在线预约挂号系统

目录

1. 开发背景 3

2. 系统描述 3

2.1 需求描述 3

2.2 功能描述 4

2.3 角色分析 4

2.4 项目预览 5

3. 系统设计 7

3.1 开发工具介绍 7

3.2 开发技术介绍 8

3.3 项目结构 9

4. 数据库设计 11

4.1 数据库分析 11

4.2 数据库概念设计 12

4.3 表结构说明 13

4.4 数据表关系 14

5. （管理端）通用模块实现 15

5.1 系统登录实现 15

5.2 系统退出实现 17

5.3 用户信息修改 17
5.4 用户密码修改 19
6. （管理端）通知管理模块 21
6.1 通知信息展示 21
6.2 通知信息添加 23
6.3 通知信息删除 24
7. （管理端）科室管理模块 25
7.1 科室信息展示 25
7.2 科室信息添加 28
7.3 科室信息修改 29
7.4 科室信息删除 30
8. （管理端）患者管理模块 31
8.1 患者信息展示 31
8.2 患者信息删除 34
9. （管理端）医师管理模块 35
9.1 医师信息展示 35
9.2 医师信息添加 39
9.3 医师信息修改 42
9.4 医师信息删除 44
10. （管理端）挂号记录管理模块 45
10.1 挂号记录信息展示 45
10.2 挂号记录删除 48

11. (客户端)挂号预约模块实现 49
11.1 医师信息浏览 49
11.2 预约记录浏览 52
11.3 普通用户注册 55
11.4 普通用户登录 58
11.5 预约记录提交 60
11.6 患者信息编辑 62

1. 开发背景

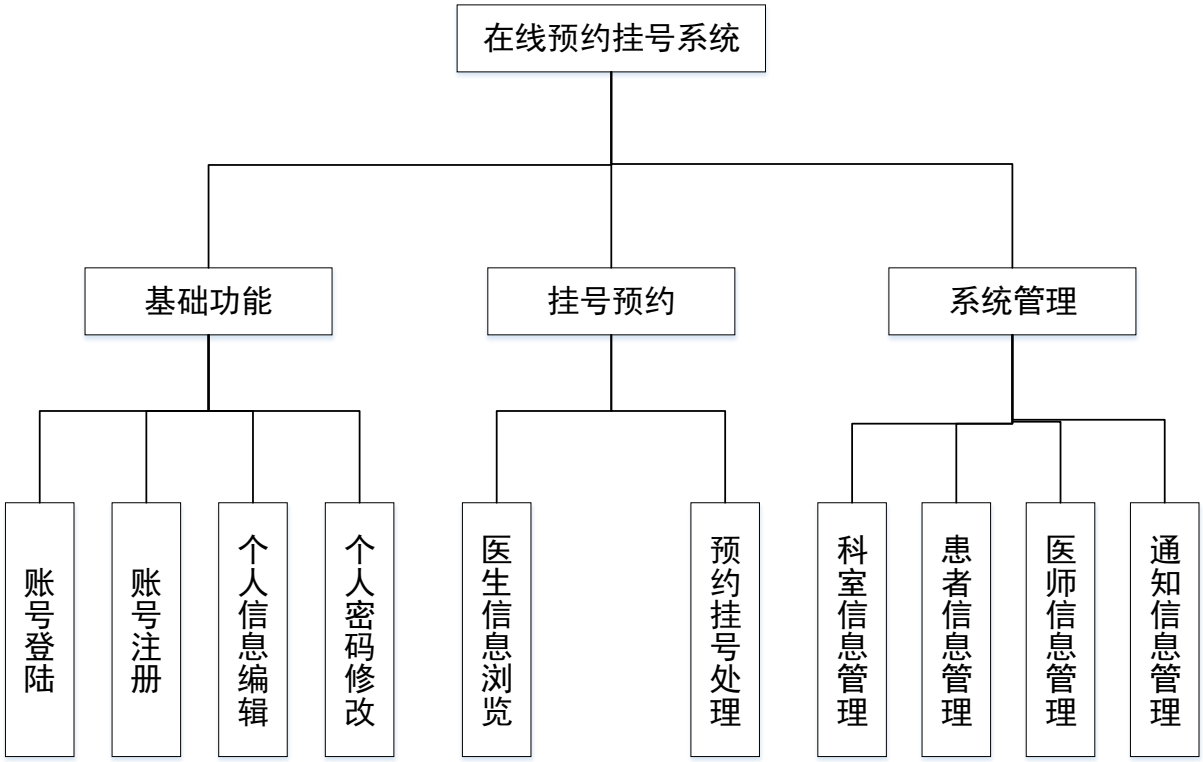
随着国民经济水平的不断提升，人们在生活质量提升的同时，对于个人健康越来越重视，越来越多的人会定期到医院进行体检，同时在生活质量提升的同时，生活压力的增大导致很多人健康出现问题，这些同样导致就医人数的增加，在这种情况下，为了提升办公效率，更好的为大众提供医疗服务，某大型综合医院决定结合当下流行的互联网开发技术开发一套在线预约挂号管理系统。

2. 系统描述

2.1 需求描述

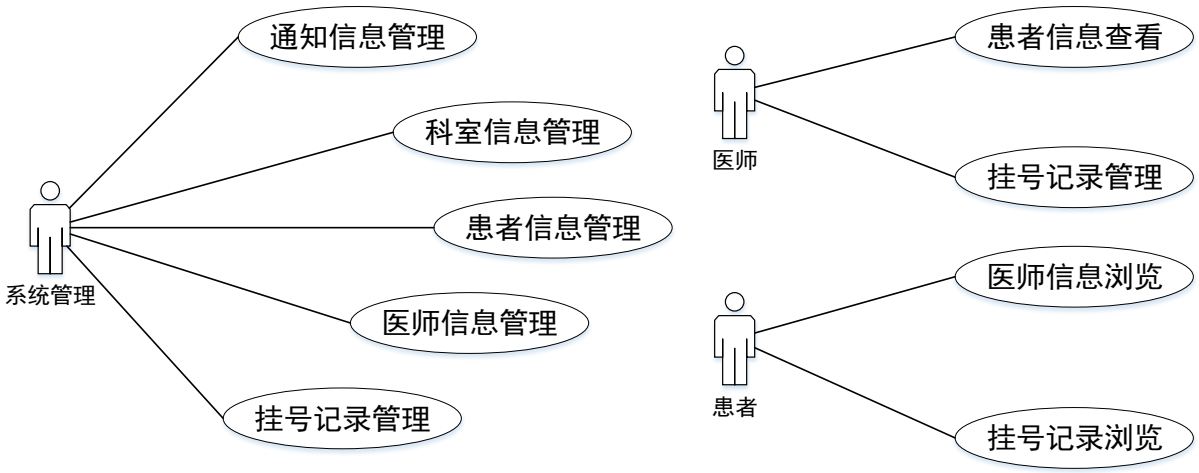
根据医院办公的实际需要以及大众网上操作的习惯，本套在线预约管理系统分为两个部分，其中一个部分是提供给医院管理人员和医师使用的管理端程序，借助这个管理程序可以很好的实现对于患者、医师、预约记录的管理，另一部分是开放的，提供给大众使用的，患者可以在线注册之后进行预约操作，同时也可以查看医院的医师信息，还可以查看自身的预约记录信息。

2.2 功能描述



如上，经过分析之后，我们最终确定本套在线预约管理系统中应该包含三方面的功能，其中基础功能模块不管是管理端程序还是客户端程序都可以使用，不同之处在于医师相关信息是由系统管理员添加的，患者信息是通过客户端程序注册的；系统管理功能是管理端程序需要实现的功能，这些功能可以满足医院日常办公和医师查询预约记录的需求；挂号预约功能是客户端程序需要实现的，在这里患者可以浏览医生的相关信息，然后进行挂号预约处理。

2.3 角色分析



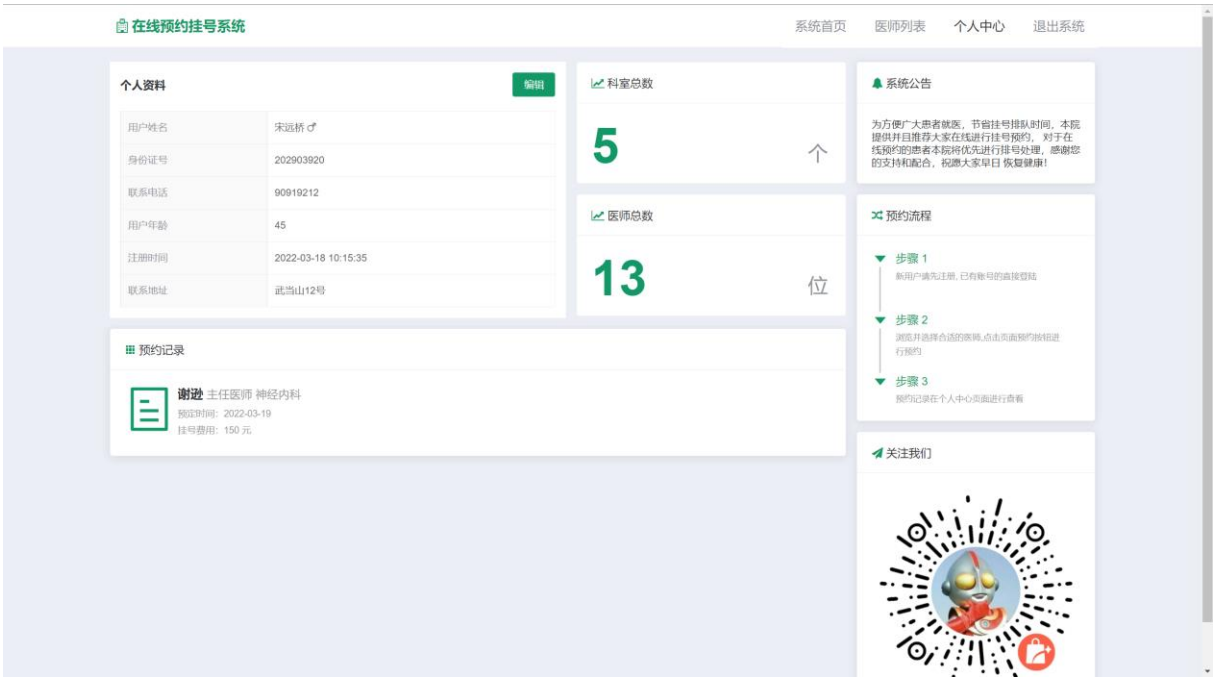
如上，经过我们的分析，系统中的用户被分为系统管理员、医师、患者三类用户，不同身份的用户可以进行的操作有所不同，在这里我们通过用例图将每个角色可以进行的操作展示出来。值得说明的是，考虑到具体实现的问题，所以我们在程序中限制系统管理员和医师仅仅可以使用管理端程序、患者仅仅可以使用客户端程序，不同身份的用户跨段调用程序并不被

允许。

2.4 项目预览

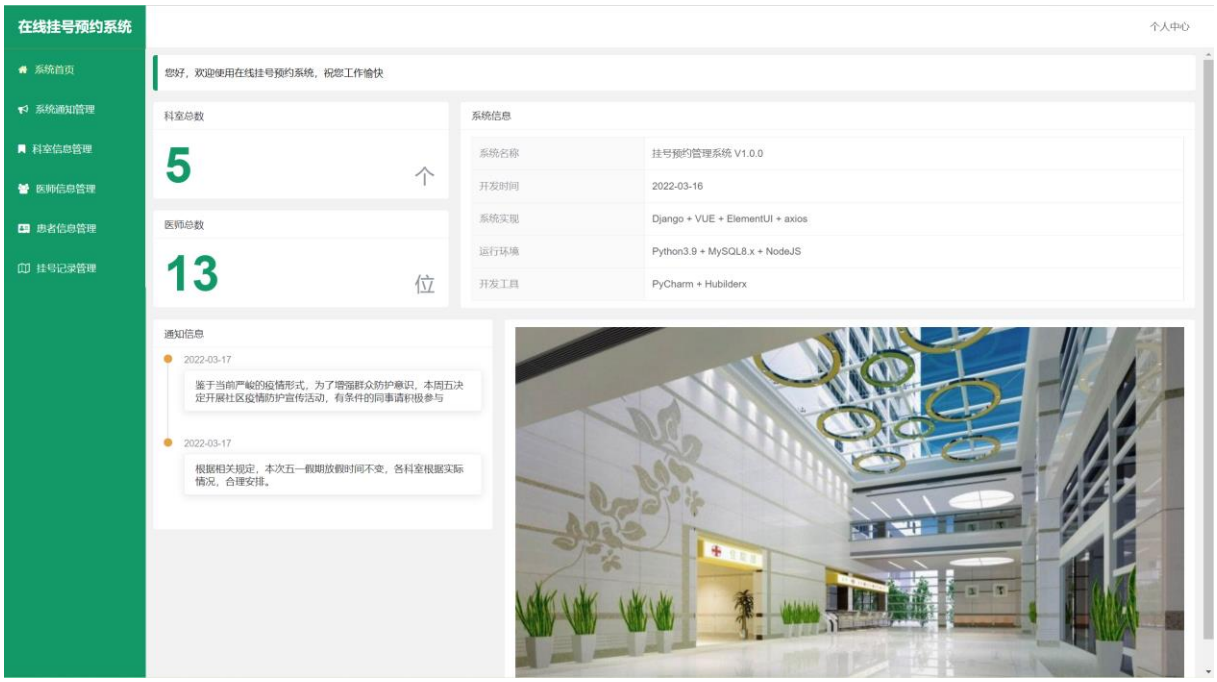
客户端首页

客户端医师列表

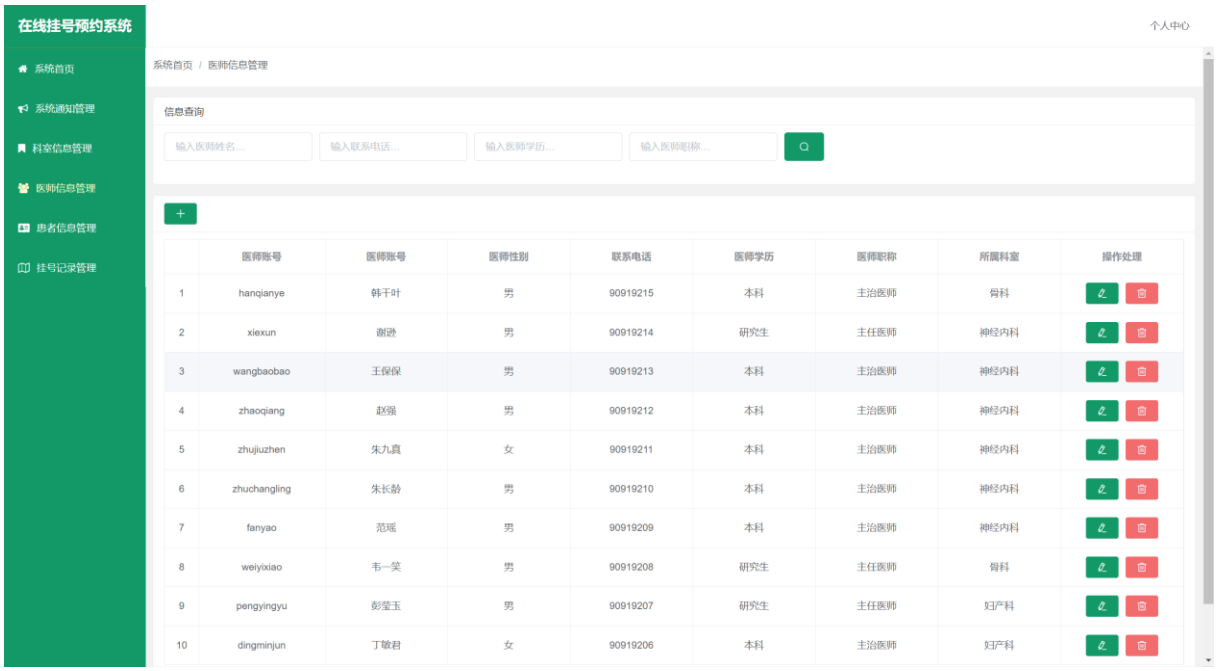


客户端个人中心

管理端登陆



管理端首页



管理端数据展示

3. 系统设计

3.1 开发工具介绍

本次在线挂号预约管理系统开发主要是在 Windows 系统下进行，采用 Python 语言开发完成，开发我们用到的工具到的工具包括 PyCharm、MySQL、Vision，下边我们对他们进行简单的介绍。

- **PyCharm**，这是一款优秀的 Python 代码编辑器，和当下流行的 IDEA 编辑器

出自同一家公司，这款编辑器不仅针对常见的代码高亮、智能补全等提供相关支持，同时对于 Django 等开发还提供相应的支持，可以大大提示开发效率

- **MySQL**，一种项目开发中常用的关系型数据库，因为体积小、开源、免费、简单易学、安装简单等特点，深受开发人员的喜爱，是大多数互联网公司的首选，我们将系统中的数据统一存储到这个数据库中，然后借助程序将这些数据读取出来，显示在页面上。
- **HBulider,** 这是一款国产的前端代码编辑器，我们可以借助完成前端代码的编辑工作，提供了代码高亮、语法提示等功能，特别是针对 Vue 创建项目方面提供了很好的支持，和 VSCode 一样，这也是一款开源呢免费且广受开发人员喜爱的一块代码编辑器
- **Vision**，Office 办公组件之一，安装之后，可以用来绘制相应的图形，帮助我们更好的理解系统的情况，建立对系统清晰的认识，是开发中最常使用的建模工具。

3.2 开发技术介绍

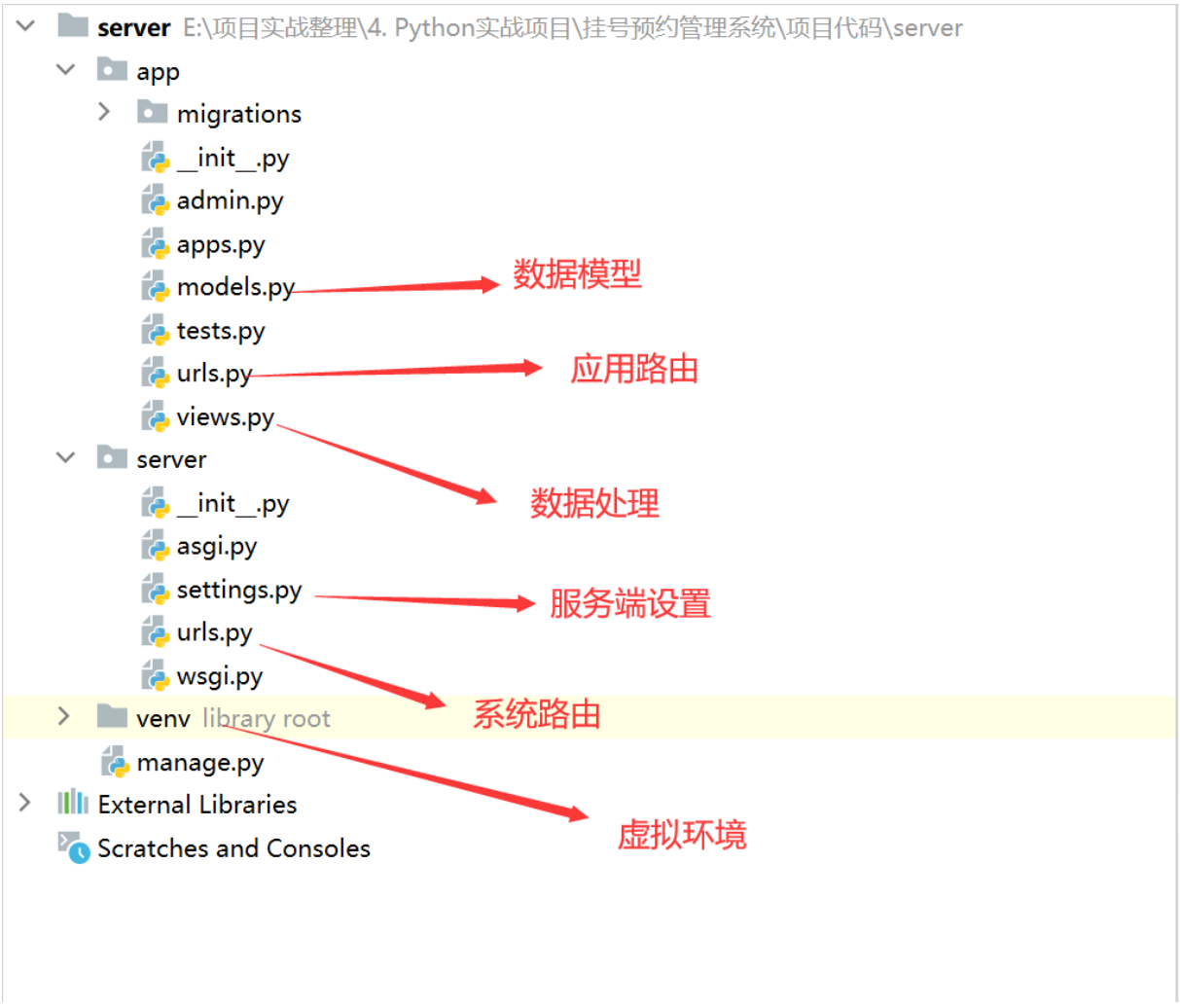
本项目采用 B/S 结构的方式，其中使用了 Django、Vue 等开发技术，这里我们对这些数据做相应的介绍。

- **Django，**Python 开发 Web 应用程序的框架，它帮助我们解决了开发中很多问题，比如模板、数据处理等等，这些使得我们只需要使用 Django 就可以完成大多数的开发任务
- **Vue2.x，**一个国产前端框架，也是当下流行的前端开发工具之一，和传统的 Java Web 开发方式不同，使用 Vue 开发，我们可以把前端作为一个单独的项目，这种方式也是当下最流行的

- **ElementUI**，一个前端可以使用 UI 框架，其中对网页中常见的元素进行了封装，比如表格、布局、按钮等等，借助这些组件可以快速构建一个好看的前端应用程序，减少了手动编写样式的麻烦

3.3 项目结构

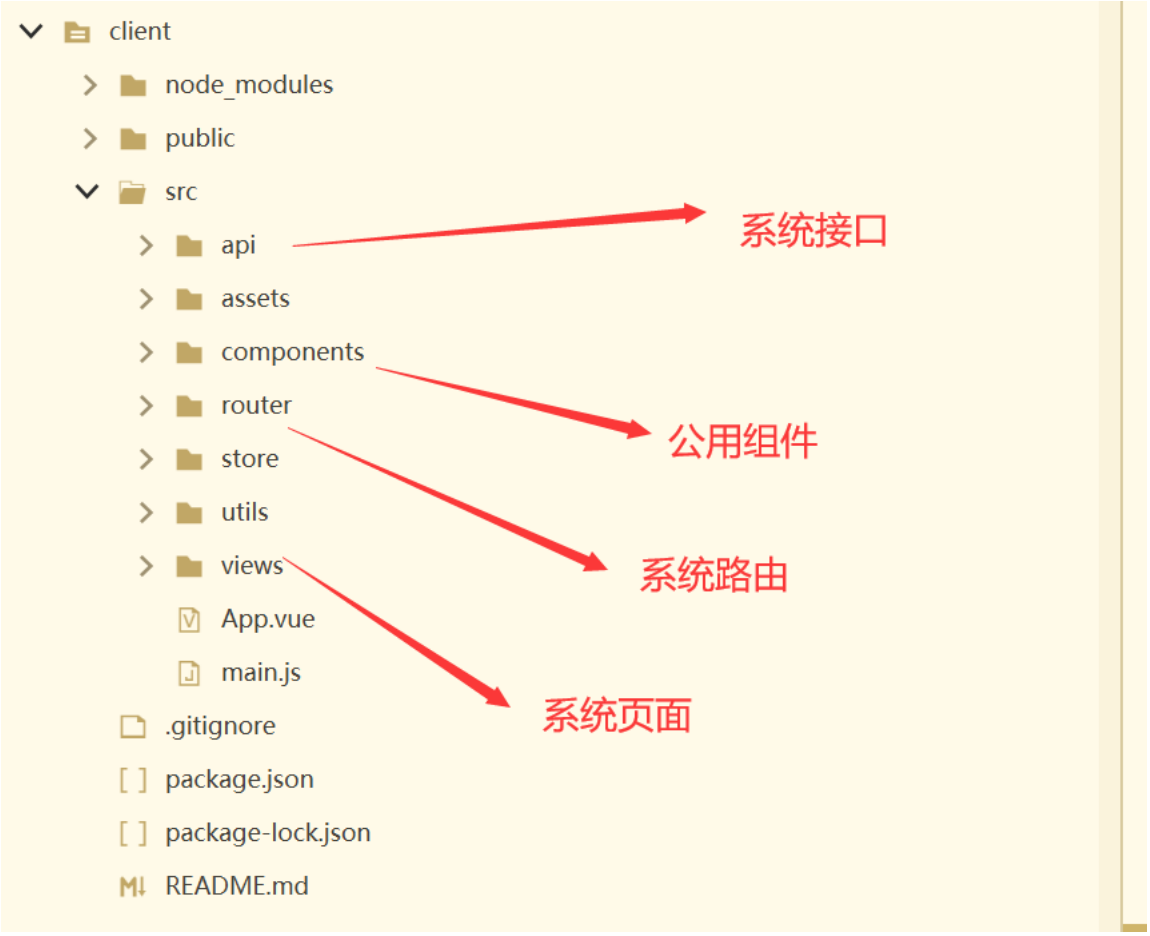
对于软件开发来说，一方面编程语言需要按照特定的结构对项目代码进行解析，另一方面，我们希望自己编写的代码是可读性良好，并且拥有很好的可维护性，要满足这些需要，对项目结构进行规划是进行开发之前必须准备的事情。



服务端代码结构


manage
node_modules
public
src
api
assets
components
router
store
utils
views
App.vue
main.js
.gitignore
package.json
package-lock.json
README.md
系统API
公用组件
路由设置
状态管理器
系统页面

管理端代码结构

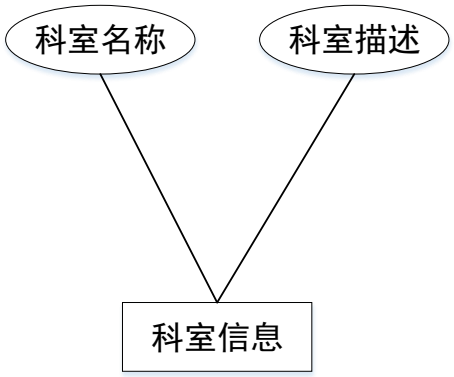


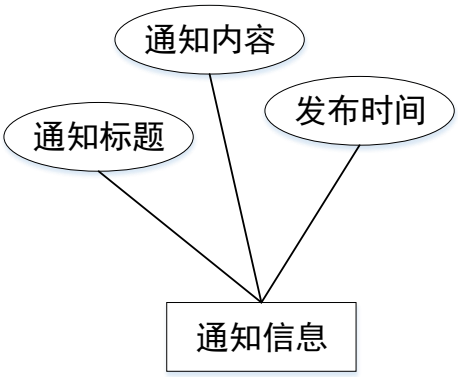
客户端代码结构

4. 数据库设计

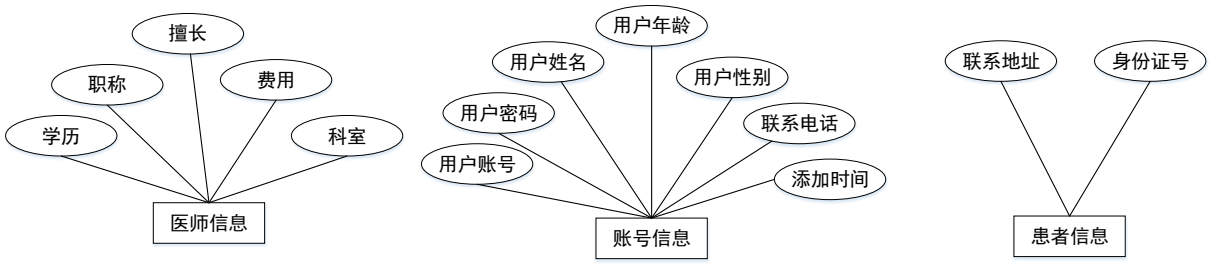
4.1 数据库分析

通过之前的分析，我们对于在线预约挂号系统有了一个比较清晰的认识，了解了系统中需要包含的功能和要求，为了使系统的数据可以正常的存取，在这里，我们对数据库的情况进行简单的分析。

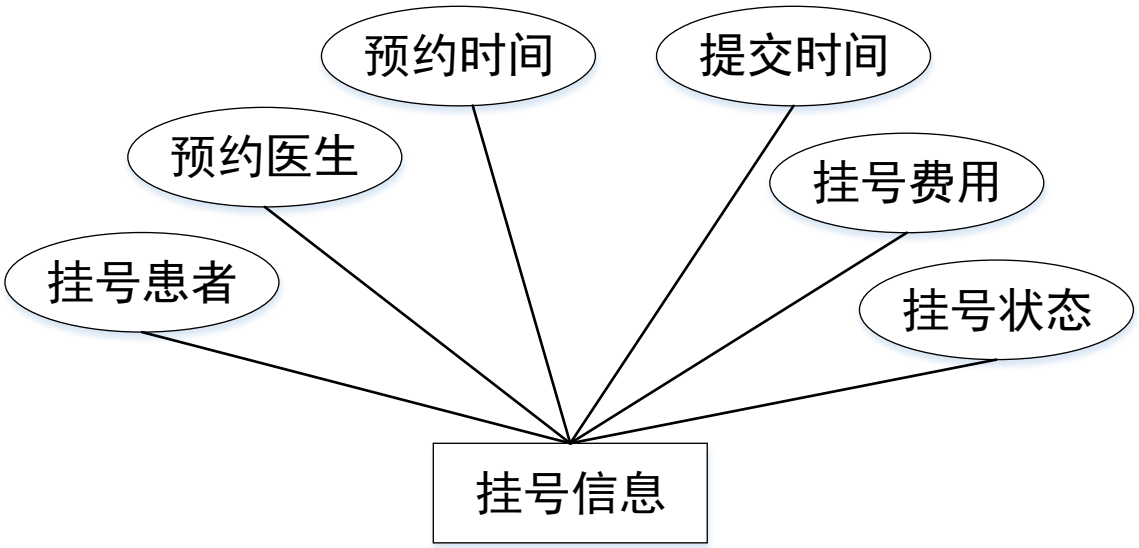




如上，这些是系统基础信息，包括系统科室信息、通知信息，具体每部分包含的在这里可以清楚的看到。

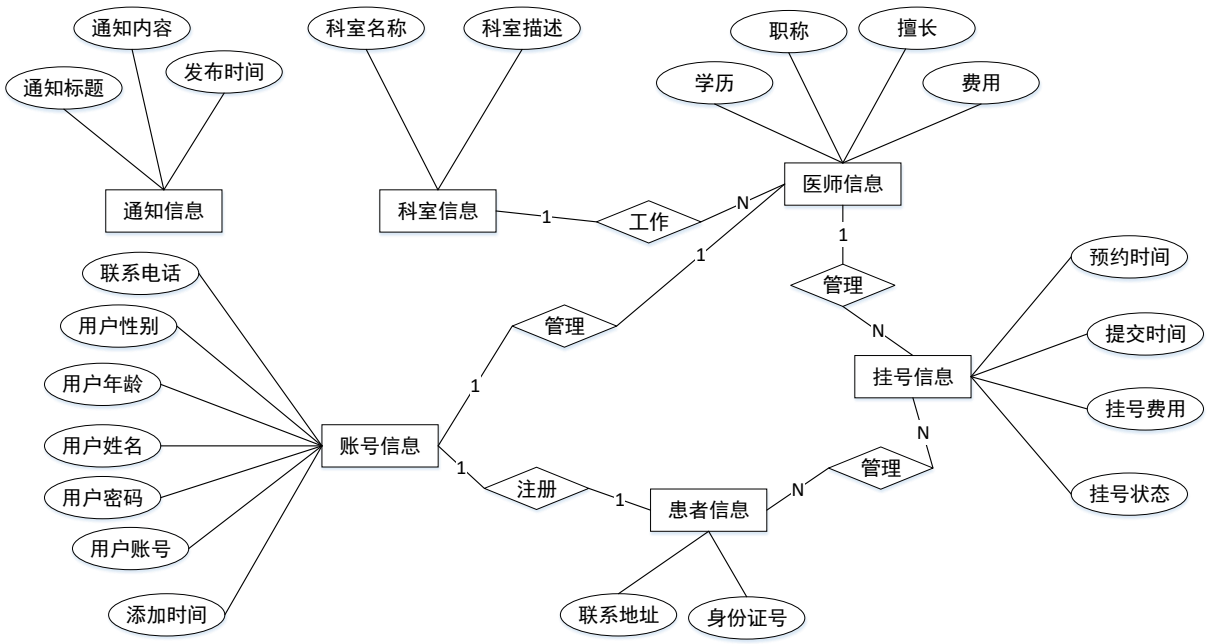


如上，这是系统中医师和患者相关的信息，我们将医师和患者共有的信息保存在账号信息表中，每个医师和患者独自的信息则保存在独立的信息表中。



如上，这是系统的挂号信息，在这里我们将记录患者挂号相关的信息方便之后查询。

4.2 数据库概念设计



这是我们使用 Vision 绘制的在线挂号预约管理系统数据模型 ER 图，通过这张 ER 图，我们可以清晰的看到系统包括了科室、通知、医师、患者等这样一些实体，同时还可以看到每个

实体包括的属性，这样我们对数据库的情况就有了一个大概的认识。

4.3 表结构说明

通过之前的分析，我们确定了数据库的基本情况，然后借助相应的 SQL 语句完成了系统的创建工作，为了方便之后的理解，这里对数据库中表的结构做相关的介绍。

（1）通知信息表，用来存储通知信息，其中包括了通知标题、通知内容等

字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号
title	varchar(32)	NO	通知标题
detail	varchar(125)	NO	通知详情
put_time	varchar(10)	NO	发布时间

（2）科室信息表，用来存储科室信息，其中包括了科室名称、科室描述等

字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号
name	varchar(32)	NO	科室名称
detail	varchar(125)	NO	科室详情

（3）账号信息表，用来存账号信息，其中包括了账号、密码、姓名等

字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号
user_name	varchar(32)	NO	用户账号
pass_word	varchar(32)	NO	用户密码
name	varchar(20)	NO	用户姓名
gender	varchar(2)	NO	用户性别
age	int	NO	用户年龄
phone	varchar(11)	NO	联系电话
create_time	varchar(19)	NO	添加时间
type	int	NO	用户身份

（4）医师信息表，用来存医师信息，其中包括了账号、密码、姓名等

字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号
record	varchar(32)	NO	医师学历
job	varchar(20)	NO	医师职称
good	varchar(125)	NO	专长说明
total	double	NO	挂号费用
office_id	int	NO	所属科室

（5）患者信息表，用来存患者信息，其中包括了联系地址等

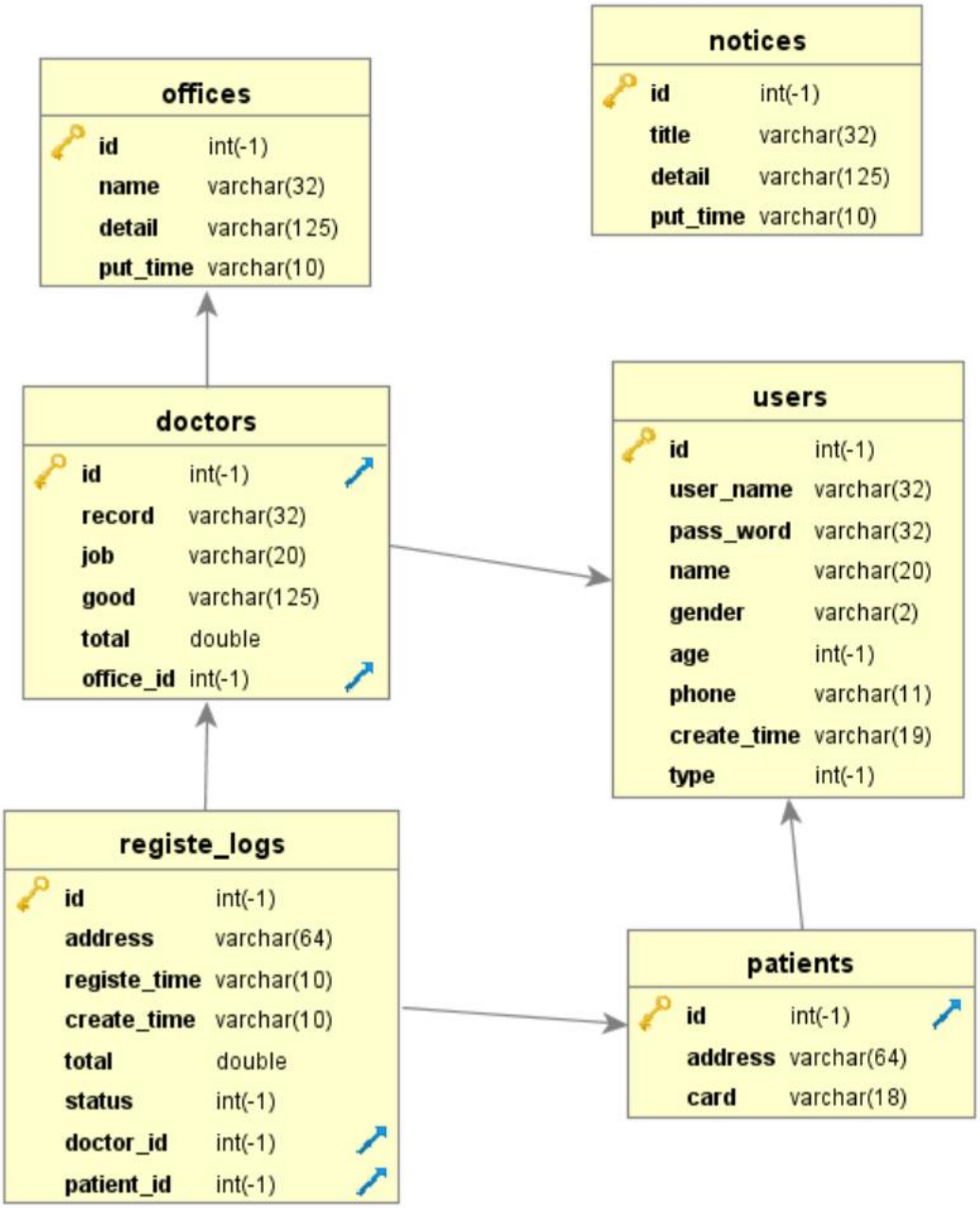
字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号
address	varchar(64)	NO	联系地址
card	varchar(18)	NO	身份证号

（6）预约记录表，用来存预约记录信息，其中包括了联系地址等

字段	数据类型	允许为 NULL	备注说明
id	int	NO	记录编号

registe_time	varchar(10)	NO	预约时间
create_time	varchar(10)	NO	提交时间
total	double	NO	挂号费用
status	int	NO	挂号状态
doctor_id	int	NO	预约医师
patient_id	int	NO	挂号患者

4.4 数据表关系



为了方便了解数据库的具体情况，我们在这里绘制了一张数据库关系展示图，在这张图中详细展示了每个数据表以及它所包含的内容，同时把数据表之间的关系详细的罗列出来，对于了解数据表有极好的意义。

5. （管理端）通用模块实现

5.1 系统登录实现

如上所示，这是我们在线挂号预约管理系统系统登录的页面，当用户访问指定的登录地址时，可以看到这样一个页面，在这里正确的输入用户的登录凭证信息，经过系统判断之后，决定用户是否可以进入到系统中。

```
<div class="login-container">
    <div class="login-win">
        <div class="login-body">
            <div class="login-title">
                挂号预约系统
            </div>
            <div class="login-form">
                <el-form :model="loginForm" :rules="rules" ref="loginForm">
                    <el-form-item style="margin-bottom:30px" prop="userName">
                        <el-input type="text" v-model="loginForm.userName" suffix-
icon="el-icon-user-solid"
                            placeholder="请输入您的账号"></el-input>
                    </el-form-item>
                    <el-form-item prop="passWord">
                        <el-input  type="password"  v-model="loginForm.passWord"
suffix-icon="el-icon-lock"
                            placeholder="请输入您的密码"></el-input>
                    </el-form-item>
                    <el-form-item>
                        <el-button  style="margin-top:  25px;  width:
```

```
100%;background-color: #009966;"
                                    @click="submitForm('loginForm')"  type="primary">用户登录</el-button>
                            </el-form-item>
                        </el-form>
                    </div>
                </div>
            </div>
```

这是我们登录页面具体的 HTML 代码，在这里主要是提供了一个表单，需要用户输入自己的用户名和密码，点击其中的登录按钮之后，就可以向服务器发送一个登录请求了。

```
def login(request):

    userName = request.POST.get('userName')
    passWord = request.POST.get('passWord')
    flag = int(request.POST.get('flag'))

    user = models.Users.objects.filter(userName=userName)

    if (user.exists()):

        user = user.first()

        if (flag == 0) & (user.type == 2) :

            return SysView.warn('普通用户无权使用管理功能')

        elif (flag == 1) & (user.type != 2):
            return SysView.warn('管理员或教师无法登陆')

        else:
            if user.passWord == passWord:

                token = uuid.uuid4()

                resl = {
                    'token': str(token)
                }

                cache.set(token, user.id, 60*60*60*3)

                return SysView.successData(resl)
            else:
                return SysView.error('用户密码输入错误')
```

```
    else:
        return SysView.error('用户名输入错误')
```

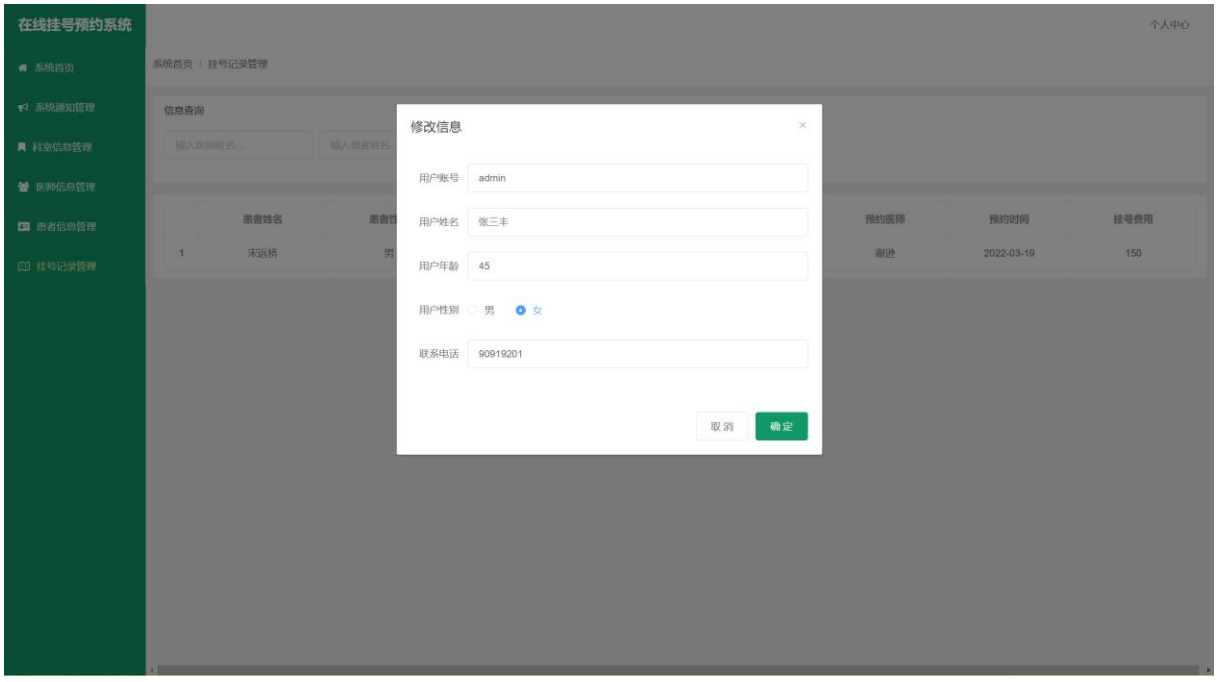
当用户点击登录按钮将填写的用户名和密码信息发送到服务器之后，在服务端会经过上述这段代码的处理，首先我们通过输入的用户名进行查找用户信息，然后再通过查找到的用户进行比对密码，中间任何一个步骤出现异常，都不会正常登录。

5.2 系统退出实现

```
def exit(request):

    token = request.GET.get('token')

    cache.delete(token)

    return BaseView.success()
```

在系统中提供了供用户退出登录状态的功能，当用户点击退出登录按钮时，会发送一个退出登录的请求到服务端，服务端进行清除登录信息的操作并且跳转到登录页面，这样就完成了退出登录状态的功能。

5.3 用户信息修改



如上所示，系统提供了供当前用户修改个人信息的功能，在这里可以将当前用户的信息呈现到表单中，方便对信息进行修改

```
<el-dialog title="修改信息" width="600px" :append-to-body="true" :visible.sync="showUpdInfoFlag">
    <el-form label-width="80px" :model="userForm">
        <el-form-item label="用户账号">
            <el-input v-model="userForm.userName" placeholder="请输入用户账号…" autocomplete="off"></el-input>
```

```
            </el-form-item>
            <el-form-item label="用户姓名">
                <el-input v-model="userForm.name" placeholder="请输入用户姓名…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item label="用户年龄">
                <el-input v-model="userForm.age" placeholder="请输入用户年龄…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item label="用户性别">
                <el-radio-group v-model="userForm.gender">
                    <el-radio label="男"></el-radio>
                    <el-radio label="女"></el-radio>
                </el-radio-group>
            </el-form-item>
            <el-form-item label="联系电话">
                <el-input v-model="userForm.phone" placeholder="请输入联系电话…"
autocomplete="off"></el-input>
            </el-form-item>
        </el-form>
        <div slot="footer" class="dialog-footer">
            <el-button @click="showUpdInfoFlag = false">取 消</el-button>
            <el-button type="primary" @click="updInfo()">确 定</el-button>
        </div>
</el-dialog>
```

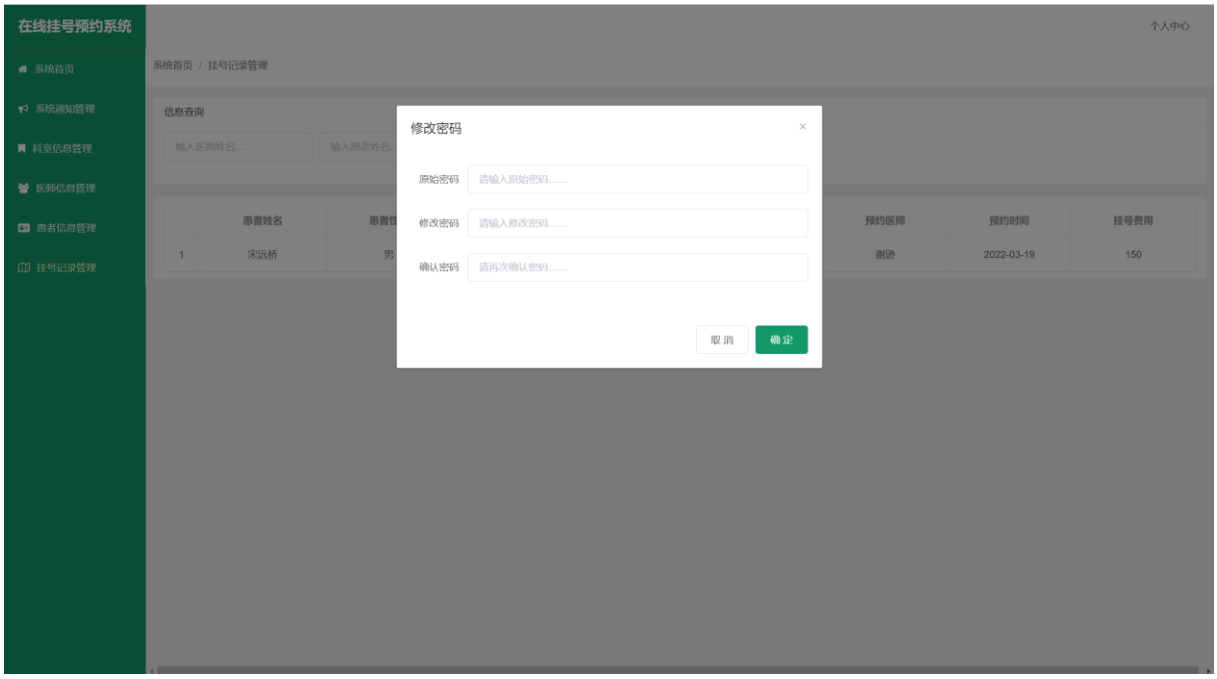
供用户进行信息修改的页面，主要是一个 HTML 表单，它并不复杂，如上就是具体实现的 HTML 代码了，当用户点击修改按钮的时候，浏览器渲染页面，然后将这个修改信息的表单呈现用户面前。

```
def updSessionInfo(request):

    loginUser = SysView.getLoginUser(request.POST.get('token'))

    if loginUser['type'] == 2:

        models.Patients.objects. \
            filter(user__id=loginUser['id']).update(
            address=request.POST.get('address'),
            card=request.POST.get('card')
        )

    models.Users.objects. \
        filter(id=loginUser['id']).update(
        userName=request.POST.get('userName'),
        name=request.POST.get('name'),
```

```
            gender=request.POST.get('gender'),
            age=request.POST.get('age'),
        )
```

当用户修改信息完成之后，点击页面中的提交按钮，可以将自己的信息发送到服务器，服务器中对这些信息进行处理，调用相关的修改用户信息方法，最终实现用户信息修改的功能。

5.4 用户密码修改



如上所示，这是我们提供的供用户进行修改密码的界面，在这里登录用户可以修改自己的密码，需要注意的是，这里用户需要完全的输入正确的原始密码，并且两次确认无误之后才可以完成密码的修改工作。

```
<el-dialog title="修改密码" width="600px" :append-to-body="true" :visible.sync="showUpdPwdFlag">
    <el-form label-width="80px" ref="userPwd" :model="userPwd" :rules="userPwdFormRules">
        <el-form-item label="原始密码" prop="oldPwd">
            <el-input type="password" v-model="userPwd.oldPwd" placeholder="请输入原始密码……" autocomplete="off">
            </el-input>
        </el-form-item>
        <el-form-item label="修改密码" prop="newPwd">
            <el-input type="password" v-model="userPwd.newPwd" placeholder="请输入修改密码……" autocomplete="off">
            </el-input>
        </el-form-item>
        <el-form-item label="确认密码" prop="rePwd">
            <el-input type="password" v-model="userPwd.rePwd" placeholder="请再次确认密码……" autocomplete="off">
```

```
                </el-input>
            </el-form-item>
        </el-form>
        <div slot="footer" class="dialog-footer">
            <el-button @click="showUpdPwdFlag = false">取 消</el-button>
            <el-button type="primary" @click="updPwd('userPwd')">确 定</el-button>
        </div>
</el-dialog>
```

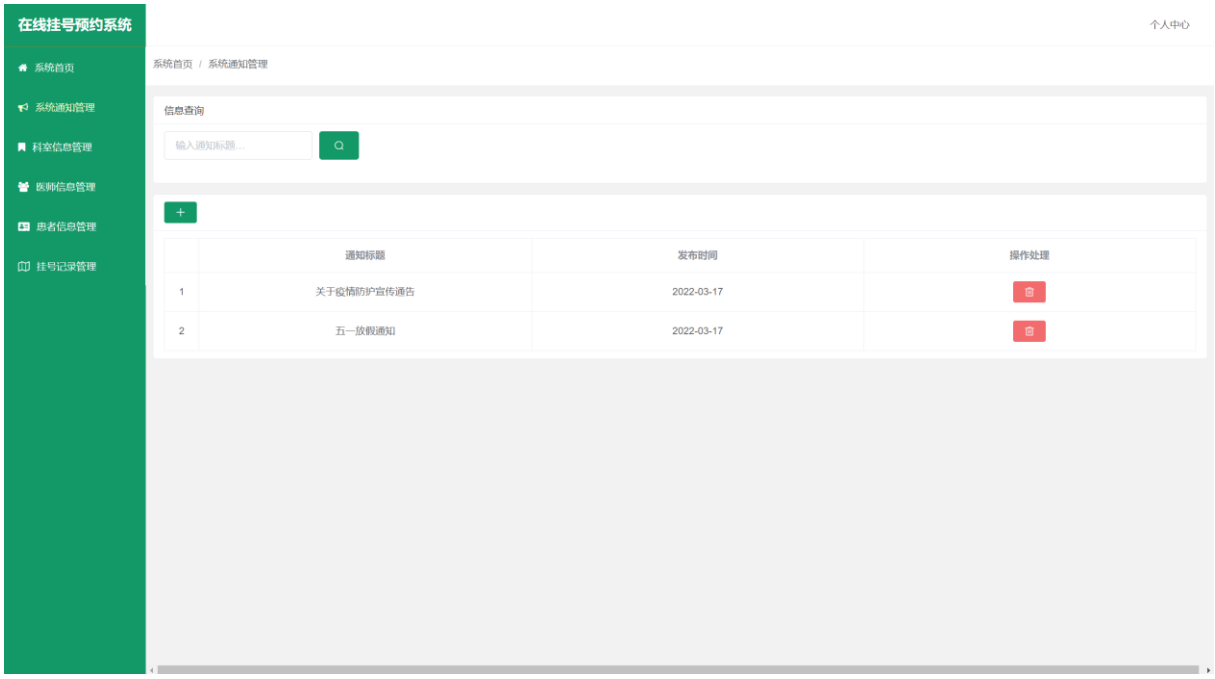
对于上边这个修改密码的功能界面，它主要是一个修改的表单，具体实现这个界面的代码如上所示，当用户点击修改按钮之后，经过浏览器的渲染，最终呈现给用户一个修改个人信息的功能页面。

```
def updSessionPwd(request):

    loginUser = SysView.getLoginUser(request.POST.get('token'))

    models.Users.objects. \
        filter(id=loginUser['id']).update(
        passWord=request.POST.get('newPwd'),
    )

    return BaseView.success()
```

用户在完全正确的输入密码之后，点击提交按钮，修改的密码发送到服务器，服务器找到登录时记录的用户信息，然后修改对应的密码信息，并且修改登录中记录的用户信息，这样就完成了密码修改的操作。

6. （管理端）通知管理模块

6.1 通知信息展示



如上所示，这是系统通知信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
    <el-card shadow="never">
    <div slot="header">
        信息查询
    </div>
    <div>
        <el-form :inline="true" :model="qryForm">
            <el-form-item >
                <el-input  v-model="qryForm.title"  placeholder="输入通知标题…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item>
                <el-button  type="primary"  icon="el-icon-search"
@click="getPageInfo(1, 10)"></el-button>
            </el-form-item>
        </el-form>
    </div>
</el-card>

<el-card shadow="never">
    <div slot="header">
```

```
            <el-button type="primary" size="mini"
                       icon="el-icon-plus" @click="showAddWin()"></el-button>
        </div>
        <div>
            <el-table v-loading="loading"
                element-loading-text="拼命加载中" element-loading-spinner="el-icon-loading"
                element-loading-background="rgba(124, 124, 124, 0.8)" :data="pageInfos" border>
                <el-table-column align="center" type="index"></el-table-column>
                <el-table-column align="center" prop="title" label="通知标题"></el-table-column>
                <el-table-column align="center" prop="putTime" label="发布时间"></el-table-column>
                <el-table-column align="center" label="操作处理">
                    <template slot-scope="scope">
                        <el-button icon="el-icon-delete"
                            type="danger" size="mini" @click="delInfo(scope.row.id)"></el-button>
                    </template>
                </el-table-column>
            </el-table>
            <el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
                @current-change="handleCurrentChange" :current-page="pageIndex" :page-sizes="[10, 20, 50]"
                :page-size="pageSize" layout="total, sizes, prev, pager, next, jumper" :total="totalInfo">
            </el-pagination>
        </div>
        </el-card>
```

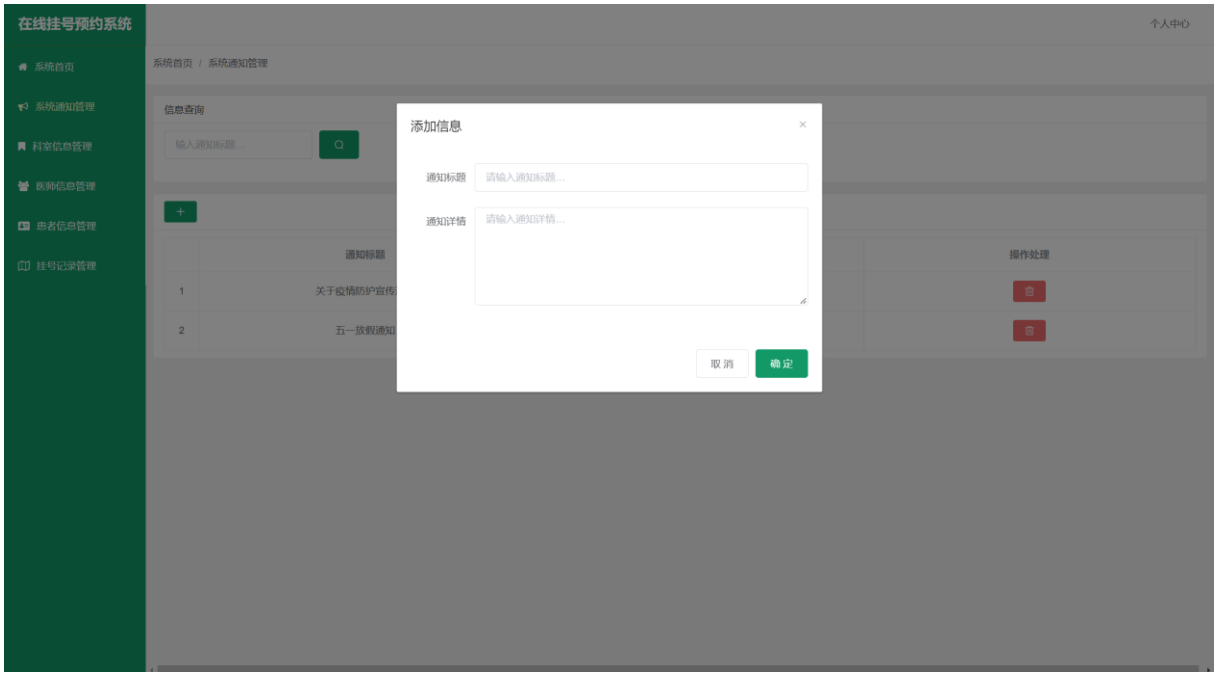
如上所示，这是我们为了实现通知信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的通知信息数据。

```
def getPageInfo(request):

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    title = request.GET.get('title')

    qruery = Q();

    if BaseView.isExit(title):
        qruery = qruery & Q(title__contains=title)
```

```
data = models.Notices.objects.filter(qruery).order_by("-putTime")

paginator = Paginator(data, pageSize)

resl = []

for item in list(paginator.page(pageIndex)):
    temp = {
        'id': item.id,
        'title': item.title,
        'detail': item.detail,
        'putTime': item.putTime,
    }
    resl.append(temp)

temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                           paginator.page(pageIndex).paginator.num_pages,
                           paginator.count, resl)

return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询通知信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

6.2 通知信息添加



如上所示，这是我们提供的通知信息添加的页面，当点击页面上的操作按钮时，可以进入到这个页面中，在这里输入正确的信息，然后点击提交按钮就可以将数据添加到系统数据库中了。

```
<el-dialog title="添加信息" width="600px" :visible.sync="showAddFlag">
    <el-form label-width="90px" :model="noticesForm">
        <el-form-item label="通知标题">
            <el-input v-model="noticesForm.title"
                placeholder="请输入通知标题···" autocomplete="off"></el-input>
        </el-form-item>
        <el-form-item label="通知详情">
            <el-input type="textarea" v-model="noticesForm.detail"
                :rows="6" placeholder="请输入通知详情···" autocomplete="off"></el-
input>
        </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
        <el-button @click="showAddFlag = false">取 消</el-button>
        <el-button type="primary" @click="addInfo()">确 定</el-button>
    </div>
</el-dialog>
```

如上所示，这是具体的通知信息添加的处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
def addInfo(request):

    models.Notices.objects.create(
                                        title=request.POST.get('title'),
                                        detail=request.POST.get('detail'),
                                        putTime=time.strftime("%Y-%m-%d",
time.localtime())
                                    )

    return BaseView.success()
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受添加通知信息处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

6.3 通知信息删除

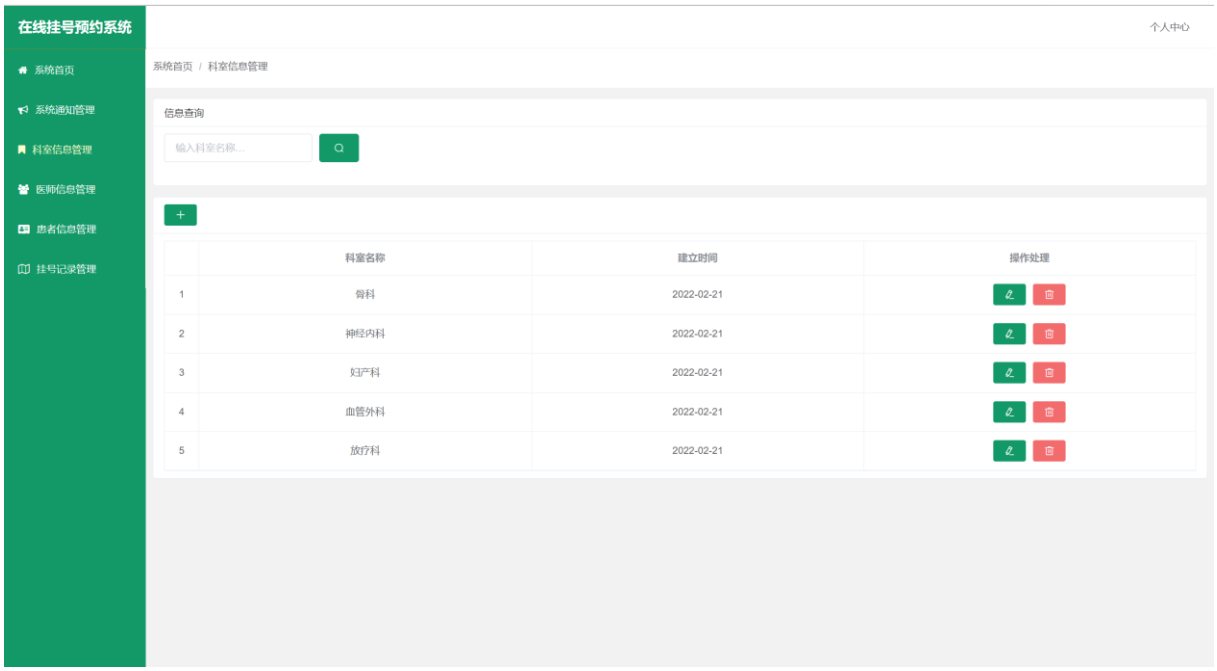
```
def delInfo(request):

    models.Notices.objects.filter(id=request.POST.get('id')).delete()
```

```
        return BaseView.success()
```

在客户端点击删除按钮时，会将要删除信息的主键信息发送到服务端，服务端有专门负责处理这类请求的方法，具体代码如上所示，我们通过发送的主键，查询到具体要删除的内容，然后调用具体的相关的方法，这样就实现了数据的删除功能。

7. （管理端）科室管理模块

7.1 科室信息展示



如上所示，这是系统科室信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
    <el-card shadow="never">
    <div slot="header">
        信息查询
    </div>
    <div>
        <el-form :inline="true" :model="qryForm">
            <el-form-item >
                <el-input v-model="qryForm.name" placeholder="输入科室名称…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item>
                <el-button type="primary" icon="el-icon-search"
@click="getPageInfo(1, 10)"></el-button>
            </el-form-item>
        </el-form>
    </div>
```

```
</el-card>

<el-card shadow="never">
    <div slot="header">
        <el-button type="primary" size="mini"
                   icon="el-icon-plus" @click="showAddWin()"></el-button>
    </div>
    <div>
        <el-table v-loading="loading"
            element-loading-text="拼命加载中" element-loading-spinner="el-icon-loading"
            element-loading-background="rgba(124, 124, 124, 0.8)" :data="pageInfos" border>
                <el-table-column align="center" type="index"></el-table-column>
                <el-table-column align="center" prop="name" label="科室名称"></el-table-column>
                <el-table-column align="center" prop="putTime" label="建立时间"></el-table-column>
                <el-table-column align="center" label="操作处理">
                    <template slot-scope="scope">
                        <el-button icon="el-icon-edit"
                                   type="primary" size="mini" @click="showUpdWin(scope.row)"></el-button>
                        <el-button icon="el-icon-delete"
                                   type="danger" size="mini" @click="delInfo(scope.row.id)"></el-button>
                    </template>
                </el-table-column>
        </el-table>
        <el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
            @current-change="handleCurrentChange" :current-page="pageIndex" :page-sizes="[10, 20, 50]"
            :page-size="pageSize" layout="total, sizes, prev, pager, next, jumper" :total="totalInfo">
        </el-pagination>
    </div>
    </el-card>
```

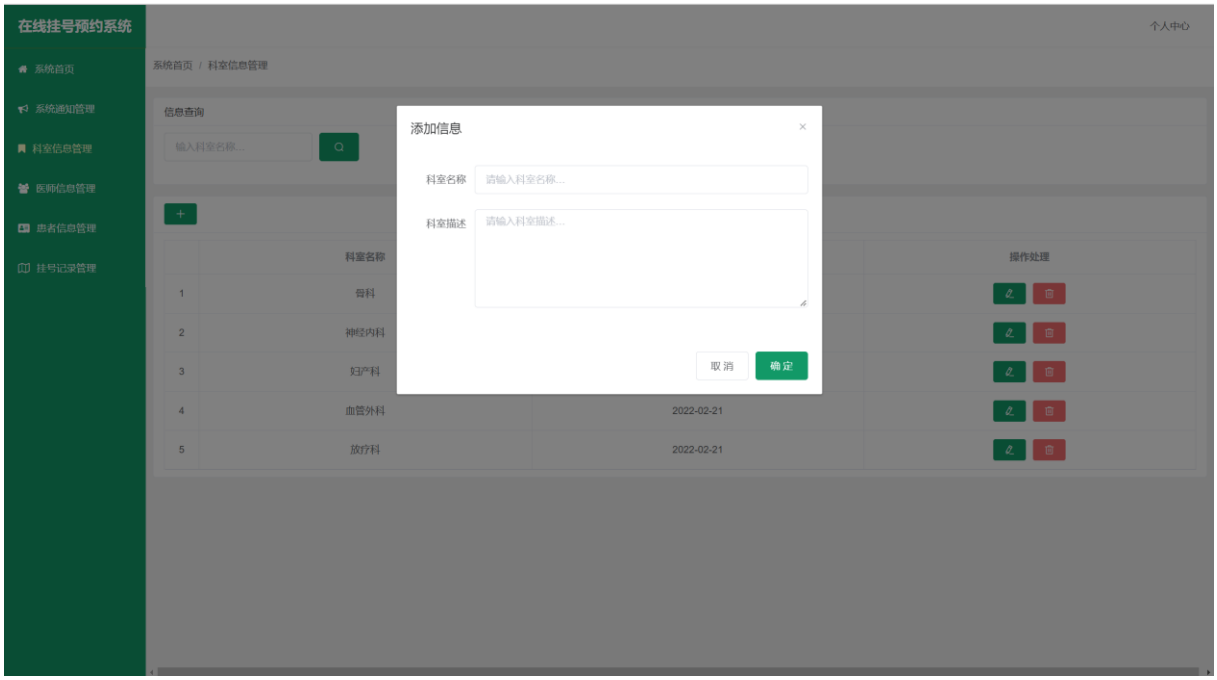
如上所示，这是我们为了实现科室信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的科室信息数据。

```
def getPageInfo(request):

    pageIndex = request.GET.get('pageIndex', 1)
```

```
    pageSize = request.GET.get('pageSize', 10)
    name = request.GET.get('name')

    qruery = Q();

    if BaseView.isExit(name):
        qruery = qruery & Q(name__contains=name)

    data = models.Offices.objects.filter(qruery).order_by("-putTime")

    paginator = Paginator(data, pageSize)

    resl = []

    for item in list(paginator.page(pageIndex)):
        temp = {
            'id': item.id,
            'name': item.name,
            'detail': item.detail,
            'putTime': item.putTime,
        }
        resl.append(temp)

    temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                               paginator.page(pageIndex).paginator.num_pages,
                               paginator.count, resl)

    return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询科室信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

7.2 科室信息添加



如上所示，这是我们提供的科室信息添加的页面，当点击页面上的操作按钮时，可以进入到这个页面中，在这里输入正确的信息，然后点击提交按钮就可以将数据添加到系统数据库中了。

```
<el-dialog title="添加信息" width="600px" :visible.sync="showAddFlag">
    <el-form label-width="90px" :model="officesForm">
        <el-form-item label="科室名称">
            <el-input v-model="officesForm.name"
                placeholder="请输入科室名称…" autocomplete="off"></el-input>
        </el-form-item>
        <el-form-item label="科室描述">
            <el-input type="textarea" v-model="officesForm.detail"
                :rows="6" placeholder="请输入科室描述…" autocomplete="off"></el-
input>
        </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
        <el-button @click="showAddFlag = false">取  消</el-button>
        <el-button type="primary" @click="addInfo()">确  定</el-button>
    </div>
</el-dialog>
```

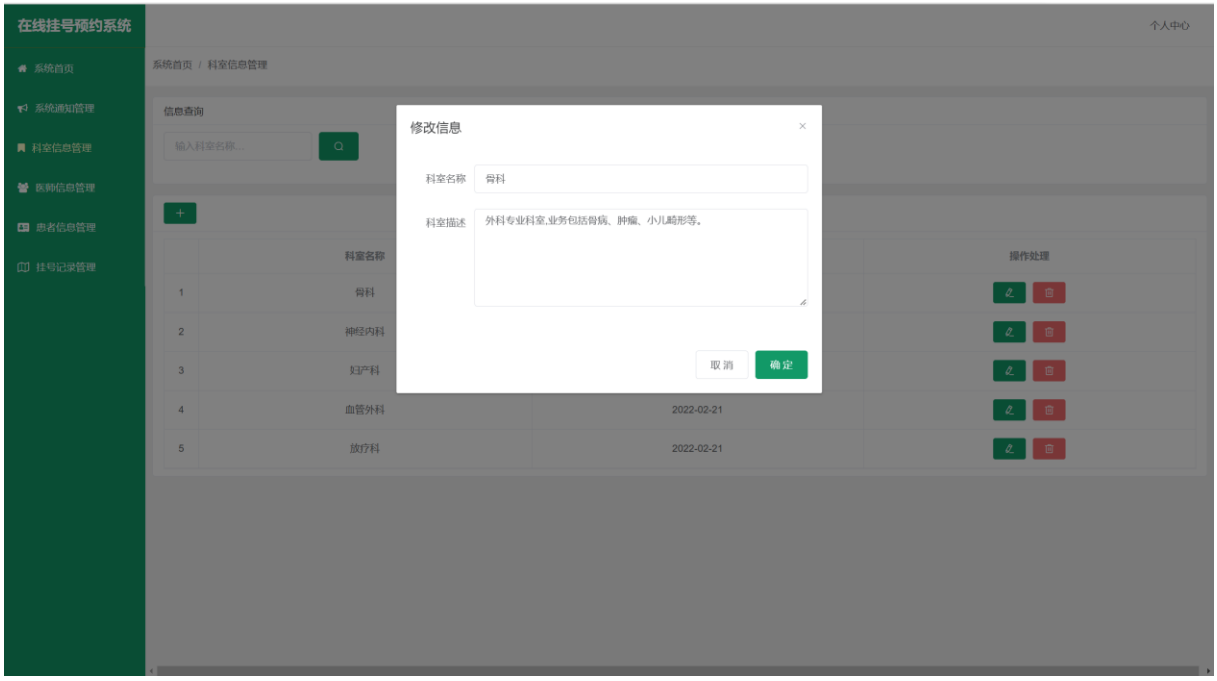
如上所示，这是具体的科室信息添加的处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
def addInfo(request):
```

```
        models.Offices.objects.create(
            name=request.POST.get('name'),
            detail=request.POST.get('detail'),
            putTime=time.strftime("%Y-%m-%d", time.localtime())
        )

        return BaseView.success()
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受添加科室信息处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

7.3 科室信息修改



如上所示，这是我们实现科室信息修改的页面，当用户点击修改按钮之后，可以进入到这个修改的页面，在这个页面中会先显示出要修改的内容，在进行编辑之后，点击提交按钮，就完成了数据的修改功能。

```
<el-dialog title="修改信息" width="600px" :visible.sync="showUpdFlag">
    <el-form label-width="90px" :model="officesForm">
        <el-form-item label="科室名称">
            <el-input v-model="officesForm.name"
                placeholder="请输入科室名称…" autocomplete="off"></el-input>
        </el-form-item>
        <el-form-item label="科室描述">
            <el-input type="textarea" v-model="officesForm.detail"
                :rows="6" placeholder="请输入科室描述…" autocomplete="off"></el-
```

```
input>
            </el-form-item>
        </el-form>
        <div slot="footer" class="dialog-footer">
            <el-button @click="showUpdFlag = false">取 消</el-button>
            <el-button type="primary" @click="updInfo()">确 定</el-button>
        </div>
</el-dialog>
```

如上所示，这是我们具体的修改页面前端实现主要代码，在这里我们通过表单的形式，将后台获取的修改信息呈现出来，这样用户并不用输入要修改的信息，就可以直接进行编辑，提升用户体验。

```
def updInfo(request):

    models.Offices.objects. \
        filter(id=request.POST.get('id')).update(
        name=request.POST.get('name'),
        detail=request.POST.get('detail'),
    )

    return BaseView.success()
```

当用户编辑完之后，可以点击页面的提交按钮，将修改信息的内容提交到服务器，在服务端程序中包含了处理这类请求的控制器，这段代码就是具体负责科室信息修改的代码，在这里我们先获取了表单中修改的信息，然后通过要修改信息的主键获取到修改之前的内容，之后调用修改方法，完成数据的修改工作，最后会向用户响应一个结果，经过渲染之后，用户就看到了修改的结果。

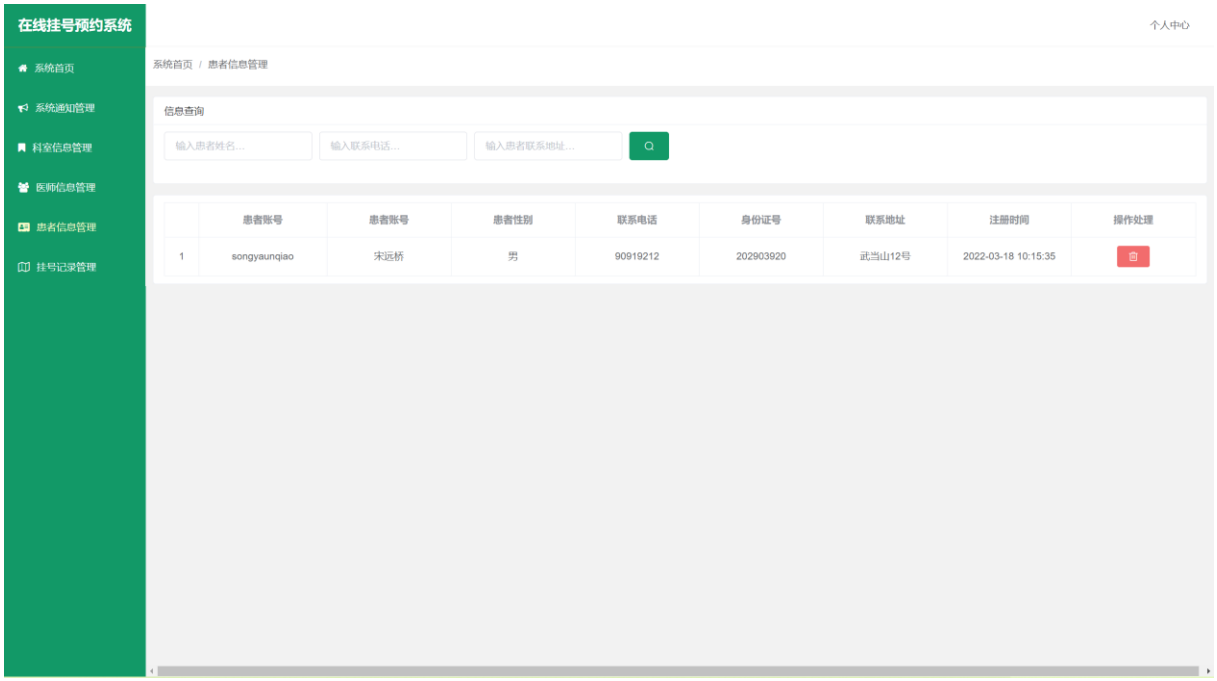
7.4 科室信息删除

```
def delInfo(request):

    if models.Doctors.objects.filter(office__id=request.POST.get('id')).exists():

        return BaseView.warn('存在关联医师无法移除')
    else:

        models.Offices.objects.filter(id=request.POST.get('id')).delete()
        return BaseView.success()
```

在客户端点击删除按钮时，会将要删除信息的主键信息发送到服务端，服务端有专门负责处理这类请求的方法，具体代码如上所示，我们通过发送的主键，查询到具体要删除的内容，然后调用具体的相关的方法，这样就实现了数据的删除功能。

8. （管理端）患者管理模块

8.1 患者信息展示



如上所示，这是系统患者信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
<el-card shadow="never">
    <div slot="header">
        信息查询
    </div>
    <div>
        <el-form :inline="true" :model="qryForm">
            <el-form-item >
                <el-input v-model="qryForm.name" placeholder="输入患者姓名…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item >
                <el-input v-model="qryForm.phone" placeholder="输入联系电话…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item >
                <el-input v-model="qryForm.address" placeholder="输入患者联系地
址…" autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item>
                <el-button type="primary" icon="el-icon-search"
```

```
@click="getPageInfo(1, 10)"></el-button>
                </el-form-item>
            </el-form>
        </div>
</el-card>

<el-card shadow="never">
        <div>
            <el-table v-loading="loading"
                element-loading-text="拼命加载中" element-loading-spinner="el-icon-loading"
                element-loading-background="rgba(124, 124, 124, 0.8)" :data="pageInfos" border>
                    <el-table-column align="center" type="index"></el-table-column>
                    <el-table-column align="center" prop="userName" label="患者账号"></el-table-column>
                    <el-table-column align="center" prop="name" label="患者账号"></el-table-column>
                    <el-table-column align="center" prop="gender" label="患者性别"></el-table-column>
                    <el-table-column align="center" prop="phone" label="联系电话"></el-table-column>
                    <el-table-column align="center" prop="card" label="身份证号"></el-table-column>
                    <el-table-column align="center" prop="address" label="联系地址"></el-table-column>
                    <el-table-column align="center" prop="createTime" label="注册时间"></el-table-column>
                    <el-table-column align="center" label="操作处理">
                        <template slot-scope="scope">
                            <el-button icon="el-icon-delete"
                                type="danger" size="mini" @click="delInfo(scope.row.id)"></el-button>
                        </template>
                    </el-table-column>
            </el-table>
            <el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
                @current-change="handleCurrentChange" :current-page="pageIndex" :page-sizes="[10, 20, 50]"
                :page-size="pageSize" layout="total, sizes, prev, pager, next, jumper" :total="totalInfo">
            </el-pagination>
        </div>
```

```
    </el-card>
```

如上所示，这是我们为了实现患者信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的患者信息数据。

```
def getPageInfo(request):

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    name = request.GET.get('name')
    phone = request.GET.get('phone')
    address = request.GET.get('address')

    qruery = Q();

    if BaseView.isExit(name):
        qruery = qruery & Q(user__name__contains=name)

    if BaseView.isExit(name):
        qruery = qruery & Q(user__phone__contains=phone)

    if BaseView.isExit(address):
        qruery = qruery & Q(address__contains=address)

    data = models.Patients.objects.filter(qruery).order_by("-user__createTime")

    paginator = Paginator(data, pageSize)

    resl = []

    for item in list(paginator.page(pageIndex)):
        temp = {
            'id': item.user.id,
            'userName': item.user.userName,
            'passWord': item.user.passWord,
            'name': item.user.name,
            'gender': item.user.gender,
            'age': item.user.age,
            'phone': item.user.phone,
            'createTime': item.user.createTime,
            'type': item.user.type,
            'address': item.address,
            'card': item.card,
        }
        resl.append(temp)
```

```
    temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                                        paginator.page(pageIndex).paginator.num_pages,
                                        paginator.count, resl)

    return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询患者信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

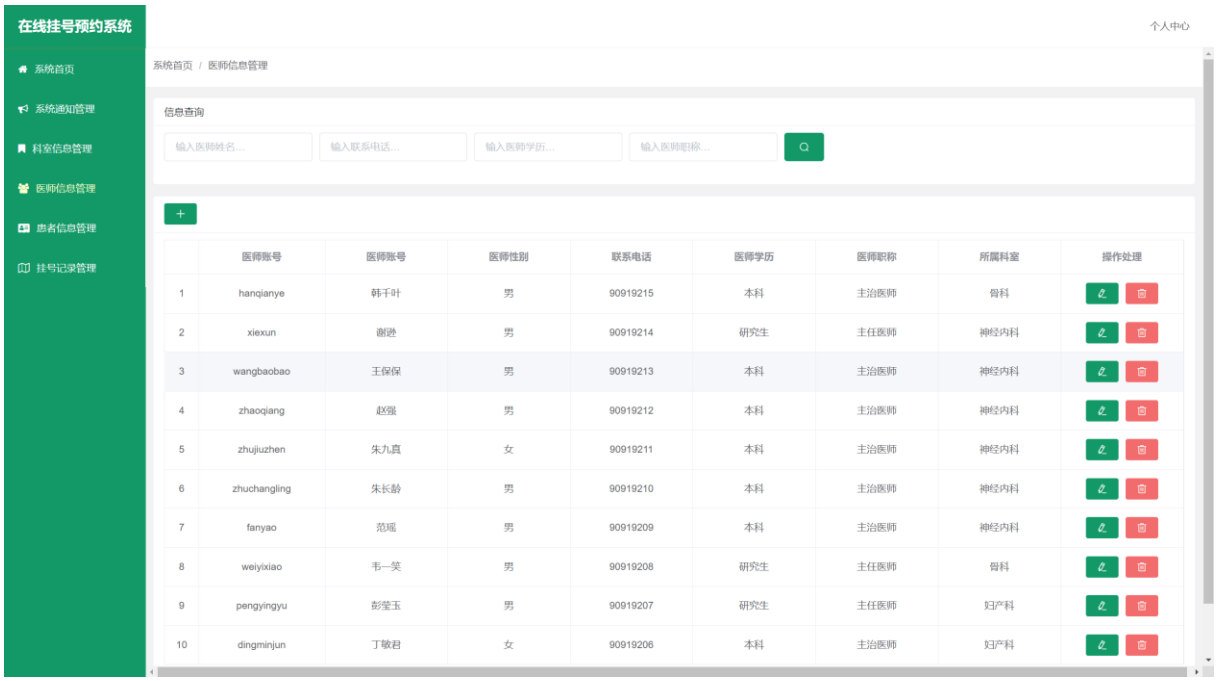
8.2 患者信息删除

```
@transaction.atomic
def delInfo(request):

    models.RegisteLogs.objects.filter(patient__user__id=request.POST.get('id')).delete()

    models.Patients.objects.filter(user__id=request.POST.get('id')).delete()

    models.Users.objects.filter(id=request.POST.get('id')).delete()

    return BaseView.success()
```

在客户端点击删除按钮时，会将要删除信息的主键信息发送到服务端，服务端有专门负责处理这类请求的方法，具体代码如上所示，我们通过发送的主键，查询到具体要删除的内容，然后调用具体的相关的方法，这样就实现了数据的删除功能。

9. （管理端）医师管理模块

9.1 医师信息展示



如上所示，这是系统医师信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
    <el-card shadow="never">
    <div slot="header">
        信息查询
    </div>
    <div>
        <el-form :inline="true" :model="qryForm">
            <el-form-item >
                <el-input v-model="qryForm.name" placeholder="输入医师姓名…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item >
                <el-input v-model="qryForm.phone" placeholder="输入联系电话…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item >
                <el-input v-model="qryForm.record" placeholder="输入医师学历…"
autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item >
                <el-input v-model="qryForm.job" placeholder="输入医师职称…"
```

```
autocomplete="off"></el-input>
                </el-form-item>
                <el-form-item>
                    <el-button type="primary" icon="el-icon-search"
@click="getPageInfo(1, 10)"></el-button>
                </el-form-item>
            </el-form>
        </div>
</el-card>

<el-card shadow="never">
        <div slot="header">
            <el-button type="primary" size="mini"
                    icon="el-icon-plus" @click="showAddWin()"></el-button>
        </div>
        <div>
            <el-table v-loading="loading"
                element-loading-text="拼命加载中" element-loading-spinner="el-icon-
loading"
                element-loading-background="rgba(124, 124, 124, 0.8)" :data="pageInfos"
border>
                    <el-table-column align="center" type="index"></el-table-column>
                    <el-table-column align="center" prop="userName" label="医师账号
"></el-table-column>
                    <el-table-column align="center" prop="name" label="医师账号
"></el-table-column>
                    <el-table-column align="center" prop="gender" label="医师性别
"></el-table-column>
                    <el-table-column align="center" prop="phone" label="联系电话
"></el-table-column>
                    <el-table-column align="center" prop="record" label="医师学历
"></el-table-column>
                    <el-table-column align="center" prop="job" label="医师职称"></el-
table-column>
                    <el-table-column align="center" prop="officeName" label="所属科室
"></el-table-column>
                    <el-table-column align="center" label="操作处理">
                        <template slot-scope="scope">
                            <el-button icon="el-icon-edit"
                                type="primary" size="mini"
@click="showUpdWin(scope.row)"></el-button>
                            <el-button icon="el-icon-delete"
                                type="danger" size="mini"
@click="delInfo(scope.row.id)"></el-button>
```

```
                        </template>
                    </el-table-column>
            </el-table>
            <el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
                    @current-change="handleCurrentChange" :current-page="pageIndex" :page-sizes="[10, 20, 50]"
                    :page-size="pageSize" layout="total, sizes, prev, pager, next, jumper" :total="totalInfo">
            </el-pagination>
        </div>
        </el-card>
```

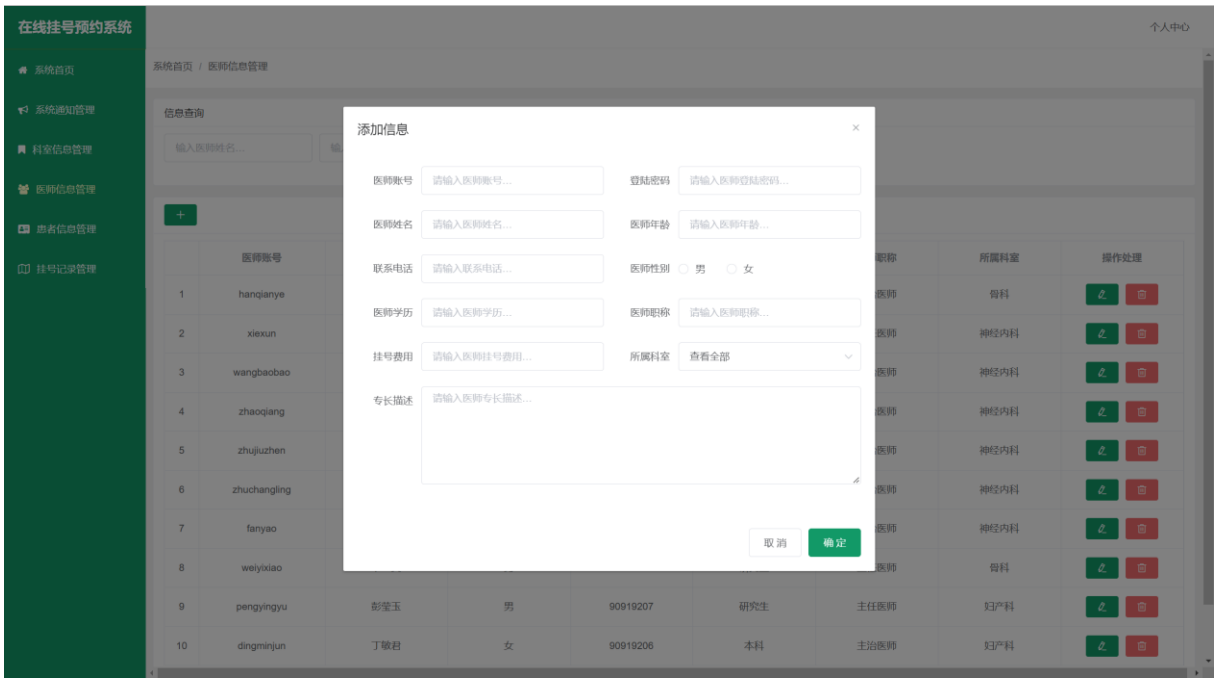
如上所示，这是我们为了实现医师信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的医师信息数据。

```
def getPageInfo(request):

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    name = request.GET.get('name')
    phone = request.GET.get('phone')
    record = request.GET.get('record')
    job = request.GET.get('job')

    qruery = Q();

    if BaseView.isExit(name):
        qruery = qruery & Q(user__name__contains=name)

    if BaseView.isExit(name):
        qruery = qruery & Q(user__phone__contains=phone)

    if BaseView.isExit(record):
        qruery = qruery & Q(record__contains=record)

    if BaseView.isExit(job):
        qruery = qruery & Q(job__contains=job)

    data = models.Doctors.objects.filter(qruery).order_by("-user__createTime")

    paginator = Paginator(data, pageSize)

    resl = []
```

```
    for item in list(paginator.page(pageIndex)):
        temp = {
            'id': item.user.id,
            'userName': item.user.userName,
            'passWord': item.user.passWord,
            'name': item.user.name,
            'gender': item.user.gender,
            'age': item.user.age,
            'phone': item.user.phone,
            'createTime': item.user.createTime,
            'type': item.user.type,
            'record': item.record,
            'job': item.job,
            'good': item.good,
            'total': item.total,
            'officeId': item.office.id,
            'officeName': item.office.name,
        }
        resl.append(temp)

    temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                               paginator.page(pageIndex).paginator.num_pages,
                               paginator.count, resl)

    return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询医师信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

9.2 医师信息添加



如上所示，这是我们提供的医师信息添加的页面，当点击页面上的操作按钮时，可以进入到这个页面中，在这里输入正确的信息，然后点击提交按钮就可以将数据添加到系统数据库中了。

```
<el-dialog title="添加信息" width="750px" :visible.sync="showAddFlag">
    <el-form label-width="90px" :model="doctorsForm">
        <el-row :gutter="15">
            <el-col :span="12">
                <el-form-item label="医师账号">
                    <el-input v-model="doctorsForm.userName"
                              placeholder="请输入医师账号…"
autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
            <el-col :span="12">
                <el-form-item label="登陆密码">
                    <el-input v-model="doctorsForm.passWord"
                              type="password" placeholder="请输入医师登陆密码…"
autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
        </el-row>
        <el-row :gutter="15">
            <el-col :span="12">
                <el-form-item label="医师姓名">
                    <el-input v-model="doctorsForm.name"
```

```
                                placeholder="请输入医师姓名…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="医师年龄">
                        <el-input v-model="doctorsForm.age"
                                  placeholder="请输入医师年龄…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="联系电话">
                        <el-input v-model="doctorsForm.phone"
                                  placeholder="请输入联系电话…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="医师性别">
                        <el-radio-group v-model="doctorsForm.gender">
                            <el-radio label="男"></el-radio>
                            <el-radio label="女"></el-radio>
                        </el-radio-group>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="医师学历">
                        <el-input v-model="doctorsForm.record"
                                  placeholder="请输入医师学历…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="医师职称">
                        <el-input v-model="doctorsForm.job"
                                  placeholder="请输入医师职称…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
```

```
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="挂号费用">
                        <el-input v-model="doctorsForm.total"
                                  placeholder="请输入医师挂号费用…" autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="所属科室">
                        <el-select  style="width:100%;"  v-model="doctorsForm.officeId" placeholder="请选择科室">
                            <el-option label="查看全部" value=""></el-option>
                            <el-option v-for="(item, index) in offices"
                                :key="index"  :label="item.name"  :value="item.id"></el-option>
                        </el-select>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-form-item label="专长描述">
                <el-input v-model="doctorsForm.good" type="textarea" :rows="6"
                        placeholder="请输入医师专长描述…" autocomplete="off"></el-input>
            </el-form-item>
        </el-form>
        <div slot="footer" class="dialog-footer">
            <el-button @click="showAddFlag = false">取 消</el-button>
            <el-button type="primary" @click="addInfo()">确 定</el-button>
        </div>
</el-dialog>
```

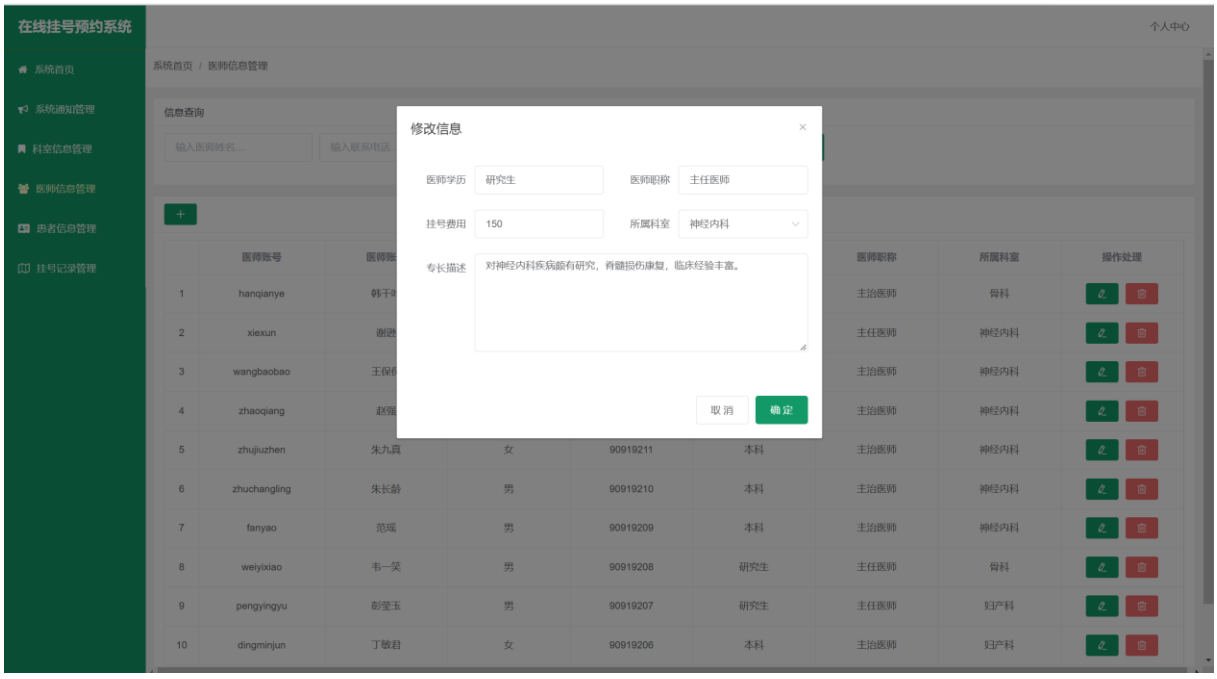
如上所示，这是具体的医师信息添加的处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
@transaction.atomic
def addInfo(request):

    user = models.Users.objects.create(
        userName=request.POST.get('userName'),
        passWord=request.POST.get('passWord'),
        name=request.POST.get('name'),
        gender=request.POST.get('gender'),
        age=request.POST.get('age'),
```

```
            phone=request.POST.get('phone'),
            type=request.POST.get('type'),
            createTime=time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        )

        models.Doctors.objects.create(
            user=user,
            record=request.POST.get('record'),
            job=request.POST.get('job'),
            good=request.POST.get('good'),
            total=request.POST.get('total'),
            office=models.Offices.objects.get(id=request.POST.get('officeId'))
        )

        return BaseView.success()
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受添加医师信息处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

9.3 医师信息修改



如上所示，这是我们实现医师信息修改的页面，当用户点击修改按钮之后，可以进入到这个修改的页面，在这个页面中会先显示出要修改的内容，在进行编辑之后，点击提交按钮，就完成了数据的修改功能。

```
<el-dialog title="修改信息" width="600px" :visible.sync="showUpdFlag">
```

```
    <el-form label-width="90px" :model="doctorsForm">
        <el-row :gutter="15">
            <el-col :span="12">
                <el-form-item label="医师学历">
                    <el-input v-model="doctorsForm.record"
                              placeholder="请输入医师学历…"
autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
            <el-col :span="12">
                <el-form-item label="医师职称">
                    <el-input v-model="doctorsForm.job"
                              placeholder="请输入医师职称…"
autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
        </el-row>
        <el-row :gutter="15">
            <el-col :span="12">
                <el-form-item label="挂号费用">
                    <el-input v-model="doctorsForm.total"
                              placeholder="请输入医师挂号费用…"
autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
            <el-col :span="12">
                <el-form-item label="所属科室">
                    <el-select  style="width:100%;"  v-model="doctorsForm.officeId"
placeholder="请选择科室">
                        <el-option label="查看全部" value=""></el-option>
                        <el-option v-for="(item, index) in offices"
                            :key="index"  :label="item.name"  :value="item.id"></el-
option>
                    </el-select>
                </el-form-item>
            </el-col>
        </el-row>
        <el-form-item label="专长描述">
            <el-input v-model="doctorsForm.good" type="textarea" :rows="6"
                placeholder="请输入医师专长描述…" autocomplete="off"></el-
input>
        </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
```

```
            <el-button @click="showUpdFlag = false">取 消</el-button>
            <el-button type="primary" @click="updInfo()">确 定</el-button>
        </div>
</el-dialog>
```

如上所示，这是我们具体的修改页面前端实现主要代码，在这里我们通过表单的形式，将后台获取的修改信息呈现出来，这样用户并不用输入要修改的信息，就可以直接进行编辑，提升用户体验。

```
def updInfo(request):

    models.Doctors.objects. \
        filter(user__id=request.POST.get('id')).update(
        record=request.POST.get('record'),
        job=request.POST.get('job'),
        good=request.POST.get('good'),
        total=request.POST.get('total'),
        office=models.Users.objects.filter(id=request.POST.get('officeId')).first()
    )

    return BaseView.success()
```

当用户编辑完之后，可以点击页面的提交按钮，将修改信息的内容提交到服务器，在服务端程序中包含了处理这类请求的控制器，这段代码就是具体负责医师信息修改的代码，在这里我们先获取了表单中修改的信息，然后通过要修改信息的主键获取到修改之前的内容，之后调用修改方法，完成数据的修改工作，最后会向用户响应一个结果，经过渲染之后，用户就看到了修改的结果。

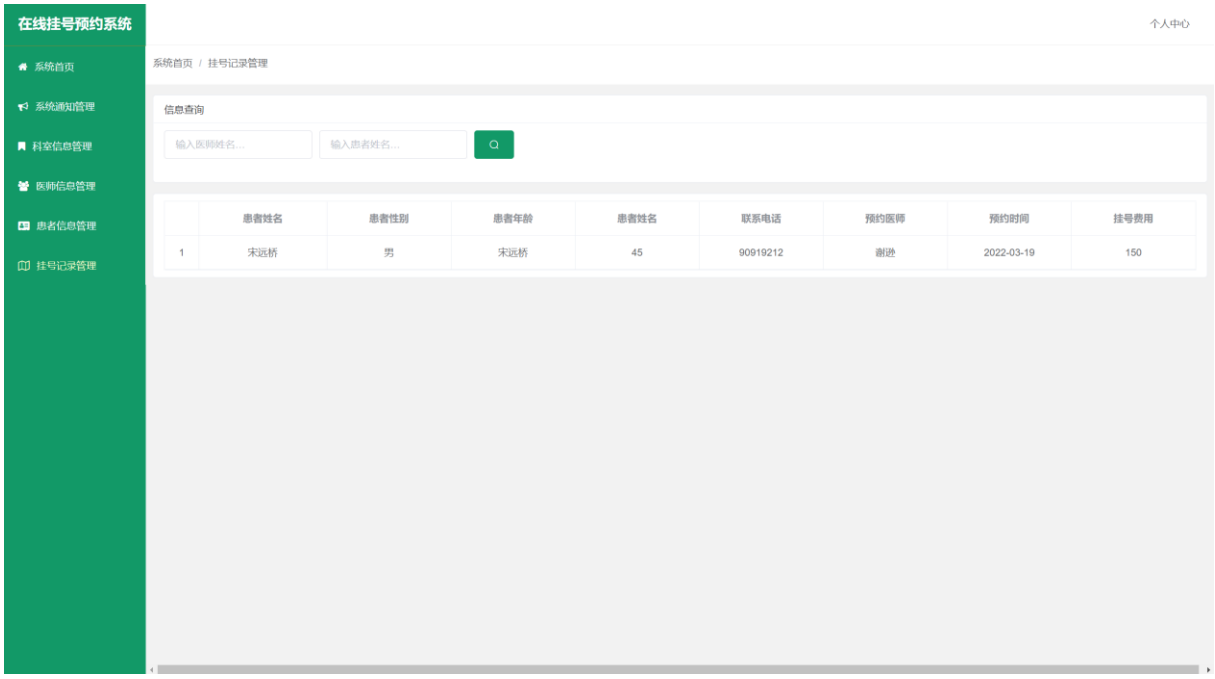
9.4 医师信息删除

```
def delInfo(request):

    if models.RegisteLogs.objects.filter(doctor__user__id=request.POST.get('id')).exists():

        return BaseView.warn('存在关联内容无法移除')
    else:

        models.Doctors.objects.filter(user__id=request.POST.get('id')).delete()
        models.Users.objects.filter(id=request.POST.get('id')).delete()

        return BaseView.success()
```

在客户端点击删除按钮时，会将要删除信息的主键信息发送到服务端，服务端有专门负责处理这类请求的方法，具体代码如上所示，我们通过发送的主键，查询到具体要删除的内容，然后调用具体的相关的方法，这样就实现了数据的删除功能。

10. （管理端）挂号记录管理模块

10.1 挂号记录信息展示



如上所示，这是系统挂号记录信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
<el-card shadow="never">
    <div slot="header">
        信息查询
    </div>
    <div>
        <el-form :inline="true" :model="qryForm">
            <el-form-item>
                <el-input v-model="qryForm.doctorName" placeholder="输入医师姓名…" autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item>
                <el-input v-model="qryForm.paientName" placeholder="输入患者姓名…" autocomplete="off"></el-input>
            </el-form-item>
            <el-form-item>
                <el-button type="primary" icon="el-icon-search" @click="getPageInfo(1, 10)"></el-button>
            </el-form-item>
        </el-form>
    </div>
```

```
</el-card>

<el-card shadow="never">
    <div>
        <el-table v-loading="loading" element-loading-text="拼命加载中" element-loading-spinner="el-icon-loading"
            element-loading-background="rgba(124, 124, 124, 0.8)" :data="pageInfos" border>
            <el-table-column align="center" type="index"></el-table-column>
            <el-table-column align="center" prop="patientName" label="患者姓名"></el-table-column>
            <el-table-column align="center" prop="patientGender" label="患者性别"></el-table-column>
            <el-table-column align="center" prop="patientName" label="患者年龄"></el-table-column>
            <el-table-column align="center" prop="patientAge" label="患者姓名"></el-table-column>
            <el-table-column align="center" prop="patientPhone" label="联系电话"></el-table-column>
            <el-table-column align="center" prop="doctorName" label="预约医师"></el-table-column>
            <el-table-column align="center" prop="registeTime" label="预约时间"></el-table-column>
            <el-table-column align="center" prop="total" label="挂号费用"></el-table-column>
            <el-table-column align="center" label="操作处理">
                <template slot-scope="scope">
                    <el-button icon="el-icon-delete"
                        type="danger" size="mini" @click="delInfo(scope.row.id)"></el-button>
                </template>
            </el-table-column>
        </el-table>
        <el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
            @current-change="handleCurrentChange" :current-page="pageIndex" :page-sizes="[10, 20, 50]"
            :page-size="pageSize" layout="total, sizes, prev, pager, next, jumper" :total="totalInfo">
        </el-pagination>
    </div>
</el-card>
```

如上所示，这是我们为了实现挂号记录信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的 挂号记

录信息数据。

```
def getPageInfo(request):

    loginUser = SysView.getLoginUser(request.GET.get('token'))

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    doctorName = request.GET.get('doctorName')
    paientName = request.GET.get('paientName')

    qruery = Q();

    if loginUser['type'] == 1:
        qruery = qruery & Q(doctor__user__id=loginUser['id'])

    if loginUser['type'] == 2:
        qruery = qruery & Q(patient__user__id=loginUser['id'])

    if BaseView.isExit(doctorName):
        qruery = qruery & Q(doctor__user__name__contains=doctorName)

    if BaseView.isExit(paientName):
        qruery = qruery & Q(patient__user__title__contains=paientName)

    data = models.RegisteLogs.objects.filter(qruery).order_by("-createTime")

    paginator = Paginator(data, pageSize)

    resl = []

    for item in list(paginator.page(pageIndex)):
        temp = {
            'id': item.id,
            'registeTime': item.registeTime,
            'createTime': item.createTime,
            'total': item.total,
            'status': item.status,
            'patientId': item.patient.user.id,
            'patientName': item.patient.user.name,
            'patientGender': item.patient.user.gender,
            'patientAge': item.patient.user.age,
            'patientPhone': item.patient.user.phone,
            'patientAddress': item.patient.address,
            'patientCard': item.patient.card,
```

```
                'doctorId': item.doctor.user.id,
                'doctorJob': item.doctor.job,
                'doctorPhone': item.doctor.user.phone,
                'doctorName': item.doctor.user.name,
                'doctorOfficeName': item.doctor.office.name,
            }
            resl.append(temp)

        temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                                   paginator.page(pageIndex).paginator.num_pages,
                                   paginator.count, resl)

        return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询挂号记录信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

10.2 挂号记录删除

```
def delInfo(request):

    models.RegisteLogs.objects.filter(id=request.POST.get('id')).delete()

    return BaseView.success()
```

在客户端点击删除按钮时，会将要删除信息的主键信息发送到服务端，服务端有专门负责处理这类请求的方法，具体代码如上所示，我们通过发送的主键，查询到具体要删除的内容，然后调用具体的相关的方法，这样就实现了数据的删除功能。

11. (客户端)挂号预约模块实现

11.1 医师信息浏览

如上所示，这是系统医师信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
<el-card style="margin-bottom: 15px;">
    <div slot="header">
 <span class="el-icon-menu text-primary"></span>  医师列表
</div>
    <div>
 <el-row v-loading="loading"
     element-loading-text="拼命加载中" element-loading-spinner="el-icon-loading"
     element-loading-background="rgba(124, 124, 124, 0.8)" :gutter="15">
          <template v-for="item in pageInfos">
               <el-col :span="8">
                    <div @click="showRegisteWin(item)"    class="doctor-panle">
                         <div class="doctor-img"></div>
                         <div class="doctor-info">
                              <div class="doctor-info-item">
                                   <span                                   class="doctor-info-
name">{{ item.name }}</span>
                                   <span class="doctor-info-title">{{ item.job }}</span>
                                   <span                                   class="doctor-info-
title">{{ item.officeName }}</span>
                              </div>
```

```
                                <div class="doctor-info-good">
                                    擅长: {{ item.good }}
                                </div>
                            </div>
                        </div>
                    </el-col>
                </template>
            </el-row>
<el-pagination v-if="pageTotal > 1" style="margin-top: 15px;" @size-change="handleSizeChange"
    @current-change="handleCurrentChange" :current-page="pageIndex"
    :page-size="pageSize" layout="total, prev, pager, next" :total="totalInfo">
</el-pagination>
</div>
</el-card>
```

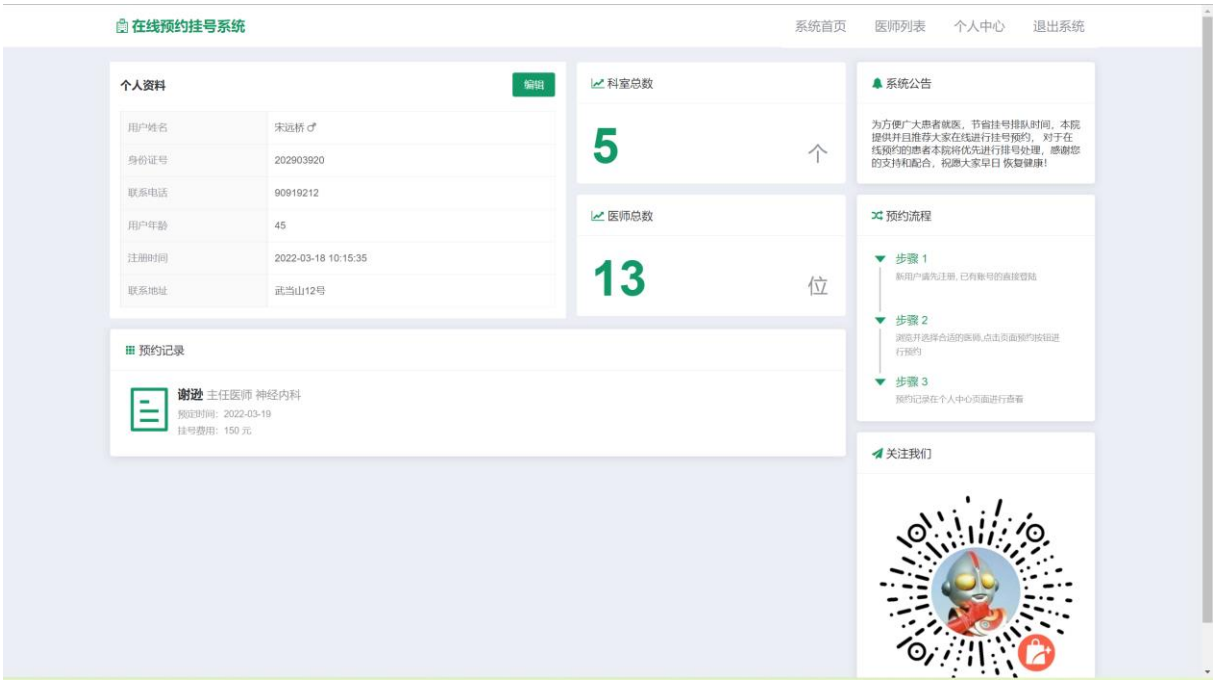
如上所示，这是我们为了实现医师信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的 XXX 信息数据。

```
def getPageInfo(request):

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    name = request.GET.get('name')
    phone = request.GET.get('phone')
    record = request.GET.get('record')
    job = request.GET.get('job')

    qruery = Q();

    if BaseView.isExit(name):
        qruery = qruery & Q(user__name__contains=name)

    if BaseView.isExit(name):
        qruery = qruery & Q(user__phone__contains=phone)

    if BaseView.isExit(record):
        qruery = qruery & Q(record__contains=record)

    if BaseView.isExit(job):
        qruery = qruery & Q(job__contains=job)

    data = models.Doctors.objects.filter(qruery).order_by("-user__createTime")

    paginator = Paginator(data, pageSize)
```

```
    resl = []

    for item in list(paginator.page(pageIndex)):
        temp = {
            'id': item.user.id,
            'userName': item.user.userName,
            'passWord': item.user.passWord,
            'name': item.user.name,
            'gender': item.user.gender,
            'age': item.user.age,
            'phone': item.user.phone,
            'createTime': item.user.createTime,
            'type': item.user.type,
            'record': item.record,
            'job': item.job,
            'good': item.good,
            'total': item.total,
            'officeId': item.office.id,
            'officeName': item.office.name,
        }
        resl.append(temp)

    temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                                        paginator.page(pageIndex).paginator.num_pages,
                                        paginator.count, resl)

    return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询 医师信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

11.2 预约记录浏览



如上所示，这是系统预约记录信息展示功能提供的页面，我们以表格的形式提供了数据分页显示的需求，同时在页面中还提供了进行数据编辑的功能，这里我们对实现它的具体过程进行详细的介绍。

```
<el-card style="margin-top: 15px;">
    <div slot="header">
        <span class="el-icon-s-grid text-primary"></span> 预约记录
    </div>
    <div>
        <el-row v-if="totalInfo > 0"  :gutter="15">
            <template v-for="item in pageInfos">
                <el-col :span="8">
                    <div class="register-panel">
                        <div class="register-icon">
                            <span class="el-icon-tickets"></span>
                        </div>
                        <div class="register-title">
                            <span class="register-doctor-name">{{ item.doctorName }}</span>
                            <span class="register-doctor-desc">{{ item.doctorJob }}</span>
                            <span class="register-doctor-desc">{{ item.doctorOfficeName }}</span>
                        </div>
                        <div class="register-detail">
                            <span  class="register-detail-item"> 预 定 时 间 ：
```

```
{{ item.registeTime }}</span>
                                    <span  class="register-detail-item"> 挂 号 费 用 ：
{{ item.total }} 元</span>
                                </div>
                            </div>
                        </el-col>
                    </template>
                </el-row>
                <el-empty v-else description="暂无相关记录"></el-empty>
            </div>
        </el-card>
```

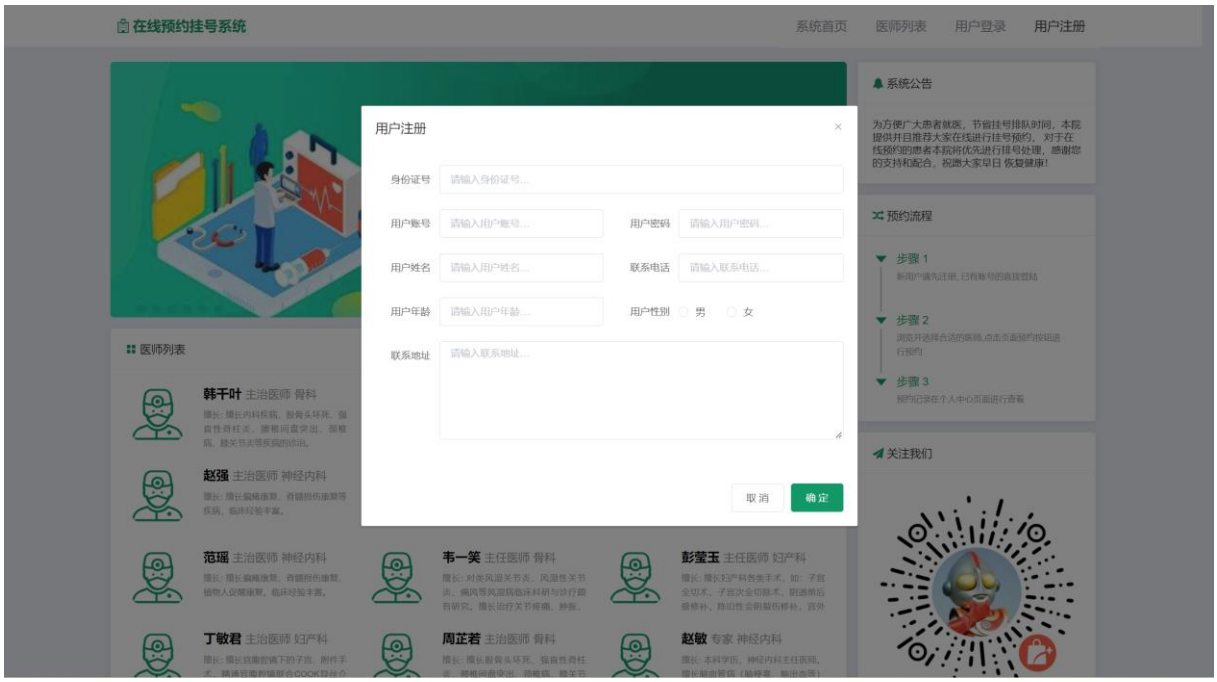
如上所示，这是我们为了实现 预约记录信息展示页面而提供的前端主要实现代码，在这里我们通过循环的方式将后台获取的数据显示到页面上，这样大家就看到了系统存储的 预约记录信息数据。

```
def getPageInfo(request):

    loginUser = SysView.getLoginUser(request.GET.get('token'))

    pageIndex = request.GET.get('pageIndex', 1)
    pageSize = request.GET.get('pageSize', 10)
    doctorName = request.GET.get('doctorName')
    paientName = request.GET.get('paientName')

    qruery = Q();

    if loginUser['type'] == 1:
        qruery = qruery & Q(doctor__user__id=loginUser['id'])

    if loginUser['type'] == 2:
        qruery = qruery & Q(patient__user__id=loginUser['id'])

    if BaseView.isExit(doctorName):
        qruery = qruery & Q(doctor__user__name__contains=doctorName)

    if BaseView.isExit(paientName):
        qruery = qruery & Q(patient__user__title__contains=paientName)

    data = models.RegisteLogs.objects.filter(qruery).order_by("-createTime")

    paginator = Paginator(data, pageSize)

    resl = []

    for item in list(paginator.page(pageIndex)):
```

```
            temp = {
                'id': item.id,
                'registeTime': item.registeTime,
                'createTime': item.createTime,
                'total': item.total,
                'status': item.status,
                'patientId': item.patient.user.id,
                'patientName': item.patient.user.name,
                'patientGender': item.patient.user.gender,
                'patientAge': item.patient.user.age,
                'patientPhone': item.patient.user.phone,
                'patientAddress': item.patient.address,
                'patientCard': item.patient.card,
                'doctorId': item.doctor.user.id,
                'doctorJob': item.doctor.job,
                'doctorPhone': item.doctor.user.phone,
                'doctorName': item.doctor.user.name,
                'doctorOfficeName': item.doctor.office.name,
            }
            resl.append(temp)

        temp = BaseView.parasePage(int(pageIndex), int(pageSize),
                                   paginator.page(pageIndex).paginator.num_pages,
                                   paginator.count, resl)

        return BaseView.successData(temp)
```

之所以我们可以在页面代码中，通过循环的方式将数据显示到页面上，这主要是因为当用户访问服务器的时候，会将查询 预约记录信息的请求参数发送到服务端，在服务端有专门接受这类请求的控制器，而具体处理分页请求的代码如上，通过这段代码，我们接受了系统查询数据集合的参数，然后调用 业务层相关的处理方法，再由业务层的相关项目代码调用 数据处理层的代码，最终我们获取了符合要求的数据，将这些信息返回，经过渲染之后，呈现出来。

11.3 普通用户注册



如上所示，这是我们提供的用户注册，当点击页面上的注册菜单时，可以进入到这个页面中，在这里输入正确的信息，然后点击提交按钮就可以实现注册了。

```
   <el-dialog title="用户注册" width="700px" :visible.sync="showRegisterFlag">
<el-form label-width="90px" :model="registerFrom">
      <el-form-item label="身份证号">
            <el-input v-model="registerFrom.card"
                        placeholder="请输入身份证号…" autocomplete="off"></el-
input>
      </el-form-item>
      <el-row :gutter="15">
                  <el-col :span="12">
                        <el-form-item label="用户账号">
                              <el-input v-model="registerFrom.userName"
                                          placeholder="请输入用户账号…"
autocomplete="off"></el-input>
                        </el-form-item>
                  </el-col>
                  <el-col :span="12">
                        <el-form-item label="用户密码">
                              <el-input v-model="registerFrom.passWord" type="password"
                                          placeholder="请输入用户密码…"
autocomplete="off"></el-input>
                        </el-form-item>
                  </el-col>
            </el-row>
```

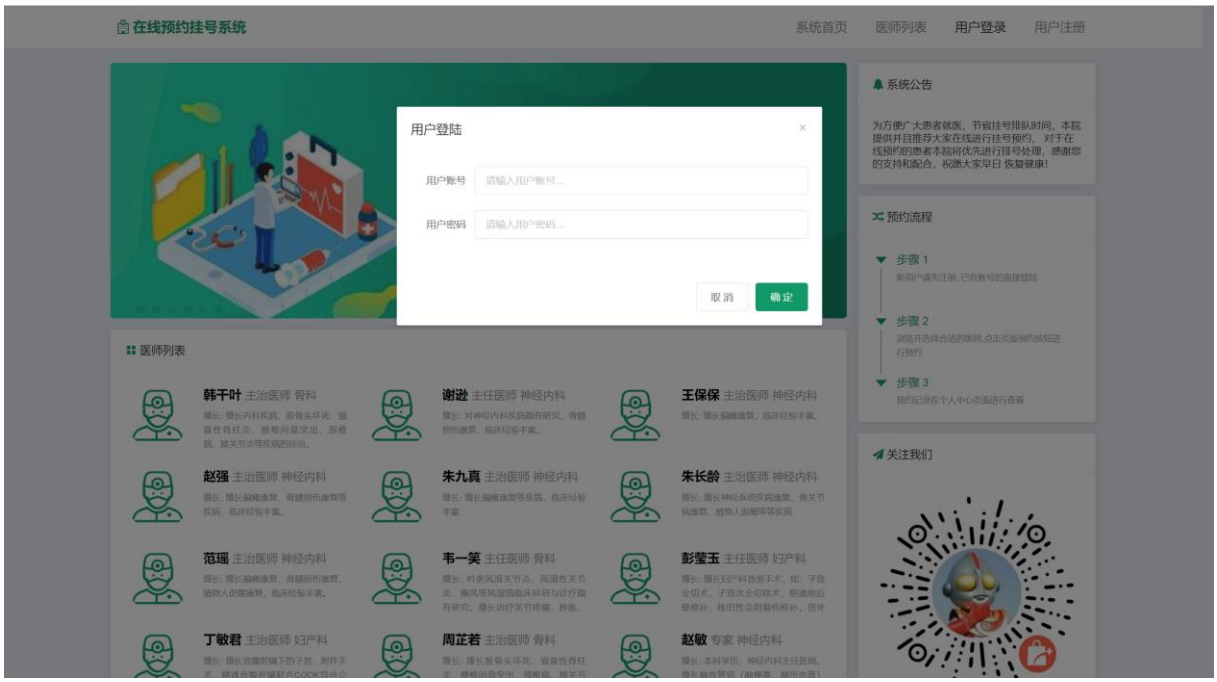
```
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="用户姓名">
                        <el-input v-model="registerFrom.name"
                                  placeholder="请输入用户姓名…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="联系电话">
                        <el-input v-model="registerFrom.phone"
                                  placeholder="请输入联系电话…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="用户年龄">
                        <el-input v-model="registerFrom.age"
                                  placeholder="请输入用户年龄…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="用户性别">
                        <el-radio-group v-model="registerFrom.gender">
                            <el-radio label="男"></el-radio>
                            <el-radio label="女"></el-radio>
                        </el-radio-group>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-form-item label="联系地址">
          <el-input v-model="registerFrom.address" type="textarea"
                           rows="6"  placeholder="请输入联系地址…"
autocomplete="off"></el-input>
      </el-form-item>
        </el-form>
<div slot="footer" class="dialog-footer">
      <el-button @click="showRegisterFlag = false">取 消</el-button>
      <el-button type="primary" @click="register()">确 定</el-button>
</div>
</el-dialog>
```

如上所示，这是具体的用户注册的处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
@transaction.atomic
def addInfo(request):

    user = models.Users.objects.create(
        userName=request.POST.get('userName'),
        passWord=request.POST.get('passWord'),
        name=request.POST.get('name'),
        gender=request.POST.get('gender'),
        age=request.POST.get('age'),
        phone=request.POST.get('phone'),
        type=request.POST.get('type'),
        createTime=time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    )

    models.Patients.objects.create(
        user=user,
        address=request.POST.get('address'),
        card=request.POST.get('card')
    )

    return BaseView.success()
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受用户注册处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

11.4 普通用户登录



如上所示，这是我们提供的用户登陆页面，当点击页面上用户登陆菜单时，可以进入到这个页面中，在这里输入正确的信息，然后点击确定按钮就可以实现登陆。

```
    <el-dialog title="用户登陆" width="600px" :visible.sync="showLoginFlag">
    <el-form label-width="90px" :model="loginForm">
        <el-form-item label="用户账号">
            <el-input v-model="loginForm.userName"
                      placeholder="请输入用户账号..." autocomplete="off"></el-input>
        </el-form-item>
            <el-form-item label="用户密码">
            <el-input v-model="loginForm.passWord" type="password"
                      placeholder="请输入用户密码..." autocomplete="off"></el-input>
        </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
        <el-button @click="showUpdFlag = false">取 消</el-button>
        <el-button type="primary" @click="login()">确 定</el-button>
    </div>
</el-dialog>
```

如上所示，这是具体的用户登陆处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
def login(request):

    userName = request.POST.get('userName')
```

```
    passWord = request.POST.get('passWord')
    flag = int(request.POST.get('flag'))

    user = models.Users.objects.filter(userName=userName)

    if (user.exists()):

        user = user.first()

        if (flag == 0) & (user.type == 2) :

            return SysView.warn('普通用户无权使用管理功能')

        elif (flag == 1) & (user.type != 2):
            return SysView.warn('管理员或教师无法登陆')

        else:
            if user.passWord == passWord:

                token = uuid.uuid4()

                resl = {
                    'token': str(token)
                }

                cache.set(token, user.id, 60*60*60*3)

                return SysView.successData(resl)
            else:
                return SysView.error('用户密码输入错误')
    else:
        return SysView.error('用户名输入错误')
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受添加用户登陆处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

11.5 预约记录提交

如上所示，这是我们提供的预约记录提交的页面，当点击页面上的操作按钮时，可以进入到这个页面中，在这里输入正确的信息，然后点击提交按钮就可以将数据添加到系统数据库中了。

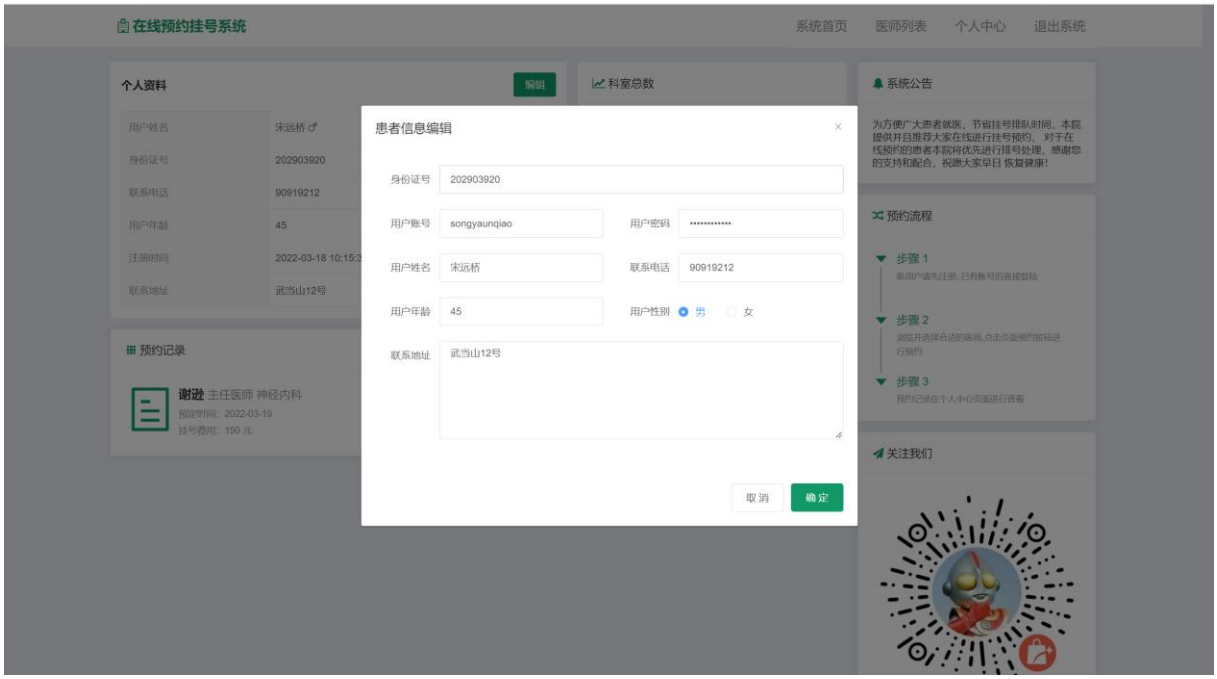
```
<el-dialog title="挂号预约" width="600px" :visible.sync="showRegisteFlag">
    <el-form label-width="90px" :model="registeForm">
        <el-form-item label="挂号费用">
            <el-input v-model="registeForm.total" disabled
                placeholder="请输入挂号费用…" autocomplete="off"></el-input>
        </el-form-item>
        <el-form-item label="预约时间">
            <el-date-picker style="width: 100%;"
                v-model="registeForm.registeTime"
                type="date" value-format="yyyy-MM-dd" placeholder="选择具体时
间">
            </el-date-picker>
        </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
        <el-button @click="showRegisteFlag = false">取  消</el-button>
        <el-button type="primary" @click="addInfo()">确  定</el-button>
    </div>
 </el-dialog>
```

如上所示，这是具体用户预约挂号的处理页面主要代码，在这里我们设置了一个添加数据的表单，这些表单会随着用户的请求响应到客户端，经过渲染之后，就可以呈现在操作人员面前了。

```
def addInfo(request):

    user = SysView.getLoginUser(request.POST.get('token'))

    qruery = Q()
    qruery = qruery & Q(patient__user__id=user['id'])
    qruery = qruery & Q(doctor__user__id=request.POST.get('doctorId'))
    qruery = qruery & Q(registeTime=request.POST.get('registeTime'))

    if models.RegisteLogs.objects.filter(qruery).exists():

        return BaseView.warn('预约记录已存在')
    else:
        models.RegisteLogs.objects.create(
            total=request.POST.get('total'),
            status=request.POST.get('status'),
            registeTime=request.POST.get('registeTime'),
            createTime=time.strftime("%Y-%m-%d", time.localtime()),
            patient=models.Patients.objects.filter(user__id=user['id']).first(),

doctor=models.Doctors.objects.filter(user__id=request.POST.get('doctorId')).first()
        )

        return BaseView.success()
```

在填写完添加信息的具体内容之后，点击页面上的提交按钮，表单中的数据就被发送到服务端，在服务端有专门接受用户预约挂号的处理的控制器，经过它的解析之后，会调用相关的实现代码实现数据添加的功能。而上边这段代码就是具体的数据添加处理代码，这里我们先是获取了表单中的数据，然后调用了业务层的处理代码，之后再由业务层的处理代码调用数据处理层的代码，经过这样一系列的处理之后，我们将数据添加到项目中，当一切执行完成之后，系统给出响应内容，浏览器解析之后，呈现到用户面前。

11.6 患者信息编辑



如上所示，这是我们实现用户修改的页面，当用户点击修改按钮之后，可以进入到这个修改的页面，在这个页面中会先显示出要修改的内容，在进行编辑之后，点击提交按钮，就完成了数据的修改功能。

```
<el-dialog title="患者信息编辑" width="700px" :visible.sync="showUpdUserFlag">
    <el-form label-width="90px" :model="userFrom">
        <el-form-item label="身份证号">
            <el-input v-model="userFrom.card"
                      placeholder="请输入身份证号…" autocomplete="off"></el-input>
        </el-form-item>
        <el-row :gutter="15">
            <el-col :span="12">
                <el-form-item label="用户账号">
                    <el-input v-model="userFrom.userName"
                              placeholder="请输入用户账号…" autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
            <el-col :span="12">
                <el-form-item label="用户密码">
                    <el-input v-model="userFrom.passWord" type="password"
                              placeholder="请输入用户密码…" autocomplete="off"></el-input>
                </el-form-item>
            </el-col>
```

```
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="用户姓名">
                        <el-input v-model="userFrom.name"
                                  placeholder="请输入用户姓名…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="联系电话">
                        <el-input v-model="userFrom.phone"
                                  placeholder="请输入联系电话…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-row :gutter="15">
                <el-col :span="12">
                    <el-form-item label="用户年龄">
                        <el-input v-model="userFrom.age"
                                  placeholder="请输入用户年龄…"
autocomplete="off"></el-input>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="用户性别">
                        <el-radio-group v-model="userFrom.gender">
                            <el-radio label="男"></el-radio>
                            <el-radio label="女"></el-radio>
                        </el-radio-group>
                    </el-form-item>
                </el-col>
            </el-row>
            <el-form-item label="联系地址">
                <el-input v-model="userFrom.address" type="textarea"
                          rows="6"  placeholder="请输入联系地址…"
autocomplete="off"></el-input>
            </el-form-item>
        </el-form>
        <div slot="footer" class="dialog-footer">
            <el-button @click="showUpdUserFlag = false">取 消</el-button>
            <el-button type="primary" @click="updLoginUser()">确 定</el-button>
        </div>
```

```
</el-dialog>
```

如上所示，这是我们具体的修改页面前端实现主要代码，在这里我们通过表单的形式，将后台获取的修改信息呈现出来，这样用户并不用输入要修改的信息，就可以直接进行编辑，提升用户体验。

```
def updSessionInfo(request):

    loginUser = SysView.getLoginUser(request.POST.get('token'))

    if loginUser['type'] == 2:

        models.Patients.objects. \
            filter(user__id=loginUser['id']).update(
            address=request.POST.get('address'),
            card=request.POST.get('card')
        )

    models.Users.objects. \
        filter(id=loginUser['id']).update(
        userName=request.POST.get('userName'),
        name=request.POST.get('name'),
        gender=request.POST.get('gender'),
        age=request.POST.get('age'),
    )
```

当用户编辑完之后，可以点击页面的提交按钮，将修改信息的内容提交到服务器，在服务端程序中包含了处理这类请求的控制器，这段代码就是具体负责用户信息修改的代码，在这里我们先获取了表单中修改的信息，然后通过要修改信息的主键获取到修改之前的内容，之后调用修改方法，完成数据的修改工作，最后会向用户响应一个结果，经过渲染之后，用户就看到了修改的结果。