# 学习网址

https://docs.djangoproject.com/zh-hans/5.2/

https://www.runoob.com/django/django-intro.html

# 官网

https://www.djangoproject.com/

# 文档地址

https://docs.djangoproject.com/zh-hans/5.2/

# 基本概念

Django 是一个由 Python 编写的一个开放源代码的 Web 应用框架。使用 Django，只要很少的代码，Python 的程序开发人员就可以轻松地完成一个正式网站所需要的大部分内容，并进一步开发出全功能的 Web 服务。

Django 遵循 MVC（Model-View-Controller）架构，但在 Django 中更常被称为 MTV（Model-Template-View）。

# Django的哲学

## 1、DRY（Don't Repeat Yourself）

避免重复代码，提倡复用（如模板继承、模型继承）。

## 2、约定优于配置

默认提供合理配置（如自动生成 Admin 界面），减少决策成本。

## 3、快速开发

从原型到生产环境均可高效推进

# 内置功能



# 核心特点

## 1、快速开发

Django 提供了大量内置功能，如认证、管理后台、表单处理等，让开发者专注于业务逻辑，而非底层实现。

## 2、自动化管理后台

只需简单的模型定义，即可生成强大的后台管理界面，支持增删改查。

## 3、ORM 数据库映射

Django 内置 ORM (Object-Relational Mapping)，可以让开发者使用 Python 类与数据库交互，无需编写 SQL。

## 4、强大的 URL 路由

使用正则表达式灵活定义 URL，轻松实现页面路由。

## 5、模板引擎

内置强大的模板系统，支持逻辑判断、循环处理，方便渲染 HTML 页面。

## 6、国际化支持

Django 支持多语言国际化，非常适合全球化应用。

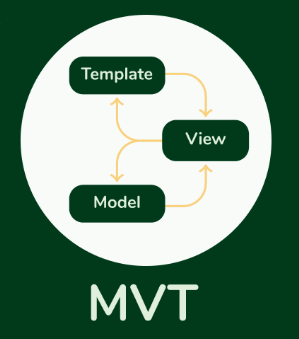
## 7、高安全性

内置多种安全保护措施，如防止 SQL 注入、XSS 攻击、CSRF 攻击等。

## 8、丰富的社区与扩展

大量开源的第三方库，如 Django REST framework、Django CMS 等，快速扩展功能。

# MVT模型



Django 中采用了 **MVT** 设计模式，类似于 MVC，但有一些区别

## Model (模型)

与数据库交互，处理数据的创建、读取、更新、删除。

## Template (模板)

负责页面渲染，生成最终的 HTML 内容。

## View (视图)

Django 的 View 更偏向于控制器的角色，接收请求并决定使用哪个模板和数据。

## 流程

1、用户访问 URL，请求被 Django 的 urls.py 映射到相应的 View。

2、View 处理业务逻辑，调用 Model 获取数据。

3、View 将数据传递给 Template。

4、Template 渲染 HTML，最终返回给用户。

# Django的适用场景

1、内容管理系统（CMS）（如新闻网站、博客）

2、社交平台（用户系统、动态发布）

3、电子商务网站（订单管理、支付集成）

4、API 后端（结合 Django REST framework）

# 不适合的场景

1、超高性能要求的实时系统（如高频交易平台，推荐 Go 或 Rust）

2、极度轻量级的微服务（可以考虑 Flask 或 FastAPI）

# Django vs 其他框架



# Django优势

## 1、开箱即用

无需额外安装插件即可实现完整功能。

## 2、安全性

自动防范 SQL 注入、XSS、CSRF 等常见攻击。

## 3、社区活跃

丰富的第三方库（如 Django REST framework、Django CMS）

# Django安装

## 安装方法

pip install Django

安装完成后，你可以通过运行以下命令验证 Django 是否成功安装：

python3 -m django --version

## 创建项目

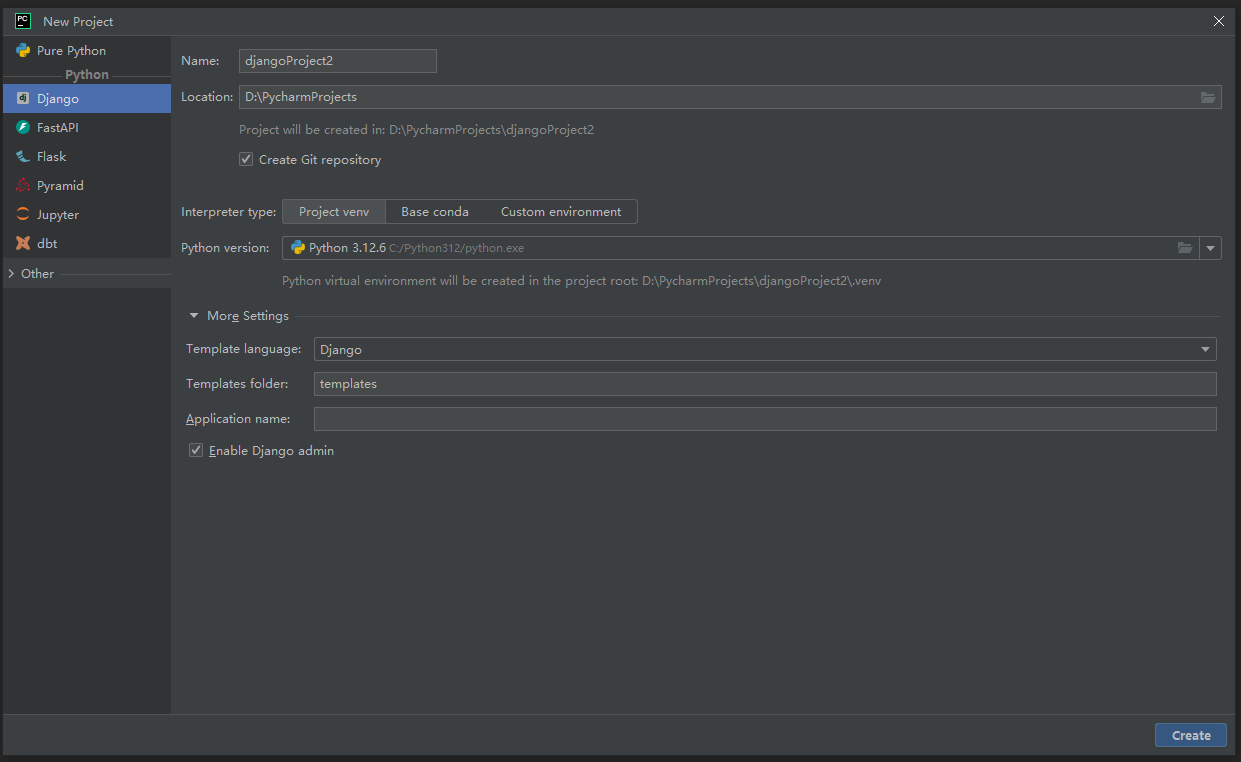
### 使用命令行创建项目

django-admin startproject 项目名称

### 使用 Pycharm 创建项目

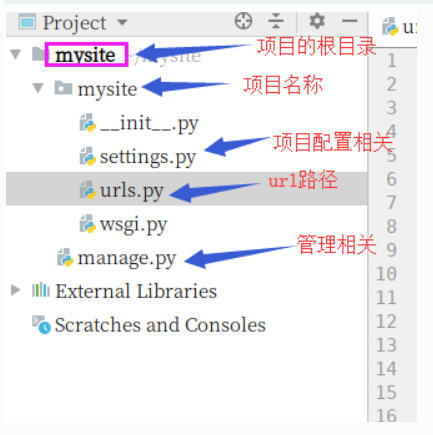
#### 操作步骤

file ---> new project ---- 选择Django ---> 配置路径和项目名称 ---> 配置环境(默认用系统环境) ----> 点击create(完成创建)



#### ****项目目录介绍****

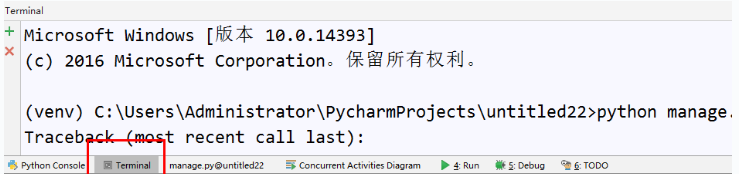


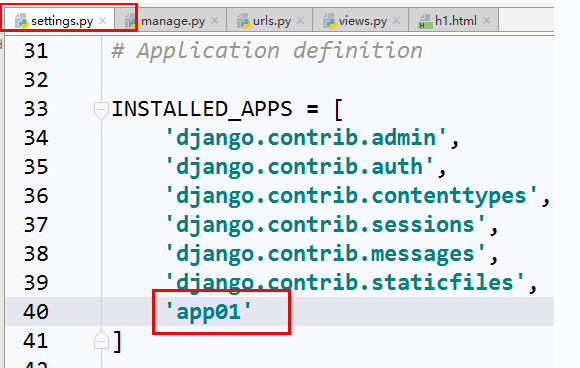


## 创建应用

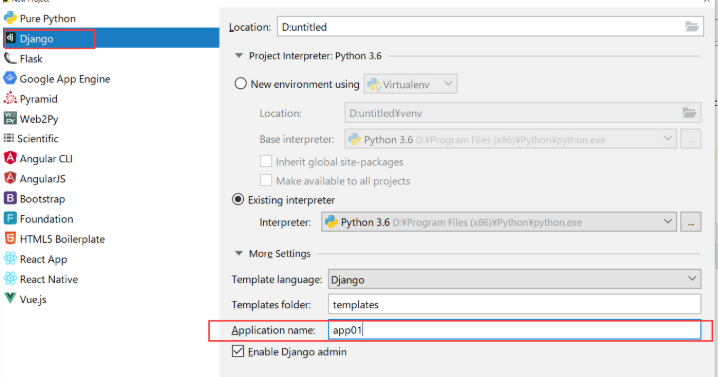
### 使用命令行模式创建应用

**python manage.py startapp** 应用名



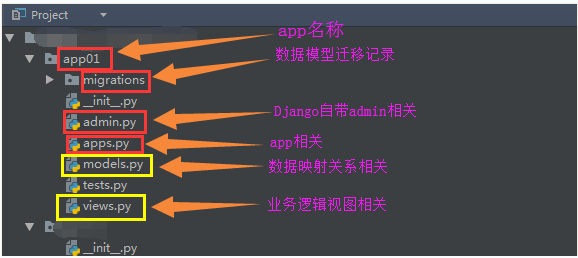


### ****使用 Pycharm 创建应用****



注意：使用命令行创建的应用，一定要在 **settings.py-->INSTALLED\_APPS** 中加入你的应用名字进行注册, 使用Pycharm创建的应用则不用。

### 认识应用结构



## 项目与应用的关系

### 核心概念

1、项目 (Project) = 整个网站

2、应用 (App) = 网站中的功能模块

### 关系类比

项目 = 一栋大楼

应用 = 大楼里的各个房间

大楼提供基础设施，房间实现具体功能

### 层次结构



### 职责分工

#### 项目负责

1、全局配置 (settings.py)

2、路由分发 (urls.py)

3、数据库设置

4、中间件配置

#### 应用负责

1、具体功能实现

2、数据模型 (models.py)

3、业务逻辑 (views.py)

4、模板文件

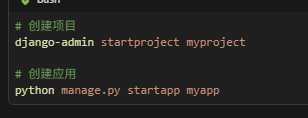
### 数量关系

1、1个项目 : N个应用

2、项目必须有，应用可选

3、应用可以在不同项目间复用

### 创建命令



### 配置关系



### 设计原则

#### 1、单一职责

每个应用只负责一个功能领域

#### 2、高内聚低耦合

应用内部紧密相关，应用间松散连接

#### 3、可复用性

应用可以在不同项目中重复使用

## 运行项目

### 使用命令行运行项目

python manage.py runserver

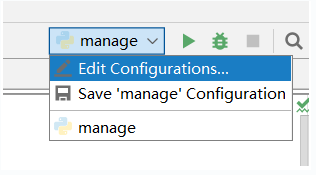
#指定端口号运行

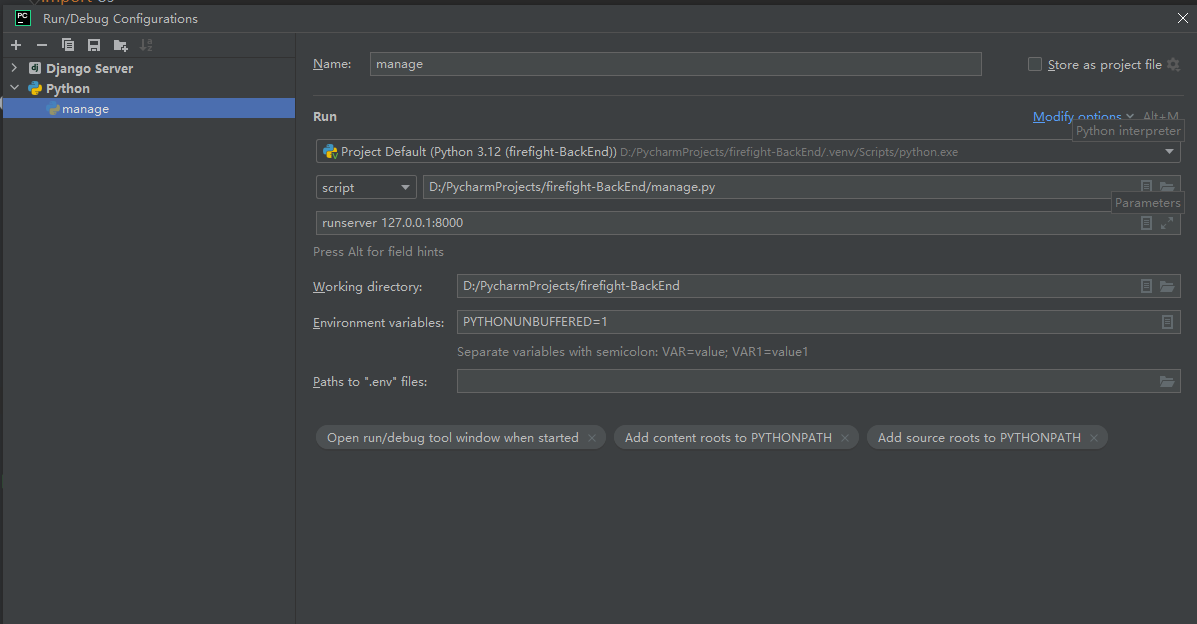
python manage.py runserver 80

**注意：**要在manage.py同级目录执行命令。

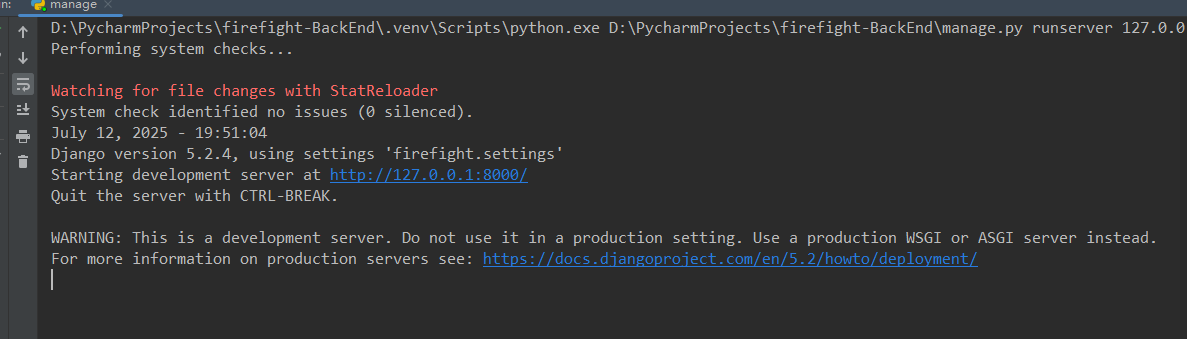
### Pycharm 启动项目

点击右上角的绿色的三角按钮。如需更改配置信息，请下拉选择 edit configurations，进行配置。





配置完以上信息之后，直接按 Ctrl+shift+F10 运行一下 manage.py 文件。



# Django创建第一个项目

## Django 管理工具

安装 Django 之后，您现在应该已经有了可用的管理工具 django-admin，Windows 如果没有配置环境变量可以用 django-admin。

我们可以来看下django-admin 的命令介绍



## 创建第一个项目

### 操作步骤

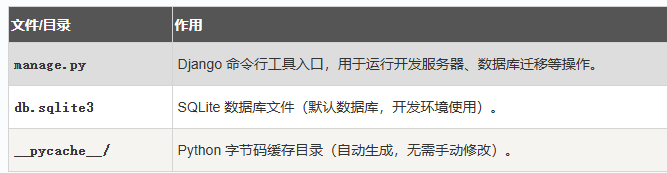
使用 django-admin 来创建 HelloWorld 项目

django-admin startproject HelloWorld

创建完成后我们可以查看下项目的目录结构



### ****项目根目录 (HelloWorld/)****



### ****主项目配置目录 (HelloWorld/HelloWorld/)****

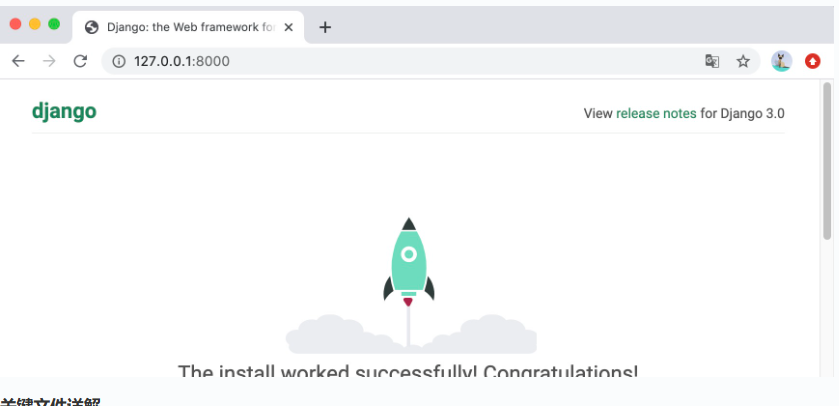


## 启动服务器

我们进入 HelloWorld 目录输入以下命令，启动服务器：

python3 manage.py runserver 0.0.0.0:8000

0.0.0.0让其它电脑可连接到开发服务器，8000 为端口号。如果不说明，那么端口号默认为 8000。在浏览器输入你服务器的 ip（这里我们输入本机 IP 地址： 127.0.0.1:8000） 及端口号，如果正常启动，输出结果如下：



## 关键文件详解

### ****settings.py（核心配置）****



### ****urls.py（路由配置）****



### ****manage.py（项目管理脚本）****



## 视图和URL配置

在先前创建的 HelloWorld 目录下的 HelloWorld 目录新建一个 views.py 文件，并输入代码：



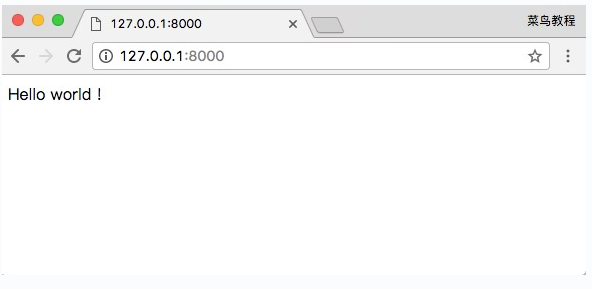
接着绑定 URL 与视图函数。打开 urls.py 文件，删除原来代码，将以下代码复制粘贴到 urls.py 文件中：



整个目录结构：



完成后，启动 Django 开发服务器，并在浏览器访问打开浏览器并访问



**注意：**项目中如果代码有改动，服务器会自动监测代码的改动并自动重新载入，所以如果你已经启动了服务器则不需手动重启。

## path()函数

### 基本概念

Django path() 可以接收四个参数，分别是两个必选参数：route、view 和两个可选参数：kwargs、name。

### 语法

path(route, view, kwargs=None, name=None)

**route：** 字符串，定义 URL 的路径部分。可以包含变量，例如 <int:my\_variable>，以从 URL 中捕获参数并将其传递给视图函数。

**view：** 视图函数，处理与给定路由匹配的请求。可以是一个函数或一个基于类的视图。

**kwargs（可选）：** 一个字典，包含传递给视图函数的额外关键字参数。

**name（可选）：** 为 URL 路由指定一个唯一的名称，以便在代码的其他地方引用它。这对于在模板中生成 URL 或在代码中进行重定向等操作非常有用。

# django-admin

## 创建新项目

django-admin startproject 项目名称

这个命令会在当前目录下创建一个新的 Django 项目，包含基本的项目结构：

manage.py：项目管理脚本

项目名称/：项目主目录

\_\_init\_\_.py

settings.py：项目设置文件

urls.py：URL 路由配置

wsgi.py：WSGI 应用入口

## 创建新应用

虽然通常使用 manage.py 来创建应用，但也可以通过 django-admin

django-admin startapp 应用名称

这会创建一个新的 Django 应用，包含：

migrations/：数据库迁移文件目录

\_\_init\_\_.py

admin.py：管理后台配置

apps.py：应用配置

models.py：数据模型定义

tests.py：测试代码

views.py：视图函数

## 检查项目配置

django-admin check

这个命令会检查你的 Django 项目是否有配置错误，包括：

模型定义是否正确

URL 配置是否有效

模板设置是否正确

静态文件配置等

## 数据库迁移

Django 使用迁移系统来管理数据库模式变更：

django-admin makemigrations # 创建迁移文件

django-admin migrate # 应用迁移到数据库

## 创建超级用户

django-admin createsuperuser

这个命令会引导你创建一个可以访问 Django 管理后台的超级用户。

## 常用命令

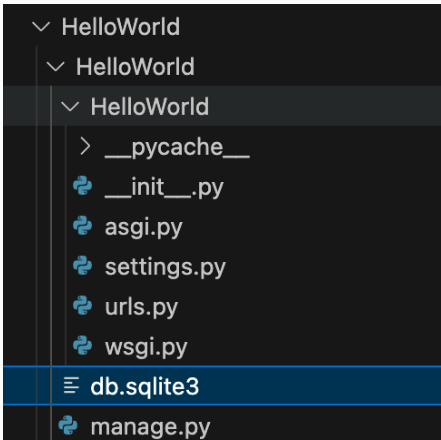


# Django项目结构解析

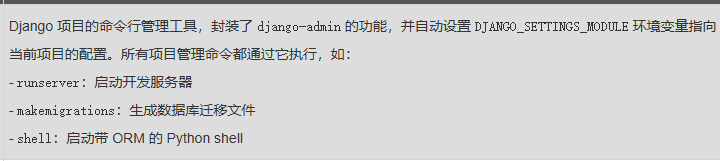
使用 django-admin 来创建HelloWorld项目

django-admin startproject HelloWorld

创建完成后我们可以查看下项目的目录结构



## manage.py

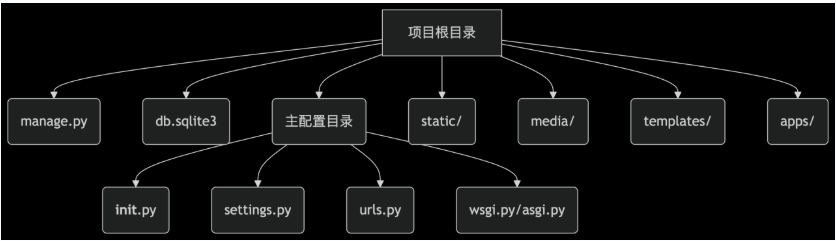


## db.sqlite3

Django 默认使用的 SQLite 数据库文件，在首次执行 migrate 命令后自动生成。开发环境适用，生产环境建议换用 PostgreSQL/MySQL。文件位置由 settings.py 中的 DATABASES['default']['NAME'] 定义。

## \_\_pycache\_\_/

Python 解释器生成的字节码缓存目录，包含 .pyc 文件，用于加速模块加载。无需提交到版本控制（应在 .gitignore 中忽略）。



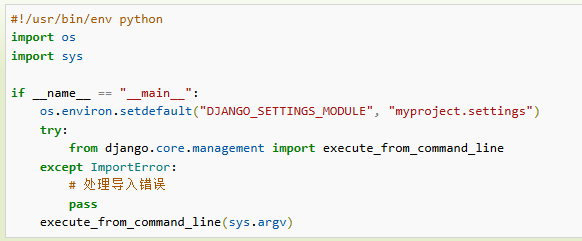
## 核心文件

### manage.py

#### 基本概念

manage.py 是 Django 项目的命令行工具入口，它提供了许多有用的命令。

#### 文件内容



#### 常用命令示例

##### python manage.py runserver

启动开发服务器

##### python manage.py migrate

应用数据库迁移

##### python manage.py createsuperuser

创建管理员账户

### settings.py

#### 基本概念

settings.py 是 Django 项目的配置文件，包含所有重要的设置

#### 文件内容



### urls.py（URL 调度中心）

#### 基本概念

路由调度中心

#### 文件内容



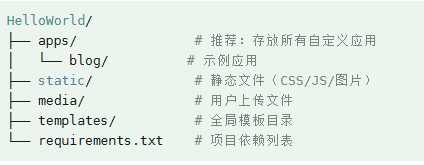
include()：实现路由模块化，将不同应用的路由分离到各自的 urls.py。

namespace：用于反向解析 URL 时避免命名冲突。

### wsgi.py & asgi.py



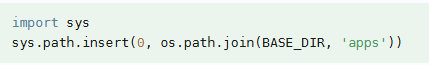
## 扩展目录



### apps/ 目录（推荐结构）

将应用集中管理，避免散落在项目根目录。

需在 settings.py 中配置 Python 路径



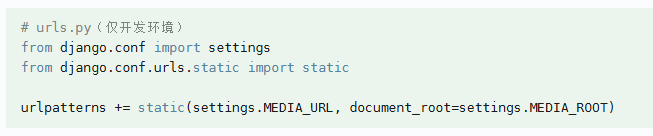
### 静态文件与媒体文件

#### static/

存放 CSS、JavaScript、图片等，通过 STATIC\_URL 访问。

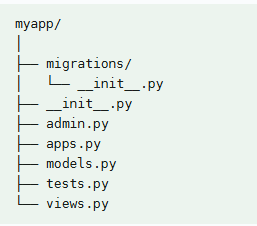
#### media/

用户上传的文件（如头像），通过 MEDIA\_URL 访问。需配置服务器在开发时提供访问：



## Django应用结构

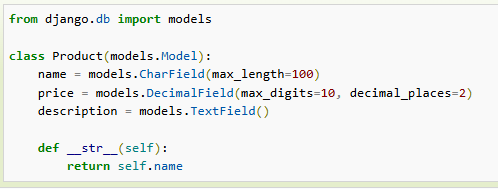
在 Django 中，一个项目可以包含多个应用。使用 python manage.py startapp myapp 创建一个新应用后，会生成以下结构



### 主要应用文件

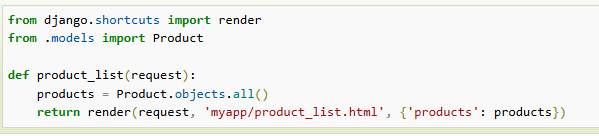
#### models.py

定义数据模型，与数据库表对应



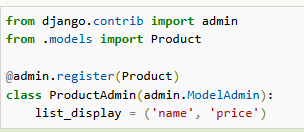
#### views.py

处理业务逻辑，返回响应



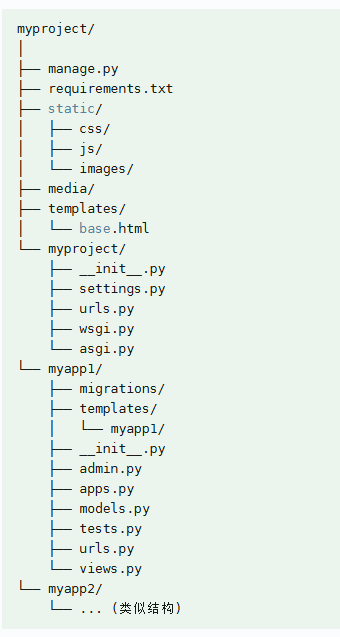
#### admin.py

配置Django管理后台



## 完整项目结构示例

一个典型的 Django 项目完整结构如下：



## 重要目录说明

### migrations目录

存储数据库迁移文件，Django 使用这些文件来跟踪模型变更并同步到数据库。

### static目录

存放静态文件（CSS, JavaScript, 图片等），在部署时会收集到指定位置。

### templates目录

存放 HTML 模板文件，Django 使用模板语言动态生成页面。

### media目录

存储用户上传的文件，需要在settings.py中配置



## 实践建议

### 1、项目与应用分离

保持每个应用的独立性，便于复用

### 2、环境配置

使用不同的 settings 文件区分开发和生产环境

### 3、静态文件管理

开发时使用 STATICFILES\_DIRS，生产时使用 collectstatic

### 4、URL 设计

在应用级别定义 URL，然后在项目级别包含.

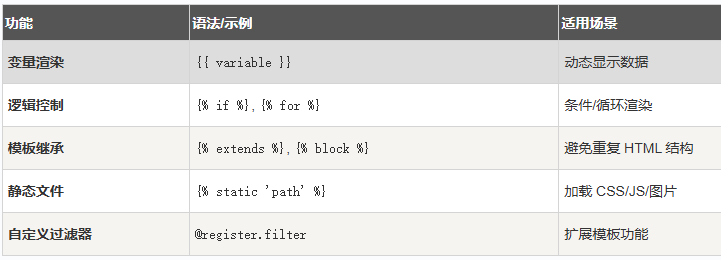
### 5、模板组织

为每个应用创建子目录存放模板。

# 模版

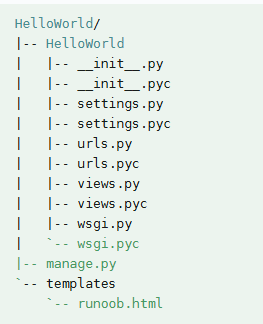
## 基本概念

Django 的模板系统（Template System）是用于将业务逻辑（Python）与展示层（HTML）分离的核心组件，它允许开发者通过简单的标签和变量动态生成 HTML 页面。模板是一个文本，用于分离文档的表现形式和内容。

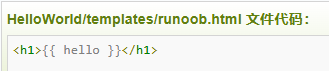


## 模版应用实例

将在 HelloWorld 目录底下创建 templates 目录并建立 runoob.html文件，整个目录结构如下：



runoob.html 文件代码如下



接下来我们需要向 Django 说明模板文件的路径，打开 HelloWorld/HelloWorld/settings.py文件，修改 **TEMPLATES** 中的 DIRS 为 [BASE\_DIR / "templates"]，如下所示:



我们现在修改 views.py，增加一个新的对象，用于向模板提交数据



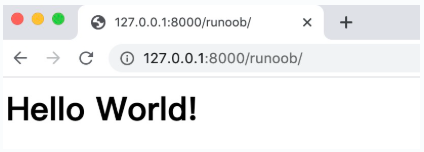
我们这里使用render来替代之前使用的 HttpResponse,render 还使用了一个字典 context 作为参数。context 字典中元素的键值 **hello** 对应了模板中的变量 {{ hello }}



接下来我们进入 HelloWorld 目录输入以下命令，启动服务器

python3 manage.py runserver 0.0.0.0:8000

再次访问 **http://127.0.0.1:8000/runoob**，可以看到页面



这样我们就完成了使用模板来输出数据，从而实现数据与视图分离。

## 模版标签

### 语法

{% tag %}

### 常用的模版标签



## 变量

### 语法

view：｛"HTML变量名" : "views变量名"｝

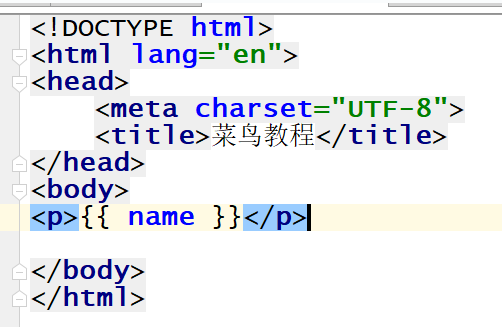
HTML：｛｛变量名｝｝

### 示例



templates 中的 runoob.html ：





再次访问 http://127.0.0.1:8000/runoob，可以看到页面：



## 列表

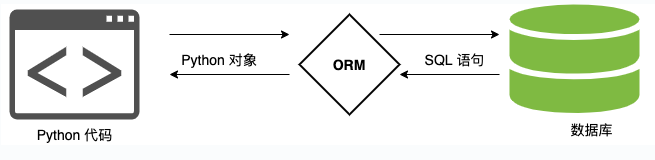
# 模型

## 基本概念

Django 对各种数据库提供了很好的支持，包括：PostgreSQL、MySQL、SQLite、Oracle。Django 为这些数据库提供了统一的调用API。 我们可以根据自己业务需求选择不同的数据库。

## Django ORM

Django 模型使用自带的 ORM。对象关系映射（Object Relational Mapping，简称 ORM ）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。ORM 是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。



### 使用 ORM 的好处

1、提高开发效率。

2、不同数据库可以平滑切换。

### 使用 ORM 的缺点

1、ORM 代码转换为 SQL 语句时，需要花费一定的时间，执行效率会有所降低。

2、长期写 ORM 代码，会降低编写 SQL 语句的能力。

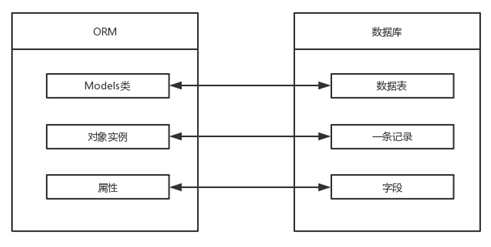
### ORM 解析过程

1、ORM 会将Python代码转成为 SQL 语句。

2、SQL 语句通过pymysql传送到数据库服务端。

3、在数据库中执行SQL语句并将结果返回。

### ORM 对应关系表



## 数据库配置

### 创建数据库

创建 MySQL 数据库( ORM 无法操作到数据库级别，只能操作到数据表)语法

create database 数据库名称 default charset=utf8; # 防止编码问题，指定为 utf8

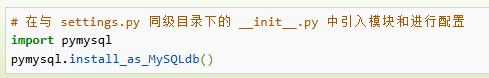
### 配置DATABASES 配置项

我们在项目的 settings.py 文件中找到 DATABASES 配置项，将其信息修改为：



### 使用pymysql模块连接mysql数据库

在与 settings.py 同级目录下的 \_\_init\_\_.py 中引入模块和进行配置



## 定义模型

### 创建应用APP

Django 规定，如果要使用模型，必须要创建一个 app。我们使用以下命令创建一个 TestModel 的 app

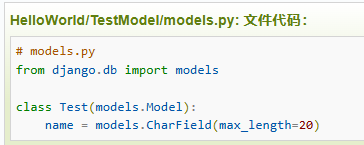
django-admin startapp TestModel

目录结构如下：



### 修改models.py文件

我们修改 TestModel/models.py 文件，代码如下：



以上的类名代表了数据库表名，且继承了models.Model，类里面的字段代表数据表中的字段(name)，数据类型则由CharField（相当于varchar）、DateField（相当于datetime）， max\_length 参数限定长度。

### 安装app

在 settings.py 中找到INSTALLED\_APPS这一项，如下：



### 在命令行运行



看到几行 "Creating table…" 的字样，你的数据表就创建好了。

表名组成结构为：应用名\_类名（如：testmodel\_test）



注意：尽管我们没有在 models 给表设置主键，但是 Django 会自动添加一个 id 作为主键。

## 数据库操作

我们在 HelloWorld 目录中添加 testdb.py 文件（下面介绍），并修改urls.py



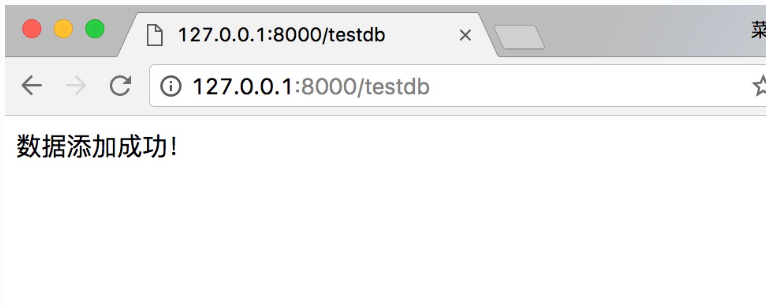
### 添加数据

添加数据需要先创建对象，然后再执行save函数，相当于SQL中的INSERT。



访问 **http://127.0.0.1:8000/testdb** 就可以看到数据添加成功的提示。

输出结果如下：



### 获取数据

Django提供了多种方式来获取数据库的内容，如下代码所示：



### 更新数据

修改数据可以使用save()或update()



### 删除数据

删除数据库中的对象只需调用该对象的delete()方法即可



## models模块

### 基本概念

Django 的 models 模块是 ORM（对象关系映射）系统的核心，让你能用 Python 代码操作数据库。

### 作用

from django.db import models

1、数据库表的 Python 表示

2、字段类型定义

3、数据库操作方法

4、查询接口

### 核心组件

#### Model 基类



1、models.Model 是所有模型的基类

2、继承后自动获得数据库操作能力

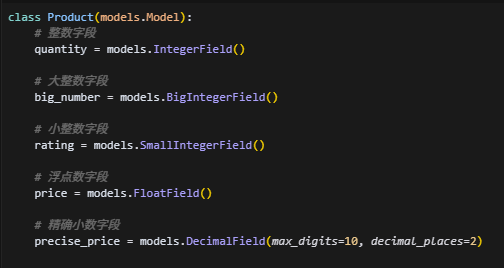
3、每个模型类对应一个数据库表

#### 字段类型 (Field Types)

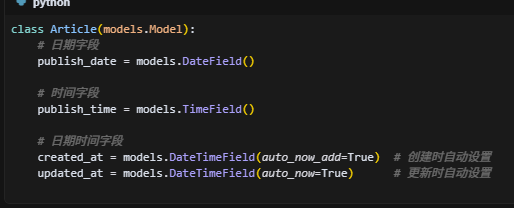
##### 文本字段



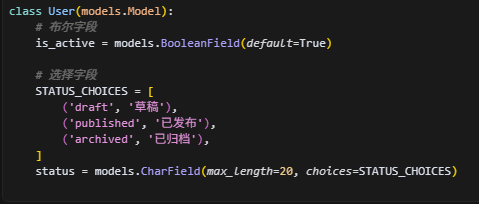
##### 数字字段



##### 日期时间字段



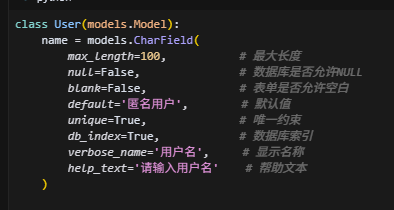
##### 布尔和选择字段



##### 关系字段



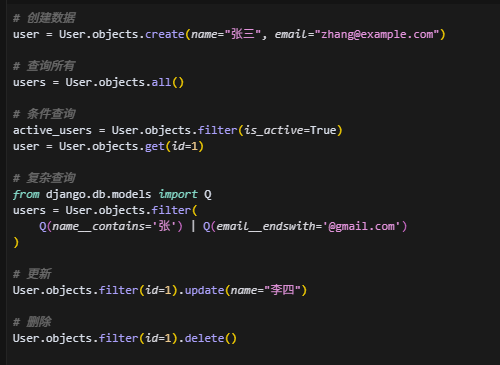
#### 字段选项



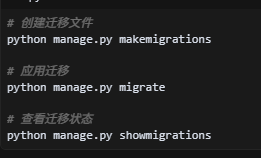
#### 模型方法



#### 查询管理器



#### 数据库迁移



### 关键优势

**1、ORM 映射**: 用 Python 对象操作数据库

**2、数据库无关**: 支持多种数据库后端

**3自动化**: 自动生成 SQL 语句

**4、类型安全**: 字段类型验证

**5、关系处理**: 简化外键和关联查询

**6、迁移管理**: 自动处理数据库结构变更

# 表单

## HTTP请求

HTTP协议以"请求－回复"的方式工作。客户发送请求时，可以在请求中附加数据。服务器通过解析请求，就可以获得客户传来的数据，并根据URL来提供特定的服务。

## GET方法

### 创建search.py 文件

创建一个 search.py 文件，用于接收用户的请求



### 创建search\_form.html 表单

在模板目录 templates 中添加 search\_form.html 表单

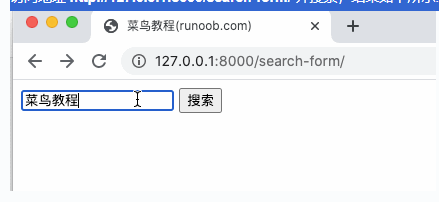


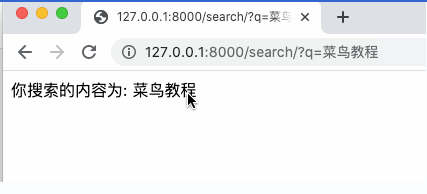
### 修改urls.py文件



### 访问

访问地址 **http://127.0.0.1:8000/search-form/** 并搜索，结果如下所示:





## POST方法

提交数据时更常用 POST 方法。我们下面使用该方法，并用一个URL和处理函数，同时显示视图和处理请求。

### 创建post.html文件



在模板的末尾，我们增加一个 rlt 记号，为表格处理结果预留位置。表格后面还有一个 {% csrf\_token %} 的标签。csrf 全称是 Cross Site Request Forgery。这是 Django 提供的防止伪装提交请求的功能。POST 方法提交的表格，必须有此标签。

### 创建search2.py文件

在HelloWorld目录下新建 search2.py 文件并使用 search\_post 函数来处理 POST请求



### 修改urls.py文件



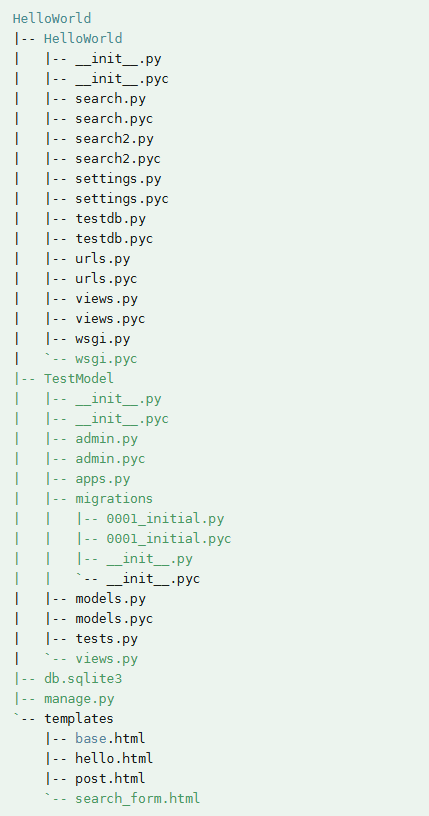
### 访问

访问 **http://127.0.0.1:8000/search-post/** 显示结果如下：





### 目录结构



## Request对象

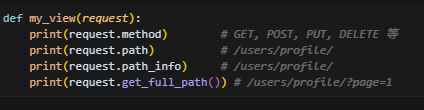
### 基本概念

Django 中的 Request 对象是处理 HTTP 请求的核心，包含了所有请求相关的信息。

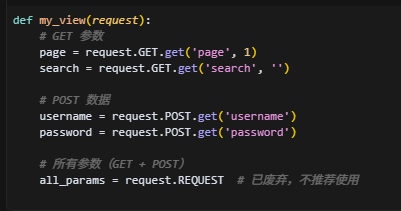


### 核心属性

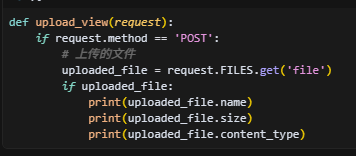
#### 1、请求方法和路径



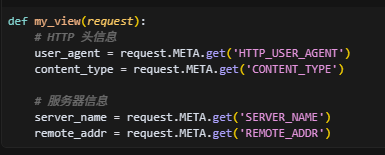
#### 2、请求数据



#### 3、文件上传

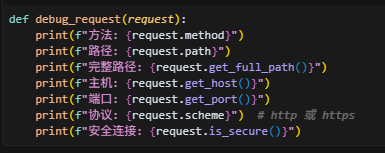


#### 4、请求头信息

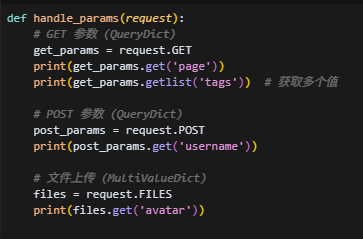


### 详细属性列表

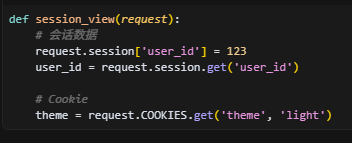
#### 基本信息



#### 请求参数

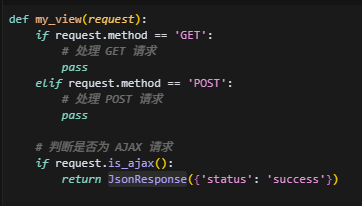


#### 会话和cookie

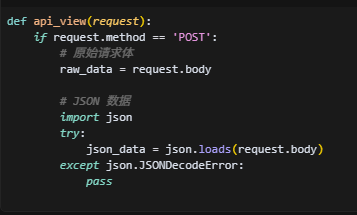


### 常用方法

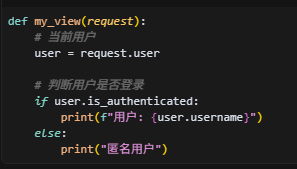
#### 1、判断请求类型



#### 2、获取请求体

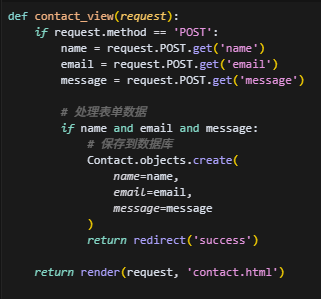


#### 3、用户信息

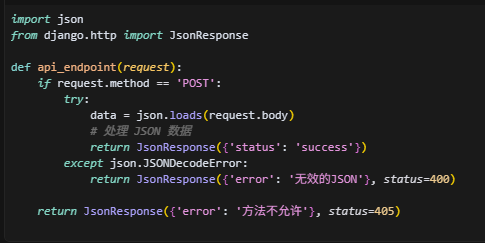


### 实际应用示例

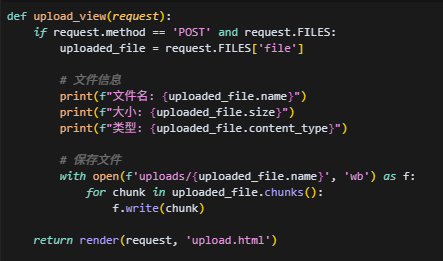
#### 1、表单处理



#### 2、API接口



#### 3、文件上传



### 注意事项

1、线程安全: 每个请求都有独立的 Request 对象

2、只读属性: 大部分属性是只读的

3、编码处理: Django 自动处理字符编码

4、安全性: 自动进行 CSRF 保护

## QueryDict对象

### 基本概念

## HttpResponse对象

### 基本概念

Django 中的 HttpResponse 对象用于向客户端返回 HTTP 响应，是视图函数的标准返回值。

### 基本用法



### 设置响应属性



### HttpResponse 子类

#### HttpResponseRedirect

### 响应处理最佳实践

1、状态码: 正确设置 HTTP 状态码

2、内容类型: 明确指定 Content-Type

3、安全头: 设置安全相关的响应头

4、缓存策略: 合理设置缓存控制

5、错误处理: 统一的错误响应格式

## JsonResponse对象

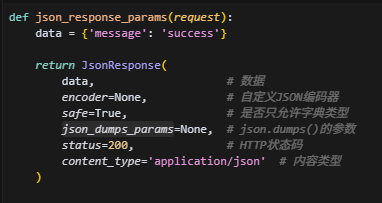
### 基本概念

Django 的 JsonResponse 是 HttpResponse 的子类，专门用于返回 JSON 格式的响应。

### 基本用法



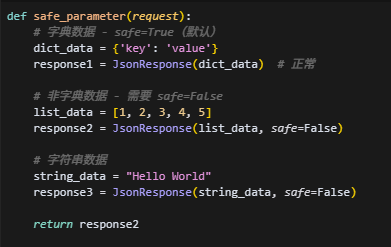
### 构造函数参数



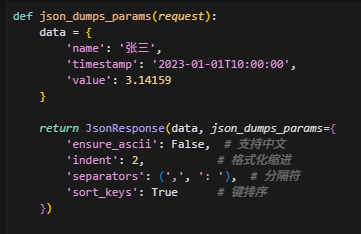
### 核心参数

#### safe

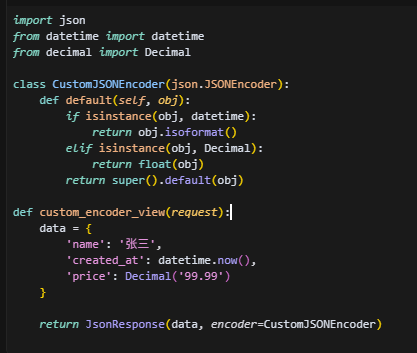
是否只允许字典数据类型



#### json\_dumps\_params

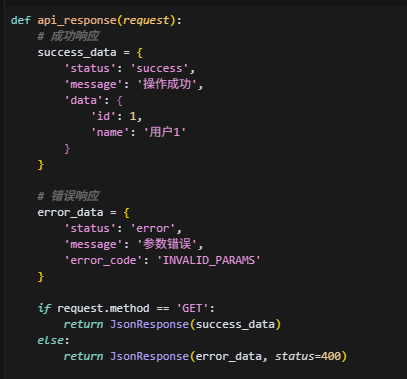


#### encoder



### 应用场景

#### 1、API响应格式



### 最佳实践

**1、统一响应格式**: 保持 API 响应格式一致

**2、错误处理**: 合理设置状态码和错误信息

**3、数据验证**: 返回前验证数据完整性

**4、性能优化**: 避免返回过大的 JSON 数据

**5、安全考虑**: 防止敏感信息泄露

# 视图

## 基本概念

一个视图函数，简称视图，是一个简单的 Python函数，它接受 Web 请求并且返回 Web 响应。响应可以是一个 HTML 页面、一个 404 错误页面、重定向页面、XML 文档、或者一张图片。

无论视图本身包含什么逻辑，都要返回响应。代码写在哪里都可以，只要在 Python 目录下面，一般放在项目的 views.py 文件中。每个视图函数都负责返回一个 HttpResponse 对象，对象中包含生成的响应。

视图层中有两个重要的对象：请求对象(request)与响应对象(HttpResponse)

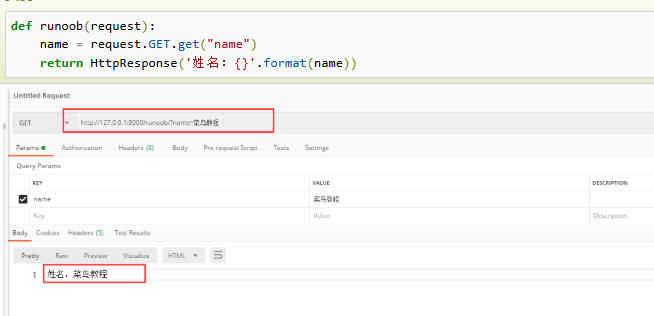
## 请求对象: HttpRequest 对象（简称 request 对象）

### GET属性

数据类型是 QueryDict，一个类似于字典的对象，包含 HTTP GET 的所有参数，有相同的键，就把所有的值放到对应的列表里。

取值格式：对象.方法。

get()：返回字符串，如果该键对应有多个值，取出该键的最后一个值。



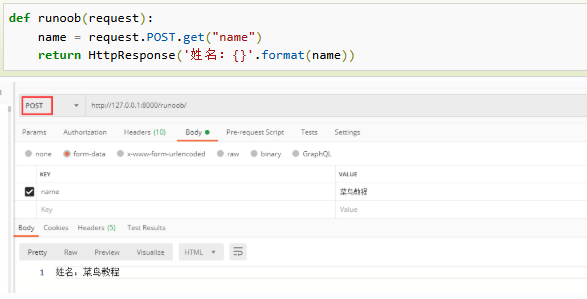
### POST属性

1、数据类型是 QueryDict，一个类似于字典的对象，包含 HTTP POST 的所有参数。

2、常用于form表单，form 表单里的标签 name 属性对应参数的键，value 属性对应参数的值。

取值格式：对象.方法

get()：返回字符串，如果该键对应有多个值，取出该键的最后一个值。



### body属性

# 路由

## 基本概念

路由简单的来说就是根据用户请求的 URL 链接来判断对应的处理程序，并返回处理结果，也就是URL与Django的视图建立映射关系。

Django 路由在 urls.py 配置，urls.py 中的每一条配置对应相应的处理方法。Django不同版本 urls.py 配置有点不一样。

### Django1.1.x 版本

**url() 方法**：普通路径和正则路径均可使用，需要自己手动添加正则首位限制符号。



### Django 2.2.x之后的版本

path：用于普通路径，不需要自己手动添加正则首位限制符号，底层已经添加。

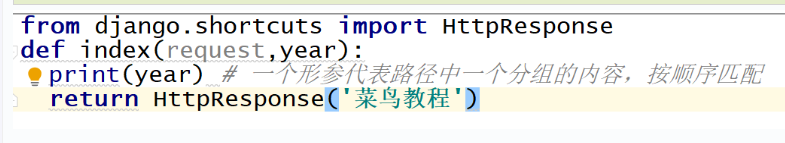
re\_path：用于正则路径，需要自己手动添加正则首位限制符号。



## 正则路径中的无名分组

无名分组按位置传参，一一对应。views 中除了 request，其他形参的数量要与 urls 中的分组数量一致。





## 正则路径中的有名分组

### 语法

(?P<组名>正则表达式)

有名分组按关键字传参，与位置顺序无关，views 中除了 request，其他形参的数量要与 urls 中的分组数量一致， 并且 views 中的形参名称要与 urls 中的组名对应。

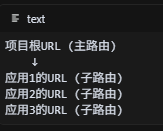




## 路由分发(include)

### 基本概念

路由分发是Django中将URL请求分配给不同应用（app）处理的机制。它允许项目的主URL配置将特定的URL模式委托给各个应用的URL配置文件处理。



### 为什么需要路由分发

1、模块化管理: 每个应用管理自己的URL

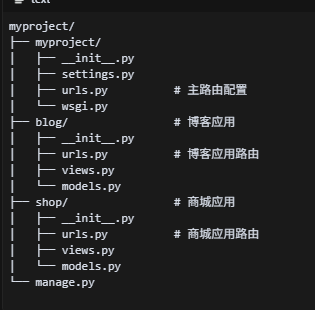
2、代码组织: 避免所有URL集中在一个文件中

3、可维护性: 便于团队协作和代码维护

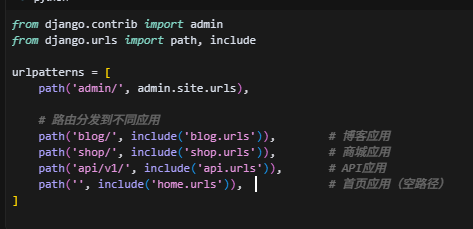
4、可重用性: 应用可以在不同项目中重用

### 代码示例

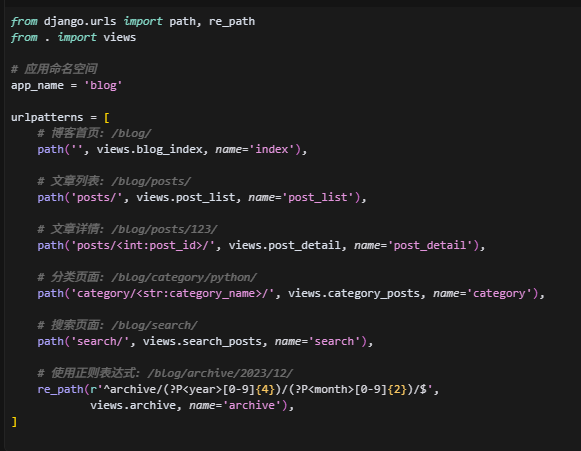
#### 1、项目结构



#### 2、主项目URL配置 (myproject/urls.py)



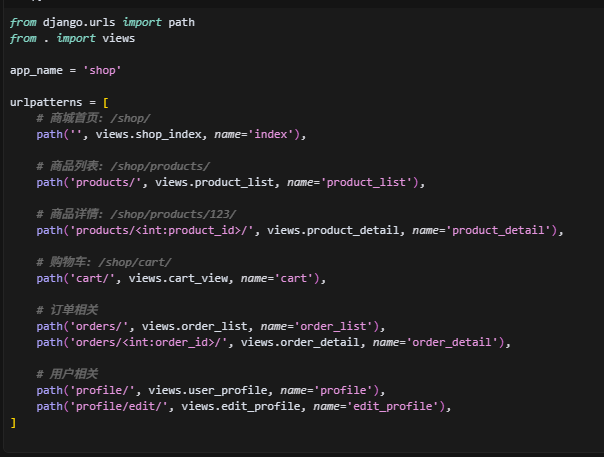
#### 3、博客应用URL配置 (blog/urls.py)



#### 4、博客应用视图 (blog/views.py)



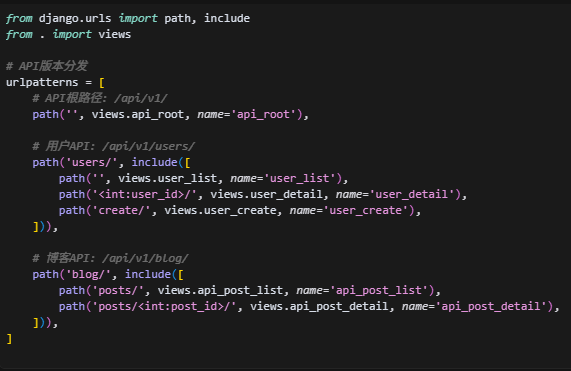
#### 5、商城应用URL配置 (shop/urls.py)



#### 6、商城应用视图 (shop/views.py)



#### 7、API应用URL配置 (api/urls.py)



### 总结

**1、分层管理**: 主项目管理应用级路由，应用管理具体功能路由

**2、模块解耦**: 每个应用独立管理自己的URL配置

**3、可扩展性**: 新增应用只需在主路由中添加一行include

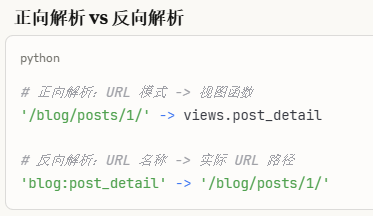
**4、命名空间**: 通过app\_name避免不同应用间的URL名称冲突

通过路由分发，Django项目能够保持良好的结构和可维护性，特别适合大型项目的开发和管理。

## 反向解析

### 基本概念

Django 路由中的反向解析是指通过 URL 名称来生成实际的 URL 路径，而不是直接在代码中硬编码 URL。



### 为什么需要反向解析

1、避免硬编码：URL 变更时不需要修改多处代码

2、维护性更好：集中管理 URL 配置

3、减少错误：避免手动拼接 URL 的错误

### URL命名

#### 1、基本命名



#### 2、带命名空间的命名



### 在视图中使用反向解析

#### 使用 reverse() 函数



#### reverse\_lazy()函数

##### 基本概念

reverse\_lazy() 和 reverse() 的作用基本相同，都是根据 URL 名称反向生成 URL 路径。但关键区别是：

reverse() 立即执行 URL 解析

reverse\_lazy() 延迟到实际需要时才执行解析

##### 使用场景

基于类的视图（Class-Based Views）中定义 success\_url



##### 为什么需要 lazy

在 Django 启动时，URL 配置是按特定顺序加载的。如果在类定义阶段就使用 reverse()，可能会遇到以下问题：



## 命名空间

### 基本概念

Django 路由中的命名空间（namespace）是一种组织和管理 URL 的方式，用于避免不同应用间的 URL 名称冲突，并提供更清晰的 URL 结构。命名空间就像给 URL 名称加上了一个前缀，让你可以通过 namespace:name 的形式来引用 URL。

### 应用命名空间

#### 1、在应用的 urls.py 中定义

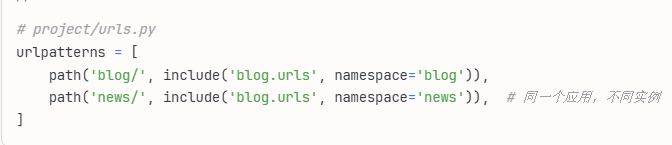


#### 2、在项目主 urls.py 中包含



### 实例命名空间

当你需要在同一个项目中多次使用同一个应用时，可以使用实例命名空间：



### 使用命名空间

#### 1、在模板中使用



#### 2、在视图中使用



#### 3、在表单中使用



### 实际示例

假设你有一个博客应用和一个商店应用



现在你可以清晰地区分：

blog:index - 博客首页

blog:detail - 博客文章详情

shop:index - 商店首页

shop:detail - 商品详情

### 好处

#### 1、避免命名冲突

不同应用可以有相同的 URL 名称

#### 2、代码更清晰

一眼就能看出 URL 属于哪个应用

#### 3、便于维护

重构时不会影响其他应用的 URL

#### 4、模块化

每个应用的 URL 独立管理

### 注意事项

1、如果没有定义 app\_name，就不能使用应用命名空间

2、命名空间和 URL 名称用冒号（:）分隔

3、嵌套的 include 可以创建多层命名空间

### 总结

命名空间是 Django 项目中管理复杂 URL 结构的重要工具，特别是在大型项目中非常有用。

## path函数

### 基本语法

path(route, view, kwargs=None, name=None)

### 参数

#### route (路由模式)

类型: 字符串

作用: 定义URL匹配模式

特点: 不使用正则表达式，使用简化的路径转换器语法

#### view (视图函数)

类型: 视图函数或类视图

作用: 处理匹配URL的请求

#### kwargs (可选)

类型: 字典

作用: 传递额外参数给视图函数

#### name (可选)

类型: 字符串

作用: 为URL模式命名，便于反向解析

### 路径转换器详解

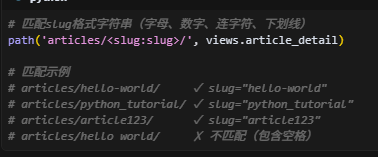
#### 1、str - 字符串转换器（默认）



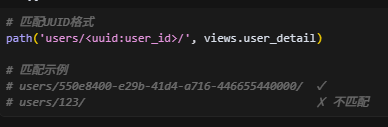
#### 2、int - 整数转换器



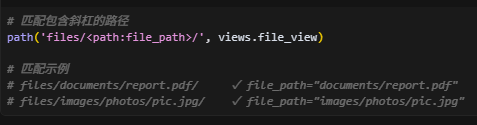
#### 3、slug - Slug转换器



#### 4、uuid - UUID转换器

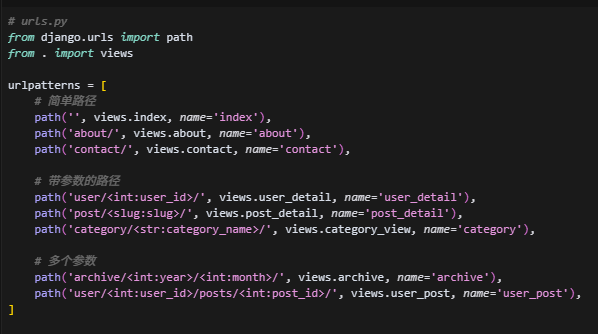


#### 5、path - 路径转换器



### 示例

1、基本用法示例



2、对应的视图函数



## include函数

### 基本概念

include() 函数允许你将 URL 配置模块化，把不同应用的 URL 配置分离到各自的文件中。



### 语法

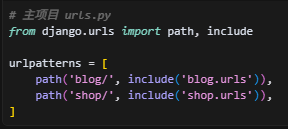


### 作用

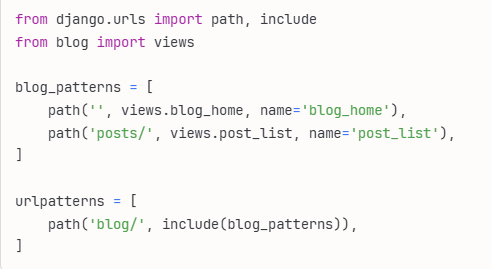
将URL模式委托给其他URLconf模块处理，实现路由分发

### 主要用法

#### 1、包含其他应用的URLs



#### 2、包含URL模式列表



# Django Admin 管理工具

# Django ORM - 单表实例

# Django Form 组件

# Django 用户认证（Auth）组件

# Django 视图 - FBV 与 CBV

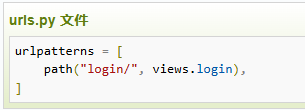
## 基本概念

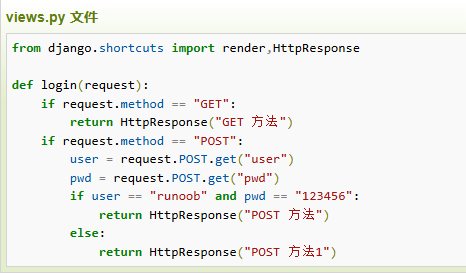
FBV（function base views） 基于函数的视图，就是在视图里使用函数处理请求。**CBV（class base views）** 基于类的视图，就是在视图里使用类处理请求。

## FBV

就是使用函数处理用户的请求，查看下面的示例：

路由配置：





如果我们在浏览器中直接访问 http://127.0.0.1:8000/login/ ，输出结果为：



### 应用场景

1、简单的业务逻辑：逻辑相对单一，不需要复杂的继承

2、快速开发：原型开发或小型项目

3、自定义程度高：需要精确控制每个细节

4、新手友好：学习成本低，逻辑清晰

## CBV

基于类的视图，就是使用了类来处理用户的请求，不同的请求我们可以在类中使用不同方法来处理，这样大大的提高了代码的可读性。

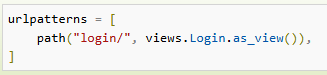
定义的类要继承父类 View，所以需要先引入库：

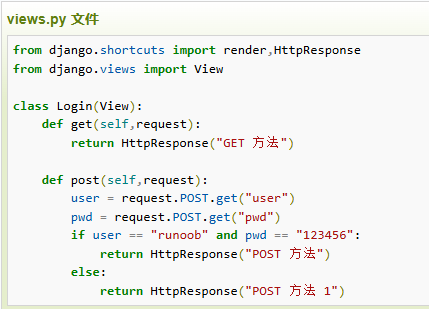
from django.views import View

执行对应请求的方法前会优先执行 dispatch 方法(在get/post/put...方法前执行)，dispatch() 方法会根据请求的不同调用相应的方法来处理。

在我们前面学到的知识都知道 Django 的 url 是将一个请求分配给可调用的函数的，而不是一个类，那是如何实现基于类的视图的呢？ 主要还是通过父类 View 提供的一个静态方法 as\_view() ，as\_view 方法是基于类的外部接口， 他返回一个视图函数，调用后请求会传递给 dispatch 方法，dispatch 方法再根据不同请求来处理不同的方法。

路由配置：





如果我们在浏览器中直接访问 http://127.0.0.1:8000/login/ ，输出结果为：



### 应用场景

1、标准的 CRUD 操作：增删改查逻辑

2、代码复用：多个视图有相似逻辑

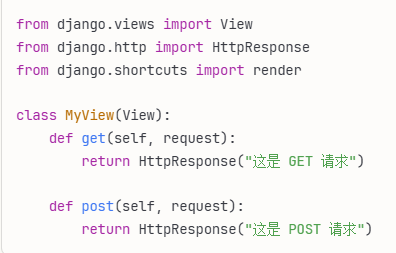
3、复杂业务逻辑：需要继承和扩展

4、REST API：符合 RESTful 设计

## View类

### 基本概念

View 类是Django 提供的所有基于类的视图的基础类，它封装了处理 HTTP 请求的通用逻辑。



### 核心机制

#### HTTP 方法分发

View 类会根据 HTTP 请求方法自动分发到对应的方法



#### as\_view() 方法

这是 View 类的关键方法，用于将类转换为可调用的视图函数



### 常用属性



### 核心方法



### 与函数视图的对比

#### 1、函数视图



#### 2、类视图



### View 类的优势

#### 1、代码组织

（1）不同 HTTP 方法的处理逻辑分离

（2）便于代码复用和继承

（3）更好的代码结构

#### 2、Django 生态

（1）与 Django 的其他 CBV 类无缝集成

（2）支持 Django 的认证和权限系统

（3）更好的表单处理支持

# 使用pycharm创建Django项目

https://www.runoob.com/pycharm/pycharm-django.html

[PyCharm Pro](https://www.jetbrains.com/zh-cn/pycharm/editions/)

# Django 结构工具窗口使用方法

https://www.jetbrains.com/zh-cn/help/pycharm/django-structure-tool-window.html

# 常用命令

## python manage.py runserver

# 启动开发服务器

## python manage.py migrate

# 执行数据库迁移

## python manage.py createsuperuser

# 创建超级用户

## python manage.py collectstatic

# 收集静态文件

## python manage.py shell

# 进入 Django shell

# manage.py文件

## 基本概念

manage.py 是 Django 项目的命令行管理工具，用于执行各种管理任务。

## 核心作用

### 1、环境配置

设置 Django 配置模块

### 2、错误处理

提供友好的错误提示

### 3、命令路由

将命令行参数传递给 Django 的管理系统

### 4、项目入口

作为整个 Django 项目的管理入口点

# 使用django框架，我想查看执行的sql语句

## 方法一

# Django rest framework

# 问题

## 1、django.db.utils.OperationalError: (1050, "Table 'alarm\_records' already exists")

### 解决方法

#### 1、确认表确实存在

用数据库客户端（如 Navicat、DBeaver、命令行等）确认 alarm\_records 表确实存在，并且结构和 Django 模型一致。

#### 2、标记迁移为已执行（fake migrate）

让 Django 只记录迁移已完成，不实际执行 SQL。

在项目根目录下执行：

python manage.py migrate project 0005 --fake

含义：

project：你的 app 名称

0005：你的迁移文件编号（即 0005\_alarmrecords\_devicemonitoringvalues\_and\_more.py）

这样 Django 会在 django\_migrations 表中记录这次迁移已完成，但不会再去创建表。

#### 3、检查迁移状态

执行下面的命令：

python manage.py showmigrations

确认 project 应用下 0005\_alarmrecords\_devicemonitoringvalues\_and\_more 前面有 [X]，表示已应用

#### 4、继续后续迁移

如果还有其他未迁移的内容，直接执行

python manage.py migrate

#### 总结

1、只要表结构和模型一致，--fake 是最安全的做法。

2、千万不要直接删表或手动改迁移文件，否则可能导致数据丢失或迁移系统混乱。