

Django项目框架

1.Django框架介绍与安装

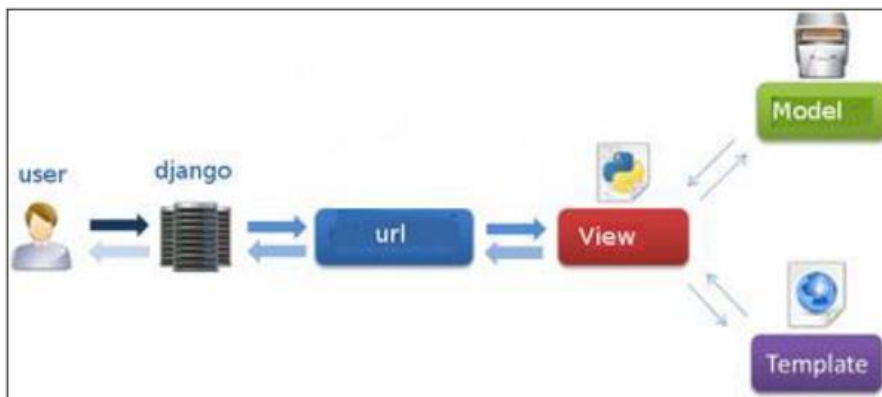
1.1 django介绍

Django发布于2003年，是目前Python语言影响力最高和最成熟的网络框架。最初Django是用于制作在线新闻系统的Web站点，之后被大众所接受并广泛使用。



经过多年开发积累，Django已经越发成熟，适应开发者的需求。目前开发者使用Django Web框架可以在几个小时内将一个Web应用程序的设计落地为一个真实的系统。在Django中集成了众多功能强大的模块，使用这些模块可以处理众多Web开发的麻烦，这也是Django相较于其他Python Web框架的优势。开发者在使用Django作Web开发时只需要专注于编写应用程序，不需要再自行设计新的功能与工具，借助Django可以完成绝大多数Web开发的工作。并且Django是一个免费、开源的Web框架，开发者可以基于Django进行适合业务的二次开发。

django的逻辑关系非常清晰



1.2 选择Django的理由：

1.功能完善、要素齐全：该有的、可以没有的都有，自带大量常用工具和框架，无须你自定义、组合、增删及修改。**2.完善的文档：**经过十多年的发展和完善，Django有广泛的实践案例和完善的在线文档。开发者遇到问题时可以搜索在线文档寻求解决方案。**3.强大的数据库访问组件：**Django的Model层自带数据库ORM组件，使得开发者无须学习其他数据库访问技术（SQL、pymysql、SQLAlchemy等）。**4.灵活的URL映射：**Django使用正则表达式管理URL映射，灵活性高。新版的2.0，进一步提高了URL编写的优雅性。**5.丰富的Template模板语言：**类似jinja模板语言，不但原生功能丰富，还可以自定义模板标签，并且与其ORM的用法非常相似。**6.自带后台管理系统admin：**只需要通过简单的几行配置和代码就可以实现一个完整的后台数据管理控制平台。**7.完整的错误信息提示：**在开发调试过程中如果出现运行错误或者异常，Django可以提供非常完整的错误信息帮助定位问题。

1.3 Django框架安装

安装语法：

```
pip install django
```

结果如下图：



```
Terminal: Local +
(c) Microsoft Corporation. 保留所有权利。

(python_best) C:\Users\ASUS\Desktop\pro_code\project>pip install django
Collecting django
  Using cached https://files.pythonhosted.org/packages/70/22/ed1943c0ef2be99ade872f49a20aa4cfc70eb4ffc866fc61f-py3-none-any.whl
Collecting asgiref<4,>=3.3.2 (from django)
  Using cached https://files.pythonhosted.org/packages/17/8b/05e225d11154b8f5358e6a6d277679c9741ec0339d1e451c4-py3-none-any.whl
Collecting pytz (from django)
  Using cached https://files.pythonhosted.org/packages/70/94/784178ca5dd892a98f113cdd923372024dc04b8d40abe77c-py2.py3-none-any.whl
Collecting sqlparse>=0.2.2 (from django)
  Using cached https://files.pythonhosted.org/packages/14/05/6e8eb62ca685b10e34051a80d7ea94b7137369d8c0be5c3b1-py3-none-any.whl
Collecting typing-extensions; python_version < "3.8" (from asgiref<4,>=3.3.2->django)
  Using cached https://files.pythonhosted.org/packages/2e/35/6c4fff5ab443b57116cblaad46421fb719bed2825664e8fe-sions-3.10.0.0-py3-none-any.whl
Installing collected packages: typing-extensions, asgiref, pytz, sqlparse, django
Successfully installed asgiref-3.3.4 django-3.2.4 pytz-2021.1 sqlparse-0.4.1 typing-extensions-3.10.0.0
```

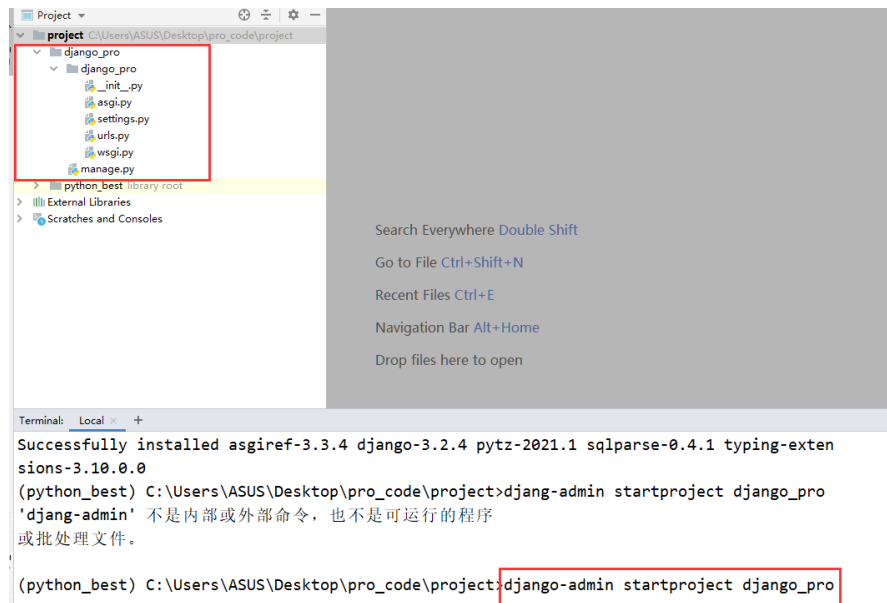
2.实现第一个Django项目

2.1创建项目

创建语法：

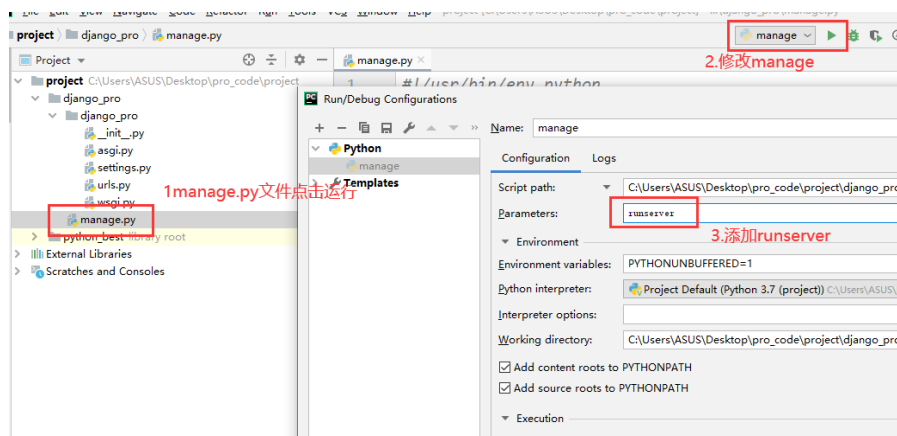
```
django-admin startproject django_pro
```

执行方法如下图。

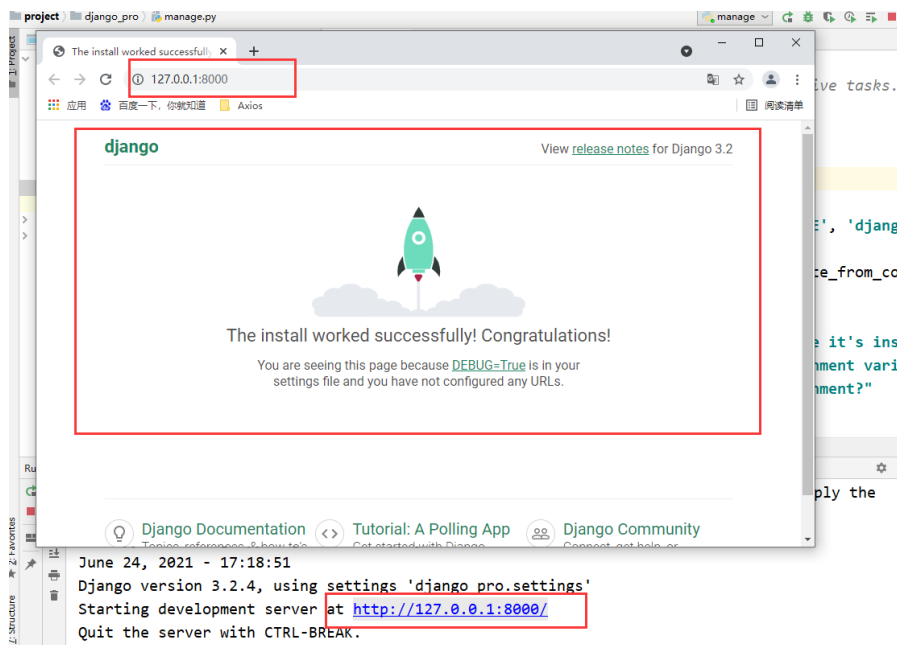


2.2启动项目

启动方法：

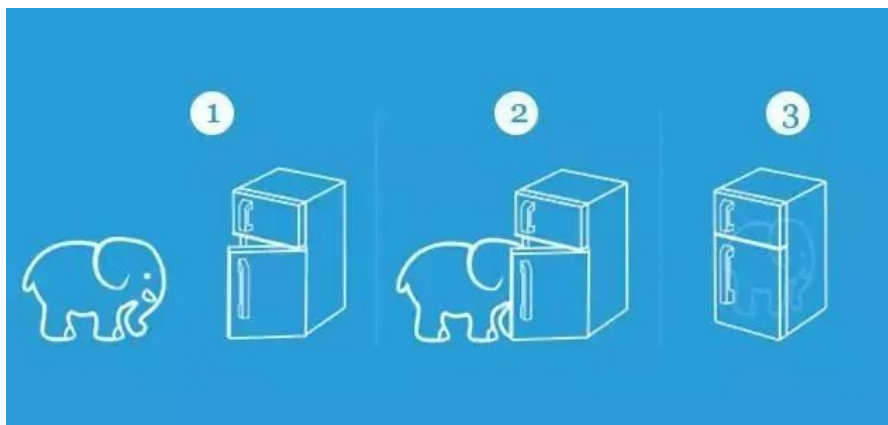


效果如下图：



2.3 创建应用

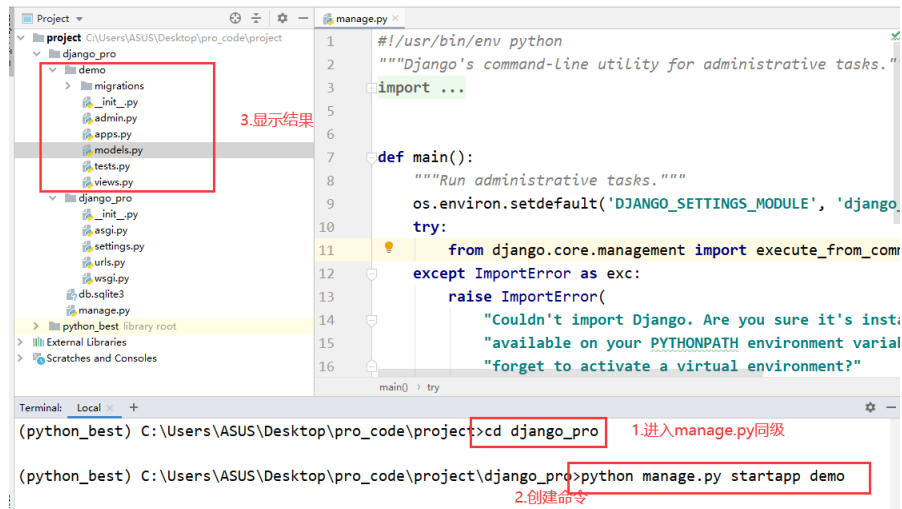
把大象放冰箱



创建语法：

```
python manage.py startapp demo
```

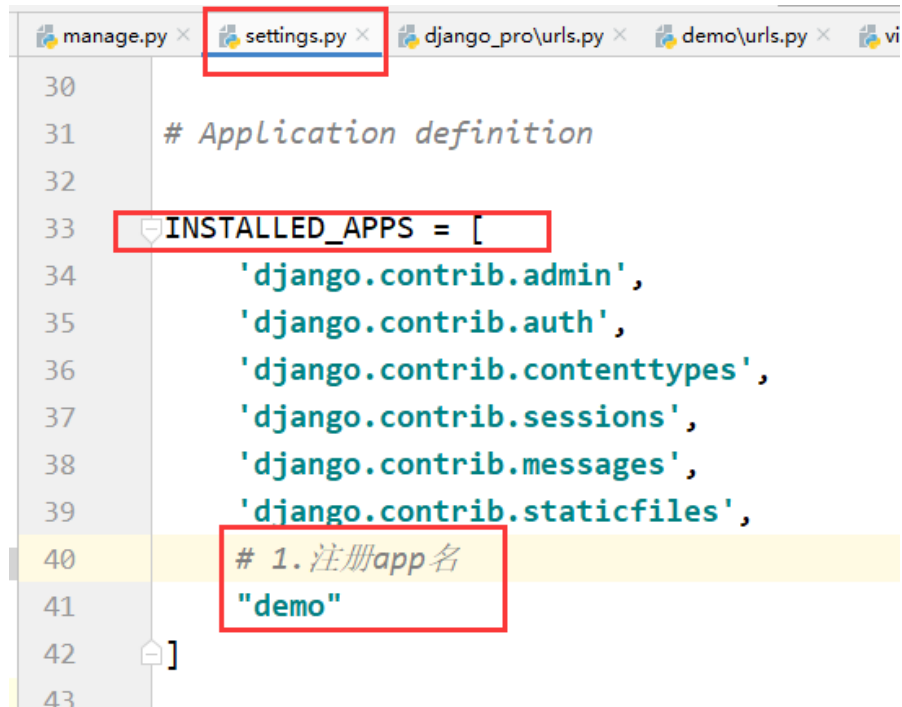
运行结果：



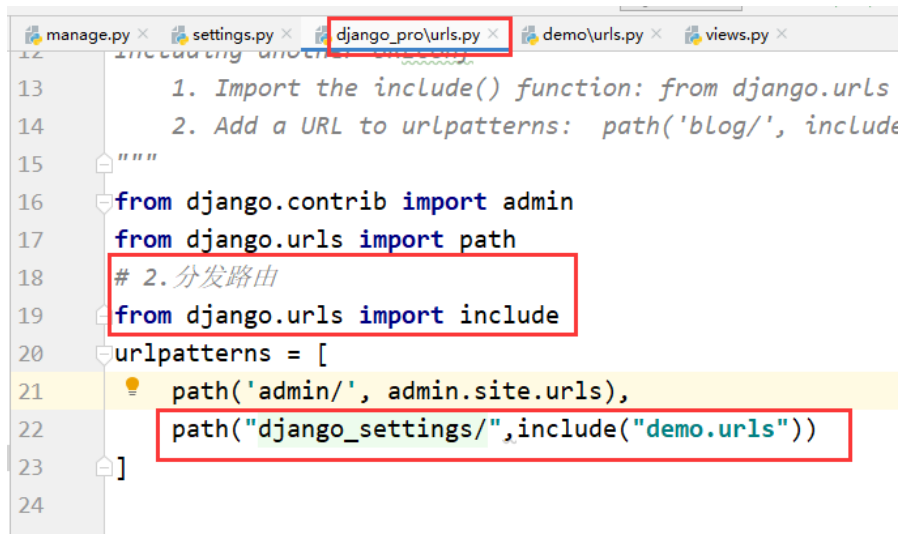
2.4 第一个项目视图创建

重点五步：

2.3.1注册app名



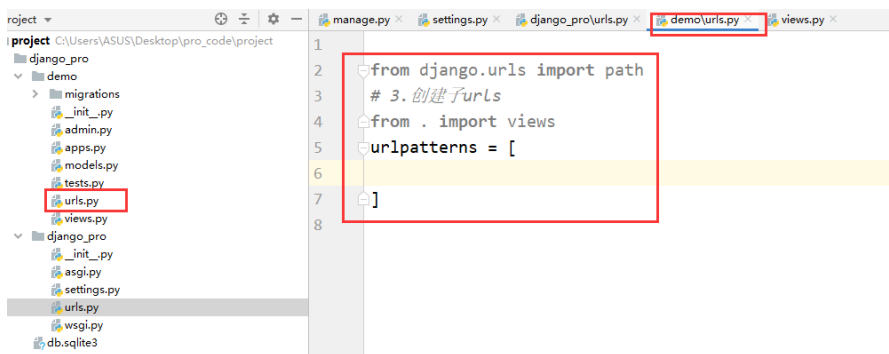
2.3.2分发路由



The screenshot shows the Django Pro URLs configuration in `urls.py`. The file is open in an IDE with several tabs: `manage.py`, `settings.py`, `django_pro/urls.py` (active), `demo/urls.py`, and `views.py`. The code includes comments in Chinese and Python code for importing `include` and `path` functions, and adding a URL pattern for `demo.urls`.

```
12 including another django project
13 1. Import the include() function: from django.urls
14 2. Add a URL to urlpatterns: path('blog/', include
15
16 from django.contrib import admin
17 from django.urls import path
18 # 2. 分发路由
19 from django.urls import include
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path("django_settings/", include("demo.urls"))
23 ]
24
```

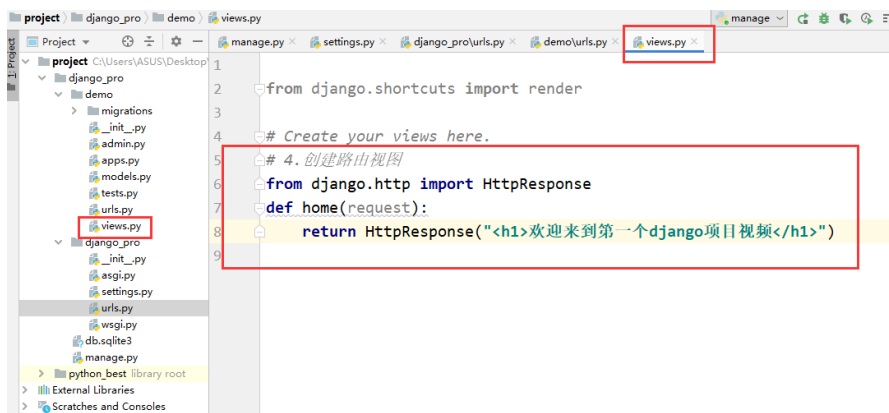
2.3.3 创建子路径urls



The screenshot shows the Django Pro URLs configuration in `demo/urls.py`. The file is open in an IDE with several tabs: `manage.py`, `settings.py`, `django_pro/urls.py`, `demo/urls.py` (active), and `views.py`. The code includes comments in Chinese and Python code for importing `path` and `views` functions, and adding a URL pattern for `views`.

```
1 from django.urls import path
2 # 3. 创建子urls
3 from . import views
4 urlpatterns = [
5
6 ]
7
8
```

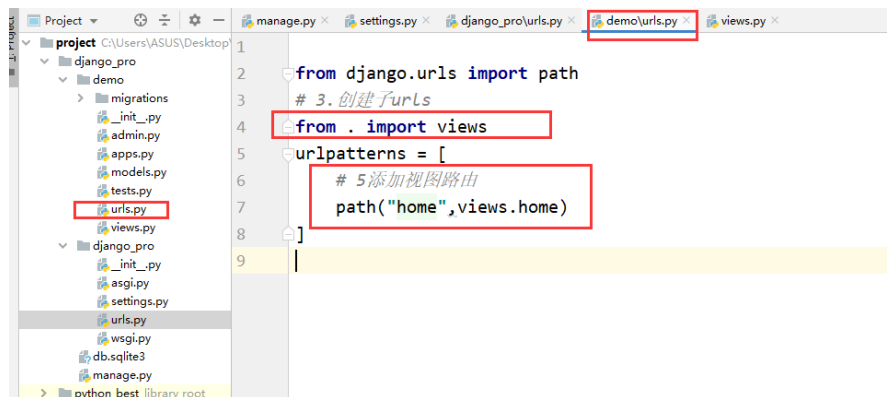
2.3.4 创建函数视图



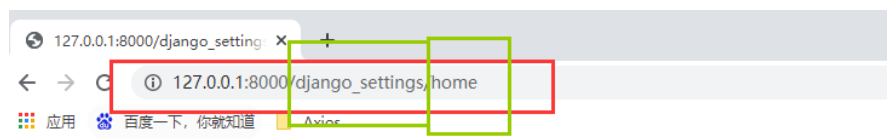
The screenshot shows the Django Pro Views configuration in `views.py`. The file is open in an IDE with several tabs: `manage.py`, `settings.py`, `django_pro/urls.py`, `demo/urls.py`, and `views.py` (active). The code includes comments in Chinese and Python code for importing `render` and `HttpResponse` functions, and defining a `home` function that returns an `HttpResponse`.

```
1 from django.shortcuts import render
2
3 # Create your views here.
4 # 4. 创建路由视图
5 from django.http import HttpResponse
6 def home(request):
7     return HttpResponse("<h1>欢迎来到第一个django项目视频</h1>")
8
9
```

2.3.5 配置视图路由



运行结果:



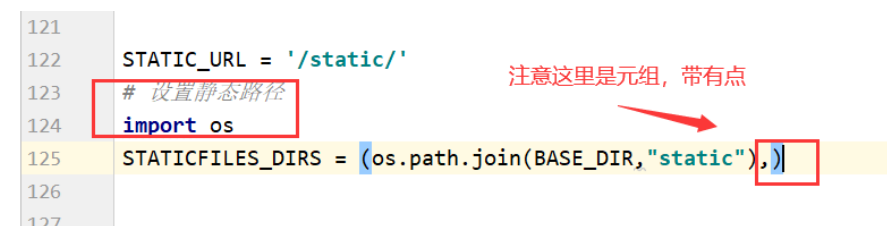
欢迎来到第一个django项目视频

一级

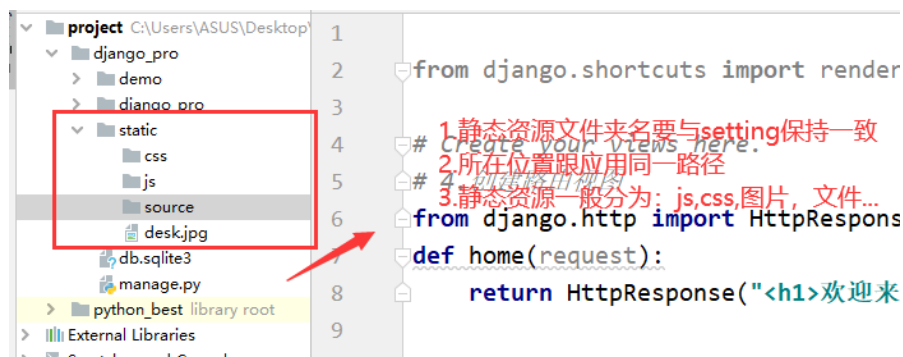
二级

3.静态资源加载

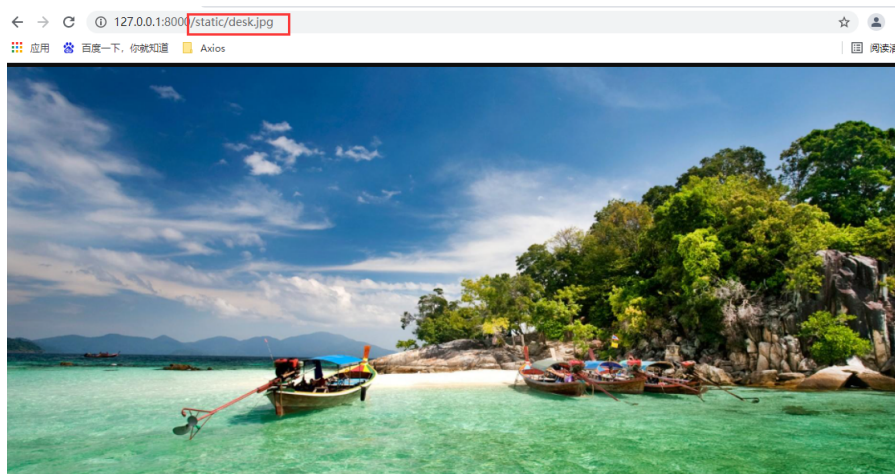
设置静态资源:



加载静态资源

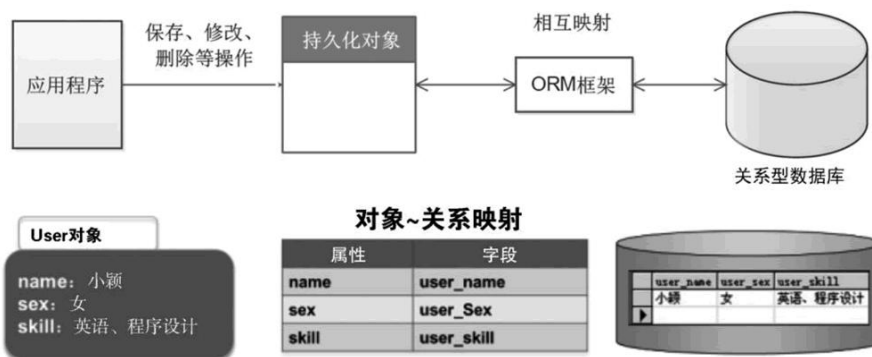


运行结果：



4.ORM操作

Django模型功能本质上是使用了一个数据库的概念：对象关系映射。对象关系映射(Object Relational Mapping, 简称ORM)是用于实现面向对象编程语言里不同类型系统的数据之间的转换。ORM 在业务逻辑层和数据库层之间充当了桥梁的作用，通过使用描述对象和数据库之间的映射的元数据，将程序中的对象自动持久化到数据库中。



1.数据库模块安装：

```
pip install pumysql
```

2.settings配置数据库


```
DATABASES = {
    # 配置mysql数据库
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'demos',
        'USER': 'root',
        'PASSWORD': 'mysql'
    }
}
```

3.同settings同级init.py配置

```
import pymysql
pymysql.install_as_MySQLdb()
```

4.创建数据库demos

创建语法:

```
create database demos charset="utf8";
```

5.在demo应用下的models中创建表:

```
from django.db import models

# Create your models here.

class User(models.Model):
    # 创建字段
    # charField字符串类型
    username = models.CharField(max_length=18,verbose_name="用户名")
    # int类型 -2147483648到2147483647
    age = models.IntegerField(verbose_name="年龄")
    # 日期类型
    birth = models.DateField(verbose_name="出生日期")
    # 性别
    FEMALE = 0
    MALE = 1
    GENDERS = (
        (FEMALE,"女"),
        (MALE,"男")
    )
    gender =
models.IntegerField(choices=GENDERS,default=MALE,verbose_name="性别")
```

```
# 添加表名
class Meta:
    db_table = "user"
    verbose_name = "用户信息表"

# 查看打印信息
def __str__(self):
    return self.username
```

6.数据库迁移

a.先生成本地数据库文件

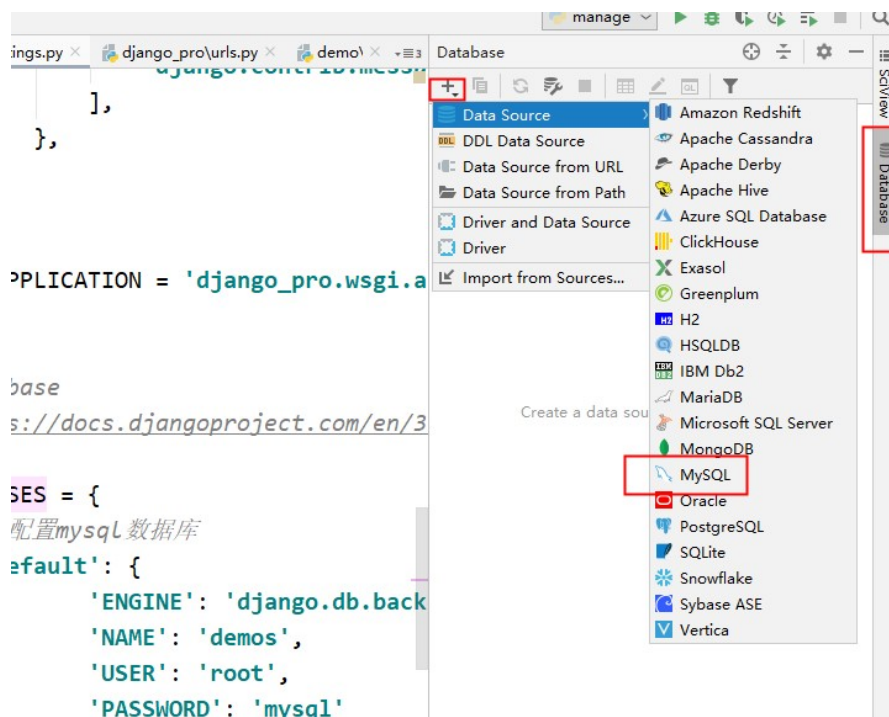
```
python manage.py makemigrations
```

b.推送到MySQL数据库

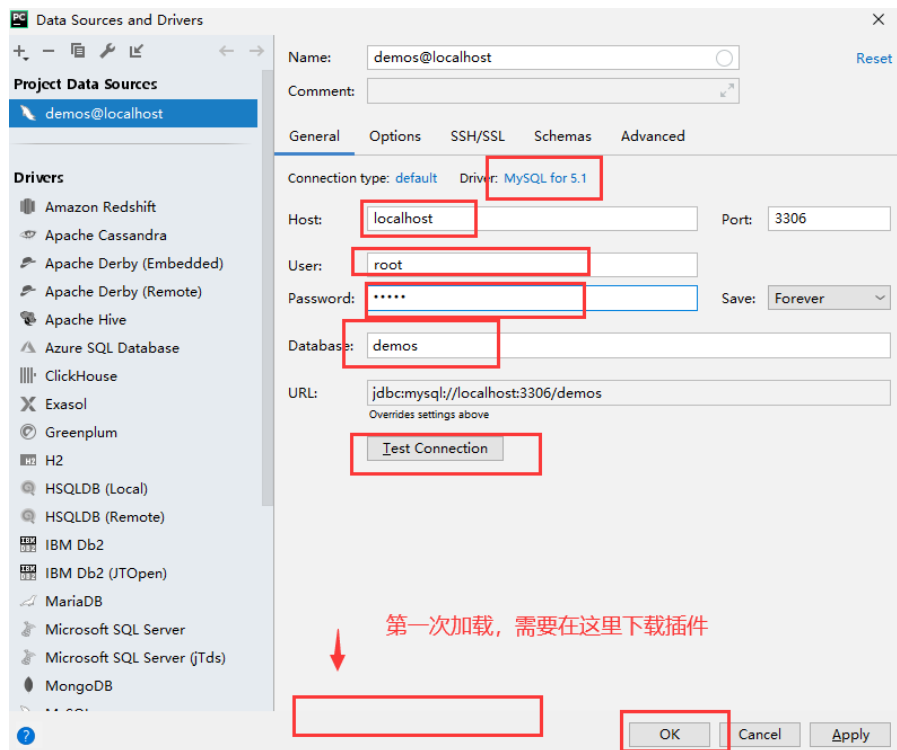
```
python manage.py migrate
```

7.查看数据库数据

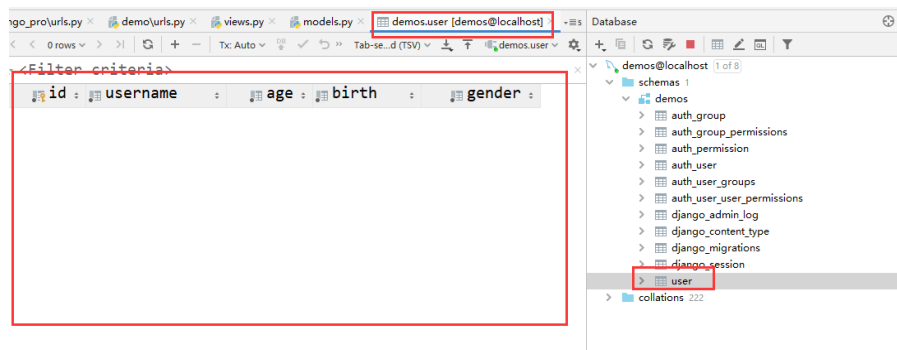
a.Pycharm设置数据库



b.测试数据库



8. 查看结果



9. ORM语法操作

```
from .models import User
from datetime import datetime
# ORM操作:增删查改
def get_orm(request):
    # 添加数据
    User.objects.create(
        username = "张三",
        age = 18,
        birth = datetime.now(),
        gender = 1
    )
    # 获取数据
    user_list = User.objects.filter(username="张三")
    data = {
        "username": user_list[0].username
```

```

        "username" : user_list[0].username,
        "gender" : user_list[0].gender,
        "age" : user_list[0].age,
        "birth" : user_list[0].birth,
    }
    # HttpResponse发送字符串
    # return HttpResponse("数据库列表信息: {}".format(data))
    # 发送JSON数据
    return JsonResponse(data)

```

添加路由

```

# orm视图路由
path("get_orm",views.get_orm)

```

运行结果：



增加和删除

```

from .models import User
from datetime import datetime
# ORM操作:增删查改def get_orm(request):
def get_orm(request):
    #修改数据
    # 先获取再修改
    # user_obj = User.objects.filter(username = "张三")
    # user_obj[0].username = "李四"
    # user_obj[0].save()
    # 删除数据
    User.objects.filter(username = "李四")[0].remove()
    # 获取数据
    user_list = User.objects.filter(username="张三")
    data = {
        "username" : user_list[0].username,
        "gender" : user_list[0].gender,
        "age" : user_list[0].age,
        "birth" : user_list[0].birth,
    }
    # HttpResponse发送字符串
    # return HttpResponse("数据库列表信息: {}".format(data))

```

```
# return HttpResponse( 数据库列表信息: {}".format(data))
# 发送JSON数据
return JsonResponse(data,json_dumps_params={"ensure_ascii":False})
```

5.模板操作

a.再Settings中配置模板

```
# 模板配置
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,"templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

b.与static同目录下创建templates文件夹,文件夹名保持和settings一直



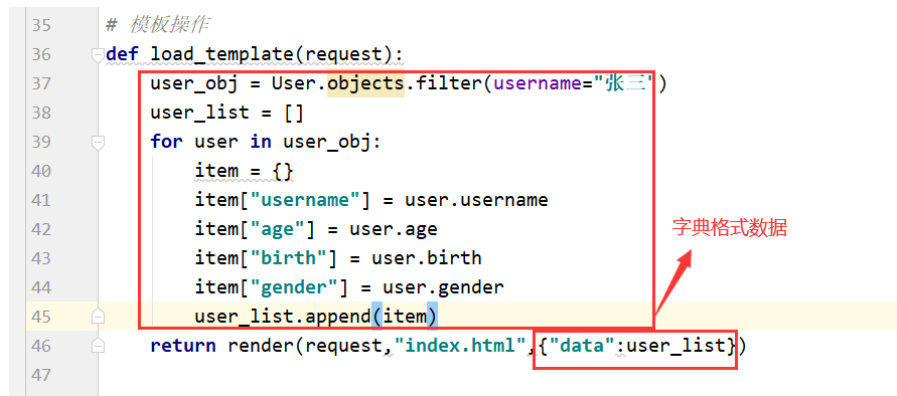
c.配置路由和视图



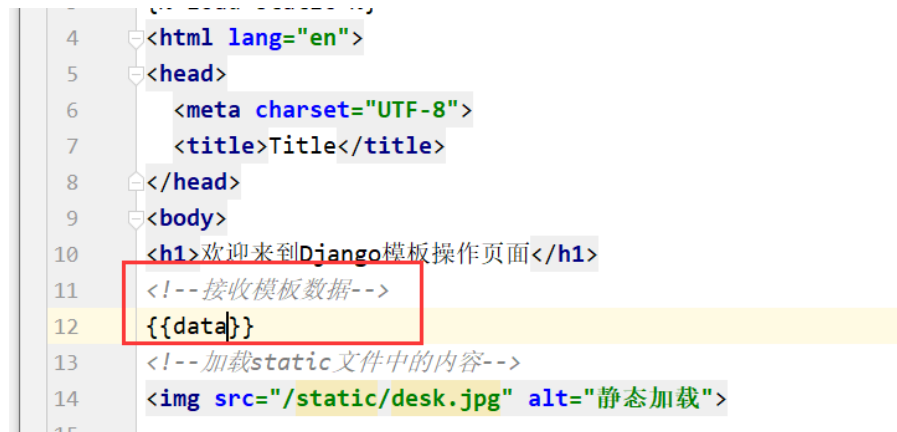
d.运行结果



e.模板数据



f.模板接收数据



g.运行结果



6.项目实操 - 智慧考试系统

6.1项目描述：

"智慧校园"是互联网基础下智慧化工作。如今推出一款智慧考试项目，项目中包含注册，登录，验证码制作，邮箱安全验证，成为机构，创建考试模板，设置考试信息，开始考试，考试提交，考试排名等功能。

6.2 项目实施：

1.创建项目

```
django-admin startproject online_school
```

2.创建index应用

```
python manage.py startapp index
```

3.配置settings

注册应用

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    # 注册首页应用  
    "index",]  

```

模板设置

```
# 加载模板文件  
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, "templates")],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    ],  
]
```

数据库设置

```
DATABASES = {  
    # 配置mysql数据库  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'online_school',  
        'USER': 'root',  
        'PASSWORD': 'mysql'  
    }  
}
```

静态文件设置


```
STATIC_URL = '/static/'  
#加载静态文件  
STATICFILES_DIRS = (os.path.join(BASE_DIR,"static"),)
```

4.路由分发

```
# 使用路由分发  
from django.urls import include  
urlpatterns = [  
    path("",include("index.urls")),]
```

5.index子路由

```
# 导入视图  
from . import views  
urlpatterns = [  
    # 首页显示路径  
    path("",views.index),]
```

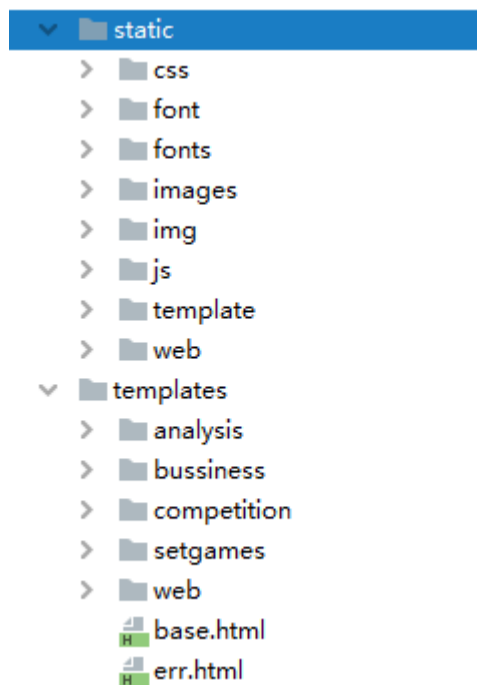
6.views视图

```
# 首页显示  
def index(request):  
    # 设置cookie  
    request.session['uid'] = ""  
    request.session['username'] = ""  
    return render(request,"web/index.html")
```

7.数据库迁移

```
python manage.py makemigrations  
python manage.py makemigrate
```

8.模板文件和静态文件



9. 首页显示



10. 注册页面的验证码制作

注册

×

邮箱

hugo.zhang@example.com

密码

#p@ssw0d123

确认密码

#p@ssw0d123

验证码

T D q J E X

已有账户, 去登录

注册

11.redis模块安装

安装语法:

```
pip install redis
```

12.在index应用下创建connectredis文件



13.在connectredis同级目录下创建addcode, 验证码图片制作

```
import random
from PIL import Image, ImageDraw, ImageFilter, ImageFont
class CodeGen(object):
    def __init__(self, text_str=None, size=None, background=None):
        """
        text_str: 验证码显示的字符组成的字符串
        size: 图片大小
        background: 背景颜色
        """
```

```

        self.text_str =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
YZ'

        self.size = (150, 50)
        self.background = 'rgb(74,101,128)'
        self.text_list = list(self.text_str)

def create_pic(self):
    """
    创建一张图片
    """
    self.width, self.height = self.size
    self.img = Image.new("RGB", self.size, self.background)
    # 实例化画笔
    self.draw = ImageDraw.Draw(self.img)

# 图画上创验证码
def create_text(self, font_type, font_size, font_color, font_num, start_xy):
    """
    font_type: 字体
    font_size: 文字大小
    font_color: 文字颜色
    font_num: 文字数量
    start_xy: 第一个字左上角坐标,元组类型, 如 (5,5)
    功能: 画文字
    """
    font = ImageFont.truetype(font_type, font_size)
    check = random.sample(self.text_list, font_num)
    self.draw.text(start_xy, " ".join(check), font=font, fill=font_color)
    return check

def get_pic_code():
    __cg = CodeGen()
    __cg.create_pic()
    __check = __cg.create_text("web/static/font/simsun.ttc", 24, (255, 165, 123),
6, (7, 7))
    return __cg.img, __check

if __name__ == '__main__':
    print(get_pic_code())

```

14.视图获取和返回验证码

```

# 验证码
def login_vcode(request):

    b = BytesIO() # 字节

```

```

b = bytesIO() # 字节
img, check = get_pic_code() # 图片和验证码
# print(check)
img.save(b, format='png') # 将图片转换成字节类型.png图片

vcode = base64.b64encode(b.getvalue()) # 将图片对象内容转成二进制
# uuid生成唯一主键
sign = str(uuid.uuid1())

'''
把临时生成的验证码存储到redis
'''

str_code = ''.join([str(i) for i in check]).lower()
print("验证码: ",str_code)
cod.set(sign, str_code) # 把验证码存储起来

return JsonResponse({
    "status": 200, # 状态码
    'vcode': vcode.decode('utf-8'), # 图片
    'sign': sign # 验证码
})

```

15.配置验证码路由

```

# 验证码路径
path("login_vcode/",views.login_vcode),

```

16.运行结果：

注册

邮箱
hugo.zhang@example.com

密码
#p@ssw0d123

确认密码
#p@ssw0d123

验证码
E 5 D 4 B 8

点击验证码更换

已有账户, 去登录 注册

17.添加register应用

```
pip install register
```

18.settings配置

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    # 注册首页应用  
    "index",  
    # 注册应用  
    "register"  
]
```

19.配置路由分发

```
# 使用路由分发  
from django.urls import include  
urlpatterns = [  
    path("",include("index.urls")),  
    path("register/",include("register.urls")),  
]
```

20.配置register子路由

```
# 导入视图  
from . import views  
urlpatterns = [  
    # 基本注册  
    path("signup",views.signup),]
```

21.创建模型

```
from django.db import models  
  
# Create your models here.  
  
# 创建用户信心  
class User(models.Model):
```

```

    username = models.CharField(max_length=32,null=True,verbose_name="用户名")
    password = models.CharField(max_length=100,null=True,verbose_name="密码")
    email = models.CharField(max_length=100,null=True,verbose_name="邮箱")
    last_login = models.DateField(verbose_name="最后登录")
    date_join = models.DateField(verbose_name="创建时间")
    #激活用户
    is_staff = models.IntegerField(default=0,verbose_name="会员")
    is_active = models.IntegerField(default=0,verbose_name="活跃")

class Meta:
    db_table = "register_user"
    verbose_name = '用户信息'
    verbose_name_plural = '用户信息'

    def __str__(self):
        return self.email

from shortuuidfield import ShortUUIDField

class Profile(models.Model):
    """
    用户信息类
    """
    # 用户状态
    GUEST_USER = 11
    NORMAL_USER = 22
    COMPANY_USER = 33

    USER_SRC = (
        (GUEST_USER, '游客用户'),
        (NORMAL_USER, '普通用户'),
        (COMPANY_USER, '机构用户'),
    )
    MALE = 1
    FEMALE = 0

    GENDER = (
        (MALE, '男'),
        (FEMALE, '女'),
    )

    UNVERIFIED = 0
    ACTIVATED = 1

    USER_STATUS = (

```

```

        (UNVERIFIED, '未验证'),
        (ACTIVATED, '已激活')
    )

    uid = ShortUUIDField(verbose_name='用户id', max_length=32,
                        blank=True, null=True, unique=True)
    # 用户相关
    user_src = models.IntegerField(verbose_name='用户来源',
                                   choices=USER_SRC, default=GUEST_USER)
    user_status = models.IntegerField(verbose_name='用户状态',
                                      choices=USER_STATUS, default=ACTIVATED)
    # 用户基本信息
    name = models.CharField(verbose_name='姓名', max_length=32,
                            blank=True, null=True)
    email = models.CharField(verbose_name='邮箱', max_length=40,
                             blank=True, null=True)
    sex = models.IntegerField(verbose_name='性别', choices=GENDER,
                              default=0)
    age = models.IntegerField(verbose_name='年龄', default=0)
    phone = models.CharField(verbose_name='手机号', max_length=11,
                             blank=True, null=True)

    class Meta:
        db_table = "profile"
        verbose_name = '用户信息'
        verbose_name_plural = '用户信息'

    def __str__(self):
        return self.name

    def data(self):
        return {
            'uid': self.uid,
            'numid': 1000000 + self.pk,
            'name': self.name or '',
            'nickname': self.name,
        }
}

```

22.视图路由

```

from django.http import HttpResponse, JsonResponse
from django.shortcuts import render
from index.connectredis import cod
from .models import User, Profile
# Create your views here.
from datetime import datetime
import uuid

```



```

# 获取注册提交数据
def signup(request):
    email = request.POST.get('email')
    password = request.POST.get('password')
    password_again = request.POST.get('password_again')
    sign = request.POST.get('sign')
    vcode = request.POST.get('vcode')
    # print(email,password,password_again,sign,vcode,"sdfdsfs")
    if password != password_again: # 两次密码不一样, 返回错误码300002
        return JsonResponse({
            'status': 300002,
            'message': "两次密码不一样"
        })
    result = cod.get(sign) # 校验vcode, 逻辑和登录视图相同
    # print(result)
    if not (result and (result.decode('utf-8') == vcode.lower())):
        return JsonResponse({
            "status": 300003,
            "message": "验证码错误"
        })
    # 验证是否存在
    if User.objects.filter(email__exact=email).exists(): # 检查数据库是否存在该用户
        return JsonResponse({
            "status": 300004,
            "message": "邮箱已存在"
        }) # 返回错误码300004
    username = email.split('@')[0] # 生成一个默认的用户名
    if User.objects.filter(username__exact=username).exists():
        username = email # 默认用户名已存在, 使用邮箱作为用户名
    User.objects.create( # 创建用户, 并设置为不可登录
        username=username,
        last_login=datetime.now(),
        date_join=datetime.now(),
        email=email,
        password=password,
    )
    Profile.objects.create( # 创建用户信息
        name=username,
        email=email
    )
    sign = str(uuid.uuid1()) # 生成邮箱校验码
    cod.setex(sign,email,1800) # 在redis设置30min时限的验证周期
    return JsonResponse({
        # 返回JSON数据
        'status': 200,
        'message': "ok",
        'email': email,

```

```
'sign': sign
})
```

跳转视图

```
# 邮箱发送信息
def signup_redirect(request):
    """
    :param request: 请求对象
    :return: email邮箱地址, sign注册验证码
    """
    email = request.GET.get('email', '')
    sign = request.GET.get('sign', '')
    return render(request, 'web/sign_email.html', {
        'email': email,
        'sign': sign
    })
```

运行结果：



邮箱settings配置

```
# 设置邮箱发送链接
# send e-mail
DOMAIN = "http://127.0.0.1:8899"
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
EMAIL_HOST = 'smtp.qq.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = '846213108@qq.com'
EMAIL_HOST_PASSWORD = 'kfgglqhweyhsbdga'
```

注册码



发送邮件视图

```
from django.conf import settings
from django.core.mail import send_mail
# 邮件验证
def sendemail(request):
    to_email = request.GET.get('email', '') # 在url中获取的注册邮箱地址
    sign = request.GET.get('sign', '') # 在url中获取的sign标识
    # 点击发送邮件
    email_result = cod.setex(to_email, 60, 1)
    # print("eeeeeee", email_result)
    if email_result: # 检查用户是否在同一时间多次点击发送邮件
        title = '[Quizz.cn用户激活邮件]' # 定义邮件标题
        sender = settings.EMAIL_HOST_USER # 获取发送邮件的邮箱地址
        url = settings.DOMAIN + '/register/email_notify?email='+ to_email +
        '&sign=' + sign # 回调函数
        msg = '您好, Quizz.cn管理员想邀请您激活您的用户, 点击链接激活。'
        {}'.format(url) # 邮件内容
        ret = send_mail(title, msg, sender, [to_email],
                        fail_silently=True) # 发送邮件并获取发送结果
    if not ret:
        return JsonResponse({
            "status": 300006,
            "message": "发送错误"
        }) # 发送出错, 返回错误码300006
    cod.setex(to_email, 1800, 1)
    # 正常发送,
    return JsonResponse({
        "status": 200,
        "message": "ok",
    }) # 返回空JSON数据
    else:
        return JsonResponse({
            "status": 300005,
```

```
"message": "发送邮件错误"
}) # 如果用户发送邮件错误, 返回错误码300005
```

配置路由

```
# 邮箱验证
path("signup_redirect", views.signup_redirect),
# 发送邮件
path("sendmail", views.sendmail),
```

运行结果:



QQ邮箱



点击邮箱连接验证视图函数

```
# 验证邮箱发送:
def email_notify(request):
    email = request.GET.get('email', '') # 获取要验证的邮箱
    sign = request.GET.get('sign', '') # 获取校验码
    signcode = cod.get(sign) # 在redis校验邮箱
    # print("邮箱验证", email, sign, signcode)
    if not signcode:
        return render(request, 'err.html',
                        # 校验失败返回错误视图
                        {"errtitle": "校验码超时", "errmsg": "由于您过长时间未校验邮件, 导致校验失
```

```

败"}}
    if not (email == signcode.decode('utf-8')):
        return render(request, 'err.html',
            # 校验失败返回错误视图
{"errtitle": "校验码错误", "errmsg": "您的校验码出现错误, 激活失败"})
    try:
        user = User.objects.get(email=email) # 获取用户
        # print(user,"用户")
    except User.DoesNotExist:
        user = None
    if user is not None: # 激活用户
        user.is_active = True
        user.is_staff = True
        user.save()
        profile = Profile.objects.filter( # 配置用户信息
            name=user.username,
            email=user.email,
        ).first()
        # print(profile)
        profile.user_src = Profile.NORMAL_USER # 配置用户为普通登录用户
        profile.save()

        request.session['uid'] = profile.uid # 配置session
        request.session['username'] = profile.name
        return render(request, 'web/index.html', { # 渲染视图, 并返回已登录信息
            'user_info': profile.data,
            'has_login': True,
            'msg': "激活成功",
        })
    else:
        return render(request, 'err.html',
            {"errtitle": "激活失败", "errmsg": "我们没有找到您的注册信息, 激活失败"}) #
    校验失败返回错误视图

```

验证路由

```

# 邮箱回执验证
path("email_notify", views.email_notify)

```

运行结果：

