# 语言：Python

## 创建Django程序

1. 配置环境：
2. 创建文件夹
3. 创建专用虚拟环境python -m venv <environment\_name>
4. 选择解析器（命令面板）
5. 安装Django库python -m pip install django
6. 创建Django项目

* 创建Project文件夹：django-admin startproject <project\_name> [path]

1. 创建并首次运行Django应用
2. 创建App文件夹：python manage.py startapp <app\_name> [path]
3. 创建App路由文件urls.py
4. 修改Project路由文件urls.py
5. 修改视图views.py
6. 创建调试器（launch.json）
7. 首次运行/调试
8. 创建Template模板
9. 注册app（settings.py | INSTALLED\_APPS）
10. 创建templates两层文件夹结构
11. 创建父模板base.html文件
12. 创建子模板.html文件
13. 修改视图views.py
14. 修改路由urls.py
15. 运行或调试
16. 引用静态文件
17. 添加引用到路由urls.py（urlpatterns += staticfiles\_urlpatterns()）
18. 添加静态文件夹（两层结构）
19. 在模板中添加引用：{% load static %}
20. 修改模板.html
21. 创建模型、迁移
22. 修改模型models.py
23. 创建迁移脚本：python manage.py makemigrations
24. 迁移到数据库：python manage.py migrate
25. 创建表单
26. 创建表单文件forms.py
27. 修改模板.html
28. 添加样式CSS
29. 修改路由urls.py
30. 修改视图views.py
31. 运行/调试
32. 创建超级管理员
33. 创建超级管理员账户：

python manage.py createsuperuser --username=<username> --email=<email>

1. 添加路由：path('admin/', admin.site.urls),
2. 运行
3. 创建requirements.txt文件

创建：pip freeze > requirements.txt

使用：pip install -r requirements.txt

# 目标：

创建一个Django app：

* 包含3个页面
* 引用一个base模板

# 先决条件Prerequisites：

* 安装Python扩展包
* 在OS中安装Python3
* 在VS Code中指定为OS中的Python解析器

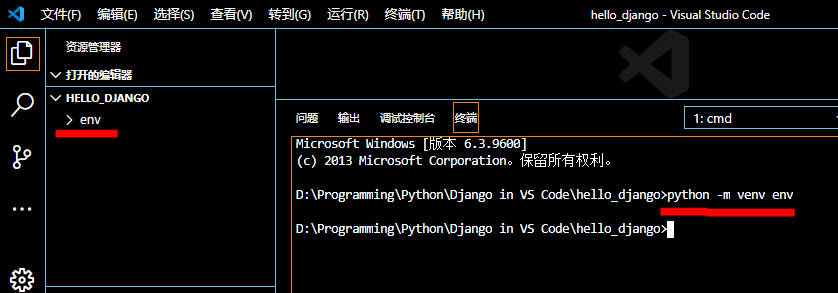
# 步骤：

## 1.创建虚拟环境与安装库

### 1.创建Project专用的虚拟环境

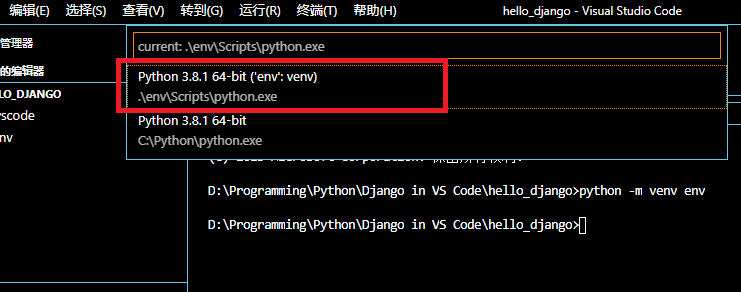
1.创建Project文件夹hello\_django

2.在VS Code中打开目标文件夹，在终端中进入Project文件夹，然后输入python -m venv env



3.将Python解析器选择为专用的虚拟环境解析器。

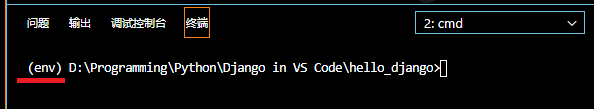




4. **确认。**

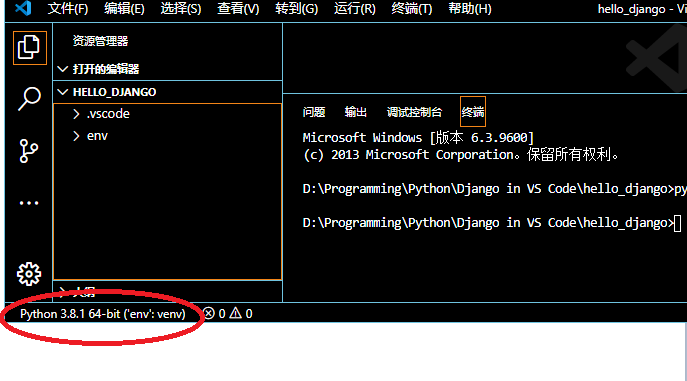
激活虚拟环境：

* 配置完毕后会自动激活该虚拟环境。可以通过创建一个新的终端，查看其路径前是否有“env”



* 如没有激活，可以在终端中输入命令：env\scripts\activate

解析器：确认是否使用了虚拟环境中的解析器，可以通过界面左下角检视：



#### 故障与排除

VS Code设置中有规定虚拟环境的路径，如果虚拟环境命名与设置不同，则VSC无法找到虚拟环境；结果会造成直接造powershell的解析器中执行之后的命令。

查找VCS的设置中的虚拟环境路径的设置：python.env

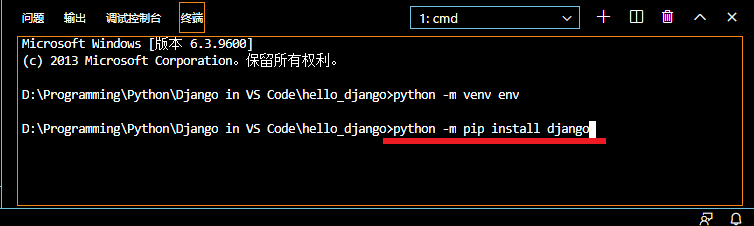
图形用户界面, 文本, 应用程序

描述已自动生成

### 2.安装Django库

* 在终端中输入命令：

python -m pip install django



Django将自动安装在当前专用虚拟环境中。

## 2.创建并运行一个简易Django App

**Django Project的通用知识：**

* 一个Django Project由若干site-level的配置文件，和一个或多个app组成，并可以部署到一个web主机上来创建一个完整的web应用。
* Django Project的组成：配置文件和app
* 可以部署到web主机
* 目的是创建一个web应用
* Project与app之间的关系：
* 一个Project可以包含多个app；
* 在Project中的每个app实现一个独立功能；
* 一个app可以在多个Project中；
* 一个app仅仅是遵循Django约定的Python包。
* Project和app的创建：
* 先创建Project来约束app；
* 再创建个别app实现具体功能。
* 实现这一目的，可调用django-admin工具，该功能是Django管理命令行工具之一，安装好Django包后可以直接调用。

### 创建Django project

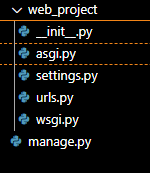
#### 1.创建Django project文件夹

创建一个名为web\_project的项目文件夹，在终端中输入命令：

django-admin startproject web\_project .

注意，该命令以“.”结尾，用作指明是在当前文件夹中创建project文件夹。

执行命令后，当前文件夹中会创建Django框架的文件夹和文件：



manage.py：当前项目的Django命令行管理实用程序。

一个以web\_project命名的文件夹，文件夹包括以下文件：

* \_\_init\_\_.py：

一个空的文件，用于标识本文件夹是一个Python模块包（module package）。此时可以在代码中引用该包中的py文件（如a.py存在与web\_project文件夹中，则可以使用from web\_project import a；否则会报错。）

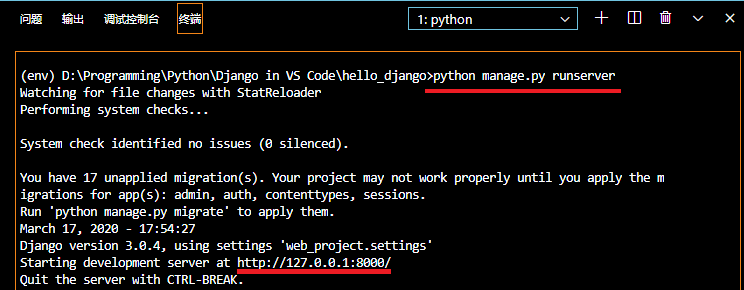
* asgi.py：用于处理具有异步功能的标准接口。
* settings.py：包含Django项目的设置
* urls.py：包含一个Django内容的表，通常用于设置路由。
* wsgi.py：

兼容WSGI的Web服务器为您的项目提供服务的入口点。 通常，此文件保持原样，因为它为生产Web服务器提供了挂钩。

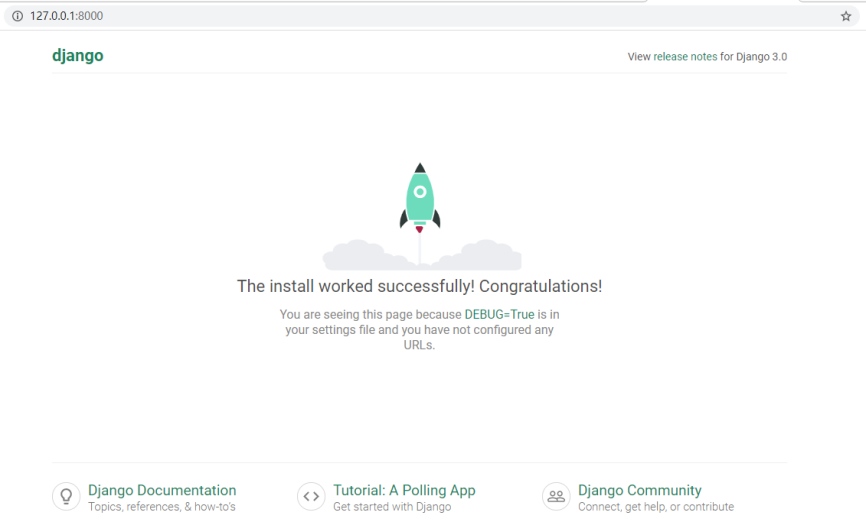
#### 2.运行Django App

在终端中输入以下命令，在专用虚拟环境中运行app：

python manage.py runserver

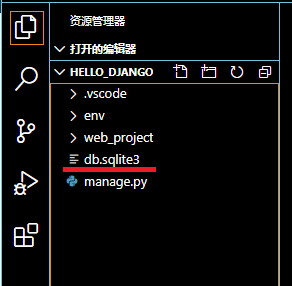


网页使用Django发开服务器（Django's development server）运行，默认地址为<http://127.0.0.1:8000/>，默认端口为8000。



* 默认数据库

当首次运行服务器时，将会自动创建默认SQLite数据库db.sqlite3



* Django发开服务器（Django's development server）

只作本地开发调试使用。当部署到web主机时，wsgi.py模快负责连接到web主机服务器。

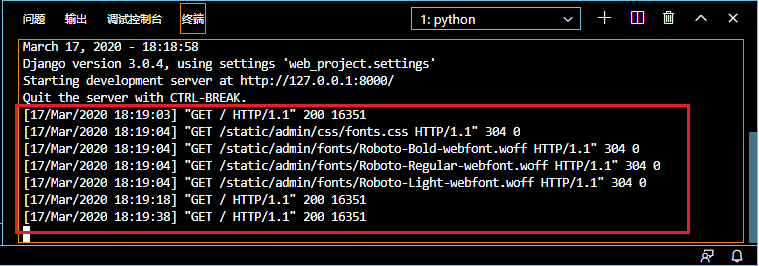
* 指定端口

使用以下命令指定端口：

python manage.py runserver 5000

* 服务器日志（server log）

在终端中显示：



#### 3.停止Django发开服务器

1.关闭浏览器窗口

2.在终端中使用组合键：Ctrl+C

### 创建一个Django app

#### 1.使用startapp工具创建app

在终端中输入命令：

python manage.py startapp hello



执行命令后将自动创建app文件夹和文件：

migrations 文件夹：存放管理数据库的命令文件

\_\_init\_\_.py：声明当前文件夹是一个python包

admin.py：用于创建admin后台管理界面

apps.py：app应用程序配置文件

models.py：包含定义数据对象的模型类

test.py：app的测试文件

views.py：包含定义网站页面的视图函数

#### 2.修改视图views

from django.http import HttpResponse

# Create your views here.

def home(request):

    return HttpResponse("Hello, Django!")

#### 3.创建app路由文件urls.py

在hello/文件夹中创建urls.py，用于定义路由函数。

from django.urls import path

from hello import views

urlpatterns = [

    path("", views.home, name="home"),

]

#### 4.修改project路由文件urls.py

修改project文件夹下的urls.py，让其包含app的路由设置。

from django.contrib import admin

from django.urls import include,path

urlpatterns = [

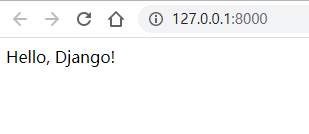
    path('admin/', admin.site.urls),

    path("", include("hello.urls")),

]

#### 5.保存并运行

* 保存所有修改的文件Ctrl+K S
* 运行app：python manage.py runserver
* 打开浏览器登录：http://127.0.0.1:8000/

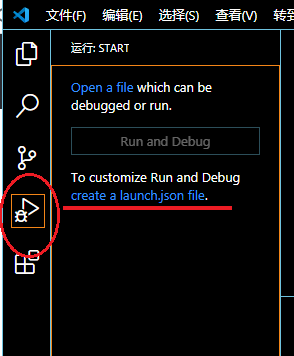


### 创建调试器配置文件

以上均是使用命令在终端中运行project。在VS Code中可以使用调试器快捷地运行并调试project。

#### 1.创建调试配置文件launch.json

切换到调试窗口》点击创建launch.json文件



在命令栏中选择Python环境



然后选择Django调试器配置



VS Code自动创建配置文件launch.json文件：

{

    // 使用 IntelliSense 了解相关属性。

    // 悬停以查看现有属性的描述。

    // 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387

    "version": "0.2.0",

    "configurations": [

        {

            "name": "Python: Django",

            "type": "python",

            "request": "launch",

            "program": "${workspaceFolder}\\manage.py",

            "args": [

                "runserver",

                "--noreload",

                "5000"

            ],

            "django": true

        }

    ]

}

"${workspaceFolder}/manage.py"：告诉VS Code的Python解析器运行manage.py

args：参数列表。

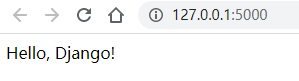
参数列表中的“runserver，--noreload，5000”，相当于运行命令python manage.py runserver --noreload 5000

"django": true：允许VS Code可以调试Django页面模板（templates）

保存launch.json文件。

#### 2.运行调试

按下“F5”进行调试。



### 调试器的使用

每次进行调试，VS Code将自动保存修改的文件。

记得在每次调试后在终端中按下Ctrl+C停止运行app，否则新的调试进程与旧的调试会因为使用相同的端口而报错。

#### 实验操作：

1.修改路由文件hello/urls.py，增加路由列表。

    path("hello/<name>", views.hello\_there, name="hello\_there"),

代码中的第一个参数定义了路由"hello/"，并接受一个名为name的变量，然后将变量值传递给views.hello\_there函数。

2.修改views.py文件

import re

from datetime import datetime

from django.http import HttpResponse

def home(request):

    return HttpResponse("Hello, Django!")

def hello\_there(request, name):

    now = datetime.now()

    formatted\_now = now.strftime("%A, %d %B, %Y at %X")

    # Filter the name argument to letters only using regular expressions. URL arguments

    # can contain arbitrary text, so we restrict to safe characters only.

    match\_object = re.match("[a-zA-Z]+", name)

    if match\_object:

        clean\_name = match\_object.group(0)

    else:

        clean\_name = "Friend"

    content = "Hello there, " + clean\_name + "! It's " + formatted\_now

    return HttpResponse(content)

3.在代码中设置断点：now = datetime.now()

4.进行调试（F5）。

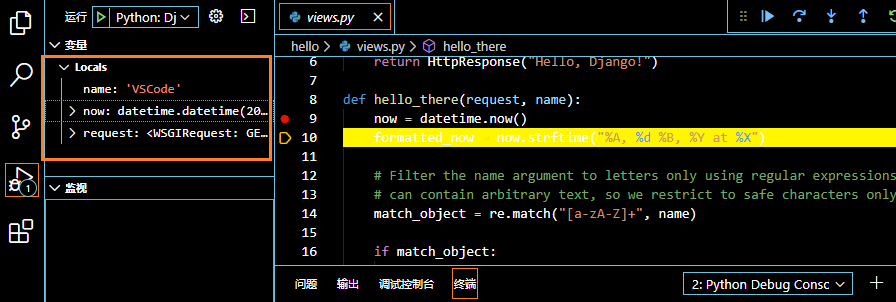
注意，在调试时状态栏将显示为：



5.在浏览器中导航到地址：

http://127.0.0.1:8000/hello/VSCode

6.在调试器中运行到下一行，在调试和运行面板中查看变量name和now的值。



* 在调试模式下，双击变量可以对变量值进行修改。尤其是在调试过程中发现返回的变量值出现错误时，对变量值进行修改然后继续调试。

7.当调试程序中断时，在调试控制台中输入代码，可以试验相关变量、方法的返回值。

如：在调试控制台中输入以下命令：

now.strftime("%a, %d %B, %Y at %X")

now.strftime("%a, %d %b, %Y at %X")

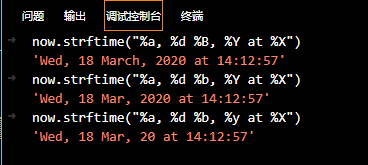
now.strftime("%a, %d %b, %y at %X")

将会返回：

'Wed, 18 March, 2020 at 14:12:57'

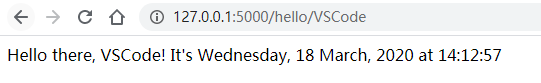
'Wed, 18 Mar, 2020 at 14:12:57'

'Wed, 18 Mar, 20 at 14:12:57'



* 在调试模式下，当在调试控制台发现想要的代码时，可以在代码修改器中修改，但不会在当前调试中产生结果。在重启调试后才会得到目标的结果。
* 在调试控制台中也可以显示异常，而这些异常有时不会再终端中显示。例如，如果在“运行”视图的“调用堆栈”区域中看到“在异常时暂停”消息，请切换到调试控制台以查看异常消息。

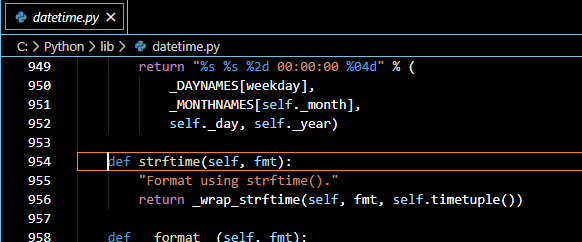
8.继续调试（F5）



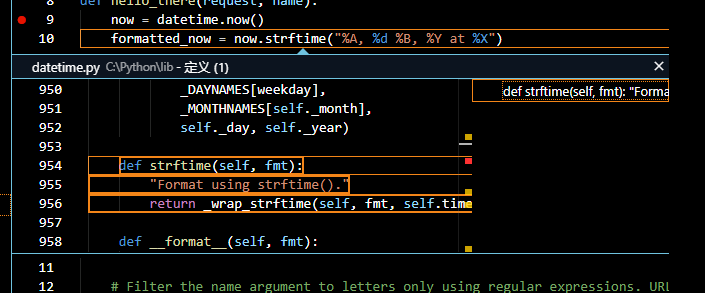
### 查看函数定义

两种方式：

1.鼠标放置于函数或方法，右键“转到定义”；或F12



2. 鼠标放置于函数或方法，右键“快速查看”=》“查看定义”；或Alt+F12

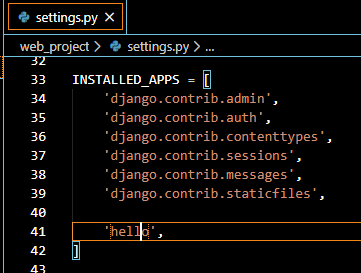


### 使用模板（template）渲染页面

* 生成一个页面可以有两种方式：1.使用Python代码；2.使用模板。
* 通常避免直接使用Python代码生成页面，因为这样会将app直接暴露在跨站点脚本攻击（cross-site scripting (XSS) attacks）的风险中。
* 更好的做法是使用模板（template）完全将HTML代码与Python代码分离。Python代码专注于处理数据；HTML代码专注于页面渲染。
* Django模板中的HTML文件包含占位符，预留给后台数据的加载。
* Django模板可以继承，在父页面处理通用标记，子页面处理各自专有的标记。

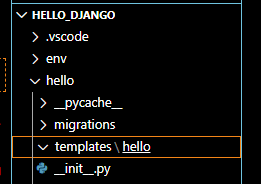
#### 1.在project设置中添加注册app

打开web\_project/settings.py文件，在INSTALLED\_APPS列表中添加当前app名称。



#### 2.经典两层文件夹结构

在app文件夹中创建templates文件夹，然后在templates文件夹中创建app名称的文件夹。



注：这种两层文件夹结构是典型的Django约定

#### 3.创建HTML模板文件

在templates/hello文件夹中创建hello\_there.html.

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8" />

        <title>Hello, Django</title>

    </head>

    <body>

        <strong>Hello there, {{ name }}!</strong> It's {{ date | date:"l, d F, Y" }} at {{ date | time:"H:i:s" }}

    </body>

</html>

注释：

1.页面包含两个数据值的占位符，分别是“name”和“date”。

2.页面中使用“|”来提示HTML显示时使用的格式形式。

#### 4.修改视图views文件

声明调用渲染render

from django.shortcuts import render

修改hello\_there 函数：

def hello\_there(request, name):

    return render(

        request,

        'hello/hello\_there.html',

        {

            'name': name,

            'date': datetime.now()

        }

    )

注释：

1.调用django.shortcuts.render方法加载模板并向模板提供模板的上下文context。

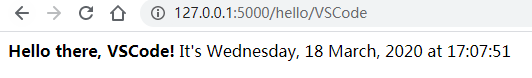
2.上下文context是变量集合，用于向模板传递变量。

3.render的参数：1.request是请求的对象；2.引用模板的路径；3.上下文集合context

4.在实践中模板名和视图中函数名通常是相同的。

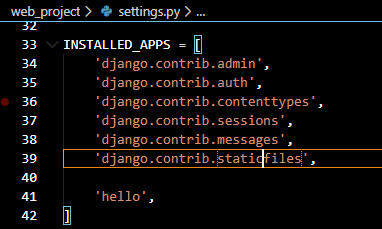
5.在以上代码中，Views只专注处理数据值，Template专注于页面的标记和格式。

#### 5.运行（Ctrl+F5）。



### 引用静态文件

* 静态文件是在Web应用中针对某些请求而按原样返回的内容，例如CSS文件。
* 引用静态文件需要在settings.py文件的INSTALLED\_APPS列表中包含django.contrib.staticfiles。这通常包含在默认设置里。



* 在实际部署中，对静态文件引用设置比在Django开发服务器的设置要复杂，并需要将settings.py文件设置DEBUG=False。（关闭调试模式）

#### 1.修改路由：在Project的路由文件中添加引用路由

修改web\_project/urls.py文件，添加引用：

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

添加路由格式：

urlpatterns += staticfiles\_urlpatterns()

#### 2.添加静态文件：在App文件夹中添加静态文件

1.在hello文件夹下创建static文件夹；在static文件夹下创建hello文件夹。

* 注：这种两层的结构，能避免多个app之间的冲突。因为每个app在static中都有专有的文件夹。

2.在static/hello文件夹中添加css文件site.css

.message {

    font-weight: 600;

    color: blue;

}

#### 3.修改模板：在模板中添加引用和修改HTML代码

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8" />

        <title>Hello, Django</title>

        {% load static %}

        <link rel="stylesheet" type="text/css" href="{% static 'hello/site.css' %}" />

    </head>

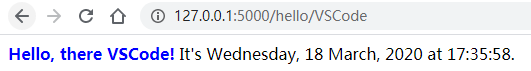
    <body>

        <span class="message">Hello, there {{ name }}!</span> It's {{ date | date:'l, d F, Y' }} at {{ date | time:'H:i:s' }}.

    </body>

</html>

#### 4.运行



### 使用collectstatic 命令

* 在web部署时，常常要将所有app的静态文件集合到一个单一文件夹中。在Django中可以使用命令python manage.py collectstatic。

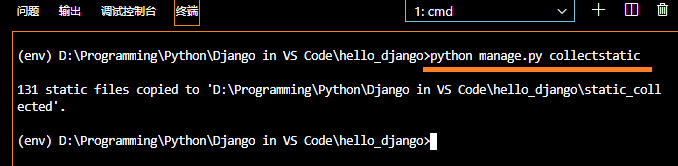
#### 1.在settings.py文件中设置静态文件路径STATIC\_ROOT

添加代码：

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static\_collected')

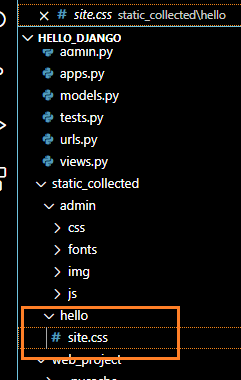
#### 2.在终端中执行collectstatic 命令：

python manage.py collectstatic



执行的结果是：

* 在project文件夹下创建一个文件夹，其文件夹名就是settings.py文件STATIC\_ROOT的值。
* 将project下的所有app的静态文件复制到目标文件夹中。



注：通常在部署前执行该命令。

### 创建一个可供多个子模板引用的父模板

#### 1.创建父模板（base page template）及其样式

* 父模板通常包含一系列通用部分，如CSS文件引用，脚本文件引用等；还包含block标签，让子模板能自定义其页面内容。
* block标签由{% block <name> %} 和 {% endblock %}定义，并对应地标记在父模板和子模板中。

1.在template/hello文件夹中创建layout.html。

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8"/>

    <title>{% block title %}{% endblock %}</title>

    {% load static %}

    <link rel="stylesheet" type="text/css" href="{% static 'hello/site.css' %}"/>

</head>

<body>

<div class="navbar">

    <a href="{% url 'home' %}" class="navbar-brand">Home</a>

    <a href="{% url 'about' %}" class="navbar-item">About</a>

    <a href="{% url 'contact' %}" class="navbar-item">Contact</a>

</div>

<div class="body-content">

    {% block content %}

    {% endblock %}

    <hr/>

    <footer>

        <p>© 2018</p>

    </footer>

</div>

</body>

</html>

注释： {% url %}标记用了引用路由设置中对应的url名称，而不是相对路径。

2. 添加static/hello/site.css样式文件

.navbar {

    background-color: lightslategray;

    font-size: 1em;

    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;

    color: white;

    padding: 8px 5px 8px 5px;

}

.navbar a {

    text-decoration: none;

    color: inherit;

}

.navbar-brand {

    font-size: 1.2em;

    font-weight: 600;

}

.navbar-item {

    font-variant: small-caps;

    margin-left: 30px;

}

.body-content {

    padding: 5px;

    font-family:'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}

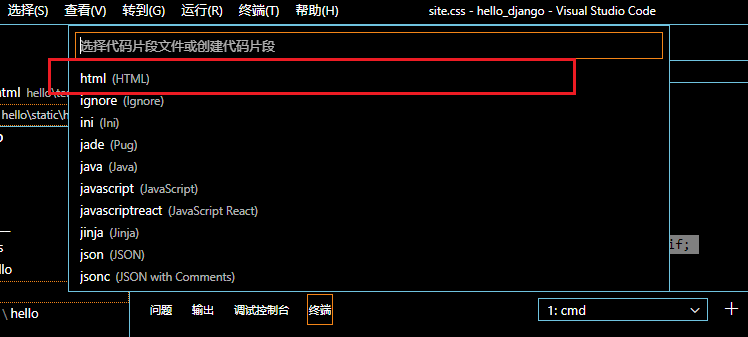
#### 2.创建一个代码片段（code snippet）

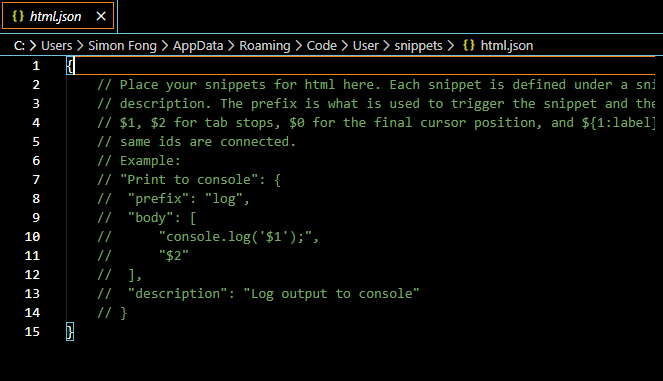
* 创建一个代码片段可以节省初始化一个新模板（template）的时间，提供一个一致的代码片段指向一个单一的来源，减少因赋值代码时产生的错误。

1.文件=》首选项=》用户代码片段



2.选择html，VS Code会自动创建或打开原有的html.json文件





3.修改html.json

{

    "Django Tutorial: template extending layout.html": {

        "prefix": "djextlayout",

        "body": [

            "{% extends \"hello/layout.html\" %}",

            "{% block title %}",

            "$0",

            "{% endblock %}",

            "{% block content %}",

            "{% endblock %}"

        ],

        "description": "Boilerplate template that extends layout.html"

    },

}

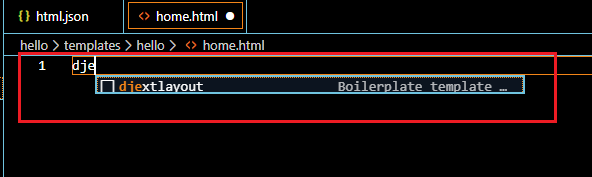
以上设定的目的：

当开始输入代码片段前缀时，如该处的djext，VS Code将自动提供设定的代码片段。还可以使用Insert Snippet命令选择一段代码片段。

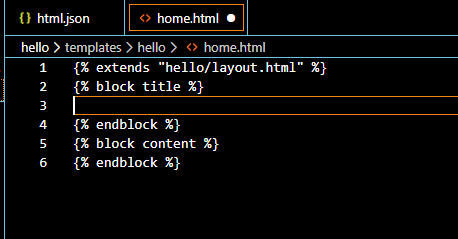
#### 3.创建子模板并应用代码片段

1.在template/hello文件夹中创建子模板home.html

向该模板输入djext时，将会提示设定好的代码片段



可以直接按回车键，则代码片段会自动加载在模板中。



然后继续输入其余代码：

{% extends "hello/layout.html" %}

{% block title %}

<title>Home</title>

{% endblock %}

{% block content %}

<p>Home page for the Visual Studio Code Django tutorial.</p>

{% endblock %}

2.在template/hello文件夹中创建子模板about.html、contact.html

about.html

{% extends "hello/layout.html" %}

{% block title %}

<title>About us</title>

{% endblock %}

{% block content %}

<p>About page for the Visual Studio Code Django tutorial.</p>

{% endblock %}

contact.html

{% extends "hello/layout.html" %}

{% block title %}

<title>Contact us</title>

{% endblock %}

{% block content %}

<p>Contact page for the Visual Studio Code Django tutorial.</p>

{% endblock %}

#### 4.修改app的路由文件urls.py

    path("about/", views.about, name="about"),

    path("contact/", views.contact, name="contact"),

5.修改视图文件views.py

# Replace the existing home function with the one below

def home(request):

    return render(request, "hello/home.html")

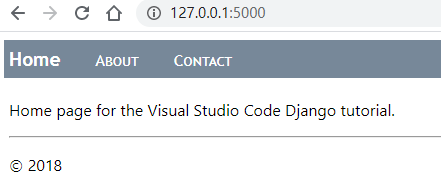
def about(request):

    return render(request, "hello/about.html")

def contact(request):

    return render(request, "hello/contact.html")

#### 5.运行



### 处理数据，数据模型和迁移

* Django使用模型（models）将对象映射到数据库。模型是一个派生于django.db.models.Model的Python类。
* app的模型类放置在models.py。

通常的流程是：

1.修改models.py文件。

2.运行命令python manage.py makemigrations，在migration文件夹中生成一个新的migration脚本，用于记录Django中对数据对象的修改

3.运行命令python manage.py migrate，执行最新的migration脚本，更新数据库。

* 通常，程序员只能使用模型类来储存、检索数据，而具体的细节由Django执行。除此之外，当需要初始化数据库数据集时，可以在执行migration命令初始化数据库框架后，使用Django管理工具loaddata命令，将数据集写入数据库。
* Django默认使用db.sqlite3数据库，可以使用如SQLite browser等工具直接在数据表中添加、删除记录。但必须避免修改数据库框架，否则将不能和app定义的模型同步。即若需要修改数据库框架，只能使用makemigrations和migrate命令。
* 在开发环境中默认使用db.sqlite3数据库。sqlite数据库能有效处理中小流量的网站（每日小于100k次访问量）。同时该类型数据库只适用于单台计算机，不能在多服务器场景下使用。
* 在实际的应用中，则需要考虑使用如MySQL、SQL Server等类型的数据库。

#### 定义模型

* Django中所有模型类都是派生自django.db.model.Models，并均存放在models.py文件中。
* 在数据库中，每个模型将自动生成一个唯一的ID字段，字段名为“id”。
* 其他字段类型有CharField（有限文本）、TextField（无限文本）、EmailField（电子邮箱）、RLField（url）、IntegerField（整数）、DecimalField（小数）、BooleanField（布尔值）、DateTimeField（日期时间）、ForeignKey（外键）和ManyToMany（多对多）等。
* 每个字段可以带有属性值，如max\_length。
* blank=True：字段是可选的。
* null=true：允许空值。
* choises：指明字段值只能在设定的若干值中选择。

实例：

from django.db import models

from django.utils import timezone

class LogMessage(models.Model):

    message = models.CharField(max\_length=300)

    log\_date = models.DateTimeField("date logged")

    def \_\_str\_\_(self):

        """Returns a string representation of a message."""

        date = timezone.localtime(self.log\_date)

        return f"'{self.message}' logged on {date.strftime('%A, %d %B, %Y at %X')}"

注释：

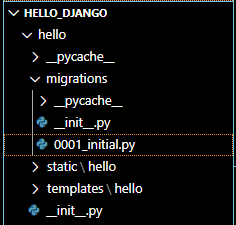
* 模型可以包含方法
* 模型通常都包含一个\_\_str\_\_方法，用于返回一个字符串实例。

#### 迁移（Migrate ）到数据库

在终端中进入project文件夹，然后执行生成迁移命令：

python manage.py makemigrations

VS Code会在app的migration文件夹下创建一个迁移脚本



执行迁移命令：



注：有时迁移时会出错，可以检查打开的终端是否调试的终端。

#### 实例：通过模型操作数据库

实例目标：

1.添加一个app页面，用于提交消息

2.修改home页面，用于显示这些消息。

步骤：

1.定义表单form

在hello文件夹中创建表单文件forms.py。Django表单包含模型中的字段，存放在forms.py文件中。

from django import forms

from hello.models import LogMessage

class LogMessageForm(forms.ModelForm):

    class Meta:

        model = LogMessage

        fields = ("message",)   # NOTE: the trailing comma is required

2.创建模板template

在template/hello文件夹中创建模板log\_message.html：

{% extends "hello/layout.html" %}

{% block title %}

    Log a message

{% endblock %}

{% block content %}

    <form method="POST" class="log-form">

        {% csrf\_token %}

        {{ form.as\_p }}

        <button type="submit" class="save btn btn-default">Log</button>

    </form>

{% endblock %}

注：

* 模板中包含一个表单form
* {% csrf\_token %}是一个防御机制，防止跨网页请求仿冒。

3.在CSS中添加表单样式

input[name=message] {

    width: 80%;

}

4.修改url文件，添加路由：

    path("log/", views.log\_message, name="log"),

5.修改视图views：

from django.shortcuts import redirect

from hello.forms import LogMessageForm

from hello.models import LogMessage

def log\_message(request):

    form = LogMessageForm(request.POST or None)

    if request.method == "POST":

        if form.is\_valid():

            message = form.save(commit=False)

            message.log\_date = datetime.now()

            message.save()

            return redirect("home")

    else:

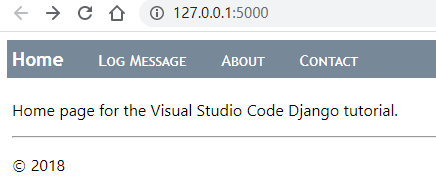
        return render(request, "hello/log\_message.html", {"form": form})

6.在导航栏中添加新页面入口

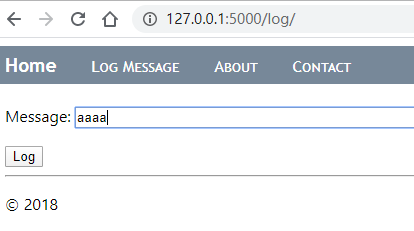
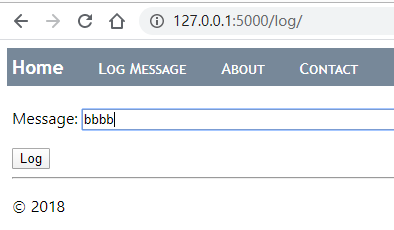
    <!-- Insert below the link to Home -->

    <a href="{% url 'log' %}" class="navbar-item">Log Message</a>

7.运行



8提交任意值。

9.停止运行app。

10.修改Home模板，显示所有提交值。

{% extends "hello/layout.html" %}

{% block title %}

    Home

{% endblock %}

{% block content %}

    <h2>Logged messages</h2>

    {% if message\_list %}  <!-- 如果页面接受到任何数据，则生成表格 -->

        <table class="message\_list">

            <thead>

            <tr>

                <th>Date</th>

                <th>Time</th>

                <th>Message</th>

            </tr>

            </thead>

            <tbody>

            {% for message in message\_list %}  <!-- 循环message列表 -->

                <tr>

                    <td>{{ message.log\_date | date:'d M Y' }}</td>

                    <td>{{ message.log\_date | time:'H:i:s' }}</td>

                    <td>

                        {{ message.message }}

                    </td>

                </tr>

            {% endfor %}

            </tbody>

        </table>

    {% else %}      <!-- 如果页面没有接受到任何数据，则生成提示信息 -->

        <p>No messages have been logged. Use the <a href="{% url 'log' %}">Log Message form</a>.</p>

    {% endif %}

{% endblock %}

11.修改CSS文件

.message\_list th,td {

    text-align: left;

    padding-right: 15px;

}

12.修改视图views，

引入新的类

from django.views.generic import ListView

删除原有的Home函数，创建HomeListView类：

class HomeListView(ListView):       #该类派生自ListView

    """Renders the home page, with a list of all messages."""

    model = LogMessage      #声明model属性与LogMessage模型绑定

    def get\_context\_data(self, \*\*kwargs):       #定义方法，用于生成模板需要的context

        context = super(HomeListView, self).get\_context\_data(\*\*kwargs)

        return context

13.修改路由urls.py

添加引用：

from hello.models import LogMessage

定义变量：

home\_list\_view = views.HomeListView.as\_view(

    queryset=LogMessage.objects.order\_by("-log\_date")[:5],  # :5 limits the results to the five most recent

    context\_object\_name="message\_list",

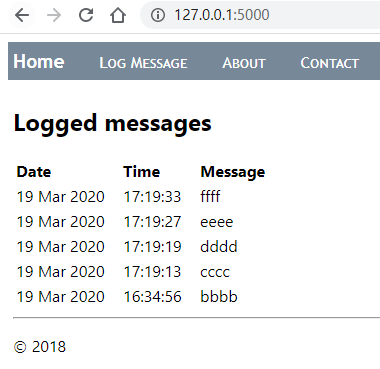
    template\_name="hello/home.html",

)

删除原有home页面的路由，插入新路由：

    path("", home\_list\_view, name="home"),

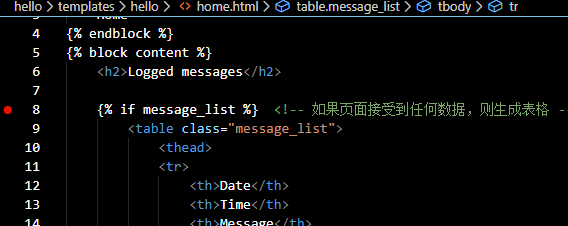
14.运行。



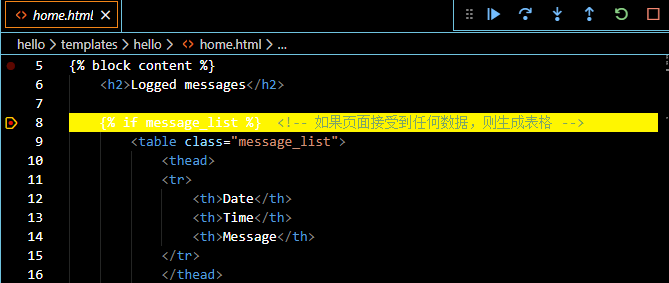
### 在模板中使用调试

当调试配置launch.json文件中的"django": true时，VS Code可以在模板中进行调试。

#### 1.在模板中设置断点



#### 2.运行调试



### 为开发环境创建requirements.txt文件

requirements.txt文件描述应用程序的依赖项。当应用程序在多个开发者之间进行共享时，使用requirements.txt文件可以描述应用程序的依赖项，而不需要复制虚拟环境中所有的文件。

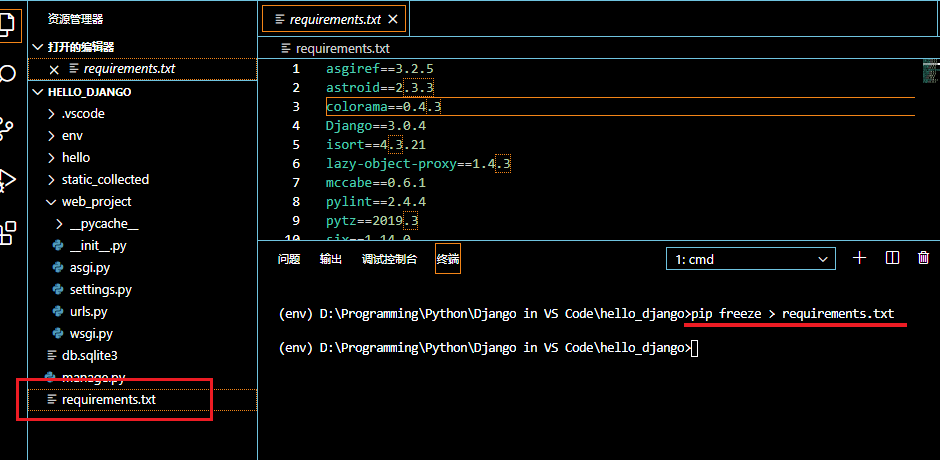
#### 创建requirements.txt文件

requirements.txt文件通常使用pip freeze命令创建：

1.进入设定的虚拟环境并打开终端。

2.执行命令pip freeze > requirements.txt

会自动在project文件夹中创建requirements.txt文件，当中记录了虚拟环境中的参数。



#### 应用requirements.txt文件

使用pip install -r requirements.txt命令，VS Code会根据requirements.txt文件自动创建虚拟环境。

### 创建超级管理员并启用管理界面

Django自带可供认证的管理员界面。

管理员界面通过内建的django.contrib.admin模块实际执行，并在project设置settings.py文件中的INSTALLED\_APPS声明。

认证由内建的django.contrib.auth模块实际执行，并在project设置settings.py文件中的INSTALLED\_APPS声明。

#### 1.创建超级管理员账户

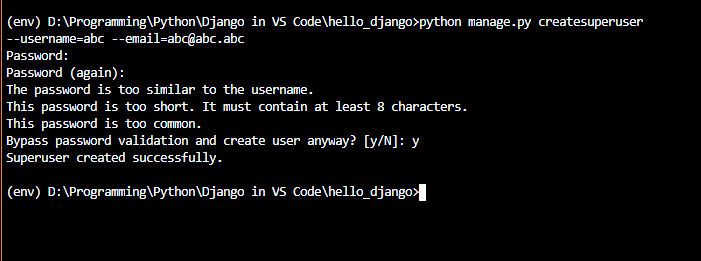
在虚拟环境终端中执行命令：

python manage.py createsuperuser --username=<username> --email=<email>

username：用户名

email：邮箱

执行命令过程中会提示输入密码。

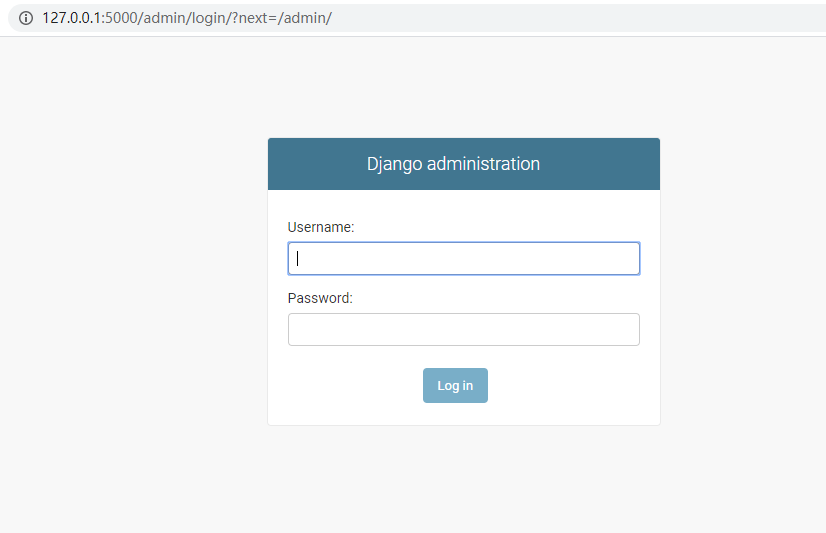


2.修改project级别的路由urls.py

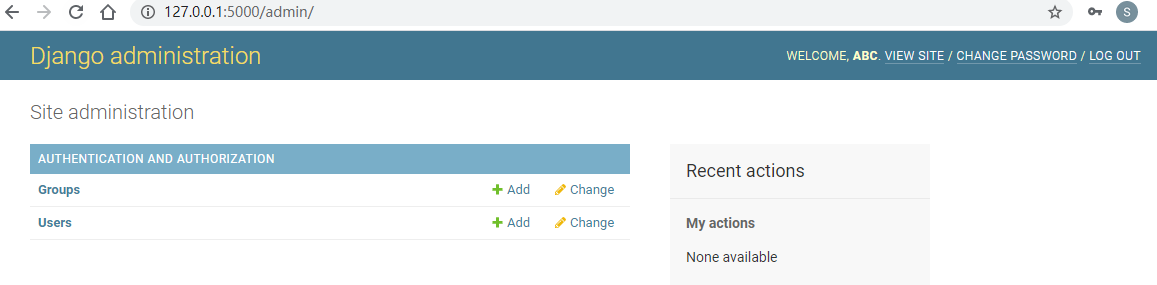
添加admin界面路由

path('admin/', admin.site.urls),

3.运行并导航到/admin



4.输入用户名和密码，进入管理员界面。



### 延伸：使用Docker扩展（未学习）

# 命令索引

|  |  |  |
| --- | --- | --- |
| 指令 | 目的 | 说明 |
| django-admin.py startproject <project\_name> [path] | 创建Django Project | project\_name:项目名称  path：安装路径。通常使用“.”表示在当前文件夹中创建 |
| python manage.py startapp <app\_name> [path] | 创建Django App | app\_name:应用名称  path：安装路径 |
| python manage.py runserver [port] | 启动开发服务器() | port：端口，默认为8000 |
| python -m venv <environment\_name> | 创建专用虚拟环境 | environment\_name：定义虚拟环境文件夹名 |
| <environment\_name>\scripts\activate | 激活虚拟环境 | environment\_name：定义虚拟环境文件夹名 |
| python -m pip install <package\_name> | 安装库 | package\_name：Python库名称 |
| python manage.py -h <command\_name> | 查看指定命令的使用方法 | command\_name：命令名称 |
| python manage.py | 查看所有使用方法的列表 |  |
| python manage.py collectstatic | 把静态目录下的所有文件拷贝至 STATIC\_ROOT 目录 |  |
| python manage.py makemigrations | 生成migration脚本 |  |
| python manage.py migrate | 将migration脚本迁移到数据库 |  |
| pip freeze | 列出当前环境中安装的库和版本 |  |
| pip freeze > <file\_name> | 将当前环境中安装的库信息输出到文件 | <file\_name>：输出的文件名，通常是requirement.txt |
| pip install -r <file\_name> | 根据指定文件，在当前环境中自动安装库 | <file\_name>：包含安装库信息的文件名 |
| python manage.py createsuperuser --username=<username> --email=<email> | 创建超级管理员 | <username>：用户名  <email>:邮箱 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 扩展：

**django-admin工具：**

Django的任务管理命令行工具。

**语法：**

django-admin <command> [options]

该命令等价于：

manage.py <command> [options] 和 py -m django <command> [options]

**显示帮助信息：**

**django-admin help** ：显示用法信息和每个应用程序提供的命令列表。

**django-admin help --commands** ：显示所有可用命令的列表。

**django-admin help <command>** ：显示给定命令的描述及其可用选项的列表。

# END