城市养老殡葬管理系统
系统设计与开发报告

摘要

本报告介绍了城市养老殡葬管理系统的设计与开发，重点关注数据库的设计与性能优化。系统主要提供养老、殡葬服务信息管理、健康数据监控、老年人健康管理及机构预约等功能。报告首先分析了项目的可行性，然后详细描述了数据库的设计过程，包括概念模型、逻辑设计和物理设计。

数据库使用 MySQL 存储数据，通过 EXPLAIN 和 EXPLAIN ANALYZE 对 SQL 查询进行了优化，以提升数据库的响应速度和处理能力。在系统实现方面，前端使用 HTML、CSS 和 JavaScript，后端采用 Flask 框架处理数据交互。报告还包括了负载测试、性能监控以及跨浏览器和移动端适配性测试，确保系统的稳定性和高可用性。

最后，报告讨论了系统的运行维护，包括数据库安全、数据备份与恢复，以及系统更新与优化。

关键词：

1. 数据库设计
2. MySQL 优化
3. 系统维护

目录

一、问题定义

1.1. 项目背景

2020 年我国第七次人口普查数据显示，我国大陆少儿人口（0-14 岁）、劳动年龄人口（15-59 岁）和老年人口（60+岁）占总人口的比重分别为 17.95%、63.35%和 18.7%。按 60 岁及以上标准计算的老年人口数量第一次超少儿人口，标志着我国人口年龄结构向老龄化发展[1]。这意味着老年人养老的需求极大，且后续也将持续增长。

同时，根据国家统计局数据，2020 年中国出生率为 6.77‰；死亡率为 7.37‰；总人口为 141175 万人，较上年末减少 85 万人，自然增长率为-0.60‰。中国正式步入人口负增长时代[2]。除了出生率走低，年轻人的生育意愿下降，我国的离婚率也持续走高，结婚率持续下降。民政部 2020 年公布的数据显示，2019 年结婚率为 6.6‰，比上年降低 0.7 个千分点，离婚率 3.36‰却比上年增长 0.2 个千分点[3]。年轻人的婚育意愿降低，大龄夫妻离婚的案例也在升高。

在更先步入人口老龄化的日本，老年人“孤独死”的案例层出不穷。“孤独死”也称“孤立死”，是指“死亡时没有任何人看护或知晓，并且距离被发现有一定时间间隔”的社会现象[4]。我们中国作为有着“尊老爱幼”的传统美德的文明古国，“孤独死”不应该是我们任何人希望看到的局面，也不会是我们期待的未来。

种种迹象都表明，在未来的养老服务中，会出现越来越多的既无子女，又无伴侣的老人。随着年纪增大，老年人的各类基础病发生的风险也会大大增加，很有可能不能自理养老，甚至独自死去。为了完善养老体系，社会养老将在未来越来越普遍。为了方便老人自主安排从养老到体面的死去的一系列过程，本文基于上海市公共数据开放平台[5]给出的数据，提出了一个城市养老殡葬管理系统。该系统不仅能方便老人自主选择需要的养老、殡葬机构，还能帮助机构进行自主优化管理，以及政府职能部门的监管规范。

1.2. 项目目的与意义

本项目提出的城市养老殡葬管理系统旨在通过信息化手段解决上述问题，构建一个集成的智能管理平台，帮助老年人选择合适的养老及殡葬机构，并实时监控其健康状况。系统的主要功能包括：

- **机构信息服务：**提供详细的养老与殡葬机构信息，帮助老年人及其家属选择合适的服务。
- **健康监测与紧急通知：**实时监测老年人的健康状况（如心率等生理指标），在异常情况下及时通知家属或相关人员。
- **评价与优化管理：**用户可以对机构进行评价与评分，帮助机构优化服务质量。
- **政府监管：**政府职能部门可以通过系统实时监控养老与殡葬机构的运营，确保服务质量与合法性。
- **预约功能：**用户可以方便地预约实地访问养老机构，进一步了解服务情况。

该系统的实施将有效提高养老和殡葬服务的透明度和监管效率，帮助老年人群体实现更加独立和体面的晚年生活，减轻家庭负担，同时为政府部门提供更加高效的监管手段。

二、可行性分析

2.1. 技术可行性

本项目采用现代主流技术架构，前端使用 HTML、CSS、JavaScript 进行页面设计与交互，后端使用 Flask 框架与 Python 语言进行开发，数据库采用 MySQL 进行数据存储。所有这些技术都是当前开发中广泛使用且成熟的技术，具有很高的稳定性和可扩展性。

- **前端技术：**HTML、CSS 和 JavaScript 是构建现代 Web 应用的基本技术，能够实现动态交互和响应式设计，适配不同设备，提供良好的用户体验。
- **后端技术：**Flask 是一个轻量级 Web 框架，适合中小型项目开发，具有快速开发、易于扩展的优势。使用 Python 语言，具有丰富的开发生态系统，能够高效处理后台逻辑和数据交互。
- **数据库技术：**MySQL 是一个成熟的关系型数据库管理系统，具备高性能和高可扩展性，广泛应用于各类系统中，能够有效处理大量数据的存储与查询。
- **数据来源：**上海市公共数据开放平台提供的养老院和殡葬服务机构的详细信息，数据可靠且符合项目需求，为系统的设计与实施提供了必要的数据支持。

因此，从技术角度来看，项目的实施是可行的，所选技术栈能够满足项目需求，保证系统的稳定性和高效性。

2.2. 经济可行性

本项目的经济可行性主要体现在以下几个方面：

- **成本投入：**采用开源技术（如 Flask、MySQL、HTML、CSS 和 JavaScript）大大降低了开发成本。开发过程中，除了人员工资和基础设施成本外，不需要购买昂贵的商业软件或工具。
- **长远效益：**通过提升养老和殡葬服务的管理效率、优化服务质量，将促进行业的发展并吸引更多的用户使用该系统。政府监管部门能够更高效地进行市场监管，避免资源浪费，并确保行业合规性，长期来看将带来较好的社会效益和经济效益。
- **市场需求：**随着老龄化问题的日益加剧，养老和殡葬行业的需求持续增长，系统能够为老年人提供更好的服务，也帮助机构提升管理效率。这样具有社会意义的项目将具有较强的市场潜力。

综合来看，本项目的开发投入相对较低，但长期收益显著，具备较高的经济可行性。

2.3. 操作可行性

项目的操作可行性体现在以下几点：

- **用户友好性：**前端界面将采用简洁、直观的设计，使得各类用户（包括老年人、家属、机构管理人员和政府监管人员）都能方便地使用系统。
- **系统维护性：**Flask 框架与 MySQL 数据库易于维护和扩展，开发人员能够快速定位问题并进行修复。系统的模块化设计也使得后期的功能扩展更加灵活，适应不断变化的需求。

由于系统采用现有的成熟技术，并通过简化的用户界面设计，操作可行性较高，适合不同背景和技术水平的用户使用。

2.4. 法律可行性

本项目在法律可行性方面考虑了以下几个方面：

- **数据隐私保护：**项目涉及到老年人的健康数据，必须遵守相关的数据保护法规。根据《中华人民共和国个人信息保护法》及相关法律规定，

系统将确保用户的个人数据仅用于服务目的，且数据存储和传输过程中将采取加密措施，防止数据泄露。

- **医疗健康数据合规：**由于系统涉及健康数据的实时采集与反馈，项目需确保符合国家关于医疗健康数据的法律法规，例如《电子健康档案管理办法》、国家卫生健康委员会的相关政策等，保证数据采集和处理的合法性。
- **政府监管合规：**系统为政府监管部门提供运营监控功能，确保系统能够帮助政府部门依法合规地对养老和殡葬行业进行监管，并提供所需的报告与数据支持。

因此，在法律层面，项目将遵循现有法律法规，确保数据安全、隐私保护和合法合规性，具有较高的法律可行性。

三、需求分析

3.1. 用户分析

3.1.1. 一般用户

普通访问者，可能是潜在的养老或殡葬服务需求者，或者关注养老和殡葬服务的公众。

他们的需求为：访问公开的养老和殡葬机构详细信息；举报机构或看护人员的不当行为，保障老年人的权益；查看其他用户对机构的评价和评分。因此给予的权限为：查看所有公开的机构信息；提交举报信息。

3.1.2. 老年用户

使用养老或殡葬服务的老年人，可能居住在养老机构或使用居家养老服务。

他们的需求为：通过可穿戴设备或智能设备实时上传自己的健康数据。查看自己的健康数据。选择和预约合适的养老或殡葬服务。对使用的服务进行反馈和评价。因此给予的权限为：一般用户的权限；上传和查看自己的健康数据；紧急呼救；查看和预约养老或殡葬服务。

3.1.3. 家属、看护用户

老年用户的家属或看护人员，负责老年人的日常照料和健康管理，需要与至少一位老年用户绑定，获取其健康数据。

他们的需求为：实时查看绑定老年用户的健康数据和健康状况；接收健康异常通知和报警信息，及时采取措施；为老年用户选择和预约合适的养老或殡葬服务。因此给予的权限为：一般用户的权限；查看绑定老年用户的健康数据和健康状况；接收健康异常通知和报警信息；查看和预约养老或殡葬服务。

3.1.4. 养老、殡葬机构管理人员

养老和殡葬机构的管理人员，负责机构的日常运营和管理。

他们的需求为：维护和更新机构的详细信息；查看和处理用户反馈和评价，改进服务质量。因此给予的权限为：查看所有公开的机构信息；管理和更新认证的机构信息；查看和处理用户反馈和评价。

3.1.5. 民政部监管职员

政府民政部门的监管人员，负责监督养老和殡葬机构的运营情况。

他们的需求为：实时监控养老和殡葬机构的运营数据，确保合规运营；处理公众和用户提交的举报信息，采取监管措施。因此给予的权限为：实时监控所有机构的运营数据；查看和处理举报信息。

3.2. 需求用例

根据系统的功能和不同用户的需求，可以将该系统划分为以下几个功能用例进行实现。

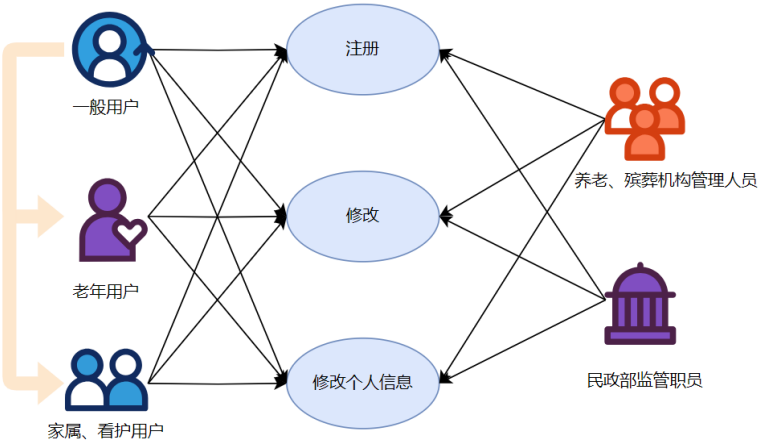
3.2.1. 注册登录和个人信息管理功能

本系统一共区分了五种不同的用户类型：普通用户；老年用户；家属、看护用户；养老、殡葬机构管理人员；民政部监管职员。用户在注册和登录之后拥有一种用户身份，不同的类型拥有不同的用户权限。在注册后，用户可以对自己的部分信息进行管理和修改。

- 普通用户仅需填写相关的基本身份信息即可注册。
- 老年用户需要进行实名认证。

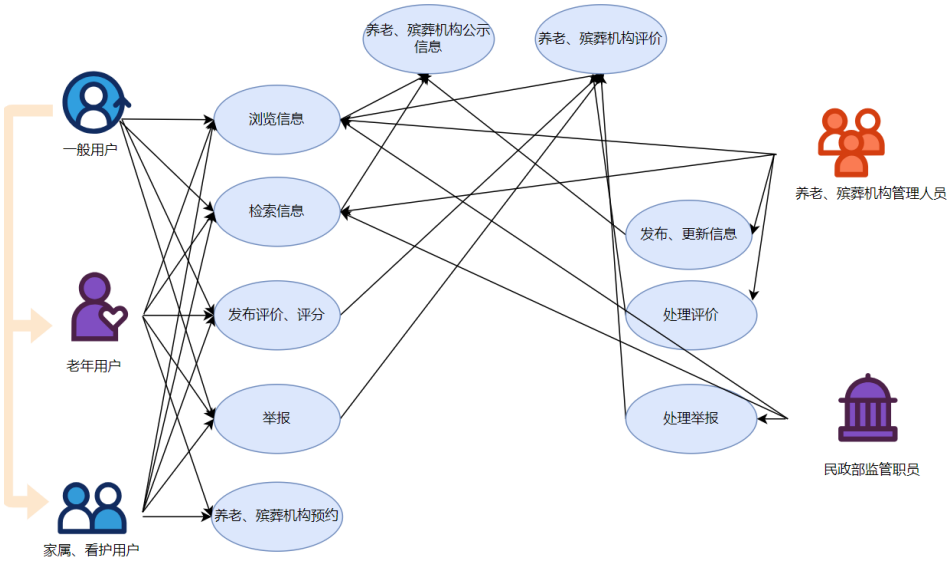
- 家属、看护用户需要实名认证，并和已存在的至少一位老年用户绑定。
- 养老、殡葬机构管理人员需要上传资质证明。
- 民政部监管职员由管理部门进行统一的注册。

系统将对用户的注册信息进行检查和核实，确认用户身份，以保证系统的正常运行和信息的准确真实。



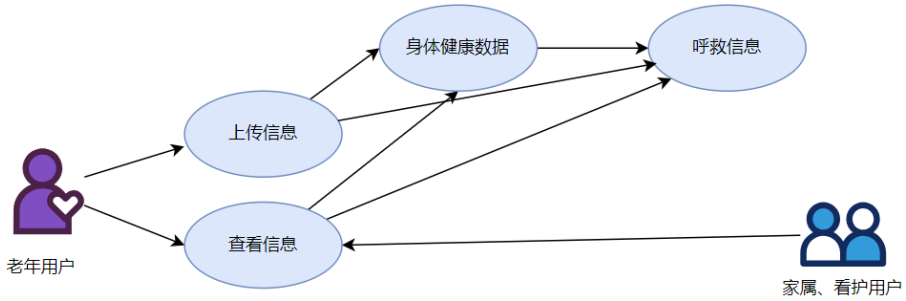
3.2.2. 养老、殡葬机构信息公示系统

- **一般用户**：养老、殡葬机构信息的浏览和查询，查看发布评价、评分，进行举报。
- **老年用户，家属、看护用户**：在一般用户的基础上，添加养老、殡葬机构预约
- **养老、殡葬机构管理人员**：养老、殡葬机构信息的浏览和查询，发布、更新养老、殡葬机构信息，处理用户评价。
- **民政部监管人员**：养老、殡葬机构信息的浏览和查询，处理举报。



3.2.3. 健康管理系统

- **老年用户：**上传和查看自己的健康数据，自主呼救或在检测到身体数据异常时自动呼救。
- **家属、看护用户：**查看对应老人的健康数据，接收健康异常通知和报警信息。



3.3. 数据字典

用户（User）：

属性名	类型	约束	描述
UserID	int	primary key	用户的唯一标识
Username	varchar(20)	not null	用户的登录名
Password	varchar(50)	not null	用户的密码
PhoneNumber	varchar(15)	not null	用户的联系电话

用户身份 (UserRoles)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识
RoleID	Int	Not null	身份标识
Role	varchar(10)	not null	用户的身份

一般用户 (GeneralUser)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识

老年用户 (SeniorUser)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识
RealName	varchar(20)	not null	用户的真实姓名
IDNumber	varchar(18)	not null	用户的身份证号码
Residence	varchar(50)	not null	用户的居住地址

老年用户健康数据 (SeniorHealthData)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on SeniorUser.UserID	用户的唯一标识
HeartRate	int	not null	用户的心率

家属、看护用户 (FamilyCaregiver)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识
RealName	varchar(20)	not null	用户的真实姓名
IDNumber	varchar(18)	not null	用户的身份证号码

家属老年用户关系 (FamilySeniorRelation)：

属性名	类型	约束	描述
FamilyCaregiverID	int	primary key foreign key on FamilyCaregiver.UserID	家属、看护用户的唯一标识
SeniorUserID	int	primary key foreign key on SeniorUser.UserID	老年用户的唯一标识

机构管理人员 (Staff)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识
Position	varchar(20)	primary key	用户的职务

管理机构 (StaffInstitution)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on Staff.UserID	用户的唯一标识
InstitutionID	int	primary key foreign key on ElderCareInstitution. InstitutionID	用户管理的机构

民政部监管职员 (Supervisor)：

属性名	类型	约束	描述
UserID	int	primary key foreign key on User.UserID	用户的唯一标识
Position	varchar(20)	not null	用户的职务
EmployeeNumber	varchar(20)	primary key	用户的工号

养老机构 (ElderCareInstitution)：

属性名	类型	约束	描述
InstitutionID	int	primary key	养老机构的唯一标识
StreetTown	varchar(50)	not null	养老机构所属的街道或乡镇
InstitutionName	varchar(50)	unique	养老机构的名称
BedCount	int	not null	养老机构的床位数
OperationYearMonth	varchar(10)		养老机构的执业时间
Address	varchar(100)	not null	养老机构的地址
PhoneNumber	varchar(15)		养老机构的联系电话
PostalCode	varchar(10)		养老机构的邮政编码
OperationMode	varchar(10)		养老机构的运营模式
LegalRepresentative	varchar(20)	not null	养老机构的法定代表人

殡葬机构 (FuneralInstitution)：

属性名	类型	约束	描述
InstitutionID	int	primary key	殡葬机构的唯一标识

InstitutionName	varchar(50)	unique	殡葬机构的名称
Address	varchar(100)	unique	殡葬机构的地址
PhoneNumber	varchar(15)	not null	殡葬机构的联系电话
PostalCode	varchar(10)		殡葬机构的邮政编码
ContactPerson	varchar(20)	not null	殡葬机构的联系人
InstitutionType	varchar(20)	not null	殡葬机构的机构性质
ServiceScope	varchar(50)	not null	殡葬机构的服务范围

养老机构评价 (ElderCareInstitutionEvaluation)：

属性名	类型	约束	描述
EvaluationID	int	primary key	评价的唯一标识
UserID	int	not null	用户的唯一标识
InstitutionID	int	not null foreign key on ElderCareInstitution. InstitutionID	机构的唯一标识
Evaluation	text	not null	评价内容
Rating	int		评分
ReportInfo	text		举报信息

殡葬机构评价(FuneralInstitutionEvaluation):

属性名	类型	约束	描述
EvaluationID	int	primary key	评价的唯一标识
UserID	int	not null	用户的唯一标识
InstitutionID	int	not null foreign key on FuneralInstitution. InstitutionID	机构的唯一标识
Evaluation	text	not null	评价内容
Rating	int		评分
ReportInfo	text		举报信息

预约 (Reservation)：

属性名	类型	约束	描述
ReservationID	int	primary key	预约的唯一标识
UserID	int	not null foreign key on User.UserID	用户的唯一标识
InstitutionID	int	not null foreign key on ElderCareInstitution. InstitutionID	机构的唯一标识
Role	varchar(10)	not null	用户的身份
ReservationTime	datetime	not null	预约时间
ReservationStatus	int	check 1 or 2 or 3 or 4	预约状态（未预约(1)，已预约(2)，已完成(3)，已逾期(4)）

呼救 (EmergencyCall)：

属性名	类型	约束	描述
EmergencyCallID	int	primary key	呼救的唯一标识
SeniorUserID	int	not null foreign key on SeniorUser.UserID	老年用户的唯一标识

CallType	varchar(4)	check “self” or “auto”	呼救类型（自主呼救(self)或自动呼救(auto)）

呼救用户 (EmergencyCallUser)：

属性名	类型	约束	描述
EmergencyCallID	int	primary key foreign key on EmergencyCall. EmergencyCallID	呼救的唯一标识
FamilyCaregiverID	int	primary key foreign key on FamilyCaregiver.UserID	家属用户的唯一标识
HeartRate	text	not null	身体健康数据

权限 (Permissions)：

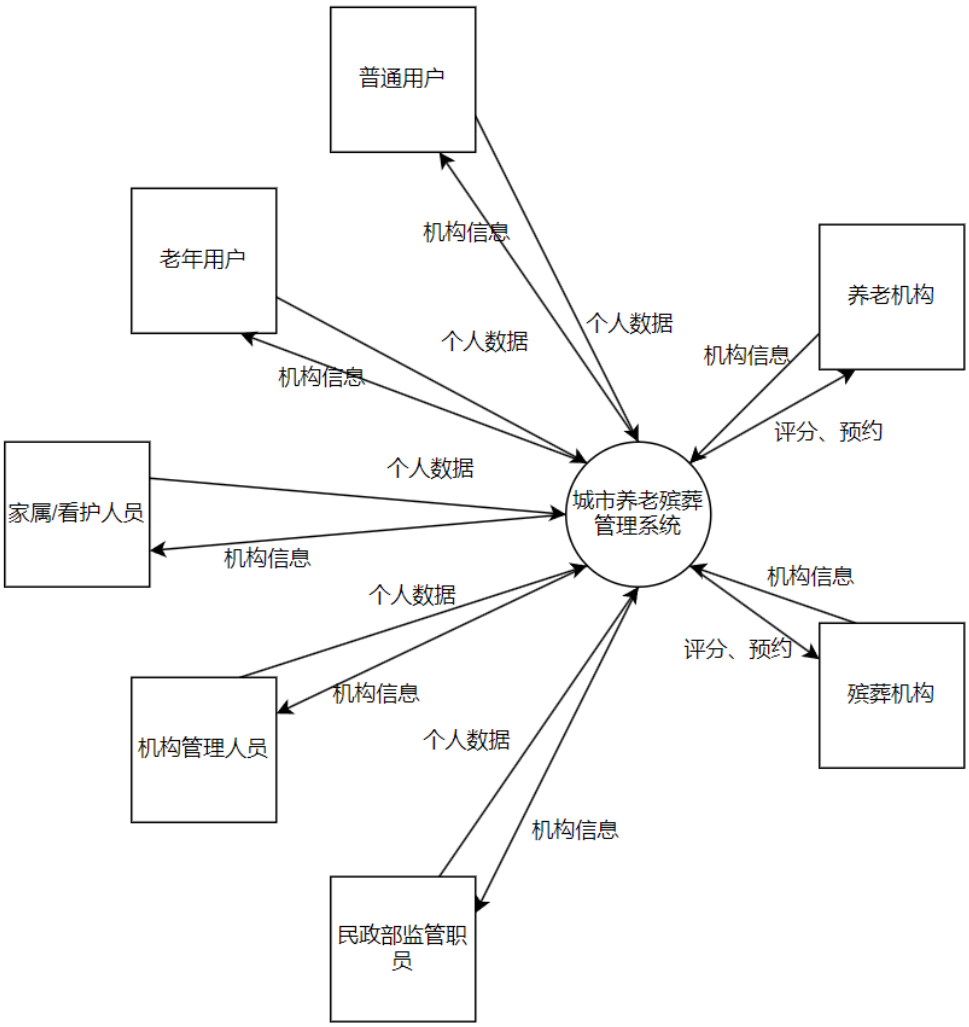
属性名	类型	约束	描述
PermissionID	int	primary key	权限的唯一标识
PermissionName	varchar(50)	not null	权限的名称
PermissionDescription	varchar(50)	not null	权限的描述

角色权限 (RolePermissions)：

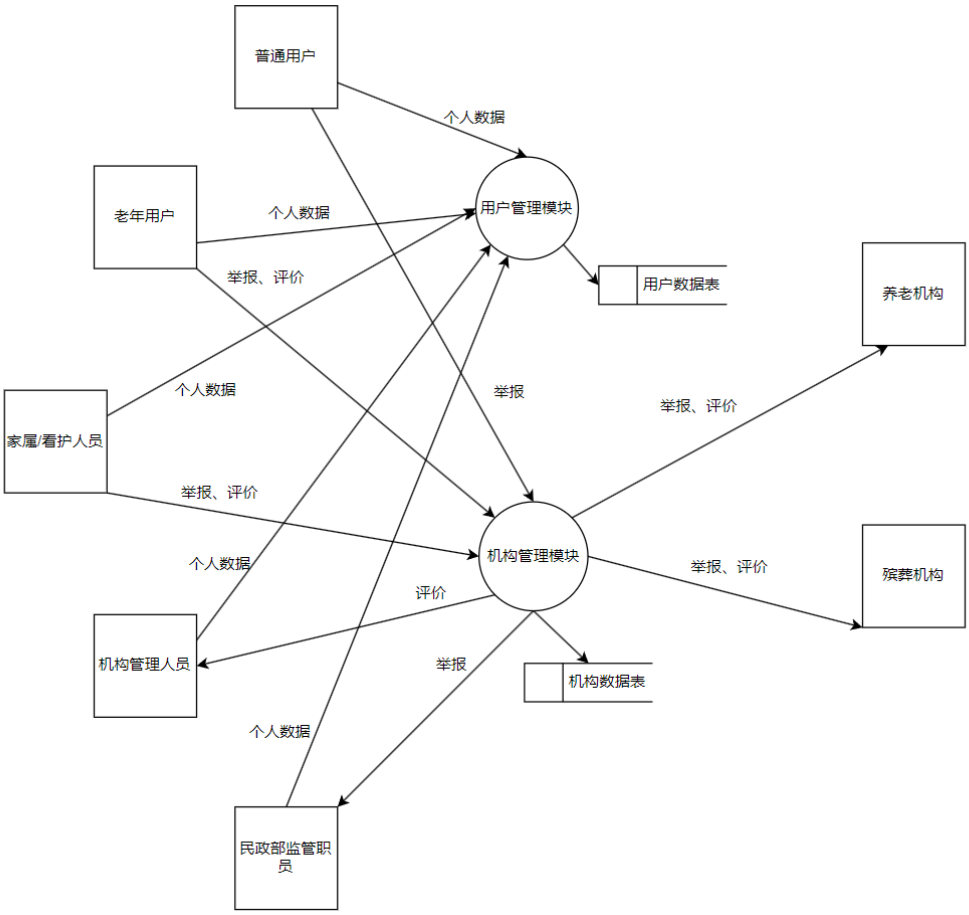
属性名	类型	约束	描述
RoleID	int	foreign key on UserRole.RoleID	身份的唯一标识
PermissionID	int	foreign key on Permission.PermissionID	权限的唯一标识

3.4. 数据流图

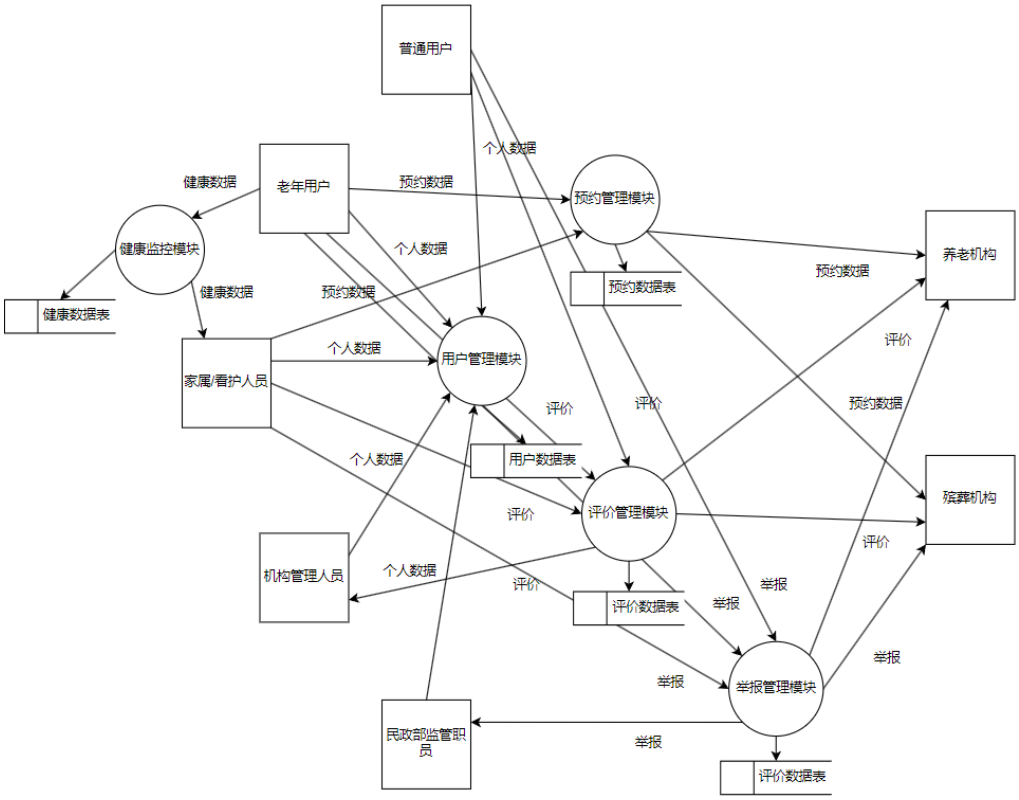
3.4.1. 0层数据流图


普通用户
老年用户
养老机构
家属/看护人员
城市养老殡葬管理系统
机构管理人员
殡葬机构
民政部监管职员
机构信息
个人数据
个人数据
机构信息
个人数据
机构信息
评分、预约
个人数据
机构信息
机构信息
个人数据
评分、预约
个人数据
机构信息

3.4.2. 1层数据流图


普通用户
个人数据
老年用户
个人数据
用户管理模块
举报、评价
用户数据表
养老机构
家属/看护人员
个人数据
举报
举报、评价
举报、评价
机构管理模块
个人数据
举报、评价
殡葬机构
评价
机构管理人员
举报
机构数据表
个人数据
民政部监管职员

3.4.3. 2 层数据流图



四、总体设计

4.1. 数据库设计

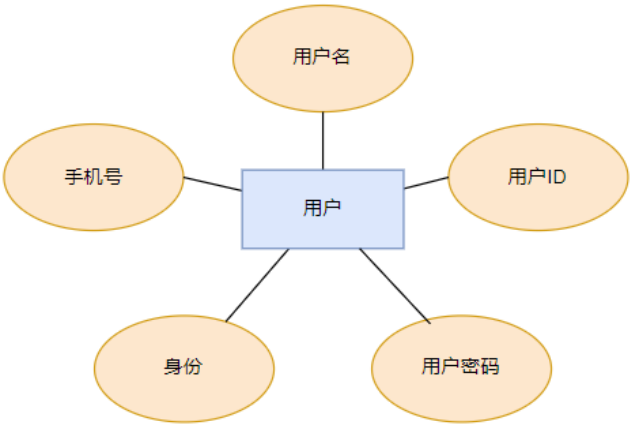
4.1.1. 概念模型

4.1.1.1. 实体属性设计

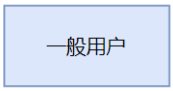
1. 用户相关实体

本系统面向的用户有普通用户；老年用户；家属、看护用户；养老、殡葬机构管理人员；民政部监管职员，这五种用户具有用户所共有的部分属性，因此提取出用户这个抽象类 ，将以上五个实体作为用户的派生，其属性分别为

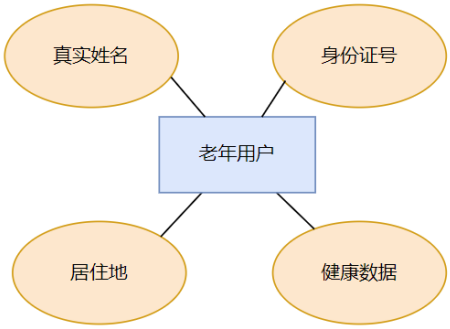
用户：用户名，用户 ID，用户密码，身份，手机号。



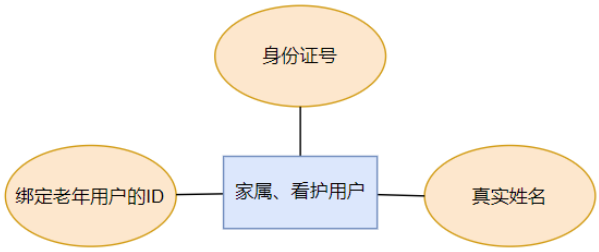
一般用户：继承用户所有属性。



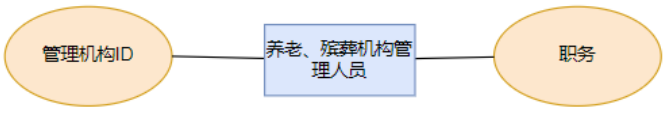
老年用户：继承用户所有属性，真实姓名，身份证号，居住地址，健康数据。



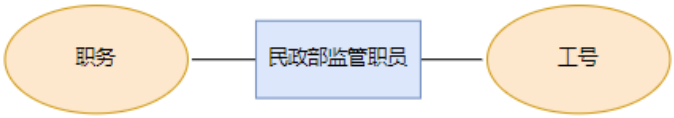
家属、看护用户：继承用户所有属性，真实姓名，身份证号，绑定老年用户的 ID。



养老、殡葬机构管理人员：继承用户所有属性，职务，管理机构 ID。

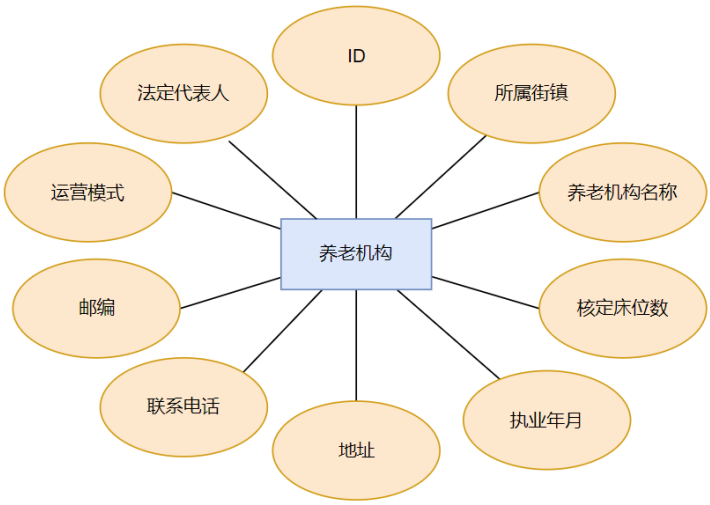


民政部监管职员：继承用户所有属性，工号，职务。

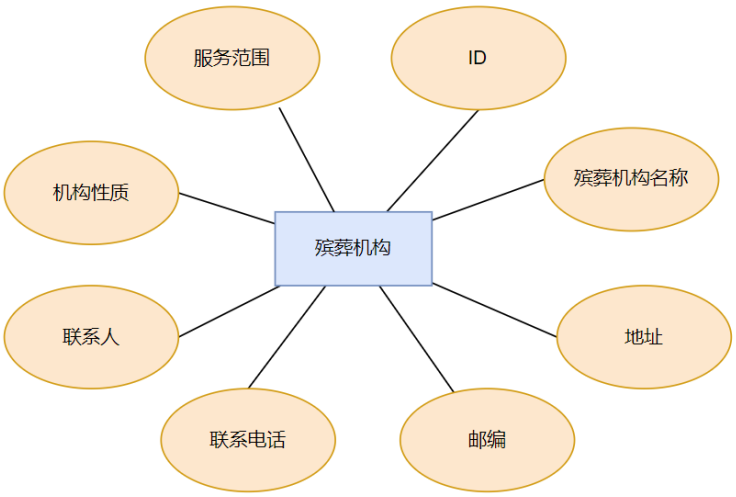


2. 机构相关实体

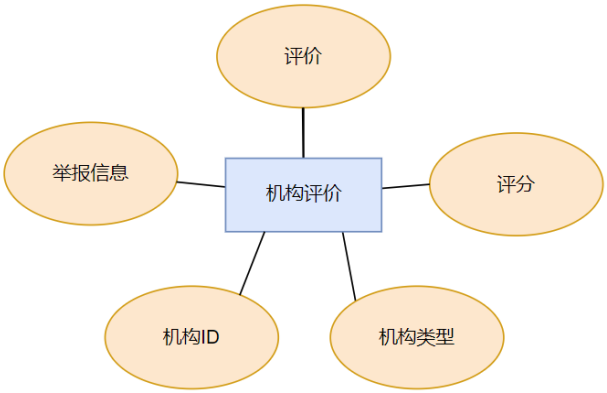
养老机构：根据上海市公共数据平台中关于上海市养老机构的数据[6]，设置以下属性：ID，所属街镇，养老机构名称，核定床位数，执业年月，地址，联系电话，邮编，运营模式，法定代表人。其中养老机构的运营模式有三种：公建公营、公建民营、民建民营[7]。



殡葬机构：根据上海市公共数据平台中关于殡葬服务代理机构的数据[8]，设置以下属性：ID，殡葬机构名称，地址，邮编，联系电话，联系人，机构性质，服务范围。



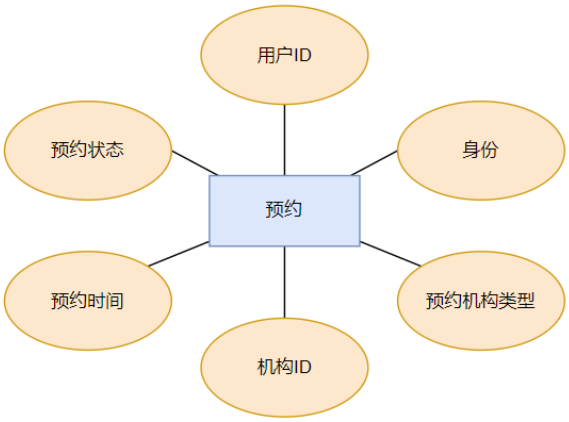
机构评价：为帮助用户更好选择机构，设计关于机构评价的实体，设置以下属性：机构类型、机构 ID、评价，评分，举报信息。



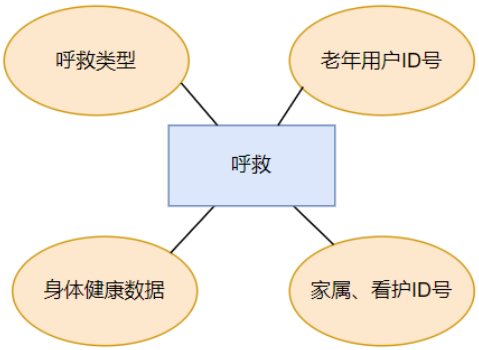
3. **其他实体**

除了用户和机构这两方面的主要的实体，要完成整个系统的功能，还需要其他一些实体，具体如下：

预约：用户 ID，身份，预约机构类型，机构序号，预约时间，预约状态（未预约，已预约，已完成，已逾期）。

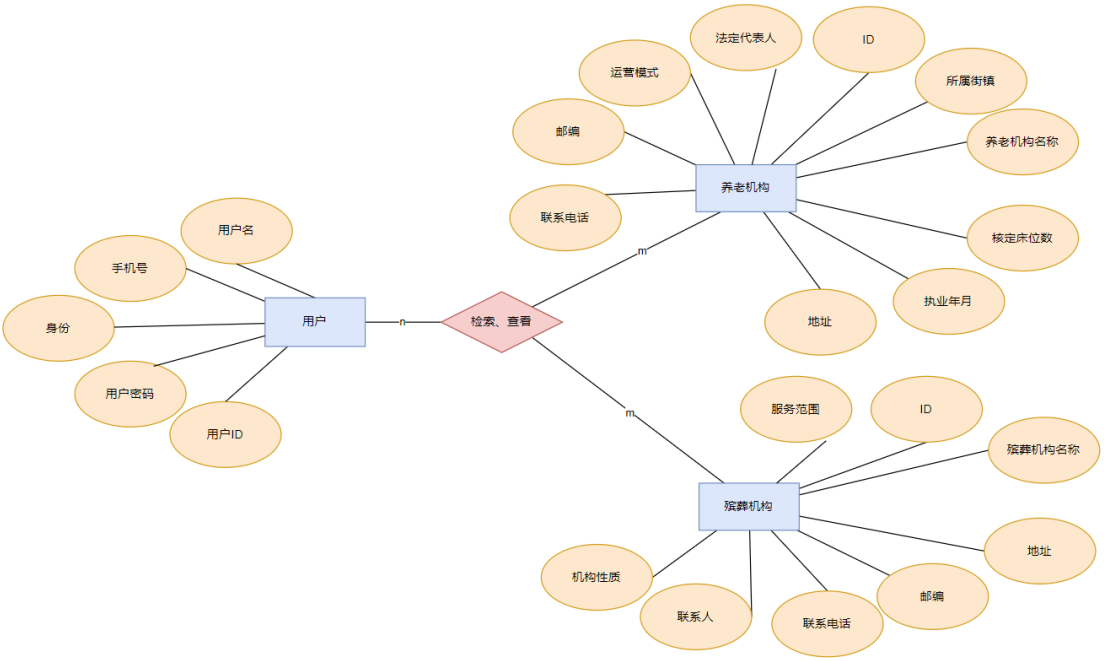


呼救：老年用户 ID 号，对应家属、看护 ID 号，身体健康数据，呼救类型（自主呼救，自动呼救）。

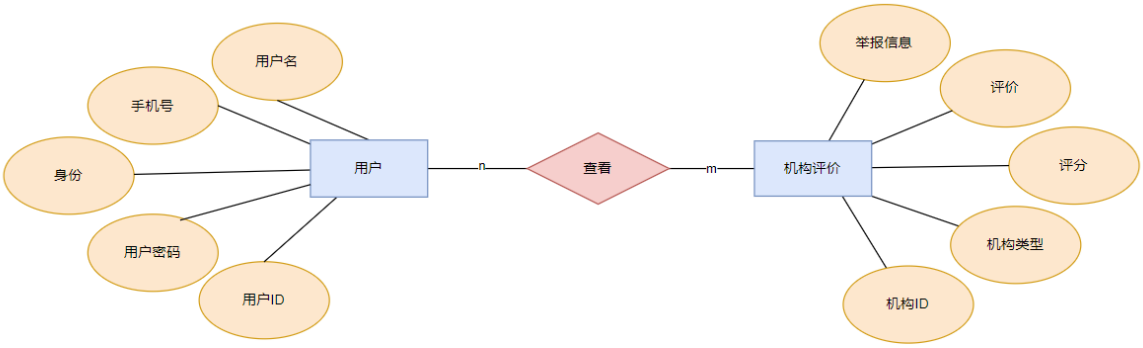


4.1.1.2. 实体关系设计

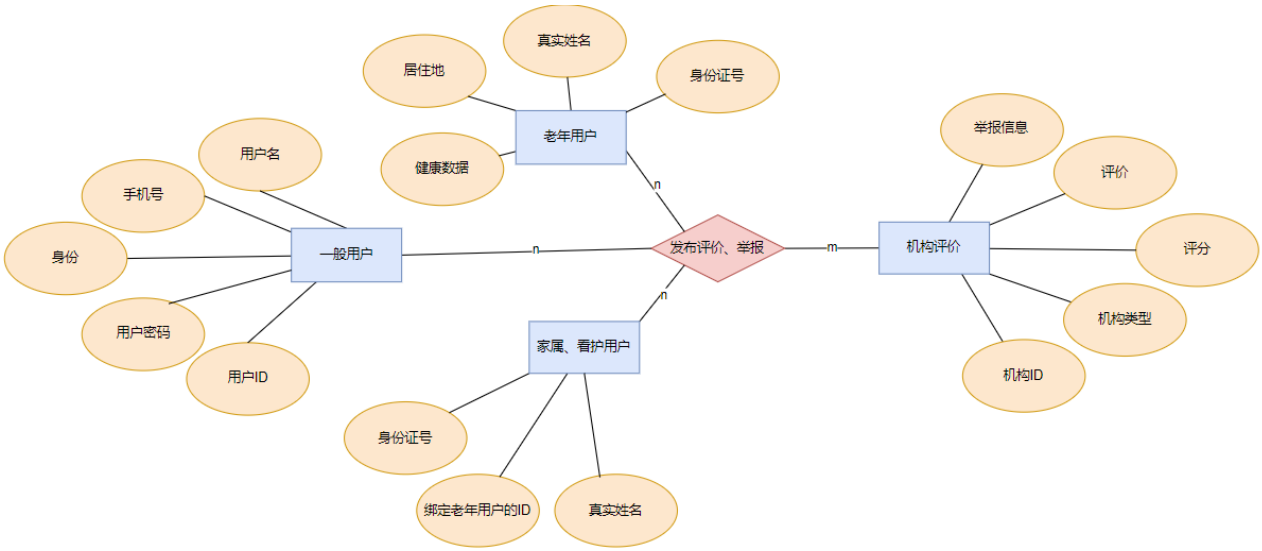
1. **用户可检索、查看机构**



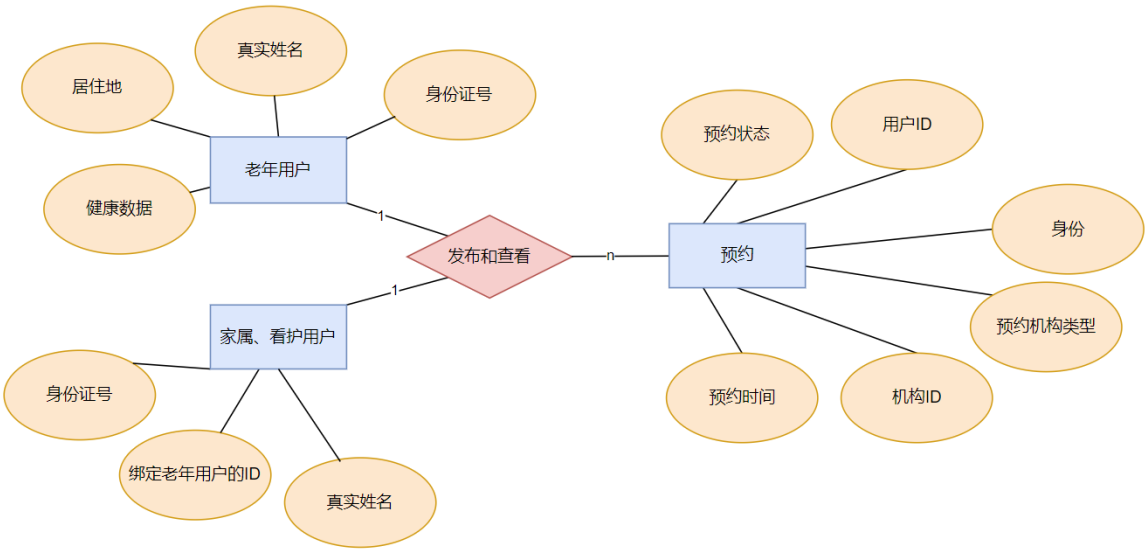
2. 用户可查看机构评价：



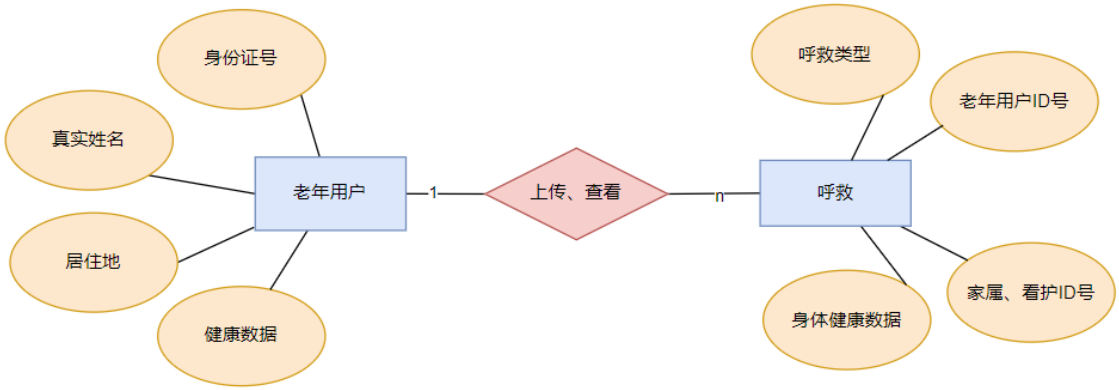
3. 一般用户；老年用户；家属、看护用户可发布评价、举报机构：



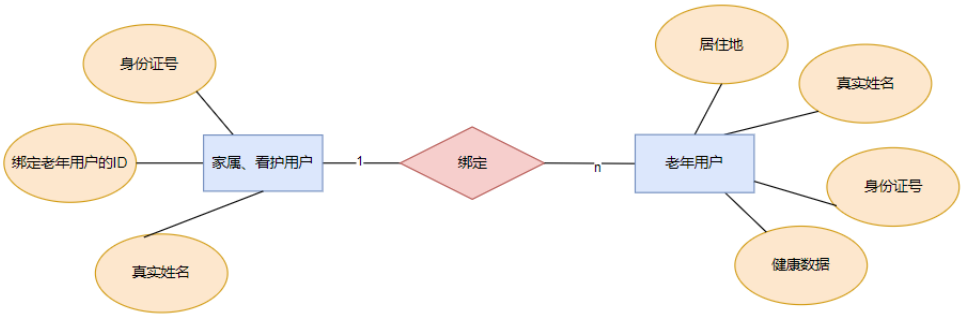
4. 老年用户；家属、看护用户可以预约机构服务：



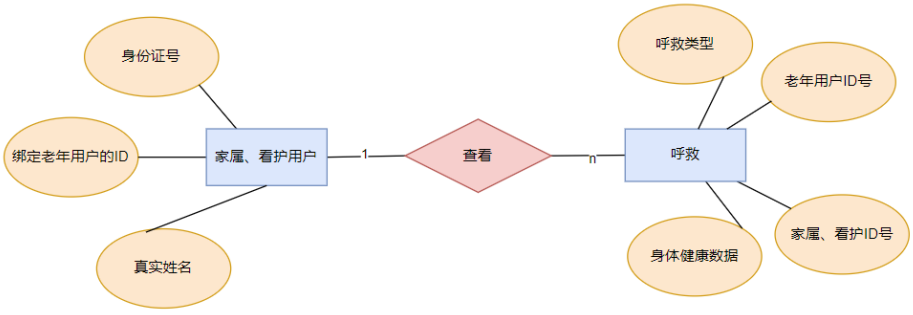
5. 老年用户可以进行呼救：



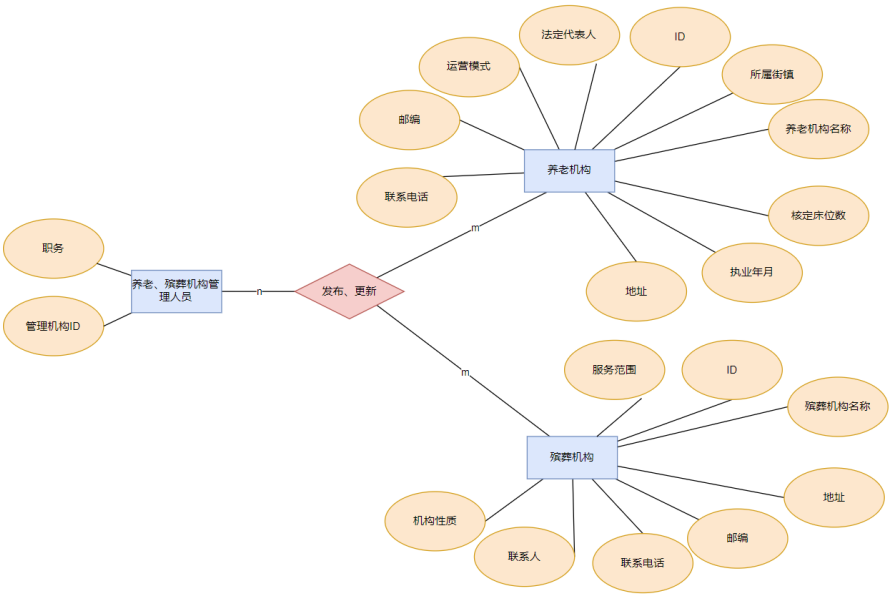
6. 家属、看护用户至少绑定一名老年用户：



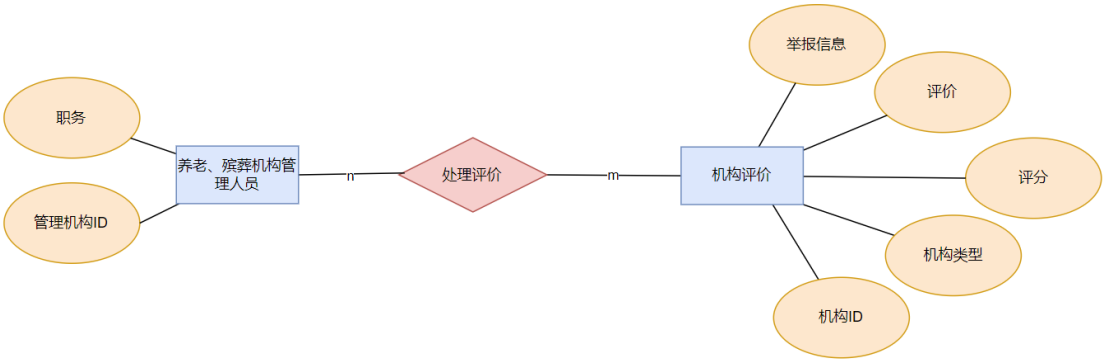
7. 家属、看护用户可以查看呼救：



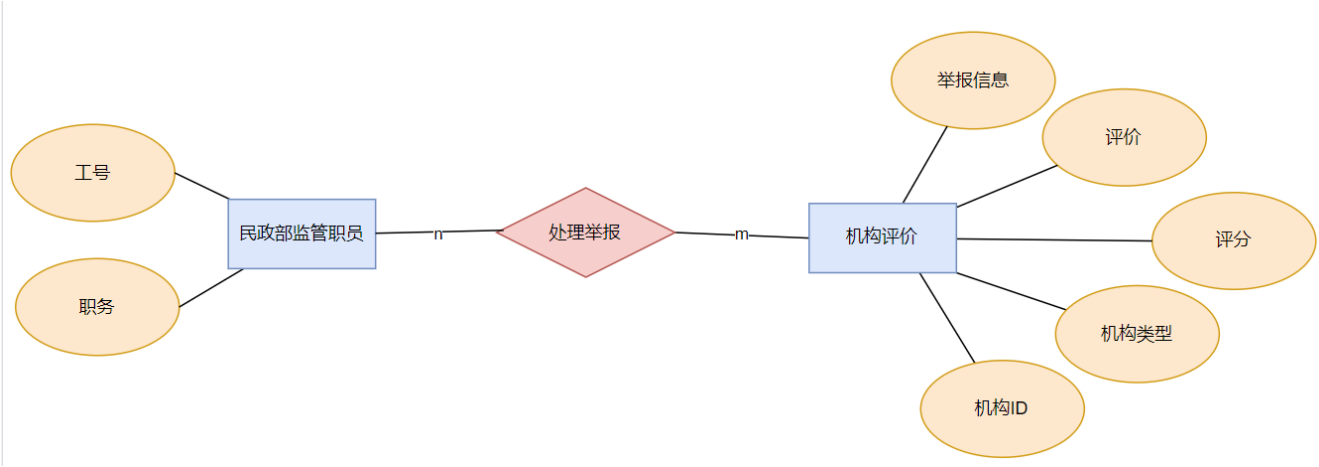
8. 养老、殡葬机构管理人员可以发布更新机构信息：



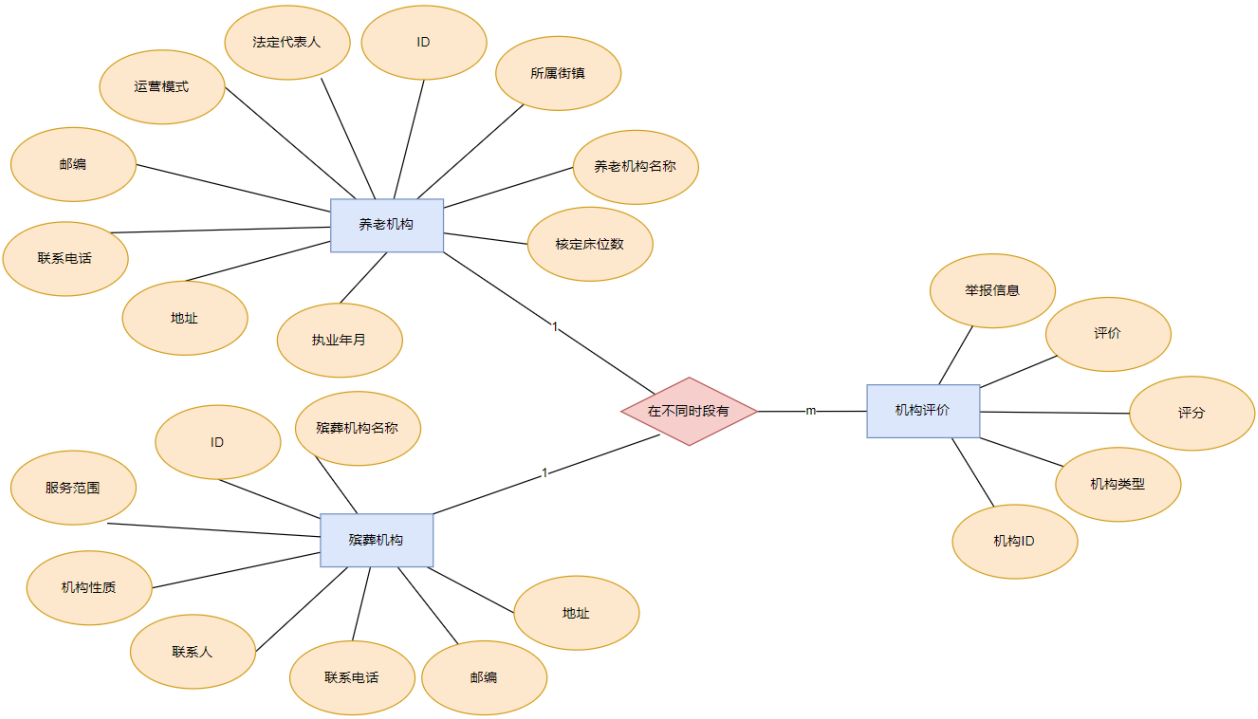
9. 养老、殡葬机构管理人员可以处理评价



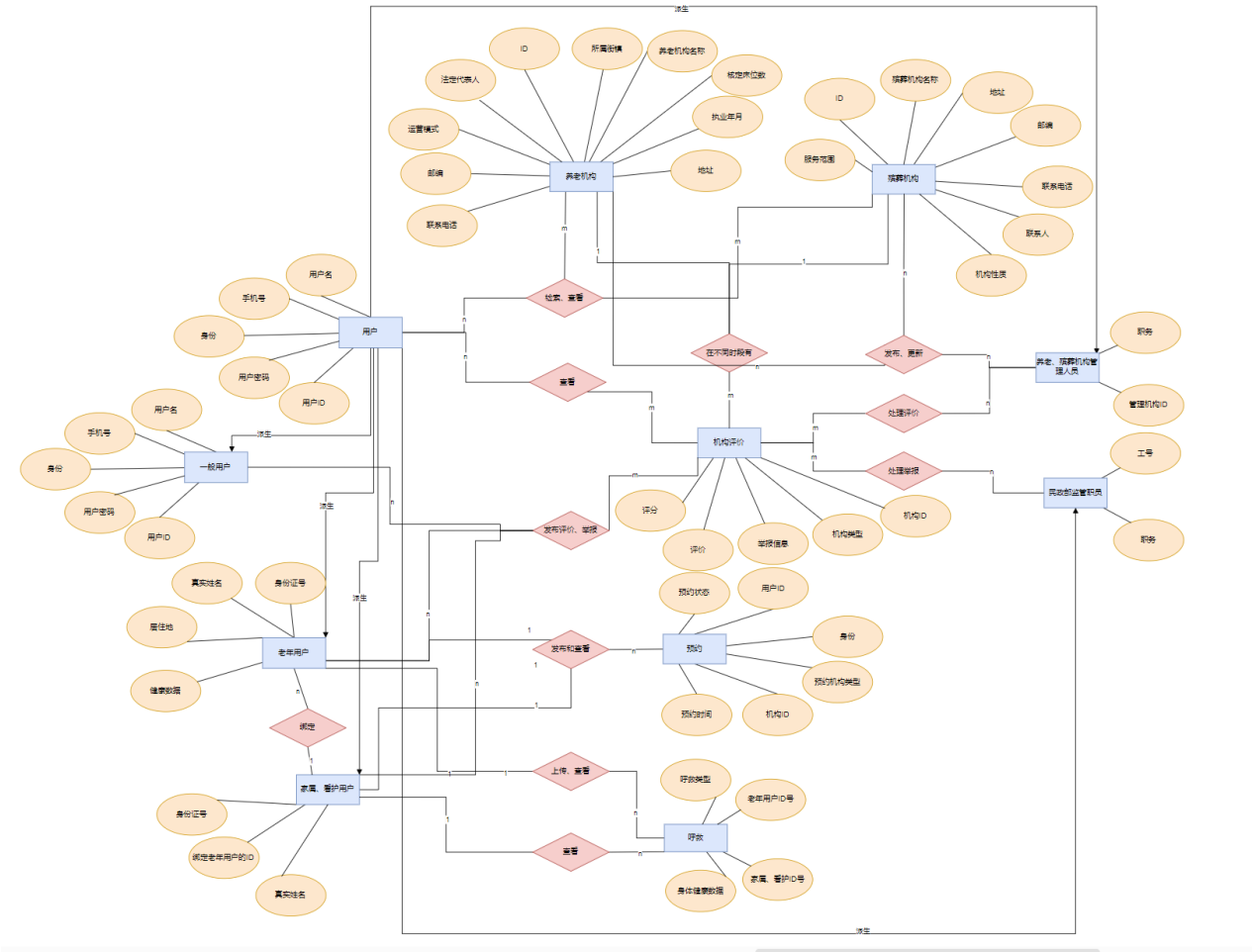
10. 民政局监管职员可以处理举报：



11. 机构在不同时间拥有不同的评价：


法定代表人
ID
运营模式
所属街镇
邮编
养老机构名称
养老机构
联系电话
核定床位数
地址
执业年月
1
举报信息
评价
在不同时段有
m
机构评价
评分
ID
殡葬机构名称
机构类型
服务范围
1
殡葬机构
机构ID
机构性质
地址
联系人
联系电话
邮编

4.1.1.3. 全局 E-R 图



4.1.2. 逻辑设计

4.1.2.1. 关系模式设计

1. 将实体转化为关系模式

用户 =（用户 ID、用户名、密码、身份、手机号）

一般用户 =（用户 ID）

老年用户 =（用户 ID、真实姓名、身份证号、居住地、健康数据）

家属、看护用户 =（用户 ID、老年用户 ID、真实姓名、身份证号）

养老、殡葬机构管理人员 =（用户 ID、职务、管理机构）

民政部监管职员 =（用户 ID、职务、工号）

养老机构 =（养老机构 ID、所属街镇、养老机构名称、核定床位数、执业年月、地址、联系电话、邮编、运营模式、法定代表人）

殡葬机构　　=（殡葬机构 ID、殡葬机构名称、地址、邮编、联系电话、联系人、机构性质、服务范围）

机构评价　　=（机构类型、机构 ID、评价、评分、举报信息）

2. 将实体之间的关系转化为关系模式

1 对 1 关系：将一方的主键加入另一方并设置为外码，加入关系本身的属性；1 对 n 关系：将 1 方的主码加入 n 方作为外码，将关系本身的属性加入 n 方；n 对 m 关系：将关系转换一个一个关系模式，将双方主码加入其中设为外码并将关系属性加入其中，转化结果如下：

用户　　=（用户 ID、用户名、密码、身份、手机号）

一般用户　　=（用户 ID）

老年用户　　=（用户 ID、真实姓名、身份证号、居住地、健康数据）

家属、看护用户　=（用户 ID、老年用户 ID、真实姓名、身份证号）

养老、殡葬机构管理人员　=（用户 ID、职务、管理机构）

民政部监管职员　=（用户 ID、职务、工号）

养老机构　　=（养老机构 ID、所属街镇、养老机构名称、核定床位数、执业年月、地址、联系电话、邮编、运营模式、法定代表人）

殡葬机构　　=（殡葬机构 ID、殡葬机构名称、地址、邮编、联系电话、联系人、机构性质、服务范围）

机构评价　　=（机构类型、机构 ID、评价、评分、举报信息）

预约　　=（用户 ID、预约机构类型、机构 ID、身份、预约时间，预约状态活动）

呼救　　=（老年用户 ID、家属、看护用户 ID、身体健康数据、呼救类型）

4.1.2.2. 将关系模式转换为 3NF 范式

将多值属性进行拆分，转化为 1NF 范式；消除部分依赖，转化为 2NF 范式；消除传递依赖，转化为 3NF 范式。转化结果如下：

用户　　= (用户 ID, 用户名, 密码, 手机号)

用户身份　= (用户 ID, 身份)

一般用户　= (用户 ID)

老年用户　= (用户 ID, 真实姓名, 身份证号, 居住地)

老年用户健康数据　= (用户 ID, 健康数据)

家属用户　= (用户 ID, 真实姓名, 身份证号)

家属老年用户关系　= (家属用户 ID, 老年用户 ID)

管理人员　= (用户 ID, 职务)

管理机构　= (用户 ID, 管理机构)

监管职员 = (用户 ID, 职务, 工号)

养老机构 = (养老机构 ID, 所属街镇, 养老机构名称, 核定床位数, 执业年月, 地址, 联系电话, 邮编, 运营模式, 法定代表人)

殡葬机构 = (殡葬机构 ID, 殡葬机构名称, 地址, 邮编, 联系电话, 联系人, 机构性质, 服务范围)

机构评价 = (评价 ID, 机构类型, 机构 ID, 评价, 评分, 举报信息)

预约 = (预约 ID, 用户 ID, 预约机构类型, 机构 ID, 身份, 预约时间, 预约状态)

呼救 = (呼救 ID, 老年用户 ID, 呼救类型)

呼救用户 = (呼救 ID, 家属用户 ID, 身体健康数据)

以上关系模式即为转化出的 3NF 范式，其中不存在多值属性，部分依赖和传递依赖。

4.1.2.3. 关系模式优化

上述关系模型中的数据冗余主要用于建立数据之间的联系，目的是消除部分和传递依赖以及提高数据使用的便利性。这些冗余是必要的，例如在各个表中重复使用的 ID 等相关信息。

非必要的冗余非常少，通常体现为将各个实体的属性与 ID 相关联，在表中使用 ID 来替代整个实体的属性，从而减少了不必要的数据冗余。

4.1.3. 物理设计

4.1.3.1. 事务访问特性分析

建立数据库索引应遵循以下原则：

- 在查询频率高、数据量大的数据表上建立索引，以提高查询效率。
- 在经常作为查询条件且区分度高的字段上建立索引。
- 由于索引会降低插入和更新的效率，因此应尽量控制索引的数量。

在本系统中，访问频率较高的表包括：用户；老年用户；老年用户健康数据；家属、看护用户；机构管理人员；民政部监督人员；养老机构；殡葬机构；机构评价。因此，应在这些表上建立索引。

这些表都有唯一标识 ID，可以唯一区分数据，所以可以在其主键 ID 上建立索引。此外，对于一些经常在查询中使用的字段，如名字，也可以建立索引。

4.1.3.2. 索引结构构建

表名	属性	索引类型
用户	用户 ID	主键索引
	用户名	普通索引
	手机号	普通索引
用户身份	身份 ID	普通索引
老年用户	用户 ID	主键索引
	身份证号	唯一索引
老年用户健康数据	用户 ID	主键索引
家属、看护用户	用户 ID	主键索引
	身份证号	唯一索引
机构管理人员	用户 ID	主键索引
民政部机构人员	用户 ID	主键索引
	工号	唯一索引
养老机构	机构 ID	主键索引
	机构名称	普通索引
殡葬机构	机构 ID	主键索引
	机构名称	普通索引

4.2. 应用系统设计

4.2.1. 系统架构设计

系统将采用 **B/S（浏览器/服务器）架构**，该架构能够通过浏览器直接访问应用，无需安装额外客户端。

技术栈：

前端技术：HTML、CSS、JavaScript

- **HTML：**构建网页结构。
- **CSS：**页面样式和布局。
- **JavaScript：**前端交互，处理用户操作。

后端技术：Flask（Python 框架）

- **Flask：作为轻量级的 Web 框架，处理前端请求，进行数据处理，并返回结果。**

数据库：MySQL

- **使用 MySQL 来存储系统的数据，**如用户信息、预约记录、健康数据等。

B/S 架构说明：

用户通过浏览器与服务器进行交互，浏览器端负责呈现用户界面，后端服务器处理业务逻辑并与数据库进行交互。

系统通过 API 接口（RESTful API）实现前后端数据交互。

4.2.2. 系统功能模块设计

4.2.2.1. 用户管理模块

- **功能**：用户注册、登录、权限管理。
- **用户角色**：管理员、普通用户、家属用户等，依据角色分配不同权限。
- **前端界面**：用户输入信息，提交注册/登录表单，后端验证用户信息并返回结果。
- **后端功能**：用户身份验证、角色分配、JWT 令牌生成。

4.2.2.2. 养老机构管理模块

- **功能**：查看养老机构资料、评价、预约等功能。
- **前端界面**：展示各养老机构的详细信息，支持评价和预约操作。
- **后端功能**：查询养老机构数据、保存评价、处理预约请求。

4.2.2.3. 健康监测模块：

- **功能**：实时获取老年用户的健康数据（如心率等），当出现异常时进行报警。
- **前端界面**：展示健康数据，提供实时监控界面。
- **后端功能**：接收来自设备的数据，进行分析并生成报警信息。

4.2.2.4. 预约管理模块

- **功能**：用户可预约访问养老机构，显示预约状态。
- **前端界面**：显示可预约的机构列表，选择预约时间，提交预约。
- **后端功能**：处理预约请求，更新数据库中的预约状态。

4.2.2.5. 通知与呼叫模块：

- **功能**：当老年用户健康数据异常或在紧急情况下发出呼救，家属和相关人员能够及时接收到通知。
- **前端界面**：展示紧急呼叫按钮，处理通知弹窗。
- **后端功能**：接收紧急呼叫数据并推送通知。

4.2.3. 前端设计

用户界面设计：

- 界面简洁、易于操作。
- 提供用户登录/注册界面，养老机构信息展示，预约和评价功能。
- 支持不同设备（手机、平板、电脑）的自适应布局，保证用户在不同平台上的使用体验。

4.2.4. 后端设计

Flask 框架设计：

- 使用 Flask 框架来实现用户请求处理、路由、数据库交互。
- 每个功能模块对应一个或多个 API，处理前端请求并返回 JSON 数据。

API 设计：

- 设计 RESTful API 接口用于前后端数据交互。例如：POST /register、POST /login、GET /institutions（查询机构列表）、POST /evaluate（提交评价）等。

五、详细设计

5.1. 数据库设计

5.1.1. 数据表设计

5.1.1.1. 用户表（User）

该表用于存储用户的基本信息，如用户名、密码、联系电话等。系统支持多种用户类型（如管理员、普通用户、家属用户等），通过 UserRoles 表来区分不同角色的权限。

外键关系：无，UserID 是主键，其他表通过 UserID 关联。

5.1.1.2. 用户身份表（UserRoles）

该表用于存储用户的身份信息，每个用户只能有一个身份标识，如管理员、普通用户等。通过 UserID 外键关联到 User 表。

外键关系：UserID 引用 User 表。

5.1.1.3. 一般用户表（GeneralUser）

用于存储一般用户的基本信息，通过 UserID 外键关联到 User 表。

外键关系：UserID 引用 User 表。

5.1.1.4 老年用户表（SeniorUser）

用于存储老年用户的详细信息，如真实姓名、身份证号、居住地址等。此表通过 UserID 外键与 User 表关联。

外键关系：UserID 引用 User 表。

5.1.1.5 老年用户健康数据表（SeniorHealthData）

用于存储老年用户的健康监测数据（如心率等）。此表通过 UserID 外键与 SeniorUser 表关联，记录每个用户的健康状况。

外键关系：UserID 引用 SeniorUser 表。

5.1.1.6. 家属/看护用户表（FamilyCaregiver）

用于存储家属或看护用户的信息，如姓名、身份证号等。通过 UserID 外键关联到 User 表。

外键关系：UserID 引用 User 表。

5.1.1.7. 家属老年用户关系表（FamilySeniorRelation）

用于表示家属用户与老年用户之间的关系。每条记录代表一个家属用户与一个老年用户的关联关系。

外键关系：FamilyCaregiverID 引用 FamilyCaregiver 表，SeniorUserID 引用 SeniorUser 表。

5.1.1.8. 机构管理人员表（Staff）

用于存储管理养老机构的工作人员信息，记录其职位。通过 UserID 外键与 User 表关联。

外键关系：UserID 引用 User 表。

5.1.1.9. 管理机构表（StaffInstitution）

用于表示每个管理人员负责的养老机构。每个管理人员可以管理多个机构。

外键关系：UserID 引用 Staff 表，InstitutionID 引用 ElderCareInstitution 表。

5.1.1.10. 民政部监管职员表（Supervisor）

用于存储监管职员的基本信息，包括工号和职务。通过 UserID 外键与 User 表关联。

外键关系：UserID 引用 User 表。

5.1.1.11 养老机构表（ElderCareInstitution）

存储养老机构的详细信息，包括名称、地址、床位数等。用户可以查看这些信息并进行预约。

外键关系：无，InstitutionID 是主键，其他表通过 InstitutionID 关联。

5.1.1.12 殡葬机构表（FuneralInstitution）

存储殡葬机构的详细信息，如名称、地址、联系人等。用户可以查看这些信息并进行预约。

外键关系：无，InstitutionID 是主键，其他表通过 InstitutionID 关联。

5.1.1.13 养老机构评价表（ElderCareInstitutionEvaluation）

存储用户对养老机构的评价信息，包括评分和评价内容。通过 UserID 和 InstitutionID 外键关联 User 表和 ElderCareInstitution 表。

外键关系：UserID 引用 User 表，InstitutionID 引用 ElderCareInstitution 表。

5.1.1.14 殡葬机构评价表（FuneralInstitutionEvaluation）

存储用户对殡葬机构的评价信息。通过 UserID 和 InstitutionID 外键关联 User 表和 FuneralInstitution 表。

外键关系：UserID 引用 User 表，InstitutionID 引用 FuneralInstitution 表。

5.1.1.15 预约表（Reservation）

存储用户对养老机构的预约信息，包括预约时间、预约状态等。通过 UserID 和 InstitutionID 外键关联 User 表和 ElderCareInstitution 表。

外键关系：UserID 引用 User 表，InstitutionID 引用 ElderCareInstitution 表。

5.1.1.16 呼救表（EmergencyCall）

用于存储老年用户发出的呼救信息，记录呼救类型等。通过 SeniorUserID 外键关联 SeniorUser 表。

外键关系：SeniorUserID 引用 SeniorUser 表。

5.1.1.17 呼救用户表（EmergencyCallUser）

用于存储家属用户与老年用户之间的呼救信息，记录家属用户的心率等健康数据。

外键关系：EmergencyCallID 引用 EmergencyCall 表，FamilyCaregiverID 引用 FamilyCaregiver 表。

5.1.1.18 权限表（Permissions）

用于存储系统中的权限信息，每条记录代表一种系统权限。

外键关系：无。

5.1.1.19 角色权限表（RolePermissions）

用于定义用户角色与权限之间的关系，支持不同角色的权限管理。通过 RoleID 和 PermissionID 外键关联 UserRoles 表和 Permissions 表。

外键关系：RoleID 引用 UserRoles 表，PermissionID 引用 Permissions 表。

5.1.2. 数据库关系与操作

在本部分中，将详细描述如何利用数据库表进行常见的增、删、查、改等操作，并结合系统中的实际业务逻辑进行实现。数据库操作包括插入数据、查询数据、更新数据以及删除数据等，下面逐一进行说明：

插入操作

在用户注册过程中，系统会先向 User 表插入用户的基本信息。

- **插入用户基本信息：**当新用户注册时，系统首先会将用户的基本信息（如用户名、密码、联系电话等）插入到 User 表。

- **插入用户身份信息：**注册完成后，根据用户的角色（如普通用户、家属用户等），系统将向 UserRoles 表插入相应的角色信息，确保用户身份的正确设置。

查询操作

查询操作通常是通过 JOIN 操作来从多个表中检索相关信息。以查询某个老年用户的健康数据为例，系统会通过 JOIN 语句连接 SeniorUser 表和 SeniorHealthData 表来获取老年用户的详细信息。

更新操作

当老年用户的健康数据（如心率）发生变化时，系统需要更新 SeniorHealthData 表中的数据。更新操作通常通过 UPDATE 语句来完成。

删除操作

删除操作通常是通过 DELETE 语句来进行的。在删除用户时，系统需要保证删除操作的顺序和完整性，确保不会产生孤立的记录。

此过程遵循外键约束的原则，确保删除操作不会违反数据的完整性，避免数据丢失或孤立记录。

5.1.3. 数据库优化

为了提高系统的性能，特别是在处理大规模数据时，数据库优化至关重要。以下是几种常见的优化手段：

建立索引

为了提高查询效率，系统应在常用查询字段上建立索引。例如，在 UserID 和 InstitutionID 上建立索引，能够加速用户信息查询及养老机构相关操作。

这些索引能够显著加快查询速度，尤其是当数据量较大时，能避免全表扫描。

SQL 优化

对于复杂的查询，使用 JOIN 时可以结合索引进行优化，以提高查询效率。例如，在查询多个表时，避免全表扫描，并确保在连接条件中使用索引字段：

通过在连接条件中使用索引字段，可以提高多表查询的效率，减少查询时的性能瓶颈。

5.1.4. 数据库安全性与完整性

确保数据库的安全性和完整性对于系统的稳定运行至关重要。以下是几种常见的安全性和完整性设计：

完整性约束

在数据库设计中，完整性约束用于确保数据的准确性和一致性。例如，SeniorHealthData 表中的 HeartRate 字段不能为空（NOT NULL），系统必须确保每个老年用户都拥有心率记录。

该约束确保了数据的完整性，防止用户健康数据缺失。

5.2. 应用系统设计

5.2.1. 系统架构设计

前端与后端交互：使用 RESTful API，前端通过 HTTP 请求与后端进行数据交换。

前端通过 AJAX（或 Fetch API）发起请求，后端通过 Flask 路由处理请求，并返回 JSON 数据。

使用 JWT 进行用户身份认证，每次请求需要在请求头中携带 JWT token。

5.2.2. 功能模块详细设计

5.2.2.1 用户管理模块

前端设计：

- 用户输入用户名、密码、联系方式等信息，提交到后端进行注册/登录
- 登录成功后，前端通过 JavaScript 保存 JWT token，用于后续接口请求。

后端设计：

- 用户表 User 存储用户名、密码、角色信息。
- 登录接口：POST /login，接收用户名和密码，验证后生成 JWT token 返回前端。
- 注册接口：POST /register，接收用户信息，将其存入数据库。

数据库设计：

设计 User 表，包含字段：UserID（主键）、Username、Password、PhoneNumber、Role。

5.2.2.2. 养老机构管理模块

前端设计：

- 显示养老机构列表及其详细信息，提供预约按钮，点击后提交预约请求。
- 提供评价功能，用户可以对机构进行评分并写评价。

后端设计：

- 机构表 ElderCareInstitution 存储养老机构的详细信息（名称、地址、联系方式等）。
- 评价接口：POST /evaluate，接受用户评价、评分等数据，并存入 ElderCareInstitutionEvaluation 表。
- 预约接口：POST /reserve，接受用户预约请求，将其存入 Reservation 表。

数据库设计：

ElderCareInstitutionEvaluation 表用于存储评价信息，Reservation 表用于存储预约信息。

5.2.2.3. 健康监测模块

前端设计：

显示实时健康数据（如心率等），并在数据异常时给出警告。

后端设计：

- 健康数据表 SeniorHealthData 存储每个老年用户的健康数据（如心率等）。
- 定时任务或外部接口获取设备的健康数据，并存入数据库。
- 触发报警通知家属或相关人员。

数据库设计：

SeniorHealthData 表包含字段：UserID（外键）、HeartRate（心率）、Timestamp（时间戳）。

5.2.2.4. 预约管理模块

前端设计：

- 用户选择养老机构和预约时间，提交预约请求。
- 展示预约状态（未预约、已预约、已完成、已逾期等）。

后端设计：

预约接口：POST /reserve，接受预约请求，处理数据并更新 Reservation 表中的状态。

数据库设计：

Reservation 表包含字段：ReservationID（主键）、UserID（外键）、InstitutionID（外键）、ReservationTime、ReservationStatus。

5.2.2.5. 通知与呼叫模块

前端设计：

显示紧急呼叫按钮，用户可点击发起呼叫，家属可接收通知。

后端设计：

- 呼叫数据表 EmergencyCall 存储呼叫信息（包括呼叫类型、呼叫时间等）。
- 呼叫时触发系统发送通知到相关家属或看护人员。

数据库设计：

EmergencyCall 表包含字段：EmergencyCallID、SeniorUserID（外键）、CallType（自主/自动）。

5.2.3. API 接口设计

用户管理：

POST /login：接收用户名、密码，验证成功后返回 JWT token。

POST /register：接收用户信息，注册新用户。

养老机构管理：

GET /institutions：查询所有养老机构。

POST /evaluate：提交养老机构评价。

POST /reserve：提交预约请求。

健康监测：

GET /healthdata：查询老年用户的健康数据。

六、数据库建立、应用系统实现与功能调试

6.1. 数据库建立

6.1.1. 数据准备

为每一张表按照字段和要求准备具体的数据。其中养老机构和殡葬机构的数据来自上海公共数据开放平台。

6.1.2. 数据库环境搭建

数据库安装与配置：在开发环境中安装 MySQL 数据库，配置数据库实例和用户权限。

```
# create_db.py
import mysql.connector
from db_config import DB_CONFIG

conn = mysql.connector.connect(
    host=DB_CONFIG['host'],
    user=DB_CONFIG['user'],
    password=DB_CONFIG['password']
)

cursor = conn.cursor()

cursor.execute("CREATE DATABASE IF NOT EXISTS pension_management;")

print("数据库创建成功！")

cursor.close()
conn.close()

```

数据库连接配置：设置数据库连接配置文件，确保应用系统能够正确连接到数据库。

```
# MySQL 配置信息
DB_CONFIG = {
    'host': 'localhost',
    'port': 3306,
    'user': 'xxx',
    'password': 'xxxxxxx',
    'database': 'pension_management'
}
```

测试数据库连接：使用简单的查询操作测试数据库连接是否正常，确保没有权限或配置错误。

6.1.3. 数据库表创建

SQL 语句编写： 根据设计文档和数据字典，编写 SQL 语句创建各个表，确保表结构和字段与设计一致。

```
# 创建 Users 表
cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS Users (
        UserID INT PRIMARY KEY,
        Username VARCHAR(20) NOT NULL,
        Password VARCHAR(255) NOT NULL,
        PhoneNumber VARCHAR(15) NOT NULL
    );
""")

```

数据插入： 向表中插入数据。

```
def import_csv_to_mysql(csv_path, table_name):
    try:
        df = pd.read_csv(csv_path)

        df = df.where(pd.notnull(df), None)

        for i, row in df.iterrows():
            row = handle_nan(row)  # 处理 NaN
            sql = f"INSERT INTO {table_name} ({', '.join(df.columns)}) VALUES ({', '.join(['%s'] * len(row))})"
            cursor.execute(sql, tuple(row))

       conn.commit()
        print(f"Data from {csv_path} has been inserted into {table_name}.")
    except Exception as e:
        print(f"Error inserting data from {csv_path} into {table_name}: {e}")
        conn.rollback()


tables_and_files = {

'Users': 'Users.csv'
}

for table_name, csv_path in tables_and_files.items():
    if os.path.exists(csv_path):
        import_csv_to_mysql(csv_path, table_name)
    else:
        print(f"CSV file {csv_path} not found.")

```

6.2. 应用系统实现

6.2.1. 前端开发

- **前端框架与技术选型：** 前端采用 HTML、CSS 和 JavaScript 技术，设计简洁、响应式的用户界面，确保用户在不同设备（PC、手机、平板）上都能良好使用。
- **页面布局设计：** 根据系统功能模块，设计系统页面布局。例如，用户管理页面展示所有用户信息、老年用户健康数据页面展示心率等信息。
- **前端交互设计：** 使用 JavaScript 实现交互效果，例如表单验证、动态数据加载等功能。
- **前端与后端数据交互：** 使用 Ajax 或 Fetch API 与后端交互，获取和提交数据。确保前端可以通过 HTTP 请求与后端的 RESTful API 进行数据交互。

6.2.2. 后端开发

- **Flask 框架搭建：** 使用 Flask 框架搭建后端应用，创建必要的路由和视图函数，处理前端请求。
- **用户认证与授权：** 实现用户登录、注册功能，使用 JWT 或 Session 来验证用户身份。

6.2.3. 功能模块开发

- **用户管理模块：** 实现用户的注册、登录、权限管理等功能。
- **健康监测模块：** 实时监测老年用户的健康数据，并将数据展示在前端页面。
- **预约管理模块：** 用户可以通过系统进行预约，系统记录每次预约的时间和状态。
- **机构管理模块：** 提供养老机构和殡葬机构的管理功能，展示机构信息，允许用户对机构进行评价。

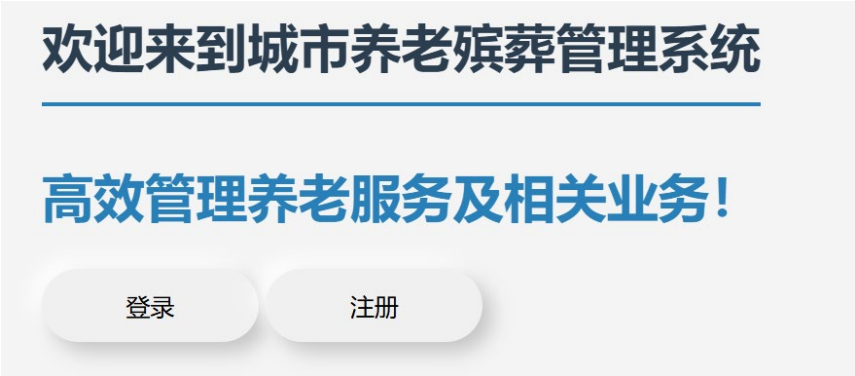
6.2.4. 前后端联调

- **接口联调：** 前后端开发人员共同测试接口，确保数据能够正确传递，并根据接口文档修复错误。
- **数据验证与错误处理：** 在后端进行数据验证，确保用户输入的数据合法，并处理潜在的错误（如数据库连接错误、用户输入无效等）。

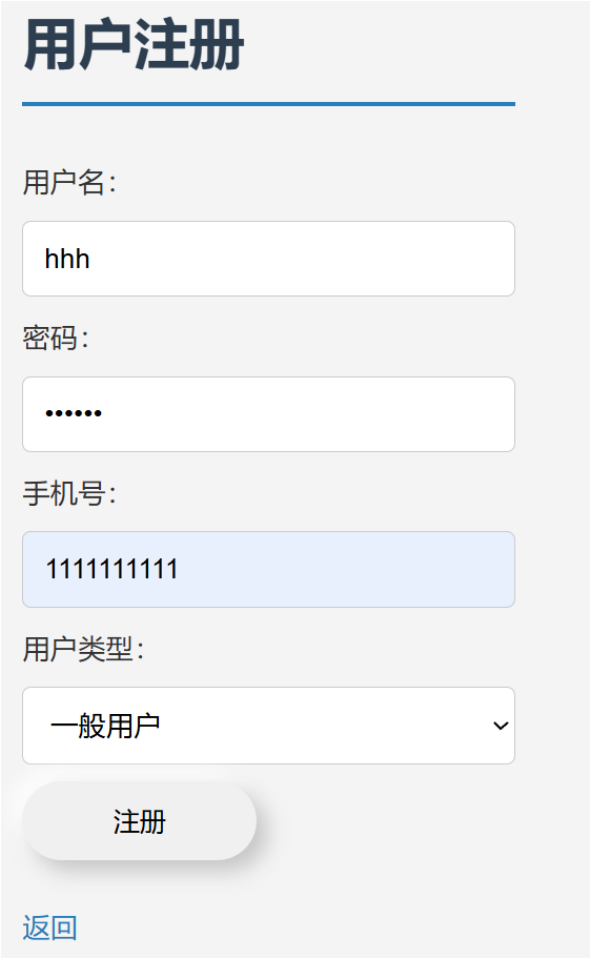
6.3. 功能调试

6.3.1. 用户注册、登录

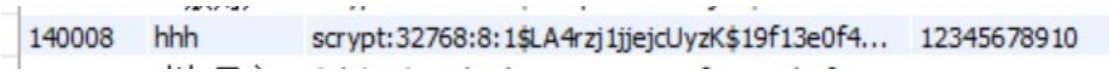
欢迎主页：

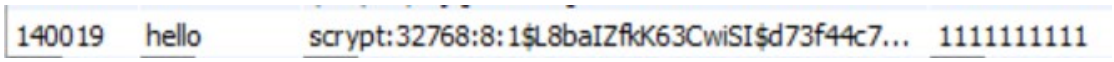


注册，根据不同用户类型有不同的信息要求：

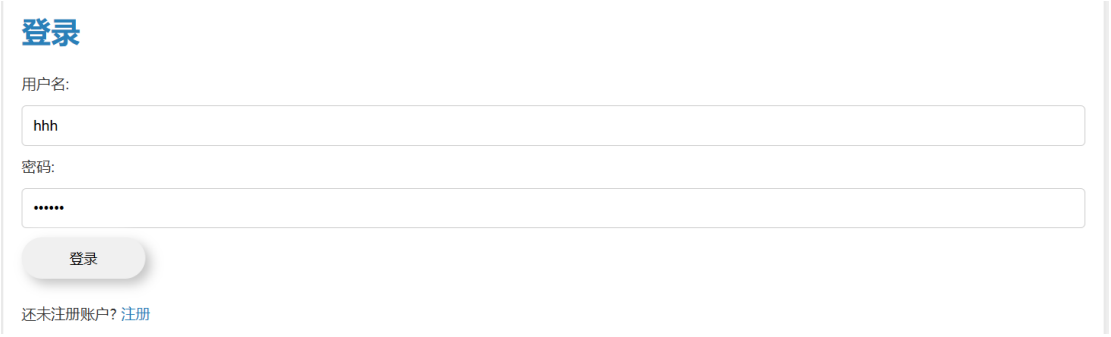


注册成功后密码通过哈希加密后存储到数据库中

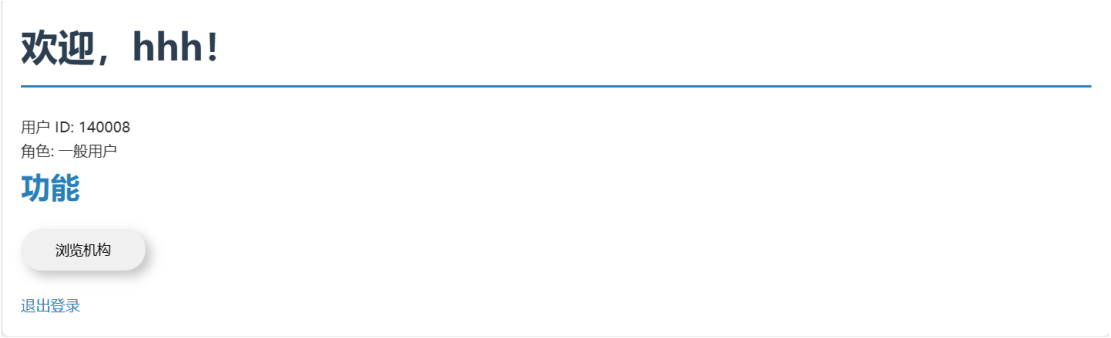




登录：

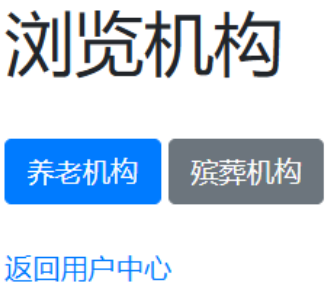


用户仪表盘：

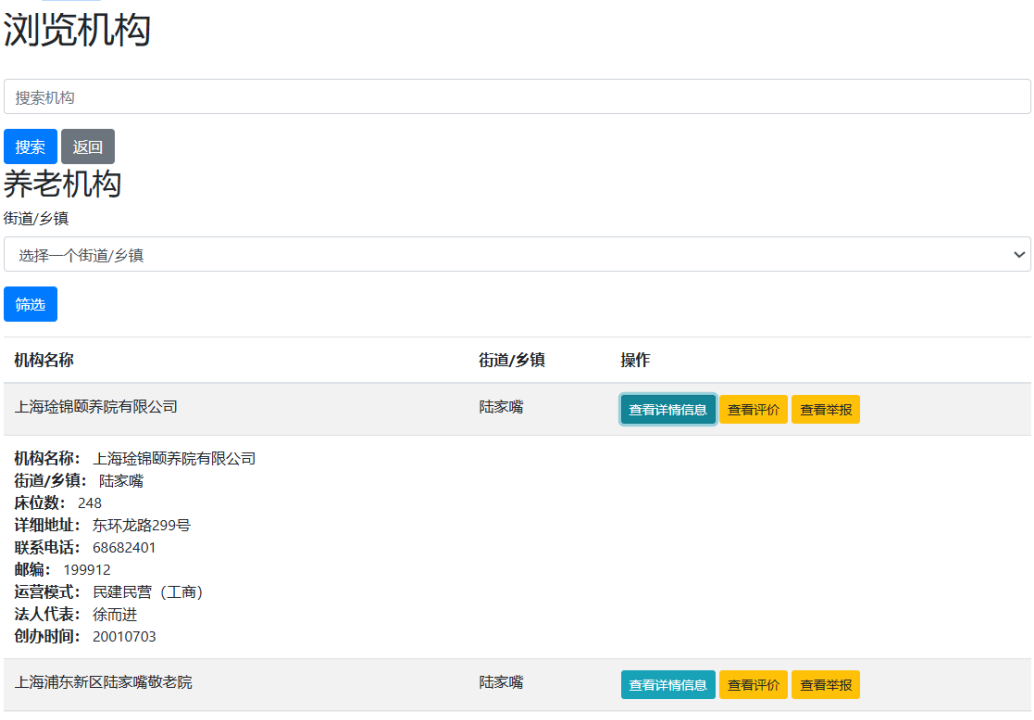


6.3.2. 浏览机构信息

选择机构类型：

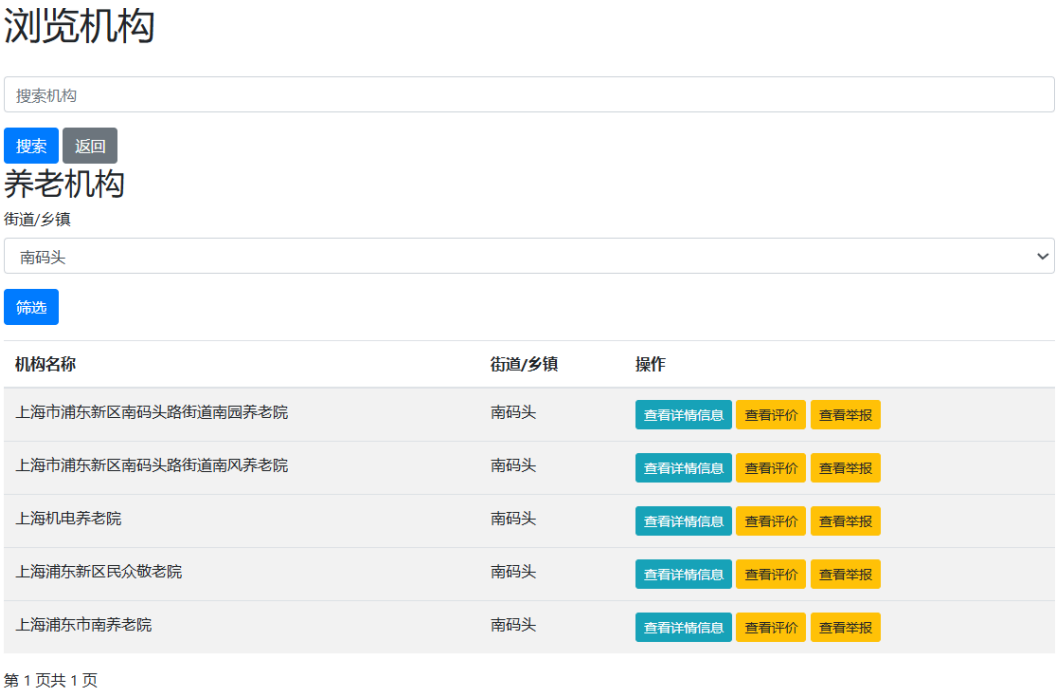


查看基本信息：



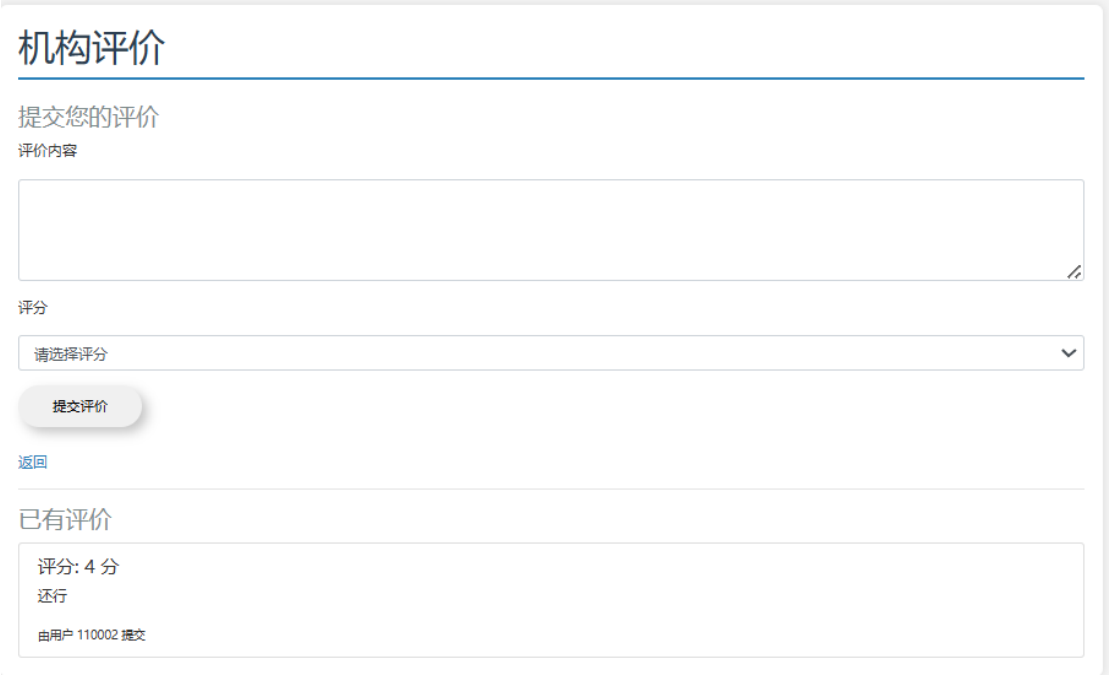
通过名字关键词搜索机构（陆家嘴）：

通过街道/乡镇（南码头）筛选机构：

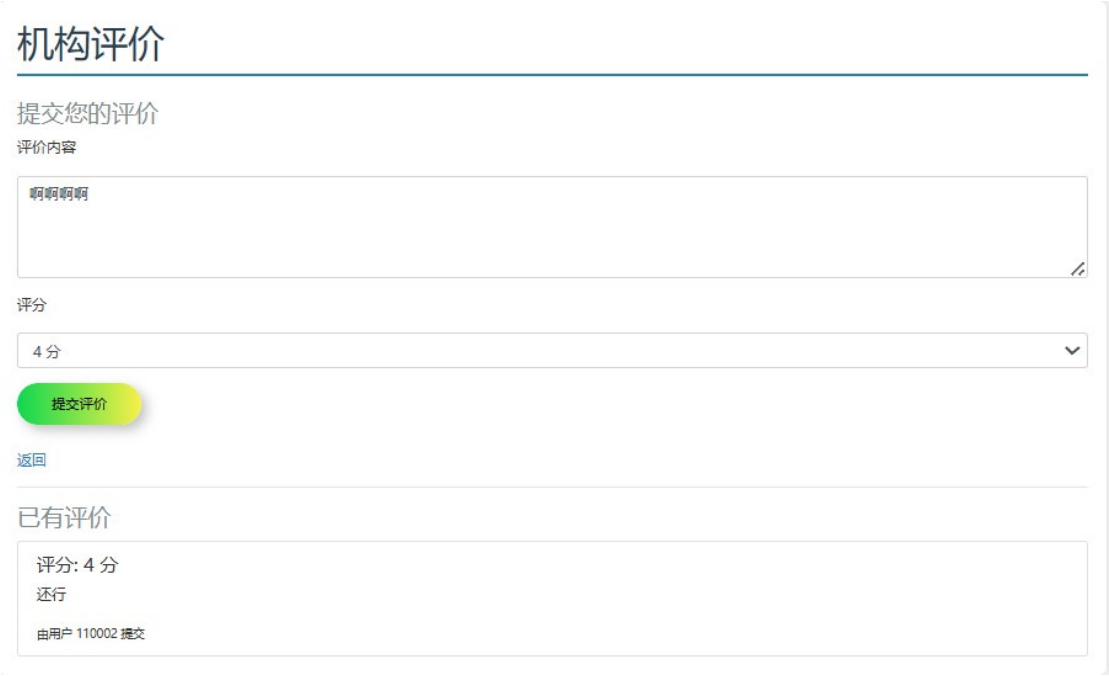


6.3.3. 机构评价、举报信息

浏览评价信息：

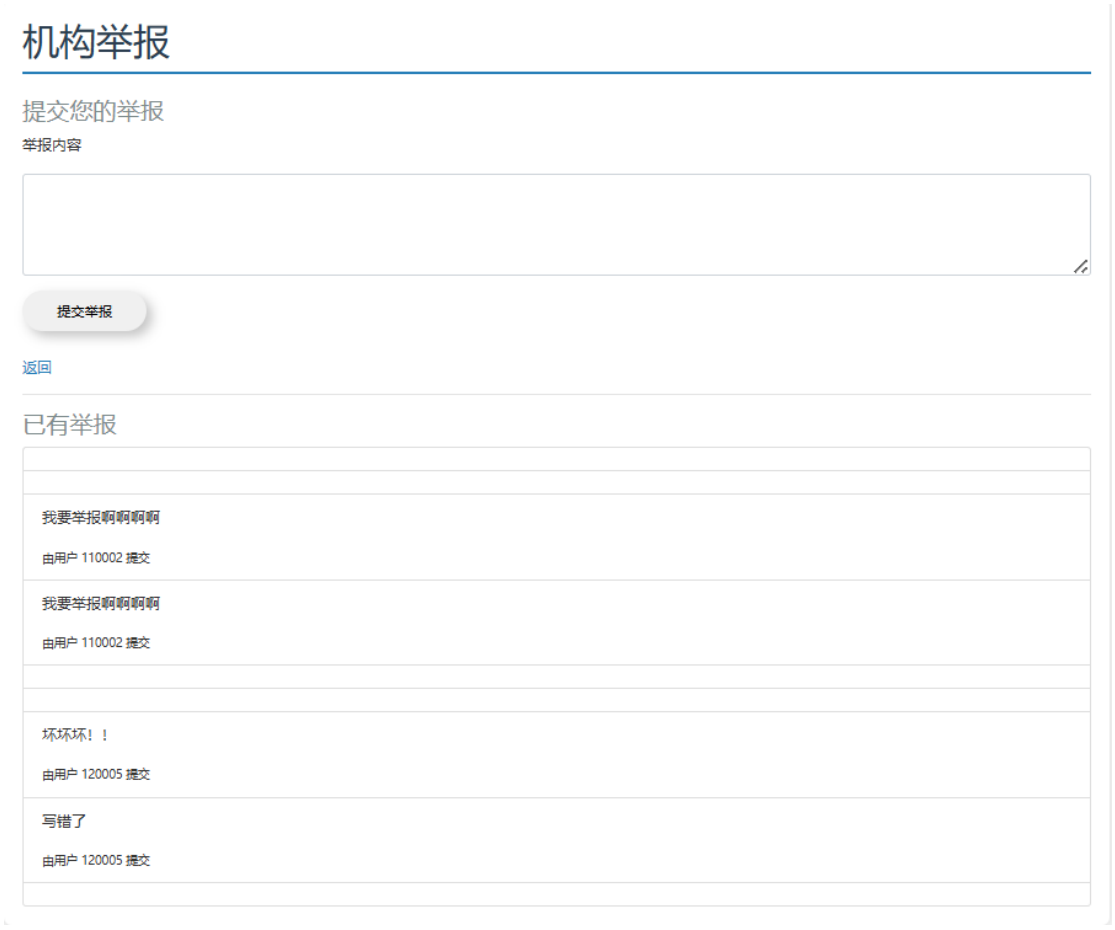


一般用户没有权限提交评价：

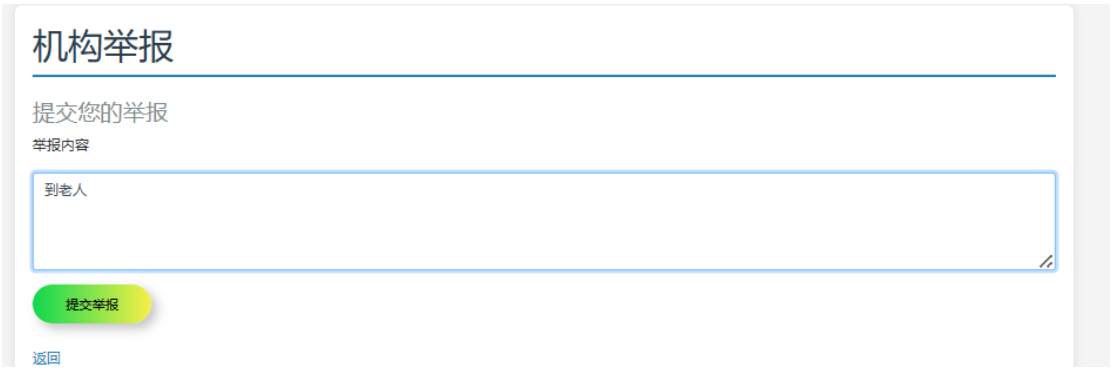



机构评价
提交您的评价
评价内容
啊啊啊啊
评分
4 分
提示
您没有权限进行评价
关闭

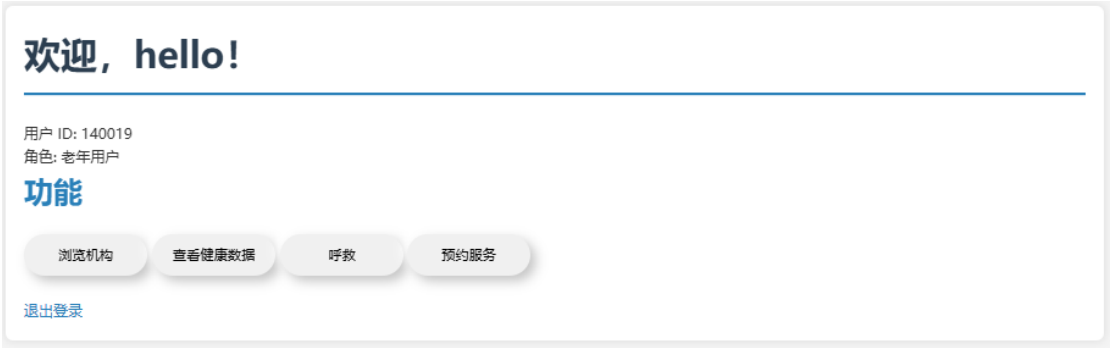
浏览举报信息：

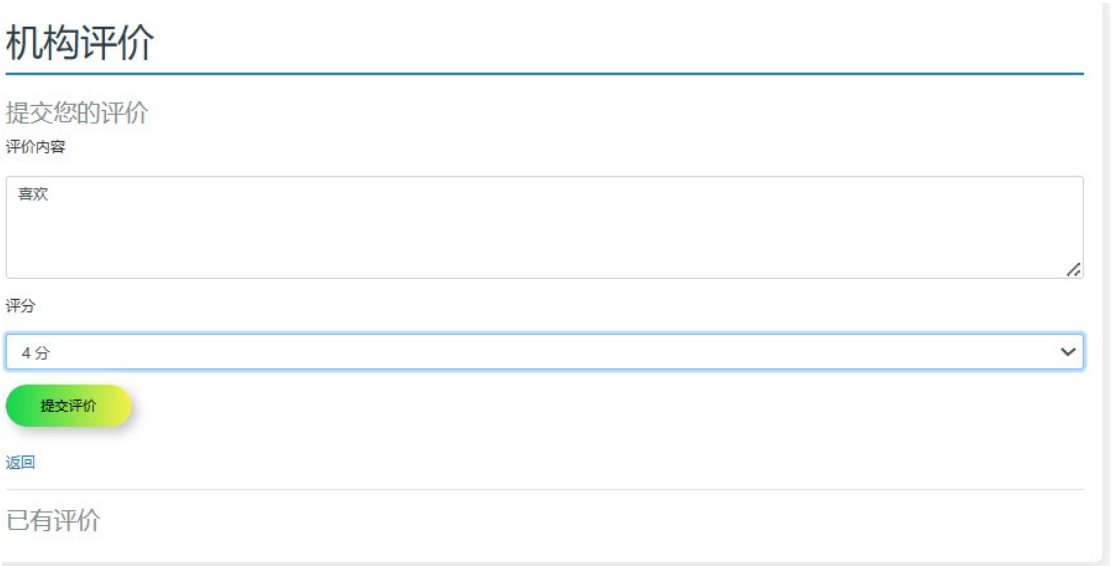

机构举报
提交您的举报
举报内容
提交举报
返回
已有举报
我要举报啊啊啊啊
由用户 110002 提交
我要举报啊啊啊啊
由用户 110002 提交
坏坏坏！！
由用户 120005 提交
写错了
由用户 120005 提交

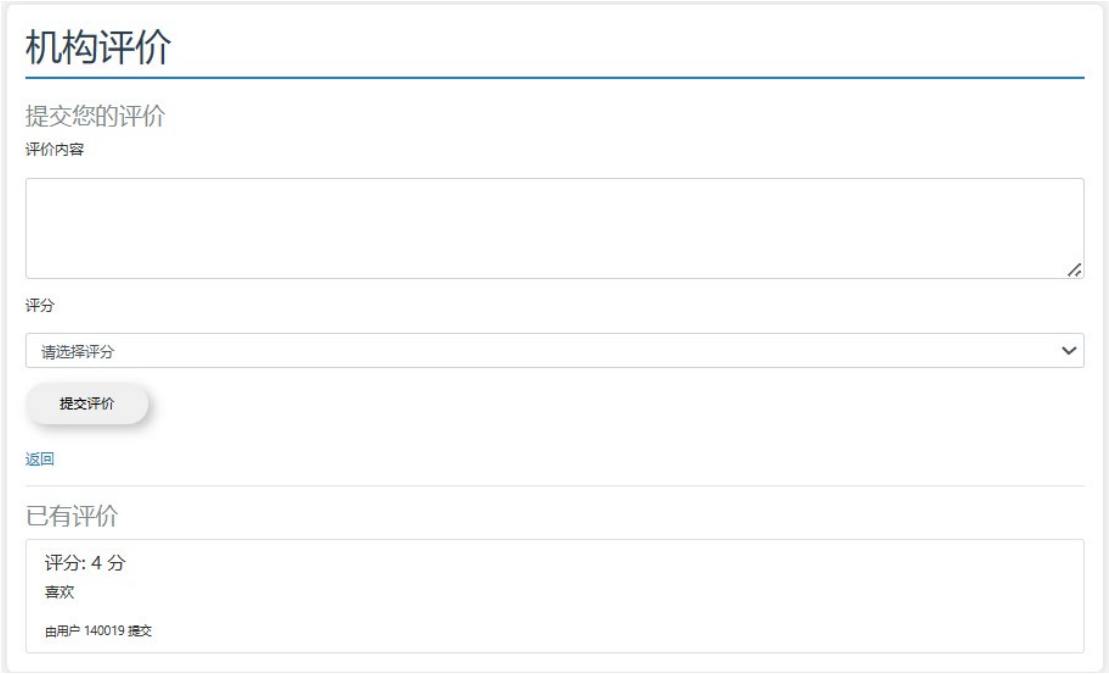
提交举报：



老年、家属/看护用户可以进行评价：

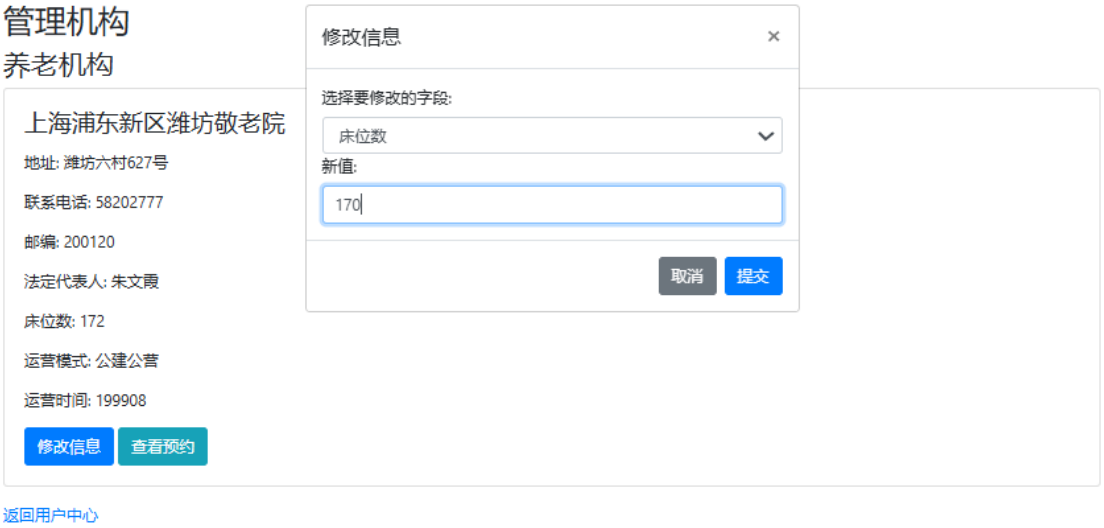


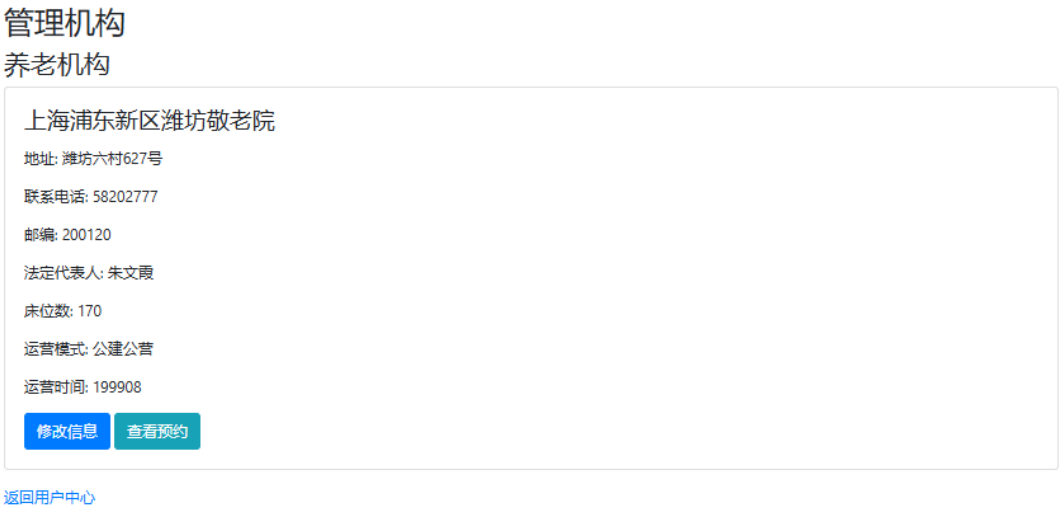




评价更新成功。

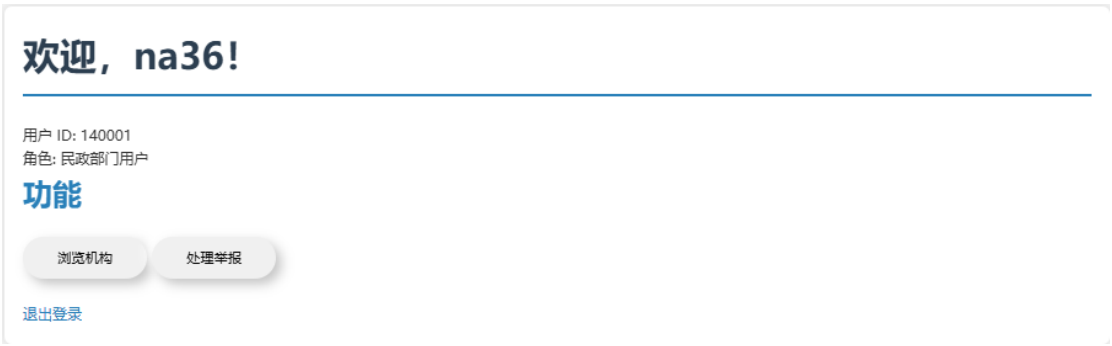
机构管理人员可以更新所管理的机构信息：





床位数更新成功。

民政部门用户可以处理举报信息：



管理举报信息

养老机构举报

机构名称: 上海瑜锦颐养院有限公司
机构ID: 200001
举报内容: 已处理
编辑举报| 删除举报
机构名称: 上海瑜锦颐养院有限公司
机构ID: 200001
举报内容: 坏坏坏！！
编辑举报| 删除举报
机构名称: 上海瑜锦颐养院有限公司
机构ID: 200001
举报内容: 写错了
编辑举报| 删除举报
机构名称: 上海浦东新区金枫敬老院
机构ID: 200005
举报内容: 你坏
编辑举报| 删除举报

殡葬机构举报

机构名称: 上海天使殡葬服务中心
机构ID: 210007
举报内容: 你坏!
编辑举报| 删除举报
机构名称: 上海市闸北区宝龙殡葬服务部
机构ID: 210019
举报内容: 怎么这样
编辑举报| 删除举报

返回用户中心

编辑举报信息：

编辑举报信息

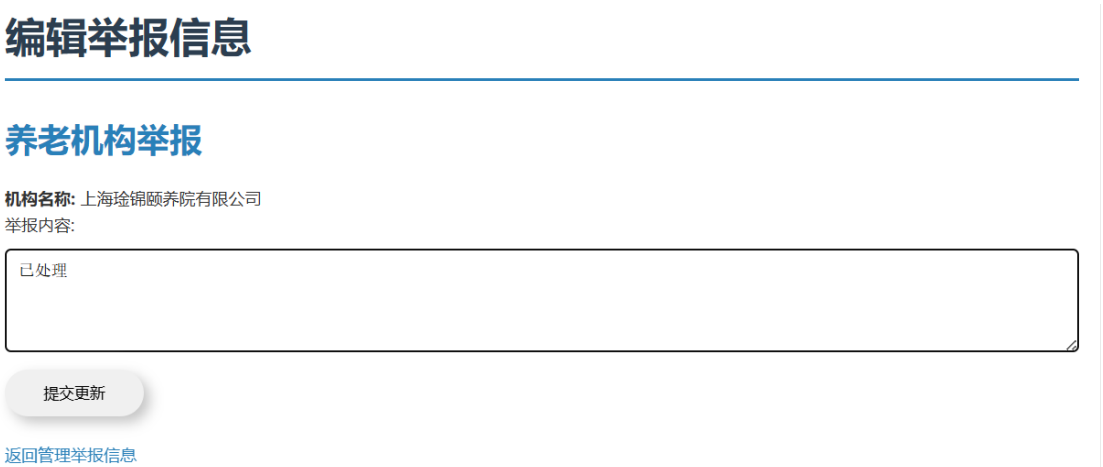
养老机构举报

机构名称: 上海瑜锦颐养院有限公司
举报内容:

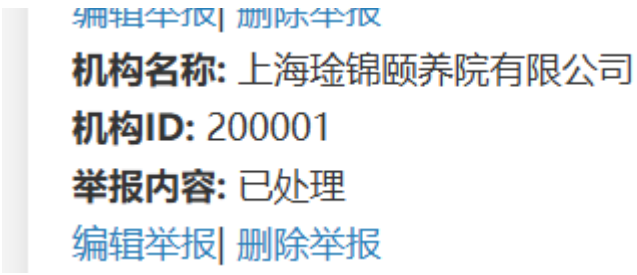
坏坏坏！！

提交更新

返回管理举报信息

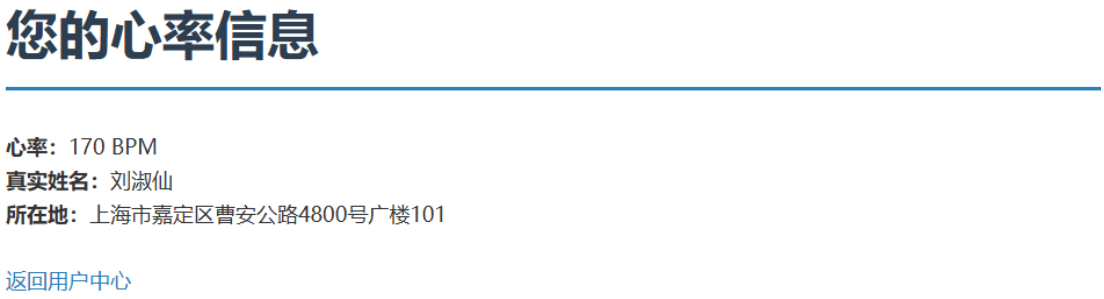


更新成功



6.3.4. 健康数据和呼救

老年用户可以查看自己的心率：



可以进行手动呼救(self)，当心率超出正常值(HeartRate < 40 BPM 或 HeartRate > 160 BPM)自动呼救(auto)：

您的呼救记录

呼救编号	呼救类型
500002	auto
500011	self
500015	self

发起新的呼救

自动呼救

返回用户中心

家属、看护用户可以查看绑定的老年人的健康数据：

您关联老年用户的心率信息

心率： 170 BPM
真实姓名： 刘淑仙
所在地： 上海市嘉定区曹安公路4800号广楼101

返回用户中心

家属、看护用户可以查看绑定的老年人的呼救：

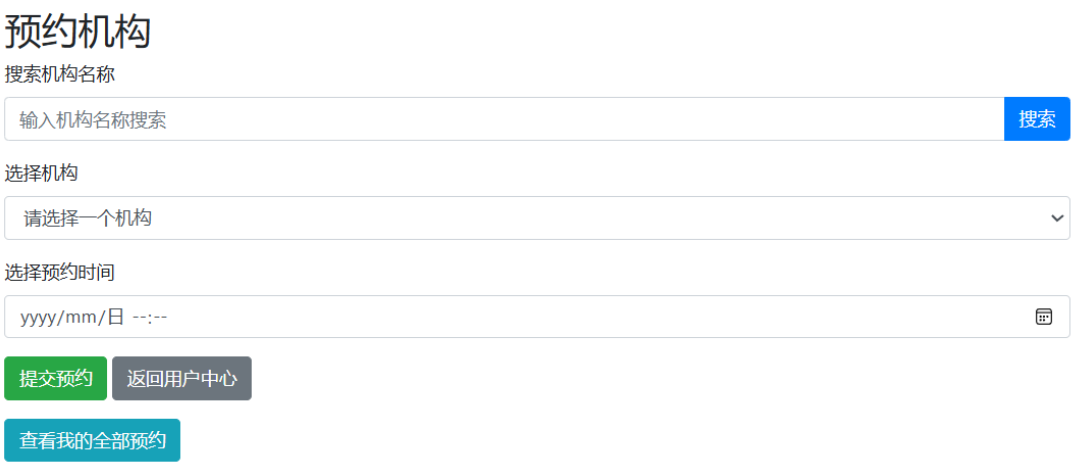
您关联老年用户的呼救记录

呼救编号	心率	老年用户编号	真实姓名	地址
500002	170	110006	刘淑仙	上海市嘉定区曹安公路4800号广楼101
500011	170	110006	刘淑仙	上海市嘉定区曹安公路4800号广楼101
500015	170	110006	刘淑仙	上海市嘉定区曹安公路4800号广楼101

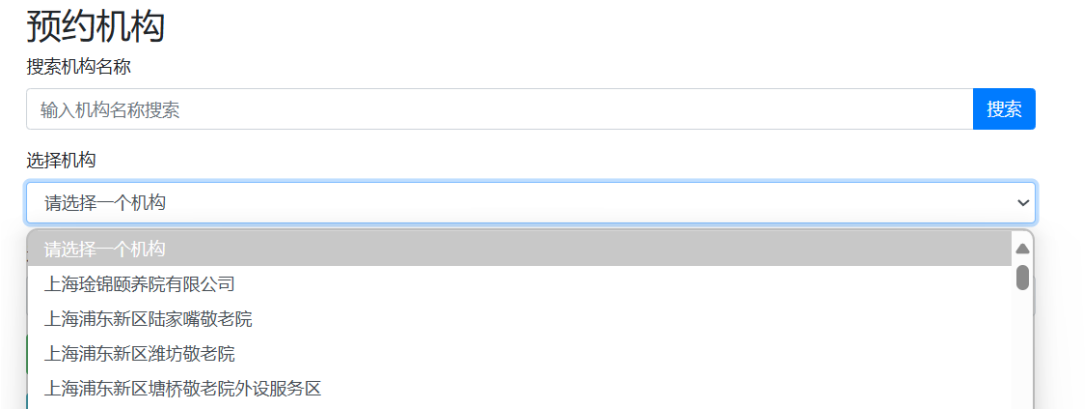
返回用户中心

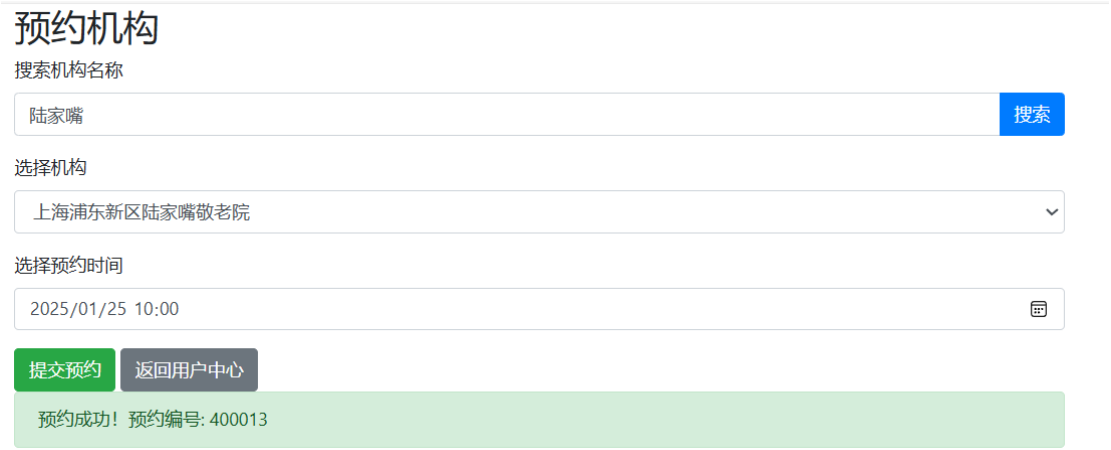
6.3.5. 机构预约

老年用户和家属、看护用户可以预约机构服务：



可以通过关键词搜索，也可以直接选择：





可以查看所有预约信息：

机构管理人员可以管理对应机构的预约信息：

管理机构

养老机构

上海浦东新区陆家嘴敬老院

地址: 崂山二村48号

联系电话: 58876896

邮编: 200120

法定代表人: 瞿文

床位数: 93

运营模式: 公建公营

运营时间: 199912

修改信息 查看预约

预约信息

预约 ID: 400004

用户 ID: 120001

预约时间: 2024/10/22 08:00:00

状态: 服务已完成

切换状态为 预约成功

预约 ID: 400010

用户 ID: 120005

预约时间: 2024/12/27 22:30:00

状态: 待机构确认

切换状态为 预约成功

预约 ID: 400013

用户 ID: 120006

预约时间: 2025/1/25 18:00:00

状态: 待机构确认

切换状态为 预约成功

返回用户中心

可以修改预约状态：

预约 ID: 400013

用户 ID: 120006

预约时间: 2025/1/25 18:00:00

状态: 预约成功

切换状态为 待机构确认 切换状态为 服务已完成

返回用户中心

七、数据库性能测试、系统综合测试，改进与完善

7.1. 数据库性能负载测试

为了评估 pension_management 数据库在高并发条件下的性能表现，我使用 sysbench 工具进行了负载测试。该测试模拟了多线程并发访问数据库的实际场景，主要通过执行读取、写入、更新等操作来测量数据库的吞吐量、延迟、错误率和资源消耗。

7.1.1. 测试设置

- **测试模式**：oltp_read_write 模式，模拟混合型的事务负载，包括读、写及其他类型操作。
- **线程数**：10 个并发线程。
- **测试时长**：60 秒。
- **数据库**：pension_management。
- **SQL 查询总数**：超过 60 万条。

```
1. sysbench --mysql-host=xxx.xx.xxx.xxx --mysql-user=root --mysql-password=xxxxxx --mysql-db=pension_management --threads=10 --time=60 --report-interval=1 --db-driver=mysql oltp_read_write prepare
2.
```

```
1. sysbench --mysql-host=xxx.xx.xxx.xxx --mysql-user=root --mysql-password=xxxxxx --mysql-db=pension_management --threads=10 --time=60 --report-interval=1 oltp_read_write run
2.
```

7.1.2. 主要性能指标

1. **吞吐量（TPS 和 QPS）：**
   - **事务每秒数（TPS）：**
     - 测试开始时，TPS 为 490.08，结束时为 509.08，表现出良好的稳定性。
     - 总事务数为 30,415 个，平均每秒处理 506.70 个事务。
   - **查询每秒数（QPS）：**
     - 测试过程中，QPS 在 60 秒内逐渐上升，从 9,812.67 增至 10,210.66，表明数据库能够在高负载下稳定执行查询操作。
     - 总查询数为 608,501 个，平均每秒处理 10,137.42 个查询。

2. **延迟：**
   - **95 百分位延迟**：测试中的 95%请求的响应时间为 29.72 毫秒，显示出在大多数情况下数据库能够快速响应请求。
   - **最大延迟**：最大延迟为 92.57 毫秒，表明在某些情况下响应时间会有所波动，但整体表现仍在可接受范围内。
   - **平均延迟**：平均延迟为 19.73 毫秒，表现出较为稳定的响应时间。
3. **错误率：**
   - 测试过程中，共发生了 12 次错误，约为每秒 0.20 次错误。错误率较低，表明数据库在高负载下的稳定性良好，能够可靠地处理大多数事务。
4. **重连接次数：**
   - 在整个测试过程中没有出现重连接现象，reconnects 为 0，表明数据库连接稳定，无需重新建立连接。
5. **线程公平性：**
   - 每个线程的平均事件数为 3,041.5，标准差为 101.06，表明各个线程间的负载分布较为均匀，未出现过多的负载不均问题。
   - 每个线程的执行时间平均为 59.99 秒，标准差为 0.01 秒，显示出线程之间的执行时间差异极小，线程调度较为公平。

```
SQL statistics:
    queries performed:
        read:                            425978
        write:                           121681
        other:                           60842
        total:                           608501
    transactions:                        30415  (506.70 per sec.)
    queries:                             608501 (10137.42 per sec.)
    ignored errors:                      12     (0.20 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          60.0243s
    total number of events:              30415

Latency (ms):
         min:                                    11.71
         avg:                                    19.73
         max:                                    92.57
         95th percentile:                        29.72
         sum:                                599941.40

Threads fairness:
    events (avg/stddev):           3041.5000/101.06
    execution time (avg/stddev):   59.9941/0.01
```

7.1.3. 性能分析与总结

- **吞吐量**：在 10 个并发线程的负载下，数据库每秒可以处理超过 10,000 条查询，并且事务处理能力稳定，TPS 值在测试过程中略有上升，表明系统能够在高负载下保持较好的性能。
- **延迟**：虽然有个别请求的最大延迟达到了 92.57 毫秒，但 95%的请求延迟均低于 30 毫秒，表明大部分请求响应速度较快，符合业务需求。
- **错误率和重连接**：低错误率和无重连接现象表明系统在高并发下依然保持高稳定性，能够可靠地处理大量并发请求。
- **线程调度公平性**：系统能够有效地分配任务给各个线程，保证了负载的均衡分配，从而提高了整体性能。

7.2. 系统综合测试

在本项目的系统综合测试阶段，我们开展了多项测试工作，以确保系统的各个功能模块能够在不同的环境和使用条件下稳定运行。除了数据库的性能负载测试外，我们还进行了黑盒测试、跨浏览器兼容性测试、以及移动端与 PC 端的适配性测试，以确保最终系统在实际部署时的高可靠性和良好用户体验。

7.2.1. 黑盒测试

在黑盒测试阶段，来自不同用户群体的测试人员对系统的核心功能进行了全面验证，重点检查了系统的功能正确性和业务流程的完整性。测试人员根据需求文档与功能设计，不涉及代码实现细节，专注于系统的输入与输出，验证了数据库查询、数据处理、用户交互等功能模块的有效性。

各种常见的业务场景和边界条件被测试人员反复验证，确保系统能够稳定地处理多种用户操作并输出正确的结果。通过多轮反馈与修复，保证了系统在实际用户使用中的稳定性。

7.2.2. 跨浏览器兼容性测试

我们在多个常见的浏览器环境下进行了测试，确保系统的前端页面在不同浏览器中的渲染效果一致性。测试涵盖了 Chrome、Firefox、Edge、Safari 等主流浏览器，验证了页面布局、交互功能、表单提交等核心功能的表现。

通过兼容性测试，我们发现并修复了一些浏览器特有的兼容性问题，使得系统能够在各大浏览器中流畅运行，提供一致的用户体验。

7.2.3. 移动端与 PC 端适配性测试：

在系统测试过程中，特别关注了移动端与 PC 端的适配性问题。测试团队在不同的操作系统环境下进行了广泛测试，包括 iOS、Android 系统上的浏览器兼容性，以及各类 PC 设备上的表现。

在移动端测试中，我们验证了系统的响应式设计，确保页面能够自动适应不同分辨率的手机屏幕，优化了触控操作的交互体验。此外，系统在低带宽和不同网络环境下的表现也得到了充分测试，以确保用户在各种网络条件下的稳定访问。

PC 端的测试确保了系统在桌面浏览器中的表现，包括页面加载速度、功能按钮、数据展示等都能够顺利操作，确保了桌面用户的流畅体验。

7.3. 改进与完善

在数据库性能优化中，SQL 查询的优化是提高系统响应速度和数据库效率的关键步骤。通过使用 EXPLAIN 和 EXPLAIN ANALYZE 来查看查询的执行计划，可以帮助我们识别查询瓶颈，并采取合适的优化措施。以下是对两条查询的分析及优化建议。

7.3.1. 查询分析—EXPLAIN 结果

查询 1：通过 InstitutionName 查询特定的养护院

```
1. EXPLAIN SELECT * FROM eldercareinstitution WHERE InstitutionName = '上海市浦东新区金桥镇张桥养护院';
2.
```

执行计划分析：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	eldercareinstitution	NULL	const	InstitutionName,idx_institution_name	InstitutionName	203	const	1	100.00	NULL

- **查询类型**：SIMPLE，表示查询没有使用联合查询或子查询。
- **使用的索引**：该查询使用了 InstitutionName 索引。通过 const 类型的连接，它能够快速定位到符合条件的记录。
- **行数估算**：rows 值为 1，表示该查询预期返回 1 行记录。
- **过滤比例**：filtered 为 100%，表示所有返回的记录都是符合查询条件的。

优化建议：

- **索引优化**：此查询使用了 InstitutionName 列的索引，可以有效地减少查询的扫描行数。若 InstitutionName 列的数据量较大，索引已经足够高效，因此此查询已经进行了合理优化。

查询 2：查询带有评分条件的养护院信息

```
1. EXPLAIN SELECT e.InstitutionName, ev.Rating
2. FROM eldercareinstitution e
3. JOIN elderCareinstitutionevaluation ev ON e.InstitutionID = ev.InstitutionID
4. WHERE ev.Rating > 4;
5.
```

执行计划分析：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ev	NULL	ALL	InstitutionID	NULL	NULL	NULL	13	33.33	Using where
1	SIMPLE	e	NULL	eq_ref	PRIMARY	PRIMARY	4	pension_management.ev.InstitutionID	1	100.00	NULL

- **查询类型**：SIMPLE，表示这是一个简单的查询。
- **ev 表**：查询首先扫描了 elderCareinstitutionevaluation 表（type: ALL），这表明没有使用索引，而是进行了全表扫描，可能会导致性能问题。查询过滤条件是 ev.Rating > 4，但未使用该列的索引来加速查询。
- **e 表**：对于 eldercareinstitution 表，使用了 eq_ref 类型的连接，这表示使用了主键索引 PRIMARY，通过外键与 ev.InstitutionID 进行匹配。每次匹配到 ev.InstitutionID，e 表的匹配结果是唯一的。
- **行数估算**：ev 表预计扫描 13 行记录，e 表预计扫描 4 行。
- **过滤比例**：ev 表的 filtered 为 33.33%，表示大部分记录不满足 ev.Rating > 4 的条件。

优化建议：

- **添加索引**：ev.Rating 列应当建立索引，以避免全表扫描，从而减少查询时间。
- **使用复合索引**：如果查询中经常使用 InstitutionID 和 Rating 两个列，可以考虑为这两个列创建一个复合索引，以加速查询效率。
- **考虑使用 INNER JOIN**：如果数据量较大，考虑是否需要进一步优化 JOIN 操作，确保其效率。

7.3.2. 查询分析—EXPLAIN ANALYZE 结果

查询 3：通过 InstitutionName 查询特定的养护院

```
1. EXPLAIN ANALYZE SELECT * FROM eldercareinstitution WHERE InstitutionName = '上海市浦东新区金桥镇张桥养护院';
2.
```

执行计划分析：

```
1. EXPLAIN
2. -> Rows fetched before execution  (cost=0..0 rows=1) (actual time=700e-6..700e-6 rows=1 loops=1)
3.
```

- **实际执行时间**：查询执行的实际时间非常短，仅为 700 微秒。
- **行数**：rows=1 表示查询返回了一行数据，符合预期。
- **优化方向**：由于查询执行时间非常短，因此不需要进一步优化。

7.3.3. 总结与优化方案

- **查询 1** 已经使用了有效的索引，并且查询条件简单，性能较好，无需优化。
- **查询 2** 由于缺少对 Rating 列的索引，导致查询在 eldercareinstitutionevaluation 表上进行了全表扫描。可以通过以下方式优化：
    1. 为 Rating 列添加索引，减少全表扫描的开销。
    2. 如果频繁使用 InstitutionID 和 Rating 两列进行查询，建议创建一个复合索引。
- **查询 3** 执行非常高效，表现良好，不需要优化。

优化建议的总结：

1.使用索引优化 WHERE 子句中频繁使用的字段。

2.对于 JOIN 操作，确保连接条件列（如 InstitutionID）已经建立索引。

3.对于频繁使用的组合查询条件，考虑创建复合索引。

八、系统运行维护

系统运行与维护是保证系统稳定性和高效性的重要工作。本章将简要介绍系统监控、性能优化、安全性保障、数据备份与恢复、更新与升级等方面的内容。

8.1 系统监控与性能维护

为了确保系统持续稳定运行，需要进行实时监控，及时发现和解决问题。

1. **服务器监控：**监控服务器的 CPU 使用、内存占用、磁盘空间等，避免资源耗尽。
2. **数据库监控：**定期检查数据库的性能，包括慢查询和资源使用情况，确保数据库高效运行。
3. **应用监控：**监控应用程序的响应时间、错误率等，确保用户体验不受影响。
4. **日志管理：**集中管理和分析系统日志，及时发现并解决潜在问题。

性能优化：

1.定期检查和优化数据库查询，确保系统响应速度。

2.使用压力测试工具模拟高并发情况，评估系统性能。

8.2 安全性维护

为了保障数据安全，需要采取一系列措施来防止攻击和数据泄露。

1. **漏洞扫描：**定期使用工具扫描系统漏洞，及时修补。
2. **数据加密：**对敏感数据进行加密，保护数据传输和存储的安全。
3. **权限管理：**实施严格的权限控制，确保只有授权人员能够访问敏感数据。

故障处理：

1.制定应急预案，应对系统出现故障时的快速处理。

2.配置实时故障监控，及时发现并处理问题。

8.3 数据备份与恢复

数据丢失或损坏可能会导致系统不可用，因此定期进行数据备份非常重要。

1. **备份策略：**结合全量备份和增量备份，定期保存数据。
2. **备份存储：**备份数据保存在本地和云端，防止单点故障。
3. **恢复测试：**定期进行恢复演练，确保备份数据可用。

8.4 系统更新与升级

系统更新与升级是保证系统性能和安全的关键。

1. **功能更新**：根据需求对系统进行功能升级。
2. **安全补丁**：定期安装安全补丁，修复已知的漏洞。
3. **性能优化**：通过优化数据库和应用结构，提高系统响应速度。

升级流程：

1.在升级前进行充分的测试，确保不影响现有功能。

2.采用平滑升级方式，避免系统停机，确保用户正常使用。

参考文献

[1] 彭希哲. 老龄化背景下的人口年龄结构[J/OL]. 上海交通大学学报(哲学社会科学版), 2023, 31(2): 14-24[2024-05-22]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST 2023&filename=SHJX202302002&v=. DOI:10.13806/j.cnki.issn1008-7095.2023.02.002.

[2] 翟振武, 金光照. 中国人口负增长：特征、挑战与应对[J/OL]. 人口研究, 2023, 47(2): 11-20[2024-05-22]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST 2023&filename=RKYZ202302002&v=.

[3] 杨菊华, 孙超. 我国离婚率变动趋势及离婚态人群特征分析[J/OL]. 北京行政学院学报, 2021(2): 63-72[2024-05-22]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST 2021&filename=XZXY202102008&v=. DOI:10.16365/j.cnki.11-4054/d.2021.02.008.

[4] 高强, 李洁琼, 孔祥智. 日本高龄者“孤独死”现象解析及对中国的启示[J/OL]. 人口 学刊, 2014, 36(1): 41-53[2024-05-22]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ& dbname=CJFD2014&filename=RKXK201401005&v=.

[5] 上海市公共数据开放平台[EB/OL]. [2024-05-24]. https://data.sh.gov.cn/index.html.

[6] 上海市养老机构-上海市民政局-上海市公共数据开放平台[EB/OL]. [2024-05-24]. https://data. sh.gov.cn/view/detail/index.html?type=cp&&id=AA9002014005&&dataset_name=%5Bobject%20HT MLHeadingElement%5D.

[7] 鱼小兰, 何佳洁. 我国不同类型养老机构运营现状、问题及对策分析[J/OL]. 卫生软科学, 2022, 36(7): 30-34[2024-05-22]. https://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ& dbname=CJFDLAST2022&filename=WRKX202207007&v=.

[8] 殡葬服务代理机构-上海市民政局-上海市公共数据开放平台[EB/OL]. [2024-05-24]. https://data. sh.gov.cn/view/detail/index.html?type=cp&&id=AA9002014013&&dataset_name=%5Bobject%20HT MLHeadingElement%5D.