Pet's Caregiver 宠物信息管理系统概要设计规约文档

目录

1. 引言

1.1 概要设计依据

- 杜庆峰老师上课所讲的概要设计基本原则;
- Pet's Caregiver-需求分析规约

1.2 参考资料

1. 《SpringBoot 更好的开发》
2. 《HTML5+CSS3 从入门到精通》

3. 《更好的软件架构，更好的设计》

4. 《Design Pattern》

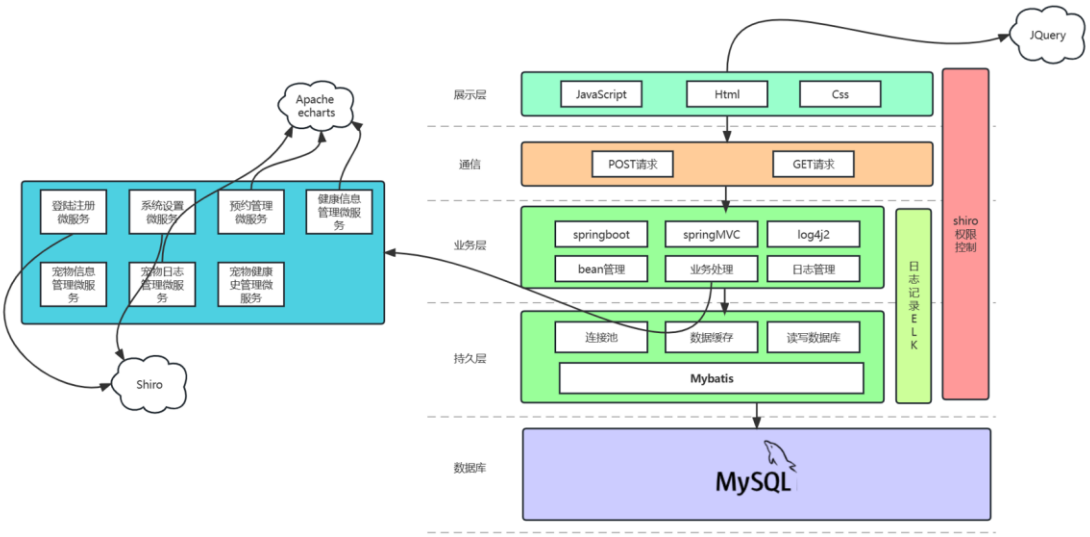
1.3 假定和约束

本项目开发主要受软件工程课程的约束，因此本项目的假定和约束如下所示：

1. 项目开发期限为 2 个月，时间为 2022 年 11 月~12 月底；
2. 项目开发无经费，设备条件为 2 台 Windows 操作系统电脑和 1 台 Mac 操作系统电脑以及阿里云平台等；
3. 项目在开发前线上通过问卷调研的方式收集了 104 份问卷，并据此制定了用户画像；
4. 在交流过程中，我们每三天线下汇报工作进度，同时通过 Github 进行代码协作管理。

2. 概要设计

2.1 系统总体架构设计



SSM（Spring+SpringMVC+MyBatis）框架集由 Spring、MyBatis 两个开源框架整合而成（SpringMVC 是 Spring 中的部分内容）。常作为数据源较简单的 web 项目的框架。

Spring 就像是整个项目中装配 bean 的大工厂，在配置文件中可以指定使用特定的参数去调用实体类的构造方法来实例化对象。也可以称之为项目中的粘合剂。Spring 的核心思想是 IoC（控制反转），即不再需要程序员去显式地 new 一个对象，而是让 Spring 框架帮你来完成这一切。

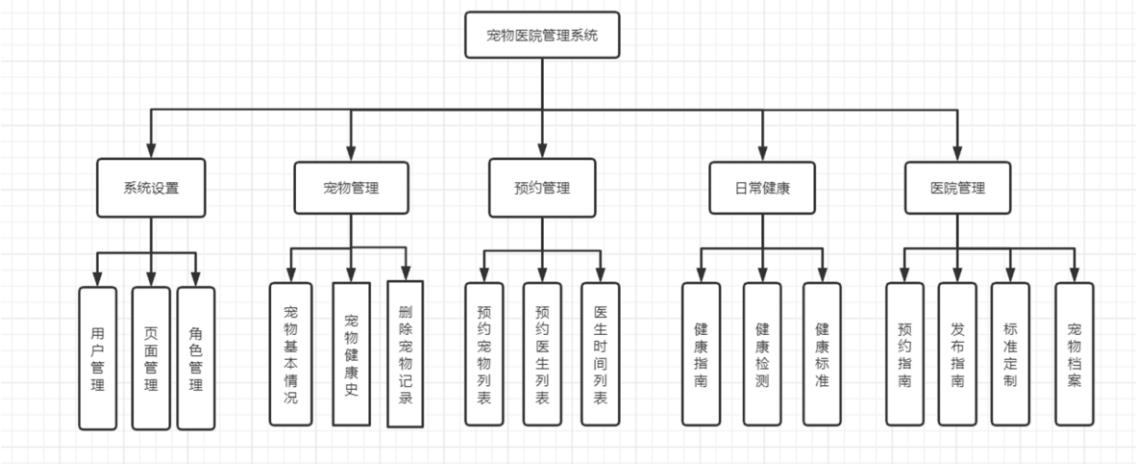
SpringMVC 在项目中拦截用户请求，它的核心 Servlet 即 DispatcherServlet 承担中介

或是前台这样的职责，将用户请求通过 HandlerMapping 去匹配 Controller，Controller 就是具体对应请求所执行的操作。SpringMVC 相当于 SSH 框架中 struts。

mybatis 是对 jdbc 的封装，它让数据库底层操作变的透明。mybatis 的操作都是围绕一个 sqlSessionFactory 实例展开的。mybatis 通过配置文件关联到各实体类的 Mapper 文件，Mapper 文件中配置了每个类对数据库所需进行的 sql 语句映射。在每次与数据库交互时，通过 sqlSessionFactory 拿到一个 sqlSession，再执行 sql 命令。

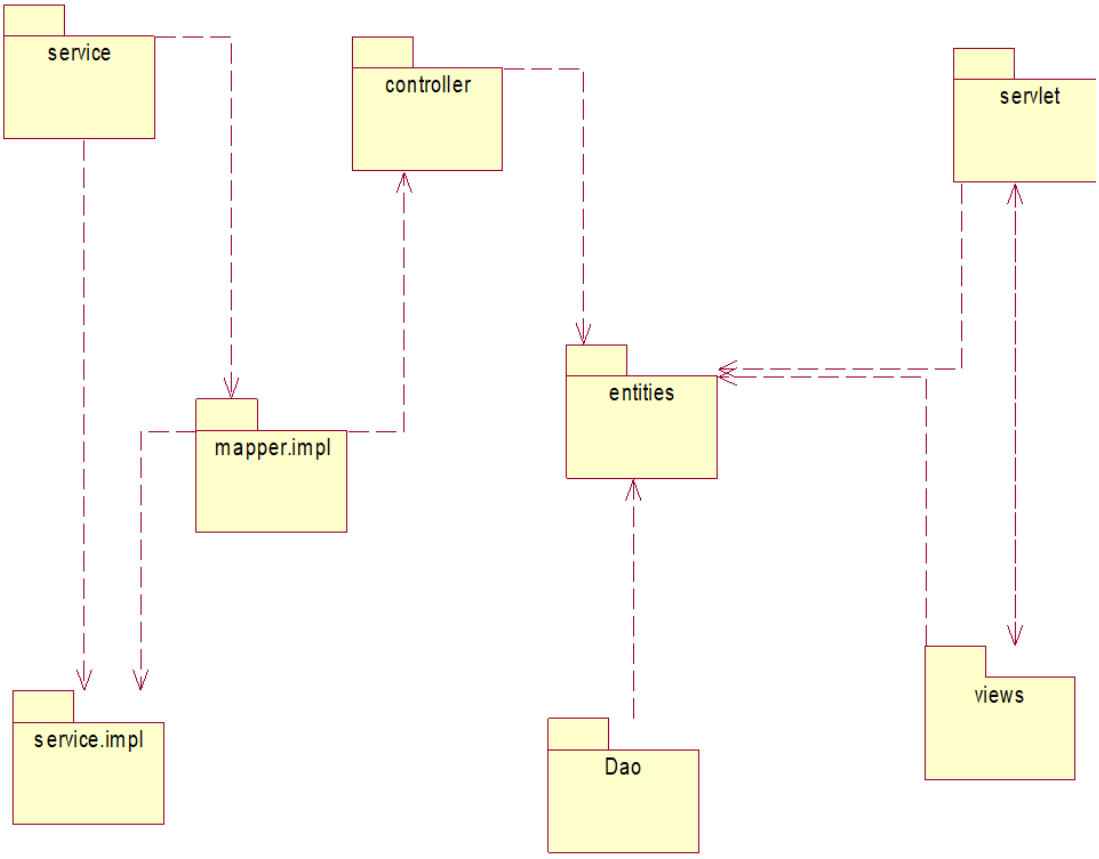
页面发送请求给控制器，控制器调用业务层处理逻辑，逻辑层向持久层发送请求，持久层与数据库交互，后将结果返回给业务层，业务层将处理逻辑发送给控制器，控制器再调用视图展现数据。

2.2 软件功能总体设计

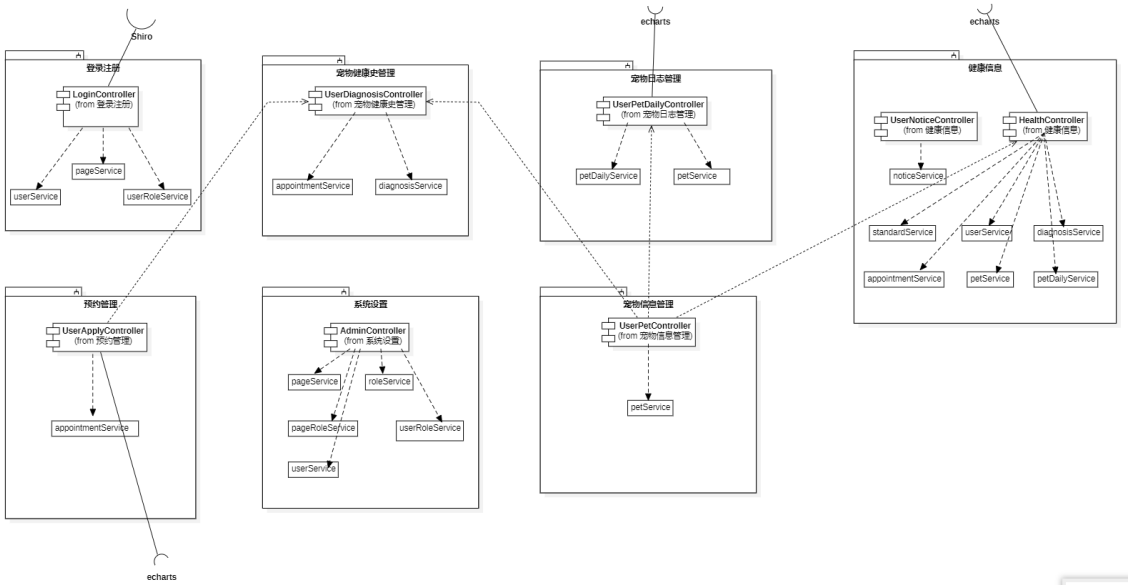


2.3 系统软件结构设计

体系结构包图:

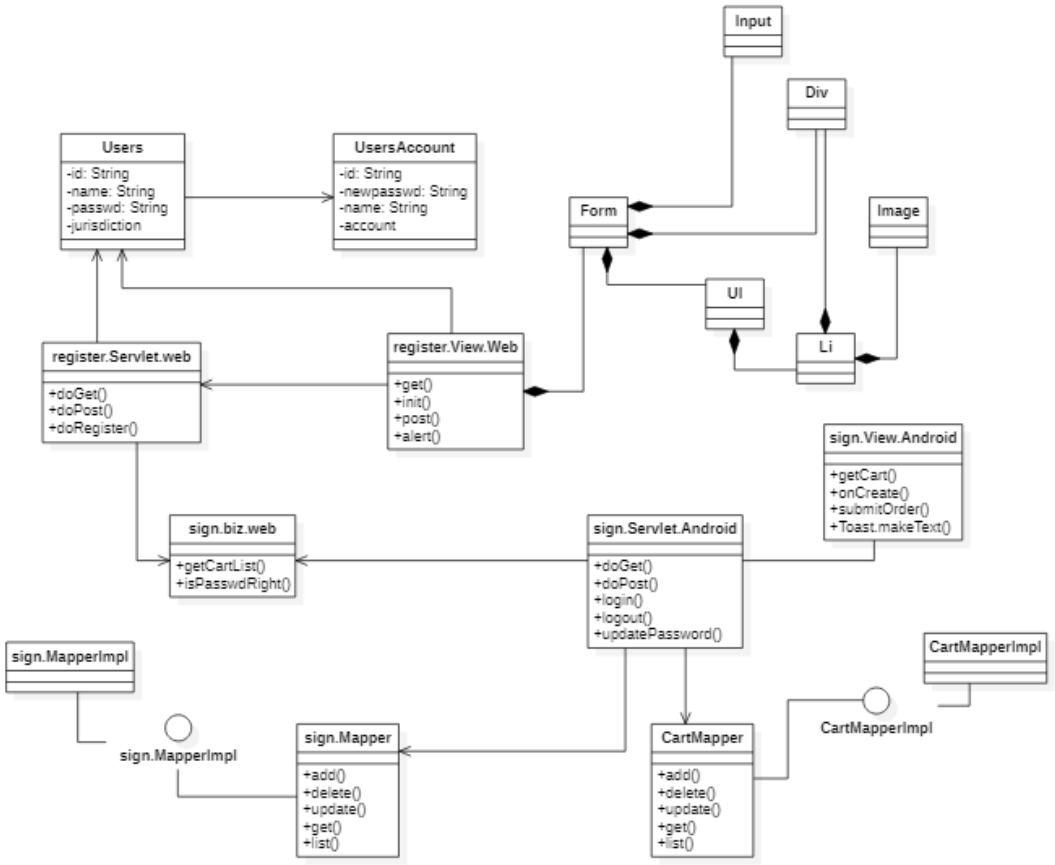


软件架构设计:



2.4 接口设计

2.4.1 登录注册微服务：



a. **login**

方法签名：public ResultMap login(String username, String password)

实际调用方法：LoginController.login

a. **logout**

方法签名：public String logout()

实际调用方法：LoginController.logout

a. **doRegist**

方法签名：public ResultMap doRegist(User user)

实际调用方法：LoginController.doRegist

a. **isPasswordRight**

方法签名：public boolean checkUserPassword(String password)

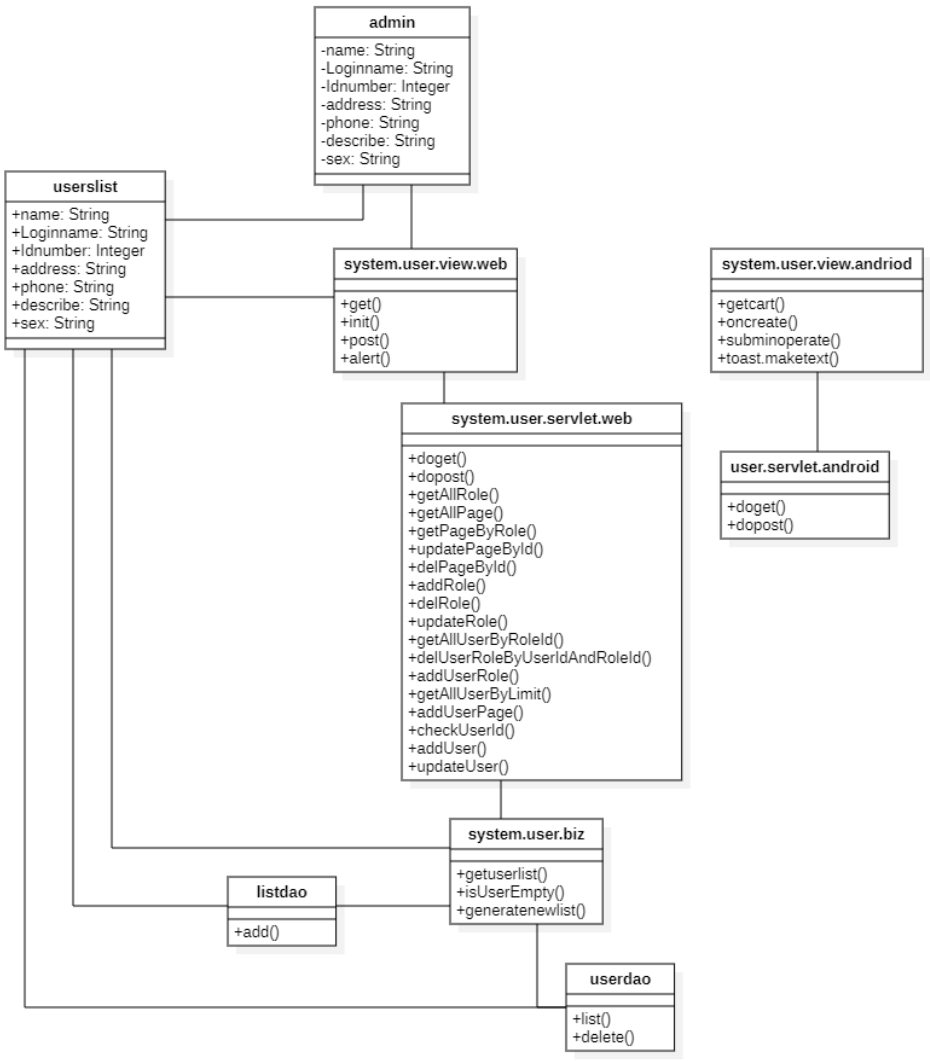
实际调用方法：LoginController.checkUserPassword

a. **updatePassword**

方法签名：public String updatePassword(String password)

实际调用方法：LoginController.updatePassword

2.4.2 系统设置微服务：



a. **getAllRole**

方法签名：public List<Role> getAllRole()

实际调用方法：AdminController.getAllRole

a. **getAllPage**

方法签名：public List<Page> getAllPage()

实际调用方法：AdminController.getAllPage

a. **getPageByRole**

方法签名：public Object getPageByRole(Integer roleId)

实际调用方法：AdminController.getPageByRole

a. **updatePageById**

方法签名：public ResultMap updatePageById(Page page)

实际调用方法：AdminController.updatePageById

a. **delPageById**

方法签名：public ResultMap delPageById(Integer id)

实际调用方法：AdminController.delPageById

a. **addRole**

方法签名：public String addRole(String name)

实际调用方法：AdminController.addRole

a. **delRole**

方法签名：public String delRole(int id)

实际调用方法：AdminController.delRole

a. **updateRole**

方法签名：public String updateRole(Integer id, String name)

实际调用方法：AdminController.updateRole

a. **addPageRoleByRoleId**

方法签名：public String addPageRoleByRoleId(Integer roleId, Integer[] pageIds)

实际调用方法：AdminController.addPageRoleByRoleId

a. **getAllUserByRoleId**

方法签名：public Object getAllUserByRoleId(Integer roleId, String roleNot, Integer page, Integer limit)

实际调用方法：AdminController.getAllUserByRoleId

a. **delUserRoleByUserIdAndRoleId**

方法签名：public ResultMap delUserRoleByUserIdAndRoleId(String userId, Integer roleId)

实际调用方法：AdminController.delUserRoleByUserIdAndRoleId

a. **addUserRole**

方法签名：public String addUserRole(Integer roleId, String[] userIds)

实际调用方法：AdminController.addUserRole

a. **getAllUserByLimit**

方法签名：public Object getAllUserByLimit(UserParameter userParameter)

实际调用方法：AdminController.getAllUserByLimit

a. **delUser**

方法签名：public String delUser(Long[] ids)

实际调用方法：AdminController.delUser

a. **addUserPage**

方法签名：public String addUserPage(Long userId, Model model)

实际调用方法：AdminController.addUserPage

a. **checkUserId**

方法签名：public User checkUserId(Long userId)

实际调用方法：AdminController.checkUserId

a. **addUser**

方法签名：public String addUser(User user)

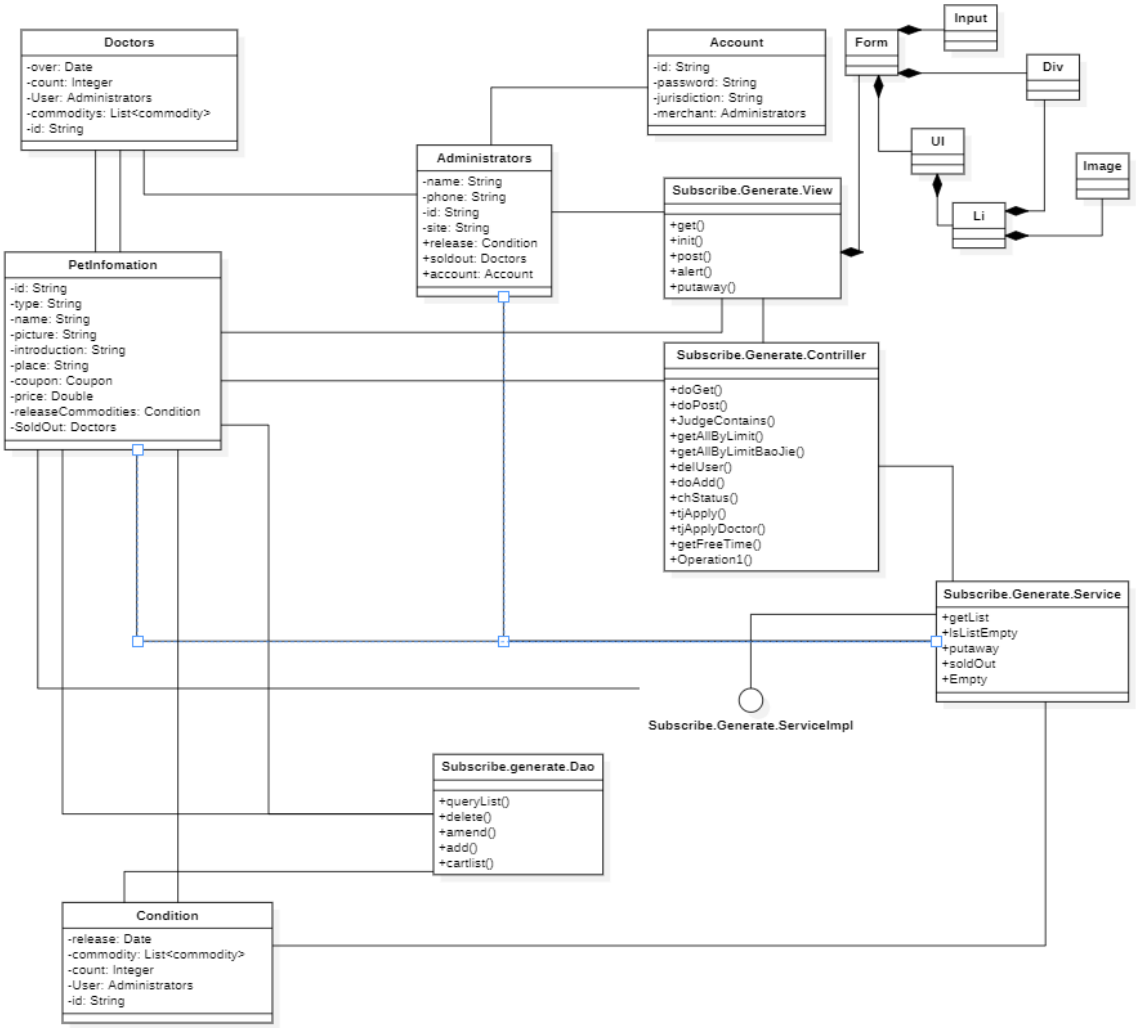
实际调用方法：AdminController.addUser

a. **updateUser**

方法签名：public String updateUser(User user, Long oldId)

实际调用方法：AdminController.updateUser

2.4.3 预约管理微服务



a. **getAllByLimit**

方法签名：public Object getAllByLimit(Appointment appointment)

实际调用方法：UserApplyController.getAllByLimit

a. **getAllByLimitBaoJie**

方法签名：public Object getAllByLimitBaoJie(Appointment appointment)

实际调用方法：UserApplyController.getAllByLimitBaoJie

a. **delUser**

方法签名：public String delUser(Long id)

实际调用方法：UserApplyController.delUser

a. **doAdd**

方法签名：public String doAdd(Appointment appointment)

实际调用方法：UserApplyController.doAdd

a. **chStatus**

方法签名：public String chStatus(Appointment appointment)

实际调用方法：UserApplyController.chStatus

a. **tjApply**

方法签名：public String tjApply(Model model)

实际调用方法：HealthController.tjApply

a. **tjApplyDoctor**

方法签名：public String tjApplyDoctor(Model model)

实际调用方法：HealthController.tjApplyDoctor

a. **getFreeTime**

方法签名：public Object getFreeTime(Long id, String date)

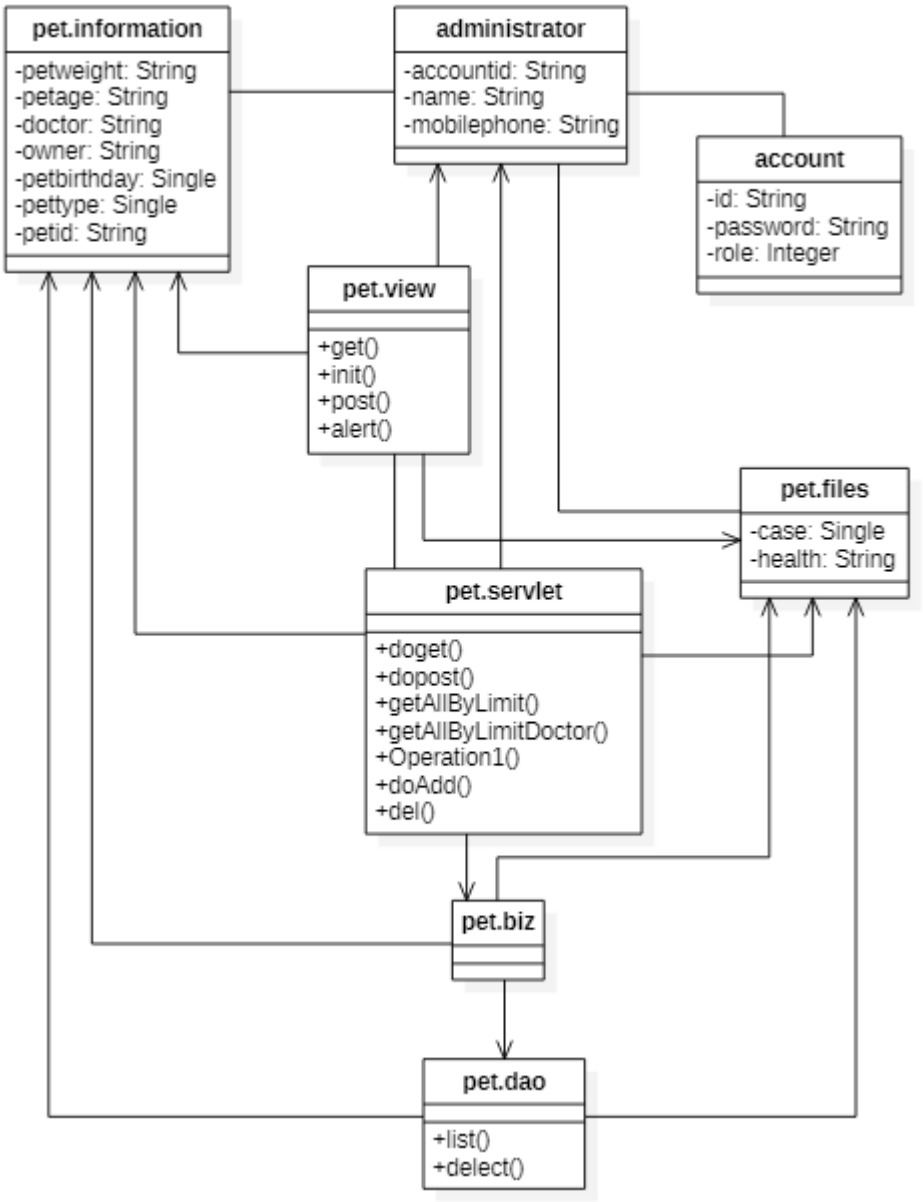
实际调用方法：HealthController.getFreeTime

a. **doAdd**

方法签名：public String doAdd(Diagnosis diagnosis)

实际调用方法：UserDiagnosisController.doAdd

2.4.4 宠物信息管理微服务



a. **getAllByLimit**

方法签名：public Object getAllByLimit(Pet pojo)

实际调用方法：UserPetController.getAllByLimit

a. **getAllByLimitDoctor**

方法签名：public Object getAllByLimitDoctor(Pet pojo)

实际调用方法：UserPetController.getAllByLimitDoctor

a. **doAdd**

方法签名：public String doAdd(Pet pojo)

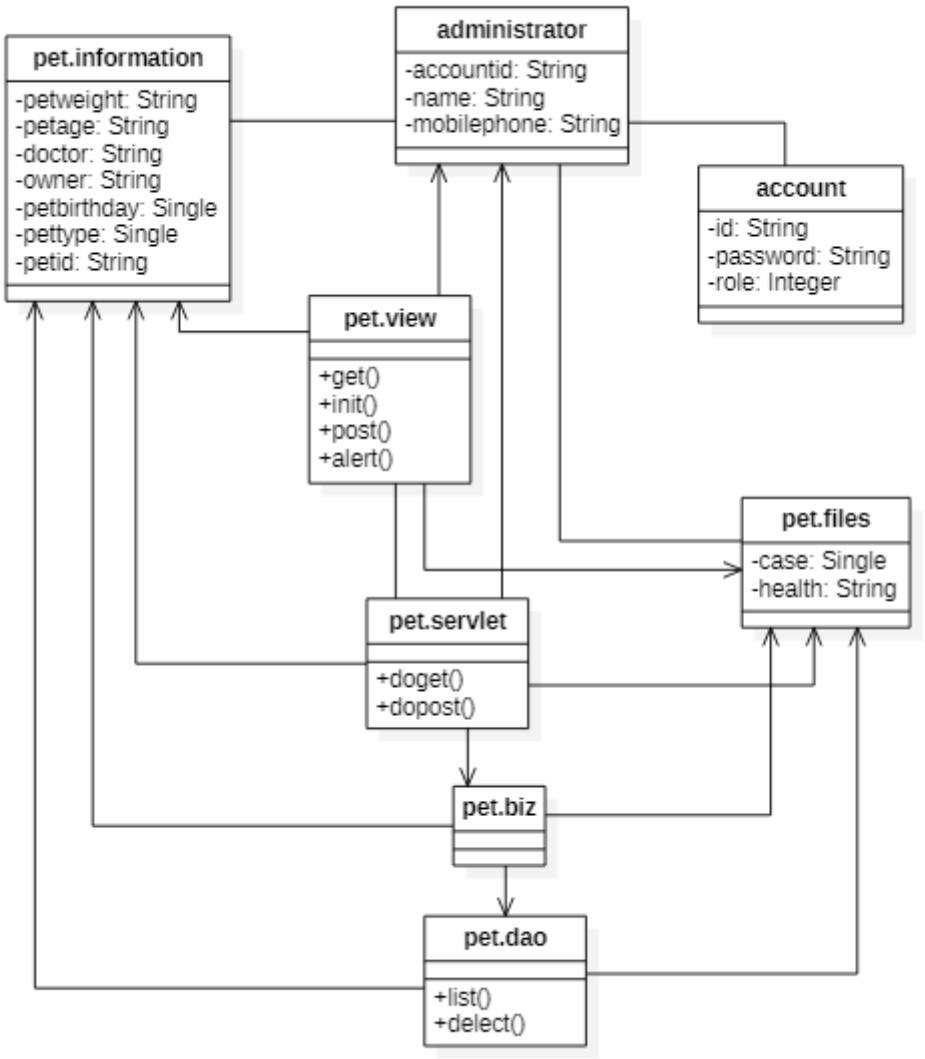
实际调用方法：UserPetController.doAdd

a. **del**

方法签名：public String del(Long id, Long type)

实际调用方法：UserPetController.del

2.4.5 宠物日志管理微服务



a. **getAllByLimit**

方法签名：public Object getAllByLimit(PetDaily pojo)

实际调用方法：UserPetDailyController.getAllByLimit

a. **getAllByLimitDoctor**

方法签名：public Object getAllByLimitDoctor(PetDaily pojo)

实际调用方法：UserPetDailyController.getAllByLimitDoctor

a. **doAdd**

方法签名：public String doAdd(PetDaily pojo)

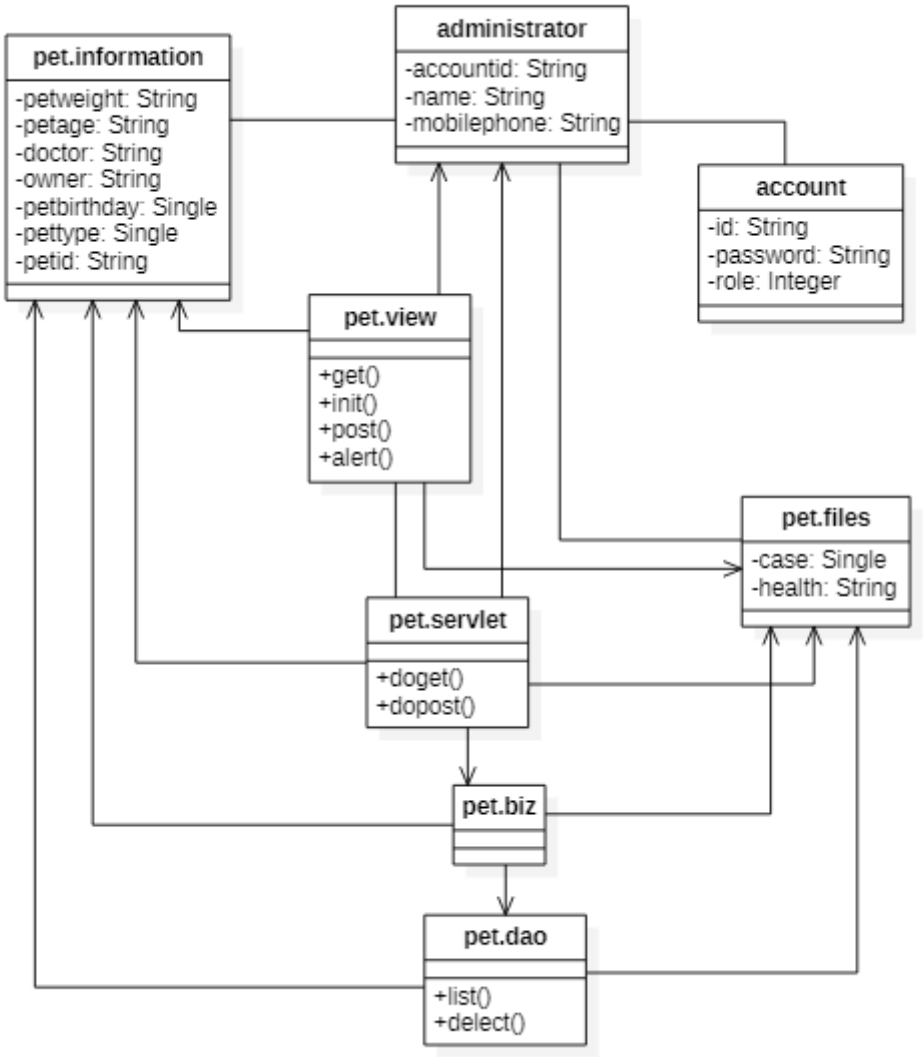
实际调用方法：UserPetDailyController.doAdd

a. **del**

方法签名：public String del(Long id)

实际调用方法：UserPetDailyController.del

2.4.6 宠物健康史管理微服务



a. **getAllByLimit**

方法签名：public Object getAllByLimit(Diagnosis diagnosis)

实际调用方法：UserDiagnosisController.getAllByLimit

a. **getAllByLimitDoctor**

方法签名：public Object getAllByLimitDoctor(Diagnosis diagnosis)

实际调用方法：UserDiagnosisController.getAllByLimitDoctor

a. **doAdd**

方法签名：public String doAdd(Diagnosis diagnosis)

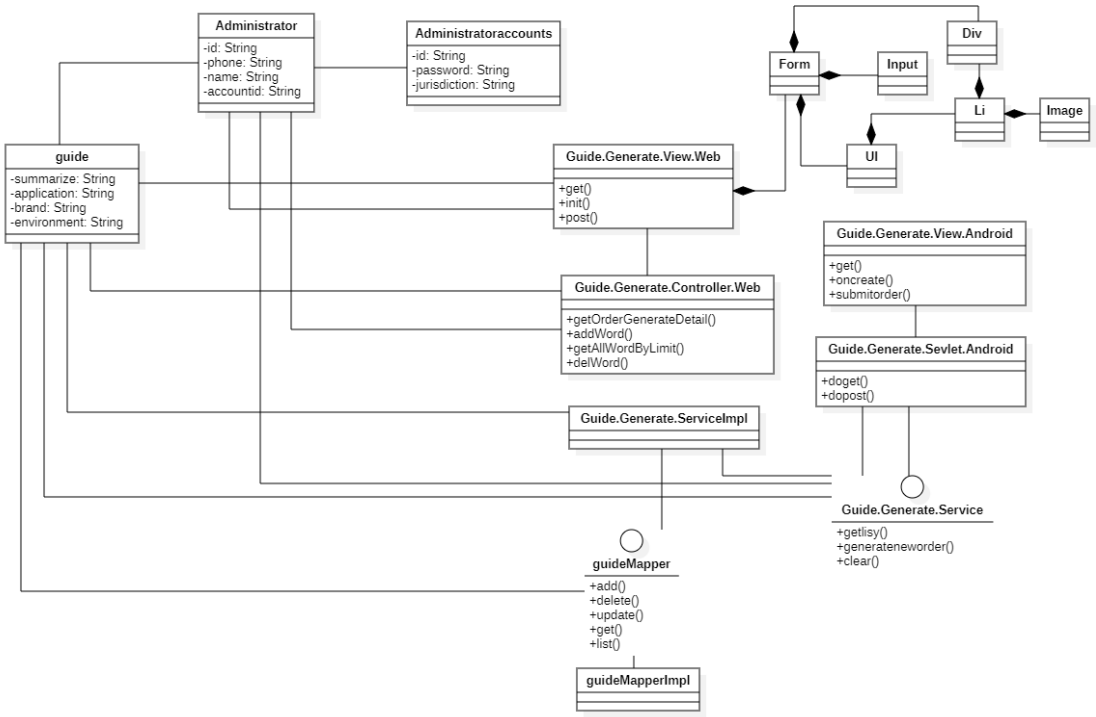
实际调用方法：UserDiagnosisController.doAdd

a. **del**

方法签名：public String del(Long id)

实际调用方法：UserDiagnosisController.del

2.4.7 健康信息微服务



a. **addWord**

方法签名：public String addWord(Notice notice)

实际调用方法：UserNoticeController.addWord

a. **getAllWordByLimit**

方法签名：public Object getAllWordByLimit(Notice word)

实际调用方法：UserNoticeController.getAllWordByLimit

a. **delWord**

方法签名：public String delWord(String[] ids)

实际调用方法：UserNoticeController.delWord

a. **getGuideGenerateDetail**

方法签名：public String getGuideGenerateDetail(Long id, Model model)

实际调用方法：UserNoticeController.getGuideGenerateDetail

2.5 界面设计

2.5.1 登录注册子系统

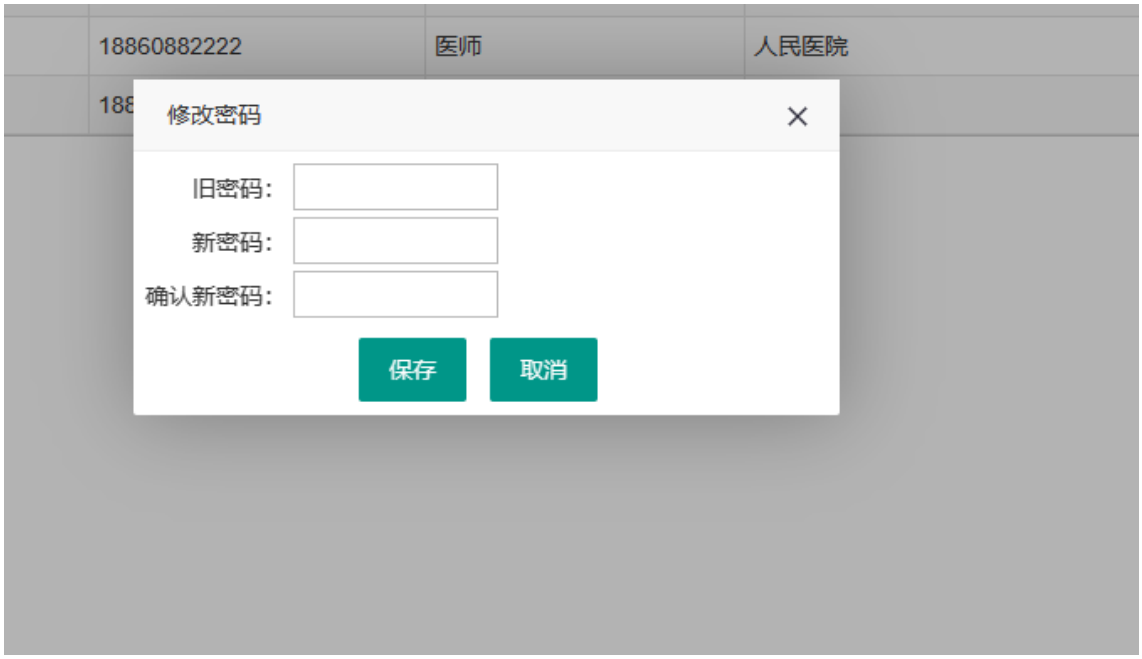
- 登录界面

输入用户名密码即可登录

- 注册界面

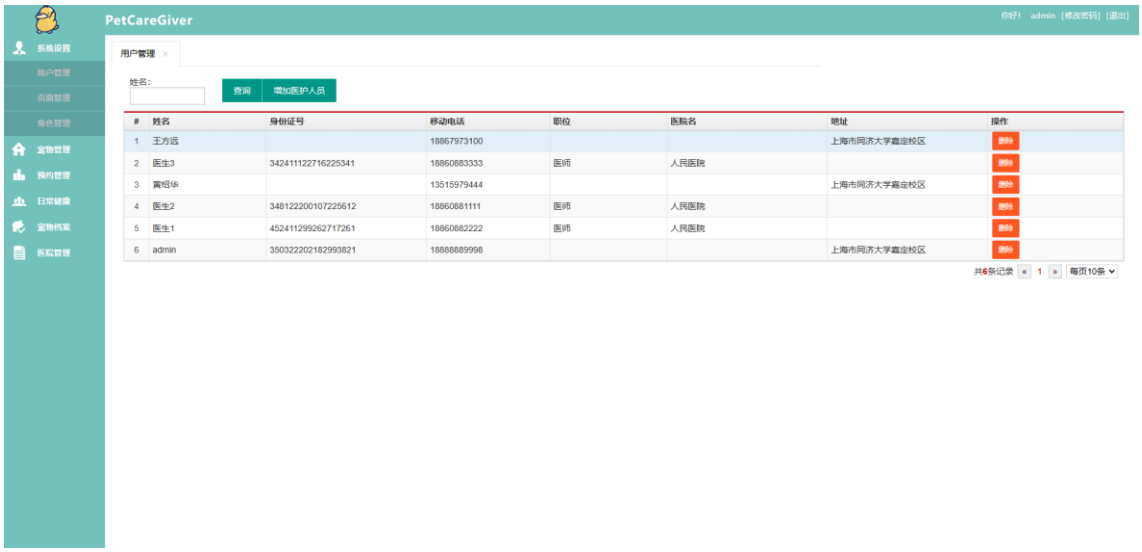
注册时，已注册的手机号不可再次注册

- 修改密码



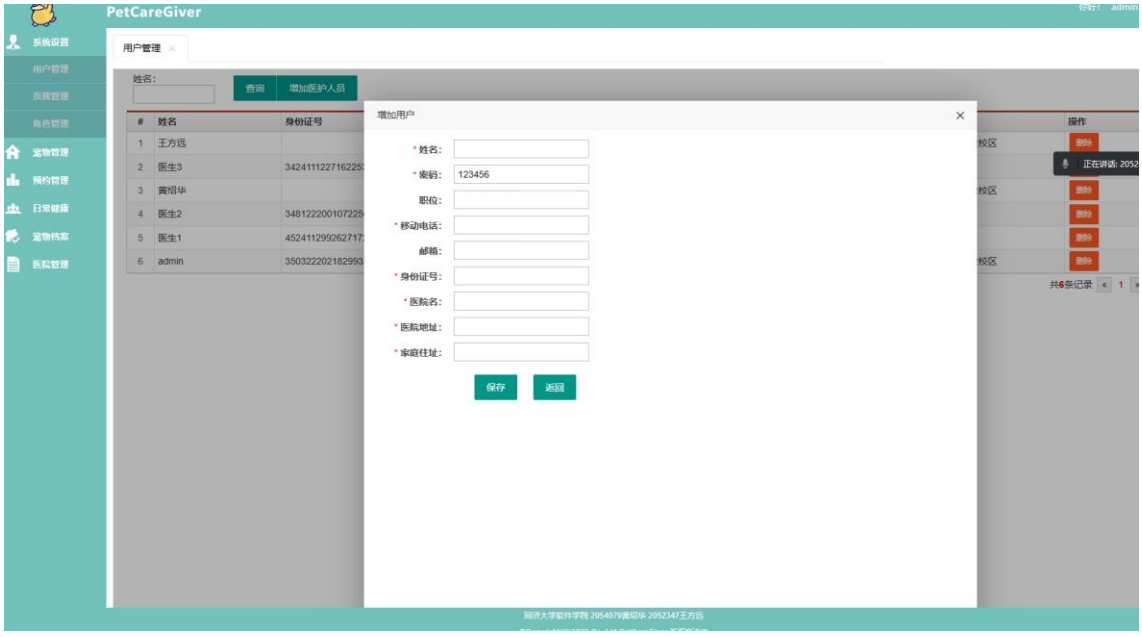
2.5.2 系统设置子系统

- 用户管理

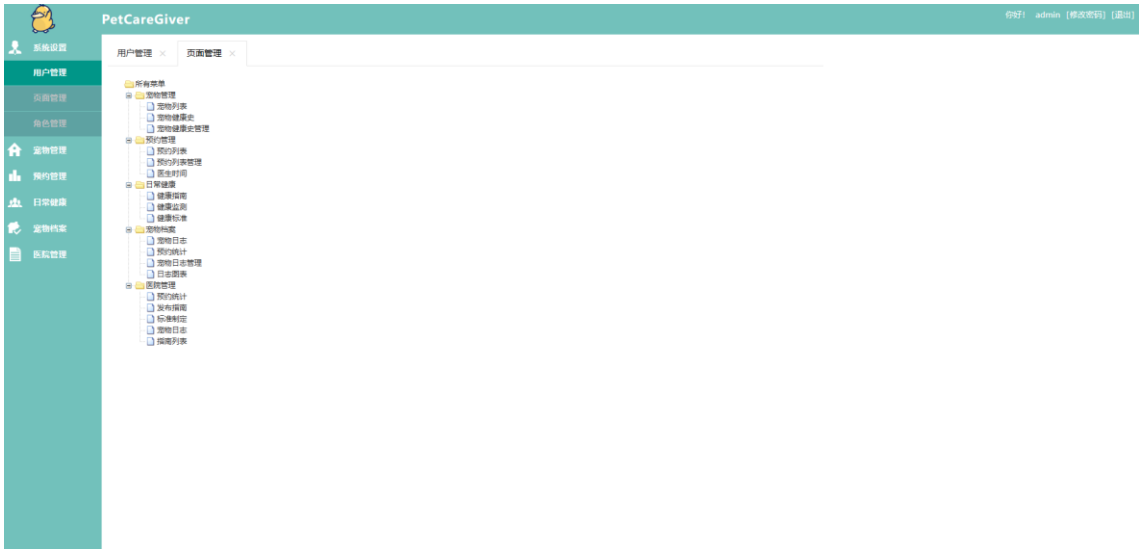


查看用户列表，可以选择删除或新增操作

- 添加医生

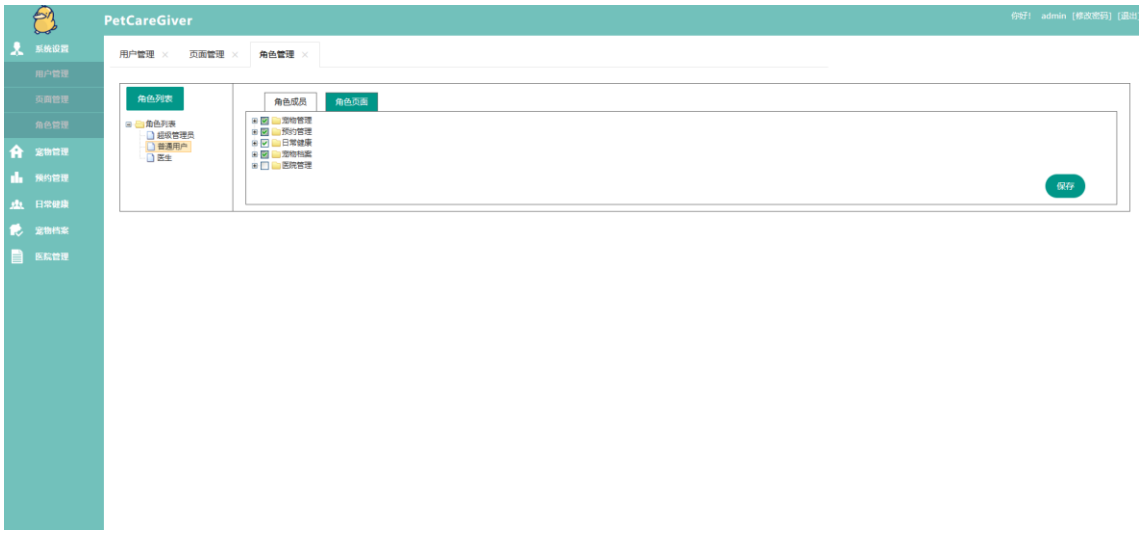


- 页面管理



查看页面文件树，可以编辑或删除

- 角色管理

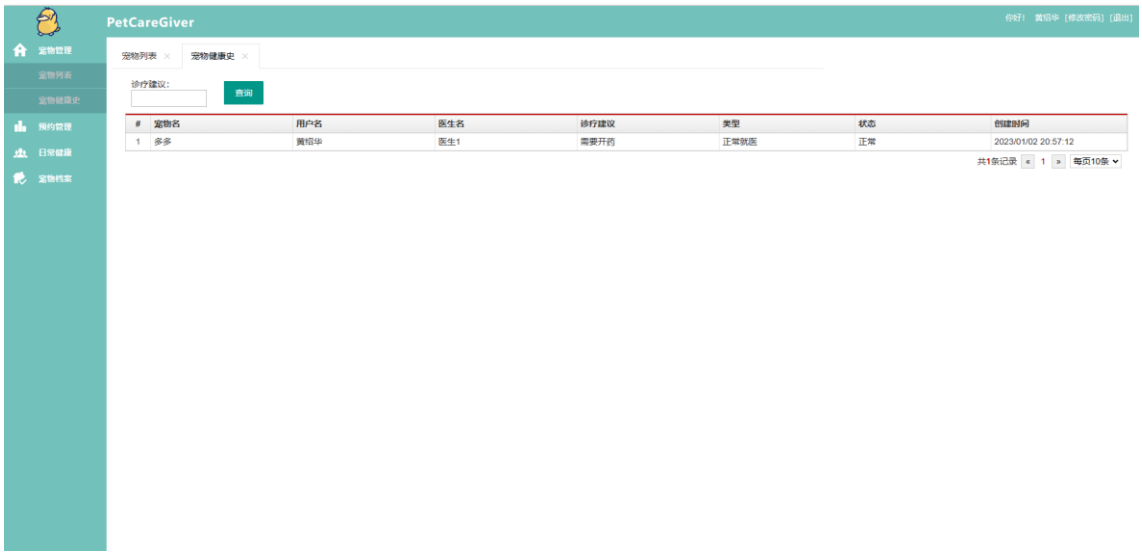


查看各个角色的页面权限，管理员可编辑

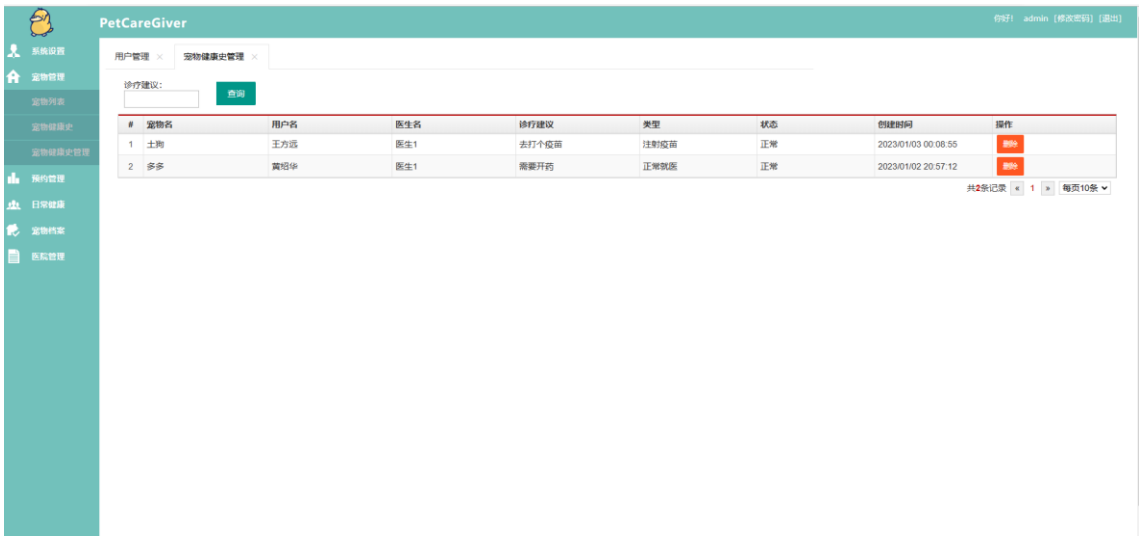
2.5.3 宠物管理子系统

- 宠物列表

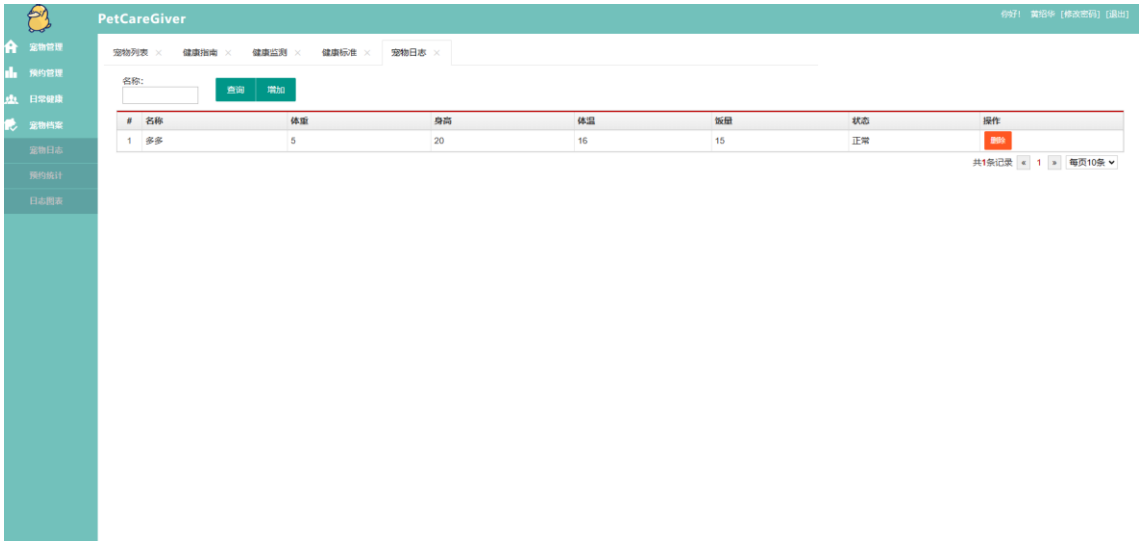
- 宠物健康史



- 宠物健康史管理



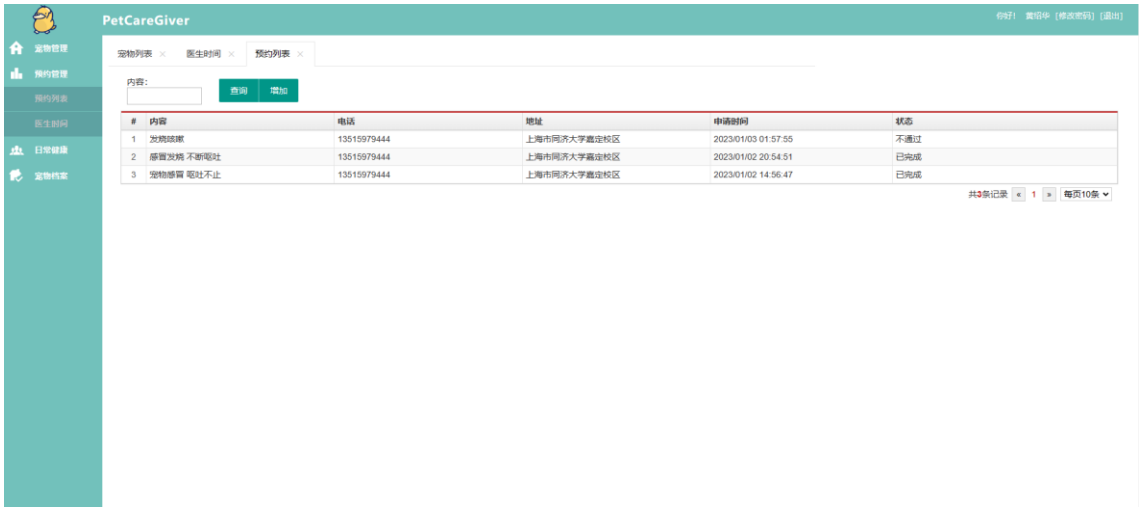
- 宠物日志



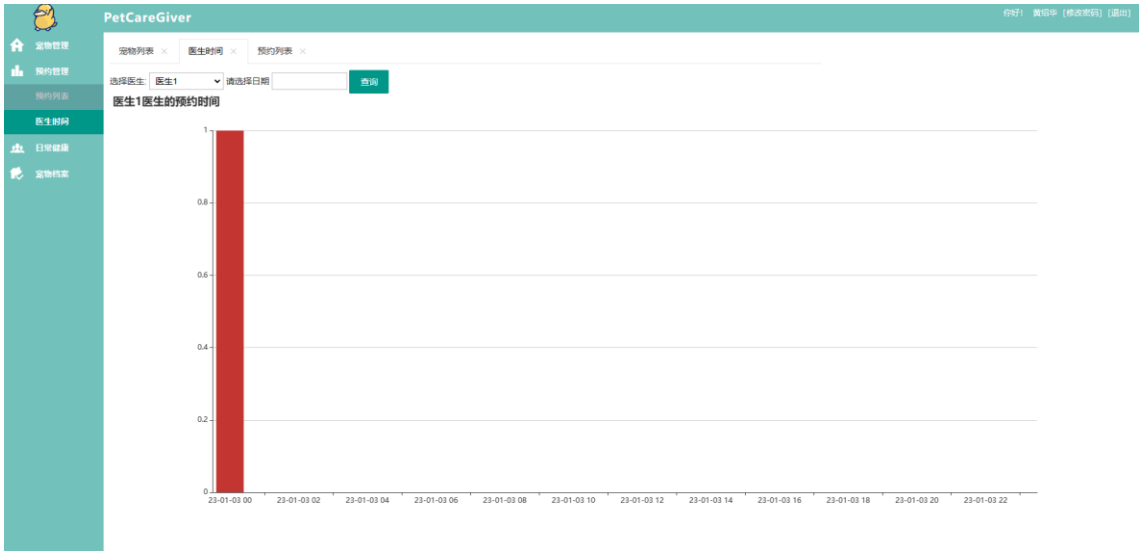
- 日志图表

2.5.4 预约管理子系统

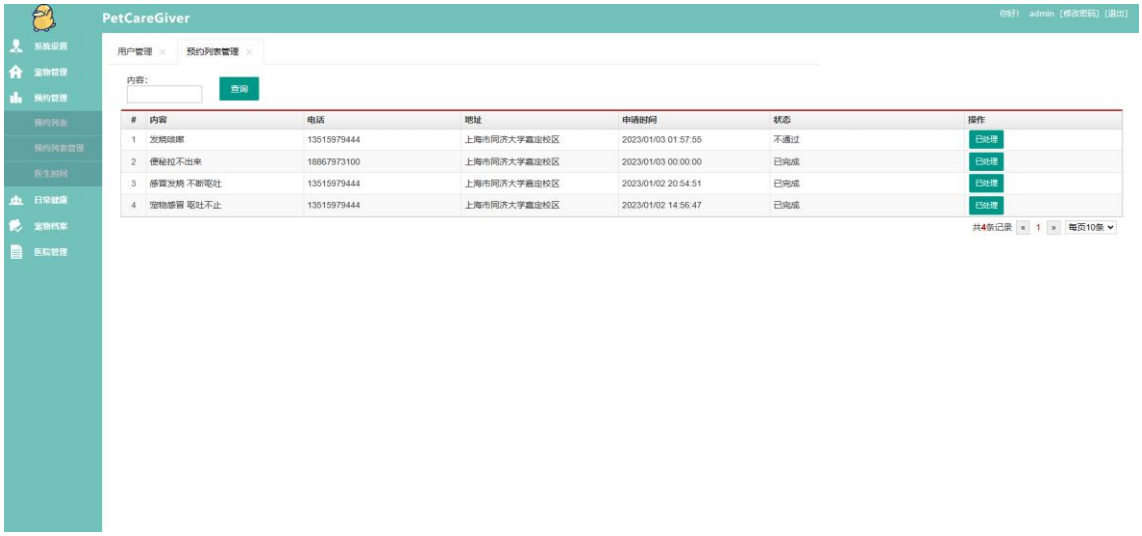
- 预约列表



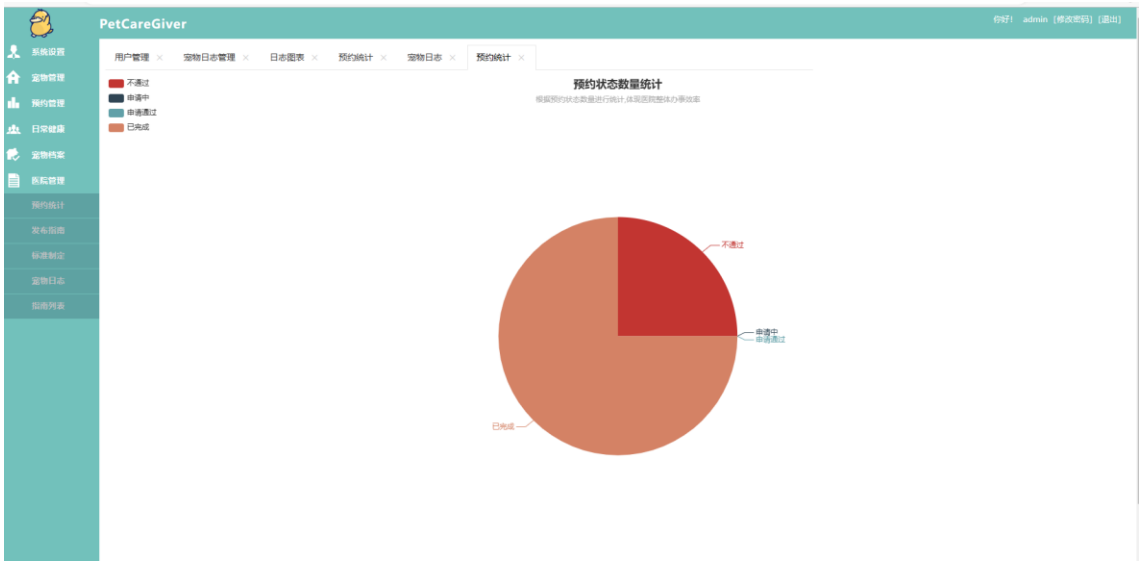
- 医生时间



- 预约列表管理

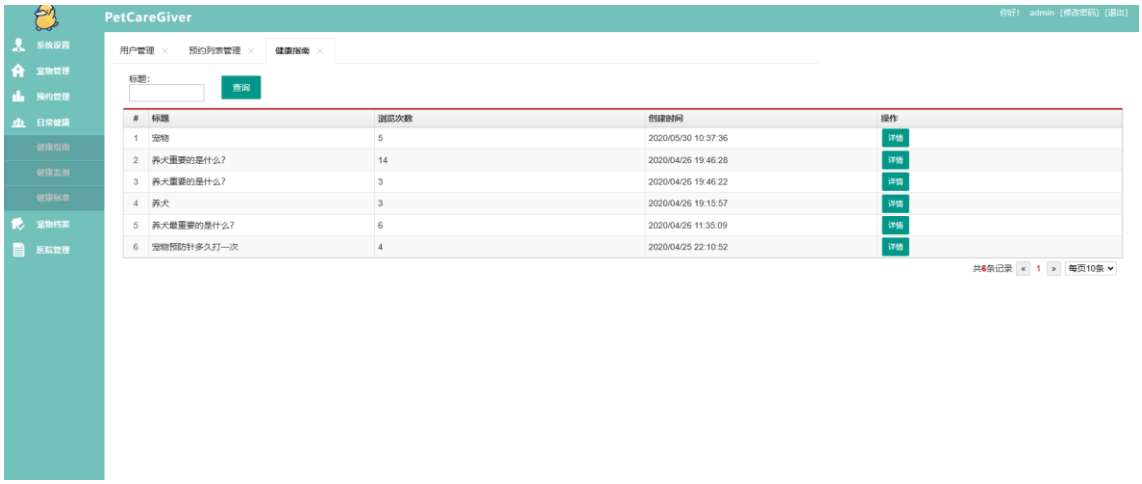


- 预约统计

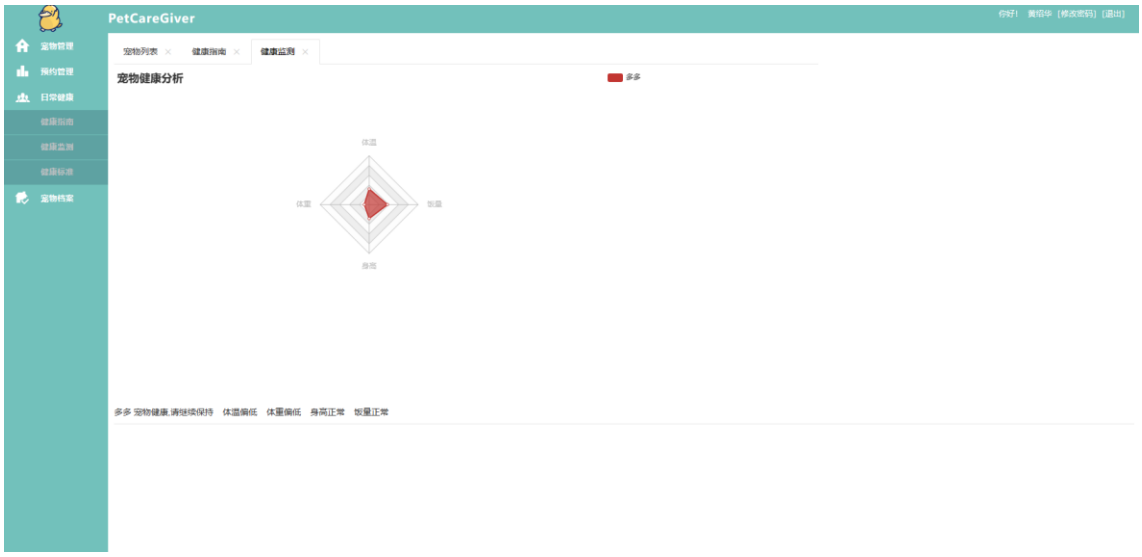


2.5.5 健康子系统

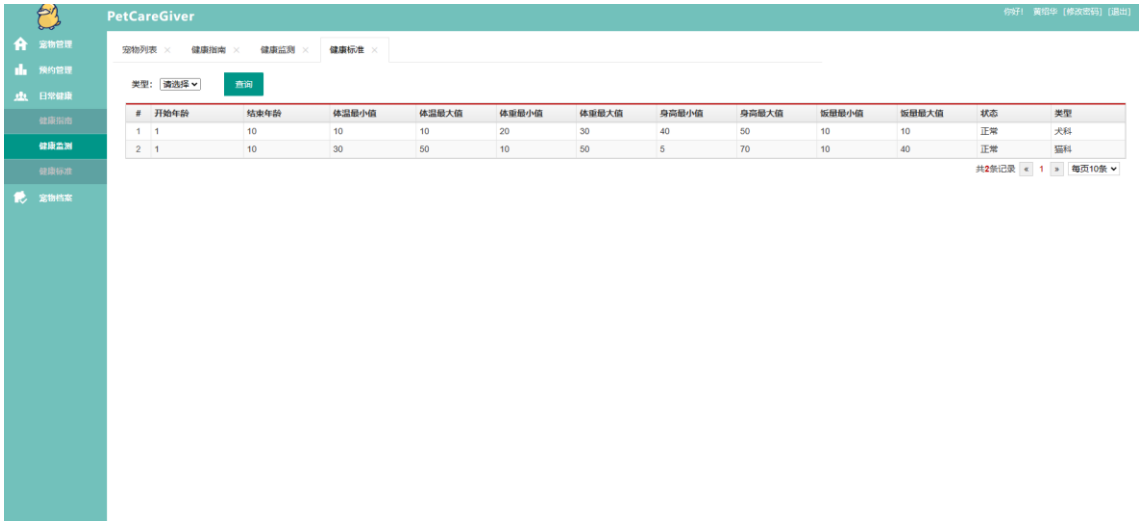
- 健康指南



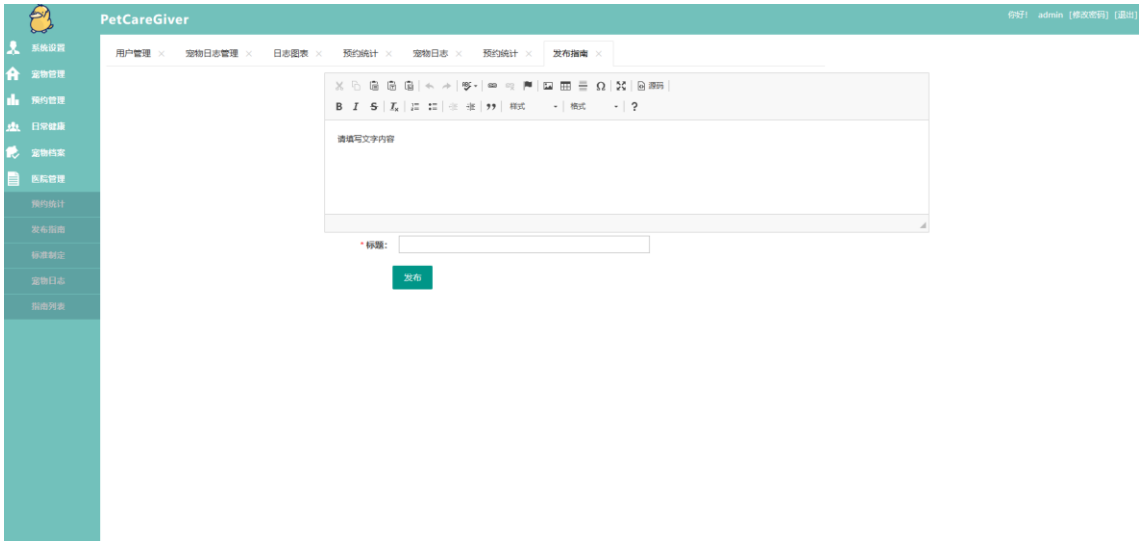
- 健康监测分析



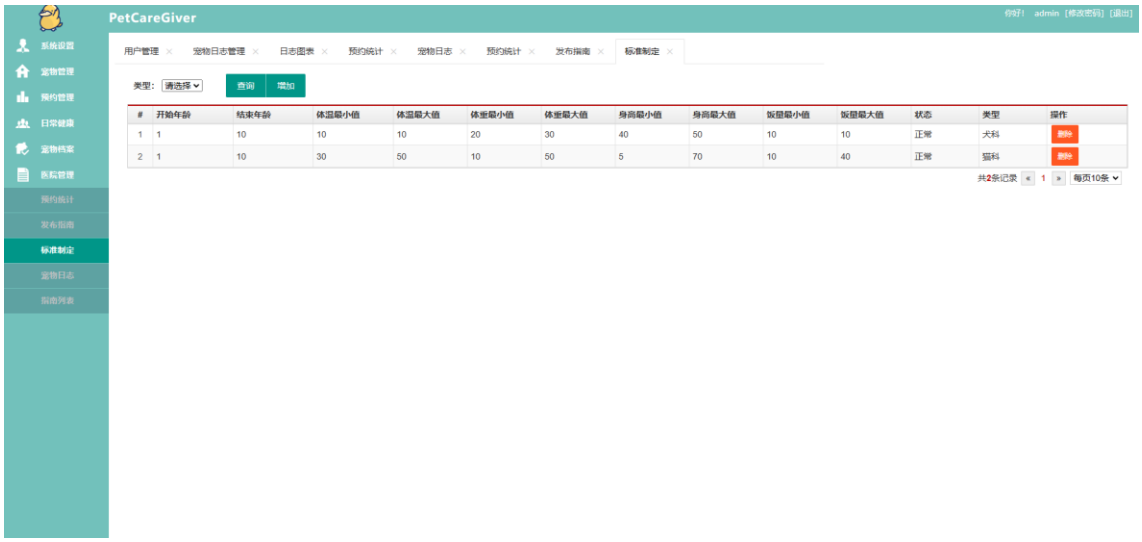
- 健康标准



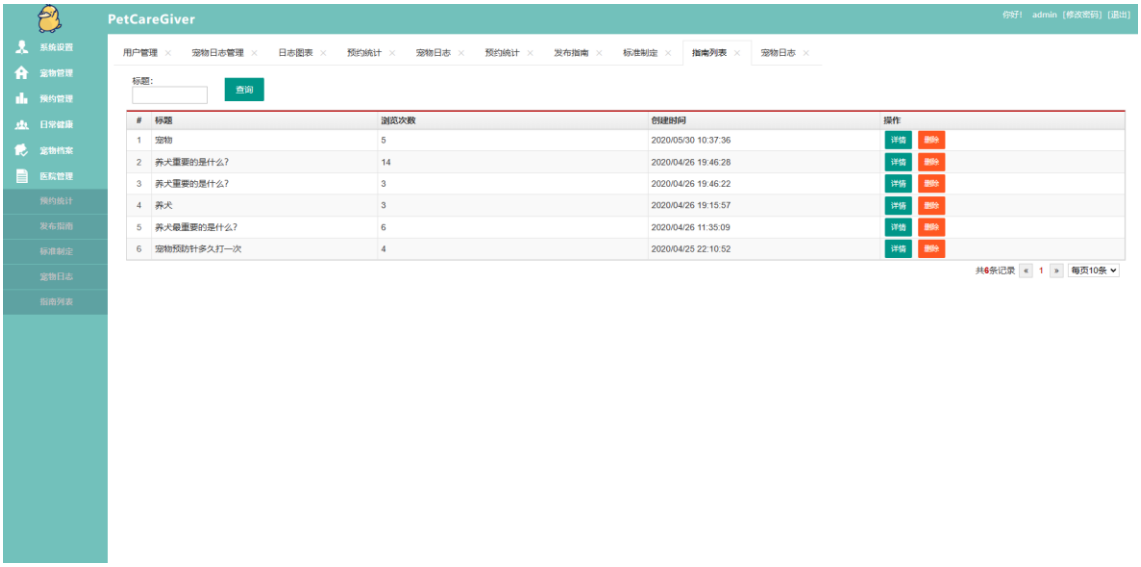
- 发布指南



- 标准制定



- 指南列表

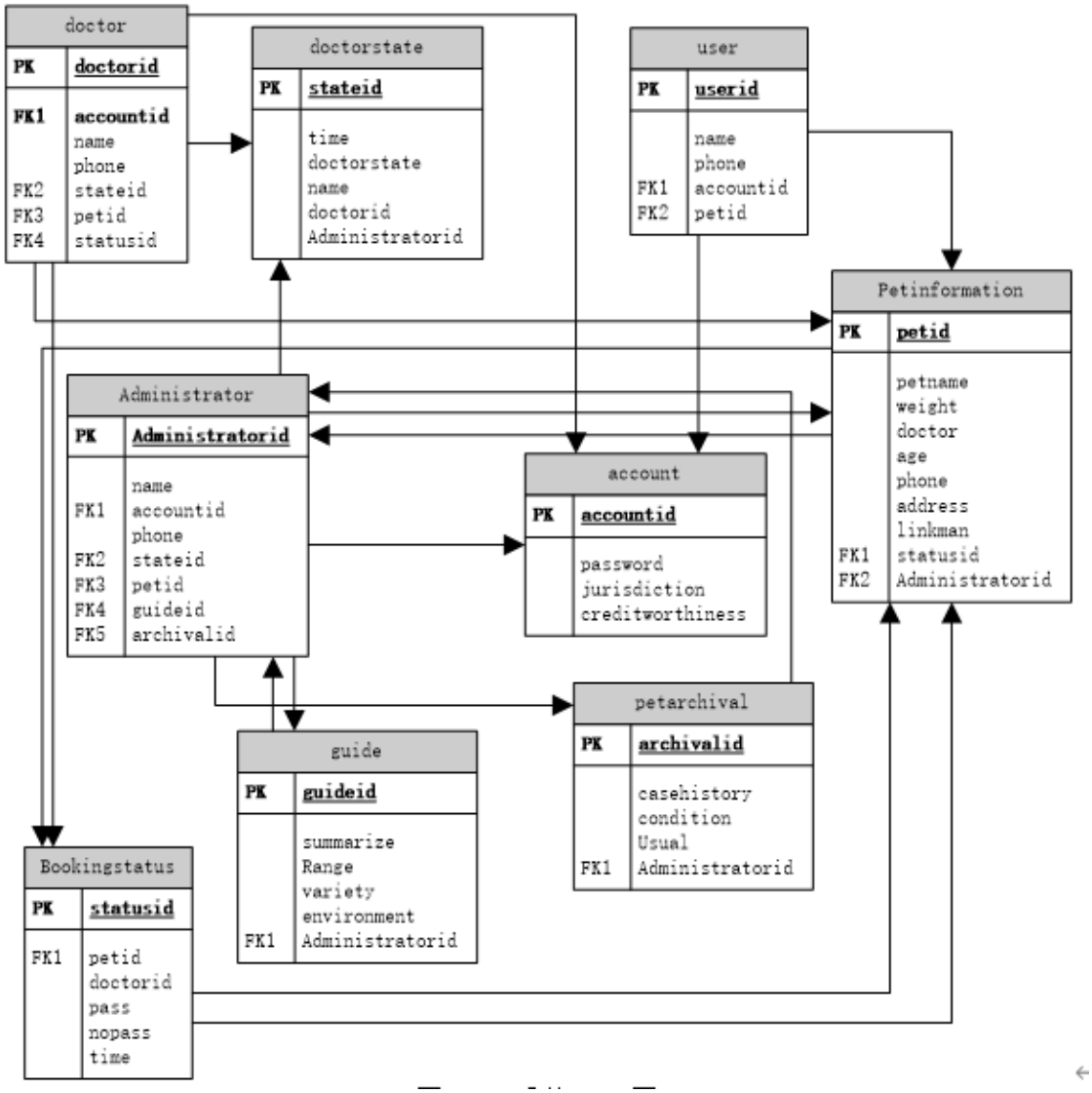


2.6 数据库设计

2.6.1 逻辑设计

在数据建模阶段，我们已经设计了每个子系统的类图和序列图，结合需求分析阶段的数据需求分析，我们详细设计了整体系统数据库的逻辑结构：首先分析系统的实体和关系，根据实体和关系构建系统整体 E-R 图。

E-R 图如图所示：



2.6.2 物理设计

1. 用户(role)数据表，角色的身份权限，见下表

role 表

序号	名称	类型	长度	是否主键	说明
1	Id	varchar	10	是	用户编号
2	Desc	Varchar	10		用户身份
3	Name	varchar	30		用户姓名

1. 角色页数(role_page)数据表，用于存储角色事务的页数，见下表

role_page 表

序号	名称	类型	长度	是否主键	说明
1	Rp_Id	Number	10	是	业务编号
2	Role_id	Number	10		角色编号
3	Page_id	Number	30		页数

1. 标准(standard)数据表，用于存储宠物的标准情况，见下表

standard 表

序号	名称	类型	长度	是否主键	说明
1	Id	int	30	是	编号
2	Age_min	Varchar	25		年龄最小
3	Age_max	Number	20		年龄最大
4	Temp_min	Number	20		温度最小
5	Temp_max	Number	20		温度最大
6	Weight_min	Number	20		体重最小
7	Weight_max	Number	20		体重最大
8	Height_min	Number	20		高度最小
9	Height_max	Number	20		高度最高
10	Appetite_min	Number	20		进食量最小
11	Appetite_max	Number	20		进食量最大

12	Type	Number	20		种类
13	Status	Number	20		状态

1. 用户信息(user)数据表，用于存储用户信息，见下表

user 表

序号	名称	类型	长度	是否主键	说明
1	Id	int	30	是	编号
2	Age	Number	20		年龄
3	Name	varchar	30		姓名
4	Password	varchar	30		密码
5	Phone	Number	20		电话
6	Adress	varchar	30		地址

1. 用户角色(user_role)数据表，用于存储用户信息，见下表

user_role 表

序号	名称	类型	长度	是否主键	说明
1	Ur_Id	Number	10	是	业务编号
2	User_id	Number	10		用户编号
3	Role_id	Number	30		角色编号

1. 业务(appointment)数据表，用于业务订单信息，见下表

appointment 表

序号	名称	类型	长度	是否主键	说明

1	Id	varchar	30	是	订单编号
2	pet_id	bigint	20		宠物编号
3	user_id	bigint	20		用户编号
4	doctor_id	bigint	20		医生编号
5	app_time	datetime	20		App 时间
6	info	varchar	255		信息
7	create_time	datetime	20		创建时间
8	status	int	20		状态
9	phone	varchar	255		电话
10	address	varchar	255		地址

1. 诊断(diagnosi)数据表，用于存储宠物诊断接种信息，见下表

diagnosi 表

序号	名称	类型	长度	是否主键	说明
1	Id	bigint	20	是	编号
2	pet_id	bigint	20		宠物编号
3	user_id	bigint	20		用户编号
4	doctor_id	bigint	20		医生编号
5	info	varchar	255		信息
6	type	int	20		类型
7	status	int	20		状态
8	create_time	datetime	20		创建时间

（8）指南(notice)数据表，用于记录科室信息，见下表

notice 表

序号	名称	类型	长度	是否主键	说明
1	Id	bigint	20	是	编号
2	content	Varchar	255		内容
3	view_count	bigint	20		计数
4	create_time	datetime	20		创建时间
5	title	varchar	255		标题

（9）页数(page)数据表，用于网页系统 url 的信息，见下表

page 表

序号	名称	类型	长度	是否主键	说明
1	page_id	int	20	是	分页编号
2	parent_id	int	20		父母编号
3	name	varchar	255		姓名
4	url	Varchar	255		链接
5	level_type	int	20		水平类型
6	level_index	int	20		水平指数
7	delete_flag	int	1		删除
8	desc	Varchar	255		描述

（10）宠物(pet)数据表，用于存储用户宠物信息，见下表

pet 表

序号	名称	类型	长度	是否主键	说明
1	Id	Varchar	110	是	编号
2	Name	Varchar	50		姓名
3	Weight	Number	11		体重
4	Height	Number	20		身高
5	Type	Number	20		种类
6	Birthday	Date	100		生日

（12）宠物信息(pet_daily)数据表，用于存储宠物身体信息，见下表

pet_daily 表

序号	名称	类型	长度	是否主键	说明
1	Id	int	30	是	编号
2	Pet_id	Varchar	25		宠物编号
3	User_id	Number	20		用户编号
4	Temp	Number	20		温度
5	Weight	Number	20		体重
6	Height	Number	20		身高
7	Appetite	Number	20		进食情况
8	Status	Number	20		状态
9	Time	Data	20		创建时间

2.6.3 设计考虑

数据库范式

由于改系统数据表较多，且业务需求设计到各张表频繁的增删改查，因此满足一定的范式，以保证不会发上插入、删除和更新的操作异常，同时降低数据的冗余。对于本系统设计的关系模式而言，满足了如下范式：

- 第一范式（1NF）。在 PetCareGiver 的关系模式中，所有的域都是原子性的，数据表的每一项都是不可分割的原子项。
- 第二范式（2NF）。PetCareGiver 数据库表中的每个实例或记录均可以被唯一地区分。
- 第三范式（3NF）。PetCareGiver 的关系中每个关系都不包含已在其他关系出现的非主关键字信息，任何非主属性不得传递依赖于主属性。

存储查询优化

随着系统发布日期的延长，数据库中的数据项将会越来越多，负载也会也会越来越大，较慢的查询效率必然会影响整体系统的响应时间。对此，我们对数据库表建立了必要的索引，以加快数据库的查询速度。对此我们制定了以下原则来建立简单索引和组合索引：

- 默认对表的外键建立索引
- 对查询中频繁出现的查询条件对应的字段建立索引
- 对于非必要的字段尽量不建立索引，以避免增加数据库的存储

除此之外，我们还对于某些复杂查询建立视图，以简化查询，方便管理。

2.7 系统出错处理设计

出错信息

本系统涉及用户的个人隐私、个人宠物的一些重要信息，因此对于一些严重的错误建立完备的出错处理机制和系统故障发生后的补救措施是十分重要的。对此我们列举了如下的常见出错信息和预计的系统处理方法。由于某些错误可以避免，因此对于特定的软错误而言，我们直接在前端予以提示和修正。对于一些可能人为造成的错误而言，如误删误改操作而言，我们提供完备的确认机制；对于一些关键信息，如密码等，系统提供相应的保密和加密措施。

以下以一览表的形式说明每种可能出现的软错误和硬错误发生时，系统输出信息

的形式，含义及处理方法。

错误类型	错误信息	处理方法	系统输出信息
TCP 连接错误	连接超时或断开	尝试重新连接，特定时间后输出错误信息	尝试失败后输出对应的网络错误信息，显示在日志中
系统部分自定义错误	用户名密码错误	不予登录	系统输出输入错误提示信息，登陆失败
	电话号码输入重复	不予注册	系统输出错误的电话号码重复信息，登陆失败
	日期信息错误	不予填入	系统前端提示日期错误
	更新宠物日志信息错误	不予更改	系统提示用户日志信息填入错误
程序错误	空指针错误	服务停止重启	系统提示内部错误并重启服务
	内部程序错误	服务重启	系统提示内部错误并重启服务
其他错误	服务器故障	无	系统提示服务器故障信息并等待人为修复重启

2.6.2 补救措施

由于错误类型不尽相同，错误的原因也各有差异，因此对于不同的错误我们采取不同的变通措施，列举如下：

- **对于软错误**，如用户输入非法信息，可能会导致后端数据库的操作错误和程序内部发生错误，系统直接在前端的输入或修改操作中对数据本身进行验证，分析错误类型，并给出相应的错误提示语句，从用户输入层面避免输入信息的非法，从而从一定程度上避免因输入非法带来的故障。

- **对于硬错误**，如数据库连接错误和网络连接错误而言，由于错误类型较少且原因明确，可在前端输出对应的提示信息，或直接将相应日志输出在前端页面中；而对于程序内部的错误，可以在程序编写阶段设置对应的异常捕获程序和抛出异常语句，在出错时输出相应的错误语句，将服务或程序重启，避免整个业务的故障停滞。维护人

员可在输出日志中查看错误信息及时修正。

- 采取适当的后备技术，如当数据库受到攻击或删库时，通过定期转储数据库，对数据库定期备份避免非法攻击带来的不可逆故障。
- 制定完备的故障恢复和重启技术，对于因程序内部错误导致的服务故障，采取服务重启并从头开始运行来保证业务逻辑的正常执行。
- 对于用户密码等信息，将提供 MD5 加密，在数据库中将密码加密存储。