发布的求租信息
用户功能	提供登录、注册、登出功能，管理员可以维护用户的列表、权限等
浏览功能	普通用户和中介可以浏览出租房屋信息、求租信息
审核功能	管理员可以审核普通用户成为中介
鉴权功能	需要确认用户身份，确认用户拥有操作的权限

各类关系及其特性：

- 依赖/使用关系：
  - 前端依赖后端提供的接口
  - 后端依赖前端传值

元素接口：

- 前端与后端
  - 查询所有出租房屋信息
  - 查询所有用户求租信息
  - 查询用户个人发布的出租房屋信息
  - 查询用户个人发布的求租信息
  - 查询个人租房信息
  - 用户租房
  - 用户提交请求成为中介
  - 用户登录/登出
  - 管理员审核用户成为中介的请求

3.5.3 可变性

后端提供了丰富的接口API，方便前端展示以及数据获取，所有服务均由后端提供，管理员可以直接通过 API 接口查看以及修改整个系统的状态，交互性强，可指定化程度高。

4 视图之间关系

4.1 视图之间关系说明

- 用例视图
  - 4+1 视图的核心
  - 确定了系统边界、系统用戶、功能和场景，其它 4 个视图需要围绕这些信息进行设计
- 逻辑视图
  - 所有视图中最不可或缺的视图
  - 对系统职责进行逐级划分，并且对接口进行描述
  - 逻辑架构元素决定了开发组织，逻辑元素的边界和接口也是后续多个开发组织之间进行接口控制的关系依据
- 开发视图

    - 对逻辑架构元素，描述其代码开发流程
    - 开发视图描述了代码开发路线所包含的逻辑架构元素
- 部署视图
    - 描述代码与软件环境的部署关系
    - 描述软件环境与物理环境的部署关系
- 运行视图

    - 唯一一个描述系统动态信息的视图
    - 描述逻辑架构元素之间的交互关系、组件间的交互关系

4.2 视图-视图关系

4.2.1 运行视图和开发视图的关系：

开发视图一般偏重于程序包在编译时期的静态依赖关系，而这些程序运行起来后会表现为对象、线程、进程，运行视图比较关注这些运行时单元的交互问题。

4.2.2 部署视图和运行视图的关系

运行视图特别关注目标程序的动态执行情况，而部署视图重视目标程序的静态位置问题；部署视图还要考虑软件系统和包括硬件在内的整个 IT 系统之间是如何相互影响的。

4.2.3 用例视图和部署视图的关系

部署视图围绕用例视图部署

五种架构视图的关注点各有侧重。逻辑视图侧重功能需求；开发视图侧重开发期间质量属性；运行视图侧重运行期质量属性；部署视图侧重安装和部署需求；用例视图侧重数据需求。

5 需求与架构之间的映射

需求	*架构*
后端服务可以部署在多机器上	采用面向服务的架构（或微服务架构），各个模块分别作为微服务
后端快速开发，方便维护、添加内容	采用 MVC 模式，将业务模型、控制器实现代码分离，使一个程序可以使用不同的表现形式，方便后期维护添加功能
前端快速开发，方便维护	采用 MVC 模式，快速开发，方便后期维护添加功能

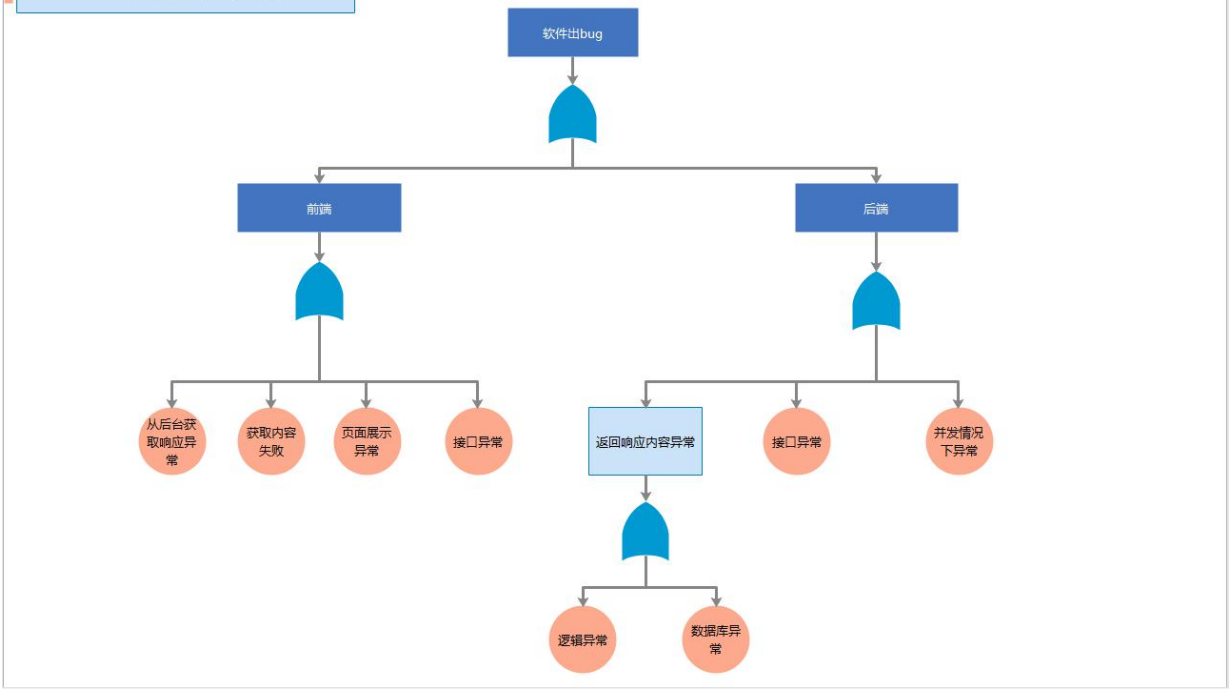
6 附录

6.1 术语表

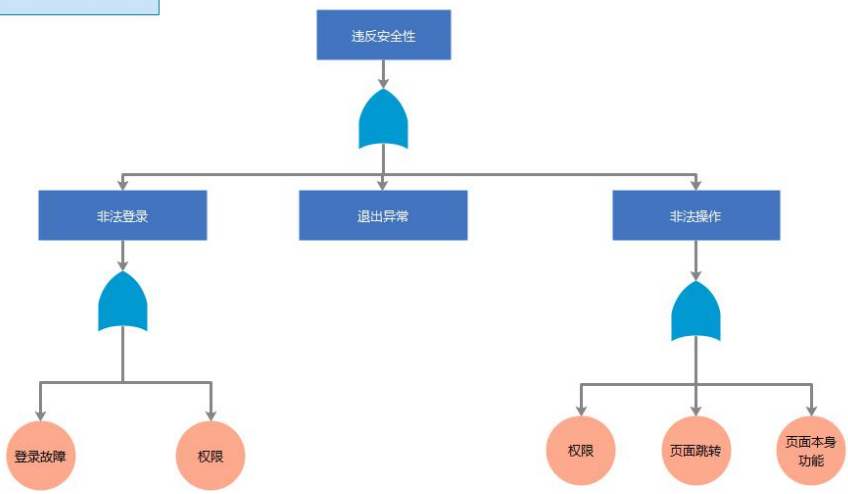
术语	*解释*
后端	运行在服务器上的程序，为用戶的请求提供服务功能
前端	运行在浏览器中的程序
Spring	Spring 是一个轻量级的 Java 开源框架。它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构。
SpringMVC	springmvc 是一种基于 MVC 设计模型的请求驱动类型的轻量级 web 框架，MVC 是项目开发中的一种开发模式，而最为 Spring 框架中重要的功能模块，SpringMVC 正是实现了这种开发结构，能更加简答、快速的开发 web 项目。
微服务	微服务架构是一项在云中部署应用和服务的新技术。

6.2 故障树-割集树模型

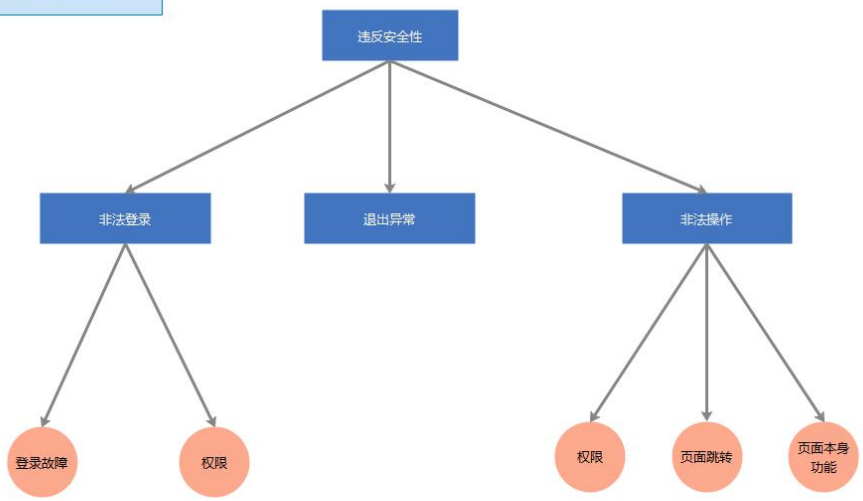
RHO系统故障树模型



违反安全性故障树模型



违反安全性割集树



RHO系统割集树