# 五色石考试系统开发手册

版本：V0.0.2

撰写人：申晏键

撰写时间：2024年3月21日

# 目录

[五色石考试系统用户手册 1](#_Toc161910003)

[技能栈 2](#_Toc161910004)

[前端 2](#_Toc161910005)

[后端 2](#_Toc161910006)

[运维部署 2](#_Toc161910007)

[运行架构 2](#_Toc161910008)

[前端 2](#_Toc161910009)

[后端 2](#_Toc161910010)

[数据库 3](#_Toc161910011)

[开发-后端 3](#_Toc161910012)

[后端连接数据库的改Hosts主机名技巧 3](#_Toc161910013)

[第三方库说明 3](#_Toc161910014)

[权限说明 4](#_Toc161910015)

[开发-前端 4](#_Toc161910016)

[开发环境测试 5](#_Toc161910017)

[生产环境测试 5](#_Toc161910018)

[部署时 5](#_Toc161910019)

[docker-compose暴露端口 6](#_Toc161910020)

[license的cron任务手动启动 8](#_Toc161910021)

[部署过程 9](#_Toc161910022)

[数据库创建过程 10](#_Toc161910023)

[1. 建库 10](#_Toc161910024)

[2. 建表（或更新表结构） 11](#_Toc161910025)

[3. 从无到有初始化数据库 12](#_Toc161910026)

[postman新建超管并登录 12](#_Toc161910027)

[初始化用户菜单 13](#_Toc161910028)

[初始化系统设置默认数据 14](#_Toc161910029)

[初始化建库成功，可正常访问 16](#_Toc161910030)

# 技能栈

## 前端

vite构建VUE项目，大量使用ElementPlus等组件。

## 后端

使用python+Django+ORM，使用了一些django相关组件

## 运维部署

Linux、docker-compose

# 运行架构

具体参数参考docker-compose.yaml、Dockerfile中参数细节。

总共分三个容器：

## 前端

Nginx反向代理

1. 静态资源文件，如vue打包构建的dist文件、文档下载、视频播放。
2. 转发/api/相关的请求路径给es\_backend。

## 后端

uwsgi启动Django项目，接受前端Nginx传过来的http协议的socket数据。

## 数据库

mysql数据库的目录配置外挂于服务器主机路径配置，更安全，不会因为容器运行不稳定丢失数据。此外后端的volumn目录和数据库的数据目录的volumn是同一挂载。

# 开发-后端

## 后端连接数据库的改Hosts主机名技巧

内部使用docker-network网络，因此命名容器名后，互相之间可以直接以容器名为host主机名，直接访问，参考django项目连接数据库的命名。

电脑萤幕的截图

描述已自动生成

（技巧：因为部署时，其连接es\_mysql的Host，如果本机开发测试，推荐hosts劫持本机的127.0.0.1为es\_mysql，可以省下在开发、部署时该连接mysql的Host名）

## 第三方库说明

如 requirements.txt 部分建议手动更新，为了docker打镜像时，体积更小，版本指定更准确，建议避免直接引入 如 numpy pandas等过于大的第三方包，导致臃肿。

电脑屏幕截图

描述已自动生成

另外，django、crosheaders、rest\_framework等第三方库直接平铺下，是因为方便第三方库代码调试、不同程度的小规模更改。

例如，django作为十几年前出现的web框架，部分自定义程度并没官方描述那么高，处于业务需求我直接进入改了一部分代码。

## 权限说明

鉴权是个开发时间很长的事情，我一度使用guardian来进行“对象级”鉴权颗粒度，但是事后发现，无论是开发时间还是代码可阅读性看，时间成本较为高昂。

遂在开发中期决定舍弃使用guardian组件，鉴权部分，我推荐自建表，将用户、目标对象做外键映射的关联即可。

# 开发-前端

## 开发环境测试

开发环境测试时，不需要注释此行，即本地让VUE的baseURL就是 <http://127.0.0.1:8000/>。

npm run dev即可

文本

描述已自动生成

## 生产环境测试

生成环境打包时，需要注释此行，即本地让VUE的baseURL空。

文本

描述已自动生成

npm run build即可打包到dist目录。

# 部署时

将docker容器需要的文件上传即可，这样减小docker体积，没必要的数据不需要上传。

在linux上新建`exam\_system`

图形用户界面, 应用程序

描述已自动生成

具体上传的内容部分，参考考本的交付的exam\_system.tar.gz的压缩包部分即可。

## docker-compose暴露端口

可以指定暴露es\_front的80端口，为了减小安全风险，其他容器的ports部分可以注释，参考如下：

（此处如mysql、backend的ports开启是为了调试而已）

version: '3'  
  
services:  
 # 后端  
 backend:  
 build: ./backend  
 container\_name: es\_backend  
 hostname: es\_backend  
 # 同 docker-network 内部通信不需要开放端口，其Dockerfile也不需要EXPOSE端口，减小非必要暴露  
 ports:  
 - 8000:8000  
 networks:  
 - es  
 volumes:  
 - ./mysql/volume:/app/volume/  
 restart: always  
 depends\_on:  
 - db  
  
 # 前端  
 front:  
 build: ./front  
 container\_name: es\_front  
 hostname: es\_front  
 ports:  
 - "80:80"  
 networks:  
 - es  
 volumes:  
 - ./mysql/volume:/mnt  
 restart: always  
  
 # 数据库  
 db:  
 image: mysql:5.6  
 container\_name: es\_mysql  
 hostname: es\_mysql  
 environment:  
 - MYSQL\_ROOT\_PASSWORD=1qaz@WSX  
 - TZ=Asia/Shanghai  
 - SET\_CONTAINER\_TIMEZONE=true  
 - CONTAINER\_TIMEZONE=Asia/Shanghai  
 # docker network内部通信 不需要暴露，减小非必要暴露  
 ports:  
 - 3306:3306  
 volumes:  
 - ./mysql/conf:/etc/mysql/conf.d  
 - ./mysql/conf:/etc/mysql/mysql.conf.d  
 # 此mysql的docker数据目录在 /usr/data  
# - ./mysql/data:/var/lib/mysql  
 - ./mysql/data:/usr/data  
 - ./mysql/logs:/var/log/mysql  
 - /etc/localtime:/etc/localtime:ro  
 # 建库，容器每次生成默认触发此init.sql脚本  
 - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql  
 networks:  
 - es  
 restart: always  
  
networks:  
 es:  
 driver: bridge

## license的cron任务手动启动

因为需要校验license和防破解，因此使用如下代码文件，利用cron自动添加每小时计算。license的有效期校验根据这 每小时增加一个小时数 来做时间校验。

文本

描述已自动生成

目前在es\_backend的Dockerfile中，我写了cron定时任务，

即图中的RUN部分的cron定时任务，在该debian的docker系统未生效，原因暂未知，我暂未细究，但是可以当作我方部署时的一个trick，部署时需要手动启动此cron。

文本

描述已自动生成

### 部署过程

docker exec -it es\_backend bash

crontab -e（因为在docker容器中执行bash，moba可能无法复制，推荐手动输入，因为手动输入，暂且按 \* \* \* \* \* 来看看每分钟是否执行本命令）

```vim

\* \* \* \* \* cd /app/scripts/;/usr/local/bin/python /app/scripts/timer\_add.py >> /var/log/timer.log 2>&1

```

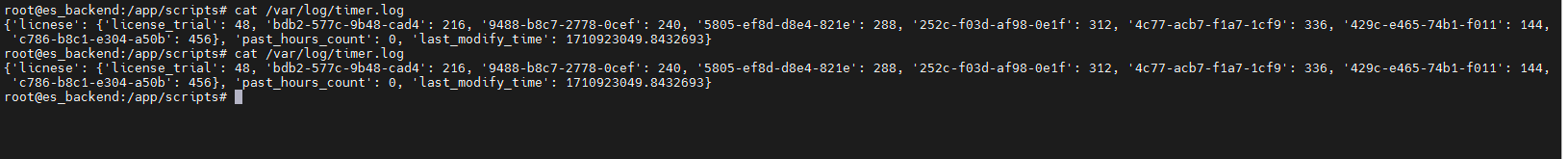
/etc/init.d/cron start && /etc/init.d/cron status

出现如下说明cron运行没有问题

图形用户界面, 文本, 应用程序

描述已自动生成

静等一分钟后，cat /var/log/timer.log ，看到正确的dict数据打印，无报错，说明无问题



再次crontab -e

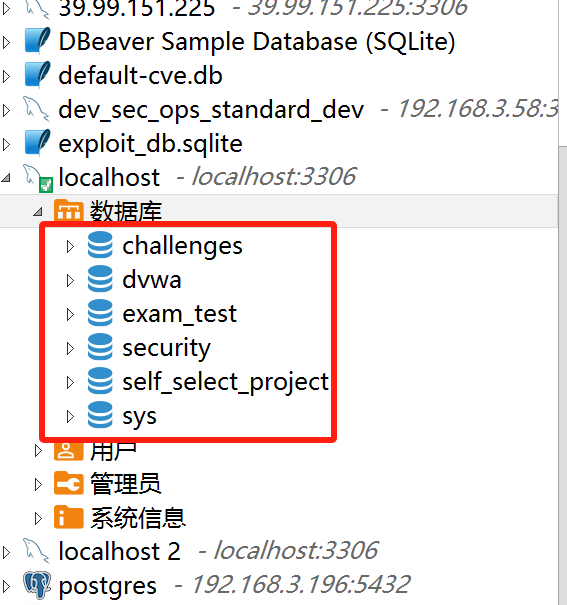
将 \* \* \* \* \* 改为 10 \* \* \* \* 即每个小时运行即可

# 数据库创建过程

## 建库

如果没有exam\_system数据库，则需要重新建立，过程如下：

目前我的mysql无exam\_system数据库



运行mysql命令

# 防止编码问题，指定为 utf8

**create** **database** exam\_system **default** charset = utf8;

创建exam\_system数据库，如下

图形用户界面

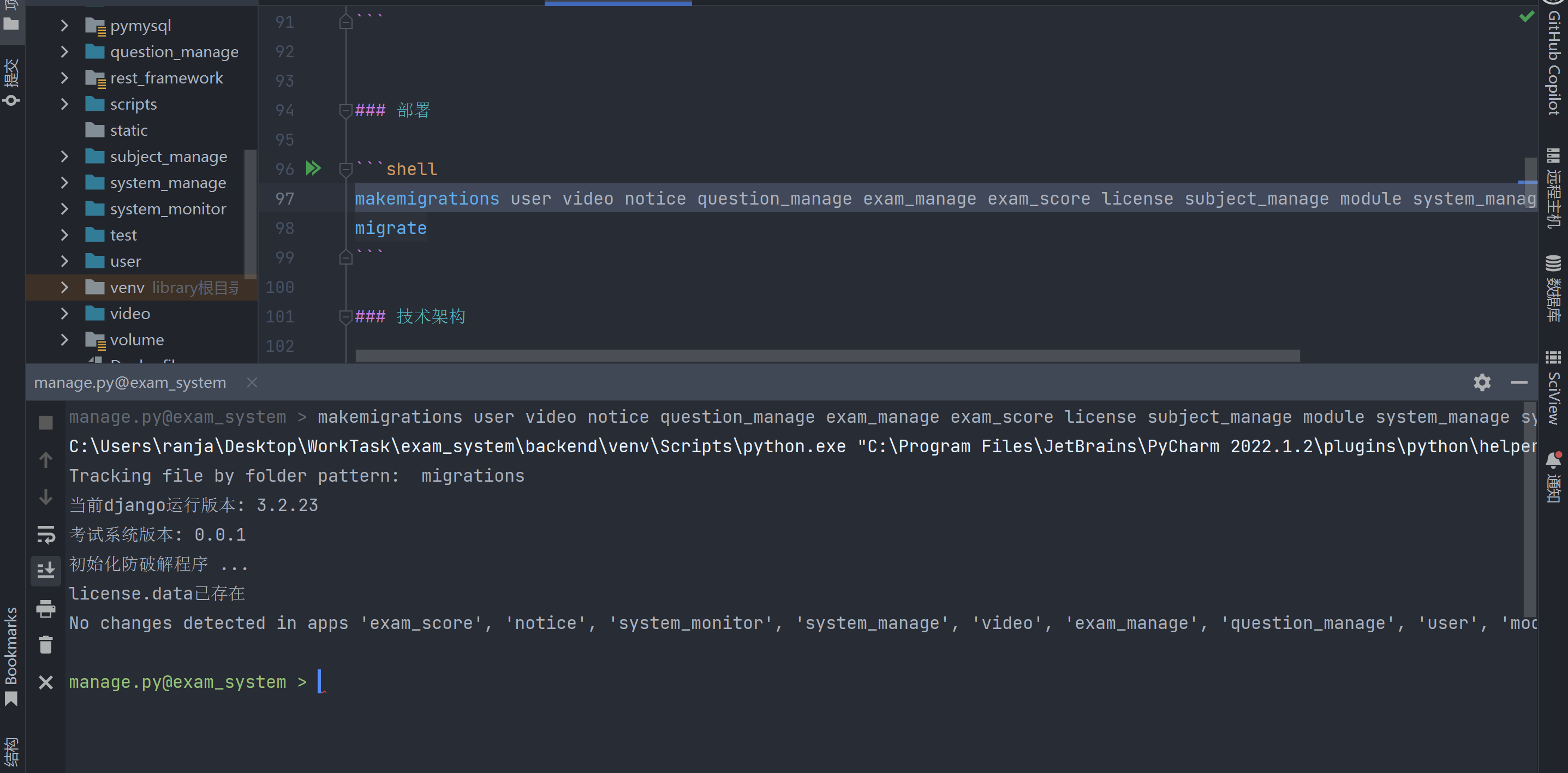
低可信度描述已自动生成

## 建表（或更新表结构）

因为我们使用的Django作为后端，建表使用Django的manage脚本命令进行。

命令可参考backend目录的readme，获取参考命令。

```shell  
python manage.py makemigrations user video notice question\_manage exam\_manage exam\_score license subject\_manage module system\_manage system\_monitor ftp  
python manage.py migrate  
```



文本

描述已自动生成

接下来数据库建表完成，如下

图形用户界面, 应用程序

描述已自动生成

## 从无到有初始化数据库

因为本步骤是从无到有建库建表，因此需要一些解放放入初始化数据，初始化数据直接通过一些接口就可以放入关键数据，存在安全风险，因此我推荐在给用户部署数据库时直接使用已存在初始化数据的数据库来部署，并将此初始化默认数据的步骤在未来需要更进一步移除。

### postman新建超管并登录

postman接口数据，如下

图形用户界面, 文本

描述已自动生成

使用002 admin 的账密可以登录该平台系统。

电脑萤幕的截图

描述已自动生成

### 初始化用户菜单

接口如下。将文件以form-data形式传输入该接口

电脑萤幕的截图

描述已自动生成

### 初始化系统设置默认数据

该数据，无论是新建，还是使用，仅使用id=1的数据条目。因此在该restfull接口，只关注id=1的数据内容。

需要初始化系统默认数据，接口如下

图形用户界面, 文本

描述已自动生成

因为是添加的第一条数据，因此直接访问如下接口确认是否是id=1的数据条目：

图形用户界面, 文本

描述已自动生成

如果要更改此条目，参考如下的postman数据，其实在用户正常使用过程中，修改的都是id=1的数据。

屏幕的截图

描述已自动生成

### 初始化建库成功，可正常访问

图形用户界面

描述已自动生成

图形用户界面, 应用程序

描述已自动生成