# 第三部分 Web端

# 配置开发环境

## 任务描述

**内 容**：安装并配置开发环境。

**学 时**：2

**知识点**：Python运行环境，PyCharm开发环境配置，Flask web开发设计

**重点**：

**难点**：

## 授课思路

开发环境涉及到Python运行环境配置、PyCharm IDE配置、Flask web开发框架的设计等几方面的内容。要按照先后步骤，一步一步的实现。

## 任务指导

1. 下载Python并设置运行环境
2. 安装并设置Python运行环境
3. 安装配置PyCharm开发环境
4. 集成Git
5. Flask概述及学习目的和意义
6. 安装Flask及其依赖包
7. Web架构及项目配置文件
8. 构建一个简单的html主页面
9. 编写启动脚本，启动主程序

## 任务实现

1. 下载python

Python最新源码，二进制文档，新闻资讯等可以在Python的官网查看，Python官网：

<https://www.python.org/>

你可以在以下链接中下载 Python 的文档，包括 HTML、PDF等格式的文档，Python文档下载地址：

<https://www.python.org/doc/>

推荐使用Python3.6及以上版本。

1. 安装并设置Python运行环境

下载完合适版本的Python后（和自己操作系统相匹配的版本），按照提示进行默认安装，安装完以后要进行环境变量的设置，此处以windows 10 操作系统为例（假设安装目录是D:\python3目录）：

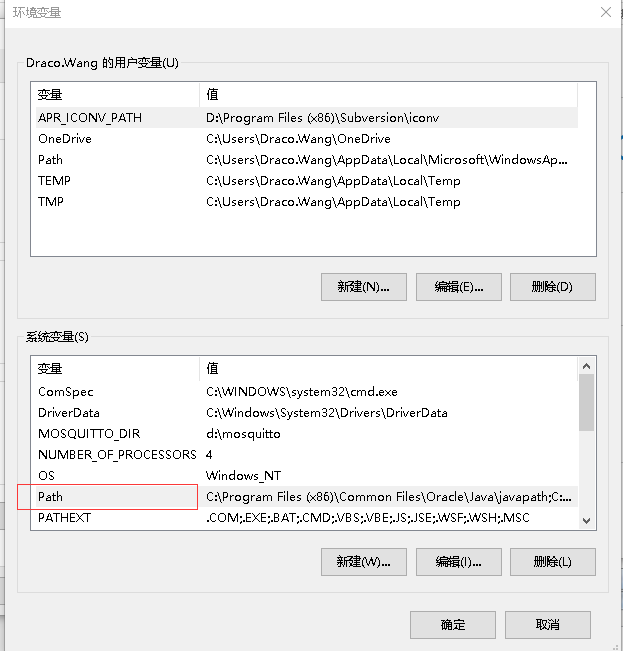
在“计算机”图标上，点击鼠标右键，选择属性



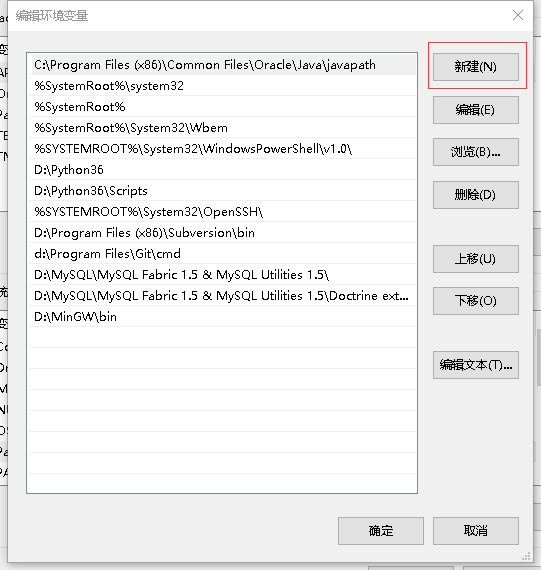
点“高级系统设置”， 在出现的界面中选择“环境变量”



点击环境变量后出现如下画面：



选择系统变量中的Path，点编辑按钮，如下所示：



点“新建”按钮，添加下面的两行代码：

D:\Python3\Scripts\ 和 D:\Python3\

确定后退出。

要验证安装的python是否正常工作，打开命令行窗口，输入python，出现下面的画面：



1. 安装配置PyCharm

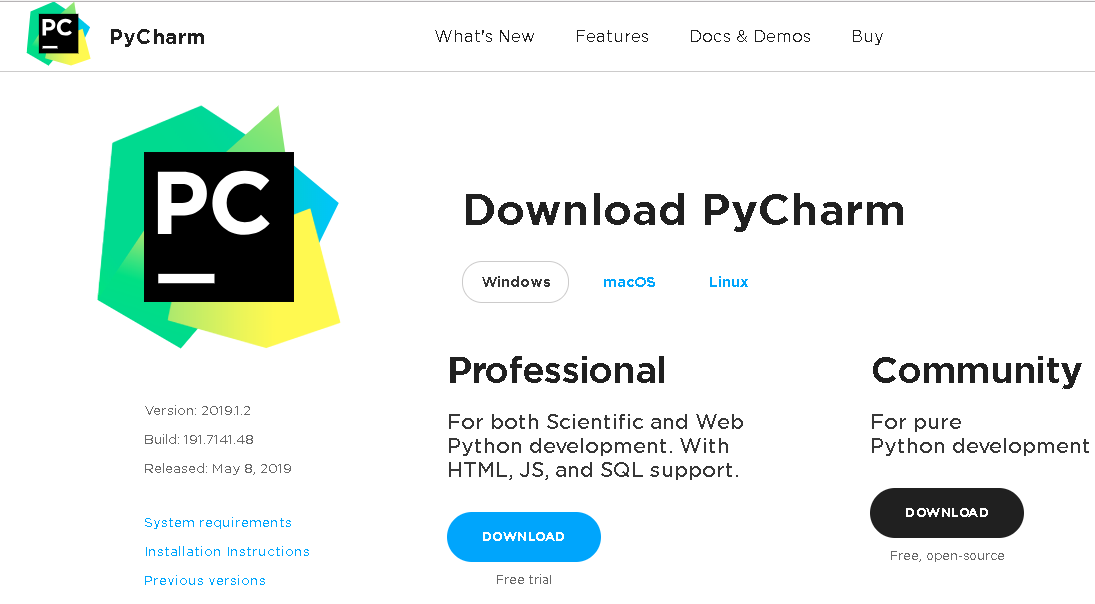
PyCharm 是一款功能强大的 Python 编辑器，具有跨平台、易使用的特点，

并且该IDE还提供了一些高级功能，用于支持Flask框架下的专业Web开发

PyCharm for windows版本的下载地址如下：

[http://www.jetbrains.com/pycharm/download/#section=windows](http://www.jetbrains.com/pycharm/download/" \l "section=windows)

进入该网站后，我们会看到如下界面:



选择Community版本，下载

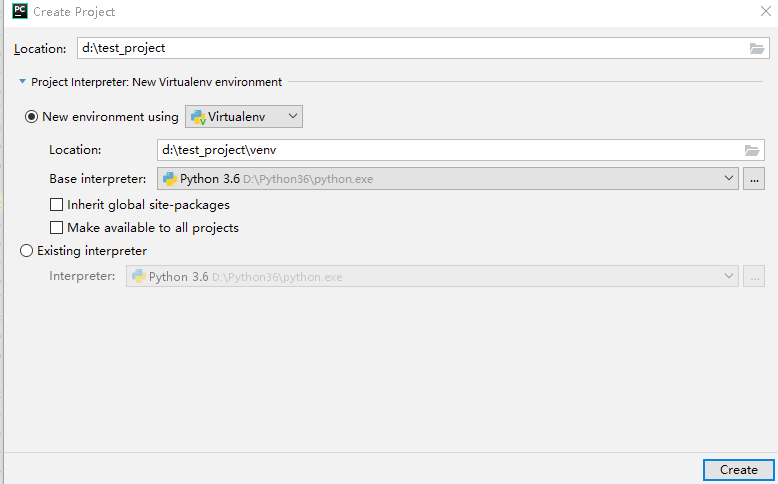
下载完以后，点击安装，选择好文件路径，直接默认安装即可。

1. 集成Git
2. Flask概述

Flask是Python编写的Web 微服务框架, 利用Flask,加上模板技术,我们可以使用Python语言快速实现一个网站或Web服务。

1. 安装Flask及其依赖包

打开PyCharm开发工具，点File—>new project

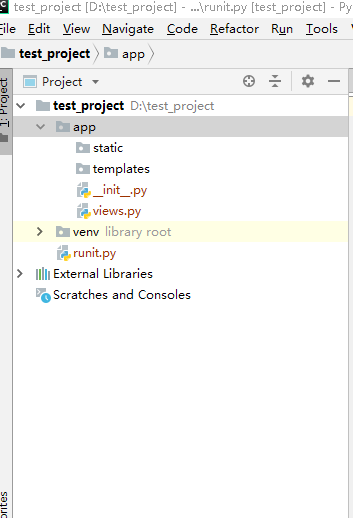


Location是项目所在路径，此处为d:\test\_project

使用Virtualenv，解释器自动默认前面已经安装好的Python3.6

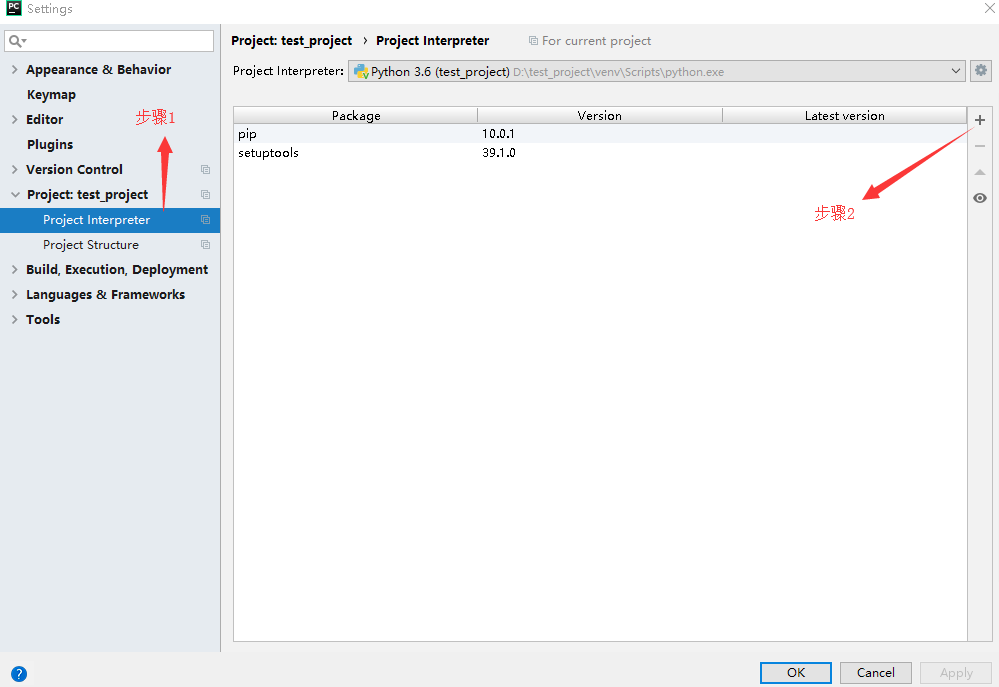
点create按钮，创建一个新的项目框架。

Flask是一个web开发框架， 要使用它开发web应用， 就要遵守它的内部约定， 主目录下必须有templates和static文件夹，分别存放html页面文件和css、js、image文件等，标准的Flask项目结构如下所示：

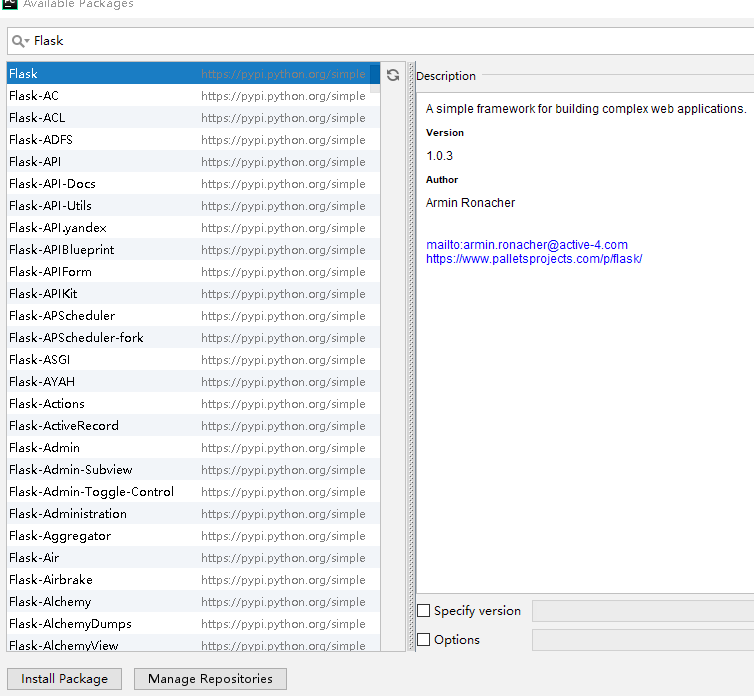


App是项目主文件目录，它里面包括templates和static两个文件夹，还有\_\_init\_\_.py和views.py文件。

要使用Flask开发框架，需要手动安装第三方开发包，方法是选择File🡪setting, 如下所示：



根据步骤提示，点project Interpreter，选择步骤2里的“+”号



在搜索框里，输入Flask，点击左下角的“Install Package”安装即可。

如果在开发过程中需要用到其他的包，安装方法同上。

1. Web架构及项目配置文件

App目录下有个 \_\_init\_\_.py和views.py两个主要的文件，其中\_\_init\_\_.py是项目的主配置文件，也是项目运行的初始化程序，可以放一些初始化操作需要的变量、数据库连接符等，\_\_init\_\_.py文件的代码如下：

from flask import Flask

app = Flask(\_\_name\_\_)

views.py文件是用来设置页面导航的， 里面的每一个函数都映射到一个或 多个请求的 URL上，views.py文件的代码如下：

from app import app

@app.route('/login.html')  
@app.route('/')  
def index():  
 return render\_template("login.html")

1. 构建一个简单的html主页面

* 在app\templates目录下， 新建login.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
 Hello, 北京交大  
</body>  
</html>

1. 编写启动脚本，启动主程序

修改app目录下的\_\_init\_\_.py文件，Flask程序对象的创建在\_\_init\_\_.py文件里完成：

from flask import Flask   
  
from flask\_sqlalchemy import SQLAlchemy  
from flask\_socketio import SocketIO

app = Flask(\_\_name\_\_)

db = SQLAlchemy(app)

socketio = SocketIO(app, async\_mode='threading')

在项目根目录下新建 runit.py文件，用来启动整个flask web程序，代码如下:

from app import app,socketio

socketio.run(app,host='127.0.0.1',debug=True,port=5000)

在代码空白处，点击鼠标右键，在菜单中选择“run runit.py”,出现如下画

面：



在浏览器打开<http://127.0.0.1:5001/,则系统根据views.py>文件里定义的 导航规则@app.route('/')，自动转到login.html页面。

# 用户登录

## 任务描述

**内 容**： 设计用户登录功能

**学 时**：2

**知识点**：数据库检索和条件查询

**重点**： Flask框架下的Web数据基本操作

**难点**：

## 授课思路

## 任务指导

1. 新建用户实体类文件
2. 新建服务层业务处理文件
3. 构建用户登录主页面
4. 接收用户输入
5. 和数据库比对用户名和密码
6. 数据判断和页面跳转

## 任务实现

1. 新建用户实体类

在 app主目录下，新建mod\_user目录，在mod\_user文件夹下新建models.py文件，定义用户实体类，代码如下：

class User(db.Model,UserMixin):  
 \_\_tablename\_\_ = 'sys\_user'  
 #colums  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(80), unique=False, nullable=True)  
 password = db.Column(db.String(80), unique=False, nullable=True)  
 real\_name = db.Column(db.String(80), unique=False, nullable=True)  
 isactive = db.Column(db.String(20), nullable=True)  
 email = db.Column(db.String(80), nullable=True)  
 created = db.Column(db.DateTime, nullable=True)

其中，id是表的主键，自增长列；

Username是用户名，real\_name是用户真实姓名，email用来找回密码。

1. 新建用户服务类

在 app\mod\_user主目录下，新建controllers.py文件，实现数据库操作，代码如下：

from app.mod\_user.models import User

def get\_all\_user():  
 return User.query.all()  
  
def select\_by\_id(id):  
 return User.query.filter\_by(id=id).first()

def select\_by\_password(name,password):  
 return User.query.filter\_by(username=name,  
 password=md5(password)).first()

1. 构建用户登录页面

在app\templates目录下，新建login.html：

<form role="form" action="login" >  
 <div class="form-group">  
 <label for="exampleInputEmail1">用户名</label>  
 <i class="fa fa-envelope"></i>  
 <input type="text" placeholder="manager" class="form-control"   
 name="username" id="username" >  
 </div>  
 <div class="form-group">   
 <label for="exampleInputPassword1">登录密码</label>  
 <i class="fa fa-lock"></i>  
 <input type="password" placeholder="1q" class="form-control"   
 name="password" id="exampleInputPassword1" >  
 </div>  
 <div class="form-actions">  
 <label class="checkbox"> <input type="checkbox" class="uniform"   
 value=""> 保存密码</label>  
 <button type="submit" class="btn btn-primary">登录</button>  
 </div>  
</form>

其中：

* form的action属性是login；
* 用户名和密码对应的html元素是input输入框，需要设置name属性，方便后端接收输入的值。

登录按钮的前端代码如下所示：

<button type="submit" class="btn btn-primary">登录</button>

是个button元素，类型是submit

1. 接收用户输入

在app目录下，新建mod\_auth文件夹， 在mod\_auth文件夹下新建forms.py文件，代码如下：

import app.mod\_user.controllers as user\_c

@app.route('/login', methods=['GET', 'POST'])  
def do\_login\_do():  
 if request.method == 'GET':  
 return render\_template("login.html")  
  
 username = request.form['username']  
 password = request.form['password']  
 user = user\_c.select\_by\_password(username,password)

If user:

return redirect(url\_for('index'))

* 定义路由

@app.route('/login', methods=['GET', 'POST'])

通过@app装饰器为函数do\_login\_do定义了一条路由，所有以POST方式提交到/login的请求都由do\_login\_do函数处理；

* 使用request.form方法接收表单数据。

username = request.form['username']

password = request.form['password']

1. 调用数据库操作，比对用户名和密码

user = user\_c.select\_by\_password(username,password)

其中 user\_c.select\_by\_password()方法调用的是服务层中对应的数据库操作

1. 数据判断和导航

if user:  
 return redirect(url\_for('index'))

如果数据正确，就直接跳转到index主页面。

# 系统管理员信息维护

## 任务描述

**内 容**： 维护系统管理员信息

**学 时**：2

**知识点**： 编辑现有的数据记录

**重点**：

**难点**：

## 授课思路

深入了解Flask中template模板的知识，理解模板中的循环遍历数据的功能

## 任务指导

1. 增加左侧菜单
2. 构建系统用户列表页面
3. 编写页面和后台绑定的代码
4. 添加数据库操作方法

## 任务实现

1. 增加左侧菜单

在app\templates目录下， 新建leftsidebar.html页面文件，用来管理所有的目录菜单，增加一个用户列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">系统设置模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/userlist">  
 <span class="sub-menu-text">系统用户信息</span></a></li>  
 </ul>  
</li>

然后在app\mod\_user\forms.py文件中，增加对应/userlist的路由信息

@app.route('/userlist')  
def list\_all\_users():  
 user = c.get\_all\_user()  
 return render\_template("user/userlist.html",userlist= user)

1. 新建用户列表页面

在app\templates\user目录下，新建userlist.html页面文件

<table id="datatable1" cellpadding="0">  
 <thead>  
 <tr>  
 <th>登录名</th>  
 <th>用户名称</th>  
 <th class="hidden-xs">创建日期</th>  
 <th>是否有效</th>  
 <th class="hidden-xs">页面风格</th>  
 <th class="hidden-xs">编辑</th>  
 </tr>  
 </thead>  
 <tbody>  
  
 {% for data in userlist %}  
 <tr class="gradeX">  
 <td>{{data.username}}</td>  
 <td>{{data.real\_name}}</td>  
 <td class="hidden-xs">{{data.created}}</td>  
 <td class="center">{{data.isactive}}</td>  
 <td class="center hidden-xs">{{data.theme}}</td>  
 <td>  
  
 <div class="btn-group dropdown">  
 <button class="btn ">  
 <i class="fa fa-gear"></i>  
 <span class="caret"></span>  
 </button>  
 <ul class="dropdown-menu">  
 <li>  
 <a href="#"   
 onclick="javascript:selectOneRecord({{data.id}})">编辑</a>  
 </li>  
 <li>  
 <a href="/deleteuser/{{data.id}}">删除</a>  
 </li>  
  
 </ul>  
 </div>  
 </td>  
 </tr>  
  
 {% endfor %}

其中

{% for data in userlist %}

………

{% endfor %}

就是模板中的代码块，上面例子里的代码就是一个简单的for循环语句，里面可以使用任何的html元素，userlist变量就是由后台代码传递过来的用户列表数据。

用户编辑功能是调用的一个ajax操作，把当前记录的ID传递给后台，也可以使用其他方式，保证把当前记录的id和状态传递给后台，并显示详细信息即可。

1. 后台添加编辑用户的路由方法

@app.route('/edituser/<int:id>')  
def edit\_user(id):  
 selectuser = User()  
 if (id != 0):  
 selectuser = c.select\_by\_id(id)  
 else:  
 selectuser.id = 0  
   
 return render\_template("user/userform.html",selectuser= selectuser)

这个方法返回到用户的编辑页面userform.html，并把选中的记录对应的属性值带到了编辑页面。



1. 操作数据库

在userform.html页面中，保存按钮提交到后台（任务8中的路由方法），最终调用

c.update\_data(user)

即控制层的update\_data方法，把数据更新到数据库中。

# 系统管理员密码修改

## 任务描述

**内 容**：系统管理员的密码删除功能

**学 时**：2

**知识点**：验证原密码的正确性

**重点**：

**难点**：

## 授课思路

单独设置修改密码按钮，点击密码出现三个输入框，验证原密码，输入两次新密码。

## 任务指导

1. 新建修改密码html页面
2. 后台添加路由规则
3. 验证原密码是否正确
4. 接收页面参数
5. 更新数据库

## 任务实现

1. 新建修改密码的html代码

在app/templates目录下，打开topbar.html页面,添加如下代码：

<form role="form" action="/changepassword" method="post">  
 < <div class="form-group">  
 <label class="col-sm-6 control-label">原密码</label>  
 <div class="col-sm-4">  
 <input type="password" name="originalpass" class="form-control"  
 id="originalpass" >  
 </div>  
  
 <label class="col-sm-6 control-label">新密码</label>  
 <div class="col-sm-4">  
 <input type="password" name="newpass" class="form-control"  
 id="newpass" >  
 </div>  
 <label class="col-sm-6 control-label">确认新密码</label>  
 <div class="col-sm-4">  
 <input type="password" name="newpass2" class="form-control"  
 id="newpass2" >  
 </div>  
</div>

1. 后台添加路由规则

打开app\mod\_user目录下的forms.py文件，添加一条路由规则：

@app.route('/changepassword', methods=['POST'])  
def do\_changepass():  
 user\_id = session.get('userid')  
 user = c.select\_by\_id(user\_id)  
  
 ……  
  
 return render\_template("index.html")

1. 验证原密码是否正确

originalpass = request.form['originalpass']  
 if md5(originalpass) != user.password:  
 flash('原始密码不正确！！', 'error')  
 ……

把当前用户的原始密码和输入的密码对比，如果不正确直接返回主页面，并显示提示信息。

1. 接收页面参数

\_user = User()  
 \_user.id = user.id  
 \_user.username = user.username  
 \_user.real\_name = user.real\_name  
 \_user.email = user.email  
 password = request.form['newpass']  
 \_user.password = md5(password)  
 ……

在客户端验证两次输入的新密码是否匹配，若匹配则后端接收新密码的值进行md5加密操作。

1. 更新数据库

c.update\_password(\_user)

flash('修改成功', 'info')

调用服务层的数据库操作方法，更新到数据库中，并在页面上显示操作成功的消息。

# 老年人模块增删改查

## 任务描述

**内 容**：老年人模块的基本操作

**学 时**：2

**知识点**：数据的增加、删除、修改和查询

**重点**：

**难点**：

## 授课思路

基于Flask框架，系统的学习一个功能模块的增、删、改、查等功能。

## 任务指导

1. 新建老年人实体类文件
2. 新建服务层业务处理文件
3. 新建老年人列表页面
4. 增加左侧导航菜单
5. 列表页面和后台数据绑定
6. 新建老年人编辑页面
7. 在列表页面添加跳转方法
8. 新建后台路由规则

## 任务实现

1. 新建老年人实体类文件

在 app主目录下，新建mod\_oldperson文件夹，在mod\_oldperson文件夹下，新建models.py文件，代码如下：

class OldPersoninfo(db.Model):  
 \_\_tablename\_\_ = 'oldperson\_info'  
 id = db.Column(db.Integer,autoincrement=True, primary\_key=True)  
 ORG\_ID = db.Column(db.Integer)  
 CLIENT\_ID = db.Column(db.Integer)  
 username = db.Column(db.String(50))  
 gender = db.Column(db.String(5))  
 phone = db.Column(db.String(50))  
 id\_card = db.Column(db.String(50))  
 birthday = db.Column(db.DateTime)  
 checkin\_date = db.Column(db.DateTime)  
 checkout\_date = db.Column(db.DateTime)  
 imgset\_dir = db.Column(db.String(200))  
 profile\_photo = db.Column(db.String(200))  
 room\_number = db.Column(db.String(50))  
 firstguardian\_name = db.Column(db.String(50))  
 firstguardian\_relationship = db.Column(db.String(50))  
 firstguardian\_phone = db.Column(db.String(50))  
 firstguardian\_wechat = db.Column(db.String(50))  
 secondguardian\_name = db.Column(db.String(50))  
 secondguardian\_relationship = db.Column(db.String(50))  
 secondguardian\_phone = db.Column(db.String(50))  
 secondguardian\_wechat = db.Column(db.String(50))  
 health\_state = db.Column(db.String(50))

具体字段说明，请参考“数据库文档设计.docx”

1. 新建服务层业务处理文件

在 app\mod\_oldperson文件夹下，新建controllers.py文件，代码如下：

from app.mod\_oldperson.models import OldPersoninfo  
from app import db  
  
def get\_all\_data():  
 return OldPersoninfo.query.all()  
  
def select\_by\_id(id):  
 return OldPersoninfo.query.filter\_by(id=id).first()  
  
def delete\_by\_id(id):  
 record = OldPersoninfo.query.filter\_by(id=id).first()  
 db.session.delete(record)  
 db.session.commit()  
  
#update or insert data  
def update\_insert\_data(data):  
 if (data.id == 0):  
 db.session.add(data)  
 #data.imgset\_dir = data.imgset\_dir+'\\'+str(data.id)  
 db.session.commit()  
 else:  
 \_data = data.\_\_dict\_\_  
   
 \_data.pop('\_sa\_instance\_state')  
  
 OldPersoninfo.query.filter\_by(id=data.id).update(\_data)  
 db.session.commit()

代码中包括了添加数据，编辑数据，删除数据和查询等常见的数据库操作方法。

1. 新建老年人列表页面

在app\templates文件夹下，新建oldperson目录，在这个目录下新建oldpersoninfolist.html文件:

<div class="box-body">  
 <table id="datatable2" cellpadding="0" cellspacing="0"   
 border="0" class="datatable table table-striped   
 table-bordered table-hover">  
 <thead>  
 <tr>  
 <th>编号</th>  
 <th>姓名</th>  
 <th>性别</th>  
 <th>房间号</th>  
 <th >入园日期</th>  
 <th>出生日期</th>  
 <th>身份证号</th>  
 <th>健康状况</th>  
  
 <th class="hidden-xs">编辑</th>  
 </tr>  
 </thead>  
 <tbody>  
  
 {% for data in listdata %}  
 <tr class="gradeX">  
 <td>{{data.id}}</td>  
 <td>{{data.username}}</td>  
 <td class="hidden-xs">{{data.gender}}</td>  
 <td>{{data.room\_number}}</td>  
  
 <td>{{data.checkin\_date.strftime('%Y-%m-%d')}}</td>  
 <td>{{data.birthday.strftime('%Y-%m-%d')}}</td>  
 <td>{{data.id\_card}}</td>  
 <td>{{data.health\_state}}</td>  
 <td>  
  
 <div class="btn-group dropdown">  
 <button class="btn　 btn-xs btn-info dropdown-toggle"   
 data-toggle="dropdown">  
 <i class="fa fa-gear"></i>  
 <span class="caret"></span>  
 </button>  
 <ul class="dropdown-menu">  
 <li>  
 <a href="#" onclick="javascript:selectdata({{data.id}})">编辑</a>  
 </li>  
 <li>  
 <a href="/setoldpersonid/{{data.id}}" >获取头像</a>  
 </li>  
 <li>  
 <a href="/deleteoldpersoninfo/{{data.id}}" >删除</a>  
 </li>  
  
 </ul>  
 </div>  
 </td>  
 </tr>  
 {% endfor %}  
 </tbody>

其中

{% for data in listdata %}

………

{% endfor %}

就是模板中的代码块，上面例子里的代码就是一个简单的for循环语句，里面可以使用任何的html元素，listdata变量就是由后台代码传递过来的老年人列表数据。

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个老年人信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">老年人模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/oldpersoninfolist">  
 <span class="sub-menu-text">老年人信息列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

在 app\mod\_oldperson文件夹下，新建forms.py文件：

Import app.mod\_person.controllers as c

@app.route('/oldpersoninfolist')  
def list\_all\_oldpersoninfo():  
 \_listdata = c.get\_all\_data()  
 return render\_template("oldperson/oldpersoninfolist.html",  
 listdata= \_listdata)

* 导入服务层的数据库操作方法

Import app.mod\_person.controllers as c

* 定义路由

@app.route('/oldpersoninfolist')

通过@app装饰器为函数list\_all\_oldpersoninfo定义了一条路由，所有以GET方式匹配到/oldpersoninfolist的请求都由list\_all\_oldpersoninfo函数处理；

* 返回模板文件

从数据库取出所有数据后，返回到list页面文件，并把数据回传给页面

return render\_template("oldperson/oldpersoninfolist.html",  
 listdata= \_listdata)

最终效果图如下所示：



1. 新建老年人编辑页面

在app\templates\oldperson\目录下，新建oldpersoninfoform.html页面，

<form class="form-horizontal" action="/saveoldpersoninfo"   
method="post" role="form">  
<input type="hidden" name="record\_id" value="{{selectdata.id}}">  
  
 <div class="form-group">  
 <label class="col-sm-3 control-label">姓名</label>  
 <div class="col-sm-2">  
<input class="form-control" id="focusedInput"   
 name="username" type="text" value="{{selectdata.username}}">  
 </div>  
  
<label class="col-sm-3 control-label">房间号</label>  
<div class="col-sm-2">  
 <input class="form-control"   
 value="{{selectdata.room\_number}}"  
 name="room\_number" id="disabledInput" type="text" >  
 </div>  
</div>

……

页面form的action属性是 “/saveoldpersoninfo”。

1. 在列表页面添加跳转方法

打开app\templates\oldperson目录下的oldpersoninfolist.html,在列表代码中添加如下Javascript代码：

<li>  
 <a href="#" onclick="javascript:selectdata({{data.id}})">编辑</a>

</li>

……

function selectdata(id)  
{  
 $.ajax({  
 url: '/editoldpersoninfo'+'/'+id,  
 contentType: 'application/json;charset=UTF-8',  
 cache:false,  
 dataType: 'html',  
 success: function (response) {  
 var result = $('#box-config').append(response).find('#box-config').html();  
 $('#box-config').html(result);  
  
 $('#box-config').modal('show');  
 }  
 });  
}

1. 增加路由规则

打开app\mod\_oldperson目录下的forms.py文件，添加对应/saveoldpersoninfo的一条路由：

@app.route('/saveoldpersoninfo',methods=['POST'])

def save\_oldpersoninfo():  
record = OldPersoninfo()  
record.id = int(request.form['record\_id'])  
record.username = request.form['username']  
record.room\_number = request.form['room\_number']

c.update\_insert\_data(record)  
listdata = c.get\_all\_data()

……  
return  
render\_template("oldperson/oldpersoninfolist.html", listdata=listdata)

* 定义路由

@app.route('/saveoldpersoninfo', methods=[ 'POST'])

通过@app装饰器为函数save\_oldpersoninfo定义了一条路由，所有以POST方式提交到/saveoldpersoninfo的请求都由save\_oldpersoninfo函数处理；

* 使用request.form方法接收表单数据。

record.id = int(request.form['record\_id'])

record.username = request.form['username']

……

* 更新数据库

c.update\_insert\_data(record)  
listdata = c.get\_all\_data()

# 老年人模块头像管理

## 任务描述

**内 容**：管理老年人的头像数据信息

**学 时**：2

**知识点**：websocket数据推送，web页面调用python脚本

**重点**：

**难点**：

## 授课思路

头像管理模块是系统中的一个重要功能，涉及到OpenCV脚本的调用，websocket数据的推送等功能。

## 任务指导

1. 新建老年人头像信息列表页面
2. 增加左侧导航菜单
3. 列表页面和后台数据绑定
4. 从Open CV获取老年人头像
5. 把图片信息推送到web页面

## 任务实现

1. 新建老年人头像信息列表页面

在app\templates\oldperson目录下，新建oldpersonimagelist.html文件:

<div id="imagelistid" class="row">  
  
 {% for image in imagelist %}  
  
 <div class="col-md-3 isotope-item">  
 <div class="filter-content">  
 <img src="img/oldperson/{{selectdata.id}}/{{image}}"  
 alt="" class="img-responsive">  
 <div class="hover-content">  
 <h4>操作</h4>  
 <a class="btn btn-success hover-link"  
 href="#" onclick="setImage('{{image}}')" title="设为头像">  
 <i class="fa fa-edit fa-1x"></i>  
 </a>  
 <a class="btncolorbox-button"  
 href="img/oldperson/{{selectdata.id}}/{{image}}"   
 title="点击放大">  
 <i class="fa fa-search-plus fa-1x"></i>  
 </a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
  
</div>

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个老年人头像信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">老年人模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/oldpersonimagelist">  
 <span class="sub-menu-text">老年人头像列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

打开app\mod\_oldperson文件夹下的forms.py文件，增加一条路由规则：

@app.route('/oldpersonimagelist')  
def list\_all\_oldpersonimage():  
 \_id = session.get('oldpersonid')  
 selectdata = c.select\_by\_id(\_id)  
  
 \_image\_dir = app.static\_folder+'/img/oldperson'+'/'+str(\_id)  
 \_images = []  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
 else:  
 # list all the image of this folder  
 \_images = os.listdir(\_image\_dir)  
 return render\_template("oldperson/oldpersonimagelist.html",  
 selectdata=selectdata, imagelist=\_images)

上述代码的作用就是遍历所选老人所在的文件夹下的所有头像文件，并把数据回传给前端页面。

1. 从Open CV获取老年人头像

这个任务的功能是调用后台OpenCV的脚本，启动一个摄像头，为老年人拍照，把各种表情的照片存入特定的文件夹，同时利用websocket技术推送到web页面上，进行实时展示。

* 首先在老年人信息列表页面增加获取头像的入口代码，方法是打开 app\templates\oldperson目录下的oldpersoninfolist.html,在列表代码里添加一个导航规则：

<li>  
 <a href="/setoldpersonid/{{data.id}}" >获取头像</a>  
</li>

* 打开app\mod\_person目录下的forms.py文件，增加一条路由规则：

@app.route('/setoldpersonid/<int:id>')  
def set\_id\_session(id):  
 session['oldpersonid'] = id  
 selectdata = c.select\_by\_id(id)  
  
 \_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
 os.popen(\_cmd)  
  
 #check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/oldperson' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()  
  
 return redirect('/oldpersonimagelist')

上述代码首先接收页面传递过来的老年人编号ID，然后启动OpenCV的拍照脚本：

\_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
os.popen(\_cmd)

接下来的代码放到了一个多线程里运行，作用是监测特定文件夹下有没有文件变化，如果有新的图片文件，就直接推送到web页面上：

#check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/oldperson' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()

1. 把图片信息推送到web页面

利用websoccket技术，把后台得到的头像信息实时推送到web端展示，后台推送代码如下：

def run\_file\_monitor(\*\*name):  
   
 path\_to\_watch = r''+str(name['data'].imgset\_dir)+'/'+str(name['data'].id)  
 before = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
   
 while 1:  
 time.sleep(2)  
 after = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
 added = [f for f in after if not f in before]  
 removed = [f for f in before if not f in after]  
 if added:  
 print("Added: ", added)  
 socketio.emit('image\_data',{'id':   
 name['data'].id,'image':added[0]},   
 namespace='/oldpersonimage')  
 if removed:  
 print("Removed: ", ", ".join(removed))  
 before = after

使用socketio框架实现信息推送，前台页面需要设置接收代码，打开app\templates\oldperson目录下的oldpersonimagelist.html,

<script src="js/websocket/socket.io.js"></script>

<script>  
  
var socket;

function connet\_ws()  
{  
 namespace = '/oldpersonimage'  
 socket = io.connect(location.protocol + '//' + document.domain + ':' + location.port + namespace,{'reconnection delay': 2000});  
  
  
  
 socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });  
}

……

首先引用socket.io.js框架：

<script src="js/websocket/socket.io.js"></script>

和后台ws服务建立连接，并建立消息监听回调函数：

function connet\_ws()

……  
socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });

当有消息到达时，调用addDiv脚本，把图片信息展示在页面上：

function addDiv(id,image)  
{  
  
 $("#imagelistid").prepend("<div class='col-md-3 isotope-item'>"+  
 "<div class='filter-content'>"+  
 "<img src='img/oldperson/"+id+"/"+image+  
 "'"+" class='img-responsive' />"+  
 "<div class='hover-content'>"+  
 "<h4>操作</h4>"+  
 "<a class='btn btn-success hover-link'   
 href='#' title='设为头像'>"+  
 "<i class='fa fa-edit fa-1x'></i>"+  
 "</a>"+  
 "<a class='btn colorbox-button'   
 href='img/oldperson/"+id+"/"+image+"'"+"   
 title='点击放大'>"+  
 "<i class='fa fa-search-plus fa-1x'></i>"+  
 "</a></div></div>"+  
 "</div>");  
  
}

# 老年人模块报表统计

## 任务描述

**内 容**：老年人模块数据统计功能

**学 时**：2

**知识点**：ECharts报表设计

**重点**：

**难点**：

## 授课思路

本模块涉及到数据统计分析，ECharts报表设计等功能。教授学生从不同角度对数据进行分析整合，使用不同的图表形式进行展现。

## 任务指导

1. 新建老年人数据统计页面
2. 增加页面导航并添加路由规则
3. 引入ECharts
4. 使用ECharts展现数据

## 任务实现

1. 新建老年人数据统计页面

在app\templates\oldperson目录下，新建oldpersonstatistic.html页面：

<div class="box-title">  
 <h4><i class="fa fa-signal"></i>年龄统计图</h4>  
   
 </div>  
 <div class="box-body" id="main" style="width: 500px;height:350px;">  
  
 </div>  
 </div>  
 <!-- /BOX -->  
</div>  
<div class="col-md-6">  
 <!-- BOX -->  
 <div class="box border inverse">  
 <div class="box-title">  
 <h4><i class="fa fa-bar-chart-o"></i>监护人分布图</h4>  
 <div class="box-body" id="main2"   
 style="width: 500px;height:350px;">  
  
 </div  
 </div>

</div>

1. 增加页面导航并设置路由规则

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个老年人统计分析的菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">老年人模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/oldpersonimagelist">  
 <span class="sub-menu-text">老年人统计分析</span></a></li>  
 </ul>  
</li>

打开app\mod\_oldperson\目录下的forms.py文件，添加一条路由规则：

@app.route('/oldpersonstatistic')  
def run\_oldperson\_statistic():  
 return render\_template("oldperson/oldpersonstatistic.html")

1. 引入ECharts

从ECharts官网下载JS文件，

官网地址是：https://echarts.baidu.com/download.html

下载源文件压缩包，把echarts.min.js文件复制出来放到static\js\echarts目录下，编辑app\templates\oldperson目录下的oldpersonstatistic.html，引入JS文件：

<script type="text/javascript" src="js/echarts/echarts.min.js"></script>

1. 使用ECharts展示数据

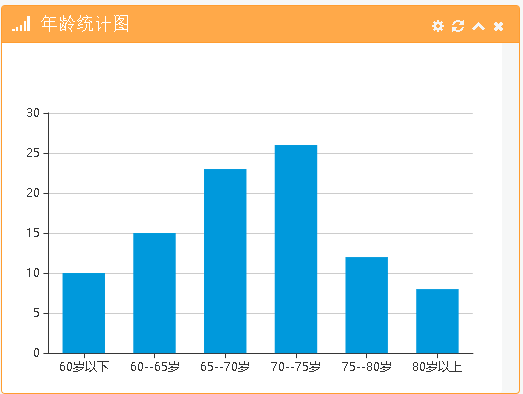
打开app\templates\oldperson目录下的oldpersonstatistic.html文件，添加JS脚本：

<script>  
 // 基于准备好的dom，初始化echarts实例  
 var myChart = echarts.init(document.getElementById('main'));  
  
 // 指定图表的配置项和数据  
 option = {  
 color: ['#3398DB'],  
 tooltip : {  
 trigger: 'axis',  
 axisPointer : { // 坐标轴指示器，坐标轴触发有效  
 type : 'shadow' // 默认为直线，可选为：'line' | 'shadow'  
 }  
 },  
 grid: {  
 left: '3%',  
 right: '4%',  
 bottom: '3%',  
 containLabel: true  
 },  
 xAxis : [  
 {  
 type : 'category',  
 data : ['60岁以下', '60--65岁', '65--70岁', '70--75岁',   
 '75--80岁', '80岁以上'],  
 axisTick: {  
 alignWithLabel: true  
 }  
 }  
 ],  
 yAxis : [  
 {  
 type : 'value'  
 }  
 ],  
 series : [  
 {  
 name:'人数',  
 type:'bar',  
 barWidth: '60%',  
 data:[10, 15, 23, 26, 12, 8]  
 }  
 ]  
};

myChart.setOption(option);

</script>

最终效果图如下所示：



# 工作人员模块增删改查

## 任务描述

**内 容**：工作人员模块的基本操作

**学 时**：2

**知识点**：数据的增加、删除、修改和查询

**重点**：

**难点**：

## 授课思路

基于Flask框架，系统的学习一个功能模块的增、删、改、查等功能。

## 任务指导

1. 新建工作人员实体类文件
2. 新建服务层业务处理文件
3. 新建工作人员列表页面
4. 增加左侧导航菜单
5. 列表页面和后台数据绑定
6. 新建工作人员编辑页面
7. 在列表页面添加跳转方法
8. 新建后台路由规则

## 任务实现

1. 新建工作人员实体类文件

在 app主目录下，新建mod\_employee文件夹，在mod\_employee文件夹下，新建models.py文件，代码如下：

class EmployeeInfo(db.Model):  
 \_\_tablename\_\_ = 'employee\_info'  
  
 id = db.Column(db.Integer, primary\_key=True)  
 ORG\_ID = db.Column(db.Integer)  
 CLIENT\_ID = db.Column(db.Integer)  
 username = db.Column(db.String(50))  
 gender = db.Column(db.String(5))  
 phone = db.Column(db.String(50))  
 id\_card = db.Column(db.String(50))  
 birthday = db.Column(db.DateTime,nullable=True)  
 hire\_date = db.Column(db.DateTime,nullable=True)  
 resign\_date = db.Column(db.DateTime,nullable=True)  
 imgset\_dir = db.Column(db.String(200))  
 profile\_photo = db.Column(db.String(200))  
 DESCRIPTION = db.Column(db.String(200))  
 ISACTIVE = db.Column(db.String(10))  
 CREATED = db.Column(db.DateTime)  
 CREATEBY = db.Column(db.Integer)  
 UPDATED = db.Column(db.DateTime)  
 UPDATEBY = db.Column(db.Integer)  
 REMOVE = db.Column(db.String(1))

具体字段说明，请参考“数据库文档设计.docx”

1. 新建服务层业务处理文件

在 app\mod\_employee文件夹下，新建controllers.py文件，代码如下：

def get\_all\_data():  
 return EmployeeInfo.query.all()  
  
def select\_by\_id(id):  
 return EmployeeInfo.query.filter\_by(id=id).first()  
  
def delete\_by\_id(id):  
 record = EmployeeInfo.query.filter\_by(id=id).first()  
 db.session.delete(record)  
 db.session.commit()

代码中包括了添加数据，编辑数据，删除数据和查询等常见的数据库操作方法。

1. 新建工作人员列表页面

在app\templates文件夹下，新建oldperson目录，在这个目录下新建employeeinfolist.html文件:

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个工作人员信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">工作人员模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/employeeinfolist">  
 <span class="sub-menu-text">工作人员信息列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

在 app\mod\_employee文件夹下，新建forms.py文件：

@app.route('/employeeinfolist')  
@login\_required  
def list\_all\_employeeinfo():  
 \_listdata = c.get\_all\_data()  
 return render\_template("oldperson/employeeinfolist.html",  
 listdata= \_listdata)

* 导入服务层的数据库操作方法

Import app.mod\_employee.controllers as c

* 定义路由

@app.route('/employeeinfolist')

通过@app装饰器为函数list\_all\_employeeoninfo定义了一条路由，所有以GET方式匹配到/employeeoninfolist的请求都由list\_all\_employeeoninfo函数处理；

* 返回模板文件

从数据库取出所有数据后，返回到list页面文件，并把数据回传给页面

return render\_template("oldperson/employeeinfolist.html",  
 listdata= \_listdata)

最终效果图如下所示：



1. 新建工作人员编辑页面

在app\templates\oldperson\目录下，新建employeeinfoform.html页面，

开发方法与任务5里的功能类似

1. 在列表页面添加跳转方法

打开app\templates\oldperson目录下的employeeinfolist.html,在列表代码中添加如下Javascript代码：

<a href="#" onclick="javascript:selectdata({{data.id}})">编辑</a>

1. 新建后台路由规则

打开app\mod\_employee目录下的forms.py文件，添加对应/saveemployeeinfo的一条路由：

@app.route('/saveemployeeinfo',methods=['POST'])

def save\_employeeinfo():

……  
return  
render\_template("oldperson/employeeinfolist.html", listdata=listdata)

* 定义路由

@app.route('/saveemployeeinfo', methods=[ 'POST'])

通过@app装饰器为函数save\_employeeinfo定义了一条路由，所有以POST方式提交到/saveemployeeinfo的请求都由save\_employeeinfo函数处理；

* 使用request.form方法接收表单数据。

record.id = int(request.form['record\_id'])

record.username = request.form['username']

……

* 更新数据库

c.update\_insert\_data(record)  
listdata = c.get\_all\_data()

# 工作人员模块头像管理

## 任务描述

**内 容**：管理工作人员的头像数据信息

**学 时**：2

**知识点**：websocket数据推送，web页面调用python脚本

**重点**：

**难点**：

## 授课思路

功能实现同任务6老年人模块头像管理一样。

## 任务指导

1. 新建工作人员头像信息列表页面
2. 增加左侧导航菜单
3. 列表页面和后台数据绑定
4. 从Open CV获取工作人员头像
5. 把图片信息推送到web页面

## 任务实现

1. 新建工作人员头像信息列表页面

在app\templates\oldperson目录下，新建employeeimagelist.html文件:

<div id="imagelistid" class="row">  
  
 {% for image in imagelist %}  
  
 <div class="col-md-3 isotope-item">  
 <div class="filter-content">  
 <img src="img/employee/{{selectdata.id}}/{{image}}"  
 alt="" class="img-responsive">  
 <div class="hover-content">  
 <h4>操作</h4>  
 <a class="btn btn-success hover-link"  
 href="#" onclick="setImage('{{image}}')" title="设为头像">  
 <i class="fa fa-edit fa-1x"></i>  
 </a>  
 <a class="btncolorbox-button"  
 href="img/employee/{{selectdata.id}}/{{image}}"   
 title="点击放大">  
 <i class="fa fa-search-plus fa-1x"></i>  
 </a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
  
</div>

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个工作人员头像信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">工作人员模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/employeeimagelist">  
 <span class="sub-menu-text">工作人员头像列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

打开app\mod\_employee文件夹下的forms.py文件，增加一条路由规则：

@app.route('/employeeimagelist')  
def list\_all\_employeeimage():  
 \_id = session.get('employeeid')  
 selectdata = c.select\_by\_id(\_id)  
  
 \_image\_dir = app.static\_folder+'/img/employee'+'/'+str(\_id)  
 \_images = []  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
 else:  
 # list all the image of this folder  
 \_images = os.listdir(\_image\_dir)  
 return render\_template("oldperson/employeeimagelist.html",  
 selectdata=selectdata, imagelist=\_images)

上述代码的作用就是遍历所选工作人员所在的文件夹下的所有头像文件，并把数据回传给前端页面。

1. 从Open CV获取工作人员头像

这个任务的功能是调用后台OpenCV的脚本，启动一个摄像头，为工作人员拍照，把各种表情的照片存入特定的文件夹，同时利用websocket技术推送到web页面上，进行实时展示。

* 首先在工作人员信息列表页面增加获取头像的入口代码，方法是打开 app\templates\oldperson目录下的employeeinfolist.html,在列表代码里添加一个导航规则：

<li>  
 <a href="/setemployeeid/{{data.id}}" >获取头像</a>  
</li>

* 打开app\mod\_employee目录下的forms.py文件，增加一条路由规则：

@app.route('/setemployeeid/<int:id>')  
def set\_id\_session(id):  
 session['employeeid'] = id  
 selectdata = c.select\_by\_id(id)  
  
 \_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
 os.popen(\_cmd)  
  
 #check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/employee' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()  
  
 return redirect('/employeeimagelist')

上述代码首先接收页面传递过来的工作人员编号ID，然后启动OpenCV的拍照脚本：

\_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
os.popen(\_cmd)

接下来的代码放到了一个多线程里运行，作用是监测特定文件夹下有没有文件变化，如果有新的图片文件，就直接推送到web页面上：

#check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/employee' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()

1. 把图片信息推送到web页面

利用websoccket技术，把后台得到的头像信息实时推送到web端展示，后台推送代码如下：

def run\_file\_monitor(\*\*name):  
   
 path\_to\_watch = r''+str(name['data'].imgset\_dir)+'/'+str(name['data'].id)  
 before = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
   
 while 1:  
 time.sleep(2)  
 after = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
 added = [f for f in after if not f in before]  
 removed = [f for f in before if not f in after]  
 if added:  
 print("Added: ", added)  
 socketio.emit('image\_data',{'id':   
 name['data'].id,'image':added[0]},   
 namespace='/employeeimage')  
 if removed:  
 print("Removed: ", ", ".join(removed))  
 before = after

使用socketio框架实现信息推送，前台页面需要设置接收代码，打开app\templates\oldperson目录下的employeeimagelist.html,

<script src="js/websocket/socket.io.js"></script>

<script>  
  
var socket;

function connet\_ws()  
{  
 namespace = '/employeeimage'  
 socket = io.connect(location.protocol + '//' + document.domain + ':' + location.port + namespace,{'reconnection delay': 2000});  
  
  
  
 socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });  
}

……

首先引用socket.io.js框架：

<script src="js/websocket/socket.io.js"></script>

和后台ws服务建立连接，并建立消息监听回调函数：

function connet\_ws()

……  
socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });

当有消息到达时，调用addDiv脚本，把图片信息展示在页面上：

function addDiv(id,image)  
{  
  
 $("#imagelistid").prepend("<div class='col-md-3 isotope-item'>"+  
 "<div class='filter-content'>"+  
 "<img src='img/employee/"+id+"/"+image+  
 "'"+" class='img-responsive' />"+  
 "<div class='hover-content'>"+  
 "<h4>操作</h4>"+  
 "<a class='btn btn-success hover-link'   
 href='#' title='设为头像'>"+  
 "<i class='fa fa-edit fa-1x'></i>"+  
 "</a>"+  
 "<a class='btn colorbox-button'   
 href='img/employee/"+id+"/"+image+"'"+"   
 title='点击放大'>"+  
 "<i class='fa fa-search-plus fa-1x'></i>"+  
 "</a></div></div>"+  
 "</div>");  
  
}

# 工作人员模块报表统计

## 任务描述

**内 容**：工作人员模块数据统计功能

**学 时**：2

**知识点**：ECharts报表设计

**重点**：

**难点**：

## 授课思路

同任务7老年人模块报表统计的功能一样

## 任务指导

1. 新建工作人员数据统计页面
2. 增加页面导航并添加路由规则
3. 引入ECharts
4. 使用ECharts展现数据

## 任务实现

1. 新建工作人员数据统计页面

在app\templates\oldperson目录下，新建employeestatistic.html页面：

<div class="box-title">  
 <h4><i class="fa fa-signal"></i>离/入职对比图</h4>  
   
 </div>  
 <div class="box-body" id="main" style="width: 500px;height:350px;">  
  
 </div>  
 </div>  
 <!-- /BOX -->  
</div>

1. 增加页面导航并设置路由规则

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个工作人员统计分析的菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">工作人员模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/employeestatistic">  
 <span class="sub-menu-text">工作人员统计分析</span></a></li>  
 </ul>  
</li>

打开app\mod\_oldperson\目录下的forms.py文件，添加一条路由规则：

@app.route('/employeestatistic')  
def run\_employee\_statistic():  
 return render\_template("oldperson/employeestatistic.html")

1. 引入ECharts

从ECharts官网下载JS文件，

官网地址是：https://echarts.baidu.com/download.html

下载源文件压缩包，把echarts.min.js文件复制出来放到static\js\echarts目录下，编辑app\templates\oldperson目录下的employeestatistic.html，引入JS文件：

<script type="text/javascript" src="js/echarts/echarts.min.js"></script>

1. 使用ECharts展示数据

打开app\templates\oldperson目录下的employeestatistic.html文件，添加JS脚本：

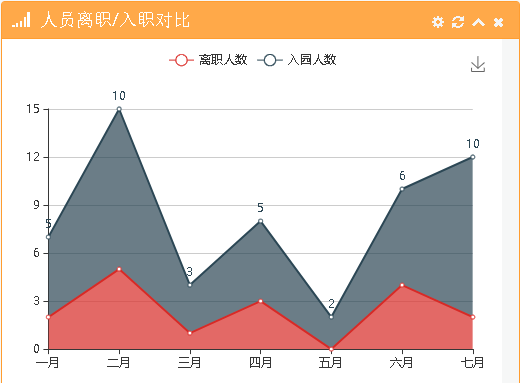
<script>  
 // 基于准备好的dom，初始化echarts实例  
 var myChart = echarts.init(document.getElementById('main'));  
  
 // 指定图表的配置项和数据  
 option = {  
 title: {  
 text: ''  
 },  
 tooltip : {  
 trigger: 'axis',  
 axisPointer: {  
 type: 'cross',  
 label: {  
 backgroundColor: '#6a7985'  
 }  
 }  
 },  
 legend: {  
 data:['离职人数','入园人数']  
 },  
 toolbox: {  
 feature: {  
 saveAsImage: {}  
 }  
 },  
 grid: {  
 left: '3%',  
 right: '4%',  
 bottom: '3%',  
 containLabel: true  
 },  
 xAxis : [  
 {  
 type : 'category',  
 boundaryGap : false,  
 data : ['一月','二月','三月','四月','五月','六月','七月']  
 }  
 ],  
 yAxis : [  
 {  
 type : 'value'  
 }  
 ],  
 series : [  
 {  
 name:'离职人数',  
 type:'line',  
 stack: '总量',  
 areaStyle: {normal: {}},  
 data:[2, 5, 1, 3, 0, 4, 2]  
 },  
 {  
 name:'入园人数',  
 type:'line',  
 stack: '总量',  
 label: {  
 normal: {  
 show: true,  
 position: 'top'  
 }  
 },  
 areaStyle: {normal: {}},  
 data:[5, 10, 3, 5, 2, 6, 10]  
 }  
 ]  
};

;

myChart.setOption(option);

</script>

最终效果图如下所示：



# 义工信息模块增删改查

## 任务描述

**内 容**：义工信息模块的基本操作

**学 时**：2

**知识点**：数据的增加、删除、修改和查询

**重点**：

**难点**：

## 授课思路

基于Flask框架，系统的学习一个功能模块的增、删、改、查等功能。

## 任务指导

1. 新建义工实体类文件
2. 新建服务层业务处理文件
3. 新建义工列表页面
4. 增加左侧导航菜单
5. 列表页面和后台数据绑定
6. 新建义工编辑页面
7. 在列表页面添加跳转方法
8. 新建后台路由规则

## 任务实现

1. 新建义工实体类文件

在 app主目录下，新建mod\_volunteer文件夹，在mod\_volunteer文件夹下，新建models.py文件，代码如下：

class VolunteerInfo(db.Model):  
 \_\_tablename\_\_ = 'volunteer\_info'  
  
 id = db.Column(db.Integer, primary\_key=True)  
 ORG\_ID = db.Column(db.Integer)  
 CLIENT\_ID = db.Column(db.Integer)  
 name = db.Column(db.String(50))  
 gender = db.Column(db.String(5))  
 phone = db.Column(db.String(50))  
 id\_card = db.Column(db.String(50))  
 birthday = db.Column(db.DateTime,nullable=True)  
 checkin\_date = db.Column(db.DateTime,nullable=True)  
 checkout\_date = db.Column(db.DateTime,nullable=True)  
 imgset\_dir = db.Column(db.String(200))  
 profile\_photo = db.Column(db.String(200))  
 DESCRIPTION = db.Column(db.String(200))  
 ISACTIVE = db.Column(db.String(10))  
 CREATED = db.Column(db.DateTime)  
 CREATEBY = db.Column(db.Integer)  
 UPDATED = db.Column(db.DateTime)  
 UPDATEBY = db.Column(db.Integer)  
 REMOVE = db.Column(db.String(1))

具体字段说明，请参考“数据库文档设计.docx”

1. 新建服务层业务处理文件

在 app\mod\_volunteer文件夹下，新建controllers.py文件，代码如下：

def get\_all\_data():  
 return VolunteerInfo.query.all()  
  
def select\_by\_id(id):  
 return VolunteerInfo.query.filter\_by(id=id).first()  
  
def delete\_by\_id(id):  
 record = VolunteerInfo.query.filter\_by(id=id).first()  
 db.session.delete(record)  
 db.session.commit()

代码中包括了添加数据，编辑数据，删除数据和查询等常见的数据库操作方法。

1. 新建工作人员列表页面

在app\templates\oldperson目录下新建volunteerinfolist.html文件:

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个义工信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">义工信息模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/volunteerinfolist">  
 <span class="sub-menu-text">义工信息列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

在 app\mod\_volunteer文件夹下，新建forms.py文件：

@app.route('/volunteerinfolist')  
@login\_required  
def list\_all\_volunteerinfo():  
 \_listdata = c.get\_all\_data()  
 return render\_template("oldperson/volunteerinfolist.html",  
 listdata= \_listdata)

* 导入服务层的数据库操作方法

Import app.mod\_volunteer.controllers as c

* 定义路由

@app.route('/volunteerinfolist')

通过@app装饰器为函数list\_all\_volunteerinfo定义了一条路由，所有以GET方式匹配到/volunteerinfolist的请求都由list\_all\_volunteerinfo函数处理；

* 返回模板文件

从数据库取出所有数据后，返回到list页面文件，并把数据回传给页面

return render\_template("oldperson/volunteerinfolist.html",  
 listdata= \_listdata)

最终效果图如下所示：



1. 新建义工编辑页面

在app\templates\oldperson\目录下，新建volunteerinfoform.html页面，

开发方法与任务5里的功能类似

1. 在列表页面添加跳转方法

打开app\templates\oldperson目录下的volunteerinfolist.html,在列表代码中添加如下Javascript代码：

<a href="#" onclick="javascript:selectdata({{data.id}})">编辑</a>

1. 新建后台路由规则

打开app\mod\_volunteer目录下的forms.py文件，添加对应/savevolunteerinfo的一条路由：

@app.route('/savevolunteerinfo',methods=['POST'])

def save\_volunteerinfo():

……  
return  
render\_template("oldperson/volunteerinfolist.html", listdata=listdata)

* 定义路由

@app.route('/savevolunteerinfo', methods=[ 'POST'])

通过@app装饰器为函数save\_volunteerinfo定义了一条路由，所有以POST方式提交到/savevolunteerinfo的请求都由save\_volunteerinfo函数处理；

* 使用request.form方法接收表单数据。

record.id = int(request.form['record\_id'])

record.username = request.form['username']

……

* 更新数据库

c.update\_insert\_data(record)  
listdata = c.get\_all\_data()

# 义工模块头像管理

## 任务描述

**内 容**：管理义工的头像数据信息

**学 时**：2

**知识点**：websocket数据推送，web页面调用python脚本

**重点**：

**难点**：

## 授课思路

功能实现同任务6老年人模块头像管理一样。

## 任务指导

1. 新建义工头像信息列表页面
2. 增加左侧导航菜单
3. 列表页面和后台数据绑定
4. 从Open CV获取义工头像
5. 把图片信息推送到web页面

## 任务实现

1. 新建义工头像信息列表页面

在app\templates\oldperson目录下，新建volunteerimagelist.html文件:

<div id="imagelistid" class="row">  
  
 {% for image in imagelist %}  
  
 <div class="col-md-3 isotope-item">  
 <div class="filter-content">  
 <img src="img/volunteer/{{selectdata.id}}/{{image}}"  
 alt="" class="img-responsive">  
 <div class="hover-content">  
 <h4>操作</h4>  
 <a class="btn btn-success hover-link"  
 href="#" onclick="setImage('{{image}}')" title="设为头像">  
 <i class="fa fa-edit fa-1x"></i>  
 </a>  
 <a class="btncolorbox-button"  
 href="img/volunteer/{{selectdata.id}}/{{image}}"   
 title="点击放大">  
 <i class="fa fa-search-plus fa-1x"></i>  
 </a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
  
</div>

1. 增加左侧导航菜单

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个义工头像信息列表菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">义工模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/volunteerimagelist">  
 <span class="sub-menu-text">义工头像列表</span></a></li>  
 </ul>  
</li>

1. 列表页面和后台数据绑定

打开app\mod\_volunteer文件夹下的forms.py文件，增加一条路由规则：

@app.route('/volunteerimagelist')  
def list\_all\_volunteerimage():  
 \_id = session.get('volunteerid')  
 selectdata = c.select\_by\_id(\_id)  
  
 \_image\_dir = app.static\_folder+'/img/volunteer'+'/'+str(\_id)  
 \_images = []  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
 else:  
 # list all the image of this folder  
 \_images = os.listdir(\_image\_dir)  
 return render\_template("oldperson/volunteerimagelist.html",  
 selectdata=selectdata, imagelist=\_images)

上述代码的作用就是遍历所选义工所在的文件夹下的所有头像文件，并把数据回传给前端页面。

1. 从Open CV获取义工头像

这个任务的功能是调用后台OpenCV的脚本，启动一个摄像头，为义工拍照，把各种表情的照片存入特定的文件夹，同时利用websocket技术推送到web页面上，进行实时展示。

* 首先在义工信息列表页面增加获取头像的入口代码，方法是打开 app\templates\oldperson目录下的volunteerinfolist.html,在列表代码里添加一个导航规则：

<li>  
 <a href="/setvolunteerid/{{data.id}}" >获取头像</a>  
</li>

* 打开app\mod\_employee目录下的forms.py文件，增加一条路由规则：

@app.route('/setvolunteerid/<int:id>')  
def set\_id\_session(id):  
 session['volunteerid'] = id  
 selectdata = c.select\_by\_id(id)  
  
 \_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
 os.popen(\_cmd)  
  
 #check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/volunteer' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()  
  
 return redirect('/volunteerimagelist')

上述代码首先接收页面传递过来的义工编号ID，然后启动OpenCV的拍照脚本：

\_cmd = r'python d:\oldperson\app\mod\_gui\opencv.py '  
os.popen(\_cmd)

接下来的代码放到了一个多线程里运行，作用是监测特定文件夹下有没有文件变化，如果有新的图片文件，就直接推送到web页面上：

#check if the folder exits  
 \_image\_dir = app.static\_folder + '/img/volunteer' + '/' + str(id)  
 if not os.path.exists(\_image\_dir):  
 os.mkdir(\_image\_dir)  
  
 # start file monitor program, need run in a single thread  
 \_file\_monitor = threading.Thread(target=run\_file\_monitor,  
 kwargs={'data': selectdata})  
 \_file\_monitor.start()

1. 把图片信息推送到web页面

利用websoccket技术，把后台得到的头像信息实时推送到web端展示，后台推送代码如下：

def run\_file\_monitor(\*\*name):  
   
 path\_to\_watch = r''+str(name['data'].imgset\_dir)+'/'+str(name['data'].id)  
 before = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
   
 while 1:  
 time.sleep(2)  
 after = dict([(f, None) for f in os.listdir(path\_to\_watch)])  
 added = [f for f in after if not f in before]  
 removed = [f for f in before if not f in after]  
 if added:  
 print("Added: ", added)  
 socketio.emit('image\_data',{'id':   
 name['data'].id,'image':added[0]},   
 namespace='/volunteerimage')  
 if removed:  
 print("Removed: ", ", ".join(removed))  
 before = after

使用socketio框架实现信息推送，前台页面需要设置接收代码，打开app\templates\oldperson目录下的volunteerimagelist.html,

<script src="js/websocket/socket.io.js"></script>

<script>  
  
var socket;

function connet\_ws()  
{  
 namespace = '/volunteerimage'  
 socket = io.connect(location.protocol + '//' + document.domain + ':' + location.port + namespace,{'reconnection delay': 2000});  
  
  
  
 socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });  
}

……

首先引用socket.io.js框架：

<script src="js/websocket/socket.io.js"></script>

和后台ws服务建立连接，并建立消息监听回调函数：

function connet\_ws()

……  
socket.on('image\_data', function(msg) {   
 addDiv(msg.id,msg.image);  
 });

当有消息到达时，调用addDiv脚本，把图片信息展示在页面上：

function addDiv(id,image)  
{  
  
 $("#imagelistid").prepend("<div class='col-md-3 isotope-item'>"+  
 "<div class='filter-content'>"+  
 "<img src='img/volunteer/"+id+"/"+image+  
 "'"+" class='img-responsive' />"+  
 "<div class='hover-content'>"+  
 "<h4>操作</h4>"+  
 "<a class='btn btn-success hover-link'   
 href='#' title='设为头像'>"+  
 "<i class='fa fa-edit fa-1x'></i>"+  
 "</a>"+  
 "<a class='btn colorbox-button'   
 href='img/volunteer/"+id+"/"+image+"'"+"   
 title='点击放大'>"+  
 "<i class='fa fa-search-plus fa-1x'></i>"+  
 "</a></div></div>"+  
 "</div>");  
  
}

# 义工模块报表统计

## 任务描述

**内 容**：义工模块数据统计功能

**学 时**：2

**知识点**：ECharts报表设计

**重点**：

**难点**：

## 授课思路

同任务7老年人模块报表统计的功能一样

## 任务指导

1. 新建义工数据统计页面
2. 增加页面导航并添加路由规则
3. 引入ECharts
4. 使用ECharts展现数据

## 任务实现

1. 新建义工数据统计页面

在app\templates\oldperson目录下，新建volunteerstatistic.html页面：

<div class="box-title">  
 <h4><i class="fa fa-signal"></i>义工年龄分布图</h4>  
   
 </div>  
 <div class="box-body" id="main" style="width: 500px;height:350px;">  
  
 </div>  
 </div>  
 <!-- /BOX -->  
</div>

1. 增加页面导航并设置路由规则

在app\templates目录下， 打开leftsidebar.html页面文件，增加一个工作人员统计分析的菜单：

<li class="has-sub">  
 <a href="javascript:;" class="">  
 <i class="fa fa-table fa-fw"></i>   
 <span class="menu-text">义工模块</span>  
 <span class="arrow"></span>  
 </a>  
 <ul class="sub">  
 <li><a class="" href="/volunteerimagelist">  
 <span class="sub-menu-text">义工统计分析</span></a></li>  
 </ul>  
</li>

打开app\mod\_volunteer\目录下的forms.py文件，添加一条路由规则：

@app.route('/volunteerstatistic')  
def run\_volunteer\_statistic():  
 return render\_template("oldperson/volunteerstatistic.html")

1. 引入ECharts

从ECharts官网下载JS文件，

官网地址是：https://echarts.baidu.com/download.html

下载源文件压缩包，把echarts.min.js文件复制出来放到static\js\echarts目录下，编辑app\templates\oldperson目录下的volunteerstatistic.html，引入JS文件：

<script type="text/javascript" src="js/echarts/echarts.min.js"></script>

1. 使用ECharts展示数据

打开app\templates\oldperson目录下的volunteerstatistic.html文件，添加JS脚本：

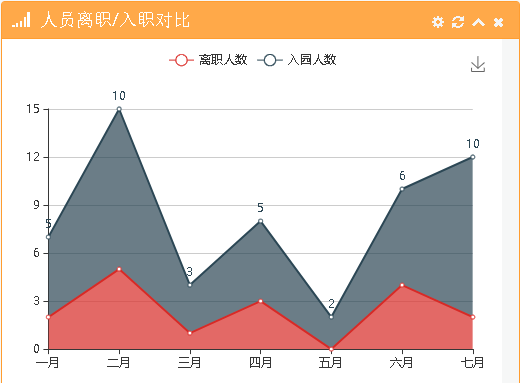
<script>  
 // 基于准备好的dom，初始化echarts实例  
 var myChart = echarts.init(document.getElementById('main'));  
  
 // 指定图表的配置项和数据  
 option = {  
 title: {  
 text: ''  
 },  
 tooltip : {  
 trigger: 'axis',  
 axisPointer: {  
 type: 'cross',  
 label: {  
 backgroundColor: '#6a7985'  
 }  
 }  
 },  
 legend: {  
 data:['离职人数','入园人数']  
 },  
 toolbox: {  
 feature: {  
 saveAsImage: {}  
 }  
 },  
 grid: {  
 left: '3%',  
 right: '4%',  
 bottom: '3%',  
 containLabel: true  
 },  
 xAxis : [  
 {  
 type : 'category',  
 boundaryGap : false,  
 data : ['一月','二月','三月','四月','五月','六月','七月']  
 }  
 ],  
 yAxis : [  
 {  
 type : 'value'  
 }  
 ],  
 series : [  
 {  
 name:'离职人数',  
 type:'line',  
 stack: '总量',  
 areaStyle: {normal: {}},  
 data:[2, 5, 1, 3, 0, 4, 2]  
 },  
 {  
 name:'入园人数',  
 type:'line',  
 stack: '总量',  
 label: {  
 normal: {  
 show: true,  
 position: 'top'  
 }  
 },  
 areaStyle: {normal: {}},  
 data:[5, 10, 3, 5, 2, 6, 10]  
 }  
 ]  
};

;

myChart.setOption(option);

</script>

最终效果图如下所示：



# 数据管理模块实时报表

## 任务描述

**内 容**：实时数据展示设计

**学 时**：2

**知识点**：OpenCV监测事件、API接口设计

**重点**：

**难点**：

## 授课思路